Hubbl, a gym bookings manager

Miquel de Domingo i Giralt

May 22, 2022

Some dedications to the people I care for.

Contents

List of Figures

List of Tables

1. Introduction, motivation, purpose and project goals

1.1 Introduction

On November 2019 the first cases of a new virus were coming into light. With similar symptomps as a regular flu, nobody would have imagined that the humanity would be at the edge of collapsing. The weeks went by and within few months, most countries of the world were completely shut down. It was not until the summer that people was able to go outside, again. Restaurants, hotels and any business that relied on the clients for their survival, had to face many restrictions imposed by the governors. One of such business where the gyms. The owners had to limit the total capacity of their installations and classes. Most companies had to find a new system which helped them handle appointents, since nearly any of them had an appointment system integrated in their respective applications.

1.2 Motivation

As a front-end engenieer and considering myself someone who really enjoys designing and later developing user interfices, I realised that I had a great oportunity in front of me in order to test myself. At my current job in Additio, a company which develops applications for the teachers, I have had the opportunity to start learning more about UI/UX. Nevertheless, I had to attach to creatain rules, in order to respect the design system of the application. Now, however, I had the opportunity of being responsible for each part of the process of developing an application: from designing the database and impementing the API, to designing the UI and implementing such design. Without constraints, I would have try and explore as many technologies and systems as I felt like.

Furthermore, as a regular user of my gym's booking application, I had few issues with it. Even though I have only experienced one side of the application, as a client, I believe that some parts could be improved, as in anything. One of such parts are both the user interface and the user experience of it.

1.3 Project goals

The goal of this final degree project is to develop a full stack application, which includes:

- Designing the database.
- Implementing an API in order to interact with the database.
- Briefly designing the user interface.
- Implement the required front end applications, web based, in order to access the system.

Fundamentally, the hubbl application will allow the creation of gym zones, which are explained in the next sections, which will allow the clients to create the needed appointments.

As a future implementation yet not being a priority, the system will also have:

- An analytics page which would provide information about the statistics of the gym.
- A subscription system for the clients.

2. Viability

The study of the viability of the project will take into account from the technical resources to the echonomical cost of the project.

2.1 Techical resources

Since the application is a web-based application, it only requires a computer. Such computer should have installed all the required software, which is covered in further sections, so that the development of the project can be done without running into any issue.

2.2 Echonomical costs

The echonomical costs of the projects will only take into account the amount of hours the developer and designer have needed to develop the application, since the hardware requirements are already owned by the developer. However, an application can never be considered as finished, since there will always be bugs, new features and client requests that will keep improving the software. Therefore, in order to evaluate the total cost of the project, the estimated hours required both to design and develop the application will be multiplied by a fixed €/hr quantity[1].

In Spain, where the project is being developed, JavaScript developers earn around 13.97€/hour (≈ 14.00€/hour). For the ui designers, the average is around 15.38€/hour (≈ 15.50€/hour). Therefore, the total costs of the project can be described in the following table

Process	**Hours**	**Price**	**Total**
Design	96h	15.50€/hour [1]	1488€
Development	629h	13.97€/hour [2]	8806€
Total	725h	-	10294€

Table 2.1: Total costs table

[1] A more specific description of the tasks with their time can be found in the Working packages section

3. Methodology

3.1 Introduction

In this chapter, it will be briefly explained what project management will be utilised and the development methodology. Both decisions are crucial and will have effect in how the project is planned and evolves.

3.2 Project management: *PMBOK*

The methodology used to develop the overall project has been the *PMBOK* which has been explained in one of the subjects of the degree. Such methodology ha become a standard in the project management world. Since it is considered as the book of books when it comes to project management, it is definitely useful to be used in any project, which, most likely, will require some sort of management.

The PMBOK methodology is explained in the *Project Management Body of Knowledge* book, in which such standards and guidelines are explained more in depth. The book is produced and updated by the *Project Management Institute* (*PMI*) and it currently has 6 editions, the latest one released in 2017.

The procedure is based in five process groups, which are:

1. **Initiating**: the initiating processes are those which are performed in order to define the project or an upcoming phase of an existing project, and obtain permission to execute the project or phase.

2. **Planning**: the planning processes are those which are required to clearly define the scope, the objectives and the course of action of the project.

3. **Execution**: the planning processes are those which are performed to complete the work that has been defined in the previous processes in order to satisfy the project specifications.

4. **Monitoring and controlling**: the processes involved in this group, are required to track, regulate and review the progress and performance of the project. It identifies areas in which the plan has to change and initiates the corresponding changes.

5. **Closing**: the closing processes are those which are performed in order to finalize all activities across the above process groups so the project or phase can be closed.

3.3 Development methodology

To structure and organise the project, working packages[1] have been used in order to ensure a temporised and structured development. Following such structure

[1] See Working packages for more information.

has been extremely useful to simplify how are tasks managed and temporized. In order to organise the tasks, an *AGILE* approach has been used, even though the project has been done all by myself. Each subgroup of the *development* branch has been considered as an *epic* and each working package as a *story*, which has as many tasks as required. This approach has been used for each project and has made the development of the project easier to manage.

In order to simplify the process of *epic*, *story* and *task* creation, the Jira software from Atlassian has been used. Jira is a project management software developed by Atlassian which helps from teams to single users to easily manage their projects. Even though it is oriented to AGILE methodologies, it can be used by any type of project [3]. It has been extremely useful as it can be integrated with many tools, one being GitHub, where the code is hosted. The GitHub integration provides a lot of utilities, yet one of the most important is the fact that it tracks your branches, commits and pull requests. Using the three elements, I have been able to automate the process of starting and completing tasks, alongside of the Jira automation. The following diagram, describes the workflow of the tasks inside Jira:

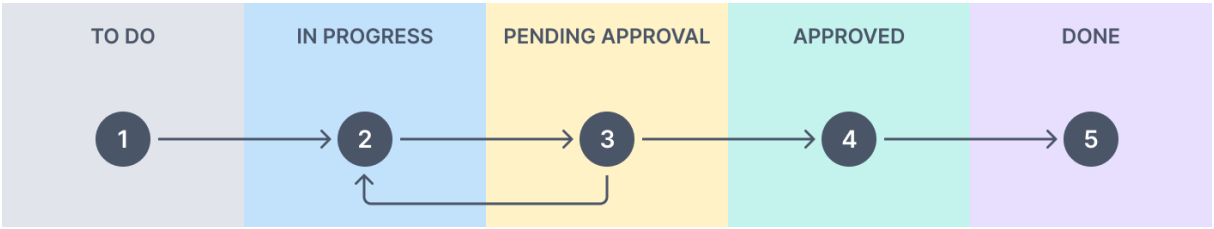


Figure 3.1: 5 steps of the Jira workflow

All the tasks would have to go from step 1 to step 5 in order to be considered as finished. Each step and the automation is as follows:

1. All the tasks that have been selected for the sprint start at the first step, at the *to-do* column.

2. For each task, a new git branch has to be created with the name of the Jira task tag, for instance `HBL-185`[2].
   When the branch is created and pushed to GitHub, Jira will move the task automatically to the *in progress* column. [4]

3. While in progress, each commit made is tracked in Jira. Additionally, Jira also supports *smart commits* which are words prefixed with a hash (#). With the smart commits, task properties can be changed. In this case, only the `#time` has been used which increments the amount of time spent in each task. Tracking time has been useful to know more precisely how much time has been spent overall.

4. Once the task is finished, a new pull request is made which triggers the continuous integration workflows set up at the GitHub repository. These workflows can also be watched by Jira and automate operations in function

[2] The `HBL` prefix is assigned when a new Jira project is created.

of the workflow result. If the task does not pass one of the workflows, the task is moved back to the *in progress* tab. Alternatively, if the checks pass, it is moved to the *approved* column.

5. Finally, in order to be considered *done*, the pull request of the task in question has to be merged. The merge will trigger another automated process which moves the task from *approved* to *done*.

In order to stick to the process as much as possible, tasks moved to *done*, should not be moved back. Instead, a new *bug* type task was created which would start the process again.

4. Framework and previous concepts

4.1 Introduction

This chapter will cover the knowledge that is required in order to properly develop a booking application. In short terms, it is required to know how[1] gyms work, regarding class and room (or non-class) appointments. Furthermore, it is important to have some knowledge on software artchitecture and development, in order to provide value with the application.

4.2 Gym overview

There are few issues to address regarding the appointment system of a gym. In general, there are two types of appointments to be made. The first is the most simple and basic, which is an appointment made to a class or event organized by the gym. Such appointment will have a maximum capacity, for example, the gym may have a fixed number of spinning machines, and it would be nonsense to allow more appointments that spinning bikes are in a spinning class. The second appointment would be an appointment that is made to access the gym for an interval of time. Most gyms have zones in which their clients can access anytime, and exercise themselves using the mahcines or equipment they feel like using. Therefore, the gym has to provide their clients the possibility of accessing their infrastructures within the interval the client wants to work out.

4.3 Development overview

The application will be a web-based application, meaning that it is expected to be used within a browser, accessing a specific URL. Using a web environment completely avoids the native app implementation. One of the most common issues developers face is the fact that, in order to develop an app for different OS, they can rarely use a common code base. Each OS may require a different language or a different implementation, difficulting the addition of new elements or even maintaining the app itself.

Therefore, developing an application for the browser removes such issues. That itself does not solve all the problems, as once again there are multiple browsers, all with their different implementation. It appears a new difficulty for developers, who have to know if the application will be cross-browser functional. Nonetheless, most modern browsers have support for JavaScript, which simplifies the development process as the application can be built with the same language, independently of the browser.

JavaScript is far from being a perfect language, plus developing a complex application with vanilla JavaScript can be a tough task and very time demanding. Luck-

[1] It is obvious that the may be different implementations, for different gyms. However, the implementation that has been chosen is the one that I thought was most accurate, based on my experience in different gyms.

ily, there exist multiple frameworks which are production-ready and extremely simplify the process of developing applications. One of the most wellknown frameworks is React. React itself does not solve all issues, since, for example, it does not provide a browser routing solution, yet it is simple to understand. Once again, companies or even developers who saw what was React lacking, as the routing solution, started developing what is known as a meta-framework, which is a framework built upon another framework. This is the case of NextJS, a Javascript meta-framework (as it is a React framework), which tries to solve most of React flaws. There exist many more JavaScript frameworks or even other solutions such as WebAssembly, which allows running non-JavaScript applications in browsers. To sum up, a developer whose goal is to create an application for the browser should be comfortable with such concepts.

On the other side of the application, there is the persistence layer. The persistence layer is none other that the part of the application that allows the system to *persist* the data of the users. The user will interact with the system, and such interaction will, most likely, output a result. Such result should be stored in the form that the application requires it, as it will add value to the product (it is rarely difficult to find an application that does not have the need of storing any data). It appears a new field of knowledge which is how the data is stored and how such data is exchanged with the client application,

Nowadays, there are hundreds of solutions for such problems. Again, it is a matter of convenience when it comes to choosing the languages and databases which will allow the communication and persistence of the data. For the database, it relies on the application type. For example, if the application relies on fast queries, a ram-based database would be a good solution. On the other hand, an application that contains many tables, each with different relationships, such as the one developed in this project, an SQL database would be a better solution. From here, a comparison with the different SQL tools should be made, as each tool has different benefits and drawbacks. Next, the developer would need knowledge in server-client communication. Using a REST API is sometings the go-to solution, however, once again, the server architecture should be chosen taking into account the application requirements and needs. To sum up, the developer should be comfortable knowing different solutions and architectures, in order to find the solution that fits most the application.

5. Requirements specifications

5.1 Introduction

This chapter will cover the system requirements specification (*SRS*) for the software being developed. The goal of this chapter is to establish the basis for what will be developed, taking into account user needs, functional and non-functional requirements.

5.1.1 Purpose

The application's objective is to provide a simple, scalable and powerful platform to handle gymnasium appointments. Gym owners will be able to create their gym zones and modify the constraints such as total capacity, available hours, and so on. On the other side, gym clients will be able to book an appointment to the gyms they have access to.

Therefore, in a single application, gym owners and their workers will be able to control everything that happens inside the gym; while at the same time being able to apply changes and modifications as wanted.

5.1.2 Definitions

There are some definitions to be explained before moving to the next section:

- *Gym*. The gym will represent the company overall, not the infrastructure. Such infrastructure is called *virtual gym*.
- *Virtual gym*. A virtual gym will be the representation of a gymnasium in the application. A *virtual gym* can have multiple *gym zones*.
- *Gym zone*. A gym zone will be the places where clients will book appointments. A more specific definition for gym zones can be found in further sections.
- *User*. The *user* is the main person who uses the software. In this case, it is the owner and their workers.
- *Template events*. Even though the class concept is explained in further sections, it is important to make a distinction between a *class* and *class template*, *template events* or *template events* (as they are named in the database). On the one hand, events will be assigned to a gym zone and scheduled. On the other hand, *template events* will be used to schedule and link to a zone.
- *Archiving* vs *Deleting*. The purpose of archiving anything in the database means it will be stored but not visible. In order to edit it again, the archived element has to be *recovered*. On the other hand, a deleted element will permanently be deleted from the database.

5.2 Overall description

5.2.1 Introduction

This section provides a general explanation about the application, how it is intended to work, the different *personas* to whom it is focused and a brief introduction to the software's functionalities.

5.2.2 Product description

The product will provide a rich interface for the gym owners and their coworkers in order to manage their gym. Managers will create virtual gyms, each of one with different constrains, which will be forwarded to their gym zones. Each gym zone will be of a certain type, for example, a cardio zone, a free-weight zone or even a powerlifting zone. Such and more characteristics will be determined by the workers or who is responsible for the creation of the zones. Once determined the capabilities of each zone, the clients will have the possibility to make their reservations.

This software is going to be interesting since it is mainly focused on the managing of appointments. There exist multiple manager systems, some are gym-focused yet there are few that provide an exclusive focus to managing the appointments. It is interesting to have an application that is that specific because most of the companies already have their client management system and other tools. The COVID has changed many things extremely fast and some gymnasiums have not been able to adapt fast enough. That is because using other management tools would mean to change the entire managing software of the company, in order words, starting from zero again. In the future, however, it would be nice to provide a client management system aswell, including subscriptions and so on.

5.2.3 User needs

After having briefly defined the application, two *personas* can be identified: the owner or worker of the gym, and the client.

In the following sections, a brief explanation will be given about the functional requirements, yet such requirements will be explained more in depth in later sections.

5.2.3.1 Owner or worker

Such persona needs entire access to the management system, and it is the main user of it. For this persona, the following needs can be defined:

- *Ability to create virtual gyms and gym zones*. It is the main purpose of the application. It has to provide all the tools that are required to have an above average managing system.
- *Modify virtual gyms and gym zones*. Constraints may change during time. Overall capacity may increase, more cardio machines may be added, and,

with these changes, the gym zones will have to adapt to such real life modifications. There has to exist an interface that provides an easy and simple tool to control it.

- *Ability to manage clients*. The income of the gym is based on the amount of clients they have, and such clients have to be subscribed into the system. However, the application is not a client management system. It purely focuses on the fact of appointments, yet the client still has to be registered in the application.

- *Ability to manage appointments*. The workers have to be able to create, modify or delete the appointments at any time. This process has to be fast and simple, since it is more than usual to have cancellations, clients without reservation, and many more.

- *Ability to manage guided events*. Some gym zones will be marked as a guided class zone. Therefore, a schedule will have to be set up for such zone in order to let the user know that.

Furthermore, two user profiles have to be differentiated: the *owner* and the *worker*. The owner needs more control than the worker and some additional needs are[1]:

- *Ability to manage workers*. The owner has to be able to add, remove or update their workers. There must exist an interface that allows such control.

- *Ability to manage the privileges of the workers*. In large gymnasium franchises, there may exist different types of workers, each with different tasks. For instance, some may only be able to manage guided sessions, others will manage the client subscriptions and so on. Therefore, a privilege system is required to define a hierarchy in the company.

- *Ability to manage gym trainers*. It is important to know what trainers will be responsible for each guided class. For instance, some clients may prefer some trainers than others, when choosing a guided class.

5.2.3.2 Client

The client has little interaction with the system. However, a management application for clients would be pointless if clients could not book their sessions. That is why it will exist another platform to provide access to the client. Such client, will have the following needs:

- *Ability to create, modify and delete appointments*. The clients will make the most use of the reservation system. There has to exist a simple and intuitive interface that allows the clients to interact with the system.

- *Ability to visualise guided events*. In case the client prefers booking a place for a guided class, it should be able to visualise all the possible events for a day or week, for a zone.

[1] Some workers may also have access to such capabilities, depending on the permission they have recieved by the owner.

5.3 Product functionalities

The goal of this section is to explain how would the basic flow of the application be. From the owner arranging virtual gyms, the events and assigning the trainers; to the client being able to manage their appointments. By exemplifying the application behavior, it will be easier to determine the functional requirements of the software.

5.3.1 Gym

When registering, the owner will be required to enter the gym information and will be able to customise other information such as the gym contact data. Such settings will be common to all clients and workers, as they will be part of such gym.

5.3.2 Virtual gym and gym zones

The interface has to be intuitive both for the user and the client, while at the same time providing as much utility as possible. Before being able to create appointments, the client needs a place to create such appointments. Therefore, the first mission of the owner is to define each zone. A gym company may have one or more virtual gyms. As stated before, a virtual gym is the representation of the gym infrastructure in the system. Each virtual gym will have different characteristics and information as: the gym location, the total capacity, the opening and closing hours and much more. This virtual gyms can be modified as they may change time to time. If ever needed, the fact of deleting the different virtual gyms will also be possible.

Once the virtual gym has been set up, it can have gym zones. Each gym zone will also have its characteristics, most of them similar to the virtual gym ones. However, there is a characteristic that is important to mention: the *type* of the gym zone. Each zone is required a type since some zones may be suitable for events and some not. Such types will be predefined, however the user will have the ability to create their own types. Nonetheless, there will still be the need of distinction between class zones and non-class zones.

5.3.3 Events, trainers and workers

After having defined the structure of the gym, the owner or the workers of the gym can create and schedule events for the different gym zones. Events can not be overlapped in the same zone, yet some events can be scheduled at the same time in different zones. Furthermore, the same class can be done in multiples zones, with different trainers.

In order to keep such consistency, the gym owners will create *template events*, which will have specific characteristics. Each template class will be unique, and it will be used to create *events*. In short, the client will create an appointment to a class, which will be a scheduled template class. Nonetheless, appointments will also be required for non-class gym zones, as it is needed to know how many users

will access such zone in a certain time. With the explained relation, owners and workers will not have the need to create a class with all the details every single time, rather simply scheduling already created template events.

Additionally, a trainer manager system is required. However, a trainer it is a worker as well, only it does not interact with the application. Hence, in the same management view, the owner, and if any worker is allowed to, will be able to create trainers and workers.

5.3.4 Clients

Before continuing, the system still lacks of the main income of the gymnasium: the client. As detailed before, the goal of this application is not to provide a client management system, though it could be a functionality to be implemented in the future. Therefore, the client will also be related to the gym entity.

There has to be, definitely, a small client management. However, this management does not need to know about all the information it is kept about the user in the gym. It will only need:

- Personal information such as the name, last name and so on. It should include an email to log in to the software.

That is it. From there, the client will have access to another view of the application in which they will be able to make the reservations.

5.3.5 Creating appointments

When the client has been able to connect to the platform, they will be able to visualise the virtual gyms of the gym they belong to. There, the client will see the availability of the different gym zones, the events offered in that zone and other relevant information. From there, they will create their appointments which will be persisted in the system.

After the appointment has been made, the client will be able to modify it and remove it, without the need of any explanation.

5.4 Functional requirements

5.4.1 Introduction

In this section, the above described requirements will be explained more in depth. The functional requirements are defined as the core functionalities of the system. All requirements exposed in the section are considered essentials and the system would be considered incomplete if it did not satisfy such requisites.

The requisites will be exposed with their code name and a number (as *FR-1*), their priority and a brief description. Three levels of priority have been defined:

1. **High priority (3)**. Such requisites must be fulfilled by the application in order to provide a proper user experience.

2. **Medium priority (2)**. Such requisites should be fulfilled by the application in order to provide an above average user experience.
3. **Low priority (1)**. Such requisites would provide an excellent user experience, yet they are not mandatory.

Furthermore, the overall application will be separated in three different front end applications:

1. **Landing app**. Application which contains the landing application of the product.
2. **Core app**. Main application of the product, where the user does most of the work.
3. **Appointments app**. Application used by the gym client in order to book their appointments.

5.4.2 Landing app - Product information

The application must have a landing page in which the potential user can find the different features offered by the application.

- **FR-1 (3)**. The user needs to know the different features of the product in a single page.
- **FR-2 (3)**. The user needs to be able to be redirected to the register and login page, from the landing page. Such page is the core page.
- **FR-3 (2)**. The user needs to be able to find a contact page in the landing page and ask for the desired information by providing personal information.
- **FR-4 (3)**. The user needs to find information about the data privacy. No personal data and information is intended to be sold, nor shared.

5.4.3 Core app - User registration

The user, in this case the owner, has to be able to register. In the beginning, the application will be completely free, so the gym owners will have complete access only by registering. A future implementation would be to provide additional features only to the users with a subscription.

- **FR-5 (3)**. The user has to provide personal information in order to register.
- **FR-6 (3)**. The user has to provide information about the gym in order to register.
- **FR-7 (3)**. The user has to provide an email and a password to access the application (log in).
- **FR-8 (3)**. The user needs to be able to see how their data is kept (personal data privacy).
- **FR-9 (1)**. The user should see the information for the different premium and free plans.
- **FR-10 (1)**. The user could have a free trial for the premium features.

5.4.4 Core app - Home page

In the home page view, the user will manage their virtual gyms and gym zones. Furthermore, it will be able to create, update and remove the template events which will be then linked to gym zones schedule.

5.4.4.1 Virtual gyms

The user will enter the home page and will see the list of their virtual gyms. If no virtual gym has been created, they will be asked if they want to create a virtual gym (which will be optional). From the same view, they will be able to create, modify and archive the different virtual gyms, aside from accessing the gym zones of that virtual gym.

- **FR-11 (3)**. The view has to display a list or a grid with the different virtual gyms of the user.
- **FR-12 (3)**. Each virtual gym item has to provide an option to visualise and update all the information of that virtual gym. This means changing both the virtual gym *metadata*, and the virtual gym constraints.
- **FR-13 (3)**. The user has to access the virtual gym view by clicking on a virtual gym.
- **FR-14 (3)**. The user has to access the gym zones view by clicking on a virtual gym.
- **FR-15 (2)**. The user has to be able to search for a virtual gym by its name.
- **FR-16 (1)**. Each virtual gym item has to provide an option to archive that virtual gym (and, consequentially, all the gym zones).
- **FR-17 (1)**. There virtual gym list has to provide an option to programmatically sort the virtual gyms (e.g. by ascending or descending capacity).
- **FR-18 (1)**. There virtual gym list has to provide an option to manually sort the virtual gyms.

5.4.4.2 Gym zones

Once a virtual gym has been created and the user has clicked on the card representing it, the user will be redirected to another view in which the zones of the gym will be displayed. Similarly, as in the virtual gyms view, if the user has not created any gym zone, they will be prompted to do so, if wanted.

- **FR-19 (3)**. The view has to display a list or a grid with the different gym zones of the selected virtual gym.
- **FR-20 (3)**. Each gym zone has to provide an option to visualise and update all the information of that gym zone. This means changing both the gym zone *metadata*, and the gym zone constraints.
- **FR-21 (3)**. The view has to display which zones are class type and wich not.
- **FR-22 (2)**. The user has to see the gym zone view by clicking on it.
- **FR-23 (1)**. Each gym zone has to provide an option to archive the zone.

5.4.4.3 Class gym zone schedule

When the user has created a class gym zone, that zone will have a schedule (or a calendar). Inside this view, the user will be able to see the schedule or calendar from the selected gym zone.

- **FR-24 (3)**. The view has to provide an option to visualise the scheduled events.
- **FR-25 (3)**. From the calendar, the user has to be able to create a guided session (an event template can be used in order to make the process faster). Such *scheduled event* must include: a start and end time, the trainer which will be the guide, a maxiumum capacity and other information.
- **FR-26 (3)**. From the calendar, the user has to be able to create an event.
- **FR-27 (2)**. From the calendar, the user has to be able to see future scheduled events.
- **FR-28 (1)**. From the calendar, the user has to be able to see past scheduled events.
- **FR-29 (1)**. The user has to be able to create guided sessions which are repeated when wanted, from the chosen interval.
- **FR-30 (1)**. After creating a new class on a gym zone, the application has to *recommend* a trainer for that class.

5.4.5 Core app - Events page

Even though this view is simple, it is important to separate the schedule view of a gym zone from the template events. The current view will only provide the necessary tools to create, edit, delete or archive the different event templates.

- **FR-31 (3)**. The user has to be able to see the gym list of event types.
- **FR-32 (3)**. There has to be an option in the menu which allows the user to create, update and delete event types.
- **FR-33 (3)**. The user has to be able to see the gym list of template events.
- **FR-34 (3)**. There has to be an option in the menu which allows the user to create, update and delete template events.
- **FR-35 (2)**. The user has to be able to archive the event types.
- **FR-36 (1)**. The user has to be able to archive template events.

5.4.6 Core app - Workers page

This view has to allow the user to manage the workers of the gym enterprise and set the different permissions.

- **FR-37 (3)**. The user has to be able to create a new worker, providing personal information and credentials for such worker to user the app. Furthermore, it has to allow the user to set the permissions for that worker.
- **FR-38 (3)**. The same view has to provide an option to update the permissions and the personal information of the worker.
- **FR-39 (3)**. The view has to provide a table to list all the workers.
- **FR-40 (1)**. The user has to be able to set the working hours of each worker.
- **FR-41 (1)**. The view has to provide an input to search the by name, last name or other characteristics the different workers.

5.4.7 Core app - Trainers page

This view has to allow the user to manage the trainers of the gym enterprise.

- **FR-42 (3)**. The user has to be able to create a new trainer, providing personal information and credentials for such worker to user the app.
- **FR-43 (3)**. The same view has to provide an option to update the trainer personal information.
- **FR-44 (3)**. The view has to provide a table list all the trainers.
- **FR-45 (1)**. The user has to be able to set the working hours of each trainer.
- **FR-46 (1)**. The view has to provide an input to search the by name, last name or other characteristics the different trainers.

5.4.8 Core app - Clients page

This view has to allow the user to manage the clients of the gym enterprise.

- **FR-47 (3)**. The user has to be able to create a new client, providing personal information and credentials for such worker to user the app. Alternatively, the client can register himself (**FR-65**) with a gym code.
- **FR-48 (3)**. The same view has to provide an option to update the client personal information.
- **FR-49 (3)**. The view has to provide a table list all the clients.
- **FR-50 (1)**. The view has to provide an input to search the by name, last name or other characteristics the different clients.

5.4.9 Core app - Settings page

This view has to provide enough settings to customizer the behavior of the behavior and the gym information.

- **FR-51 (3)**. The view has to allow the user to log out.
- **FR-52 (3)**. As an owner, the settings menu has to allow the user to change the gym name and gym characteristics.
- **FR-53 (3)**. As an owner or worker, the settings page has to allow the user to change their personal information.
- **FR-54 (3)**. As an owner or worker, the settings page has to allow the user to change their password.
- **FR-55 (1)**. The settings menu has to provide an option to change the application theme (light and dark) and persist it.
- **FR-56 (1)**. The settings menu has to provide an option to customise the main colours of the user interface.

5.4.10 Core app - Analysis page

The goal of this view is to provide some metrics to *what is happening at the gym*. However, **as it is not the main purpose of the application**, all the requirements from each will be marked with *low priority*.

- **FR-57 (1)**. The view has to provide a select option to choose what virtual gym wants to be analised.
- **FR-58 (1)**. After having selected a virtual gym, the view has to display in which intervals the virtual gym is most crowded at the current date.
- **FR-59 (1)**. After having selected a virtual gym, the view has to display in which intervals the virtual gym is most crowded during an interval.
- **FR-61 (1)**. After having selected a virtual gym, the view has to display what events will have more clients during an interval.
- **FR-62 (1)**. After having selected a virtual gym, the view has to display what events will have more clients during at the current date.
- **FR-63 (1)**. After having selected a virtual gym, the view has to provide a filter to check what gym zones are more crowded during an interval.
- **FR-64 (1)**. After having selected a virtual gym, the view has to provide a filter to check what gym zones are more crowded at the current date.

5.4.11 Client app

The client application will reuse most of the content from the Core application. However, as it is a client, all the functionalities of modifying and deleting will no appear, as the client does not have access to such views.

- **FR-65 (3)**. The client has to provide personal information and a gym code in order to register.
- **FR-66 (3)**. The client has to provide an email and password to access the application (log in).
- **FR-67 (3)**. The home page has to display a list or a grid with the different virtual gyms of the gym they are subscribed.
- **FR-68 (3)**. On clicking any virtual gym, the home view has to display the gym zones of the selected virtual gym.
- **FR-69 (3)**. The calendar should display the scheduled events, week by week.
- **FR-70 (2)**. On clicking a scheduled event, the client will be able to book a place, if the event is not full.
- **FR-71 (3)**. If the user clicks on a non-class gym zone, a form has to be shown to choose the interval in which they want to make the appointment. This includes the date of the appointment, the duration and the starting hour.
- **FR-72 (3)**. On selecting a date and a duration, the server has to return what available starting hours there are. If there is no capacity for such selections, the user will not be able to create the appointment.

5.4.12 Client app - Settings page

The personal information, which is nearly the settings page for the client, has to display most of their information. Additionally, it can show some user stats *for the geeks*.

- **FR-73 (3)**. The view has to allow the user to log out.
- **FR-74 (3)**. The view has to allow the client to change their personal information.
- **FR-75 (3)**. The view has to allow the client to change their password.
- **FR-76 (3)**. The view has to display a list with all the upcoming appointments of the user.
- **FR-77 (1)**. If wanted, the user has to be able to see a list with all the past appointments.
- **FR-78 (1)**. The settings menu has to provide an option to change the application theme (light and dark) and persist it.
- **FR-79 (1)**. The view has to display an analytics table to see which days have they accessed a virtual gym.

5.5 Non-functional requirements

A non-functional requirement is a requirement that defines system attributes such as security, reliability, maintainability, scalability and usability. Therefore, such requisites do not describe information to keep nor functionalities to implement, rather characteristics of such functionalities. In the system, the following non-functional requirements can be defined:

- **NFR-1 (3)**. The system has to provide a secure authentication method.
- **NFR-2 (2)**. The application has to provide different authentication methods in order to differentiate the gym owner, the gym worker and the gym client.
- **NFR-3 (2)**. The landing and client web applications must provide a responsive interface, so that the application can be seen both in small and large screens.
- **NFR-4 (1)**. The core application has to provide a responsive interface up to a point. In smaller screens such as phones, some functionalities would not be easy to implement nor to use.
- **NFR-5 (2)**. The three web applications have to provide an accessible and semantic HTML structure.

5.6 Requirements dependency matrix

Due to the large quantity of requirements that have been determined for the development of the project, and due to the low relationship between them, the dependencies will be shown individually, in order to avoid displaying a nearly empty table. Using the following notation, the result is more consise and can be understood better.

Each dependency will use the following format: **[FR-1, FR-2, FR-3] → [FR-4, FR-5]**. It states that the functional requirements 1, 2 and 3 depend on the 4 and the 5.

