

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Disseny i implementació d'un servei de commutació de flufos multimèdia

Document: Memòria

Alumne: David Martínez Álvarez, u1939690

Tutor: Josep Lluís Marzo Lazaro

Departament: Arquitectura i Tecnologia de Computadors

Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): Juny 2019

Universitat de Girona

ESCOLA POLITÈCNICA SUPERIOR

GRAU D'ENGINYERIA INFORMÀTICA

TREBALL FINAL DE GRAU

Design and Implementation of a Multimedia Flow Switch Service

Document:
Report

Author:
David MARTÍNEZ, u1939690

Professor:
Dr. Jose Luis MARZO

Department:
Arquitectura i Tecnologia
de Computadors (ATC)

June, 2019



Universitat
de Girona



Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 1.1 | Motivations and purpose | 3 |
| 1.2 | Project objectives | 4 |
| 1.3 | Project structure | 4 |
| 2 | Methodology | 5 |
| 2.1 | Agile based on PXP | 6 |
| 3 | Planning | 9 |
| 3.1 | Customer requirements | 9 |
| 3.2 | Sub-projects definition | 10 |
| 3.3 | Tasks definition | 14 |
| 3.4 | Requirement Traceability Matrix | 20 |
| 3.5 | Planning schedule | 21 |
| 4 | Feasibility study | 23 |
| 4.1 | Technical concepts study | 23 |
| 4.2 | Technology study | 23 |
| 4.3 | Legal requirements study | 23 |
| 4.4 | Economic study | 24 |
| 4.5 | Conclusion | 27 |
| 5 | Framework and background knowledge | 29 |
| 5.1 | Concepts definition | 30 |
| 5.2 | Environments definition | 32 |
| 5.2.1 | Testing environment | 32 |
| 5.2.2 | Production environment | 33 |
| 5.3 | Project associates | 34 |
| 6 | System requirements | 35 |
| 6.1 | Functional requirements | 35 |
| 6.2 | Non-functional requirements | 37 |
| 7 | Studies and decisions | 39 |
| 7.1 | Hardware | 39 |
| 7.2 | Multimedia Flow Switch Server | 39 |
| 7.2.1 | WebRTC Technology | 39 |
| 7.2.2 | NAT problem | 41 |
| 7.2.3 | WebRTC Media Server | 42 |
| 7.2.4 | Decisions | 43 |
| 7.2.5 | MediaSoup SFU Server | 46 |
| 7.2.6 | Software libraries | 49 |
| 7.3 | <i>SaaS</i> -based application | 50 |
| 7.3.1 | Back-end | 50 |
| 7.3.2 | Front-end | 52 |
| 7.4 | <i>MWA</i> : <i>MFSS</i> integration with the <i>SaaS</i> -based application | 55 |
| 7.5 | Git client choice & practices | 55 |

| | | |
|-----------|--|------------|
| 7.6 | Documentation practices | 57 |
| 7.7 | Source code editor choice | 58 |
| 7.8 | Final software used and technical requirements | 59 |
| 7.9 | Marketing study | 60 |
| | 7.9.1 MWA name | 60 |
| | 7.9.2 User interfaces design | 60 |
| | 7.9.3 Logos | 61 |
| 7.10 | Business feasibility study | 63 |
| | 7.10.1 Value Proposition | 64 |
| | 7.10.2 Business competitive study | 66 |
| 8 | System analysis and design | 69 |
| 8.1 | System architecture | 69 |
| 8.2 | System requirements analysis | 70 |
| | 8.2.1 Functional requirements analysis | 70 |
| | 8.2.2 Non-functional requirements analysis | 75 |
| 8.3 | Use case diagram | 77 |
| 8.4 | Database design | 78 |
| 8.5 | Modules design | 81 |
| 8.6 | Front-end interfaces | 96 |
| 9 | Implementation and testing | 97 |
| 9.1 | <i>Flouescent</i> Server | 97 |
| 9.2 | React client | 100 |
| 9.3 | Web Socket protocol | 107 |
| 9.4 | Recordings | 109 |
| 9.5 | Final project structure | 111 |
| 10 | Deployment and Results | 113 |
| 10.1 | Deployment | 113 |
| 10.2 | Results | 116 |
| 11 | Conclusions | 119 |
| 12 | Future Work | 123 |
| 13 | Bibliography | 125 |
| 14 | User manual | 127 |
| 14.1 | Normal user manual | 127 |
| 14.2 | Admin user manual | 128 |

1 Introduction

This project consists of developing a scalable *Multimedia Flow Switch Server (MFSS)*. The resulting server is used to create a real-time *Meetings Web-based Application (MWA)*, considered as a *Multimedia Flow Switch Service*. The *MFSS* enables to manage participants media flows efficiently under a real-time meeting. The *MWA* provides a simple professional web application, using the *MFSS* services.

1.1 Motivations and purpose

Several Spanish companies continue to use traditional face-to-face meetings. The major motivation of this project is the necessity of having a simple, efficient, and affordable software that could communicate company members remotely. It becomes a complete IT project, as it demands system and programming skills.

One interesting finding for a computer engineer is to prove itself, making a complete IT project as a final thesis. It is especially helpful for an engineer to work and fit different theoretical concepts learned during the degree. This process provides enough practical experience to be prepared to go into the market world.

I've found WebRTC and real-time communications interesting since I've been experimenting with them at *Backbone networks and public data services* subject. People know about several popular and high used real-time applications like WhatsApp, Skype, Viber, or WeChat, those applications generally attend single users demands, though they do not provide company oriented solutions.

Although professional real-time meetings software already exist, most of them work under deprecated or inefficient technologies. Recent years have seen the emergence of new powerful tools, several times more efficient and scalable.

Real-time multimedia sharing services are diverse and frequently user-oriented. User-oriented services are optimized to work on one-by-one user conferences, or with reduced casual meetings. A company environment is more complex and needs efficient performance on many-to-many user meetings.

Real-time communications software is designed differently depending on its purpose. If the software is based on direct connections between users (peer-to-peer), then their real-time multimedia flows can be sent directly. On the other hand, it generates a scalability problem, as each user has to send his/her media to all the others, and receive all the other user's media by independent flows. As the project purpose is to develop a company-oriented meeting service, it is necessary to implement a central server that handles all media in real-time, and then redirects them correctly. A relevant server design enables to solve the scalability problem, as media flows would be fully controlled.

This project does not only implement software that could meet the needs of a particular company, it also implements a Meetings Web-based Application (*MWA*) *SaaS*-based providing real-time meeting functionalities to the stakeholders corporations. A Software as a Service (*SaaS*) is a software licensing and delivery model, in which software is licensed on a subscription basis and is centrally hosted.

As stated, the principal project purpose is to design and develop a Multimedia Flow Switch Server (*MFSS*), however, to test its correctness, the *MWA* is contemplated. The *MWA* is expected to be simple and limited, getting much less importance than *MFSS* development.

1.2 Project objectives

According to the purpose, the project objectives are:

1. Exploring and studying modern and efficient technologies for use in the *MFSS*.
2. Finding *MFSS* core tools that fill the project purpose or implementing a new one.
3. Developing a simple and functional application (*MWA*) *SaaS*-based to provide *MFSS* meetings services to those companies who want it.
4. Deciding some commercial aspects of *MWA*, such as names, value proposition, and service plans.
5. Providing understandable code implementations and complete documentation.

1.3 Project structure

The EPS bachelor's degree guide contains an example of a structure as a reference for final degree projects. However, some methodologies may need minor structural changes.

According to the guide, the *Feasibility study* section should be placed after the introduction and before the methodology. However, it was decided to place the feasibility study before the methodology and planning schedule, as the last ones can directly interfere with it. The project planning may vary depending on the methodology used and should typically be considered as a part of the feasibility study.

Consequently, the structure has been modified in the early sections, being from 1 to 4:

1. *Introduction.*
2. *Methodology.*
3. *Planning.*
4. *Feasibility study.*

2 Methodology

It is necessary to select a suitable methodology [1] [2] to perform an optimal project development. Methodologies are important for enterprises, especially for project managers, and each company follows its preferred way. The following description provides a brief comparison between the most used project methodologies. However, they are usually used with some modifications to fulfill particular necessities:

Agile

For many project teams, it is considered as an ideal methodology. Agile has methodologies within itself, such as Scrum and Kanban. There are some discrepancies about Agile that should be considered more as frameworks. However, they are used to develop services with their specifications, causing them to be considered as a methodology. Agile is based on developing automated tasks at determined timescales called sprints. This procedure is suited for projects that require flexibility and have an uncertain level of complexity, like new products or services.

Scrum

This is a framework for Agile methodology. It is comprised of five main values: Commitment, courage, focus, openness, and respect. Its objective is to develop complex products through collaboration, responsibility, and interactive progress using fixed-length iterations. Scrum differs from Agile in how it operates on certain events, roles, and artifacts.

Kanban

Kanban is another framework used in Agile methodology. It matches the work to the team's capacity. Its main goal is to ensure details done as fast as possible with no fixed-length iterations.

Lean

All methodologies have their pros and cons, and depending on what is chosen impacts on different product quality and development costs. Lean methodology is usually known as an ideal "Quality/Price" option. In other words, it maximizes customer value while minimizes waste. This procedure is suited for teams that are interested in transforming how they conduct doing business.

Waterfall

As some of the previous methodologies are relatively new, Waterfall is one of the traditional ones. It stresses the importance of having clear documentation, revealing that if a developer leaves the job in the middle of the project, someone else can start where he/she left. Agile is an "upgrade" of this methodology and faces many issues like non-adaptive design constraints, the poor consumer feedback and delayed testing period. This procedure is suited for projects that have been done previously with low change for surprises.

An ideal solution for this project in consideration of previously studied methodologies is Agile. However, this project is not collaborative, and the application is not developed gradually, as Agile represents. Consequently, it is necessary to find a way to adapt Agile methodology on a solo project.

2.1 Agile based on PXP

It is possible to find some useful adaptations, discovering that people have doubts about this exploring a little bit through forums. Personal Extreme Programming[3] (PXP) is a project methodology designed especially for solo software projects. The following PXP principles allow developers to be more flexible and responsive to changes:

- 1) It is essential to take control of daily work.
- 2) Developers have to learn from their development processes variations to improve their performance.
- 3) All tasks need to be accompanied by continuous testing.
- 4) Defects should be detected as early as possible to fix it at start-up phases when the cost is lower.
- 5) Developers should try to automate as much as possible their daily work.
- 6) Provide a coding standard to unify the project programming style can be helpful for future improvements or modifications.
- 7) As a solo methodology, a simple design is an ideal option to avoid complications.
- 8) Developers should release small pieces of software to ensure its correct behavior and successfully testing.

The PXP process phases are similar to Scrum. The main difference is that PXP has some environment adaptations for solo projects. **Table 1** shows a brief description of each phase. **Figure 1** illustrates the graphical representation of these phases:

| Phase | Description |
|---------------------------------|--|
| Requirements | The first thing to do is to create a document with functional and non-functional requirements of the system. |
| Planning | Assemble a set of tasks based on objectives and requirements, and then estimate their effort. Here is where major decisions have to be taken (e.g., technologies and programming languages). |
| Iteration initialization | Like Scrum, iterations should take 1 to 2 weeks. It is essential to choose a set of tasks that has maximum priority. In the case of a tie, then select ones that have less estimate effort. |
| Design | Model all system modules and functions that will be implemented. |
| Implementation | There are three phases: Test preparation and code implementation. When it works, refactor. |
| System testing | Verify that the working implementation meets the project requirements. |
| Retrospective | Iteration analysis of all data and then check effort estimations. Adapt the methodology if it is not working as ideally as it should. |

Table 1: PXP phases description.

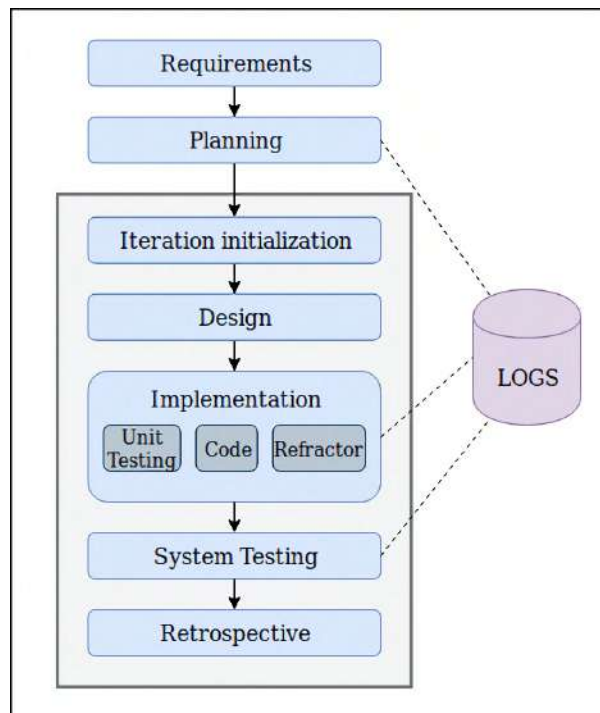


Figure 1: PXP process phases.

This adapted methodology is the most suitable for this project. Agile based on PXP is the chosen one for this project with few adaptations by needs. BCDS research group methodology is quite similar to the Scrum model, working by teams of two or four people. PXP methodology is working as an adaptation for this project. However, it is possible to make little changes at some phase.

The project planning is defined through the PXP requirements and the planning schedule. It has the following structure:

- 1) Customer requirements definition.
- 2) Sub-projects definition focused on the requirements.
- 3) Task definitions for each sub-project.
- 4) Gantt diagram and timeline elaboration of sub-projects and its tasks.

3 Planning

Project planning must be done based on the scope, objectives, and customer requirements of the project. The customer requirements are the core for the detailed system requirements, explained later in the *System requirements* section.

As my own project manager, I have decided to follow the PMBOK guide [4] practices. The project manager [5] has the overall authority and responsibility for managing the project according to the project plan.

The project range includes the planning, studies, design, development, testing, and deployment phases, without considering any changes in requirements. It is especially relevant for the Meetings Web-based Application *MWA* software meet all customer requirements. Furthermore, it has to provide full documentation and manuals. Once all documentation and code are complete, then the project is completed. The project code is not outsourced, as all work is performed internally.

3.1 Customer requirements

Companies attend to their customer's requirements to develop software applications for them. This project is being performed in a completely autonomous way. Consequently, I am my own customer.

The following list defines customer requirements based on project objectives:

- CR1** Find an appropriate technology¹ to design and implement the Multimedia Flow Switch Server *MFSS*.
- CR2** Develop the *MFSS* based on the previously chosen technology.
- CR3** Design and implement a company *SaaS*-based application.
- CR4** Integrate the *MFSS* with the company *SaaS*-based application, resulting in a *Meetings Web Application (MWA)*.
- CR5** Perform a simple marketing and business feasibility studies to provide some commercial vision for *MWA* software.
- CR6** Deploy the *MWA* providing complete documentation and manuals.

The customer requirements have some dependencies between them. **Table 2** shows the dependency matrix, which displays those internal dependencies.

¹An appropriate technology refers to the most accurate one according to project objectives

| | CR1 | CR2 | CR3 | CR4 | CR5 | CR6 |
|-----|-----|-----|-----|-----|-----|-----|
| CR1 | | | | | | |
| CR2 | X | | | | | |
| CR3 | | | | | | |
| CR4 | X | X | X | | | |
| CR5 | | | | | | |
| CR6 | X | X | X | X | | |

Table 2: Customer requirements dependencies matrix.

To perform the *MFSS*, it is necessary to first find an appropriate technology, being **CR2** dependent on **CR1**. The *MFSS* integration with *SaaS*-based application needs both software to be made, being **CR4** dependent on **CR2** and **CR3**. Finally, it is not possible to provide documentation and manuals before the *MWA*, being **CR6** dependent on **CR4**.

3.2 Sub-projects definition

The next step is to define modular and independent sub-projects that allows passing from the initial to the final project environment, with all customer requirements and project objectives accomplished.

Sub-projects definition has been performed by defining a deductive reasoning flow, represented through a Project diagram. The project diagram is partially shown divided into three divided parts: The initialization, the development, and the finalization.

The initialization phase is composed of the testing environment preparation and basic studies. In other words, it is composed of the sub-projects that have to be performed before any code development.

It is essential to define a testing environment before starting with customer requirements. Environment preparation consists basically of desktop computer availability. As the *MWA* application is defined as *SaaS*-based, it needs a desktop computer that enables local execution and testing. Once the testing environment is prepared, then it is possible to research an appropriate technology for the *MFSS*. The technology has to accomplish the project requirements and objectives. This procedure is deeply documented in the *Studies and decisions* section.

The technology research opens two workflows, depending on its result. An ideal situation is one where it is found an appropriate technology for the *MFSS*, although it is uncertain. If the technology is found, the next sub-project would be to analyze, configure, and append it to the server. Otherwise, it would be necessary to implement a technology that fulfills customer requirements and project objectives demands. This open scenario requires the diagram to be divided into two possible paths, one for each sub-project workflow. **Figure 2** illustrates the initialization phase of the Project diagram, including the defined sub-projects.

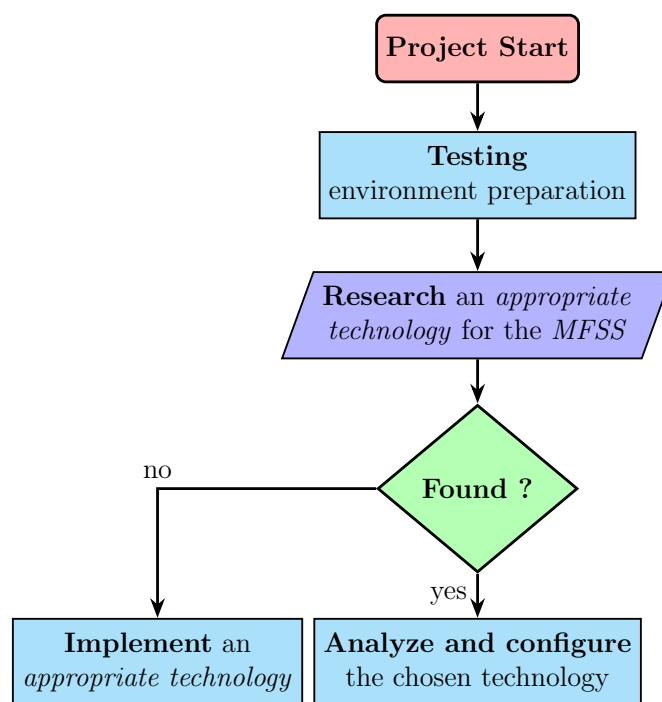


Figure 2: Project diagram: Initialization (1/3).

The development phase is composed of the whole application modules design and development. It also contains the marketing & business feasibility studies. It begins developing the *MFSS* through configuring and integrating the founded or implemented technology.

In parallel with *MFSS*, it is essential to design and develop the *SaaS*-based application that integrates the *MFSS*. This application is professional-oriented, offering different service plans. Since the *SaaS*-based application has no previous dependencies, it could be performed in the project beginning after *Testing environment preparation* sub-project.

With *MFSS development: Technology integration* and *SaaS-based application: Design and development* sub-projects performed, the next one consists of their integration by creating the final *MWA*. Furthermore, the integration includes software testing and error correction processes, which can revert steps backward to both *SaaS*-based application and *MFSS* development to solve bugs or design problems.

In parallel with all the previous defined sub-projects, there is one more requirement with no previous dependencies, the *MWA* marketing and business studies. It would be better to perform those studies before *MWA* creation to check if the application is commercially feasible. If it is not, it can be an important indicator that the application approach should be revised.

The phase is not completed, as it is essential to write the final project report. It is important to recall that this is an end-of-degree project, causing a remarkable amount of time ($\approx 1/3$) to be spent on the report writing. It is written in parallel with all the other sub-project. **Figure 3** illustrates the initialization and development phases of the Project diagram, including the initialization and development defined sub-projects.

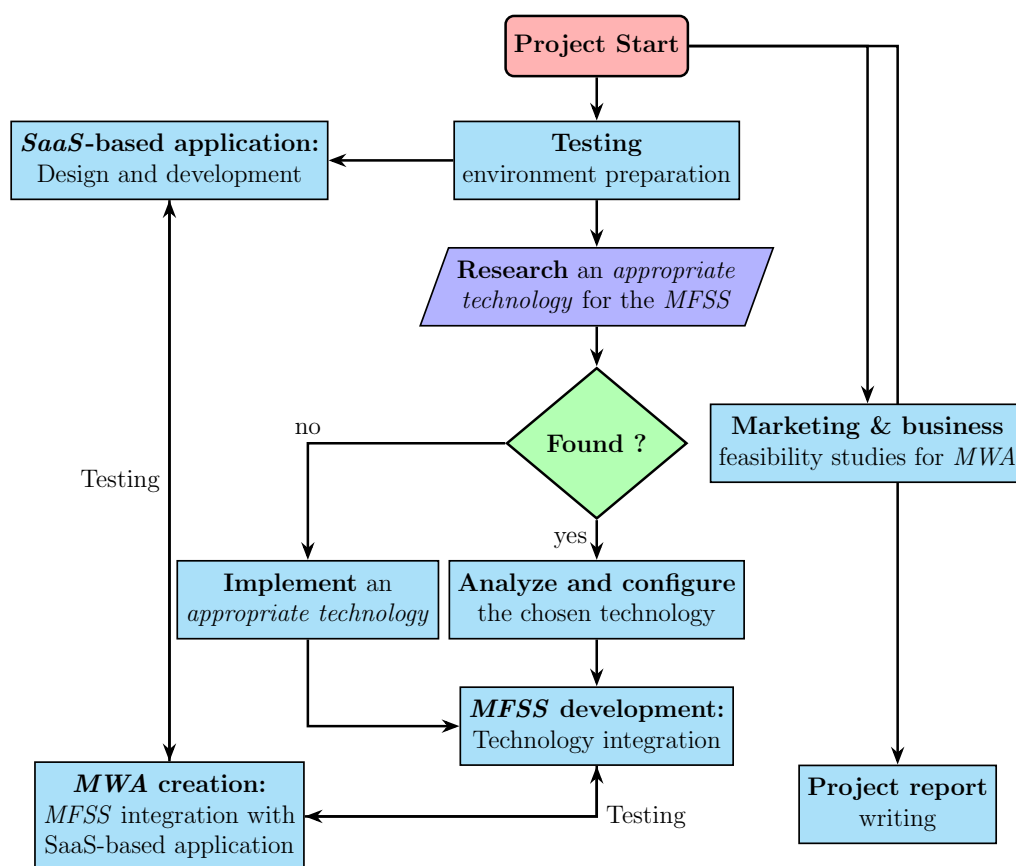


Figure 3: Project diagram: Initialization and development (2/3).

The finalization phase is composed of the *MWA* deployment, documentation, and manuals. It integrates all the development phase sub-projects, enabling the project completion.

The *MWA* is prepared for production, although it is necessary to prepare a production environment for subsequent deployment. *MWA* documentation and manuals receive special attention, becoming an important part. It is important to note that this process only contemplates *MWA* related information, excluding the project report. The documentation is based on the source code for possible future developers. The manuals are user-based, providing a basic user guide. **Figure 4** illustrates the integration, development, and finalization phases of the Project diagram, including all sub-projects.

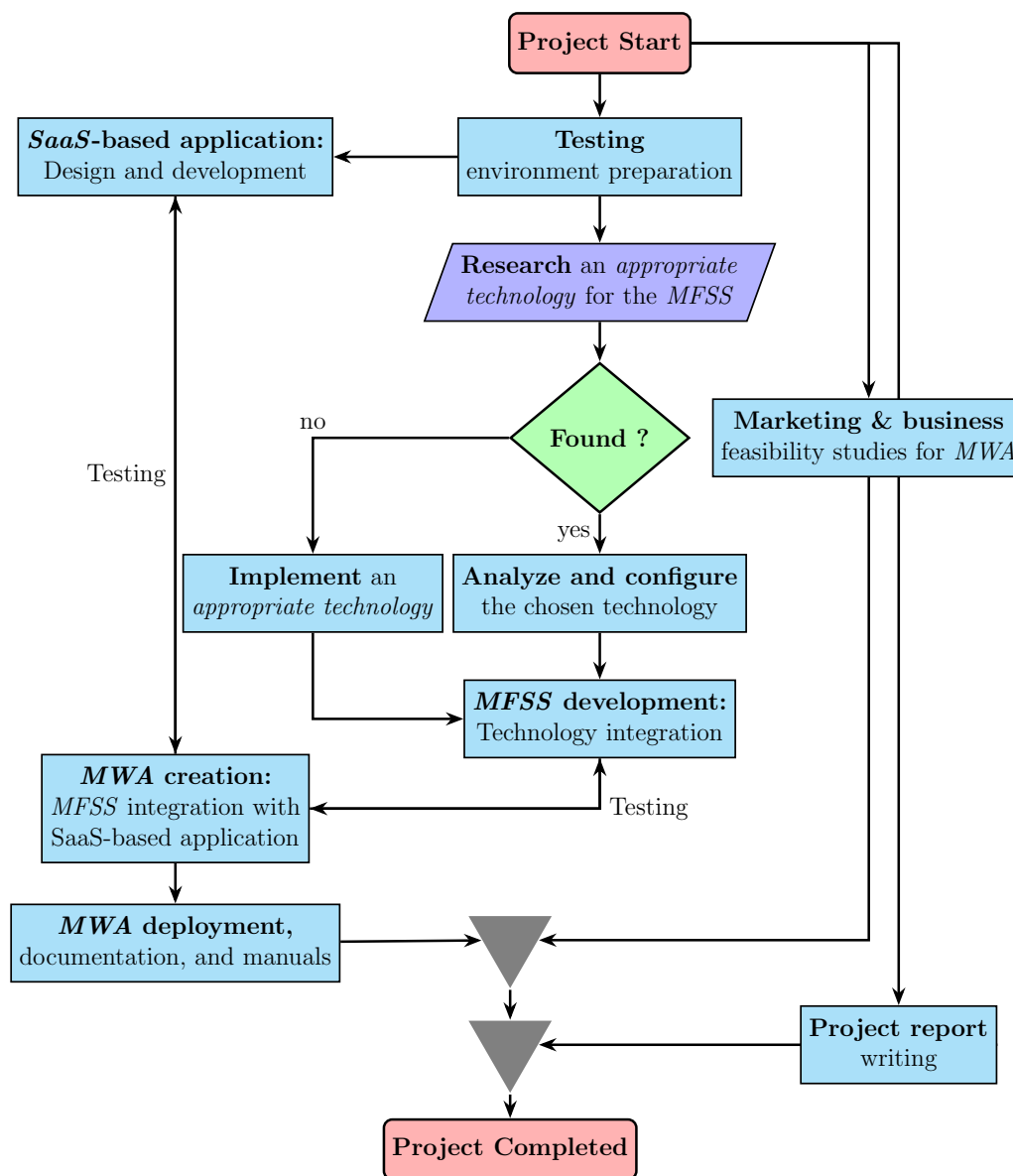


Figure 4: Project diagram: Initialization, development, and finalization (3/3).

3.3 Tasks definition

It is essential to define a set of tasks for each sub-project. Those tasks enables a complete and accurate planning schedule elaboration. All sub-project tasks are defined in a detailed manner.

As there are many sub-projects, they are bonded together in work areas for better understanding and readability. The following list defines those work areas with their corresponding sub-projects:

- **Environment preparation.**
 - Testing environment preparation.
- **MFSS development.**
 - Research an appropriate technology for the *MFSS*.
 - Analyze and configure the chosen technology or implement an appropriate one.
 - *MFSS* development: Technology integration.
- **SaaS-based application development.**
 - *SaaS*-based application: Design and development.
- **MWA development.**
 - *MWA* creation: *MFSS* integration with *SaaS*-based application.
 - *MWA* deployment, documentation, and manuals.
- **Marketing & business feasibility studies.**
 - Marketing & business feasibility studies for *MWA*.
- **Project report.**
 - Project report writing.

The execution order is fairly clear except with the *Marketing & business feasibility studies for MWA* sub-project. As said in the *Sub-projects definition* section, it is performed before *SaaS-based application: Design and development* as it can influence the *SaaS*-based application user interfaces.

As a result, the sub-projects are ordered as follows:

- S1** Testing environment preparation.
- S2** Research an appropriate technology for the *MFSS*.
- S3** (a) Analyze and configure the chosen technology.
or
(b) Implement an appropriate technology.
- S4** *MFSS* development: Technology integration.
- S5** Marketing & business feasibility studies for *MWA*.
- S6** *SaaS*-based application: Design and development.
- S7** *MWA* creation: *MFSS* integration with *SaaS*-based application.
- S8** *MWA* deployment, documentation, and manuals.
- S9** Project report writing.

The **S3a** and **S3b** sub-projects have been defined as the third sub-project with two different workflows, depending on the **S2** sub-project result.

1) Testing environment preparation

As a cloud *SaaS*-based application, the *MWA* does not require a sophisticated testing environment. Thanks to BCDS research group IT equipment, many virtualized servers and desktop computers are available to be included in project environments.

The following list shows all this sub-project related tasks, considering that software could be developed and tested at the same computer:

- S1.1** Desktop computer availability. It is possible to perform all the sub-projects except *MWA deployment, documentation, and manuals* using a single desktop computer, although it is possible to have multiple computer environments.
- S1.2** Prepare a testing environment for those computers. The computers only need a desktop-based OS, an internet connection, and software development compilers, editors, and tools defined at the *Framework and background knowledge* section.

2) Research an appropriate technology for the *MFSS*

This sub-project contains all necessary tasks to research an appropriate technology for the *MFSS*. It could be done after the *Testing environment preparation* sub-project, although it is not strictly necessary.

The following list shows all this sub-project related tasks, starting from the customer requirements needs and ending with the final technology decision:

- S2.1** Technology requirements definition based on customer requirements needs.
- S2.2** First search for appropriate technology.
- S2.3** Limit the search results based on project needs.
- S2.4** Limit the search results based on an analysis of each technology (e.g., capacity, performance, scalability, and utility).
- S2.5** Limit the search results with some testing processes.
- S2.6** Decide an ideal option by considering all obtained data from limitation processes. If there is no feasible candidate, create and develop a new one.

3a) Analyze and configure the chosen technology

Requirements:

- S1** Testing environment preparation.
- S2** Research an appropriate technology for the *MFSS*.

This sub-project tasks are dependent on the chosen technology and are performed in case of having a feasible technology candidate. The most significant **S3a** tasks are related to learning all necessary configurations to integrate the technology by creating the *MFSS*.

The following list shows all this sub-project related tasks:

- S3a.1** Analyze the chosen technology and other projects based on it.
- S3a.2** Search a proper free framework that implements the technology. If it exists, integrate it by creating the *MFSS*. Otherwise, implement a new one.
- S3a.3** Use a simple technology application to test it at the testing environment.

3b) Implement an appropriate technology

Requirements:

- S1** Testing environment preparation.
- S2** Research an appropriate technology for the *MFSS*.

This sub-project consists of implementing *MFSS* technology. Its tasks are performed in case of not having found a feasible technology candidate.

