

UNIVERSITY OF TWENTE

DESIGN PROJECT REPORT

Peer Review Scheduler

Group 1

Ibrahim Teymurlu

Salih Eren Yuceturk

Ferhat Ege Darici

Zheyu Dong

Zinan Guo

Boxuan Wang

Supervisor

Dr. Petra van den Bos

Contents

1 Domain Analysis	5
1.1 Context	5
1.2 Problem Description	5
1.3 Our Solution	6
2 System proposal	7
2.1 Requirements Proposal	7
2.2 Prototype Proposal	8
2.3 Project Presentation	8
2.4 Results of Meetings	9
2.5 Time Planning	9
2.5.1 Planning	10
2.5.2 Designing	11
2.5.3 Development	11
2.5.4 Testing	11
3 Requirements Analysis	12
3.1 Approach	12
3.2 Stakeholder Requirements	12
3.2.1 Functional Requirements	12
3.2.2 Non-Functional Requirements	14
3.3 System Requirements	14
3.3.1 Must Have	15

3.3.2	Nice to Have	15
3.3.3	Will not Have	15
4	Global and Architectural Design	17
4.1	Global Design Choices	17
4.2	Key Design Pillars	17
4.3	Technology Used	18
4.3.1	Programming Language	18
4.3.2	Frameworks and Libraries	18
4.3.3	Architectural Design Choices	19
4.4	Diagrams	20
4.4.1	Code Structure	20
4.4.2	Front-End Code Structure	21
4.4.3	Back-End Code Structure	21
4.4.4	Class Diagram	22
4.4.5	State Diagram	25
4.4.6	Use Case Diagram	26
5	Interface, Functionality and Algorithm Design	29
5.1	Design Iterations	29
5.1.1	Design Phase One	29
5.1.2	Design Phase Two	30
5.1.3	Final Design	31
5.2	Design Choices	31

5.2.1	System Interface and Usability	31
5.2.2	File Upload	31
5.2.3	Manual Data Input	32
5.2.4	Preview Table	32
5.2.5	Submit and Refresh Buttons	32
5.2.6	Preview Table and Schedule Table Editing	33
5.2.7	Schedule Table Highlighting	33
5.2.8	File Downloading	34
5.2.9	Use of Database	34
5.2.10	Algorithm	34
6	Testing the System	37
6.1	Test Plan	37
6.1.1	Approach	37
6.1.2	Unit Testing	37
6.1.3	Usability Testing	37
6.1.4	Functionalities to be tested	38
6.2	Risk Analysis	38
6.2.1	Performance Risks	39
6.2.2	Usability Risks	40
6.2.3	Functionality Risks	40
6.3	Test Results	40
6.3.1	Unit Tests	40
6.3.2	Test for "test_test_route"	43

6.3.3	Test for "test_preview_generator_uploaded_data" . . .	44
6.3.4	Test for "test_preview_generator_uploaded_file" . . .	45
6.3.5	Test Result	46
6.3.6	Usability Tests	46
7	Evaluation	49
7.1	Design and Development Process	49
7.2	Team Evaluation	50
7.3	Final Results	54
8	User Guide	55
8.1	Instalment of Web Application	55
8.2	Run Tests	57
8.3	Use of Application	57
8.3.1	Manual Data Entry	58
8.3.2	File Upload for Data Entry	58
8.3.3	Editing Data	58
8.3.4	Highlighting	58
8.3.5	Downloading and Re-Generating the Schedule	59
8.3.6	Error message	60

1 Domain Analysis

In this section of the report, the context of the problem will be described and evaluated. Followingly, the current solution to the problem and the team's approach to improving the solution will be described.

1.1 Context

Large groups of students in various modules often work together on their projects as teams. These teams may perform multiple peer reviews throughout the course of their module. Schedules must be created for this purpose. The usual way of doing this is to do it by hand or via Excel. This is slow and error-prone. It is also increasingly difficult to do as the number of groups goes up.

1.2 Problem Description

In the master course Programming Principles, Patterns, and Processes at the University of Twente, students work on projects in teams. As a part of the course, teams are supposed to perform peer reviews. The pairing process needs scheduling which includes team names, review place, and review time.

The peering process is as follows. There are three review sessions. In the first two review sessions, two (twin) teams, which are paired by the teacher, review each other. In the last review session, these teams again review each other, but every team is also reviewed by a second review team.

The problem is to generate schedules for the last review session. In the schedule, every team should present once, and attend the review sessions of two other teams. Every team should only be in one place at one time. Additionally, the second review team should not be reviewed by the presenting or twin teams.

1.3 Our Solution

The goal is to create a website to use for this purpose. The product should allow the user to automate, speed up, and process, with no errors in schedule creation. The flexibility provided by the web interface will also allow the team to implement potential extra functions for the teacher to use.

2 System proposal

This section of the report is dedicated to the requirement specification phase of the project. In this phase, the team decided on requirements and proposed prototypes for further feedback.

2.1 Requirements Proposal

The initial idea of our supervisor on the product was an application or web application for automating the scheduling process of the peer reviews for the master course Programming Principles, Patterns, and Processes at the University of Twente. The expected product was going to be used mainly by the teachers. After the first meeting, the main focus of the team was to decide on the correct requirements according to the product goals presented by the supervisor. Firstly, the matching algorithm will take presenting teams and pair teams paired by the teacher, available rooms, and timeslots as input. Secondly, a third team that will review the presenting team will not be reviewed by either the presenting team or the review team. Finally, if a team has to attend two consecutive peer reviews, then these two peer reviews have to be scheduled in the same room to minimize disruptions. With these requirements in focus, we were able to get the algorithm to work correctly.

Besides the above-mentioned requirements which are necessary for the algorithm to work, we also had more sub-discussions to further specify requirements. The division of the type of requirements was initially made by dividing the requirements into functional or non-functional requirements. After presenting the requirements proposal to our supervisor, we decided to divide the requirements into stakeholder requirements and system requirements. The stakeholder requirements are the user stories from a teacher's perspective that are divided into functional and non-functional requirements. The system requirements are the finalized version of our initial requirements. The ultimate alteration to the system requirements was the change of the structure to must-have, nice to have, and will not have structure from functional/non-functional structure.

Furthermore, additional requirements were added for the design of the product during the meetings. However, no major changes have been made

to the requirements besides their structure of them.

2.2 Prototype Proposal

As there was no initial mock-up prototype of the product, the team is required to create mock-up prototypes. Our supervisor wanted the design to be minimalistic and easily understandable. The team initially had three different designs including hand-drawn sketches and mock-ups created with Figma. After presenting the different designs, we decided to use the current design by critically thinking about the feedback received from the supervisor. Since our focus is creating a minimalistic design that is usable, understandable, and efficient, multiple things were removed from the prototype.

Firstly, a login page is not necessary for the product as we agreed with the supervisor. Thus, the login page design was removed from the prototype. Also, the prototype has a one-page design with a couple of buttons which all are of use for different actions. The decision to have a one-page design has helped the design to be greatly simplified. Additionally, the initially created button that helps to regenerate the schedule was removed as the main generate button also has this functionality.

2.3 Project Presentation

During the development process, the project teams gave three presentations. These presentations were to inform the other teams of the current state of the project and receive feedback from other groups.

The first presentation was about the project concept. We described the problem and the proposed solution. We presented potential design heuristics and a time plan. We received useful feedback in this presentation session. The questions helped us decide on the requirements in better detail.

The second presentation was about presenting our first iteration of work, and the UI design. We had a lot of material to present and our designs were well received. We presented multiple options and asked for other groups' opinions.

For the third presentation, our product was almost finalized. We talked about testing and explained the system in detail with our UML diagrams, and showed a demo of the system. We received useful feedback on our UML diagrams and the first draft of our report.

In the final week of the module, we did a final presentation to the chair of the FMT group, the group to which our supervisor belongs. We showed our demo to the teachers, we also talked about design, development phase, testing, and future work.

2.4 Results of Meetings

Our initial meeting with the supervisor was scheduled via an email discussion, where it was decided that meetings could be conducted either online or offline based on the availability of both parties. Offline meetings were held at the Zilverling building where our supervisor's office is located. Online meetings were held via Teams. The scheduling of meetings was updated each week as the team and the supervisor mutually agreed on availability. The specifics of the next meeting, including its date, time, and format, were decided at the end of each meeting.

During meetings, we started by presenting our progress on the product. Thus, we were able to get the necessary feedback from the start of the meetings. Then, we asked our questions to make sure we are on the right track. The meetings during the planning and designing phases were the ultimate guideline for the upcoming meetings. Later on, the meetings were to fix, test and improve the product. Overall, each meeting had its usefulness and contributed to the final product.

2.5 Time Planning

The Gantt chart below is the planned timeline for the development process of the Peer Review Scheduler, created on the web app Click-Up.

In the first week, the whole process was divided into 4 main phases: planning, designing, development, and testing.

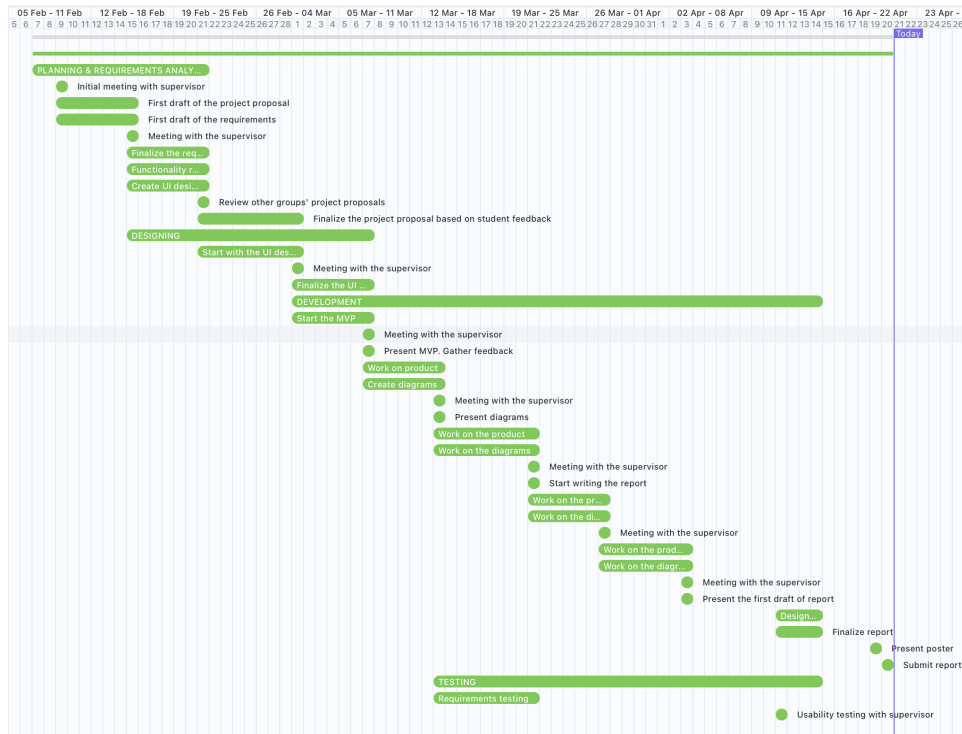


Figure 1: Planning

2.5.1 Planning

The goal of the planning process is to discuss with the supervisor and understand the project thoroughly in order to start with the initial design. The phase consists of two weeks. During those weeks, the team aimed to:

- Meet with the supervisor to get feedback
- Decide on the requirements
- Peer review
- Finalize the project proposal

2.5.2 Designing

The design phase is to design the UI. The phase consists of three weeks. Our aim was to:

- Start the design
- Meet with the supervisor to get feedback
- Finish the design

2.5.3 Development

In this phase, coding and reporting procedures started. The phase consists of a month. Our aim was to:

- Start the MVP
- Create diagrams
- Start the report
- Finish the product
- Finalize the report

2.5.4 Testing

In this phase, we started to test the code. The phase consists of around three weeks. Our aim was to:

- Make sure the schedule meets all the requirements
- Erase bugs from both frontend and backend

3 Requirements Analysis

3.1 Approach

During this phase, we utilized agile methodology to iteratively implement, test and improve our requirements.

