Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Generador estático web para facilitar tareas en la docencia

Static web generator to ease teaching tasks

Carlos Díaz Calzadilla

La Laguna, June 29, 2023

D. **Casiano Rodríguez León**, con N.I.F. 42020072S profesor Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

I N F O R M A

Que la presente memoria titulada:

"*Generador estático web para facilitar tareas en la docencia*"

ha sido realizada bajo su dirección por D. **Carlos Díaz Calzadilla**, con N.I.F. 43380941C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a June 29, 2023

Agradecimientos

Quisiera agradecer en primer lugar, a mi tutor Casiano Rodríguez León por su continuo acompañamiento y reccurrente, preocupación y grado de compromiso tanto en el plano personal como académico. Asimismo quisiera también reconocer la labor y agradecer a todo el profesorado de la Escuela Técnica Superior de Ingeniería y Tecnología por los contenidos, herramientas y ayudas que me han dado a lo largo de mi paso por la universidad. Por ultimo, agradecder también a mi familia y amigos que me han estado apoyando siempre y han hecho que gracias a ellos esto fuera posible.

Licencia

Abstract

En la educación, el uso de las tecnologías hoy en día es un factor fundamental, por ello, el profesorado se ha tenido que ir adaptando a estos nuevos sistemas. El tener acceso a Internet y a una cantidad casi indefinida de herramientas ayuda a que las labores de dociencia puedan ser más ágiles, pero a la vez esto implica que la información no esté centralizada, haciendo que esté dispersa por estas herramientas, como usando GitHub para repositorios, el campus virtual para entregar las tareas y evaluar al alumnado. El objetivo de este trabajo es realizar un generador estático que ofrezca una página web que permita tener todo este contenido centralizado, haciendo que el docente tenga toda la información a mano y le sea más fácil el apoyo en las herramientas externas, ofreciendo una plantilla para modificar en caso necesario.

Palabras clave: educación, generador estático, Vue, Vite, GitHub, docencia, plantilla

Abstract

In education, the use of technology today is a main factor, therefore, teachers have had to adapt to these new systems. Having access to the Internet and an almost indefinite number of tools helps teaching tasks to be more agile, but at the same time this implies that the information isn't centralized, being scattered throughout these tools, such as using GitHub for repositories, the virtual campus to deliver homework and assess students. The objective of this work is to create a static generator that offers a web page that allows all this content to be centralized, making the teacher have all the information at hand and making it easier for them to use external tools, offering a template to modify and ,if necessary.

Keywords: education, static generator, Vue, Vite, GitHub, teaching, template

Contents

1 **Introduction** 1
1.1 Context . . . 1
1.2 Background and current state of the art . . . 1
1.2.1 GitHub Education Methodology . . . 2
1.3 Objectives . . . 3
1.4 Development methodology . . . 3

2 **Technologies used** 4
2.1 Scripts . . . 4
2.2 Template . . . 4
2.3 Client framework: Vue . . . 5
2.3.1 Single-File Components . . . 5
2.3.2 API Styles . . . 6
2.3.3 State management . . . 6
2.4 Static Site Generator . . . 7
2.4.1 VuePress . . . 8
2.4.2 VitePress . . . 8
2.4.3 Comparison, why VitePress? . . . 9
2.5 JamStack . . . 9
2.6 Firebase . . . 10

3 **System design** 11
3.1 System Architecture . . . 11
3.2 Design and architecture of the application . . . 12
3.2.1 Project Structure . . . 18
3.2.2 User Interface . . . 21

4 **Implementation** 22
4.1 Script implementation . . . 22
4.1.1 Manual . . . 22
4.1.2 Script menu . . . 22
4.1.3 Main Script . . . 25
4.2 Template implementation . . . 29
4.2.1 Store . . . 29
4.2.2 Components . . . 31
4.3 Final system . . . 39

5 **Quick guide** 45
5.1 Download GitHub template . . . 45

5.2 Firebase project 46
5.3 GitHub authentication 47
5.4 Create environment 49
5.5 Deployment 50
5.6 Set up with the gh-education script 52

6 Summary and conclusions **56**

7 Resumen y conclusiones **57**

A Project repositories **58**

List of Figures

2.1 Simple Single-File Components example 5
2.2 Vuex state cycle 7
2.3 Firebase authentication suppliers 10

3.1 Basic layers of application 12
3.2 Main page initial design 13
3.3 Teams initial design 14
3.4 Student initial design 15
3.5 General initial design 15
3.6 General login view design 16
3.7 Login view design 16
3.8 Register view design 17
3.9 External dependencies 17
3.10 Script project structure 18
3.11 Template project structure 20
3.12 Palette colour for dark mode 21
3.13 Palette colour for ligh mode 21

4.1 Template Vitepress files 29
4.2 UML template diagram 31
4.3 Vue Lifecycle 32
4.4 SDK Firebase configuration 33
4.5 Navigation bar 39
4.6 Home Page 40
4.7 Schedule 41
4.8 Lessons - Student view 41
4.9 Tasks - Student view 42
4.10 Units - Student view 42
4.11 Teams of an organization 43
4.12 Specific team information 43
4.13 Login view 44
4.14 User logged view 44

5.1 Template repository use 45
5.2 Template repository created 46
5.3 Firebase project creation 46
5.4 Firebase project home page 46
5.5 Firebase create app 47
5.6 Firebase app SDK 47
5.7 Firebase Authentication 48

5.8 Firebase Authentication GitHub provider 48
5.9 Firebase GitHub App 49
5.10 Firebase GitHub App 49
5.11 Firebase GitHub App Tokens 50
5.12 Authorising GitHub App 51
5.13 Allow GitHub pages domain in Firebase 51
5.14 Script Firebase Project Creation 54
5.15 Script Firebase List Projects 55

Listings

4.1 Get cursor position function 23
4.2 Change cursor position function 23
4.3 Get input key function 23
4.4 Main choose menu function 24
4.5 Clone repository function 26
4.6 Firebase project menu 26
4.7 Firebase Project SDK 27
4.8 Refresh firebase configuration file 27
4.9 Deploy Script 28
4.10 Store mutations 29
4.11 Login action 30
4.12 Logout action 31
4.13 Authentication component data 33
4.14 Template Auth component 33
4.15 Login and logout methods 34
4.16 Units mardown example 34
4.17 Units component prop and dynamic import 35
4.18 Ghcard component template 35
4.19 Team component template 36
4.20 Teams component data 37
4.21 Team component template 37
4.22 Teams component function 38

Chapter 1

Introduction

The aim of this chapter is to explain about the problem this system wants to solve, a small introduction of the system and the work methodology used for the development of the project.

1.1 Context

Nowadays, with all the new devices, the Internet with all of this powerful cloud services, having everything on hand is easier. Teachers need to access a lot of different files or web pages in order to get student information, like the virtual campus to add the tasks, evaluate the students, to add all the units and files, having student GitHub repositories, their commits or teams of an organization. In order to solve this problem, this system has been developed.

1.2 Background and current state of the art

Teaching is important to maintain knowledge over the years and to understand the world around us to know how to solve problems,... Nowadays, all the information is online getting anything easy and fast. Like the information changes from paper to online, teaching has also had to adapt to this new situation. New environments for the teachers, new ways to teach using tools from Internet, etc. That's why, in 1990s was the first introduction to Learning Management Systems (13). LMS is a software application for the administration, documentation, tracking, reporting, automation, and delivery of educational courses,... that was designed to identify training and learning gaps, using analytical data and reporting focused on online learning delivery but supporting a range of uses, acting as a platform for online content, including courses.

If this situation is moved in how to teach computing science, there is a tool that can be one of the most used now is Git (GitHub and GitHub Classroom). The way to use or whether or not using Version Control System in teaching can be difficult to do. First, Git is a Version Control System that provides support for tracking changes and storing files. This information, (files, changes, history,...) is grouped in repositories. One of the most important differences from Git to the others VCS is that is distributed (everything is mirrored in all the developer's computer).

On the other hand, the most popular development platform for Git Version Control is GitHub. It provides a web graphical interface that help the users to manage, check, copy, download,... their or others repositories. Also GH is a social platform for developers to share their projects, help the community with some bugs or fix errors.

Finally, there is an GitHub API called GitHub Classroom that was created to ease and improve the use of Git and GitHub for education. GitHub is a good system to have your own code, or history and information about it. But it lacks the concept of the assignments and classrooms in the education environment. It's true that in GH it is allowed to create teams and organizations. The teams are usually used to have small groups and to develop one task. Otherwise, the organizations are more like a classroom for code. This is why GHC is important, because it changes the concept of an organization to a classroom. As we said earlier, it adds the assignments that need to be associated with a a visible repository (template repository). Automatically GHC creates an invitation link that when the student accepts it, a repository (like the template of the assignment) will be cloned.

1.2.1 GitHub Education Methodology

Once we know the basics we can talk about how GitHub Education Methodology works. But first it is important to remember how GitHub classroom works, using as reference the GitHub documentation (5). GitHub Classroom is a teaching tool that lets teachers and school administrators create and manage digital classrooms and assignments. It allows teachers create assignments for individual students or groups of students, set due dates, and track assignments on your teacher's dashboard. Additionally, GitHub Classroom has many features that simplifies tasks like providing feedback, grading assignments, and integrating your existing teaching tools.

In order to answer this question it is important to make researches about the topic. After a few researches, most of articles tell us about using GitHub as a collaborative platform for education, explaining in each of them the methodology used to achieve that objective. But let's talk about two of them that are interesting. First, there is an article (7) extracted from a course (8) that explains the use of GitHub in the education environment. It's interesting because there are a lot of links to registration in the platform, creating profiles, the use of git,... So it's dedicated to people that knows git or not, for all the public.

On the other side, the article (14) mentions the use of GitHub in education. In this article, the first part is an introduction to CSCW(Computer Supported Cooperative Work) and education with a simple explanation about LMS (Learning Management Systems) and GitHub. The next part is the most related with this section. They work using exploratory research methods and they find the use of this tool in the education environment, the different ways to use it in this environment, benefits and motivations. This article also shows the different ways of using GitHub for education and there are two:

- Submission platform: This can be achieved in two ways. On the one hand, using a public repository as base, and then the students can fork it (seeing all the other forked repositories as well). This allows students to cross-reference different solutions and encourages student collaboration. On the other hand, it sets up a private repository for each student and individual students can only see other repositories when explicitly given access (needing to set up repositories and permissions manually). This has some benefits as having transparency of activity, allowing teachers to monitor student's progress, activity and participation.

- Hosting Course Material: Having the class hosted allows the teacher to upload content as notes, reading materials, exercises and homework. Using it this way, it allows the students and other teachers to create issues or pull requests in order to suggest course improvements.

1.3 Objectives

The objective of this work is to design and develop a functional system that allows the teachers to have an easy way to monitor the students having all in the same site. This system is going to be divided into to subsystems. The first is the template that the user can download and change it to his / her liking. The other one is a bash script used for downloading the template, changing the most important things of it, deploying the final result,... In general, the system aims to achieve this following specific objectives:

- Authorisation for teacher and students allowing you to restrict certain content.
- It's connected with GitHub, having information about students GitHub's profiles.
- Easy deployment and easy to use, giving the user a quick guide.
- The script allows user to change GitHub organization or refresh it to see teams.

The results of this project aim to be a useful and functional system focused on teachers. It also can be, an starter version for an idea that can be more and more complex. Also, it needs to have main things developed like authentication, easy development, optimised and easy for modification having a great understandable structure of files.

1.4 Development methodology

This project has been developed in different phases. In the first phase an exhaustive search was made on the different technologies available to be able to implement the system, comparing them and seeing which can fit better the system to solve the problems.

In the phase of development of this system Agile methodologies were used, having meetings one each week. In this meetings we usually try to answer this questions:

- What have we got up to now?
- Is there any new feature for the system?
- Are there any new errors?

Also, once a month or in the last meetings, we tried to install and run the system in a new environment to see new errors or the different ways of the system. For example, in OS, the output shows some characters that are not shown in Linux.

The git version control tool has been used with GitHub for some coordination tasks as well as using it for creating a project board. That allows teachers have all the tasks in the same place and an easier way to control what features and errors we are fixing or adding.

Chapter 2

Technologies used

Nowadays there are a lot of different technologies to use for a project. Often, choosing one technology instead of others can be a hard decision for the developer and he / she usually takes the most familiar one, but advantages and disadvantages need to be known, future projection or life of the technology, that is to say, if the technology is new or it is old. The aim of this project is to create a template for teachers to get all the information of the education in the same site easily. So, this chapter describes the technologies used to carry out this project and the justification for the choice made.

2.1 Scripts

On the other hand, the script that the users are going to use to create the template and the initial configuration for it. For this one, it was developed a bash script because this one will be more flexible if we want to modify the directory or run bash commands.

It is important to mention that the script is divided into 4 different scripts. This is because in order to get a script that it is easy to understand to modify and to work with it is better to have it divided. All of them are developed using Bash because it is one of the languages that I have used the most. Also, bash allows to use all the bash commands and make the integration of some tools like Firebase easy. Another important thing is it is a powerful programming language that usually is used to make automated scripts, making programs with interactive menus to ease user experience,...

2.2 Template

On the one hand, as it was said before, the aim of this project is to create a static generator for a template. So in this section we are going to talk about the technologies used in the template side. Knowing this, we need to decide what programming language will be used for creating this system. As we want this as a cloud service because it will be easy to access and to use, one good way is to develop the system using JavaScript + Node. Another important reason to take this language instead of others is because we will work using a Vue Static Generator it so is important to develop the system using JavaScript and Markdown.

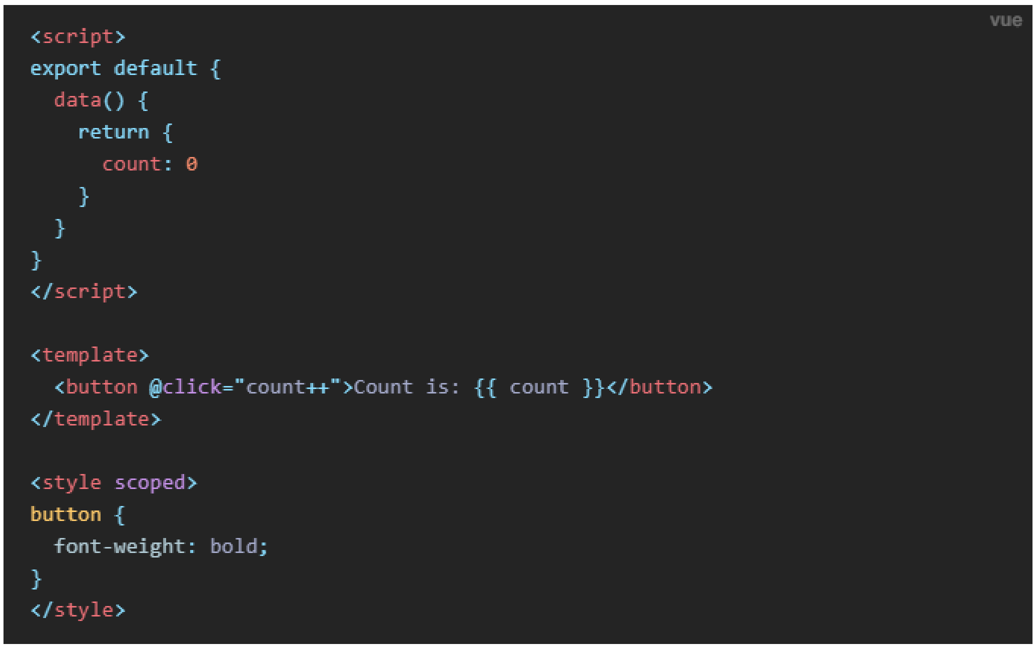
2.3 Client framework: Vue

As it has been said, for developing the script a Vue Static Generator will be used. In order to understand how this works we need a quick explanation about Vue and some features. Vue (10) is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces.

This framework covers most of the common features needed in frontend development, so Vue is designed to be flexible and adoptable. Despite the flexibility, the core knowledge about how Vue works is shared across all these use cases. Vue has some elements that make the difference with other frameworks like Single-File Components, API styles,...

