Building a web application for student project allocation

Bachelor Project in Computer Science

Andreas Twisttmann Askholm(aaskh20)

Advisor: Marco Chiarandini

2023-06-01

Repository containing all files from this project:
`https://github.com/WaterCres/Bachelor`
Website on which the appplication is hosted:
`https://adsigno.sdu.dk/`

Contents

Abstract

På enkelte fakulteter på Syddansk Universitet, udgiver underviserne en liste af tilgængelige emner når de studerende skal arbejde på projekter. Ud fra denne liste indgiver de studerende en prioriteret liste over hvilke emner de helst vil arbejde med, hvorefter emnerne bliver fordelt baseret på de studerendes præferencer. Denne fordeling af projekterne sker med et program kontrolleret via kommandolinien og som kun er tilgængeligt på en enkelt computer.
I denne rapport beskriver jeg arbejdet med at udvikle en web applikation hvis formål er at virke som en grafisk brugerflade til fordelings programmet, og gøre det tilgængeligt over internettet for at gøre brugen af programmet nemmere og mere tilgængelig.

I rapporten beskriver jeg funktionaliteten af fordelings programmet, hvorfor jeg valgte at bruge Django og SQLite som base i udviklingen af applikationen og giver eksempler på alternativer til disse.
Jeg gennemgår hvordan applikationen blev gjort tilgængelig over internettet ved hjælp af Gunicorn og NGINX, vejen fra kode til kørende applikation og hvordan applikationen blev testet .
Til slut beskriver jeg hvad der kan gøres for at gøre applikationen tilgængelig og brugbar i et miljø udenfor Syddansk Universitet og giver bud på funktioner der kan implementeres i fremtiden for at øge funktionaliteten og forbedre brugeroplevelsen af applikationen.

Introduction

In this project the main goal is to build a web application for student project allocation. Every year at the University of Southern Denmark, students from several different study programmes have to work on various projects relating to their specific study programme. For some of those projects the students are presented with a list of available topics, offered by their teachers. Each student, or group of students, must be assigned to a topic to work on. The allocation of the topics is based on the preferences of the students. Some of these cases involve hundreds of students and topics to be paired in a way that satisfies everyone involved, be it the teachers who offer the topics or the students who have to work on them.
To solve the problem of allocating students to topics an implementation of the algorithm described in [2] is used.
The algorithm has been implemented as a python program, which currently runs as a console application. The users of the application has requested a user friendly interface to increase accessibility and usability, especially in relation to submitting projects and starting the allocation process. Visualising the results of the allocation was also requested as a way to show everyone involved that the assignment is in fact the most fair solution to a given case.

The workflow for project allocation is as follows:
When the time comes for a group of students to select a project, they are presented with a list of all topics available. From this list each student, or group of students, selects the topics on which they wish to work, and submits a prioritized list of their chosen topics. At a predefined deadline the submitted lists are put in to the application, used for allocating and the results of this allocation is published. This workflow is visualised in Figure 1. The purpose of the web application is to automate as much of this workflow as possible.

In building the application I have used skills and knowledge gathered during the study of computer science at SDU.

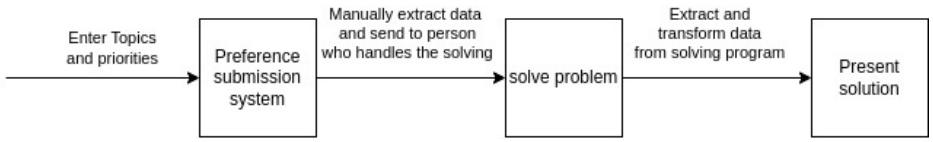

Figure 1: Diagram showing the workflow of the process of assigning students to topics

While building a web application is in itself a complex task, a substantial part of this project has been the research of available tools and deciding why some were more suitable than others.

Several sub-goals have been defined in order to break down the project into smaller and more accessible chunks, to more easily measure progress and provide a path from start to end. These sub-goals will constitute the chapters of this report.

Terminology in regards to the assignment program

Topic:
A subject on which a student can work.
Team:
A group of students assigned to the same topic.
Project:
A specific instance of a Topic and a Team. e.g. Team "a" in Topic 7.
Case:
A collection of students and topics.
Scenario:
A specific assignment of students to projects in a case.
Group:
A set of students with the same preferences to be assigned to the same topic.

The Application

Before starting the work on the web application I need to know the requirements, in order to make a decision on the best way to proceed. As the first step I need to know how the implementation of the allocation algorithm [2] work and how communication between the web application and the program are to be handled.

The implementation is written in python and can be installed as a module through pip. For the program to solve the problem it makes use of a 3rd party solver, Gurobi [1], in order to perform the calculations needed.
This program, when given a case to solve, computes an optimal solution to the problem and saves that solution, along with logs containing information about it, as files on the computer on which it is running.

Input

As input the allocation program requires four files containing information about the case to perform the computation:

- A file containing information about all the available topics to be assigned to students.
- A file containing information about the students to be assigned.
- A file containing information about what type of student can be assigned to which topics.
- A file specifying how many groups each advisors is able to manage.

When a user wants to start the computation the allocation software expects two arguments, a path to the folder containing the required files and a set of key:value pairs specifying the parameters needed to perfom the computation. These parameters are the following:

- `allsol`: Indicating whether or not all the best solutions should be presented or only one of them.
- `instability`: Indicating whether unstable solutions are allowed. Instability is when an incentive exists for a student to change to another topic, for example when a student has been assigned to their 3rd priority but their 1st or 2nd priority still has capacity left to receive other students.

- `expand_topics`: Indicating whether the file containing information about the topics is already expanded into teams.
- `groups`: Indicating whether groups are predefined and locked or if students can be added to a group during the computation.
- `Wmethod`: Which method should be used to determine if a solution is optimal.
- `cut_off_type`: Some students may have a special type giving them an advantage regarding the priority list. If such a type should be taken into consideration it is given as this parameter.
- `cut_off`: A number specifying the worst priority a student with the `cut_off_type` can be assigned to.
- `prioritize_all`: A true/false check, if this is set to true all topics will be appended to the priority lists of all students, tied and the lowest priority.
- `allow_unassigned`: Whether or not students can be placed into placeholder projects in the case there is not enough capacity.
- `min_preferences`: The minimum number of preferences a student must have submitted.
- `solution_file`: Path to where the file containing the solution should be placed.

All the input parameters of the computation should be adjustable by the user from an easy to use interface. Since the number of students and topics could be hundreds, importing them into the web application manually one at a time was not feasible and should be handled by importing the data from an excel document. The decision to use excel documents was made based on the intended users familiarity with the previous workflow of extracting the data from a database as an excel document.

Output

When a solution has been found a number of files are created, in which the solution is stored. These files also store some statistics of the solution and logs of the computation. Specifically the output are files containing the following:

- Statistics of the solution such as: number of students, number of teams and the number of students who was assigned to priority n.
- A list of students with incentive to want another topic.
- The assignment of students to projects for the solution.
- A summary of the projects used in the computation.
- A summary of all students' list's of preferences.
- A summary of the students used in the computation.

This output data should then be read by the web application and presented in a user-friendly and transparent manner, such that none of the advisors or students feel "cheated" when the solution is presented. The presentation of the solution should be through relevant diagrams and tables.

Tools

After reading through the code and documentation of the assignment application, a list of requirements to the web application was compiled.
The web application has to:

1. Allow input data to be uploaded by an administrator.
2. Interact with the python application.
3. Store data persistently in a database.
4. Visualise the results of the assignment process.
5. Be accessible over the internet.
6. Be easy to test and update, via automation.