We have created a product backlog which is a prioritized list of requirements, features, and user stories of the project. We then introduced these requirements to the client and based on the feedback we kept on improving them. This phase continued until both the team and the client were satisfied. While writing down the requirements we followed the SMART strategy which stands for specific, measurable, achievable, relevant, and time-bounded requirements.

Further changes were indeed made throughout the development phase as per the liking of the client/supervisor.

3.2 Stakeholder Requirements

In this section, the project requirements will be listed as user stories. Below are listed the Functional and non-functional requirements. The requirements are organized by priority from highest to lowest.

3.2.1 Functional Requirements

1. As a teacher, I want to be able to input two teams who reviewed each other in the first session.
2. As a teacher, I want to be able to type the two teams of the first session as input by hand.
3. As a teacher, I want to be able to upload a file containing teams from the first session as input.
4. As a teacher, I want the program to have a default setting for the variables, and the option to change each manually.

5. As a teacher, I want to be able to click a button to start the scheduling process.
6. As a teacher, I want the program to make sure that no team needs to do more than one review session for a single time slot, and no team should have two consecutive sessions in different rooms.
7. As a teacher, I want the program to make sure that any 2nd review team should not be reviewed by the presenting team, or the twin team that is attending the same presentation.
8. As a teacher, I want the schedule to be flexible regarding the number of teams. The teacher would make sure there are only an even number of teams.
9. As a teacher, I want the schedule to be flexible regarding the number of rooms.
10. As a teacher, I want the application to produce a schedule, including which group should review another specific group in what room and at which time.
11. As a teacher, I want the generated schedule to be in a table format.
12. As a teacher, I want the tables to be editable.

This requirement was added during the design process. Since the supervisor thinks it would be helpful to modify the schedule.

13. As a teacher, I want to be able to download the generated schedule.
14. As a teacher, I want the downloaded schedule to be in CSV format.
15. As a teacher, I want to be able to highlight the same teams, rooms, and time slots.

During the design process, the teacher realized that it would be convenient for her to check the correctness of the schedule when she can easily see the rooms and the time slots for one team, and highlighting the same team in each row could be effective.

16. As a teacher, I want the system to give clear and informative error messages when errors occur or when the system cannot perform a requested task

17. As a teacher, I want to be able to press a random button and have the program regenerate the schedule.
18. As a teacher, I want to be able to highlight the same teams, rooms, and time slots.
19. As a teacher, I want the submit button and regenerate button to be combined as one button.

This requirement is added during the testing phase. At first, we have two buttons separated in the design, but the supervisor thinks it would be more reasonable to combine two buttons because they have the same use.

3.2.2 Non-Functional Requirements

1. As a teacher, I want the user interface to be simple and user-friendly.
2. The schedule should be produced in less than 10 seconds.
3. The system should generate a schedule that ensures no team will need to attend two consecutive peer reviews in two rooms.
4. The system should generate a schedule that ensures that an additional review team will not be reviewed by the presenting team and pair team.

3.3 System Requirements

In this section, the final version of the requirements will be presented. The team decided to use a must-have, nice-to-have, and will-not-have structure. The structure is adapted for the purpose of clarity.

“Must have” requirements will include the ones that are strictly expected by our client. The requirements the client liked but specified as not necessary will be listed below the “Nice to Have” section. The “Will not Have” section will have the requirements that are misunderstood by the team and/or not wanted by the client after the first two interviews.

3.3.1 Must Have

1. The application must have a default setting for the first session reviews.
2. The application must provide a correct schedule for the last session.
3. The application must provide editing of the given schedule.
4. The application must give the exporting option to the user.
5. The application must let the user choose the number of teams.
6. The application must let the user choose the number and location of rooms.
7. The application must alert the user in case of an anomaly.
8. The application must be user-friendly and easy to use.

3.3.2 Nice to Have

1. The application could be integrated with Canvas so that it can be easily accessed by the users.
2. The application should be optimized for speed and be able to produce the schedule in less than 10 seconds.
3. The schedule produced by the application should be in an Excel-like table format, which is easy to read and understand.
4. The application should have fewer buttons and more scroll-type inputs to make it more user-friendly and intuitive.
5. The application should be able to let the user highlight the information of the same type to make it easier to review and revise data.

3.3.3 Will not Have

1. The application will not generate a schedule having an additional review team reviewing a presenting team and pair the team with which it has been scheduled a peer review.

-
2. The application will not generate a schedule having a team attending two consecutive peer reviews in two different rooms.

4 Global and Architectural Design

This section of the report will discuss the global design choices, key design pillars, and other design parts of the project such as technologies that have been used and diagrams made to illustrate the application to the client.

4.1 Global Design Choices

The peer review scheduler system aims to automate the process of peer review meeting scheduling and reduce the labor-intensive work procedures.

This system should be implemented in the form of a web application so that it can be accessed on any OS environment without impairing user experience.

A user-friendly interface is essential to any web application. The user interface of this system should be intuitive and easy to use. The interface should provide the user with clear instructions on how to generate and regenerate peer review schedules, modify preview table data and download generated peer review schedules. The interface should also provide clear feedback to users when something goes wrong and actionable guidance on how to correct the error. The user interface should be responsive by using modern frontend technologies such as AngularJs, restful service such as Flask.

Testing is a crucial part of software development. This system will be tested extensively and thoroughly so that the system can work as expected. The testing will cover all aspects of the web application, including testing of scheduling algorithms, data validation, and API functionalities. The testing will be done by an automated testing framework such as unittest or pytest.

4.2 Key Design Pillars

The designing of the system aims to improve usability and enhance user experience. In order to develop a system that meets the requirements and expectations of users, users are closely involved in the design and development process. During the project, the client was provided with design

concepts, prototypes, and diagrams, and the client was interviewed in order to receive fruitful feedback. Designs are iterated and revised based on feedback to create more effective and impactful design solutions. The design process and potential outcomes are openly communicated with the client in order to strengthen the relationship with the client. Therefore, user involvement is the key design pillar carried out throughout the entire design process, from ideation to implementation, to create designs that are intuitively usable.

4.3 Technology Used

In this section, we will explore the used technologies throughout the development phase.

4.3.1 Programming Language

Python is the primary programming language in this project. There are several reasons for this choice. Firstly, Python is a programming language that is easy to learn and use. Python has clear and concise syntax, and it has a large community of developers who share knowledge and offer help. Secondly, Python has a large library of frameworks to make web development easier and more efficient, for example, flask is the backend framework that is written in Python. Besides, Python also allows for faster development times, making it more efficient and time-saving to build web applications. Last but not least, Python is the programming language that everyone in the team has experience with.

4.3.2 Frameworks and Libraries

The peer review scheduling algorithm is essential to this project. Z3 Theorem Prover package is chosen as the library dedicated to handling schedule generation. Z3 has been proven to be very efficient and powerful for solving the boolean satisfiability problem. It can handle large and complex problems with high efficiency. Z3 also works well with several programming languages, especially Python. An API has been developed to support

some programming languages, including C++, .NET, and Python. It's quite convenient to integrate Z3 into existing software projects using the Z3 API.

Several other libraries for solving boolean satisfiability problems are also taken into consideration in this project. For example, Optaplanner developed in Java has been regarded as one of the options. While Optaplanner is also efficient and powerful, it is verbose and has a steep learning curve compared to Z3, which is less suitable for a lightweight project. And it is not possible to integrate Optaplanner with Python and Flask, making it a less optimal choice.

Flask is a web framework written in Python. Flask has been chosen as the backend framework for this project. Flask is a reputable framework for being lightweight and suitable for smaller projects compared to Django, which requires extensive knowledge and has a steep learning curve. Flask is adopted by many leading companies, such as Netflix, Reddit, Uber and Airbnb. With Flask, it is quick and convenient to set up a server development environment, making Flask an optimal option for Python-based university projects with smaller codebases. Scalability and modularity are essential features that Flask supports.

The front-end framework that has been chosen in this project is AngularJS. AngularJS is a modern front-end framework characterized by many excellent features, such as two-way data binding, and modular and MVC architecture. The two-way data binding features make it easy to manage the data within the application. The data can be synchronized between the model and the view automatically. The feature of modularity and MVC architecture of AngularJS significantly reduces the verbose work of code maintenance, since from the beginning of the development, functionalities and interfaces are divided into different components and developed in a modular way. Using AngularJS also enhances the characteristics of scalability because new components can be added to existing projects without causing conflicts.

4.3.3 Architectural Design Choices

The team initially started to use a functional programming approach as it is an easier way of coding thanks to Python. As the application improved, the downsides of the functional programming started to show up when creating multiple schedules for multiple teams, time slots, and rooms. We discussed

this matter within the team and with the supervisor and therefore ended up deciding to switch to the Object-Oriented-Programming (OOP) approach. By doing so, it was easier to create schedules as everything shifted from using lists to objects. Simply put, to add one more team to the table, it was only needed to refer to the class carrying that responsibility and using the methods within to create one more object.

Overall, the team started with functional programming and then migrated to Object-Oriented-Programming due to the above-discussed downsides of functional programming and their easy fixes with the introduction of OOP.

4.4 Diagrams

4.4.1 Code Structure

This section illustrates the code structure, including frontend and backend, of the project. Among the folders shown on Figure 2, folder “designProject” contains frontend code. And folder “static” contains the necessary elements including images, html file to build frontend interface. Folder “node_modules” contains built-in angular libraries. Folder “services” contains backend code. Whereas folder “algorithm” contains code for the algorithm that handles schedule generation, and folder “test” contains the test files necessary for testing.

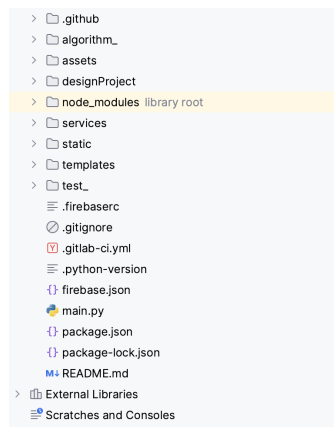


Figure 2: Code Structure

4.4.2 Front-End Code Structure

Under folder "designProject/src/app" lies the frontend code implementation as illustrated in Figure 3. The folder "schedulerHandler" contains files of all of the functionalities necessary for generating a schedule. The folder "fileDownloadService" contains files of the file downloading functionality for the schedule.