[FR-11, FR-13, FR-14, FR-16, FR-17, FR-18] → [FR-12]. In order to visualise, edit or delete a virtual gym, such virtual gym has had to be created previously.

[FR-19, FR-20, FR-21, FR-22, FR-23] → [FR-12]. The gym zones are required to exist inside a virtual gym. If such virtual gym has not been created, the user can not visualise, edit or delete the gym zone.

[FR-19, FR-21, FR-22, FR-23] → [FR-20]. In order to visualise, edit or delete a gym zone, such gym zone has had to be created previously.

[FR-24, FR-25, FR-26, FR-27, FR-28, FR-29, FR-30] → [FR-20]. A calendar is required to exist inside a gym zone. If such gym zone has not been created, the user can not visualise, edit or delete the events of the calendar.

[FR-31, FR-35] → [FR-32]. In order to visualise, edit or delete an event type, such event type has had to be created previously.

[FR-33, FR-36] → [FR-36]. In order to visualise, edit or delete an event template, such event template has had to be created previously.

[FR-38, FR-39, FR-40, FR-41] → [FR-27]. In order to visualise, edit or delete a worker, such worker has had to be created previously.

[FR-43, FR-44, FR-45, FR-46] → [FR-42]. In order to visualise, edit or delete a trainer, such trainer has had to be created previously.

[FR-48, FR-49, FR-50, FR-51] → [FR-47]. In order to visualise, edit or delete a client, such client has had to be created or registered previously.

[FR-66, FR-67, FR-68, FR-69, FR-70, FR-71, FR-72] → [FR-65]. In order to interact with the application, the client must have had registered previously.

6. Planning

This chapter will cover the planning of the project. After having defined the requirements and the matrix of dependencies of such requirements, working packages will be defined, which will ensure an organized development using tasks.

6.1 Working packages

Each's table corresponds to one of the working packages. Since the *Development* branch of the working packages tree the most extensive, its tables are explained after the *Testing* branch, and so are the package numbers.

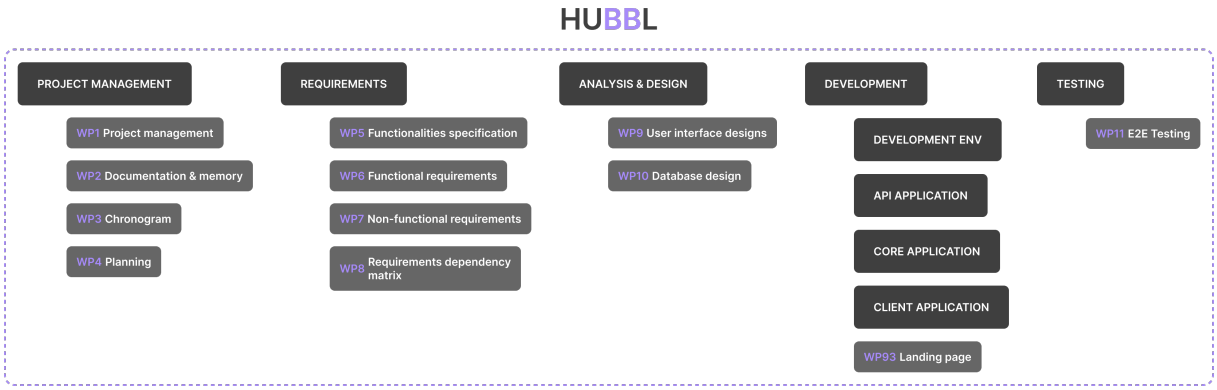


Figure 6.1: Structure of the working packages at the root level

The image above shows how the working packages have been organized. Each column represents a module, which are:

- Project management.
- Requirements.
- Analysis and design.
- Development.
- Testing.

Since the development branch is very extensive, each project has been extracted in smaller working packages.

6.1.1 Project management

The *project management* branch contains all the working packages that are related any task that involves the specification of how the project will be managed in order to be successful.

Package name	**WP1**: Project management
Description	Working package description.
Estimated time	8h
Tasks	**T1**: Project document writing. **T2**: Project document review.
Results	Project document to be submitted to the jury.

Table 6.1: Package one's table - Project management

Package name	**WP2**: Documentation and memory
Description	Wording of some chapters of the memory.
Estimated time	8h
Tasks	**T1**: Wording of the project introduction. **T2**: Wording of the requirements. **T3**: Wording of the analysis and design. **T4**: Wording of the studies and decisions. **T5**: Wording of the project development.
Results	Project document to be submitted to the jury.

Table 6.2: Package two's table - Documentation and memory

Package name	**WP3**: Chronogram
Description	Define the working chronogram for the project development.
Estimated time	4h
Tasks	**T1**: Temporal planning of the working packages.
Results	Gantt's project diagram.

Table 6.3: Package three's table - Chronogram

Package name	**WP4**: Planning
Description	Structure the working packages.
Estimated time	4h
Tasks	**T1**: Planning of the working packages.
Results	Chapter 6 of the memory document.

Table 6.4: Package four's table - Planning

Package name	**WP5**: Functionalities specification
Description	Analysis of the functionalities of the application.
Estimated time	8h
Tasks	**T1**: Users identification. **T2**: Define user needs (owner/worker and client). **T3**: Define product functionalities
Results	Sections 5.2 and 5.3 of the memory document.

Table 6.5: Package five's table - Functionalities specification

6.1.2 Requirements

The *requirements* branch contains all the tasks that are related with the analysis and definition of all the requirements and functionalities of each application.

Package name	**WP6**: Functional requirements
Description	Analysis and definition of the functional requirements of the application.
Estimated time	12h
Tasks	**T1**: Use case diagram. **T2**: Functional requirements' specification for the *core* application. **T3**: Functional requirements' specification for the *client* application. **T4**: Functional requirements' specification for the *landing* application. **T5**: Functional requirements review and make changes if needed.
Results	Section 5.4 of the memory document.

Table 6.6: Package six's table - Functional requirements

Package name	**WP7**: Non-functional requirements
Description	Analysis and definition of the non-functional requirements of the application.
Estimated time	4h
Tasks	**T1**: Initial non-functional requirements' specification. **T2**: Non-functional requirements review and make changes if needed.
Results	Section 5.5 of the memory document.

Table 6.7: Package seven's table - Non-functional requirements

Package name	**WP8**: Requirements dependency matrix
Description	Definition of the requirements' dependency matrix.
Estimated time	4h
Tasks	**T1**: Matrix dependency specification once the functional and non-functional requirements have been specified.
Results	Section 5.6 of the memory document.

Table 6.8: Package eight's table - Requirements dependency matrix

6.1.3 Analysis and design

The *analysis and design* branch contains all the working packages that are related with the design and analysis of the system interface and architecture.

Package name	**WP9**: User interface designs
Description	Definition of the requirements' dependency matrix.
Estimated time	96h
Tasks	**T1**: Design system specification. **T2**: Prototype most of the user interfaces. **T3**: Design review. Ensure it satisfies the requirements previously specified.
Results	User interface prototypes which provide a vague view of the application's look and feel.

Table 6.9: Package nine's table - User interfaces designs

Package name	**WP10**: Database design
Description	Definition of the application's database.
Estimated time	4h
Tasks	**T1**: Design of the applications' database structure. **T2**: Schema and structure review. **T3**: Analysis of the database structure using PostgreSQL.
Results	Database structure in which the application's data will be persisted.

Table 6.10: Package ten - Database design

6.1.4 Testing

The *testing* branch contains the working package that is related with the testing of the application.

Package name	**WP11**: E2e testing
Description	End-to-end testing implementation for the applications.
Estimated time	96h
Tasks	**T1**: *API* aplication e2e testing. **T2**: *Core* application e2e testing. **T3**: *Client* application e2e testing. **T4**: *Landing* application e2e testing.
Results	End-to-end tests which ensure the correct behaviour of each application, and that any futher new features and updates do not break the application.

Table 6.11: Package eleven's table - E2e testing

6.1.5 Development

The *development* branch is probably the most extensive, as it is the most important part of the project. The branch consists of 73 working packages in total. Nevertheless, most of the packages follow the same structure. For instance, as it is seen in the Api application, each web service package follows the same structure: a working package to create an entity, another one to update it, a third one to delete it, and a final one to fetch the entity. The structure is the same for each web service, which easily increases the number of working packages.

6.1.5.1 Development env

The first group that is in the *development* branch is the *development env* (or *development environment*). This part is as important as any other application, since

it will define the structure of the project. Furthermore, the continuous integration has to be set up. Using the Nx build system, such feature becomes extremely simple and scalable. One of the options it offers is to run a command to as many projects as wanted. Not only so, that you can run the commands to the affected projects. Such feature reduces even more the CI execution time.

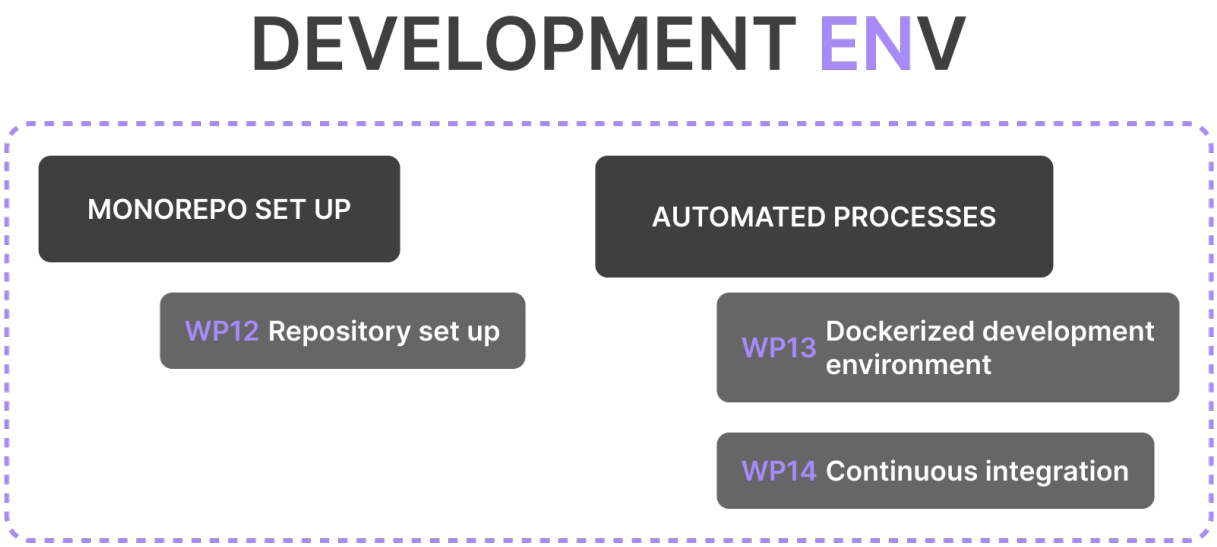


Figure 6.2: Api application working packages diagram

The *development environment* group is composed of the following working packages:

Package name	**WP12**: Repository set up
Description	Prepare the Nx monorepo.
Estimated time	4h
Tasks	**T1**: Generate an `nx-workspace` ready to develop.
Results	Initial structure of the monorepo.

Table 6.12: Package twelve's table - Repository set up

Package name	**WP13**: Dockerized development environment.
Description	Dockerize the database, the test database and the API application.
Estimated time	4h
Tasks	**T1**: Dockerize the main database and the test database. **T2**: Dockerize the API application using a NodeJS image.
Results	The dockerization of the databases and the REST API.

Table 6.13: Package thirteen's table - Dockerized development environment

Package name	**WP14**: Continuous integration.
Description	Set up Github Actions for CI.
Estimated time	8h
Tasks	**T1**: Set up CI for the *API* application. **T2**: Set up CI for the *core* application. **T3**: Set up CI for the *client* application. **T4**: Set up CI for the *landing* application. **T5**: Set up CI for the monrepo libraries.
Results	An automated process for testing the code developed.

Table 6.14: Package fourteen's table - Continuous integration

6.1.5.2 Api application

The working package groups of the *api* application have been designed so that each web service of the application is divided in multiple working packages with tasks that are very specific and simple.

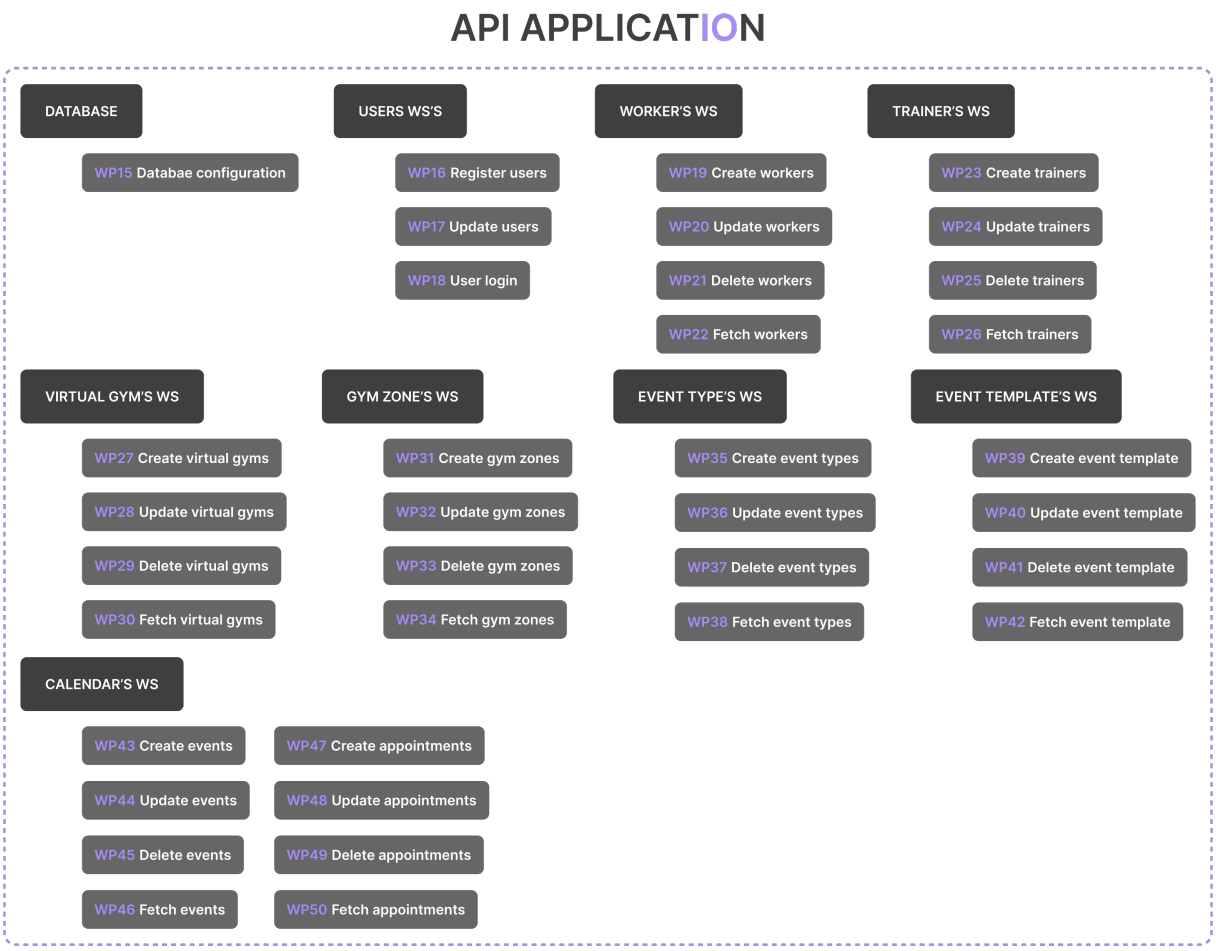


Figure 6.3: Api application working packages diagram

The *api* group is composed of the following working packages:

Package name	**WP15**: Database configuration.
Description	Configure the application, so that is connected the database.
Estimated time	1h
Tasks	**T1**: Configure the API with the dockerized database.
Results	Connection of the api with the database.

Table 6.15: Package fifteen's table - Database configuration

Package name	**WP16**: Register users
Description	Allow the users to register.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the owner register endpoint. **T4**: Create the client register endpoint.
Results	Creation of a user.

Table 6.16: Package sixteen's table - Register users

Package name	**WP17**: Update users
Description	Allow the users to update their information.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the user update endpoint.
Results	Updation of user's information

Table 6.17: Package seventeen's table - Update users

Package name	**WP18**: User login
Description	Allow the users to log in to the application.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the session cookie validation endpoint. **T4**: Create the login endpoint.
Results	Endpoints to login and validate old user sessions.

Table 6.18: Package eighteen's table - User login

Package name	**WP19**: Create workers
Description	Allow the creation of workers.
Estimated time	6h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the workers create endpoint.
Results	Creation of workers.

Table 6.19: Package nineteen's table - Create workers

Package name	**WP20**: Update workers
Description	Allow the update of workers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the update workers endpoint.
Results	Updation of workers.

Table 6.20: Package twenty's table - Update workers

Package name	**WP21**: Delete workers
Description	Allow the deletion of workers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the delete workers endpoint.
Results	Deletion of workers.

Table 6.21: Package twenty-one's table - Delete workers

Package name	**WP22**: Fetch workers
Description	Allow the fetch of workers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the fetch workers endpoint.
Results	Fetching of workers.

Table 6.22: Package twenty-two's table - Fetch workers

Package name	**WP23**: Create trainers
Description	Allow the creation of trainers.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the create trainers endpoint.
Results	Creation of trainers.

Table 6.23: Package twenty-three's table - Create trainers

Package name	**WP24**: Update trainers
Description	Allow the update of trainers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the update trainers endpoint.
Results	Updation of trainers.

Table 6.24: Package twenty-four's table - Update trainers

Package name	**WP25**: Delete trainers
Description	Allow the deletion of trainers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the delete trainers endpoint.
Results	Deletion of trainers.

Table 6.25: Package twenty-five's table - Delete trainers

Package name	**WP26**: Fetch trainers
Description	Allow the fetch of trainers.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the fetch trainers endpoint.
Results	Fetching of trainers.

Table 6.26: Package twenty-six's table - Fetch trainers

Package name	**WP27**: Create virtual gyms
Description	Allow the creation of virtual gyms.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the virtual gym creation endpoint.
Results	Creation of virtual gyms.

Table 6.27: Package twenty-seven's table - Create virtual gyms

Package name	**WP28**: Update virtual gyms
Description	Allow the update of virtual gyms.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the virtual gym update endpoint.
Results	Updation of virtual gyms.

Table 6.28: Package twenty-eight's table - Update virtual gyms

Package name	**WP29**: Delete virtual gyms
Description	Allow the deletion of virtual gyms.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the virtual gym deletion endpoint.
Results	Deletion of virtual gyms.

Table 6.29: Package twenty-nine's table - Delete virtual gyms

Package name	**WP30**: Fetch virtual gyms
Description	Allow the fetch of virtual gyms.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the virtual gym fetch endpoint.
Results	Fetching of virtual gyms.

Table 6.30: Package thirty's table - Delete virtual gyms

Package name	**WP31**: Create gym zones
Description	Allow the creation of gym zones.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the gym zone create endpoint.
Results	Creation of gym zones.

Table 6.31: Package thirty-one's table - Create gym zones

Package name	**WP32**: Update gym zones
Description	Allow the update of gym zones.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the gym zone update endpoint.
Results	Updation of gym zones.

Table 6.32: Package thirty-two's table - Update gym zones

Package name	**WP33**: Delete gym zones
Description	Allow the deletion of gym zones.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the gym zone delete endpoint.
Results	Deletion of gym zones.

Table 6.33: Package thirty-three's table - Delete gym zones

Package name	**WP34**: Fetch gym zones
Description	Allow the fetching of gym zones.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the gym zone fetch endpoint.
Results	Fetching of gym zones.

Table 6.34: Package thirty-four's table - Fetch gym zones

Package name	**WP35**: Create event types
Description	Allow the creation of event types.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event types create endpoint.
Results	Creation of event types.

Table 6.35: Package thirty-five's table - Create event types

Package name	**WP36**: Update event types
Description	Allow the update of event types.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event types update endpoint.
Results	Updation of event types.

Table 6.36: Package thirty-six's table - Update event types

Package name	**WP37**: Delete event types
Description	Allow the deletion of event types.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event types delete endpoint.
Results	Updation of event types.

Table 6.37: Package thirty-seven's table - Delete event types

Package name	**WP38**: Fetch event types
Description	Allow the fetch of event types.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event types fetch endpoint.
Results	Fetching of event types.

Table 6.38: Package thirty-eight's table - Fetch event types

Package name	**WP39**: Create event templates
Description	Allow the creation of event templates.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event templates create endpoint.
Results	Creation of event templates.

Table 6.39: Package thirty-nine's table - Create event templates

Package name	**WP40**: Update event templates
Description	Allow the update of event templates.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event templates update endpoint.
Results	Updation of event templates.

Table 6.40: Package forty's table - Update event templates

Package name	**WP41**: Delete event templates
Description	Allow the deletion of event templates.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event templates delete endpoint.
Results	Deletion of event templates.

Table 6.41: Package forty-one's table - Delete event templates

Package name	**WP42**: Fetch event templates
Description	Allow the fetch of event templates.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the event templates fetch endpoint.
Results	Fetching of event templates.

Table 6.42: Package forty-two's table - Fetch event templates

Package name	**WP43**: Create events
Description	Allow the creation of events.
Estimated time	8h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the events fetch endpoint.
Results	Creation of events.

Table 6.43: Package forty-three's table - Create events

Package name	**WP44**: Update events
Description	Allow the update of events.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the events update endpoint.
Results	Updation of events.

Table 6.44: Package forty-four's table - Update events

Package name	**WP45**: Delete events
Description	Allow the deletion of events.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the events delete endpoint.
Results	Deletion of events.

Table 6.45: Package forty-five's table - Delete events

Package name	**WP46**: Fetch events
Description	Allow the fetching of events.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the events fetch endpoint.
Results	Fetching of events.

Table 6.46: Package forty-six's table - Fetch events

Package name	**WP47**: Create appointments
Description	Allow the fetching of appointments. It is important to create an algorithm that ensures that a gym zone does not have more appointments than its capacity.
Estimated time	16h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T3**: Create the available hours endpoint. **T4**: Create the appointments create endpoint.
Results	Creation of appointments.

Table 6.47: Package forty-seven's table - Create appointments

Package name	**WP48**: Update appointments
Description	Allow the update of appointments. It should only allow to cancel the appointment.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T4**: Create the appointments update endpoint.
Results	Updation of appointments.

Table 6.48: Package forty-eight's table - Update appointments

Package name	**WP49**: Delete appointments
Description	Allow the deletion of appointments. It should only allow to cancel the appointment.
Estimated time	4h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T4**: Create the appointments delete endpoint.
Results	Deletion of appointments.

Table 6.49: Package forty-nine's table - Delete appointments

Package name	**WP50**: Fetch appointments
Description	Allow the fetch of appointments.
Estimated time	6h
Tasks	**T1**: Create the services required. **T2**: Create the controllers required. **T4**: Create the appointments fetch endpoint.
Results	Fetching of appointments.

Table 6.50: Package fifty's table - Fetch appointments

6.1.5.3 Core application

The *core* group of working packages follows a similar approach as the one described in Development.

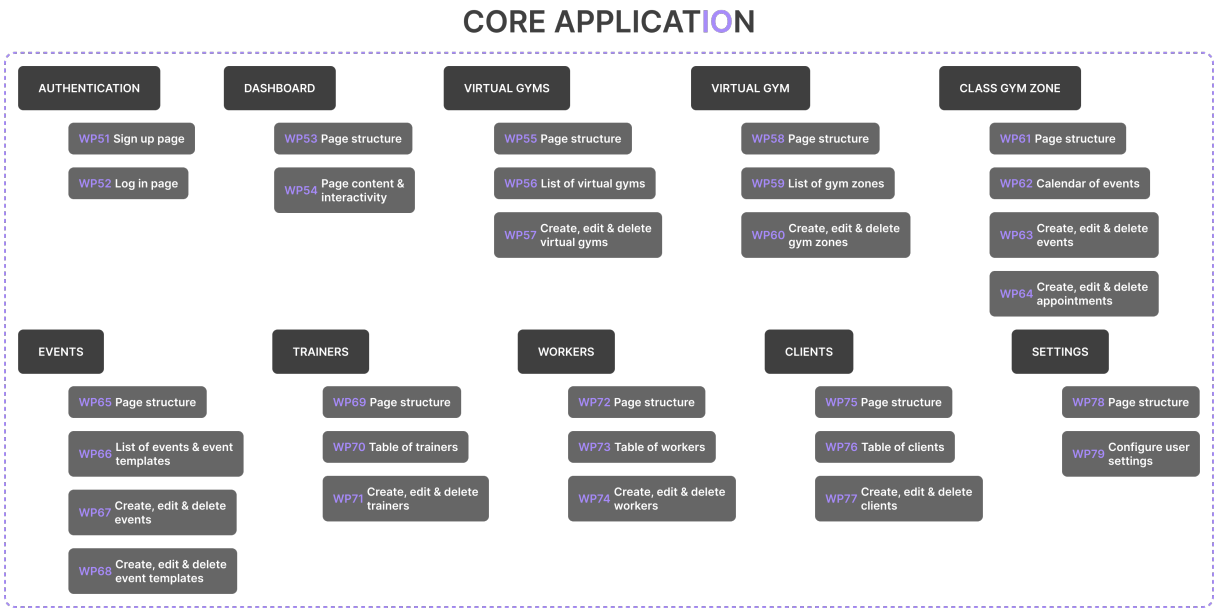


Figure 6.4: Core application working packages diagram

The *core* group is composed of the following working packages:

Package name	**WP51**: Sign up page
Description	Develop the sign up page
Estimated time	16h
Tasks	**T1**: Set up the page structure. **T2**: Sign up as an owner.
Results	Allow the users to register to the system.

Table 6.51: Package fifty-one's table - Sign up page

Package name	**WP52**: Log in page
Description	Develop the log in page.
Estimated time	16h
Tasks	**T1**: Set up the page structure. **T2**: Log in as an owner. **T3**: Log in as a worker.
Results	Allow the user to log in to the system.

Table 6.52: Package fifty-two's table - Log in page

Package name	**WP53**: Page structure (Dashboard)
Description	Apply the design of the dashboard page.
Estimated time	4h
Tasks	**T1**: Apply the design of the dashboard page.
Results	The scaffolding of the dashboard page.

Table 6.53: Package fifty-three's table - Page structure (Dashboard)

Package name	**WP54**: Page content and interactivity (Dashboard)
Description	Add the interactivity to the dashboard page.
Estimated time	12h
Tasks	**T1**: Add virtual gyms' interactivity. **T2**: Add gym zones' interactivity. **T3**: Add trainers' interactivity. **T4**: Add event templates' interactivity. **T5**: Add events' interactivity.
Results	User experience of the dasboard page.

Table 6.54: Package fifty-four's table - Page content and interactivity

Package name	**WP55**: Page structure (Virtual gyms)
Description	Apply the design of the virtual gyms page.
Estimated time	4h
Tasks	**T1**: Apply the design of the virtual gyms page.
Results	The scaffolding of the virtual gyms page.

Table 6.55: Package fifty-five's table - Page structure (Virtual gyms)

Package name	**WP56**: List of virtual gyms (Virtual gyms)
Description	Display the list of virtual gyms.
Estimated time	4h
Tasks	**T1**: List the virtual gyms of the gym. **T2**: List the gym zones inside each virtual gym.
Results	The page of virtual gyms.

Table 6.56: Package fifty-six's table - List of virtual gyms (Virtual gyms)

Package name	**WP57**: Create, edit and delete virtual gyms (Virtual gyms)
Description	Create the modal to create, edit and delete virtual gyms.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete virtual gyms.
Results	The modal to interact with virtual gyms.

Table 6.57: Package fifty-seven's table - List of virtual gyms (Virtual gyms)

Package name	**WP58**: Page structure (Virtual gym)
Description	Apply the design of the virtual gym page.
Estimated time	4h
Tasks	**T1**: Apply the design of the virtual gym page.
Results	The scaffolding of the virtual gym page.

Table 6.58: Package fitfty-eight's table - Page structure (Virtual gym)

Package name	**WP59**: List of gym zones (Virtual gym)
Description	List the gym zones of a virtual gym.
Estimated time	8h
Tasks	**T1**: List all class gym zones. **T2**: List all non-class gym zones.
Results	The page of a virtual gym.

Table 6.59: Package fifty-nine's table - List of gym zones (Virtual gym)

Package name	**WP60**: Create, edit and delete gym zones (Virtual gym)
Description	Create the modal to create, edit and delete gym zones.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete gym zones.
Results	The modal to interact with gym zones.

Table 6.60: Package sixty's table - Create, edit and delete gym zones (Virtual gym)

Package name	**WP61**: Page structure (Class gym zone)
Description	Apply the design of the class gym zone page.
Estimated time	4h
Tasks	**T1**: Apply the design of the class gym zone page.
Results	The scaffolding of the class gym zone page.

Table 6.61: Package sixty-one's table - Page structure (Class gym zone)

Package name	**WP62**: Calendar of events (Class gym zone)
Description	Display the calendar of events of the gym zone.
Estimated time	8h
Tasks	**T1**: Design the calendar of events. **T2**: Add interactivity to the calendar.
Results	The calendar of events for the gym zone.

Table 6.62: Package sixty-two's table - Calendar of events (Class gym zone)

Package name	**WP63**: Create, edit and delete events (Class gym zone)
Description	Modal to create, edit and delete events.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete events.
Results	The modal to interact with events.

Table 6.63: Package sixty-three's table - Create, edit and delete events (Class gym zone)

Package name	**WP64**: Create, edit and delete appointments (Class gym zone)
Description	Modal to create, edit and delete appointments.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete appointments.
Results	The modal to interact with appointments.

Table 6.64: Package sixty-four's table - Create, edit and delete appointments (Class gym zone)

Package name	**WP65**: Page structure (Events)
Description	Apply the design of the events page.
Estimated time	4h
Tasks	**T1**: Apply the design of the events page.
Results	The scaffolding of the events page.

Table 6.65: Package sixty-five's table - Page structure (Events)

Package name	**WP66**: List of events and event templates (Events)
Description	Display the list of events and event templates.
Estimated time	6h
Tasks	**T1**: List the events. **T2**: List the event templates
Results	The events page.

Table 6.66: Package sixty-six's table - List of events and event tempaltes (Events)

Package name	**WP67**: Create, edit and delete events (Events)
Description	Modal to create, edit and delete events.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete events.
Results	The modal to interact with events.

Table 6.67: Package sixty-seven's table - Create, edit and delete events (Events)

Package name	**WP68**: Create, edit and delete events templates (Events)
Description	Modal to create, edit and delete events templates.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete events templates.
Results	The modal to interact with events templates.

Table 6.68: Package sixty-eight's table - Create, edit and delete events templates (Events)

Package name	**WP69**: Page structure (Trainers)
Description	Apply the design of the trainers page.
Estimated time	4h
Tasks	**T1**: Apply the design of the trainers page.
Results	The scaffolding of the trainers page.

Table 6.69: Package sixty-nine's table - Page structure (Trainers)

Package name	**WP70**: Table of trainers (Trainers)
Description	Table of trainers.
Estimated time	4h
Tasks	**T1**: Display the table of trainers.
Results	The table of trainers.