2.3.1 Single-File Components

When we are working with Vue projects there are some files called Single-File Components (also known as *.vue files). These files encapsulates the component's logic (JavaScript), template (HTML), and styles (CSS) in a single file. The way developers use this type of files is to create a separate item that can be used in different files or in other Components. The syntax of these files is always the same, like in this basic example of Figure 2.1.



```vue
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Figure 2.1: Simple Single-File Components example

As shown in the picture, the components has a few language blocks:

1. <template>: Each *.vue file can contain at most one top-level <template> block, where its content will be extracted and passed on to *@vue/compiler-dom*, pre-compiled into JavaScript render functions, and attached to the exported component as its render option.

2. <script>: Each *.vue file can contain at most one <script> block (excluding <script setup>), that is executed as an ES Module and the default export should be a Vue component options object, either as a plain object or as the return value of defineComponent.

3. <script setup>: Each *.vue file can contain at most one <script setup> block (excluding normal <script>). The script is pre-processed and used as the component's setup() function, which means it will be executed for each instance of the component. Top-level bindings in <script setup> are automatically exposed to the template. For more details, see dedicated documentation on <script setup>.

4. <style>: A single *.vue file can contain multiple <style> tags. A <style> tag can have scoped or module attributes (see SFC Style Features for more details) to help encapsulate the styles to the current component. Multiple <style> tags with different encapsulation modes can be mixed in the same component.

5. <custom block>: Additional custom blocks can be included in a *.vue file for any project-specific needs, for example a <docs> block. These blocks depend on the block integration that is used in the project.

2.3.2 API Styles

As it was explained in section of Single-File Components, there is a component logic, this one can be written in two different styles: Options API and Composition API. The one that is used in this project is Options API. Here the definition of component's logic is using an object for options such as data, methods, mounted and watch. Properties defined by options are exposed on this inside functions. Let's explain all of them:

- data: This option is used to declare reactive state of a component. The option value should be a function that returns an object. Vue will call the function when creating a new component instance, and wrap the returned object in its reactivity system.

- methods: In this option is where all the methods component instance are added . Vue automatically binds the *this* value for methods so that it always refers to the component instance. This ensures that a method retains the correct *this* value if it's used as an event listener or callback. That's why it is important to avoid using arrow functions when defining methods, as that prevents Vue from binding the appropriate *this* value.

- computed: Computed properties ease to create a property that can be used to modify, manipulate, and display data within your components in a readable and efficient manner. They are used when you want to display basic operations using some values in the data model.

- watch: Computed properties allow to declare compute derived values. However, there are cases where we need to perform "side effects" in reaction to state changes - for example, mutating the DOM, or changing another piece of state based on the result of an async operation. That's why the watchers are used for.

2.3.3 State management

The state of the system is going to be always changing so it's important to have a state management. As the project is developed using Vue + Vite, there is a npm package called Vuex (12) that aims to handle the state of the system. It's important to have this because the communication between components in Vue is a bit complex and it's necessary to have a parent relation in order to pass the data as properties. But, it becomes easier to store the data with this tool. As it's shown in the Figure 2.2, there is a flow that is in constant change.

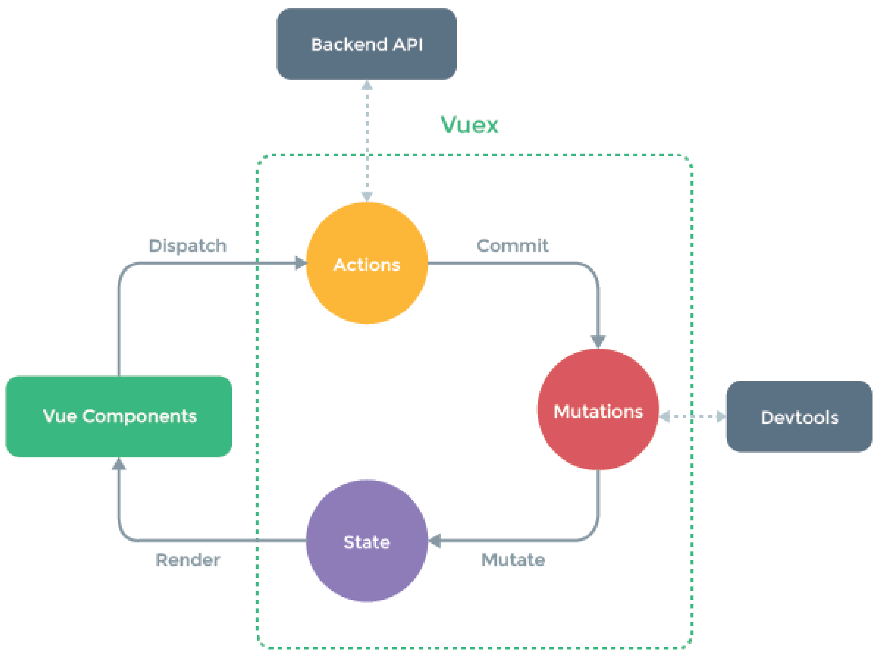


Figure 2.2: Vuex state cycle

First of all, the state, that is, a single object that contains all your application level state and serves as the "single source of truth." This also means you will usually have only one store for each application. Secondly, the mutations are the only way to actually change state in a Vuex store is by committing a mutation. Vuex mutations are very similar to events: each mutation has a string type and a handler.The handler function is where we perform actual state modifications, and it will receive the state as the first argument. Finally, the actions are similar to mutations but the differences being that:

- Instead of mutating the state, actions commit mutations.
- Actions can contain arbitrary asynchronous operations.

2.4 Static Site Generator

After knowing all the different technologies used for the template or the script it is important to tell how it is going to generate that system and the technologies used for it. First of all, Static website Generators are ready-made code frameworks on the basis of which static web pages are developed. Unlike content management systems that retrieve content from databases, static site generators create the HTML files of a page using a script from input files stored in a file system at the time of creation. Having a static website has a few advantages like:

- High speed: The projects that are created using a static website generator are fast to navigate for users. As we said before, processing of files occurs at a time when pages are being created and not only at the time of querying the page.
- Content version control: Structurally, the content elements are no different from other components of the code base, which is why version management can be set up without any problems.
- Security: Another advantage is that they offer small potential attack. This risk potential is low because of the single access by the user.
- Maintenance: The components required for these sites are low, but they have to be available and working only during the development process so it will be easy to maintain.

The SSG has many advantages but it also has some weakness:

- Real-time content: Having an static generator, doesn't allow users to get content dynamically or in real time
- User interactivity: All the content is static so the user retention tends to be low and limited.

There are lots of different static generators like next, Gatsby, Jekyll, Hugo,... In my case, as I have been working for a long time using Vue, there are two of them that are the most important: Vuepress and Vitepress.

2.4.1 VuePress

VuePress (11) is a markdown-centered static site generator. It allows to write the content (docs, blogs, etc) in Markdown, then VuePress will generate a static site to host them. VuePress is a sing-page application (SPA) and it is powered by Vue 2 and Vue Router.

Routes are generated according to the relative path of your markdown files. Each Markdown file is compiled into HTML with markdown-it and then processed as the template of a Vue component. This allows to use directly Vue inside your Markdown files and it is great when you need to embed dynamic content.

Also, during development, we start a normal dev-server, and serve the VuePress site as a normal SPA. On the other hand, during build, we create a server-rendered version of the VuePress site and render the corresponding HTML by virtually visiting each route. This approach is inspired by Nuxt's nuxt generate command and other projects like Gatsby.

2.4.2 VitePress

VitePress (9) is VuePress little brother and it is built on top of Vite. Vite is a build tool that aims to provide a faster and leaner development experience for modern web projects. It's true that, VitePress is currently in alpha status so it's new and doesn't have as many plugins, updates,... as VuePress. Also, VitePress uses Vue 3, instead of Vue 2.

Incidentally Vite solves these problems really well: server starts nearly instant, an on-demand compilation that only compiles the page being served, and lightning-fast HMR. Plus, there are a few additional design issues in VuePress that VitePress aim to fix.

2.4.3 Comparison, why VitePress?

Now that the two static generators are known, it's time to decide which of them is the chosen one for the project. Here are some of the most remarkable advantages of Vitepress:

- Home Layout: The home page is something important to have the client's attention. In this point VuePress has a vertical layout, column style feature but Vite one is horizontal and the feature comes in card style.
- Sidebar: In this case, the sidebar of VitePress is better designed, easy to modify and has a good presentation.

On the other hand, VitePress and VuePress also have some disadvantages like:

- External tools: VitePress has less components and plugins than VuePress. But it is true that the basic ones are available.
- Age: Vitepress is still young at its 1.0.0-alphaxx and vuepress already v2.x and more mature.
- Documentation: One important thing is that the documentation is a bit behind its development.
- VuePress Sidebar: It's true that the design of the sidebar of VitePress is better. In the case of VuePress the configuration is a bit complex, it's not possible to change it in run time, only in build time

2.5 JamStack

To understand better the reason of the technologies used and the way in which they will influence the architecture of the system it's important to explain JamStack (6). This is a web development architecture that is based on the delivery of static content through a CDN (Content Delivery Network). This architecture is characterised by separating application logic and content management from visual presentation, allowing greater scalability, security and loading speed. The name of JamStack comes from:

- JavaScript: It is responsible for the logic of the application and the interaction with the user. This logic is executed on the client side by using JavaScript frameworks and libraries such as React, Angular or Vue.js.
- APIs: They are responsible for the management of the data and business logic of the application. In a Jamstack application, using an API allows to invoke these functionalities from the JavaScript layer through HTTP requests.
- Markup: It is responsible for the visual presentation of the application. This layer is implemented using HTML, CSS, and JavaScript files that are delivered over a CDN and rendered directly in the user's browser.

Using JamStack, the website code runs all on client scripts meaning that has more advantages over other traditional development approaches such as being its much faster, safer, having lower risk and it makes easier to scale the web.

2.6 Firebase

In order to have JamStack approach Firebase (2) will be used as an external API and it is an app development platform that helps to build, grow apps and game users loves it and it's backed by Google. Firebase has a lot of features like authentication system, database, deployment and analytic for the app created. In this project, Firebase will be used to fulfil two different tasks:

- Authentication: Firebase authentication, allows to have a system that can restrict content to different users or use it for accessing to other features. As shown in the Figure 2.3 Firebase allows to have a lot of different suppliers like Google, GitHub or use a mail with password one.
- Hosting: When the template is correctly modify, the user can deploy the site to the project hosting site.

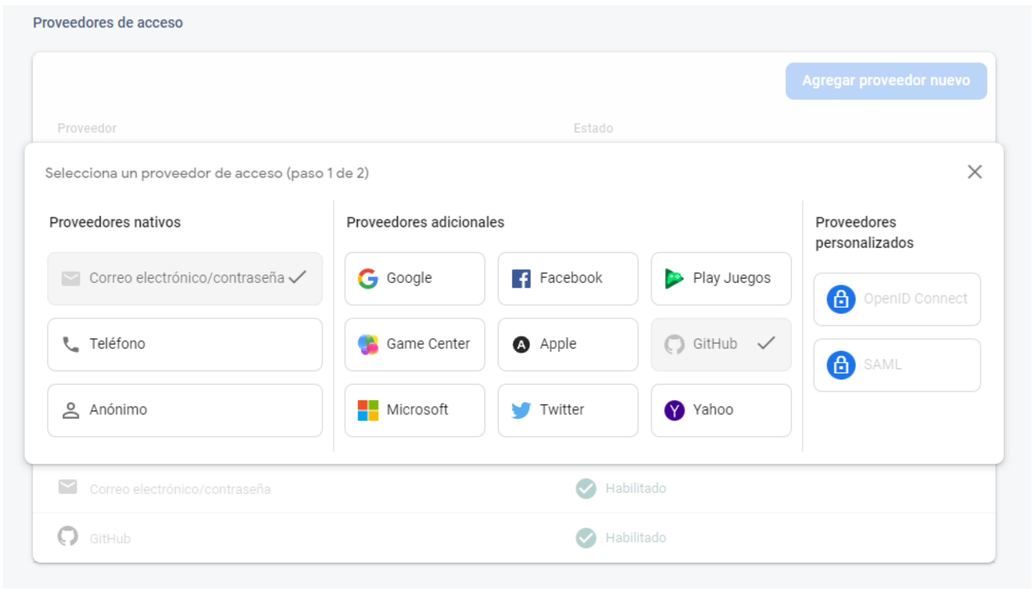


Figure 2.3: Firebase authentication suppliers

Chapter 3

System design

Once we have analysed the available technology, it's time to talk about the system design. These two steps, the choice of technology and design feed each other, and the decision in one of them can affect the other and vice versa. In this chapter the design of the system will be shown, without talking about functionality. This step is important because the design of the system will be affecting directly the way to implement the system.

3.1 System Architecture

First of all, one of the main decisions was what type of architecture to take, standalone or a client-server approach. In this section both of them will be analysed and the reason for the action taken.

Local vs Client-Server

To begin with the project, the first action needed to take is to choose the architecture of the project it was whether it should be a local application or some kind of connection would be needed in network. To make this decision, lets see the advantages and disadvantages of each way:

- A local application is much easier and simpler to develop. The response speed of the device is greater since the data is stored locally and does not depend on the network connection. There is no need to worry about variables that arise when talking about this type of architecture such as synchronisation, optimising waiting times or security.
- A client-server application is more complex, not only because it requires two developments and it is necessary to design an architecture, but also because a series of problems typical of this type of system is introduced, such as errors or security, synchronisation. Many of these problems, rather than a solution, usually require a strategy that minimises their effects. As advantages, this type of architecture offers more dynamism.

Since both options have pros and cons, in this system we want to have dynamic changes and have everything on a web page, so the best is to have a local application but it will be deployed in a host page. So this application will be a local system, with dynamic changes (thanks to Vue developer server mode), an storage for users authentication that will be managed by Firebase. It won't be a client-server because of the complexity for the user to lunch both. For the user, to have to lunch the client and server can be a bit troublesome. So, like there isn't so much information to store, it can be locally and the authentication by an external system.

3.2 Design and architecture of the application

This section describes the design of the application at the software level, the organization of the project and its main components. Following the design and style fundamentals stated at the beginning of this report, a strongly typed and well-structured architecture has been chosen, divided in two subsystem, as shown in Figure 3.1. With this idea, the aim is to have separated the script for cloning template, basic configuration, etc and having the template that can be modify in every moment.

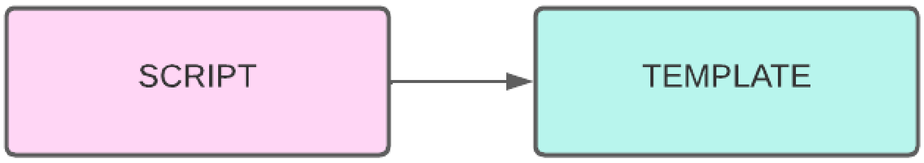


Figure 3.1: Basic layers of application

Script

Now, let's talk about how the script is designed. The aim of this script is to guide the user through all the initial configuration of the template and the deployment. The design of the performance of the script should be following these steps:

1. Clone repository: In this step, the script have to download the template from GitHub and clone it in the current folder.

2. Initial configuration: This is a quick step that aims to get an initial information about the user like subject name, Academic year or the GitHub organization for example.

3. Authentication: To deploy the project Firebase will be used for it. In Chapter 2 we talk about the Authentication with Firebase. In this step, the script installs the dependencies and then the user logs in.

4. Firebase project: In order to link the system with a Firebase project, the script must ask the user to choose one of his / her own projects or to create new one.

5. Link with app: When the project is created, the script needs to change the default project of the system with the project chosen.

6. Build site: When the app id and all the information of the Firebase project is changed it's time to build the VitePress site.

7. Deployment: The last step is to deploy the system in Firebase cloud hosting system.

Template

With all the functionality of the script explained, the next step is to show how the site is going to be shown in the client. So, in order to make a initial design of the system, four representations (that can be changed depending on the technologies used or the new structure in the future) have been done of the designs that would have the main page and some of the others available have been made.

All the start information will be on the **main page** with the title of the system and some of the most important features. As its shown in the Figure 3.2, the design has a navigation bar where it should be all the different tabs of the site allowing the user to change pages easily and fast. Then the app title and quick description of the system, there can be some buttons or links to the code of the template and the logo on the other side. And at the bottom, add a description of all the tabs and the features of the system.

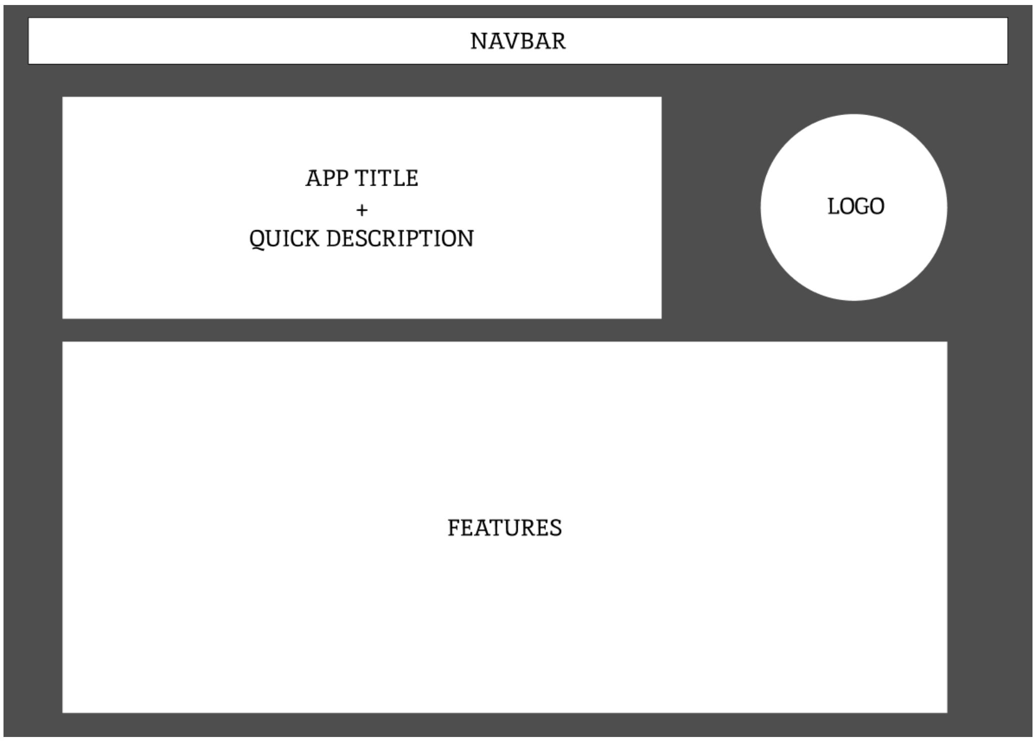


Figure 3.2: Main page initial design

Secondly, the tab related with **teams**, needs to have an special design. In the Figure 3.3 the navigation bar is in the same position, because its aim to be in all the pages to guide the user and make easier the navigation. The system, to create the teams is going to use a GitHub Organization and a dynamic query. So to it's important to add the name of the current organization. And then, create a quick design for a GitHub card (a component that we will going to talk later) having the name of the team, a picture and then some of the initial important information like the GitHub profile, the repositories and the notification of that team.

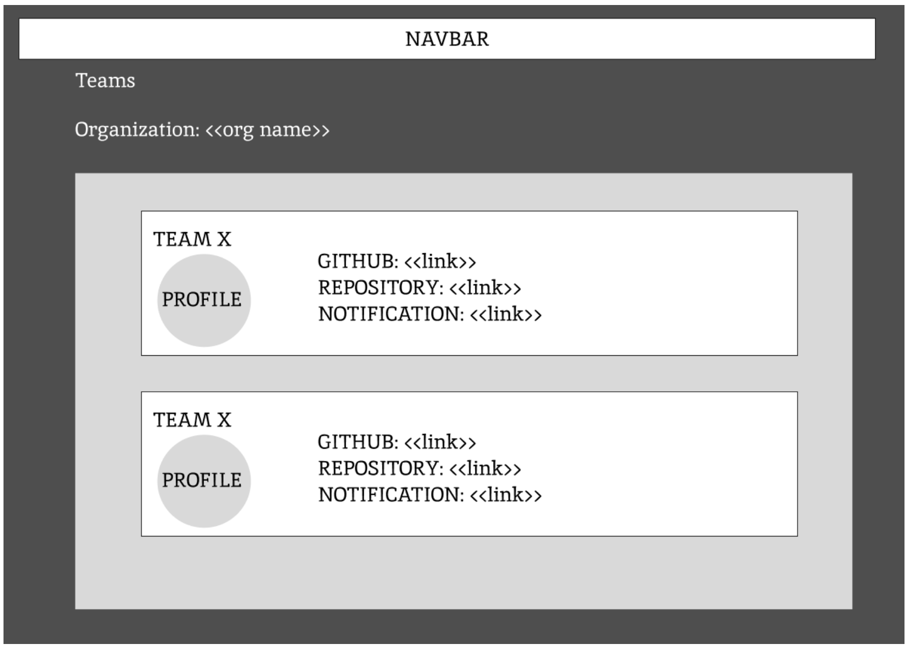


Figure 3.3: Teams initial design

As there is an special design for the teams tab, it should be one for the **student** tab too (that in the final system the name will be Team, because is the specific information about one Team not the student information). In the Figure 3.4, the navigation bar stays in the same position. Then, this tab should be opened when the user presses one of the teams card. So it’s important to know what students we are talking about, then there should be the name of him / her. The initial information is the same as in the teams tab, but there will be more information here. The idea is to have here all of the student recent projects, for example 10 of the recent one. It also can be good to have a link to the commits, issues and summary of each project.

At last but not least, there is a design for all the other tabs that show some links or information like the tasks, units or lessons. As shown in Figure 3.5, this design is an standard and can be modified in order to use in different tabs. For example, this is a good template to show the units, lessons and it can also be used for the tasks.

It’s important to add a quick tutorial to the system to guide the user if he / she has some doubts about how to use it. In it it can be shown the features like authentication, tabs available and the functionality of them can be shown. It is also important to talk about the configuration of the site, where are all the published units, lessons and tasks, and the files to change, etc are placed.

One of the features of the system was the login. The first way to add this is to create another element into the navigation bar to get to the login page. Then this page, can be in three different views. The Figure 3.6 shows how the initial login or register view can be. In this view there are two buttons, one for changing the login view and the other for the register view.

If the user presses the login view, as we can see in the Figure 3.7, the login form will be opened. This is an standard form where the user can add the email and password registered. Also there is an option at the bottom of the form that will allow the user to change to the register view.

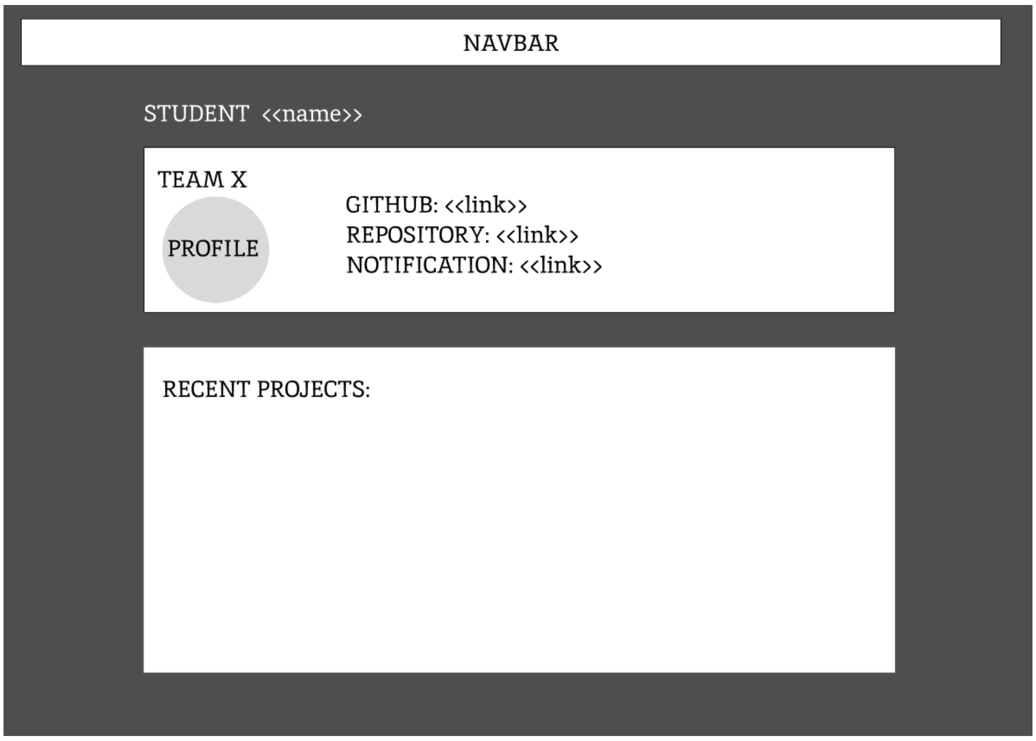


Figure 3.4: Student initial design

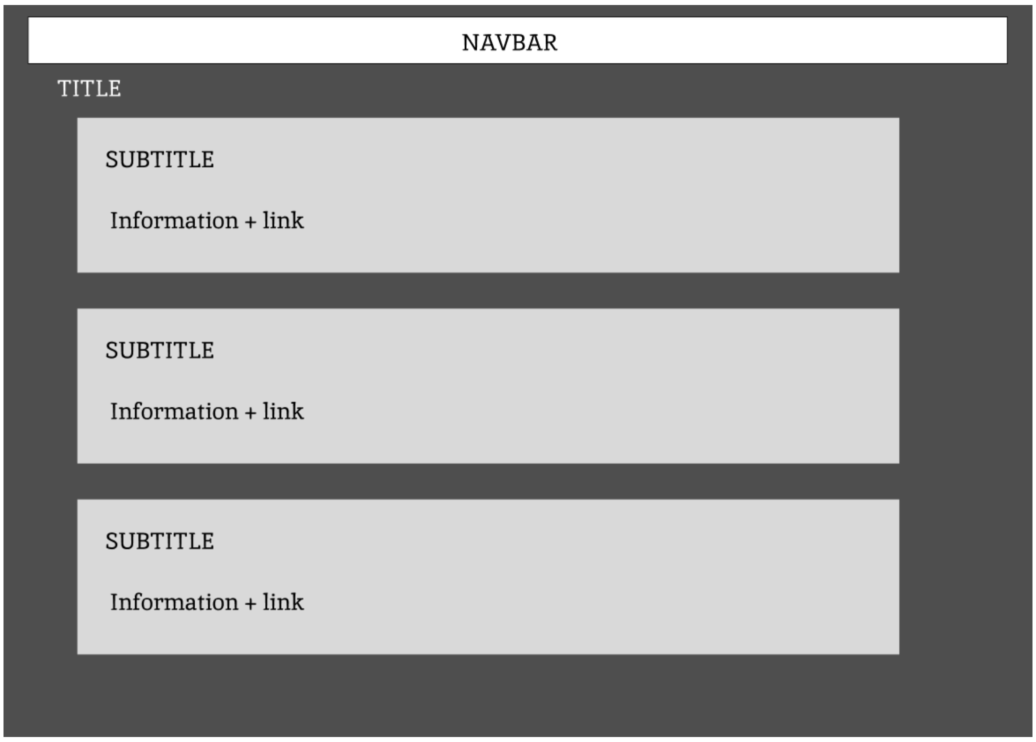


Figure 3.5: General initial design

Also, if the user presses the register button or the register link in the login view, the register form will show up like the one in Figure 3.8. This one has more options needed to register the user in the Firebase project like the username, email, password and a confirmation of it. Then there is also the possibility to change to login view.

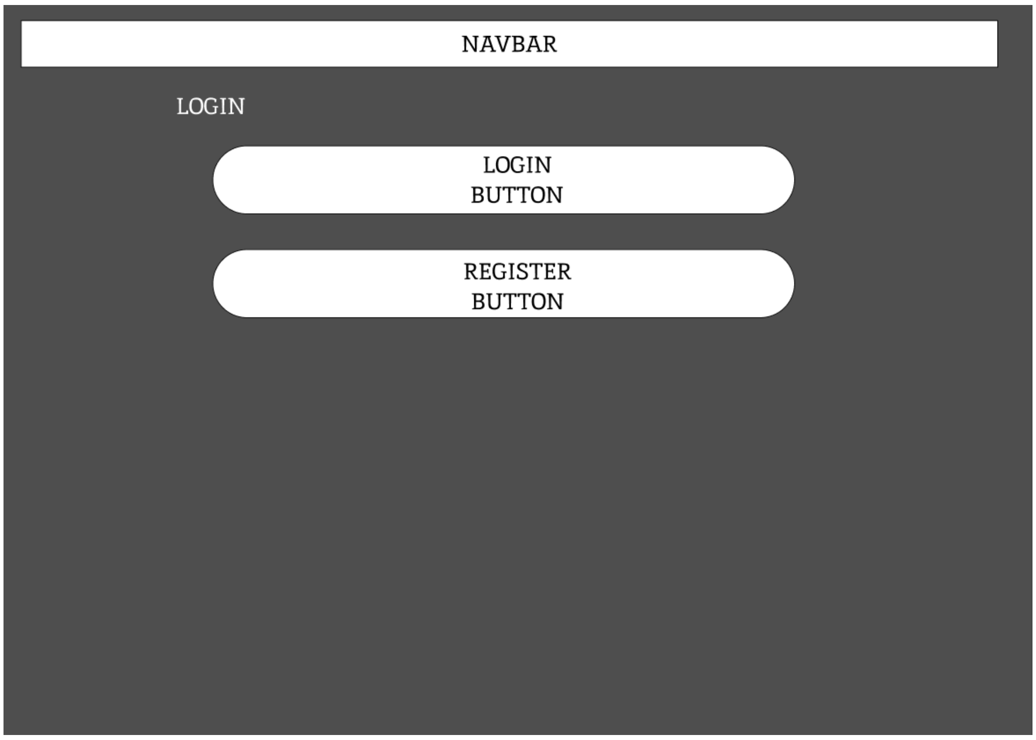


Figure 3.6: General login view design

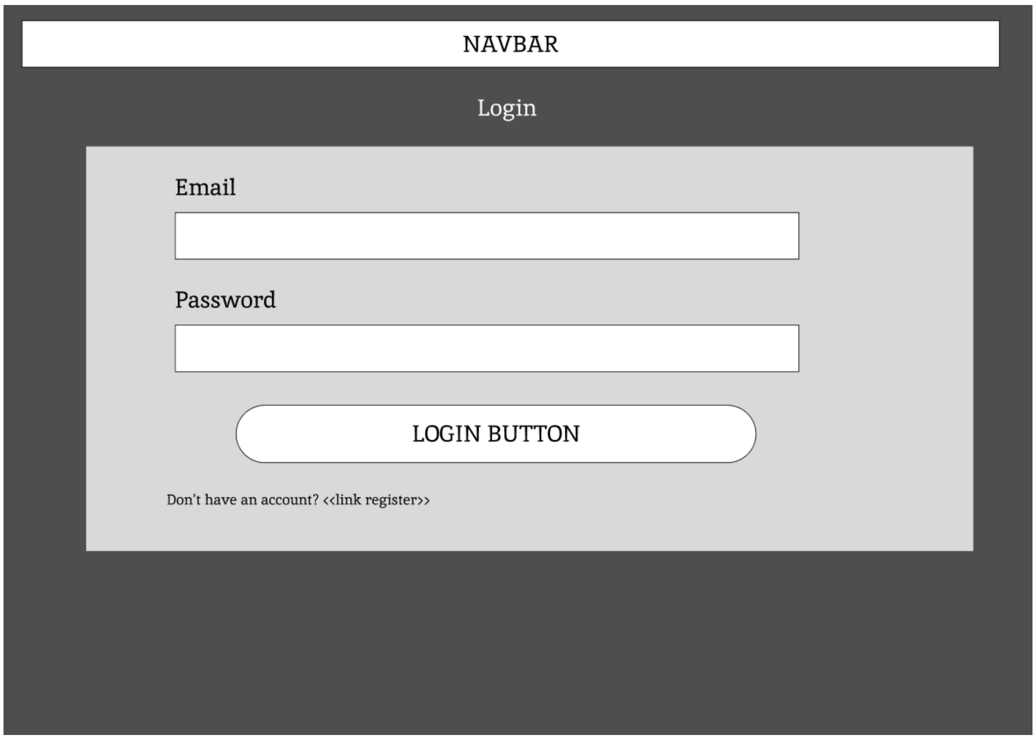


Figure 3.7: Login view design

External dependencies

The Figure 3.1 shows the initial idea for the design of the complete system, but as we have already said, this system will have an authentication subsystem for the template. In order to achieve this, it will be necessary to store somewhere all of that user information: name, email, password,... Because of that, Firebase was used as an external tool with which there is no need for the user to lunch a server or something like that, because the

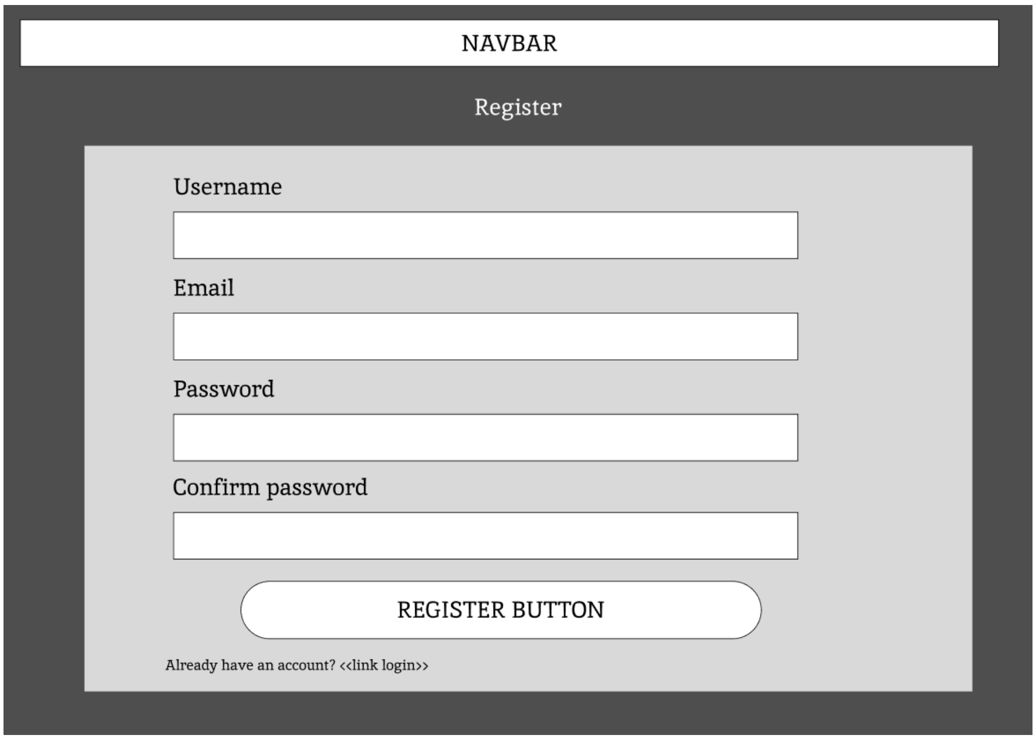


Figure 3.8: Register view design

user will create a Firebase project using the script and it's only necessary to write the name of the app. That's why the system was designed as shown in Figure 3.9.

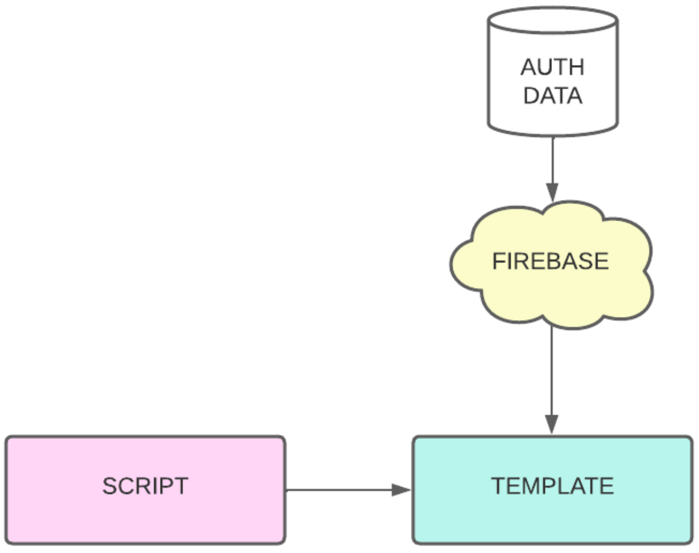


Figure 3.9: External dependencies

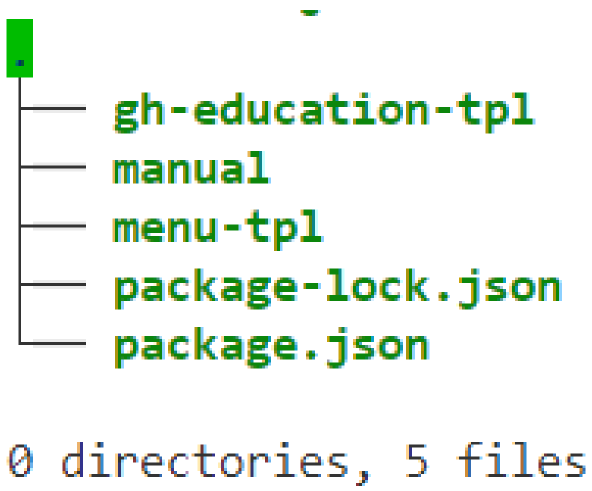
3.2.1 Project Structure

First all, this project contains two subsystems. On the one hand, we have the script that creates the template, configures and deploys it. On the other hand, the template is the site that the user can navigate through and use it for uploading information, see the teams,... So this section will be split in two.

Script Structure

Let's begin with the Script. As it has been told before, this scripts aims for cloning or creating the site, creating a basic configuration and deploy it to the cloud. In total,as shown in Figure 3.10, there are five files that make up the whole of this script functionally, but only three of them are important:

- *manual*: This is a manual created for having initial information about the script. In this manual there is some information like, description of the project, options available, version,..
- *menu-tpl*: In order to make the initial script easier to modify and understand this one was created. It aims to create an arrow system for making user's navigation easier. With this script, there is no need for the user to add options manually, it will only the arrows buttons be needed to choose the option and then press enter to confirm it.
- *gh-education-tpl*: At last but not least, the main script that is the one in charge of cloning the template, creating the Firebase app and project, does the initial configuration, calls the other scripts and builds and deploy the site.