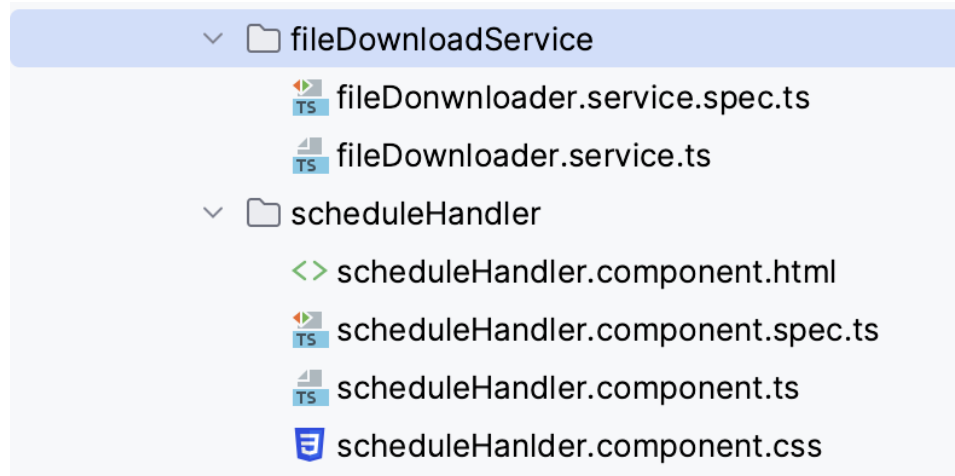


Figure 3: Front End Code Structure

4.4.3 Back-End Code Structure

Under folder "services/previewGenerator" lies the backend code implementation of previewGenerator. It contains three classes "previewDataTable", "previewGeneratorOfUploadedData" and "previewGeneratorOfUploadedFile", which correspond to the classes in class diagram respectively. Under folder "services/scheduleRetriever" lies the backend code implementation of schedule retriever. It contains four classes "uploadedFile", "manuallyFilledData", "filledDataScheduleRetriever" and "uploadFileScheduleRetriever", which correspond to the classes in class diagram respectively.

Under folder "algorithm_" lies the code implementation of algorithm. It contains two classes "Algorithm" and "Schedule", which correspond to the classes in the class diagram respectively.

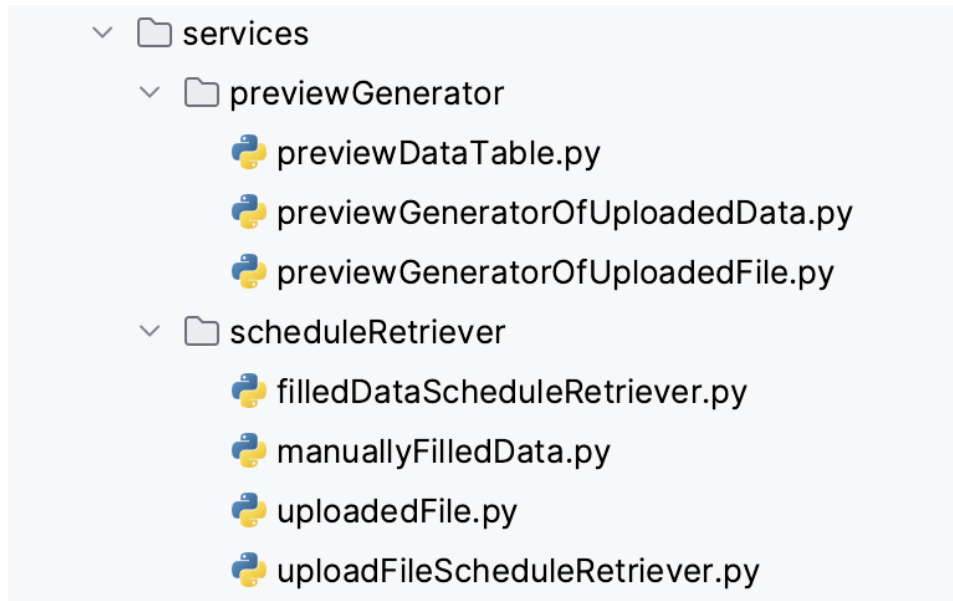


Figure 4: Back End Code Structure

Under folder "test_" lies the code implementation of unit tests for the algorithm and flask application.

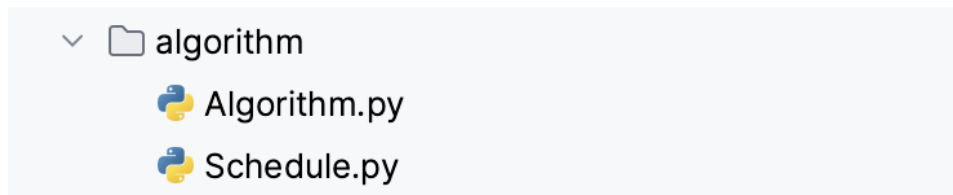


Figure 5: Algorithm Code Structure

4.4.4 Class Diagram

In this class diagram shown in Figure 6, 10 classes are created to model the structure of the peer review scheduler system. These classes can be separated into frontend classes and backend classes.

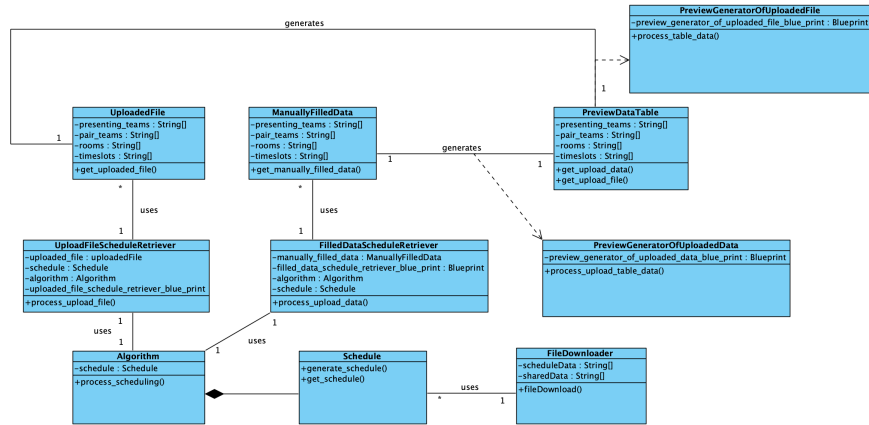


Figure 6: Class Diagram

Back-End Classes

The “uploadedFile” class represents the file that the user uploads. This class is characterized by arrays of presenting_teams, pair_teams, rooms and timeslots. For example, a valid uploaded file may look like this:

Presenting_team	Pair_team	Locations	Timeslots
Team1	Team2	Horst915	11:00-11:15
Team2	Team1	Horst916	11:20-11:35
Team3	Team4		11:40-11:55
Team4	Team3		12:00-12:15
Team5	Team6		
Team6	Team5		
Team7	Team8		
Team8	Team7		

This class is also characterized by a function "get_uploaded_file()" which is used to get the uploaded file from other classes.

The “ManuallyFilledData” class represents the data that the users manually input. This class is characterized by attributes of arrays of presentint_teams, pair_teams, rooms and timeslots. For example, the user can input the numbers 8, 2, and 4 for teams, locations, and time slots respectively, which will result in an array of 8 presenting teams and pair teams, an array of 2 rooms and an array of 4 time slots. This class is also featured by a function of "get_manually_filled_data()" which is used to get filled data from other classes.

The “PreviewGeneratorOfUploadedFile” class represents the service that generates the preview table from the uploaded file. This class is characterized by a function “process_table_data()” which utilizes the “UploadedFile” class to generate the preview table. And this class is also featured by an attribute of "preview_generator_of_uploaded_file_blue_print" which is used to used to define routes and methods of flask app.

The “PreviewGeneratorOfUploadedData” class represents the service that generates the preview table from the uploaded data. This class is characterized by a function “process_upload_table_data()” which utilizes the “ManuallyFilledData” class to generate the preview table. This class is also featured by an attribute of “preview_generator_of_uploaded_data_blue_print” which is used to define routes and methods of flask app.

The “UploadedFileScheduleRetriever” class represents the service that generates the schedule for uploaded files, which refers to the “uploadedFile” class. This class is featured by an attribute of uploaded_file referring to “uploadedFile” class, an attribute of “schedule” referring to “Schedule” class, and an attribute of “algorithm” referring to “Algorithm” class and an attribute of "uploaded_file_schedule_retriever_blue_print". This class is also characterized by a function “process_upload_file()”, which uses the uploaded file and algorithm to generate a schedule.

The “FilledDataScheduleRetriever” class represents the service that generates the schedule for manually filled data, which refers to the “ManuallyFilledData” class. This class is featured by an attribute of "manually_filled_data" referring to “ManuallyFilledData” class, an attribute of “schedule” referring to “Schedule” class, an attribute of “algorithm” referring to “Algorithm” class and an attribute of "filled_data_schedule_retriever_blue_print". This class is also characterized by a function “process_filled_data()” which uses manually filled data and an algorithm to generate a schedule.

The "Schedule" class represents the current schedule that is generated by the algorithm. This class is generated by "Algorithm" class and used by both "UploadFileScheduleRetriever" and "FilledDataScheduleRetriever" to send the schedule as requested. The "Schedule" class is featured by two functions "generate_schedule()" and "get_schedule()".

The "Algorithm" class represents the algorithm used to generate the schedule. This class is featured by an attribute "schedule" referring to the "Schedule" class. This class is also characterized by a function "process_scheduling()", which utilizes z3 solver to generate correct schedules.

Front-End Classes

The "FileDownloader" class represents the service to handle the user request of downloading schedule to local machine. This class is characterized by an attribute of "scheduleData" which represents the schedule data sent back from the backend and an attribute of "sharedData". This class is also featured by the function "fileDownload()" to handle the file downloading request.

4.4.5 State Diagram

The state machine diagram represents the behavior of the peer review scheduling system. The system starts at an empty state, represented by a filled circle. Before it receives any instructions from a user, it remains in an "Idle" state. In the "Idle" state, the user can either upload a file or fill in data manually, which will cause a transition to the state of "preview table automatically generated". If the user clicks the submit button to submit the input file or data, the system will transition to the state of "schedule generated". In the state of "schedule generated", the user can either choose to regenerate the schedule, which will cause a transition back to the state of "schedule generated", or request for download, which will lead to a transition to the state of "downloading completed", or chooses to do nothing, then the system will transition to the state of "end", which is represented by a partially filled circle.

