
# Capstone Project

## Swap Classes

Luísa Maria Pereira Araújo

up201904996

07/2022

# Index

# 1. Description

This project's main goal is to build a web platform that allows students to request a class change with other students.

For each class swap there is only need for one request (only one student must file the form). When a student specifies all the other ones involved in the exchange, everyone needs to accept the request. When it doesn't involve other students, the application must match the request with other compatible ones.

The students don't need to authenticate themselves in the website to make a request nor to confirm it. After filling out the request form, the first student (in the form) will receive an email with a confirmation link. If this student doesn't confirm their part on the request (through the link sent) before a certain amount of time, then the request is dumped (timeout). If the first student confirms it on time, the second student will then receive their confirmation link through email as well. If timeout doesn't occur (the second student confirms on time) then they will close the request and swap classes. As a note, the e-mails indicated in the requests need to be university emails.

This website has an authentication system since only registered administrators (staff) have access to a few features, such as importing each semester's data and exporting all the concluded requests.

# 2. Goal

For this project a few goals were defined taking in consideration the total duration (late March - late June) and how many people would be working on it (just one person). The main goal was to build a website where a student could file a request to swap classes with another student. As it is going to be seen later, this was accomplished and there is even another request option (not part of the project's goal) where a student can just file for a request indicating the target class. In this case, requests are only concluded if there is a match with another student.

Other goals:

1. Import files of a current semester (students, curricular units, classes, class placements, schedules and composed classes).
2. Export the results of new class placements.
3. Send emails with a dynamically generated link where students can confirm their identity and will in the request.
4. Create links where students can check their request current status.
5. Have options to clean all information stored in the database, enable or disable requests, change the time it takes for requests to be cancelled and to view all requests and their respective status.
6. Build an authentication system.
7. Check for a request validity, including checking if there aren't any schedule overlaps.

More details about the goals can be found in section 3 (Use) where are available all implemented user stories.

# 3. Use

In the analysis and design phase, two actors were identified: Student and Administrator.

## 3.1. Student Perspective

User Stories:

1. As a Student, I want to be able to file for a request so that I can swap classes with another student.
2. As a Student, I want to confirm my part in the request through email, so that no one can swap classes with me without my consent.
3. As a Student, I want to have a link, so that I can check the status of my request.
4. As a Student, I want to file for a single request, so that I have the possibility to swap classes if someone else wants to enroll in my class.

Note: The first, second and third user stories were part of this project's proposed goals. The fourth one, as opposed to the other two, wasn't part of the project's goals.

## 3.2. Administrator's Perspective

User Stories:

1. As an Administrator, I want to import objects, so that the database will be filled.
2. As an Administrator, I want to export all the current concluded requests, so that I can then officialize all of them.
3. As an Administrator, I want to clean everything in the database, so that no unnecessary data is stored.
4. As an Administrator, I want to change the amount of time it takes to occur timeout, so that I can adjust it if it's currently an unnecessary long or short period.
5. As an Administrator, I want to log-in, so that I can have the necessary permissions to execute my functions.
6. As an Administrator, I want to log-out, so that no one can access my account.
7. As an Administrator, I want to view all requests, so that I can see the status of every request.
8. As an Administrator, I want to enable requests, so that students are allowed to file for their class exchanges.
9. As an Administrator, I want to disable requests, so students can no longer file for new ones.
10. As an Administrator, I want to make matches between single requests, so students can swap classes with other students.
11. As an Administrator, I want to see every match made, so that I can be aware if there is any incoherence in the current data.

Note: All the above user stories except for the tenth and eleventh were part of this project's proposed goals.

# 4. Development

## 4.1. Process

This project followed some Software Engineering conventions. For example, the Kanban Board, Git Flow and an Incremental model.

Kanban Board:

The board contained the following columns:

Product Backlog – Where all user stories could be found initially.

Product Iteration – The user stories expected to be implemented in each iteration.

In Progress – The user stories and other small tasks (such as a bug fix) being implemented at that moment.

Done – Finished user stories (were implemented and tested).

Dumped – Tasks that were no longer an objective for this project (at least until the deadline).

