STARS GUI: A Web Application for the Analysis of Astronomical Schedules

Néstor Campos Gestal

Resumen—El objetivo principal de las observaciones astronómicas es observar la mayor cantidad de planetas en un periodo de tiempo fijado. Para ello, la mayoría de los telescopios tienen un planificador, que es un software que planifica las diferentes tareas/planetas que debe de observar el telescopio en un periodo de tiempo fijado teniendo en cuenta restricciones, que un científico pueda dar para una determinada observación como, por ejemplo, la fase lunar o la luminosidad del sol. El Instituto de Ciencias del Espacio (ICE) está involucrado en una gran variedad de proyectos de planificadores para telescopios terrestres y en satélites. El problema es que el resultado que sacan estos planificadores es difícil de entender, por ello surge la necesidad de crear una herramienta de visualización que permita entender fácilmente el resultado de los planificadores. Este artículo presenta el proceso de desarrollo para crear una aplicación web que permita representar el resultado de los planificadores astronómicos de una forma fácil de entender a través de información estadística y gráficos de los planetas en la planificación.

Palabras clave— Planificaciones Astronómicas, Telescopios, Planificación, Aplicaciones Web, JavaScript, NodeJS, VueJS, Bootstrap, Element Plus, PostgreSQL, SQL, HTML, CSS

Abstract— The main objective of astronomical observations is to observe as many planets as possible in a given period of time. To do this, most telescopes have a scheduler, which is a software that plans the different tasks/planets that the telescope must observe in a given period of time taking into account constraints, that a scientist can give for a certain observation such as the moon phase or the luminosity of the sun. The Institute of Space Sciences (ICE) is involved in a wide variety of scheduling projects for space and ground-based telescopes. The problem is that the result that these planners give is difficult to understand, therefore, there is a need to create a visualization tool that allows to easily understand the result of the schedulers. This article introduces the development process for creating a web application that can represent the output of astronomical schedulers in an easy-to-understand way through statistical information and plots of targets included in the plan.

Index Terms—Astronomical Scheduling, Telescopes, Scheduling, Web Applications, JavaScript, NodeJS, VueJS, Bootstrap, Element Plus, PostgreSQL, SQL, HTML, CSS

1 Introduction

THE automatic planning and scheduling of astronomical observations from space and ground based observatories have become extremely important for large surveys because they facilitate the coordination of multiple instruments and observatories located at different parts, they provide a fast and dynamic reaction to changes in the environmental conditions, and they try to maximize the scientific return [1].

The main objective of astronomical observations is to efficiently observe different targets/celestial bodies in the sky. A telescope could have many targets planned to observe in one night or a given period. The objective is to plan the observations of as many targets as possible, spending the smallest time possible between them and taking into consideration the constraints it should have. These constraints are mainly related with the visibility of targets, the cadence of observations, ephemerides of astronomical events, moon phase, etc. Due to the large number of parameters that need to be taken into account, most of the observatories/telescopes have a scheduler. A scheduler is a software tool that plans when tasks should be executed based on some criteria to order them. It identifies whether adequate resources are available to carry out the plan (e.g. the type of telescope, its location, the instrument), specifies the date and time of the tasks to be performed, considering the constraints (e.g. elevation of target, moon phase, angular distance) required for each target, and provide an optimized observation plan to the telescope.

In order to facilitate the understanding of the results provided by the scheduler, a visualization planning tool is necessary. The goal of this paper is to design, develop and test a tool essential not only to display the plan, but to allow scientists to analyze it from an astronomical point of view in a graphical and more clearly way by showing statistics and plots of targets included in the plan.

The structure of this article is as follows, chapter 2 describes the state of the art and background of the project. Chapter 3 describes the objectives of the project. Chapter

- *Contact e-mail: Nestor.Campos@autonoma.cat*
- *Mention: Information Technology*
- *Academic advisor: Víctor García Font (Departament d'Enginyeria de la Informació i de les Comunicacions Area de Ciències de la Computació i Intel·ligència Artificial)*
- *Course 2021/22*

4 explains the methodology and planning, followed by the development cycle (Analysis, Design, Develop and Testing) in chapters 5, 6, 7 and 8. Then a discussion of the results is described in chapter 9 and to finalize, the conclusions and future lines of work of the project in chapters 10 and 11.

2 Background

The Institute of Space Sciences (ICE) is involved in a several scheduling projects. ICE is developing a software framework that provides scheduling tools for the planning of observations and the statistical analysis of produced plans which is called STARS (Scheduling Telescopes as Autonomous Robotic System).

This software framework is composed by:

- STARS Core: a C++ library that provides a general abstraction of the scheduling data model and algorithms
- STARS Pre-Scheduler: it receives a list with the targets to be observed and its constraints, and it provides the possible windows of time to observe the target
- STARS Scheduler: a scheduler which implementation is based on STARS Core. It receives the parameters provided by the Pre-Scheduler and it makes and optimized schedule using artificial intelligence algorithms. ICE has implemented different schedulers for different projects, ground-based telescopes and spacecraft-based telescopes

The goal of this work is to upgrade an outdated visualization tool developed by ICE, in order to adapt it to new requirements. Since being an application developed in JAVA, it has the disadvantage that it must be installed on a local computer, and it is also not accessible from anywhere for any type of device.

For this reason, explained above, it arises the need to create a web application that allows the scientist to analyze schedules with different views, diagrams, and statistical data with customizable graphics. In addition, this application will allow having different roles within the system which will allow to be a distribution of responsibilities among users. From now on, we will call it STARS GUI (STARS Graphical User Interface).