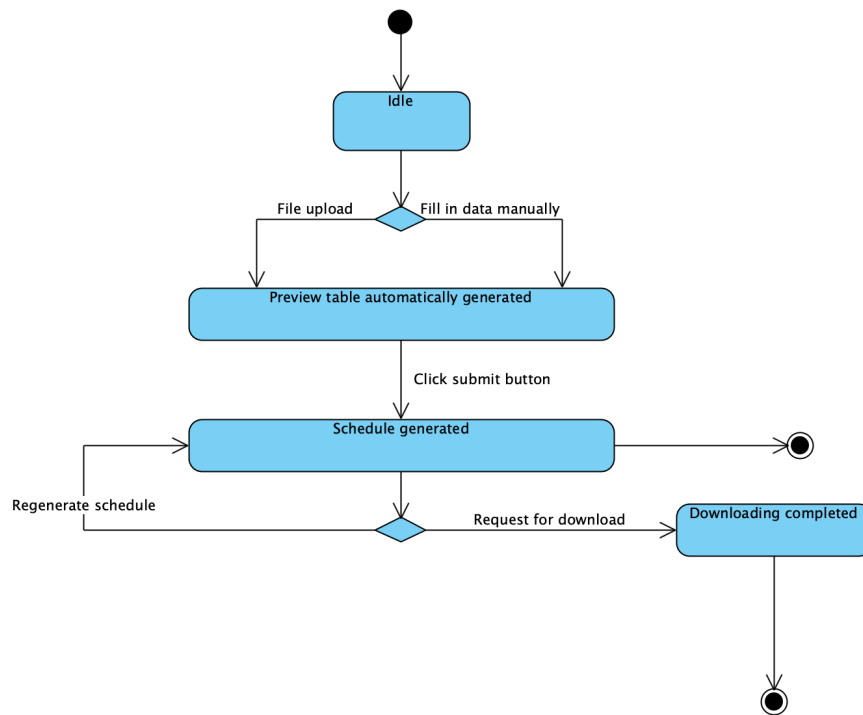


Figure 7: State Diagram

4.4.6 Use Case Diagram

This use case diagram illustrates the functionality of the peer review scheduler system. The system has two main actors, “User” and “Peer review scheduler service”. The user is who uses the system. The peer review scheduler service is responsible for handling file uploading, downloading, preview table generation and schedule generation.

The “User” has several use cases, including “upload file”, “fill in data manually”, “request to regenerate schedule” and “request to download schedule”. The “Peer review scheduler service” has several use cases as well, including “generate preview table”, “generate schedule” and “generate downloading file”.

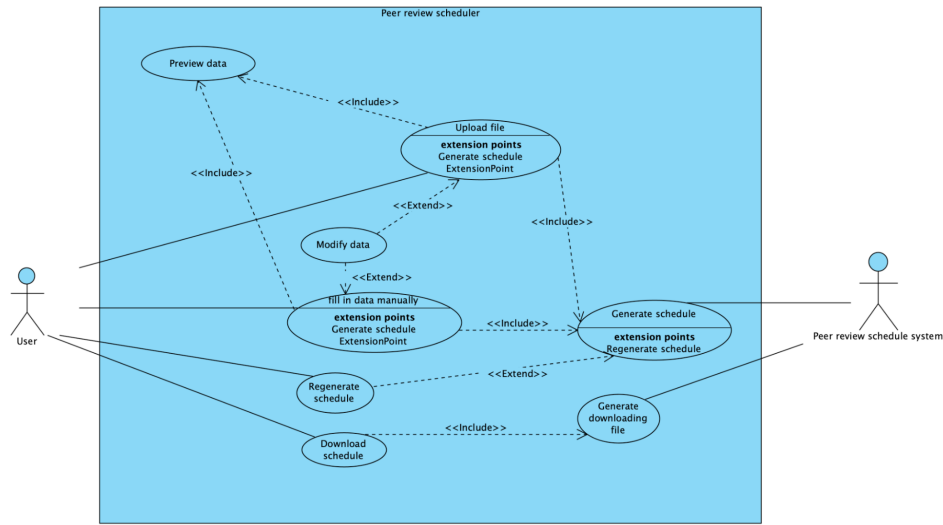


Figure 8: Use Case Diagram

There are several relationships between the actors and use cases. The “User” is associated with all of the user-related use cases, while the “Peer review schedule service” is associated with all of the “Peer review scheduler service” related use cases.

What’s more, the “fill in data manually” use case includes the “generate preview table” use case, as after the data is filled, the system will automatically generate a preview table on the interface. The “fill in data manually” use case also includes the “generate schedule” use case, as the system must generate a schedule once the data is submitted by the user.

The “upload file” use case includes the “generate preview table” use case, as after the file is uploaded, the system will also automatically generate a preview table on the interface. The “upload file” use case also included the “generate schedule” use case, as the system must generate a schedule once the uploaded file is submitted by the user.

The “modify data” use case extends both “upload file” and “fill in data manually” use cases. Because modifying data is an optional action for the user to do after they upload the file or fill in the data manually.

The “request to regenerate schedule” use case includes “generate schedule” use case, as the system must generate a schedule if the user requests to regenerate a schedule.

The “request to download schedule” use case includes the “generate downloading file” use case, as the system must generate downloading files if the user desires to download the schedule.

5 Interface, Functionality and Algorithm Design

The purpose of this section is to provide a clear understanding of the design choices. It involves the translation of requirements gathered during the analysis and design phases into a blueprint for the actual implementation of the web application. This section provides a detailed description of the user interface and functionality design.

5.1 Design Iterations

We have been through different design phases to reach the final design. In each design phase, we integrate the feedback from our supervisor into the design. And we also tested the design with fellow students to gain extra feedback. By presenting, receiving feedback, making changes and presenting again, we are able to create a design that met the requirements and expectations of our supervisor.

5.1.1 Design Phase One

We presented the first initial design in Figure 12: 1 based on the requirements and discussions. The concept includes file uploading, manual data input, a preview table and a final schedule. There is another initial design in Figure 13:2 that we came up with. This design was made by using figma. It serves the same functionalities but contains several web pages.

Presenter	Team	2nd Review	Place	Time
Team 1	Team 2	Team 12	RT 50208	11:45 - 11:50
Team 3	Team 4			
...	...			

Presenter	Team	This page will be auto update	Place	Time
Team 1	Team 2		RT 50208	11:45 - 11:50
Team 3	Team 4			
...	...			

Figure 9: 1

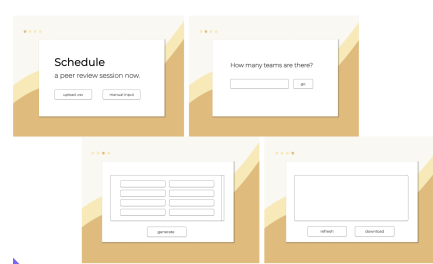


Figure 10: 2

Figure 11: Design Phase 1.1

This is the third initial design presented to our supervisor. This design was also made on figma. It uses a scenery picture as part of the design to make it more visually appealing.

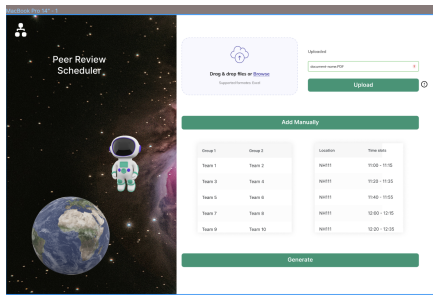


Figure 12: 1

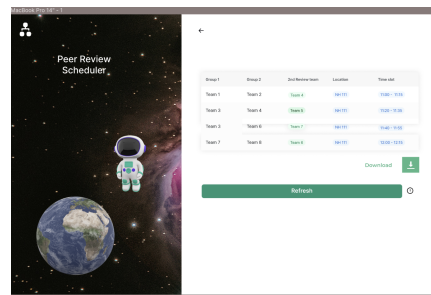


Figure 13: 2

Figure 14: Design Phase 1.2

Fruitful feedback was given by the supervisor on our initial design drafts. The table editing function in the first initial design, the function logic of the second design, and the UI design of the third initial design were liked. The supervisor would also prefer the application to be a single-page application to make it simple to use.

5.1.2 Design Phase Two

During this phase, we presented a design that is based on the requirements and the feedback from the initial designs. We improved the design by combining the advantages of each initial design of design phase one, such as the scenery picture, the table editing function, the design of the buttons,

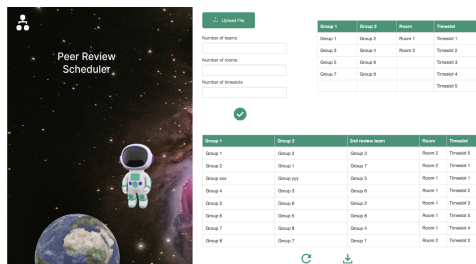


Figure 15: Design phase 2

the layout of the interface and the idea of a single page application.

5.1.3 Final Design

During this phase, we presented a final design that is liked by the supervisor. We improved the design based on the feedback from the supervisor. For example, a highlight function to mark the same kind of data would be helpful, and the submit and the regenerate button can be combined since they serve the same purpose.

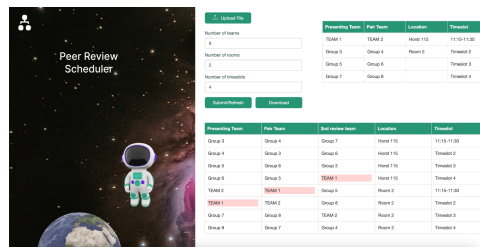


Figure 16: Final design

5.2 Design Choices

5.2.1 System Interface and Usability

The first choice design concerns the system's look and usability. For this system, it is important that the user interface is intuitive and easy to use. During the design of the system interface, the client's feedback was constantly taken into consideration and changes were made accordingly, so that the final design meets the client's requirements.

5.2.2 File Upload

The second design choice concerns the ability to upload a file in the format of Excel. It has been decided that the user is allowed to upload a file so that the user can be spared the efforts of manually modifying the data on

the preview table if the user already has a file of real data. When a file is uploaded, the file name and format will be shown on the interface. The user is also given the option to clear the file that has been uploaded by clicking on the “clear file” button.

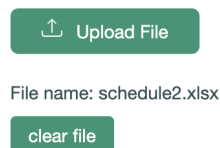


Figure 17: File Upload

5.2.3 Manual Data Input

The third design choice concerns the ability to manually input the data. The user will find this feature useful because this enables the user to fill in the placeholders in the form of numbers instead of preparing a file, which is more convenient. The input fields are sanitized such that only positive numbers are allowed and on invalid inputs, error messages will be given.

5.2.4 Preview Table

The fourth design choice concerns the feature of a preview table displaying data extracted from uploaded file or manually input data in an intuitive way. The preview table will automatically show up on the interface once the actions of uploading file or manually filling data is done. This preview table contains four columns, from left to right, presenting_team, pair_team, rooms, and time slots respectively. This way, the user is given a visualized way to view the data.

5.2.5 Submit and Refresh Buttons

The fifth design choice concerns the feature of combining the submit button with the refresh button. In the initial design, the user was given the option to use both the submit button and the refresh button. The submit button is

↑ Upload File

Number of teams

Number of rooms

Number of timeslots

Submit/Refresh

Presenting Team	Pair Team	Location	Timeslot
Group 1	Group 2	Room 1	Timeslot 1
Group 3	Group 4	Room 2	Timeslot 2
Group 5	Group 6		Timeslot 3
Group 7	Group 8		Timeslot 4

Figure 18: Preview Table

used to submit the first request of the retrieving schedule. And the refresh button is used to submit the request of regenerating the schedule after the first submission. The options were discussed with the client who indicated that these two buttons have duplicate functions and should be combined as one button, which is more in line with the principle of a user-friendly and intuitive interface.

