CCfit GYM

Jackson Ferreira dos Santos 2020091

ABSTRACT

This is a project thought and developed during the very end of the last semester of the Higher Diploma in Science Computing course at CCT College. The students were given a booking system scenario to work on and that’s where CCfit Gym – slots booking proposal came from. The project idea was based on making a real-life system based on the current pandemic situation (COVID-19). This application is responsible for providing the user the possibility to sign themselves up, making payments and book session all through the web application. It also allows the administrator and the teachers to check each day schedules, manage user classes and registers besides introducing extra features.

ACKNOWLEDGEMENTS

I would like to express my very great appreciation to all CCT college employees, everyone involved in the learning process from the administrative departments to the principal, in special the lectures I had that provided me with the possibility to obtain an enormous amount of knowledge and experience throughout the whole year of 2020.

I would also like to thank all my classmates and other students I had contact in mentoring sections, workshops and other college events.

Last but not least, I am grateful for my family and friends who have always supported and believed in me during this year.

CONTENT

[CHAPTER 1: INTRODUCTION 5](#_Toc62925757)

[Project context: 5](#_Toc62925758)

[Why is this a good project? 6](#_Toc62925759)

[Main goal: 6](#_Toc62925760)

[Objectives: 6](#_Toc62925761)

[Areas to cover 6](#_Toc62925762)

[Personal challenges 7](#_Toc62925763)

[Project Plan: 7](#_Toc62925764)

[Waterfall methodology: 7](#_Toc62925765)

[Planning process 9](#_Toc62925766)

[CHAPTER II: LITERATURE REVIEW 10](#_Toc62925767)

[Backend: 10](#_Toc62925768)

[Python 10](#_Toc62925769)

[Django 11](#_Toc62925770)

[MVT (Model-View-Template) 11](#_Toc62925771)

[Frontend: 11](#_Toc62925772)

[HTML/CSS 12](#_Toc62925773)

[JavaScript 12](#_Toc62925774)

[Database: 12](#_Toc62925775)

[SQLite 13](#_Toc62925776)

[PostgreSQL 13](#_Toc62925777)

[Deployment: 14](#_Toc62925778)

[Heroku 14](#_Toc62925779)

[Additional features: 15](#_Toc62925780)

[xhtml2pdf 15](#_Toc62925781)

[Stripe 16](#_Toc62925782)

[CHAPTER III: SYSTEM ANALYSIS & DESIGN 17](#_Toc62925783)

[Functional Requirements: 17](#_Toc62925784)

[Main Requirements 17](#_Toc62925785)

[Subsystems requirements 17](#_Toc62925786)

[User Interface Design: 25](#_Toc62925787)

[Data requirements: 35](#_Toc62925788)

[Database design: 36](#_Toc62925789)

[Functional design: 38](#_Toc62925790)

[CHAPTER IV: IMPLEMENTATION OF THE SYSTEM 42](#_Toc62925791)

[Architecture: 42](#_Toc62925792)

[Technologies implemented: 42](#_Toc62925793)

[Frontend 42](#_Toc62925794)

[Backend 42](#_Toc62925795)

[Database 42](#_Toc62925796)

[Other 43](#_Toc62925797)

[Implementation of the system: 43](#_Toc62925798)

[Database 43](#_Toc62925799)

[Backend 47](#_Toc62925800)

[Frontend 51](#_Toc62925801)

[Testing 54](#_Toc62925802)

[Problems encountered: 55](#_Toc62925803)

[CHAPTER V: TESTING & EVALUATION 57](#_Toc62925804)

[System access: 57](#_Toc62925805)

[Edit user information: 58](#_Toc62925806)

[Book classes: 59](#_Toc62925807)

[Generate invoice: 60](#_Toc62925808)

[Make payments: 61](#_Toc62925809)

[Check bookings: 61](#_Toc62925810)

[Manage users: 62](#_Toc62925811)

[CHAPTER VI: CONCLUSIONS & FURTHER WORK 63](#_Toc62925812)

[Conclusions 63](#_Toc62925813)

[Further work 64](#_Toc62925814)

[APPENDIX A: TEST INFORMATION 65](#_Toc62925815)

[Source code: 65](#_Toc62925816)

[Access to the system: 65](#_Toc62925817)

[User credentials: 65](#_Toc62925818)

[REFFERENCES: 67](#_Toc62925819)

# CHAPTER 1: INTRODUCTION

## Project context:

In consequence of the current pandemic global situation we have been living, it was thought to develop a booking system that helps gyms owners to manage their establishment regarding the number of people allowed to be indoors at the same time while training, designate the instructors for each one the classes, and provide users with the experience of doing everything online, from registering to making the online payment and finally booking the slot session.

As the business had to adapt to the restriction, the gyms started to control the flow of users they have during the day and split that in sections so they can still provide their services.

This project in specific is related to the creation of a web-based application for the CCfit gym which like the others, had to adapt its services.

Considering the scenario described above, here are the functionalities the system will provide for each one of the system roles:

* Administrator:
* Manage users: Edit all the information related to the users and teachers such as personal info, role (user, teacher or admin), and the type of subscription whether it “All Classes” or “Workout Only”.
* Check classes: Check the daily schedule for each class by generating reports so the admin can designate the instructors for the classes.
* Generate Invoice: Manage and generate invoices for each user, whether if it is “Enrolment Fee” or “Monthly payment”.
* Teacher:
* Check classes: Check the daily schedule for each class by generating reports so the admin can designate the instructors for the classes.
* User:
* My bookings: Check all the classes and workouts booked since the user joined the gym.
* Payment Enrolment fee: Redirects the user to the payment page to pay the enrolment fee.
* Payment Monthly payment: Redirects the user to the payment page to pay the monthly payment.
* All roles:
* Edit Settings: Everyone with access to the website can edit the system settings regarding the email and password.
* Edit Profile: This section allows the user to provide the personal info and the type of subscription (Workout only or All Classes).

## Why is this a good project?

Apart from being a practical example of some of the many technologies I got in contact with during this course, I believe this is a very useful and flexible project in which can be applied to many scenarios, not only for gyms but all the businesses that were somehow affected can make use of the booking system with a few changes.

### Main goal:

Develop a booking system web-based application mirroring a real-life scenario.

### Objectives:

* Research of all new technologies to be used in the project;
* Build a consistent backend;
* Build a modern frontend design;
* Test the system as a whole;
* Make a detailed report about the project;
* Extra and current features.

### Areas to cover

This project will have the same structure of a monolith1 making use of some of the following frontend technologies:

* HTML (HyperTextMarkup Language);
* SAAS CSS (Cascading Style Sheets);
* Javascript.

All the logic behind the website was built on the backed part and counts with the technologies mentioned below:

* Python, as the programming language;
* Django, as the web framework;
* Model-View-Template (MVT) as the architectural pattern.

Regarding the database:

* Relational Database (POSTGRESQL), to store the website data
* pgAdmin software to manage the data from POSTGRESQL;
* Relational Database (SQLite) for tests.

Other technologies:

* HTTP Methods;
* Heroku (PaaS platform).

Management techniques:

* Waterfall methodology;
* Use of the Gantt chart.

### Personal challenges

The following challenges were faced while doing this project:

* PostgreSQL database connection;
* Website deployment;
* Work with a real payment process platform (Stripe);
* Generate PDF reports;
* Deal with web framework documentation (Django).

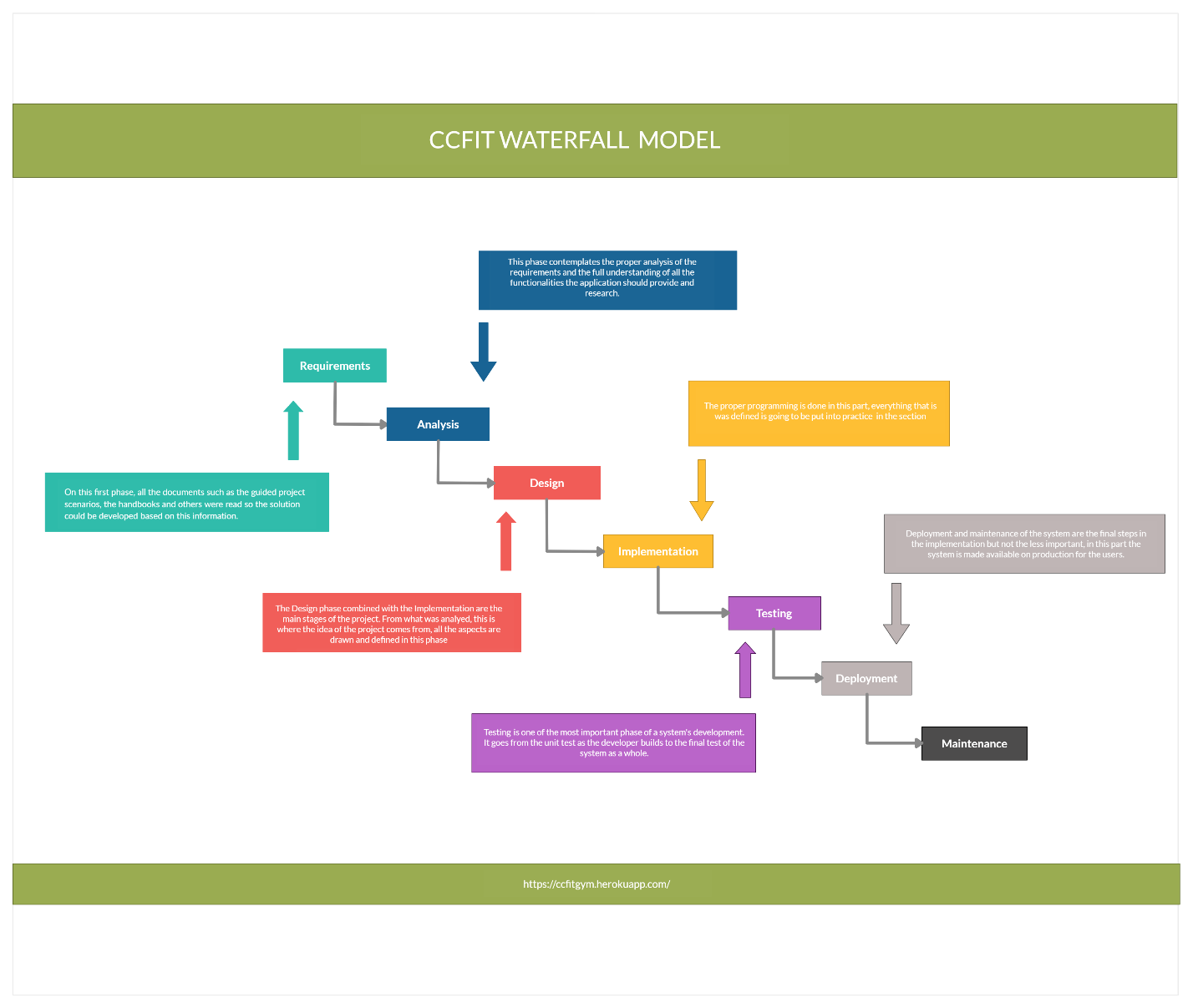
## Project Plan:

Taking into account that I have previously worked with most of the technologies I used for this project and that it’s a small-sized project, I made use of only one project management methodology as great part of the project was planned in mind before even start. For that reason, the Waterfall methodology was used when structuring the project, I believe that this linear and straightforward approach would meet all my needs.

### Waterfall methodology:

“The waterfall model has this name because each phase of the project unfolds in the next, continuing downwards like a waterfall”, according to (ProjectManager, 2007). The methodology was applied to the project based on the following seven pillars:

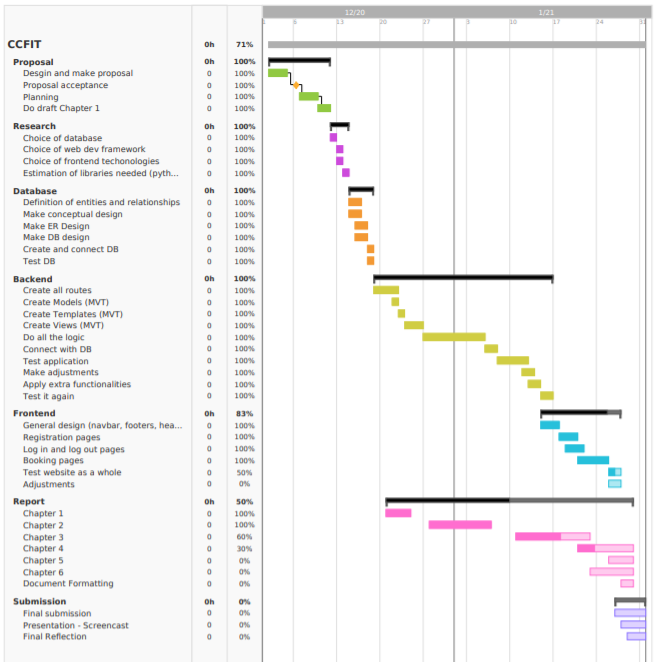
* **Requirements**: On this first phase, all the documents such as the guided project scenarios, the handbooks and others were read so the solution could be developed based on this information.
* **Analysis**: This phase contemplates the proper analysis of the requirements, research, and a full understanding of all the functionalities the application should provide.
* **Design**: The Design phase combined with the implementation are the main stages of the project. From what it was analysed, this is where the idea of the project came from, all the aspects were drawn and defined in this phase.
* **Implementation**: The proper programming is done in this part, everything that it was defined was put into practice in the section.
* **Testing**: Testing is not left aside as being extremely important to the system's development in general. It went from the unit test as I was building to the final test of the system as a whole.
* **Deployment**: Deployment and maintenance of the system are the final steps in the implementation but not the less important too, in this part the system is made available on production for users.
* **Maintenance**: This is not included in the scope but as this is a project that will kept being worked on afterwards, the maintenance of the system will be provided to users when the system is made available.



**Illustration 1: ccfit waterfall model**

### Planning process

For this project, the Gantt chart technique was used to gather and manage all the requirements and tasks ahead. The sequence had to be carefully thought and analysed so the planning could be followed and the deadlines met.



**Illustration 2: Gantt chart**

# CHAPTER II: LITERATURE REVIEW

The chapter aims to present the academic research carried out, in which is the base of all the decisions taken concerning this project throughout the project cycle. The following four topics bullet-pointed are considered the ones with most importance for the project development:

* Backend;
* Frontend;
* Database;
* Deployment;
* Additional features.

Even though many other topics were researched, these four were by far the most important for the development process combined with the planning as the concepts were being understood.

## Backend:

As being a crucial component for system development in general, the language used in the backend was carefully chosen after reviewing what it was learned during the whole year and analysing which one would allow me to meet to the project requirements and add features in a timely manner. In order to achieve these goals, I decided to use python as the language used to develop the web-based application.

### Python — Programming with ease.. Ep -2 | by Junaid KHAN | MediumPython

First of all, Python is a general-purpose language, which means that it can be used for different types of programming: Desktop GUI applications, websites, web application and scientific and numeric research (data science). Secondly, I chose python owing to the fact that I have previously worked with it and the stack used in this project for other web-based applications, I am very keen on the data science field where python is predominant.

1

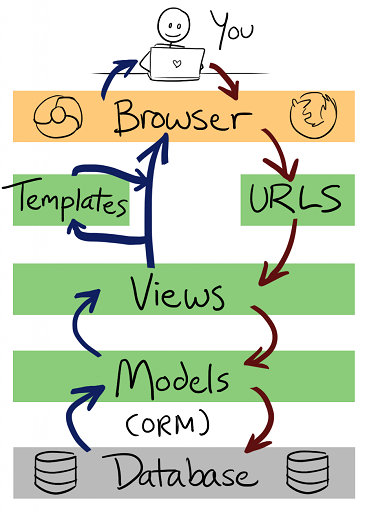
1 Source: <https://medium.com/@junaidapj/python-programming-with-ease-7561313490ca>

### Django

Django is the web framework used for this project. A web framework is a set of packages and modules that provide the developer with a faster web development process. By its own definition, “Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design” (Django, 2005). It’s considered the most famous and used python web framework nowadays, its main characteristic and advantage over the other frameworks is the MVT design pattern.

### MVT (Model-View-Template)

Web-based applications are usually projects that require many components, most of the times it’s needed to have a database system behind to store the data and retrieve later on. In other occasions, the visual part of the website is essential, an e-commerce for instance where the user needs to have a navigable environment. For these and other reasons that the MVT pattern exists, it’s necessary to highlight that the MVT differs from the MVC (Model-View-Controller) in some aspects:



* **Model**: The model layer is responsible for managing the data through the application where all the data displayed or processed comes from, it’s usually associated with e database system files;
* **View**: The view is where all the logic stays, it’s responsible not only for keeping the functions but doing the webpages routing from the system to the browser;
* **Template**: The template layer function is pretty straightforward as its task is receiving data from the views and general design (HTML/CCS and others).

2

## Frontend:

From everything that it was learned throughout this year, the basics of the frontend are HTML, CSS and JavaScript concerning the DOM3 are definitely the most important when designing your webpages. There are also some frameworks that provide ready components for customization such as Bootstrap, Material UI, Bulma and others. 4

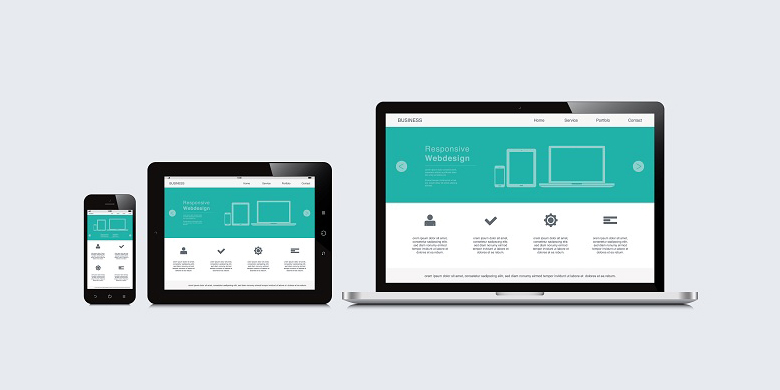
2 Source: <https://medium.com/@humble_bee/django-basics-for-a-beginner-5d864e6aa084>

3 DOM: The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.. (Robie, 2001)

4 Source: <https://medium.com/level-up-web/amazingly-useful-html-css-and-javascript-tools-and-libraries-d73b10fbae29>

### HTML/CSS

The design of the website was built without using any framework, this choice was made because it’s usually hard to custom the style of a webpage when using a framework as the code is provided ready for example nav bars, menus and grids. When the template is built from scratch there is way more liberty to draw the page the way is desired, even though it’s more arduous work.

Another crucial aspect that it was implemented to the project is the responsive website concept which is the compatibility between all screen sizes when opening the website in the browser, that means that the components of a webpage are readjusted from the mobile device to the Desktop as well as the opposite.