One of the advantages of creating a web application is that it will be accessible from everywhere, it won't have to be installed on a local computer requiring the installation of some program and it will be always updated.

3 Objectives

The main objective of this project is to design and develop a visualization planning tool to analyze and compute efficient observational plans (we can call it also optimized schedule) for telescopes in space and ground-based astronomical observatories. Therefore, to achieve this goal, a set of sub-objectives need to be satisfied:

- Analyze the project's viability and plan its development through an agile philosophy (Low priority)
- Investigate and study the current technology involved in the outdated visualization tool. In order to understand from firsthand how it is designed and how it works (Low priority).
- Collect the requirements of the project with the stakeholders (High priority)
- Implement a backend architecture to receive requests from the client and process the schedule fast. This backend should use C++ executables from the schedulers developed by ICE to compute and generate optimized schedules (High priority)
- Implement a frontend architecture to display the results provided by the backend providing statistical information and charts so the physicist can analyze schedules (High Priority)

4 Methodology

The methodology used in this project is the agile methodology. Agile is among today's most popular software development methodologies, accepted in all-size companies [2].

The point of using Agile is that having multiple sprints throughout the project will allow us to change direction as needed. The fact that STARS GUI will depend on several projects, will lead to a large number of constant changes, in which new requirements will arise from time to time that will have to be added and implemented, so it is essential to have the flexibility of this methodology.

4.1 Description

The phases during every sprint consist of the four typical stages of software development: Analysis, Design, Develop, and Testing.

- Analyze: Define the requirements necessary to fulfill the functionality that is requested
- Design: Once the requirements have been obtained, and we know what to do, this phase consists on studying how to implement a functionality
- Develop: Developing the software following the previous steps
- Test: Test the different bugs of the implementation already done and check if it matches the requirements

4.2 Planning

The project was planned with a total of 3 sprints where Sprint 1 was planned to finish for the first progress delivery (November 14th) and Sprint 2 and 3 finished for the second progress delivery (December 19th). The duration of every sprint was:

- Sprint 1: 5 weeks (Oct 11-15, Oct 18-22, Oct 25-29, Nov 1-5, Nov 8-12)
- Sprint 2: 2 weeks (Nov 15-19, Nov 22-25)
- Sprint 3: 3 weeks (Nov 29 – Dec 3, Dec 6-10, Dec 13-17)

We can see these sprints and a breakdown in how each of the functionalities have been developed in the Gantt

diagrams of Fig. 1.

After the second progress delivery, a couple of weeks were dedicated to create the final paper and documentation of the project and to finish implementing some functionalities that were not completely done in the last sprints.

After the final report delivery, 2 weeks were to prepare the slides to present the project in public and to finish this paper of the project.

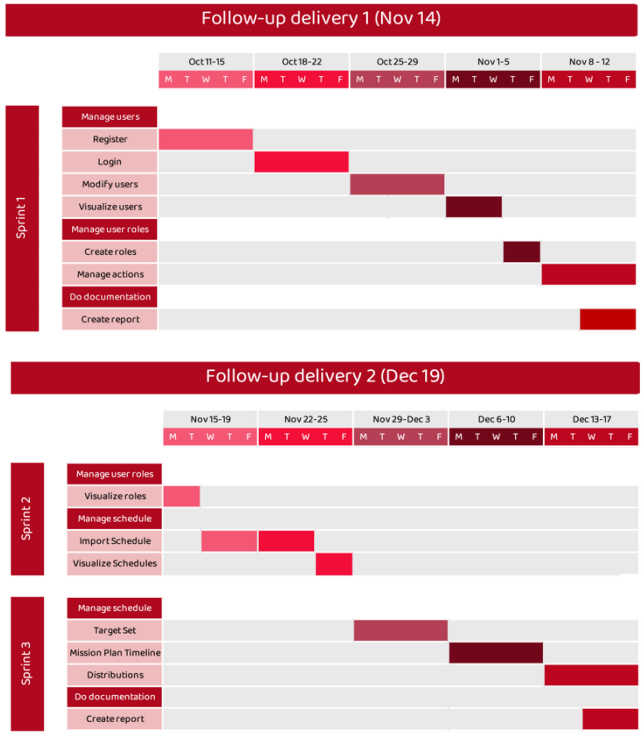

Fig. 1: Gantt Diagrams

5 Requirement Analysis

For the analysis phase of the project, several meetings have been held with stakeholders (people from different projects that will use STARS GUI in the future) to identify and capture requirements. All the requirements have been described and specified in the SRS (Software Requirement Specification) document, where requirements have been added as the project progressed in the sprints. Below are some of the more general and summarized requirements.

- The STARS GUI system shall manage user roles
- When the user connects to STARS GUI system, it shall provide the user with the ability to log in. Username and password shall be used for the authentication.
- When a user registers in the system filling out a form. An email is sent to the user for confirmation. The user should click in the link sent on the email to validate its account
- Any user recently registered and authenticated in the system will have the role of User (without permissions). This is a user that can't do any actions in the system, only login, log-out and view the profile.
- The user roles of the system shall be User (without permissions), User, Project Manager and Administrator
- Once a user is authenticated and registered, a user with the role of Project Manager or Administrator should give the user permissions to do actions in the system
- If successful login, the system shall associate the user with the user roles/privileges and configure the GUI according to the user's profile
- The system shall let the user import a schedule
- The system shall let the user see the list of schedules
- The system shall let the user display a target set distribution
- The system shall let the user to generate the mission plan timeline letting the user to modify the dates
- The system shall let the user to generate histograms indicating the number of observations per target, the distribution of "exposure times", the distribution of waiting time between observations and let the user compare these histograms with other schedules

6 Design

This chapter deals with the design of the application. The following sections explain the use cases of the system and the general architecture of the system that is composed by a backend and a frontend architecture. To save all the data of the user, a database is designed that is part of the backend architecture.