5.2.6 Preview Table and Schedule Table Editing

The sixth design choice concerns the feature of enabling users to edit the preview table and schedule table. The first option is to enable one-way editing. For example, when the user edits the `presenting_team` name in the preview table, the same team names in the schedule table will be modified synchronously. But it doesn't work the other way around. The second option is to enable two-way editing, which means if the user modifies data on the schedule data, the same data on the preview table will be modified synchronously as well.

5.2.7 Schedule Table Highlighting

The seventh design choice concerns the feature of highlighting information of the same kind. It has been discussed with the client who indicated that highlighting information of the same kind in the schedule table, for example, same team names, same rooms, and time slots would be effective

for reviewing and revising the schedule.

5.2.8 File Downloading

The eighth design choice concerns the feature of downloading the generated file. The first option that was discussed is to enable the user to download history files from a specific date. The second option is to only provide the user with the ability to download the most recently generated file. These options were discussed with the client, and the client indicated that there is no need to download history files as they can be backed up locally on a laptop. Therefore the second option was adopted during the design.

5.2.9 Use of Database

The last design concerns the use of a database. Multiple options have been discussed with the client regarding the use of a database. The first to implement a user management system, including functionalities of signing up, logging in, and logging out. The second one involves storing history schedules according to date. As indicated by the client, the system is for personal use for the time being, and in the future, a following project is responsible for integrating the system into Canvas, and the user management system will be used, therefore it would be redundant to implement a user management system in this project. The client also indicated a preference to store history files locally instead of in a database. Hence it has been decided that it is not necessary to use an information management system such as a database.

5.2.10 Algorithm

This design choice concerns the correctness of the generated schedule. The detailed design is as follows:

The algorithm is implemented as a class, which takes a list including names of teams, rooms, and time slots as input. In the initialization, a schedule object would be created. The `process_scheduling` method is used to generate the schedule. First a dictionary that maps teams' names, rooms' names, and

time slots' names to their corresponding index would be created according to the input. We use the index rather than the name itself because the index is easier for adding the requirements.

```
# Input format: [[("G1", "G2"), ("G3", "G4"), ("G5", "G6"), ("G7", "G8")], ["R1", "R2"], ["T1", "T2", "T3", "T4"]]
class Algorithm:

    def __init__(self):
        self.schedule = Schedule()

    # Generate and return the schedule.
    def process_scheduling(self, from_input):

        # Initial the list of names of the teams, rooms and timeslots.
        team_list = list(chain(*from_input[0]))
        rooms_list = from_input[1]
        slots_list = from_input[2]
        teams_num = len(team_list)
        rooms_num = len(rooms_list)
        slots_num = len(slots_list)

        # Map names into index (start from 1), then z3 will do the operation based on the index, for example
        # defaultdict(<class 'str'>, {1: 'Group1', 2: 'Group2', 3: 'Group3', 4: 'Group4', 5: 'Group5',
        # 6: 'Group6', 7: 'Group7', 8: 'Group8'})
        teams_map_to_index_dict = defaultdict(str)
        rooms_map_to_index_dict = defaultdict(str)
        slots_map_to_index_dict = defaultdict(str)
        for index, team in enumerate(team_list):
            teams_map_to_index_dict[index + 1] = team
        for index, room in enumerate(rooms_list):
            rooms_map_to_index_dict[index + 1] = room
        for index, slot in enumerate(slots_list):
            slots_map_to_index_dict[index + 1] = slot
```

Figure 19: Initialization of the algorithm

Then a board is created for the z3 solver to solve the scheduling problem, and it represents the information about the teams, rooms and time slots. This board has five columns, the first column is the presenting team and the second column is the pair team. These two columns are fixed according to the rule of the pair review.

The remaining columns represent additional teams, rooms and time slots respectively. The solver will attempt to fill these three columns.

Next, several constraints are added to the solver to ensure that the scheduling problem is solved correctly. For example, each name of the teams in the row of additional teams must be different, each team must appear in three different time slots, and the corresponding presenting team and pair team for a team shouldn't be the additional team for that team. The correctness

```

# Create a z3 solver.
solver = Solver()

# Create a board for solver to solve.
board = [[Int(f"m_{i}_{j}") for j in range(3)] + [Int(f"t_{i}"), Int(f"r_{i}")] for i in range(teams_num)]

# Condition for initialize the presenting and the pair team.
# [[1, 2], [2, 1], [3, 4], [4, 3], [5, 6], [6, 5], [7, 8], [8, 7]]
initialize = []
for i in range(1, teams_num + 1, 2):
    initialize.append([i, i + 1])
    initialize.append([i + 1, i])
initialize_con = [board[i][j] == initialize[i][j] for j in range(2) for i in range(teams_num)]

```

Figure 20: Initialize solver, board and constraint for presenting and pair teams

of the algorithm is ensured by adding all the requirements as constraints and adding to the z3 solver.

```

# condition: for an additional team, corresponding presenting and pair team should not be the additional team
# when it is presenting
# loop through additional team
for i in range(teams_num):
    # loop through presenting team
    for j in range(teams_num):
        solver.add(
            If(board[i][2] == board[j][0], And(board[j][2] != board[i][0], board[j][2] != board[i][1]), True))

```

Figure 21: One example of requirements

At the end we feed the board to the solver, the solver will return a result if there exists a solution. After changing back to the group name from the index, result would be ordered by the name of the rooms and the name of time slots, then an schedule object would be created using the result. This object would be returned by the function and send to the schedule retrievers.

```

# Return the schedule object.
self.schedule.generate_schedule(res)
return self.schedule
else:
    # Result not exist.
    self.schedule.generate_schedule("RESULT NOT EXISTED")
    return self.schedule

```

Figure 22: Return the schedule object or give ERROR message

6 Testing the System

The purpose of this section is to provide an overview of the system testing phase of the project and present the results of the testing activities. This section aims to provide a comprehensive understanding of the testing process and the quality of the application under test.

6.1 Test Plan

6.1.1 Approach

Our product consists of two parts: the front-end and the back-end. There are various popular test approaches for testing the back end, including unit testing, integration testing, and API testing. There are also several test approaches for testing the front end, including usability testing, accessibility testing and performance testing. Based on the structure and the requirements of our project, we chose to perform unit testing and usability testing, and the corresponding plans are shown below.

6.1.2 Unit Testing

We plan to do some unit testing for our back-end, including the functionality for Flask and the algorithm.

For the functionality of Flask, there is not that much to test so we only plan to implement one test case to test whether the front-end and the back-end are connected correctly.

To test the correctness of the Algorithm, we plan to implement several test cases based on the requirements, to make sure that the result is valid.

6.1.3 Usability Testing

In this section, the usability testing strategy is illustrated. Usability testing is essential to the software development process. This testing involves testing

the web application with a group of users to identify potential usability issues and areas of improvement. The goal of the usability testing is to ensure that the peer review scheduler system is easy and pleasing to use, efficient and smoothly functioning.

We plan to set up several interviews to let other people tell us what they think about our system. We also plan to do the manual testing ourselves. After we fixed most of the bugs and finished most of the functionality, we will run the selenium automation test to ensure that the system does everything as intended.

6.1.4 Functionalities to be tested

In this section, a list of functionalities to be tested is presented in the perspective of what the system does from a user. These functionalities cover all of the project requirements. As the requirements might change, these functionalities may also vary in the future. These functionalities are tested using the frontend testing framework Selenium. The level of risk indicates how important it is to test the functionality.

Functionality to be Tested	Level of Risk
Upload File	High
Clear Uploaded File	Medium
Manually Fill Data	High
Generate Preview Table	High
Preview Table Editing	High
Generate Schedule	High
Schedule Table Editing	High
Schedule Table Highlighting	High
File Download	High
Regenerate Schedule	High

Testing results are in the next page (Figure 25):

6.2 Risk Analysis

The purpose of this section is to identify potential risks associated with this project and provide strategies for mitigating these risks. Specifically, this

Project: DesignProject*

Command	Target	Value
1 ✓ open	/static/	
2 ✓ set window size	1512x874	
3 ✓ click	id=teams	
4 ✓ type	id=teams	8
5 ✓ type	id=rooms	2
6 ✓ type	id=timeslots	4
7 ✓ send keys	id=timeslots	\$(KEY_ENTER)
8 ✓ click	css=#dataTable tr:nth-child(1) > td:nth-child(1) > .editableTeamInput	
9 ✓ click	css=.col-sm-5 > .row	
10 ✓ type	css=#dataTable tr:nth-child(1) > td:nth-child(1) > .editableTeamInput	Team1
11 ✓ type	css=#dataTable tr:nth-child(1) > td:nth-child(2) > .editableTeamInput	Team2
12 ✓ type	css=#dataTable tr:nth-child(1) > td:nth-child(3) > .editableTeamInput	NH115
13 ✓ type	css=#dataTable tr:nth-child(1) > td:nth-child(4) > .editableTeamInput	11:15-11:30
14 ✓ click	css=.downloadButton	

Figure 23: 1

Running 'Test1'	Time
1. open on /static/ OK	18:50:05
2. setWindowSize on 1512x874 OK	18:50:06
3. click on id=teams OK	18:50:06
4. type on id=teams with value 8 OK	18:50:06
5. type on id=rooms with value 2 OK	18:50:06
6. type on id=timeslots with value 4 OK	18:50:06
7. sendKeys on id=timeslots with value \$(KEY_ENTER) OK	18:50:06
8. click on css=#dataTable tr:nth-child(1) > td:nth-child(1) > .editableTeamInput OK	18:50:06
9. click on css=.col-sm-5 > .row OK	18:50:07
10. type on css=#dataTable tr:nth-child(1) > td:nth-child(1) > .editableTeamInput with value Team1 OK	18:50:07
11. type on css=#dataTable tr:nth-child(1) > td:nth-child(2) > .editableTeamInput with value Team2 OK	18:50:07
12. type on css=#dataTable tr:nth-child(1) > td:nth-child(3) > .editableTeamInput with value NH115 OK	18:50:07
13. type on css=#dataTable tr:nth-child(1) > td:nth-child(4) > .editableTeamInput with value 11:15-11:30 OK	18:50:07
14. click on css=.downloadButton OK	18:50:08
'Test1' completed successfully	18:50:08

Figure 24: 2

Figure 25: Testing Results

report focuses on risks related to performance, usability, and functionality.

6.2.1 Performance Risks

One potential risk associated with this project is poor performance. such as slow load times, poor response times, or frequent crashes. In our project, the main risk can arise when it takes a long time for the algorithm to generate the schedule or/and when the algorithm generates a false schedule. To mitigate these risks, we will conduct thorough performance testing throughout the development process. This testing will involve testing scenarios under each constraint to identify potential performance issues before they affect users.

6.2.2 Usability Risks

Usability risks are another potential concern in this project. These risks can arise when the software is difficult to use, leading to user frustration and errors. To mitigate these risks, we will conduct extensive usability testing throughout the development process. This testing will involve testing the web application with real users to identify potential usability issues and gather feedback on how to improve the user experience.

6.2.3 Functionality Risks

Functionality risks can arise when the software does not perform as intended, resulting in bugs, errors, or data corruption. To mitigate these risks, we will conduct thorough testing of all software features and functions. This testing will involve identifying potential issues through unit testing for the algorithm and each of the functionalities, and fixing any issues before they affect users.