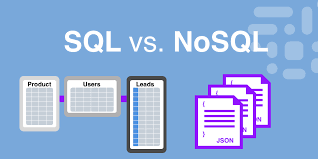
5

### JavaScript

Not much JavaScript was used in this project as many libraries from the pip were installed and handled using python. However, the following were used in some part of the code to make some small adjustments on the frontend:

* **Ajax**6**:** Send asynchronous data from a certain type of date (date) to the server-side;
* **Jquery**7**:** Handle placeholders for input type date fields.

## Database:



For the database choice is always good to have in mind what type of database will meet the project needs regarding if it’s relational or non-relational. Both have similarities but there are some variants that make plenty of difference when choosing one or the other.

8

5 Source: <https://freshsparks.com/why-responsive-design-is-important/>

6 Jquery: jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. (jquery, 2015)

7 AJAX: AJAX = **A**synchronous **J**avaScript **A**nd **X**ML. AJAX is not a programming language. AJAX just uses a combination of: 1. A browser built-in XMLHttpRequest object (to request data from a web server) 2. JavaScript and HTML DOM (to display or use the data). (AJAX, 2015)

8 Source: <https://acodez.in/sql-and-nosql-an-overview/>

* **Relational**

Relational databases or SQL databases are systems that store data in form of tables, columns and rows, it is called SQL database because it can be queried using SQL (Structured Query Language). It works by creating tables with different schemas9 and linking them between keys. For these reasons is considered more organized and structured than the Non-relational (Pawlan, 2017).

* **Non-Relational**

On the contrary, non-relational databases don’t build different schemas for each schema, instead, it creates one document that stores the data in the form of JSON10 document and a key for each one of them. This allows the system to have more flexibility and adaptability as it doesn’t make any use of queries for different schemas (Pawlan, 2017).

For this project the type used was SQL databases for the following reasons:

* Handles SQL which facilitates when testing (Data manipulation);
* Great integration with the framework choice;
* Much more experience using relational databases;

During the development of the project two SQL databases were used, SQLite and PostgreSQL. In the beginning, the plan was to use SQLite as it comes when downloading Django but as the development process went by and new features were being implemented there was the need on changing for a better database.

### SQLite

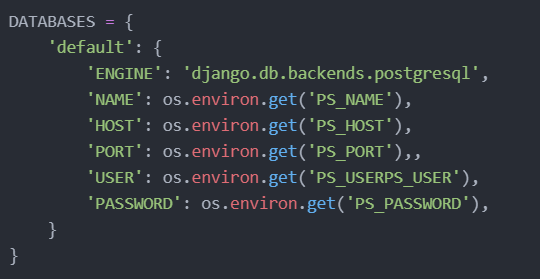
SQLite “is a local file as [Django is a server-side framework](https://data-flair.training/blogs/django-tutorial/) and it treats your computer as the host when you actually run the server in command line/terminal”, (Flair, 2019). The SQLite is an encoded file which it’s very useful and doesn’t require any setup, it was used during 50% of the implementation until the need of changing to a database that can be deployed (additional feature) came up. As it was researched and defined that the deployment platform would be Heroku, the change was successfully made.

### PostgreSQL

PostgreSQL an open-source relational database management system that is used by many small and medium-sized companies. One of the main reasons for using PostgreSQL is because of its open-source nature makes it easy to upgrade or extend. In PostgreSQL, you can define your own data types, build custom functions, and even write code in another programming language (e.g. Python) without recompiling the database, (Romanowski, 2020). After changing to PostgreSQL, the website had everything available online, without any local dependencies.

9 schema: is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. (White, 2020)

10 JSON: (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript programming language. (JSON, 2005)



**Illustration 3: postgresql settings**

## Deployment:

Deployment is the process of making a system or service available for users, it’s a common concept between IT professionals as it’s considered the last phase of projects. This process includes a series of measures that need to be taken to be successful.

Developers nowadays can have their applications deployed either choosing a host application11 using its own infrastructure or hiring a cloud provide services (IaaS12/PaaS13) such as GCP (Google Cloud Platform), Microsoft Azure, Amazon Web Services (sumologic, 2021) or Heroku itself which can be used for free to a certain extent.

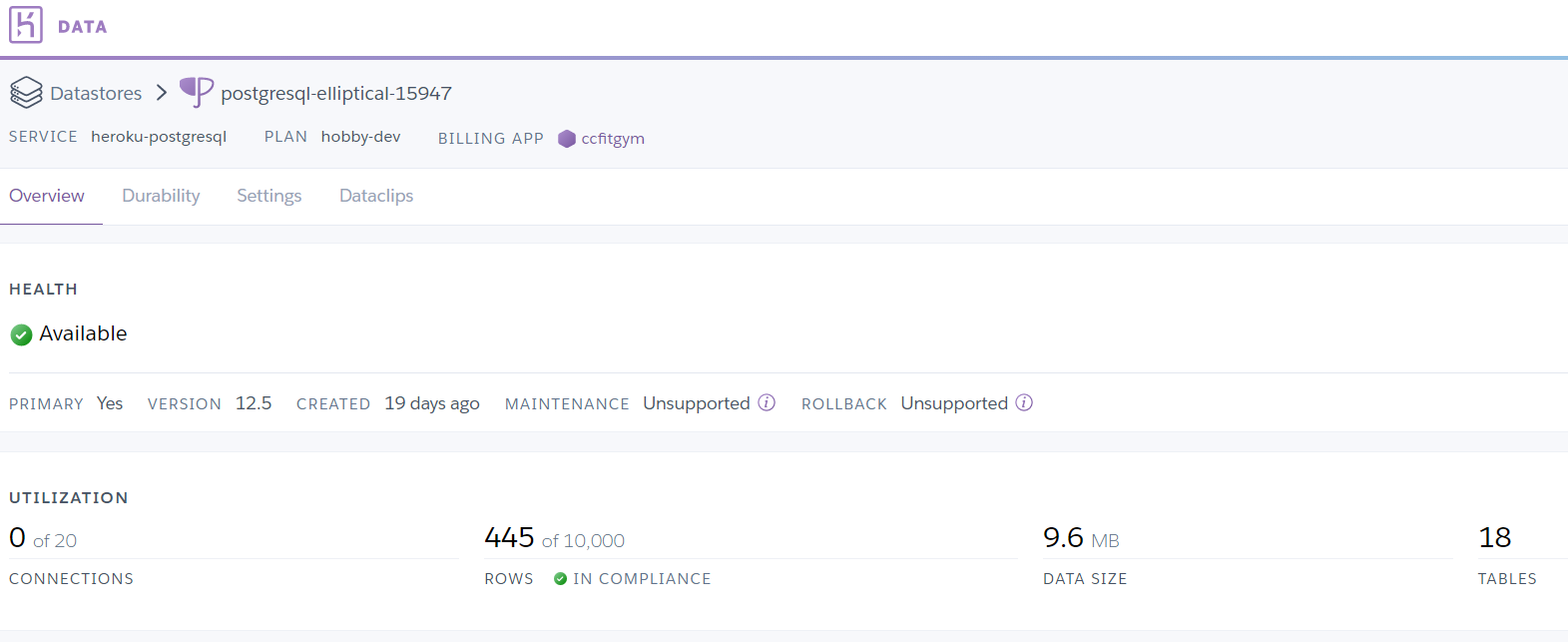
### Heroku

Heroku is a container-based cloud Platform as a Service (PaaS) that according to (Chai, 2021) is a cloud computing model where a third-party provider delivers hardware and software tools to users over the internet. Usually, these tools are needed for application development. Heroku was the platform I used to host the CCfit web application, it provides an environment free with many resources for several types of programming languages, including python. I could not only deploy my application but connect to a free instance of a PostgreSQL connection, which allowed me to have the live database both test and production environment.

11 host application: A hosted application is any piece of software that is running on another provider’s infrastructure rather than on-premise. (Intricately, 2020)

12 IaaS: [Infrastructure-as-a-Service (IaaS)](https://www.ibm.com/cloud/infrastructure) is a cloud-computing offering in which a vendor provides users access to computing resources such as servers, storage and networking. Organizations use their own platforms and applications within a service provider’s infrastructure. (IBM, 2017)

13 PaaS: [Infrastructure-as-a-Service (IaaS)](https://www.ibm.com/cloud/infrastructure) is a cloud-computing offering in which a vendor provides users access to computing resources such as servers, storage and networking. Organizations use their own platforms and applications within a service provider’s infrastructure. (IBM, 2017)



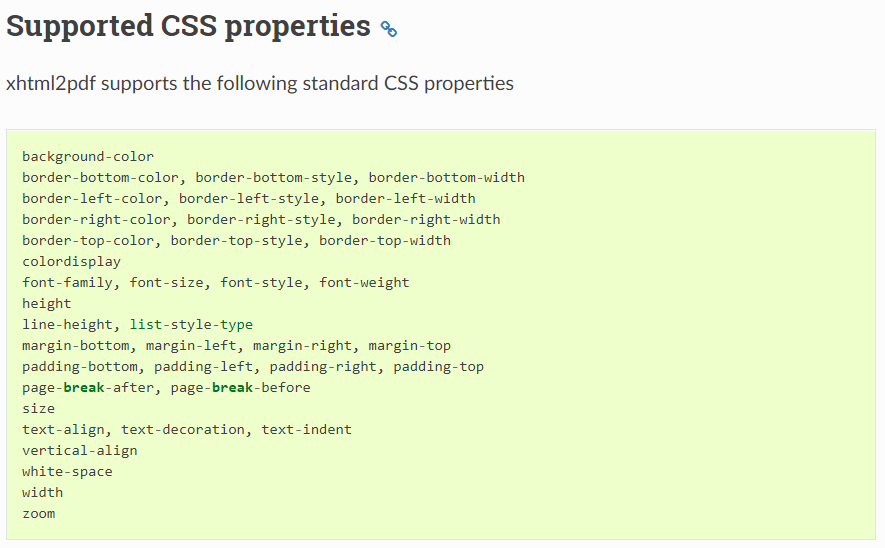
**Illustration 4: postgresql/Heroku set up**

## Additional features:

For this work, some extra functionalities were implemented in the project to make the system as integrated as possible with other systems and platforms to provide the user with a better experience.

### xhtml2pdf

Xhtml2pdf is a library of the python package manager whose main function is to generate pdfs for development and projects using the Django framework. The code for its use is very simple and is provided through the link <https://xhtml2pdf.readthedocs.io/en/latest/>. Another important detail regarding the use of this library is that the pdf layout is assembled using the knowledge of HTML and CSS, but it is also important to remember that not all CSS attributes are supported as shown in the example below.

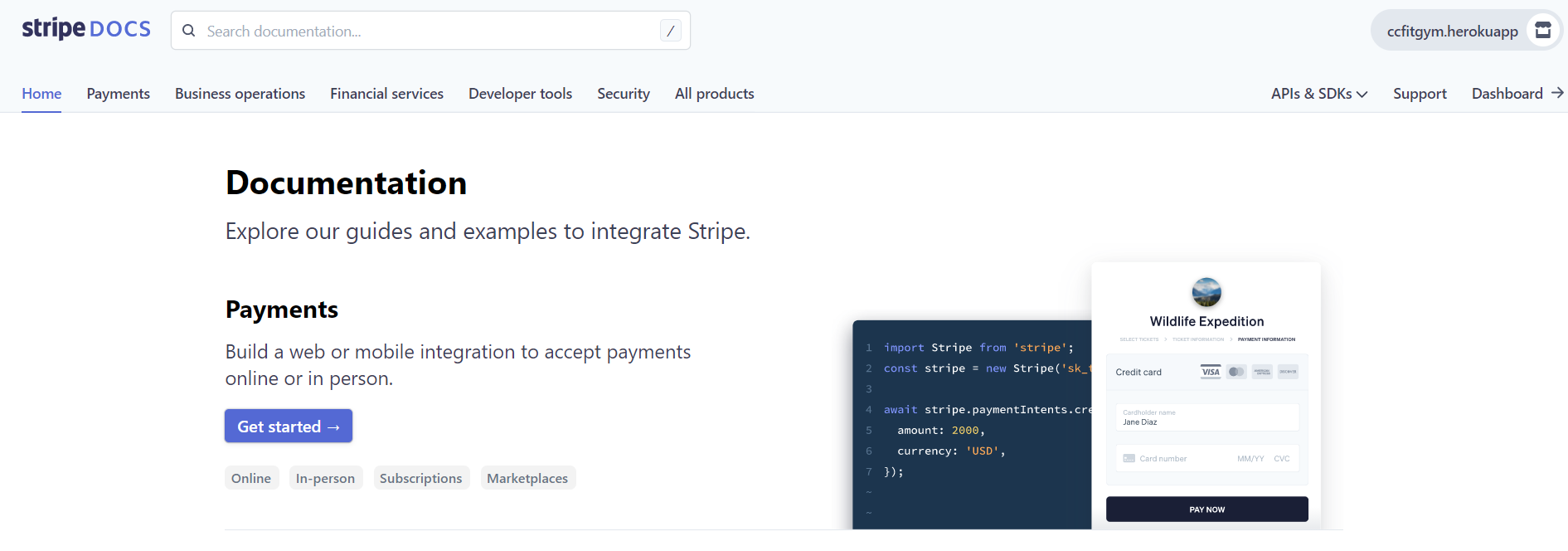


**Illustration 5: supported css properties**

### Stripe

The payment method platform used for this was Stripe, which provides a ready API so that developers can work with the system and implement it in their projects. It is a system that does not require any fee and can be associated with small, medium and large companies.

The platform also offers two types of environments, the test and the production so that the developer can make the validations in the code before the system is ready for use.



**Illustration 6: stripe documentation**

# CHAPTER III: SYSTEM ANALYSIS & DESIGN

CCfit Gym is the web-based application built for gyms owner using the Pandemic as the scenario where the owners have to meet the government requires and manage the entry of users, the number of users per session and allocate instructors for these classes. This chapter will describe in detail the whole process for users, teachers and the website administrator.

## Functional Requirements:

Considering that there are three types of users (user, teacher, admin):

### Main Requirements

* Users and teachers must sign up to the system (admin already registered);
* Everyone must edit profile page as soon as they sign in;
* Users need to pay the enrolment fee to book classes;
* Admin needs to set the instructors as “teachers” in the system as soon as they sign in;
* Only users can book classes;
* Users can check all their own bookings;
* Admin and teachers can check any session and class at any day;
* Admin must generate invoices for users;
* Users can print the invoice and pay in person or by online payment method (Stripe)
* Admin can manage users (edit personal information)

### Subsystems requirements

REGISTRATION – SIGN UP

**Admin**

As this application was thought for one particular admin ([ccfitgym@gmail.com](mailto:ccfitgym@gmail.com)), it won’t be necessary to sign up as its data will be set up in the database since the very beginning.

**Teacher | Users**

Users and Teachers will need to create an account through the signup page providing the following information:

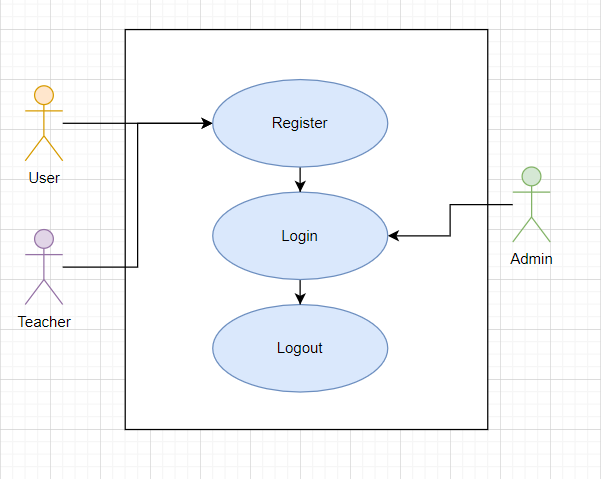
* E-mail address;
* First name;
* Last Name;
* Password.

LOG IN

**Admin | Teachers | Users**

Everyone one needs to provide the info down below in order to log in to the system;

* E-mail address;
* Password.



**Illustration 7: log in**

EDIT PROFILE

**Admin | Teachers | Users**

Everyone needs to edit their profile page as soon as they log in to the system, in this profile page section they need to provide:

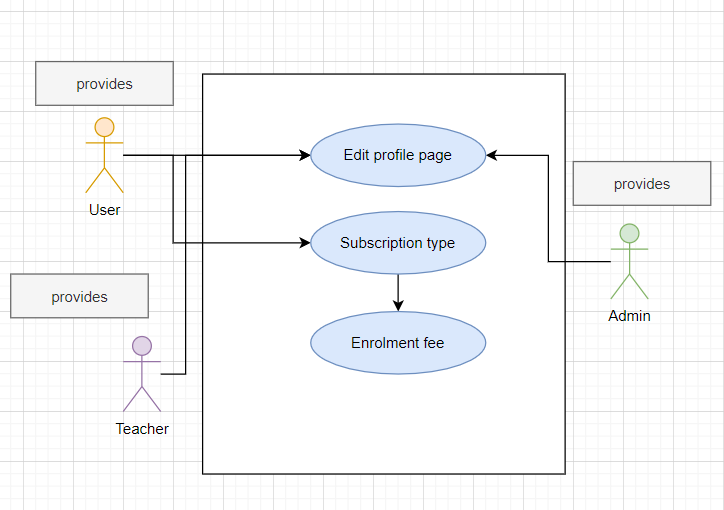
* Nickname;
* Gender;
* Birthdate;
* Address;
* Complement;
* County;
* Country;
* Prefix;
* Phone number;
* **Subscription type** (USERS ONLY).
  + Workout Only - €35
  + All Classes - €50

**Subscription type** is the field in which the user will define and will have its enrolment fee and access to the system based on.

PAYMENT: ENROLMENT FEE

**User**

As soon as the user logs in and finish editing the profile page information, a button named “Enrolment Fee” will appear in the home screen and the user will be able to click, print the invoice, download it and pay via Stripe if desired.



**Illustration 8: enrolment fee payment**

MAXIMUM USERS PER SESSION

**Admin**

There is a table used for informing the maximum number of people for a certain class and it’s also set up already. The only one allowed to modify this information is the admin trough the Django admin section.



**Illustration 10: MaxSession table**

This is the pattern established by the owner but it can be changed any time and the logic of number maximum of users per session will follow this info.

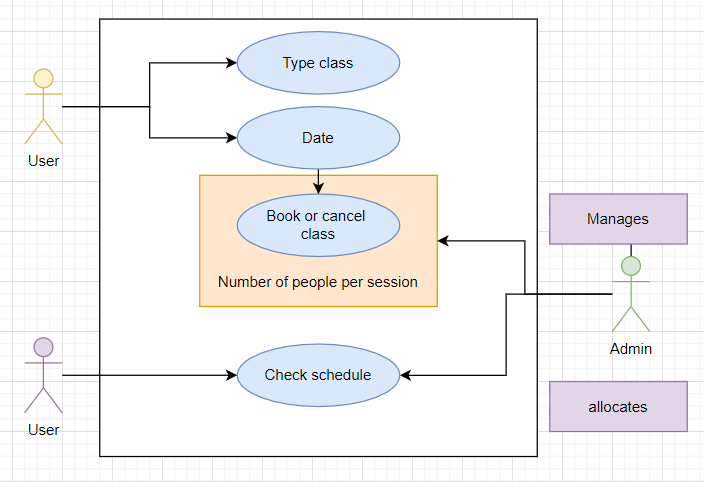
BOOK A CLASS

**User**

After paying the enrolment fee and receiving the email confirming the payment the user is able now to book classes according to the type of subscription provided. If it is a workout only subscription, the button for the other classes will be disabled whenever the tries to book a class.