6.1 Use cases

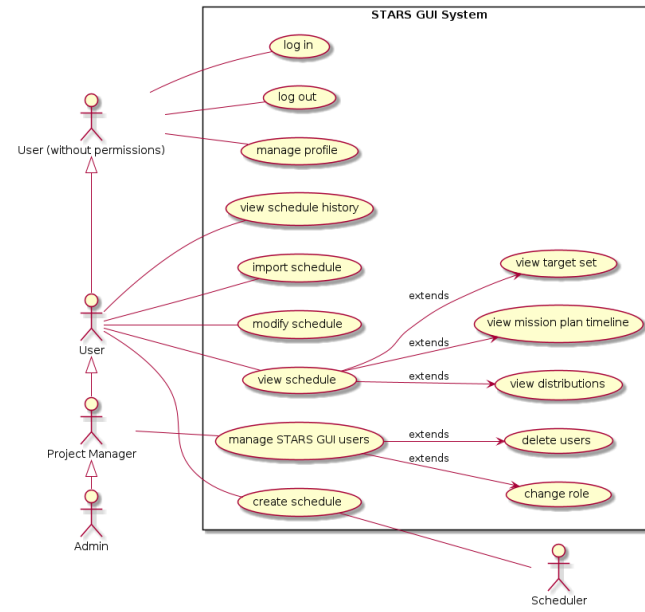

Fig. 2: Use cases diagram

Above in Fig. 2 are the actions that the system can perform and how the different users in the system interact with it.

The user will have the typical login and logout operations as well as modify their profile in the system. In addition, the user can import a schedule already optimized in a specific file format that he/she has in the local computer and see the different graphics that the system offers (target set, mission plan timeline and distributions). The user can create an optimized schedule using a scheduler, for this the web application will have to use an executable. The user can see a list of the different schedules and modify them. Finally, the admin and project manager can modify the status and role of any user in the system. For example, when a user register into the system and it has already validated its account, one administrator or project manager should change its role to User so the user can use the functionalities of the system.

6.2 Architecture

The system is based on a client-server architecture shown at Fig. 3 where on one side we have the frontend that will send requests to the server and get responses from it. On other side, we have the backend which is composed by the server that will manage all the petitions made by the font-end and do queries to the database when needed. The server will also have the possibility to use the executables provided by the scheduler when needed to optimize a plan (this function is still not implemented).

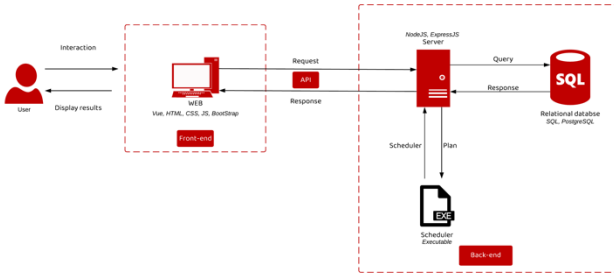

Fig. 3: System Architecture

6.3 Backend Design

In this project, we are going to use the popular architectural pattern known as MVC (Model-View-Controller) but with some modifications.

MVC enables to split the web application into specific sections that have their own individual purpose. For example, we will have a section where we will have all the queries to the database. This will make the code much easier to read and more reusable if we want to do the same query in different parts of the code. This also makes the application more scalable, and it lets other people recognize every part of the application easily. Got to section A.1 in the appendix for more information on how MVC works.

As shown in Fig. 4, the backend of the web application is composed by the next modules:

- Views: all the content in this module will be provided by the front-end
- Router: this module will manage all the routes of the application and redirect the requests to different controllers. The point of having a router is that we can have different controllers for one route.
- Controller: in this module we will process all the information using the Model when needed
- Model: this module will interact with the Database and the Scheduler when needed

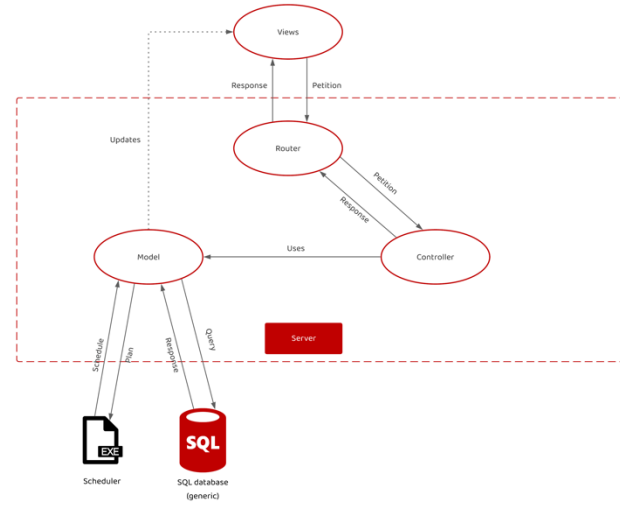

Fig. 4: Backend Architecture

6.4 Frontend Design

For the design of the frontend of the application, meetings have been held with the stakeholders to discuss about the layout and design of the page. During these meetings a paper prototype has also been made using a graphic program which has served to have a first idea on how the design of the application shall be.

The front end will be based on views and web components. These components encapsulate the client code (HTML, CSS, JavaScript) so it can be reused and scale depending on which functionalities are going to be added.

