SEG Agile Group Project

- Adriel Aiach-Kohen k1631099 Student Number: 1604954
- Anttoni Pykalisto k1631083 Student Numebr: 1608665
- Jaman Salique k1630580 Student Number: 1625442
- Muhammad Junaid Mehmood k1630622 Student Number: 1680371
- Tomas Sleyman k1631322 Student Number: 1632764

Planning

We decided to use Trello as it provided a simple yet comprehensive platform to create a Kanban board containing user stories and team progress tracking. Due to the agile nature of the project, it was essential that each team member understood the tasks and which stage of development they were in. Boards for the to-do list and user stories were created, we assigned tasks to team members which helped plan future objectives during our team meetings and the boards were updated accordingly.

The Kanban board contains all the user stories that we felt had to be too implemented, initially each user stories is listed as a feature. Once the features has been assigned, the corresponding card is moved into the development list, upon completion it is listed as a completed task. The simplicity of tracking our progress allows each team member to easily understand updates and provides increased transparency. We also included a list of assumptions, where we eliminate any ambiguities throughout the development of the project. A similar structure is applied to our to-do Kanban board, where tasks are broken down and distributed. The Kanban boards are available for viewing on Trello for a more detailed view.

Design

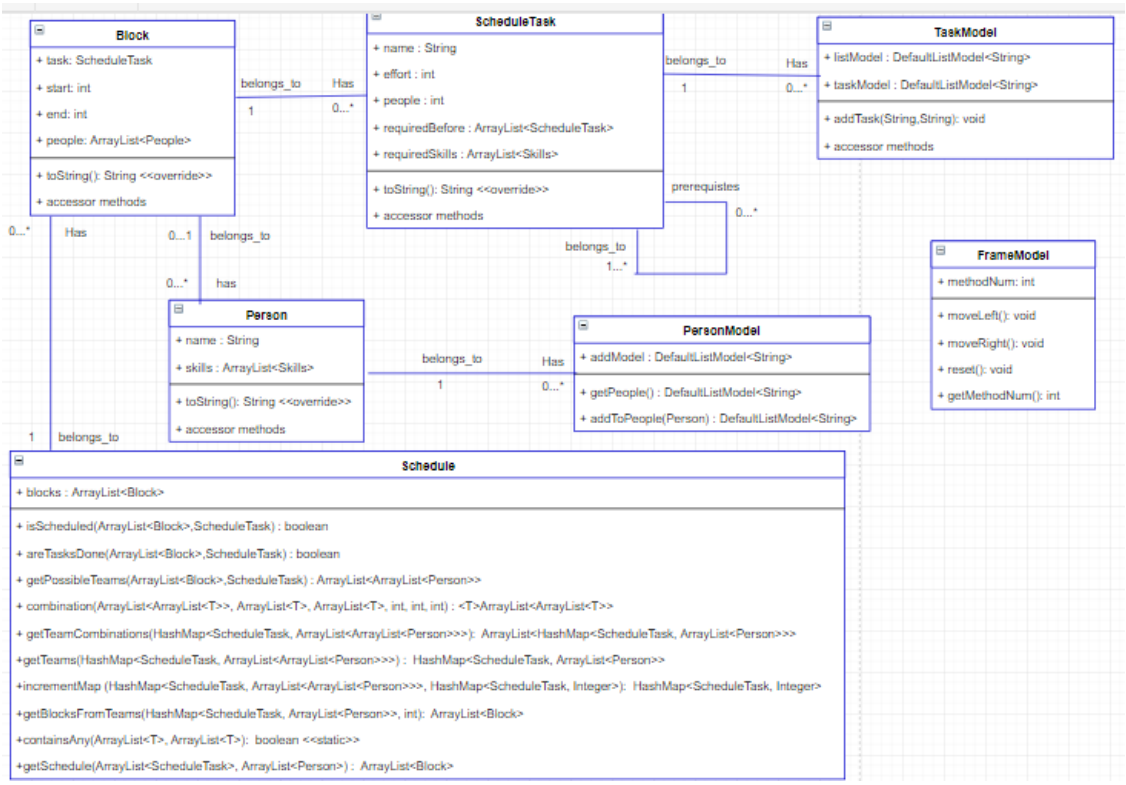
We wanted to develop a project based on structured and efficient code, as it increases to the potential to develop it further. Due to the changing nature of agile development, it is essential that external parties are able to understand the code and work with it easily. Clean code can be subjective, but some practices are commonly used and improve the efficiency of group work. We decided to follow the model-view-controller design pattern to structure our code, which allowed us to work on one aspect of the application without affecting others. This also opens the door to future developments and the ability to work independently yet collectively. The architecture pattern breaks the business logic down to the model and the user-interface to the view. The controller provides a platform to link the model and view together through updating the view and processing the user's interaction.

Essential features may have been omitted as a result of the assumptions we made throughout the development of the application. This highlights the importance of ensuring well-structured and maintainable code, essentially allowing our application to adapt with changing environments. High extensibility translates into less time for a developer to adapt the code and develop new functionality.