Ideas – Tasks that weren't part of the project's objective for evaluation but would be interesting to implement later.

Git Flow:

A few branches were created when implementing a new feature, such as schedule and single requests.

Iterative Development:

The development was iterative. Each iteration lasted approximately two weeks.

Even though the Kanban board model can be useful when working with other people in the same project, it was still useful so I knew what tasks were more relevant and could organize what to do in each iteration. As this model suggests, after every iteration there was an available version of this application with more features than the previous version.

## 4.2. Technologies

A framework makes web applications easier to conceive since, most of them follow some design patterns and have a lot of different functionalities to explore. Django is built using the Model-View-Template (MVT) architectural design pattern and was the chosen framework for mainly back-end development. A few advantages of Django:

- Developed and implemented in Python – Python is a language known for having tons of packages and libraries for almost every paradigm and for being "non-programmer friendly".
- DRY principle – Don't Repeat Yourself
- ORN Libraries
- Scalability

A few useful built-in features of Django:

- Django Authentication System – Already built authentication system.
- Django Administrator User Interface
- Django Interactive Shell
- Django Super User

# 5. Implementation (Technical Details)

## 5.1 Set-up and Run for Linux (Ubuntu)

### 5.1.1. Set-up instructions

Install Python3:

```
1.  sudo apt-get update
2.  sudo apt-get install python3.8
```

Install pip:

```
1.  python3 get-pip.py
2.  python3 –m pip install --upgrade pip
```

Django needs a virtual environment to execute an application. Install Pipenv (virtual environment):

```
1.  sudo apt install pipenv
```

Install Django:

```
1.  pipenv install django
```

Note: How to create a project and an application:

```
1.  pipenv shell
2.  django-admin createproject Swapping
3.  django-admin startapp app
```

Install dependencies:

```
1.  pipenv shell
2.  sudo apt-get install libpq-dev
3.  pip install psycopg2
4.  pip install python-csv
```

Install PgAdmin (desktop):

```
1.  sudo apt install pgadmin4-desktop
```

Install PostgreSQL (12):

```
1.  sudo apt-get install postgresql-12
```

### 5.1.2. Run Instructions

Run the code (from the root of the project – Swapping):

```
1.  pipenv Shell
2.  cd src/
3.  python manage.py runserver
```

### 5.1.3. Other useful commands

If there is any change in the file models.py then there will be a need to make new migrations:

```
1.  python manage.py makemigrations
```

After a new file is added to the folder migrations (Swapping/app/migrations/) there will be a need to migrate this new information to the database (translate it in SQL):

```
1.  python manage.py migrate
```

To create a new Super User (django feature):

```
1.  python manage.py createsuperuser
```

Check for any issues:

```
1.  python manage.py check
```

## 5.2. Database

The database is relational and it is built using PostgreSQL (version 12.9), since it has a few advantages, such as:

- Being open source;
- Doesn't require any license;
- Having an extensive set of functionalities;
- Scalability;
- Reliability;

Since Django has a built-in feature that allows programmers to write their models (tables) in python and then migrate it to a database (more details about it in the section Django built-in features), no SQL code was written manually.

However, the SQL (generated) code, like tables and constraints, can be consulted in two different ways:

1. Using PgAdmin4
2. Using PostgreSQL commands through the shell

### 5.2.1. Database Details:

Engine: “django.db.backends.postgresql”;

Name: “swapping”;

User: “postgres”;

Password: “.”;

Host (used during development): localhost (127.0.0.1);

Port: 5432;

### 5.2.2. Access database via PgAdmin

To access the database via PgAdmin use the Password indicated above.