If sub-project tasks have to be performed, they may need to be reconsidered and probably be a part of more sub-projects. With this in mind, the following list shows all this sub-project related tasks:

S3b.1 Search information about the appropriate design of multimedia flow switch servers.

S3b.2 Search information about signaling protocols.

S3b.3 Design an efficient switch according to the studied information.

S3b.4 Think about a way to control the switch and distribute multimedia flows.

S3b.5 Think about a way to reroute flows to an external endpoint allowing recordings.

S3b.6 In-depth technology testing.

The **S3** sub-project has two possible workflows, *(a)* or *(b)* depending on the **S2** sub-project result. However, the project planning studies only consider the most probably **S2** sub-project result, having found an appropriate technology.

4) *MFSS* development: Technology integration

Requirements:

S1 Testing environment preparation.

S2 Research an appropriate technology for the *MFSS*.

S3 (a) Analyze and configure the chosen technology.

or

(b) Implement an appropriate technology.

This sub-project consists of integrating the chosen or implemented technology in a server by creating the *MFSS*.

The following list shows all this sub-project related tasks:

S4.1 Basic server implementation.

S4.2 Integrate the chosen or implemented technology in the performed server by creating the *MFSS*.

S4.3 In-depth *MFSS* testing.

5) Marketing & business feasibility studies for *MWA*

As explained before, this sub-project has no other sub-project dependencies. It consists of making a marketing and business feasibility studies for the *MWA*. The marketing study provides some additional marketing value, such as commercial names, slogans, and plans. The business feasibility study analyzes the hypothetical situation of a *MWA* commercialization.

The following list shows all this sub-project related tasks:

S5.1 Marketing study elaboration.

- (a) Explore some basic well practices to think about the *MFSS* name, slogans, and other marketing fields.
- (b) Explore some business criteria for designing correct and profitable service plans.

S5.2 Business feasibility study elaboration.

- (a) Value proposal study.
- (b) Competition analysis.
- (c) Business plans definition.

6) *SaaS*-based application: Design and development

Requirements:

S1 Testing environment preparation.

This sub-project has only one previous dependency, the *Testing environment preparation* sub-project, and does not depend on the *MFSS* development process. It consists of many studies about usable candidates technologies and decision making (e.g., web application design). Once studies are complete, then the *SaaS*-based application is developed and minimally documented.

The following list shows all this sub-project related tasks:

S6.1 Study and decide some technologies, programming languages, and important libraries to use at *SaaS*-based application.

S6.2 Perform the *SaaS*-based application server (back end) design according to customer requirements.

S6.3 Perform the *SaaS*-based application interface (front end) design according to customer requirements.

S6.4 Develop and test the back end server.

S6.5 Develop and test the front end interface and integrate it correctly with the server.

S6.6 Provide minimal source code documentation. Although a specific documentation sub-project exists after *MWA* creation, it is important to document some code functionalities to be correctly interpreted later.

7) *MWA* creation: *MFSS* integration with *SaaS*-based application

Requirements:

- S1** Testing environment preparation.
- S2** Research an appropriate technology for the *MFSS*.
- S3** (a) Analyze and configure the chosen technology.
or
(b) Implement an appropriate technology.
- S4** *MFSS* development: Technology integration.
- S5** *SaaS*-based application: Design and development.

This sub-project depends directly on the *MFSS development: Technology integration* and the *SaaS-based application: Design and development* ones. It consists of those two sub-project integrations by creating the *MWA* application.

The following list shows all this sub-project related tasks:

- S7.1** Design a communication protocol between software.
- S7.2** Integrate the *MFSS* with the *SaaS*-based application by creating the *MWA*.
- S7.3** In-depth testing.

8) *MWA* deployment, documentation, and manuals

Requirements:

- S1** Testing environment preparation.
- S2** Research an appropriate technology for the *MFSS*.
- S3** (a) Analyze and configure the chosen technology.
or
(b) Implement an appropriate technology.
- S4** *MFSS* development: Technology integration.
- S5** *SaaS*-based application: Design and development.
- S6** *MWA* creation: *MFSS* integration with *SaaS*-based application

These sub-project goals are to deploy, document, and write user manuals for the *MWA*. The deployment process consists of a production environment preparation and the software source code build. It is essential to provide comprehensive documentation and user basic manual. The documentation covers both source code documentation and the necessary procedures to install *MWA* on other computers.

The following list shows all this sub-project related tasks:

S8.1 Production environment preparation.

S8.2 *MWA* deployment process development.

S8.3 Provide a comprehensive software generalized documentation following the most used standards.

S8.4 Provide a basic user manual.

9) Project report writing

The project report writing is performed in parallel with all the other procedures. However, it is possible that some sections may require specific task dependencies essential for their documentation.

The following list shows all this sub-project related tasks:

S9.1 Report writing.

3.4 Requirement Traceability Matrix

Requirement Traceability Matrix (RTM) captures all requirements proposed by the customer or software development team and their traceability in a single document. In other words, it is a matrix that maps and traces user requirements with, in this case, sub-projects and tasks. The central purpose of RTM is to ensure that all tasks are covered. No functionality should miss while doing software testing.

The project planning has started from customer requirements (CR), and it has been moving forward through sub-projects definition and their tasks. The traceability matrix adds a graphical point of view, making the customer requirements as clear as possible. **Figure 5** exposes the RTM matrix.

| | S1 | | S2 | | | | | S3 | | | S4 | | | S5 | | S6 | | | | | | S7 | | | S8 | | | | S9 | |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| | 1.1 | 1.2 | 2.1 | 2.2 | 2.3 | 2.4 | 2.5 | 2.6 | 3.1 | 3.2 | 3.3 | 4.1 | 4.2 | 4.3 | 5.1 | 5.2 | 6.1 | 6.2 | 6.3 | 6.4 | 6.5 | 6.6 | 7.1 | 7.2 | 7.3 | 8.1 | 8.2 | 8.3 | 8.4 | 9.1 |
| CR1 | | | X | X | X | X | X | X | | | | | | | | | | | | | | | | | | | | | | |
| CR2 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | | | | | | | | | | | | | | | | |
| CR3 | X | X | | | | | | | | | | | | | | | X | X | X | X | X | X | | | | | | | | |
| CR4 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | | | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| CR5 | | | | | | | | | | | | | | X | X | | | | | | | | | | | | | | | |
| CR6 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

Figure 5: Requirements Traceability Matrix (RTM).

3.5 Planning schedule

Gantt diagrams are one of the most used tools to visualize work plannings and provide an indication of how a project is evolving. It is a representation of pending tasks according to the predicted time to perform them.

The predicted time is represented horizontally, in which each column represents a determinate amount of time (e.g., days, weeks). The process evolution is defined vertically, from the beginning to the end. The project started by the middle of March 2019, and the deadline is on June 11th, having exactly 13 available weeks. The days are placed on the horizontal axis. The tasks are placed on the vertical axis, ordered by the expected realization order (from left to right). A square represents each task occupying the space of its time frame.

The most common way to make a Gantt diagram is with a spreadsheet-type editor or an online application. However, it can be generated with other less common software. The selected tool to make this project Gantt diagram is the *TeamGantt* online application [6], which provides a free version for one single project. **Figure 6** illustrates the resulting diagram.

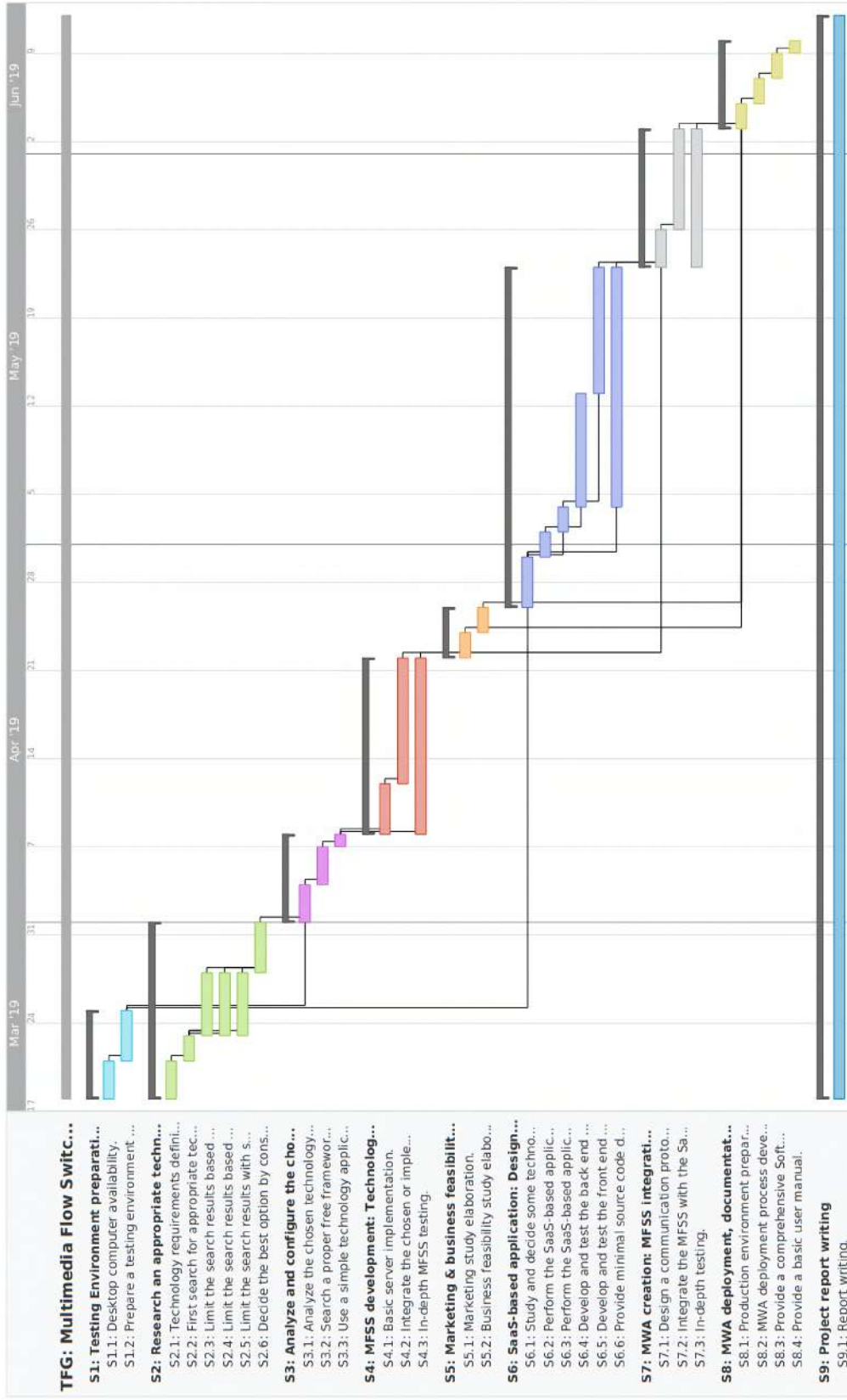


Figure 6: Gantt Diagram.

4 Feasibility study

This section exposes the whole Meetings Web-based Application *MWA* software feasibility study, based on the project planning. It is important not to confuse this feasibility study with the marketing & business study, defined as **S5** sub-project. The first one only takes into account the application related functionalities, and the other ones are based on the company that might commercialize the software.

According to the official fifteen *ECTS* credits dedicated to this end-of-degree project, the elaboration should take about 375 hours. These hours together with the available weeks (13) results on about 4.12 hours daily spent time.

A feasibility study starts with the requirements and project planning, according to the PMBOK guide. The *MWA* feasibility study covers technical concepts, technology, legal and economic studies.

4.1 Technical concepts study

Technical concepts knowledge is particularly important and has been suitably acquired during the degree. As the *Motivations and purpose* section explains, this end-of-degree work is a complete IT project where several theoretical concepts are covered. The subjects coursed during the degree provide all the necessary knowledge to perform the Multimedia Flow Switch Server *MFSS*.

4.2 Technology study

The number of existent technologies is continually growing, enabling software functionalities to be performed in many ways. The computer engineering four-year degree provides enough knowledge to make studies, and also decide which technology is better to apply. Therefore, sufficient technology knowledge can be guaranteed.

4.3 Legal requirements study

All software must meet their corresponding legal requirements, based on the provided services. Those requirements are different depending on the software utilization country. As the *MWA* is developed under European law, only European regulations are taken into account, such as GDPR (General Data Protection Regulation). The knowledge acquired at legislation and professional ethics subject ensure that *MWA* meets all necessary legal requirements.

4.4 Economic study

The economic study is an important aspect to find out if the *MWA* is economically feasible, taking into account that *MWA* is developed under an IT company. The study scope considers the design and implementation cost, excluding indirect expenses (e.g., hardware equipment, electricity costs). The *MWA* only uses free or open software, being not necessary to consider any additional license expense. Consequently, the study only takes into account the following costs:

- software analysis and design costs.
- Implementation costs.
- Production environment preparation costs.

The software developer salary is considered as **27€/hour**, which is the top nine European software engineer average [7]. The salary is estimated based on 40 hours a week and 28 days vacation job.

The project schedule defined at the *Planning* section and the calculated salary allows for making a cost table. It includes all project tasks related to *MWA* development, showing the expected time and their own engineer/developer costs. It is important to note that the **S9** is not included, as it is not related to *MWA* software development.

| Code | Task description | Hours | Cost (€) |
|--------------------|--|------------|--------------|
| S1.1 | Desktop computer availability. | 2 | 54 |
| S1.2 | Prepare a testing environment for those computers. | 3 | 81 |
| S2.1 | Technology requirements definition based on client requirements needs. | 8 | 216 |
| S2.2 | First search for appropriate technology. | 15 | 405 |
| S2.3 | Limit the search results based on project needs. | 3 | 81 |
| S2.4 | Limit the search results based on an analysis of each technology. | 5 | 135 |
| S2.5 | Limit the search results with some testing processes. | 5 | 135 |
| S2.6 | Decide the best option by considering all obtained data from limitation processes. | 5 | 135 |
| S3.1 | Analyze the chosen technology and other projects based on it. | 5 | 135 |
| S3.2 | Search a proper free framework that implements the technology. | 12 | 324 |
| S3.3 | Use a simple technology application to test it at the testing environment. | 6 | 162 |
| S4.1 | Basic server implementation. | 10 | 270 |
| S4.2 | Integrate the chosen or implemented technology in the performed server by creating the <i>MFSS</i> . | 26 | 702 |
| S4.3 | In-depth <i>MFSS</i> testing. | 10 | 270 |
| S5.1 | Marketing study elaboration. | 8 | 216 |
| S5.2 | Business feasibility study elaboration. | 8 | 216 |
| S6.1 | Study and decide some technologies, programming languages, and important libraries to use at <i>SaaS</i> -based application. | 5 | 135 |
| S6.2 | Perform the <i>SaaS</i> -based application server (back end) design according to client requirements. | 10 | 270 |
| S6.3 | Perform the <i>SaaS</i> -based application interface (front end) design according to client requirements. | 17 | 459 |
| S6.4 | Develop and test the back end server. | 20 | 540 |
| S6.5 | Develop and test the front end interface and integrate it correctly with the server. | 35 | 945 |
| S6.6 | Provide minimal source code documentation. | 5 | 135 |
| S7.1 | Design a communication protocol between software. | 15 | 405 |
| S7.2 | Integrate the <i>MFSS</i> with the <i>SaaS</i> -based application by creating the <i>MWA</i> . | 5 | 135 |
| S7.3 | In-depth testing. | 5 | 135 |
| S8.1 | Production environment preparation. | 10 | 270 |
| S8.2 | <i>MWA</i> deployment process development. | 15 | 405 |
| S8.3 | Provide a comprehensive software generalized documentation following the most used standards. | 2 | 54 |
| S8.4 | Provide a basic user manual. | 5 | 135 |
| Grand Total | | 280 | 7,560 |

Table 3: Estimated cost table.

As **Table 3** shows, this *MWA* economic study results in **280** work hours with a **7,560€** estimated cost. It is difficult to estimate the average cost of an application, such it depends on many aspects. However, there are some online applications dedicated to solving this problem. One of the best is the Estimate My App [8], which seems to be a reliable option [9] to obtain an approximated idea that can be compared with the predicted costs. This online tool only asks for different specs that the desired application has to contain, and then compute an estimated cost result based on computer learning algorithms.

The following list shows the Estimate My App input data provided, selected by customer requirements:

- **1. How big is your app?** Small.
- **2. What level of UI would you like?** Basic.
- **3. Users & Accounts** Email / Password Sign Up & Multi-tenant Accounts.
- **4. User Generated Content** Dashboard & File Uploading & User Profiles & Audio / Video processing.
- **6. Social & Engagement** Messaging.
- **7. Billing & eCommerce** Subscription plans & Product Management.
- **10. Security** SSL Certificate based Security & DoS protection.

All the provided input generates an estimation cost of 26,100\$ ($\approx 23,400\text{€}$), as shown at **Figure 7**. While it is true that this estimation cannot be fully trusted, the obtained cost is more than 3 times the previously obtained costs. It demonstrates that *MFSS* is economically feasible.

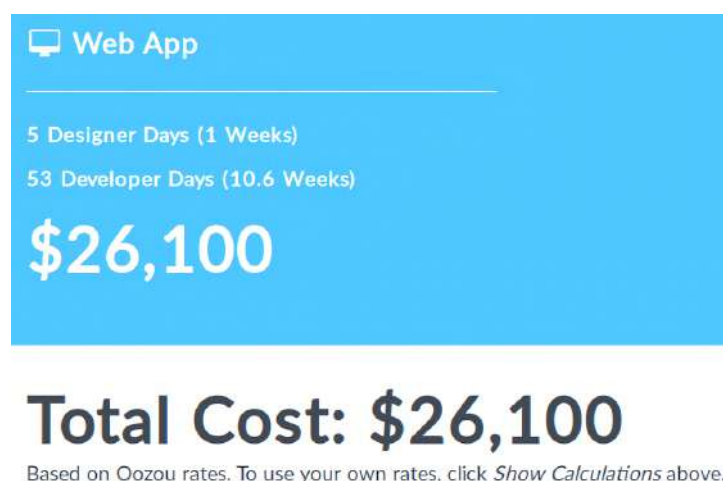


Figure 7: Estimate My App estimation cost for *MWA*.

4.5 Conclusion

The previous technical knowledge, technology, legal and economic studies have proven that the **MWA is feasible** based on the following arguments:

- Required technical knowledge is known as explained at previous technical concepts study.
- Right technologies are used as explained at previous technology study. Those technologies are studied and selected in the *Studies and decisions* section.
- European legal requirements are satisfied, as explained at previous legal requirements study.
- The resulting economic study cost is feasible, as explained at previous economic study.

5 Framework and background knowledge

This section defines essential technical concepts to understand the following studies and design, like acronyms, words, and concepts. **Table 4** shows some of these basic concepts.

| Concept | Definition |
|-----------------|---|
| OS | Acronym referred to <i>Operating System</i> , which is software that allows a user to run other applications on a computing device. |
| DB | Acronym referred to <i>Database</i> , which is a collection of information that is organized being easily accessed, managed, and updated. |
| DBMS | Acronym referred to <i>Database Management System</i> , which is a software for creating and managing databases. |
| SaaS | Acronym referred to <i>Software as a Service</i> , which is a software distribution model available to customers over the Internet. |
| MFSS | Acronym referred to the <i>Multimedia Flow Switch Server</i> . |
| MWA | Acronym referred to <i>Meeting Web-based Application</i> . This service includes the MFSS together with the SaaS-based application, being a complete Multimedia Flow Switch Service. |
| VM | Acronym referred to <i>Virtual Machine</i> , which is an emulated computer system created using software resources. It uses hardware resources like CPU, RAM and disk storage but is isolated from other software on the computer. |
| Daemon | A daemon is a software process that runs in the background. |
| REST API | A REST API is an Application Program Interface that uses HTTP requests to GET, PUT, POST, and DELETE data. A REST API is based on REST technology, an architectural style, and an approach for communications often used in web services development. |
| SQL DB | SQL is a communication standard programming language used in SQL databases for data manipulating and querying. SQL databases are relational model based databases based on predefined schemas and relations. |
| NoSQL DB | NoSQL databases have dynamic schemas for unstructured data, storing data in many ways, adding flexibility. |

Table 4: Basic concepts definition table.

As the whole application programming languages depend on future analysis and are still unknown, this section keeps these decisions as pending. Programming language decisions are exposed in the *Studies and decisions* section.

5.1 Concepts definition

This section shows other more advanced concepts necessary to understand all studies, analysis, and implementations.

Version control system

It is helpful to use a Version Control System (VCS) at software projects, and most especially when the software is developed by teams. VCS are usually stand-alone applications that add revision control support to projects. Revision control allows reverting documents to previous versions correcting possible mistakes. In the case of multiple developers working on parallel, these systems are necessary to ensure there are no code conflicts between them.

One of the most popular VCS is Git [10] [11]. Git is an open-source distributed VCS, which means that it has a remote repository stored at one server and a local repository stored in each local developer computer. A developer could update the remote repository when developed a new feature or corrected some bugs, and all the other project participants can update their local repositories from the remote one. **Table 5** shows the basic Git vocabulary definitions essential for the *Studies and decisions* section understanding. This vocabulary can be seen graphically in **Figure 8**.

| Concept | Definition |
|-------------------|---|
| Repository | A repository is frequently an appropriately named directory with a .git suffix. All of the Git administrative and control files would be present in the hidden .git sub-directory. |
| Clone | A local clone of a remote repository. |
| Branch | It is defined as an active line of development. A project might have an arbitrary number of branches, although the working tree is associated with just one of them, named <i>current</i> branch. |
| Master | The main project branch. All other branches must hang from it. |
| Commit | It represents a project state at a specific time. Each commit is followed by a title and body message that explain the state changes compared with the previous one. |
| Fetch | Fetching a branch means to get a branch description update from a remote repository. |
| Pull | Pulling a branch means to fetch and merge it. |
| Push | Pushing a branch means to pull the remote one, merge it with local commits and after push it to the remote repository. |

Table 5: Git vocabulary definitions.

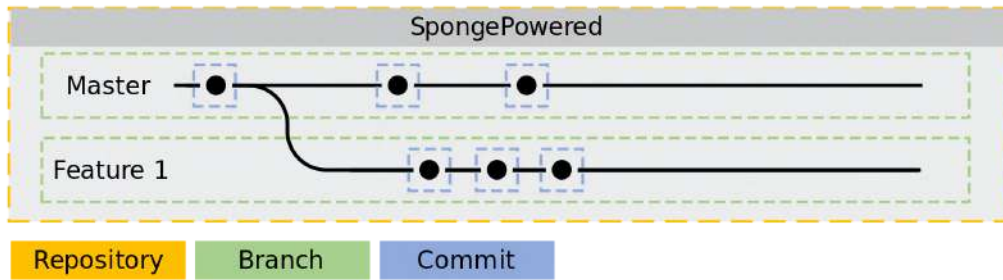


Figure 8: Git concepts graphical view.

Remote connections

As *MWA* is implemented based on the *SaaS* model, the final application has to be hosted on a remote server. Therefore, it is necessary to briefly explain how to manage remote servers, and the most popular way is using a Secure Communication Protocol.

Secure Communication protocols allow remote encrypted connections to be able to communicate with other computers over insecure networks. The most popular communication protocol is the Secure Shell (*SSH*), a software package that enables secure shell file transfers [12]. **Figure 9** illustrates graphically the Secure Shell protocol.

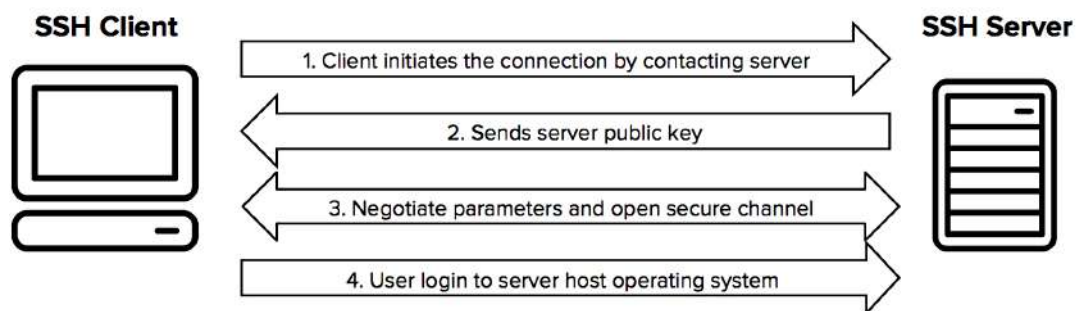


Figure 9: Secure Shell protocol graphical view.

Source code editor

Another significant aspect is to define a source code editor, as the chosen one is the backbone for both code and report writing. While programming languages are still undefined, the source code editor is a conceptual framework that enables code writing despite the language used. Therefore, the code is written with a source code editor and after compiled and executed with the proper language tools.

Usually, source code editors are composed of a comprehensive suite of editing and debugging tools, and lightweight integration with other services. An ideal option has to be chosen in the *Studies and decisions* section.

Latex

Latex [14] is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents although it is used for almost any form of publishing.

All the project is being written using Latex as a result of previous experience in research papers. It is necessary for the testing environment to have a source code editor for report writing and Latex software and libraries for report compilation and visualization.

Some report diagrams and graphs are created through the Latex *pgfgantt* [15] package. All project tables are created using LibreOffice Writer [16].

5.2 Environments definition

This project is formed by testing and production environments. Thanks to BCDS research group IT equipment, many virtualized servers and desktop computers are available to be included in project environments.

5.2.1 Testing environment

The initial testing environment preparation is the first sub-project, defined as **S1** sub-project. This environment is the development scope where the software is developed and tested.

This environment could be installed and replied at different computers, and it is possible to have multiple synchronized computers (e.g., home computer, netbook, work computer). For this reason, the private BCDS GitLab Git service is used to synchronize versions easily in all testing computers. GitLab is a Git service that offers a free and open version of its software, that could be installed inside a business and education infrastructures such as BCDS. Consequently, the project is stored with no space limitations.

Figure 10 illustrates graphically the testing environment. The testing environment could be implemented under different OS distributions, although they must meet at least the following requirements:

- **Internet connection:** An internet connection is necessary to research information, consult dictionaries, and download software libraries.
- **Web browser:** One of the sub-projects defined in the *Planning* section is the *SaaS-based application: Design and development S6* sub-project. It is essential to have a JavaScript compatible browser to test the application under development.
- **Webcam:** At least one webcam is necessary to test the *MFSS*.
- **Git client:** A git client software is necessary to work with remote BCDS GitLab code and report repositories.

- **Source code editor:** A source code editor is required for both code development and reports writing. It would be better if all testing environments use the same editor.
- **Latex software and packages:** It is necessary to install the Latex software and libraries to all testing environment computers to compile the report and visualize the resulting document.
- **Code compilers/runners:** Code compilers and runners are essential for code testing during the development. Those have to be replied in all testing computers and differ depending on future technologies and programming language choices.
- **Other devices with a browser, webcam, and microphone:** Other devices are expected to test some application functionalities, such as room conferencing. These devices would access the application through a browser inside a local LAN network, and use a webcam and a microphone to participate in room conferencing.
- **OS distributions:** The host OS distribution must be Linux, Mac OS, or Windows. They have to be desktop based distributions to be able to run Latex and use most of the source code editors, compilers, and browsers.

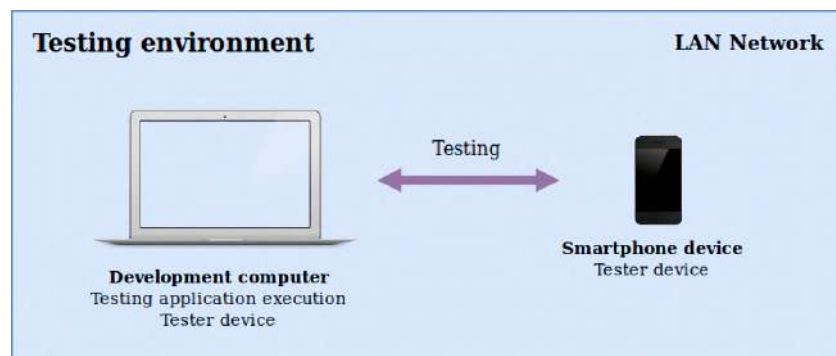


Figure 10: Testing environment.

5.2.2 Production environment

The production environment preparation is a part of **S8** sub-project. This environment is the production service scope where the Meetings Web-based Application *MWA* is maintained after the deployment.

The environment could be installed and replied, although *MWA* is deployed only in one server. Each time a new stable version of *MWA* becomes available, all changes are pushed to the project Git repository, and then the production server automatically updates, compiles, and executes the code.

Figure 11 illustrates graphically the production environment. The production environment could be implemented under different OS distributions, although they must meet at least the following requirements:

- **Internet connection:** An internet connection is necessary to enable remote connections for maintenance purposes, software and security updates, and client connections to serve and make use of the software.
- **Git:** Git client software is necessary to work with the project remote Git repository to receive *MWA* updates.
- **Code compilers/runners:** It is essential to have code compilers and runners for an efficient production code generation, execution, and deployment. These code compilers and runners differ depending on future technologies and programming language choices.
- **Daemon managers:** The host must guarantee the software availability and failure recovery. In the case of a software crash or system restart, it is necessary to configure a trusted daemon system that can restore the software automatically.
- **Security considerations:** As the production server hosts the *MWA* it is crucial to take the required legal actions, and reliable methods to ensure proper data security (e.g., integration, encryption).
- **OS distributions:** The host OS distribution must be Linux or Windows and server-based (without GUI), which serves necessary files and execute the *MWA* optimally.

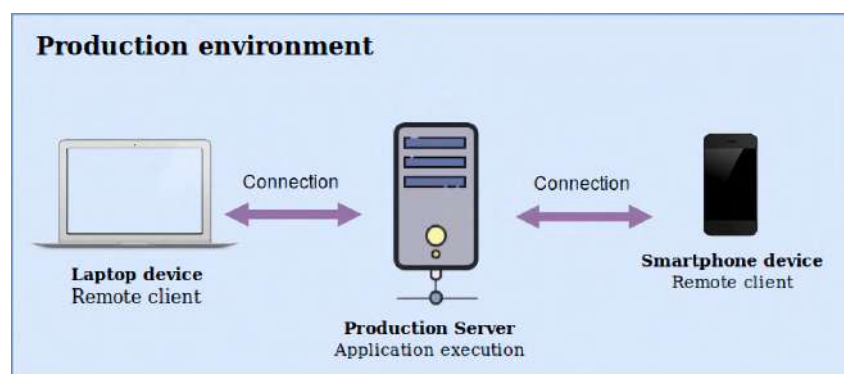


Figure 11: Production environment.

5.3 Project associates

The following description shows the project-related people team:

Dr. Jose Luis Marzo Lazaro

End-of-degree project supervisor.

Jose L Marzo is a professor at the Department of Computer Architecture and Technology at Girona University and leads the BCDS research group [17].

David Martínez Álvarez

Computer engineer student at Girona University, as the project owner.