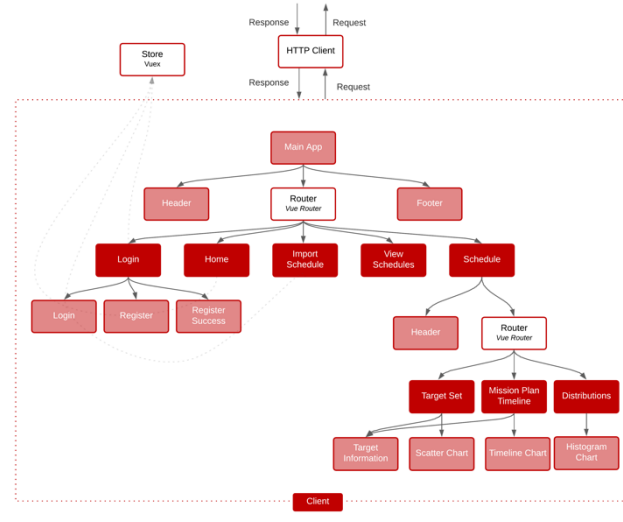

Fig. 5: Frontend Architecture

Fig. 5 shows the design of the frontend architecture and how the different views and components will be related. It is important to make clear the difference between component and view. We will understand a view as a "page" of the web app, and a component as a small box/part of the view.

In Fig. 5, boxes with a strong red color are the views while boxes with a softer red tone are the components.

The client will have the main component/view (Main App) that will be composed by the header and the footer of the web page. We will also have a router that will handle the frontend framework itself to display different views depending on the user's actions. A better example of how this process works is shown in Fig. 6.

The Store will serve as a centralized store of information for all the components in the application, this will serve to communicate and share information between different components, (for example, if we want to pass information from the scatter chart component to the target information component, the target information component will use the store to get the information that the scatter chart component has put there).

The HTTP Client will be used from any component to send a request to the server.

The Main App component will be the principal component of the application, it can be understood as if it were the root of the tree. This component is composed by the next elements:

- Header Component: this component is composed by a navigation bar with different options
- Router: this will be a plugin of the frontend framework. Its function is to show one component/view or another depending on which option of the menu bar the user clicks, we can see in this example how the different elements of the navbar in the header component have the tag `<router-link>`, and the `:to` attribute is the component that the router should show the user
- Footer Component: this will be used to show typical information that goes in the footer of the websites

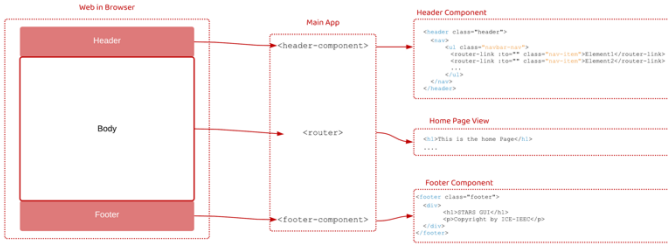

Fig. 6: Frontend component example

6.5 Database

One of the most important elements of a web application is the design of the database because it is where all the user's data is stored. Therefore, it is important that this is a safe element.

This project uses a relational database. One of the main points of relational databases is their referential integrity. Referential integrity refers to the accuracy and consistency of data. This data integrity is achieved by using these primary and foreign keys.

Another advantage and important point for our project is that relational databases have easy data navigation, and the data is well structured. This is important because this project will be for the consumption of scientist, and most scientist doesn't have experience in technology concepts, and probably most of them have worked at some point with relational databases rather than non-relational databases.

6.5.1 Database Design

At Fig. 7 below, we can see the entity-relationship model of the database. We can see the relations between the different entities. It is important to remark that a user could pertain to several projects, and the projects could have several users, but a user can have only one role in a project. Currently the relationship that users can belong to different projects is not implemented because this has to be studied in more detail in the future.

To manage the roles of the users, there are two entities to do it, the entity User Role which contains the different types of roles that a user can have, and the Permissions entity which contains the different actions the user can do such as Read, Write, See Schedules…

When a user imports a schedule to the system, the server saves this file in a folder in the file system, and it saves the path, name and other details of this file in the database.

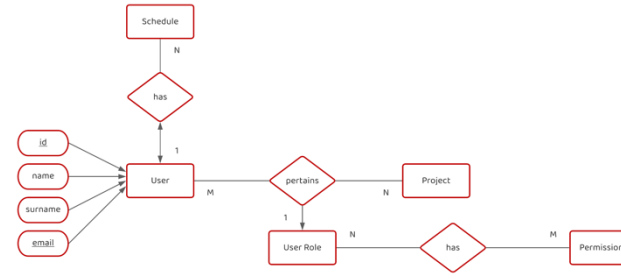

Fig. 7: ER model of the STARS GUI Database

7 Implementation

In this phase is where the coding part is done. To do it, the design presented in the previous section was applied.

In this chapter, firstly, we present how was the development process and which tools were used for coding.

Secondly, we explain the implementation using the design of the previous sections in chapter 6, these sections are the backend, frontend and database. In each of these sections we first make an explanation of the technologies used, then an explanation of the internal structure and finally an explanation of what each of the components of the structure do.

7.1 Development Process

All the developed code has been stored in a Gitlab repository on an ICE server. The advantage of using repositories

is that it contains the different versions of the application, having a history with the changes made on the original and on each new version.

For each new functionality a new branch was created which was revised before making the merge to the main branch.

We also used the interactive board that offers GitLab to carry follow-up of the functions to implement.

Visual Studio Code was used as the main IDE to develop code, with the addition of some plugins.

7.2 Backend

7.2.1 Technologies

The backend of this project is developed in NodeJS [4]. The appearance of NodeJS has brought about a revolution in the world of JavaScript (JS). Node.js is an open-source environment designed to enable server-side JavaScript application development.