Table 6.70: Package seventy's table - Table of trainers (Trainers)

Package name	**WP71**: Create, edit and delete trainers (Trainers)
Description	Modal to create, edit and delete trainers.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete trainers.
Results	The modal to interact with trainers.

Table 6.71: Package seventy-one's table - Table of trainers (Trainers)

Package name	**WP72**: Page structure (Workers)
Description	Apply the design of the workers page.
Estimated time	4h
Tasks	**T1**: Apply the design of the workers page.
Results	The scaffolding of the workers page.

Table 6.72: Package seventy-two's table - Page structure (Workers)

Package name	**WP73**: Table of workers (Workers)
Description	Table of workers.
Estimated time	4h
Tasks	**T1**: Display the table of workers.
Results	The table of workers.

Table 6.73: Package seventy-three's table - Table of workers (Workers)

Package name	**WP74**: Create, edit and delete workers (Workers)
Description	Modal to create, edit and delete workers.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete workers.
Results	The modal to interact with workers.

Table 6.74: Package seventy-four's table - Table of workers (Workers)

Package name	**WP75**: Page structure (Trainers)
Description	Apply the design of the trainers page.
Estimated time	4h
Tasks	**T1**: Apply the design of the trainers page.
Results	The scaffolding of the trainers page.

Table 6.75: Package seventy-five's table - Page structure (Trainers)

Package name	**WP76**: Table of trainers (Trainers)
Description	Table of trainers.
Estimated time	4h
Tasks	**T1**: Display the table of trainers.
Results	The table of trainers.

Table 6.76: Package seventy-six's table - Table of trainers (Trainers)

Package name	**WP77**: Create, edit and delete trainers (Trainers)
Description	Modal to create, edit and delete trainers.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete trainers.
Results	The modal to interact with trainers.

Table 6.77: Package seventy-seven's table - Table of trainers (Trainers)

Package name	**WP78**: Page structure (Settings)
Description	Apply the design of the settings page.
Estimated time	4h
Tasks	**T1**: Apply the design of the settings page.
Results	The scaffolding of the settings page.

Table 6.78: Package seventy-eight's table - Page structure (Settings)

Package name	**WP79**: Configure user settings (Settings)
Description	Set up the form to update user settings.
Estimated time	8h
Tasks	**T1**: Log out the user. **T2**: Configure the basic fields of the user. **T3**: Update the user password. **T4**: Update the gym information as an owner.
Results	The settings page.

Table 6.79: Package seventy-nine's table - Configure user settings (Settings)

6.1.5.4 Client application

The *landing* group of working packages follows a similar approach as the one described in Development.

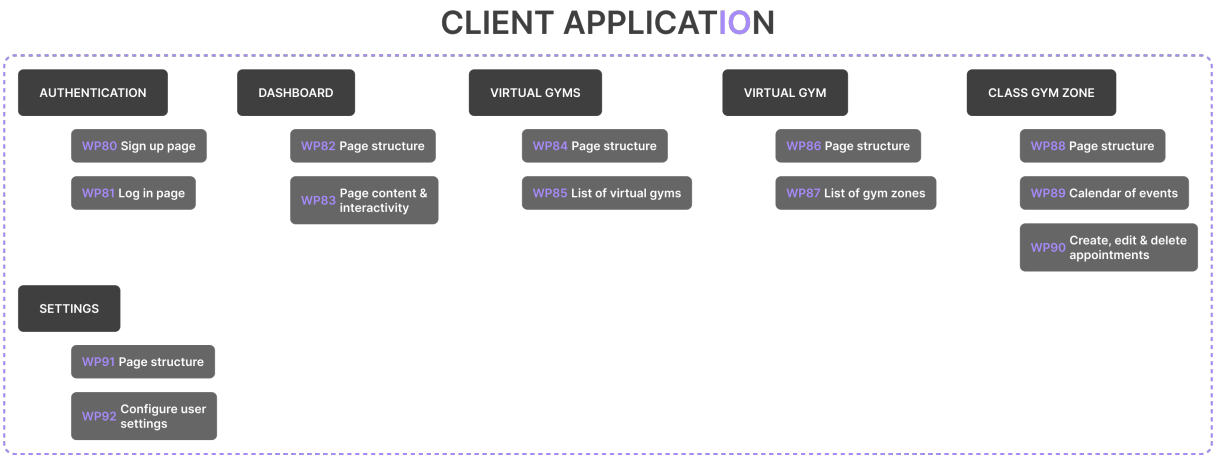


Figure 6.5: Client application working packages diagram

The *client* group is composed of the following working packages:

Package name	**WP80**: Sign up page
Description	Develop the sign up page
Estimated time	8h
Tasks	**T1**: Set up the page structure. **T2**: Sign up as a client.
Results	Allow the clients to register to the system.

Table 6.80: Package eighty's table - Sign up page

Package name	**WP81**: Log in page
Description	Develop the log in page.
Estimated time	8h
Tasks	**T1**: Set up the page structure. **T2**: Log in as a client.
Results	Allow the user to log in to the system.

Table 6.81: Package eighty-one's table - Log in page

Package name	**WP82**: Page structure (Dashboard)
Description	Apply the design of the dashboard page.
Estimated time	4h
Tasks	**T1**: Apply the design of the dashboard page.
Results	The scaffolding of the dashboard page.

Table 6.82: Package eighty-two's table - Page structure (Dashboard)

Package name	**WP83**: Page content and interactivity (Dashboard)
Description	Add the interactivity to the dashboard page.
Estimated time	6h
Tasks	**T1**: Add virtual gyms' interactivity. **T2**: Add gym zones' interactivity.
Results	User experience of the dasboard page.

Table 6.83: Package eighty-three's table - Page content and interactivity

Package name	**WP84**: Page structure (Virtual gyms)
Description	Apply the design of the virtual gyms page. Should reuse the core's page.
Estimated time	4h
Tasks	**T1**: Apply the design of the virtual gyms page.
Results	The scaffolding of the virtual gyms page.

Table 6.84: Package eighty-fourt's table - Page structure (Virtual gyms)

Package name	**WP85**: List of virtual gyms (Virtual gyms)
Description	Display the list of virtual gyms.
Estimated time	4h
Tasks	**T1**: List the virtual gyms of the gym. **T2**: List the gym zones inside each virtual gym.
Results	The page of virtual gyms.

Table 6.85: Package eighty-five's table - List of virtual gyms (Virtual gyms)

Package name	**WP86**: Page structure (Virtual gym)
Description	Apply the design of the virtual gym page. Should reuse the core's page.
Estimated time	4h
Tasks	**T1**: Apply the design of the virtual gym page.
Results	The scaffolding of the virtual gym page.

Table 6.86: Package eighty-six's table - Page structure (Virtual gym)

Package name	**WP87**: List of gym zones (Virtual gym)
Description	List the gym zones of a virtual gym.
Estimated time	4h
Tasks	**T1**: List all class gym zones. **T2**: List all non-class gym zones.
Results	The page of a virtual gym.

Table 6.87: Package eighty-seven's table - List of gym zones (Virtual gym)

Package name	**WP88**: Page structure (Class gym zone)
Description	Apply the design of the class gym zone page. Should reuse the core's page.
Estimated time	4h
Tasks	**T1**: Apply the design of the class gym zone page.
Results	The scaffolding of the class gym zone page.

Table 6.88: Package eighty-eight's table - Page structure (Class gym zone)

Package name	**WP89**: Calendar of events (Class gym zone)
Description	Display the calendar of events of the gym zone.
Estimated time	4h
Tasks	**T1**: Design the calendar of events. **T2**: Add interactivity to the calendar.
Results	The calendar of events for the gym zone.

Table 6.89: Package ninety's table - Calendar of events (Class gym zone)

Package name	**WP90**: Create, edit and delete appointments (Class gym zone)
Description	Modal to create, edit and delete appointments.
Estimated time	8h
Tasks	**T1**: Modal to create, edit and delete appointments.
Results	The modal to interact with appointments.

Table 6.90: Package ninety's table - Create, edit and delete appointments (Class gym zone)

Package name	**WP91**: Page structure (Settings)
Description	Apply the design of the settings page. Should reuse the core's page.
Estimated time	4h
Tasks	**T1**: Apply the design of the settings page.
Results	The scaffolding of the settings page.

Table 6.91: Package ninety-one's table - Page structure (Settings)

Package name	**WP92**: Configure user settings (Settings)
Description	Set up the form to update client settings.
Estimated time	4h
Tasks	**T1**: Log out the client. **T2**: Configure the basic fields of the client. **T3**: Update the client password.
Results	The settings page.

Table 6.92: Package ninety-two's table - Configure user settings (Settings)

6.1.5.5 Landing application

The landing application is a simple application that displays information about the system and acces to the main application.

Package name	**WP93**: Design the landing application
Description	Set up the form to update client settings.
Estimated time	16h
Tasks	**T1**: Display basic information about the system. **T2**: Display information about the data privacy. **T3**: Display a contact form.
Results	The landing application.

Table 6.93: Package ninety-three's table - Landing page

6.2 Traceability matrix

Due to the large quantity of requirements and working packageds that have been determined for the development of the project, and due to the low relationship between them, the traceability matrix will be shown individually, in order to avoid displaying a nearly empty table. Using the following notation, the result is more

consise and can be understood better.

Each dependency will use the following format: **[FR-1] → [WP-1, WP-2, WP-3]**. It states that the functional requirements 1 will be covered in the working package 1, 2 and 3.

- **[FR-1, FR-2, FR-3, FR-4] → [WP-93]**
- **[FR-5, FR-6] → [WP-51]**
- **[FR-7] → [WP-52]**
- **[FR-8, FR-9, FR-10] → [WP-93]**
- **[FR-11, FR-13, FR-14, FR-15, FR-17] → [WP-56, WP-28, WP-29, WP-30]**
- **[FR-12, FR-16] → [WP-57, WP-27, WP-28, WP-29]**
- **[FR-19, FR-21, FR-22] → [WP-59, WP-34]**
- **[FR-20, FR-23] → [WP-60, WP-31, WP-32, WP-33]**
- **[FR-24, FR-27, FR-28] → [WP-62, WP-43, WP-44, WP-45, WP-47, WP-48, WP-49]**
- **[FR-25, FR-26, FR-29, FR-30] → [WP-63, WP-34, WP-46, WP-50]**
- **[FR-31, FR-33] → [WP-66, WP-38, WP-42, WP-46]**
- **[FR-32, FR-35] → [WP-67, WP-35, WP-36, WP-37]**
- **[FR-34, FR-36] → [WP-68, WP-39, WP-40, WP-41]**
- **[FR-37, FR-38, FR-40] → [WP-74, WP-19, WP-20, WP-21]**
- **[FR-39, FR-41] → [WP-73, WP-22]**
- **[FR-42, FR-43, FR-45] → [WP-70, WP-23, WP-24, WP-25]**
- **[FR-44, FR-46] → [WP-71, WP-26]**
- **[FR-47, FR-48] → [WP-77]**
- **[FR-49, FR-50] → [WP-76]**
- **[FR-51, FR-52, FR-53, FR-54, FR-55, FR-56] → [WP-79, WP-17]**
- **[FR-65] → [WP-80, WP-16]**
- **[FR-66] → [WP-81, WP-17]**
- **[FR-67, FR-68] → [WP-85, WP-30]**
- **[FR-69] → [WP-89, WP-46]**
- **[FR-70, FR-71, FR-72] → [WP-90, WP-43, WP-44, WP-45, WP-47, WP-48, WP-49]**

- **[FR-73, FR-74, FR-75, FR-76, FR-77, FR-78] → [WP-92, WP-17]**
- **[NFR-1] → [WP-16, WP-18]**
- **[NFR-2] → [WP-52]**
- **[NFR-3] → [WP-93]**
- **[NFR-4] → [WP-53, WP-55, WP-58, WP-61, WP-65, WP-69, WP-72, WP-75, WP-78]**
- **[NFR-5] → [WP-53, WP-55, WP-58, WP-61, WP-65, WP-69, WP-72, WP-75, WP-78, WP-82, WP-84, WP-86, WP-88, WP-91, WP-93]**

6.3 Roadmap

The roadmap contains an estimation of how the development of the application should. Nonetheless, it is just an estimation and many things may occur that could change such schedule. The projec that is expected to take up most of the amount of tiume is the *core* app, as it is the onw that contains the most amount of logic, followed by the server. Even though the projects are structured sequentially, bugs or errors may occur while developing, for instance, a web service does not return what it should[1], which would imply to have it fixed while another app is being developed.

[1]The applications will be tested, yet tests do not ensure a 100% correctness of the application. If the test does not contemplate all corner-cases, there may be a bug.

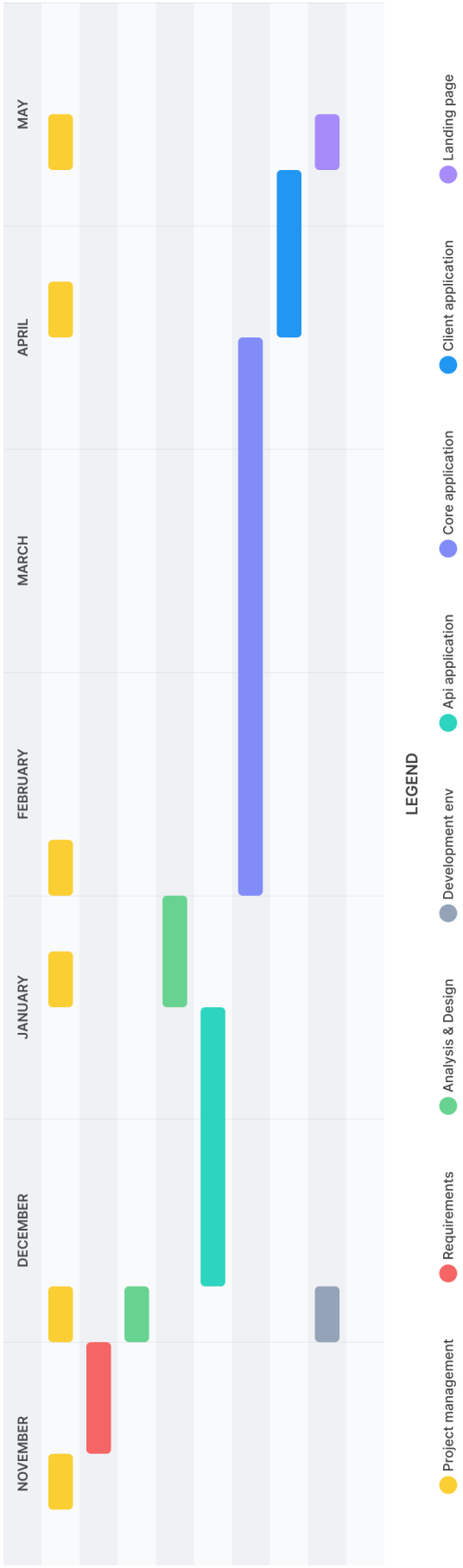


Figure 6.6: Expected application development roadmap.

7. Studies and decisions

7.1 Introduction

This chapter will cover the different aspects that have been taken into account in order to choose the technology stack used to develop the system.

7.2 Project structure

Being able to solve problems rapidly, or even minimizing such errors, it is a must in order to provide a secure and stable application. Since *test driven development* (or *behavior driven development*) was designed, more and bigger companies started incorporating such approach when developing new projects. Nowadays, testing the code that is being developed is unquestionable.

Aside from the development method used, it is indisputable to use a version control system. The VCS that takes the gold medal is *git* which is used alongside *GitHub*. From there, the applications can be structured in two forms:

- *Multirepo*. The multirepo approach means to have multiple applications in different repositories. The main benefits of such approach are the fact that teams can separately work in the repository while at the same time the repository is kept smaller and cleaner.

- *Monorepo*. The monorepo approach is the opposite of the multirepo: all the applications are kept in the same repository. Such approach allows maintaining build and deployment patterns altogether. However, application versioning may be harder.

Since all the applications are build using the same language (*TypeScript*) and the same package manager (*npm*), the monorepo approach has been used.

7.2.1 TDD and CI

The test driven development approach has been the method to ensure that what was being written would do as expected, even when new features are added. Both client and server applications include unit and end-to-end (*e2e*) tests.

Unit tests allow the developer to test a feature by isolating all its dependencies. For instance, each method of a concrete service from the backend has it own unit tests which completely mock its dependencies and ensure the behaviour is as expected. On the other hand, the e2e contexts are more complex, since they test, for example, one of the different use cases. An example would be to simulate the behaviour of the user in a client app and expect the DOM to properly change.

With testing, comes continuous integration which allows the code to be constantly tested. By constantly checking the newly added code, future errors can be prevented and *post-deployment* breaks or bugs can be reduced. The chosen

continuous integration platform is *GitHub Actions*[1]

7.2.2 Nx

There exist many systems or libraries that allow the development team to maintain a healthy and scalable monorepo (examples are *Bazel* or *Pants*). In this system, since the language is the same for both web and server applications, the monorepo build framework used that has been chosen is *Nx*. As stated in their repository: *Nx is a smart and extensible build framework to help you architect, test, and build at any scale*.

Figure 7.1: Nx logo

Using such tool, managing the applications becomes extremely easy to manage both JavaScript and TypeScript monorepos. Other key features are:

- The different applications have been *generated* using the plugins also developed and maintained by the team behind Nx.
- The fast creation of shared libraries.
- Testing and building only the affected code. Therefore, in unit testing only the files that have been changed, reduces the amount of tests to be done on each push.
- All the created libraries and applications include all the dependencies, scripts and tools to fast serve, test, build and deploy. Additionally, it can *lint* and format the projects.
- Provides a rich plugin ecosystem if some utilities are not included in the core package.

More specifically, it has great support with *NextJS*, which is the front-end framework used for the web apps. Being able to easily scale, maintain and test the front-end applications is a crucial requisite in this system. Such tool and its excellent documentation have simplified all aspects for setting up the different apps.

After using Nx, the application folder structure is as follows:

```
hubbl
├── apps
│   ├── api
│   ├── client
│   ├── client-e2e
```

[1]The *workflow* files of the platform can be found in the `.github/workflows` folder of the repository.

```
│   ├── core
│   ├── core-e2e
│   ├── landing
│   └── landing-e2e
├── libs
│   ├── data-access
│   │   ├── api
│   │   └── contexts
│   ├── shared
│   │   ├── models
│   │   └── types
│   ├── ui
│   │   └── components
│   └── utils
│       └── src
└── tools
    └── generators
        └── component
```

There are other files in the root folder of the repository, yet they have been excluded as the main folders are `apps` and `libs`. On the one hand, the `apps` folder contains the source code and additional files to build and compile each one of them. The client apps have two folders: the source code folders (`core`, `landing` and `client`) and their respective e2e test folders (`core-e2e`, `landing-e2e` and `client-e2e`). On the other hand, the folder `libs` contains all the libraries which do not belong to an application, but rather used in many of them. As the Nx documentation states, the `libs` folder should contain: *Libs contain services, components, utilities, etc. They have well-defined public API.* The front-end applications have been generated using the `@nrwl/next` utility, while the back end has been generated with `@nrwl/express`. The libraries have been created with the `@nx/workspace` which provides a lot of utils for the overall workspace [5]. Finally, Nx provides an utilities' library that can help the team with common tasks. Such tools are kept in the `tools` folder. In this case, there is only one tool, `component`, whih is a generator that when run generates the basic component structured used in `libs/ui/components`.

7.3 Technology stack

This section will cover the different technologies that are specifically used to develop the different applications.

7.3.1 TypeScript

These days JavaScript is the most used language around the world for many reasons. Nevertheless, due to the fact of not having types, it makes some applica-

tions harder to debug and more error-prone. That is one of the reasons why Microsoft developed TypeScript, which is a strict syntactical superset of JavaScript. The code written in TypeScript is transcompiled to JavaScript at build time.

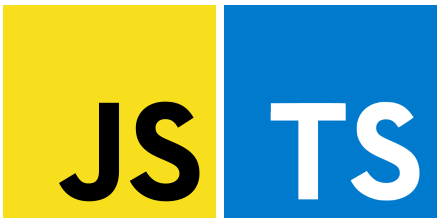


Figure 7.2: JavaScript and TypeScript logos

By using TypeScript we provide a highly productive environment when developing the different apps. Not only reduces the amount of painful bugs due to type errors, but also provides all the benefits of the ECMA script.

7.3.2 Front end - Web applications

Nowadays, there are many JavaScript front-end frameworks all with their benefits and disadvantages. Angular, React, Vue and Svelte are the most known and used frameworks. Since I am most experienced with React and NextJS is built upon React, it has been the chosen option. Furthermore, on October 21st of 2021, the NextJS team released its 12th version which included a Rust compiler that makes rebuilds and fast refreshes[2] extremely faster.

7.3.2.1 NextJS

As stated before, NextJS has been chosen as the front-end framework to build the applications with. This framework is built on top of Node.js which enables React based web application functionalities such as server-side rendering (SSR) and static websites. In the NextJS home page, they provide the following description: *Next.js gives you the best developer experience with all the features you need for production: hybrid static & server rendering, TypeScript support, smart bundling, route pre-fetching, and more. No config needed* [6].

Figure 7.3: NextJS logo

A NextJS project has to follow a certain structure. The NextJS bundler will create a new page for the application for each file that is created inside the `pages` folder. For example, in the `core` application, there are the following pages[3]:

[2]The fast refresh is an utility provided by JavaScript bundlers which checks for the changes that have been made with the current bundle, and rebundles only the changed code, allowing the developer to instantly see the changes in the development environment.

[3]Not all the folders have been included, as there are pages nested within other pages; it is a reduced folder structure.

```
apps/core/pages
├── pages
│   └── index.tsx
├── 404
├── _app.tsx
├── auth
│   ├── signup
│   │   └── index.tsx
│   └── login
│       └── index.tsx
├── clients
│   └── index.tsx
├── dashboard
│   └── index.tsx
├── _document.tsx
├── events
│   └── index.tsx
├── index.tsx
├── settings
│   └── index.tsx
├── trainers
│   └── index.tsx
├── virtual-gyms
│   └── index.tsx
└── workers
    └── index.tsx
```

In this case, instead of defining files, I opted for the `<folder-name>/index.tsx` structure. NextJS will check for each folder and when it finds a file with the `index.tsx`[4], generates the logic required to display the page in the URL of `<folder-name>`. If a folder is nested within another folder, as it is the case of the `auth` folder, the URL would be the same, but nested> `auth/signup` and `auth/login`. Aside from folders, there are special files which are specially treated by the NextJS compiler. For example, the `_app.tsx` is the entry point of the application and the content of it will be the parent of each page[5].

One of the other important features NextJS provides is `code-splitting` [7]. Code splitting consists in splitting the content provided by the server to the chrome in different bundles. Therfore, when the user loads the website, instead of providing all HTML, CSS and JS files, it only recieves what is required. In React applications this is extremely handy, since the React content is provided as a single gigantic JavaScript file.

[4] The files can end with `jsx` (JavaScript React file), `js` (JavaScript file) or `tsx` (TypeScript React file).

[5] This is very useful for SEO purposes, as an example.

7.3.2.2 Mui

Once the web framework have been decided, the applications need to share a common user interface to provide the user experience of using the same product. Keeping such similarities can be difficult if the enterprise or company does not have a well-defined design system. Therefore, creating my own design system was out of scope, and a components' library, as they are called, was needed. Again, there are multiple libraries to be used, yet there is one that shines above all others, which is the `mui` library. The library is open-source and maintained by `mui-org` [8].

This library is based on the Google Material design system. However, it has an extremely powerful API to define your own design system, without the need of starting from 0. The developer has the possibility to inject the customised theme to all the components of the API included in the library. Furthermore, it includes an icon pack, which are the same as in the Google Material.

Figure 7.4: Mui library's logo

7.3.3 Back end - Server application

For the backend application, many languages can be used. However, since NodeJS provides a runtime environment for JavaScript, which allows building applications outside the browser. So, with the help on Nx, the backend is also written in TypeScript.

7.3.3.1 Database

The application has many relationships, which would be harder to maintain in a NoSQL database. Therefore, using a relational database is a better solution. At the first glance, I opted for the MySQL database since it is one of the most used SQL databases, nowadays. However, I have more experience with PostgreSQL and I had no issues with it. In the end, the chosen database has been PostgreSQL [9].

Figure 7.5: PostgreSQL's logo

As explained before the database is run in a docker container. Viewing the data in the terminal is not an issue, yet pgAdmin is a graphic user interface which simplifies such visualisation. It is the equivalent of MySQL Workbench for PostgreSQL. With such tool, schemas in general can be simply viewed and edited. Additionally, the JetBrains DataGrip application has been also used, which has allowed to generate the ERD once the database was defined.

7.3.3.2 ExpressJS

ExpressJS is one of the many back-end frameworks that exist in the JavaScript world, nowadays. It is a minimal and flexible framework which provides a fast methodology to develop APIs. It is considered the most popular back-end framework of JavaScript, up to a point that other back-end meta-frameworks are ExpressJS based (an example is NestJS). As stated in their landing page, ExpressJs is a: *Fast, unopinionated, minimalist web framework for Node.js* [10].

Figure 7.6: ExpressJS's logo

7.3.3.3 TypeORM

Finally, the TypeORM library is used to provide a bidirectional connection from the server to the database. TypeORM is a TypeScript ORM (*object relational mapper*) whose goal is to provide an agnostic model-database mapping. By using TypeScript decorators and defining a connection, TypeORM will generate the entities or documents specified, depending on the database to which is connected. Such agnosticism allows to re-use the models with different databases [11].

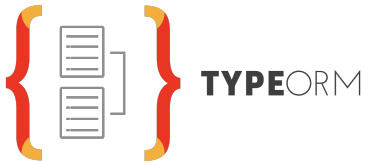


Figure 7.7: TypeORM's logo

In conjunction with Nx, the models that are used to generate the entities and their relations, can be used in the front-end applications. Therefore, such models can be shared and easily maintained.

7.3.4 Tests

Last but not least, the test stack also had to be defined. In this case, the provided frameworks for testing by Nx were kept, since they are some of the most common testing libraries nowadays.

7.3.4.1 Unit testing - Jest

Both `@nrwl/express` and `@nrwl/next` generators set up the application with Jest. Jest is an open-source unit and mock testing framework developed and maintained by Facebook. It is one of the most popular libraries nowadays, since it is extremely powerful and simple. It offers a powerful CLI tool to control the tests which has an interactive mode that allows the developer only to run *affected* tests[6]. Other benefits of Jest are that require zero configuration and ensures isolation between tests. Isolation is extremely important when running tests, to ensure that each test does not influence other's results. As stated in their home page: *Jest is a delightful JavaScript Testing Framework with a focus in simplicity* [12].

Figure 7.8: Jest's logo

Therefore, Jest is used in all the applications to unit tests the services, controllers, components and so on from all the applications. Furthermore, since there is a GitHub Action, CI can be easliy acomplished.

7.3.4.2 Unit testing - `@testing-library`

Aside from unit testing API functions and so on, it is important to also test the different React components, ensuring they behave as expected. Again, there are multiple choices to develop component unit tests, yet the solution chosen is the one which is installed with `@nrlw/next`: `@testing-library`. This package provides an API to easily test React components. It is a very light library which uses functions on top of `react-dom` and `react-dom/test-utils`.

Figure 7.9: Testing library's logo

[6] Nx also takes care of running only affected tests.

7.3.4.3 Client e2e testing - Cypress

Cypress is more than a tool for testing. It provides a graphical user interface to see what is being tested, where it fails and other features. It is an extremely powerful tool for UI testing, which has always been a difficult subject. As stated in their website, cypress enables you to: *write faster, easier and more reliable tests* [14].

Figure 7.10: Cypress's logo

In order to use cypress, the project needs a specific folder structure which can be a bit confusing to set up. However, the `@nrlw/next` plugin already sets up this environment, which is kept in the `*-e2e` folders[7].

7.3.4.4 Server e2e testing - Jest and Supertest

Finally, the libraries that are installed to develop end-to-end test for express, in `@nrwl/express` are: Jest and Supertest. The Jest library has already been covered. The Supertest is an HTTP assertions library that allows the developer to create Node.js HTTP tests. It uses the SuperAgent library, which is an HTTP client for Node.js. Using Supertest is one of the most common methods to develop such tests.

[7] In the system case, the different cypress environments are in: `client-e2e`, `core-e2e` and `landing-e2e`.

8. Analysis and system design

8.1 Use case diagram

8.2 Database diagram

8.2.1 Introduction

In a previous section, it has been explained that the DataGrip application has been used to create the database diagram. The process to create such diagram has been:

1. Create the entities with TypeORM. The synchronization of TypeORM with the empty database, will create all the tables, sequences, indexes and so on that have been defined. Therefore, the use of SQL is minimum and can easily be debugged, if needed.
2. Set up a connection to the dockerized database with DataGrip.
3. Generate the ERD with the detected tables.

Such tool allows faster diagrams visualisation and avoids the pain of having to keep ERD diagrams manually updated on any minor change,

8.2.2 Diagram

The resulting diagram is:

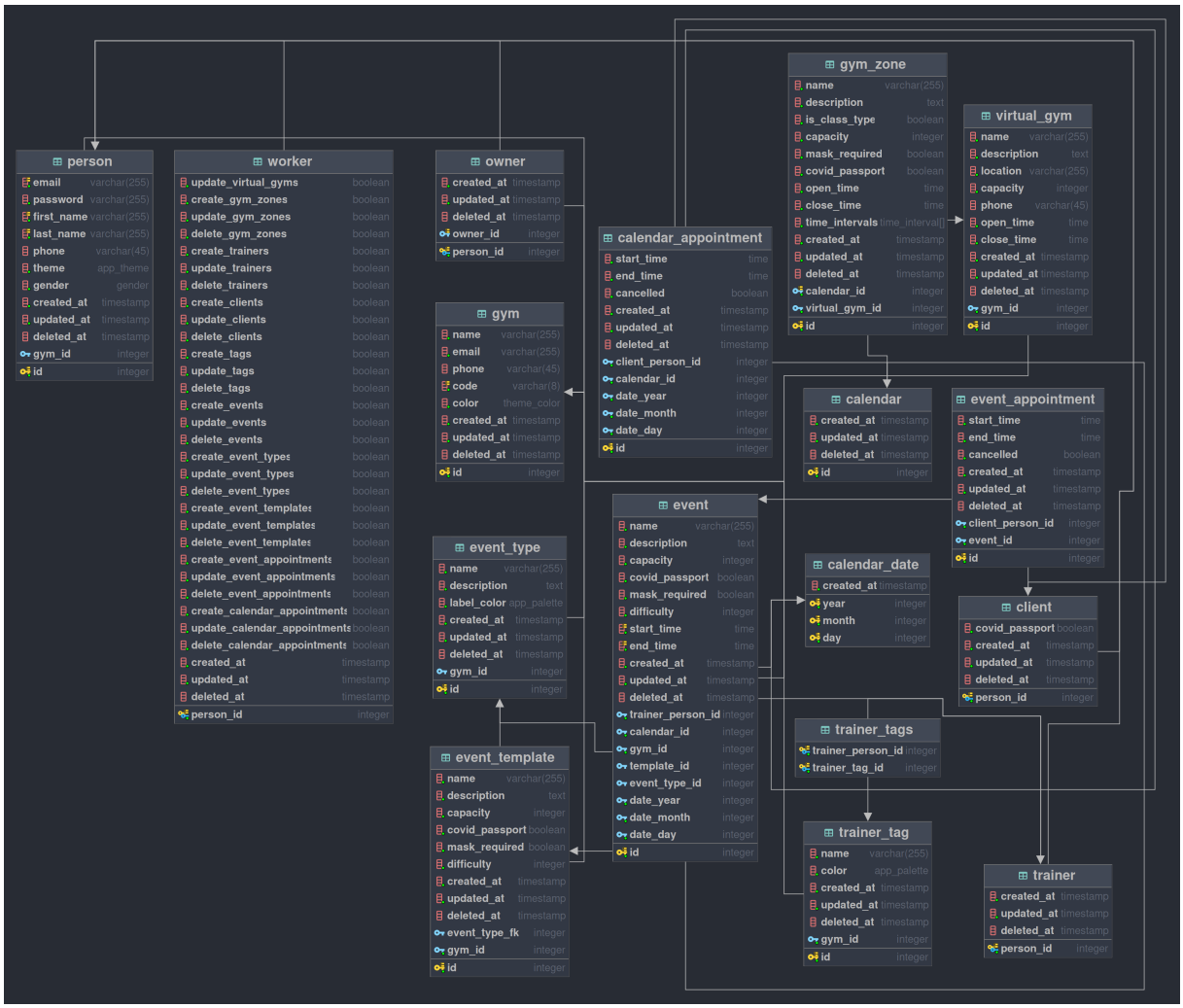


Figure 8.1: Database diagram

8.2.2.1 Introduction

In the following sections each table will be explained and a brief description of their fields will be given. Each field will be represented using the following schema:

- **<field name>** (`<field type>`) [`<field constraints>`] → <field description>

There are some fields that are common to the different entities, independently of their relations. Such fields are:

- **created_at** (`timestamp`) [`DEFAULT NOW()`] → Keeps the *timestamp* at which the entity has been created.
- **updated_at** (`timestamp`) [`DEFAULT NOW()`] → Keeps the *timestamp* at which the entity has been last updated.
- **deleted_at** (`timestamp`) [] → Keeps the *timestamp* at which the entity has been dele-ted.