A virtual machine and an accompanying domain name was provided by SDU for the purpose of hosting the application.

Following the requirements gathering the next step in the process was to figure out which tools and technologies could be used to build an application fulfilling these requirements. First of all I needed to find a framework in which I could build the application.

Framework

According to Matt Makai in his book Full stack Python [12], "Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications." In essence the framework handles all the underlying functionality of a web application, like URL routing and input form handling. These are functions not imediatly visible to the user. In addition, the framework handles the rendering of templates into dynamic HTML pages, which is what the user sees in their browser.

Since the web application has to interact closely with the assignment application I decided to use a python framework.

Of the many available python web frameworks, Django and Flask consistently appeared in every search query I made. According to a survey [16] made by Stack Overflow, they are also the most widely used python frameworks, which suggests they are robust frameworks, with a high availability of helpful resources.

Terminology in regards to python web frameworks

View:
A view defines the response to an http request.
Template:
A file used to generate the desired content.
Decorator:
A way to modify a function by wrapping it in another function.
Render:
Turning a template into an HTML page.

Flask

Flask [17] is presented as a minimal framework which only includes the basic functionalities a developer needs to build a website using python [20], these basic functionalities mostly consists of routing and rendering.
Everything beyond the very basics is performed by 3rd-party extensions, for instance if the developer wants to store data in a database, a Flask application generally relies on an extension for the application to communicate with the database.

Flask uses a templating language called Jinja2, which allows a developer to write html templates which includes functions with a python like syntax. These functions make it possible to, for example, iterate over a list and for every item in that list, create a component on the page.

Figures 2 and 3 below, shows two simple examples of a Flask application.
In a Flask application views are defined as functions, and URL routes are defined using decorators such as `@app.route('/')`.
Decorators can also be used to separate views based on HTTP methods, allowing one template to be rendered for a GET request, and another template for a POST request on the same URL.

With regards to security in a Flask application [18] only cross-site scripting is prevented, and only when HTML pages are generated using Jinja2. Cross-site scripting [11] is a type of attack where a malicious party executes malicious code on a user's computer, through a vulnerable website. Other types of attacks such as cross site request forgery [10] where a forged request is send to a user, forcing the user user to perform an action on the website,

are handled by extensions. Management of http request headers, including security headers, are handled by yet another extension.

The reliance of extensions allows a developer building a Flask application to only include the functionalities needed for that particular application. This approach means the application will indeed be minimal, however it might complicate the development process when the developer has to find the right or best extension to perform a given task.

Django

Django is presented as a framework that includes everything needed to develop a web application "from concept to completion as quickly as possible" [6]. Django includes an object-relational mapper (ORM) which allows a developer to define a relational database as a collection of python classes, illustrated in Figure 4, and makes it possible to write queries as python expressions instead of SQL statements. Django officially supports five different databases, PostgreSQL, MariaDB, MySQL, Oracle and SQLite [4], extensions can be installed to add support for other databases if needed. Django automatically creates an admin page from which entries in the database can be created, deleted and edited.

Django natively supports two templating languages, Jinja2 and Django templating language, they use a very similar syntax [19], Jinja2 being inspired by Django[1].

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Figure 2: An example of a very simple Flask application, in which a user is presented with a simple page showing "Hello, World!" when navigating to the URL "/".

```python
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Figure 3: An example of a minimal flask application using templates, in which a user is shown the contents of "hello.html" when navigating to "/hello/" or "/hello/<name>".

[1]Mentioned briefly in the jinja2 documentation

Views in Django, like in Flask, are generally written as functions, however class based views, where views are instead written as python classes are supported. URL routing is handled in a separate file, `urls.py`, in which a URL is mapped to a view function.

By default projects made using Django is organized in a predefined folder structure, automatically created by Django, illustrated in Figure 5.

Protection against several kinds of attacks, such as cross site scripting, cross site request forgery and SQL injections are included in Django by default.

The everything-included approach of Django allows a developer to build a web application by "filling in the blanks", this approach does however limit the freedom of the developer to choose which components to include.
In the podcast "Talk python to me" episode 149 [9] at timestamp 40:24, Nicholas Hunt-Walker summarizes the approach as follows: "...it encourages rapid development as long as you want to do things the Django way, which

---

```python
from django.db import models


class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)


class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

Figure 4: An example from the Django documentation of a simple database with two tables, Musician and Album.

```
setup/
│
├── example/
│   │
│   ├── migrations/
│   │   └── __init__.py
│   │
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
│
├── setup/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
└── manage.py
```

Figure 5: An example of the default folder structure of a Django project.

is oftentimes a good way to do things, but if you want to go outside that norm, you're going to have to pull apart some code to do that."

I decided to use the Django framework because the features it includes is a good fit for this project.

Persistent data

After deciding on a framework, the next step was to make a decision on which type of database to use. I found that the book Full Stack Python [13] was a great resource for finding information about commonly used relational databases, in particular MySQL, PostgreSQL and SQLite which are all supported by Django.

SQLite support is built in to python [7] enabling communication between the python application and the database without further configuration, while MySQL and PostgreSQL both require the inclusion of a separate driver libraries to facilitate communication.
SQLite is the default when starting a new Django project, changing to another supported database is however well documented and several step-by-step guides exists[2,3].
MySQL and PostgreSQL both work by having their own separate server processes which handle reads and writes to the database. The reliance on this server process makes it possible to have the database running on a remote host machine, allowing several applications, or multiple instances of the same application, to use the same database while not necessarily running on the same machine.
Access to a MySQL or PostgreSQL Database is restricted to only allow authorized users, or processes, to read and/or write to the database.
SQLite on the other hand relies on the file system of the host machine to perform reads and writes directly to an ordinary file. This approach limits the access to an SQLite database to processes running on the host machine, with no way of controlling the access to the file besides the access control provided by the host operating system.

When choosing a database for a web application a major consideration is the concurrency aspect, a user should be able to read from and write to the database without disrupting the work of other users of the application.

[2]How to connect mySQL to django:
https://www.javatpoint.com/how-to-connect-mysql-to-django

[3]How to Start Django Project with a Database(PostgreSQL):
https://stackpython.medium.com/how-to-start-django-project-with-a-database-postgresql-aaa1d74659d8

With concurrency in mind, the Django documentation advises against the use of SQLite [5], in an environment with many concurrent operations involving the database. Strategies do however exists to reduce the likelihood of concurrency errors, like increasing the timeout value and using the Write-Ahead logging mode [23, 15].
Considering the expected number of users is well within the range of what the SQLite documentation calls "low to medium" while also stating that "Generally speaking, any site that gets fewer than 100K hits/day should work fine with SQLite" [14], and since support for it is build into python I decided to use SQLite as the database for the application.
The extra work required to use either MySQL or PostgreSQL, was deemed unnecessary since SQLite would be sufficient for this project.

Architecture of the application

The architecture of the application is split into two parts, the UI a user sees and interacts with, and the underlying functionality, commonly referred to as the frontend and backend respectively. In this section I will describe considerations and decisions made regarding both, starting with the frontend.

Responsive and dynamic pages

Modern websites are generally expected to be usable on many different kinds of devices while having a consistent UI across varying screen sizes, in order to meet this expectation the responsiveness is a key part of designing the HTML templates [8].