There are a few conditions for some actions that the user will not be able to perform:

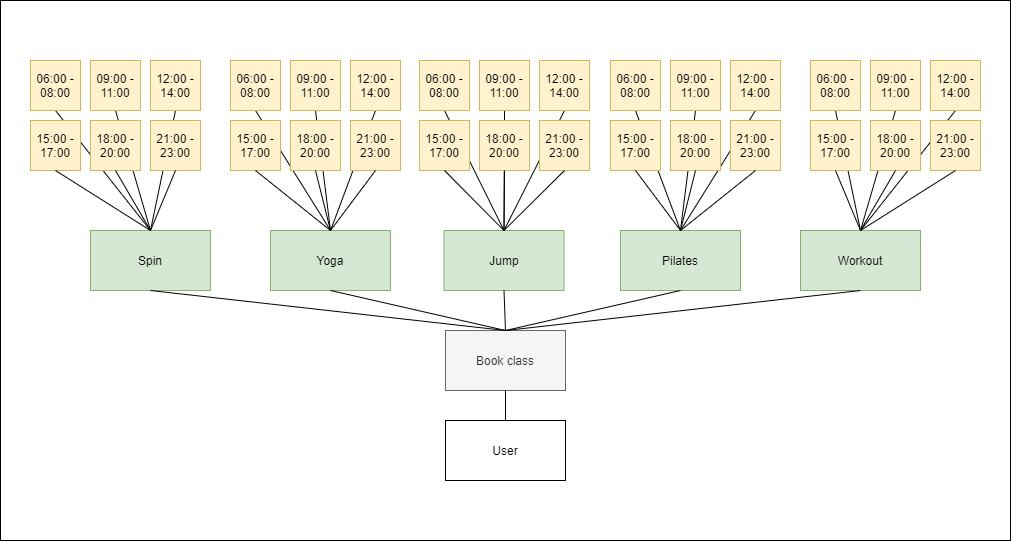
* The user is not allowed to book more than one class per day;
* The user must cancel a class in order to book others on the same day;
* The user can only book a class until 30 min before the session starts;
* If the session is full the user will not have access to book that session but if someone cancels it, a slot will be available for users to book.
* As long as the number max of people is not reached, the users can book a class.



**Illustration 11: book a class**

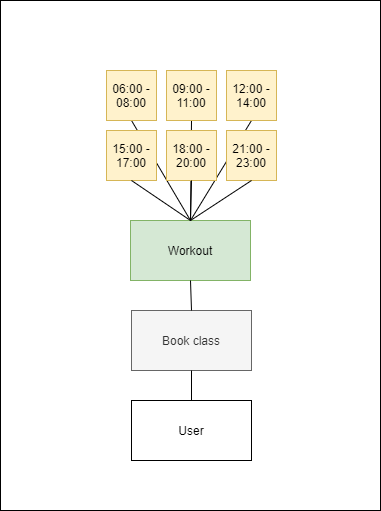
The following flowchart represents the relation between class and sessions from the user view when he has both “All Classes” and “Workout Only” subscription:

**All Classes**



**Illustration 12: “All classes” relation**

**Workout Only**



**Illustration 13: “Workout only” relation**

GENERATE INVOICE

**Admin**

The admin has access to the generate invoice page that allows him to manage invoices for:

* Enrolment fee
  + View invoice
  + Mark as paid
* Monthly payment
  + View invoice
  + Sent to user
  + Mark as paid

The **Enrolment fee** is generated once when the user finishes registering and informing the personal info, whereas the **Monthly Payment** is generated every month when the user books the class for the subsequent month.

Beforehand, it’s important to have in mind that there is a table called invoice which handles all the invoices for all the users, and to manage the payments there is a field named “status” and its possible values are Generate, Requested and, Paid.

By clicking in **View Invoice,** the admin will have access to the pdf containing the user invoice. The Invoice table has a field called status and the **Mark as Paid** button sets the payment status to paid which means that the user has paid in person and not via stripe. The **Send to user** button sets the status from generate to requested and sends an email to the user notifying that the invoice was generated.

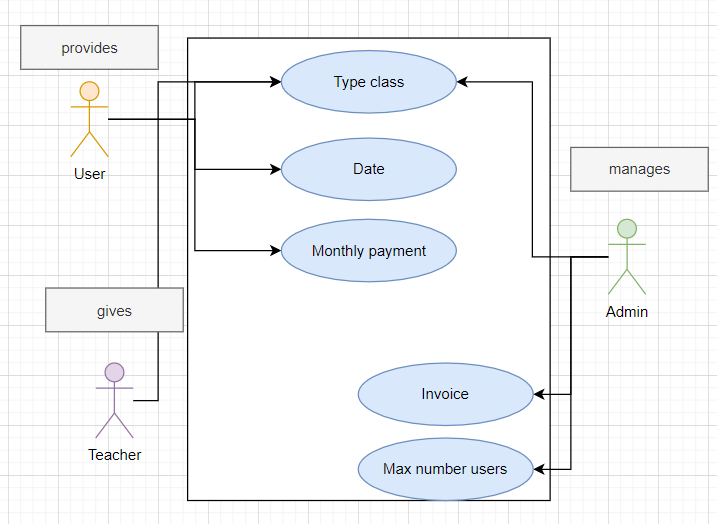
PAYMENT: MONTHLY PAYMENT

**Admin**

In regards the monthly payment, the admin monthly sent the invoice to the user by clicking in the “Send to user” button as said in the section above and mark as paid also pressing the button in case the user pays in cash, otherwise if the payment is made by stripe the system makes all the validations update the invoice status.

**User**

When the system generates the invoice and the admin sends it to the user, a new button “Monthly payment” is shown on the home page where the user can have access to the pdf invoice and make the payment via stripe.



**Illustration 14: monthly payment**

MANAGE USERS

**Admin**

In this page, the admin can update the users’ information and such as personal data but also the type of subscription in case the user wants to swap from “Workout only” to “All classes” and also inactivate the user blocking him from booking classes in case the left the gym or didn’t pay.

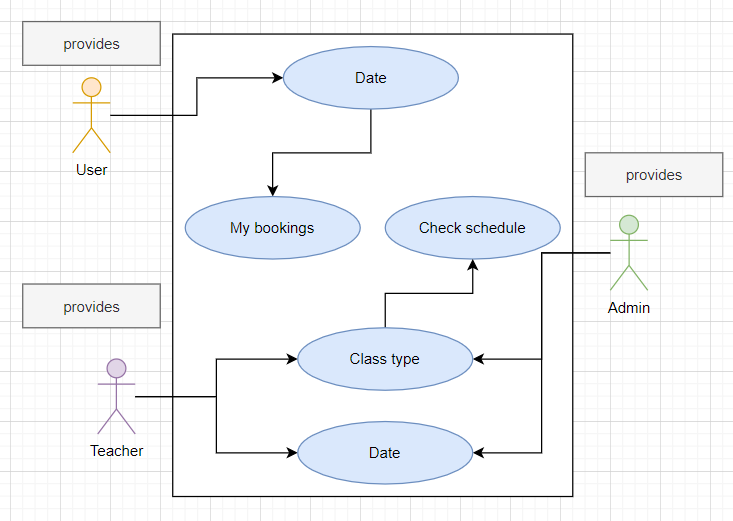
CHECK CLASSES

**User**

The user can check all the session he’s been since he joined the gym by clicking in “My bookings” in the home page

**Admin | Teacher**

The admin and teachers can check the schedule of any particular class at any time any day, generate the pdf schedule and download it by clicking in “Check classes” in the home page



**Illustration 15: check classes**

## 

## User Interface Design:

The user interface was designed with the aim of making the user experience as easy and intuitive as possible. In addition to the registration and login page, all others have the same style, all stored in the base.html file. This file contains the following items:

* Navigation bar;
* Background colour;

The following wireframes show how the all the pages are supposed to look like:

Sign Up

Creation of the user account to have access to the system:



**Illustration 16: sign up page**

Log In

Log in screen with the email and password:

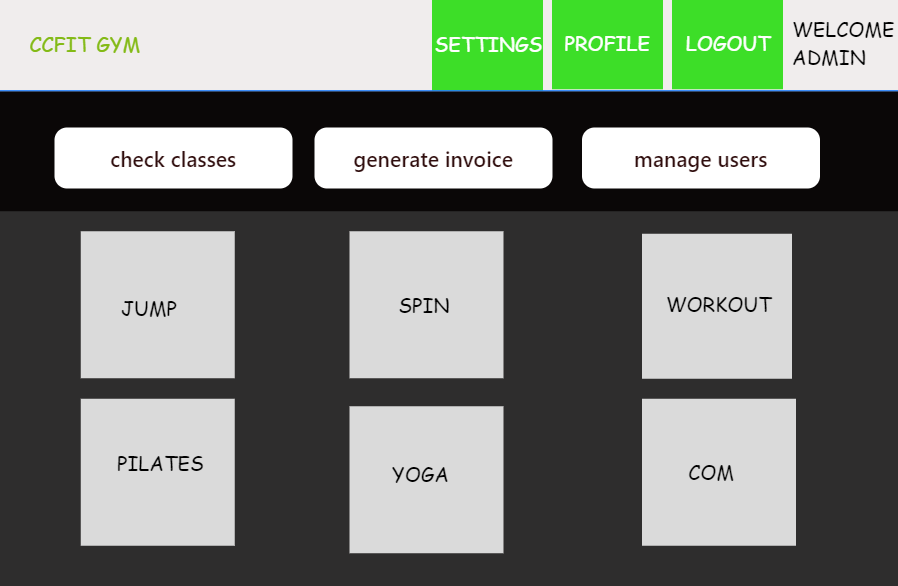


**Illustration 17: log in**

Home Page

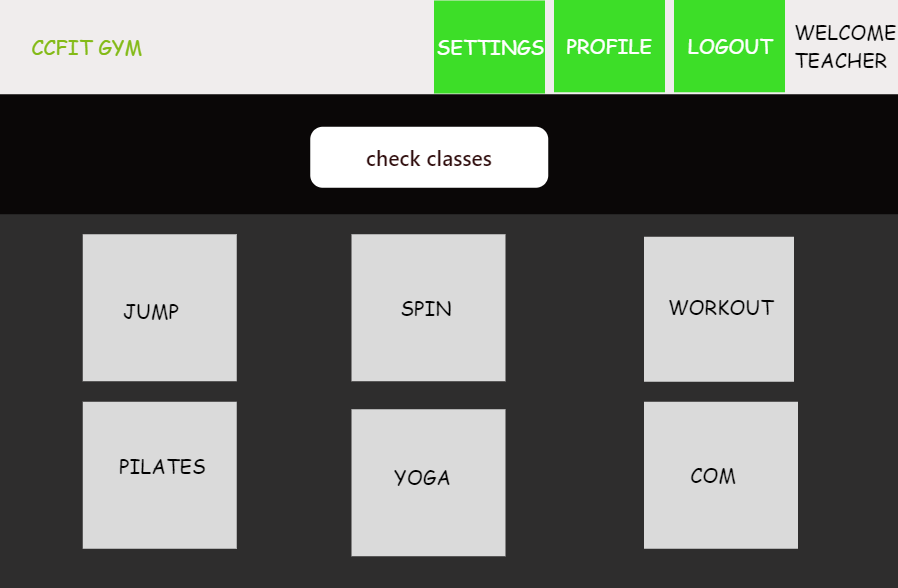
The home page has the same layout for all the users but a few different functions for each type:

**Admin**



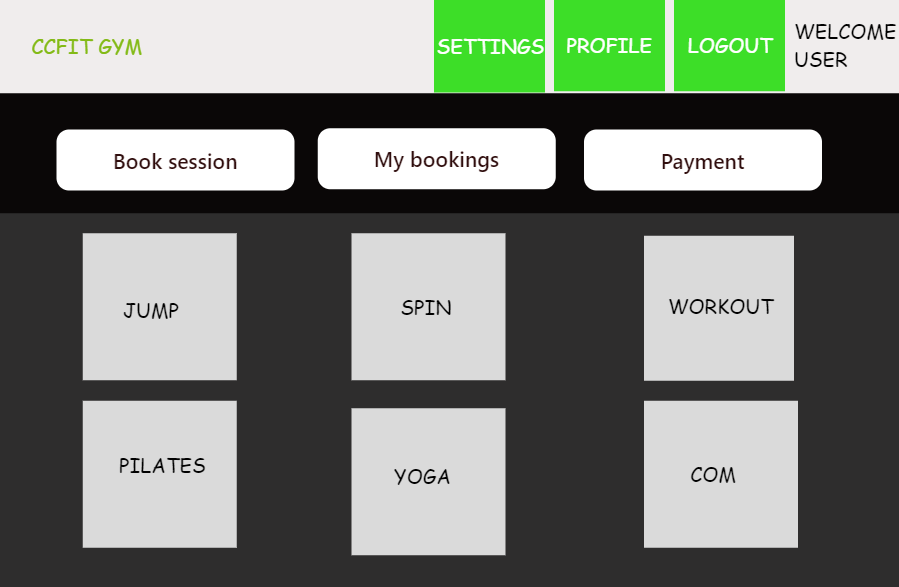
**Illustration 18: user home page**

**Teacher**



**Illustration 19: teacher home page**

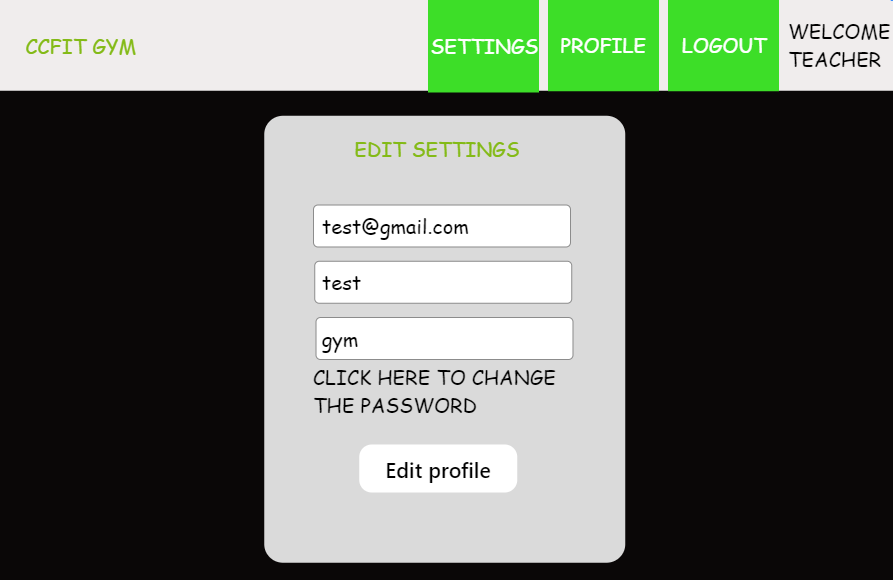
**User**



**Illustration 20: admin home page**

Edit Settings

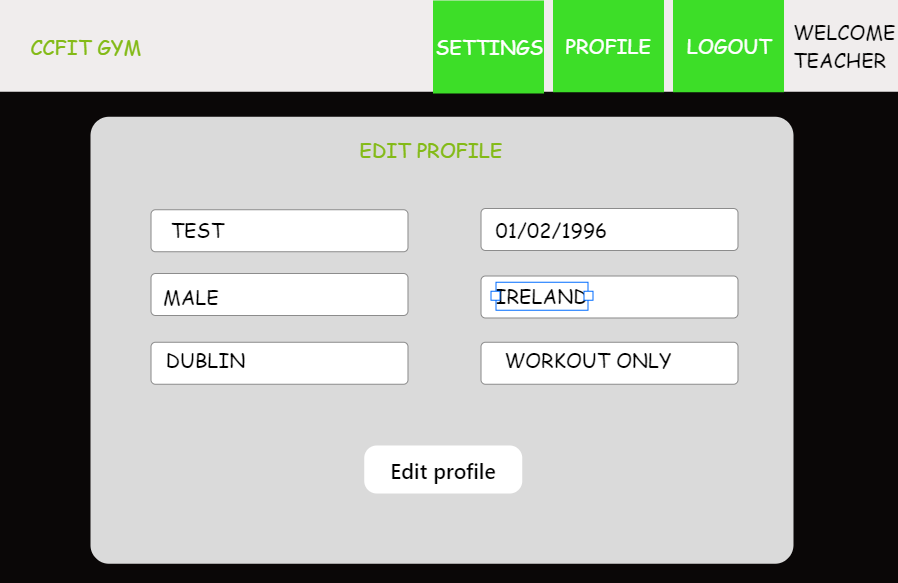
These settings are informed at the signup act but the user can change at any time:



**Illustration 21: edit settings**

Edit Profile

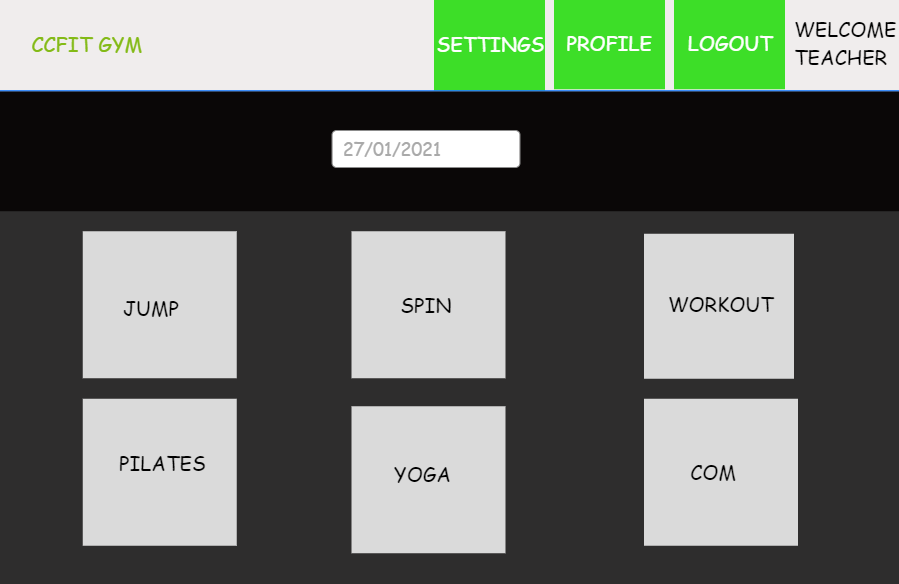
In this section the user provides the personal info to the system. It also provides the type of subscription he will have if either “Workout only” or “All classes”:



**Illustration 22: edit profile**

Booking Page

The user has access to the booking page through the button in the home page that redirects to the below page. The user needs to inform the day he wants to book the class for and after that the buttons of the classes concerning the type of subscription, he paid for will be available for booking:



**Illustration 23: booking page**

Choose Session

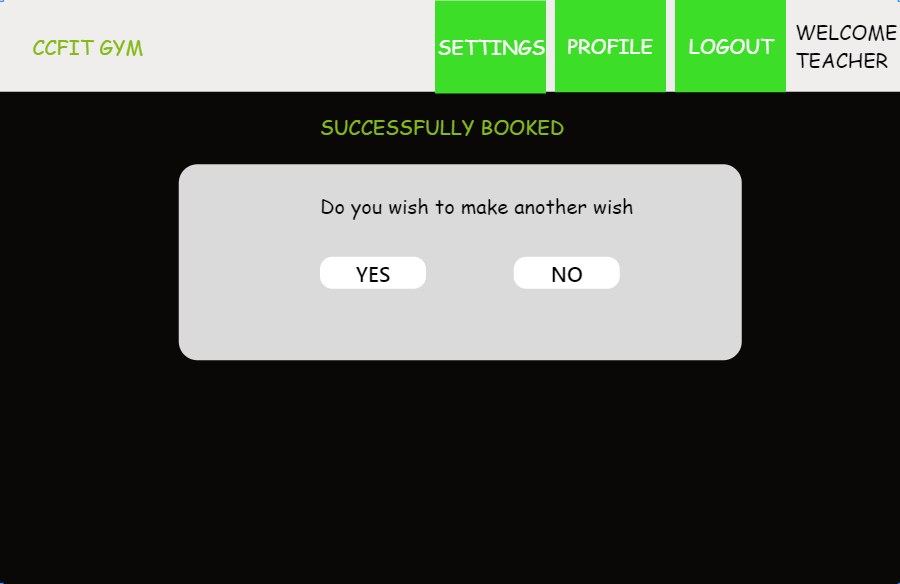
After selecting the classes desired the system redirects the user for the session page, in which the user is ablet to select the time he wants to book the workout:



**Illustration 24: choose session**

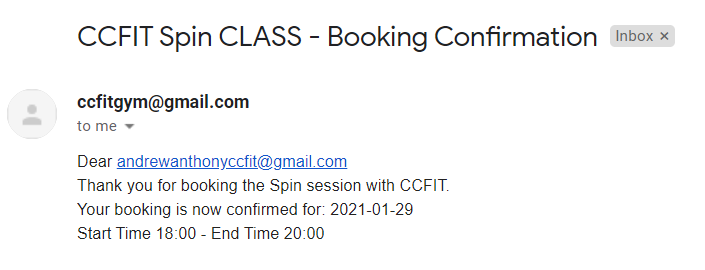
Booking Confirmation

As soon as the user confirm the booking a confirmation message is shown:



**Illustration 25: booking confirmation**

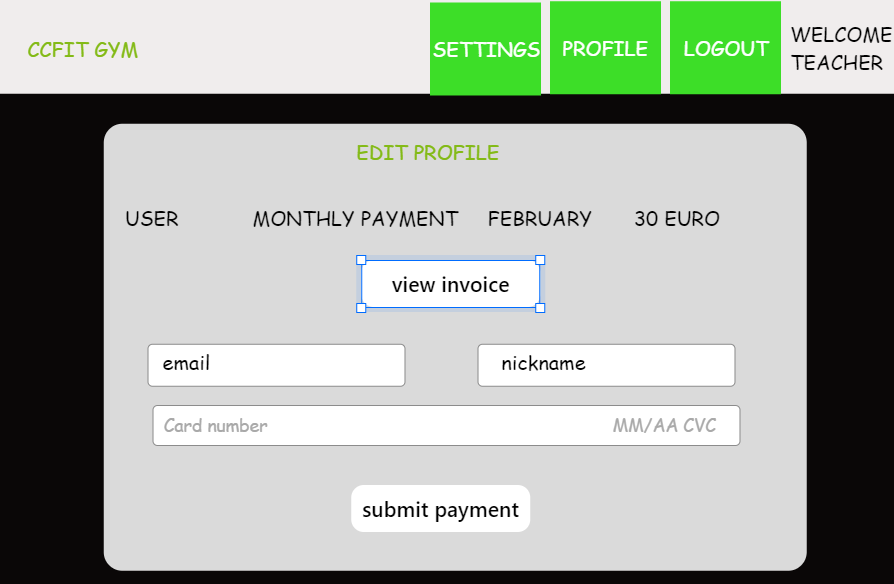
The user also receives an email confirming that he booked a class at the time given.



**Illustration 26: email confirmation**

Payment

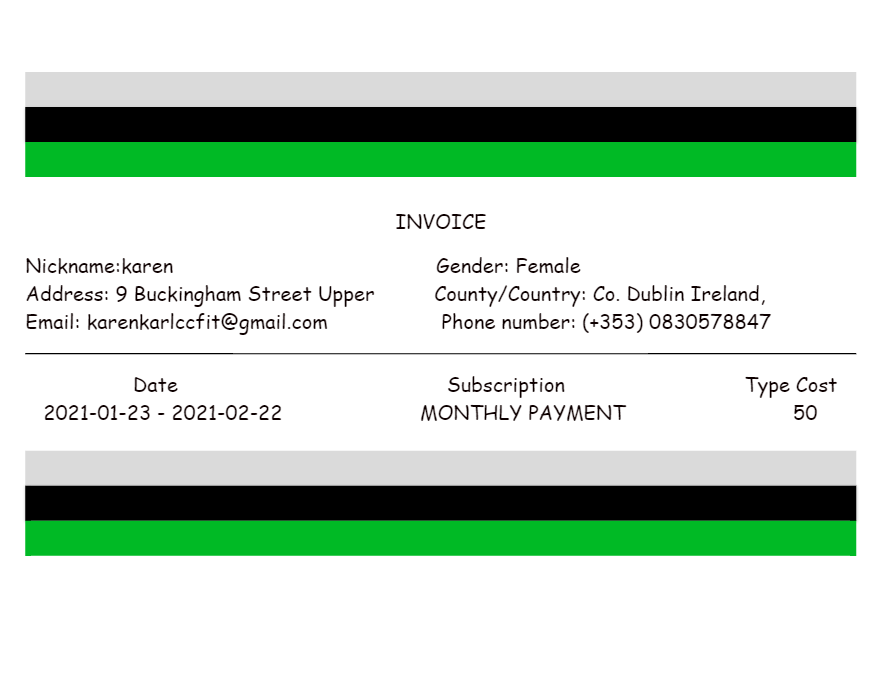
The user can also make payments using the website on the “Enrolment fee” or the “Monthly payment” buttons. It’s only needed to check the invoice and provide the Credit or Debit card details:



**Illustration 27: payment**

PDF Invoice

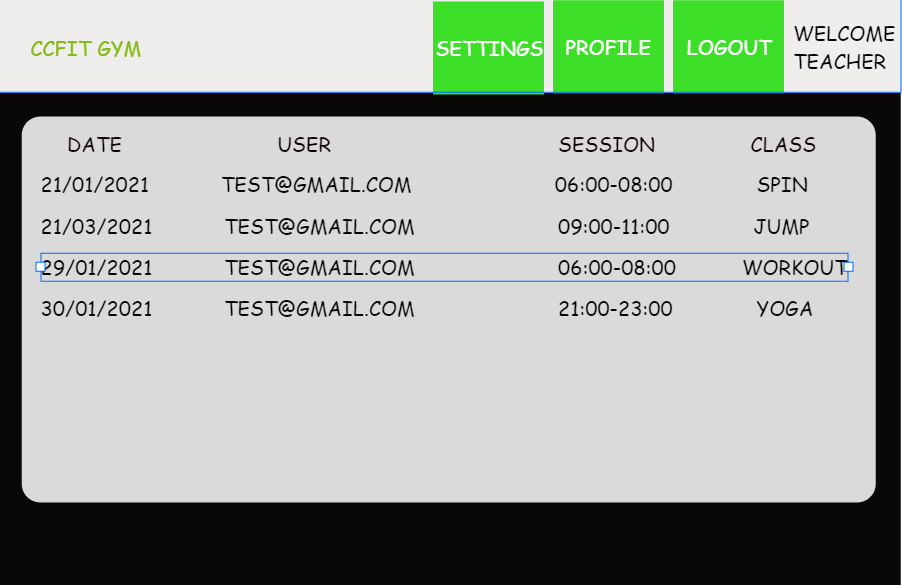
As it is shown in the picture the user has access to view the invoice in a PDF file in case he doesn’t want to pay the invoice via Stripe:



**Illustration 28: pdf invoice**

My bookings

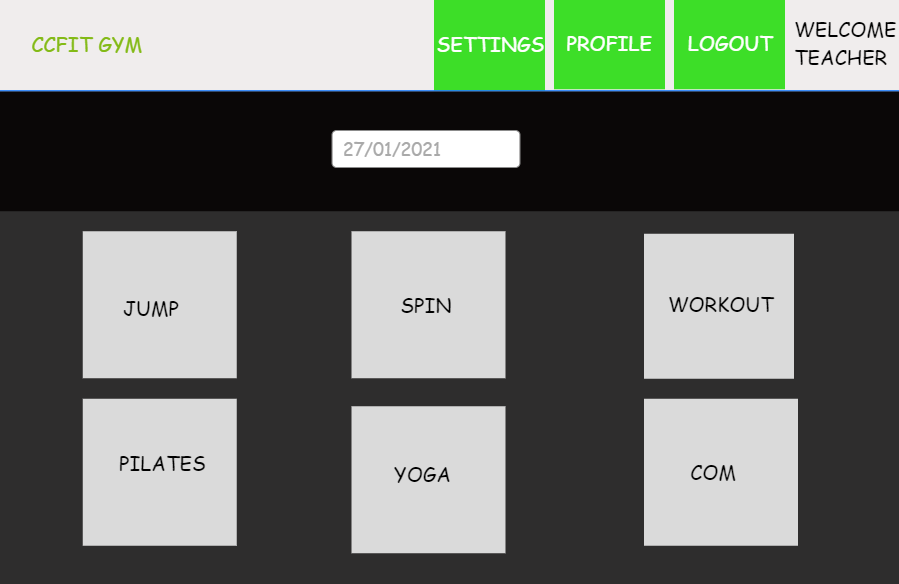
After all this process, the user is able to check the classes in the “My bookings” section:



**Illustration 29: my bookings**

Check Classes

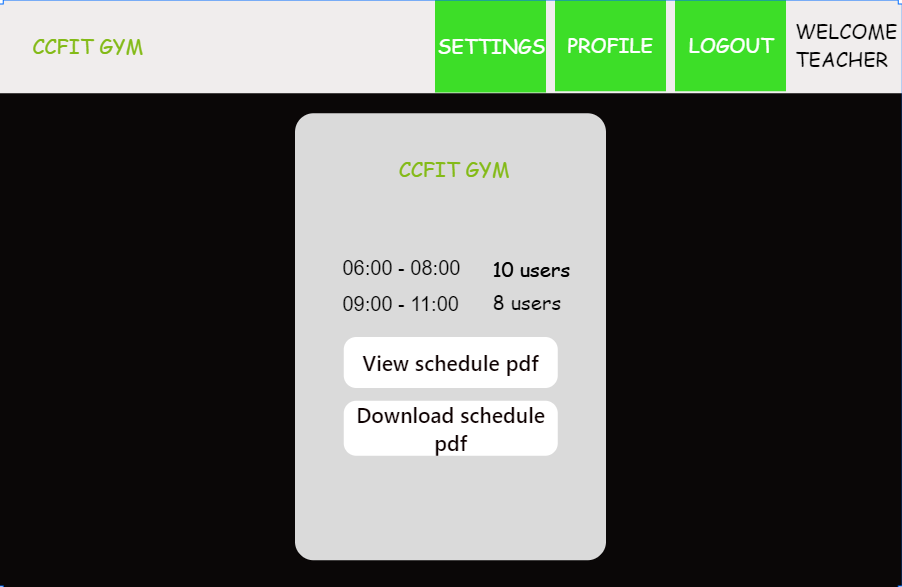
The admin and users have access to a page called “Check classes” in which they can print the schedule of any class at any day:



**Illustration 30: admin check classes panel**

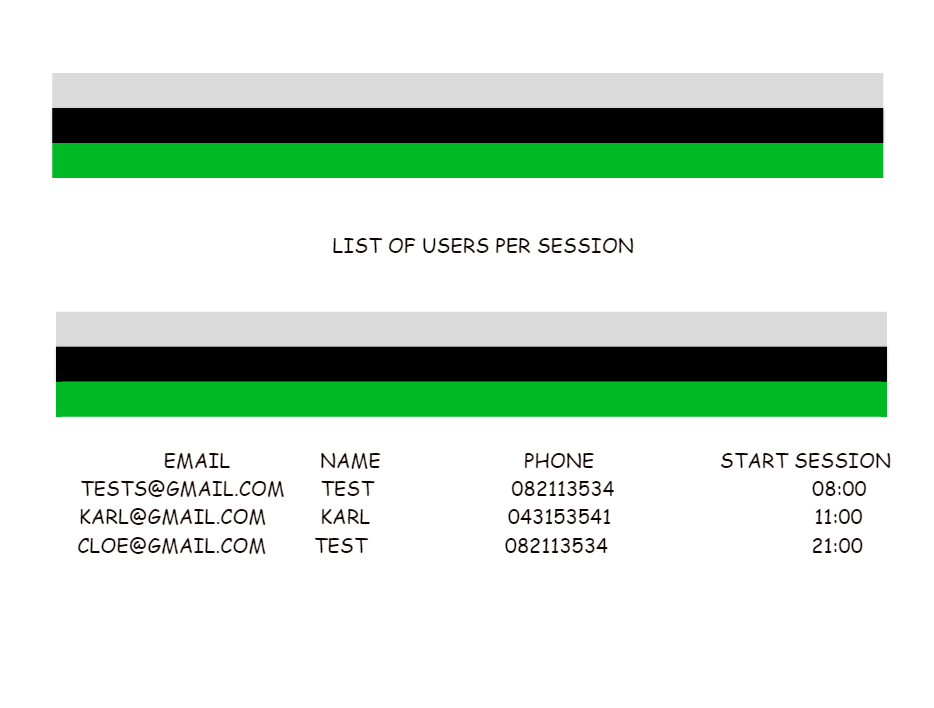
Print Schedule

A report in PDF format is also available in case they want to print out the schedule:



**Illustration 31: print schedule**

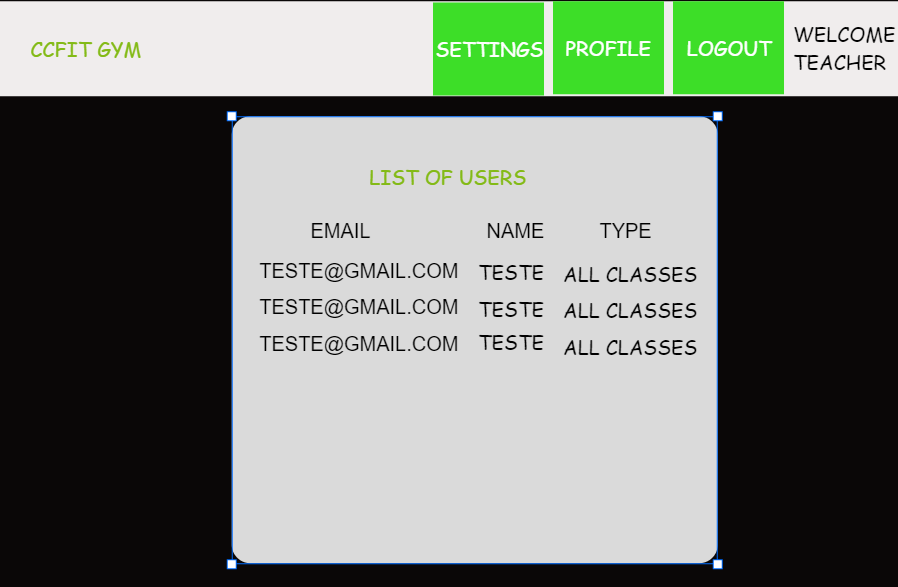
This is the PDF output:



**Illustration 32: pdf output**

Check Session Detail

The check session detail is provided in case the admin just want to see the list on the website:



**Illustration 33: check session detail**

Generate Invoice

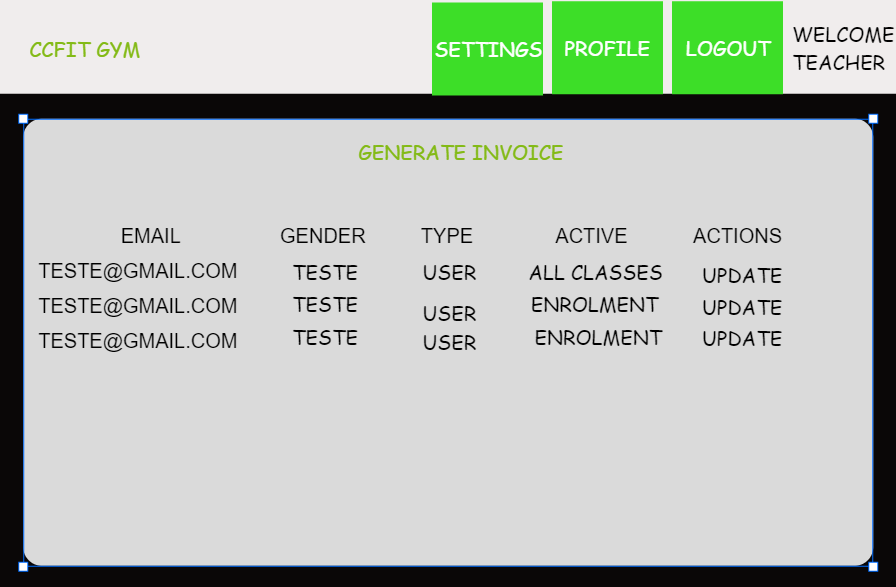
The admin is responsible for managing the invoice for each one of the users through the “Generate Invoice page:



**Illustration 34: generate invoice**

Manage Users

The manage users screen was developed for the user to manage the users and teachers, being able to change their personal and system information:



**Illustration 35: manage users**

Manage Users Detail

All the fields are available to the admin as he manages the whole system through the application:



**Illustration 36: managers users detail**

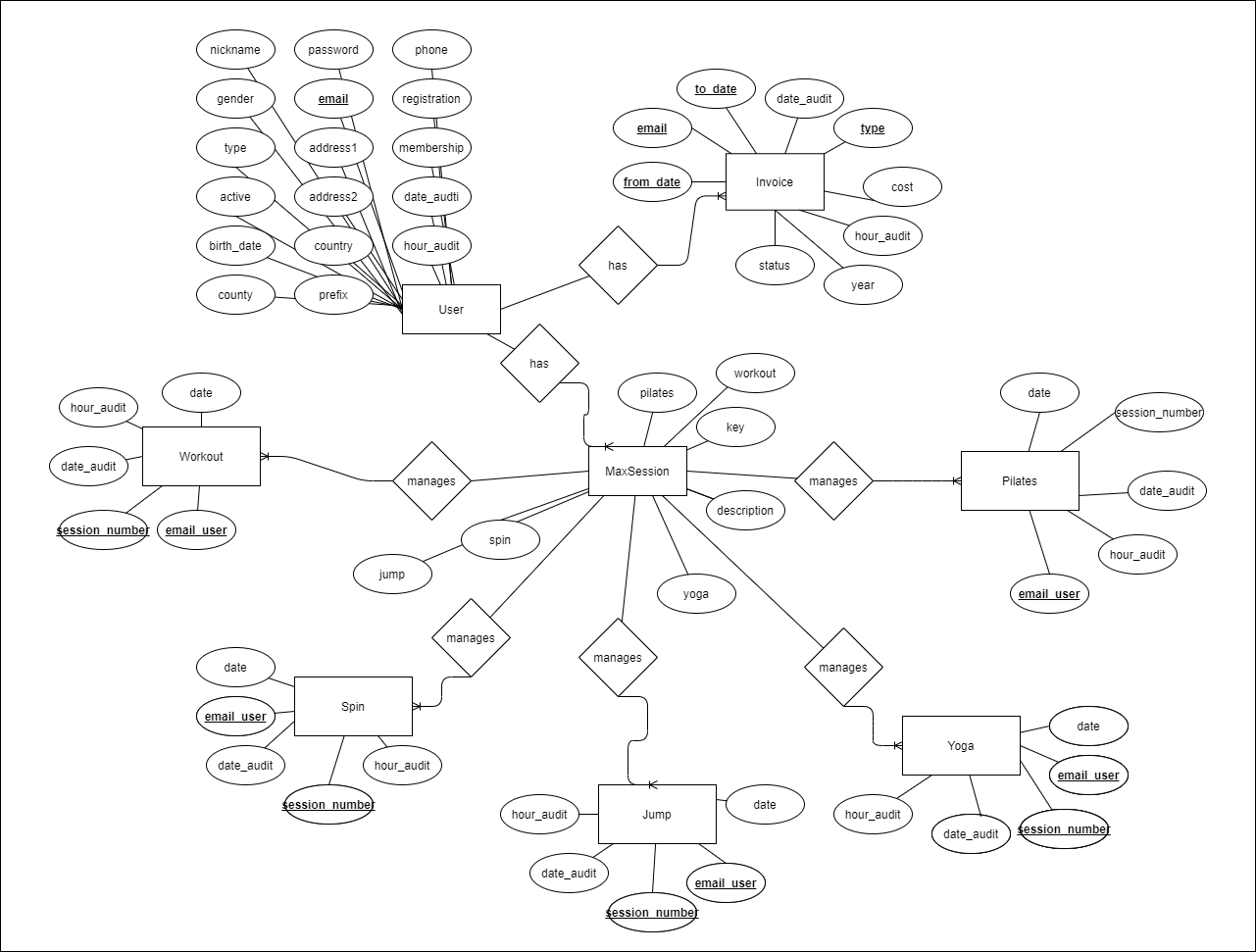
## Data requirements:

As informed previously, it was used the relational Database for this project. In this type of the database, it’s important to have all the entities well defined as well as the relationships between them to make them interact with each other. The entity is associated with a table in a system development scenario as being an object that contains attributes.

The entities for this project are:

* User: Table in charge to store all the personal and information related to the system.
* MaxSession; Table that contains the setup information of the system such as the maximum number of users per session and subscription costs information:
* Invoice: Table that stores all the invoices for all the users regarding the enrolment fee and monthly payments.
* Workout: Table where all the workout sessions are stored.
* sessions Spin: Table where all the spin sessions are stored.
* Yoga: Table where all the yoga sessions are stored.
* Pilates: Table where all the pilates sessions are stored.

Down below you can check on the ER diagram14 built with all the entities and its relationships:



**Illustration 37: ccfit ER diagram**

## Database design:

For the database design, it was thought of dividing the tables into three categories:

* Invoices;
* Main core;
* Classes.

**Invoice**

The invoice category counts only with the “Invoice” table that deals with the invoice of all users.

14 ER Diagram: Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important information: **The major entities within the system scope**, and the**inter-relationships among these entities.** (Paradigm, 2020)

**Main core**

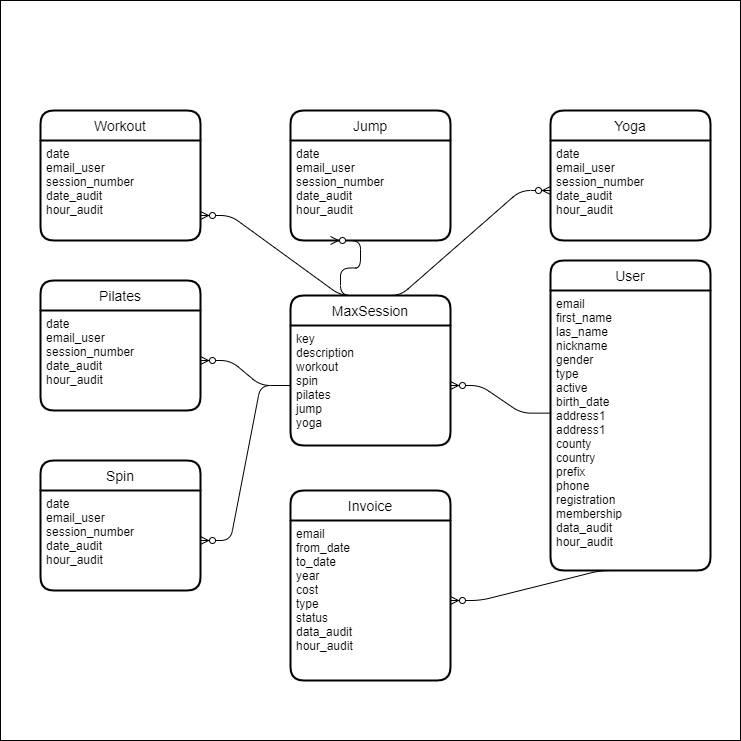
The main core has this name because contains the table where most of the setup of the project is retrieved from. The is composed by the “User” table with the personal and system data of all users, specifically the type of membership (workout only or all classes) where the classes permissions will be based on. The other table is the “MaxSession” that holds the number max of people per class.

**Classes**

This category counts with distinct tables for each one of the classes (workout, pilates, jump, yoga and spin). The reason why the table was created separately is due to the further work that will be done on this project, some particularities (different fields) will be added for each one of the tables that won’t let the data be stored in only one as it is suggested in order to maintain the performance. An example of that is if users from the yoga class are borrowing the yoga mat with the gym or bringing their own one or in case the dance (not implemented yet) that there will be different styles of dance and this *dance\_style* attribute being a relevant information only to the dance entity. All these fields will be added and these types of information will be added to them.

Logical design:

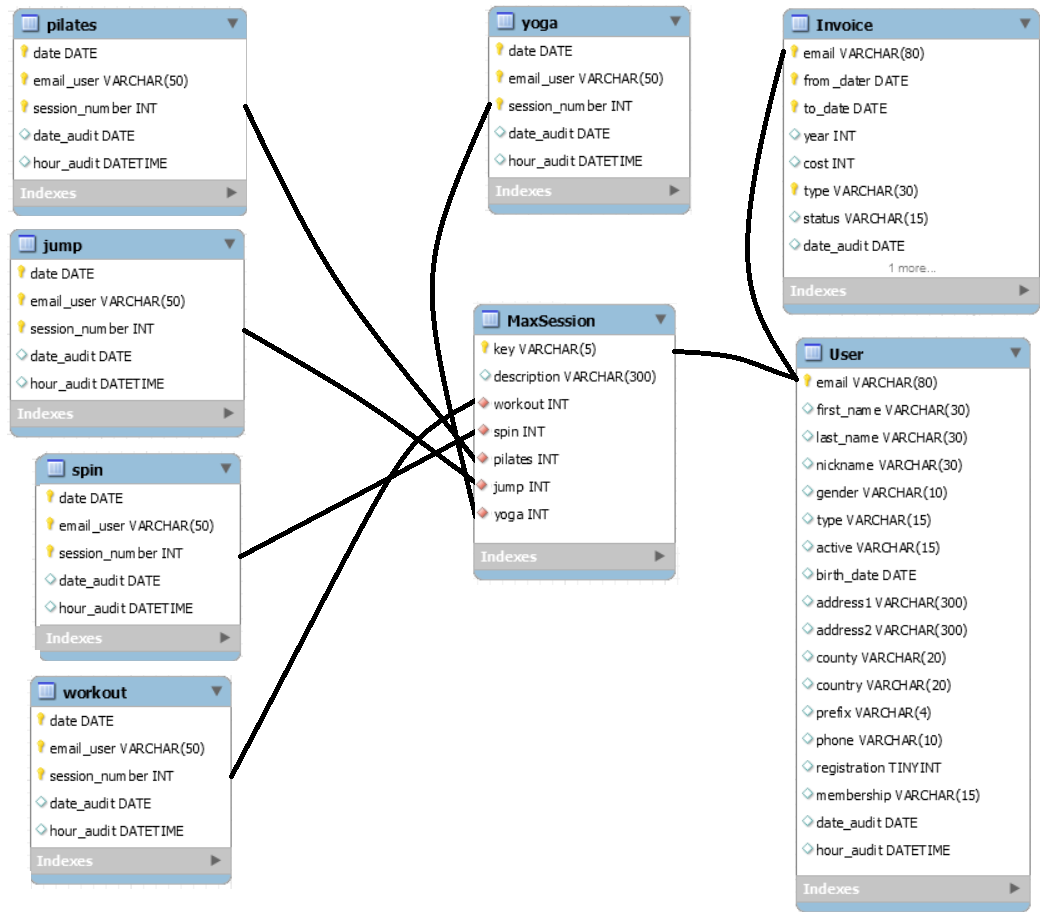
The following image shows how the tables are related to each other:



**Illustration 38: logical design**

Physical design:

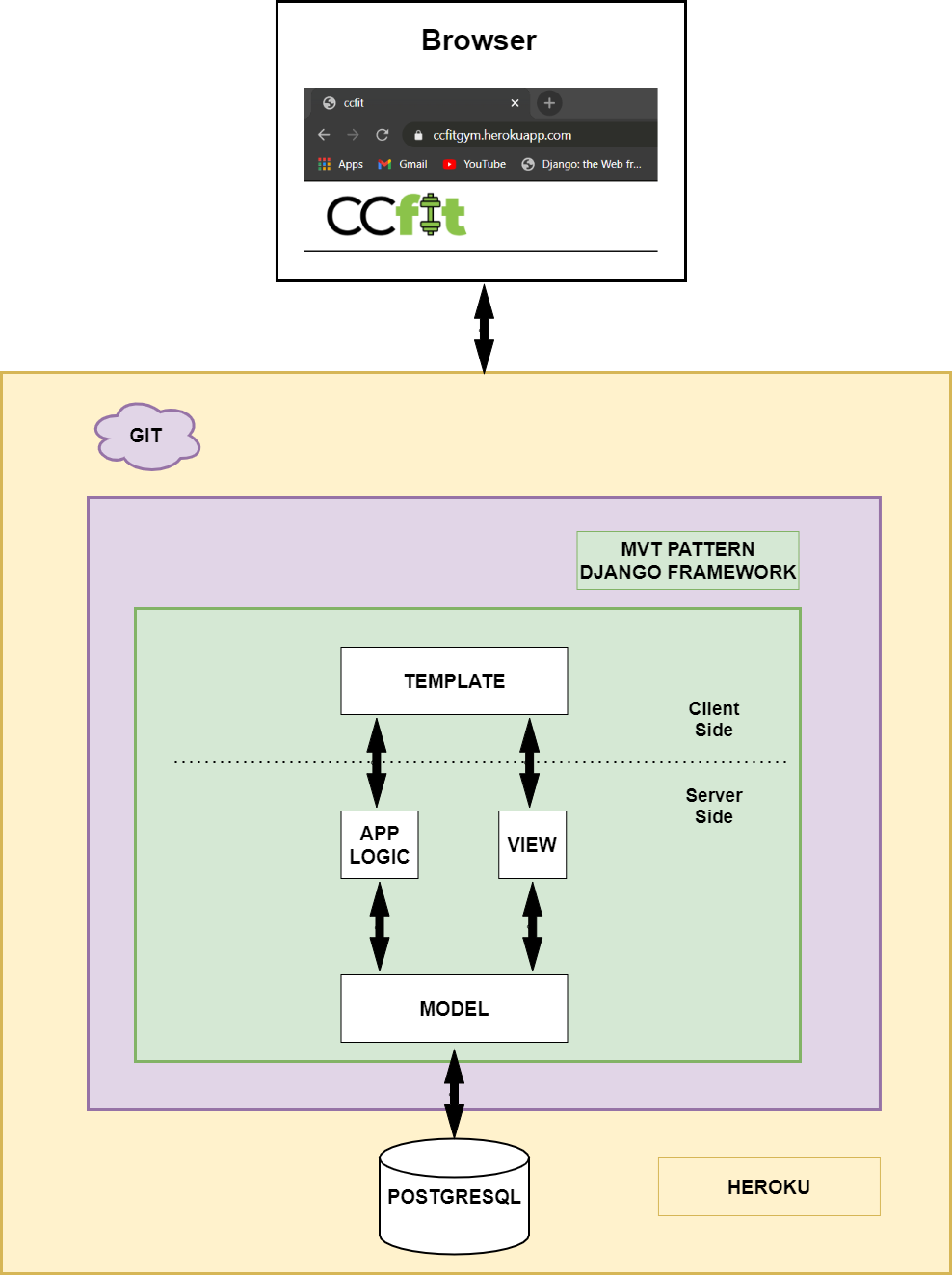
The image down below is the representation of the physical design.



**Illustration 39: physical design**

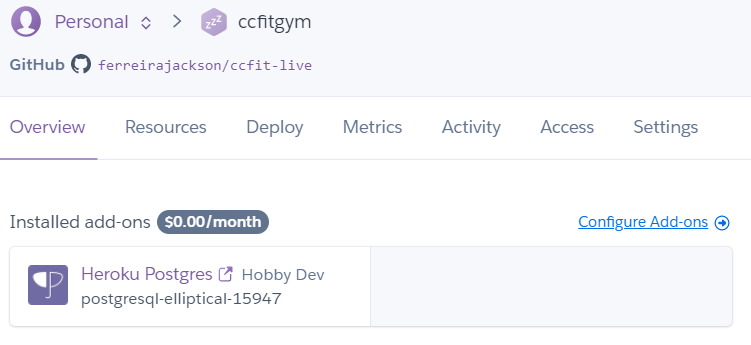
## Functional design:

The architecture used in this project is considered standard as it contains a database, the MVT architecture pattern which is similar to the MVC (Model View Controller), a web framework that takes care of the Client and Server-side and a cloud platform to host the website:



**Illustration 40: ccfit architecture design**

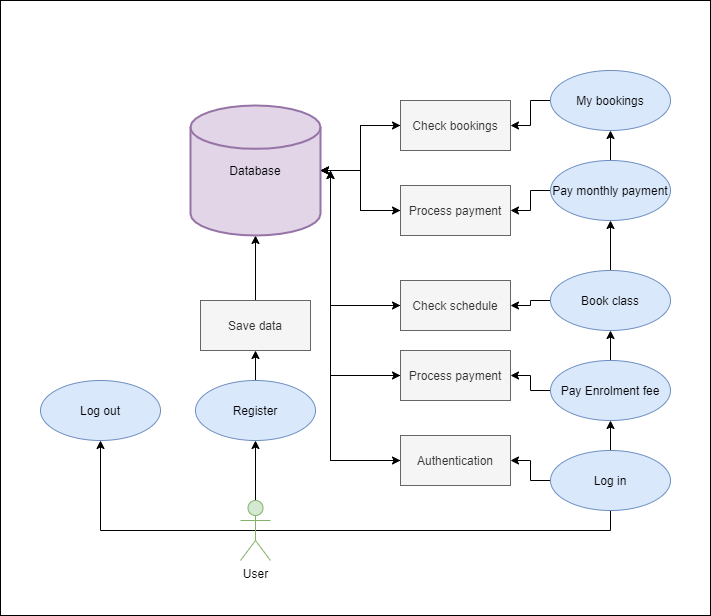
As it was said previously in this report the database chosen for the project was PostgreSQL. For the major part of the programming the local environment was used but as the development process went by it was felt the necessity to host the database instance online as part of the deployment process. Fortunately, Heroku allows the association of a free PostgreSQL instance for an account. Regarding the server and client-side, both are handled by the Django web framework



**Illustration 41: postgresql instance**

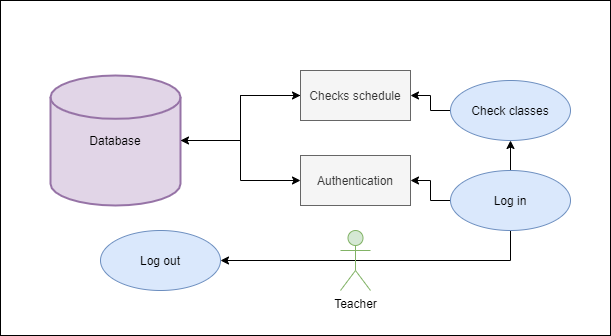
The following diagrams show all the capabilities that all the users have when using the system.

**Users**



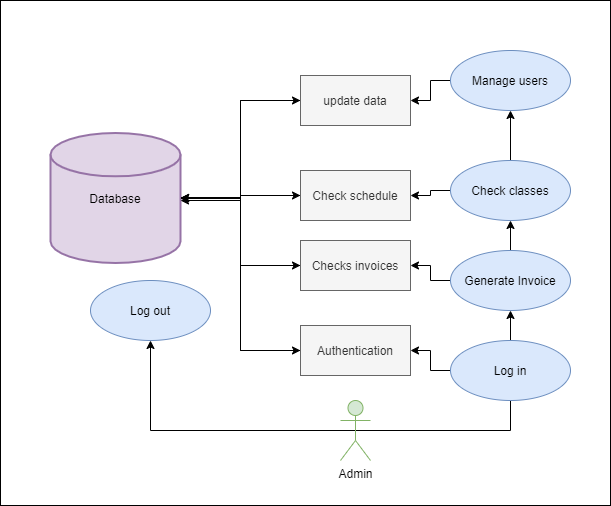
**Illustration 42: user’s functions**

**Teacher**



**Illustration 43: teacher’s functions**

**Admin**



**Illustration 44: admin’s functions**

# CHAPTER IV: IMPLEMENTATION OF THE SYSTEM

## Architecture:

The CCfit system is an application developed using the basic architecture for websites made with Django Python and PostgreSQL. The decision of using these technologies relies on the facility of developing a web application in an agile process and the fact that using high-level web frameworks like Django speed up the development process when combined with a system database that is supported by Django like PostgreSQL. It also makes use of the MVT pattern which is of common use in many projects as keeping the best practices when developing websites.

