COMPSCI5059: Software Engineering M

Project Report

By:

——- SLAVLINGS ——-

Nadezhda Dimitrova - 2172605D
Ventsislav Antov - 2504299A
Tereza Buckova - 2200528B
Filip Marinov - 2175499M

School of Computing Science
UNIVERSITY OF GLASGOW

An assessed team project submitted to the University of Glasgow in accordance with the requirements for the degree SOFTWARE DEVELOPMENT (MSC) in the School of Computing Science. Supervised by:

DR RON POET

MARCH 2020

Table of Contents

1 | Design Specification

"Before the start of each term or semester, the class directors produce a list of teaching requirements which we must try and fill. Our administrator will then attempt to find suitable staff and organise training for them. All teaching requests must be approved by the PTT Director."

Following this specification, we identified the following user stories:

0. As a user I can quit the app at any point.
1. As a course director I can produce a list of teaching requirements.
2. As a course director I can produce a teaching request for a course.
3. As an admin I can try to find suitable staff.
4. As an admin I can organise training for the suitable staff.
5. As a PTT director I can approve a teaching request.

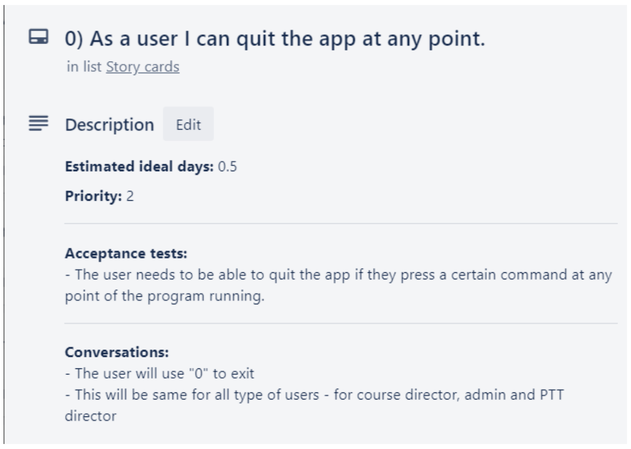
Additionally, we decided to include a `quit` and `go back` functionality to improve the user experience.

The standalone app with the described functionality was designed with the following assumptions:

- The app does not require a database.
- Every course director directs only one class.
- To teach a course only one training for a teacher is required.
- Teacher can have multiple teaching skills.
- If the administrator does not add a training requirement, the teaching request can not be approved by the PTT director.
- One course can only have one teacher.
- There is only one PTT Director and only one administrator.
- The teaching requirements can only be rewritten instead of adding new ones to the old ones in the list.
- The functionality is designed only for the administrator, PTT Director and Course Directors.

2 | User Stories

The following User Stories were identified with 1 ideal day = 8 ideal hours. Priority is expressed as 1 (must have), 2(should have), 3(will not have.)



0) As a user I can quit the app at any point.
in list Story cards

Description Edit

Estimated ideal days: 0.5

Priority: 2

Acceptance tests:
- The user needs to be able to quit the app if they press a certain command at any point of the program running.

Conversations:
- The user will use "0" to exit
- This will be same for all type of users - for course director, admin and PTT director

Figure 2.1: User story 0

0) "As a user I can quit the app at any point."

Estimated ideal days: 0.5

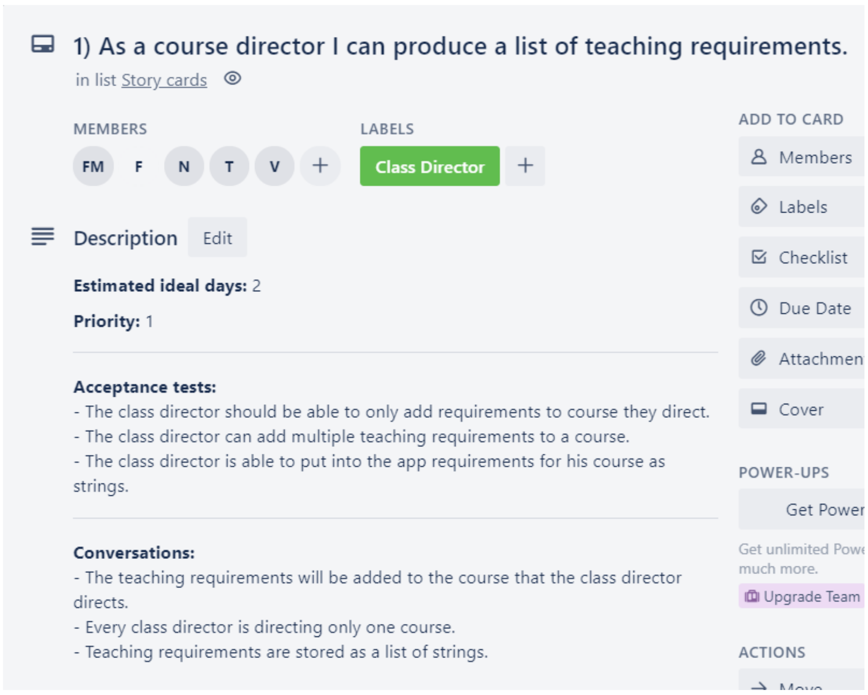
Priority: 2

Acceptance tests:

- The user needs to be able to quit the app if they press a certain command at any point of the program running.

Conversations:

- The user will use "-1" to exit.
- This will be same for all type of users - for course director, admin and PTT director.



1) As a course director I can produce a list of teaching requirements.

Estimated ideal days: 2

Priority: 1

Acceptance tests:
- The class director should be able to only add requirements to course they direct.
- The class director can add multiple teaching requirements to a course.
- The class director is able to put into the app requirements for his course as strings.

Conversations:
- The teaching requirements will be added to the course that the class director directs.
- Every class director is directing only one course.
- Teaching requirements are stored as a list of strings.

Figure 2.2: User story 1

1) "As a course director I can produce a list of teaching requirements."

Estimated ideal days: 2

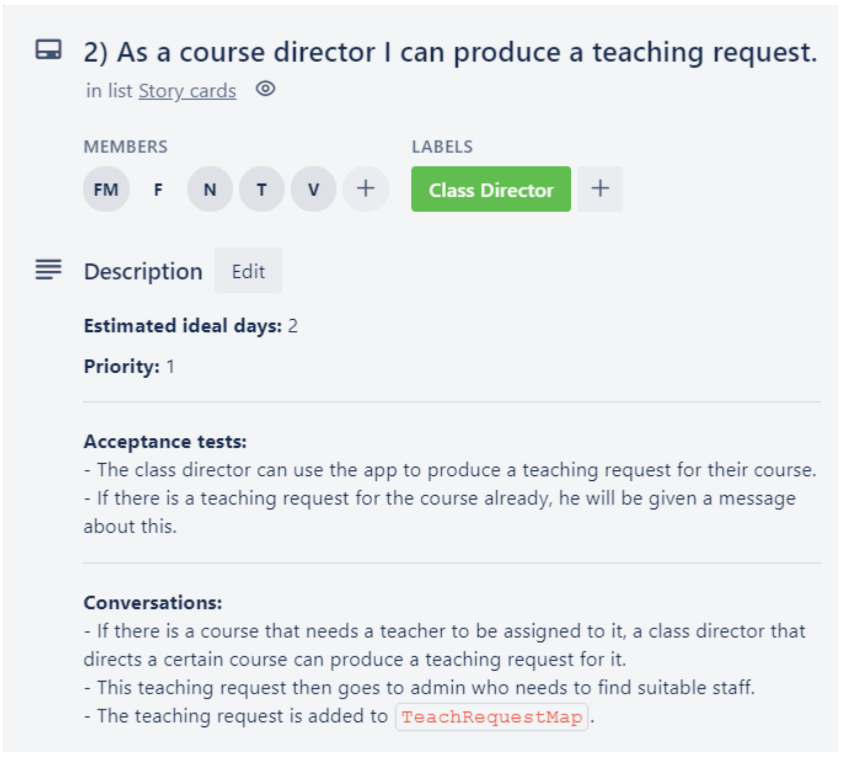
Priority: 1

Acceptance tests:

- The course director should be able to only add requirements to the course they direct.
- The course director can add multiple teaching requirements to a course.
- The course director is able to put the requirements into the app for his course as strings.