In conclusion, by identifying and mitigating performance, usability, and functionality risks, we can ensure that the final software product meets the needs of users and operates as intended.

6.3 Test Results

6.3.1 Unit Tests

Back-End

```
def test_test_route(self):
    response = self.client.get('/test')
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.mimetype, 'application/json')
    self.assertIn(b'Welcome to the test', response.data)
```

Figure 26: Back End Unit test

Algorithm

There are two classes in the unit test file: Result class and MyTestCase class.

Result class is used to generate the testing instance. First, it will generate random numbers of teams, rooms, and time slots, then the schedule would be generated and stored as a variable of the class. We've also implemented several getter methods for this class, making it easier for each of the test cases to gain the result of the algorithm and compare certain information.

```
# create the result object
class Result:

    def __init__(self):
        # next three lines is used to create random combinations
        self.team_count = (random.randint(4, 6)) * 2
        self.timeslot_count = random.randint(2, 5)
        self.room_count = random.randint(2, 5)
        self.from_input = []
        # names of teams
        self.tmp = []
        for i in range(self.team_count):
            if i % 2 == 1:
                self.tmp.append((i, i + 1))
        self.from_input.append(self.tmp)
        self.from_input.append([self.room_count])
        self.from_input.append([self.timeslot_count])
        algorithm = Algorithm(self.from_input)
        self.result = algorithm.process_scheduling()

    # return result in json format
    def get_result(self):
        return self.result
```

Figure 27: Result Class

Each test case is defined as a method within the MyTestCase class. We implemented nine test cases to ensure the algorithm would produce the correct schedule. These test cases represent the following constraints, which are:

1. The number of rows of the result matrix must be the same as the total number of teams, and the number of columns of the result matrix must equal to 5.
2. Each team must appear exactly three times in the schedule.
3. The number of different rooms in the schedule must be less than or equal to the input.

4. The number of different time slots in the schedule must be less than or equal to the input.
5. Each team must only appear once in a row in the result matrix.
6. Each team must only appear once in the column of additional teams.
7. Any additional team should not be reviewed by the presenting team, nor the pair team that is attending the same presentation.
8. Each team must only appear once in a timeslot.
9. No team should have two consecutive sessions in different rooms.

```

172     # test for the requirement:
173     # if one team appear in consecutive timeslots, then the corresponding room must be the same
    ± PinkGlove
174 ▶ def test_consecutive_timeslot_for_one_team_must_in_same_room(self):
175     result = Result()
176     result_check = result.get_result_in_matrix()
177     if result_check != "HELL NO":
178         for i in range(len(result_check)):
179             current_team = result_check[i][0]
180             for j in range(len(result_check)):
181                 if i != j:
182                     if (result_check[j][1] == current_team or result_check[j][2] == current_team) and abs(
183                         int(result_check[i][4].split(" ")[-1]) - int(result_check[j][4].split(" ")[-1])) == 1:
184                         self.assertEqual(result_check[i][3], result_check[j][3])
185     else:
186         self.assertEqual(True, True)

```

Figure 28: Algorithm Unit Test

Figure 29 is the coverage report of the unit test, showing how much of the code is covered in the Algorithms file during the test. Here we set the number of teams, rooms and time slots manually because we can only test whether the algorithm is correct or not when there's an answer.

```

(venv) (base) pinkglove@PinkGlove-2 test_ % coverage run -m Test_Algorithm
.....
Ran 9 tests in 15.291s

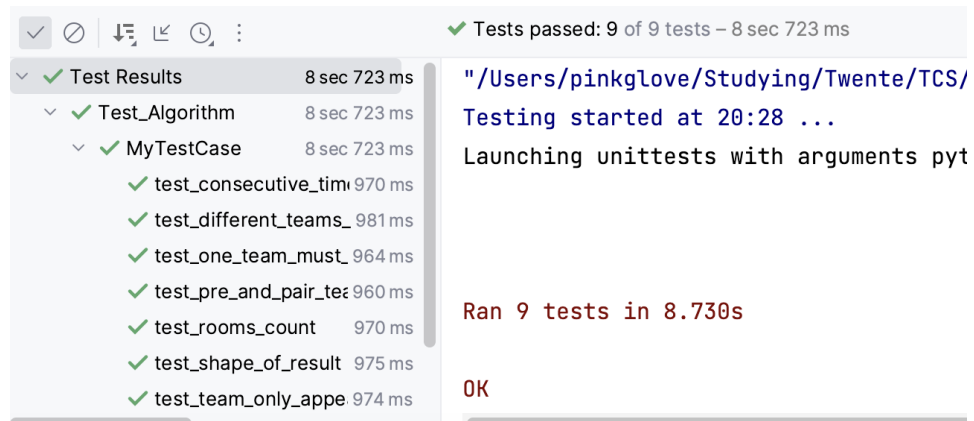
OK
(venv) (base) pinkglove@PinkGlove-2 test_ % coverage report -m
Name                                                                    Stmts  Miss  Cover   Missing
-----
/Users/pinkglove/Studying/Twente/TCS/Module 11/Project/Project/algorithm_/Algorithm.py  74      2   97%  136-137
/Users/pinkglove/Studying/Twente/TCS/Module 11/Project/Project/algorithm_/Schedule.py  11      1   91%    8
Test_Algorithm.py                                                         128     10   92%  45, 82, 96, 108, 120, 130, 141, 156, 171, 187
TOTAL                                                                      213     13   94%

```

Figure 29: Result Matrix

It shows that line 136-137 is not used, this is because these lines are used to return an error message when the given input does not have a corresponding schedule. The reason for 10 missing lines in Test_Algorithm is the same, these lines are used to handle the situation of error as well.

Overall, these unit tests ensure that our algorithm is correct, and the coverage test shows that all parts of the algorithm are tested.



```
✓ Tests passed: 9 of 9 tests – 8 sec 723 ms
Test Results 8 sec 723 ms
  Test_Algorithm 8 sec 723 ms
    MyTestCase 8 sec 723 ms
      ✓ test_consecutive_time 970 ms
      ✓ test_different_teams_ 981 ms
      ✓ test_one_team_must_ 964 ms
      ✓ test_pre_and_pair_teams_ 960 ms
      ✓ test_rooms_count 970 ms
      ✓ test_shape_of_result 975 ms
      ✓ test_team_only_appends 974 ms
"/Users/pinkglove/Studying/Twente/TCS/
Testing started at 20:28 ...
Launching unittests with arguments pyt
Ran 9 tests in 8.730s
OK
```

Figure 30: Test Results

Flask Testing

The Flask testing code is implemented in the file "test/Flask_Tests.py". In this test file, there are three test cases, they are test_test_route, test_preview_generator_uploaded_data, and test_preview_generator_uploaded_file respectively.

Since the algorithm testing already covers file "fillDataScheduleRetriever" and "uploadFileScheduleRetriever", they will not be tested in this section again.

6.3.2 Test for "test_test_route"

The test function sends a GET request to the '/test' route using the Flask test client and asserts that the response status code is 200 (which means that

the request was successful), the response mimetype is 'application/json', and the response data contains the welcome message "Welcome to the test".

```
def test_test_route(self):
    response = self.client.get('/test')
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.mimetype, 'application/json')
    self.assertIn(b'Welcome to the test', response.data)
```

Figure 31: Test Route

6.3.3 Test for "test_preview_generator_uploaded_data"

This test is meant to verify the functionality of a Flask application when handling a POST request to the /table-data endpoint with some uploaded data, which contains the number of teams, rooms, and timeslots.

The test sends this data to the Flask application using the client.post() method, which returns a response object. This response object contains a JSON-formatted data string. This data string is loaded into a Python object using the json.load() method, and then checked for certain properties.

The self.assertEqual() method is used to verify that the response status code is 200, and to compare the values of "presenting_team", "pair_team", "room" and "timeslot" in the JSON data with expected values defined in res_json.

```
def test_preview_generator_uploaded_data(self):
    uploadedData = MultiDict([('teams', '8'), ('rooms', '2'), ('timeslots', '4')])
    res_json = self.generate_result()
    response = self.client.post('/table-data', data = uploadedData)
    file_obj = io.StringIO(response.data.decode('utf-8'))
    json_data = json.load(file_obj)
    self.assertEqual(response.status_code, 200)
    for i in range(len(json_data['result'])):
        self.assertEqual(json_data['result'][i]['presenting_team'], res_json[i]['presenting_team'])
        self.assertEqual(json_data['result'][i]['pair_team'], res_json[i]['pair_team'])
        self.assertEqual(json_data['result'][i]['room'], res_json[i]['room'])
        self.assertEqual(json_data['result'][i]['timeslot'], res_json[i]['timeslot'])
```

Figure 32: Test Uploaded Data

6.3.4 Test for "test_preview_generator_uploaded_file"

The purpose of this test function is to test the functionality of a Flask endpoint which accepts an Excel file as input, processes it, and returns the result in JSON format.

The function starts by creating a test data in the form of a dictionary, which is then converted into a pandas dataframe. This dataframe is written to an Excel file using the `pd.ExcelWriter` module.

The Excel file is then read using the `BytesIO` module, which creates a file-like object from a bytes buffer. This file-like object is then passed as a parameter to the Flask endpoint using the `client.post` method.

The response from the endpoint is then converted to a string and read back into a JSON object.

The test function checks that the status code of the response is 200, and then iterates through the result to check that the `presenting_team`, `pair_team`, `room`, and `timeslot` in the expected and actual results are equal.

```
def test_preview_generator_uploaded_file(self):
    #Create mock up uploaded excel file
    data = {'presenting_team': ['Group1', 'Group2', 'Group3', 'Group4', 'Group5', 'Group6', 'Group7', 'Group8'],
           'pair_team': ['Group8', 'Group7', 'Group6', 'Group5', 'Group4', 'Group3', 'Group2', 'Group1'],
           'room': ['Room1', 'Room2', None, None, None, None, None, None],
           'timeslot': ['Timeslot1', 'Timeslot2', 'Timeslot3', 'Timeslot4', None, None, None, None]}
    df = pd.DataFrame(data)
    excel_data = BytesIO()
    with pd.ExcelWriter(excel_data, engine='xlsxwriter') as writer:
        df.to_excel(writer, index=False, sheet_name='Sheet1')
    excel_data.seek(0)
    #Send the post request with uploaded file and get the response
    response = self.client.post('/upload-table-data', data={'file': (excel_data, 'test.xlsx')})
    file_obj = io.StringIO(response.data.decode('utf-8'))
    #Convert the response data to json
    json_data = json.load(file_obj)
    res_json = self.generate_result()
    #Check if the status code is equal to 200
    self.assertEqual(response.status_code, 200)
    for i in range(len(json_data['result'])):
        # Check if the response data corresponds to expected result
        self.assertEqual(json_data['result'][i]['presenting_team'], res_json[i]['presenting_team'])
        self.assertEqual(json_data['result'][i]['pair_team'], res_json[i]['pair_team'])
        self.assertEqual(json_data['result'][i]['room'], res_json[i]['room'])
        self.assertEqual(json_data['result'][i]['timeslot'], res_json[i]['timeslot'])
```

Figure 33: Test Uploaded File

6.3.5 Test Result



Figure 34: Test Result Flask Test

6.3.6 Usability Tests

To conduct the usability test, we worked with another design project team who were paired with us to give us a peer review. During the usability testing, a list of functionalities that the user is supposed to accomplish by using our system is presented. We asked each of the team members to perform the task without giving them detailed instructions. And then we observed and took notes. We also consulted with our client every week and gained feedback.