To help in the development of the templates I use the CSS framework Bootstrap, which includes many predefined components which can be used in the design of the UI. Bootstrap components are designed with responsiveness in mind, it allows a developer to use a component which will render differently depending on screen size, like the navigation bar at the top of the page illustrated in Figure 6 and 7.

This use of predefined responsive components instead of explicitly defining what small, medium and large screens are and designing a page for each, reduces the complexity of designing a modern web UI.

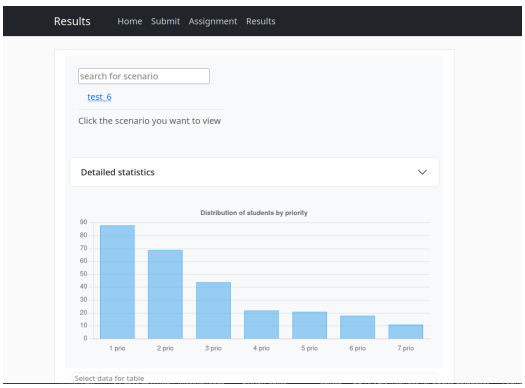

Figure 6: The results page of the application in its default view.

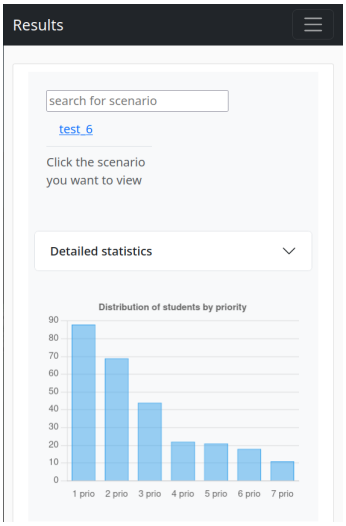

Figure 7: The results page of the application in a narrow view. The navigation bar at the top, has been collapsed into a "hamburger menu".

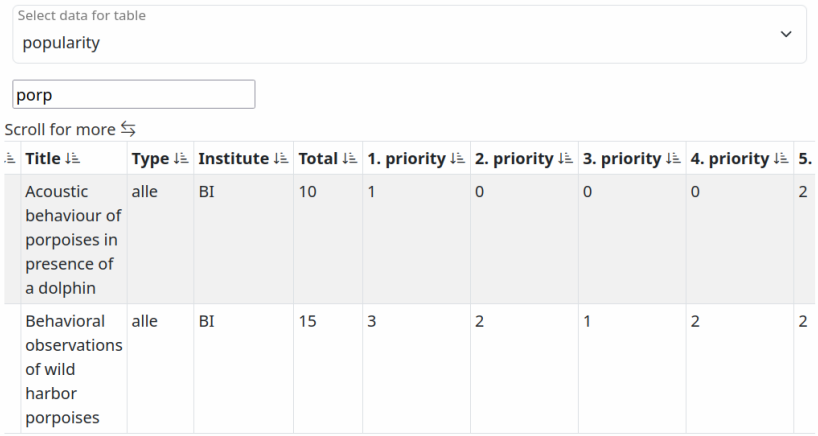

Figure 8: An example of a table which is too wide to be shown in its entirety and where a user has used the search field to limit the displayed entries to those containing the string "porp".

The Django template language supports the inclusion of one template in another and the creation of blocks[4], these features in combination means a single file can define the UI across multiple pages.
This approach ensures a consistent UI without repeating code, and if the design needs to be changed the change can be made in a single file instead of changing the same thing in all the templates.

In addition to having responsive pages I also wanted the pages to be somewhat dynamic, meaning some elements on the page would react or change in response to the user interaction without re-rendering the entire template. Bootstrap does include some dynamic components, like an accordion in which advanced options can be hidden, but more advanced features would have to be build using some form of JavaScript. I decided to include the JavaScript framework VueJS in my templates, which allowed me to build small components with the necessary functions and data, and then attach these components to elements defined in my HTML templates.

The combination of VueJS, Django Templating language and bootstrap allows me to build, among other things, tables which will be populated with data chosen from a drop-down menu, that are sortable and searchable on all fields and if the table is populated with too much data to fit the screen on which it is shown, the user can scroll sideways to view the rest of the data, example shown in Figure 8.

[4]Also supported by other template languages, e.g. Jinja2.

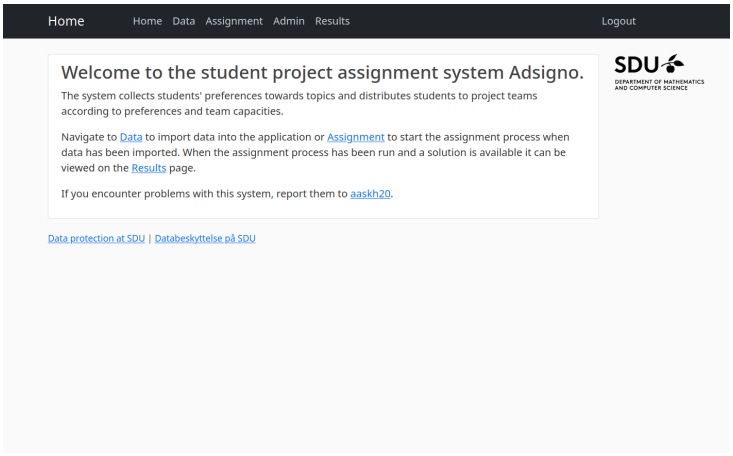

Figure 9: Home page

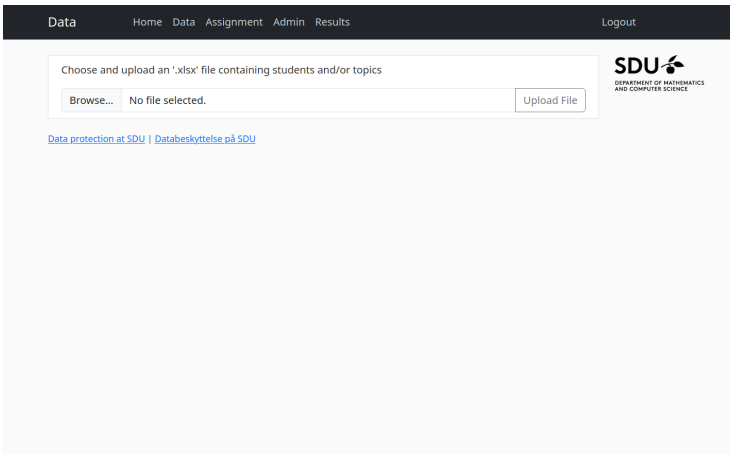

Figure 10: Data page

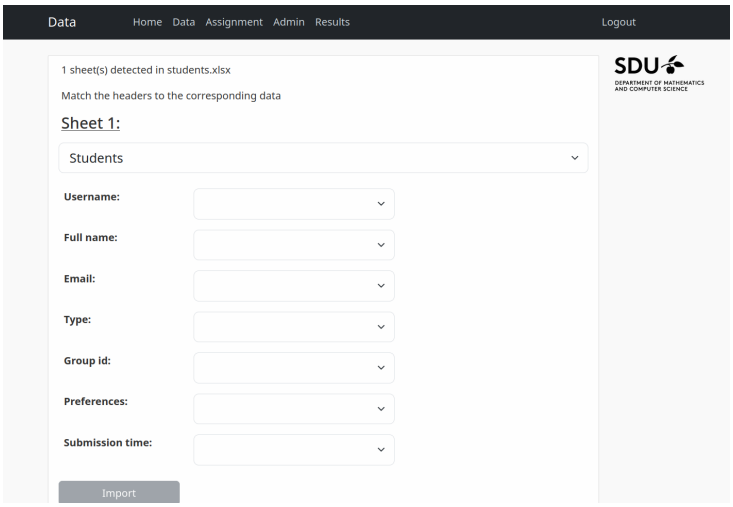

Figure 11: Header/data matching page

Workflow

When designing the web site, I wanted to match the workflow shown in Figure 1 as close as possible, with a page for each part aiming for a user experience as intuitive and familiar to the users as possible. This approach resulted in four distinct pages: home, data, assignment and results. All pages have the same navigation bar at the top allowing for easy navigation between them, and an additional page for administrators to assign permissions and make changes to the stored data.