Conversations:

- All teaching requirements must be met in order for the teacher to be qualified as suitable for teaching the course
- The teaching requirements will be added to the course that the course director directs.
- Teaching requirements are stored as a list of strings.


2) As a course director I can produce a teaching request.

in list Story cards

MEMBERS: FM F N T V +

LABELS: Class Director +

Description Edit

Estimated ideal days: 2

Priority: 1

Acceptance tests:
- The class director can use the app to produce a teaching request for their course.
- If there is a teaching request for the course already, he will be given a message about this.

Conversations:
- If there is a course that needs a teacher to be assigned to it, a class director that directs a certain course can produce a teaching request for it.
- This teaching request then goes to admin who needs to find suitable staff.
- The teaching request is added to `TeachRequestMap`.

Figure 2.3: User story 2

2) "As a course director I can produce a teaching request for a course."

Estimated ideal days: 2

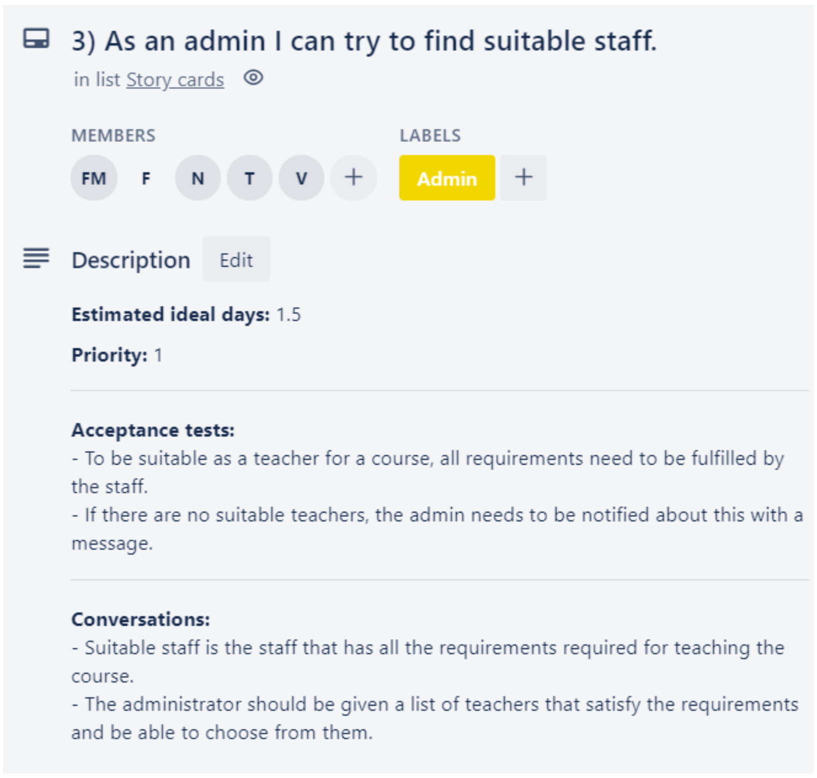
Priority: 1

Acceptance tests:

- If the teaching request has already been added, it will not be added again.
- If the teaching request has not been added already, it will be added.

Conversations:

- If there is a course that needs a teacher to be assigned to it, a course director that directs a certain course can produce a teaching request for it.
- This teaching request then goes to admin who needs to find a suitable teacher.
- The teaching request has to be added to TeachRequestMap.


3) As an admin I can try to find suitable staff.

in list Story cards

Estimated ideal days: 1.5

Priority: 1

Acceptance tests:
- To be suitable as a teacher for a course, all requirements need to be fulfilled by the staff.
- If there are no suitable teachers, the admin needs to be notified about this with a message.

Conversations:
- Suitable staff is the staff that has all the requirements required for teaching the course.
- The administrator should be given a list of teachers that satisfy the requirements and be able to choose from them.

Figure 2.4: User story 3

3) "As an admin I can try to find suitable staff."

Estimated ideal days: 1.5

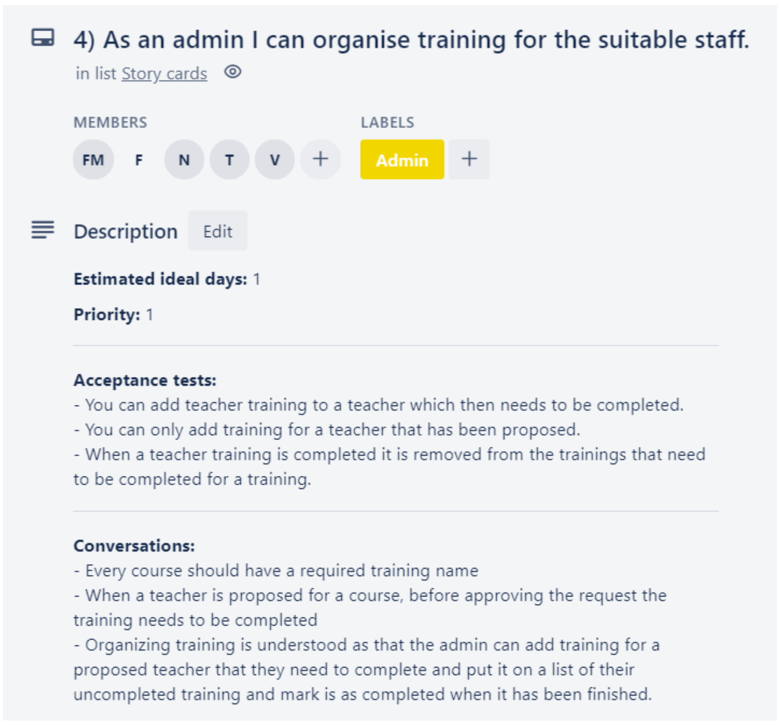
Priority: 1

Acceptance tests:

- If a course has no teaching requirements set, no suitable staff will be found and then the admin will be notified.
- Staff are found to be suitable only if all teaching requirements of the course are contained within their list of skills.
- The admin has to be able to propose the suitable teacher for a teacher position if there is a teaching request for the course.

Conversations:

- Finding suitable staff is understood as searching for a teacher who fits the requirements and proposing them for the position.
- Suitable staff are the staff that have all the requirements required for teaching the course.
- The requirements are a list of strings.
- The administrator should be given a list of teachers that satisfy the requirements and be able to choose from them.


4) As an admin I can organise training for the suitable staff.
in list Story cards

MEMBERS
FM F N T V +

LABELS
Admin +

Description Edit

Estimated ideal days: 1

Priority: 1

Acceptance tests:
- You can add teacher training to a teacher which then needs to be completed.
- You can only add training for a teacher that has been proposed.
- When a teacher training is completed it is removed from the trainings that need to be completed for a training.

Conversations:
- Every course should have a required training name
- When a teacher is proposed for a course, before approving the request the training needs to be completed
- Organizing training is understood as that the admin can add training for a proposed teacher that they need to complete and put it on a list of their uncompleted training and mark is as completed when it has been finished.