Carrying out test-driven development has allowed us to efficiently develop our software, shorten development time and increases productivity. Following this development technique we initially wrote an automated test case which required an improvement or implementation of a feature, then produced code that sufficiently passes the test case. We then re-factored the code to meet the desired expectations and standards and continued to repeat the process until the schedule planner was successfully built.

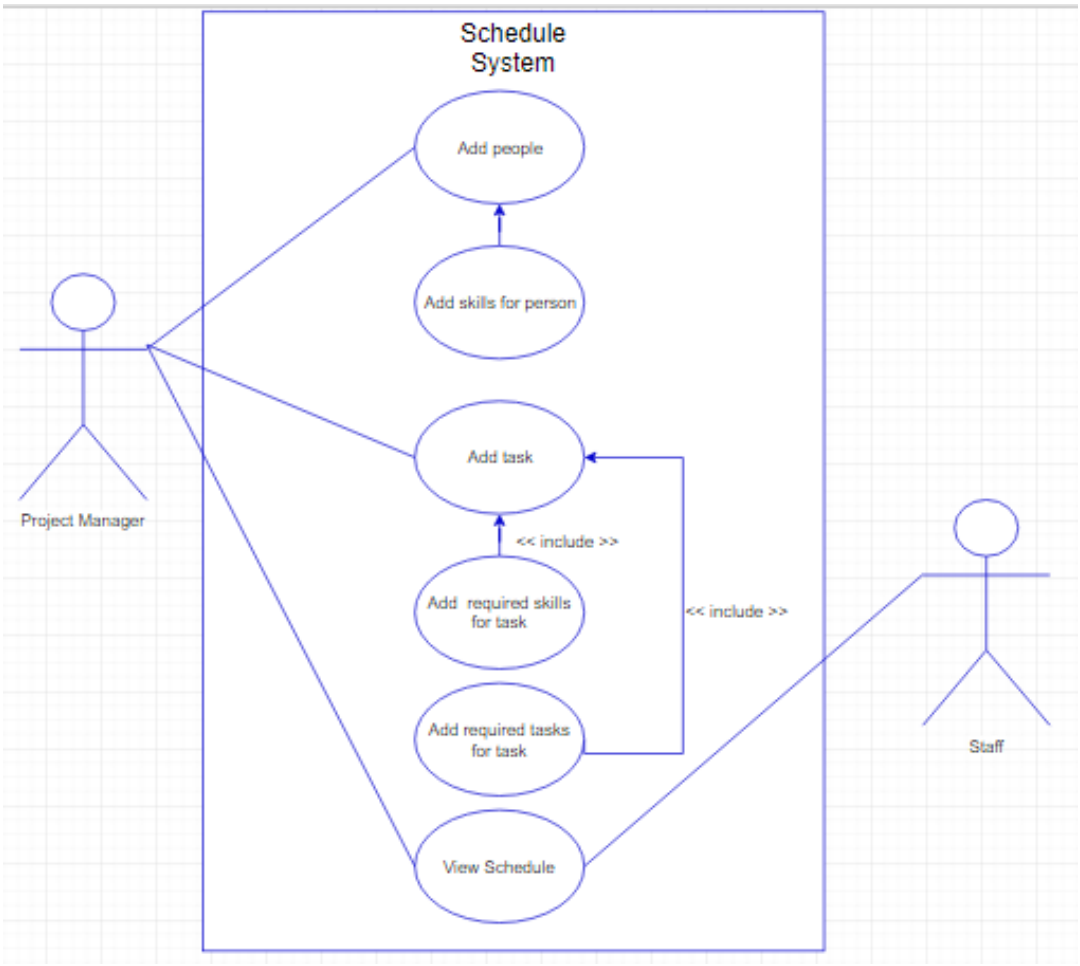
Class structure

To help show the class structure of our application we decided to create a class diagram. Our classes are Block, TaskModel, ScheduleTask, Person, PersonModel, FrameModel, and Schedule. The class diagram below shows how each class relates to each other.



Use Cases

To show interactions between different actors in our application we created a use case diagram which shows the different actors and the different operations they can do.



Use Case Description

Add people:

Use case name:	**The actor wants to add people**
Actor:	**Project Manager**
Preconditions:	**Persons name and skills need to be known**
Postconditions:	**People is added to list**
Main flow:	**1. The actor chooses to add people** **2. Actor is guided by the system to fill in required information to add the people to the list**

Add Tasks:

Use case name:	**The actor wants to add tasks**
Actor:	**Project Manager**
Preconditions:	**Tasks: name, skills required, previous tasks required, effort estimate, and number of people needed all need to be known**

Postconditions:	**Tasks is added to list**
Main flow:	**1. The actor chooses to add task** **2. Actor is guided by the system to fill in required information to add the tasks to the list**

View Schedule:

Use case name:	**The actor wants to view schedule**
Actor:	**Project Manager or Staff**
Preconditions:	**None**
Postconditions:	**None**
Main flow:	**1. The actor chooses to view schedule** **2. System displays schedule**