1. Upload a file containing teams, rooms and time slots from the first session as input.
2. Clear the file that is uploaded from the system.
3. Manually type the number of teams, rooms and time slots of the first session as input.
4. Submit the uploaded file and request for a schedule in the system.
5. Submit the manually input data and request for a schedule in the system.
6. Edit the data on the preview table in the system.
7. Edit the data on the schedule table in the system,
8. Highlight the data on the schedule table in the system.

9. Download the generated schedule in the system.
10. Regenerate the schedule in the system.

After conducting the usability test, we analyzed the result. We found out that the participants have difficulty navigating the web application, and finding out the function of editing tables and they reported frustration with on downloaded schedules not delimited by commas.

Based on the findings, we made several improvements to the web application, including redesigning the user interface, improving the table editing functionality and fixing the issue of downloaded schedules not delimited by commas. Several findings are listed below:

1. The schedule table looks a bit messy and not organized.

Solution: order the schedule table by room numbers and time slot numbers.

2. When changing the data in the preview table, the corresponding data on the schedule table would not change if changes were made before the schedule is generated.

Solution: We saved the changes in the form of form data from preview table, and send the request of schedule with the changed data to the backend. And the backend will use the changed data to generate a schedule, so that the changes in preview table will reflect on the schedule table as well.

3. When changing a certain team name in the preview table, other teams with the same name are mistakenly changed.

Solution: We came up with a solution to store the a set of indexes of table cells of same values. If any values of these indexes are changed, then update the values of all indexes.

4. After the schedule is generated, when changing the team names in the schedule table, sometimes the wrong corresponding data in the preview table is changed.

Solution: Debug if the wrong indexes of table cells are stored.

5. The functionality responsible for modifying data in preview table and schedule data only works for "presenting_team" and "pair_team", it doesn't work for "room" and "timeslot".

Solution: We implement the function responsible for modifying data in "room" and "timeslot" as well.

6. A wrong error message is given when regenerating the schedule.

Solution: A wrong value is passed to the function that is responsible for regenerating the schedule, which makes the system to give the wrong error message. Therefore, we checked the value before the value is passed to the function.

7. The downloaded schedule is not correctly delimited.

Solution: We found out that the downloaded schedule which is in CSV format is delimited by space rather than by comma. We change the delimited mark from space to comma, then the bug is fixed.

8. When the user first input the data manually to request a schedule, it worked well. However, when the user tried to delete one of the values from the input field, and uploaded a file and clicked submit button, the system still gave a result, which is not expected to happen.

Solution: We checked the fields when a file is submit, if all of the fields are valid, then it is allowed to request a schedule. Otherwise, an error message would be given.

Usability tests are an essential part of software development. By conducting usability tests, we are able to identify potential issues in our system and make plans for improvements based on the testing results, so that the user experience of the system can be improved.

7 Evaluation

In this chapter, we will discuss and evaluate various aspects of our design project, including our design and development process, team members' roles and responsibilities, and the final result. The aim of this chapter is to provide an overview of the team's strengths and areas for potential improvement throughout the process of the design project.

7.1 Design and Development Process

We started our project by designing a detailed plan with a well-structured timeline, outlining every task's scope and allowed time, from the inception in week 1 to the conclusion in week 10 of our project. Such a plan not only allowed us to set specific timelines for each task so that we can keep a reasonable workload for every week but also served as a reference and gave us a reliable way to gauge our progress and evaluate our scheduling flexibility, so that when unforeseen delays happen, we can accommodate them with ease and adapt our plan accordingly without compromising the successful completion of the project.

To make sure the project is collaborative, we used Belbin Roles and SCRUM methodology. Belbin Roles provided us with a better understanding of each team member's strengths, which assisted in the assignment of roles and distribution of responsibilities. For the SCRUM methodology, each team member took turns to be the SCRUM master, which ensured more equal participation.

We also arranged weekly meetings with our supervisor, Petra van den Bos, to report our weekly progress and receive valuable feedback, which helped us to refine the objectives of the project, improve our approach, discover and correct issues early, and stay on track. After the weekly meeting, we would always discuss what are the tasks for the next week and what progress can be reasonably expected, taking into consideration each team member's personal circumstances.

Of course, there are areas that can be improved throughout the project. For example, we should have designated a specific team member to keep meeting minutes and ensure that feedback from our supervisor was accurately

documented and fully understood. As a consequence, during the intermediate weeks, some of the feedback from our supervisor was not properly implemented in the following week, resulting in multiple revisions of the diagrams until they met our supervisor's satisfaction. On recognizing this issue, we kept better track of the feedback and avoided further unnecessary delays.

Moreover, our development of the product started earlier than planned. During the design phase, our team experimented with coding the algorithm and the outcome was satisfactory. Even though we managed to get the algorithm to work, it is a deviation from the best practice of finalizing the design choices before starting the coding phase. Consequently, our initial code structure was suboptimal, and each new addition to the code relied on the existing suboptimal structure, creating unnecessary challenges and complexities in refining it. Additionally, this reversal of the standard process resulted in delays and difficulties in completing the diagrams, as it was challenging to accurately represent the structure we had, and equally challenging to come up with new ideas ignoring the structure we had. Recognizing the issues, we adjusted our approach by halting the development, reassessing the structure, and finalizing the design diagrams, which ensured a smoother development process moving forward. We eventually managed to modify the code structure based on the improvements we made to the design, but we learned a lesson to always stick to the best practices in the development process.

7.2 Team Evaluation

Our team members naturally gravitated toward different Belbin Roles and tasks based on their personalities, interests and experience. Therefore, we have distributed responsibilities based on these factors. Below is a table of each team member's Belbin Roles, preferred tasks, and time for being the SCRUM master.

Below is also a summary of the main contribution of each team member to the project. It is worth noting that certain tasks, such as the completion of the design report, are a collaborative effort, and each team member undertook a certain part to write before we put it together and reviewed the entire report together; the parts each team member undertook will not be further specified. It is also worth noting that the list is not definitive or exhaustive,

Name	Belbin Roles	Preferred Tasks	SCRUM Master Time
Ferhat Ege Darici	Teamworker, Complete Finisher	Frontend or Backend	Week 8
Salih Eren Yüçetürk	Implementer, Plant	Frontend	Week 7
Ibrahim Teymurlu	Teamworker, Plant, Implementer	Frontend or Backend	Week 6
Boxuan Wang	Coordinator, Plant, Monitor Evaluator	Frontend or Backend	Week 4
Zinan Guo	Teamworker, Implementer	Frontend or Backend	Week 3
Zheyu Dong	Teamworker, Implementer, Complete finisher	Frontend or Backend	Week 5

as they only encompass major responsibilities. There are smaller tasks and less tangible contributions that played important roles in making our teamwork possible, which will not be listed.

Boxuan Wang:

- Participated in the specification of requirements and the writing of the project proposal
- Researched algorithm libraries and contributed to the algorithm design
- Made one of the initial UI designs and contributed to finalizing the UI design
- Participated in finalizing the diagrams
- Participated in testing
- Participated in weekly meetings, peer review sessions, and poster presentation

- Participated in writing the final report

Ferhat Ege Darici:

- Contributed to writing the requirements
- Participated in writing the project proposal
- Participated in writing the report
- Participated in supervision meetings
- Participated in diagram design
- Participated in UI design while brainstorming
- Designed and presented the poster
- Participated in weekly meetings, peer review sessions, and poster presentation

Ibrahim Teymurlu:

- Set up most of the requirements
- Was the team representative in supervisor meetings
- Contributed to architectural design
- Contributed to diagram sketching
- Participated in UI design and debugging
- Wrote design proposal and report and migrated everything to LaTeX
- Participated in weekly meetings, peer review sessions, and poster presentation
- Was the team representative in most of the presentations alongside Zheyu

Salih Eren Yuceturk:

- Contributed to writing the requirements
- Worked on the poster design
- Contributed to the UML diagram sketching
- Contributed to testing
- Contributed to the UI design
- Contributed on the written assignments and reports
- Participated in weekly meetings, peer review sessions, and the poster presentation
- Worked on the preparation of presentations

Zheyu Dong:

- Researched project frameworks and algorithm libraries
- Made one of the initial UI designs
- Made initial system diagrams and following changes
- Set up development and server environment
- Implemented frontend interface and functionalities
- Implemented backend functionalities
- Wrote system test cases and conducted tests
- Participated in writing design proposals and design reports
- Participated in client meetings, peer review sessions, and poster presentation

Zinan Guo:

- Researched the library for algorithm

- Contributed to the algorithms' design and the UI design
- Contributed to the final changes of the diagrams
- Implemented the algorithm and unit tests for the algorithm
- Conducted the major part of the manual and usability testing
- Participated in bugs fixing
- Participated in the writing of project proposal and final report
- Participated in weekly meetings, peer review sessions, and poster presentation

7.3 Final Results

The outcome of our project is a peer review scheduler that met the supervisor's requirements and expectations. We developed an algorithm that satisfied all constraints in scheduling peer review sessions. The user interface of our project is designed and revised to the supervisor's preference, resulting in a clean, intuitive, single-page design to ensure that users can easily navigate the features. For users' convenience, our project supports both Excel uploads and manual inputs of team, timeslot, and room numbers, offering users the flexibility to choose their preferred data entry method. Our project also includes features such as editing the team, room, and timeslot names and refreshing and downloading the schedule.

The efficiency of the algorithm is satisfactory for our supervisor. However, we acknowledge that it still has great potential to be faster. However, we have only several weeks for the whole project. Moreover, the scheduler is designed for the master course in Programming Principles, Patterns, and Processes at the University of Twente, and it is very unlikely that this course will involve a huge number of teams. Therefore, we decided that making the algorithm faster and more scalable is not an important requirement to be implemented. Still, one of the future works of this project is to explore more libraries instead of z3 and do some experiments on which libraries can produce the quickest result. Another future direction is to integrate the scheduler with Canvas to improve usability.

8 User Guide

In this section, the guide to install and run the web application and tests is provided. The IDE tool in this example is Pycharm. users are welcome to use any IDE tools, but we are not responsible for any error occurs if user is not using Pycharm.