```
.
├── gh-education-tpl
├── manual
├── menu-tpl
├── package-lock.json
└── package.json

0 directories, 5 files
```

Figure 3.10: Script project structure

Template Structure

On the other hand, the template structure is bigger and a bit complex as shown in Figure 3.11. The structure is defined by VitePress when the project is initially created. First of all, there is a *package.json* where all the dependencies are stored that the systems needs to run are stored. All the files that are going to be needed to serve the site are in the folder *docs* that we are going to analyse:

- *.vitepress*: This is one of the most important folders of all the project. In this folder there are images, configuration for the site, style-sheets, JavaScript files and some markdown files. Beginning with the a*assets* folder, it has two images that are the logo for dark and light mode for the site.

- *components*: Then we have the components, that allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organised into a tree of nested components:

  - *Auth*: This component is responsible for managing everything related with authentication. First of all it has the template part where the authentication view is defined and secondly it has the methods for registering, logging the user in or out.
  - *Ghcard*: This one is related to student and teams. The aim of this component is to create a card view in the template and add all the student information in it.
  - *General* (*Lessons*, *Tasks* and *Units*): These three components aim is creating the template for lessons, tasks and units. Also, they take information from a JavaScript file, where all the lessons, tasks and units published are and then add it to the page.
  - *Team*: This one aims to take all the information of each team, showing it using the *Ghcard* component and having more information about the team.
  - *Teams*: At last but not least this component has to take the information from the query to the GitHub organization, split it and call the *Student* component with all of it.

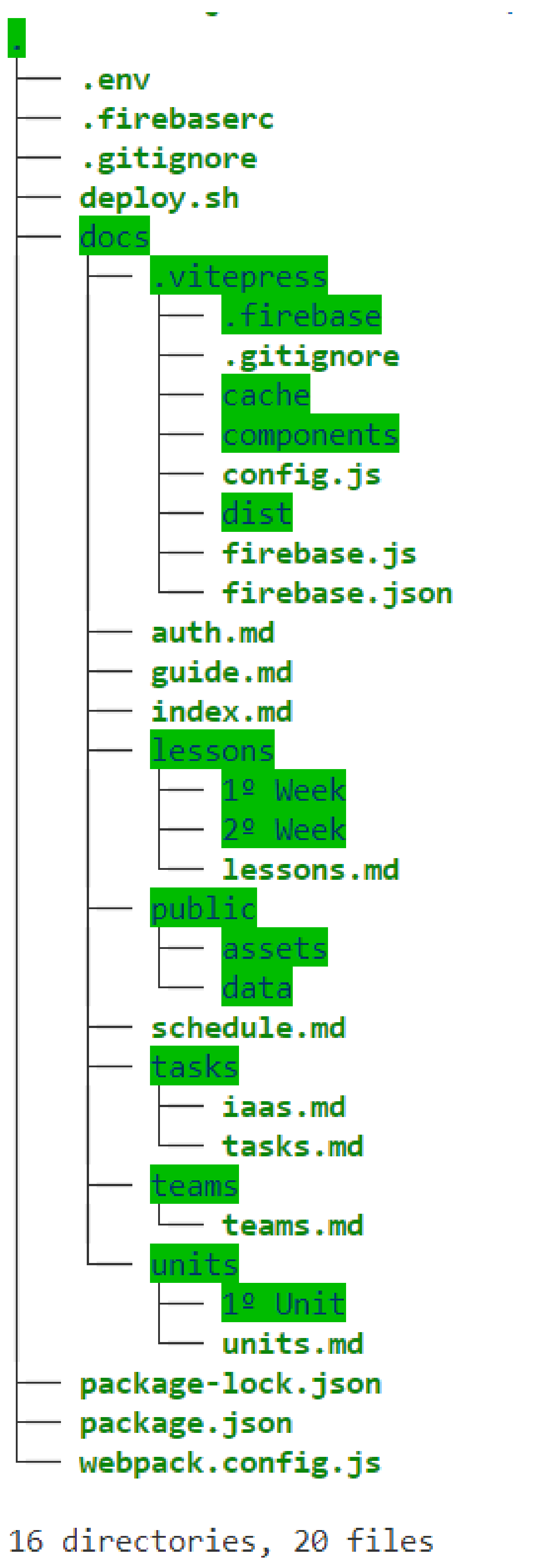
- *config.js*: This file is the configuration of the site that VitePress is going to generate. In this file we can change navbar, add plugins (in my case one that is related to working with markdown), set the headers of footer,...

- *firebase.js*: One of the files that is important for creating the configuration of Firebase. It has the initial configuration (apiKey, projectId, appId,...) that when the user creates his / her own project it will be modified. Also in this JavaScript file the app and authentication are initialised.

- *firebase.json*: When creating the Firebase project the system will ask a few questions. The aim of this file is to delete those questions so the system takes default answers from this file.

- *public*: In this folder, there are some of the files that contains information as the lessons, units and tasks published. Otherwise, the *teams.js* file contains the results of the query done by getting the information from the GitHub organization.

- *markdown files*: Finally these files thanks to how VitePress works, allow to add markdown and then when they serve, there will be a translation from markdown to HTML site with this code. So all the other files are written in markdown but then that ones are going to be traduced to HTML code. Here we have all the different pages such as authentication, quick guide, index (main page), schedule, tasks, teams, units and lessons.



```
.
├── .env
├── .firebaserc
├── .gitignore
├── deploy.sh
├── docs
│   ├── .vitepress
│   │   ├── .firebase
│   │   ├── .gitignore
│   │   ├── cache
│   │   ├── components
│   │   ├── config.js
│   │   ├── dist
│   │   ├── firebase.js
│   │   └── firebase.json
│   ├── auth.md
│   ├── guide.md
│   ├── index.md
│   ├── lessons
│   │   ├── 1º Week
│   │   ├── 2º Week
│   │   └── lessons.md
│   ├── public
│   │   ├── assets
│   │   └── data
│   ├── schedule.md
│   ├── tasks
│   │   ├── iaas.md
│   │   └── tasks.md
│   ├── teams
│   │   └── teams.md
│   └── units
│       ├── 1º Unit
│       └── units.md
├── package-lock.json
├── package.json
└── webpack.config.js

16 directories, 20 files
```

Figure 3.11: Template project structure

3.2.2 User Interface

Once we have talked about the system structure, another important thing is the user interface. In order to get a comfortable and easy way to navigate through, an initial search was carried out to know where each component should be (knowing what VitePress allows and the difficulties that the implementation could entail). It's important that VitePress adds a button into the navigation bar that allows the user to have dark or light mode of the site (it can be set dark or light by default too), so this is necessary to know to choose the colour palette. That's why, we used two different colour palettes focusing both of them in the green colour as we can see the dark one in Figure 3.12 and the light in Figure 3.13.

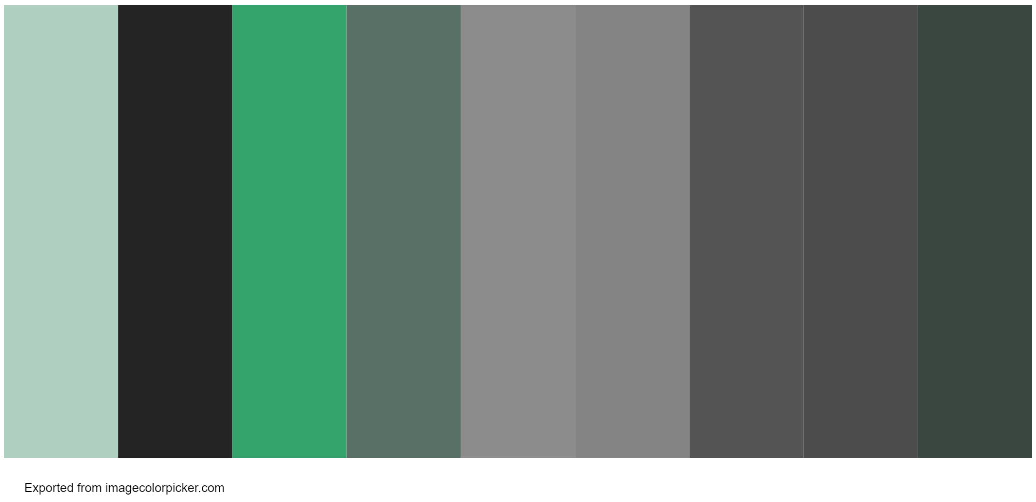


Figure 3.12: Palette colour for dark mode

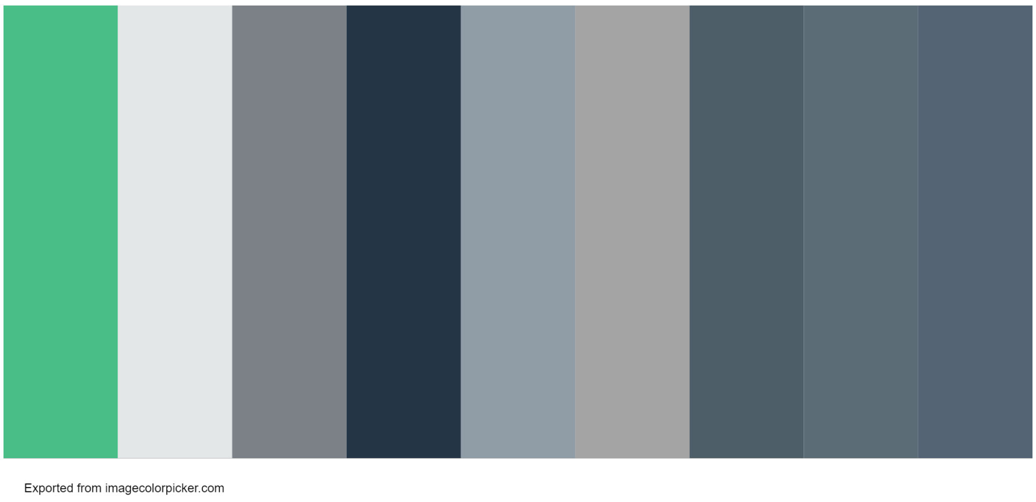


Figure 3.13: Palette colour for ligh mode

Chapter 4

Implementation

In this chapter the system will be described from a lower level of abstraction and some points of interest will be shown. Its important to mention that the project code will not be shown and explained entirely, but only those most notable fragments that allow an overview of the system to be outlined. In the annex to this report it is attached the repository that contains all the source code of the project together with the tag of the last commit made on a valid date for the delivery of this work.

4.1 Script implementation

This section seeks to make a deeper explanation of how the script development is. As it has been told all across the document, this script has been divided in important files allowing to have the content well structured and making it easier to understand and modify.

4.1.1 Manual

The first file and the easiest to explain of all. This file is a manual file that shows an small guide of how the application works. In this manual, there is a quick description, the author and also the options of the application. The options are the initial like:

- *–version*: Shows the version of the main script
- *–help*: Shows program help
- *–manual*: Shows the manual

4.1.2 Script menu

Making an easy experience for the user when using the script was one of the main objectives of this script. In order to create an option menu and letting the user to navigate through it only using the arrow keys will make the navigation easier than typing the option number of the chosen one. So this script aims to improve the navigation through the options of the script.

In order to this, there are some important functions that we need to talk about. First of all, it's important to know where is the cursor or the current option selected. There is the function used to get the position of the cursor, that is shown in the Listing 4.1. The *IFS* are the characters used as word delimiter by the command read. In this command, there are three options that have been used. The option *-s* makes the read silent and *-dR* tell read to stop at the R character instead of the default newline. After the options, there is `$'\E[6n'` that is the escape code and the storage the result into the ROW var that then it will be displayed.