6 System requirements

System requirements are defined according to project objectives and the desired final environment. As the *Project objectives* section explains, the project objectives are:

1. Exploring and studying modern and efficient technologies for use in the *MFSS*.
2. Finding *MFSS* core tools that fill the project purpose or implementing a new one.
3. Developing a simple and functional application (*MWA*) *SaaS*-based to provide *MFSS* meetings services to those companies who want it.
4. Deciding some commercial aspects of *MWA*, such as names, value proposition, and service plans.
5. Providing understandable code implementations and complete documentation.

The final environment represents a scenario generated with the objectives. System requirements have to be defined to achieve the definitive environment, grouped into functional and non-functional ones. This section contains only the requirements definition, as they are analyzed later with more detail in the *System requirements analysis* section.

6.1 Functional requirements

The functional requirements define what the system can or cannot do. They are enumerated as **FR-x**, where **x** is the requirement number:

- FR-1** The server has to switch and forward multimedia flows. It receives multiple media streams and then decides which of those media streams has to be sent to which system agent.
- FR-2** The server can perform Multimedia features, such as room conferencing through peers.
- FR-3** The server can establish connections with all agents using socket-like protocols.
- FR-4** The system shall block or restrict users from access to certain Multimedia features if their company service plans do not allow it.
- FR-5** The system shall block unauthorized users from access to their companies confidential data, settings, and management.
- FR-6** A room closes after a few seconds of all participants left.
- FR-7** In-room users have access to an invitation key.
- FR-8** It is not possible to erase previous recordings. However, they are automatically deleted after some time.
- FR-9** Any external company interested in joining the community can apply for a service plan through the web interface.

- FR-10** Companies are capable of creating and managing as many users as their service plan allows.
- FR-11** Companies can manage their user's authorization levels.
- FR-12** It is essential for users to verify their accounts after creation. A mechanism allow users to set their credentials and accept the terms of use and privacy policy.
- FR-13** Authentication is required. The system requires users to provide credentials to log in.
- FR-14** Tokens allow users to keep logged sessions.
- FR-15** Users can manage their profiles.
- FR-16** Users can use all the Multimedia features allowed by their companies service plans.
- FR-17** Authorized users can manage their companies providing confidential data, settings, and management.
- FR-18** Authorized users from system authorized companies would be able to create new companies and manage their service plans.
- FR-19** Users can create rooms providing some room settings.
- FR-20** Users can visualize their room activity history of created and participated sessions.
- FR-21** Users must provide at least the invitation key to join a room.
- FR-22** To access Restricted rooms, users must provide the invitation key but must belong to the same company as the room creator user.
- FR-23** Inside a room, users can change their video webcams at any moment.
- FR-24** Users can manage recording settings (type and set authorization) if their company service plan allows it.
- FR-25** Users can access an open record if they have participated in it. They can access a restricted record if the creator user is from the same company.
- FR-26** Authorized users can access to all recordings of their company users.
- FR-27** Authorized users can change their companies service plans through the web interface.
- FR-28** Authorized users from system authorized companies can modify service plans templates.

6.2 Non-functional requirements

The non-functional requirements define how the system should be. They are enumerated as **NFR-x**, where **x** is the requirement number:

NFR-1 Via an analysis of some marketing aspects, such as names, logos, and service plans, the system has to be more prepared for the production launch.

NFR-2 A business analysis show if the application is commercially feasible.

NFR-3 The project does not have to contemplate any payment logic.

NFR-4 The system has three service plans that are named as the market study suggests, ordered from lower to higher services.

NFR-5 The server must work with limited hardware resources and be scalable.

NFR-6 The server API must be developed following API REST standards.

NFR-7 All data (e.g., companies, users, and recordings) must be stored correctly. Sensitive information, like passwords or classified meeting recordings, must be secured using cryptographic procedures.

NFR-8 In case of a security breach and sensitive information exposure, the system must guarantee that the time needed to extract the sensible data are longer than the information life span.

NFR-9 All data must remain intact in case of failure.

NFR-10 The web interface has to be cross-platform following a responsive design.

NFR-11 The web interface design must be user-friendly and easy to use.

NFR-12 The system is required to have a 99.99% availability.

NFR-13 The service has to be suitable and easy to install.

NFR-14 The application code has to follow the chosen programming language practices.

NFR-15 Minimal and clear application documentation must be provided.

NFR-16 The application must fulfill all Spanish and European legal requirements.

7 Studies and decisions

7.1 Hardware

With the working environments exposed in the *Environments definition* section, it is necessary to specify which hardware is used.

The following description shows the three computers used in the development phase in which the testing environment is installed and their respective principal hardware resources:

Home computer

Tower desktop computer at home. It has a *Intel Core i5-7600K* CPU, 16GB RAM and 1TB free space. It is under a 1000MB optical network.

Notebook computer

Xiaomi Notebook Pro computer. It has a *Intel Core i5-8250U* CPU, 8GB RAM and 150GB free space. It is commonly under wireless networks, being used in many places.

Work computer

Tower desktop computer at *BCDS* research group. It is under a 100MB network.

For deployment, a virtual server is prepared, and its resources are configured according to production software requirements, explained in the *Deployment and results* section.

7.2 Multimedia Flow Switch Server

This section analyzes all the *MFSS* related studies and decisions. This study matches with the technology study for *MFSS S2* sub-project, performed after the testing environment preparation *S1* sub-project.

7.2.1 WebRTC Technology

Web Real-Time Communication[19], named as WebRTC, is a technology developed by Google and released in May 2011. This technology supports browser-to-browser communications for voice calling, video chat, and peer-to-peer (P2P) sharing without the need for internal or external plugins. Currently, Chrome, Firefox, and Safari browsers have native WebRTC support, via simple APIs.

Before WebRTC, there was no free, high quality, complete solutions available which enable this somewhat meaningful browser communication. WebRTC does not need a server to perform a multimedia communication, and it could be directly sent via P2P. The problem here is that a large number of user computers are under firewalls, making P2P connections hard to succeed.

WebRTC technology is widely used for performing peer-to-peer real-time multimedia transfers, following a P2P low-level communication architecture. This P2P architecture can be materialized through direct peer connections or with an intermediary server, which enables additional advantages.

WebRTC Advantages

- **Powerful:** It is simple to add WebRTC to websites and mobile applications compared to other existing technologies.
- **Open source and active:** Most of the top technology companies (e.g., Google, Mozilla, Apple) are actively contributing and working together to keep this project alive, correcting bugs, or developing new functionalities.
- **Video, audio, and data:** WebRTC not only provides a video and audio channel. Moreover, it creates a data channel.
- **Trusted:** WebRTC is trusted by the Internet community and is frequently evolving and improving. It has a strong dedication over the last 4 or 5 years.

WebRTC Disadvantages

- **Scalability:** WebRTC is efficient in peer-to-peer communications with no intermediary servers, although it is natively not scalable. The problem occurs considering multiple video calls scenarios, which is this project case.
- **Complexity:** WebRTC has a large number of complex definitions and protocols, making it rather difficult to understand, especially for newcomers. Different browsers, codecs, plugins, ICE, SDP, and other acronyms are some of those definitions, making it difficult to understand. The most used codecs in WebRTC are H264 or VP8, although there is not a single answer. There are more different codecs options depending on the browser and its WebRTC implementation.
- **Browser dependency:** Although WebRTC is available at most of the browsers, it is not available at less used ones. It is mandatory to have WebRTC implementation and regular maintenance to be up to date frequently, or at least a trusted external plugin.

All network communication technologies suffer from security concerns, and WebRTC is not an exception. Currently, this technology is known as the most secure one. However, it is imperative to be aware of what security and privacy issues can affect WebRTC.

7.2.2 NAT problem

The IPv4 Internet addresses are still present, and it is unknown when it will be replaced by the new IPv6. While IPv4 present, WebRTC technology has to deal with the NAT problem.

NAT stands for Network Address Translation. It is a widely used mechanism for preserving IPv4 addresses and giving local network admins control of their local topology. Many NAT techniques exist, although the most common NAT device works by changing IP header information of packets passing through it.

In many home routers, the NAT techniques are complemented with firewalls, providing security and preserving IPv4 address space at the same time. This causes problems for WebRTC, which attempts to communicate peer-to-peer. The most used NAT technique is the Symmetric NAT.

A host on the LAN can send a packet to a host on the Internet. However, it is not possible to receive a packet from the Internet, trying to reach the LAN host. In other words, the NAT device blocks the packet from passing through. If no packets could reach from the Internet to a host on the LAN, WebRTC and any other IP based communication would not work.

Essentially, the communication must be initiated by the host on the LAN. NAT process is applied at the packets proceeding through the NAT device. The Symmetric NAT device changes the source IP and port of the packet, to a new source IP and port. It saves this binding between the *inside* source IP and port, and the *outside* source IP and port. This is known as a NAT binding, stored in a table at the NAT device.

NAT makes peer-to-peer connections impossible. The client inside a NAT network thinks it is reachable on one address. However, others cannot establish a connection with it, as it is blocked by the NAT device.

The Interactive Connectivity Establishment (ICE) is used by WebRTC to solve the NAT problem [20]. ICE is a framework that allows WebRTC to overcome the complexities of real-world networking. ICE can find the best path to connect peers. It may be able to do that with a direct connection between the clients. However, it works for clients where a direct attachment is not possible (e.g., NAT networks).

In the case of Asymmetric NAT (less common), ICE uses a STUN (Session Traversal Utilities for NAT) server. A STUN server allows clients to discover their public IP address and the type of NAT they are behind. This information is used to establish a media connection. The STUN protocol is defined in RFC 3489. In most cases, a STUN server is only used during the connection setup, and after an established connection, the media flows directly between clients.

If a STUN server cannot establish the connection, ICE can use TURN. Traversal Using Relay NAT (TURN) is an extension to STUN that allows Symmetric NATs to work with WebRTC. A TURN server remains in the media path after the connection has been established, relaying the media between the WebRTC peers. **Figure 12** illustrates the ICE process in case of a client under Asymmetric NAT (STUN) and Symmetric NAT (STUN + TURN).

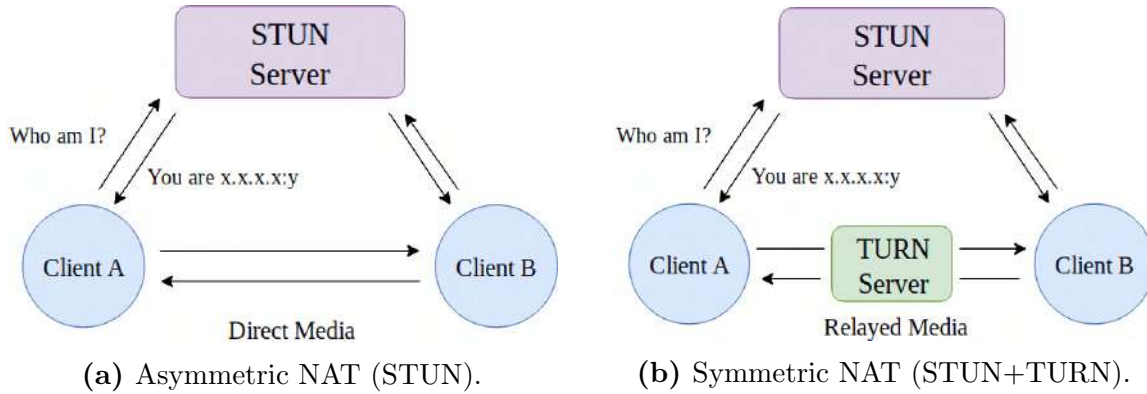


Figure 12: ICE process cases.

7.2.3 WebRTC Media Server

Scalability in peer-to-peer WebRTC communications is not a problem, although it is in a multiple participant conference. While it is true that it is tolerable to hold multiple peer calls using peer-to-peer communications, it stops being practical as the number of participants increase. In this case, a single participant has to send local multimedia flow streams (video, audio, or data) to every participant, and receive all remote streams from the other ones.

Generally, it is considered that a peer-to-peer meeting conference starts becoming unusable in more than a five participants conference, even though it depends on the participant's bandwidth. Here is where WebRTC Media Servers can highlight as they help reduce the number of sending participant streams, usually to one. They could even decrease the volume of received streams from remote participants (depending on the media server capabilities).

There are mainly three architecture types of WebRTC communication Media Servers: P2P (peer-to-peer), SFU (Selective Forward Unit), and MCU (Multipoint Control Unit).

In multi conferences, using P2P communications with each remote participant (from now R_p) result in $2 * R_p$ flows (Sending R_p and receiving R_p). With an SFU or MCU Media Server in action, all these scalability problems are drastically reduced by, at least, a half. If the selected server is an SFU model, then the client only needs to send one flow and receive R_p ($R_p + 1$ total flows). If the selected server is an MCU model, then the client only needs to send one flow and receive another one. However, the server has to multiplex all remote flows to one and send it to the participant, which implies more additional load on the server.

For example, it could be considered a 4 clients meeting case, shown in **Figure 13**. With the P2P model, it is mandatory to have $2 * R_p = 6$ open flows for each client. With the SFU model, this number could be reduced by half, having $R_p + 1 = 4$ open flows for each client. Finally, the MCU model could reduce these flows to two (send and receive ones) for each client.

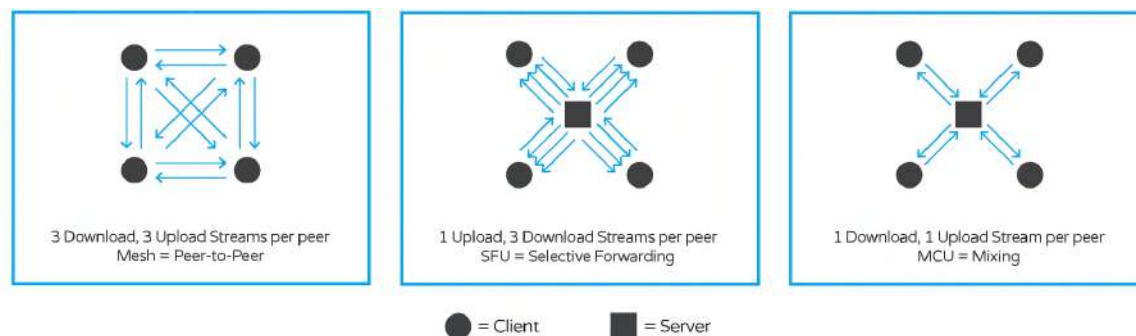


Figure 13: WebRTC Media Server architectures: P2P, SFU and MCU.

Although the MCU model seems better in scalability terms, it could expose the server to have substantial loads and sizable hardware technical requirements. As client requirements explicitly enhance the throughput-simplicity ratio, and Internet speed connections are regularly progressing, the SFU model fits better with this project requirements.

7.2.4 Decisions

The meeting functionality is the most relevant customer requirement. The previous studies result in the SFU model to be the chosen technology for the *MFSS*, completing the **S2** sub-project. Using a WebRTC Media Server provides all the previously describes advantages. Besides, the NAT problem is no longer a problem, and the *MFSS* does not have to use ICE solutions (e.g., stun, turn). Now, it is necessary to perform an analysis of existing free or open frameworks that implement an SFU architecture to integrate it later at the Multimedia Flow Switch Server *MFSS S3a* sub-project.

Many existent frameworks implement WebRTC Media Servers in most programming environments (e.g., C, node.js, php). The server has to be simple and scalable, which implies that it has to perform its communications efficiently for a meeting environment.

There are many media servers implementations, providing some functionalities that almost a regular user uncommonly uses. These functionalities only slow down the server throughput and do not contribute a noticeable benefit. In the most common cases ($\approx 95\%$), the server only needs to offer these following functionalities:

- **Signaling protocol:** The server needs to establish peer-to-peer connections with remote participants. A signaling protocol is required to start those connections and manage them.
- **Media flow handler:** The handler is the core of WebRTC Media Servers. It contains the flow switch logic.
- **Ability to record sessions:** Another relevant customer requirement is to record meeting sessions. It is necessary to manage session flows and redirect them to a multimedia endpoint, which processes them later.

It is essential to choose an ideal option according to customer requirements and previous studies. The following list shows some of the best and most used frameworks that perform WebRTC media Servers [21], being its main features compared in **Table 6**:

Intel Collaboration Suite for WebRTC: Include server SDKs (SFU and MCU) and client SDKs (JS, Android, iOS, and Windows). The MCU/SFU is Licode-based. The main problem of this technology is that it is not Open Source and is not possible to modify any server behavior.

Janus: Janus core is only a WebRTC gateway. It is not possible to achieve MCU or SFU itself, but a large number of packets and plugins can be added to expand its functionalities, being able to implement an SFU, MCU, and recording.

Licode: Originally it was an MCU, although it can now behave as an SFU. It was used by Intel as a base of their Intel Collaboration Suite core and is implemented all in the C++ programming language.

MediaSoup: MediaSoup [22] is one of the newest open-source technologies, and it is all implemented in C language wrapped in JS. This is the unique option exposed that does not natively support the record functionality itself. Since the last version launch, it is easier to redirect flows to a multimedia endpoint for external processing. This flexibility opens many options, being possible to choose any media manipulation software.

| Name | SFU | MCU | Video rec | Audio rec | OSS | License | Comments |
|---|-----|-----|-----------|-----------|-----|-------------------|--|
| Intel Collaboration Suite for WebRTC | ✓ | ✓ | ✓ | ✓ | | <i>Free</i> | Originally based on licode. |
| Janus | ✓ | ✓ | ✓ | ✓ | ✓ | <i>GPL v3</i> | Janus has a plugin model allowing it to function as MCU or SFU. Slack uses it. |
| Jitsi VideoBridge | ✓ | | ✓ | ✓ | ✓ | <i>Apache2</i> | Used by <i>Atlassian</i> and <i>talky.io</i> . Most advanced simulcast support. |
| Kurento | ✓ | ✓ | ✓ | ✓ | ✓ | <i>Apache2</i> | Stand-alone server. It can be used in the cloud with <i>elasticRTC</i> or <i>Nubomedia</i> . |
| Licode/Erizo | ✓ | ✓ | ✓ | ✓ | ✓ | <i>MIT</i> | Originally an MCU, the option to run as an SFU was added later. |
| MediaSoup | ✓ | | | | ✓ | <i>ISC</i> | New approach. An SFU all in JavaScript using node.js. |
| Medooze | | ✓ | ✓ | ✓ | ✓ | <i>GPL v2</i> | Old school MCU focused on <i>SIP interop</i> . |
| PowerMedia XMS | | ✓ | ✓ | ✓ | | <i>Commercial</i> | By dialogic. Used by <i>Softbank</i> in Japan. |
| SORA | ✓ | | ✓ | ✓ | | <i>Commercial</i> | Made by <i>Shiguredo</i> in Japan. |

Table 6: Main WebRTC media servers frameworks.

According to the studied options, an ideal solution is the MediaSoup SFU. It provides a simple JavaScript API that can be used in node.js for server-side as well as an intuitive JavaScript API for client-side, named as mediasoup-client. The multimedia flow switch is implemented using the C language, which is more efficient than other pure JavaScript libraries.

7.2.5 MediaSoup SFU Server

Before starting any implementation, it is essential to analyze and understand how MediaSoup works.

MediaSoup is developed and maintained by Versatica organization, formed by two Spaniard developers:

- **Iñaki Baz Castillo**, as the main project contributor.
- **José Luis Millán**, as a project contributor.

MediaSoup SFU receives multimedia streams from every participant in a meeting and relays them to everyone else (flow forwarding). This design leads to better throughput, performance, and less latency compared to other SFU or MCU existent implementations. It is highly scalable and requires fewer resources. The striking point about separately receiving other participants media is that they can have a personalized layout, being possible to choose which stream to render and how to display them.

According to MediaSoup official website, the main goals of MediaSoup SFU are the following:

- Be a WebRTC SFU.
- Be a node.js module in the server-side.
- Be a tiny SDK in the client-side.
- Be minimalist just handling the media layer.
- Expose modern and updated APIs in both client-side and server-side.
- Support all current WebRTC browsers.

Other arguments for MediaSoup choice are that (unlike other existing SFU) it is a module that can be integrated into a bigger application, that is accurately what this project needs. It has other significant technical features like:

- Many multimedia streams over a single ICE and DTLS transport.
- All connections are IPv6 ready.
- The WebRTC connections can be performed over UDP and TCP transmission protocols.

As said previously, MediaSoup has an agile team that periodically launches new features and correct bugs. The MediaSoup **v3** has been launched recently and in parallel with this project elaboration, on 1st May 2019. The server is developed with the new **v3** version that includes a large number of new features and performance improvements.

In order to understand MediaSoup SFU server architecture, **Table 7** shows all the necessary vocabulary.

| Concept | Definition |
|---------------------------|---|
| Worker | A worker represents a MediaSoup C++ sub-process that runs in a single CPU core and handles router instances. |
| Router | A router enables injection, selection, and forwarding of multimedia streams through Transport instances created on it. It has the semantic concept of a meeting room, although it is covered at low-level (e.g., a meeting room could involve various routers at different workers even in different physical hosts). |
| Transport | A transport connects an endpoint with a router, enabling the multimedia transmission in both directions using producer and consumer instances created on it. It can also connect routers. |
| Producer | A producer represents an audio or video source being injected into a MediaSoup router. It is created on top of a transport that defines how the media packets are carried. |
| Consumer | A consumer represents an audio or video source, being forwarded from a MediaSoup router to an endpoint. It is created on top of transport, which defines the media packets carriage. |
| Broadcaster | It is a specific use case of a producer. A producer can be a regular user or an external media producer. This opens the possibility to create streaming rooms. |
| Recording endpoint | It is a specific use case of a consumer. A consumer can be a regular user or an external endpoint that processes audio and video. This opens the possibility to create an external record system. |

Table 7: MediaSoup vocabulary definitions.

Figure 14 illustrates an example of MediaSoup SFU server architecture. The first worker hosts router 1 and router 2. Router 1 has one producer participant producing video and audio, and two consumers, one participant that consumes audio and video from the router, and a recording endpoint that consumes audio. Router 2 has one video producer broadcaster. As said previously, workers can transfer multimedia flows between them, enabling fully scalable designs. Router 2 transmits its video to router 3 and router 4 (A transport consumes audio from router 2 to router 3, and another one from router 2 to router 4). Then, both routers 3 and 4 (from workers 2 and 3) have two participants that consume video from their respective routers. This design opens many scalability possibilities, being able to divide or join routers with at execution time depending on the meeting hardware demands.

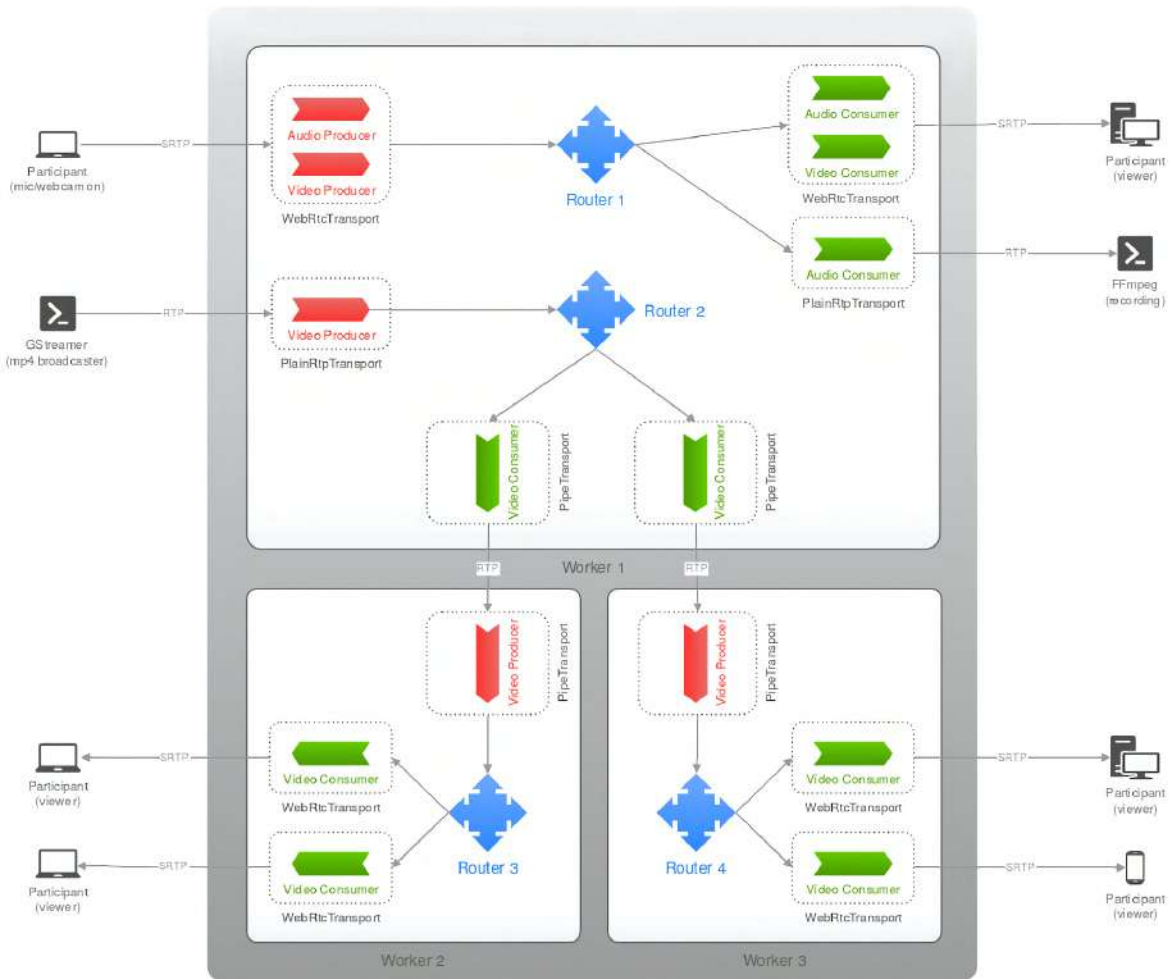


Figure 14: A MediaSoup architecture representation example (3 workers and 4 routers).

After internal discussions with the development team, **v3** roadmap was defined to improve almost all MediaSoup code, functionalities, and usage. The following points compare MediaSoup **v2** with **v3**, to learn more about how MediaSoup works:

- MediaSoup **v2** is based on peers. A peer is just a wrapper that holds transports, producers, and consumers. These peers are artificial and exist at node.js level, although they do not exist at the C++ level. Those peers require unnecessary data structures that provide no real benefit. In **v3** version, those peers are not still present, and it is up to the application to manage the concept of peers if required.
- MediaSoup **v2** depends on opaque messages (called MEDIASOUP_PROTOCOL) generated by both client-side and server-side that must be blindly exchanged between both without the application being aware of what those messages mean. In **v3** there are no longer these messages and provides real JS API calls for both mediasoup-client and mediasoup to create and manage transport, producers and consumers.

- MediaSoup **v3** makes possible to listening for RTP packets in multiple IPv4 and IPv6 addresses, making possible to run different transports at different IPs.
- It is possible in **v3** to easily inject/extract external/internal multimedia from sources/end-points such as ffmpeg, gstreamer, and more without requiring mediasoup-client.
- Each MediaSoup **v2** room simply uses a single worker and a single CPU. This works well with hundreds of participants, although it does not scale well for broadcasting scenarios with few producers and many (> 500) consumers. MediaSoup **v3** makes broadcasting scenarios possible by allowing a room to use N media worker sub-processes.
- In **v3** rooms changed their name to routers.
- MediaSoup **v3** implements sender-side bandwidth estimation to automatically accommodating the total transport bitrate to the bandwidth available in the receiver endpoint.

It is essential to test MediaSoup software to guarantee its correct behavior and efficiency. The same MediaSoup development team has a demo application example [23] available for testing as an additional component. After some code analysis and configuration testing, software behavior and efficiency meet the expectations. Some parts of this demo application are used as a reference for the Multimedia Flow Switch Server *MFSS* implementation.

7.2.6 Software libraries

Table 8 shows all libraries used in the *MFSS*. Basically, it needs MediaSoup required libraries and some package to work with system free ports.

| Library | Definition |
|----------------------|--|
| mediasoup | Cutting Edge WebRTC Video Conferencing. It contains the main MFSS logic used to manage multimedia flows. |
| protoo-server | It is a minimalist and extensible signaling framework for multi-party RTC applications. It is developed by the same team as the <i>mediasoup</i> package, which recommends using this package to perform the signaling Web Socket logic. |
| awaitqueue | JavaScript utility to enqueue async tasks for Node.js and the browser. It is also developed by the same team as the <i>mediasoup</i> package, which recommends using this package to avoid concurrent issues at room creations. |
| get-port | It is used to get a random system available port and for meeting recordings, to interact with the external endpoint. This functionality is a part of the <i>MWA</i> Recording System (<i>RS</i>) module, explained in the Studies and decisions section. |

Table 8: Libraries used in the *MFSS*.

7.3 *SaaS*-based application

The Software as a Service (*SaaS*) based application studies is a part of the **S6** sub-project. It is elaborated following modern and web-based technologies. Desktop applications were the most popular software development platform at the beginning of the twenty-first century, although now they are becoming outdated. Web applications provide many benefits compared with a traditional desktop application:

1. Desktop applications have to be deployed in each client computer, and web-applications avoid that.
2. Updates and bug fixes are easier following the same argument as deployment.
3. Web applications can be accessed from anywhere and are platform-independent. It only needs a web browser with JavaScript.
4. Support and maintenance are easier following the same argument as deployment.
5. Web applications can handle more versatile user interface designs in small devices.

Web applications only have the main disadvantage of an Internet connection requirement (It does not affect the *SaaS*-based application, as WebRTC connections need Internet connectivity).

Following the most common architecture at Web applications development, it is composed by a back-end RESTful API server and a front-end client interface application. The client interface runs at clients browsers through HTML, CSS, and JavaScript, and performs API calls to communicate with the back-end.

7.3.1 Back-end

The back-end API can be developed in many programming languages like PHP, Python, or node.js. The database can be SQL or NoSQL. The back-end is implemented in node.js, mainly for the following reasons:

- *MFSS* is developed in node.js to be able to run the Mediasoup server. As later the *SaaS*-based application has to be integrated with *MFSS*, using the same programming language avoids some significant communication problems.
- Personal experience for previously developed production node.js back-end servers.
- Developing *MFSS* and *SaaS*-based application in JavaScript simplifies the whole procedure, and the same code practices can be followed. It matches with the simplicity requirement. Furthermore, it matches with the Personal Extreme Programming (PXP) methodology.

SQL and NoSQL are possible options for the database. As defined in the *Framework and background knowledge* section, the main difference is that an SQL DB uses a strict schema and SQL processing language and NoSQL ones are unstructured and usually easy to work with object modeling. This project DB uses MongoDB NoSQL DB for the following reasons:

- A NoSQL DB does not need to learn an additional programming language, such as SQL.
- NoSQL is highly scalable, works faster, and with fewer resources than SQL ones (especially for a large amount of data).
- MongoDB is currently the most used NoSQL database. It is open-source and document-oriented that fits perfectly with project customer requirements. Moreover, it uses JSON like documents to store any data, that fits perfectly with node.js.
- There are some node.js additional libraries that enable to interact with MongoDB directly with JavaScript objects, that fits perfectly with project simplicity requirement and with PXP methodology.