### 5.2.3. Access database via shell

Connect to PostgreSQL service

```
1.  sudo –i –u postgres
```

Access PostgreSQL prompt:

```
1.  psql
```

Connect to swapping database:

```
1.  \c swapping
```

A few commands that were used and can be useful:

- Change database password:

```
1.  ALTER USER postgres PASSWORD ‘<new_password>’;
```

- See all tables in a database:

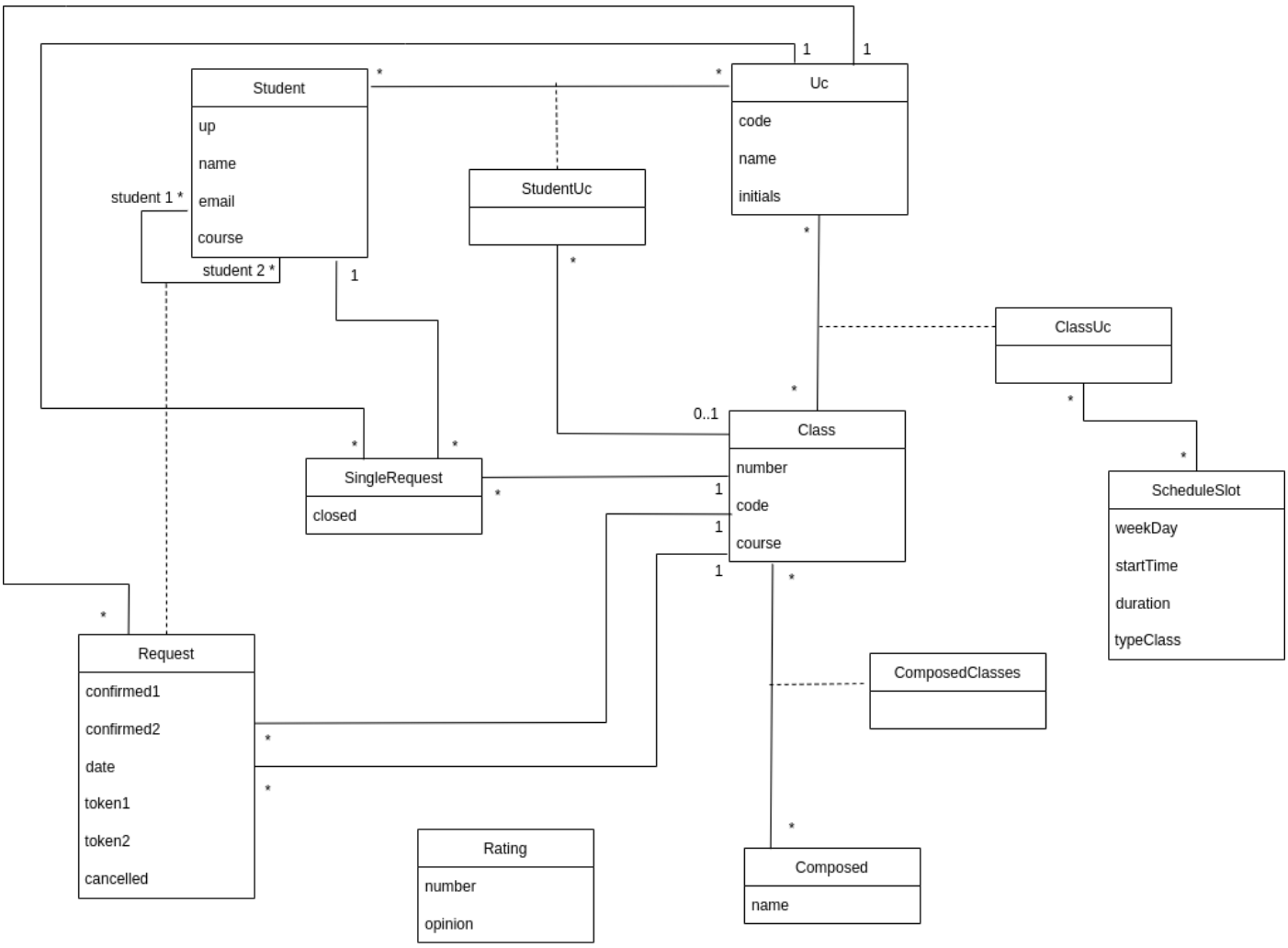
```
1.  \dt
```

- See all available databases:

```
1.  \l
```

### 5.2.4 Logical View

As a relational database, an UML diagram can represent it:




## 5.3. Make a Request

There are two types of requests mentioned in this report:

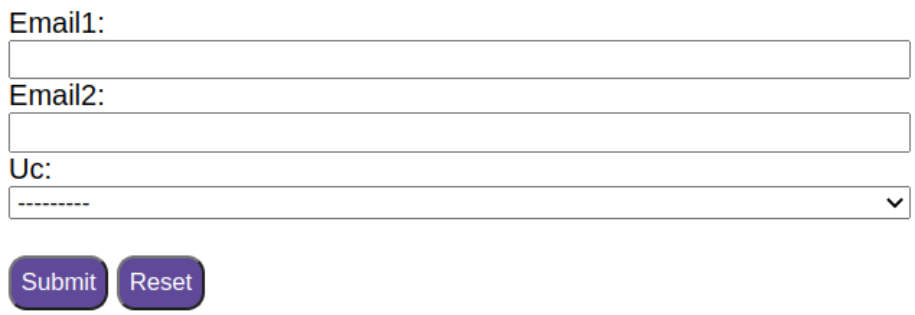
1. Two-person Request (also mentioned as duo request)
2. One-person Request (also mentioned as single request)

None of these requests guarantees a class swap between two students, either because they didn't acknowledge their part in the request or because there are no matches.

### 5.3.1. Duo Request

In this type of request, there are always two students involved. Only one of these students must file a request, but the exchange between classes must be discussed between both participants beforehand. Students aren't supposed to submit a request if they haven't previously consulted the other specified student. The reason why is that students who haven't consented to a request will then receive emails (with a confirmation link) that don't matter and should be avoided.

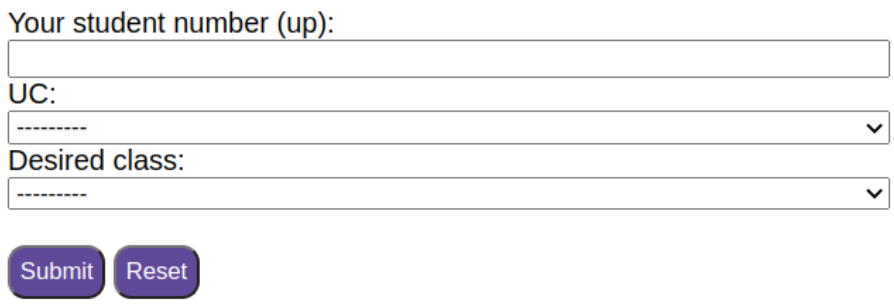
Preview of a two-person request:




### 5.3.2. Single Request

In this type of request, as opposed to what was previously discussed, only one student is involved. The results (if a student was placed in their desired class or not) depend on other single requests, so, if there aren't many for example, it's not likely a student will be placed in their desired class.

Preview of a one-person request:



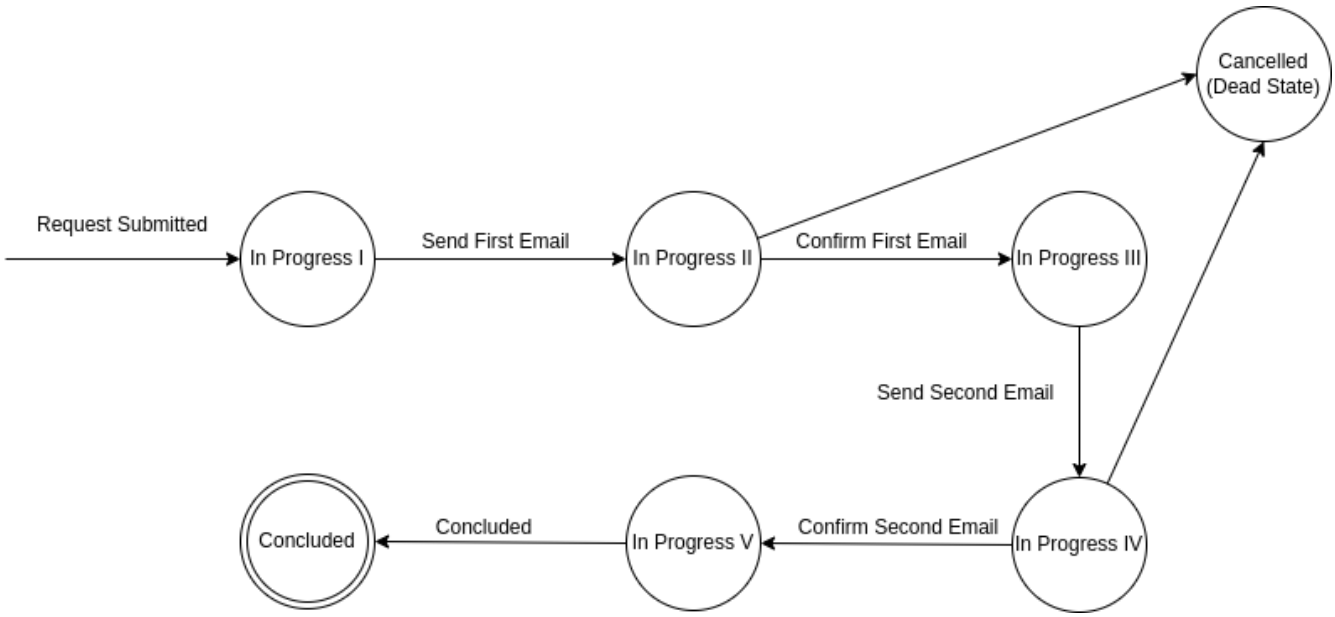

Since this wasn't a part of the project's goals, the algorithm developed to make matches is simple. It's a brute force algorithm that matches everything it finds. This solution might or might not be optimal.

## 5.4. Request States

A valid request has three main different stages:

1. In progress – At most one student confirmed;
2. Concluded – Both students confirmed;
3. Cancelled – Timeout occurred.

How a request works (passing the several states):




## 5.5. Validate a Request

After a student files a request (with two students involved), there are some criteria that need to be checked.

In a (duo) request, there are three different fields:

- Email (Student 1);
- Email (Student 2);
- Curricular Unit

After submitting a request, it's up to the application to check if all the following conditions are true:

1. Both introduced emails are valid.
2. The student exists in the database;
3. Student 1 is different from Student 2;
4. Both students are enrolled in the indicated curricular unit;
5. There won't be any overlapped classes if this request is concluded;
6. None of these students has an on going (not concluded) request for the specified curricular unit.

### 5.5.1. Emails

A valid email needs to, in most cases, start with "up", followed by nine numbers, then "@" and end with "up.pt". What comes between "@" and "up.pt" can vary. The email structure is checked using Regular Expressions.

A few examples of acceptable emails are “up201912345@fe.up.pt”, “up201912345@fc.up.pt”, “up201912345@edu.fe.up.pt”.

Some more examples of what isn’t accepted: “up123@fe.up.pt”, “up201912345.fe.up.pt”, “up201912345@gmail.com” and “student@fe.up.pt”.

Emails are sent using SMTP service. For this project, a first email was created (swappingfeup@gmail.com). Due to Google's new restrictions (started on the 1st of June 2022) about using Gmail via SMTP, this email and Gmail/Google server could no longer be used for this purpose. This was now a new challenge because, probably for similar reasons, other SMTP email servers were also unavailable.

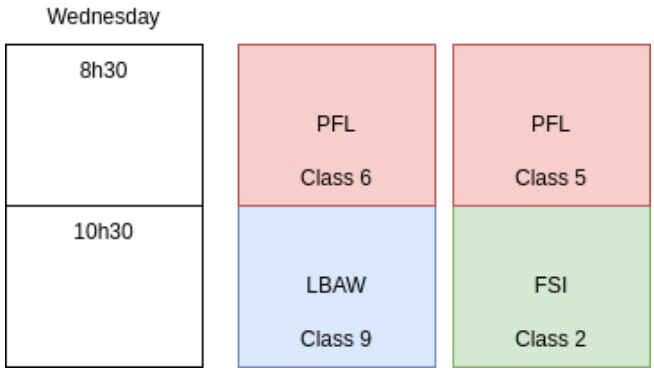
To solve this problem the application is using FEUP’s SMTP service (smtp.fe.up.pt).