```bash
# Get cursor position #
function getCursorPosition() {
    IFS=';' read -sdR -p $'\E[6n' ROW COL; echo ${ROW#*[};
}
```

Listing 4.1: Get cursor position function

When the user wants to change between options, it's important to have a function that can change that position of the cursor. Listing 4.2 shows the code of changing the position of the cursor. To understand how this works it's important to read the tutorial of how the movement in bash works. This article (4) shows how to change the position of the cursor, resets it or saves the current cursor position. As we said, to change the position of the cursor to another it is important to follow this syntax (shown in the article) \033[<L>;<C>H. It is similar to the code, but the \033 is an environment variable and <L> is a value that is passed as an argument. The <C> is the column value and is the same because we want that the cursor stays in the same column but different row.

```bash
# Change cursor position #
function setCursorPosition() {
    printf "$ESC[$1;${2:-1}H";
}
```

Listing 4.2: Change cursor position function

Now that the position of the cursor can be obtained and set, the next step is to read the input and modify it. In order to do this, as shown in Figure 4.3, as we said before, the arrows keys will be used for navigating through the menu. So in this function there is a silent read of the input and then this will change the input to up, down or enter depending on the key pressed.

```bash
# Get input key #
function setKeyInput() {
    read -s -n3 inputKey 2>/dev/null >&2
    if [[ $inputKey = $ESC[A ]]; then echo up;    fi
    if [[ $inputKey = $ESC[B ]]; then echo down;  fi
    if [[ $inputKey = ""     ]]; then echo enter; fi
}
```

Listing 4.3: Get input key function

In this script there are three more functions that beautify the output of the script adding symbols around the menu, displaying the text in colour if it is selected or normal and if it is in the center or the right of the shell. At the end, the main function, shown in Listing 4.4, is the one that uses the other side functions to make a complete menu. This one input is the available options, then it stores the first row and the total number of rows. Then, for each option it changes the cursor position and check which is the option selected to change the background color (simulating that is the current option). Then when all the options are added, the next is to check the key pressed and depending if it is up or down move the selected option and refresh the shell. This will be done until the key enter is pressed, returning the current option.

```bash
# - - MAIN FUNCTION - - #
function chooseOption() {

    for opt; do printf "\n"; done

    local rowNumber=‘getCursorPosition‘ # Get total number of rows
    local firstRow=$(($rowNumber - $#)) # Get intial row of the menu

    local currentOption=0
    while true; do
        local optionIndex=0
        for opt; do
            setCursorPosition $(($firstRow + $optionIndex))  # Change cursor position to the new option
            if [ $optionIndex -eq $currentOption ]; then
                displayTextRight 1 " [$((optionIndex+1))]   $ESC[7m $opt $ESC[27m ";  # Selected option (white background)
            else
                displayTextRight 0 " [$((optionIndex+1))]    $opt ";  # Non selected option (no background color)
            fi
            displayTextCenter " "
            rowOfSymbols 1 maxCols
            ((optionIndex++))
        done

        case ‘setKeyInput‘ in
            enter) break;; # Break while and eturn option choosed
            up)    ((currentOption--));
                   if [ $currentOption -lt 0 ]; then currentOption=$(($# - 1)); fi;;  # Change choose option to upper one
            down)  ((currentOption++));
                   if [ $currentOption -ge $# ]; then currentOption=0; fi;;  # Change choose option below
        esac
    done

    setCursorPosition $rowNumber
    printf "\n"

    return $currentOption
}
```

Listing 4.4: Main choose menu function

4.1.3 Main Script

After talking about the manual and the menu it's time to explain how the main script works and how it uses the other scripts. In this script there are a lot of variables and functions so, in order to understand better it, we will be splitting this script into different subsections depending on the type of function we are talking about (menu, output, clone, authentication,...).

Input & Output

All the scripts need to have input and output functions. In this case, for the input, all the information is going to be introduced using the menu that was explained before. Also, some of the information will be introduced manually in order to configure the system and make some quick details for the system. For the input it is important to mention that there are a few functions that create the standard menu, add a box drawed with symbols, display text centered or on the right (these two are functions too).

On the other hand, the output has more functions because it's important to take care of the quality of the output. First of all, there is a function that allows to add color for the options or the text (all the used color are defined as global variables). Also, like in some process the user needs to wait for it to be finished, there was an idea to add a progress bar. With this functionality, the user can know when the process finishes and then start with the next one (this bar wait until the process finish and can be shown in different colours).

Handling functions

Sometimes it is important to check the files or errors in order to simplify the user's experience. In this category, there is one function related with the cloning one. This aims to check if the clone folder has been already created, it can allow the user to skip the cloning process and get to another one, if not the user has to clone again the template. Also, it will be an important feature to add error handling to the system because if one command fails, the program will show the error but tries to run the other functions. The problem comes here, if that command gets information that other function can be in trouble for the final result.

Menu

As it was told in the section 4.1.2 the menu aims to get a better user experience for the user. So, in this script, when a menu is created it calls the script menu and then passes the available options that the user can choose. Also, this script has an initial function in order to set a default menu using a box (this one is created depending on the size of the current shell that the user is using) and adding a title (sometimes using the yellow for text background color). The options will be shown in the center of this box.

Cloning

The idea of having a separate script from the template is to help the user all through the process of creating, building and deploying it. This is one of the most important functions because it is going to download the template, as we can see in the Listing 4.5 from GitHub (adding a progress bar to show the user that the cloning process is running) in the current path, but the complete path will be shown in the shell and then show a successful message if this process has been done correctly.

```
## Take the template and clone it in the clone folder  ##
cloneRepo() {
    echo -ne "$(printcolor $YELLOW '# CLONING TEMPLATE #' )"
    echo -e "\n"
    checkFolder
    echo -e "\n"
    echo -ne "$(printcolor $YELLOW "Cloning in ${FILEPATH}" )"
    echo -e "\n"
    progressbar 'printcolor' $CYAN 'Clonning' & pid="$!"
    cloneCmd="git clone git@github.com:gh-cli-for-education/gh-education.git ${FILEPATH}"
    cloneCmdRun=$($cloneCmd 2>&1)
    kill -s USR1 "$pid"
    echo -ne "\n $(printcolor $GREEN 'Clone completed succesfullly' ) \n"
}
```

Listing 4.5: Clone repository function

Authentication

It is a good idea to have an authentication system because it can allow to hide or show some content depending on the type of user that is logged. For example, if there is a professor logged, he or she can have all the content in the site but show some of them to their students in some punctual point, they can also have tasks and correct it.

So, to have this, its important to know a few things about the authentication with Firebase. First, to have an authentication system with Firebase it's important to have a Firebase project linked with an app. This is because the user will log into that project. The project that is going to be used for default is an example project but the user can create one on his / her own (in this script).

First of all, the script needs to install all the dependencies of Firebase (that are in the package json of the docs project, that is why we change to that directory and run an *npm install*). After all of the packages are installed, the next step is to login the user. To make this step easier, the command *firebase login* is in charge of asking all the information like the user, password,... In case the user is new or it can be an old one, it's a good idea to add a menu for the user to show a list of the existing projects and choose one or the available to create a new one as shown in the Listing 4.6 having the options create project, list project and choose existing project.

```
while [ $exit != 0 ]
    do
        printf "\033c"
        case $choice in
            0)
                printf "\033c"
                echo -ne "$(printcolor $CYAN '# Project ID: ' ) $projectID\n"
                firebase projects:create $projectID
                firebase use $projectID
                firebase apps:create WEB $projectID
                exit=0
                ;;
            1)
                firebase projects:list
                echo -ne "$(printcolor $YELLOW '# PRESS ENTER TO EXIT #' )\n"
                read -r

                options=("Create new project" "List existing projects" "Change current project" "Exit")

                printf "\033c"
                setMenu "# FIREBASE PROJECT CREATION #"
                chooseOption "${options[@]}"
                choice=$?
                ;;
            2)
                firebase projects:list
                echo -ne "$(printcolor $CYAN '# Project ID to change: ' )"
                proejctID=""
                read -r projectID
                firebase use $projectID
                exit=0
                ;;
            3)
                echo -ne "$(printcolor $RED 'exiting...' )"
                exit=0
                ;;
        esac
    done
```

Listing 4.6: Firebase project menu

As we have said, it's important to change the default project to the new or the project that the user is going to choose. First, the project needs SDK configuration in order to communicate with the Firebase project, something similar to the Listing 4.7 with the project values. In order to do this, as shown in the Listing 4.8, there is a command that allows to get this information and, in my case it is going to be written in a file, changing the default one with the new configuration.

```
const firebaseConfig = {
        apiKey: "fill_me",
        authDomain: "fill_me",
        projectId: "fill_me",
        storageBucket: "fill_me",
        messagingSenderId: "fill_me",
        appId: "fill_me",
        measurementId: "fill_me"
};
```

Listing 4.7: Firebase Project SDK

```
## Refresh firebase configuration file with new project ##
refreshFirebaseConfig() {
    fichero="./firebase.js"
    if [[ -f $fichero ]]; then
```

```
        rm $fichero
    fi
    echo "
import { initializeApp } from 'firebase/app'
import { getAuth } from 'firebase/auth'
const firebaseConfig = " > $fichero
    firebase apps:sdkconfig WEB --project $1 --json > "./firebase-config.json"
    cat ./firebase-config.json | jq ".result.sdkConfig" >> $fichero
    echo "
const app = initializeApp(firebaseConfig);
const auth = getAuth();

export { auth }; " >> $fichero
}
```

Listing 4.8: Refresh firebase configuration file

Project creation and Deployment

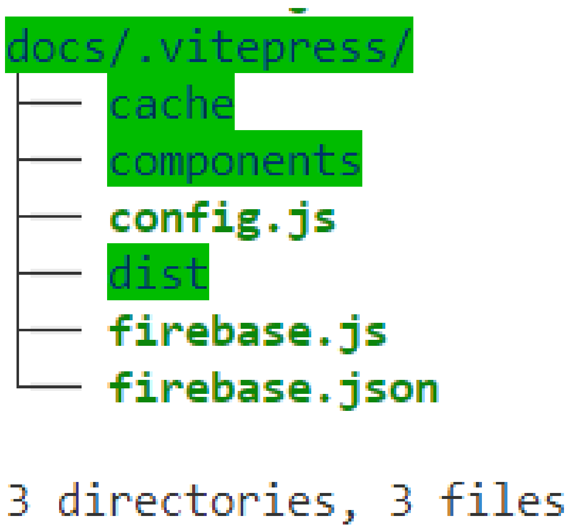
Once the user is logged in and the project configuration is all replaced its time to get the project built and deploy it. To make this step easier, there is a script in the template side called *deploy.sh*. This allows the user to deploy from template or from the script (because the template is cloned). In the script, first an environment is set with some information (repository to deploy, branch and organization), make a build of the project, then initialise the repository (to deploy to GitHub pages) and make a push. The script is shown in the Listing 4.9.

```
#!/bin/bash

# abort on errors
set -e

set -o allexport
source .env
set +o allexport

# build
npm run docs:build

# navigate into the build output directory
cd docs/.vitepress/dist

git init
git add -A
git commit -m 'deploy'

git push -f git@github.com:$VITE_ORGANIZATION/$VITE_REPOSITORY.git $VITE_BRANCH:gh-pages
# firebase deploy

cd -
```

Listing 4.9: Deploy Script

4.2 Template implementation

Once we have talked about the script implementation and all the subscripts developed it's time to go with the Template. It is important to remember that this template is created using Vitepress, so it has a recommended and a fixed structure for some files. All the information related with components, assets, CSS and configuration is going to be in a hidden folder called *.vitepress* as shown in Figure 4.1.



```
docs/.vitepress/
├── cache
├── components
├── config.js
├── dist
├── firebase.js
└── firebase.json

3 directories, 3 files
```

Figure 4.1: Template Vitepress files

First of all, the assets folder aims to have all the images that the web page is going to show, in this case there are only two logos. Next, the cache folder is generated each time we build or serve the project in order to load faster. In this case, the CSS folder can be added to the assets one and the public one has some public information for the user that will be explained later.

4.2.1 Store

As it was told in the implementation chapter, it's important to handle the state of the application to have this information shared between components. Thanks to Vuex this is possible from the creation of a store. The state of the application initially will be null, having the user information (Firebase authentication object), GitHub token and the user role. Once the state structure is known the next step is to determine the mutations (functions to change the state) as shown in the Listing 4.10. These three mutations allow to change the three properties, having two arguments, the current state and the new value of that state. These are going to be called, for example, when the user authenticate into the system these three are called.

```
mutations: {
        //Mutation to update the user state
        //Takes in two arguments, the state and the payload. When we call this mutation,
    the payload will be user object from firebase auth
        //When the user logs out, we call the mutation and the payload will be null
        setUser(state, payload) {
            state.user = payload
        },

        setToken(state, payload) {
            state.token = payload
        },

        setRole(state, payload) {
            state.role = payload
        }
}
```

Listing 4.10: Store mutations

Once the mutations are explained the next step is talk about the actions, these are the functions that are going to be called to make async operations and commit the mutations of the state. The first action is the login that uses an argument and it is the organization as shown in the Figure 4.11. In this action, the user logs in the system using the GitHub authentication provided in Firebase. The authentication displays a pop up window and, when the user is logged in the system, thanks to the authentication GitHub token, a query is made to get the role of that user in the organization. Finally these values are committed to the store allowing to use them in other components.

```
async login(context, { organization }){

    const provider = new GithubAuthProvider();
    provider.addScope('repo');
    provider.setCustomParameters({
            'allow_signup': 'false'
    });

    const result = await signInWithPopup(auth, provider)
    if (result) {

        const credential = GithubAuthProvider.credentialFromResult(result);
        const token = credential.accessToken;
        const endpoint = "https://api.github.com/user/memberships/orgs/" + organization;

        const response = await fetch(endpoint,{
            method: "GET",
            headers: {
                'X-GitHub-Api-Version': '2022-11-28',
                Authorization: `token ${token}`
            }
          })

        let organizationRole = await response.json();

        const role = (organizationRole.role == 'admin') ? 'teacher' : 'member'

        context.commit('setUser', result.user,)
        context.commit('setToken', token)
        context.commit('setRole', role)

    } else {
        throw new Error('login failed')
    }
}
```

Listing 4.11: Login action

The other action is to logout the user from the system. As it is shown in the Listing 4.12, the user session is cleared and called to the sign out Firebase function. Then when the user is logged out successfully, the default null values for the state are committed.

```
async logout(context){
    await signOut(auth)

    sessionStorage.clear();

    context.commit('setUser', null)
    context.commit('setToken', null)
    context.commit('setRole', null)
}
```

Listing 4.12: Logout action

4.2.2 Components

The most important part of the implementation comes here. The components allow to split the UI into independent and reusable, and think about each piece in isolation. This is very similar to how we nested native HTML elements but Vue implements its own component model that allows us to encapsulate custom content and logic in each component.

As we talked in the section 2.3, the component has different language blocks. All of the files that we are going to talk about has <template>, <script>, <style> and <custom blocks> following the structure that we have been talking about in this project. In some cases, in some component, other components will be used inside. To introduce components an UML diagram has been done as shown in the Figure 4.2, where all these components are going to be explained all of them going from the easiest to the most difficult one.

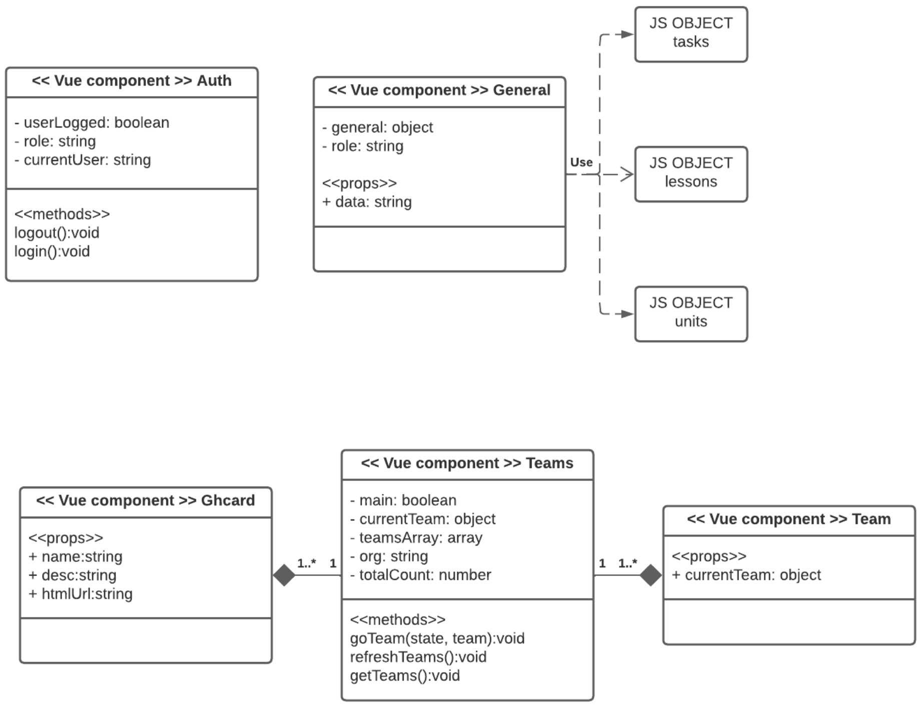


Figure 4.2: UML template diagram

Otherwise, it is important to mention that, thanks to the state management, the content between components will be shared. But in order to do this, it is important to know a bit about Vue lifecycle. Each Vue component instance goes through a series of initialization steps when it's created - for example, it needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes. Along the way, it also runs functions called lifecycle hooks, giving users the opportunity to add their own code at specific stages. So to achieve this, in each component beforeMount() is used, thanks to this, the system can check if an app has been created and then, if it is true set the store values to their value.

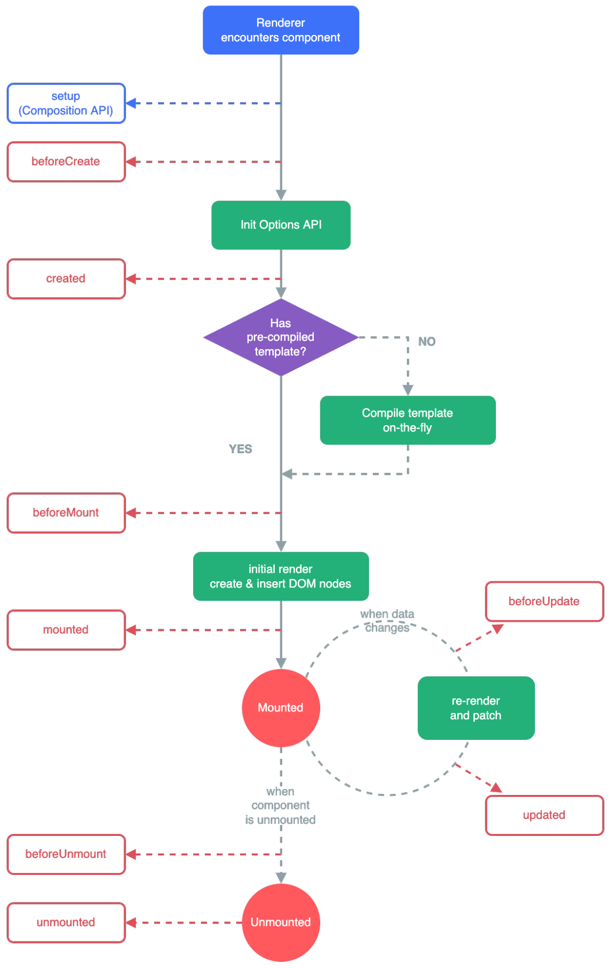
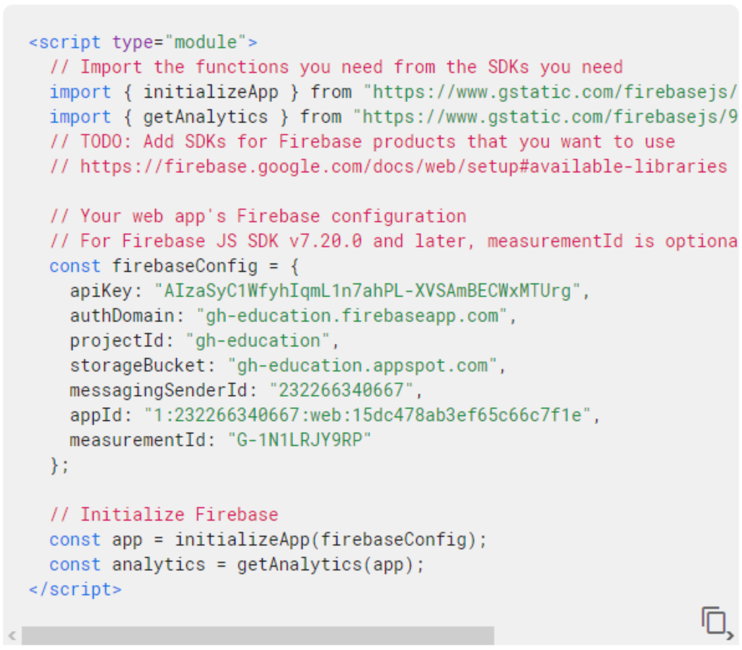


Figure 4.3: Vue Lifecycle

Auth

As the other three components have dependencies among them, lets talk first about the Auth component. As it was told in the chapter 2, there is a technology that is going to be used for the user authentication, Firebase. Firebase allows the user to have his / her own project with authentication, analytic, storage,... However, in order to login or logout the user, this is going to be handled by the store that is the one in charge to hold the user state, in this function there are only call to mutations that change the state of the application

Also, there is on thing that is important to talk about. All the related education system needs to have two roles, one for the teacher and the other for the student. That is because it makes easier to hide or show content and allows to have it on the web and only needs to show or hide. As it was told before, one of the Firebase suppliers is GitHub so it makes interesting to have the users logged there, allowing to use the organization data and with this assign the roles. It's easy to set the roles, the only thing that is important is the role of that user on that organization depending if he / she is member (student) or owner (teacher). Once this authentication type is activated, it's important to add the SDK configuration that Firebase show when the project is created as it's shown in Figure 4.4.