The home page, shown in Figure 9, is the first thing a user sees, it serves as a welcome page and contains a short description of the application.

The data page, Figure 10, is where new topics and students are imported into the application. Excel spreadsheets containing students and/or topics, examples of these files are illustrated in Figures 12 and 13, can be uploaded and the user is then transferred to a page on which the headers of the uploaded document must be matched with the required data, Figure 11. This approach means the formatting of the file is less strict, while ensuring the data is imported correctly. The matching could be done automatically based on the title of the headers, this approach was however shown to be more robust during testing.

grp_id	username	type	priority_list	full_name	email	timestamp	stype
1	dummy21	biologi	20,61,54,14,45,5,22	Placeholder for real name21	dummy21@student.sdu.dk	25-02-2022 16:15	
2	dummy22	biologi	45,29,14,54,34,27,20	Placeholder for real name22	dummy22@student.sdu.dk	25-02-2022 16:21	
3	dummy23	farmaci	33,35,53,12,16,17,24	Placeholder for real name23	dummy23@student.sdu.dk	25-02-2022 16:27	
4	dummy24	biologi	20,61,25,36,54,45,5	Placeholder for real name24	dummy24@student.sdu.dk	25-02-2022 16:39	
5	dummy25	biomedicin	34,47,15,11,26,43,62,49	Placeholder for real name25	dummy25@student.sdu.dk	25-02-2022 17:01	
7	dummy26	biomedicin	6,23,8,31,4,7,30	Placeholder for real name26	dummy26@student.sdu.dk	25-02-2022 19:12	
8	dummy27	biologi	6,13,29,20,14,5,54,61,45	Placeholder for real name27	dummy27@student.sdu.dk	25-02-2022 21:32	
9	dummy28	biologi	1,5,22,20,14,45,54	Placeholder for real name28	dummy28@student.sdu.dk	25-02-2022 23:29	
10	dummy29	biomedicin	4,13,62,34,30,3,11,27,21,43	Placeholder for real name29	dummy29@student.sdu.dk	26-02-2022 08:05	
11	dummy30	farmaci	10,24,28,59,31,60,52,41,46	Placeholder for real name30	dummy30@student.sdu.dk	26-02-2022 11:30	
12	dummy31	biomedicin	34,39,37,23,4,11,30	Placeholder for real name31	dummy31@student.sdu.dk	26-02-2022 11:43	
13	dummy32	biologi	20,61,54,45,36,14,5	Placeholder for real name32	dummy32@student.sdu.dk	26-02-2022 12:10	
14	dummy33	kemi	40,8,41,51,61,20,44,2	Placeholder for real name33	dummy33@student.sdu.dk	26-02-2022 16:06	
15	dummy34	biologi	54,45,29,61,5,34,44,25	Placeholder for real name34	dummy34@student.sdu.dk	26-02-2022 16:50	
16	dummy35	biomedicin	4,13,34,18,7,37,1,3,50,27	Placeholder for real name35	dummy35@student.sdu.dk	26-02-2022 19:53	
17	dummy36	farmaci	53,10,35,33,28,12,24	Placeholder for real name36	dummy36@student.sdu.dk	26-02-2022 21:11	
18	dummy37	biomedicin	34,4,27,30,49,5,15	Placeholder for real name37	dummy37@student.sdu.dk	26-02-2022 22:25	

Figure 12: An example of a file containing information about the students involved in a case.

Proj	Pa	Underp	Projektnavn	Min	Max	Type	Institutforkort	Institut	Minikursus obl	Min	Gruppeplacering
01	1		Acoustic behaviour of porpoises in presence o	3	4	alle	BI	Biologisk Institut	Posterfremstilling		Kerteminde Marinbiologisk Forskningscenter
02	2		Affaldsplast som et vedvarende materiale og e	3	5	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Kemi
03	3		Analyse af syntetiske og biologiske lipid-nano	3	4	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		FKF
04	4		Antibiotika på&aring: plejehjem - hvor stort er fo	3	6	Farmaci	SUND	Institut for Sundhedstjenesteforskning	Posterfremstilling,Sundhedsvidensk		Det Sundhedsfaglige Fakultet, J.B. Winsl&oslash ws Vej 19, 2. sal
05	5		BEERomics	3	6	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Institute of physics, chemistry and farmaci
06	6		Behavioral observations of wild harbor porpois	3	4	alle	BI	Biologisk Institut	Posterfremstilling,Posterfremstilling		Marine Biological Research Center Kerteminde
07	7		Biologisk datamining og eksperimentel under	3	6	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		BMB
08a	8a		Brown fat - A silver bullet against obesity?	3	5	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		Institut
08b	8b		Brown fat - A silver bullet against obesity?	3	5	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		Institut
09	9		Brug af nanopartikler til at udforske proteiner	3	4	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		Institut
10	10		Brystkræft: Mere end bare én sygdom	3	5	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		BMB, Campus
11	11		Cellulær identitet og dysfunktion i nonalkoholi	3	5	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		BMB, campus
12	12		Chromatin and gene regulation by histone pro	3	5	alle	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		Proteinforskningsgrupper/BMB
13a	13a		Design af lysaktiverede forbindelser til kemot	3	4	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		FKF&#39 s terminalrum eller U26B
13b	13b		Design af lysaktiverede forbindelser til kemot	3	4	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		FKF&#39 s terminalrum eller U26B
14	14		DNA/RNA i moderne lægemidler	3	4	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		FKF
15a	15a		DRUG LEGO : How drugs fit into proteins, usin	3	5	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Intitut
15b	15b		DRUG LEGO : How drugs fit into proteins, usin	3	5	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Intitut
15c	15c		DRUG LEGO : How drugs fit into proteins, usin	3	5	alle	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Intitut
16	16		Efflux transporters in drug delivery	3	5	Farmaci	FKF	Institut for Fysik, Kemi og Farmaci	Posterfremstilling		Bibliotek
17	17		Establishing a mouse model of lung tumorige	3	5	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		BMB
18	18		Exploring molecular mechanisms of cell death	3	4	Biologi, Biomedicin, BM	BMB	Institut for Biokemi og Molekylær Biologi	Posterfremstilling		BMB

Figure 13: An example of a file containing information about the topics to which students should be assigned.

When a case, consisting of both students and topics, has been imported into the application, the allocation process can be started from the assignment page shown in Figures 14 and 15. All the parameters for the process are presented and adjustable on this page.

On the results page the different scenarios are listed at the top, along with a search field to quickly find a specific scenario. Selecting a scenario will present the details of the scenario beginning with a bar chart, pictured in Figure 6 showing the distribution of students according to their priorities, followed by the table shown in Figure 8 containing detailed information chosen from a drop-down menu.

In order to assign permissions to users and view/update the data in the database, authorized users have access to the Django administration page, Figure 16, on which these actions can be performed.