Figure 2.5: User story 4

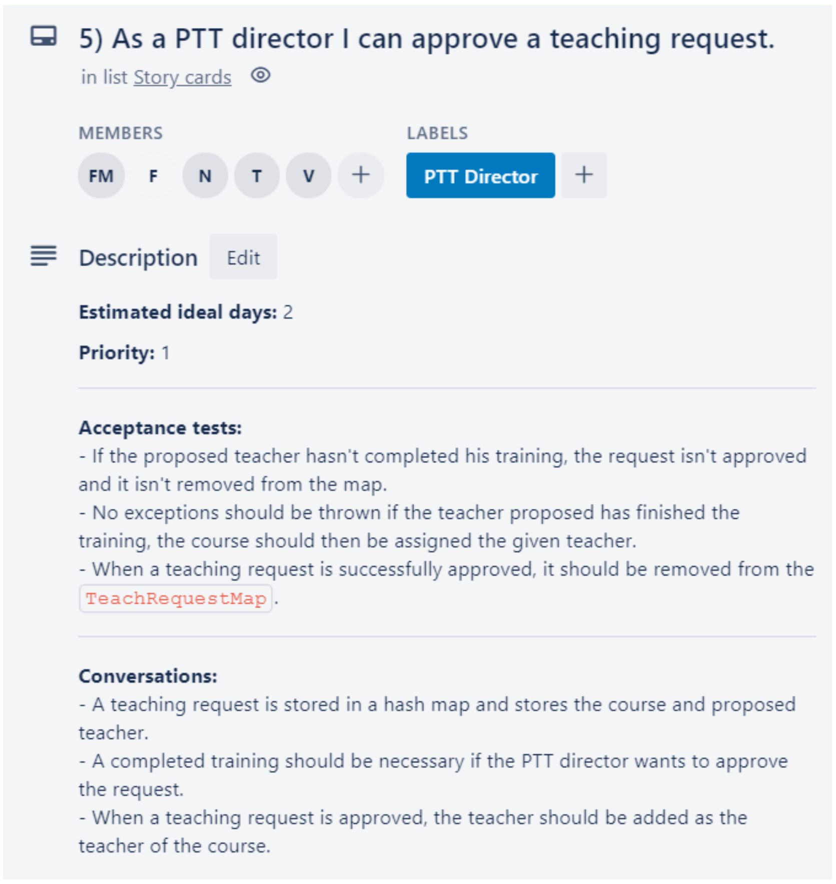
4) **"As an admin I can organise training for the suitable staff."**
Estimated ideal days: 1
Priority: 1

Acceptance tests:

- If the admin tries to add a training to teacher who already has a training with the same name, the admin is not able to add it again.
- You can only add training for a teacher that has been proposed.
- When a teacher training is completed, it is removed from the trainings that need to be completed and is added to the trainings that are completed.
- When a training that is unassigned is selected to be completed, nothing happens.

Conversations:

- Training means that before a teacher can teach a course they need to complete a specific training unit. This is separate from the teacher's qualifications. An example of a training would be "Health and Safety" training.
- Every course should store and take as a parameter a required training name.
- When a teacher is proposed for a course, before approving the request, the training needs to be completed.
- Admin can add training for a proposed teacher that needs to be completed before the teacher can teach the specific course. It needs to be put on list of the teacher's uncompleted trainings and marked as completed when it has been finished.


5) As a PTT director I can approve a teaching request.
in list Story cards

MEMBERS
FM F N T V +

LABELS
PTT Director +

Description Edit

Estimated ideal days: 2

Priority: 1

Acceptance tests:
- If the proposed teacher hasn't completed his training, the request isn't approved and it isn't removed from the map.
- No exceptions should be thrown if the teacher proposed has finished the training, the course should then be assigned the given teacher.
- When a teaching request is successfully approved, it should be removed from the `TeachRequestMap`.

Conversations:
- A teaching request is stored in a hash map and stores the course and proposed teacher.
- A completed training should be necessary if the PTT director wants to approve the request.
- When a teaching request is approved, the teacher should be added as the teacher of the course.

Figure 2.6: User story 5

5) **"As a PTT director I can approve a teaching request."**
Estimated ideal days: 2
Priority: 1

Acceptance tests:

- If there is no teaching request placed for the course, no approval can happen.
- If the proposed teacher hasn't completed his training, the request isn't approved and it isn't removed from the TeachRequestMap.
- If there is no proposed teacher, the request isn't approved and it isn't removed from the TeachRequestMap
- When a teaching request is successfully approved, it should be removed from the map.
- No exceptions should be thrown if the teacher proposed has finished the training, the course should then be assigned the given teacher.

Conversations:

- A teaching request is stored in a hash map and stores the course and proposed teacher.

- A completed training should be necessary if the PTT director wants to approve the request.
- When a teaching request is approved, the teacher should be added as the teacher of the course.

Actual/Estimated Effort

1 ideal day = 8 ideal hours

User stories

- "As a user I can quit the app at any point."
  - Estimated ideal days: 0.5
  - Actual days: 0.5
- "As a course director I can produce a list of teaching requirements."
  - Estimated ideal days: 1.5
  - Actual days: 2
- "As a course director I can produce a teaching request for a course."
  - Estimated ideal days: 1.5
  - Actual days: 2
- "As an admin I can try to find suitable staff."
  - Estimated ideal days: 2
  - Actual days: 2
- "As an admin I can organise training for the suitable staff."
  - Estimated ideal days: 1
  - Actual days: 1
- "As a PTT director I can approve a teaching request."
  - Estimated ideal days: 2
  - Actual days: 3

Developer tasks

Class	Task	Method	Est. ideal hours	Actual hours
`Employee`	1	toString()	0.5	0.5
`Teacher`	2	satisfiesTeachReqs()	0.5	1
	3	getter and setter	0.25	0.5
	4	completeTraining()	0.1	0.5
	5	completedTraining()	0.5	0.5
	6	addTraining()	1	0.5
`Course`	7	getter, setter, toString()	0.5	1
	8	requirementsFulfilled()	1	1
`Course Director`	9	getter and setter	0.5	1
	10	courseTeachRequirements()	1	1
`Admin`	11	findSuitableStaff()	0.5	1
	12	proposeTeacher()	0.5	2
	13	addTraining()	0.5	2
	14	completeTraining()	0.5	2
`PTT Director`	15	approveTeachRequest()	1	2
`Teach Request`	16	approve()	1	2
	17	proposeTeacher()	1	1
`TeachRequesMap`	18	addTeachRequest()	2	3
	19	proposeTeacher()	2	2
	20	approveTeachRequest()	1	2
`InputOutput`	21		4	16
`DataWrapper`	22		4	8
`Controller`	23	adminMenu()	6	10
	24	pttDirectorMenu()	4	8
	25	courseDirectorMenu()	4	8