8.2.2.2 Person

One of the big concepts to define in the database has been how are users stored. After comparing many approaches, a multiple table inheritance has been chosen. The base table of the user's inheritance is the *Person*. The person entity will have an id which will be the primary key of the entities that inherit from the table. Such entity contains the mandatory data each user must have:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → This is the *primary key* of the entity.
- **email** (`varchar(255)`) [`NOT NULL`] → Stores the email of the user. The email is indexed by a *unique index* (`person-email-idx`).
- **password** (`varchar(255)`) [`NOT NULL`] → Stores the password of the user which is encrypted before insert.
- **first_name** (`varchar(255)`) [`NOT NULL`] → Stores the first name of the user. This field is indexed for faster queries by first name (`person-first-name-idx`).
- **last_name** (`varchar(255)`) [`NOT NULL`] → Stores the last name of the user. This field is indexed for faster queries by last name (`person-last-name-idx`).
- **theme** (`app_theme`) [`DEFAULT "LIGHT"`] → Stores the theme used by the person in the different applications[1].
- **gender** (`char`) [`NOT NULL`] → Stores the gender of the person.
- **gym_id** (`integer`) [`NOT NULL`] → Stores the id of the `Gym` entity to which the user belongs.

8.2.2.3 Owner

The *Owner* entity is the person who has the most permissions. Therefore, instead of having all the permissions which are defined in the *Worker* entity set to `true`, the *Owner*'s are kept in a different table. Additionally, an owner will have more permissions than the ones stored in the *Worker* entity. The fields of the table are:

- **person_id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → References an id of the *Person* entity.
- **onwer_id** (`integer`) [] → References the id of the *Gym* the owner owns.

8.2.2.4 Worker

The next person type is the *Worker*. A worker has different permissions set by the owner when created. The *Worker* entity basically contains `boolean` fields which allow the system to know what permissions have been given to the worker. The permissions of the user are set using the following rule: `<operation>_<on>`. For example, the permission of *updating* the gym zones is `update_gym_zone`.

The fields of the table are

[1] This field is an enum defined in the library `@hubbl/shared/enums`.

- **person_id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → References an id of the *Person* entity.
- **update_virtual_gym** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *VirtualGym*.
- **create_gym_zones** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create a *GymZone*.
- **update_gym_zones** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *GymZone*.
- **delete_gym_zones** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete a *GymZone*.
- **create_trainers** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create a *Trainer*.
- **update_trainers** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *Trainer*.
- **delete_trainers** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete a *Trainer*.
- **create_clients** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create a
  Client.
- **update_clients** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *Client*.
- **delete_clients** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete a
  Client.
- **create_tags** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create a *TrainerTag*.
- **update_tags** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *TrainerTag*.
- **delete_tags** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete a *TrainerTag*.
- **create_events** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create an *Event*.
- **update_events** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update an *Event*.
- **delete_events** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete an *Event*.

- **create_event_types** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create an *EventType*.
- **update_event_types** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update an *EventType*.
- **delete_event_types** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete an *EventType*.
- **create_event_templates** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create an *EventTemplate*.
- **update_event_templates** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update an *EventTemplate*.
- **delete_event_templates** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to
delete an *EventTemplate*.
- **create_event_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create an *EventAppointment*.
- **update_event_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update an *EventAppointment*.
- **delete_event_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to
delete an *EventAppointment*.
- **create_calendar_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to create a *CalendarAppointment*.
- **update_calendar_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to update a *CalendarAppointment*.
- **delete_calendar_appointments** (`boolean`) [`DEFAULT FALSE NOT NULL`] → Permission to delete a *CalendarAppointment*.
- **manager_id_fk** (`integer`) [`NOT NULL`] → References the manager (*Owner*) of the *Worker*.

8.2.2.5 Trainer

The trainer table is very simple since it has no interaction with the system. It is a subset of the properties of a worker.

- **person_id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → References an id of the *Person* entity.

8.2.2.6 Client

Client is a WIP entity.

- **person_id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → References an id of the *Person* entity.
- **covid_passport** (`boolean`) [`NOT NULL DEFAULT FALSE`] → Allows the gym owners and workers to know if the user has registered the covid passport.

8.2.2.7 Gym

The *Gym* entity is relates to the company, not the infrastructure of a gym. Therefore, it works as a bridge entity that knows the preferences of the gym, such as the interface, gym location and other data. These properties can only be modified by an owner person. The table consists of the following fields:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *Gym*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *Gym*, which will be displayed to the client and workers.
- **email** (`varchar(255)`) [`NOT NULL`] → Contact email of the *Gym*.
- **phone** (`varchar(255)`) [`NOT NULL`] → Contact phone of the *Gym*.
- **code** (`varchar(8)`) [`NOT NULL`] → Unique *Gym* code used to identify the gym so the *Client* can use it when registering to the application.
- **color** (`theme_color`) [`DEFAULT "#2196F3" NOT NULL`] → Primary color of the gym which is the same with all the users[2].

8.2.2.8 VirtualGym

Once the *Gym* has been defined, the company may have multiple different infrastructures, for example, if it is a franchise. These virtual gyms are stored in the *VirtualGym* table. The virtual gym will have different constraints which can change if the workers or owners do change them. The fields of the *VirtualGym* table are:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *VirtualGym*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *VirtualGym*.
- **description** (`text`) [`NOT NULL`] → Description of the *VirtualGym*.
- **location** (`varchar(255)`) [`NOT NULL`] → Location of the *VirtualGym*.
- **capacity** (`integer`) [`NOT NULL`] → Maximum capacity of the *VirtualGym*.
- **phone** (`integer`) [] → Optional phone of the *VirtualGym*.
- **open_time** (`time`) [`NOT NULL`] → Time at which the *VirtualGym* opens.
- **close_time** (`time`) [`NOT NULL`] → Time at which the *VirtualGym* closes.
- **gym_id** (`integer`) [`NOT NULL`] → *Gym* to which the *VirtualGym* belongs.

[2] This field is an enum defined in the library `@hubbl/shared/enums`.

8.2.2.9 GymZone

After having created the *VirtualGym*, now gym zones can be created for the different virtual gyms of a gym. There are two types of zones: class and non-class zones. The class zones will have a schedule with *Event*'s create by the gym personnel, while non-class zones will have a schedule without events. At the same time, it has the constraints set by the owner and workers, similar to the *VirtualGym*.

The columns of the entity are:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *GymZone*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *GymZone*.
- **description** (`text`) [`NOT NULL`] → Description of the *GymZone*.
- **is_class_type** (`boolean`) [`DEFAULT FALSE`] → Whether it is a class or non-class zone.
- **capacity** (`integer`) [`NOT NULL`] → Maximum capacity of the *GymZone*.
- **mask_required** (`boolean`) [`NOT NULL DEFAULT TRUE`] → Whether wearing the mask is mandatory to access the *GymZone*.
- **covid_passport** (`boolean`) [`NOT NULL DEFAULT TRUE`] → Whether having registered the covid passport is mandatory to access the *GymZone*.
- **open_time** (`time`) [`NOT NULL`] → Time at which the *GymZone* opens.
- **close_time** (`time`) [`NOT NULL`] → Time at which the *GymZone* closes.
- **time_intervals** (`time_interval[]`) [`DEFAULT 30, 60, 90, 120`] → This field will only be used if the *GymZone* is a non-class zone. It defines the time intervals, in minutes, that the clients will be able to make the appointments.
- **calendar_id** (`integer`) [`NOT NULL`] → *Calendar* identifier of the *GymZone*.
- **virtual_gym_id** (`integer`) [`NOT NULL`] → *VirtualGym* to which the *GymZone* belongs.

8.2.2.10 Calendar

The *Calendar* entity is linked to a *GymZone* and is used to link the events and to know how many clients have made an *Appointment* during a time interval. The *Calendar* will always be linked with *Event*'s, and the *Event* entity will be used to determine if the *Appointment* is for a class or a non-class zone.

The fields of the table are:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *Calendar*.

8.2.2.11 CalendarDate

A *CalendarDate* is a simple entity that will contain the year, month and day of a date. It will be created *on demand*, meaning that each entry is unique, yet it will not exist until there is another entity that requires of it. With this approach, the system avoids the need to fill the table with hundreds of entries for each date. The fields are:

- **year** (`integer`) [`NOT NULL`] → Year of the *CalendarDate*.
- **month** (`integer`) [`NOT NULL`] → Month of the *CalendarDate*.
- **day** (`integer`) [`NOT NULL`] → Day of the *CalendarDate*.

In order to ensure each entry is unique, a *composite unique index* has been set which contains the three fields of the table.

8.2.2.12 EventType

The system allows the user to create events are of a specific type. For instance, the types can be: spinning, zumba, yoga and so on. There will be predefined *EventType*'s when the owner first creates the gym. However, in order to allow more flexibility while using the application, the system also allows the user to create custom *EventType*'s.

The *EventType* entity is composed by the following columns:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *EventType*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *EventType*.
- **description** (`text`) [`NOT NULL`] → Description for the *EventType*.
- **label_color** (`app_palette`) [`DEFAULT "#2196F3" NOT NULL`] → Color of the *EventType* when displayed.
- **gym_id** (`integer`) [`NOT NULL`] → References the id to which the *EventType* belongs.

8.2.2.13 EventTemplate

In order to reduce the repetitive process of creating *Event*'s for a gym zone, the system provides the utility of creating *EventTemplate*'s, which allow the owners or workers to create *Event*'s faster. With the *EventTemplate* utility, in order to create an *Event* becomes faster and less repetitive.

The fields of this entity are:

- **id** (`integer`) [`PRIMARY KEY AUTOINCREMENT`] → Unique id of the *EventTemplate*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *EventTemplate*.
- **description** (`text`) [`NOT NULL`] → Description for the *EventTemplate*.

- **capacity** (`integer`) [`NOT NULL`] → Optional description for the *EventTemplate*.
- **covid_passport** (`integer`) [`NOT NULL DEFAULT FALSE`] → Whether the event will require the client to have the covid passport or not.
- **mask_required** (`integer`) [`NOT NULL DEFAULT FALSE`] → Whether the event will require the client to wear a mask.
- **event_type_fk** (`integer`) [] → References the *EventType* of the *EventTemplate*.
- **gym_id** (`integer`) [`NOT NULL`] → References the *EventType* of the *EventTemplate*.

8.2.2.14 Event

Finally, the *Event* can be defined. An *Event* is a bridge table that contains: the *Trainer*, the *CalendarDate*, the *Calendar* to which is scheduled and the *EventTemplate*. All these identify the *Event*, that is what the entity to which the user will be able to create an appointment.

The table is defined as:

- **id** (`integer`) [`PRIMARY FIELD NOT NULL`] → Unique identifier of the *Event*.
- **name** (`varchar(255)`) [`NOT NULL`] → Name of the *Event*.
- **description** (`text`) [`NOT NULL`] → Description for the *Event*.
- **capacity** (`integer`) [`NOT NULL`] → Optional description for the *Event*.
- **covid_passport** (`integer`) [`NOT NULL DEFAULT FALSE`] → Whether the event will require the *Client* to have the covid passport or not.
- **mask_required** (`integer`) [`NOT NULL DEFAULT FALSE`] → Whether the event will require the *Client* to wear a mask.
- **start_time** (`time`) [`NOT NULL`] → Time at which the *Event* starts.
- **end_time** (`time`) [`NOT NULL`] → Time at which the *Event* ends.
- **trainer_person_id** (`integer`) [`NOT NULL`] → References the *Trainer* of the *Event*.
- **calendar_id** (`integer`) [`NOT NULL`] → References the *Calendar* of the *Event*.
- **template_id** (`integer`) [] → References the *EventTemplate* of the *Event*, if any.
- **event_type_id** (`integer`) [] → References the *EventType* of the *Event*.
- **date_year** (`integer`) [`NOT NULL`] → References the year of a *CalendarDate* of the *Event*.

- **date_month** (`integer`) [`NOT NULL`] → References the month of a *CalendarDate* of the *Event*.
- **date_day** (`integer`) [`NOT NULL`] → References the day of a *CalendarDate* of the *Event*.

8.2.2.15 EventAppointment and CalendarAppointment

Finally, the only requirement left is the appointments from the *Client*'s. The *Client* has to be able to book a session for a guided class, which means to an *Event*, or for a non-guided class, that is for a *Calendar*. So, the two entities represent each case respectively. They both share some attributes, such as the `startTime` and the `openTime`. However, the *EventAppointment* is related with an *Event*, as it will be an *Appointment* made to an *Event*. On the other hand, a *CalendarAppointment* would be an *Appointment* made in a time interval, chosen by the user, for a specific *GymZone*.

The fields of the *EventAppointment* entity are:

- **id** (`integer`) [`PRIMARY KEY NOT NULL`] → Unique identifier of the *EventAppointment*
- **client_person_id** (`integer`) [`NOT NULL`] → References the *Client* that has made the *EventAppointment*.
- **event_id** (`integer`) [`NOT NULL`] → References the *Event* to which the *EventAppointment* has been made.
- **start_time** (`time`) [`NOT NULL`] → Time at which the *EventAppointment* starts, which will be the same as the *Event*.
- **end_time** (`time`) [`NOT NULL`] → Time at which the *EventAppointment* ends, which will be the same as the *Event*.
- **cancelled** (`boolean`) [`NOT NULL DEFAULT FALSE`] → Used to know if the *EventAppointment* has been cancelled.

The fields of the *CalendarAppointment* entity are:

- **id** (`integer`) [`PRIMARY KEY NOT NULL`] → Unique identifier of the *CalendarAppointment*
- **client_person** (`integer`) [`NOT NULL`] → References the *Client* that has made the *CalendarAppointment*.
- **calendar_id** (`integer`) [`NOT NULL`] → References the *Calendar* to which the *CalendarAppointment* has been made.
- **start_time** (`time`) [`NOT NULL`] → Time at which the *CalendarAppointment* starts.
- **end_time** (`time`) [`NOT NULL`] → Time at which the *CalendarAppointment* ends.

- **cancelled** (`boolean`) [`NOT NULL DEFAULT FALSE`] → Used to know if the *CalendarAppointment* has been cancelled.
- **date_year** (`integer`) [`NOT NULL`] → References the year of a *CalendarDate* of the *CalendarAppointment*.
- **date_month** (`integer`) [`NOT NULL`] → References the month of a *CalendarDate* of the *CalendarAppointment*.
- **date_day** (`integer`) [`NOT NULL`] → References the day of a *CalendarDate* of the *CalendarAppointment*.

8.3 User interfaces

The design of the user interfaces has been a priority before diving into the front end development. Such interfices, provide a preliminar view of what show the application look and feel. The following figures, are the designs of the user interfaces that have been designed using the Figma software[3].
Designing every possible state of the application is far from achievable with such little time. However, since the core and client application will share much of their user interface, only the core views have been designed. When developing the client application, most of the components have been reused, modifying they behaviour as needed, since the client interactions are limited (for instance, the client sees the same dashboard page, yet the button to add a virtual gym is hidden). Needless to say, some views do not correspond exactly as they are in the application, mainly because some features have been modified or added as required. Finally, the following aspects have been considered when designing the views:

- The views should also be displayed as an owner used. This is due to the fact that the only differences between the views of an owner, a worker or a client, will be minimal. Some things will be hidden or not allowed, which has little effects to the final design.
- Each element that can be created (virtual gyms, gym zones, trainers and so on) must have their corresponding dialog or view in which such items are created, edited or deleated.
    - In case a dialog is used, it has to contain how it looks when: it is being created, it is being edited, and it is being loaded, for each of the previous states.

With these requirements, there are views modals that contain 4 images, each one for the described states.

[3] The figma application is a free software which can be found in: www.figma.com

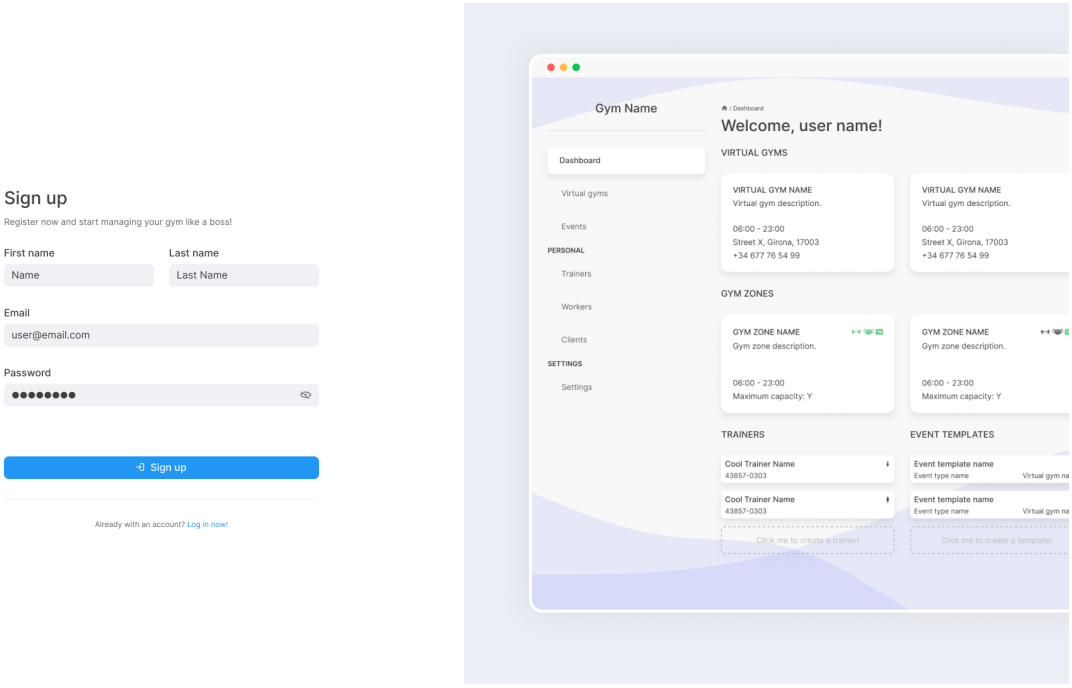


Figure 8.2: First step of the sign up page

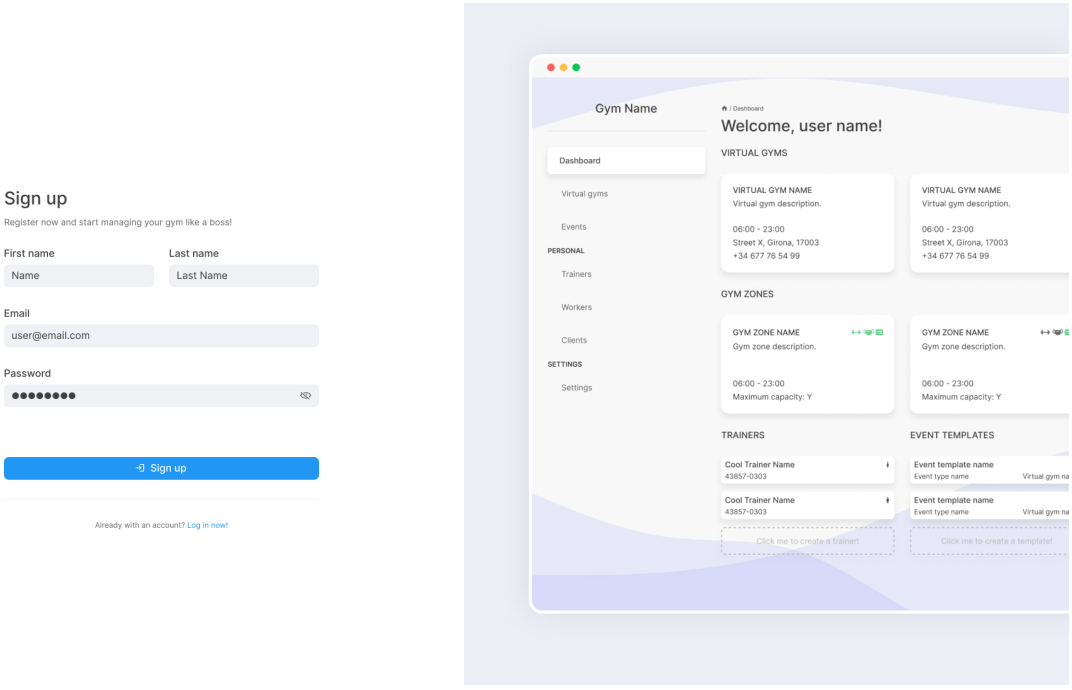


Figure 8.3: Second step of the sign up page

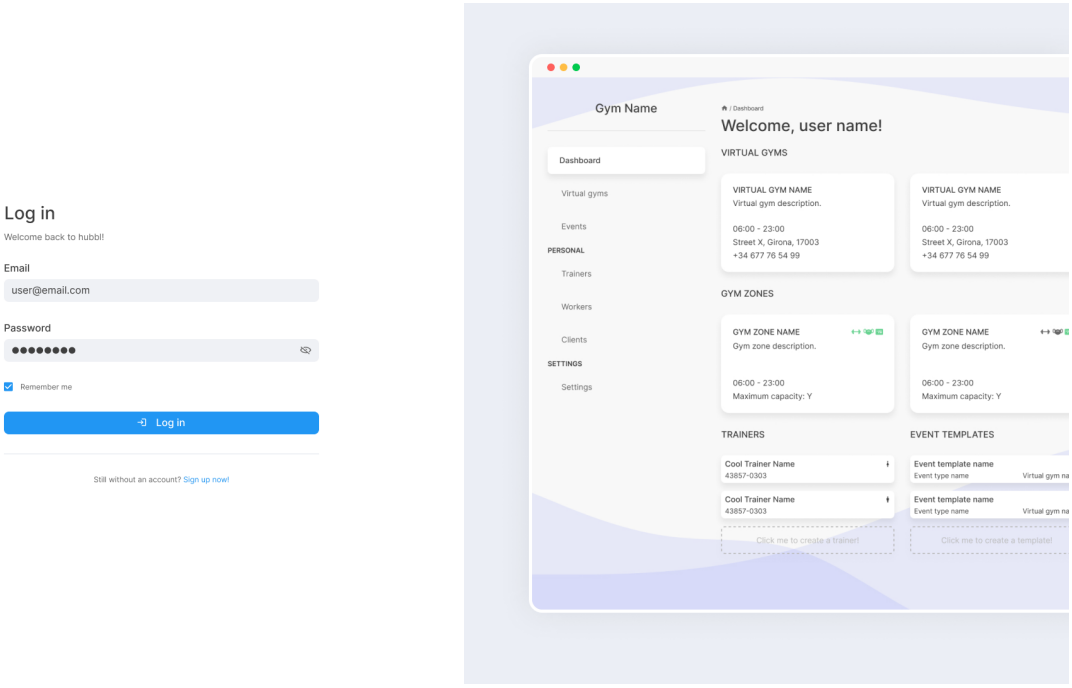


Figure 8.4: View of the login page

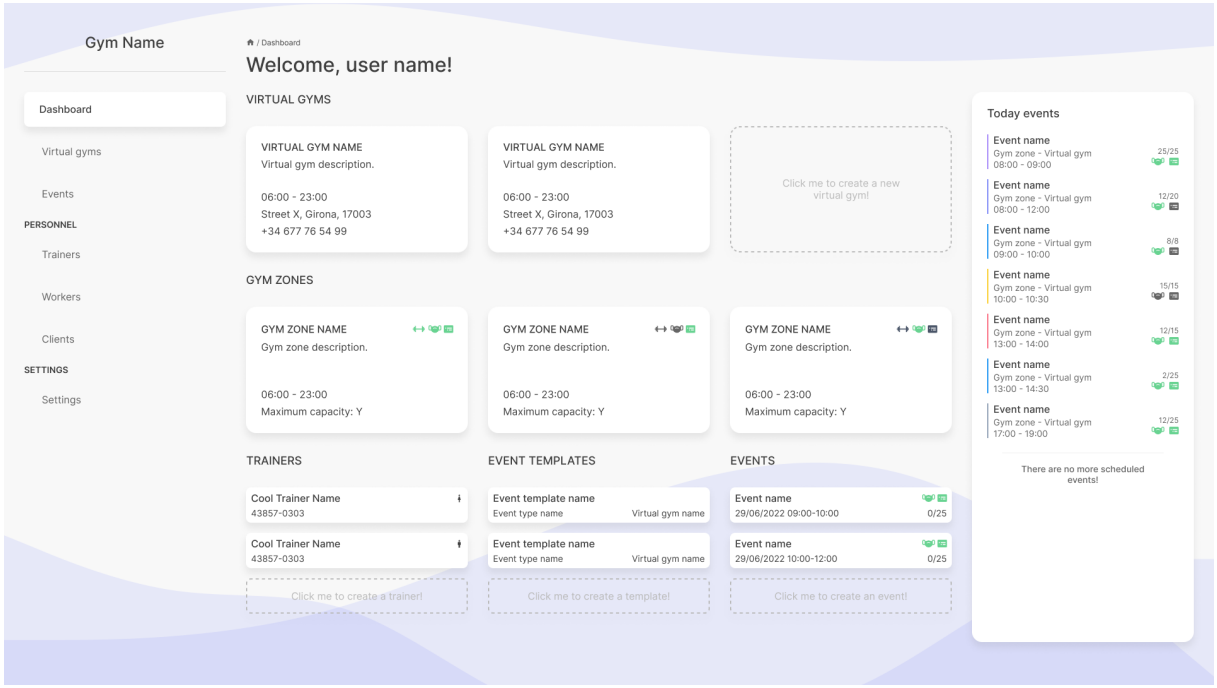


Figure 8.5: Dashboard page, displaying a summary of the gym's information

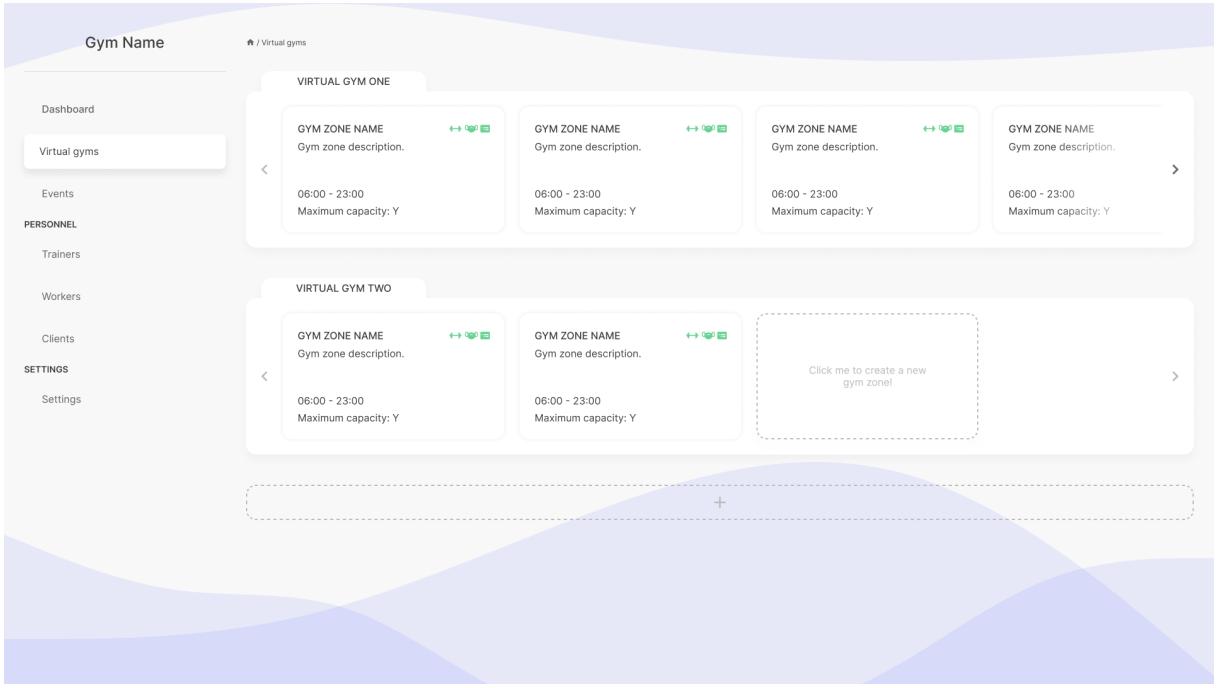


Figure 8.6: Virtual gym's page, which is accessed using the left navigation bar

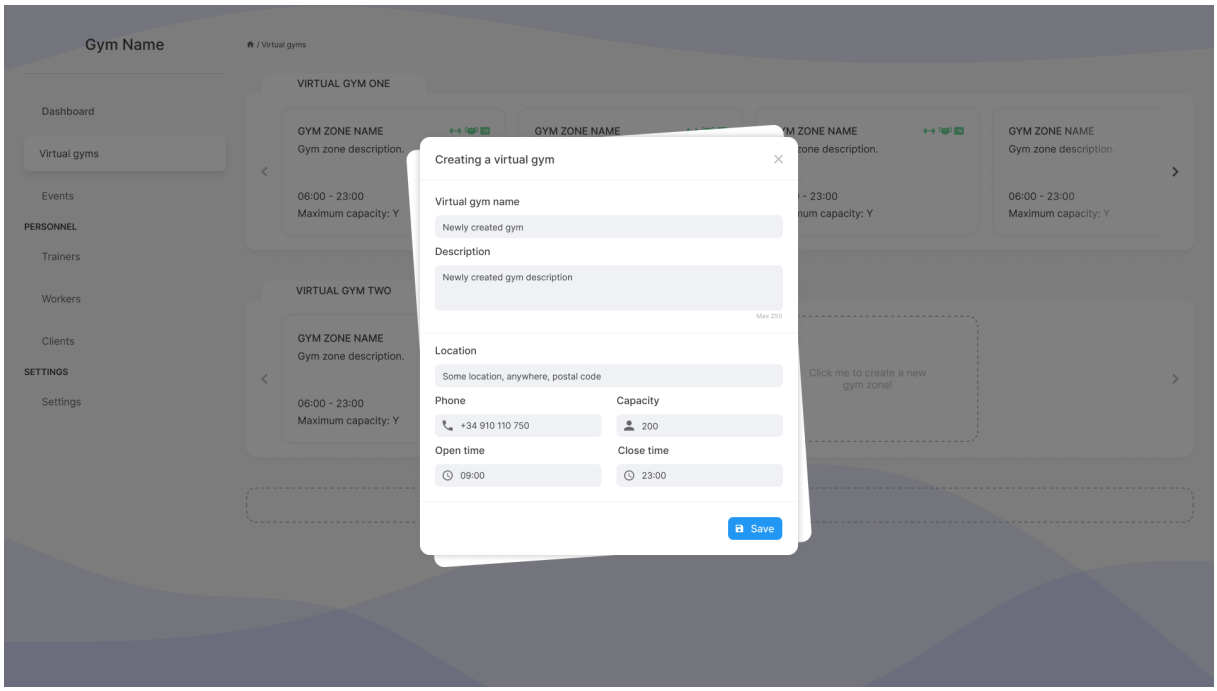


Figure 8.7: Virtual gym dialog (create state)