Once the server programming language and its database are defined, it is necessary to define which node.js additional libraries to use. **Table 9** shows all libraries used in the *SaaS*-based application back-end.

| Library | Definition |
|---------------------------|--|
| mongoose | Mongoose is an elegant MongoDB connector object modeling library for node.js. Mongoose provides a straight-forward, schema-based solution to model application data. It includes built-in type casting, validation, query building, and more features. |
| express | Express is a minimal and flexible node.js web application framework that provides a robust set of features for web and mobile applications. Creating a robust API is quick and easy. |
| helmet | Helmet allows securing Express applications by setting many HTTP headers. |
| body-parser | This library parses incoming request bodies in a Middleware before handlers, making the request body parameters easy accessible at API callbacks. |
| cookie-parser | Cookie-parser allows to header and populate an object with the request cookies. |
| jsonwebtoken | This library uses an implementation of JSON Web Tokens in node.js. This will allow managing sessions using authentication tokens. |
| bcryptjs | A library that implements an optimized bcrypt hashing algorithm 8 for node.js with no dependencies. This will be a secure method to store user passwords safely. |
| multer | Multipart/form-data handler for uploading files. It will be used for profile picture service. |
| nodemailer | Nodemailer is a module for node.js applications to allow easy email sending. The project got started back in 2010 when there was no appropriate option to send email messages, and today it is the solution most node.js users turn to by default. |
| simple-node-logger | A simple and efficient log library to beautify and send to file server logs. |

Table 9: Libraries used in *SaaS*-based back-end.

7.3.2 Front-end

The *SaaS*-based application is developed using HTML, CSS, and JavaScript. HTML language defines the webpage content, the CSS its style, and JavaScript the logic.

JavaScript is in charge of providing user interaction, API back-end interaction, and HTML content generation. JavaScript can easily interact with HTML content using the Document Object Model (DOM) [24]. When a web application is loaded, the browser creates a DOM of the page constructed as a tree of Objects, as shown in **Figure 15**. With the object model, JavaScript can create, change, and remove all HTML elements and attributes. Furthermore, it is capable to change CSS styles.

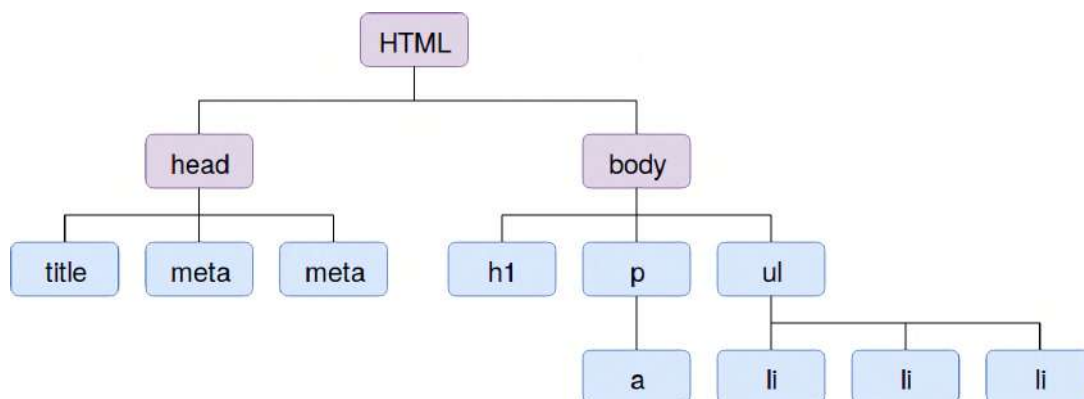


Figure 15: Example of a DOM (Document Object Model) tree.

The API logic implementation, styles, and HTML DOM interaction are parts of a complex system, impossible to implement at the scheduled time and much less in a solo project. To avoid this problem, many helper JavaScript frameworks and libraries are available and regularly increasing.

Choosing an appropriate JavaScript framework is not trivial. The following description analyzes the top 3 JavaScript frameworks of 2019 and its usage statistics[25]:

React

React is a declarative JavaScript library used for building interactive UIs. It was created by Facebook engineer Jordan Walke and launched in May 2013. It has overtaken Angular and established its dominance. Although it is not technically a Framework itself, React can create reusable user-interface components for each state that can automatically update and render data without need of reloading the page.

Usage Statistics:

- **Current websites running React:** 475K.
- **JavaScript developers that are using React:** 64.8%
- **State Of JS survey ranking:** 1st.
- **GitHub statistics:** 120.5k stars, 21k forks, and 1200 contributors.

Vue Vue is an open-source JavaScript framework that was created by an independent engineer Evan Yoy and released in 2014. The last years have increased its popularity outpacing Angular. Vue has a progressive nature that helps it to adapt to the developer's needs and offers a lightweight solution incorporating all of the best features and practices.

Usage Statistics:

- **Current websites running Vue:** 64K.
- **JavaScript developers that are using Vue:** 28.8%
- **State Of JS survey ranking:** 2nd.
- **GitHub statistics:** 125k stars, 18k forks, and 253 contributors.

Angular

Angular is a front-end web application framework based on *typescript*. Launched by google engineers in 2010 and updated in 2016, and it is the oldest modern JavaScript framework.

Usage Statistics:

- **Current websites running Angular:** 350K.
- **JavaScript developers that are using Angular:** 23.9%
- **State Of JS survey ranking:** 3rd.
- **GitHub statistics:** 44.5k stars, 11.5k forks, and 840 contributors.

Based on the description of the frameworks, the front-end is developed with React. React is chosen mainly for the following reasons:

- **Prior personal experience:** Prior experience on React project development and production deployment.
- **High-performance:** React is known for its flexibility and efficiency.
- **Abundant resources:** Maintained by a sizable company like Facebook provides substantial documentation and resources.
- **Rich community:** React JavaScript framework has now the larger developers community.
- **Popularity:** More than half of the JavaScript developers use React. It is in the first position at State Of JS survey ranking.

It is crucial to follow a design methodology to create user interfaces efficiently. A methodology has to be selected from existing ones, as they are studied and proven by design experts. The current most popular design methodologies are:

Bootstrap

This framework was born in 2012. Twitter took its social network graphic style complemented with some JQuery (Javascript DOM interaction library) features to create a solid base to begin any web design project.

Material Design

This framework was developed in 2014 by Google. It is not a framework itself, as it offers a style guide for applications. From those style definitions, a large number of implementations for different environments were born. *Material UI* is the React component implementation of Material Design.

During 2018, Google's Material design usage has been increased, and Bootstrap has been started to lose popularity. React has implementations for both Bootstrap and Material Design. Material Design is becoming popular, especially for mobile applications. For this reason, it is the chosen design methodology for the *SaaS*-based software.

At this point, enough information is set to define which JavaScript additional libraries are used at project implementation. **Table 10** shows all libraries used in *SaaS*-based application front-end with a little brief description for each one.

| Library | Definition |
|---------------------------|--|
| @material-ui/core | A react component that implements Google's Material Design. |
| @material-ui/icons | React component that hosts Google's Material Design Icons. |
| izitoast | An elegant, responsive, flexible, and lightweight notification. |
| material-table | A simple and powerful data table for React. |
| material-ui-dropzone | It provides a file-upload dropzone. |
| mediasoup-client | JavaScript client-side library for building mediasoup apps. |
| nuka-carousel | A pure React carousel component. |
| protoo-client | A minimalist and extensible node.js signaling framework. |
| query-string | Parse and stringify URL query strings. |
| react | A JavaScript library for creating user interfaces. |
| react-cookie-consent | A small, simple, and customizable cookie consent bar. |
| react-copy-to-clipboard | Copy to clipboard React component. |
| react-dom | Serves as the entry point to the DOM and server renders for React. |
| react-helmet | A React security assistant. |
| react-router-dom | DOM bindings for React Router. |
| react-scripts | Provides default react test, start, and build scripts. |
| sweetalert2 | A beautiful replacement for JavaScript's popup boxes. |
| sweetalert2-react-content | A beautiful replacement for JavaScript's popup boxes connector with react. |
| video-react | A web video player HTML5 ready React library. |

Table 10: Libraries used in *SaaS*-based application front-end.

7.4 MWA: MFSS integration with the SaaS-based application

The Meetings Web-based Application (*MWA*) is composed by the Multimedia Flow Switch Server (*MFSS*) integration with the *SaaS*-based application. It is essential to study which communication architecture would be better to use, obtaining an ideal throughput. This integration is a part of the **S7** sub-project.

The most viable form is to integrate the *MFSS* and the *SaaS*-based application API (back-end), as they are developed through Node.js programming language. It generates a new piece of software, becoming the final *MWA* back-end. All the final *MWA* application client (front-end) requests pass first through *SaaS*-based application API, which verifies authentication, authorization, and some additional parameters. In other words, if the *MWA* front-end wants to start Web Sockets communications with the *MFSS* (essential for meetings management), then it needs the API approval. **Figure 16** illustrates graphically the whole *MWA* planned architecture.

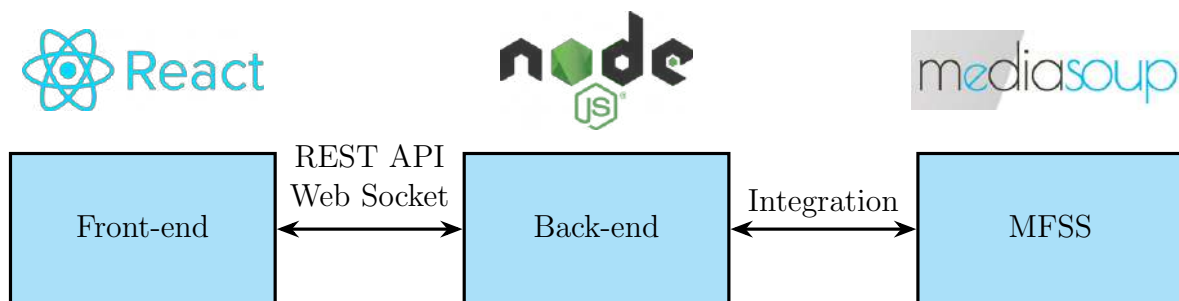


Figure 16: *MWA* planned architecture.

7.5 Git client choice & practices

Git Client software is necessary to work on multiple physical computers. One popular and flexible Git client is GitKraken, a software used to work with git repositories. The following list shows the main GitKraken features:

- Intuitive UI/UX.
- Merge conflict editor.
- Built-in code editor.
- Task tracking.
- GitHub, GitLab, Bitbucket, and Azure DevOps integration.
- Gitflow support and file history visualization.

GitKraken is not free software, although it has a limited free version. The free version can do all the actions required for this project elaboration. It can work with remote self-hosted GitLab services, being able to be linked with the BCDS GitLab service. **Figure 17** illustrates the main GitKraken user interface, with an example of this project repository, associated to BCDS GitLab.

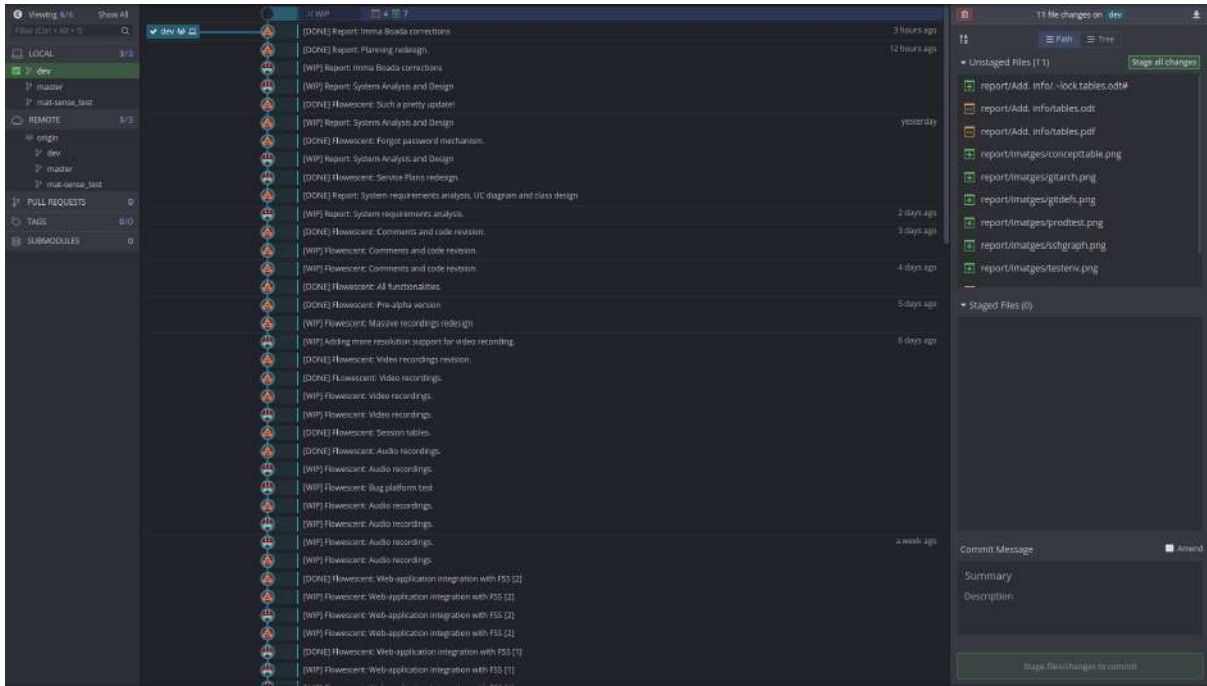


Figure 17: GitKraken project interface visualization.

The project followed Personal Extreme Programming (XP) does not specify any git practices, alleging that they are not relevant in solo projects. Consequently, neither of the most common git methodologies (based on co-working) is used, although some clear rules are used, although some clear rules are applied. The following list shows the used rules:

1. **Commit format:** When a new feature is completed, a commit composed by a title and a body is placed in the Git repository. The title and body describe the version changes. The title starts with a tag, that describes if the new version has a completed new feature (**DONE**), or it has some pending work (**WIP**). The title is followed by a short description (≈ 20 characters). A commit body message contains a detailed list of all done and pending tasks.

Figure 18 illustrates a message format example from *BCDS GitLab* repository exposing the finalization of one task.

2. **Branches distribution:** As this is a solo project, two *master* and *dev* branches are sufficient. All the development is typically performed at the *dev* branch, and each time a major feature or update is completed, the *dev* branch work is merged into *master*.

[DONE] Web-application: Root panel: Companies management

```
[ADDED] API calls: getCompanies, createNewCompany, updateCompany.
[FIX] Notifications will now be shown at top right position.
[FIX] Admin Panel is now Root Panel to avoid confusions.
[NEW] Panel Component (Root Panel). The admin user of root company can manage companies (crate, update) with restrictions.
[FIX] Added loader protection to signup component. Now it doesn't render before token verification.
[ADDED] Panel component to route /panel.
[FIX] Server routes.
[FIX] Init_db script to new schema adaption.
[ADDED] Active number (0,1) and admin_email to company schema.
[ADDED] MediaSoup dependency to package.json.
[FIX] HTTPS Initialization bug.
[ADDED] First invitation email for company admin after creation.
```

Figure 18: Commit format example.

7.6 Documentation practices

The project documentation differs from the most common documentation practices, that are mainly dedicated to team development (e.g., *Doxygen*[18] syntax, practices). The established procedure follows common sense and previous personal methods, defined by the following rules:

1. The top of each source code file must contain a descriptive comment with the following content, as exposed in **Figure 19**.
 - First line with the main project name. It is common at all files and contains exactly the text *“Multimedia Flow Switch Service®”*
 - Second line with an author copyright notice. It is common at all files and contains exactly the text *“Copyright © 2019 David Martínez - All rights reserved”*
 - Third line with a brief description of what the file code does.
2. A section title introduces the additional software libraries and files imports. It contains exactly the text *“*** LIBRARIES AND FILES IMPORTS ***”*.
3. Another section title introduces the remaining code. It contains exactly the text *“*** CODE BEGINS HERE ***”*.
4. A multi-line descriptive comment is placed at the beginning of each code function, before its description. The comment explains in a few words what the function does, with its entry parameters and return definition.
5. Before each library or file import, a one-line comment is placed to explain what the library or file does and what it is needed.
6. In most functions, the code is easily understandable, being its performed tasks easily deducted. In the case of complex code pieces, a simple line of code is placed before the piece to explain what the source code part does.

```
David, 4 days ago | 2 authors (David and others)
1 // Multimedia Flow Switch Service Software - Flowescent®
2
3 // Author: Copyright© 2019 David Martínez - All rights reserved.
4
5 // File: server.js
6 // Flowescent® main server file.
7
```

Figure 19: Top code file comment example (main server file).

7.7 Source code editor choice

As previously exposed in the *Environments definition* section, it is necessary to choose an appropriate source code editor for both programming and report writing. The following description analyzes the top 2019 editors [13]:

Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source, and cross-platform code editor developed by Microsoft. It is the most used code editor in 2018, being used by more than 34.9% of the developers. VS Code is a relative newcomer text editor (since 2015) and the most powerful and customizable one through its plugin system. It offers support to OSX, Windows, and Linux OS distributions.

Atom

Atom is a free, open-source, and cross-platform code editor that was released in 2014, now owned by Microsoft. It has a nice plugin system, although it is not at the same level as VSCode. It offers support to OSX, Windows, and Linux OS distributions.

There is another source code editor called Sublime Text, although it is not as well-known as the previous ones. It is not analyzed as is proprietary software, and free software is prioritized as said in the *Feasibility study* section.

VS Code is a lightweight and powerful source code editor, and it is half-way between a text editor and an IDE. VS Code is the project chosen source code editor mainly for the following reasons:

- It comes with built-in support for JavaScript, TypeScript, and node.js (auto completion, syntax, and debug). Those supports are helpful in Meetings Web-based Application *MWA* development.
- Its plugin system is reliable, and the plugins are generally more maintained than other source code editors ones.
- Having tried both VS Code and Atom, the first one is more fluid and efficient under the same hardware.

At this point, it is possible to define which VS Code extensions are used. **Table 11** shows the packages used name, version, and functionality.

| VS Code Package | Version | Functionality |
|---------------------------------------|---------|---|
| Beautify | 1.4.10 | Beautify JavaScript, JSON, CSS, and HTML in VS Code. |
| Code Runner | 0.9.8 | Run code snippet or code file for multiple languages. |
| Code Spell Checker | 1.7.6 | Check spell for the English language. |
| GitLens | 9.6.3 | GitLens supercharges the Git capabilities built into VS Code. |
| JavaScript (ES6) Code Snippets | -- | Contains code snippets for JavaScript in ES6 syntax for VS Code editor. |
| LaTeX Workshop | 6.5.1 | It provides all-in-one features and utilities for LaTeX typesetting with VS Code. |
| REST Client | 0.21.2 | Enables to send an HTTP request and view the response in VS Code directly. |
| Vscode-icons | 8.6.0 | It has many features, like icons customization, project auto-detection, and custom configuration. |

Table 11: VS Code packages used.

7.8 Final software used and technical requirements

The *MWA* application is developed using all the previous defined software. **Table 12** shows those software together with their respective technical requirements.

| Software | Version | Functionality | Technical Requirements |
|------------------------|---------------------|--|--|
| Mozilla Firefox | 66.0.3 | An Internet Web Browser for <i>MWA</i> application testing. | Pentium 4 or newer 512MB/2GB RAM (64-bit) 200MB free space |
| Gitkraken | 5.0.4 | Git client to easily interact with git. | Unspecified |
| VS Code | 1.33.1 | Source-code editor to implement code and report elaboration. | 1.6GHz or faster CPU 1GB RAM |
| MiKTeX TeX Live | 2.9 3.14159 2 | LaTeX implementations for Windows and Linux. | 5GB free space (full) |
| node.js | 10 | <i>MWA</i> application development process. | Unspecified |
| npm | 6.4 | | |

Table 12: All used software and their corresponding technical requirements.

The software requirements showed at **Table 12** are easily satisfied in all cases, taking into account the specified hardware in the *Testing environment* section.

7.9 Marketing study

This section performs a simple marketing study, deciding commercial Meetings Web-based Application (*MWA*) properties. This study does not follow any known methodology and is performed following a deductive analysis of some *MWA* aspects. The marketing study is one of the **S5** sub-project tasks, in particular, the **S5.1** task.

7.9.1 MWA name

The first task is to search and choose an appropriately engaging name. The name is related to the Meetings Web-based Application (*MWA*) purpose. To achieve that an online business name generator [26] has been used to generate possible names making plays on words.

The current name for the entire application is *Multimedia Flow Switch Service*, known as *MWA*. Analyzing the first name words separately, it seems that the word *flow* opens many name combinations. After *flow* word as input at the online business name generator, the results show an attractive name: *Flouescent*. It is a word composed by *flow* and *fluorescent* words, which matches perfectly. This name can be fluidly pronounced, and it is engaging. Consequently, it has been chosen as *MWA* name. From now on, the entire project application is called both *MWA* and *Flouescent*.

7.9.2 User interfaces design

The front-end user interfaces are designed using Google's Material Design guidances. The interfaces are designed at *System analysis and design* section, although as part of the marketing study, it is necessary to define the entire application standard appearance. Many open-source React Material Design templates are available to use as a start point. With some prior knowledge about React, it is possible to analyze a base template and adapt it as needed.

After a search process, the *material-sense*[27] React Material Design is the chosen Meetings Web-based Application (*MWA*) front-end template, as **Figure 20** illustrates. This is a React Material UI (React Google's Material design implementation library) simple template, showing a rich application example with wizards, charts, and ranges. This template is enough as an interface start point, and it is more lightweight than the other candidates.

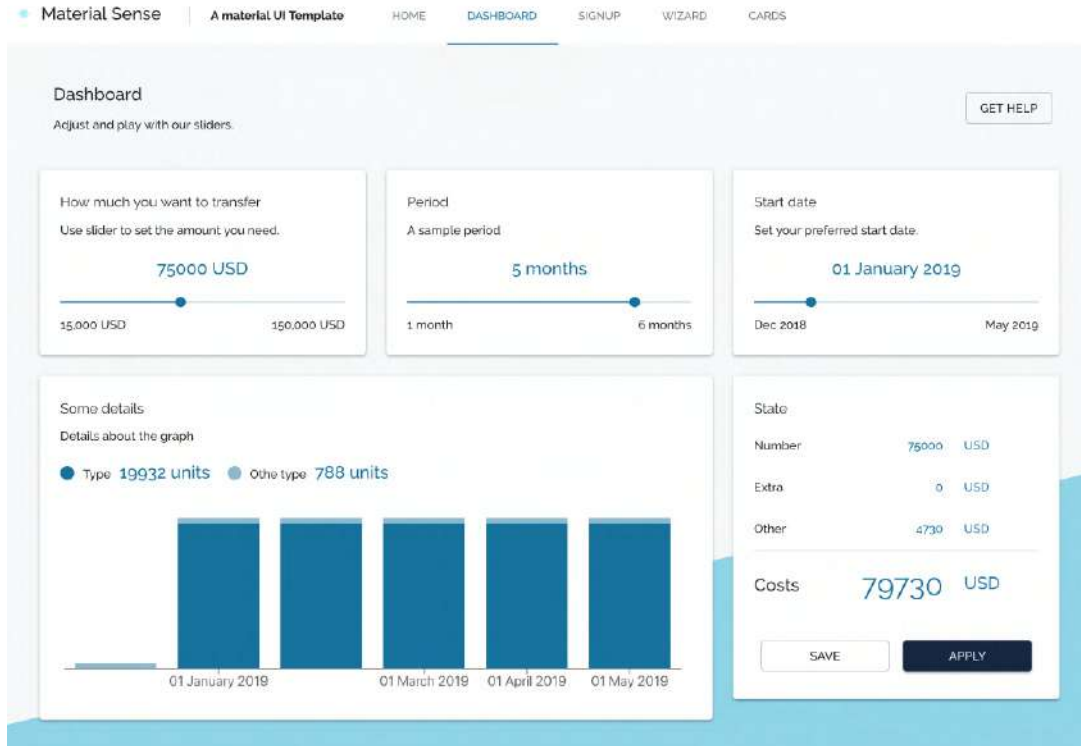


Figure 20: *material-sense* React Material UI template.

7.9.3 Logos

Logos are an essential part of a commercial product, as relevant designs typically catch more interest from customers. The principal *Flouescent* logo is defined based on the naming semantics. The objective is to design a central corporative logo and a favicon. A favicon is a little image shown at the browser page tab, together with the page title.

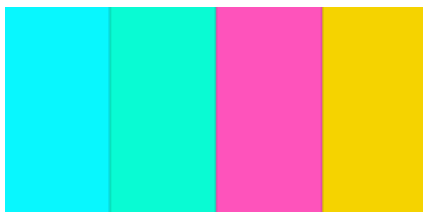
Neon lights are considerably related with *Flouescent* semantic. The bright light represents the fluorescence and tubes. The multimedia flows can be represented with some neon-based color palettes. These arguments result in a fluorescent neon colored text logo.

A neon-based font text and design software are necessary to make the logos. The design software used is Gimp [28], a free image manipulating software, licensed as freeware and developed by Spencer Kimball and Peter Mattis. The used neon-based font is Library 3 AM [29], as **Figure 21** illustrates, licensed as freeware and developed by Igor Kosinsky.

A B C D E F G H I J K
 L M N O P Q R S T U
 V W X Y Z Á Â Ã Æ
 0 1 2 3 4 5 6 7 8 9

Figure 21: *Library 3 AM* font.

It is necessary to design an appropriate dynamic neon-based color palette for applying it in the text logo, together with the *Library 3 AM* font. Inspired by the prism trend, one of 2019 creative trends is the color palette as **Figure 22a** illustrates, which is a dazzling take on popular rainbow tones. This palette should catch the viewer’s attention, although four colors are not sufficient for all the text characters. For this reason, it is adapted using middle colors. Additional color is placed between each color obtaining the definitive color palette.



(a) Original 4 colors palette.



(b) Adapted 10 colors palette.

Figure 22: *Flouescent* corporative logo colors palettes.

With all studied components applied, the **Figure 23** illustrates the resulting final *Flouescent* corporative logo.

FLOWESCENT

Figure 23: *Flouescent* corporative logo.

The first corporative logo letter *F* is user for the favicon logo, as **Figure 24** illustrates.



Figure 24: *Flouescent* favicon logo.

7.10 Business feasibility study

A feasibility study [30] provides an investigating function, addressing the question of “Is this a viable business venture?”. It helps narrow the scope of the project to identify some of the best business scenarios. The business feasibility study is one of the **S5** sub-project tasks, in particular, the **S5.2** task.

The following list provides some reasons to do a feasibility study:

- Gives focus to the project and outline alternatives.
- Narrows business alternatives.
- Identifies new opportunities through the investigative process.
- Identifies reasons not to proceed.
- Enhances the probability of success by addressing and mitigating factors early on that could affect the project.
- Provides quality information for decision making.
- Provides documentation that the business venture was thoroughly investigated.
- Helps in securing funding from lending institutions and other monetary sources.
- Helps attract equity investment.

It is important not to confuse this study with the Meetings Web-based Application (*MWA*) feasibility study. The *MWA* feasibility study only focuses on the feasibility of the software based on technical, technology, legal, and economic considerations. Instead, the business feasibility study evaluates the eventual company that may commercialize the software.

7.10.1 Value Proposition

A value proposition refers to the value that a company promises to deliver to customers should they choose to buy their product. Furthermore, a value proposition is a declaration of intent or a statement that introduces a company's brand to consumers by telling them what the company stands for, how it operated, and why it deserves their business.

As said in the *Motivations and purpose* section, *Flowescent* wants to put aside conventional face-to-face business meetings. This is the main purpose of the system and has significant commercial potential, being an ideal option to choose as a value proposition.

The following list exposes the main advantages of online meetings in front of traditional face-to-face meetings:

1. People attending the meeting **do not need to leave their work** station or wherever they are. They can attend meetings.
2. With virtual meetings, distance is not an issue. People can **take part in the conference regardless of where they are**. In other words, they can hold meetings without having to travel all the way.
3. Many people can participate in a virtual meeting since they **only have to set aside some time** for the meeting.
4. **The carbon footprint is reduced** since people do not have to travel.
5. Virtual meetings are **cheap**. Although the software service has its costs, it eventually becomes competitive because it does not have any supplementary costs (such as travel or venue cost). Only an Internet connection and the meetings software are needed, which allows everyone to attend the meeting.
6. There is **no need to spend money on hiring a venue**.
7. Technology and Internet connections are frequently improving. **Software services are continually growing** and improving its services.
8. Virtual **meetings can be fully recorded** without any extra devices or resources.

All these exposed advantages help redact the value proposition. The following list shows the main features that the application has to satisfy to obtain the maximum benefit from online video meetings advantages:

- Provide an affordable company-oriented online meeting software, as a basis.
- Offer meeting recordings functionality.
- Offer the possibility to perform meetings between different companies.
- Work on many devices as possible.

Once the advantages and features that guarantee them are set, the smartest way to design the value proposition is to point out the aspects where *Flouescent* services are superior to face-to-face meetings. After some research, the selected value proposition has been converged on the following main body:

“Save money and time”

“Flouescent is the smartest meeting solution. One click and start as many meetings as you want. Record sessions and keep an activity history. And no more money and time spent.”

Moreover, the proposition contains more descriptive parts, defined by the following points:

1. *How can I save money and time?*

“1. Join: Join our community by applying for a service plan and start managing users.”

“2. Enjoy: Create, join, and record as many meetings as you want.”

“3. Review: Check your previous sessions, display recordings, and consult activity history.”

2. *Why can I save money and time?*

“1. Less wasted time: Employees will only have to set aside the necessary time. They can participate in meetings without having to travel all the way. People attending the conference do not need to leave their work station or wherever they are.”

“2. No more venue costs: There is no longer a need to spend money on hiring a venue.”

“3. Without additional costs: No more travel or venue costs. Only a small monthly customizable tax according to each business necessities.”

What are you waiting for? Start saving now (Link to signup) and enjoy our 30 days free trial!

7.10.2 Business competitive study

Business competition is an important point to take into account this business feasibility study. It is necessary to analyze if other applications cover the previously exposed necessities and how they do it. **Table 14** shows the competitive matrix with all service plans found through a simple Internet search of the two most popular software, and which necessities cover:

| Software plans | Users | Company features | Audio meetings | Video meetings | Cross-platform | Audio record | Video record | Price (€) |
|-------------------------|-----------------------------|------------------|----------------|----------------|----------------|--------------|--------------|-----------|
| join.me LITE | Total: 5 | | ✓ | | ✓ | | | 9 |
| join.me PRO | Total: 250 | | ✓ | ✓ | ✓ | | | 17 |
| join.me BUSINESS | Total: 250 | ✓ | ✓ | ✓ | ✓ | | | 24 |
| zoom.us BASIC | Total:100 40 min limited | | ✓ | ✓ | | | | Free |
| zoom.us PRO | Total: 100 | | ✓ | ✓ | | | | 13.99 |
| zoom.us BUSINESS | Total: 100 | ✓ | ✓ | ✓ | | | | 18.99 |

Table 13: Business competitive matrix.