3 | Diagrams

3.1 Class diagrams

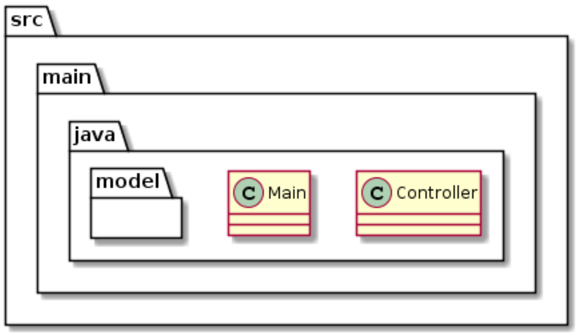


Figure 3.1: Project File Structure

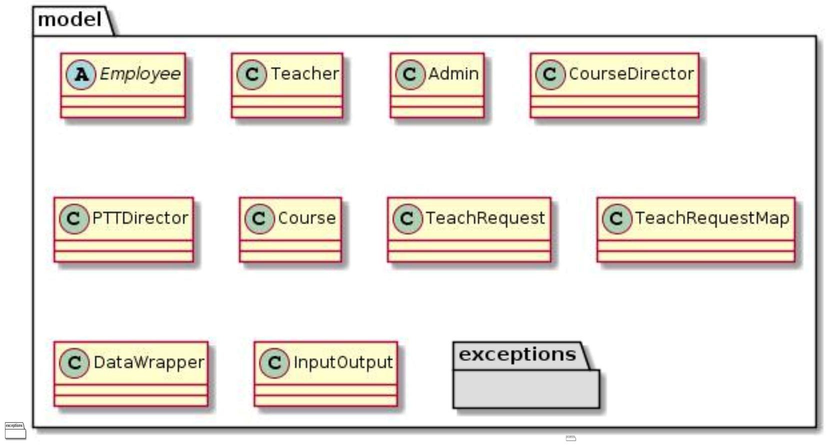


Figure 3.2: Model classes

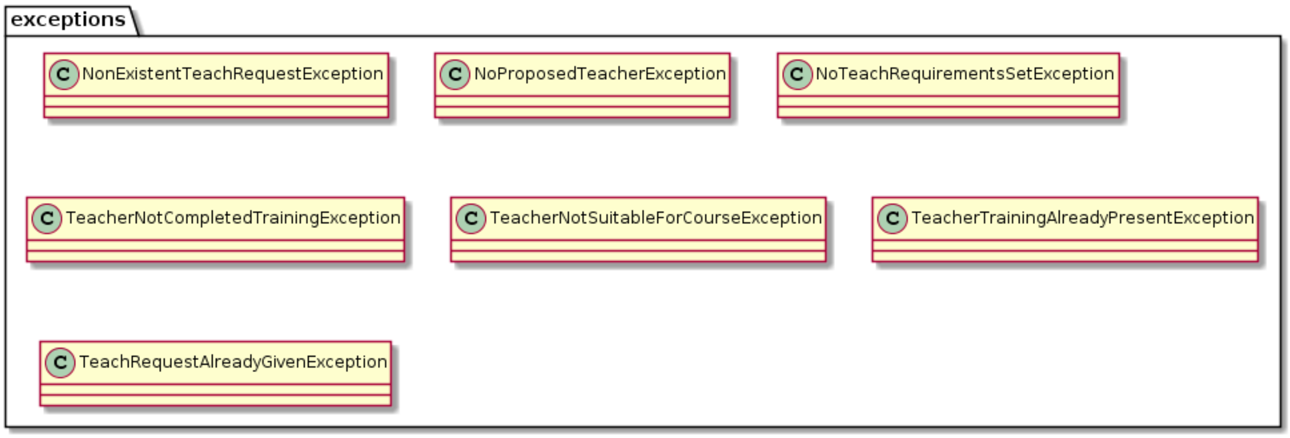

exceptions
NonExistentTeachRequestException
NoProposedTeacherException
NoTeachRequirementsSetException
TeacherNotCompletedTrainingException
TeacherNotSuitableForCourseException
TeacherTrainingAlreadyPresentException
TeachRequestAlreadyGivenException

Figure 3.3: Custom Exceptions

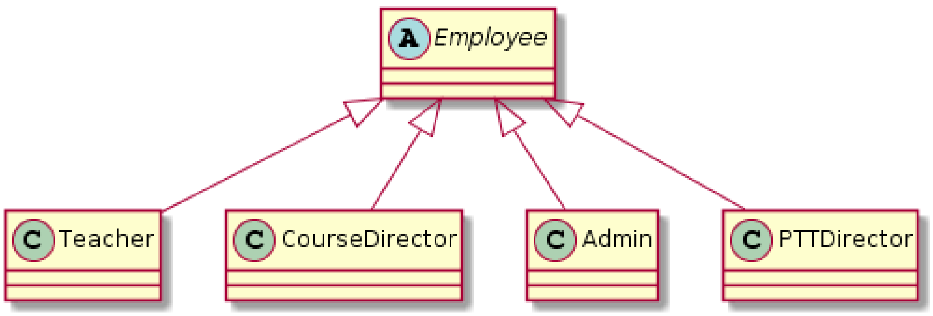
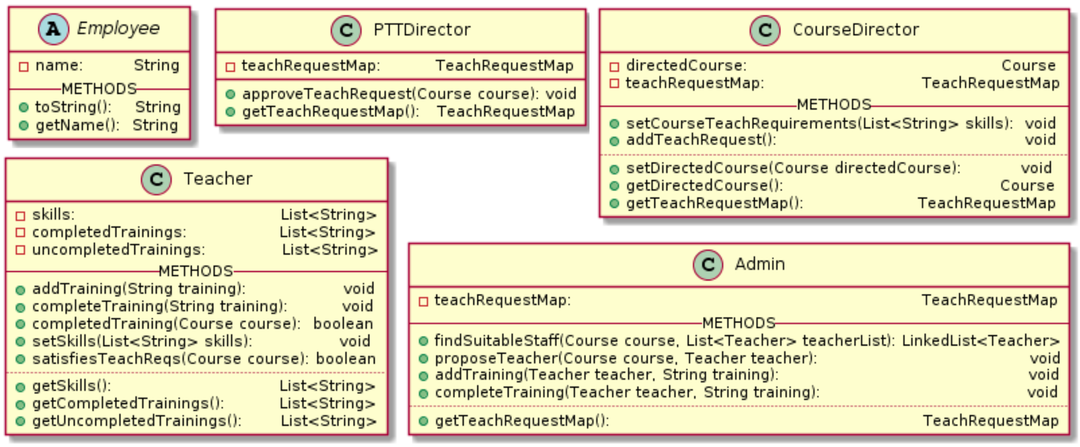
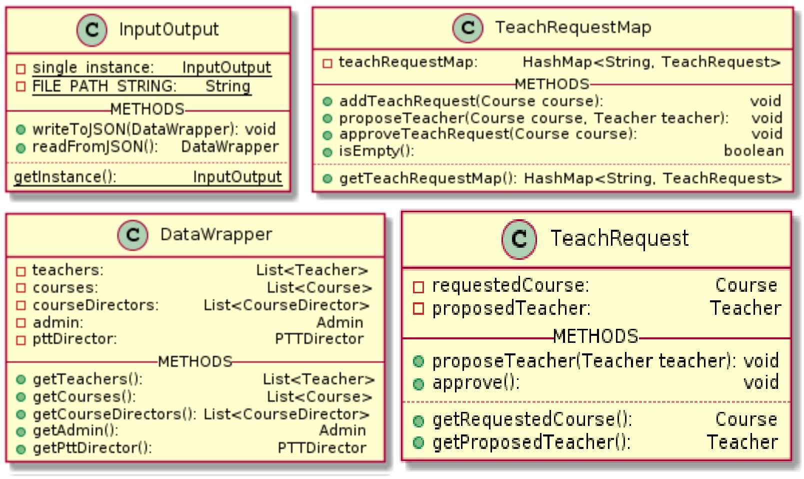


Figure 3.4: Classes extending `Employee`



Employee
name: String
METHODS
toString(): String
getName(): String
PTTDirector
teachRequestMap: TeachRequestMap
approveTeachRequest(Course course): void
getTeachRequestMap(): TeachRequestMap
CourseDirector
directedCourse: Course
teachRequestMap: TeachRequestMap
METHODS
setCourseTeachRequirements(List<String> skills): void
addTeachRequest(): void
setDirectedCourse(Course directedCourse): void
getDirectedCourse(): Course
getTeachRequestMap(): TeachRequestMap
Teacher
skills: List<String>
completedTrainings: List<String>
uncompletedTrainings: List<String>
METHODS
addTraining(String training): void
completeTraining(String training): void
completedTraining(Course course): boolean
setSkills(List<String> skills): void
satisfiesTeachReqs(Course course): boolean
getSkills(): List<String>
getCompletedTrainings(): List<String>
getUncompletedTrainings(): List<String>
Admin
teachRequestMap: TeachRequestMap
METHODS
findSuitableStaff(Course course, List<Teacher> teacherList): LinkedList<Teacher>
proposeTeacher(Course course, Teacher teacher): void
addTraining(Teacher teacher, String training): void
completeTraining(Teacher teacher, String training): void
getTeachRequestMap(): TeachRequestMap

Figure 3.5: Method and Field Descriptions of classes extending `Employee`



InputOutput
single_instance: InputOutput
FILE_PATH_STRING: String
METHODS
writeToJSON(DataWrapper): void
readFromJSON(): DataWrapper
getInstance(): InputOutput
TeachRequestMap
teachRequestMap: HashMap<String, TeachRequest>
METHODS
addTeachRequest(Course course): void
proposeTeacher(Course course, Teacher teacher): void
approveTeachRequest(Course course): void
isEmpty(): boolean
getTeachRequestMap(): HashMap<String, TeachRequest>
DataWrapper
teachers: List<Teacher>
courses: List<Course>
courseDirectors: List<CourseDirector>
admin: Admin
pttDirector: PTTDirector
METHODS
getTeachers(): List<Teacher>
getCourses(): List<Course>
getCourseDirectors(): List<CourseDirector>
getAdmin(): Admin
getPttDirector(): PTTDirector
TeachRequest
requestedCourse: Course
proposedTeacher: Teacher
METHODS
proposeTeacher(Teacher teacher): void
approve(): void
getRequestedCourse(): Course
getProposedTeacher(): Teacher

Figure 3.6: Method and Field Descriptions of classes `DataWrapper`, `InputOutput`, `TeachRequest` and `TeachRequestMap`.