Some of the reasons why NodeJS has been chosen for the development of the application backend are the following [3]:

- Asynchronous and event driven.
- High performance
- Single threaded but highly scalable processes
- Open Source
- Active user community

7.2.2 Backend Folder Structure

Fig. 8 shows the structure of the backend directories. In each of these directories are the different components of the backend explained in the section 6.3 (Backend Design).

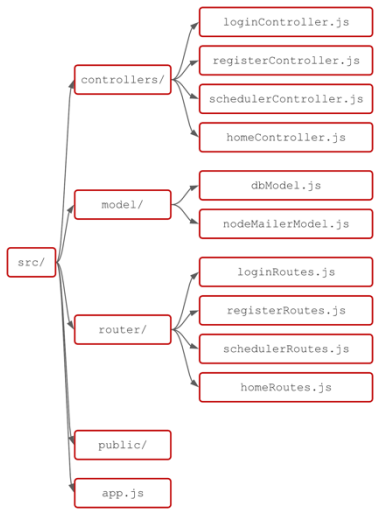

Fig. 8: Backend Folder Structure

These directories do the following functions:

- Controllers: in this folder we are going to have all the controllers of the project. For example, we have the loginController.js where we are going to manage all the requests for the login of a user.
- Model: in this folder we are going to have the model part. We can see we have the `dBModel.js` which contains all the information about how to connect to the database, it will do the queries to the database and it will be reuse by many controllers, and we also have the `nodeMailerModel.js` which will send the confirmation emails when a user registers into the system.
- Public: this folder is sent to the user and contains all the views of the application. This folder is generated by the frontend which will be explained in next section.
- Routes: this folder is where we are going to manage all the routing of the application.
- app.js: is the main file of the server, where we configure the server and where we start the routing. This file is where we get all the requests from the client.

7.2.3 Routing

Routing is one of the most important parts of the backend. It defines the way in which the client requests are handled by the application endpoints. So, it is important to understand this term in order to understand the workflow of the backend. There are several ways to implement routing in NodeJS, in this project we use ExpressJS for routing. For more information on how ExpressJS works go to appendix section A.2.

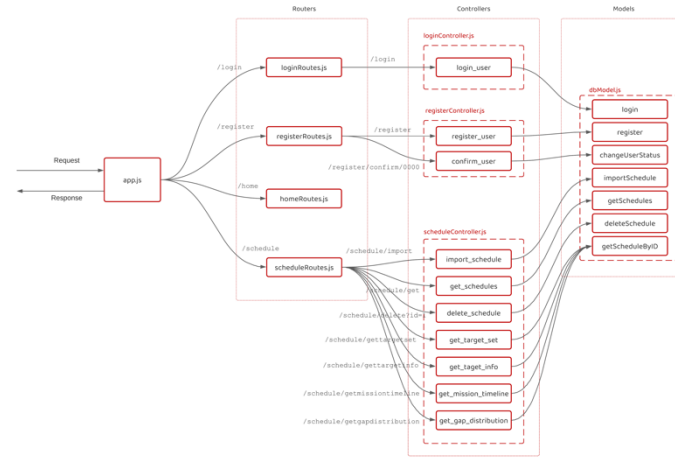

Fig. 9: Routing

In Fig. 9 we can see the routing process of this application. We can see the composition of each of the components of the backend, which client requests are received by each component and which other elements are used to process these requests.

The `app.js` file is the file that the server uses to run. This file catches all the requests and it uses the different routing elements provided by the Express-Routing (explained later) to route the requests depending on the path. Every router can have one or more controllers and these routers call the controllers to do the work.

For example, if a user clicks on the button of "Show Schedules", it will generate a request to the server with the path `/schedule/getschedules`, the `app.js` is going to catch this request and send it to the router that catches all

the `/schedule` requests. The Schedule router is going to use the Schedule Controller to use the module `get_schedule` which will use the database model to do a query to the database to get all the schedules done by the user.

7.2.4 Routes

As we have said before, the router folder contains all the router objects used to begin the routing process explained above, these routers send the information to the controllers to process the requests. We have the following routers:

- loginRoutes.js: will catch requests with `/login` at the beginning of the path. It uses the `loginController.js` to process the requests.
- registerRoutes.js: will catch requests with `/register` at the beginning of the path. It uses the `registerController.js` to process the requests.
- scheduleRoutes.js: will catch requests with `/schedule` at the beginning of the path. It uses the `scheduleController.js` to process the requests.
- homeRoutes.js: will catch requests with `/home` at the beginning of the path. It uses the `homeController.js` to process the requests.

7.2.5 Controllers

In the last section we explained how the sever uses the routes to get the information to the controllers. The controllers are where all the information is processed and where the server uses de models to access to the database or to other external resources. In this section we are going to explain how every controller treats the information received by the user. The controllers are:

- loginController.js: this controller gets the `/login` requests and logs in a user in the system, for this it makes a query to the database (using the `dbModel.js`) to see that the username and password data that the user gives matches correctly
- registerController.js: this controller gets the following requests and does the following actions:
  - If it receives a request with the path `/register`: registers a user in the system getting the information from a form that the user fills, and then saving this information in the database. It also sends a confirmation email so the user can validate the account. To do this, it uses the `nodeMalierModel.js`.
  - If it receives a request with the path `/register/confirm/XXX`: it changes the status of the user to a "validated" user when the user clicks on the confirmation email sended.
- schedulerController.js: this controller gets the following requests and does the following actions.
  - If it receives a request with the path `/schedule/import`: it receives an XML from the user, which is the optimized schedule. It transforms this XML file to a JSON file and saves both files in the server file system. It also saves in the database the path to these files and other information such as the name…
  - If it receives a request with the path `/schedule/getschedules`: it sends to the client the schedules that the user has imported
  - If it receives a request with the path `/schedule/deleteschedule`: it receives by the client the id of the schedule the user wants to delete. To do this, this module does a query to the database to know the path of file, once it knows where this schedule is, it deletes the file and also deletes all the information related in the database
  - If it receives a request with the path `/schedule/gettargetset`: it receives the id of the schedule to know what schedule should analyze and then, it returns to the client the data in the right format so the frontend can represent in a scatter chart the distribution of the planets in the sky
  - If it receives a request with the path `/schedule/getmissiontimeline`: it receives the id of the schedule to know what schedule should analyze and then, it calculates the number of observation tasks of the telescope, the number of calibrations tasks, the number of station keeping tasks and the number of gaps between each task and saves all this information in the right format to send it to the client so the frontend can represent this data in a timeline chart
  - If it receives a request with the path `/schedule/getdistribution`: it receives from the client information about the type of distribution wants to see and also the id of the schedule. It sends the information of the desired distribution to the client in the right format so the frontend can display this information in a histogram chart

7.3 Frontend

7.3.1 Technologies

The frontend of this project is developed in VueJS in version 3 [5]. Vue.js is a progressive JavaScript framework, which is used to build UIs (User Interfaces) and SPAs (Single-page Applications). I have chosen Vue because it is very easy to use if we compare it to its competitors such as Angular and React, and it has been designed for scalability and incrementality along with being easy to integrate with other libraries focused on the view layer, it has also good documentation. One of the main reasons for using Vue is that it is component-based. We write code in the form of components to import that component and reuse it wherever we need it.

For this project, we will use HTML5 and CSS3 to generate the different HTML files and make the different components that we are going to manage with Vue.

Bootstrap [6] is another framework that we are going to use in this project. It will make much easier to design

and make responsive our website.

ApexchartsJS [7] is the framework we are going to use to display the information with charts. It is very easy to use and it is compatible with Vue3.

Element Plus [8] is a UI framework that provides tools such as forms and other widgets to use in a website very easy without having to spend time in CSS to make the UI elements look good.

We will use VueRouter [9] to route and display the views the user wants when clicking in the different elements of the navigation bar at the top of the web page.

Vuex [10] allows us to have a centralized store for all the components in an application. This is very helpful when we have components that are not directly connected.

To do the requests to the server, we will use Axios [11] which is a HTTP Client based in promises.

7.3.2 Frontend Folder Structure

Fig. 10 shows the structure of the frontend directories so that you can get a better idea of how the different components are structured. Not all the files of the application are showed to not to make the figure too large.

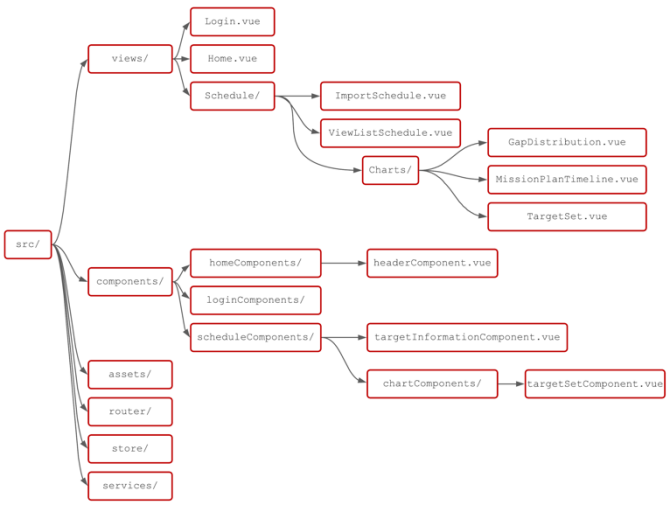

Fig. 10: Frontend Folder Structure

There are the following directories:

- Assets: in this folder is where all the images and icons are
- Components: in this folder is where we have the different components used in the views
- Router: in this folder is where the Vue Router is
- Services: in this folder is where we have all the HTTP requests to send to the server using AXIOS
- Store: in this folder is where the store of the Web App is using Vuex. Components are going to use this to interchange information between them
- Views: in this folder is where the views (main pages) are
- App.vue: is the main component of the Web App, the root component. It will be composed by the header and footer and will use the Router (VueRouter) to display different views as shown in Fig. 6.
- main.js: this is the first file that the client is going to run, this will create the Main Vue App

7.4 Database

The database used in this project is PostgreSQL. One of the main reasons why we have chosen this database is because it is open source and offers a very easy and intuitive user interface. Another advantage is that AWS (Amazon Web Services) has easy integration with PostgreSQL and it offers an easy set-up, operation, and scale PostgreSQL deployments in the cloud. So, our PostgreSQL database is hosted in AWS. There are other open-source relational SQL databases in the market such as MySQL, a popular database. If we do a comparison of performance between MySQL and PostgreSQL, PostgreSQL is better in terms of performance than MySQL as we can see in the study [12].

8 Testing

This section covers how the testing process has been implemented. The main task is to ensure that the application developed acts as the user expects it to do, investigating the application functionalities to ensure the quality of the software product under the test.

To do this, we have put the web application under an internal domain on an ICE server that only people from within ICE can access and test the application.

The idea of doing this testing is that several users (ICE physicists who are involved in several projects of space and ground-based astronomical observatories) use the application to compute schedules for the telescopes of their projects. The first tests will be done by a physicist from the Montsec Observatory [13] and another physicist from the ARIEL mission [14]. Once these users have tested the application in a period of 1-3 weeks, they will be sent a form with several questions to obtain feedback of their use.