### 5.5.2. Schedule

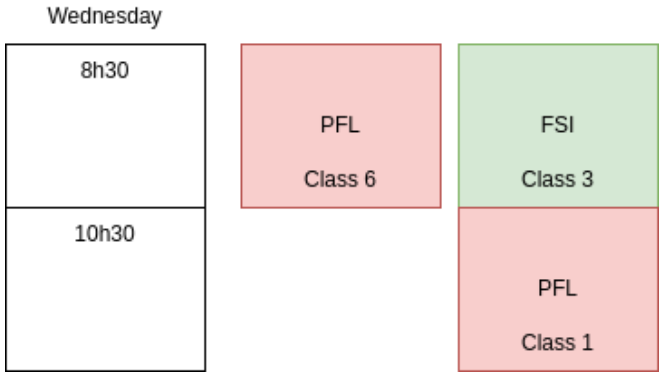
The developed application doesn’t allow students to swap classes if their new schedule is going to overlap with other practical or lab classes. Theoretical classes are allowed to be overlapped.

Examples where there isn’t any schedule overlap with other curricular units practical or lab classes:

1. PFL Class 6 stops existing, leaving an open spot that will be occupied by PFL Class 5.

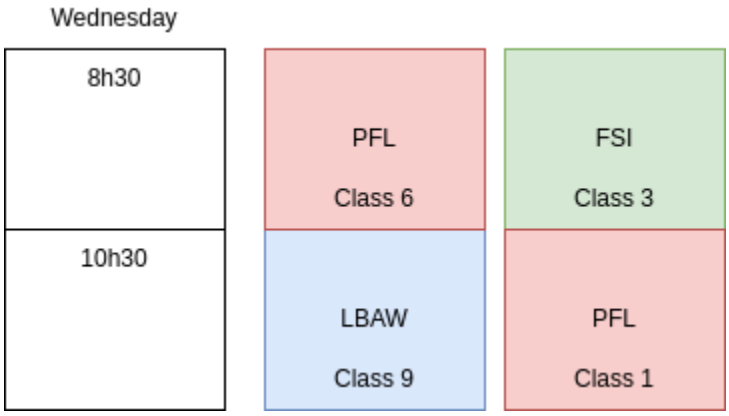



2. There is a free spot for PFL Class 1 in the left column.



An example where a request wouldn't be valid:

1. Student in PFL Class 6 can't change to PFL Class 1 because overlaps LBAW Class 9 and Student in PFL Class 1 can't change to PFL Class 6 because it overlaps FSI Class 3.




## 5.6. Permissions

Students don't need to authenticate themselves in this application since there isn't much they can do in the website besides consulting static pages and filing requests. Due to this, only administrators (department's staff) need to be authenticated to perform their roles. Since there are only two different actors here and one of them doesn't need to have an account, all logged members have the same (administrator's) permissions.

If a student or someone who isn't logged-in tries to access an administration page, then they will be redirected to a static page stating they don't have permissions to access that content.

Any authenticated member can configure the timeout, import and export files to the database, view all requests, clean all data, enable and disable either single or duo requests and see the matches for single requests.

Note: When an Administrator wants to export data there are two options: export all changed (students that changed class after a concluded request) and unchanged content or just the changes.