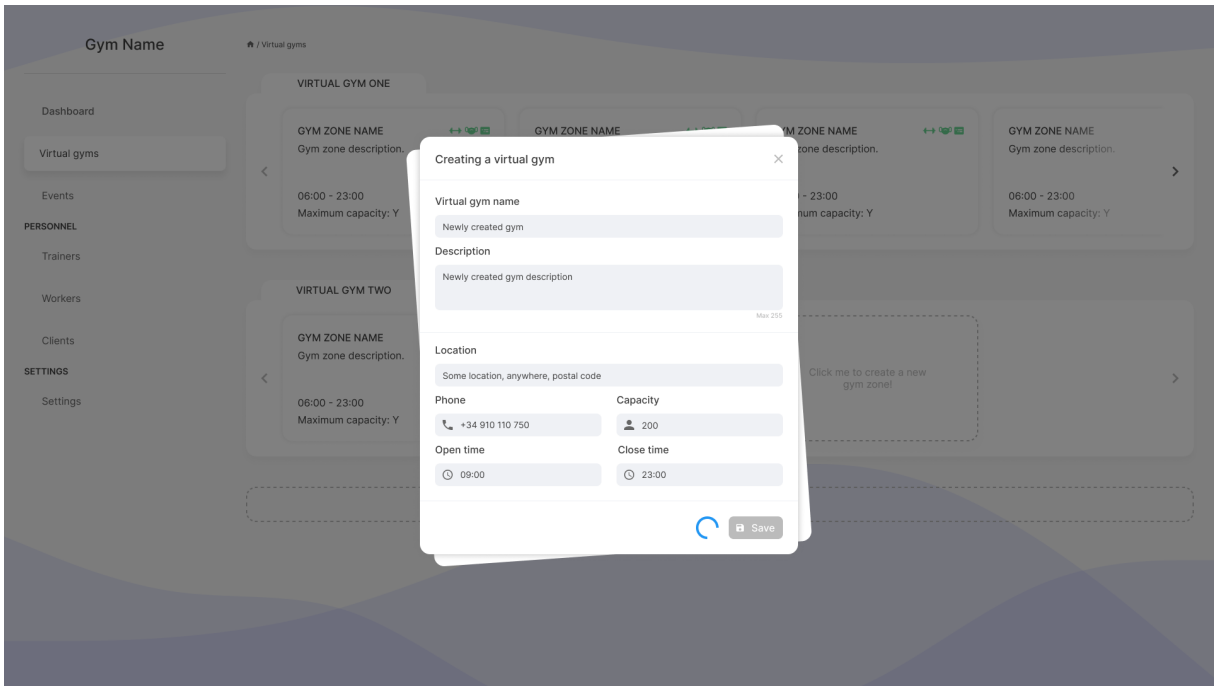


Figure 8.8: Virtual gym dialog (create-loading state)

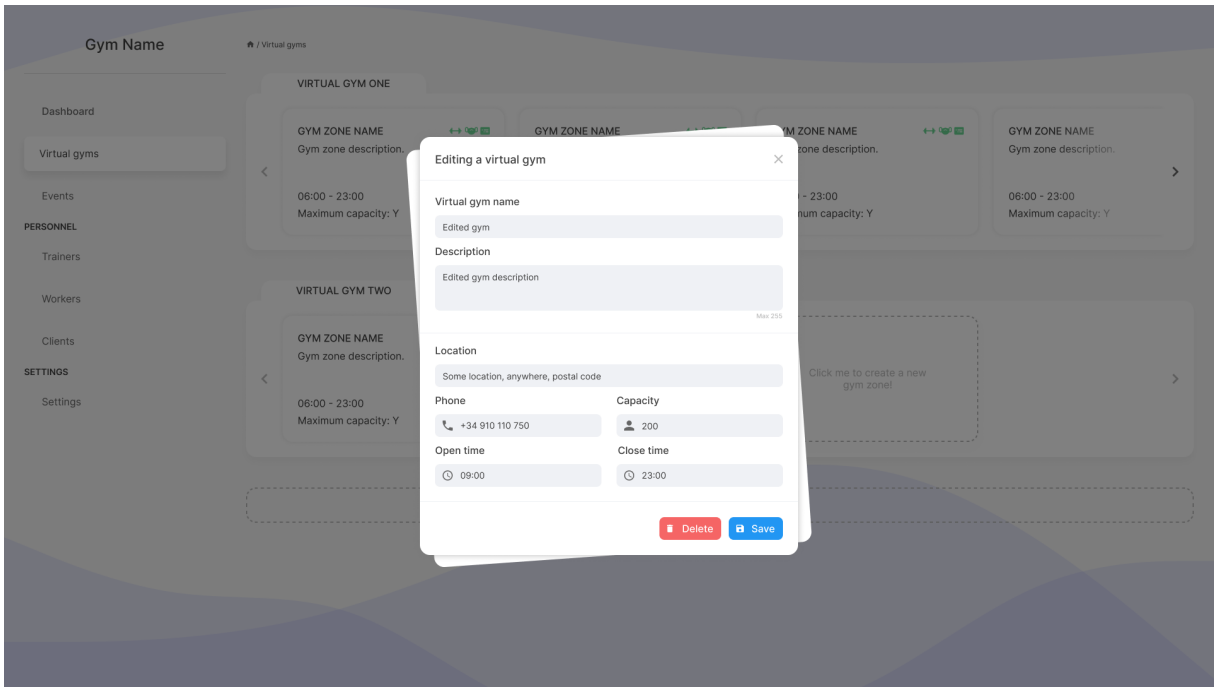


Figure 8.9: Virtual gym dialog (edit state)

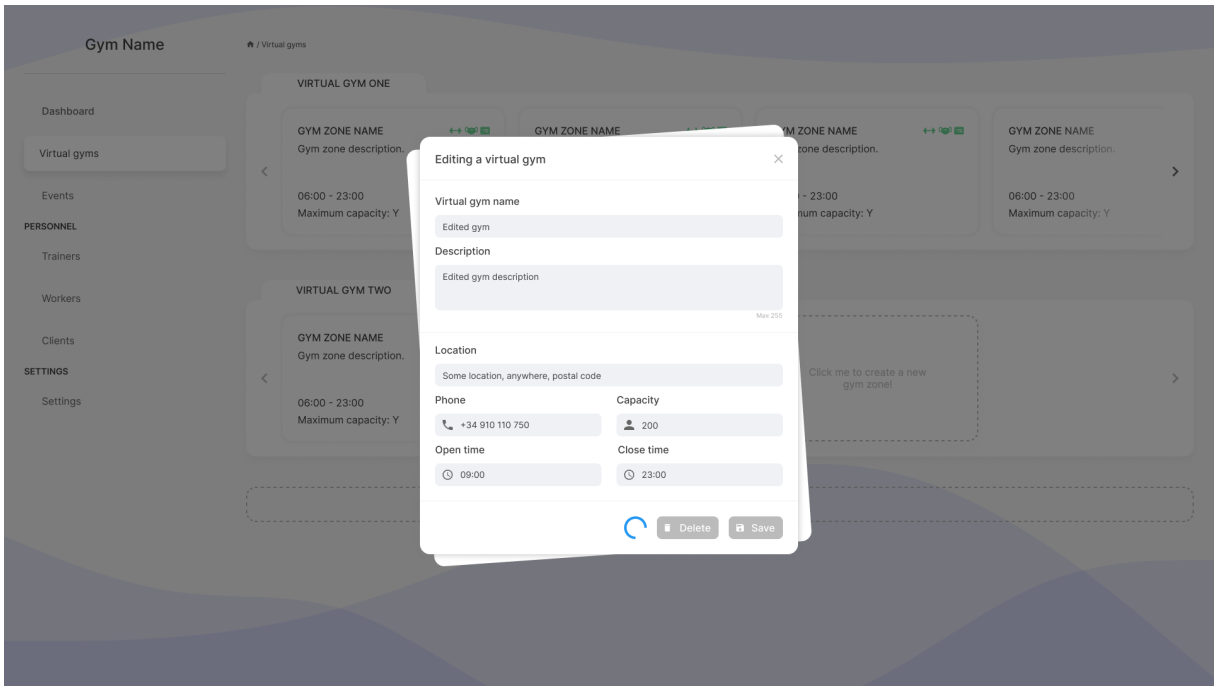


Figure 8.10: Virtual gym dialog (edit-loading state)

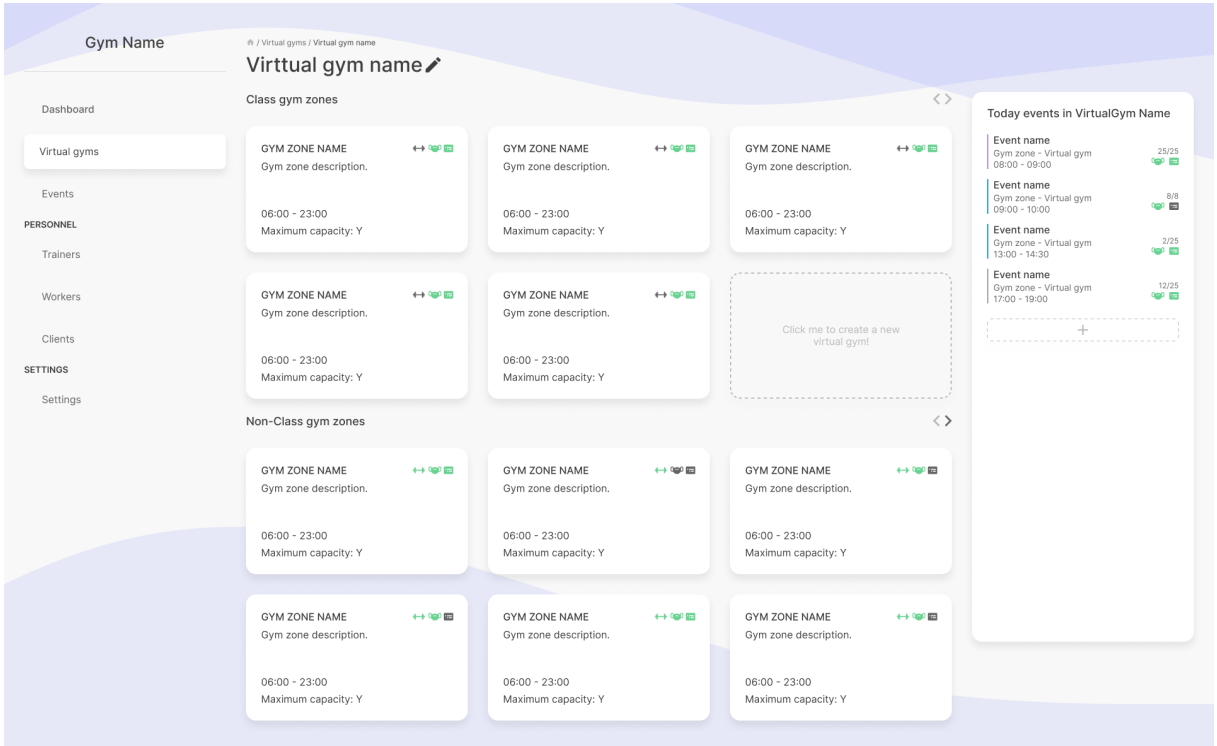


Figure 8.11: Single virtual gym view, accessed by clicking on a virtual gym

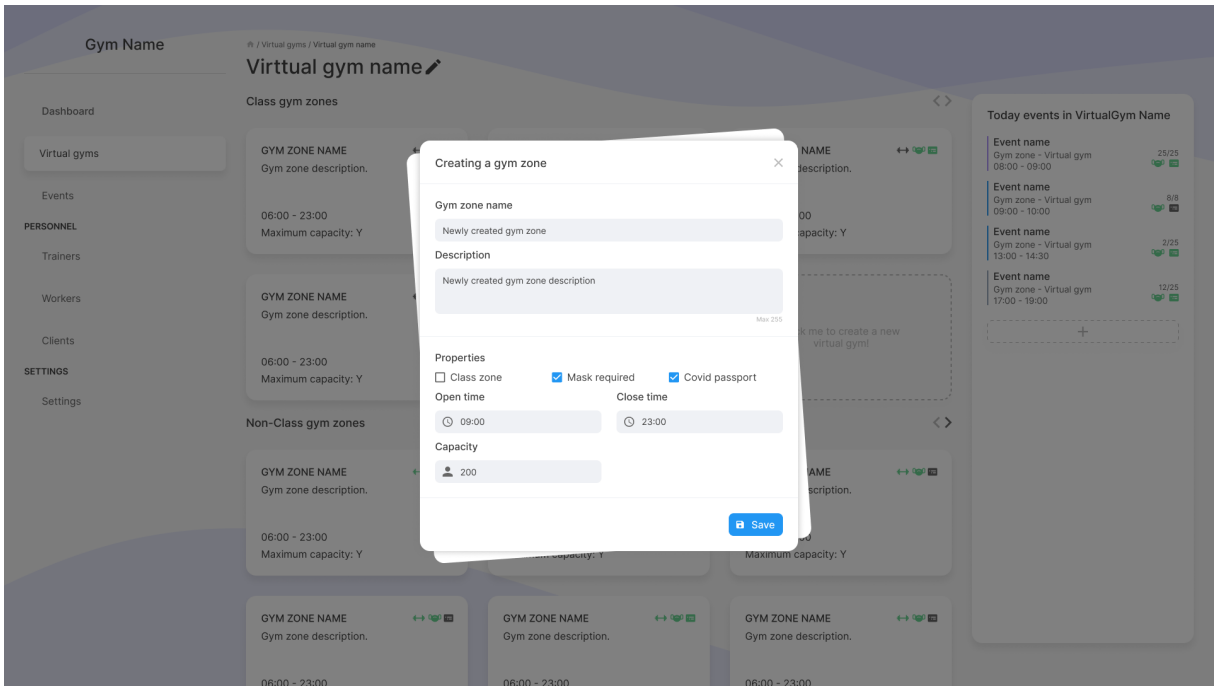


Figure 8.12: Gym zone dialog (create state)

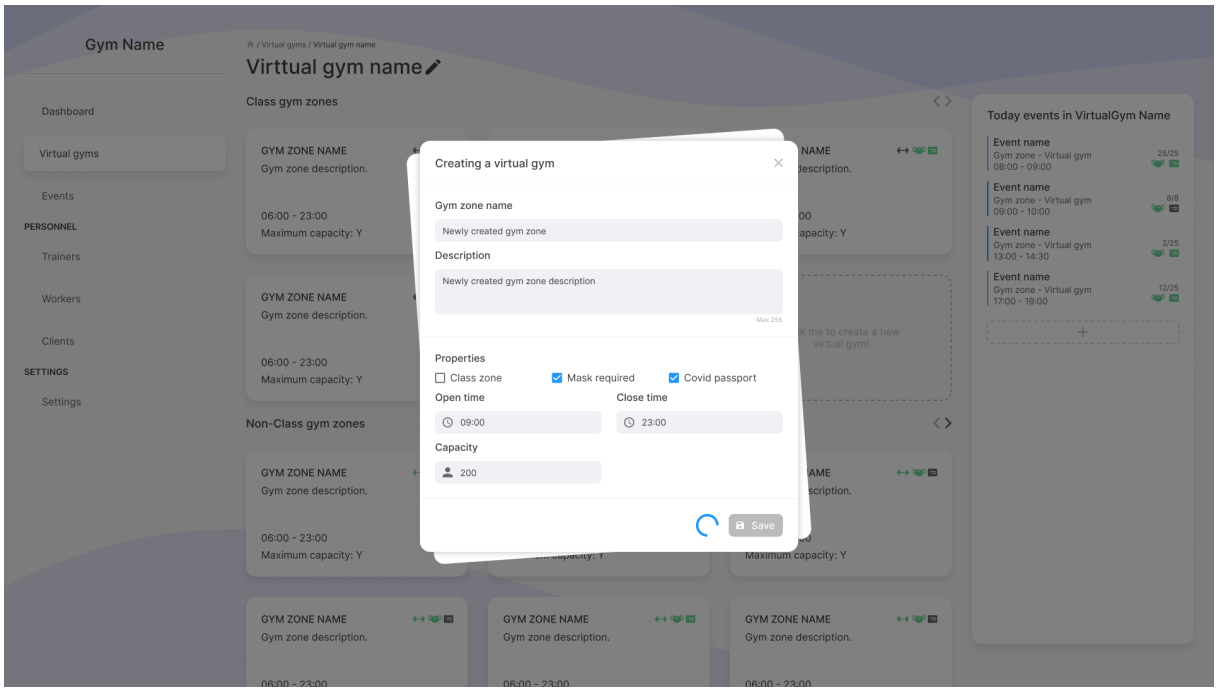


Figure 8.13: Gym zone dialog (create-loading state)

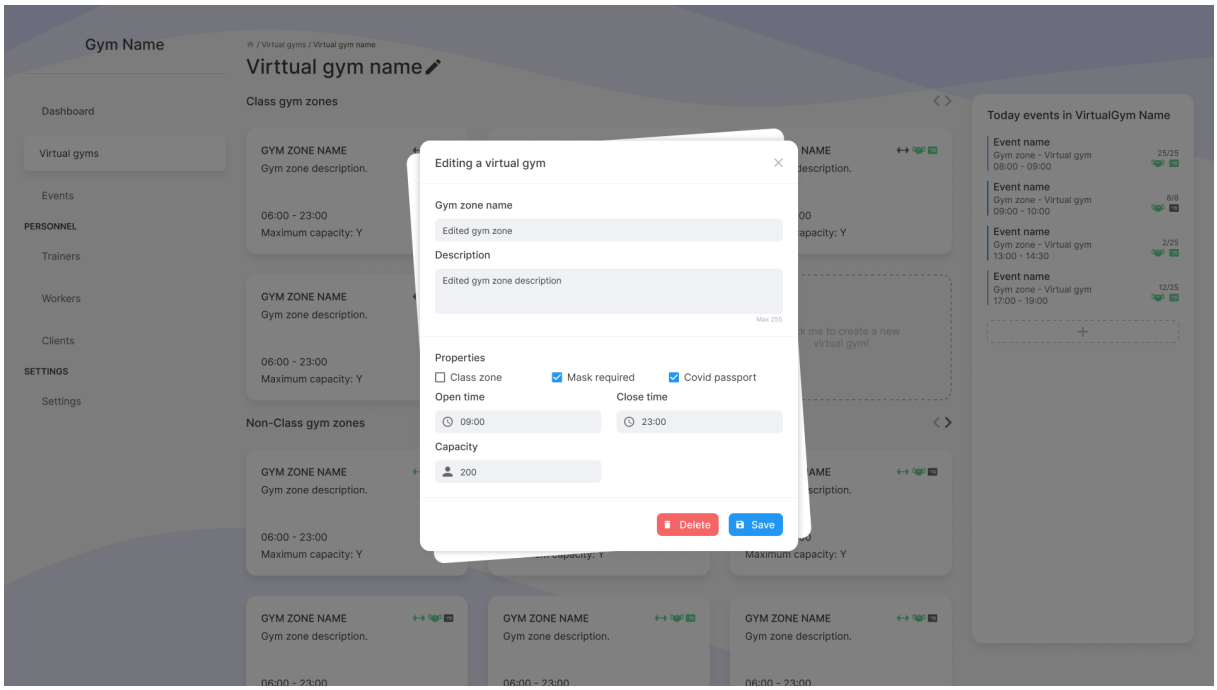


Figure 8.14: Gym zone dialog (edit state)

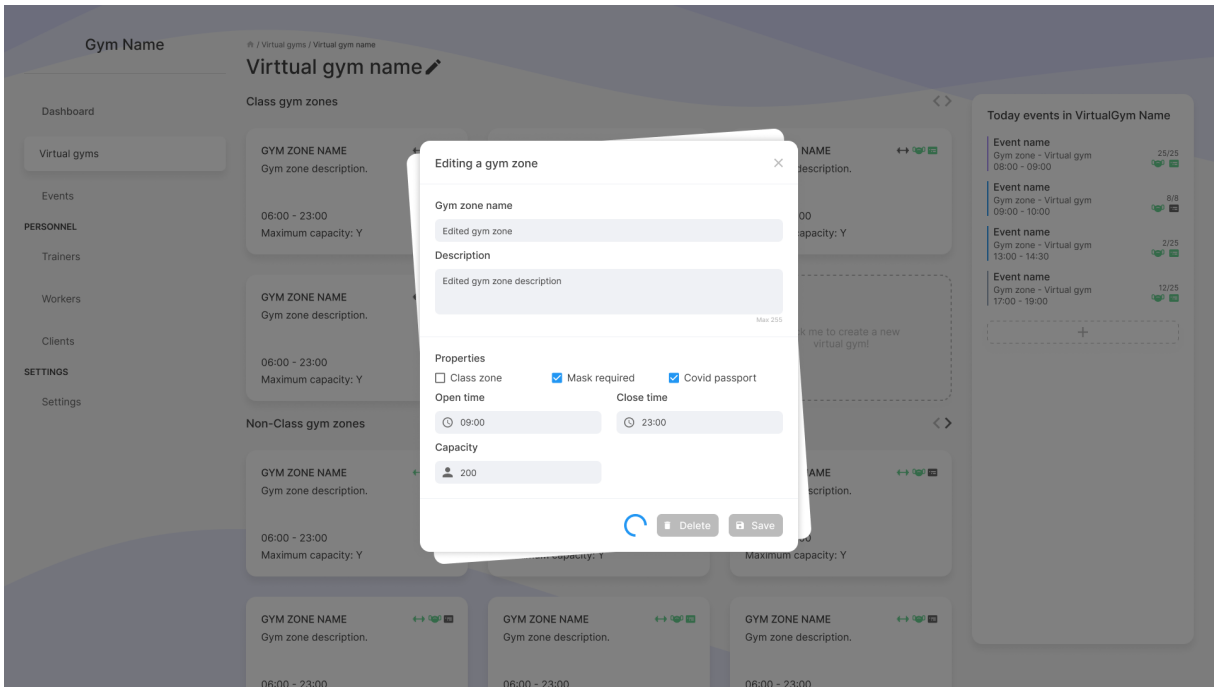


Figure 8.15: Gym zone dialog (edit-loading state)

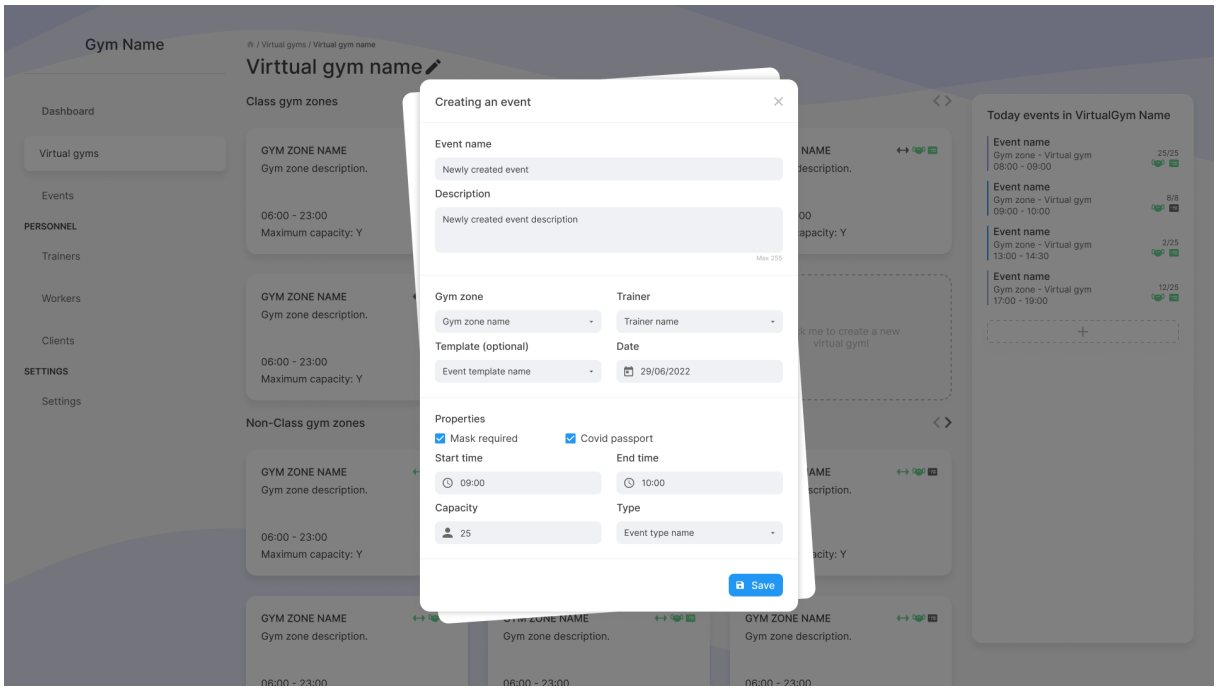


Figure 8.16: Event dialog (create state)

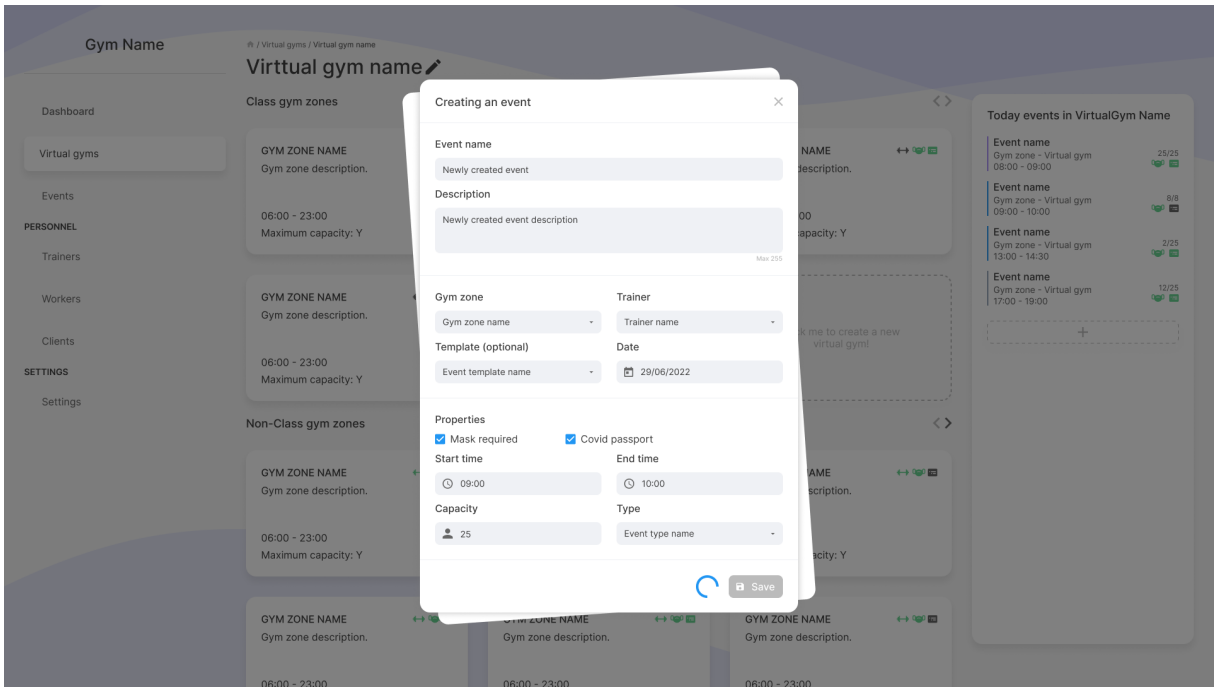


Figure 8.17: Event dialog (create-loading state)

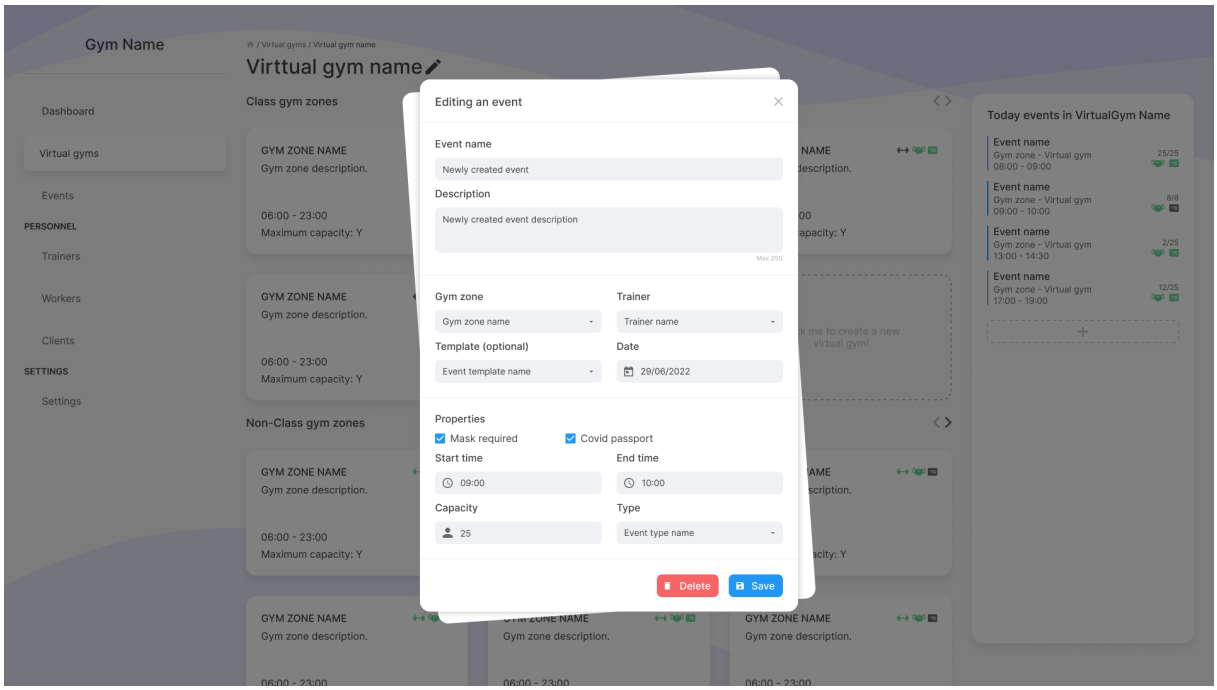


Figure 8.18: Event dialog (edit state)