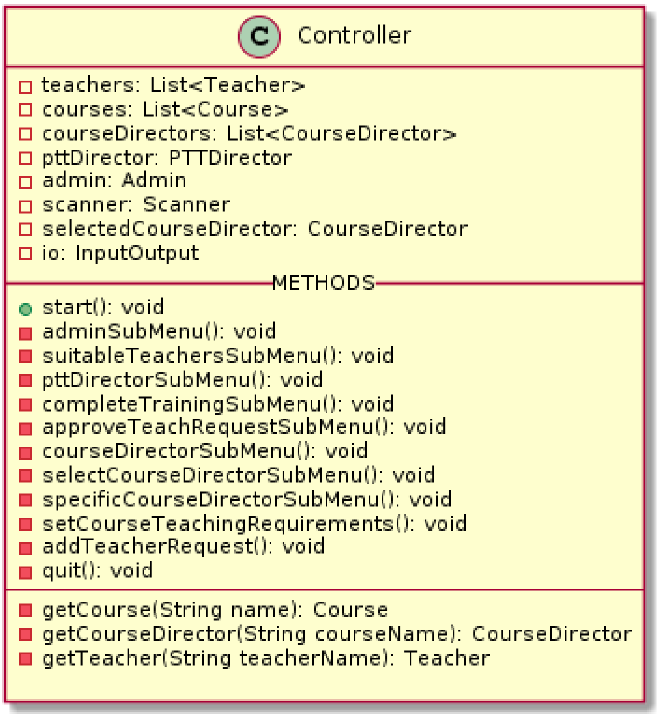

Controller
teachers: List<Teacher>
courses: List<Course>
courseDirectors: List<CourseDirector>
pttDirector: PTTDirector
admin: Admin
scanner: Scanner
selectedCourseDirector: CourseDirector
io: InputOutput
METHODS
start(): void
adminSubMenu(): void
suitableTeachersSubMenu(): void
pttDirectorSubMenu(): void
completeTrainingSubMenu(): void
approveTeachRequestSubMenu(): void
courseDirectorSubMenu(): void
selectCourseDirectorSubMenu(): void
specificCourseDirectorSubMenu(): void
setCourseTeachingRequirements(): void
addTeacherRequest(): void
quit(): void
getCourse(String name): Course
getCourseDirector(String courseName): CourseDirector
getTeacher(String teacherName): Teacher

Figure 3.7: Method and Field Descriptions of the `Controller` class.

Character	**Icon for field**	**Icon for method**	**Visibility**
–	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

Figure 3.8: Visibility Legend for the aforementioned UML diagrams.

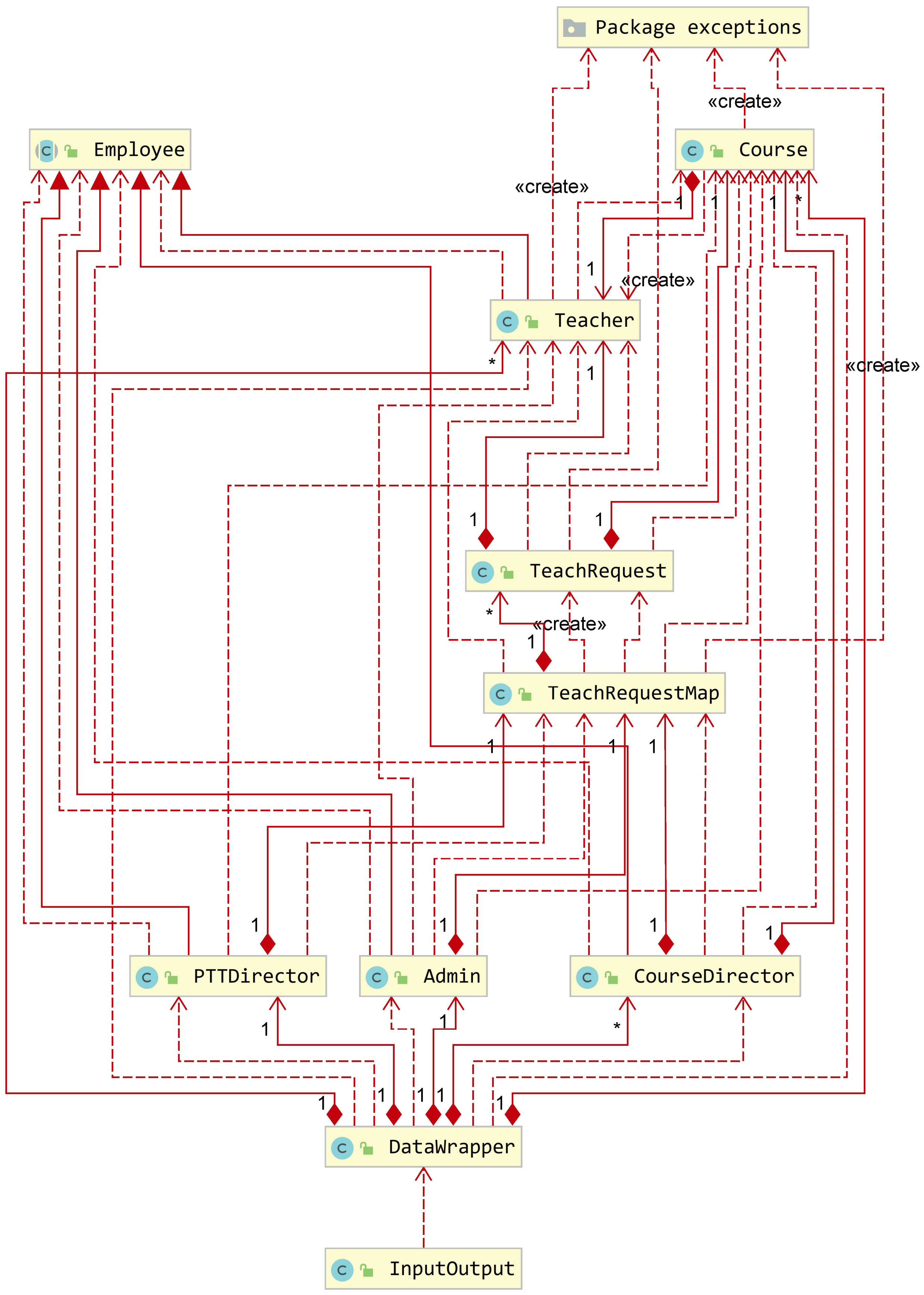


Figure 3.9: Class Diagram. Generated automatically from InteliJ.

3.2 Sequence Diagrams

The User is presented with the main menu, where he can chose if he wishes to navigate as an Admin, PTTDirector or Course Director. All three sub-menus include the options to quit or go back to the main menu, as well as additional options, unique for each role (See Figure X below.)

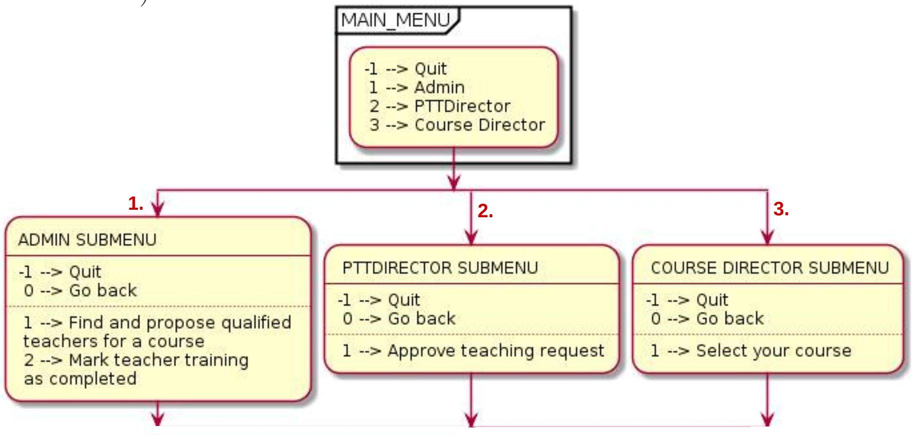


Figure 3.10: Flowchart for the main menu, splitting into 3 distinct sub-menues for each of the users: `Admin, PTTDirector` and `Course Director`.

0. "As a user I can quit the app at any point."

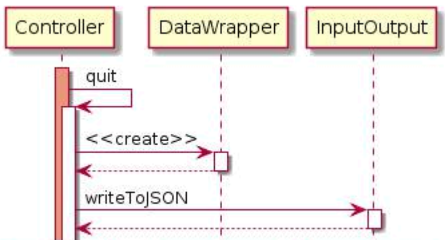


Figure 3.11: Sequence Diagram representing the User Story: *"As a user I can quit the app at any point."*

This quit functionallity is presented as an option at every submenu in the app and for the sake of simplicity will be omitted from the other sequence diagrams.