The competitive matrix provides enough information to decide feasible service plans for *Flouescent* application, taking into account the possible maintenance cost. **Table 14** shows the competitive matrix fulfilled with chosen *Flouescent service plans*.

| Software plans | Users | Company features | Audio meetings | Video meetings | Cross-platform | Audio record | Video record | Price (€) |
|----------------------------|-----------------------------|------------------|----------------|----------------|----------------|--------------|--------------|-----------|
| join.me LITE | Total: 5 | | ✓ | | ✓ | | | 9 |
| join.me PRO | Total: 250 | | ✓ | ✓ | ✓ | | | 17 |
| join.me BUSINESS | Total: 250 | ✓ | ✓ | ✓ | ✓ | | | 24 |
| zoom.us BASIC | Total:100 40 min limited | | ✓ | ✓ | | | | Free |
| zoom.us PRO | Total: 100 | | ✓ | ✓ | | | | 13.99 |
| zoom.us BUSINESS | Total: 100 | ✓ | ✓ | ✓ | | | | 18.99 |
| Flouescent STARTER | Max-room: 3 Total: 20 | ✓ | ✓ | ✓ | ✓ | | | 9.99 |
| Flouescent STANDARD | Max-room: 8 Total: 50 | ✓ | ✓ | ✓ | ✓ | | | 14.99 |
| Flouescent PREMIUM | Max-room: 20 Total: 1000 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 29.99 |

Table 14: Competitive matrix with *Flouescent* service plans.

Although the maintenance cost is no studied, a limited estimation needs to be done. The maintenance cost estimation is based on the selected technology to perform the *MFF*, which does not need much hardware resources. It is necessary to predict an estimated number of companies that would hire *Flouescent* services. The application has sufficient potential to gain a respectable professional market place.

The Spanish companies number [31] is chosen as a basis to approximate a company clients number. In April 2019 187,388 Spanish companies from small (10 employees) to large (more than 250 employees) size were officially registered. Assuming that only 0,001% of those companies are interested in *Flouescent* services, 187 companies are clients. Companies should be more interested in the premium plan, being a 37-50-100 an acceptable client distribution estimation for respective starter, standard, and premium plans. This data, together with previous defined pricing results on a **4,117€** predicted monthly gains, that, for sure, overcomes all possible maintenance costs.

In summary, this study ensures that *Flouescent* application commercialization is feasible. *Flouescent* is the only company that offers both audio and video recordings for its premium service plan. This feature is wide requested, and attracts clients to join *Flouescent* community (mainly for the premium package, much competitive). Furthermore, this purpose provides company features at all service plans and fully cross-platform support.

8 System analysis and design

This section contains all the analysis and design studies for the entire *Flouescent* application, including the Multimedia Flow Switch Server *MFSS*, and the *SaaS*-based application. All those definitions are related to all analysis and design tasks, included at **S4**, **S6**, and **S7** sub-projects.

Before going into detail, the following list defines some basic application vocabulary that is indispensable to understand the system and define better its requirements:

- **Flouescent Server:** As previously studied, the union of the *MFSS* (MediaSoup WebRTC SFU) and the *SaaS*-based application server.
- **Multimedia features:** Multimedia flows that could be handled by MediaSoup WebRTC SFU. Room conferencing is an example of *Multimedia feature* composed by audio and video multimedia flows that server commutes from and to all peers.
- **Company types:** System requirements distinct two types of companies, one with more privileges than the other. The companies with lower privileges are named as *Normal companies*, and the others with higher privileges are names as *Admin companies* or *Root companies*. *Admin companies* differs from *Normal companies* at the authorization level theoretically. In practice, only one *Admin company* exists, which is the whole application owner. The *Flouescent* owners may perform some administration tasks over all the client companies that are using its services.
- **User types:** The system requirements distinct between two users types, depending on their authorization. The users with lower privileges are named as *Normal users*, and the others with higher privileges are named as *Admin users*.
- **Room types:** The system requirements distinct between two room types, depending on its access restriction value. The more restricted rooms are named as *Restricted rooms* and the others as *Open rooms*.
- **Recording types:** The system requirements distinct between two recording types, depending on its access restriction value. The more restricted recordings are named as *Restricted recordings* and the others as *Open recordings*.

8.1 System architecture

This section shows more detailed information about the system architecture, which is briefly introduced in *Studies and decisions* section.

As **Figure 25** illustrates, the whole *Flouescent* architecture uses both *CS* (Client-Server) and *P2P* (peer-to-peer) communications. In other words, the front-end client application can communicate with the server through a REST API (*CS*) or Web Sockets (*P2P*). The server must implement a Web Socket Server to establish a connection through the *CS* model to avoid firewalls and NAT network problems. Once the connection is established, then the Web Sockets work as a *P2P* model, being both server and client able to notify each other.

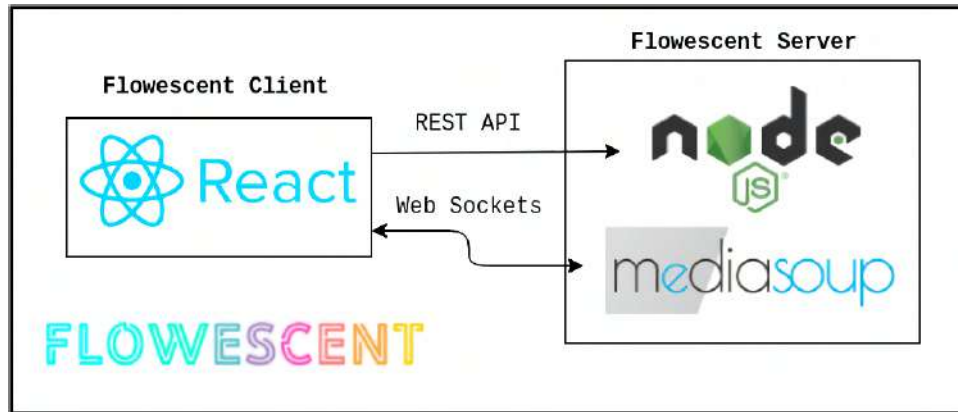


Figure 25: *Flowescent* architecture.

8.2 System requirements analysis

All system requirements are described in the *System requirements* section, although it is necessary to analyze them with all previous vocabulary and studies acquired knowledge.

8.2.1 Functional requirements analysis

The following description analyzes the system functional requirements together with an actor definition (who makes the feature or action) and a description (more detailed explanation):

FR-1 *“The server has to switch and forward multimedia flows. It receives multiple media streams and then decides which of those media streams has to be sent to which system agent.”*

Actor: *Flowescent* Server.

Description: MediaSoup server has to manage all media streams correctly implementing an SFU architecture.

FR-2 *“The server can perform Multimedia features, such as room conferencing through peers.”*

Actor: *Flowescent* Server.

Description: MediaSoup server implements an SFU architecture based on room conferencing through peers. Each peer (user inside the system) produces media streams (audio and video) to a conferencing room (meeting). Each meeting represents an independent SFU (for MediaSoup, called Router).

FR-3 *“The server can establish connections with all agents using socket-like protocols.”*

Actor: *Flowescent* Server.

Description: It is necessary for peers to establish connections with the server and vice-versa. In this case, socket web protocols are an ideal solution, which allows P2P connections (both peer and server can notify each other about configurations or changes). The *Flowescent* Socket Communication (*SC*) module is behind all this complex communications.

FR-4 *“The system shall block or restrict users from access to certain Multimedia features if their company service plans do not allow it.”*

Actor: *Flouescent* Server.

Description: The system must provide a robust mechanism to guarantee that the users do not pass their companies service plans restrictions.

FR-5 *“The system shall block unauthorized users from access to their companies confidential data, settings, and management.”*

Actor: *Flouescent* Server.

Description: The system must provide a robust mechanism to guarantee that only Admin users can manage their companies, and Normal users do not have any access.

FR-6 *“A room closes after a few seconds of all participants left.”*

Actor: *Flouescent* Server.

Description: The system must provide a robust mechanism to guarantee that the rooms close automatically after a few seconds of last peer leave.

FR-7 *“In-room users have access to an invitation key.”*

Actor: *Flouescent* Server.

Description: The system must provide a robust mechanism to guarantee that the in-room users receive an *Invitation key* or *link* associated with it to share with desired users.

FR-8 *“It is not possible to erase previous recordings. However, they are automatically deleted after some time.”*

Actor: *Flouescent* Server.

Description: Neither users nor system can manually erase a recording. They are stored for some time (actually unspecified), and after that, the system deletes it though a secure automated process.

FR-9 *“Any external company interested in joining the community can apply for a service plan through the web interface.”*

Actor: External system actor applying as company manager (already not registered to the system).

Description: A company that wants *Flouescent* services can join the community and apply for a service plan at any time. The company manager only has to select the desired service plan, and insert some valid company data (at least, a name and an email of company manager, that becomes an *Admin user* in the system after registration). It is necessary that the company manager accepts the terms of use and privacy policy according to Spanish and European regulations (mainly, the GDPR), and the web interface must provide a reliable mechanism to verify that.

FR-10 *“Companies are capable of creating and managing as many users as their service plan allows.”*

Actor: Admin users.

Description: Only Admin users can create company users, as many as the company service plan allows. To create a user at least is necessary to provide an email. The created users cannot use yet *Flouescent* services, as they need first to validate their identity. The users sign up system is designed to achieve that. If a company changes its service plan and has more users than the new plan allows, then it is not possible to create more users, but they are not deleted.

FR-11 *“Companies can manage their user’s authorization levels.”*

Actor: Admin users.

Description: Admin users are free to change at any time the authorization level of each company user. Admin users have high privileges inside companies. Therefore, company managers are responsible for any malfunction or data access caused by improper privileges.

FR-12 *“It is essential for users to verify their accounts after creation. A mechanism allow users to set their credentials and accept the terms of use and privacy policy.”*

Actor: External system actor applying as new user.

Description: A user must be verified to use *Flouescent* services. A mechanism called *Flouescent* Users Sign Up System (*USUS*) module is designed to handle sign up’s and verifying users. It is a complex and secure module that is defined and designed later. Without verification, users are unable to use the application.

FR-13 *“Authentication is required. The system requires users to provide credentials to log in.”*

Actor: Normal users.

Description: A user must provide credentials to log in at *Flouescent* application. The credentials are an email address and a password.

FR-14 *“Tokens allow users to keep logged sessions.”*

Actor: Normal users.

Description: A token is served to the user after login through cookies, avoiding user credentials transactions at every server API call. This token is active some hours or days, and then it becomes invalid. The token associates a user and a dispositive to a logged account, offering better usability, acceptability, and security.

FR-15 *“Users can manage their profiles.”*

Actor: Normal users.

Description: Users can manage all their profile personal data, such as email, username, name, surname, or profile picture. The user password could be modified, having to provide the old one for security reasons.

FR-16 *“Users can use all the Multimedia features allowed by their companies service plans.”*

Actor: Normal users.

Description: Normal users can use all multimedia features restricted only by their companies service plan specifications.

FR-17 *“Authorized users can manage their companies providing confidential data, settings, and management.”*

Actor: Admin users.

Description: Admin users are Normal users with company management extra privileges. In other words, Admin users can perform all company actions.

FR-18 *“Authorized users from system authorized companies would be able to create new companies and manage their service plans.”*

Actor: Admin users of Admin companies.

Description: Admin users of admin companies have the maximum authorization level. Theoretically, they represent the *Flouescent* administrator team, that manages all companies using *Flouescent* services. They can suspend the service to companies (due to financial debt or terms and conditions violation) or change their service plans.

FR-19 *“Users can create rooms providing some room settings.”*

Actor: Normal users.

Description: Users can create rooms at any time with custom settings. These settings are room type (authorization), room recording type, and room recording permissions. Users can set default settings to be applied to every room creation.

FR-20 *“Users can visualize their room activity history of created and participated sessions.”*

Actor: Normal users.

Description: All users can visualize their sessions history (room activity), both own created rooms and participated ones.

FR-21 *“Users must provide at least the invitation key to join a room.”*

Actor: Normal users.

Description: A room participant user that wants other users to join a room needs to share the invitation key through any communication path with them. The invitation key is necessary for room joining.

FR-22 *“To access Restricted rooms, users must provide the invitation key but must belong to the same company as the room creator user.”*

Actor: Normal users.

Description: Any user can join free to any open room if it has the necessary invitation key. If the room is restricted, then the user must belong to the same company as the room creator (if not, access is blocked).

FR-23 *“Inside a room, users can change their video webcams at any moment.”*

Actor: Normal users.

Description: While conferencing, it is necessary in some cases to change the current webcam producer. In the case of multiple webcam devices, without the cam switch option, a user would be unable to select the desired producer device, and the application would select it randomly. It is especially relevant in smartphone devices, that usually have more than one cam.

FR-24 *“Users can manage recording settings (type and set authorization) if their company service plan allows it.”*

Actor: Normal users.

Description: The recording type could be *disabled*, *audio* or *audio & video*, and the recording authorization *open* or *restricted*. If a company changes its service plan, users default settings are not erased, although are disabled since the company re-enables recordings (through a service plan improvement). Recording settings must be configured before room creation and cannot be changed later. Users have changeable default settings that apply to every room creation.

FR-25 *“Users can access an open record if they have participated in it. They can access a restricted record if the creator user is from the same company.”*

Actor: Normal users.

Description: The open record rooms are accessible by all participants, but restricted record ones only from users of the same company as the creator. Recordings become available after the room closure (if enabled).

FR-26 *“Authorized users can access to all recordings of their company users.”*

Actor: Admin users.

Description: Authorized users can access to all recordings of their company users.

FR-27 *“Authorized users can change their companies service plans through the web interface.”*

Actor: Admin users.

Description: It is easy for admin users to change their companies service plans through the web interface. An automatic procedure using the web interface is provided.

FR-28 *“Authorized users from system authorized companies can modify service plans templates.”*

Actor: Admin users of Admin companies.

Description: Admin users of Admin companies can modify service plans features at any moment, but not the number (3 plans) and the names (starter, standard, and premium). If a service plan is modified, it does not affect current active company ones (companies could maintain the old service plan benefits, or choose to change to the new one).

8.2.2 Non-functional requirements analysis

The following description analyzes the system non-functional requirements together with a group definition (system part) and a description (more detailed explanation):

NFR-1 “*Via an analysis of some marketing aspects, such as names, logos, and service plans, the system has to be more prepared for the production launch.*”

Group: Marketing study.

Description: To offer to *Flouescent* more opportunities to a hypothetical commercial launch, it would be grateful to perform a simple marketing study that provides some studied logos, names, or plans. The study is explained in the *Marketing study* section.

NFR-2 “*A business analysis show if the application is commercially feasible.*”

Group: Business study.

Description: To offer to *Flouescent* more opportunities to a hypothetical commercial launch, it would be grateful to check if the project is feasible according to the competence.

NFR-3 “*The project does not have to contemplate any payment logic.*”

Group: Monetary management.

Description: Although *Flouescent* is implemented with service plans payment subscriptions, it avoids all financial data collection. This is a final degree project, and the later possible commercial launch is out of scope.

NFR-4 “*The system has three service plans that are named as the market study suggests, ordered from lower to higher services.*”

Group: Service plans.

Description: *Flouescent* service plans have to be designed to be customizable. The problem here is that a fully customizable service plans would add complexity, being better to avoid according to project objectives. Fixing service plans number to three (fixed, studied to be an ideal option) and its names to the starter, standard, and premium (also studied) make the logic simpler.

NFR-5 “*The server must work with limited hardware resources and be scalable.*”

Group: Throughput.

Description: As one of the main objectives, all throughput aspects are carefully treated. Theoretically, MediaSoup WebRTC SFU technology should be more efficient than the significant current systems and could be executed under limited hardware systems. Native MediaSoup should work well on those systems but probably could appear some problems with recording functionality, which is more CPU expensive. In any case, recordings under limited hardware are tested in the *Results* section.

NFR-6 *“The server API must be developed following API REST standards.”*

Group: Design.

Description: Some API REST coding standards theoretically provides more code security and readability. As there are studied under solid reasoning, there is no reason to avoid them.

NFR-7 *“All data (e.g., companies, users, and recordings) must be stored correctly. Sensitive information, like passwords or classified meeting recordings, must be secured using cryptographic procedures.”*

Group: Security.

Description: The server has to be under strict security control. All data must be checked and stored carefully at the server database. Passwords and access tokens are stored under stringent hashing algorithms to avoid data extraction attacks.

NFR-8 *“In case of a security breach and sensitive information exposure, the system must guarantee that the time needed to extract the sensible data are longer than the information life span.”*

Group: Security.

Description: Related with previous data security requirements, this one guarantees that users credentials and other sensitive data could not be extracted during the information life span. The cost of the attack is exponentially much expensive than the information cost.

NFR-9 *“All data must remain intact in case of failure.”*

Group: Integrity.

Description: The data integrity is another crucial part to be taken into account. Although all processing recordings become corrupted after a failure, all the other information must remain intact.

NFR-10 *“The web interface has to be cross-platform following a responsive design. Minimal and clear application documentation must be provided.”*

Group: Design.

Description: Web applications are continually growing and coexisting with mobile applications. There is no incentive to make a mobile application, as front-end interfaces are developed to be usable at every device type through a responsive design.

NFR-11 *“The web interface design must be user-friendly and easy to use.”*

Group: Design.

Description: Usability and acceptability are the most important finding for users satisfaction. The interface components are studied to be understandable and easy-to-use.

NFR-12 *“The system is required to have a 99.99% availability.”*

Group: Security.

Description: Web services normally guarantee at least a 99.99% availability. It could be easily raised with correct system monitoring tools and fast failure recovery rates.

NFR-13 *“The service has to be suitable and easy to install.”*

Group: Usability.

Description: The documentation and user manual specifies the whole installation process step-by-step.

NFR-14 *“The application code has to follow the chosen programming language practices.”*

Group: Design.

Description: Ideal programming practices have to be followed to avoid poor quality and unmaintainable code.

NFR-15 *“Minimal and clear application documentation must be provided.”*

Group: Design.

Description: As explained before, minimal documentation has to be provided at the source-code level following the Personal Extreme Programming (*PXP*) methodology standards.

NFR-16 *“The application must fulfill all Spanish and European legal requirements.”*

Group: Legal requirements.

Description: All commercial applications must fulfill national legal regulations. The software must contain detailed legal notice and right user agreements.

8.3 Use case diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. The actors are people or entities operating under defined roles within the system.

The previously studied requirements show three system actors or user authorizations levels: Normal users, admin users, and admin users of admin companies. With all functional system requirements analyzed and grouped by actors, providing a graphic use case diagram is helpful to visualize the system scope. The **Figure 26** illustrates a simplified use case diagram with actions referenced to requirements.

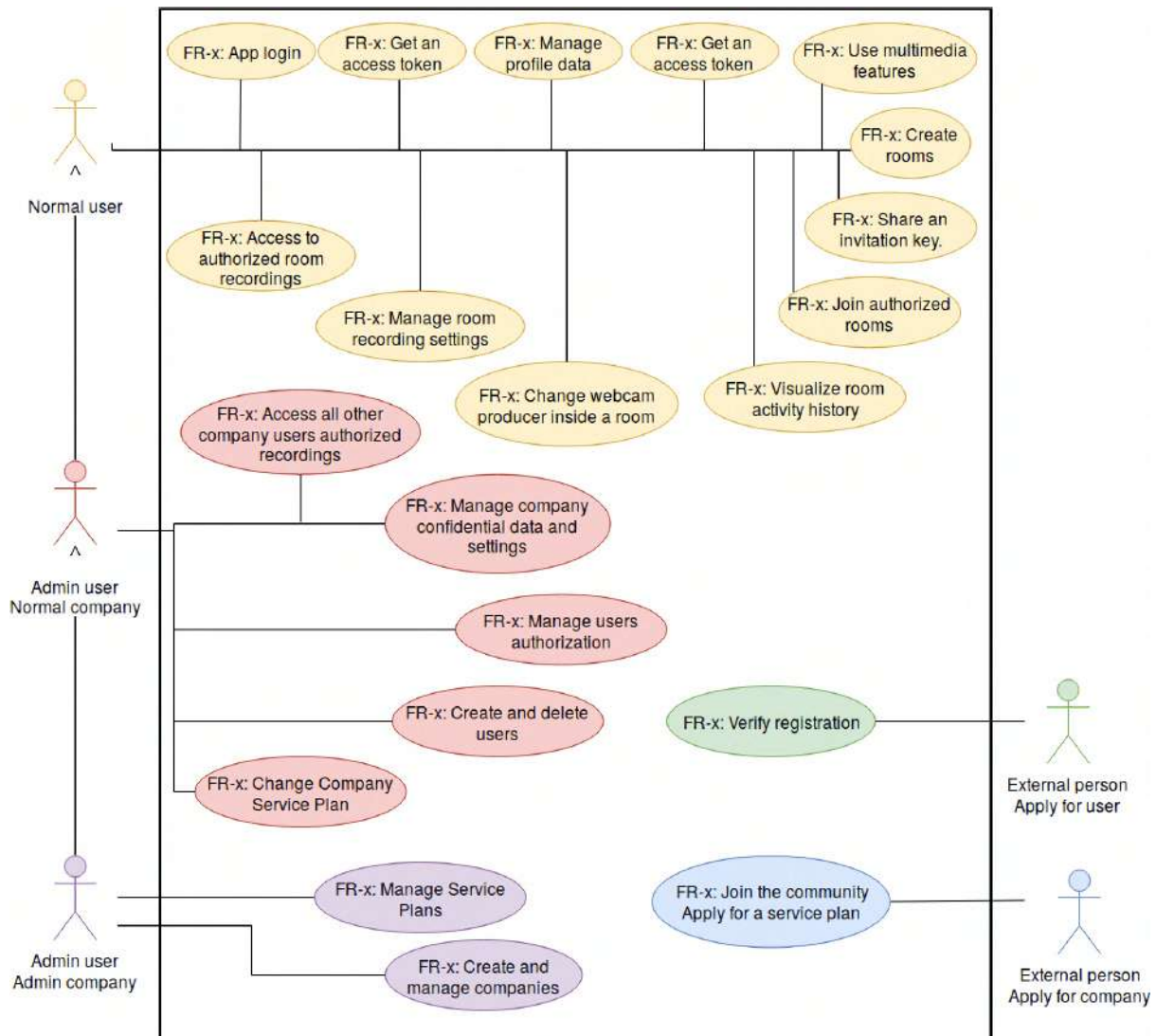


Figure 26: Flouescent simple use case diagram.

8.4 Database design

As specified in the *Studies and decisions* section, the chosen project database is MongoDB (NoSQL). Traditional SQL databases can be described as a set of interrelated entities (also named tables). MongoDB databases work with references instead of relations. Their data structures are named as *collections*. From now on, a database entity is named as *collection*, an entity instance as a *document*, and an instance column as a *field*. **Figure 27** illustrates graphically these concepts differences between SQL databases and MongoDB. **Figure 28** illustrates the database design collections and its relations, that are handled internally through MongoDB references.

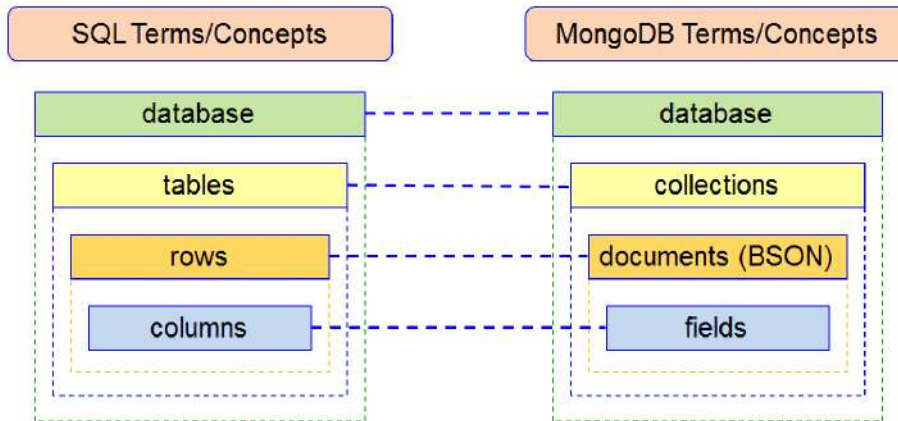


Figure 27: Database concepts SQL vs MongoDB.

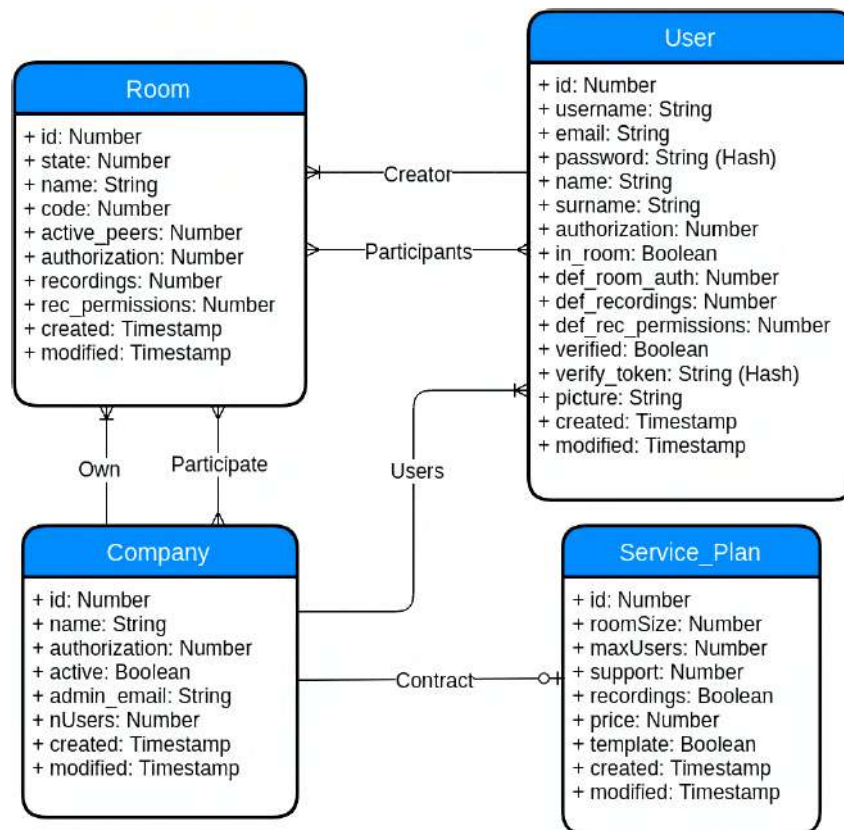


Figure 28: *Flouescent* MongoDB database design diagram.

It is important to note that room recording files are saved externally due to their size, with a room ID reference, and the server logic ensures that only authorized users could access to those files. For this reason, recordings are not included at this database design diagram, and all required fields to guarantee correct file access authorization are stored at *Room* collection.

The database diagram is quite simple, although it is well related. As exposed in the *Studies and decisions* section, this is not only a software project and this simple database design contrasts with other complex application modules, such as the *Flouescent Recording System (RS)* module.

The *Company* collection contains documents with basic company necessary information attributes like *id*, *name*, and *authorization*, that indicates if its an Admin company (1) or normal company (0). As explained in the *System requirements analysis* section, a company can be deactivated due to non-payment or legal terms violation, indicated in *active* attribute. Moreover, it have an attribute *admin_email* to indicate the original Admin user email (the user that originally created the company). The total company users number is stored as *nUsers* field for efficiency purposes. The *created* and *modified* timestamps are present for control purposes, like all other entities.

The *Service_Plan* collection contains documents with service plans related features. The three template service plans are unlinked from companies and are the matrix to apply instances. When a company applies for a service plan, it has the three templates plan options. When a service plan is contracted, then the system makes a copy of its features instancing them to the company. For this reason, the attribute *template* indicates if the service plan is a template (true, the three template service plans) or is a company instance (false). *Service_Plan* collection contains the following attributes: *roomSize* as maximum users per room, *maxUsers* as max total company users number, *support* as a support level, *recording* to indicate if companies could perform recordings (0 for nothing, 1 for audio, and 2 for both), and *price* as the monthly tax.

The *Room* collection contains documents with all room related information together with recording settings. The *state* field indicates if a room is open (0), if it is processing recordings (1), if it is closed correctly (2), or if it is closed with a record processing error (3). Furthermore, it has an identifier *name* field provided by the creator user, the open room number *code* field (from 0 to 999), and the *authorization* field that indicates if room is open or restricted. The *recording* field indicates if the room has recordings activated (0 for none, 1 for audio, and 2 for both) together with recording permission *rec_permissions* field, that is 0 or 1 whether it is open or restricted. Finally, it contains a dynamically updated *active_peers* field to make sure that the users inside a room do not pass the allowed quota of the creator user company.

The *User* collection contains documents that basically contain basic user data information like *name*, *username*, *email*, *surname*, and *picture*. In order to login, users have to provide their email and password stored hashed and salted by a 12 round *bcrypt* algorithm at *password* field. The *in_room* attribute indicates if an user is currently inside some room, to avoid that it could be at more rooms at the same time. Finally, the *def_room_auth*, *def_recordings*, and *def_rec_permissions* store the users desired default options for room authorization, record type and record permission respectively.

Company document instances are related with all other collections, *Room* (as owner company and participant company), *User* (as company users), and *Service_Plan* (as currently contracted service plan). Moreover, the *Room* has a dual relation with *User* (as room creator and room participants). The following **Table 15** shows all transformations from those relations to MongoDB references.

| Relation | From Collection | To Collection | Reference |
|---|-------------------|---------------------|---|
| Companies own rooms | Company (1) | Room (many) | Each room stores a reference to its own company. |
| Companies participate in rooms | Company (many) | Room (many) | Each room stores a list reference of participated companies. |
| Companies contract service plans | Company (0) | Service_Plan (1) | Each company stores a reference with the currently active plan. |
| Companies have users | Company (1) | User (many) | Each user stores a reference to its own company. |
| Rooms have a creator user | Room (many) | User (1) | Each room stores a reference to its creator user. |
| Rooms have user participants | Room (many) | User (many) | Each Room stores a list reference to its user participants. |

Table 15: All transformation from database relations to MongoDB references.

8.5 Modules design

All *Flouescent* modules are defined and designed in this section. Modules are system core functionalities that merge in *Flouescent* application, from server and front-end ones. Modules are shown ordered from more internal to external, because external modules may use more internal ones.

Logger

The *Flouescent* Server Logger (*L*) module is a log handler that allows other modules to print and store a server log. It is important to have a clear and understandable server log to efficiently identify problems and errors, making the implementation phase much comfortable.