```
<script type="module">
  // Import the functions you need from the SDKs you need
  import { initializeApp } from "https://www.gstatic.com/firebasejs/
  import { getAnalytics } from "https://www.gstatic.com/firebasejs/9
  // TODO: Add SDKs for Firebase products that you want to use
  // https://firebase.google.com/docs/web/setup#available-libraries

  // Your web app's Firebase configuration
  // For Firebase JS SDK v7.20.0 and later, measurementId is optiona
  const firebaseConfig = {
    apiKey: "AIzaSyC1WfyhIqmL1n7ahPL-XVSAmBECWxMTUrg",
    authDomain: "gh-education.firebaseapp.com",
    projectId: "gh-education",
    storageBucket: "gh-education.appspot.com",
    messagingSenderId: "232266340667",
    appId: "1:232266340667:web:15dc478ab3ef65c66c7f1e",
    measurementId: "G-1N1LRJY9RP"
  };

  // Initialize Firebase
  const app = initializeApp(firebaseConfig);
  const analytics = getAnalytics(app);
</script>
```

Figure 4.4: SDK Firebase configuration

Once the configuration is all done, the next step is to show the data or variables are going to be used in this component. As we know the data is defined in the script section as is shown in the Listing4.13. Now it's time to see how the template is developed. On the beginning, the approach of the template aims to have a Login, Signup form and a welcome for the user if he / she is logged in. But with this structure, the user needs to have a usermail, password and needs to register into the site. In order to allow the user to login with GitHub, Firebase has some functions to handle this and it is the same for login and register so it simplifies the views only having one button with the pop up for making this authentication with GitHub and the organization.

```
data() {
    return {
        userlogged: false,
        role: null,
        currentUser:null
    }
}
```

Listing 4.13: Authentication component data

The first part of the template as shows the Listing 4.14 is the authentication view. With the authentication in Firebase using GitHub there is no need to have a login and a register form because the GitHub function handles this checking if the user has registered the app or not. Depending of this, the pop up will show an authentication view or will log in the user.

```
<template>
    <div>
        <!-- Check if user is logged to show login buttons if not -->
        <div v-if="role == null">
            <h1> LOGIN FIREBASE (GitHub) </h1>
            <button type="button" @click="login" cla> Login </button>
        </div>
        <!-- The user is logged, it shows a welcome -->
        <div v-else>
            Welcome {{role}} {{currentUser}}
          <button type="button" @click="logout"> Logout </button>
        </div>
    </div>
</template>
```

Listing 4.14: Template Auth component

Entering into the methods, as it was said before, the store is in charge of login the user into the application so that makes the component easier to work with. In the login or logout function, there is only need to dispatch the action in the store and then setting the data depending on the method. If it is logout, setting them to null (default value) or to the new authentication values as shown in the Listing 4.15.

```
/**
 * Logout a user using Firebase auth
 */
async logout() {
    await store.dispatch('logout', {});
    this.role = null;
    this.currentUser = null;
},

/**
 * Login a user using Firebase auth with Github popup window
 */
async login() {
    await store.dispatch('login', { organization: import.meta.env.VITE_ORGANIZATION })
    const storeData = store.getters.userData;
    this.role = storeData.role;
    this.currentUser = storeData.user.displayName;
}
```

Listing 4.15: Login and logout methods

General

First of all the General component aims to show general information. In this case, the first approach was having three different components for Tasks, Lessons and Units but, the code written in these components were the same, the only thing that was different was the imports used (the public files that contains information about the Lessons, Units and Tasks). In order to fix this, there is a feature in Vue called properties. These allow to pass certain information in the component tag, as it's shown in Listing 4.16. With this feature, it is now possible to pass the file path that we want to add (depending if the site is in development or production mode), depending the markdown that we want to show we pass one or other. The other problem now is in the component, to get this information, the component needs to call the component before being mounted and to do a dynamic import adding the result value to a local component variable as shown in Listing 4.17.

```
<script setup>
    import general from '../.vitepress/components/General.vue'
    import { useData } from 'vitepress'
    const { site } = useData()
    const filePath = (site.value.base === "/") ?  "/public/data/units.js" : site.value.
    base + "/data/units.js";
</script>

# Units

<general :data="filePath"></general>
```

Listing 4.16: Units mardown example

```
  props: {
    /**
     * String with the file with the data to dynamic import
     */
     data: {
       type: String,
       required: true,
    }
  },

  async beforeMount() {
      if (getApps().length !== 0 ) {
          const storeData = store.getters.userData;
          this.role = storeData.role;
          let info = await import(/* @vite-ignore */this.data);
          this.general = info.default.data;
      }
  }
```

Listing 4.17: Units component prop and dynamic import

Ghcard

Once all the non dependent components have been explained, then there are three components that depend on others. In this section, we will be talking about the Ghcard. This components aims to create a profile card based in GitHub colours to show the teams information. As we can see in the UML of the Figure 4.2, Teams contains one or more Ghcard, this is because the teams are built with all the teams of an organization and normally it can be one or more.

The Ghcard aims to have a summary of the team shown in a card. To have this, the card will have, as shown in the Listing 4.18, first an image will be added, then at the other side some information about the team as the GitHub profile, the commits or the notifications. All the variables used in this component are props, this means that all the information is passed from another component, and then the information is replaced in this component with that data. Then in the style block, there is the CSS that allows to create the card and show it in the site.

```
<template>
  <div class="card">
    <h3 class="name">{{ name }}</h3>
    <div class="flex">
      <!-- Image for profile card picture -->
      <div class="media">
        <img :src="image" :alt="ghuser" />
      </div>
      <!-- Summary of information about the team -->
      <div class="details">
        <div class="author">
          <p class="ghuser">{{name}} github: <a :href="href">{{ ghuser }} </a> </p>
          <p class="ghuser">{{name}} <a :href="repositoryUrl"> repositorio  </a> </p>
          <p class="ghuser">{{name}} <a :href="notifications"> notificaciones </a> </p>
        </div>
      </div>
    </div>
  </div>
</template>
```

Listing 4.18: Ghcard component template

Team

Then there is another component that is being used by the Teams. This component is similar to the Ghcard one. It has only one prop that is a team. The component aims to get all the information about one team. When the user clicks on the Ghcard, that is the summary, it changes view to this page, with all the information of the current Team. In the Listing 4.21 it shows how the template is written. First it adds the user summary (profile, commits and notifications), something like the Ghcard. And then, using the information that the query wrote in the file is going to be used to get all the repositories and show for each of them, the name, a link to the commits, issues and notifications. This can be done thanks to the v-for feature that allows to show each element in a vector.

```
<template>
  <button type="button" @click="$emit('change', true)" cla> Go back </button>
  <br/>
  <div class="card">
    <br/>
    <!-- Summary of information about the team -->
    <div class="details">
      <div class="author">
        <p class="ghuser">{{currentTeam.name}} github: <a :href="currentTeam.htmlUrl">
    link </a> </p>
      </div>
      <!-- Top 10 reposiories of that Team -->
      <p class="ghuser"> Recent projects: </p>
      <li v-for="(repository, section) in currentTeam.repositories" :key="section">
        {{currentTeam.name}}  <a :href="repository.commits_url">Commits </a>, <a :href="
    repository.issues_url">Issues </a>, <a :href="repository.notifications_url">
    Notifications </a> for <a :href="repository.url"> {{repository.name}} </a>
      </li>
      </div>
  </div>
</template>
```

Listing 4.19: Team component template

Teams

Finally the Teams component that, as we said before, it uses Ghcard and team components, it's important to add them in the component section and then the data as shown in the Listing 4.20. In the important variables the current team, teamsArray that have all the array information of the organization team, org and the totalCount (total number of teams).

```
export default {
  components: {
    ghcard,
    Team
  },
  data() {
    return {
      main: true,
      currentTeam: {},
      teamsArray: [],
      org: "",
      totalCount: 0,
      role:null
    }
  }
```

Listing 4.20: Teams component data

After explaining the data, the next step is to show the template in Listing 4.21. There is a flag that allows to change between the main view (all teams) and the current team view. In the Ghcard tab, there is a click function that changes the view to the team and all the information as its shown is passed with the name of the props in the Ghcard component. Then the team uses another prop that is the currentTeam (this is all the data of the team that is clicked).

```
<template>
  <!-- Show the view of all the teams -->
  <div v-if="main && ((role !== null) || role === 'Owner')">
    <div class="flex">
      <h1> Teams </h1>
      <button type="button" class="refresh" @click="refreshTeams()"> Refresh teams <
    button>
    </div>
    <h2> Organization: {{ org }} </h2>
    <h2> Number of teams: {{ totalCount}} </h2>
    <!-- Display all teams and create a ghcard for each of them -->
    <div v-for="(team, section) in teamsArray" :key="section">
      <h3> Equipo: {{ team.name }} {{section + 1}} / {{ totalCount }} </h3>
      <br/>
      <div class="card">
        <ghcard
          :htmlUrl="team.htmlUrl"
          :name="team.name"
          :desc="team.desc">
        </ghcard>
        <button type="button" @click="goTeam(false, team)"> More information </button>
      </div>
    </div>
  </div>

  <!-- Show the view of one of the teams -->
  <div v-if="!main && ((role !== null) || role === 'Owner')">
    <h1> Team {{currentTeam.name}} </h1>
    <Team :currentTeam="currentTeam" @change="goTeam(true, team)"> </Team>
  </div>
</template>
```

Listing 4.21: Team component template

In the template function to get teams, see Listing 4.22. In order to be available in runtime, there is a refresh option so there is no need to have a static file with teams information. Instead of that, it is important to make queries to GitHub API to get the new information (taking the current organization and repository from the environment file). There are three important queries in this function. First, the endpoint aims to get all the teams from an organization. Once we have them, the next step is use the slug property of each team (similar to team name) to get the repositories of each team in the organization. With this information then, the url of each repository can be created using the GitHub base url plus repository name and the feature we want to aim to (issues, commits, notifications,...). Finally each team is created with the important information.

```
async getTeams() {
      const storeData = store.getters.userData;
      const organization = import.meta.env.VITE_ORGANIZATION;
      const base_url = "https://github.com/" + organization;
      const orgTeamsEndpoint = "https://api.github.com/orgs/" + organization + "/teams";
      const teamsArray = [];

      const orgTeamsResponse = await fetch(orgTeamsEndpoint,{
          method: "GET",
          headers: {
              'X-GitHub-Api-Version': '2022-11-28',
              Authorization: `token ${storeData.token}`
          }
        });

      const teamsOrganization = await orgTeamsResponse.json();

      for (const orgTeam of teamsOrganization) {
        const teamsRepositoryEndpoint = "https://api.github.com/orgs/"+ organization + "/
    teams/" + orgTeam.slug + "/repos";
        const orgTeamResponse = await fetch(teamsRepositoryEndpoint,{
            method: "GET",
            headers: {
                'X-GitHub-Api-Version': '2022-11-28',
                Authorization: `token ${storeData.token}`
            }
          });

        const teamRepository = await orgTeamResponse.json();
        teamRepository.forEach( repo => {
          const repository = "/" + repo.name
          repo.url = base_url + repository;
          repo.issues_url = base_url + repository + "/issues";
          repo.commits_url = base_url + repository + "/commits";
          repo.notifications_url = base_url + repository + "/notifications";
        })

        const newTeam = {
          name: orgTeam.name,
          htmlUrl: orgTeam.html_url,
          desc: orgTeam.description,
          repositories: teamRepository
        }

        teamsArray.push(newTeam);
      };