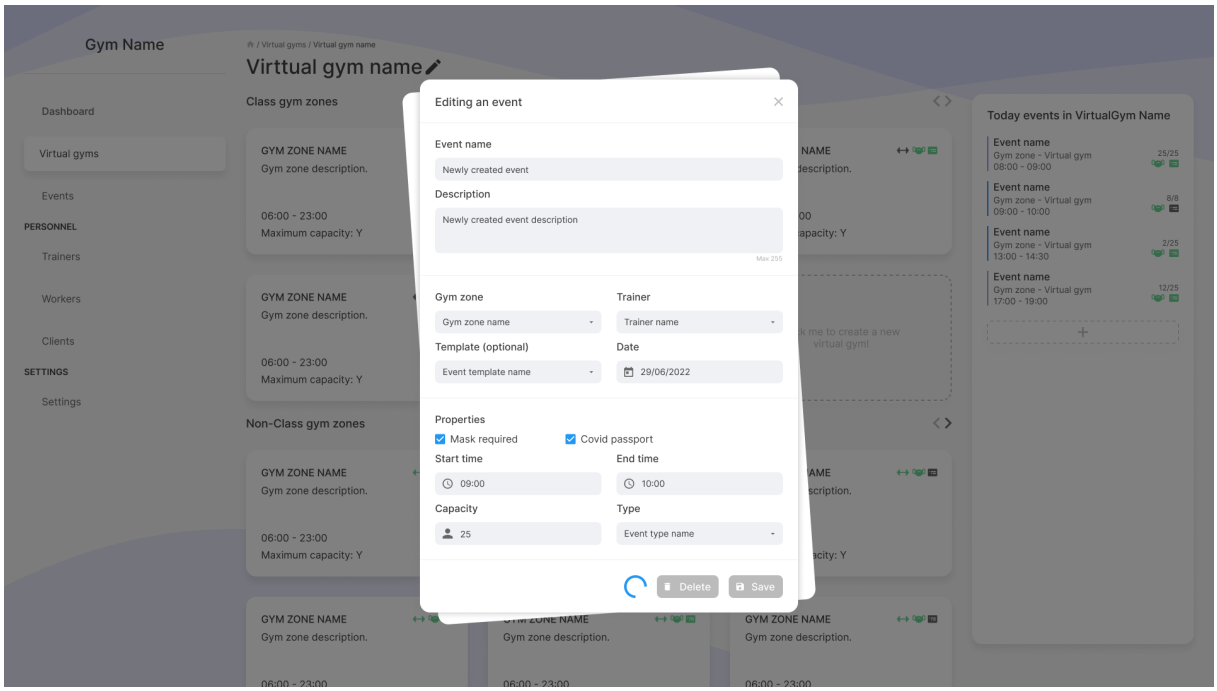


Figure 8.19: Event dialog (edit-loading state)

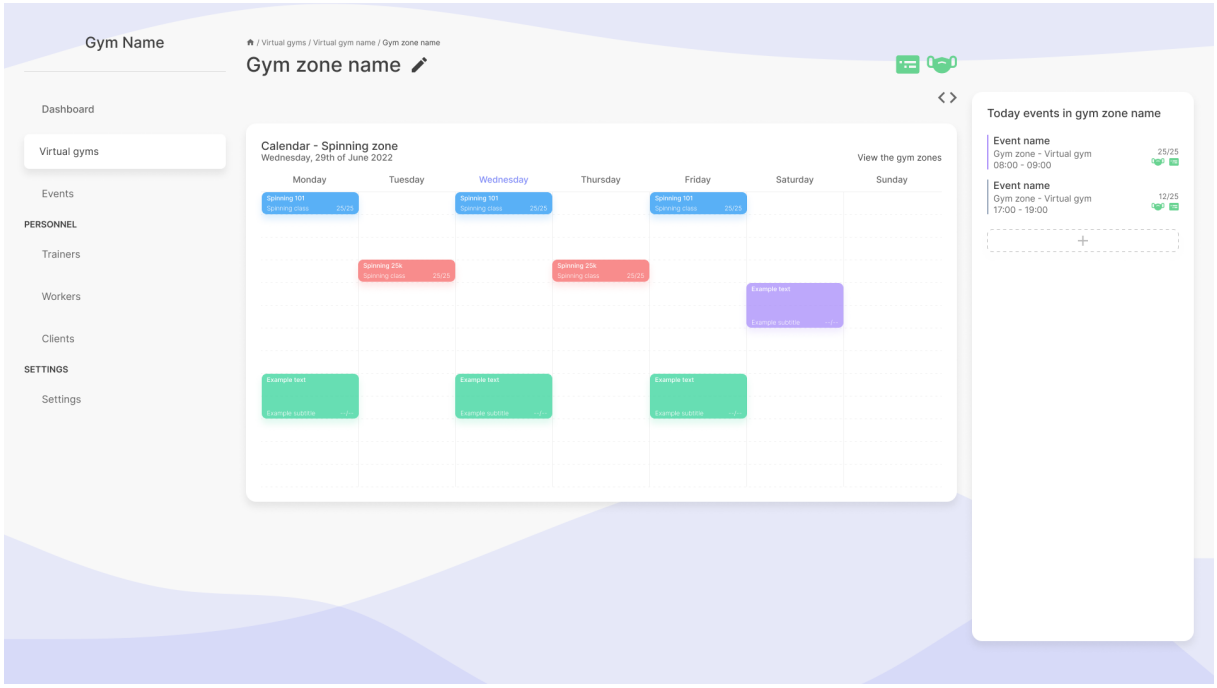


Figure 8.20: Class gym zone page, accessed by clicking on any class-type gym zone

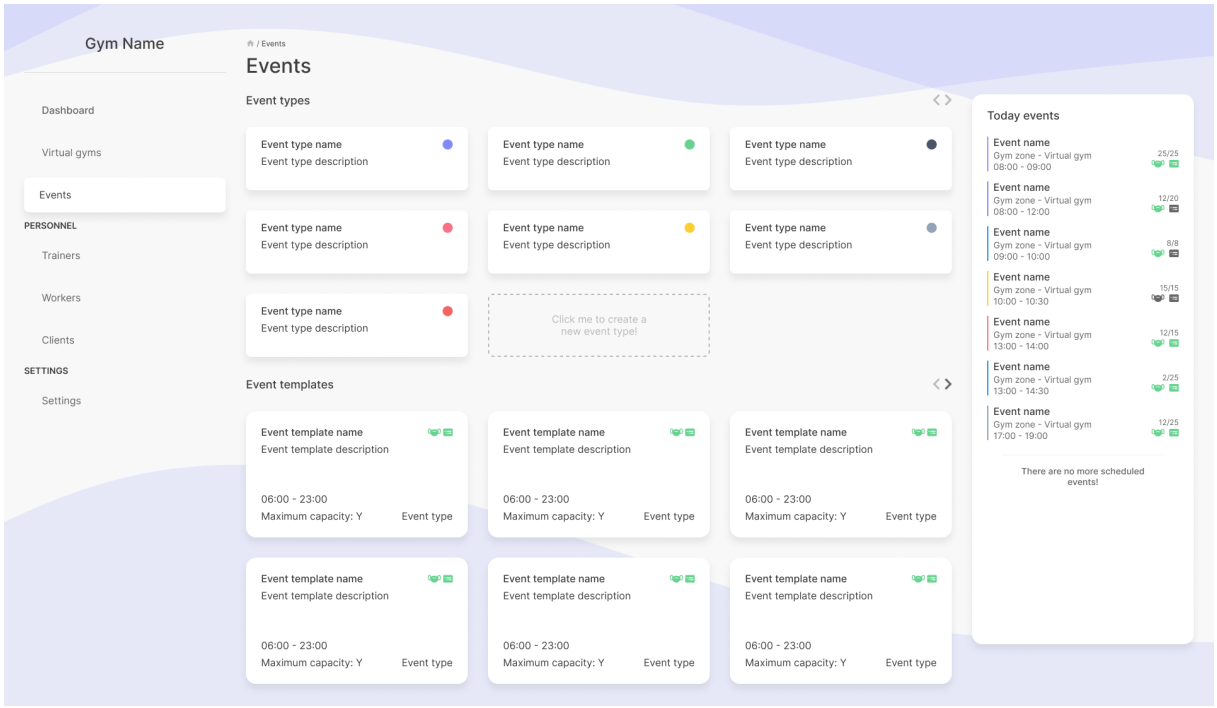


Figure 8.21: Event's page, which is accessed using the left navigation bar

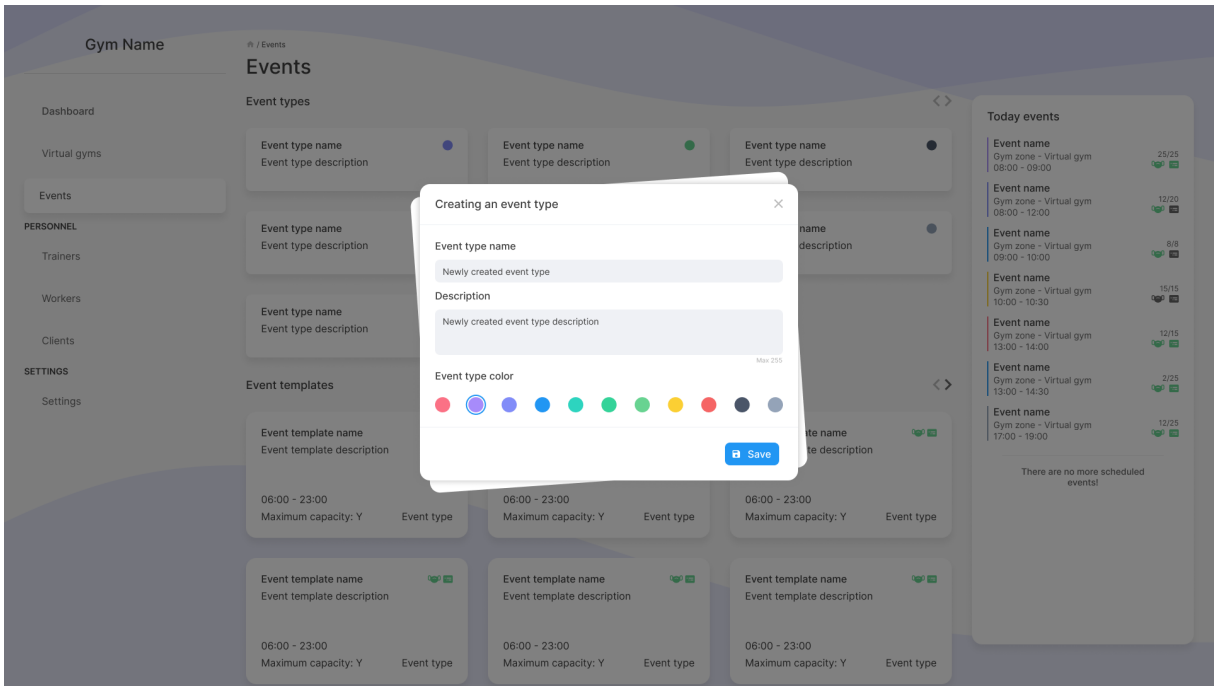


Figure 8.22: Event type dialog (create state)

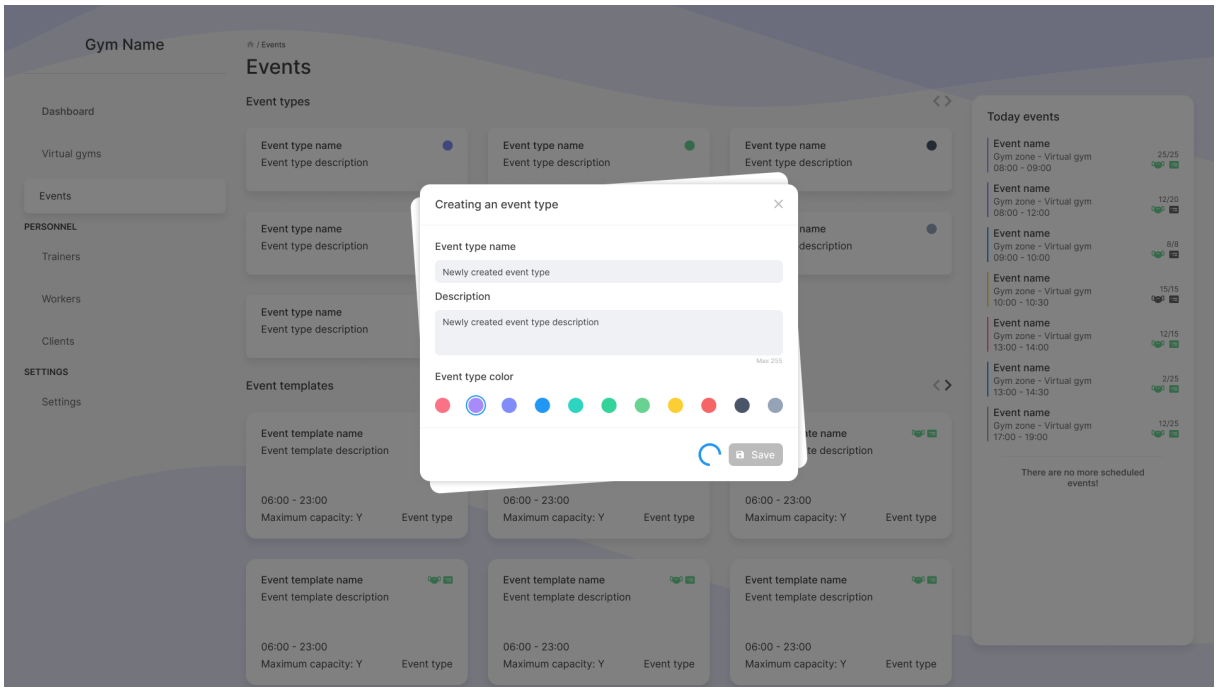


Figure 8.23: Event type dialog (create-loading state)

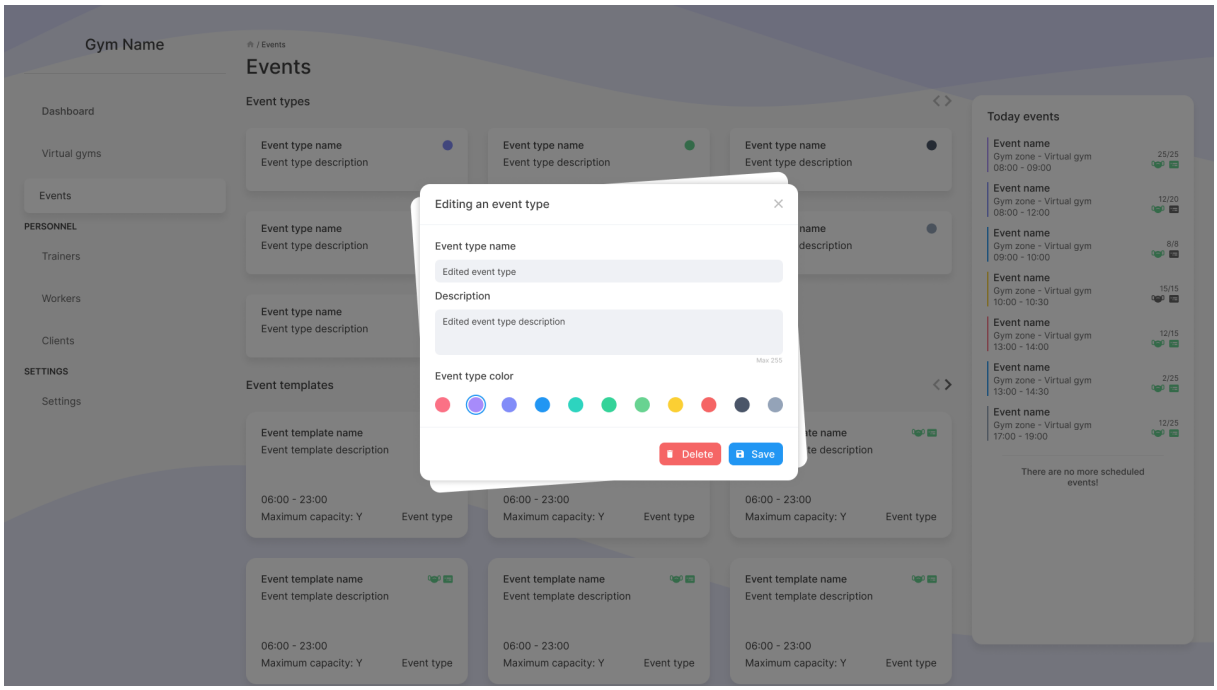


Figure 8.24: Event type dialog (edit state)

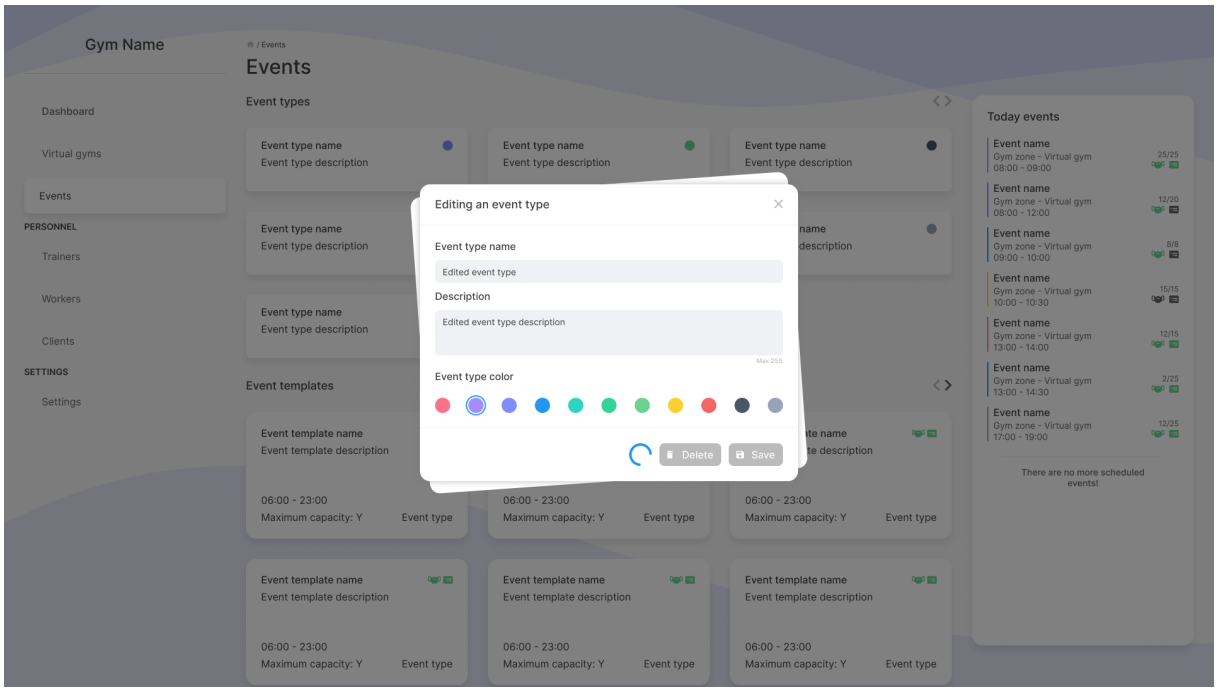


Figure 8.25: Event type dialog (edit-loading state)

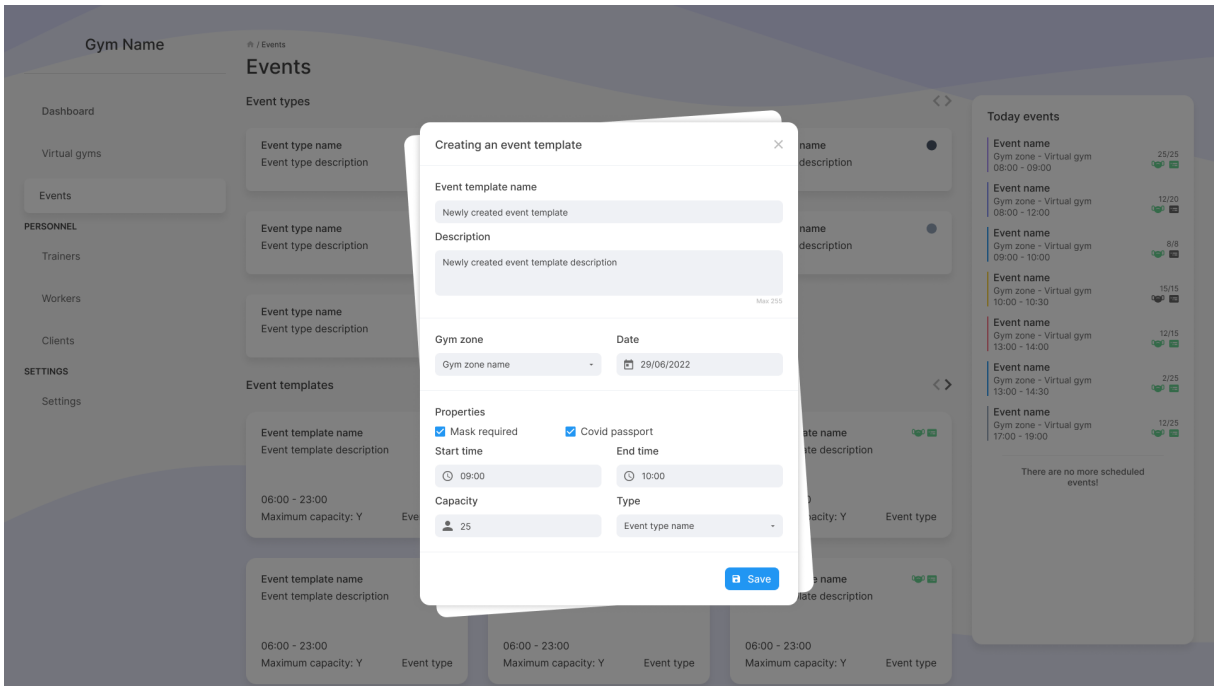


Figure 8.26: Event template dialog (create state)

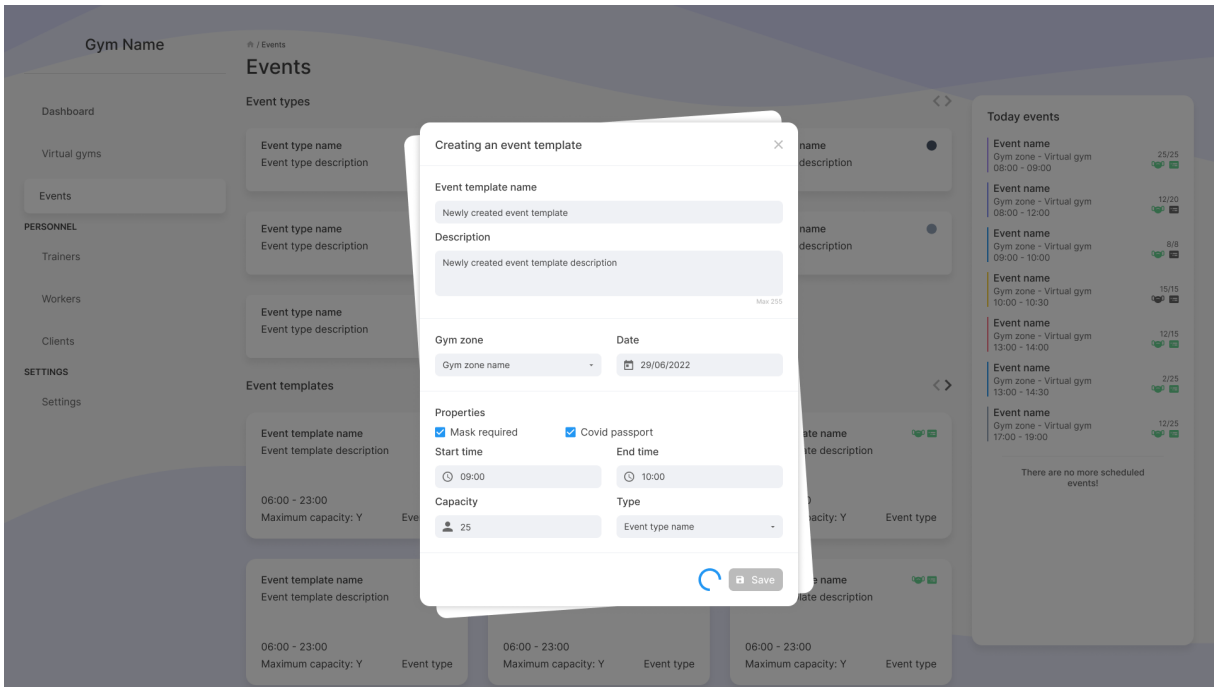


Figure 8.27: Event template dialog (create-loading state)

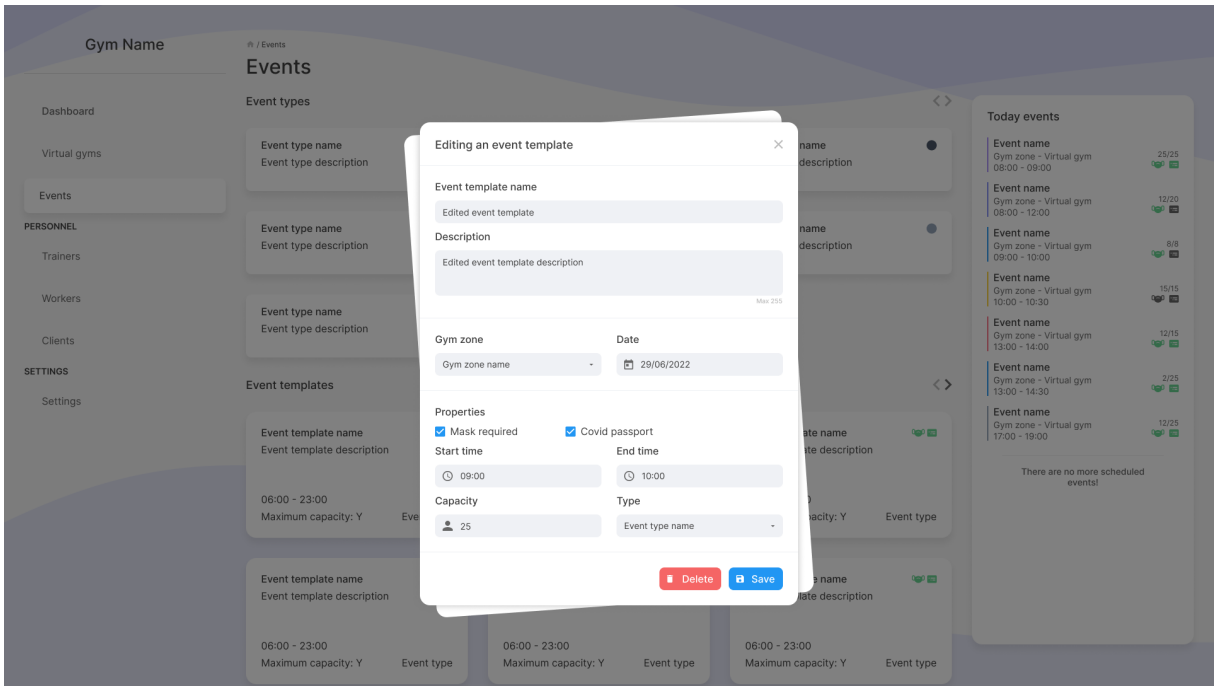


Figure 8.28: Event template dialog (edit state)

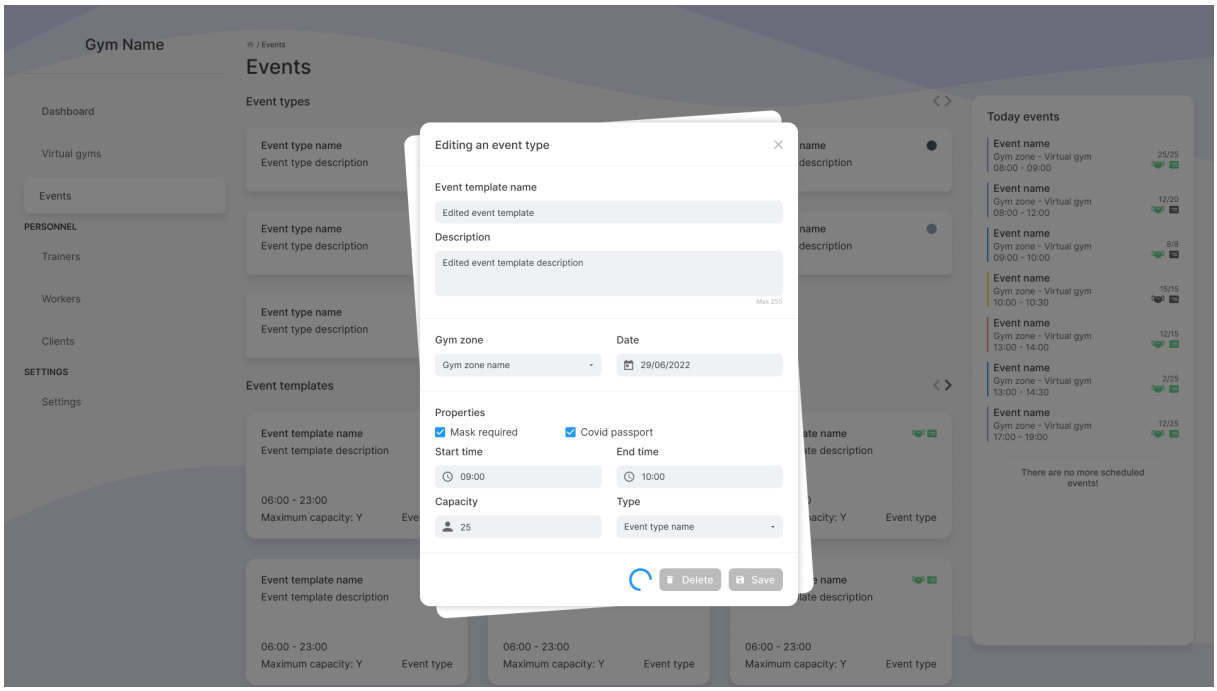


Figure 8.29: Event template dialog (edit-loading state)