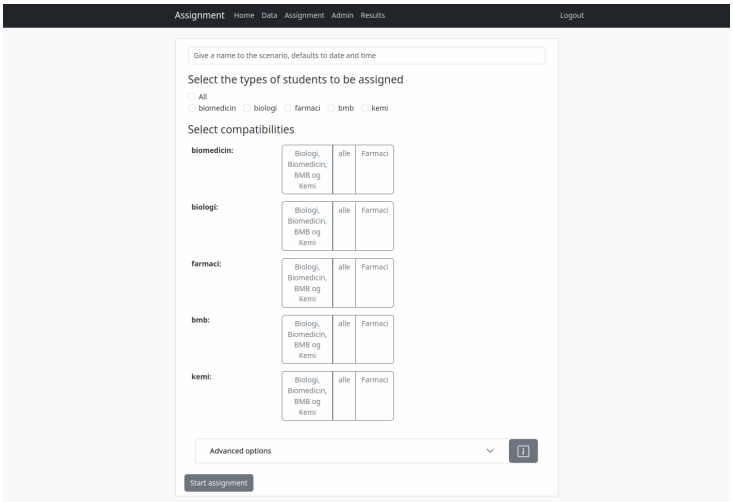

Figure 14: Assignment page, basic view

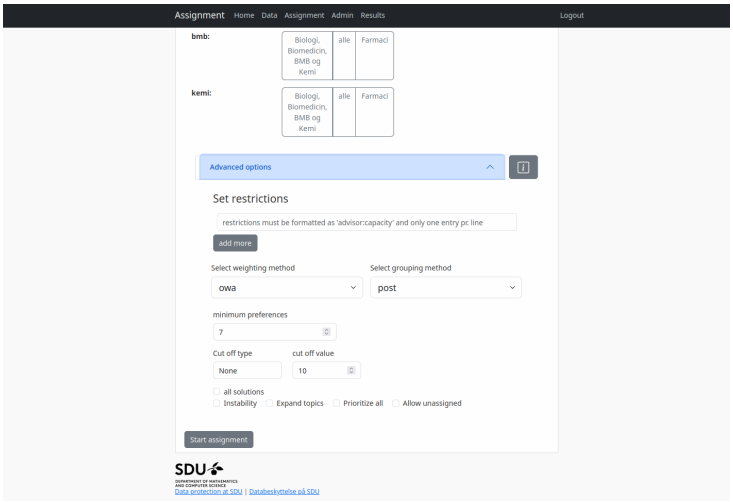

Figure 15: Assignment page, showing advanced options

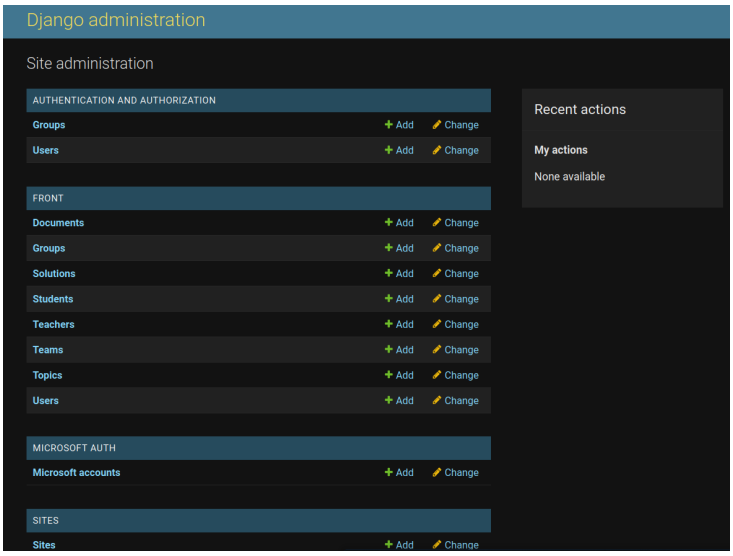

Figure 16: Administration page

```
from django.db.backends.signals import connection_created

def activate_wal_mode(sender, connection, **kwargs):
    """Enable Write ahead logging in sqlite."""
    if connection.vendor == 'sqlite':
        cursor = connection.cursor()
        cursor.execute('PRAGMA journal_mode=WAL;')

connection_created.connect(activate_wal_mode)
```

Figure 17: Code snippet enabling WAL-mode on the SQLite database during initialisation

Database

Since the database supporting the application is an SQLite instance very little configuration is needed before it can be used. When building a Django application and using an SQLite database the only configuration strictly needed is done by default and consists of two lines of code. The first line defines the engine needed to communicate with the database and the second line is a path to the file containing the database.

To addres the concerns regarding concurrency however, means a little more configuration is needed. The "timeout" value is increased to 20 this allows a thread wanting to access the database to wait for up to 20 seconds in case the database is currently locked by another thread.

Another attempt at mitigating potential problems arising from concurrent access to the database is to enable Write Ahead Logging mode (WAL-mode). This is done by including the code from Figure 17 in the `/front/__init__.py` file, which means it will be executed when the application is initialised.

By default an instance of an SQLite database will write changes directly to the database file, while saving a separate rollback journal. WAL-mode instead appends any changes to the end of a buffer file without changing the contents of the database file, these changes are written to the database file when the wal buffer reaches 1000 pages by default.

When a read operation is started a mark is set at the current end of the WAL buffer, this mark will not change during the read. The reader then reads from the beginning of the WAl buffer to the marked end of the buffer and, if the needed data was not found, it continues its operation in the original database file.

This approach means a write operation can be performed without interfering with an ongoing read operation, write operations can however only be performed by one thread at a time.

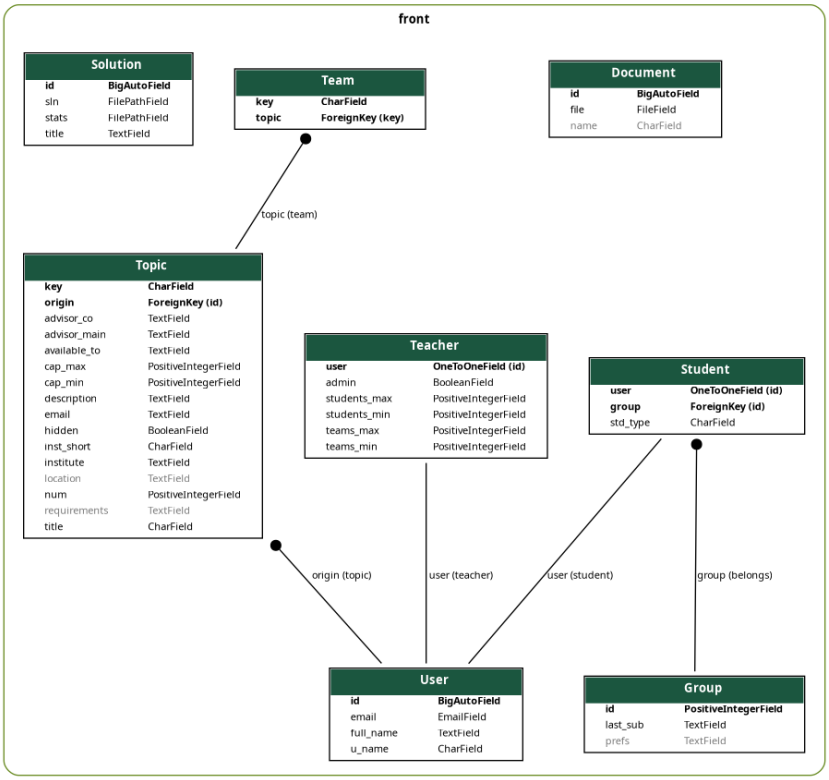

Figure 18: ER diagram of the database used in the application. Automatically generated with "django-extensions" and Graphviz.