The *L* module uses the *simple-node-logger* NPM community package, quoted at *Studies and decisions* section. It allows printing log messages both to terminal and to an external file, identifying errors even in testing (terminal) or production (file) environments. The main features of this package are the facility to change between time formats and message types, which are *trace*, *debug*, *info*, *warn*, *error* and *fatal*. **Figure 29** illustrates a small portion example of *Flouescent* Server log.

```

20-05-19 08:38:22 INFO Running 1 mediasoup Workers...
20-05-19 08:38:22 INFO Running protoo WebSocketServer...
20-05-19 08:38:22 INFO Flowescent HTTPS Server Started at port
4443
20-05-19 08:38:22 INFO MongoDB Connection Established
20-05-19 08:38:53 INFO WS connection request
[roomId:5ce24b7d26dbf00cd3ed6ae5, peerId:5ce24b5ba8b03a0cacf7de5a,
address::ffff:84.88.154.6, origin:https://84.88.154.6:4443]
20-05-19 08:38:53 INFO Creating new room
[roomId:5ce24b7d26dbf00cd3ed6ae5]
20-05-19 08:38:53 INFO create() [roomId:5ce24b7d26dbf00cd3ed6ae5,
forceH264:false]

```

Figure 29: *Flowescent* Server *L* module: Log file example.

Mailing

The Mailing (*M*) module provides email services to *Flowescent* Server. As a *SaaS* based application, it is crucial to communicate with users using an email service.

The *M* module uses the *nodemailer* NPM community package, quoted in the *Studies and decisions* section. *nodemailer* is able to connect with external email service providers (e.g., Gmail, Outlook) and send emails without any problem.

From now, the *M* module only handles essential messages. As soon as the application grows, it would be easy to create more messages or to modify existing ones. The following list exposes all current *Flowescent M* module messages:

- a) **Company Sign Up:** This email message welcomes a new company when it registers. The message is sent to the company admin user creator and provides a secure token that allows validating the creator account (Provides a direct link to the *SUS* module, explained later).
- b) **User Sign Up:** This email message is sent to a new user of a specific company, after being created by an admin user. It provides a secure token that allows validating the user account (Provides a direct link to the *SUS* module, explained later).
- c) **User Welcome:** This email message welcomes a new user once it is verified. The message is sent to the confirmed users after their successful validations through the *SUS* module. It encourages users to use the platform and customize their profiles.
- d) **User Goodbye:** This email sends a goodbye message if users are deleted by their company administrators.
- e) **User Password Reset Request:** This email message contains a password reset request. The message is sent to a user who requested password recovery. It provides a secure token that allows users to recover their password (Provides a direct link to the *SUS* module, explained later, which manages password recovery logic).
- f) **User Password Reset Confirmation:** This email message shows a confirmation of successful password recovery. The message is sent to the requested users after their successful password recovery through the *SUS* module.

Database Connection

The Database Connection (*DC*) server module provides a connection between the node.js server and the MongoDB database. MongoDB stores data in JSON format structures and node.js can comfortably work with them.

The *DC* module uses the *mongoose* NPM community package, quoted in the *Studies and decisions* section. *mongoose* is an Object Document Mapper (ODM) which can easily handle MongoDB documents and parse it to JavaScript objects. It provides a schema-based solution to model application data, including casting, validation, query building, and more.

In order to ensure *mongoose* work with *Flouescent* Server it is necessary to define collections schemas and models. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection. In other words, a schema describes all document attributes fields, such as their data types and other options (e.g., default values, required fields). Once schemas are defined, it is time to build the models. Models are fancy constructors compiled from schemas definitions. Then, a model instance is directly a document. Models are responsible for creating and reading documents from the underlying MongoDB database.

All MongoDB *Flouescent* Server database schemas are defined following the previous database design study, exposed in the *Database design* section. This results in a company, room, service plan, and user schemas, which allow building all entities models. With all collection models defined, it is possible to create, read, and modify documents in many different ways. The following table describes abstractly the model's actions used at *Flouescent* Server:

1. **Search documents:** With a model *mongoose* object, it is easy to find a document from the database. It is possible to find a single document or many of them filtering by field values (e.g., equality, inequality, elements inside lists). The obtained objects are document instances.
2. **Create documents:** It is possible to create directly document instances from a *mongoose* object, just specifying all document fields.
3. **Save documents:** Once a document instance is created or fetched, it is trivial to save them. It is essential to change the instance document desired values, and then *mongoose* saves it whenever the programmer wants.
4. **Update documents:** Although document instances can be fetched and later saved, there is a reasonable way that allows directly to update document instances from a collection that matches with some field values. This could be performed directly over a *mongoose* model object.
5. **Delete documents:** With an identical behavior as document updates, documents can be directly removed from a collection specifying the field values of the document to be deleted. Furthermore, this could be performed directly over a *mongoose* model object.

6. **Populate searches:** As explained in the *Database design* section, MongoDB works with references instead of relations. Each document field that is related to another document ID can be easily populated to access its referenced document fields. This action is equivalent to SQL join searches.

REST API Middleware

The REST API Middleware (*RAM*) server module provides Middleware functionalities to *Flouescent* Server REST API. Middleware is functions that perform specific actions based on a REST API request information. These functions can be executed both before API routes logic and for error handler. To achieve decent code practices and avoid duplicate code, Middleware can implement some specific code logic that has to be executed before multiple API calls. Furthermore, it can insert an error handler to all requests. *Flouescent* Server uses four Middleware functions: One error handler, and three user authorization verification handlers (through tokens), one for each three-level authorization specified in the *Use case diagram* section.

The *RAM* module uses internally *L* and *DC* modules. The first one is necessary to log possible errors caught by the error handler, and the second one is necessary to access database documents for authentication verifications.

The error handler must ensure that all possible errors thrown at any route are caught and processed. However, it is not that simple, because errors can be caused by multiple failures caused by incorrect customer requests, authorizations, or server errors. For this reason, a custom error type list is defined, enabling API routes error throws. If an API route thrown an error is not listed at the error handler, then the error is logged through the *L* module, and the requester is answered with an unknown server error. **Table 16** shows all defined error types.

| Error type | HTTP response code | Throw cases |
|-----------------------------|--------------------|---|
| inv_user_credentials | 403 Forbidden | Thrown when a user provides invalid credentials. |
| user_not_verified | 403 Forbidden | Thrown when an unverified user tries to access to <i>Flowescent</i> services. |
| user_no_privileges | 401 Unauthorized | Thrown when a user attempts to perform an unauthorized operation. |
| inv_params | 400 Bad Request | Thrown when a user provides invalid parameters for a specific action. |
| already_exists | 403 Forbidden | Thrown when a user attempts to create or change elements that already exist in the system and cannot be duplicated. |
| not_found | 404 Not Found | Thrown when a user tries to access to nonexistent content. |
| sp_limit_users | 401 Unauthorized | Thrown when a user attempts to create more users but its company exceeds the service plan quota. |
| sp_max_room | 403 Forbidden | Thrown when a user tries to create join a Room but it has exceeded the service plan quota. |

Table 16: *RAM* module error types.

The first user authorization handler verifies if a user is or not authenticated. Called Token verifier, this handler allows API requests to pass through if they come from authenticated users, regardless of their authorization. If the requester is not logged, then the API denies the request by a 403 Forbidden HTTP error.

The second user authorization handler verifies if a user has or not admin privileges. Called Admin verifier, this handler allows API requests to pass through if they come from an authenticated and authorized admin user. If the requester is not authorized to act, then the API denies the request by a 401 Unauthorized HTTP error.

The third and last user authorization handler verifies if a user has or not admin privileges from an admin company. Called Root verifier, this handler allows API requests to pass through if they come from an authenticated and authorized admin user from an admin company. If the requester is not authorized to act, then the API denies the request by a 401 Unauthorized HTTP error.

REST API routing

The REST API routing (*RAR*) server module provides API routing calls to *Flouescent* Server REST API, which determines the path of a request and its necessary parameters. This module contains all routing logic for all database collections.

The *RAR* module uses internally *DC* and *RAM* modules. The first one is necessary to interact with the database, and the second one is necessary for error handling and user authentication and authorization validation.

It is important to design all API routing correctly. All routes have to follow the standard practices design [32] to achieve that. Each route is designed using a route path, description, parameters, authorization, and response fields. The route path describes the API URL, the description describes its functionalities, the parameters are the thinks that the customer has to provide, the authorization shows who can use the route, and the response describes the server answer that is sent to the requester. All routes are exposed through tables for better readability. **Table 17** shows all routes related with authentication. **Table 18** shows all routes related with companies. **Table 19** shows all routes related with service plans. **Table 20** shows all routes related with rooms. **Table 21** shows all routes related with users. All those API routes are implemented in the *Implementation and testing* section.

| Route path | Description | Parameters | Authorization | Response |
|-----------------------------|---|------------------------|---------------|-----------------|
| POST / | User login. | User credentials | -- | Logged user |
| GET /me | Checks authentication. | -- | Normal user | Logged user |
| GET /logout | User logout. | -- | Normal user | -- |
| POST /token/check | Checks the user pre-sign up token. Explained in thee <i>SUS</i> module. | User pre-sign up token | -- | Unverified user |
| POST /token/validate | Validates the user pre-sign up token. Explained in the <i>SUS</i> module. | User pre-sign up token | -- | -- |

Table 17: Authentication routes (/api/auth).

| Route path | Description | Parameters | Authorization | Response |
|---------------|--|-------------------------------------|--------------------------------|------------------|
| POST / | Company creation. | Company data | -- | Created company |
| GET / | Obtains all companies. | -- | Admin user from admin company | Logged user |
| PUT / | Modify company by <i>Flowescent</i> stuff. | Company identifier and company data | Admin users from admin company | Modified company |

Table 18: Company routes (/api/companies).

| Route path | Description | Parameters | Authorization | Response |
|--------------------|--|----------------------|---------------|--------------|
| POST / | Room creation. | Room data | Normal user | Created room |
| GET / | Obtains all rooms that the logged user has access. | -- | Normal user | Rooms |
| GET / | Checks if a room is available to join. | Room code identifier | Normal user | Room |
| GET /record | Obtains the records of a specified room. | Room code identifier | Normal user | Records |

Table 19: Room routes (/api/rooms).

| Route path | Description | Parameters | Authorization | Response |
|------------------------|---|----------------------------------|-------------------------------|-------------------------|
| GET / | Get the current logged user company service plan. | -- | Normal user | Service plan |
| GET /templates | Obtains all available service plans. | -- | Normal user | Available service plans |
| POST /templates | Modify an available service plan. | Service plan identifier and data | Admin user from admin company | Modified service plan |
| PUT / | Change company service plan. | Service plan identifier | Admin User | New service plan |

Table 20: Service plans routes (/api/plans).

| Route path | Description | Parameters | Authorization | Response |
|---------------------------|---|-----------------------------------|---------------|------------------------|
| GET / | Obtains all company users. | -- | Admin user | Company users |
| POST / | Creates a new user. | User data | Admin user | New user |
| PUT / | Modifies the logged user data. | User data | Normal user | Modified user |
| PUT /room_config | Updates the default room configuration for the logged user. | Room configuration | Normal user | New room configuration |
| PUT / | Modifies user data. | User identification and user data | Admin user | Modified user |
| POST /passwd | Updates the logged user password. | Old and new user password | Normal user | -- |
| POST /upload | Upload a new profile image for the logged user. | Image | Normal user | -- |
| GET /images | Obtains an user profile image | User identifier | Normal user | Image |
| DELETE / | Deletes an user. | User identifier | Admin user | -- |
| POST /passwd_reset | Request user password reset (forgot password). | User identifier | Normal user | -- |

Table 21: User routes (/api/users).

Sign Up System

The Sign Up System (*SUS*) module enables a secure and manageable user sign-up system.

The *SUS* module uses internally *M* and *RAR* modules. The first one is necessary to send emails to users, and the second one is necessary to offer this service through API routes. In particular, it uses check and verify token routes from authentication routes, as **Table 17** shows.

As *Flouescent* is *SaaS*-based, its direct clients are companies. Those companies can create and manage their users. The problem here is that the users must own application credentials to use *Flouescent* services, and those credentials should not be known by company administrators. This is where the mailing system can be used, enabling the server to arrange credentials directly with users through email.

The following steps describe how *SUS* module works:

1. **User creation:** An admin user from any company can create users through the web interface if the company user number does not exceed the company current quota. It has to specify only the new user email.
2. **Pre sign-up token creation:** The server creates a pre-sign up token that allows users to complete and verify their accounts. This token is sent through email from the server to the user.

3. **User account completion and verification:** Using the pre-sign up system tokens, users can access the completion of their account. Users only have to click a link that redirects to their account completion and verification. Then, users are required to create set their account credentials and accept the legal terms of use and privacy policy. After that, users become verified and can now use *Flouescent* services.

The last two steps are used to create a password recovery system. When users forgot their passwords, then can request a password reset that activates a mechanism using those same steps with minimum changes. Fort this reason, this password recovery system is considered as a part of this *SUS* module.

Recording System

The Recording System (*RS*) module enables video and audio recordings from rooms. It is one of the most complex application modules.

The *RS* module uses the *L* module, necessary for possible errors or unexpected behaviors that can appear during the recording process. This module is composed of two different parts, the live flow recordings, and their later synchronization and processing.

Recordings processing cannot be performed at flow live recording time. MediaSoup can easily forward multimedia flows to a recording endpoint but divided for each participant and each flow type (audio and video), as there are considered as different producers. This recording endpoint has to manage all router producers to store all video and audio participants flow, and that is the work of the flow recording part until room closure. The problem here is the multimedia synchronization issue, as participants join and leave rooms at any moment. Timestamps are helpful to solve the issue, as the participant producers first instant is known. They are stored with participants audio and video, enabling its later synchronization and processing. **Figure 30** illustrates a graphical representation of this flow live recording *RS* module functionality.

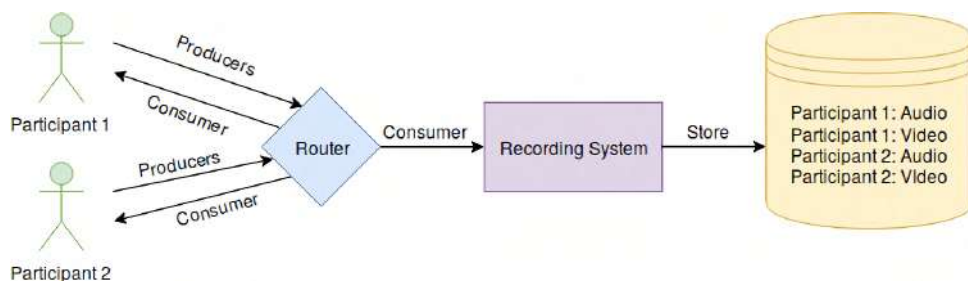


Figure 30: Room live flow recording: Two participants example.

Once a room is closed, then the recordings synchronization and processing process starts. It uses previously-stored timestamps to synchronize all media from separated components to a single merged file. The process differs between video and audio processing, as audio is easier processed. **Figure 31** illustrates a graphical representation of both audio and video synchronization and processing process.

The audio process only takes all audio producers and synchronizes and merges them through previously-stored timestamps. The resulting audio media is stored at the server and is accessible for authorized users depending on the room configuration. The video process is more complex, as it required a previous pre-processing, that enables adding some *Flouescent* marketing watermarks to all participants videos. It has another complication, as after the merge process the resulting video does not contain any audio. For this reason, it is necessary to add the previous processed audio track to the resulting video, obtaining the final video file.

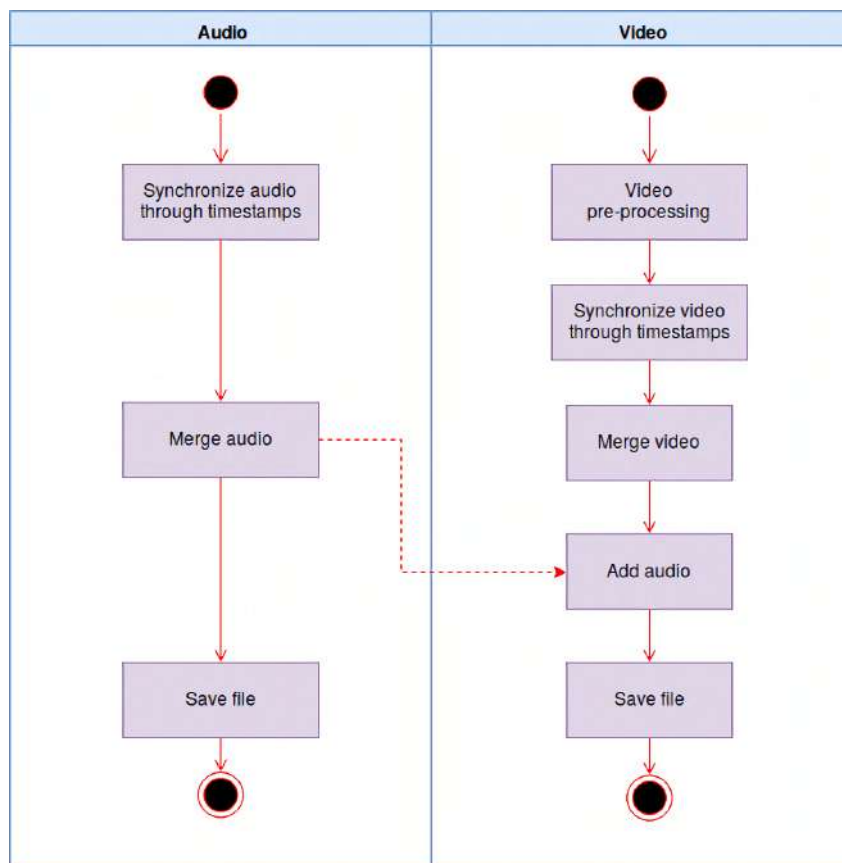


Figure 31: Recording media synchronization and processing processes.

What does this process possible is the SFU server, as all media flows must pass through it. However, the *RS* module can be placed in the front-end application, each one with their pros and cons. **Table 22** shows a comparison between this two solutions.

| Aspect | Server | Client |
|------------------------|---------------------------------------|-----------------------------------|
| Complexity | Centralized ✓ | Distributed ✗ |
| Efficiency | More powerful hardware resources ✓ | Limited hardware resources ✗ |
| Fault tolerance | Full control ✓ | Distributed control ✗ |
| Server load | High (all logic) ✗ | Low (merge logic) ✓ |
| Availability | Everybody ✓ | No guaranteed ✗ |
| Feasibility | High (server is controlled) ✓ | Low (uncontrolled behaviors) ✗ |

Table 22: Server and client recording system placement comparison.

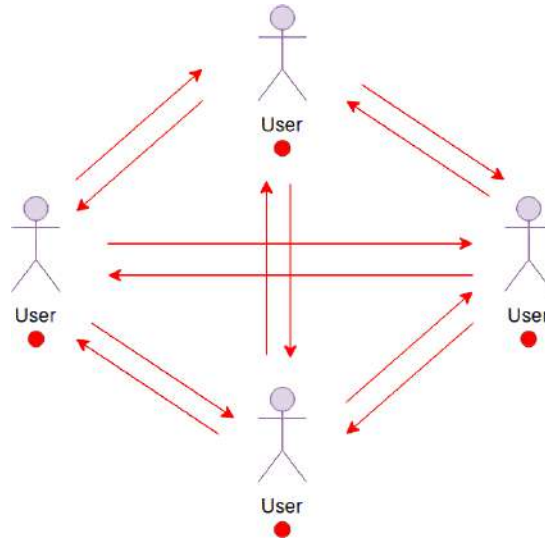
As exposed, client implementation is not feasible as it could generate no guaranteed behaviors that could be impossible to handle. The only problem with a centralized solution is the extra work added to the server.

It is fascinating thinking about how this system can be performed under a *P2P* based meetings application, where there is no central server². In this case, it is mandatory to manage recordings at the client-side, causing the problems exposed previously.

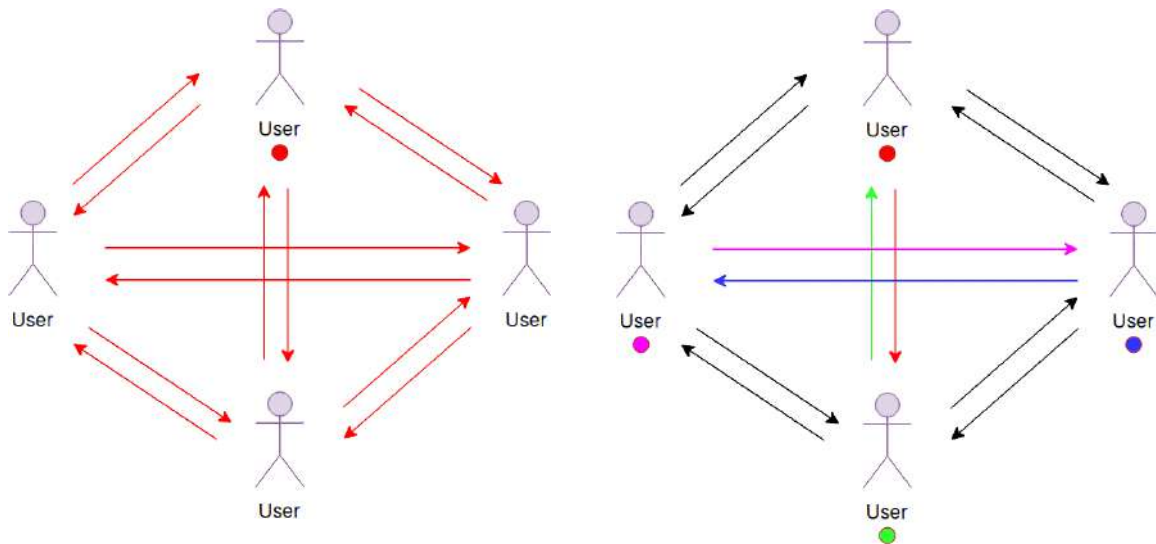
If the *RS* module development has to be placed at the client-side (e.g., *P2P* based meetings application), it can be done in many ways. The following list exposes all main possible ways, illustrated in **Figure 32**:

1. **Each user stores all meeting media:** It is robust and failure resilient, as all clients have a meeting record copy. However, it is inefficient and CPU expensive. As users can use the application in many different device types, it is a significant problem.
2. **One user stores all meeting media:** It is inefficient only in one client, as the others do not have to do anything. However, who should do the work is a significant problem. Besides, it is not robust, and a recording client failure invalidates all the meeting record.
3. **Users store their produced media:** It is the most load-balanced option. If a client gets involved in a failure, then only the failed client record part is lost. However, all client record parts are divided by all clients, causing a significant problem in deciding who, when, and how has to perform the merging and synchronization.

²In many cases, a TURN server must be used to bypass NAT restrictions, as explained in the *NAT problem* section.



(a) Case 1: Each user stores all meeting media.



(b) Case 2: One user stores all meeting media. (c) Case 3: Users store their produced media.

Figure 32: Client-side *RS* module placement: Possible solutions.

Room Management

The Room Management (*RM*) module enables multimedia meetings between users. It is one of the most complex application modules.

The *RM* module uses internally *RAR*, *DC*, and *RS* modules. The first is necessary to offer *RM* services through API routes, and the second enables the same server room management logic to interact with the database. The *RM* module is liable to manage the *RS* module, to start flow recordings at the precise moment for their later synchronization and processing. This module uses Web Socket connections to communicate client and server through a P2P architecture, although it uses some API routes. The API interface is used at room creation level. The room management communication architecture is through Web Sockets, where many verifications are performed.

Users should be able to create and join rooms in a fast way. When a room is created, then an invitation code is generated that allows other users access. This invitation code enables room joining from authorized users depending on room authorization level. The code is only available during the meeting and is destroyed later. Since more than one thousand concurrent open rooms are unlikely, a three-digit numeric code is sufficient. After room creation, any meeting participant can share the room invitation code to more users, who may join the room.

Inside a room, the front-end interface must guarantee a responsive grid visualization with all meeting participants for all screen types. Furthermore, it is necessary that the interface indicates if the room recording is enabled because users have the right to know if they are being recorded.

Web Sockets communications enable P2P communications between server and client applications. Those communications are complicated and handle multimedia rooms (e.g., creations, joins), and multimedia flows (e.g., transports, producers, consumers). As explained at *MediaSoup SFU Server* section, MediaSoup rooms are named as routers. A user would be represented in MediaSoup as a peer, but they do not exist. Instead, there are producers and consumers, that are multimedia endpoints that can both produce or consume router resources.

Web Sockets communication processes are complex. Two-room semantic concepts have to be maintained, one for server database and the other for MediaSoup routers, being the router linked with the database room. MediaSoup room data structure concept is named as *router*, to avoid confusions.

The following points explain how room and router creation and join processes works:

1. The application must ensure that a room has been created at the server database before creating its corresponding router through an API communication between client and server. After that, the client and server can establish a Web Socket communication to create and join the router and link it to its corresponding room.
2. A user that wants to join an existing room must check through an API communication if a meeting is active and if it has the authority to participate. If it has permission, then the client and server can establish a Web Socket communication.
3. When a Web Socket join request is accepted by the server, it interacts with the *DC* module to maintain an updated counter of online router users. Moreover, the server knows if a user leaves the room to decrease the online users counter.

As the creation and join processes are complicated, **Figure 33** illustrates a simplified graphical representation of the communication process.

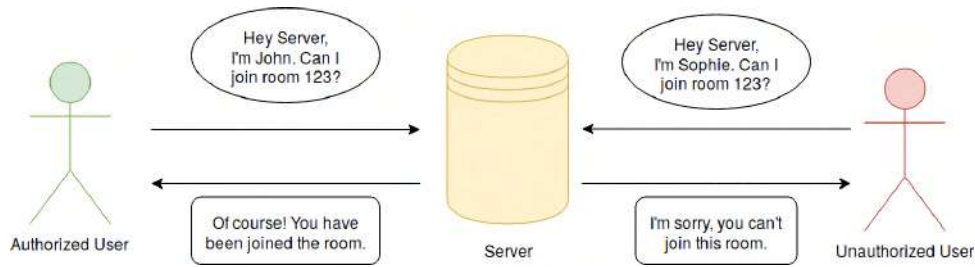


Figure 33: Simplified creation and join processes representation.

Once a room and its corresponding router are created, and the creator user joins the room, it can start to produce multimedia (audio and video). When another participant joins the meeting, it starts producing multimedia. Then, the creator and recently joined users consume the multimedia of the other. In other words, they receive video and audio from the router. The following points explain how these processes work:

1. The client and server must first arrange the multimedia codecs necessary for audio and video flow through WebRTC. The server has a list with all supported media codecs, which is transmitted to the client. If the client can use one of those codecs for audio and one for video, then urges the server to use them. If a client does not support the codecs, it cannot produce nor consume.
2. Once the codecs negotiation is done, then the clients create their send and receive transports, and transmits them to the server. The server connects them to the user joined router, and returns the connected transports.
3. With transports connected to MediaSoup router, then the clients can start producing their video and audio, which is sent to the server through sender transport through WebRTC. Furthermore, it can start consuming all the other participant's video and audio, which is sent to the client through receiver transport using WebRTC.

As produce and consume processes are complex, **Figure 34** illustrates a simplified graphical representation of the communication process between a recently joined user and the server.

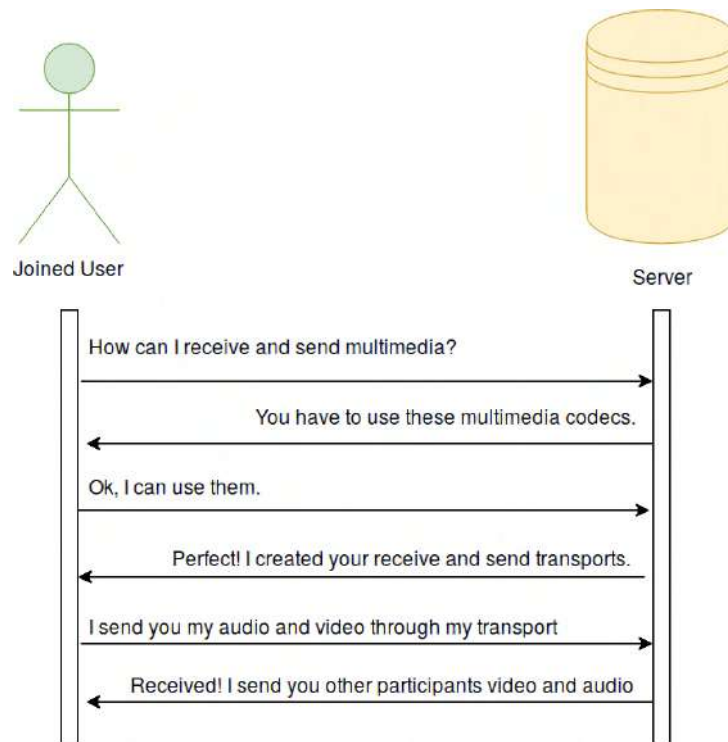


Figure 34: Simplified produce and consume processes representation.

8.6 Front-end interfaces

As explained in the *Studies and decisions* section, the front-end interfaces aspect follows the *material-sense* React template. In particular, **Table 23** shows all *Flouescent* front-end interfaces components.

| Interface component | URL route | Description | Used system modules |
|---------------------------|------------|---|--|
| Main | / | Contains the home front-end interface component. It contains the value proposition, service plans pricing and company sign up logic. | M: Used to send emails to the company admin user. RAR: Used for server API interaction. |
| Sign in | /signin | It contains the home front-end interface component. It includes the value proposition, service plans, and company sign up logic. | RAR: Used for server API interaction. |
| About us | /about | It contains the <i>Flouescent</i> marketing information that extends the value proposition exposed at the Main component. | RAR: Used for server API interaction. |
| Dashboard | /dashboard | It contains the main <i>Flouescent</i> application functionalities. It handles rooms creation and joining, sessions history visualization, and recordings access and reproduction. | RAR: Used for server API interaction. RM: Used to create and join rooms. |
| Room | /room | It contains the meeting conferencing functionality. It will interact with MediaSoup Server to send media and receive and visualize other participants media from the joined router. | RAR: Used for server API interaction. RM: Used to manage rooms and recordings (RS included at RM module). |
| Profile | /profile | It contains the profile customization logic. Users can change their profile data, picture, and account password. | RAR: Used for server API interaction. |
| Company manager | /company | It contains the company customization logic. Admin users can modify their companies data, like the current service plan. They can also manage all their company users. | RAR: Used for server API interaction. |
| Root panel manager | /panel | It contains the SaaS control logic. Admin users of Admin companies can manage all <i>Flouescent</i> companies, denying their service or changing service plans templates. | RAR: Used for server API interaction. |
| User sign up | /signup | It contains the users sign up logic. It enables users to create their accounts credentials and verify them. | RAR: Used for server API interaction. SUS: Used to give user credentials to new users and verify them. |
| Legal | /legal | It contains all legally required information about the <i>Flouescent</i> application. It includes the legal notice, data policy, and terms and conditions of use. | RAR: Used for server API interaction. |

Table 23: Front-end interfaces components.

9 Implementation and testing

This section shows all information related to the implementation and testing processes. As explained in the *Planning* section, all testing is performed at the same implementation time, testing all functionalities implemented parts through the system modules defined in the *System analysis and design* section. For this reason, both client and server applications are developed in parallel.