      this.totalCount = teamsOrganization.length;
      this.org = organization;
      return teamsArray;
    }
  }
```

Listing 4.22: Teams component function

4.3 Final system

Finally, lets see a quick look to the final design and features of the system. Thanks to Vitepress it is easier to create a navigation bar to guide the user between the pages of the application. In the configuration file, Vitepress allow to create a navigation bar in the top or a side bar that can be useful depending on the system developed. For example, for this case, the navigation bar has five different links to other pages like Home, schedule, lessons, tasks, units, teams and the login like it's shown in the Figure 4.5. It is important to say that all the pages are written using markdown and importing the Vue components or using markdown extensions that available in Vitepress.

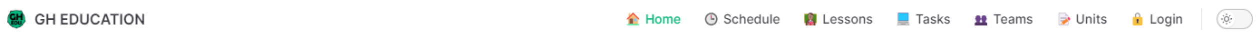
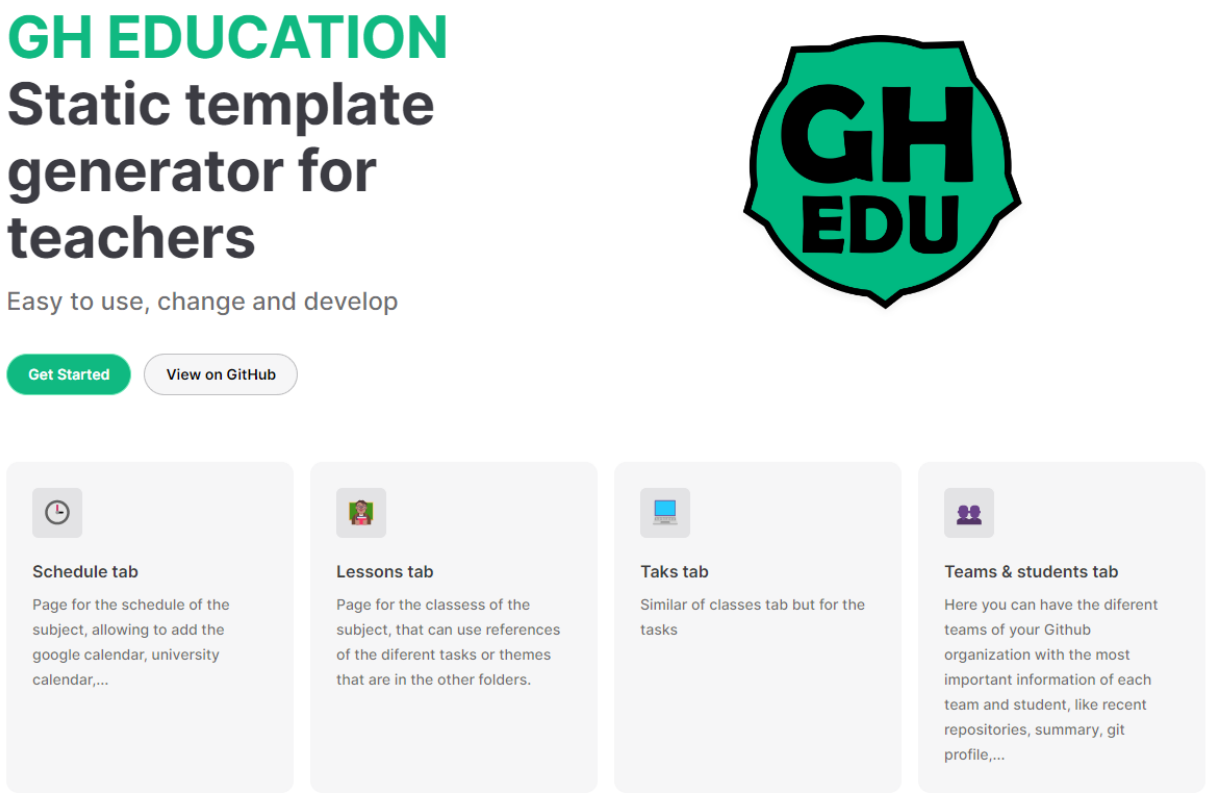


Figure 4.5: Navigation bar

So, the system has different views, depending on the state of the user some of them are going to be displayed and other not, for example, if the user is not logged, the user cannot see lessons, tasks,.. but if it is logged then that will be displayed:

1. Home Page: The homepage is the main page of the system. There is the name of the system (all of this can be customised in each markdown) explanation of the features developed in the system, one button to the GitHub template and other to a quick guide to understand how the system structure, features works. As it is shown in the Figure 4.6 thanks to the Vite YAML frontmatter that allows to create an information section(logo and buttons) and a feature section.

2. Schedule: In the education environment it is important to have a schedule where the students can get information about the classes, links for some events in Google Calendar,... In order to get this feature, in the Figure 4.7, it's shown that there is a page to have quick information about the methodology of the subject, some links to the university calendar and the Google Calendar too. This page can have exam dates too, or anything related to this topic.


GH EDUCATION
Static template generator for teachers

Easy to use, change and develop

Get Started | View on GitHub

Schedule tab

Page for the schedule of the subject, allowing to add the google calendar, university calendar,...

Lessons tab

Page for the classess of the subject, that can use references of the diferent tasks or themes that are in the other folders.

Taks tab

Similar of classes tab but for the tasks

Teams & students tab

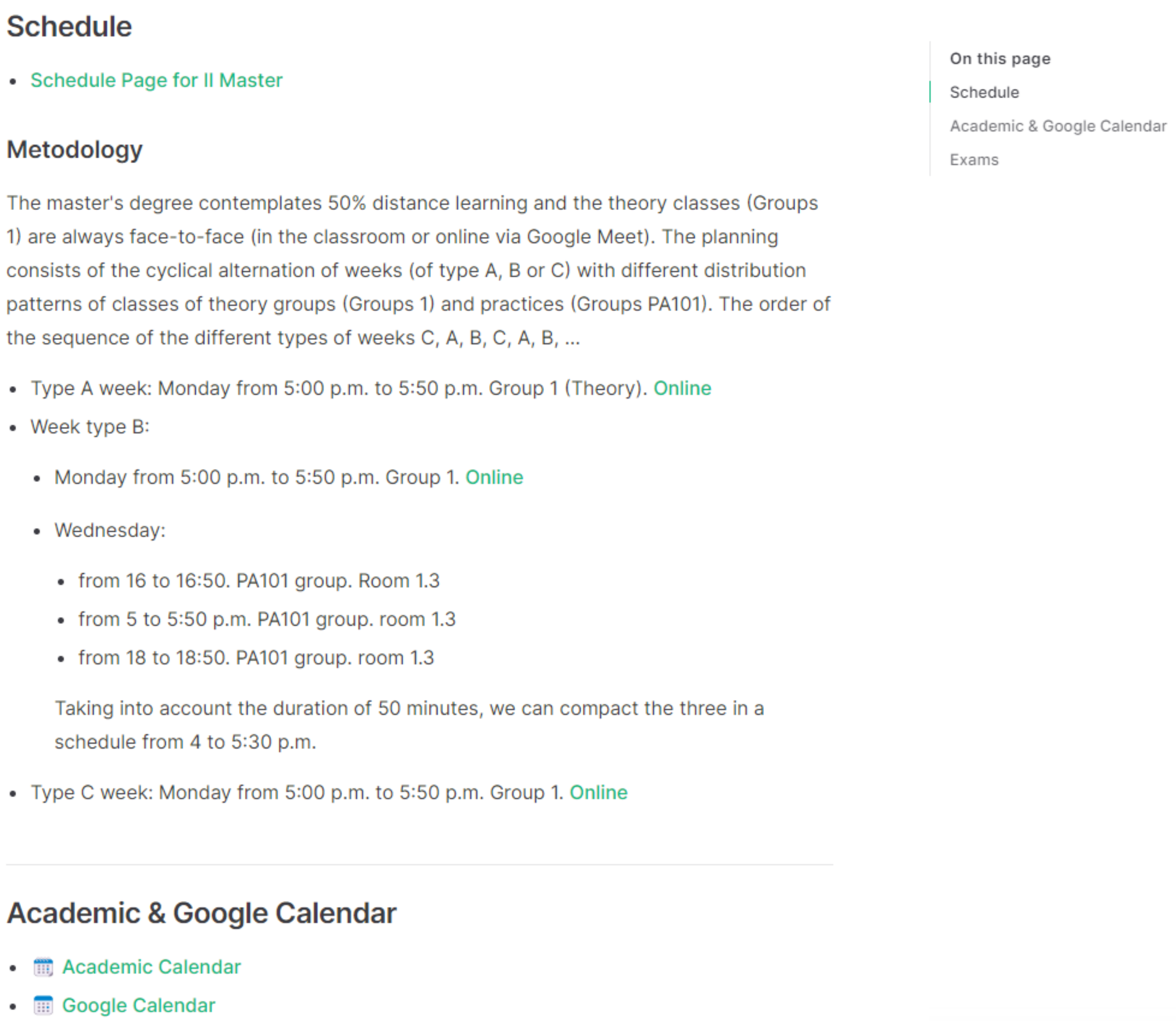
Here you can have the diferent teams of your Github organization with the most important information of each team and student, like recent repositories, summary, git profile,...

Figure 4.6: Home Page

3. Lessons, units and tasks: As is it a system that aims to be useful in a education environment, it is important to have all the content ready to be used. Remember that the lessons, units and tasks are displayed in the component *General* so the files with the information must have similar structure. First of all, lessons, tasks and units are written in markdown (file passed to the General component), having weeks in order to have an schedule for students and in each week having x lessons. Then, the lessons need the user to be authenticated. This is because the teacher can hide some content because it can be draft or not be ready to be shown. So the users with the role teacher can see this hide content but the students not. In the public folder, the teachers can set hidden true or false (its the same to not write the option). So the structure will be similar as the shown in the Figure 4.8 for lessons, Figure 4.9 fot tasks and the Figure 4.10 for the units.

4. Teams: Another important thing is to have the teams available and to be useful for the teacher. In the Figure 4.11, first it will be displayed an small summary of the teams with links like: User GitHub profile, repositories and notifications. Thanks to this, it makes easier to check information about each team as recent commits or projects that is working in. Then if the card is clicked, then the team view will be displayed. This view is to have more information about that team, for example the recent repositories that has been working with (all of them with the link to the repositories, issues and notification) making the correction task for the teachers easier, as it is shown in the Figure 4.12.

5. Login: The first approach was to add the login using the navigation bar, when the login was clicked to display user information, but apparently now that feature is not


Schedule

- Schedule Page for II Master

Metodology

The master's degree contemplates 50% distance learning and the theory classes (Groups 1) are always face-to-face (in the classroom or online via Google Meet). The planning consists of the cyclical alternation of weeks (of type A, B or C) with different distribution patterns of classes of theory groups (Groups 1) and practices (Groups PA101). The order of the sequence of the different types of weeks C, A, B, C, A, B, ...

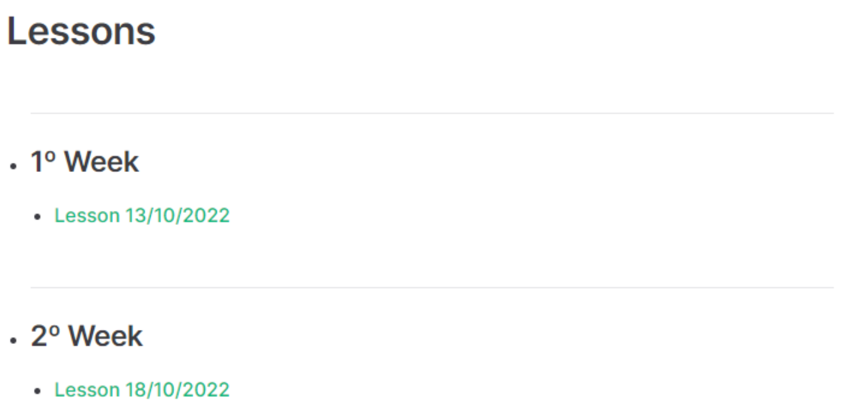
- Type A week: Monday from 5:00 p.m. to 5:50 p.m. Group 1 (Theory). Online
- Week type B:
  - Monday from 5:00 p.m. to 5:50 p.m. Group 1. Online
  - Wednesday:
    - from 16 to 16:50. PA101 group. Room 1.3
    - from 5 to 5:50 p.m. PA101 group. room 1.3
    - from 18 to 18:50. PA101 group. room 1.3

    Taking into account the duration of 50 minutes, we can compact the three in a schedule from 4 to 5:30 p.m.
- Type C week: Monday from 5:00 p.m. to 5:50 p.m. Group 1. Online

Academic & Google Calendar

- Academic Calendar
- Google Calendar

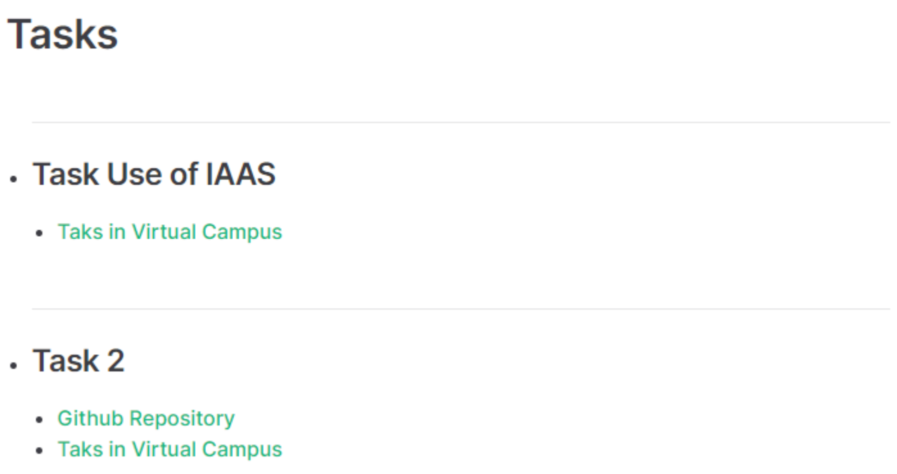
Figure 4.7: Schedule


Lessons

- 1º Week
  - Lesson 13/10/2022
- 2º Week
  - Lesson 18/10/2022

Figure 4.8: Lessons - Student view

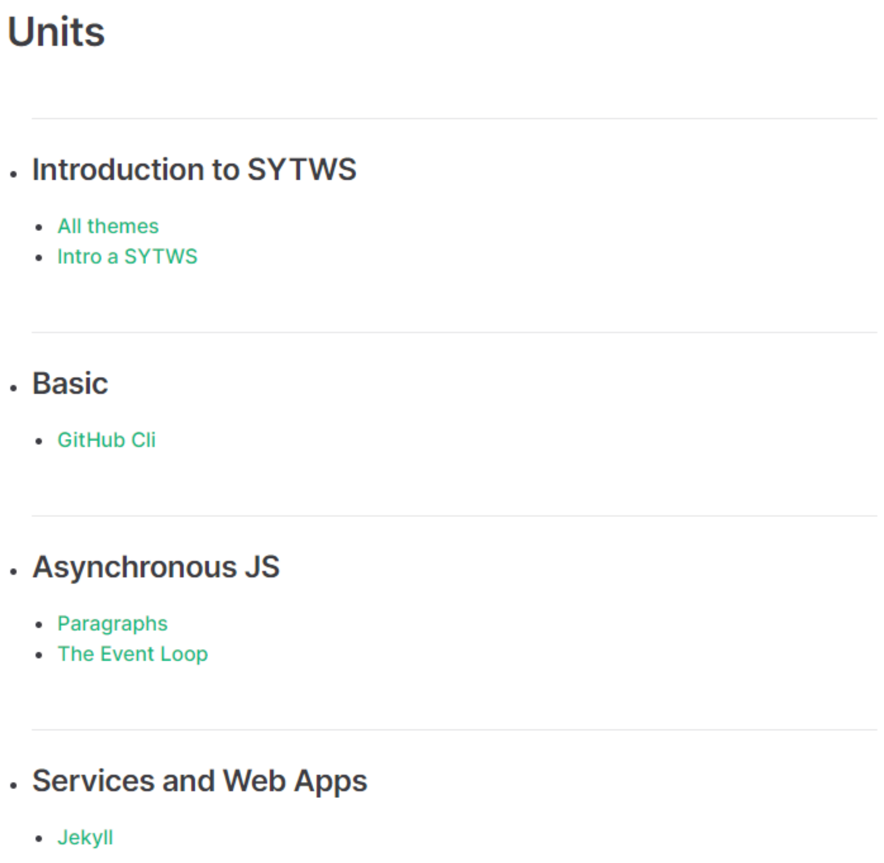
available, in the navigation bar only links are allowed. So in this case, the view has a button where the user can authenticate with GitHub, Figure 4.13 and then when the user is logged the page will display the role and name of the current user logged plus the logout button as shown in the Figure 4.14.



Tasks

- Task Use of IAAS
  - Taks in Virtual Campus
- Task 2
  - Github Repository
  - Taks in Virtual Campus

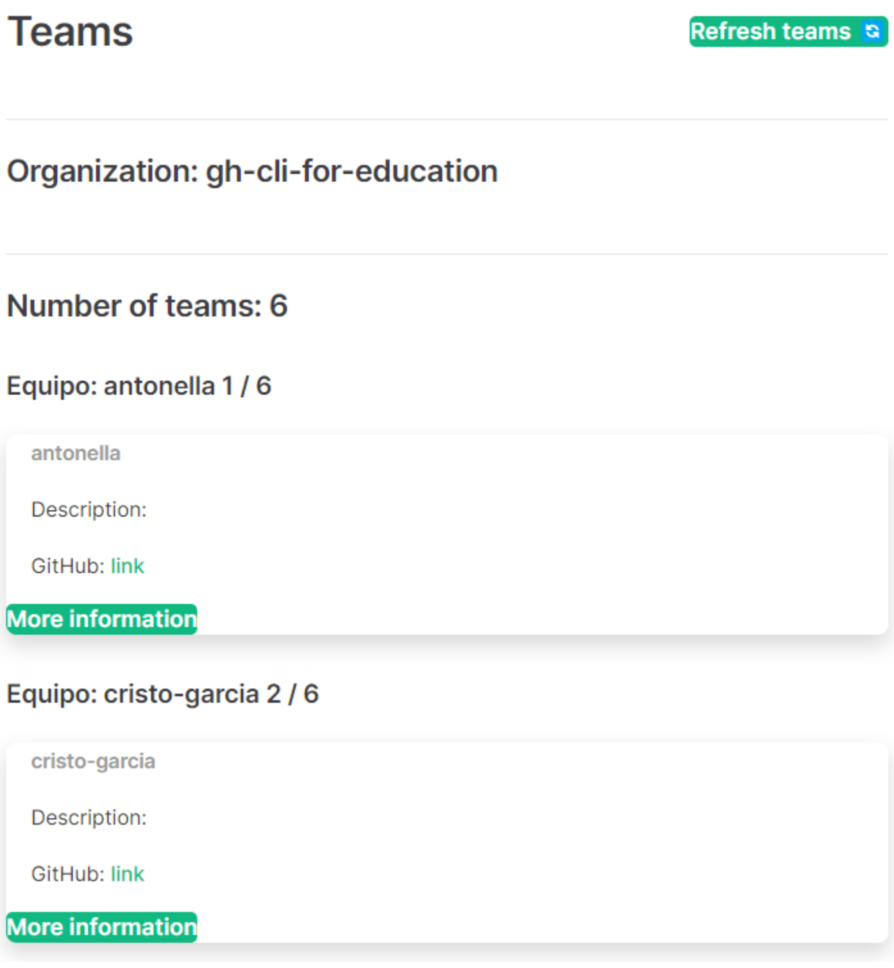
Figure 4.9: Tasks - Student view



Units

- Introduction to SYTWS
  - All themes
  - Intro a SYTWS
- Basic
  - GitHub Cli
- Asynchronous JS
  - Paragraphs
  - The Event Loop
- Services and Web Apps
  - Jekyll

Figure 4.10: Units - Student view


Teams

Organization: gh-cli-for-education

Number of teams: 6

Equipo: antonella 1 / 6

antonella

Description:

GitHub: link

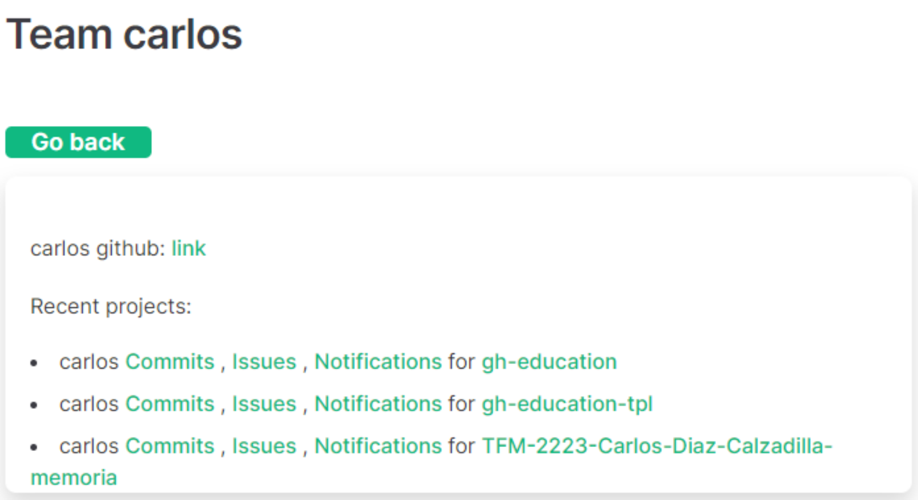
Equipo: cristo-garcia 2 / 6

cristo-garcia

Description:

GitHub: link

Figure 4.11: Teams of an organization


Team carlos

carlos github: link

Recent projects:

- carlos Commits , Issues , Notifications for gh-education
- carlos Commits , Issues , Notifications for gh-education-tpl
- carlos Commits , Issues , Notifications for TFM-2223-Carlos-Diaz-Calzadilla-memoria

Figure 4.12: Specific team information

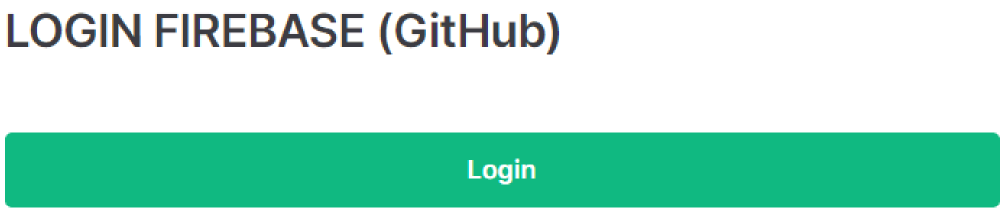


Figure 4.13: Login view

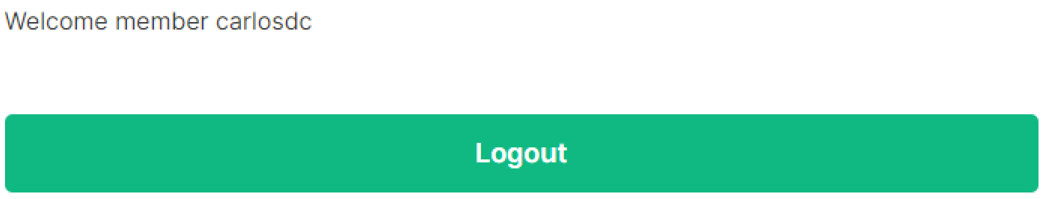


Figure 4.14: User logged view

Chapter 5

Quick guide

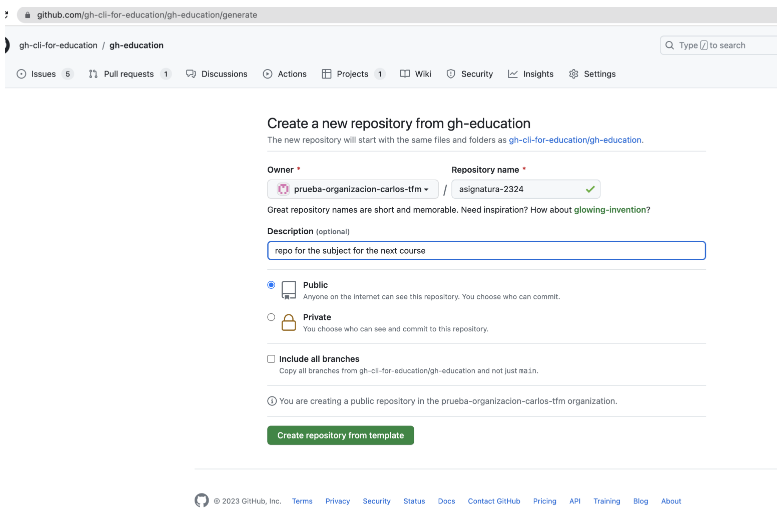
Once the system has been explained, the next step is to make a quick guide of how to use and get all the potential of this tool. As it was said in the other chapters, there are two ways to use the system.

5.1 Download GitHub template

There are some things to keep in mind before cloning the repository. First for all, it's important to create a GitHub Organization.When the organization is created, the next step is to create a repository forking the GitHub repository. Since the repository is a template, it allows to create a repository using this template, as shown in Figure 5.1.

Once the new repository is created Figure 5.2, the next step is to clone it. After this, the next step is to create the Firebase project needed to link with the system. One way to do it is issuing the following commands:

```
$ git clone --depth 1 <REPOSITORY CREATED USING TEMPLATE>
$ cd gh-education/vite-education/docs/
$ npm i
$ npm run docs:dev
```



Create a new repository from gh-education

The new repository will start with the same files and folders as gh-cli-for-education/gh-education.

Owner * prueba-organizacion-carlos-tfm / Repository name * asignatura-2324

Great repository names are short and memorable. Need inspiration? How about glowing-invention?

Description (optional): repo for the subject for the next course

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

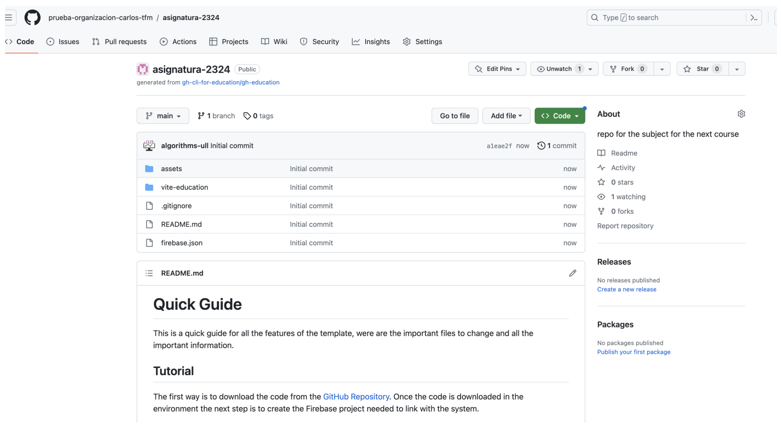
Private: You choose who can see and commit to this repository.

Include all branches: Copy all branches from gh-cli-for-education/gh-education and not just `main`.

You are creating a public repository in the prueba-organizacion-carlos-tfm organization.

Create repository from template
© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

Figure 5.1: Template repository use



prueba-organizacion-carlos-tfm / asignatura-2324

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

asignatura-2324 Public
generated from gh-cli-for-education/gh-education

Edit Pins · Unwatch 1 · Fork 0 · Star 0

main · 1 branch · 0 tags · Go to file · Add file · Code

algorithms-ull Initial commit	a1eae2f now	1 commit
assets	Initial commit	now
vite-education	Initial commit	now
.gitignore	Initial commit	now
README.md	Initial commit	now
firebase.json	Initial commit	now

About
repo for the subject for the next course
Readme · Activity · 0 stars · 1 watching · 0 forks · Report repository

Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

README.md

Quick Guide

This is a quick guide for all the features of the template, were are the important files to change and all the important information.

Tutorial

The first way is to download the code from the GitHub Repository. Once the code is downloaded in the environment the next step is to create the Firebase project needed to link with the system.

Figure 5.2: Template repository created

5.2 Firebase project

In order to create the first project, lets go to the Firebase console web where, when the user is logged in, then he / she will be able to create a project. As it is shown in the Figure 5.3, the first step is to add a name to the project (that will be identified with an ID) and set the domain that project will be lined with.

Then, for this case, there is no need to toggle on the analytic option (it's used to track user activity inside the application). Finally, the project will be created in a few minutes.

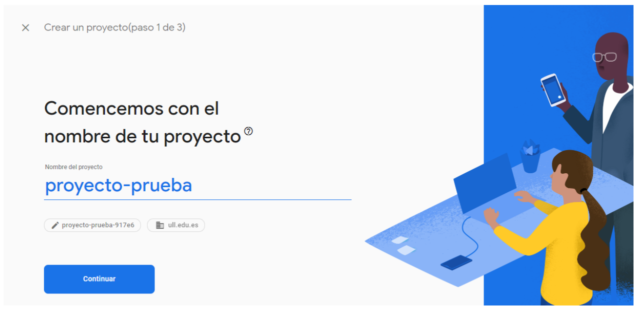


Figure 5.3: Firebase project creation

Once the project is created and the user presses the ready button, the page will be redirected to the homepage of that project, similar to Figure 5.4.

First of all, let's add an app to link with the one that is developed in the template. To do this, in the homepage, there are five icons. Each of them refers to a type of application. In this case the application developed is web so we need to click in the icon **<\>**.

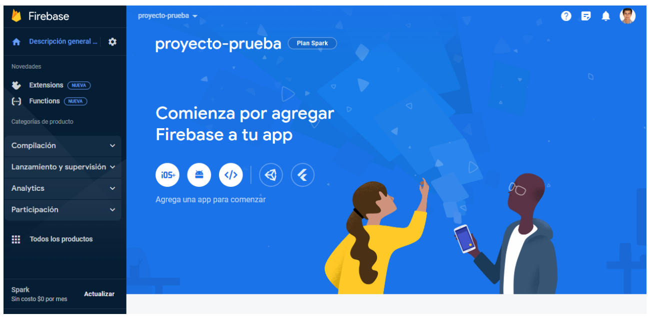
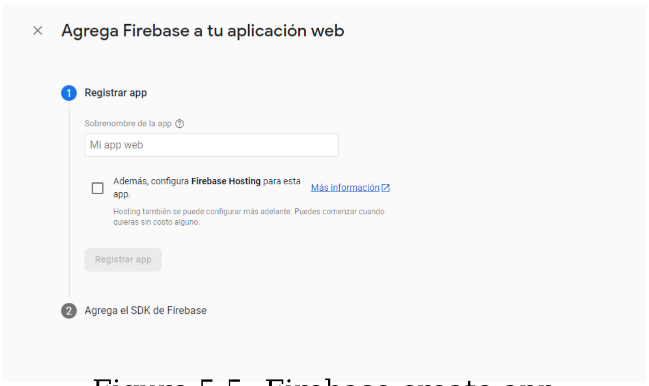


Figure 5.4: Firebase project home page

Now it's time to add the first information about our system. As we can see in the Figure 5.5, the app needs a name to be registered. It only accepts names that haven't been used for other projects. There is no need to activate the Hosting checkbox, in this example, because the project will be deployed in GitHub pages.

Once the application is created, then the Firebase SDK configuration will be shown. As the installation of Firebase has been done in the first step, there is no need to install it again.



Agrega Firebase a tu aplicación web

1 Registrar app

Sobrenombre de la app

Mi app web

Además, configura **Firebase Hosting** para esta app. Más información

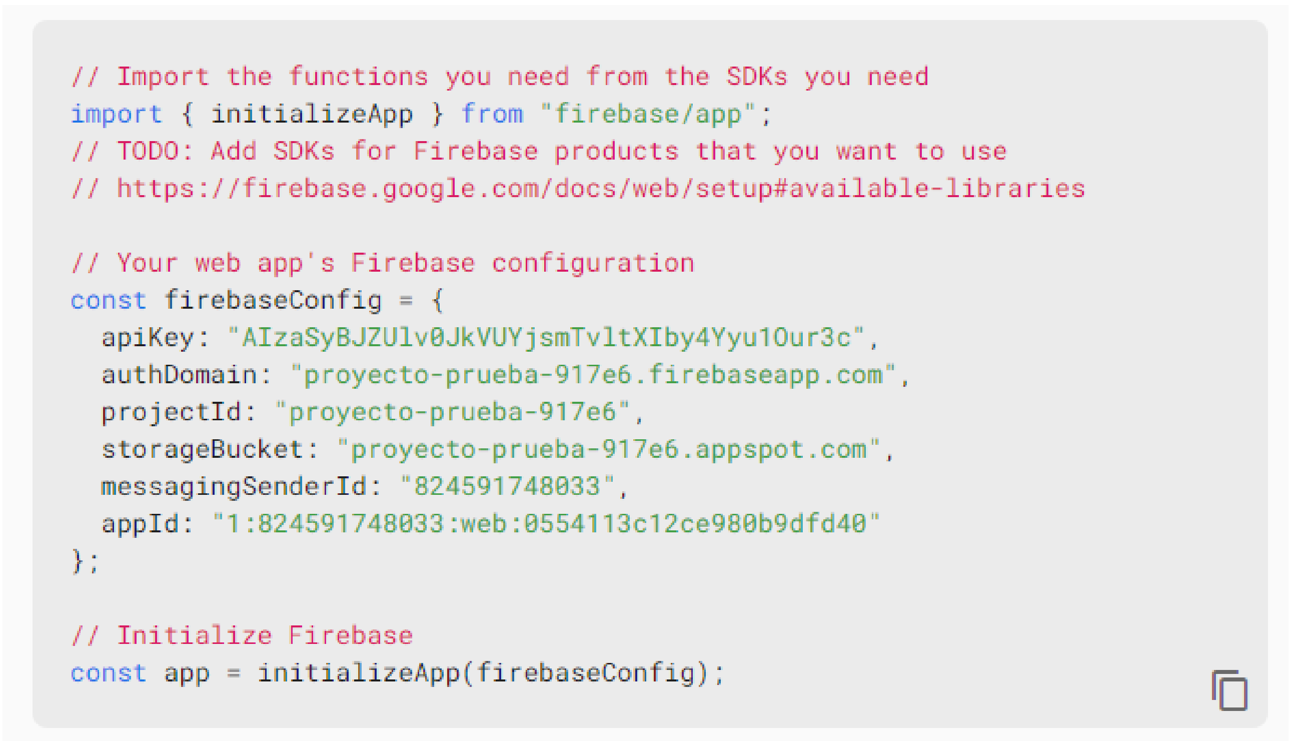
Hosting también se puede configurar más adelante. Puedes comenzar cuando quieras sin costo alguno.

Registrar app

2 Agrega el SDK de Firebase

Figure 5.5: Firebase create app

It will be similar to Figure 5.6. This information is important because it contains the credentials for the system to link with this application. So, once we have them, we will copy the *firebaseConfig* object content and replace it in the file `<YOUR-PATH>/vite-education/docs/.vitepress/firebase.js` with the content generated. This code will be also available in the configuration of the project so it won't be lost.