Tables and relations

All tables and relations used in connection with the application are defined in the `/front/models.py` file. When designing the database I started by making the tables "Topic" and "Student" since these were the entities that needed to be paired, building from this starting point the database evolved into what is shown in Figure 18. While the actual pairing is between a group of students, consisting of one or more students and a project, this relation is not explicitly present in the Entity Relation (ER) diagram or the database but rather contained in the Solution entity as the "sln" attribute. The Solution entity contains as attributes paths to the files created by the allocation application, and a title for users to easily differentiate between the solutions when they are listed on the results page.

While importing documents containing multiple students and/or topics are done on the data page, if a user wants to create, read, update or delete[5] individual entries in the database, the default admin page created when starting a new django project is utilised. On this page any user who is signed in and has the proper permissions can edit or create individual entries in the database and delete one or more entries.

[5]Commonly referred to as CRUD operations

User authentication

Some of the pages in the web application contain information which should not be publicly accessible, like the email and user name of the students being assigned to topics, and functions that only authorised users should have access to, like starting the allocation process.
In order to restrict access to these pages to authorized users, a login system has been implemented where users can login using their SDU accounts.
When a user logs in with their SDU credentials, their account will either be linked to an existing authentication user with the same email address or a new user will be created and then linked to the SDU account.

When the application is started a super user with access and permissions to do everything is created, with credentials specified in a `.env` file, this file will be described on page 26. The email specified for this user determines which SDU user is able to set permissions for all subsequent users.
Five permission levels exist, each building on top of the previous one:

1. A guest is someone who is not logged in and only has access to the welcome page and the option to login.

2. A logged in user is allowed to view the solution of an assignment.

3. Teachers are allowed to upload documents to the system.

4. Users designated as "staff" are allowed to start the assignment process and view the admin page but, by default not view or edit any data in the database.

5. Super users who are allowed to make changes on the admin page.

Super users have the ability to assign permissions to staff users, allowing them to view and/or edit specific tables in the database.

I made the decision to rely on SDU authentication for user authentication, because the application aims to solve a problem experienced at SDU.
The problem might not however be exclusive to SDU but since the authentication at SDU is done using Azure Active Directory(AAD), it is however fairly simple to migrate the application to any other organization or entity who relies on AAD for user authentication, by creating an app registration in AAD and entering three codes into the `.env` file.

Deployment of the application

The purpose of the web application is to allow multiple users to use the allocation application. To achieve this goal the web application needs to be accessible over the internet. To access the application over the internet it needs to be running on a computer with an associated url address, both a virtual computer and a url were provided by SDU to be used in this project. In this section I will discuss both the hosting of a web application and the flow from coding to deployment.

Hosting a web application

When a client wants to visit the application, they navigate to the url address of the application in their web browser, a DNS request is then send to a DNS server to match the url with the IP address of the server on which the application is running. The clients web browser then sends a request to the server, requesting to view the application. The incoming request is then routed to the application which responds with the requested page. The connections established in this flow is illustrated in figure 19.

On the virtual machine an instance of NGINX is running as a reverse proxy which forwards all incoming requests to the requested application and serves static files like images, javascript and css files. Typically requests for websites are send to port 80 for http, and port 443 for https.

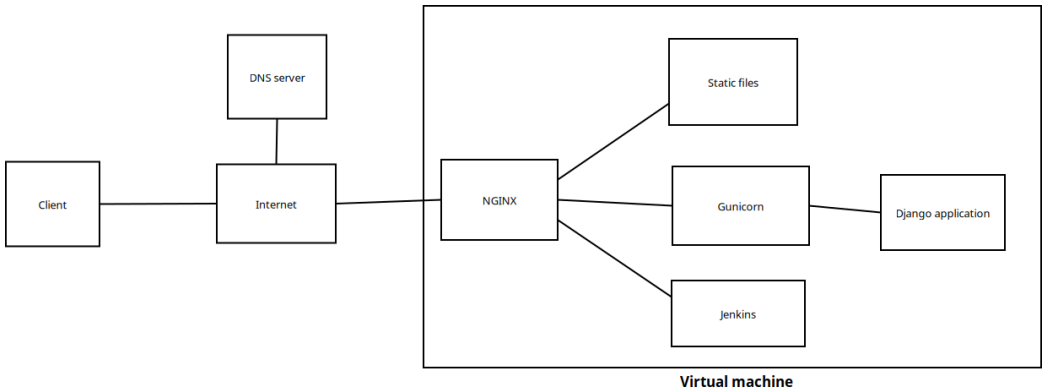

Figure 19: Diagram illustrating the connections from client to application. *NGINX* handles all requests arriving at the *Virtual machine*. If the *client* requests the *Django application*, *Gunicorn* responds with the templates from the *Django application* while *NGINX* responds with the *Static files*.

The running instance of NGINX is configured to redirect any incoming requests on port 80 to port 443 in order to ensure all traffic is encrypted using TLS[6].

TLS is a combination of asymmetric and symmetric encryption used to encrypt communication over the internet. To use TLS the server needs a certificate along with a public/private key pair[3].
When an https request reaches the server, the server responds with its certificate and public key. The client then verifies that the certificate is legitimate by asking a trusted certificate authority if the certificate is valid to ensure that the communication is between it and the expected server.
When the certificate has been verified the server and client communicates to each other what types of encryption they support, in order to use the strongest available. The client then uses the servers public key to encrypt a session key to be used for the current session and sends it to the server, which uses its private key to decrypt the session key. From that point and until the connection is closed, all requests and responses are encrypted using symmetric encryption.

In the case of this project a self signed certificate was used. A self signed certificate means the server can not be verified by an outside authority since anyone could have signed it. However the provided virtual machine itself sits behind a reverse proxy managed by SDU which handles the encryption of traffic to the internet and by extension the actual certificate used when accessing the application.

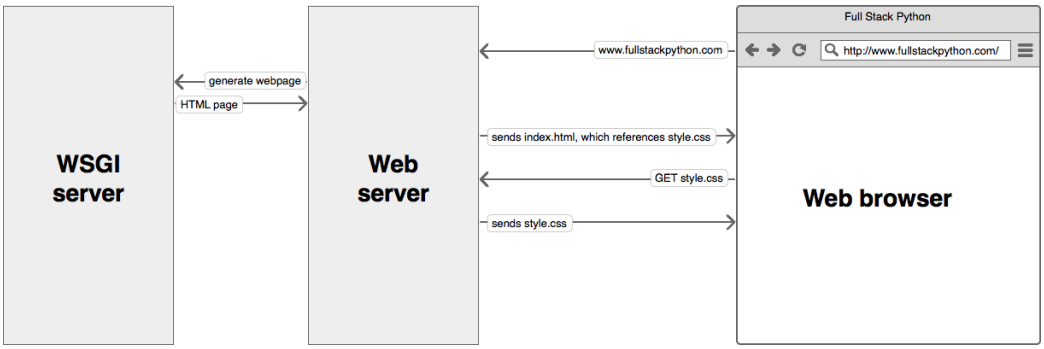

Figure 20: Diagram from Full stack Python illustrating the requests and responses between client, Web server(NGINX) and WSGI server(Gunicorn)

[6]Many online resources use SSL/TLS when referring to this encryption, the last version of SSL was however deprecated in 2015 which is why I will refer to it only as TLS

As illustrated in figure 19 the running instance of NGINX handles traffic to and from three resources: The Django application, an instance of Jenkins which will be discussed in the next section and the static files used by these applications. Since the Django application itself is just a collection of python scripts which is not runnable by a traditional web server, a web server gateway interface(WSGI) is needed to run the python application. Django does set up a minimal WSGI configuration when a new project is started and includes a lightweight development server which according to the documentation should only be used in a development environment. I chose to use Green Unicorn(Gunicorn) as the WSGI server for this particular project.
When a request from a client is send to Gunicorn its job is to handle that request and respond with an html page, the page may contain images and styling defined in a css file which is then served by NGINX, illustrated in figure 20.