It is important to explain the *Flouescent* implementation process. For a better understanding, it is described using some step-by-step points descriptions. The *Flouescent Server* and *React Client* are the first explained implementations, describing both server and React to client implementation processes. The Web Socket protocol is explained next, being recordings the last explained implementation. All implementations together converge at final *Flouescent* application.

9.1 *Flouescent* Server

Before any implementation, the server API has to offer a simple and efficient way to modify its parameters easily without having to modify different files code. It could be done using a server configuration file, named *config.js*, placed at root project directory. All these file variables can be changed to customize server parameters that could change easily between different production environments. **Table 24** shows what this file *config.js* stores.

| Variable | Content definition |
|-------------------------|--|
| mongodb_uri | Database connection path or URI. |
| ip | Server host IP address. |
| port | Port which the whole application will listen to. |
| jwt_secret | Json Web Tokens secret that will be used to generate secure user tokens. |
| log_path | Server relative path, including file name, where the log will be stored. |
| email_account | <i>Flowescent</i> email account address used for automatic email communications with users. |
| email_pass | <i>Flowescent</i> email account password used for automatic email communications with users. |
| flowescent_url | Complete application URL. Example: <i>https://flowescent.com/</i> |
| https | Object with certificate and key server relative paths where HTTPS credentials are stored. |
| images_path | Server relative path, including file name, where images will be stored. |
| recordAudio_caps | Capabilities for RTP audio transport recordings. |
| recordVideo_caps | Capabilities for RTP video transport recordings. |
| service_plans | An object that contains all service plan templates features for starter, standard and premium plans. |
| mediasoup | A big object that contains the following MediaSoup configuration parameters. |
| video_record_res | The resolution which recording video will be stored. |
| numWorkers | MediaSoup workers number (recommended the same number as system CPUs) |
| worker | Some worker options like log levels and RTC ports range. |
| router | List of all router supported media codecs. |
| webRtcTransport | Default WebRTC transport settings, like listen IP. |

Table 24: Server configuration file variables definition.

The next step is to create a self-signed SSL certificate to generate an HTTPS testing server. This can be performed using *openssl* software [33]. The generated credentials should be placed at *cert* project root sub-directory, both certificate, and key. After certificate creation, information is provided to fill all server configuration file with all testing environment parameters.

Once the testing environment configuration file filled, *Flouescent* Server can be minimally created with all previously defined API routes. The server main initialization file is called *server.js*, placed at the project root directory. The following point description explains what this *server.js* file does:

1. **Run MediaSoup Workers:** The server has to start the MediaSoup workers number specified in the configuration file.
2. **Create an express app:** An express app has to be created to implement the API interface.
 - It has a *controllers* sub-directory where the controllers are placed. The express app is instanced to use the controllers from *controllers* sub-directory at specific base routes. Each controller is implemented following the *RAR* module design defined in the *System analysis and design* section.
 - It has a *models* sub-directory where the database models and schemas are defined. Each model and schema are defined following the database design exposed in the *Database design* section. It has a *front-end* sub-directory where React client application is placed. The express app is instanced to serve static files from the *front-end/build* directory, where the production build React app is placed. The *front-end* sub-directory is the root directory for the React application.
 - It has a *utils* sub-directory where the *RAM*, *M* and *L* modules are implemented following the module design defined at *System analysis and design* section. The error handler is named as *asyncMiddleware.js*, and the authorization verifiers as *verifyToken.js*, *verifyAdmin.js* and *verifyRoot.js*. The Mailer and Logger modules are implemented under the files *mailer.js* and *logger.js*.
 - It has a *db* sub-directory where the database connection logic is placed under a *connect_db.js* file, implemented following the *DC* module designed at *System analysis and design* section. Furthermore, it contains an *init_db.js* file used for testing purposes, that erases all MongoDB database and reinitialize it with default data.
 - It has an *uploads* sub-directory where all users profile images are saved, as defined at the server configuration file.

In summary, the express app serves the React application at root route and sets API controllers routes to use the base */api* route.

3. **Run an HTTPS server:** An https server has to be created to host the express app. It uses the SSL credentials, and the listen port specified at the configuration file.
4. **Run a Web Socket server:** It is necessary to create a Web Socket server to handle Web Sockets connections and messages. As explained at *Studies and decisions* section, the *protoo-server* library is used. Once the Web Socket server is created, it can be added to the created HTTPS server. The Web Socket protocol is explained later at the *Web Socket protocol* section.

The following list shows all the controllers placed at *controllers* sub-directory:

- **Authentication controller:** Controller with all designed authentication routes. It uses the */api/auth* API base route.
- **Company controller:** Controller with all designed companies routes. It uses the */api/companies* API base route.
- **Room controller:** Controller with all designed rooms routes. It uses the */api/rooms* API base route.
- **Service plan controller:** Controller with all designed service plans routes. It uses the */api/plans* API base route.
- **User controller:** Controller with all designed users routes. It uses the */api/users* API base route.

All API routes have been tested through Visual Studio Code REST Client extension separately. Each new route implementation was followed by its corresponding testing.

9.2 React client

The React client minimal implementation has been performed in parallel with the recently explained API server. React application has been implemented following the defined front-end interfaces in the *Front-end interfaces* section and the selected Material UI template named as *material-sense*. The following points explain step-by-step the React client implementation:

1. The *material-sense* Material UI React template was cloned inside *front-end* project sub-directory. It has been used as a start point. All the code is placed under *src* sub-directory, and images to use at web interface at *images* one.
2. An API functional module was created to centralize all API calls logic at one file, accessible through all application components. It has a function for every API call and returns the response to who requested the call. This functional module was named as *api.js* and placed at root React application directory.
3. The application Top Bar was implemented internally, containing different menu visualization possibilities. Depending on user authentication and authorization, the accessible menu options may vary. All the menus are placed at *src/components/menus* folder, and the Top Bar itself is placed at *src/components* folder with *Topbar.js* file name. According to all authentication and authorization possibilities, the following list show all implemented application menus:
 - **Menu Normal:** Visualized menu when the user is not authenticated. It shows the Main, Sign in, About us and Legal components, and is placed into *MenuNorm.js* file.
 - **Menu Auth:** Visualized menu when the user is authenticated, although it is not an admin user. It shows the Dashboard, Profile, About us, and Legal components, placed into *MenuAuth.js* file.

- **Menu Admin:** Menu that is shown when a user is authenticated, and it is an admin user. It shows the same components as menu auth together with Company manager additional component, and is placed into *MenuAdmin.js* file.
- **Menu Root:** Menu that is shown when the user is authenticated, and it is an admin user from an admin company. It shows the same components as menu admin but with Root panel manager additional component, and is placed into *MenuRoot.js* file. **Figure 35** illustrates this menu visualization.



Figure 35: Menu Root visualization at Top bar.

4. All necessary components were initialized following the exposed design at *Front-end interfaces* section. Each one performs the required API calls to fetch, modify, or create application elements as needed. Internally, components handle React states to link users inputs with a JavaScript value and then interact with the API functional module as desired.

The next descriptions define how previously studied design is implemented for each front-end component, being the most important ones complemented by their corresponding graphical interfaces view. They are tested through React developer tools each time new functionality is completed. Those tools create a simple developer server that enables live code testing while programming, being able to update the browser at each source code file change.

Main component

The Main component exposes the *Flowescent* application value proposition, service plans, and companies sign up logic. The value proposition is shown as studied in the *Business feasibility study* section, together with some images designed to complement the value proposition text. Below the value proposition, the service plans pricing is placed grouped with companies sign up logic. When a service plan request is clicked from the service plans pricing, a stepper is placed at the same place the company sign up logic. This logic asks for a company name and a company admin user email and forces users to accept the terms of use and privacy policy before continuing. **Figure 36** illustrates the component service plans pricing visualization.

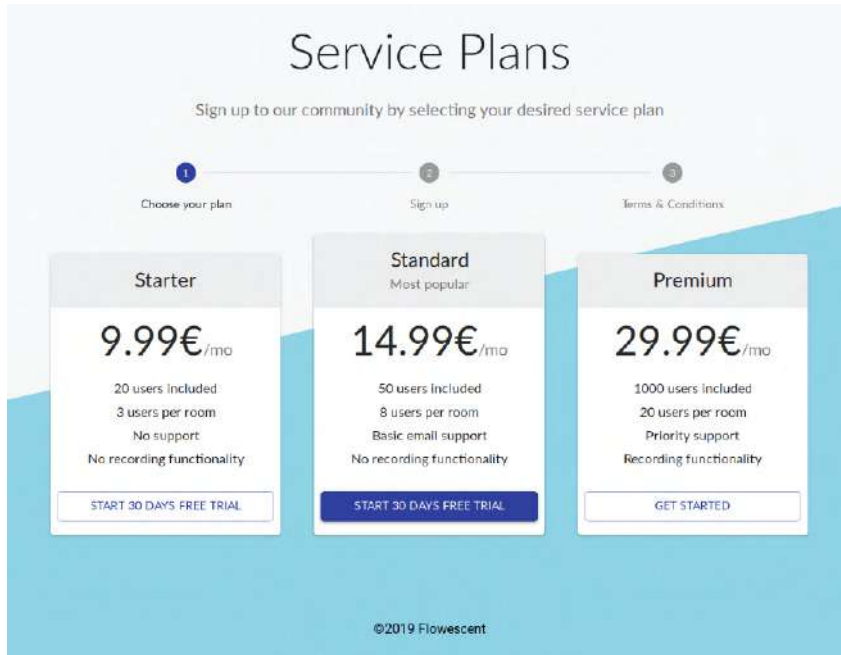


Figure 36: Service plans pricing visualization.

Sign In component

The Sign In component enables users to log-in at *Flowescent* application. It contains the login logic and provides a password recovery system. The login page consist in a simple login form with *email* and *password* input fields, a *sign in* button and a *I forgot my password* link. If a login action ends successfully, then the user is redirected to the Dashboard component. The *I forgot my password* link activates a mechanism that uses essentially the same logic of the User Sign Up component to implement password recovery functionality. **Figure 37** illustrates this component final states visualization.

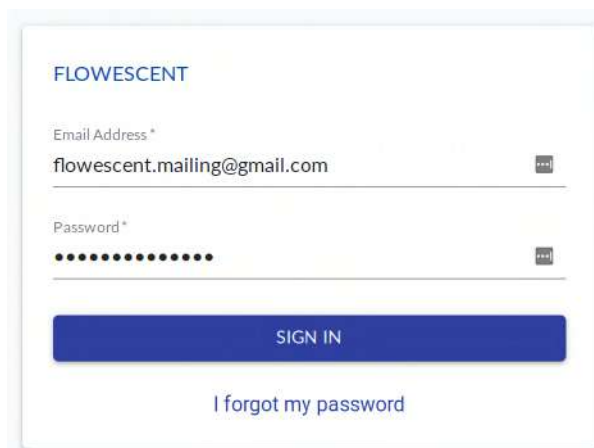
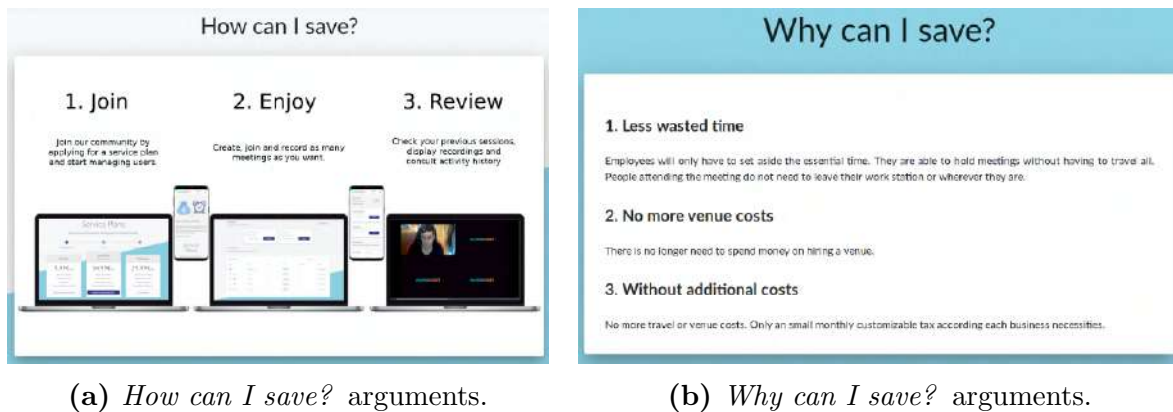


Figure 37: Sign In component parts visualization.

About Us component

The About Us component contains *Flouescent* application marketing information as an extension of the value proposition exposed in the Main component. It has a first element similar to the Main component value proposition, although with additional text that incites the client to request a 30 days free trial. Below, there are two more extra elements as part of the extended value proposition exposed at *Business feasibility study* section, the *How can I save?* and *Why can I save?* arguments. **Figure 39** illustrates this component final states visualization.



(a) *How can I save?* arguments.

(b) *Why can I save?* arguments.

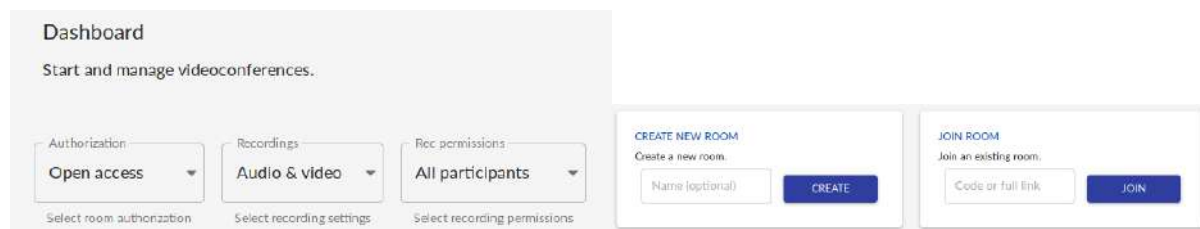


(a) Value proposition.

Figure 39: About Us component parts visualization.

Dashboard component

Dashboard component contains main *Flouescent* application functionalities. It handles room creation and joining, sessions history visualization, and recordings access and reproduction. First, at the component top are the room user default settings, with a *room authorization*, *recording type* and *recording permissions* selector fields, which enables creating rooms with customizable settings. The room creation and joining elements are at the top, divided into 2 boxes. The create box has an optional *name* input, and a *create* button in order to create a new room with specific name, and the joining box has a mandatory *code* input and a *join* button which enables router joining. The joining is performed if the code provided is valid, and the requester user has authorization. The user sessions history (owner and participation rooms) is shown in a customizable table, with *actions* (buttons), *state* (room state), *name* (room name), *date* (room modification date), *participants* (participants number) and *company* (company name of room creator) columns. The history shows all created rooms and shows different information according to the room state. If the room state is open, then one button is shown at *actions* column that enables direct room joining, and room code is shown at *status* column. Otherwise, the action buttons are the audio and video recordings, and the status no longer shows the room access code. Moreover, admin users have another table with the same structure as the first one but with all company users sessions. **Figure 41** illustrates this component main final states visualization.



(a) User default room settings.

(b) Room creation and joining boxes.

The screenshot shows the 'My sessions' section with the subtitle 'Check sessions and access to its recordings.' It contains a table with the following data:

| Actions | Status | Name | Date | Participants | Company |
|---------|------------------|------|---------------------|--------------|--------------|
| | Open - Code: 190 | -- | 6/2/2019 8:03:53 PM | 1 | root_company |
| | Closed | -- | 6/2/2019 8:03:44 PM | 1 | root_company |
| | Closed | -- | 6/2/2019 8:03:27 PM | 1 | root_company |
| | Closed | -- | 6/2/2019 7:56:53 PM | 2 | root_company |
| | Closed | -- | 6/2/2019 7:34:28 PM | 2 | root_company |

(a) User session history.

Figure 41: Dashboard component parts visualization.

Room component

The Room component contains the meeting conferencing functionality. It interacts with MediaSoup SFU Server to send media, and receive and visualize other participants media from the joined router. It starts after a room creation or joining success from the Dashboard component, or through an invitation link. This component does not contain the top bar with the menu and is not accessible through it from others. At the top left, it is placed the *Back to dashboard* button that allows users to leave rooms and go back to the Dashboard component. At the top center, it is placed a tree-digit number that is the room invitation code. If the room invitation code is clicked, then the room URL is copied to users computer clipboard. At the top right, there is an indication if the room is being recorded, which specifies if it is video or audio tracks. In the center are shown all the participants video visualized in responsive boxes being this functionality fully usable at 95% of screen types. At the bottom left it is shown the current user own video track with an option to change the producer webcam (in case of multiple connected devices), and at the bottom right it is shown a *Flouescent* logo watermark, that is shown complete if the screen is sufficiently large or reduced to favicon logo in case of small ones. All participant video tracks boxes are white bordered during some seconds when the router detects user audio. Audio tracks from remote users are not shown graphically, although they are commonly auto-reproducing. **Figure 42** illustrates this component visualization.



Figure 42: Room component visualization.

Profile component

The Profile component contains the user profiles customization logic. Users can change their profile data, picture, and account passwords. At the top, it is placed a *logout* button to manually close user sessions and a profile picture image at right, with the user's company name. There are two forms with respective submit buttons, one for changing user profile data (email, username, name, and surname) and the other for changing the current account password, providing the old one for security reasons. At the bottom, a profile image drag-and-drop box file uploader is shown with a submit button enabling user profile picture management.

Company Manager component

The Company Manager component contains companies customization logic. Admin users can modify their companies data, like current service plans. Furthermore, they can manage all their company users. The service plan manager is the top component element. It shows the current company service plan and its monthly cost. Moreover, it has a text bellow that differs depending on the service plan. For example, if the service plan is the starter or standard ones, the text encourages admin users to upgrade their companies plan, gaining more functionalities. The company users management is shown in a customizable table, with *actions* (buttons), *active* (user verification status), *email* (user email), *username* (user desired username), *name* (user name), *surname* (user surname), and *authorization* (if an user is admin or normal user) columns. Users can be created or deleted at any time, being the authorization level modifiable.

Root Panel Manager component

The Root Panel Manager component contains the *SaaS* administration control logic. Admin users of admin companies can manage all *Flouescent* registered companies (e.g., create new companies directly or disable companies services for non-payments) and modify service plan templates. The companies management is shown in a customizable table, with *actions* (modify), *active* (company state), *name* (company email), *admin email* (email of admin user company creator), *users* (company users number), *service plan* (current company service plan) and *authorization* (normal or admin companies) columns. The only editable columns are the *active* and *service plan* ones. Companies can be manually created introducing the *active*, *name*, *admin email* and *service plan* values. The service plan manager is placet at the component bottom. It has a service plan selector in order to change visualized service plan templates attributes for starter, standard and premium plans. All service plan templates fields are customizable (following some obvious rules), they can be changed only modifying desired fields values and saving it through *save template* button. **Figure 43** illustrates the service plan templates customizer visualization.

The screenshot displays the 'Service Plans' management interface. At the top, it says 'Service Plans' and 'Manage service plans templates'. There is a dropdown menu for 'Service Plan' currently set to 'Starter', followed by a blue 'SAVE TEMPLATE' button. Below this is a label 'Select a plan template'. The main area contains five customizable fields, each with a numeric input and a 'Save' icon: 'Admin Max Users' (3), 'Company Max Users' (20), 'Recordings' (Unauthorized), 'Support' (No support), and 'Price' (9.99).

Figure 43: *Flouescent* service plans templates customizer.

User Sign Up component

The User Sign Up component enables new users to create their account credentials and verify them. This component does not contain the Topbar with the menu, being inaccessible from other components. It is only accessible through a user sign up or user password recovery links. The password recovery process follows the same design as this sign-up system. The User Sign Up component is composed of a two-phase stepper. At the first step, an unmodifiable *email* input is shown, which is the email inserted by the company admin user who creates the account. The email is modifiable later, once the account is verified. The *email* input is followed by a password creation double field (for security reasons). Once the *sign up* button is clicked, the stepper renders the component at step 2, which shows terms of use and data privacy agreement checkbox. Once accepted, the user account is verified, and the application redirects to the Main component. **Figure 44** illustrates this component state visualization.

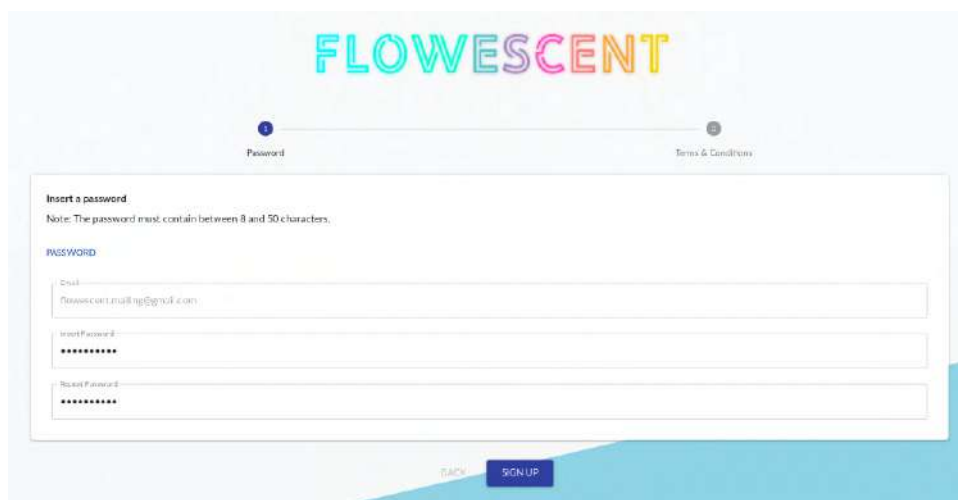
The image shows a web form for user sign-up. At the top, the word "FLOWESCENT" is displayed in a colorful, multi-colored font. Below it, a progress indicator shows two steps: "1 Password" (active) and "2 Terms & Conditions". The main form area is titled "Insert a password" and includes a note: "Note: The password must contain between 8 and 50 characters." Under the heading "PASSWORD", there are three input fields: "Email" (pre-filled with "flowescent@mail@gmail.com"), "Insert Password", and "Repeat Password", both of which are masked with asterisks. At the bottom of the form, there are two buttons: "BACK" and "SIGN UP".

Figure 44: User Sign Up component visualization.

Legal component

The Legal component contains all legally required information about *Flowescent* application. It includes the legal notice, data privacy, and terms of use and conditions.

9.3 Web Socket protocol

At this point, all *SaaS*-based application development and partial testing are done. This section explains the implementation process of the Web Socket protocol, the P2P communication protocol between the React client, and the MediaSoup SFU server. This protocol follows the previously designed *RM* module considerations. All Web Socket protocol messages are showed in **Table 25**.

| Message | From | To | Functionality |
|----------------------------------|--------|--------|--|
| connection request | Client | Server | Web Socket connection request. It can be accepted or rejected. |
| close | Client | Server | Web Socket closure listener. Peer closure is handled causing a <i>peerClosed</i> message from the server to all the other room participants. |
| getRouterRtp Capabilities | Client | Server | Client request for a router RTP capabilities. The server will answer with its capabilities. |
| join | Client | Server | Client room joins request. This is handled by the server and after notified to all existent room participants through <i>newPeer</i> message. |
| createWebRtc Transport | Client | Server | Client transport creation request. Clients can request both sender and receiver transport creations. The server will answer with the resulting created transport which the client can produce or consume. |
| connectWebRtc Transport | Client | Server | Client receiver transport connection request. The client provides a previous created receiver transport ID and its DTLS parameters. With those parameters, the server will be able to establish connections and send multimedia. |
| produce | Client | Server | It is the client produce request. It will use this message when audio or video is ready to be sent to the server. Then, the server adds the producer and consumes it at all receiver participants transports. Each producer type (audio and video) are handled separately. |
| activeSpeaker | Server | Client | Active speaker notification. It contains a data structure with information about current room peers and their actual volume levels. It is sent periodically to all room peers. |
| peerClosed | Server | Client | Peer closure notification. When a peer closes a room, then this notification will be sent to all other participants specifying who left. |
| newPeer | Server | Client | Peer join notification. When a peer joins a room, then this notification will be sent to all other participants specifying who joined. |

Table 25: *Flouescent* Web Socket protocol messages definition.

The Web Socket protocol definition is one of the lighter parts of the system. The design information studied in the *System analysis and design* together with MediaSoup SFU Server documentation recommendations enables a Web Socket protocol design without significant problems. Parallel client and server implementation allow the protocol to test its correct behavior by exchanging messages between them. It simulates all the action-cases related to the protocol at the final *Flouescent* application.

9.4 Recordings

One of the most important and marketable project functionality is meeting recording. This functionality implementation is complicated due to the complexity of audio and video tracks. Inside a meeting, users can join or leave rooms at any time, disable their cameras or mics, be only a listener, and more. This unpredictable environment causes recordings for being almost impossible or highly inefficient at many systems meeting based architectures.

MediaSoup SFU architecture enables an easy transports management that can be used for audio or/and video broadcasting, streaming, recording, among others, but it does not offer direct support for them. In other words, MediaSoup allows external endpoints that can interact with multimedia flows as they want, both producing (track imports) or consuming (track exports). At first glance, it does not seem an appropriate solution, although it opens fascinating doors related to efficiency optimization.

A question that has been done to MediaSoup developers is *Why do you not implement a built-in recording system?*, and it has a clear and convincing answer: *Because there is other software that does it better.* For example, there is a widely known and highly maintained open-source software called *ffmpeg* [34]. It manages audio and video tracks efficiently, being the MediaSoup developers answer accurately. An externalized recording system is compatible with the *RS* module design, defined in the *System analysis and design* section, offering a solution for the server load problem. These endpoints can be moved to other hosts, only dedicated to recording management.

Audio recordings are easier to handle than video ones, being processed differently. The audio recording procedure is named as Audio Recording System (*ARS*), and video recording procedure as Video Recording System (*VRS*).

Following the *RS* module previous defined design, the following list explains all steps that *ARS* implements in order to provide audio record functionality to *Flouescent* application:

1. **Flow live recording:** Explained at *RS* module design section. The *ARS* consumes all participant audio producers from a specific router while stores them at real-time one by one. As they have to be synchronized later, each file is stored with a timestamp of its joining instant.
2. **Synchronization process:** This process is started at the same time as room closure. It consists of all previous generated audio files merging to one final audio file. Each audio is delayed based on its timestamp, compared with the first participant audio one.

The *VRS* follows the *RS* module previous defined design, although complemented by some extra components. The following list explains all steps that *VRS* implements in order to provide video record functionality to *Flowescent* application:

1. **Flow live recording:** Explained at *RS* module design section. The *VRS* consumes all participant video producers from a specific router while stores them at real-time one by one. As they have to be synchronized later, each file is stored with a timestamp of its joining instant. In this case, video is more problematic than audio, because it can be transmitted through many different resolutions. The implemented system ensures that all tracks are stored in real-time with the same video format, chosen as VGA.
2. **Logo addition:** This process is started at the same time as room closure. It is essential not to forget that marketing and business aspects have a significant weight. An instant full-screen *Flowescent* logo is added at the beginning and the end of each video track recorded.
3. **Synchronization process:** This process is started after logo addition execution. It consists of all previously generated video files joining to one final video file. Each video is delayed based on its timestamp compared with the first participant video one. In this case, video is more problematic than audio. Audio can be merged easily without any problem, although videos cannot be overlapped. To visualize all videos, they are placed following a grid responsive design. This implementation is not trivial and is complicated.
4. **Audio addition:** This process is started after a successful *ARS* execution. It takes the final audio file generated by the *ARS* and overlaps it with the synchronized video file.

As *VRS* processing is much more expensive than *ARS* and requires high CPU resources, it is limited to 9 participants visualization (the first 9 participants). The audio is not limited, and all participants audio files are available at video recordings. However, only the first nine video tracks are visible.

The *RS* testing is a challenging application part. It is necessary to test it through *Flowescent* meeting rooms, forcing all other implementations to be performed before *RS* module implementation. As there are many room possible cases, it is not possible to guarantee a fully operational functioning without errors. In summary, the only way to test *RS* module correct behavior is to test it in as many room meeting configurations as possible, especially for the most common ones.

9.5 Final project structure

Figure 45 illustrates the final project directory tree structure split by server and client files.

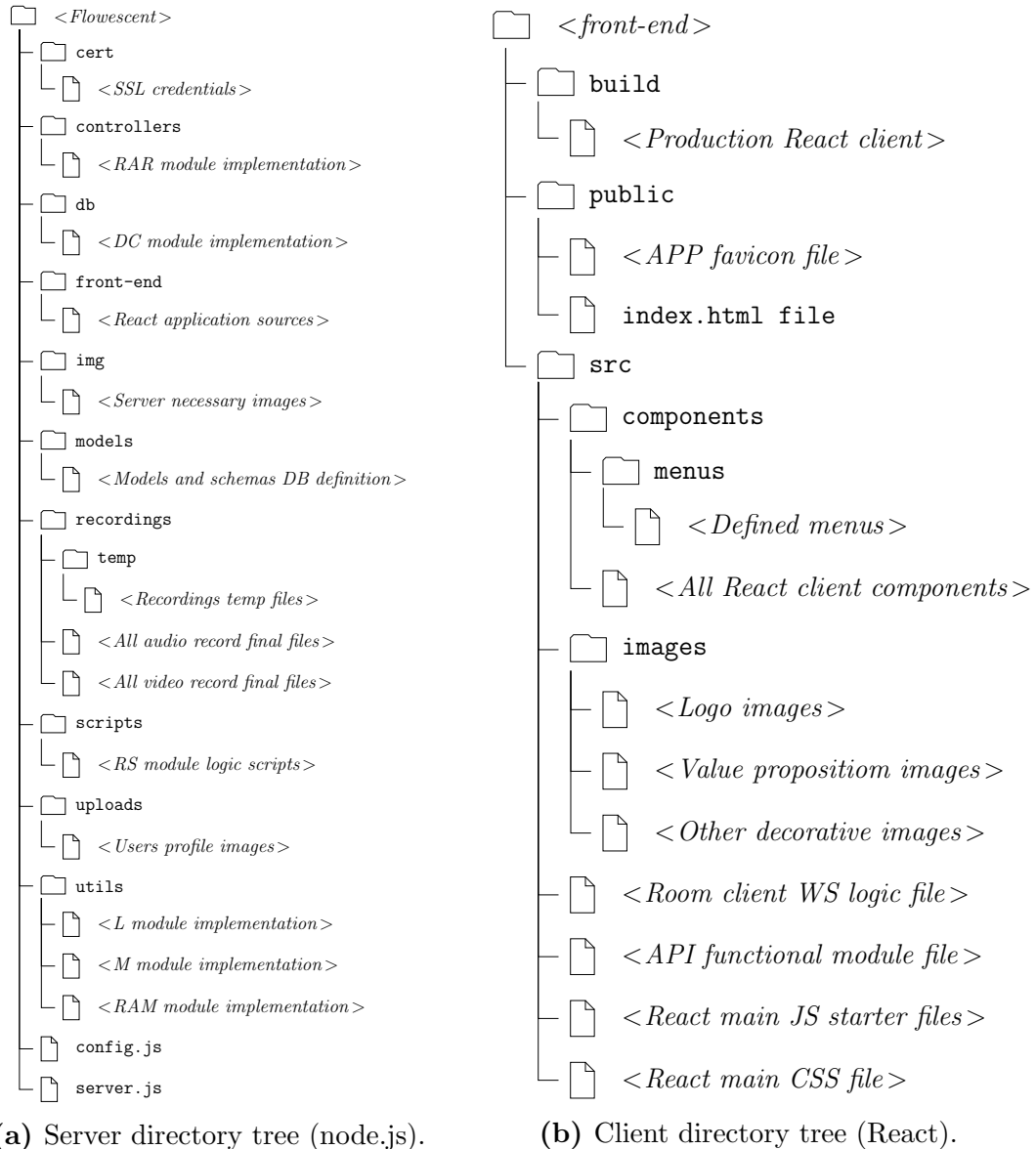


Figure 45: Final project directory tree.