8.1 Instalment of Web Application

1. Get the URL from gitlab, choose “clone with HTTPS”

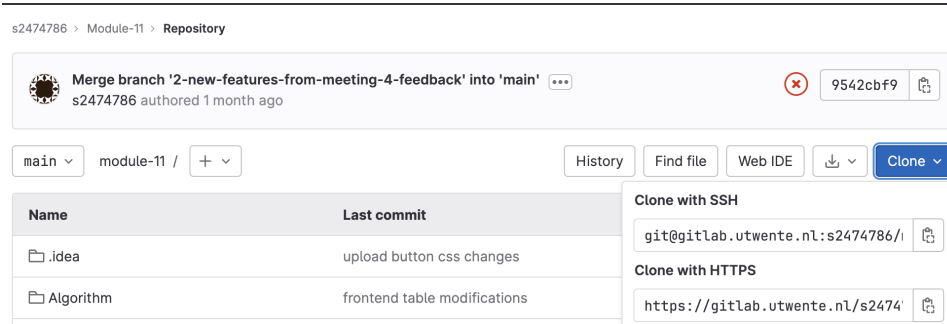


Figure 35: GitLab Cloning

2. Open Pycharm, create a new project, choose “get from VCS”, your credentials of UTwente gitlab might be requested.

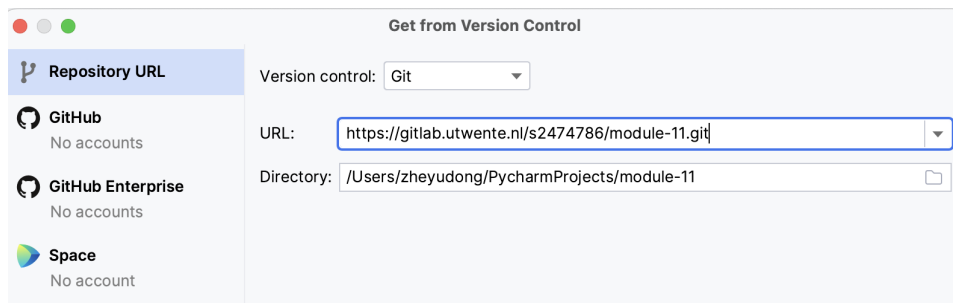


Figure 36: Version Control

3. Once the project is loaded, for windows users, go to "File -> Settings -> Project -> Python Interpreter" to configure the right python interpreter version on your machine. For mac users go to "Pycharm -> Settings -> Project -> Python Interpreter".

If user is using VS Code, don't forget to create a virtual environment for Python.

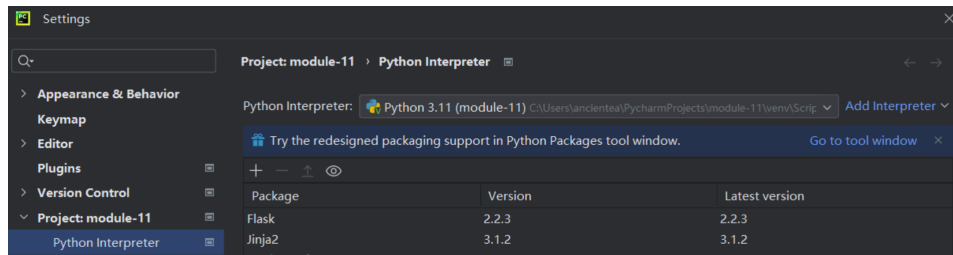


Figure 37: PyCharm Setting

4. Install following libraries in the command line: (based on your python version, examples given below use python 3.11)

- pip3 install flask
- pip3 install z3-solver
- pip3 install pandas
- pip3 install openpyxl
- pip3 install xlswriter

If user is using VS Code, please install the above package using "py -m pip install flask", same for the remaining commands. The commands above should be sufficient to run the web application. If the user find any other error message regarding "library not existed", please install corresponding libraries.

5. Right click on file main.py, choose run to start the application, and then click on the link below:

```

C:\Users\ancientea\PycharmProjects\module-11\venv\Scripts\python.exe C:\Users\ancientea\PycharmProjects\module-11\main.py
* Serving Flask app 'main'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8888

```

Figure 38: Last Step

8.2 Run Tests

To run the unit tests of the algorithm, go to the "Test" folder under the project, right click on the file "Test_Algorithm.py" or "Flask_Tests.py" and choose run. Then the tests will automatically be running.

In this section the guide to efficiently use the software, Peer Review Scheduler will be provided. All of the subsections below will refer to the following image:

8.3 Use of Application

The diagram illustrates the application's workflow with numbered steps:

- 2**: Upload File button
- 5**: Submit/Refresh button
- 6**: Download button
- 3**: Presenting Team table
- 4**: 2nd review team table

The **Presenting Team** table (3) contains the following data:

Presenting Team	Pair Team	Location	Timeslot
TEAM 1	TEAM 2	Horst 115	11:15-11:30
Group 3	Group 4	Room 2	Timeslot 2
Group 5	Group 6		Timeslot 3
Group 7	Group 8		Timeslot 4

The **2nd review team** table (4) contains the following data:

Presenting Team	Pair Team	2nd review team	Location	Timeslot
Group 3	Group 4	Group 7	Horst 115	11:15-11:30
Group 4	Group 3	Group 6	Horst 115	Timeslot 2
Group 5	Group 6	Group 3	Horst 115	Timeslot 3
Group 6	Group 5	TEAM 1	Horst 115	Timeslot 4
TEAM 2	TEAM 1	Group 5	Room 2	11:15-11:30
TEAM 1	TEAM 2	Group 8	Room 2	Timeslot 2
Group 7	Group 8	TEAM 2	Room 2	Timeslot 3
Group 8	Group 7	Group 4	Room 2	Timeslot 4

Figure 39: User Guide

8.3.1 Manual Data Entry

The user is allowed to enter the data manually. This can be obtained by filling out the boxes as shown in Figure 39, 1. The user should enter the number of teams, number of available rooms and number of available time slots for the application to generate a schedule. Simply press the "Submit/Refresh" button to generate a schedule.

8.3.2 File Upload for Data Entry

The user is also given a second way to upload the data. It is allowed to upload an Excel file to the system with the above-mentioned necessary details. For this, the user can use the "Upload File" button as marked with 2 in Figure 39. For the system to parse the file correctly, the Excel file should be formatted similarly to the preview table (slot 3 of Figure 39), i.e. the first two columns should be the names of the twin team pairs, the third column should be the names of the locations, and the fourth column should be the names of the timeslots. Afterwards, just like the manual entry, the user can press the "Submit/Refresh" button to generate a schedule.

8.3.3 Editing Data

The user can edit the produced data using the slot 3 shown in Figure 39. By simply clicking on the fields, the user is given the option to edit the fields to their liking. For instance, TEAM 1 on the image above, was Group 1 before. Also be aware that any data changed in one place will eventually be replaced in every other place as well.

8.3.4 Highlighting

To maintain visibility, the application contains highlighting. This feature can be used by hovering the mouse pointer on a slot. Please, see number 4 in Figure 39. By, hovering the mouse on the slot, "TEAM 1" the user is able to highlight the team name on every other position as well.

8.3.5 Downloading and Re-Generating the Schedule

By using the "Download" button which is marked as 6 in Figure 39, the user can download the schedule as a .csv file. Furthermore, the user is also given the option to re-generate a schedule using the "Submit/Refresh" button denoted as 5.

If the .csv file the user downloaded is separated incorrectly, that is because our system uses the comma(,) as the list separator for the .csv file, but the user's computer may be set up differently, such as using the semicolon(;) as the default list separator. We noticed that the default list separator is different between PC and Mac, and varies across different regions in the world, so our system cannot accommodate every user's computer setup.

If the user's .csv file is separated incorrectly, the user can do one of the two following things:

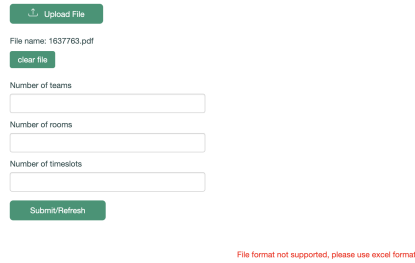
- Replace all the commas(,) in the .csv file with semicolons(;): open the .csv file with a default text editor, and use the replace function.
- Change Excel settings (assuming the user uses Excel to read the .csv file): open Excel -> options -> advanced, uncheck "use system separators", and switch the decimal separator and the thousands separator.

	A	B	C	D	E	F
1	Schedule Data					
2						
3	presenting_	pair_team	additional_t	room	timeslot	
4	Group4	Group3	Group6	Room1	Timeslot1	
5	Group5	Group6	Group3	Room1	Timeslot2	
6	Group6	Group5	Group2	Room1	Timeslot3	
7	Group1	Group2	Group5	Room1	Timeslot4	
8	Group2	Group1	Group7	Room2	Timeslot1	
9	Group8	Group7	Group1	Room2	Timeslot2	
10	Group7	Group8	Group4	Room2	Timeslot3	
11	Group3	Group4	Group8	Room2	Timeslot4	
12						
13						
14						

Figure 40: Format of the downloaded schedule

8.3.6 Error message

The user should upload the file in Excel 2010 "xlsx/xlsm/xlsx/xlsm" format, otherwise an error message would generate as shown in the Figure 41:



The screenshot shows a web form with the following elements: a green 'Upload File' button, a text field containing 'File name: 1637763.pdf', a green 'Clear file' button, three input fields labeled 'Number of teams', 'Number of rooms', and 'Number of timeslots', and a green 'Submit/Refresh' button. Below the form, a red error message reads: 'File format not supported, please use excel format'.

Figure 41: File format not supported

When enter the data manually, the user should only enter input positive numbers, otherwise an error message would generate as shown in the Figure 42:



The screenshot shows a web form with the following elements: a green 'Upload File' button, three input fields labeled 'Number of teams', 'Number of rooms', and 'Number of timeslots', and a green 'Submit/Refresh' button. The input fields contain the values '8', '2', and '-1' respectively. Below the form, a red error message reads: 'Please input an valid positive integer'.

Figure 42: Please input an valid positive integer

When the input can't generate a valid schedule, an error message would be generated as shown in the Figure 43. If this is the case, please check if the number of teams, rooms and time slots are well matched. For example, 8 teams with 2 rooms and 2 times slots are not well matched, since 2 rooms and 2 time slots cannot fit 8 teams.

The screenshot shows a web form with the following elements:

- An "Upload File" button.
- Input fields for "Number of teams" (value: 8), "Number of rooms" (value: 2), and "Number of timeslots" (value: 3).
- A "Submit/Refresh" button.
- A table with the following data:

Presenting Team	Pair Team	Location	Timeslot
Group1	Group2	Room1	Timeslot1
Group3	Group4	Room2	Timeslot2
Group5	Group6		Timeslot3
Group7	Group8		

Below the table, the error message "Result does not exist" is displayed in red text.

Figure 43: Result does not exist

When any one of the input fields is empty, an error message would be generated as shown in Figure 44:

The screenshot shows a web form with the following elements:

- An "Upload File" button.
- Input fields for "Number of teams" (value: -1), "Number of rooms" (empty), and "Number of timeslots" (empty).
- A "Submit/Refresh" button.

Below the form, the error message "Please fill in all fields" is displayed in red text.

Figure 44: Fields are not filled

Each field in the preview table and schedule table must be unique. If there are duplicate fields, after clicking "Submit/Refresh", an error message would be generated as shown in Figure 45:

The screenshot shows a web interface for team scheduling. On the left, there are three input fields: "Number of teams" with the value 8, "Number of rooms" with the value 2, and "Number of timeslots" with the value 4. Above these fields is an "Upload File" button, and below them is a "Submit/Refresh" button. To the right is a table with the following data:

Presenting Team	Pair Team	Location	Timeslot
Group1	Group1	Room1	Timeslot1
Group3	Group4	Room2	Timeslot2
Group5	Group6		Timeslot3
Group7	Group8		Timeslot4

Below the table, a red error message reads: "Please make sure there are no duplicate names !".

Figure 45: Duplicate names