From development to production

During the development of the application i used git as version control system to store the code and keep track of changes. Early in the development of the web application, it became apparent that manually accessing the virtual machine on which the application is hosted via ssh and then downloading the most recent version from git to update the version being hosted on the server, was a time consuming and inefficient way of updating the application. And that I needed a way to automate this process, the solution was to use a CI/CD tool[21].

Many different tools exists in this space so to limit the potential candidates I came up with a list of requirements the tool had to fulfil:

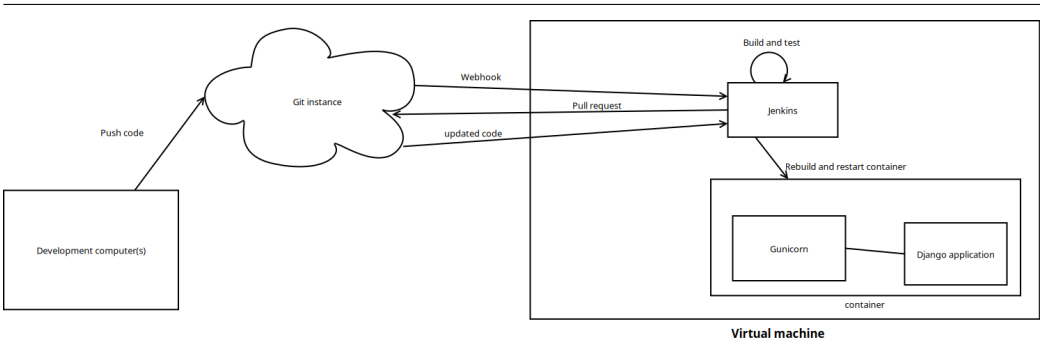

Figure 21: Diagram illustrating the flow from code to running application through git and Jenkins

1. It had to be free of charge.
   At the very least there had to be a free version I, as a student, could use for this project.

2. It had to run on the virtual machine.
   Since access to the machine was limited to connections originating from SDU's network the tool should run on the computer that hosted the application.

3. It had to support python.

4. It had to be able to build and deploy Docker containers, more on this in section .

The first two requirements in particular limited the number of potential candidates and in the end I decided to use Jenkins since it fulfilled all the requirements.
Jenkins has very good documentation and easy to follow guides, while still being a very powerful tool with many advanced features which is extensible by a large library of plug-ins. It has a webUI in which all settings can be accessed, and a complete NGINX configuration is available on the Jenkins website[22] to allow access over the internet.

Setting up the steps needed to pull and deploy the updated code was fairly straight forward since i was able to run the commands previously used to update the running instance of the application, from within Jenkins.
With the inclusion of Jenkins the path from code to running application, also called the pipeline, executes the following steps as illustrated in figure 21:

- Manually push changes to the git repository.
- A git webhook sends a "notification" to Jenkins that a push has happened.
- Jenkins pulls the updated code from the repository.
- Jenkins builds and tests the updated application in a test environment.
- Assuming the building and the tests pass, Jenkins again builds the updated container this time in the live environment, and restarts the running instance of the application with the updated container.

If the tests does not pass or the building process fails the step that failed can be viewed in the webUI.

Containerisation

The provided virtual machine runs linux as its operating system, I did not however want to limit the application to running on a specific operating system on which a user would have to install multiple specific programs and python packages in order to use the application. To run the application regardless of the operating system of the host machine, and to better control the environment in which the application would be running, containers provided the solution I needed.

Using Docker allows me to package all the individual components, NGINX, Jenkins and my web application along with Gunicorn, in separate containers each containing only what is needed to run an instance of the specific application and the only requirement for the host system is that it should be able to run Docker. It allows anyone to quickly start the applications and achieving an equivalent setup, by setting a few variables and then, with the use of Docker Compose, starting all containers with a single instruction given to Docker. All the instructions on how to build the individual containers are defined in a container specific dockerfile, this file defines which application or environment is needed and which operations should be performed in order to build the container. A Docker Compose file then defines what containers to start, the order in which they should be started and the networks and volumes available to the containers.

The variables that would need to be adjusted in order to start the applications on another machine are all contained within a `.env` file located in the same folder as the Docker Compose file. This file contains settings for both the Docker containers and for the Django application. This approach means that all variables, like where to store static files and the secret keys used for authentication, can be changed in a single place. Since this file contains secret keys it should not be uploaded to the git repository and is therefore ignored when changes are being tracked by git. Instead this file is manually transferred to the virtual machine, used for hosting the application.

Testing

When developing any kind of software, testing the application is an important step in the process. Tests are performed on the application to discover any potential bugs in the software before the application is deployed and used in a setting in which a bug could potentially be extremely disruptive and/or make the entire application unusable.
Additionally tests in which users interact with the application can be performed. These tests allows developers and users to collaborate and identify the components that work, and if some things need to be done in another way.

During the development of this application both kinds of test were performed. User testing where the application was presented to a representative of the intended end user in order receive feedback on the user experience and suggestions on features that would make the application more appealing to use. And technical tests whenever a new feature was implemented. The feature was tested by trying to break it, in order to either prevent or gracefully handle the failure of the feature. An example of this is when I tested the file uploading feature, several different file formats were uploaded as well as excel spreadsheets formatted in different ways. As a part of these tests a logging feature was created to write error messages to a file, in order to review what was attempted when the error happened.

User testing

Early in the process when the first prototype featuring the file upload and assignment pages was ready, a test session was arranged with an end user. The session started with a quick introduction to the application explaining its features, the current state and the scope of the test. The scope of this test was primarily to receive feedback on the navigation of the page, in addition to getting an impression of what features would be important to the user.
The outcome of the test was that the site was intuitive to navigate, it was however revealed that the format and content of the cells in the uploaded files would not necessarily be static or conform to the initially expected format. Furthermore some of the options on the assignment page could lead to confusion unless properly explained.

To solve these issues the data types saved to the database were reworked to better accommodate flexibility in the content of the cells of the uploaded spreadsheet, and an intermediate page was added on which the user is asked to match the headers of the uploaded file to the data needed to perform the

computation. On the assignment page some of the options were hidden under an "Advanced features" menu, with an attached section explaining what each option does.

Later in the process another user test was arranged. At that point the application had all the features and components needed to upload documents, start the computation and review the results along with the ability to log in and restrict the functionality based on the permissions given to a user. The primary scope of this test was to verify that the login and permission system worked as intended, and secondly to verify that the application could be used to solve the allocation problem and thereby solving the problem this project was based on.
This second test showed that the login and permission systems worked as intended. An issue was discovered where the user was unable to match headers to data in the uploaded document, when the application was accessed with the Microsoft Edge browser, this issue was not present when the user accessed the application with Firefox. The user was unable to start the allocation process, this issue was later revealed to be caused by misconfigured javascript which disabled the button used to start the process.
Based on feedback from the user some UI elements were redesigned in order to better match the workflow.