## Technologies implemented:

### Frontend

* HTML (HyperTextMarkup Language);
* SAAS CSS (Cascading Style Sheets);
* Javascript.

### Backend

* Python, as the programming language;
* Django, as the web framework;
* Model-View-Template (MVT) as the architectural pattern.

### Database

* Relational Database (POSTGRESQL), to store the data of the program;
* pgAdmin software to manage the data from POSTGRESQL;
* Relational Database (SQLite) for tests.

### 

### Other

* HTTP Methods;
* Heroku (PaaS platform).

## Implementation of the system:

As it was shown on the Gantt chart of the development of this project, the execution of the tasks followed a chronological order from planning to testing and documenting of everything it was done. The implementation points are the following:

* Database;
* Backend;
* Frontend;
* Testing;

It’s important to notice that deployment is not in the list because it was being made in each one of the tasks.

### Database

Regarding the database, as stated earlier, the database used at the beginning of the project was SQLite3 which comes as the standard database when Django is installed running on the machine. Because of all the projects I worked on were not deployed, the use of SQLite3 seemed enough while my idea regarding this work was just to use the local environment. However, as the project went on and I saw the possibility of deploying the application, SQLite3 had to be replaced by another database in which the cloud platform that would be used was compatible and had an integration with. Because of that, the change from SQLite3 to Heroku was made.

**SETUP**

The configuration of the database was very simple to do as mentioned in chapter II of this report and for table schema to be created, two commands to be run:



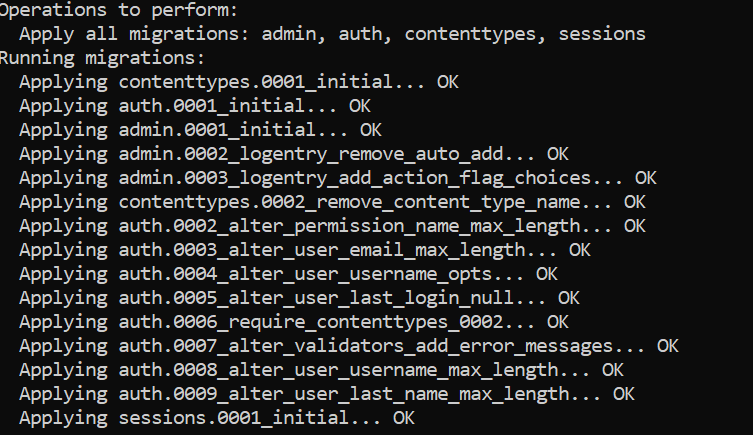
**Illustration 45: makemigrations command**

This command is responsible for creating or updating the models



**Illustration 46: migrate command**

This command commits the changes made by the makemigrations generating the following output:



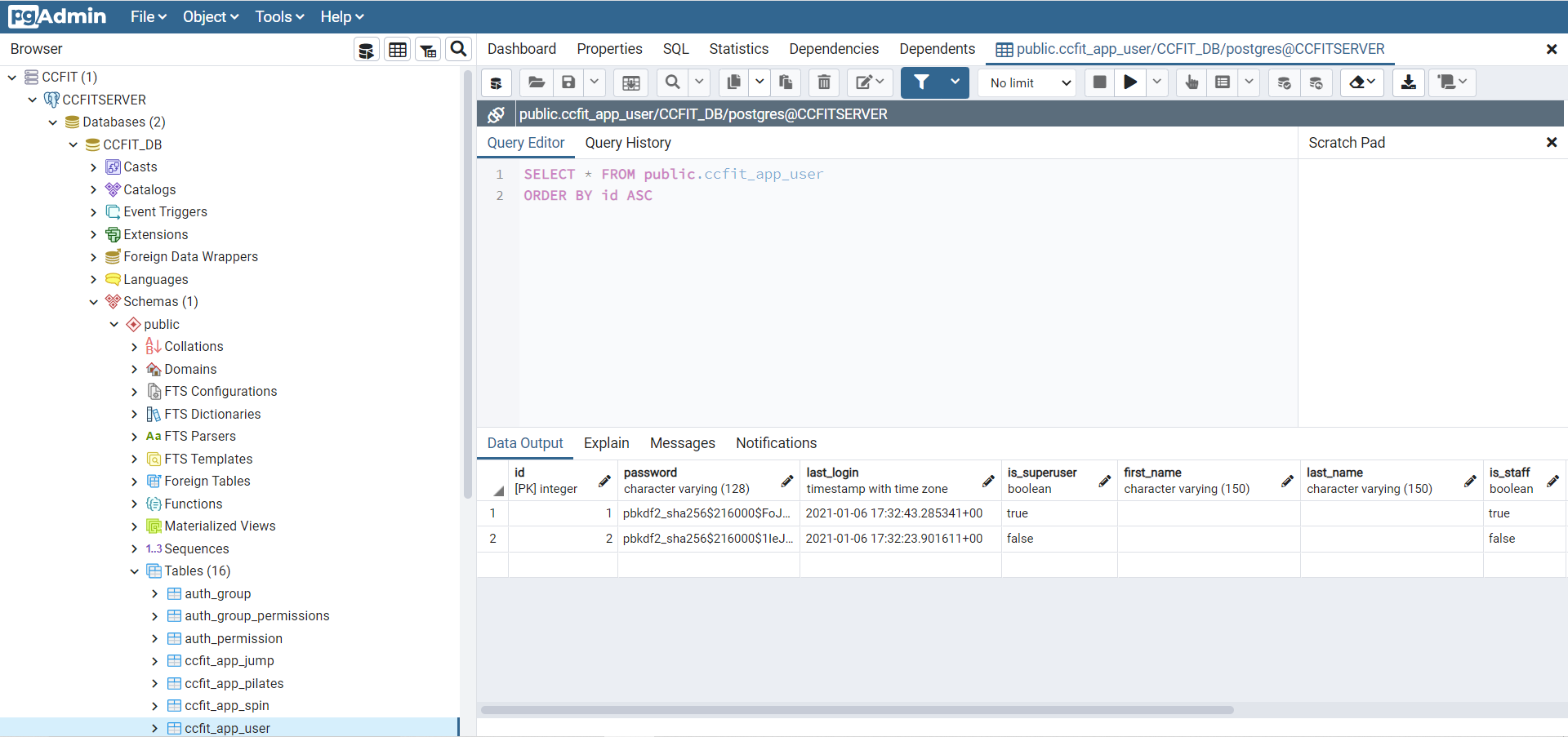
**Illustration 46: migration blog**

With that the database is configured and in case of any changes in the tables it is only necessary to run the above commands again.

**MANAGEMENT**

The pgAdmin 4 tool was used to add and delete information into the project tables. This tool allows the user to have access and operate all the CRUD15 methods via code or through the panel, also allowing the user to perform several queries on the database and take logs of the current database state.

15 CRUD: Create, Read, Update, and Delete (CRUD) are the four basic functions that models should be able to do, at most. (Codecademy, 2011)

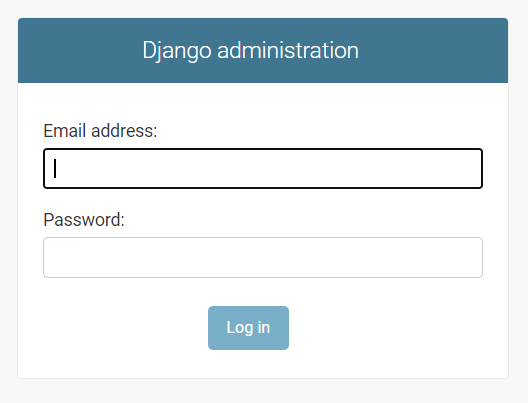


**Illustration 47: pgAdmin panel**

Another alternative widely used to access the application data is through the admin that can be accessed with the URL of the site and adding / admin at the end, for example:

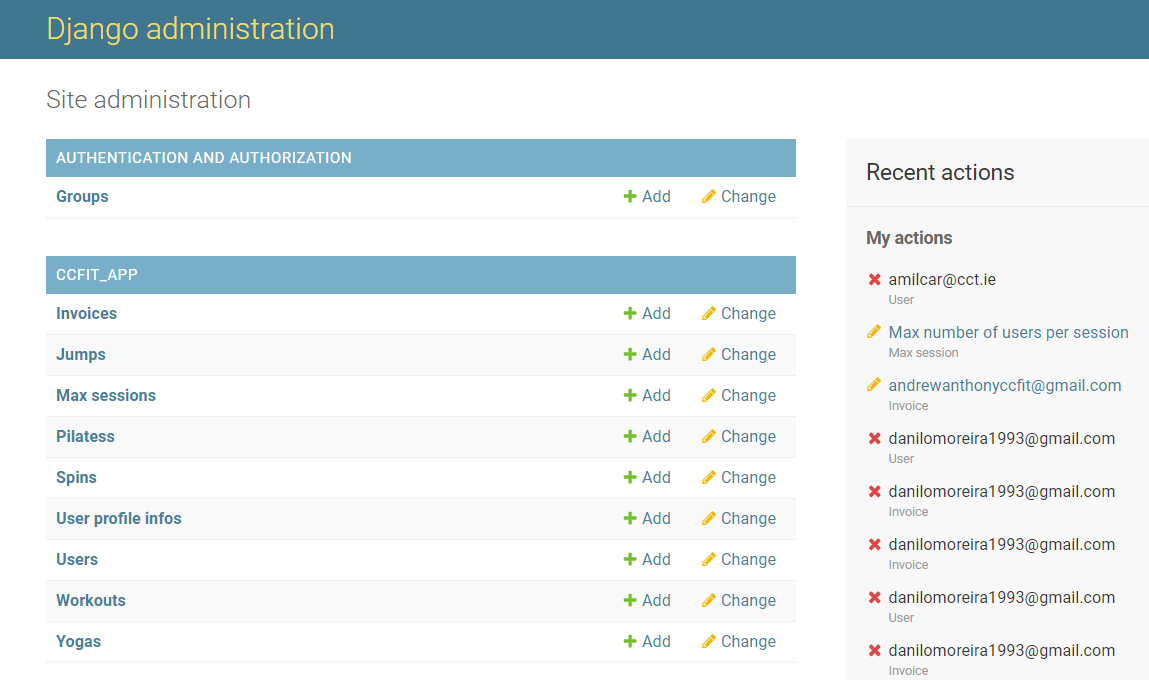
<https://ccfitgym.herokuapp.com/admin>

When accessing the link above the user is directed to the following screen but only the site administrator has access to the system.



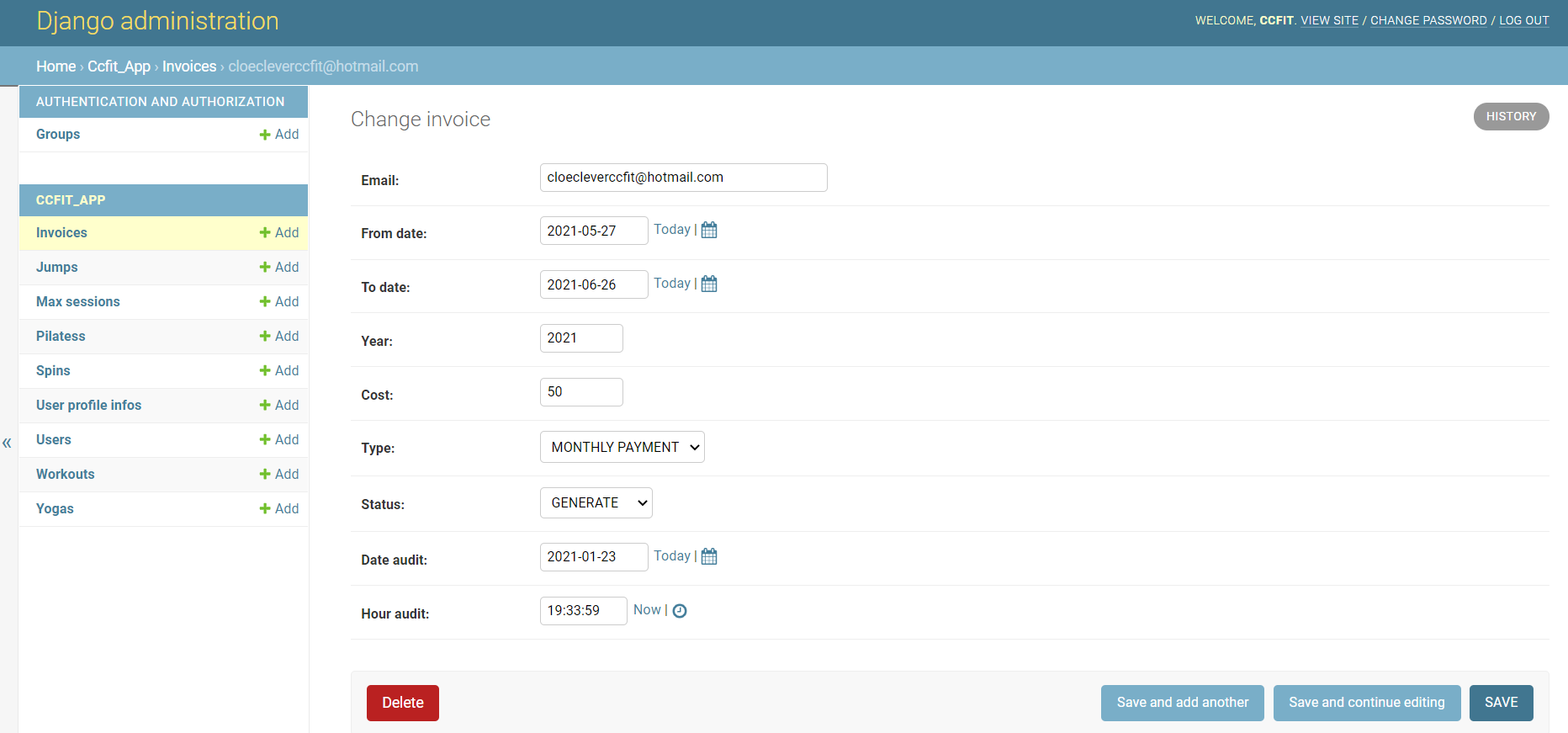
**Illustration 48: django admin panel**

After logging into the system, the administrator will be rediretcted to the screen where he will be able to see all the tables:



**Illustration 49: table panel**

And alter any information:



**Illustration 50: row panel**

### 

### Backend

The backend development process was certainly the most time consuming due to the various functionalities that were requested and the features that were planned and introduced in the system.

Required features:

* Create booking system;
* Check previous bookings;
* Print invoice;
* Change costs.

**Check booking system**

This is, in fact, the greatest functionality of the system, the CCfit system of bookings for users was implemented so that by their own cell phone or computer they can book the desired classes.

The booking functionality counts with 5 tables for each one of the classes and what the logic contained in the view.py file that holds all the backend logic does as soon as the user informs the date for the class is:

1. Validates in the user table for the membership\_type field that informs which classes the user is allowed to have access;
2. After allowing access, check which class the user chose and check the table for that class if the user has already booked a class for that day;
3. Allows the user to cancel the class and book another one, but not on the same day as the user can only book one class per day;
4. It also validates the time and does not allow the user to book the class that is less than half an hour away to start;
5. It also verifies if the maximum number of users booked for each one of the sessions for that day was reached and disables new users to book that classes until someone cancels.
6. The System also does not allow the user to book classes for days that have passed.

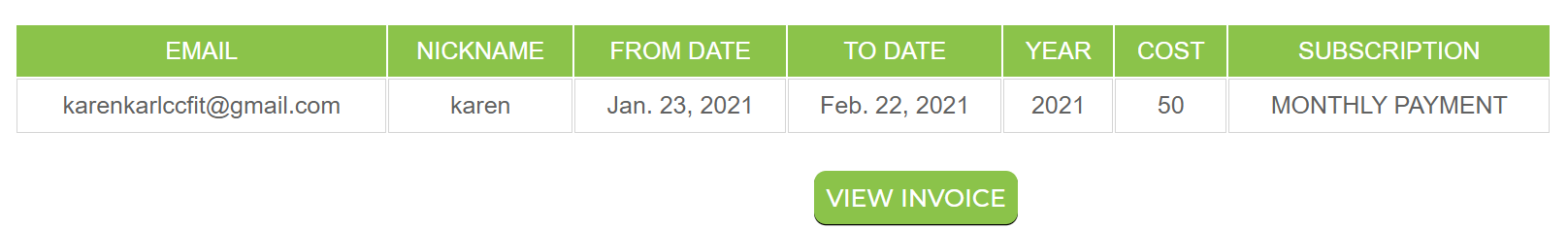
**Check previous bookings**

Access to the old bookings is available for all users by pressing a button on the home page and simply returns all bookings to that user since the first class. For teachers and admin works differently, teachers and the admin do not need to book the classes but they can check the schedule of each class and generate reports with this information.

**Print Invoice**

The system prints the invoice, not only displaying it on the payment screen but also generating a pdf with the same information in case the user wants to pay in person and not using the online option.

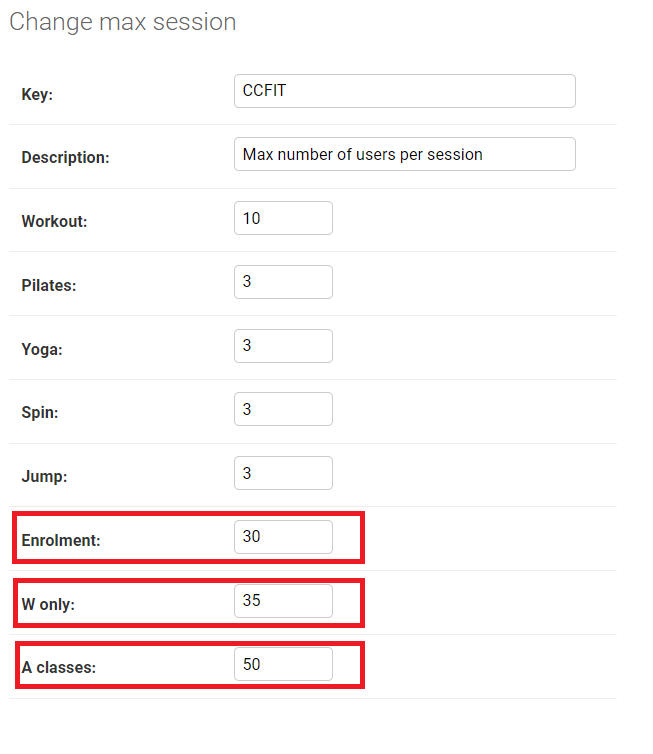
**Screen invoice:**



**Illustration 51: screen invoice**

**Change costs**

The change in the values of the membership types can only be carried out by the administrator and is made by the admin panel as explained previously in the “MaxSession” table:



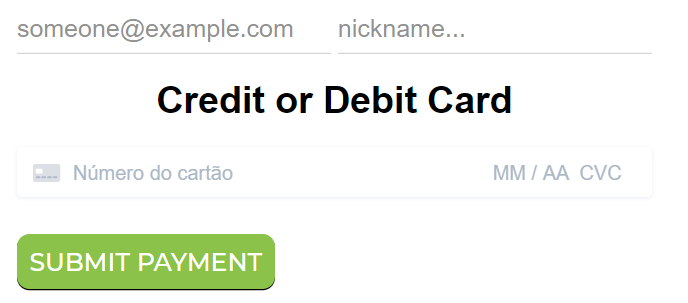
**Illustration 52: MaxSession table**

Extra features added:

* Payment system;
* Manage users;
* Change of password;
* PDF generation.