Currently the application is under testing, and it is expected to receive feedback from the users to know possible bugs or improvements that can be implemented.

9 Results

The result of this work has been the creation of a functional web application to analyze and compute efficient observational plans for telescopes in space and ground-based astronomical observatories thanks to the representation of statistical data of the schedule through charts. Fig. 11, Fig. 12 and Fig. 13 show some of the possible visualizations that the user can make on an imported schedule.

In Fig. 11 we can see the target/planet distribution in the sky. This is a scatter chart that shows where the planets that have been planned and those that have not been planned are located. We can appreciate that if the user

clicks on one target in the chart, on the right side of the screen appears information about this target, its visible periods, and more information of interest for the scientist.

In Fig. 12 we can see the mission plan timeline for a specific range of dates set by the user. This is a timeline chart where the user can see the different tasks that the telescope performs over time (Observation tasks in blue, Calibration tasks in red, Gaps that the telescope doesn't perform any operations in yellow). We can see that the user can modify these dates above the chart. If the user clicks in any task in the timeline, on the right side of the screen appears again the same component as the one in Fig. 11 which shows information about the target who has that task.

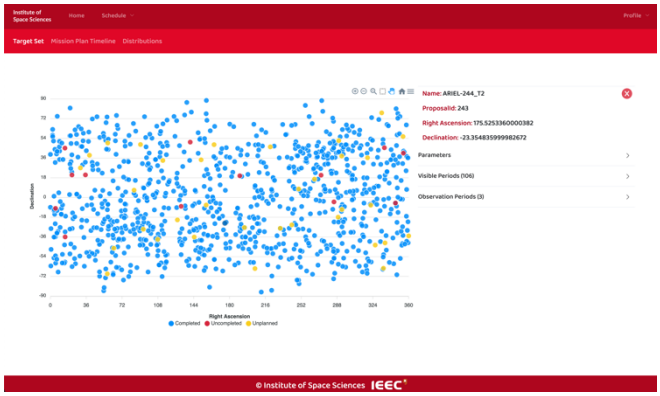

Fig. 11: Target Set View

Fig. 12: Mission Plan Timeline

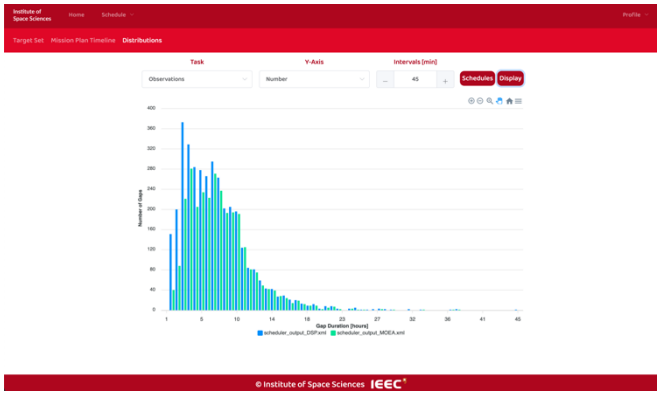

Fig. 13: Gap Distribution by Number of Gaps

In Fig. 13 we can see the distribution of any of the tasks performed by the telescope. This is a histogram where the user can see how many tasks there are based on how long each task lasts. Above the chart, the user can configure what type of task distribution wants to see and what type of data wants to represent on the Y-Axis, in this case the number of observations. There is also a dropdown menu in the button "Charts" that lets the user to add more schedules to compare, in this case we can see two schedules.

Therefore, the main objective of the project that was to design and develop a visualization planning tool to analyze and compute efficient observational plans that has been satisfactorily fulfilled.

Within the subobjectives of the project defined in chapter 3, the analysis of the viability of the project as well as the investigation of the outdated visualization tool have been satisfactorily fulfilled. To do this, as I have said before, several meetings have been held with the stakeholders where these issues have been discussed and where the collection of requirements has also been carried out.

The sub-objective of implementing the backend has been fulfilled by 80%, because it was not possible to implement the functionality of create an optimized schedule through the web application using an executable of the scheduler developed by ICE. The main reason is because as I said in chapter 1, this scheduler is an application developed in C++ and is not yet prepared to be integrated with this project. This functionality remains as future work. However, another requirement was to receive requests from the client and process the schedule fast, and this part was satisfactorily fulfilled. We can take as a key performance indicator (KPI) that the time that takes to compute the schedule file to represent the mission plan timeline (Fig. 12), which is the chart that takes most time to render, is about 5 seconds which is a great advance with respect to the previous application that was mentioned in chapter 2 that took more time to render some charts.

The sub-objective of creating a frontend that can analyze and show the results provided by the backend has been satisfactorily fulfilled as we can see in Fig. 11, Fig. 12 and Fig. 13.

10 Conclusions

To conclude, it should be noted that many of the functionalities that have been developed in this project have taken much longer than initially planned. The reason is that although I already had some knowledge of most of the technologies used, I had to see more detailed documentation to be able to implement some functionalities that required more technical knowledge.

How to assemble the infrastructure and the organization of all the files has been one of the things that has taken the most time since I had never done before a web application that required a lot of code.

The part of the frontend has required most of the time and has been the most difficult. The reason is because the frontend is the part that incorporates the largest number of frameworks and is also the visual part of the application, and always spends a lot of time trying to make it look good. It also requires a good relationship between components which also makes its development much more difficult.