```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyBJZUlv0JkVUYjsmTvltXIby4Yyu1Our3c",
  authDomain: "proyecto-prueba-917e6.firebaseapp.com",
  projectId: "proyecto-prueba-917e6",
  storageBucket: "proyecto-prueba-917e6.appspot.com",
  messagingSenderId: "824591748033",
  appId: "1:824591748033:web:0554113c12ce980b9dfd40"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

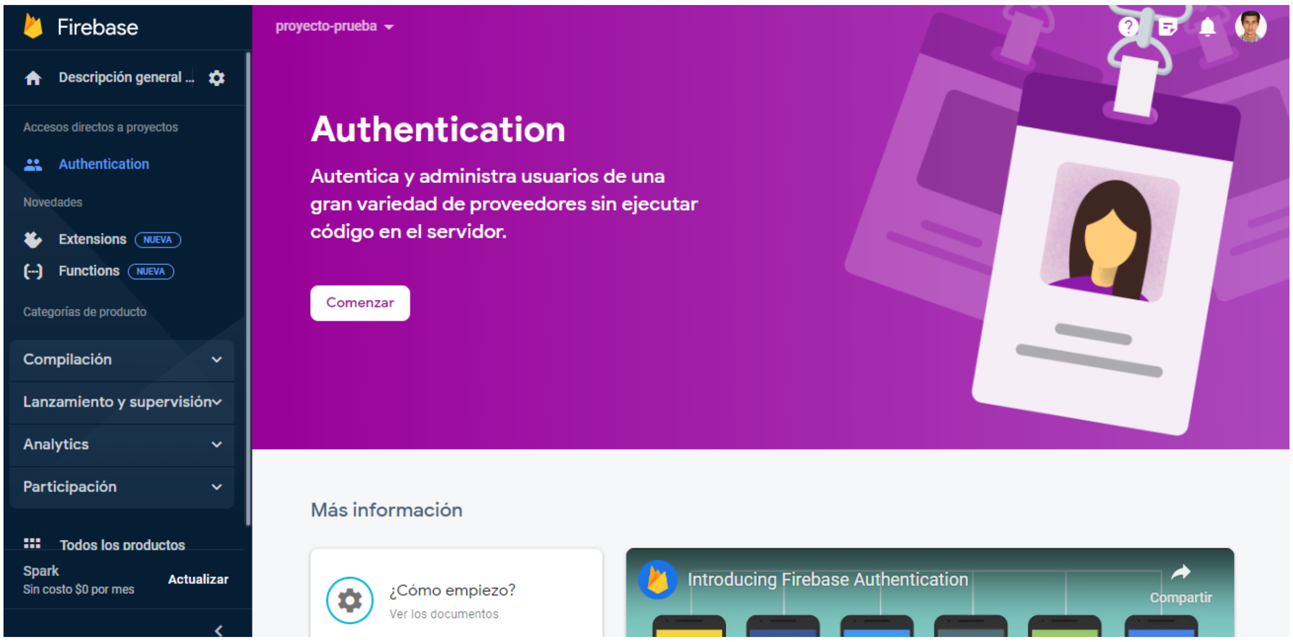
Figure 5.6: Firebase app SDK

Now the project is linked with the app, the next step is to select what type of authentication will be available for the user. To choose this, let's go to All the products section (in the project home page) and select `Authentication`. When it is selected, it will be available one view similar to the Figure 5.7 and when we press the begin button all the available sign in methods will be displayed.

5.3 GitHub authentication

Then, as we said in Chapter 4, the authentication in the system is through GitHub. This is because, getting the token of the user in GitHub makes easier to do some queries to get organization information. Then, when we select that provider, as shown in the Figure 5.8, it asks to link with GitHub using an ID and a secret.

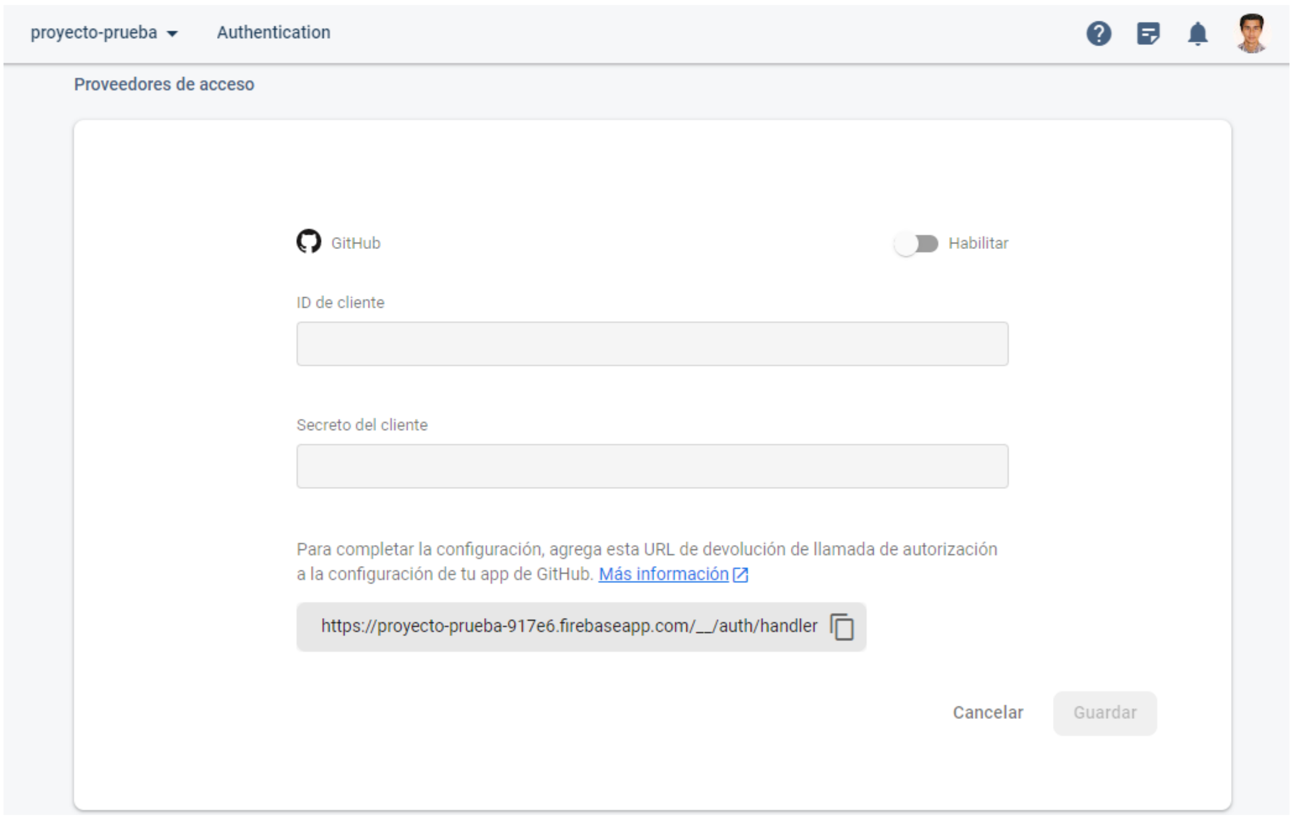
To obtain these two elements let's move to our GitHub account. First we will go to settings, then in the side navigation bar, in the integration section, we will click in



Authentication

Autentica y administra usuarios de una gran variedad de proveedores sin ejecutar código en el servidor.

Figure 5.7: Firebase Authentication



ID de cliente

Secreto del cliente

Para completar la configuración, agrega esta URL de devolución de llamada de autorización a la configuración de tu app de GitHub. Más información

https://proyecto-prueba-917e6.firebaseapp.com/__/auth/handler

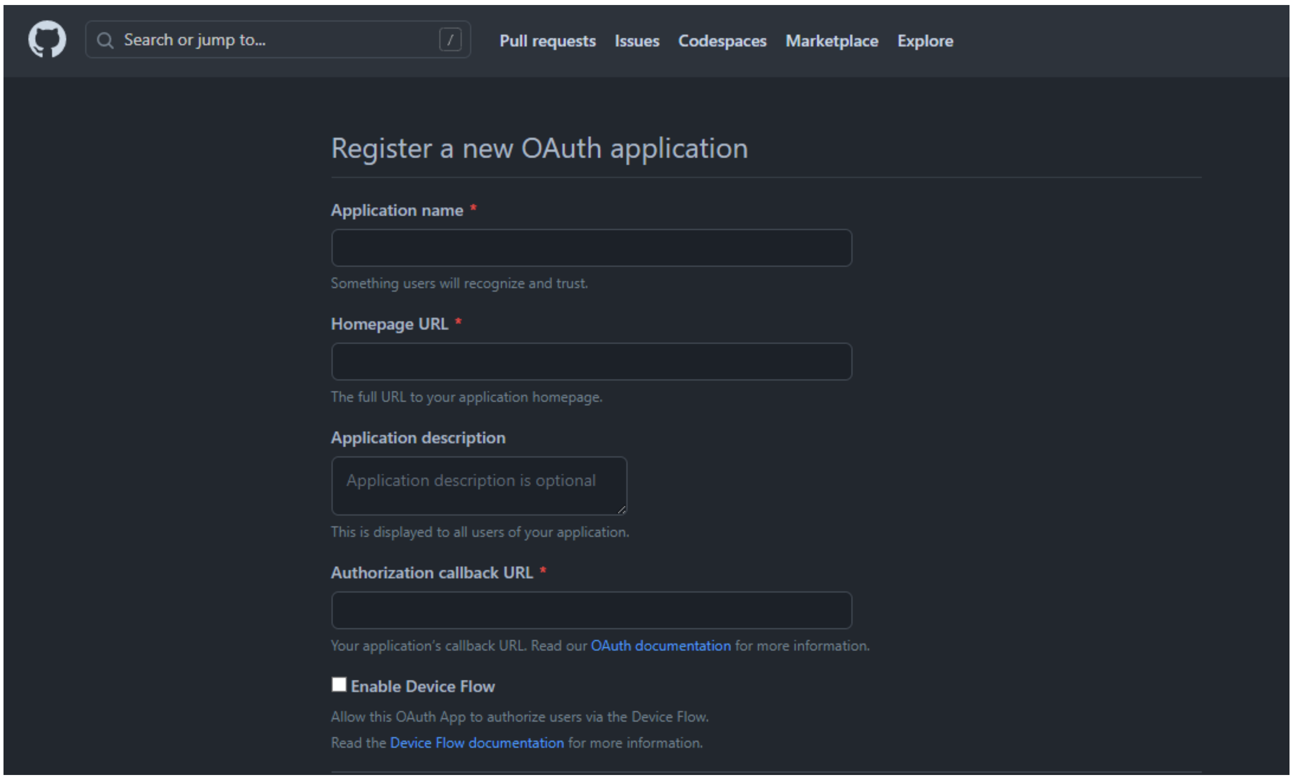
Figure 5.8: Firebase Authentication GitHub provider

applications. Here, there are three types of applications, but the one that we are going to work with are the **Authorized OAuth Apps**. To create an application let's move to the Authorized OAuth Apps and click new or follow this link. Then, this page must be similar to the picture shown in the Figure 5.9. Fill all the information:

1. Application name: Adding the name of the GitHub repository is recommended.

2. Homepage URL: Add README.md url of the repository.

3. Authorization callback URL: Copy the callback URL that Firebase displayed in Figure 5.8.


Search or jump to...

Pull requests Issues Codespaces Marketplace Explore

Register a new OAuth application

Application name *

Something users will recognize and trust.

Homepage URL *

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

Your application's callback URL. Read our OAuth documentation for more information.

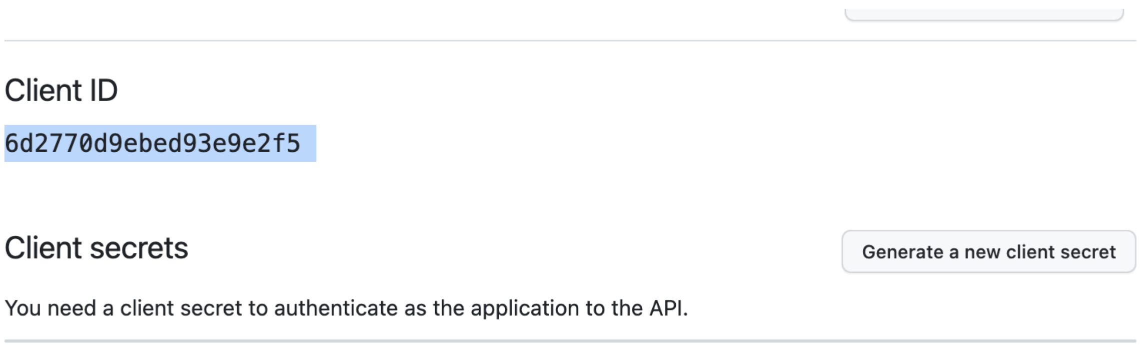
Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.

Read the Device Flow documentation for more information.

Figure 5.9: Firebase GitHub App

Once the application is created (Figure 5.11), it will generate a new client secret by clicking in the `generate a new secret button`.


Client ID

6d2770d9ebed93e9e2f5

Client secrets

Generate a new client secret

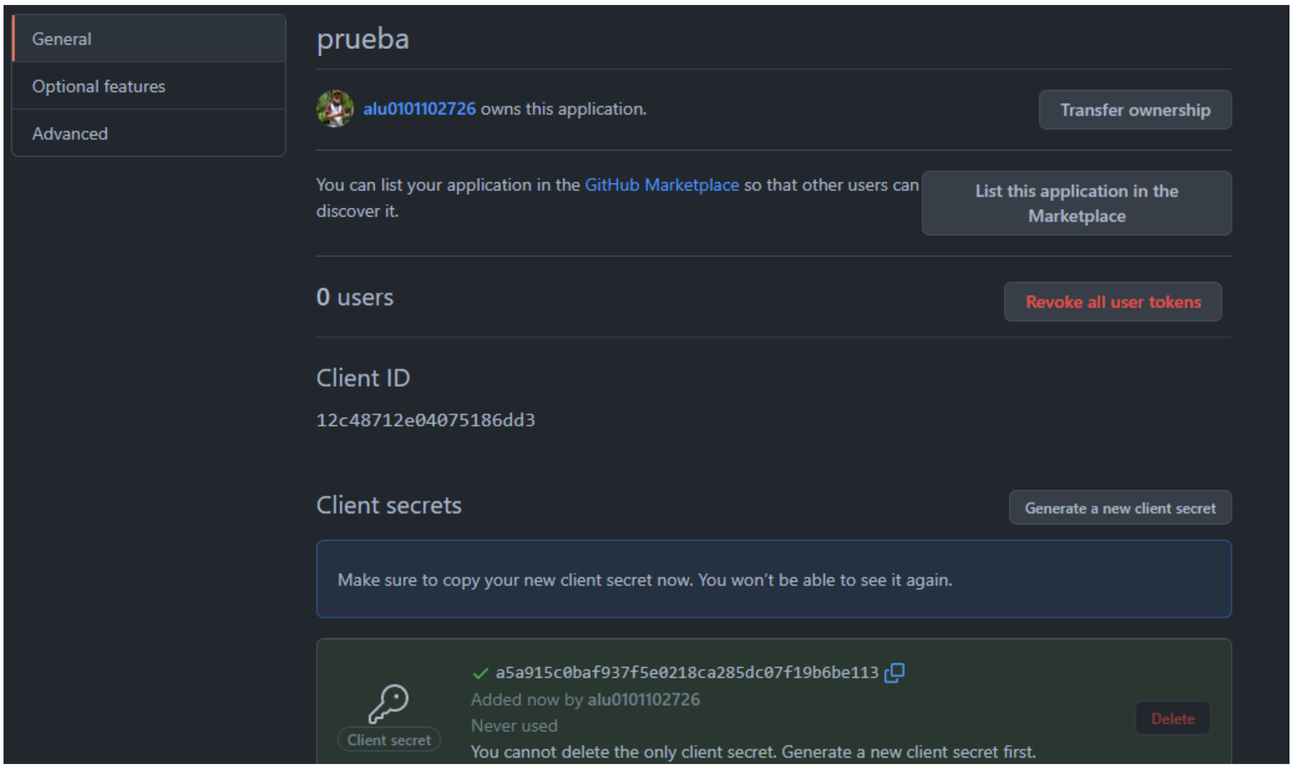
You need a client secret to authenticate as the application to the API.

Figure 5.10: Firebase GitHub App

The secret will be shown only once, so copy it before refreshing the page. Copy the client ID with the token, back to the Firebase form. Now with all this information filled, the application is ready to be used.

5.4 Create environment

With all the Firebase configuration done, the next step is to set the environment and install all the packages. To do this, follow the guide shown in the GitHub template. In the



prueba

alu0101102726 owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

0 users

Client ID

12c48712e04075186dd3

Client secrets

Make sure to copy your new client secret now. You won't be able to see it again.

✓ a5a915c0baf937f5e0218ca285dc07f19b6be113
Added now by alu0101102726
Never used
You cannot delete the only client secret. Generate a new client secret first.

Figure 5.11: Firebase GitHub App Tokens

project root there is a file called *.env* that uses environment variables to feed the system. So it's important to change that file modifying the variable *VITE_ORGANIZATION* with the GitHub organization and *VITE_REPOSITORY* with the GitHub repository. Then, there are only three steps to run the application:

1. Install dependencies: To install the dependencies run:

```
npm install
```

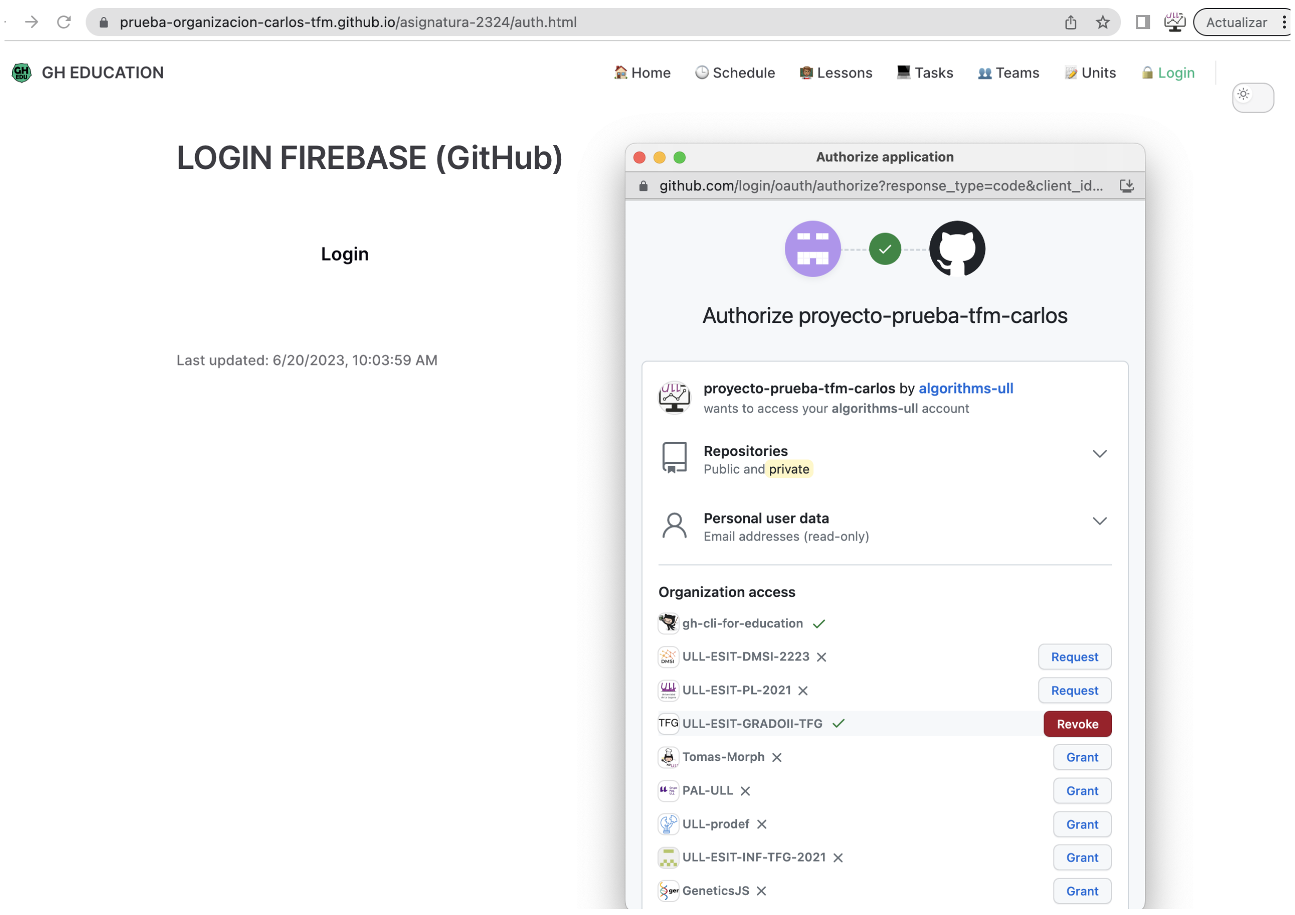
2. Run the project: The project can be tried in different modes:

```
npm run docs:dev      # Developer mode
npm run docs:build    # Build the project
npm run docs:serve    # Serve built project
```

5.5 Deployment

Finally if the user wants to deploy the content to GitHub pages, he / she needs to run the file called *deploy.sh*, located in the root of the project. In order to run this, the default branch used is main, but it can be changed editing the variable `\$VITE_BRANCH` in `.env` file. To deploy this content run:

```
chmod a+x ./deploy.sh
./deploy.sh
```


prueba-organizacion-carlos-tfm.github.io/asignatura-2324/auth.html

LOGIN FIREBASE (GitHub)

Login

Last updated: 6/20/2023, 10:03:59 AM

Authorize application

github.com/login/oauth/authorize?response_type=code&client_id...