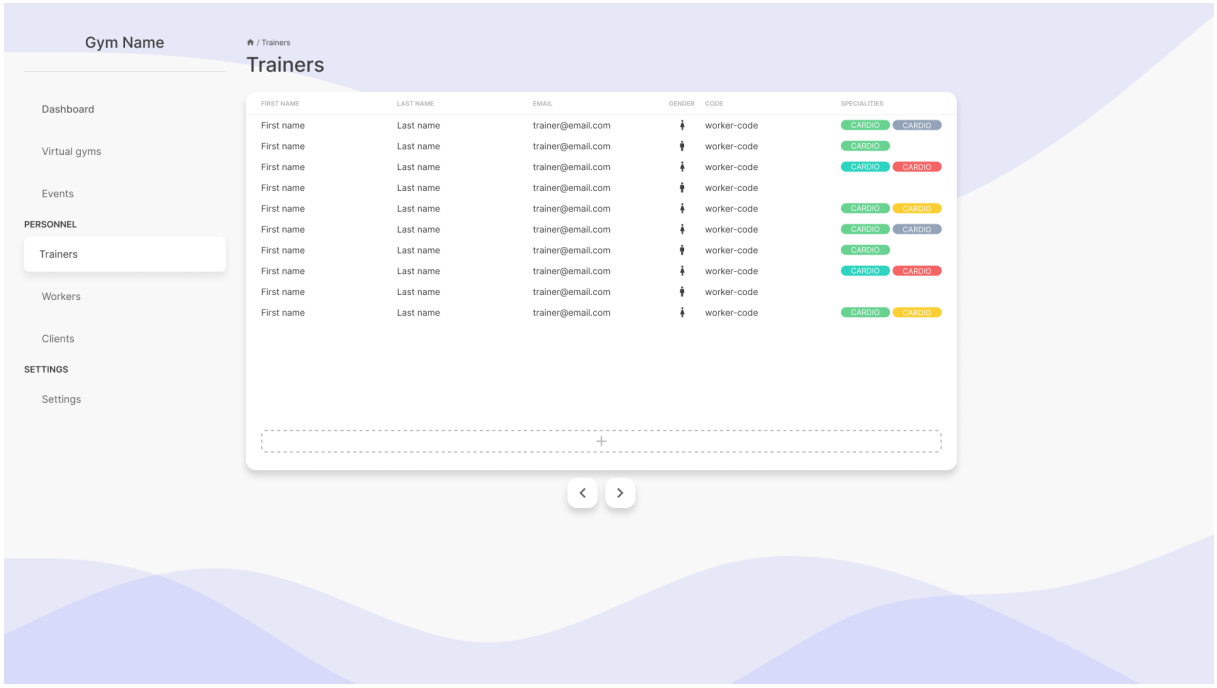


Figure 8.30: Trainer's page, which is accessed using the left navigation bar

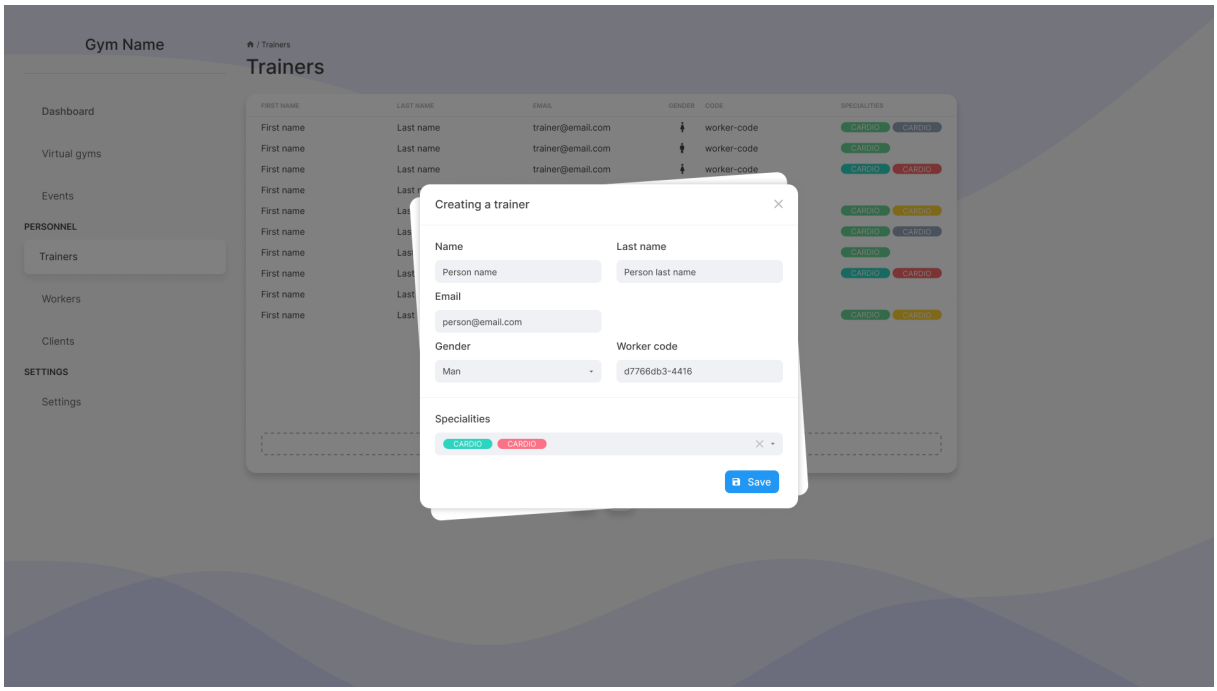


Figure 8.31: Trainer dialog (create state)

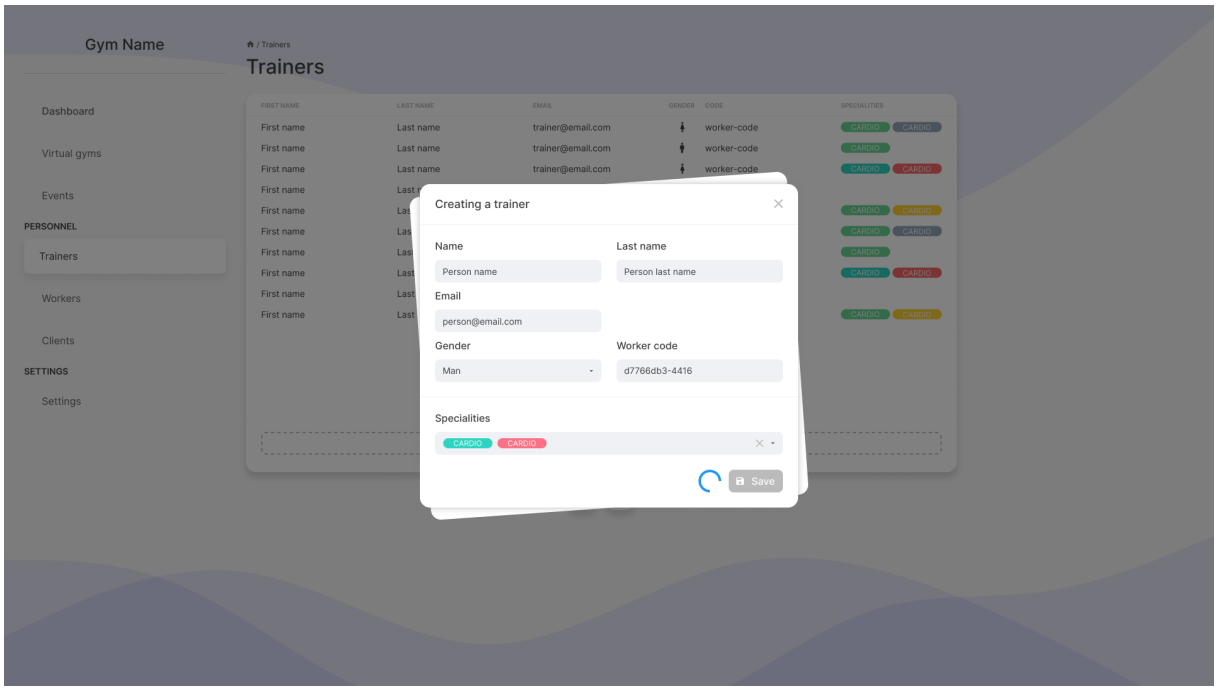


Figure 8.32: Trainer dialog (create-edit state)

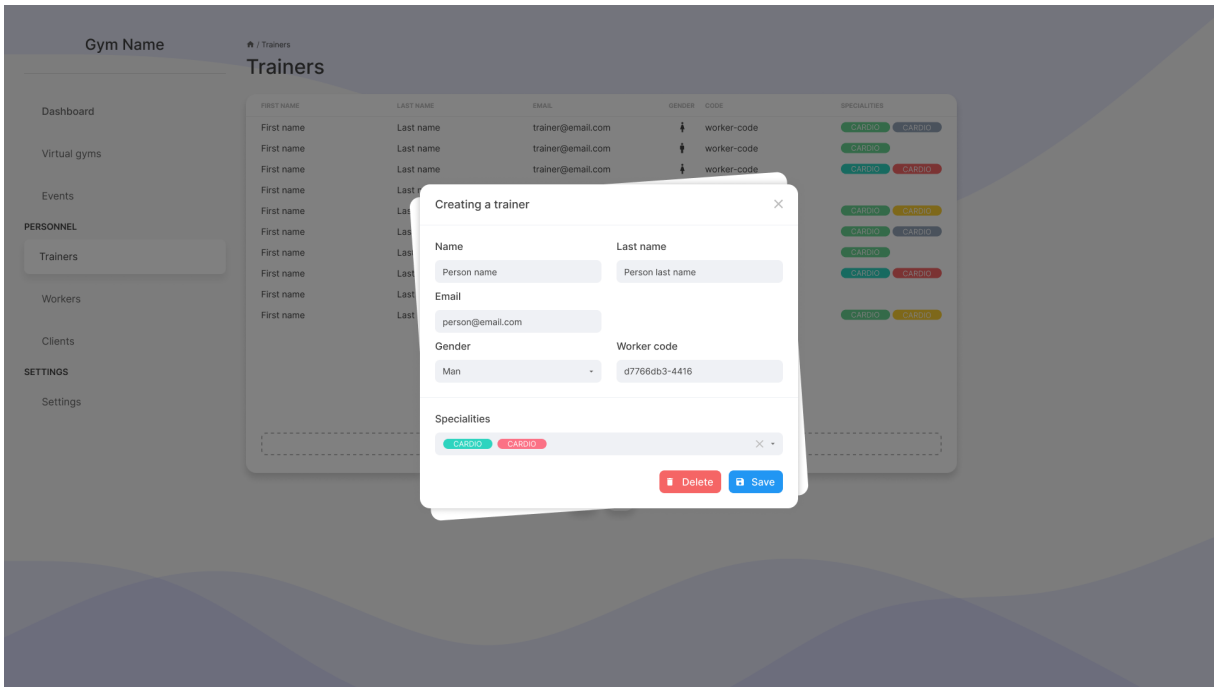


Figure 8.33: Trainer dialog (edit state)

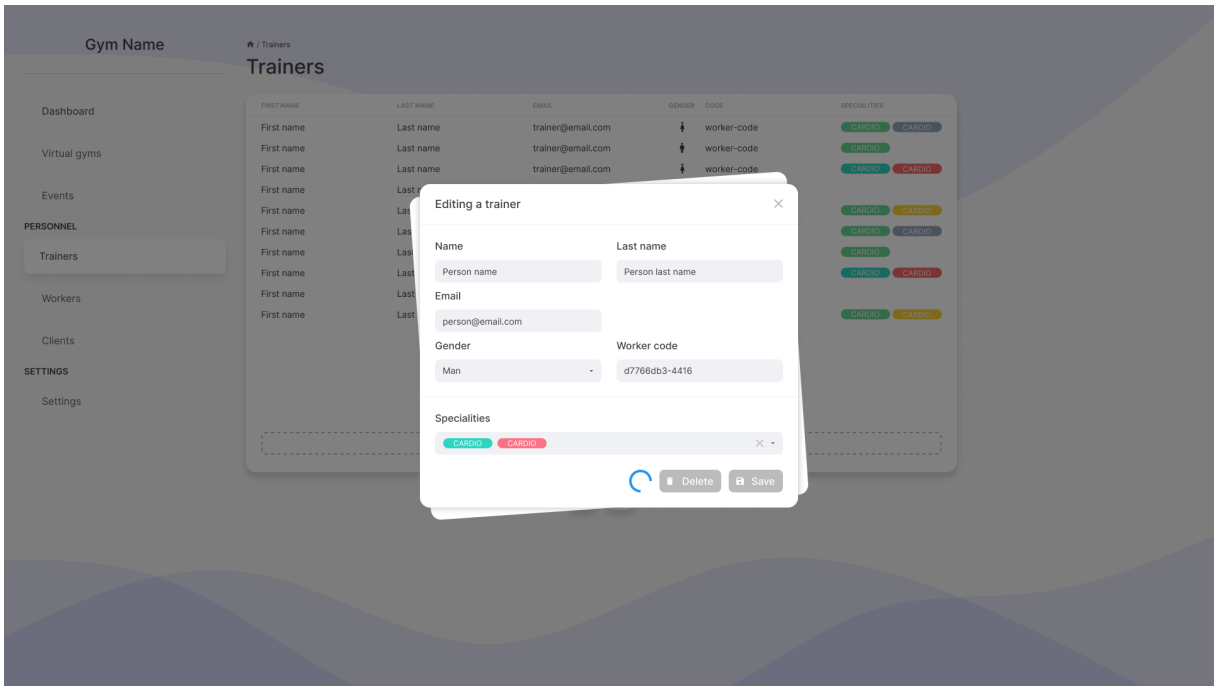


Figure 8.34: Trainer dialog (edit-loading state)

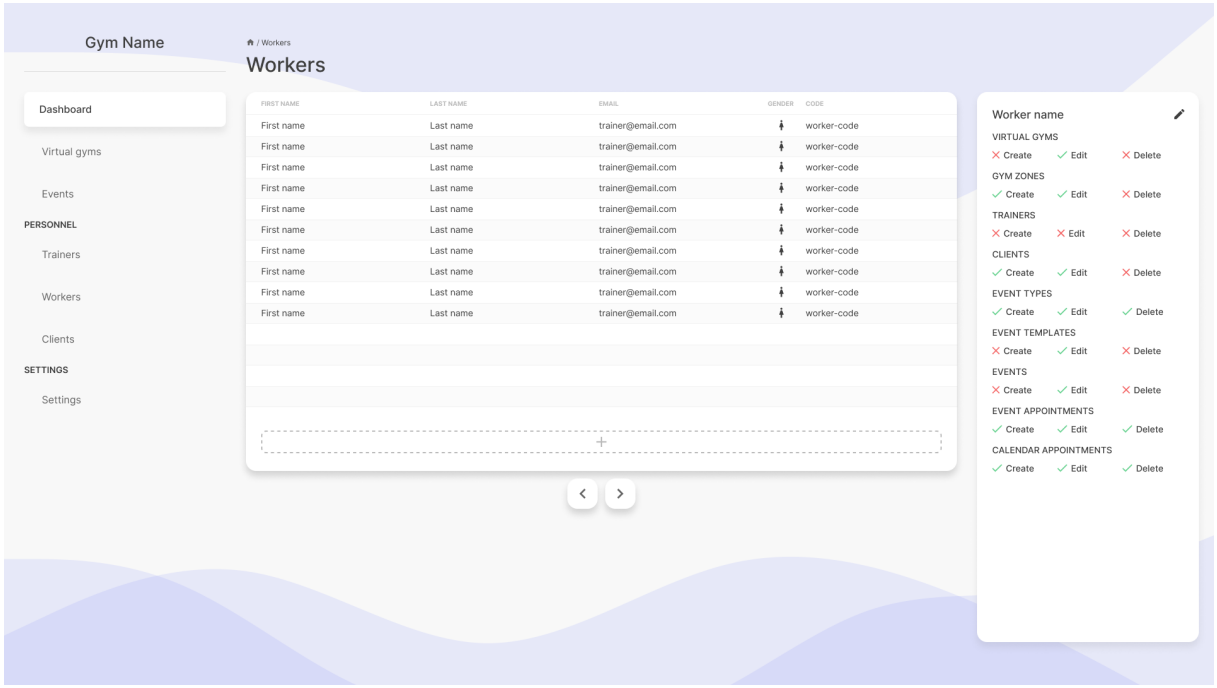


Figure 8.35: Worker's page, which is accessed using the left navigation bar

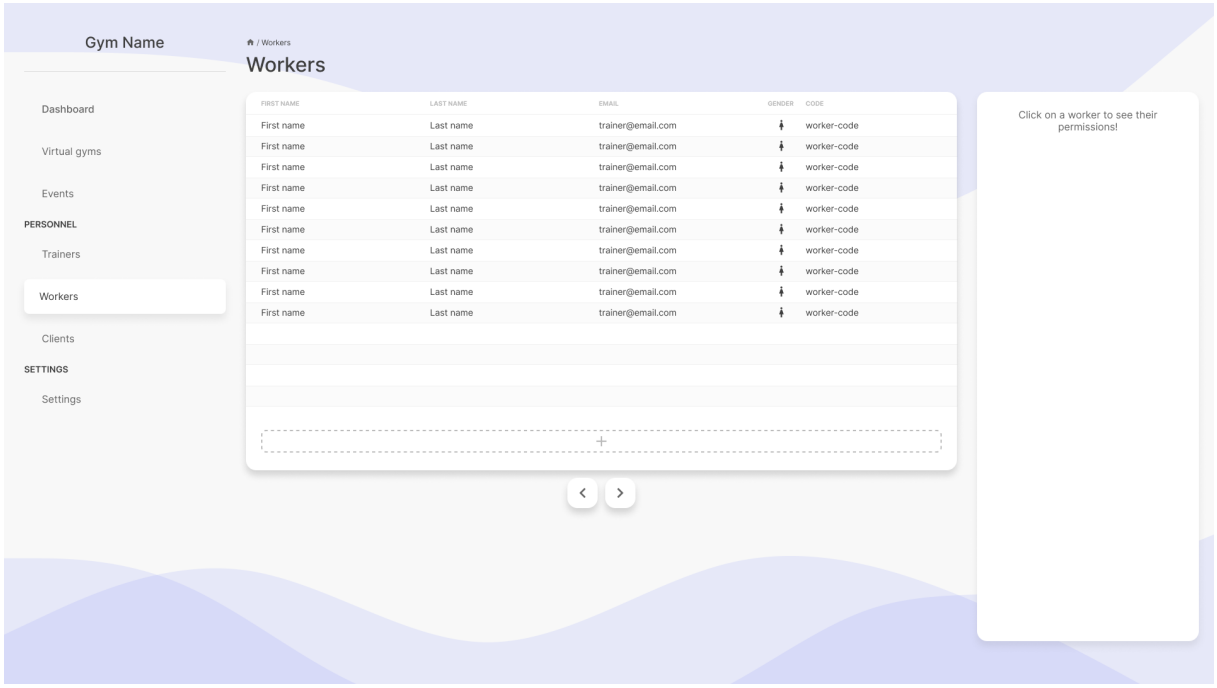


Figure 8.36: Same as previous, with a selected worker

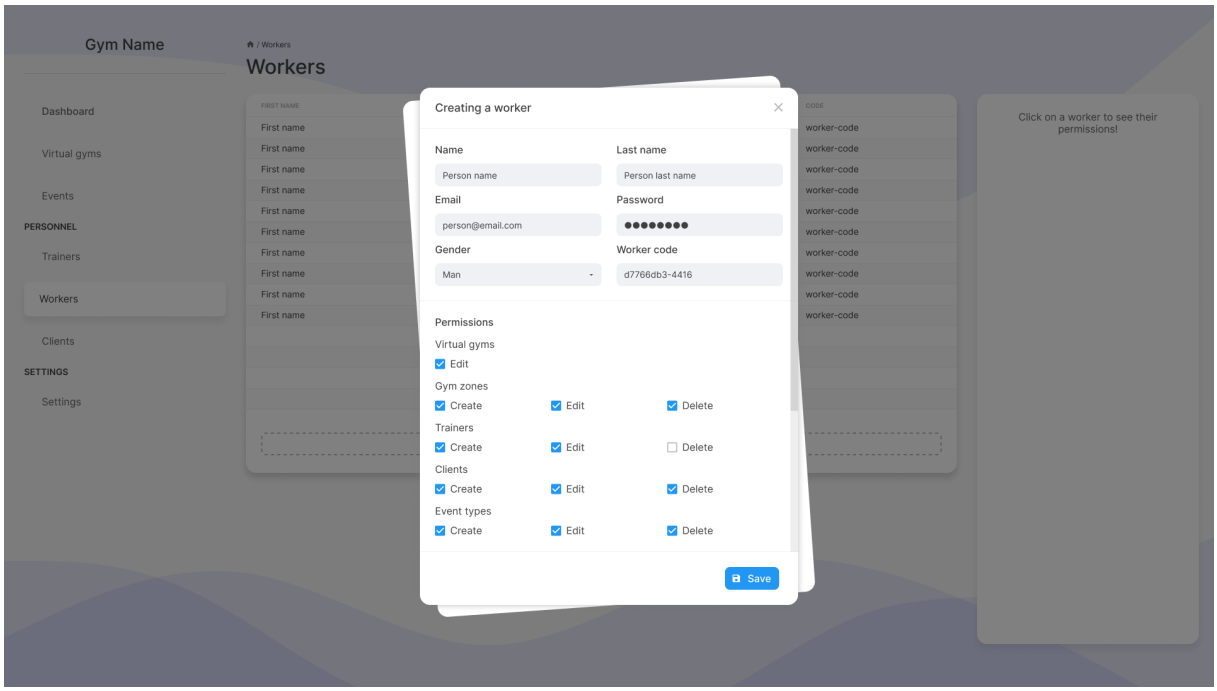


Figure 8.37: Worker dialog (create state)

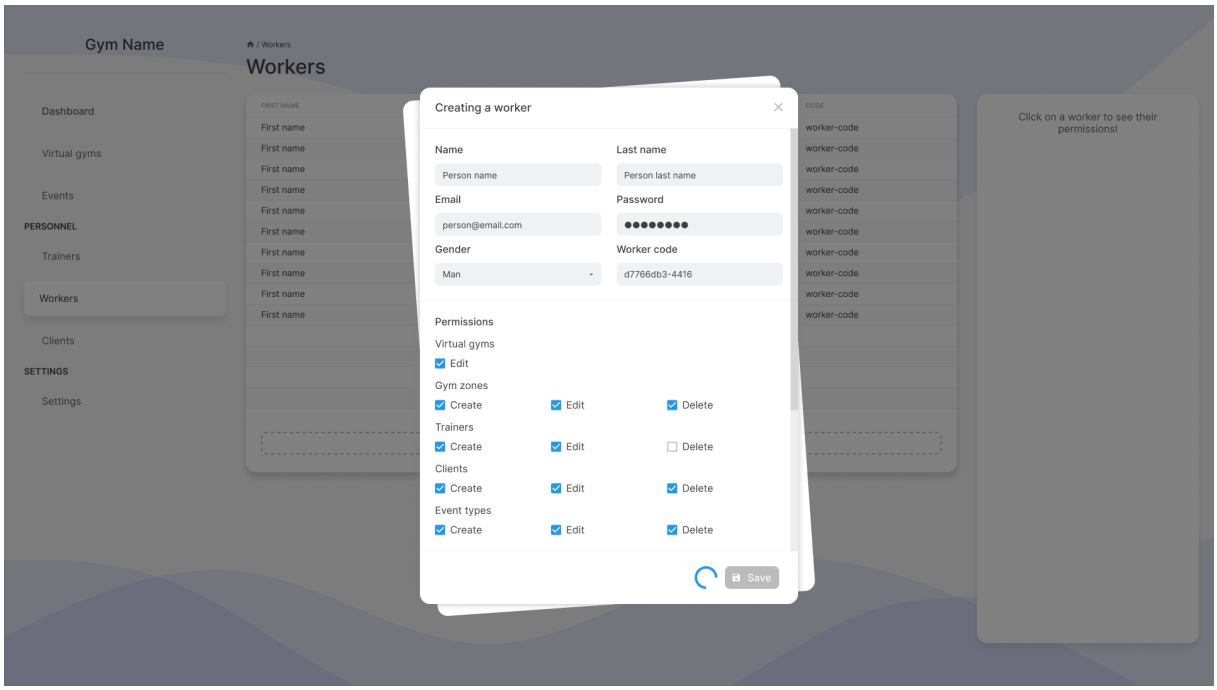


Figure 8.38: Worker dialog (create-loading state)

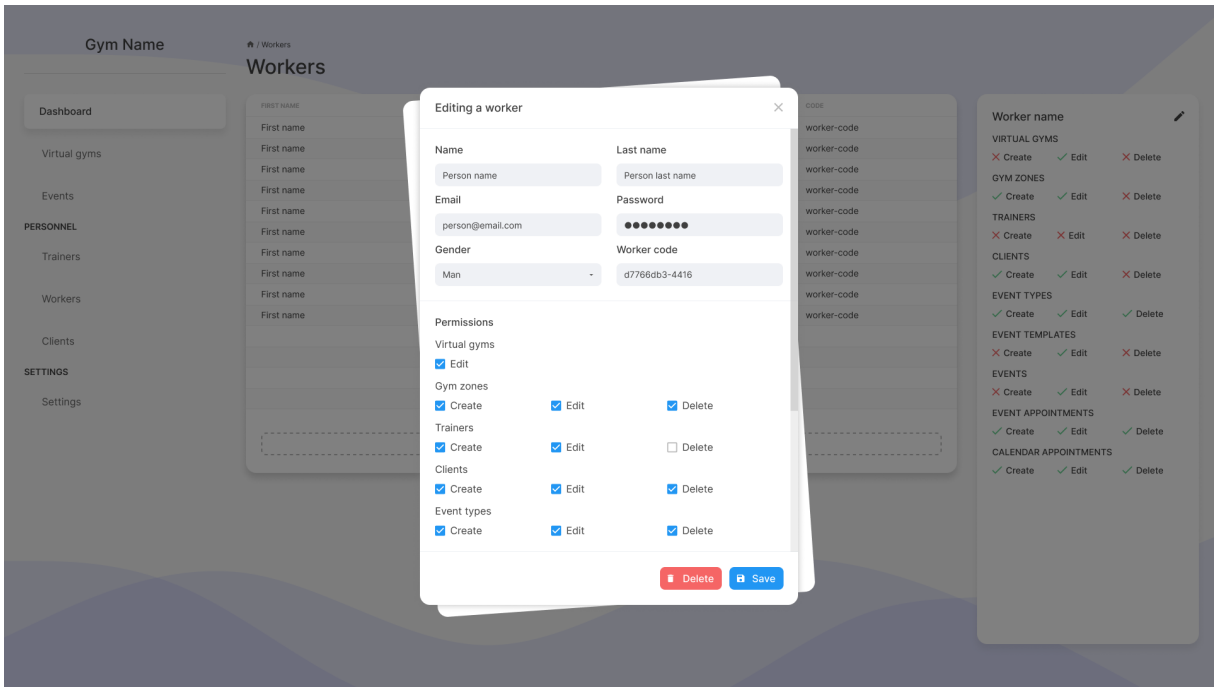


Figure 8.39: Worker dialog (edit state)

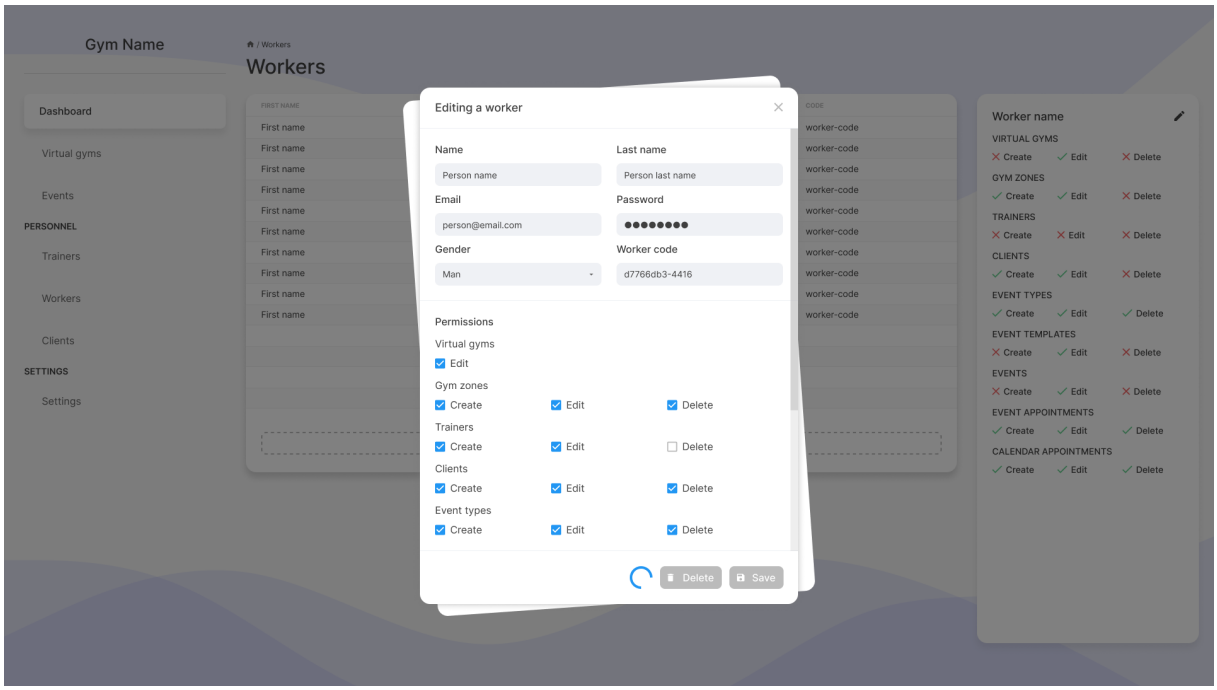


Figure 8.40: Worker dialog (edit-loading state)