10 Deployment and Results

10.1 Deployment

It is important to remember that MediaSoup works only on Linux hosts, being all other non-based Linux distributions excluded to host the application. Furthermore, there is another limitation with system ports, as *Flouescent* needs port 443 open to enable HTTPS connections for both React client and API, although it requires a considerable port range for WebRTC connections. For this reason, it cannot be deployed in port-restrictive network firewalls.

The deployment is a part of the **S8** sub-project. The deployment process may vary depending on the selected production environment. The following deployment step-by-step description explains all required settings to be performed to run a production *Flouescent* software beginning with its source code files. As an example, this process is described based on a clean Ubuntu server installation with public IP and DNS under port non-restrictive network, and follows all production environment considerations defined in the *Framework and background knowledge* section.

Basic software installation

Table 26 shows all necessary basic software installation description and installation commands for an Ubuntu server.

| Software | Description | Ubuntu server installation |
|----------------------------|--|---|
| Essential utilities | <i>gcc</i> and <i>g++</i> compilers. | <code>sudo apt-get install build-essential</code> |
| Git | Git client to download <i>Flouescent</i> source-code files from private Git remote repository. | <code>sudo apt-get install git</code> |
| OpenSSH server | SSH server which enables remote machine access. | <code>sudo apt-get install openssh-server</code> |
| Firewall | A firewall is necessary to protect the system from unauthorized access. | <ufw firewall installed by default> |
| Certbot | A tool from Electronic Frontier Foundation that enables trusted SSL certificates creation. | <code>sudo add-apt-repository ppa:certbot/certbot</code> <code>sudo apt-get update</code> <code>sudo apt-get install certbot</code> |
| Node.js | Development platform for executing JavaScript server-side code. | <code>curl -sL https://deb.nodesource.com/setup_10.x sudo -E bash -</code> <code>sudo apt-get install nodejs</code> |
| Yarn | Nodejs dependencies package manager. | <code>sudo apt-get install yarn</code> |

Table 26: Basic software installation.

Firewall configuration

A firewall must be configured to initially enable ports 22 (for SSH remote server administration), 80 (for a Certbot certificate validation) and 443 (listener port for *Flouescent* application) TCP connections. However, it must enable a port range TCP and UDP connections to handle WebRTC traffic. At this installation example, it is considered a port range from port 31000 to port 35000, which is more than enough for a small and medium application size. The selected range supports statistically 100 companies, considering the worst spike case. The worst spike is defined as 20 users using *Flouescent* services for each company.

The following steps apply the explained firewall configuration in an Ubuntu server:

1. `sudo ufw allow 22`: Enables connections from anywhere to host port 22.
2. `sudo ufw allow 80`: Enables connections from anywhere to host port 80.
3. `sudo ufw allow 443`: Enables connections from anywhere to host port 443.
4. `sudo ufw allow 31000:35000`: Enables connections from anywhere to host port range from 31000 to 35000.
5. `sudo ufw enable`: Enables firewall applying previously configured rules. It is enabled for each system boot.

TLS certificate generation

The TLS certificate generation and validation can be performed following the next points:

1. Generate the SSL certificate using Certbot. It can be performed executing the Certbot command `certbot certonly --manual`, pausing the execution when it asks for file access validation.
2. Create an Express application to serve static files. Start an HTTP node.js Express application and make the directories and files asked for Certbot accessible.
3. Confirm the domain, validating the certificate through the Certbot command.
4. Obtain the certificate. Certbot created the certificate and private key. Those file paths are inserted later at the server configuration file.

The *flaviocope's* guide^[35] contains more information about how to perform all steps to validate a TLS certificate through *certbot*.

Flouescent source code download

Flouescent source code files must be downloaded from its correspondent Git remote repository. The code its placed at project GitLab remote repository³. It is important to note that this application code is proprietary, and only authorized users can download those resources. The code repository can be downloaded using the following git command:

```
git clone repo_url
```

³TFG GitLab repository: <https://gitlab-bcds.udg.edu/david.ma/tfg>

Build React client application

It is necessary to execute three commands from the project root folder to build the React client application:

```
cd front-end (Enters to React application root folder)
yarn (Install all necessary dependencies)
yarn build (Build the React client)
```

Customize server configuration file

The last step before server execution is to customize the server configuration file, placed at the root project folder and named as *config.js*. All variables are modifiable, although the following ones must be modified:

- **ip:** Host public IP.
- **port:** *443*, as the default HTTPS port.
- **cert:** Let's encrypt certificate generated path.
- **key:** Let's encrypt private key generated path.
- **rtcMinPort:** Desired WebRTC port range low value. In this example: *31000*.
- **rtcMaxPort:** Desired WebRTC port range high value. In this example: *35000*.
- **email_account:** Desired email account address for mailing.
- **email_pass:** Desired email account password for mailing.

Server daemon execution

The server can be deployed once the configuration file is filled. It is necessary to use some node production daemon process to automate the execution (e.g., *pm2*, *forever*). *pm2* is a simple node.js daemon handler for production applications, installed through the *npm* package manager (*sudo npm install -g pm2*). The application execution starts when the following command is executed at the root project folder:

```
pm2 start server.js
```

The application can now be tested accessing *Flouescent* at the URL and port, specified in the configuration file.

10.2 Results

The *Implementation and testing* section explains all the *Flowescent* application implementation process and shows some partial results. A demo application has been prepared to enable free testing.

At this project report finalization time, the *Flowescent* demo is hosted in the *Hetzner cloud* [36], under limited hardware. Specifically, the virtual server has 1 vCPU and 2GB RAM hardware resources. The access is public, and it is available as a demo with all the service plans for free at the following URL: `flowescent.tk`.

As the demo host address can change, all necessary access information is periodically updated at this project public TFG Wiki ⁴. The wiki page contains the following information:

1. Definitive project report.
2. Project report summary.
3. Live demo information.
4. Complete legal notice, as a part of **S8** sub-project.
5. Other additional documentation.

It is necessary to perform some performance studies to analyze if the resulting *MFSS* is as scalable and efficient as it should be. The performance studies are simulations made at the production *Hetzner cloud* server limited hardware (1 vCPU and 2GB RAM). The server operation under limited hardware is one of the project requirements.

MFSS performance study

Figure 46 illustrates the *MFSS* studied performance, based on the media delay depending on meeting peers number. As the chart shows, the media delay is not appreciable in meetings with less than 5 participants. From 5 to 35 participants, the delay increases linearly, beginning with 50ms and ending with 1.8s. However, with more than 35 participants, the delay grows exponentially. As the demo production server is under limited hardware, the linear progression illustrates the medium to high CPU usage, and the exponential progression indicates a CPU collapse.

⁴TFG Wiki GitLab repository: <https://gitlab-bcds.udg.edu/david.ma/tfg-wiki>

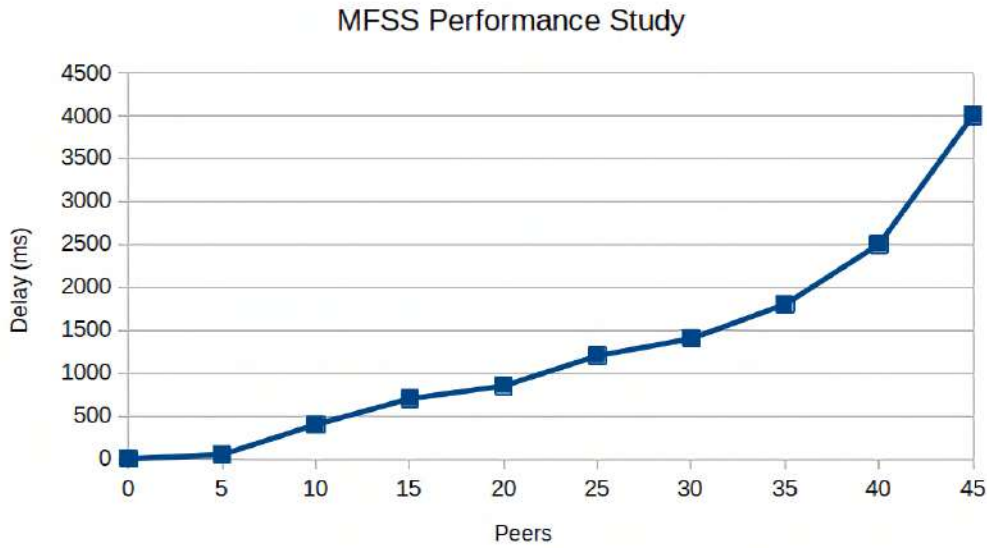


Figure 46: *MFSS* performance study.

A real-time meeting stops being practical from 15 participants, as more than 700ms delay does not allow to hold a normal conversation.

Recordings performance study

Figure 47 illustrates the audio recordings studied performance, based on the record processing time depending on meeting peers number in a 1-minute meeting. As the chart shows, the audio record processing time is not appreciable in 1-minute meetings. It begins with a 280ms delay (2 peers) and ends with a 500ms delay (10 peers). The processing time increments in a softened linear progression and more peers do not grow the processing time too much.

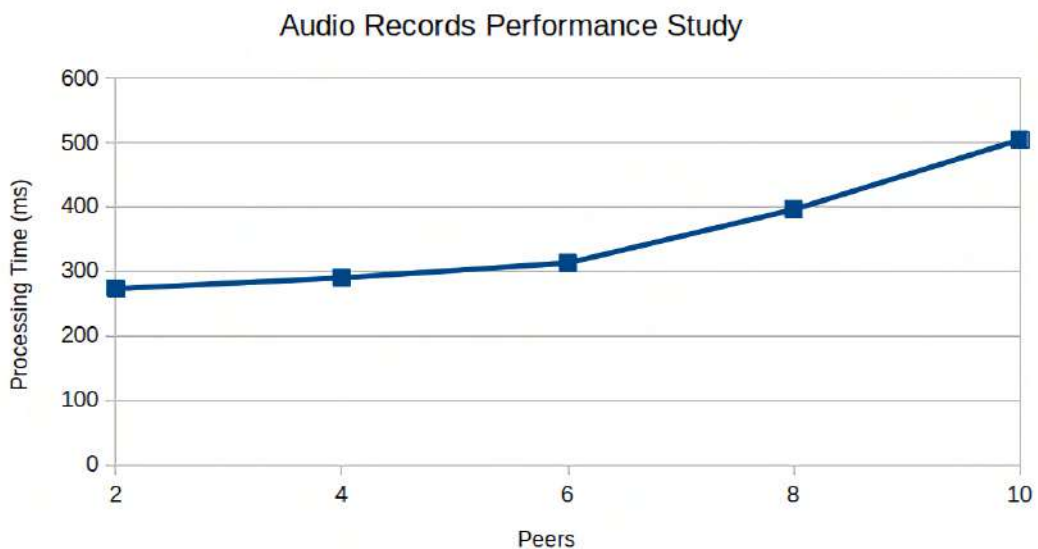


Figure 47: Audio recordings performance study.

The processing time depends more on the recording length than on the peer number. These results show affordable audio records processing time performance, as waiting a few seconds after the room closure is not a significant problem.

Final Flowescent main page

The image shows the Flowescent home page. At the top, there are two icons: a blue money bag with a white Euro symbol (€) and a blue alarm clock with a white checkmark inside. To the right of these icons is the text "Save money and time". Below this text is a paragraph: "Flowescent is the smartest meeting solution. One click and start as many meetings as you want. Record sessions and keep an activity history. And no more money and time spent." Below the paragraph is a link: "MORE REASONS TO SAVE >".

Below the icons and text is a section titled "Service Plans". Underneath the title is the text: "Sign up to our community by selecting your desired service plan".

There is a three-step process bar above the plans:

- 1 Choose your plan
- 2 Sign up
- 3 Terms & Conditions

The service plans are presented in three columns:

- Starter**
9.99€/mo
20 users included
3 users per room
No support
No recording functionality
START 30 DAYS FREE TRIAL
- Standard** (Most popular)
14.99€/mo
50 users included
8 users per room
Basic email support
No recording functionality
START 30 DAYS FREE TRIAL
- Premium**
29.99€/mo
1000 users included
20 users per room
Priority support
Recording functionality
GET STARTED

Figure 48: Flowescent home page visualization.

11 Conclusions

During the project elaboration, it has been possible implementing the theoretical concepts learned at all degree subjects, especially on the IT branch ones. The following description analyzes the most relevant degree subjects that provided the project elaboration necessary knowledge:

- **Web environment programming:** This subject provides enough knowledge to perform a correct design for the *Flouescent* web interfaces.
- **Security and data protection:** This subject provides enough knowledge to secure all users and company data correctly at the *Flouescent* server.
- **System management:** This subject provides enough system administrator skills to establish and maintain the production environment.
- **Backbone networks and public data services & Networks and Internet services configuration and maintenance:** These subjects provide enough knowledge to study, design, and implement technologies to use at Multimedia Flow Switch Server (*MFSS*).
- **Paradigms and programming languages:** The knowledge acquired in this subject together with prior experience programming languages is enough to perform complete studies about what technologies and programming languages should be used.
- **software development project:** This subject provides previous experience in software project methodologies useful for this project.
- **Operating systems project:** Operating systems experience is typically necessary for computer engineers and software developers, and this subject provides enough knowledge for it.
- **Legislation and professional ethics:** Especially since the GDPR effective date in Europe, it is essential for a software developer to meet all legal requirements, and this subject provides enough knowledge for it.

Each degree subject provides a specific area of knowledge, and it is relevant to find a way to use all acquired knowledge by implementing and joining some different theoretical concepts. It would seem easy at a glance, although there is a deep learning phase between theoretical concepts and their practical implementations.

I think that this project is essential to pass from theoretical knowledge acquired during the degree to their practical implementations in an IT project. There are many learning and research processes behind this project, as explained along all report sections, resulting in the final *Flouescent* application. It contains studies, design, and implementations for the Multimedia Flow Switch Server (*MFSS*) and the Meetings Web-based Application (*MWA*). However, the backbone of the project is *MFSS*. The *MWA* provides a basis and an interesting commercial application example that uses *MFSS* services. All the company, user, and service plan management were not crucial. However, many efforts have been taken to provide a useful case of use. The resulting *MWA* is simple and confined, although it is stable and adequate. As it has to be made to demonstrate the correct *MFSS* operation, it is made successfully.

Initially, the business feasibility study, marketing aspects, and business models were not contemplated. However, I have found these business-related aspects genuinely interesting, adding them as a project objective. The idea was not to perform structured and complete studies, but to use some logical procedures for trying to give more opportunities to *MWA* to succeed in the real market.

Results analysis

The *Flouescent* software degree of achievement is exposed based on project objectives and functional requirements. They may be proven through testing at the live demo application. **Table 27** shows the degree of achievement based on objectives and **Table 28** shows it based on both functional and non-functional system requirements.

| N | Objective | Degree of achievement |
|---|---|-----------------------|
| 1 | Exploring and studying modern and efficient technologies for use in the <i>MFSS</i> . | ✓ |
| 2 | Finding <i>MFSS</i> core tools that fill the project purpose or implement a new one. | ✓ |
| 3 | Developing a simple and functional application (<i>MWA</i>) <i>SaaS</i> -based to provide <i>MFSS</i> meetings services to those companies who want it. | ✓ |
| 4 | Deciding some commercial aspects of <i>MWA</i> , such as names, value proposition and service plans. | ✓ |
| 5 | Providing understandable code implementations and complete documentation. | ✓ |

Table 27: Degree of achievement: Project objectives.

| Functional Requirement | Degree of achievement | Non-functional Requirement | Degree of achievement |
|------------------------|-----------------------|----------------------------|-----------------------|
| FR-1 | ✓ | NFR-1 | ✓ |
| FR-2 | ✓ | NFR-2 | ✓ |
| FR-3 | ✓ | NFR-3 | ✓ |
| FR-4 | ✓ | NFR-4 | ✓ |
| FR-5 | ✓ | NFR-5 | ! ¹ |
| FR-6 | ✓ | NFR-6 | ✓ |
| FR-7 | ✓ | NFR-7 | ! ² |
| FR-8 | ✓ | NFR-8 | ✓ |
| FR-9 | ✓ | NFR-9 | ! ³ |
| FR-10 | ✓ | NFR-10 | ✓ |
| FR-11 | ✓ | NFR-11 | ✓ |
| FR-12 | ✓ | NFR-12 | ✓ |
| FR-13 | ✓ | NFR-13 | ✓ |
| FR-14 | ✓ | NFR-14 | ✓ |
| FR-15 | ✓ | NFR-15 | ! ⁴ |
| FR-16 | ✓ | NFR-16 | ✓ |
| FR-17 | ✓ | | |
| FR-18 | ✓ | | |
| FR-19 | ✓ | | |
| FR-20 | ✓ | | |
| FR-21 | ✓ | | |
| FR-22 | ✓ | | |
| FR-23 | ✓ | | |
| FR-24 | ✓ | | |
| FR-25 | ✓ | | |
| FR-26 | ✓ | | |
| FR-27 | ✓ | | |
| FR-28 | ✓ | | |

Table 28: Degree of achievement: System requirements.

¹MediaSoup SFU can work under limited hardware systems (the recording system can be affected).

²Users credentials are cryptographically secured (encrypted meetings have not been contemplated).

³All data remains intact in case of recoverable failure, as data is not backed up.

⁴All documentation is developed in a schematic form, following the PXP methodology guidelines.

Almost all project objectives and system requirements have been fulfilled, resulting in a complete project focused mainly on IT and systems management. The *SaaS*-based application provides a minimally applied case example for the Multimedia Flow Switch Server (*MFSS*). The *MFSS* is the principal project purpose and the most complex part. Another complicated part is the recordings system, having to store, synchronize, and merge all media flows. Moreover, it provides the marketing and business feasibility studies as important purposes, exposed in the project objectives section.

Complications

As the final degree project is time-limited, it is not possible to guarantee a completely stable operational application. During the project implementation, some problems and unexpected behaviors have been appearing, especially in the server configurations, recordings system, and service plans management. Furthermore, room logic is limited, allowing only the participant user role. User roles are an interesting additional point, that would add more flexibility (e.g., broadcaster, listener).

Flouescent is fully compatible with *Chrome* and *Firefox* browsers. The system operation can be no guaranteed under other browsers, as not all browsers implement WebRTC technology. For this reason, some browsers do not allow using *Flouescent* room conferencing functionality, although the other application modules should work without problems at almost 99.8% of browsers.

The problem with service plan templates is that *Flouescent* administrators would like to change them, for example, when the market or costumers environments change. This functionality is not trivial, since all existent companies that have a modified service plan may not want to replace their conditions. There is no solution to please both *Flouescent* administrators and companies. The most affordable way seems to provide to affected companies the option of maintaining their old conditions, applying only the new service plan changes to new clients. This solution has unexpected behaviors in some cases. For example, the system does not notice affected companies to update their plans.

The recordings system does not work as stable as expected. The room configuration environment where users join and leave at any moment causes significant problems trying to implement a robust system. Only complete testing of all unlimited conference configuration environments can provide a stable system. The final *Flouescent* application handles video recordings correctly at 95% of possible cases, and accurately audio recordings at 90%, being the audio more unstable to synchronize. It is important to note that users' physical devices can experience malfunctions that can be translated to a recording processing or synchronization issues impossible to avoid by the system.

Acknowledgement

I am thankful to my project supervisor Dr. Jose Luis Marzo, who provided the expertise that greatly assisted the project. I am also grateful to Dr. Immaculada Boada for assistance with project planning and feasibility studies, and Dr. Josep Soler, who helped with some initial project report structuration doubts. I would not forget to remember my family and my partner, for their continuous support, especially during the last project development weeks, which I was to work hard.

12 Future Work

There are opportunities and possibilities to improve *Flouescent*. All the work done built a solid backbone from which *Flouescent* can grow up. The following list exposes some interesting feasible functionalities that can be developed:

1. **More user in-room roles:** Currently, *Flouescent* only supports the participant user role, which forces users to produce both audio and video. It would be interesting to add more in-room roles such as listener. Listener users would join rooms without the necessity of producing their audio or video. Besides, the participant role can be updated to allow only video or only audio producers, providing a mechanism that enables media producers to pause and resume at any time.
2. **Improve browser support:** According to *Statcounter GlobalStats* browser market share in Europe [37], both *Chrome* and *Firefox* browsers have a 68.47% of market share. Although it is a decent value, it is not sufficient. It would be interesting to provide at least *Safari* support, which can grow the compatibility percentage to 85.29%.
3. **≈100% stable recording system:** As explained previously at *Conslusions* section, it is not trivial to build a complete stable recording system due to its complexity and the room environment variability. For this reason, recordings are limited to nine participants for the previous reasons and CPU limitations. Independent servers which host the recording system and performing load balancing would be a solution to avoid this problem, adding as many CPU resources as the system needs.
4. **Screen sharing:** This functionality is not trivial, and its implementation is complex. A screen sharing system would add considerable value to room meetings and opens *Flouescent* up to new markets.
5. **Service plan templates improvement:** The service plan changeable templates system has limitations, and modified plans can enter into conflict with current companies hired service plans. Another improvement would be to stabilize this system and allowing companies to update their current service plan with its new version.
6. **Mobile application:** *Flouescent* is a cross-platform web application that can be accessed through many devices with a browser and an Internet connection. However, a mobile application provides additional benefits, especially for usability, and a more proper responsive design. The decision of making a web-based application was taken to provide a cross-platform solution, taking into account that the project time is limited.

13 Bibliography

This bibliography is written following the APA (American Psychological Association) style.

References

- [1] 9 Project Management Methodologies Made Simple. (2017, March 2). Retrieved April 5, 2019, from <https://thedigitalprojectmanager.com/project-management-methodologies-made-simple/>
- [2] Agile Project Management. Retrieved April 5, 2019, from <https://www.atlassian.com/agile/project-management>
- [3] Dzhurov, Y., Krasteva, I., & Ilieva, S. (2009). Personal Extreme Programming – An Agile Process for Autonomous Developers. URL: <https://research.uni-sofia.bg/handle/10506/251>
- [4] Project Management Institute (2008). A Guide to the Project Management Body of Knowledge Retrieved (PMBOK® Guide) - Fourth Edition. Retrieved from https://www.works.gov.bh/English/ourstrategy/Project%20Management/Documents/Other%20PM%20Resources/PMBOKGuideFourthEdition_protected.pdf
- [5] Project Management Plan Template. Retrieved April 6, 2019, from <http://www.projectmanagementdocs.com.vhost.zerolag.com/project-planning-templates/project-management-plan.html>
- [6] Intuitive and Beautiful Project Planning. Retrieved April 26, 2019, from <https://www.teamgantt.com/>
- [7] Annual software Engineer Salaries in Europe, February 2019. Retrieved April 23, 2019, from <https://www.daxx.com/blog/development-trends/it-salaries-software-developer-trends-2019>
- [8] Estimate My App. Retrieved April 23, 2019, from <https://estimatemyapp.com/>
- [9] How accurate are app estimates like estimatemyapp. Retrieved April 23, 2019, from <https://www.quora.com/How-accurate-are-app-estimates-like-estimatemyapp-or-venturepact>
- [10] Git Documentation Glossary. Retrieved April 26, 2019, from <https://git-scm.com/docs/gitglossary>
- [11] Getting Started - Git basics. Retrieved April 24, 2019, from <https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61>
- [12] SSH (Secure Shell). Retrieved April 24, 2019, from <https://www.ssh.com/ssh/>
- [13] Choosing the best code editor as a web developer in 2019. Retrieved April 24, 2019, from <https://designrevision.com/best-code-editor/>
- [14] An introduction to LaTeX. Retrieved April 24, 2019, from <https://www.latex-project.org/about/>
- [15] Skala, W. (2018). Drawing Gantt Charts in LATEX with TikZ. The pgfgantt Package. URL: <http://bay.uchicago.edu/CTAN/graphics/pgf/contrib/pgfgantt/pgfgantt.pdf>
- [16] LibreOffice Writer. Retrieved April 24, 2019, from <https://www.libreoffice.org/discover/writer/>
- [17] Broadband Communications and Distributed Systems. Retrieved April 24, 2019, from <https://bcds.udg.edu/>
- [18] Doxygen: Documenting the code. Retrieved April 26, 2019, from <http://www.doxygen.nl/manual/docblocks.html>
- [19] Jakobsson, C. (2015). Peer-to-peer communication in web browsers using WebRTC A detailed overview of WebRTC and what security and network concerns exists. URL: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A852726&dswid=-7979>

- [20] Understanding WebRTC Media Connections: ICE, STUN and TURN. Retrieved April 29, 2019, from <https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html>
- [21] Overview of WebRTC Media Servers. Retrieved April 29, 2019, from <http://webrtcbydralex.com/index.php/2016/12/13/overview-of-webrtc-media-servers/>
- [22] MediaSoup. Retrieved April 29, 2019, from <https://mediasoup.org/>
- [23] mediasoup-demo v3. Retrieved April 24, 2019, from <https://github.com/versatica/mediasoup-demo>
- [24] JavaScript HTML DOM. Retrieved April 30, 2019, from https://www.w3schools.com/js/js_htmldom.asp
- [25] Top 11 JavaScript Frameworks for 2019. Retrieved April 30, 2019, from <https://www.lambdatest.com/blog/top-javascript-frameworks-for-2019/>
- [26] Business name generator. Retrieved May 5, 2019, from <https://www.businessnamegenerators.com/>
- [27] material-sense. Retrieved May 1, 2019, from <https://github.com/alexanmtz/material-sense>
- [28] Gimp software. Retrieved May 5, 2019, from <https://www.gimp.org/about/>
- [29] Library 3 AM font. Retrieved May 7, 2019, from <https://www.fontspace.com/igor-kosinsky/library-3-am>
- [30] What is a Feasibility Study?. Retrieved May 16, 2019, from <https://www.extension.iastate.edu/AGDm/wholefarm/html/c5-65.html>
- [31] Cifras PyME Empresas. Retrieved May 18, 2019, from <http://www.ipyme.org/es-ES/ApWeb/EstadisticasPYME/Documents/CifrasPYME-abril2019.pdf>
- [32] Restful API Designing guidelines - The best practices. Retrieved May 18, 2019, from <https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9>
- [33] OpenSSL - Cryptography and SSL/TLS toolkit. Retrieved May 21, 2019, from <https://www.openssl.org/>
- [34] Ffmpeg - A complete, cross-platform solution to record, convert and stream audio and video. Retrieved May 22, 2019, from <https://ffmpeg.org/>
- [35] Setup Let's Encrypt for Express. Retrieved May 23, 2019, from <https://flaviocopes.com/express-letsencrypt-ssl/>
- [36] Hetzner online. Retrieved May 27, 2019, from <https://www.hetzner.com/>
- [37] Browser Market Share in Europe. Retrieved May 28, 2019, from <http://gs.statcounter.com/browser-market-share/all/europe>

14 User manual

The user manual has been made following the project Personal Extreme Programming (*PXP*) methodology. As it detracts the documentation process, it is explained following a schematic and precise steps structure. This manual is a part of the **S8** sub-project.

As the implemented application is *SaaS*-based, only the commercialization company has to deploy or install *Flouescent* on an own server following the steps provided in the *Deployment and results* section. The software is cloud-based, and users do not have to install specific software, as they only need an Internet connection and a browser supporting JavaScript. As two authorization user types exist, both admin and normal users have their manual.

14.1 Normal user manual

Application access

1. **Account creation:** To get access at *Flouescent* services, you have to ask for an account to your company managers.
2. **Account validation:** Once your company managers create your account, you will receive an email with instructions to validate your account and set your log-in credentials. You must read and accept our legal terms of service and data privacy policy.
3. **Access:** Once your account is validated, you can log in to *Flouescent* application from any device at *Sign in* menu item providing your account credentials.
4. **Logout:** You can logout from a previously logged session easily at *Profile* menu item, clicking the *Logout* button.

Meeting conferencing

1. **Start a new meeting:** You can start a new meeting at any moment going to the *Dashboard* menu item and click *Create* button. Furthermore, you can specify a room name.
2. **Join an existing meeting:** You can join an open meeting if you are authorized and you provide the invitation code at any moment going to the *Dashboard* menu item. You have only to insert the code at *Join room* input and click the *Join* button.
3. **Change your default room settings:** You can change at any moment your room creation authorization, recordings type, and recordings permissions going to the *Dashboard* menu item and clicking the *Show settings* button.
4. **Visualize room history:** You can visualize your room creations and participations history going to the *Dashboard* menu item, under room creation and joining. If you have permission, audio and video recordings button will be enabled, and you can visualize or reproduce them as easy as a click.

Profile customizing

1. **Customize user data:** You can change your profile data such as email, username, name and surname at any time in the *Profile* menu item *Update data* form. You only have to modify the desired fields and then click the *Update data* button. Please note that the email address is unique and you cannot use it at multiple *Flouescent* accounts.
2. **Change your password:** You can change your password at any time in the *Profile* menu item *Update password* form. You only have to provide the old and new password, and then click the *Update password* button. Please note that the password must contain at least eight characters.

Get more application information

1. **About us menu:** You can consult at any moment more information about *Flouescent* services in the *About us* menu item.

Consult our legal terms

1. **About us menu:** Feel free to consult our legal terms in the *Legal* menu item.

14.2 Admin user manual

Admin users can consult the normal user manual, as they can perform all normal user actions. Admin users are considered as company managers that hire *Flouescent* services.

Join *Flouescent* community

1. **Join us:** Feel free to apply for your desired service plan in the *Home* menu item. You only have to click your desired service plan, insert the required company necessary data, and start enjoying our 30 days free trial.

Manage your company

1. **Change your service plan:** You can change your company service plan at any moment by clicking *Downgrade to* and *Upgrade to* buttons as desired at *Company* menu item.
2. **Create users:** You can create as many users as your plan allows at *Users manager* table at *Company* menu item. You have to insert the new user valid email, and optionally his/her name, surname, and permissions (normal user by default).
3. **Edit users authorization:** You can edit at any moment users authorization at *Users manager* table at *Company* menu item. You only have to click the *Edit* button at the desired user row, modify the authorization column, and then save changes by clicking the *Save* button.
4. **Users deletion:** You can delete users at any time at *Users manager* table inn the *Company* menu item. You only have to click the *Delete* button at the desired user row, and then confirm the action by clicking the *Save* button.