1. "As a course director I can produce a list of teaching requirements."

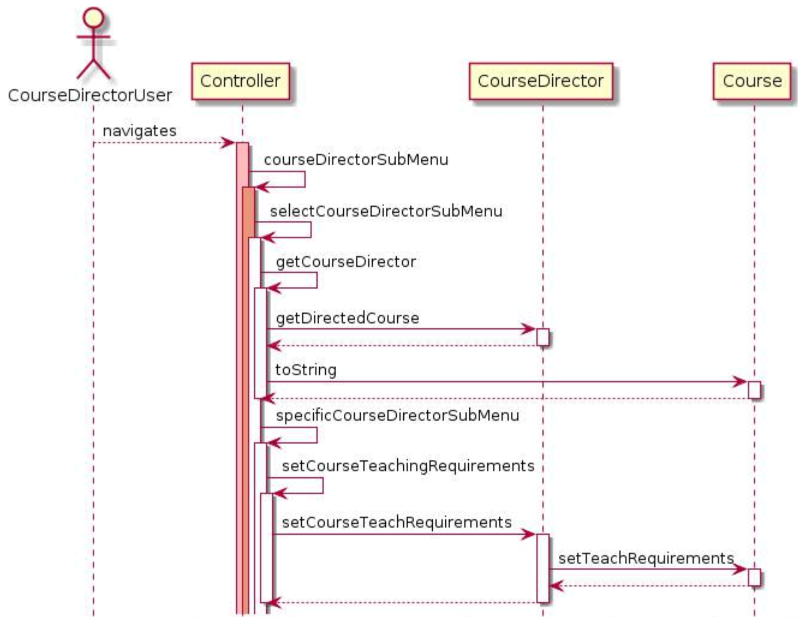


Figure 3.12: Sequence Diagram representing the User Story: *"As a course director I can produce a list of teaching requirements."*

2. "As a course director I can produce a teaching request."

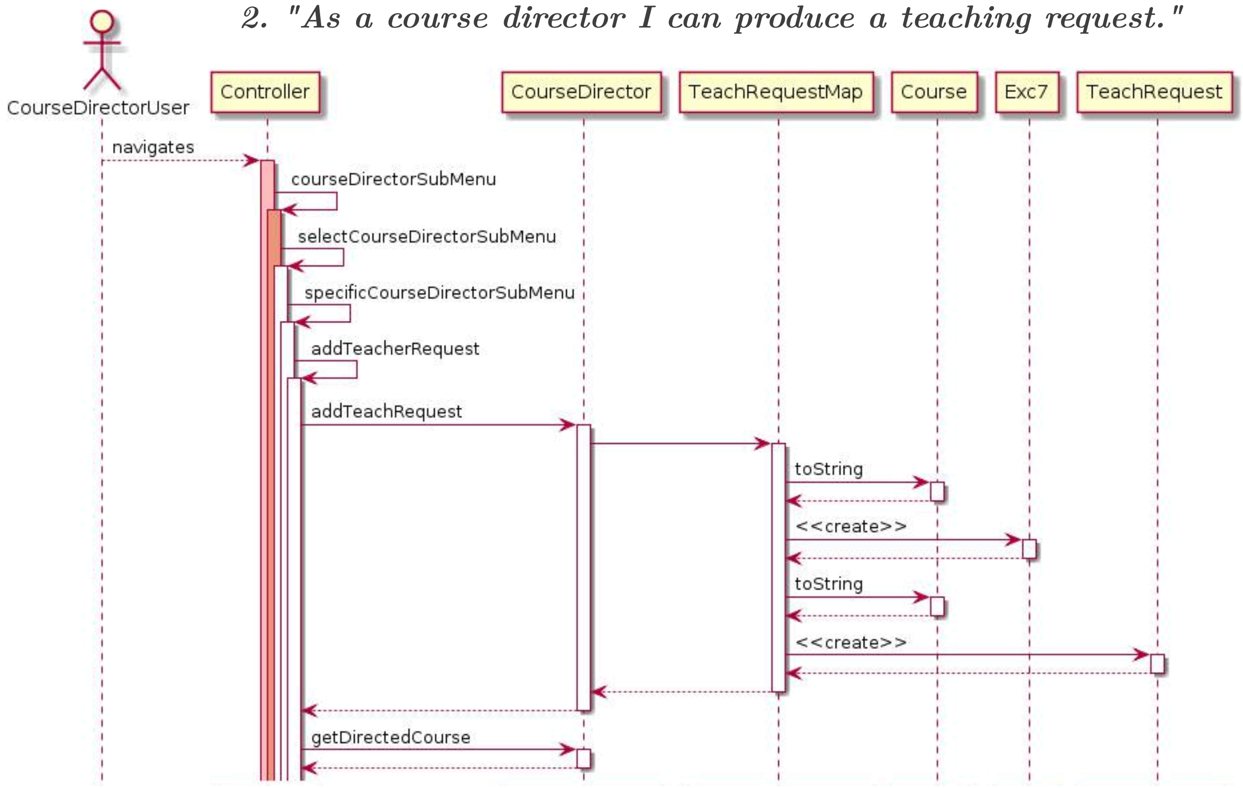


Figure 3.13: Sequence Diagram representing the User Story: *"As a course director I can produce a teaching request."* `Exc7` has been omitted for the sake of simplifying the diagram and corresponds to the `TeachRequestAlreadyGivenException`