Conclusions and future work

My goals for this project were to build an interface where authorized users would be able to upload spreadsheets containing student and topic data. The users had to be able to start the allocation process and view the results of this process. Additionally a tool had to be setup in order to facilitate quick and automatic updates.

While my application along with the running instance of Jenkins has all the features and functionality I set out to implement, there are some aspects which could and maybe should be changed in future updates, or at least should be considered if the application were to be hosted outside of the SDU environment.

At the moment the SQLite database file is stored within the docker container. As a consequence of this the entire database is purged whenever the docker image is rebuilt. Meaning the data stored in the database will be deleted when the application is updated. To remedy this two solutions exists. One solution is to store the database file outside the docker container by linking a directory outside of the docker container to a directory inside it, and setting the path to the database file accordingly in the Django settings. This solution does not require a lot of development time, it does however leave the database vulnerable to manipulation from other processes running on the server. Alternatively the database could be migrated from the current SQLite database to an instance of MySQL or PostgreSQL running in a separate container. This approach does require more development work, but allows multiple running instances of the application to access the same database in a controlled manner, even if these instances were distributed among several computers.

Currently the NGINX configuration is tailored to run on a virtual machine provided by SDU.
In order to have the application running and accessible on another computer in general, a self signed certificate should not be used, instead something like Certbot could be utilised to automate the certificate process. The host address specified in the NGINX configuration and `.env` file will also have to be changed to match the new url address associated with the application. An alternative would be hosting the application on a service like Railway, this approach leaves the configuration and handling of the reverse proxy, certificates and DNS records to the service provider and lets a developer focus on the application itself.

With regards to the login system on the application, while the transition to another Azure Active Directory is fairly simple, as I described in the login section, transitioning to a completely different service does require a bit more work. The transition would require the removal of the Microsoft authentication module currently integrated into the application, and the setup of the new system along with potentially integrating a new module if the desired login system has a Django package.

Currently no automatic testing has been implemented in Jenkins or the unit test module included in Django. This should absolutely be remedied in the future, in order to avoid the introduction of potentially application breaking bugs to the running instance of the application.
Unit tests of the individual functions in the application can be written as python scripts to utilise the included test module. In addition to unit tests, tests could be created using Selenium in order to test the flow through the application and if the design and functionality is consistent between different browsers.
These tests should be executed by Jenkins before building and restarting the application.

In addition to the aspects outlined above, I would consider the following features to be great additions to the application in the future:

- The creation of a custom admin page.
  In order to keep the UI on this page consistent with all the other pages. This would also allow administrators of the application to get a more detailed overview of the data currently stored in the database since the current admin page does not display the data in a way that suits the needs of the users very well.

- Associate student and topic data to the case in which they are involved.
  Currently when the allocation process is started all selected student types and all projects present in the database will be considered during the allocation and while this works fine as long as only one case is present in the database, it creates a problem in the future when the application must handle data from several different cases.

- Update the data submission page to allow more flexibility.
  This flexibility includes the option to overwrite/update existing data when a document is uploaded and matching entries are found, an option

to manually enter a single entry in to the database from this page and allowing other file formats, in addition to `*.xlsx`, to be uploaded.

- An option to extract data from the application in a suitable file format. At the moment all the data used in the application is only accessible through the application, the option to extract certain data e.g. the files created as output from the computation, was requested by a user involved in the user tests.

During my work on this project I have become aware of, and familiar with several tools and technologies used in the world of software development, and identified the tools I found most suitable for this project.
Decisions made in the development process, may however need reconsideration if the requirements change.

References

[1] Marco Chiarandini. Project assignment README. https://github.com/belzebuu/ProjectAssignment/blob/master/README.md, jan 2023. [Online; accessed February-2023].

[2] Marco Chiarandini, Rolf Fagerberg, and Stefano Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research*, 275(1):39–78, 2019.

[3] F5. What is ssl/tls encryption? https://www.f5.com/glossary/ssl-tls-encryption, Jan 2023. [Online; accessed May-2023].

[4] Django Software Foundation. Django databses. https://docs.djangoproject.com/en/4.2/topics/install/#database-installation, March 2023. [Online; accessed March-2023].

[5] Django Software Foundation. Django databses. https://docs.djangoproject.com/en/4.2/ref/databases/#database-is-locked-errors, March 2023. [Online; accessed March-2023].

[6] Django Software Foundation. Django frontpage. https://www.djangoproject.com/, January 2023. [Online; accessed February-2023].

[7] Python Software Foundation. sqlite3 — db-api 2.0 interface for sqlite databases. https://docs.python.org/3/library/sqlite3.html#module-sqlite3, November 2012. [Online; accessed April-2023].

[8] Vitaly Friedman. Responsive web design: What it is and how to use it. https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/, Aug 11 2018. [Online; accessed April-2023].

[9] Michael Kennedy and Nicholas Hunt-Walker. Talk python to me. https://talkpython.fm/episodes/transcript/149/4-python-web-frameworks-compared, February 2018. [Online; accessed March-2023].

[10] KirstenS. Cross site request forgery (csrf). https://owasp.org/www-community/attacks/csrf, March 2020. [Online; accessed May-2023].

[11] KirstenS. Cross site scripting (xss). https://owasp.org/www-community/attacks/xss/, January 2020. [Online; accessed May-2023].

[12] Matt Makai. *Full Stack Python*. Matt Makai, March 2014. [Online; accessed March-2023].

[13] Matt Makai. *Full Stack Python*, chapter Relational Databases. Matt Makai, March 2014. [Online; accessed March-2023].

[14] Developers of SQLite. Appropriate uses for sqlite. https://www.sqlite.org/whentouse.html, February 2005. [Online; accessed March-2023].

[15] Developers of SQLite. Write-ahead logging. https://www.sqlite.org/wal.html, July 2010. [Online; accessed March-2023].

[16] Stack Overflow. Stack overflow 2022 developer survey. https://survey.stackoverflow.co/2022/#section-most-popular-technologies-web-frameworks-and-technologies, June 2022. [Online; accessed February-2023].

[17] Pallets. Flask quickstart. https://flask.palletsprojects.com/en/2.2.x/quickstart/, August 2022. [Online; accessed February-2023].

[18] Pallets. Flask security. https://flask.palletsprojects.com/en/2.2.x/security/, August 2022. [Online; accessed February-2023].

[19] Pallets. Switching from jinja to dtl. https://jinja.palletsprojects.com/en/3.1.x/switching/#django, April 2022. [Online; accessed May-2023].

[20] Pallets. What flask is, what flask is not. https://flask.palletsprojects.com/en/2.2.x/design/#what-flask-is-what-flask-is-not, August 2022. [Online; accessed February-2023].

[21] inc. Red Hat. What is CI/CD? https://www.redhat.com/en/topics/devops/what-is-ci-cd, May 2019. [Online; accessed March-2023].

[22] Jenkins the project. Jenkins documentation - nginx. https://www.jenkins.io/doc/book/system-administration/reverse-proxy-configuration-nginx, November 2020. [Online; accessed March-2023].

[23] Simon Willison. Weeknotes: Djangocon, sqlite in django, datasette-gunicorn. https://simonwillison.net/2022/Oct/23/datasette-gunicorn, Oct 2022. [Online; accessed March-2023].

Appendix

As an appendix to this report comes a zip archive, Adsigno.zip, which contains all the code I have written in connection with this project.