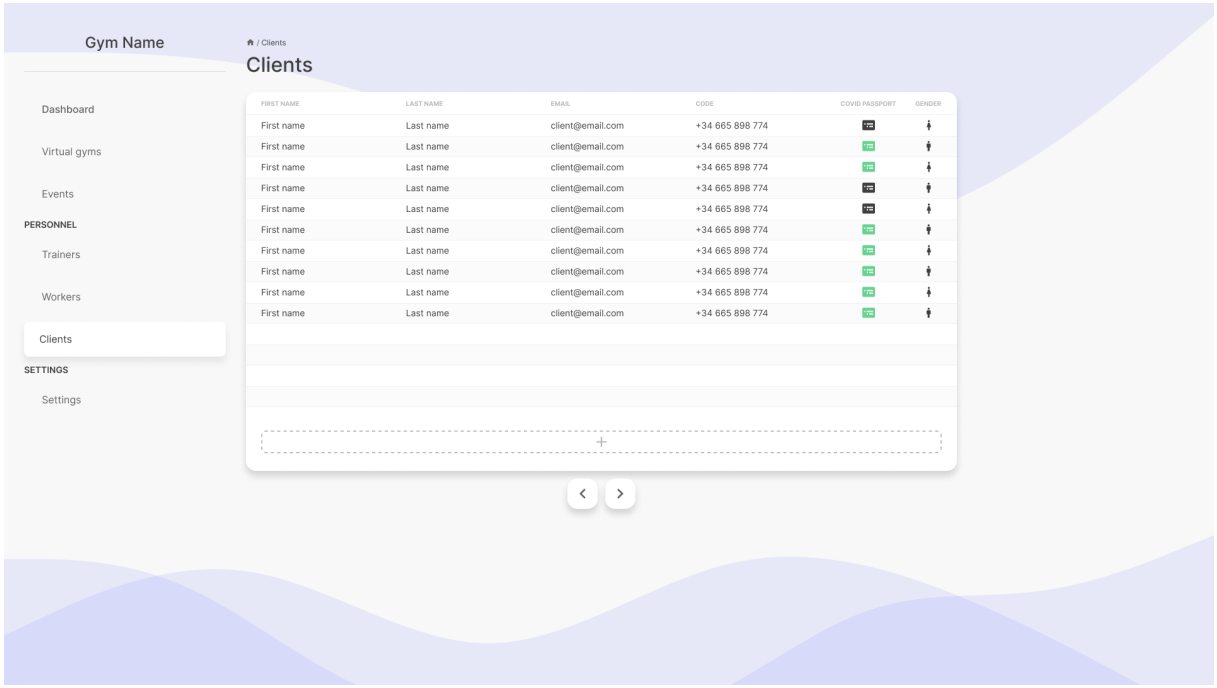


Figure 8.41: Client's page, which is accessed using the left navigation bar

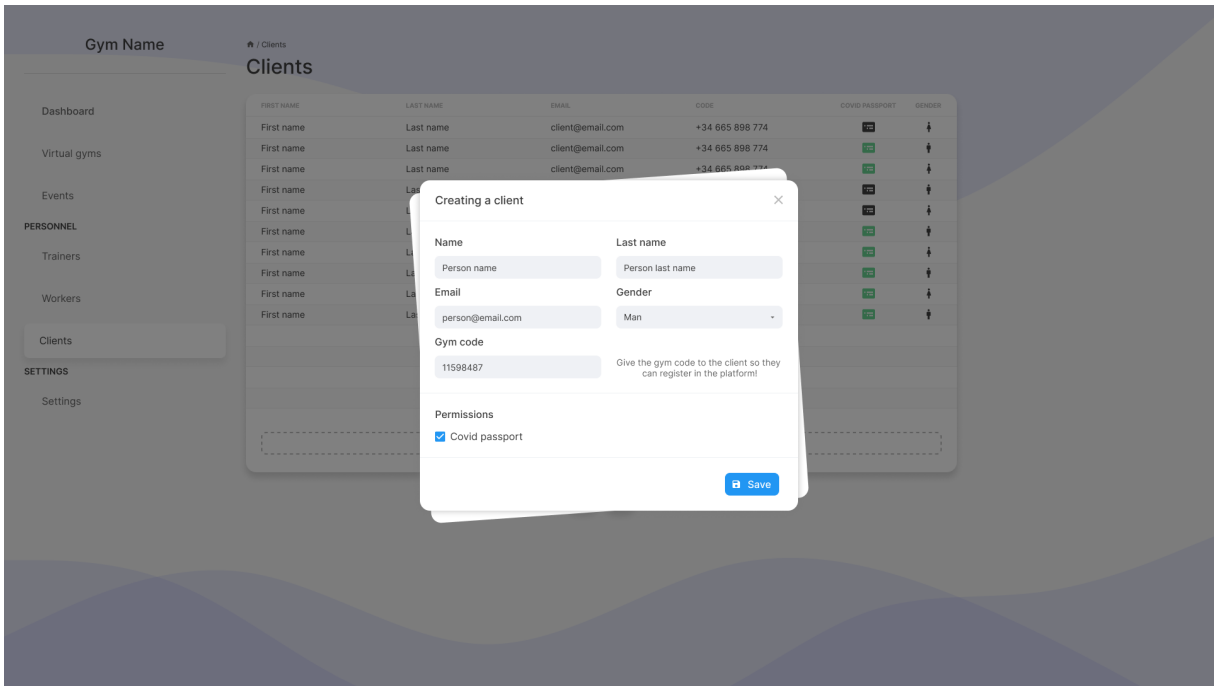


Figure 8.42: Client dialog (create state)

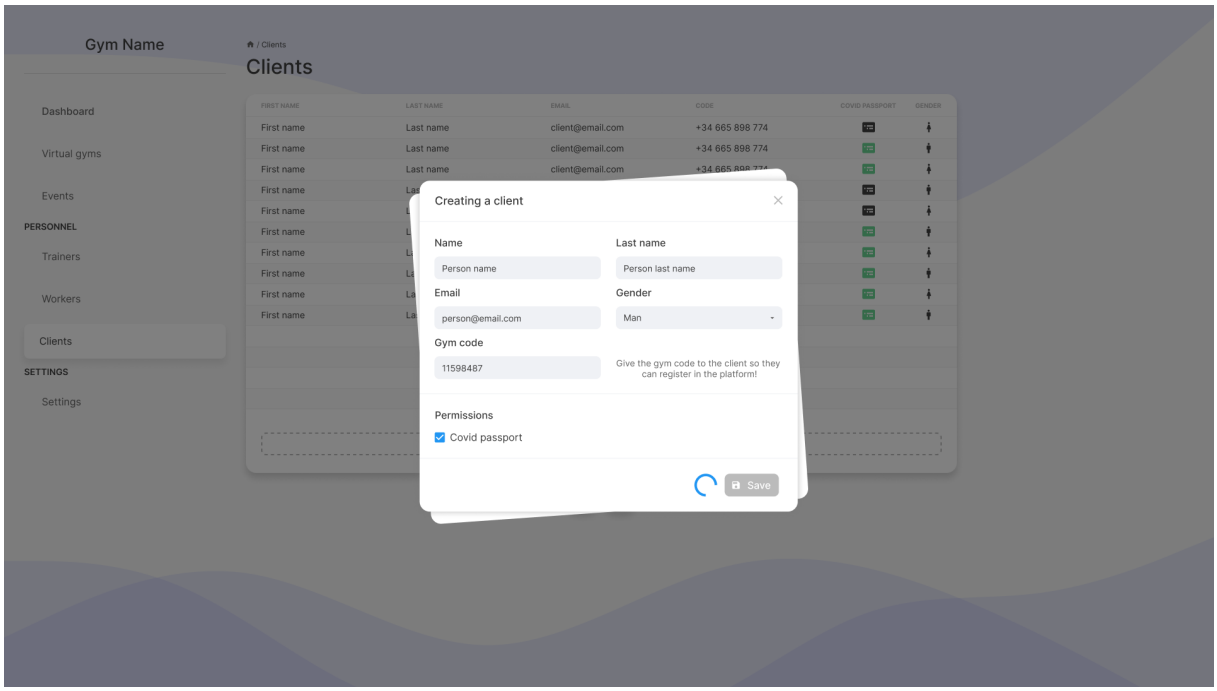


Figure 8.43: Client dialog (create-loading state)

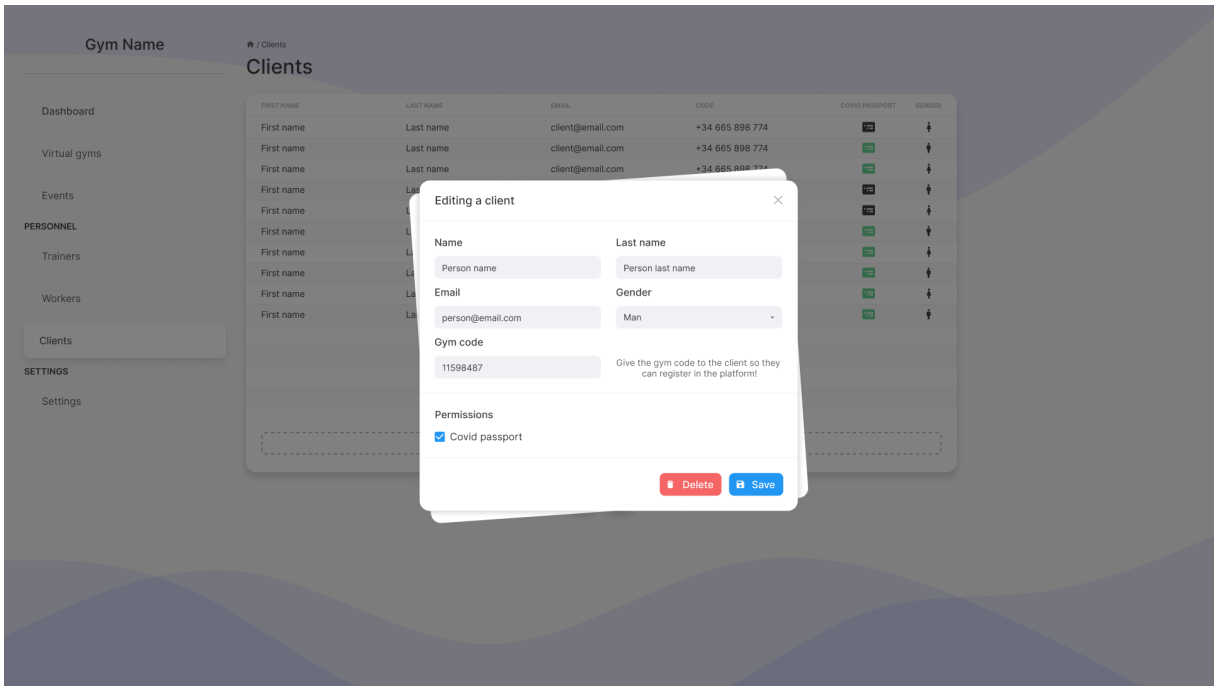


Figure 8.44: Client dialog (edit state)

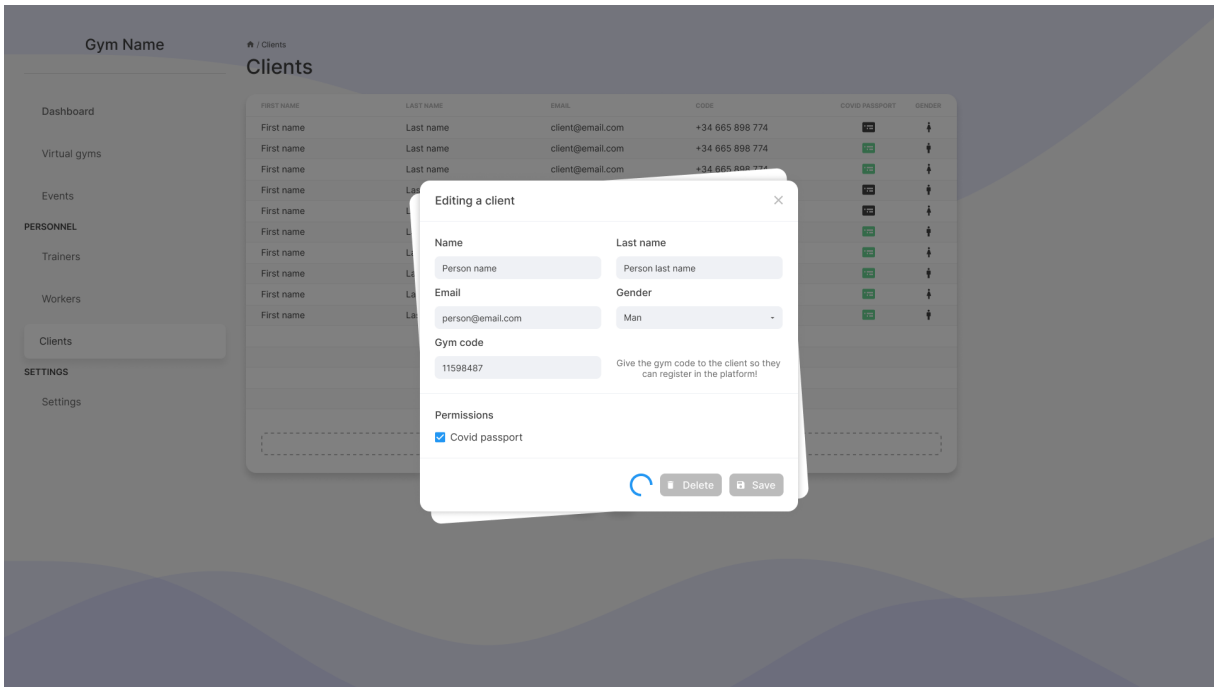


Figure 8.45: Client dialog (edit-loading state)

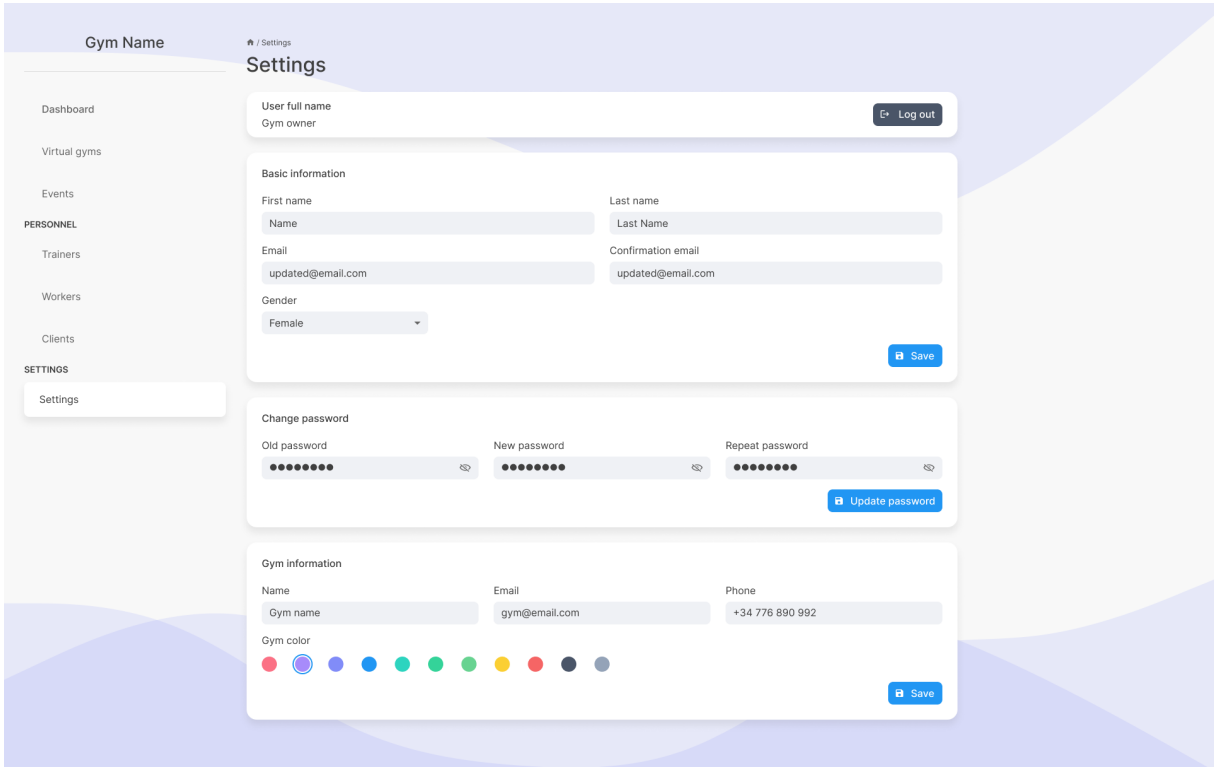


Figure 8.46: Settings page, from the owner's view

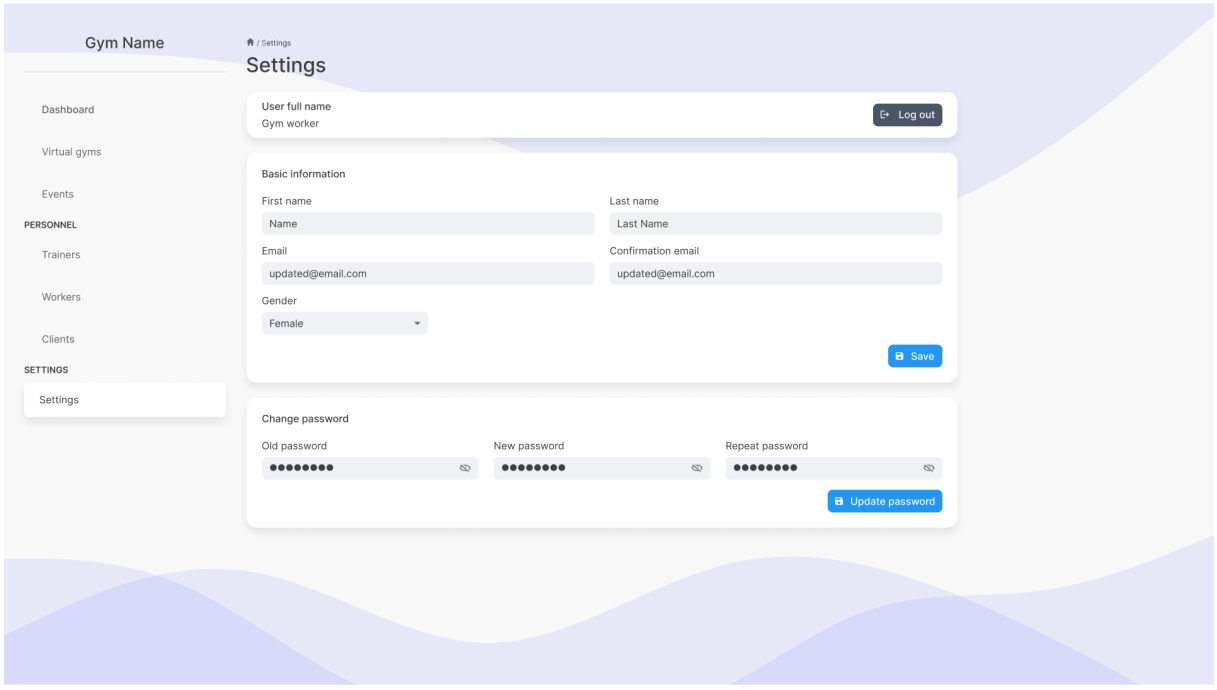


Figure 8.47: Settings page, from the worker's view

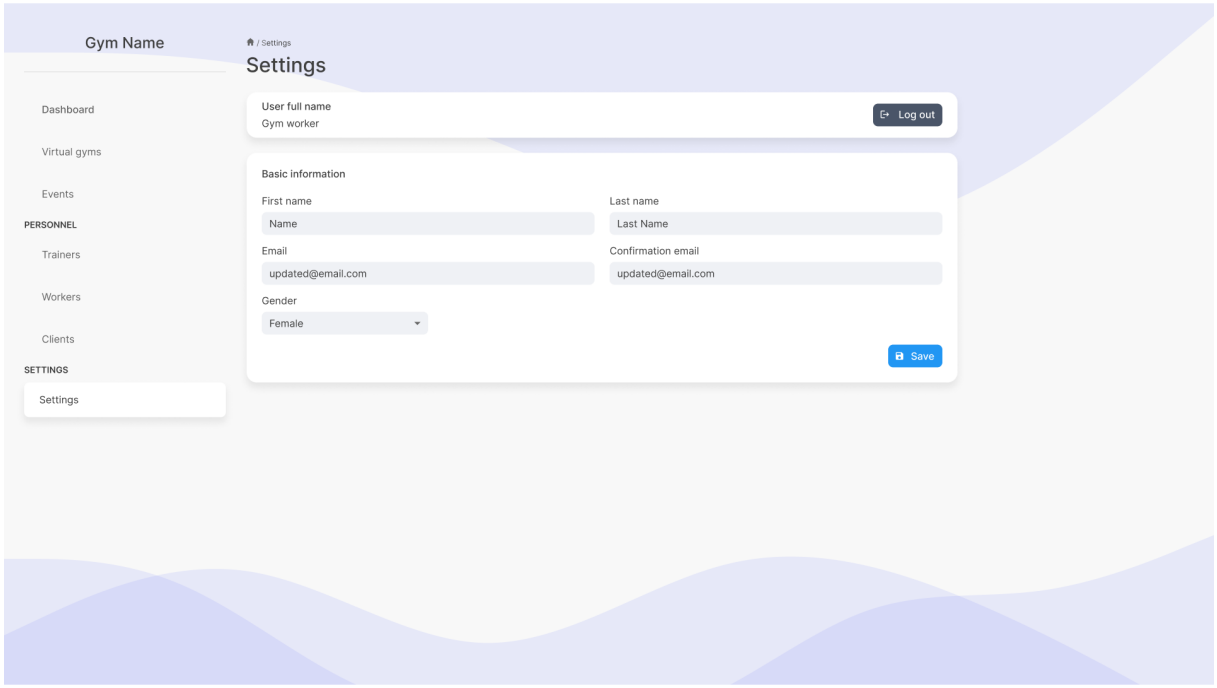


Figure 8.48: Settings page, from the client's view

9. Implementation and trials

9.1 Introduction

The goal of this chapter is to explain the development process, from the thinking of the application and its structure, to the actual development of it. It will also include and explanation of how the application has been constantly tested[1].

9.2 Organizing the idea

When I came up with the idea for the application, I ended up with a mix of thoughts that had to be organized and connected. Therefore, the first step was to write an extensive explanation of the application, the goal of the application, what it would do and what it would not do, trying to cover as many details as I could. After revising and rethinking the idea, I ended up with a potential project that *made sense* and could provide value to future clients.

Once the application had been described, defining the functionalities of such were easier. However, I needed to define a possible structure for such application. It was then when I decided to develop three applications: the landing page, the core application and the client application. I found interesting to distinguish between the client and the core application in order to simplify the development process. Mostly, when an application is expected to cover the most amount of possibilities and users, it ends which an enourmous amounts of checks and even repeated code. Knowing so, I decided to split it in two applications, while at the same time reducing the amount of boilerplate code. Knowing the existence of Nx, I had the logistic problem of maintaining a monorepo with the applications and libraries solved by such tool.

Even though it would probably change in structure, the initial idea of the architecture was there, and from it, I could start specifying the functional and non-functional requirements. It was probably the most time demanding task of the project, alongside structuring the working packages. Now that such parts were defined, I was able to visualise a roadmap for the development of the system.

9.3 Application development

I started developing the API application as it helped to define the database. It ended up being the most repetitive project to develop as there are many endpoints which are quite similar, however others were such a challenge, as involved complex queries. Once the server had been “finished”[2], I started designing the user interface. knowing available endpoints of the server helped with the thingk-

[1] Most testing involves unit and integration testing. In the API application, end-to-end testing has been implemented. For the UI applications, do not have end-to-end testing implemented due to a lack of time

[2] It is hard to consider a project finished, as product applications need constant improvements and changes in order to keep up with the market.

ing of the UI, yet at the same time it helped me realise what was lacking in the server. Therefore, after having finished the design, I got back to the API, adding what was mssing and cleaning up the code[3].

The next step was to start developing the UI applications. The core application had the most amount of logic and difficulty, which is the reason why I started with such project. I knew I would have to add a lot of components and views which would later be reused in the client application. This is the reason why the working packages for the core application have more estimated time than the client application. With Nx has been extremely easy and simple to abstract common code and keep it in the shared libraries, which can be easily used and referenced in any application. If in the future it is needed to add another application, the development process would be extremely simple, since most components could be reused.

Since then, I have been developing most of the core application. When most of it had been finished, I started developing the client application. With it, I had to start abstracting some of the code that was in the core application, so that it can be reused.

9.3.1 Continuous integration

One of the things that I explained in previous sections was the importance of having continuous integration, which would help to know if the code developed would cause errors or not. Also, with integration tests, it would be easier to verify if the component or controller work as expected. It was also one of the reasons why I chose Nx as a build tool, as it allows you to test for a specific project.

Knowing so, I defined multiple workflows [15] which would run if there were changes in the different project directiories[4]. The worflows would be run in two conditions:

1. If there is a push in the `develop` branch.
2. If a pull request if opened.

The development process would be:

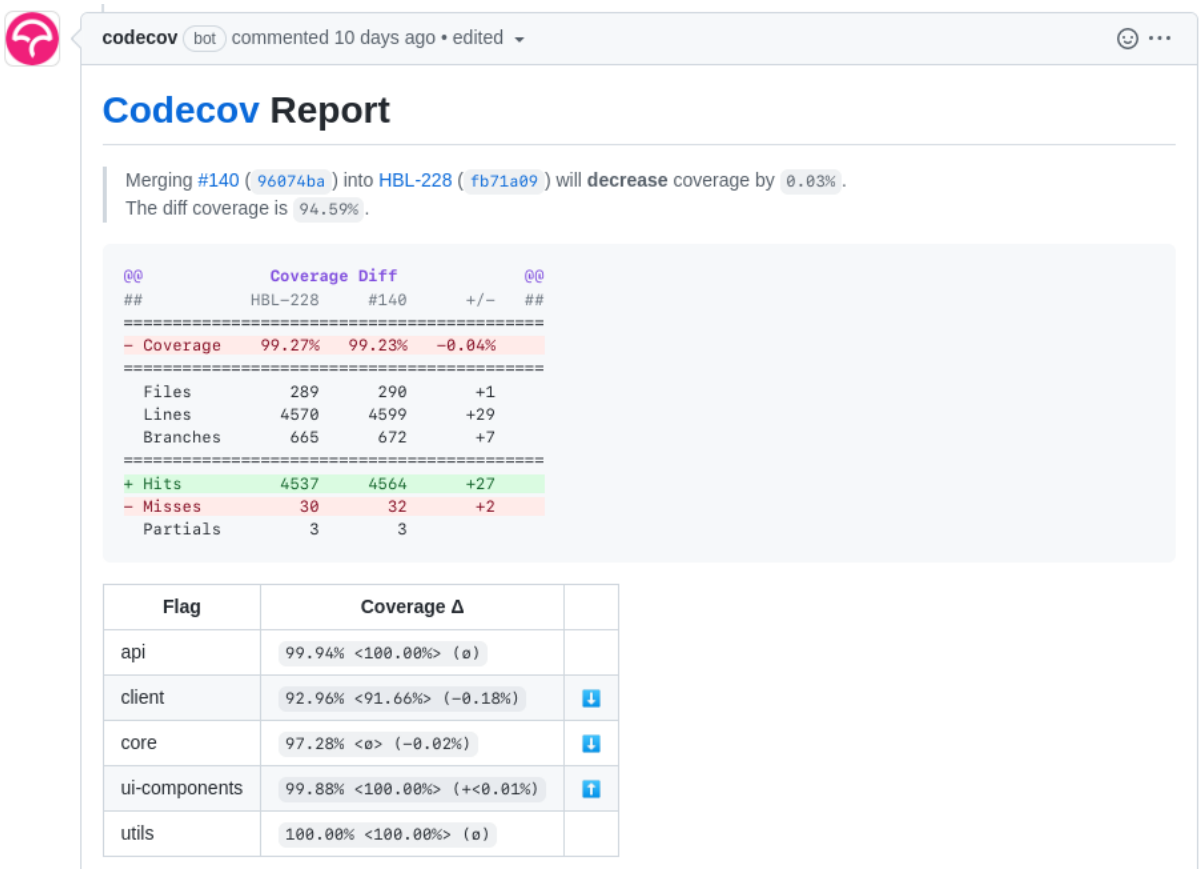
1. Creating a branch with the name of the Jira issue. Using such naming convention, when accessing an issue in Jira, it was easier to find the issue branches.
2. Adding as many changes as wanted, while also adding tests. If the changes were added in the API application, it should include unit, integration and e2e tests. If the changes where added in any library of any UI application, it should include unit and integration tests.
3. Once the task has been finished, a PR should be created which would trigger the workflows.

[3] Such thing was also expected in the roadmap, as it was my intention to revise the server after designing the UI.

[4] GitHub worflows allow you to trigger worflows on changes made in a given folder path.

4. If all checks pass, the PR can be merged into the wanted branch.

Using the PRs, it becomes really easy to identify issues, and it ensures that not a single line of coded is merged without knowing it does not pass its tests. As an addition to the checks, I also integrated CodeCov, which is a freemium tool that helps to visualise the coverage of the project[5]. The CodeCov integration adds a comment to the pull request summarizing the changes made and how such changes modify the coverage of the project.



codecov bot commented 10 days ago • edited

Codecov Report

Merging #140 (96074ba) into HBL-228 (fb71a09) will **decrease** coverage by 0.03%.
The diff coverage is 94.59%.

```
@@             Coverage Diff             @@
##           HBL-228     #140      +/-   ##
==========================================
- Coverage    99.27%   99.23%   -0.04%
==========================================
  Files          289      290       +1
  Lines         4570     4599      +29
  Branches       665      672       +7
==========================================
+ Hits          4537     4564      +27
- Misses          30       32       +2
  Partials         3        3
```

Flag	Coverage Δ	
api	99.94% <100.00%> (ø)	
client	92.96% <91.66%> (−0.18%)	⬇️
core	97.28% <ø> (−0.02%)	⬇️
ui-components	99.88% <100.00%> (+<0.01%)	⬆️
utils	100.00% <100.00%> (ø)	

Figure 9.1: Example of the comment generated by the CodeCov command

As explained before, the API server runs an e2e test. Such test requires a fake database and also takes more time to initialise, run and finish. Therefore, the end-to-end tests are not run on every merge to develop or every PR, but when the code is merged to the `main` branch. The end-to-end test starts a docker container in the GitHub action that will contain the database which will be used by the tests.

9.4 Trials

Aside from the implemented tests, the UI application has been manually tested in order to find bugs that may not have been covered by the automated tests. In the case of the API, testing the endpoints has been a bit harder, yet the Postman

[5]The CodeCov configuration file can be found in the `codecov.yaml` in the root folder of the repository.

software has been used in order to test the endpoints. In case of the UI applications, they have been tested by finding edge cases when running the appliation.

When a bug has been found, I firstly checked the tests to see why it was not being covered or if it was a bug in the test, as it is something that can happen. If the bug was found in the tests, I updated the tests and later fixed the actual code. If not, I would search the bug in the code, to later cover it with the tests.

10. Conclusions

11. Future work

A. User manual or installation

The applications should be accessed from a public URL. However, due to lack of time and money, the application has not been deployed, meaning it can not be accessed from any device, other than the local machine. In order to run the application from the local machine, the installation manual can be found in the code repositiry, as well as in the documentation for each application, in the repository aswell.

Bibliography

[1] Unspecified author. Unspecified last update. *Salario medio para Diseñador Ui en España, 2022*. April 19th 2022. https://es.talent.com/salary?job=dise%C3%B1ador+ui.

[2] Unspecified author. Unspecified last update. *Salario medio para Programador Javascript en España, 2022*. April 19th 2022. https://es.talent.com/salary?job=programador+javascript.

[3] Atlassian. Unspecified last update. *Jira Software homepage*. March 20th 2022. https://www.atlassian.com/software/jira.

[4] Unspecified author. Unspecified last update. *Use smart commits*. March 20th 2022. https://support.atlassian.com/bitbucket-cloud/docs/use-smart-commits/.

[5] Unspecified author. Unspecified last update. *Using Nx at Enterprises*. December 12th 2021. https://nx.dev/l/r/guides/monorepo-nx-enterprise#apps-and-libs.

[6] Vercel. Unspecified last update. *The React Framework for Production*. December 12th 2021. https://nextjs.org/.

[7] MDN contributors. January 28th 2022. *Code splitting*. December 12th 2021. https://developer.mozilla.org/en-US/docs/Glossary/Code_splitting.

[8] Unspecified author. Unspecified last update. *Move faster with intuitive React UI tools*. December 12th 2021. https://mui.com/.

[9] Unspecified author. Unspecified last update. *PostgresQL: The World's Most Advanced Open Source Relational Database*. December 12th 2021. https://www.postgresql.org/.

[10] Unspecified author. Unspecified last update. *4.x API*. December 13th 2021. https://expressjs.com/en/api.html.

[11] @pleerock[1]. April 28th 2022. *typeorm/typeorm*. December 13th 2021. https://github.com/typeorm/typeorm.

[12] Facebook. Unspecified last update. *Jest*. December 13th 2021. https://expressjs.com/en/api.html.

[13] @Nick McCurdy[2]. September 21st 2021. *React Testing Library*. December 13th 2021. https://testing-library.com/docs/react-testing-library/intro.

[14] Cypress.io. Unspecified last update. *Javascript End to End testing framework*. December 13th 2021. https://www.cypress.io/

[15] Unspecified author. Unspecified last update. *Using workflows*. May 22nd 2022. https://docs.github.com/en/actions/using-workflows

[1] GitHub's username.

[2] GitHub's username.