3. *"As an Admin I can attempt to find suitable staff."*

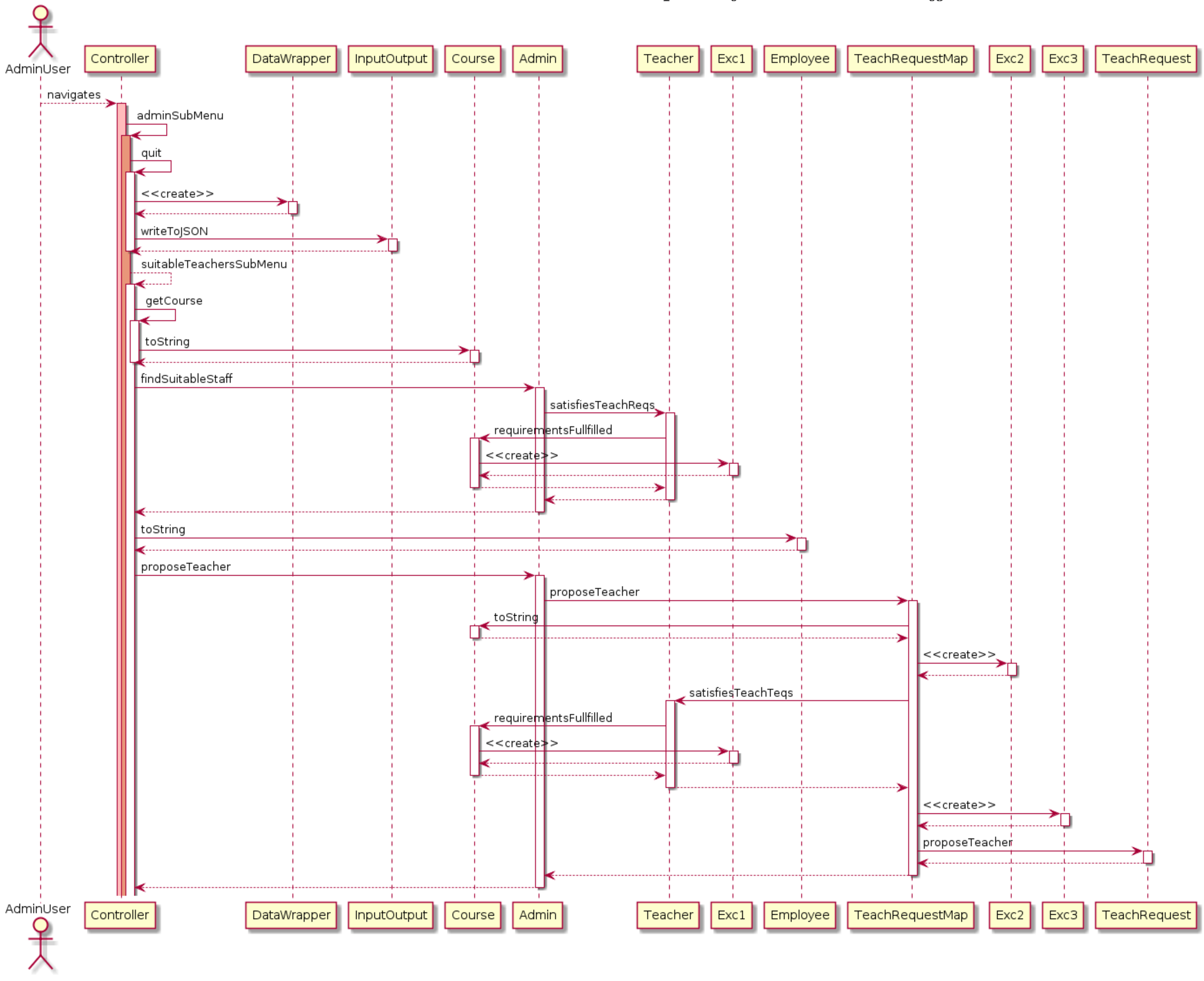


Figure 3.14: Sequence Diagram representing the User Story: *"As an Admin I can attempt to find suitable staff."* In this context 'finding' also includes the proposal of a suitable Teacher. The classes Exc1, Exc2 and Exc3 were abreviated for display purposes and correspond to the following exceptions: `NoTeachRequirementsSetException`, `NonExistentTeachRequestException`, `TeacherNotSuitableForCourseException`

4. "As an admin I can organise training for the suitable staff."

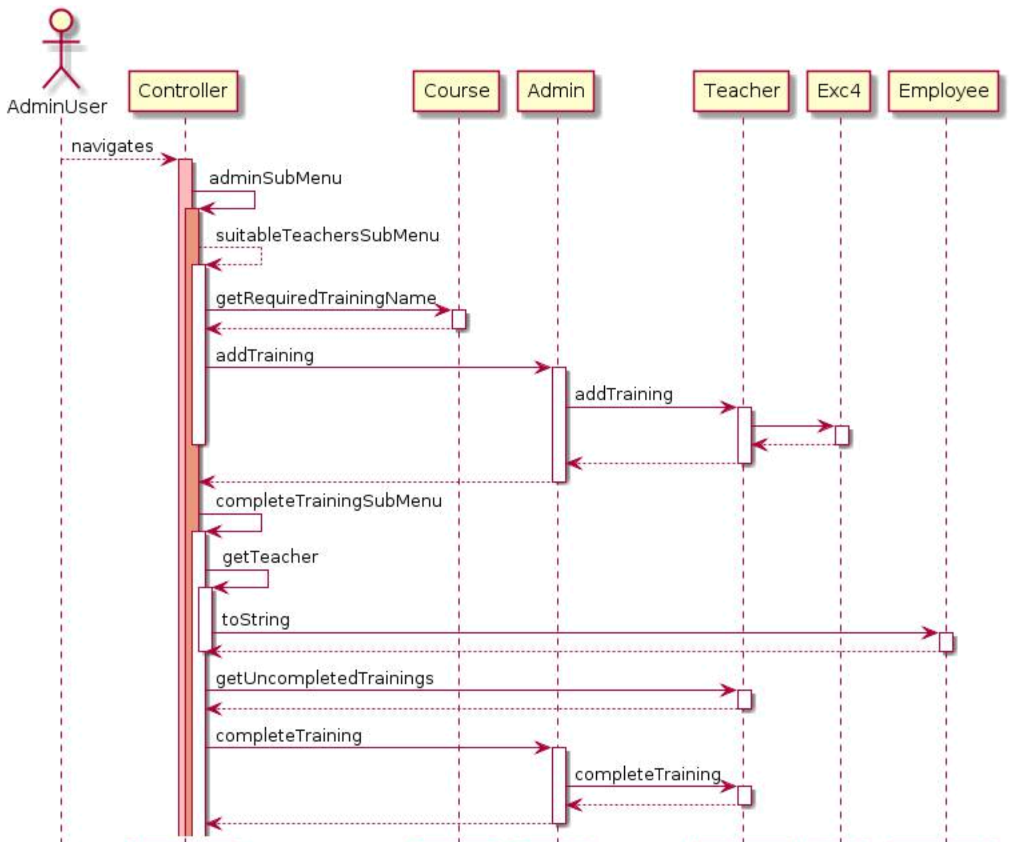


Figure 3.15: Sequence Diagram representing the User Story: *"As an admin I can organise training for the suitable staff."* The class `Exc4` was abbreviated for display purposes and corresponds to `TeacherTrainingAlreadyPresentException`.

5. "As a PTT Director I can approve a teaching request."

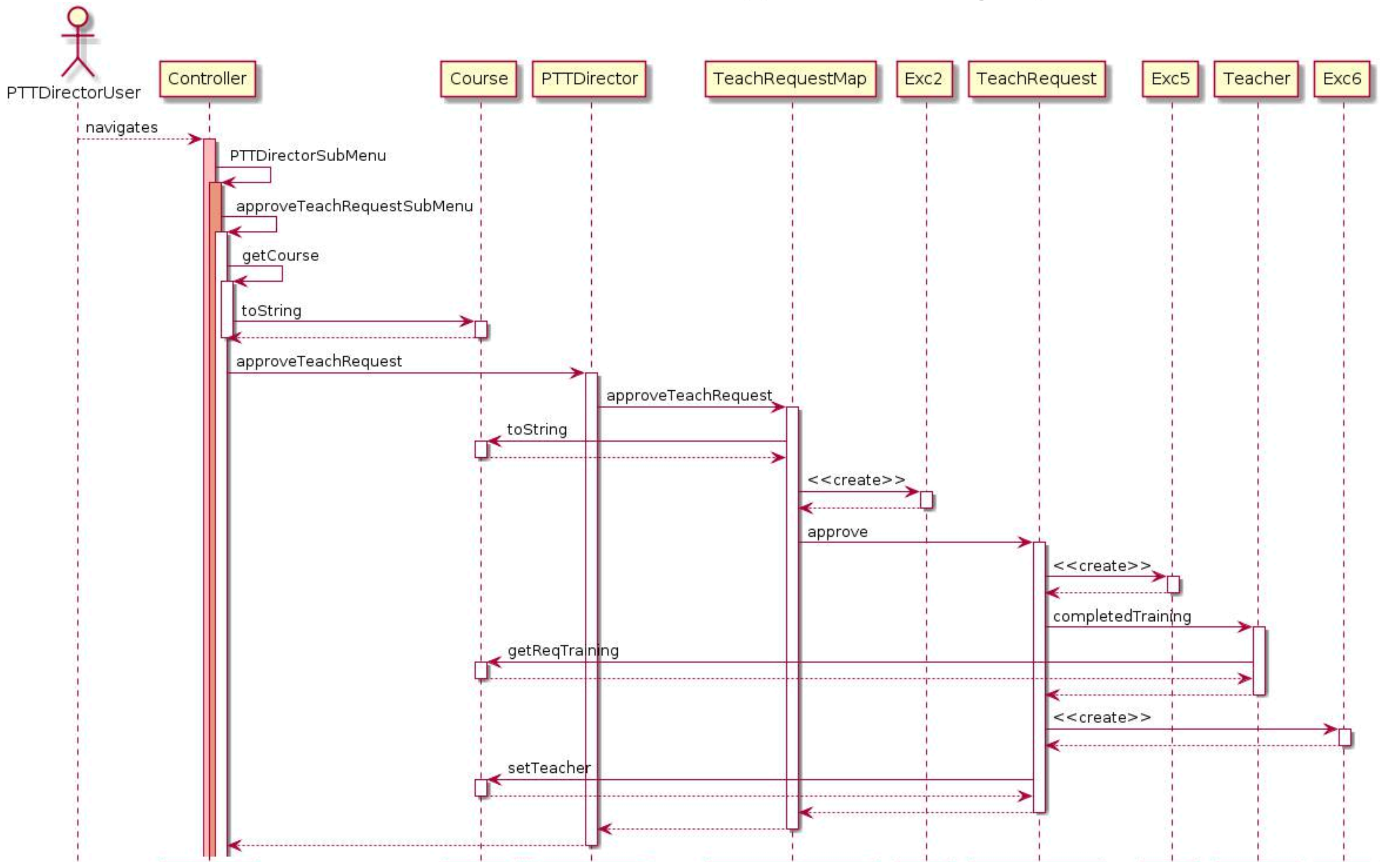


Figure 3.16: Sequence Diagram representing the User Story: *"As a PTT Director I can approve a teaching request."* The classes `Exc2`, `Exc5` and `Exc6` were abbreviated for display purposes and correspond to the following exceptions: `NonExistentTeachRequestException`, `NoProposedTeacherException`, `TeacherNotCompletedTrainingException`