## 5.7. Links

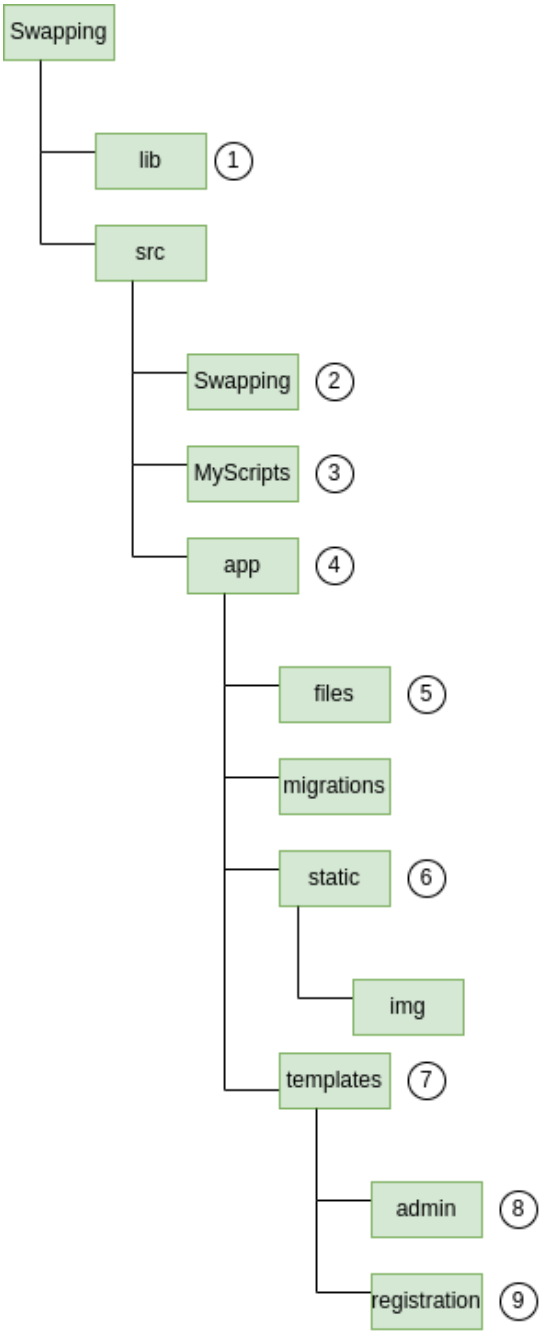
After a request is verified and submitted there are a few different dynamically generated URLs. These are the (two) confirmation links which are sent via email and a link (also sent by email) that allows students to check their request state.

## 5.8. Website Settings

There are website settings which save administrative choices, such as the request allowance (both for single and two-people/duo requests) and the current configured timeout.

Since it wasn't appropriate to create a table for only one object, these settings and preferences are stored in a json file (adminSettings.json).

## 5.9. Code Structure




1. Few informative files and images.
2. Python files, such as settings.py where project's specifications are located.
3. A few commands that were useful in the course of this project.
4. Contains several files with most of the developed code:

| File Name | Description |
|---|---|
| **admin.py** | Registers the models so they are available in Django's Admin Interface. |
| **ioFiles.py** | Procedures regarding the importation or exportation of data |
| **models.py** | Where models are created. |
| **forms.py** | All generated forms. |
| **readers.py** | Personalized procedures to read imported sigarra's csv files. |
| **settingsUtils.py** | Administrator settings (such as request allowance and timeout). |
| **urls.py** | All routes. |
| **utils.py** | Random useful functions mostly used in other procedures. |
| **validators.py** | Checks for a request validity. |
| **views.py** | All views. |

5. Mostly debug and importation files.
6. Images, Javascript and CSS files can be found here.
7. All HTML files, including the email's template.
8. HTML files regarding admin's features.
9. HTML files regarding the authentication system.

## 6. Deployment

Domain: swapclasses.fe.up.pt

E-mail: swapclasses@fe.up.pt

## 7. What could be improved

Due to time restrictions, there are a lot of useful features ideas that weren't in this project's objectives, so they were never implemented. If there was to be any continuation of this project the features or aspects I would suggest being implemented are the following:

- Front-end – At the start of this project I thought React would be a suitable technology to build the website's frontend since it is a growing community framework and has a lot of features. The front-end part of the website isn't more developed since the back-end was more essential and time-consuming.
- Tests – Even though every feature was tested to see if it was working correctly, there aren't any "official" tests. It would have been time consuming and not a big priority to write tests using an adequate package/library. Since most of the technologies, packages and libraries were only learnt during this project, learning more about test libraries would take longer.

# 8. Conclusion

During this project, I acquired a new skill set. It was challenging to develop this website alone and to learn new technologies. This made me become a more autonomous and self-taught person. I also had the opportunity to apply some concepts and algorithms I learned in my academic route. A few useful curricular units to this project I have previously completed are Databases, Database and Web Applications Laboratory, Software Engineering, Software Design and Testing, Algorithm Design, Web Languages and Technologies and Computer Security Foundations.

Overall, it was an interesting theme and a real-life problem for both students and the responsible staff. I accomplished everything I was supposed to, and more, in the given period of time.