**Payment system**

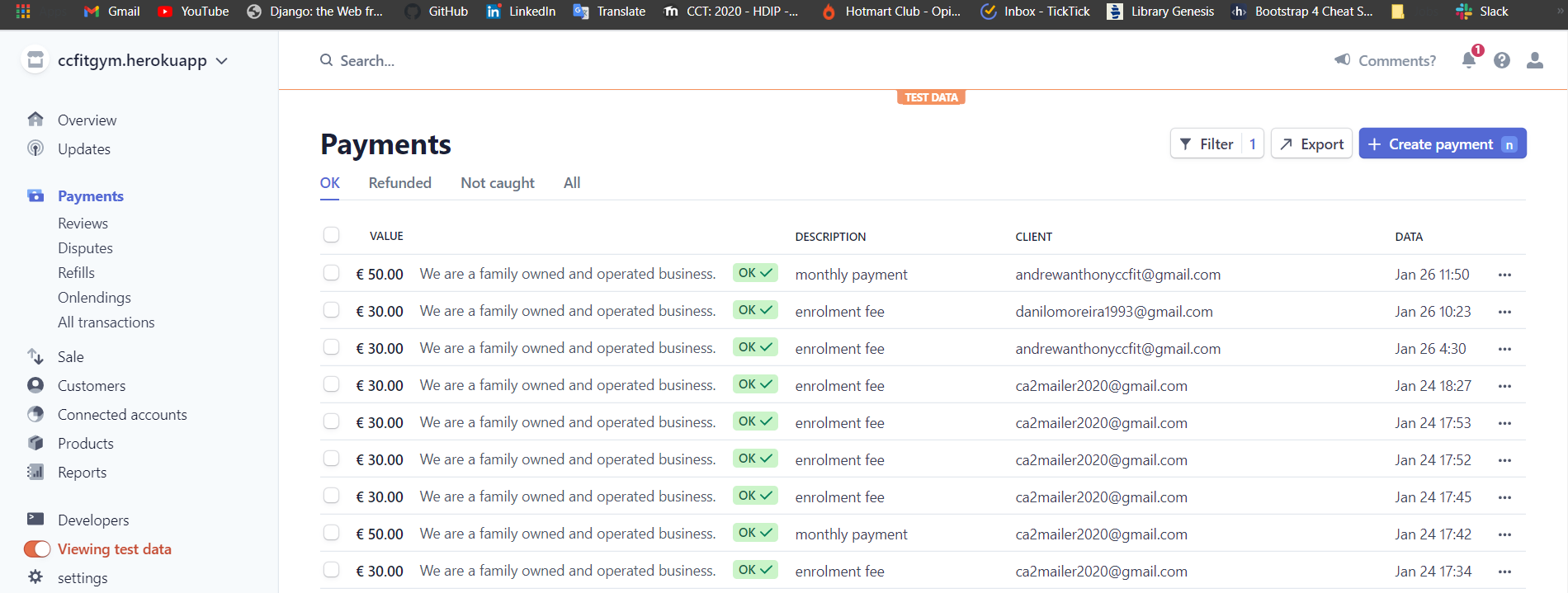
The payment system was implemented as another way to help the user to be able to make the payment all online and making the process more streamlined. The system used was Stripe due to the ease of working with when compared with PayPal, which would require much more work and it might not be feasible to implement for the project time.



**Illustration 53: payment method**

It only requires the insertion of the information in the image above so that the payer can be identified by the admin and updated in the systems.

In the following image, we can see how the admin views payments in the stripe panel.



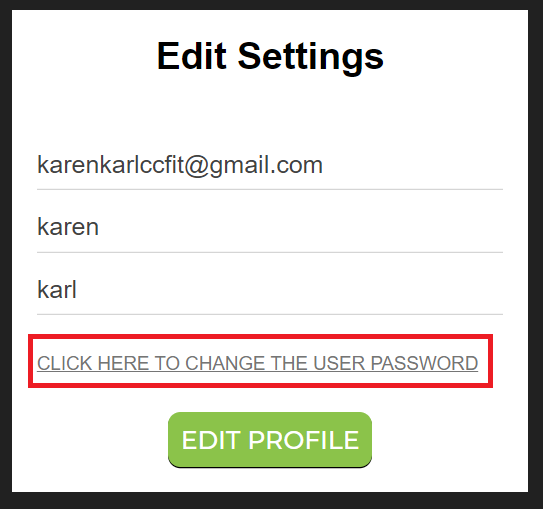
**Illustration 54: stripe panel**

**Manage users**

This section is for the admin to manage the user’s information

**Change of password**

The password change is something that was not specified in the project but it was implemented as a security measure.

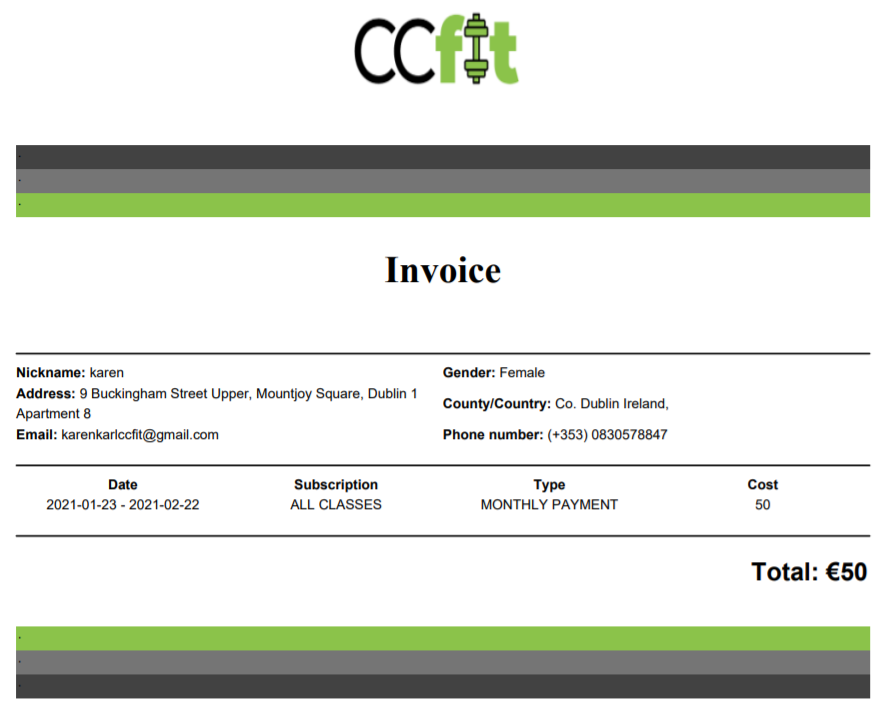
 

**Illustration 55: edit settings | Illustration 56: edit password**

**PDF generation**

As it was shown in the print invoice section, this is the invoice in the pdf format generated when the user clicks in “View Invoice”.

Pdf invoice:



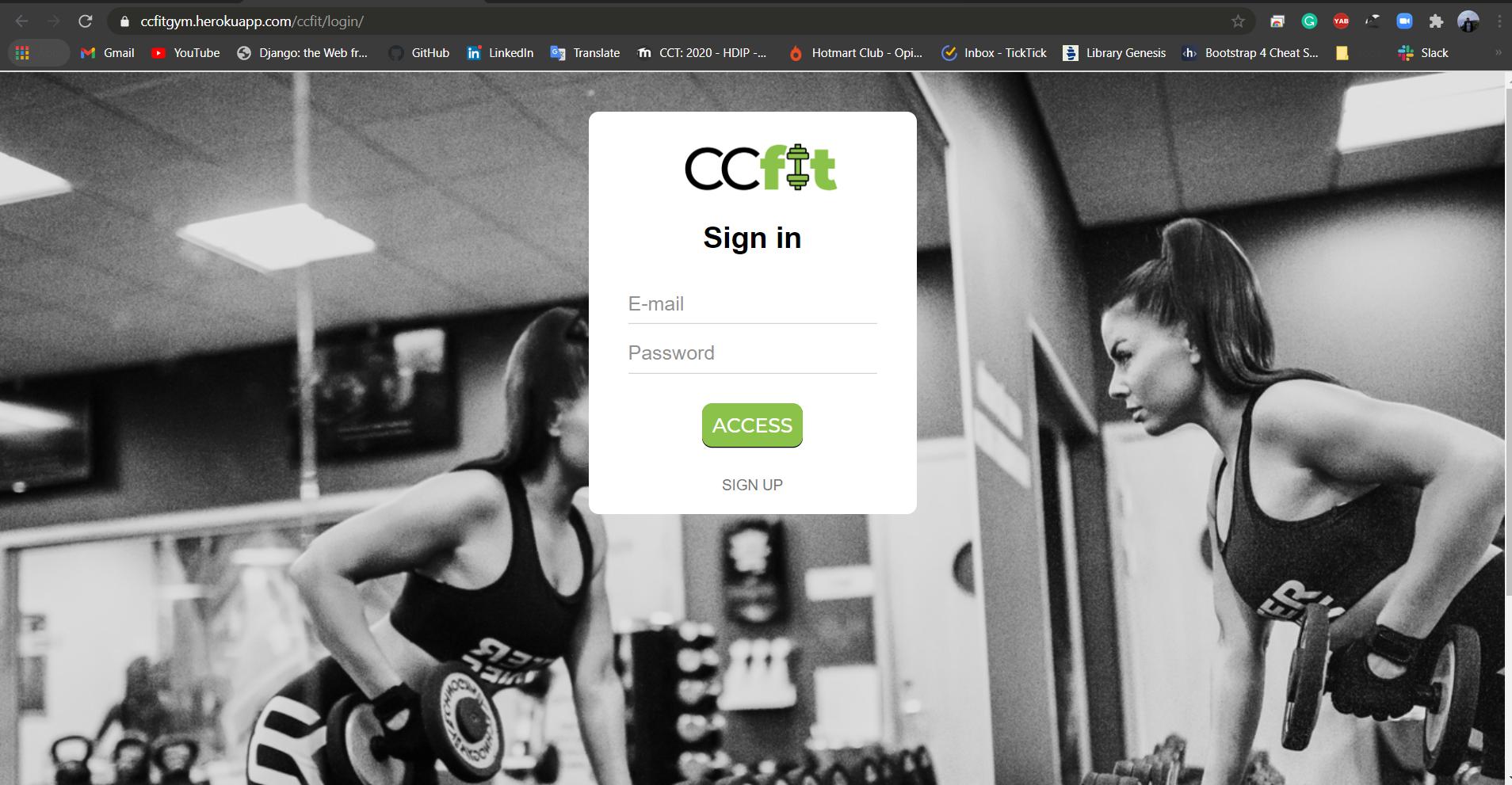
**Illustration 57: invoice output**

### Frontend

The design of the frontend was thought to try to maintain a colour pattern that had to do with the concept of the website. The logo contains a gym weigh and the colour green because green alludes to healthy things which is a great combination with physical exercise, and black to give a contrast to the green. From the definition of these two colours, the layout started to be worked on. The login and signup screens are the only ones that have a different layout from the others.

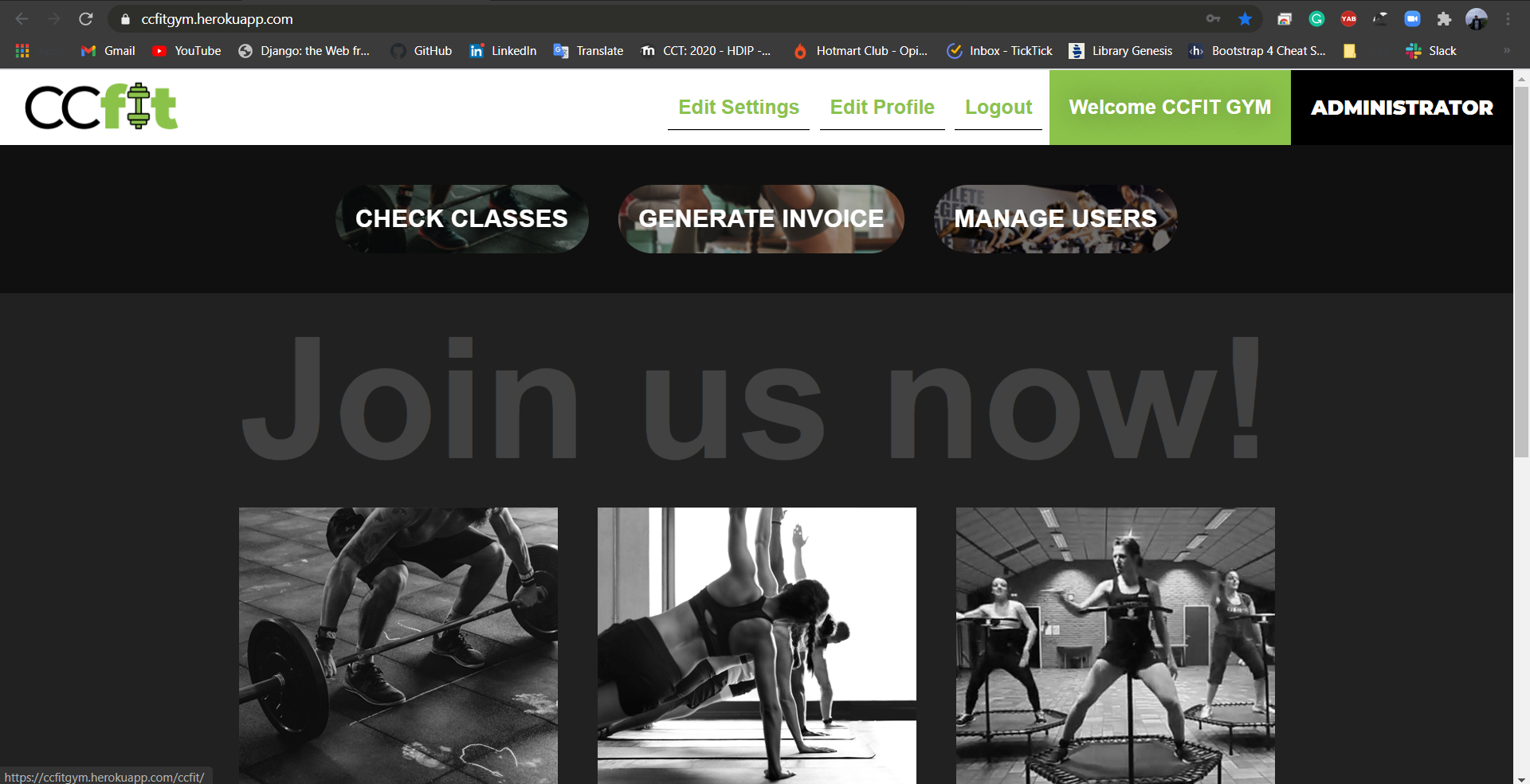
**Design**

Login and signup:



**Illustration 58: ccfit sign up page**

Other pages:



**Illustration 59: ccfit home page**

The design of the other screens is also simple and has a navbar that informs the users' name and type of user, in addition to the logout button and editing of personal information.

**Template tagging**

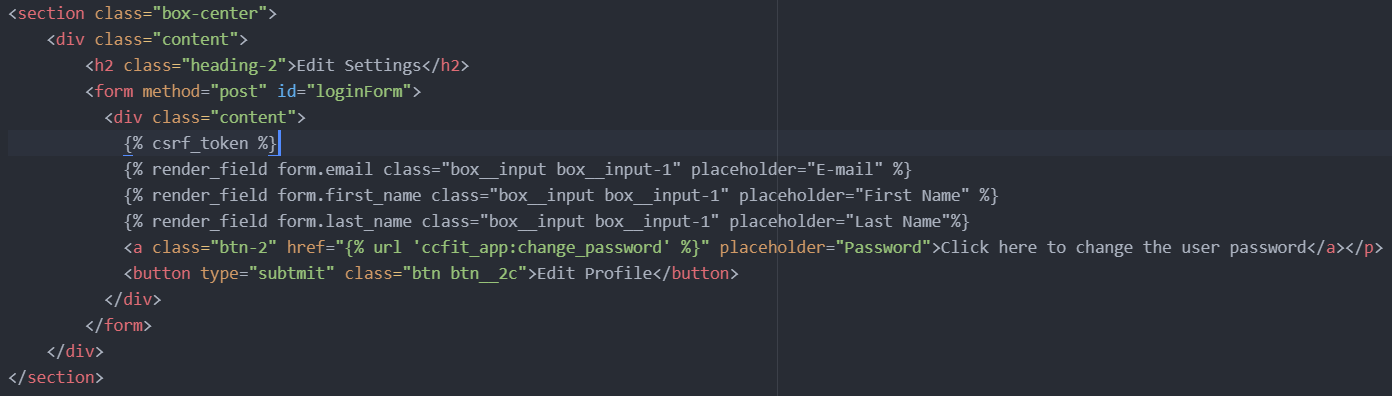
Template tagging is the name of the notation used in the HTML files on the front end to represent data coming from the server. These tags are intended to interact with the HTML tags and generate dynamic content for the web page.



**Illustration 60: template tagging code**

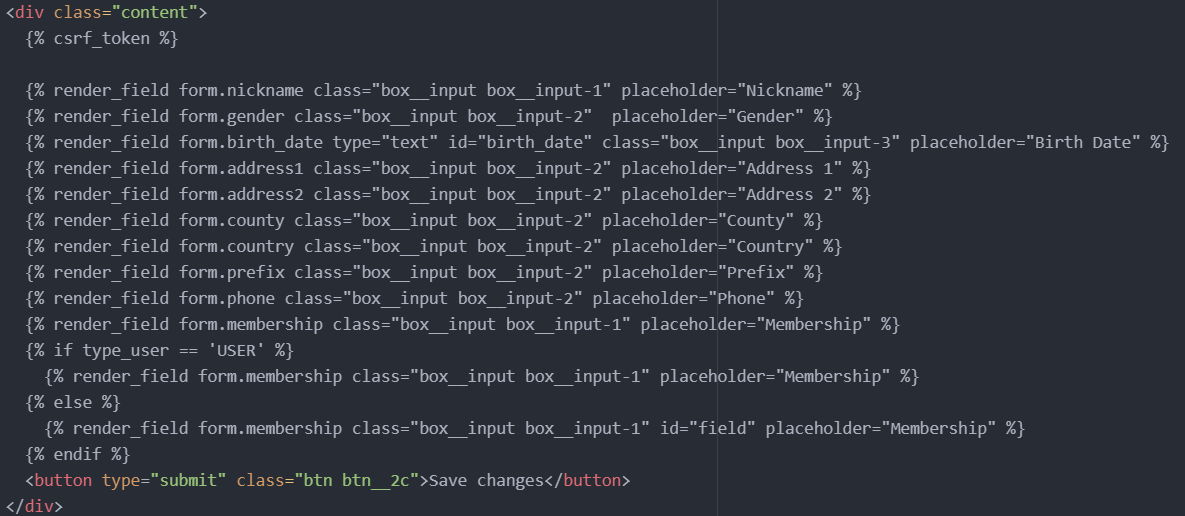
The tags are represented by curly brackets and the percentage sign and it’s possible to manipulate data in many ways like using if conditions or doing nested loops.