In the end, STARS GUI has satisfied the main objectives that were set at the beginning of the project. It is a simple, easy, very intuitive, and fast application that facilitates the understanding of the results provided by the scheduler. In addition, one of the advantages it has over the outdate application is that it lets scientists to configure the type of chart they want.

11 Future Work

STARS GUI is an application that has a lot of future and that there are still many functionalities to be developed that will be very profitable for ICE projects.

One of the main points of great interest is to make the application adaptable to many more ICE projects, since right now it only works for 2 specific projects.

Another important point to improve would be that the user could create an optimized schedule within the application itself, this means that the application should use the schedulers developed by ICE which is a C++ application to create optimized schedules and then provide the results. As we have already seen, currently, what the user does is import the already optimized schedule to the application.

Acknowledgements

I would like to express my sincere gratitude to several people. Firstly, to my family, who are always supporting me pushing me to become a better version of myself and who always give me the best advice. Secondly, my sentimental partner, for the love and support that gives me every day. Also, to the people of ICE (Juan Carlos, Francesc D., Francesc V. and Emilio) to whom I thank for giving me this project and for always being available for any doubt. Finally, to Prof. Víctor García for his advice throughout the project work.

Bibliography

[1] A. Garcia, J. Colomé. Automated planning and scheduling [Online].Available: https://www.ice.csic.es/research/experimental-research/2-uncategorised/48-automated-planning-and-scheduling
[2] Maria D. (Aug 25, 2020). Popular Software Development Methodologies and Frameworks: Full Comparison [Online]. Available:https://www.cleveroad.com/blog/software-development-methodologies
[3] Oleg K. (Nov 06, 2020). How to Benefit From Using Node.js for Your Next Project? [Online]. Available: https://procoders.tech/blog/advantages-of-using-node-js/
[4] NodeJS [Online]. Available: https://nodejs.org/es/
[5] VueJS [Online]. Available: https://v3.vuejs.org/
[6] BootStrapVue [Online]. Available: https://bootstrap-vue.org/
[7] ApexChartsJS [Online]. Available: https://apexcharts.com/
[8] ElementPlus [Online]. Available: https://element-plus.org/en-US/
[9] VueRouter [Online]. Available: https://router.vuejs.org/
[10] Vuex [Online]. Available: https://vuex.vuejs.org/
[11] Axios [Online]. Available: https://axios-http.com/docs/intro
[12] Blessing K. (Feb, 2021). Performance differences between Postgres and MySQL [Online]. Available: https://arctype.com/blog/performance-difference-between-postgresql-and-mysql/
[13] Montsec Observatory [Online]. Available: http://www.ieec.cat/content/18/oadm-montsec-observatory/
[14] ARIEL Mission [Online]. Available: https://arielmission.space

APPENDIX

A1. MODEL VIEW CONTROLLER (MVC) ARCHITECTURE

MVC is a design or architectural pattern used in software engineering. MVC separates the application into three main logical components as we can see in Fig. 14:

- Model: represents the structure of data, the format and the constraints with which it is stored. It determines how a database is structured, defining a section of the application that interacts with the database. This is where we will define the properties of a user that will be store in our database. The controller accesses the database through the model.
- View: it is where end users interact within the application.
- Controller: it controls the requests of the user and then generates appropriate response which is fed to the viewer. Typically, the user interacts with the View, which in turn generates the appropriate request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

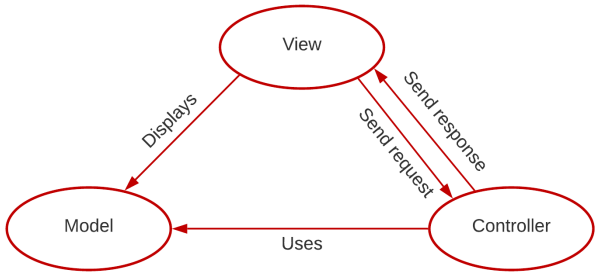

Fig. 14: MVC Architecture

A2. EXPRESS ROUTING

Express is a web framework that lets structure a web application to handle multiple different http requests at a specific url. It is used mainly for routing and for the main configuration of the application (in `app.js` file).

To do the Routing process, we use the Router Middleware provided by Express by creating a Router instance. A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a "mini-app".

Below we can see an example of how all this routing progress work in a specific case that was explained in section 7.2.3.

The case explained in section 7.2.3 is the case when a user clicks in the button of "Show schedules" and the user expects to see all the schedules that he has simulated. To do this, when the user clicks the button, the frontend sends a request with the path `/schedule/getschedule` that the `app.js` will manage.

The `app.js` when it gets this request, it uses the router provided by `scheduleRoutes.js` in line 2. This router will get all the request with the path `/schedule` at the beginning. Then this router will use the different modules in the `scheduleController` to do different actions. For example, in this case, it is going to use the module `get_schedules` that it does a qury to the database to get all the schedules of the user.

app.js

```
1 const express = require('express');
2 const scheduleRoutes=require('./routes/scheduleRoutes.js');
3 app.use('/schedule',scheduleRoutes);
```

scheduleRoutes.js

```
1 const express = require('express');
2 const router = express.Router();
3 const scheduleControl-
ler=require('../controllers/scheduleController.js');
4 router
.post('/getschedules',scheduleController.get_schedules);
5 router
.get('/deleteschedule',scheduleController.delete_schedule);
6 module.exports = router;
```

scheduleController.js

```
const get_schedules=(req,res)=>{

    db.getSchedules(req.body.username)
    .then((result)=>{
        res.status(200).json(result.rows).end();
    })
    .catch((err)=>{
        res.status(400).send({message: 'There was an er-
ror'}).end();
    })
}
 . . .
module.exports={
    get_schedules
}
```