Authorize proyecto-prueba-tfm-carlos

proyecto-prueba-tfm-carlos by **algorithms-ull**
wants to access your **algorithms-ull** account

Repositories
Public and private

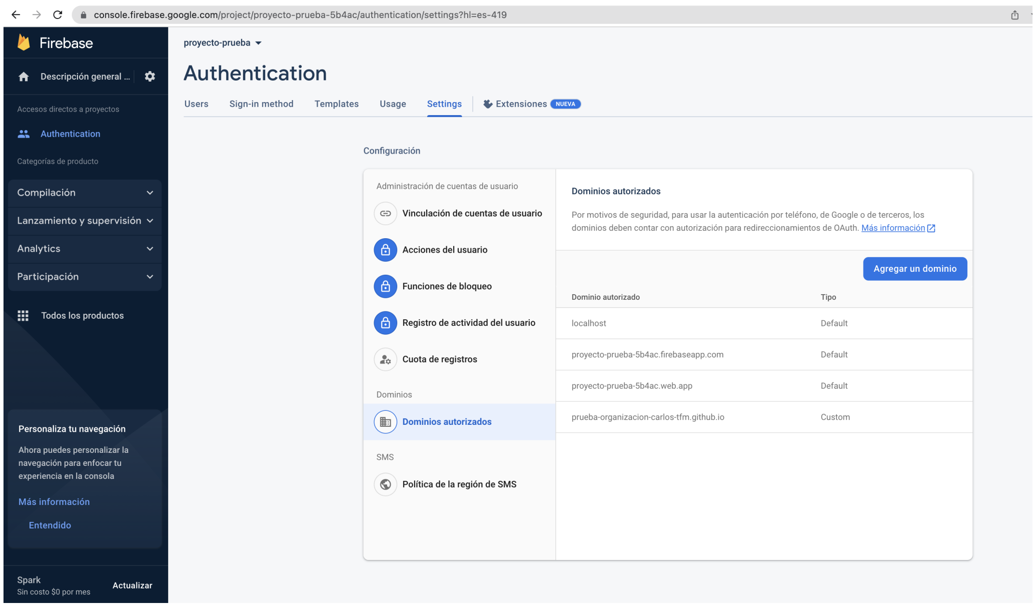
Personal user data
Email addresses (read-only)

Organization access

gh-cli-for-education ✓	
ULL-ESIT-DMSI-2223 ✗	Request
ULL-ESIT-PL-2021 ✗	Request
ULL-ESIT-GRADOII-TFG ✓	Revoke
Tomas-Morph ✗	Grant
PAL-ULL ✗	Grant
ULL-prodef ✗	Grant
ULL-ESIT-INF-TFG-2021 ✗	Grant
GeneticsJS ✗	Grant

Figure 5.12: Authorising GitHub App

Once the project is deployed, it's important to allow GitHub pages domain in Firebase. In order to do this, in the Firebase console, go to Authentication, then settings and in the Domain option, select Authorised domains. Once we are here, add a new domain copying the URL domain from GitHub pages, see Figure 5.13.


console.firebase.google.com/project/proyecto-prueba-5b4ac/authentication/settings?hl=es-419

Personaliza tu navegación

Ahora puedes personalizar la navegación para enfocar tu experiencia en la consola

Más información

Entendido

Spark
Sin costo $0 por mes

Actualizar

proyecto-prueba

Authentication

Configuración

Administración de cuentas de usuario
- Vinculación de cuentas de usuario
- Acciones del usuario
- Funciones de bloqueo
- Registro de actividad del usuario
- Cuota de registros

Dominios
- Dominios autorizados

SMS
- Política de la región de SMS

Dominios autorizados

Por motivos de seguridad, para usar la autenticación por teléfono, de Google o de terceros, los dominios deben contar con autorización para redireccionamientos de OAuth. Más información

Agregar un dominio

Dominio autorizado	Tipo
localhost	Default
proyecto-prueba-5b4ac.firebaseapp.com	Default
proyecto-prueba-5b4ac.web.app	Default
prueba-organizacion-carlos-tfm.github.io	Custom

Figure 5.13: Allow GitHub pages domain in Firebase

5.6 Set up with the gh-education script

Other option to use the system is using a script developed specific to ease all the steps done in the previous section. First task to do is to download the GitHub script repository. Once this has been done, the next step is to install all the dependencies that are going to be used in this script doing:

```
cd <YOUR-PATH>
npm install
./gh-education-tpl
```

When it is running will display a similar menu to the one shown below will display:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                         *
*                    # AVAILABLE OPTIONS #                *
*                                                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                         *
*  [1]    Clone and config template                       *
*  [2]    Deploy                                          *
*  [3]    Refresh Teams                                   *
*  [4]    Exit                                            *
*                                                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

In this menu there are three different options. Beginning with the first option, the clone and configuration template, the repository is going to be cloned in the current directory. Meanwhile the repository is cloning, there is a progress bar to let the user know that it is doing the clone task. Also, if the repository has been already cloned, then it will be deleted and replaced.

It shows the folder where the template will be downloaded and creates small configuration parameters such as the subject name, academic year and the GitHub organization to be used in the template. We can see the final results of this initial configuration shown below. In case of the information is not correct there is an small menu to show the configuration and add the information again.

```
# PROJECT INITIAL CONFIGURATION #
Subject name:  test
Academic year:  2023-2024
Github Organization:  gh-cli-for-education
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                         *
*                       # ALL CORRECT? #                  *
*                                                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                         *
*  [1]    yes                                             *
*  [2]    no                                              *
*                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Once the initial information is correct, the next step is to install the dependencies (like Firebase, Vuex, Vue,...) of the template, from the *package.json* of the project. The system will show all the packages that are being installed.

Now an important step comes in. When all the packages are installed, the next step is to log in the user into Firebase. If the user is logged the program will display the current user logged but, if not, the user need to follow an url in order to log in. Then, configure the Firebase project that we are going to use. In order to do this, a menu will be displayed:

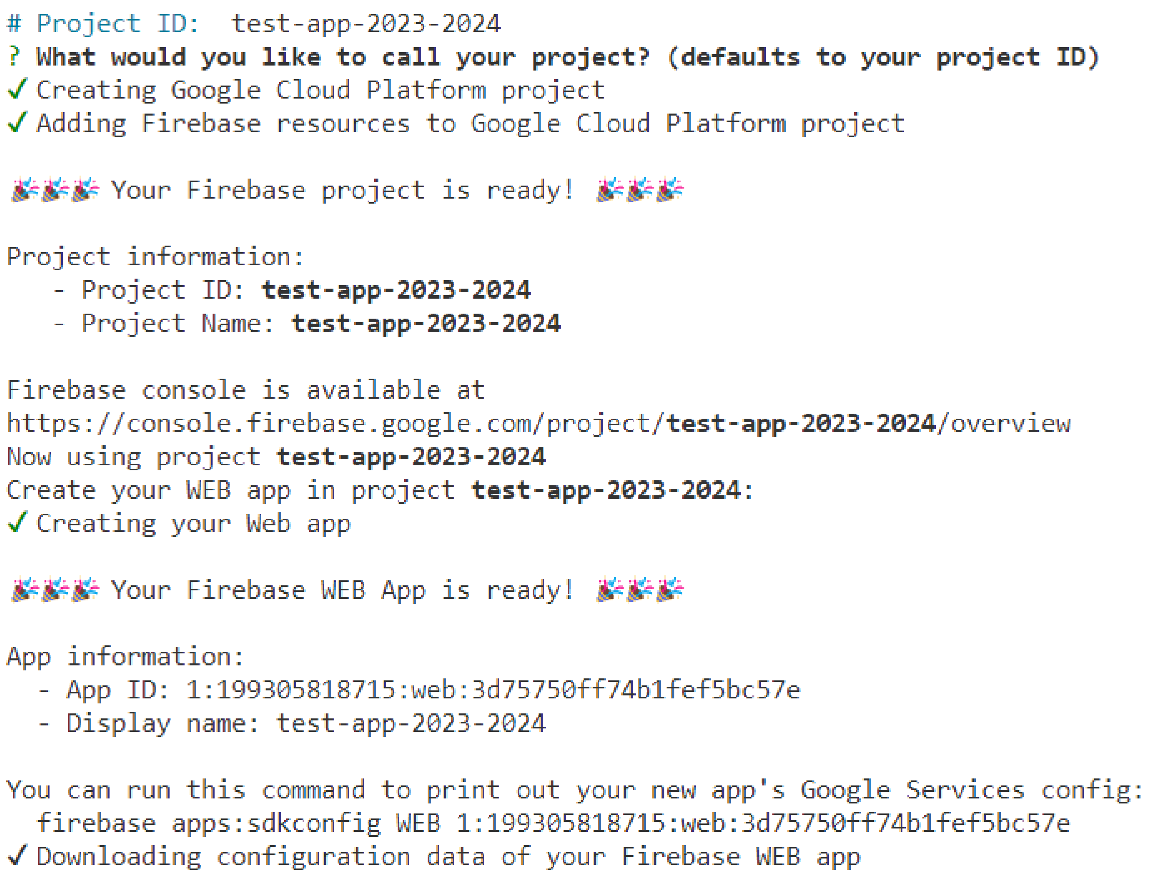
```
# FIREBASE LOGIN #
Already logged in as alu0101102726@ull.edu.es

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                             *
*                 # FIREBASE PROJECT CREATION #               *
*                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                             *
*  [1]    Create new project                                  *
*  [2]    List existing projects                              *
*  [3]    Change current project                              *
*  [4]    Exit                                                *
*                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

In this configuration the user is allowed to create Firebase project, adding a name for the project and the app (as we did in the template tutorial). The name of the project need to be a unique id, so it will use the name of the subject and the academic year. With this information, the project will be created as we can see in the Figure 5.14.

There are more features like list all the Firebase project related with the account logged as shown in the Figure 5.15 or change the project writing the id of the project to change.

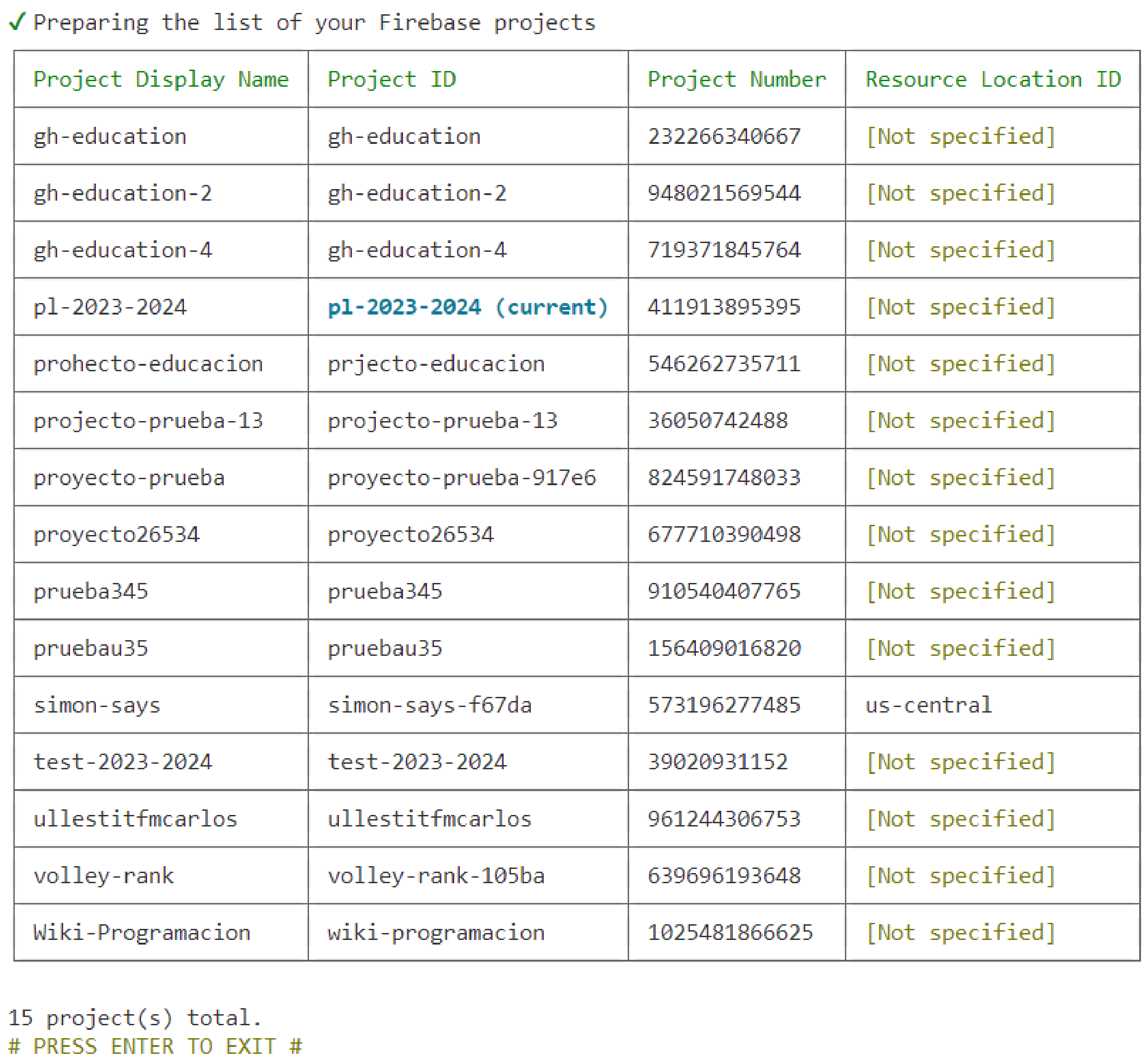
The final step for this Firebase configuration is to set the authentication of the project. When the project is created, we will save the URL of that project and navigate there. Then we will follow the steps as the template example in the Section 5.2, where we need to turn on GitHub authentication for our project and add the tokens.

Finally when all of this is changed, the Firebase project of the template will change to the one selected and the next step is to deploy, doing a build and push to an specific GitHub page. This option can be accessed directly from the initial menu.



```
# Project ID:  test-app-2023-2024
? What would you like to call your project? (defaults to your project ID)
✓ Creating Google Cloud Platform project
✓ Adding Firebase resources to Google Cloud Platform project

🎉🎉🎉 Your Firebase project is ready! 🎉🎉🎉

Project information:
   - Project ID: test-app-2023-2024
   - Project Name: test-app-2023-2024

Firebase console is available at
https://console.firebase.google.com/project/test-app-2023-2024/overview
Now using project test-app-2023-2024
Create your WEB app in project test-app-2023-2024:
✓ Creating your Web app

🎉🎉🎉 Your Firebase WEB App is ready! 🎉🎉🎉

App information:
  - App ID: 1:199305818715:web:3d75750ff74b1fef5bc57e
  - Display name: test-app-2023-2024

You can run this command to print out your new app's Google Services config:
  firebase apps:sdkconfig WEB 1:199305818715:web:3d75750ff74b1fef5bc57e
✓ Downloading configuration data of your Firebase WEB app
```

Figure 5.14: Script Firebase Project Creation



```
✓ Preparing the list of your Firebase projects
┌──────────────────────┬─────────────────────────┬────────────────┬──────────────────────┐
│ Project Display Name │ Project ID              │ Project Number │ Resource Location ID │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ gh-education         │ gh-education            │ 232266340667   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ gh-education-2       │ gh-education-2          │ 948021569544   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ gh-education-4       │ gh-education-4          │ 719371845764   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ pl-2023-2024         │ pl-2023-2024 (current)  │ 411913895395   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ prohecto-educacion   │ prjecto-educacion       │ 546262735711   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ projecto-prueba-13   │ projecto-prueba-13      │ 36050742488    │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ proyecto-prueba      │ proyecto-prueba-917e6   │ 824591748033   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ proyecto26534        │ proyecto26534           │ 677710390498   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ prueba345            │ prueba345               │ 910540407765   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ pruebau35            │ pruebau35               │ 156409016820   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ simon-says           │ simon-says-f67da        │ 573196277485   │ us-central           │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ test-2023-2024       │ test-2023-2024          │ 39020931152    │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ ullestitfmcarlos     │ ullestitfmcarlos        │ 961244306753   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ volley-rank          │ volley-rank-105ba       │ 639696193648   │ [Not specified]      │
├──────────────────────┼─────────────────────────┼────────────────┼──────────────────────┤
│ Wiki-Programacion    │ wiki-programacion       │ 1025481866625  │ [Not specified]      │
└──────────────────────┴─────────────────────────┴────────────────┴──────────────────────┘

15 project(s) total.
# PRESS ENTER TO EXIT #
```

Figure 5.15: Script Firebase List Projects

Chapter 6

Summary and conclusions

The development of this project has made me aware of the possibilities that static generators have, specifically, the case of Vitepress that combined with Vue technology makes the generated systems get very good results. In addition to this, a lot of use has been made of the GitHub API for queries in the organization. Likewise, this work has helped me to understand how the teaching environment works and most of the tools used to teach. The system is intended to be able to help teachers (more focused towards teachers in the field of computing) to have all the information more at hand, to be able to reuse and modify the template as desired as well.

Chapter 7

Resumen y conclusiones

El desarrollo de este proyecto me ha hecho ver las posibilidades que tienen los generadores estáticos, en concreto, el caso de Vitepres que combinado con la tecnología de Vue hace que los sistemas generados tengan muy buenos resultados. Además de esto, se le ha sacado mucho partido a la APÌ de GitHub para las consultas en la organización. Asimismo, este trabajo me ha ayudado a entender como funciona el entorno docente y la mayor parte de herramientas que se utilizan para impartir docencia. Con el sistema se persigue poder ayudar a docentes (más orientado a docentes en el campo de la informática) a tener toda la información más a mano, poder reutilizar y modificar la plantilla a gusto también.

Appendix A

Project repositories

The address of the GitHub repositories that contain the template developed, the script and the template, is attached to facilitate its use together with the tag of the last commit, which guarantees its status on a date prior to the delivery deadline of this work.

- Template repository
  - Latest commit: #349d46e40df3f8b505bf32eb6b284a1720b56db3
  - tag
  - release
- Script repository
  - Latest commit: #97b918fb3e9442008048794220acfe95b41231af
  - tag
  - release

Bibliography

[1] Calzadilla, C. D. Template github repository. https://github.com/gh-cli-for-education/gh-education.

[2] Firebase. Firebase guide. https://firebase.google.com/docs/guides.

[3] GitHub. Github graphql documentation. https://docs.github.com/es/graphql.

[4] HOWTO, B. P. Cursor movement. https://tldp.org/HOWTO/Bash-Prompt-HOWTO/x361.html.

[5] Inc., G. Github classroom. https://docs.github.com/en/education/manage-coursework-with-github-classroom, 2023.

[6] Jamstack. Jamstack home. https://jamstack.org.

[7] León, C. R. Curso sobre la integración de las herramientas 'github education' en el aula. https://campusvirtual.ull.es/ocw/course/view.php?id=136, 2022.

[8] León, C. R. Curso github en el aula. https://ull-ocw-github-education.github.io, 2023.

[9] VitePress. Vitepress guide. https://vitepress.vuejs.org/guide/what-is-vitepress.

[10] Vue. Vue guide. https://vuejs.org/guide/introduction.html.

[11] VuePress. Vuepress guide. https://v2.vuepress.vuejs.org/guide/.

[12] Vuex. Vuex guide. https://vuex.vuejs.org.

[13] Wikipedia. Learning management system. https://en.wikipedia.org/wiki/Learning_management_system.

[14] Zagalsky, A., Feliciano, J., Storey, M.-A., Zhao, Y., and Wang, W. The emergence of github as a collaborative platform for education. 1906–1917.