One very important tag that exists and should be commonly used is the {% csrf\_token %} which should always be used when there is a form on the page. It exists as a security measure so that the user is not able to do SQL injection through the forms. This is a tag that comes standard with Django as a security measure.



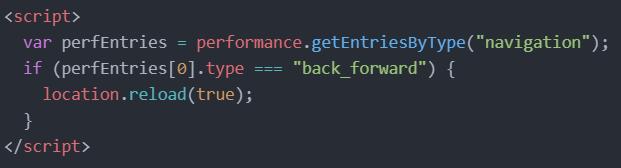
**Illustration 61: csrf token tag**

The interaction of template tagging with HTML is so great that there is a python library called widget\_tweaks that allows the use of both as if it was a normal HTML tag as in the case below where all tags contain the class and placeholder attributes and are visually processed by the browser like if they were HTML codes.



**Illustration 62: widget\_tweaks**

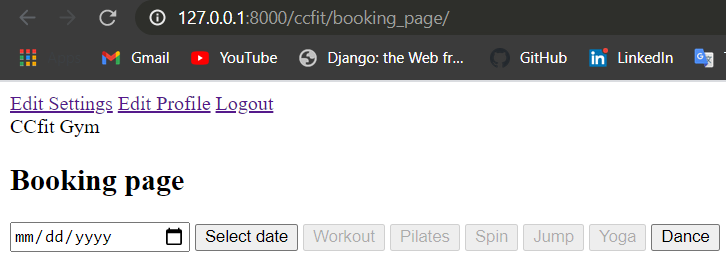
Some JavaScript codes were still used to perform some validations not handled neither with Django or HTML such as reloading the previous page when the user clicks the back button on the page.



**Illustration 63: javascript code**

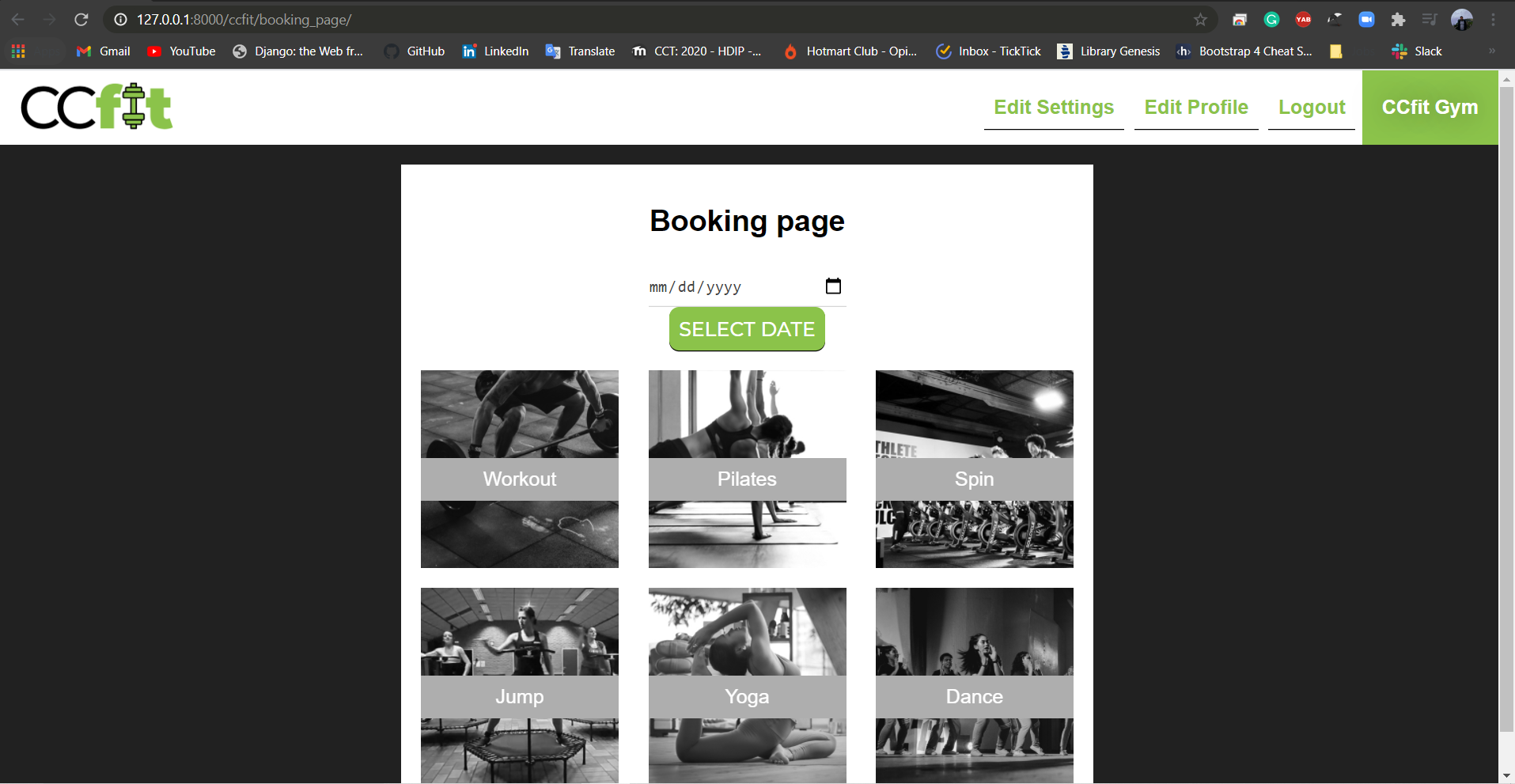
### Testing

The execution of system tests, as well as the deployment, was done in stages and in the end with double attention to all details. The first testing stage was being carried out while the backend was being built.



**Illustration 64: pure backend**

After that, the frontend was implemented and all functions were tested again with the new layout the output generated for the user is:



**Illustration 65: booking page**

One other useful resource that helps on the tests are the logs generated by the Django terminal that shorten the search for bugs:



**Illustration 66: log error**

## Problems encountered:

For the issues encountered along the development process the following table was created to individually detail them:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Action | Technology | Issue/Challenge | Solution |
| Database | Deploy or make the database online | SQLite3 | The database became obsolete when the need for deploying the database came up | The database was replaced to PostgreSQL that has more features supported by the web framework. |
| Database | Query PostgreSQL | PostgreSQL | Learning a few PostgreSQL commands manipulating data via terminal | Looked it up on the internet |
| Backend | Generate PDF | Xhtml2pdf library from pip (python package manager) | First time using this library | Took more time to implement than expected. |
| Backend | Working with permissions | Django decorators’ function - Python | To work correctly with permissions is needed to have a better understanding of how the decorators work | Looked it up on the well detailed Django documentation. |
| Frontend | Working with responsive design | HTML CSS | Couldn’t make the website responsive without having to write an enormous chunk of code | Researched and used a few Sass concepts to compile my cascading style sheets |
| Deployment | Deploy website | Heroku | Error when trying to deploy and .env file that keeps all the passwords | The error was researched through the log and it was decided to store the password in the Heroku environment variables and push that via code |

# CHAPTER V: TESTING & EVALUATION

In this chapter, all test scenarios will be detailed, exploring all the functionalities of the system. The tests were carried out in both test and production environment to make the system as functional as possible.

## System access:

**Register**

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| First registration | Store the data in the database and automatically log in the user redirecting to the home page | Correct |  |
| Register again | Display a message saying that the user is registered already | It doesn’t allow the user to register again but it doesn’t display any message, it just reloads the page with the form empty | No error message put in the code. More time needed |
| login button | Redirects to the login page | Correct. |  |
| Enrolment fee message | An enrolment fee message is displayed for the user to generate and pay the fee to start having access to the booking system | Correct. | The enrolment fee is sent after the user complete the user information because that form contains the type of subscription the user chooses. |

**Log in**

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| User registered logs in | Redirects to home page | Correct. | It also shows the User name and type of user in the navbar |
| Wrong input | Display a message saying that the input is wrong | The message doesn’t exist but the page is reloaded and the fields emptied for the insertion of new information | No error message put in the code. More time needed |
| Sign up button | Redirects to the signup page | Correct. |  |

**Log out**

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Press logout navbar button | Redirects to login page | Correct. |  |

## Edit user information:

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Change email, first name and last name | Update the email, first name and last name | Correct. |  |
| Change password | Update the password | Correct. | On the next login, the user will be asked to enter the new password. |
| Edit personal info | Create or update the personal info | Correct. |  |
| Creation of enrolment fee | If this screen is opened and edited for the first time, the “enrolment fee” button will be displayed and the enrolment fee invoice available for payment | Correct. |  |

## Book classes:

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Select and choose date | Displays correct data sets | Correct. |  |
| Don’t select date | Displays a message to insert the date | Correct. | On the next login, the user will be asked to enter the new password. |
| Press “select date” button | Checks the classes the user is allowed to book and enables access | Correct. | A small detail for that part is that there is a colour fade-in animation for all the class buttons even if the user doesn’t have access to it. As further work, I’ll be removing this animation to follow the same permission logic |
| Book a class of a date that has passed | Don’t the user to book any class | Correct. |  |
| Select a class without selecting the date | Don’t redirect to the class page | Correct. |  |
| Book class until 30 min before the class starts | Deny access to the user book a class that starts in less than 30 minutes from the time the user makes the booking | Correct | The session is not shown on the screen if the situation mentioned happens. |
| Book class that is full | Deny access for the user that tries to book a class that is full – reached the maximum number of users per session | Correct. | The number maximum of users per session is defined by the admin. |
| Book two class per day | Don’t allow the user to book to session per day and display the message that there is another session booked for that day already | Correct. |  |
| Cancel class | Remove the data concerning that class for the table and let the user book another class for the same day | Correct. |  |
| Book many classes for different days | Allows the user to book classes in advance | Correct. | As long as is two on the same day |
| Book classes for further months | Allows the user to book classes for further months and generate the invoice in the range for that period | Correct. |  |
| Cancel classes for further months | Checks if there are more classes for this further month, if not, delete the invoice. | Correct. |  |
|  |  |  |  |

## Generate invoice:

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Search field | Brings all the users searched by email | Correct. | The search works as a “like” conditions and brings any combination of words instead of having to type the whole and exact email. |
| View invoice button | Display the pdf for that invoice | Correct. |  |
| Send to user button | Changes the status of the monthly payment invoice from “GENERATE” to “REQUESTED”, enables the “monthly payment” button on the screen for that users and sends an email to the user letting him know that the invoice was generated and is ready to proceed with the payment. | Correct. |  |
| Mark as paid button | Changes the status of the monthly payment invoice from “GENERATE” or “REQUESTED” to “PAID” when the users make the payment in person. | Correct. |  |

## Make payments:

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| View invoice button | Display the invoice in the pdf format |  |  |
| Make payment | Redirects the user to the home page, give access to users to keep booking classes and take the monthly payment button out of the home page. | Correct. | On the next login, the user will be asked to enter the new password. |
| Invalid card info message | Display a message that the card is invalid while the user inputs it | Correct. | The form doesn’t allow the submission of invalid card credentials |

## Check bookings:

**User**

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| My bookings buttons | Shows all bookings made by that user since he joined the gym | Correct. |  |

**Admin and Teachers**

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Check classes schedule | Allow the admin and teachers to print the schedule of all sessions for a particular day and class | Correct. | It provides two pages where the teacher/admin can see the number of users per session and all the users’ identification. |
| Print Schedule | Shows the pdf | Correct. |  |
| Download schedule | Downloads the pdf | Correct. |  |

## 

## Manage users:

|  |  |  |  |
| --- | --- | --- | --- |
| Test input | Expected result | Actual result | Comment |
| Update button for users | Retrieves the user information and allows the admin to update. | Correct. |  |
| Update button for staff | Retrieves all the teachers and admin information but the type of membership, that’s why the admin and teachers don’t do classes | Correct. |  |
| Change user to Inactive | Don’t allow the user to book any class | Correct. | The user still has access to his account tough |

# CHAPTER VI: CONCLUSIONS & FURTHER WORK

## Conclusions

Based on the complexity of this project and the requirements that had to be met given the Ger’s garage scenario as an example and also considering that the whole project had to be planned and developed from scratch, I can surely affirm that this is a decent prototype bearing in mind that not only content from the technical classes taught during the year of 2020 was applied to the project but also the business and innovation classes concepts if considering that the idea of the project came from a necessity of a current scenario.

I believe that this was a very challenging project which enough effort had to be put in order to make something tangible, significant and usable as most of us from the Higher Diploma course in Science Computing were trying to build. It is also valid to point out that within this 1-year-course, a full-stack project to be done in 2 months is a great achievement and all the students should be proud of what they made afterwards regarding that every material as support were given to us by the lectures over the course.

Personally, I think I have learned a lot this year and especially in this project where I had to manage myself to dedicate a certain amount of time to learning new things as the project required such as the habit of working with a repository in order to have my work kept safe and for shareability matters, or looking up on internet and having a better understanding of how the cloud services work to host my web application using Heroku for example.

In short, I wasn’t expecting to learn so much about different technologies, concepts and business idea in such a short time and work on so many different projects with a great variety of programming languages during this year that I feel confident and prepare enough to get an opportunity to professionally work with IT now.

## Further work

Since the very beginning development process of this project, it was thought of giving continuity to it as this is a system that meets the necessity to many entrepreneurs due to the pandemic situation. Some of the capabilities of the system can be adapted to each type of gym services and there is even the possibility of using the booking system for different types of business, using the system to manage and book medical appointments for private clinics where there are more than one doctors and different types of appointments (orthopaedics, paediatrician, general consult) for instance. Before that happens, some small changes are to be made:

* Implement the dance classes and the types of dances;
* Refine the frontend design;
* Change the payment method to PayPal;
* Add more functions to the system such as discount vouchers and classes rating;

All these fundamentals quoted above were not implemented due to the project size and the lack of time to implement it.

# APPENDIX A: TEST INFORMATION

## Source code:

All the code for this project is available on:

Github link: <https://github.com/ferreirajackson/ccfit-live>

## Access to the system:

The website was deployed to Heroku and can be accessed through the link:

Website link: <https://ccfitgym.herokuapp.com/>

## User credentials:

In order to run the system and test the functionalities, you can either sign up with your personal email or use one of the followings:

To log as an **admin**:

**CCfit account**

Email: [ccfitgym@gmail.com](mailto:ccfitgym@gmail.com)

Password: ccfit1234!

To log as a **teacher**:

**CCfit account**

Email: [joejonesccfit@gmail.com](mailto:joejonesccfit@gmail.com)

Password: ccfit1234!

To log as a **user**:

**CCfit account**

Email: [andrewanthonyccfit@gmail.com](mailto:andrewanthonyccfit@gmail.com)

Password: ccfit1234!

The following Gmail account is given so the receipt and booking confirmations can be read (or try your own personal email)

**Gmail account**

Email: [andrewanthonyccfit@gmail.com](mailto:andrewanthonyccfit@gmail.com)

Password: ccfit1234!

**Stripe environments**

One thing to be considered is that both test and production environment for the stripe payment method API work. Both scenarios will be shown on the screencast video to demonstrate that works but **the environment provided for the lectures to use will be the test one** as the production requires real card information.

To pass over the payment test, the environment test accepts the following fake card credentials:





Typed password: **4242424242424242 12 23 456 78910**

p.s.: You can use that credential or any other as long as the card number is “4242 4242 4242 4242”.

With that being provided, the user can follow along the process.

# REFFERENCES:

* AJAX, 2015. *W3Schools.* [Online] Available at: https://www.w3schools.com/xml/ajax\_intro.asp[Accessed 27 January 2021].
* Chai, W., 2021. *TechTarget.* [Online] Available at: https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS [Accessed 19 January 2021].
* Codecademy, 2011. *Codecademy.* [Online] Available at: https://www.codecademy.com/articles/what-is-crud [Accessed 13 January 2021].
* Cohen, 2019. *workamajig.* [Online] Available at: https://www.workamajig.com/blog/project-management-methodologies [Accessed 19 January 2021].
* Django, 2005. *Django Project.* [Online] Available at: https://www.djangoproject.com/  
  [Accessed 01 January 2021].
* Flair, D., 2019. *Data Flair.* [Online] Available at: https://data-flair.training/blogs/django-database/ [Accessed 28 January 2021].
* IBM, 2017. *IaaS, PaaS and SaaS – IBM Cloud service models.* [Online] Available at: https://www.ibm.com/cloud/learn/iaas-paas-saas [Accessed 10 January 2021].
* Intricately, 2020. *Intricately.* [Online] Available at: https://www.intricately.com/glossary/hosted-application#:~:text=A%20hosted%20application%20is%20any,a%20hosted%20application%20is%20Wordpress. [Accessed 26 January 2021].
* jquery, 2015. *w3schools.* [Online] Available at: https://www.w3schools.com/jquery/jquery\_intro.asp [Accessed 26 January 2021].
* JSON, 2005. *JSON.* [Online] Available at: https://www.json.org/json-en.html [Accessed 18 January 2021].
* Paradigm, V., 2020. *Visual Paradigm.* [Online] Available at: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/ [Accessed 19 January 2021].
* Pawlan, D., 2017. *Aloa.* [Online] Available at: https://aloa.co/blog/relational-vs-non-relational-database-pros-cons#:~:text=A%20relational%20database%20is%20structured,of%20a%20laundry%20list%20order. [Accessed 21 January 2021].
* ProjectManager, 2007. *ProjectManager.* [Online] Available at: https://www.projectmanager.com/waterfall-methodology#:~:text=Waterfall%20Methodology%20Resources-,What%20Is%20the%20Waterfall%20Model%3F,created%20to%20accommodate%20those%20requirements. [Accessed 26 January 2021].
* Robie, J., 2001. *w3.* [Online] Available at: https://www.w3.org/TR/WD-DOM/introduction.html [Accessed 20 January 2021].
* Romanowski, J., 2020. *LearnSQL.com.* [Online] Available at: https://learnsql.com/blog/companies-that-use-postgresql-in-business/ [Accessed 29 January 2021].
* sumologic, 2021. *Sumo Logic.* [Online] Available at: https://www.sumologic.com/glossary/software-deployment/ [Accessed 22 January 2021].
* White, A., 2020. *Izenda.* [Online] Available at: https://www.izenda.com/relational-vs-non-relational-databases/ [Accessed 15 January 2021].