4 | Testing

The team decided to approach testing with both `JUnit` testing and user tests. The latter paid off because it directly tested the functionality of the app and discovered a number of bugs (mostly isolated in the `Controller` class), which were addressed. Some of the discovered bugs had to do with omitting the `nextLine()` method of the `Scanner` class after calling the `nextInt()` method and after incorrect user input for clearing. There were also minor bugs in the `Controller`'s menu logic and a design oversight which allowed a Course director to add multiple skills of the same name within a Course's teaching requirements. The `Controller` class and the `InputOutput` class are tested entirely with user tests and only the mentioned bugs were discovered. The `InputOutput` class' reading and writing functionality only changes the `JSON` file as desired, i.e. only when the program's functionality changes the state of the system.

The `JUnit` tests test only the functionality implemented by the team, i.e. there are no tests for methods that only call built-in Java methods. These are located in the `src/test` folder There are also no tests for methods that only make a single call to a method from another class since the functionality is tested within the corresponding class. For example, the `addTraining` and `completeTraining` methods in the `Admin` class are tested in the `Teacher` testing class instead as the `Admin` class only calls the `Teacher` methods.

More specifically, the following testing approach and results to each user story was applied:

- *"As a user I can quit the app at any point."*

  This was tested with user tests on every stage of the controller.

- *"As a course director I can produce a list of teaching requirements."*

  The course director's add requirements uses a setter method for their directed course and the teaching requirements themselves are represented by a list of Strings. Therefore, all acceptance tests are satisfied by default. The course director can add multiple requirements and can add them as Strings as, again, a list of Strings is used for that purpose.

- *"As a course director I can produce a teaching request for a course."*

  This has been entirely tested with `JUnit` tests in the `TeachRequestMapTest` class.

- *"As an admin I can try to find suitable staff."* A `JUnit` test in the `AdminTest`

  class was created to test scenarios where there are no teaching requirements set. The notification was checked via a user test. Several `JUnit` tests in the `AdminTest` and `CourseTest` classes check whether staff are found only if all teaching requirements are met by the staff's skills. The proposal functionality is tested via the `TeachRequestMapTest` class.

- *"As an admin I can organise training for the suitable staff."*

All add training functionality is tested in via `JUnit` testing in the `TeacherTest` class. The assigned training option is given only to proposed teachers because the user menu's is sequenced so that in order to assign a training, the admin needs to propose the teacher first. This is tested via a user test. The removal of training from uncompleted to completed for a training that has been assigned and the functionality of not doing that for unassigned training is tested again in the `TeacherTest` class with `JUnit` testing.

- *"As a PTT director I can approve a teaching request."*

  All testing has been done using `JUnit` testing in the `TeachRequestMapTest` class.

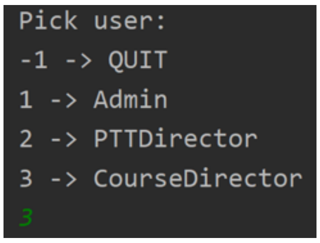
All `JUnit` test and user tests passed successfully.

5 | User Manual

In order to test the implementation, the program has been preloaded with data about the teachers and their respective skills, the courses and the course directors. This is explained in the next section.

All menus, contain the options to either quit or go back which the user can utilise by typing either `"-1"` or `"0"` respectively. The user input of `"-1"` results in the writing of all necessary information to a `JSON` file, whereas `"0"` simply takes them back to the previous menu. Unsuccessful actions are handled accordingly with the program's exceptions, but for the sake of simplicity, these are not displayed in the manual below.

The program's first menu allows the user to identify as either an Administrator, Course Director or PTT Director by pressing `1,2` or `3` respectively. Each of those redirect to additional sub-menus which have unique options for actions.



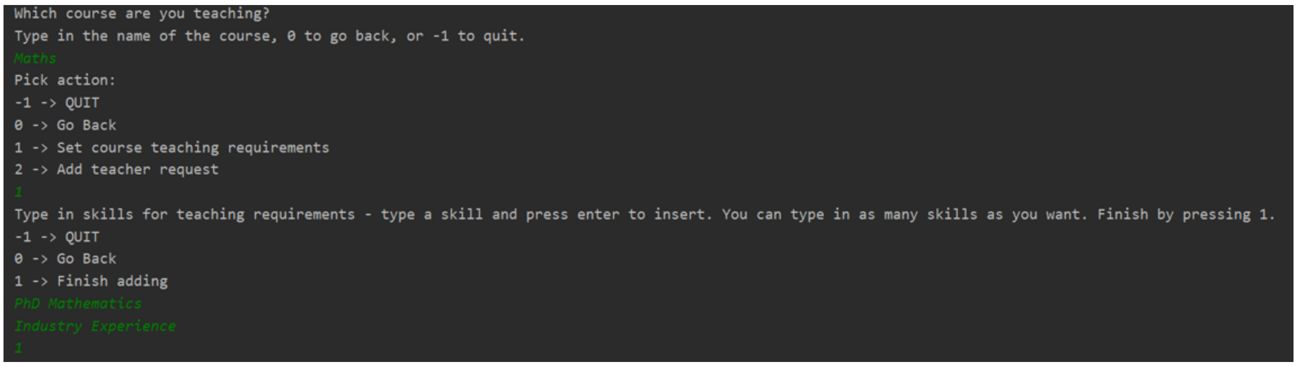
```
Pick user:
-1 -> QUIT
1 -> Admin
2 -> PTTDirector
3 -> CourseDirector
3
```

Figure 5.1: Main Menu, allowing the user to select their role by pressing 1, 2 or 3.

The Course Director Path

Following the selection of `option 3` from the main menu, the user now needs to type in the exact name of the course they direct in order to be identified. After this, they have 2 options for action:

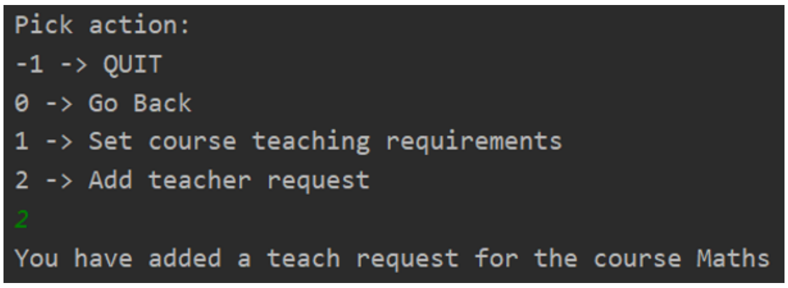
1. **Set the course teaching requirements** (which need to be contained in the skills list of a teacher in order for the teacher to be qualified to teach the course). This is done by typing each skill and then pressing Enter for however many skills are required. The process is ended by 1 + Enter .



```
Which course are you teaching?
Type in the name of the course, 0 to go back, or -1 to quit.
Maths
Pick action:
-1 -> QUIT
0 -> Go Back
1 -> Set course teaching requirements
2 -> Add teacher request
1
Type in skills for teaching requirements - type a skill and press enter to insert. You can type in as many skills as you want. Finish by pressing 1.
-1 -> QUIT
0 -> Go Back
1 -> Finish adding
PhD Mathematics
Industry Experience
1
```

Figure 5.2: Course Director Submenu: 1. Setting the course teaching requirements functionality.

2. **Add a teach request for the course he directs** from the same course director menu. This is done by pressing 2 + Enter . The program will display a confirmation message notifying that the user has added a teach request for the course.



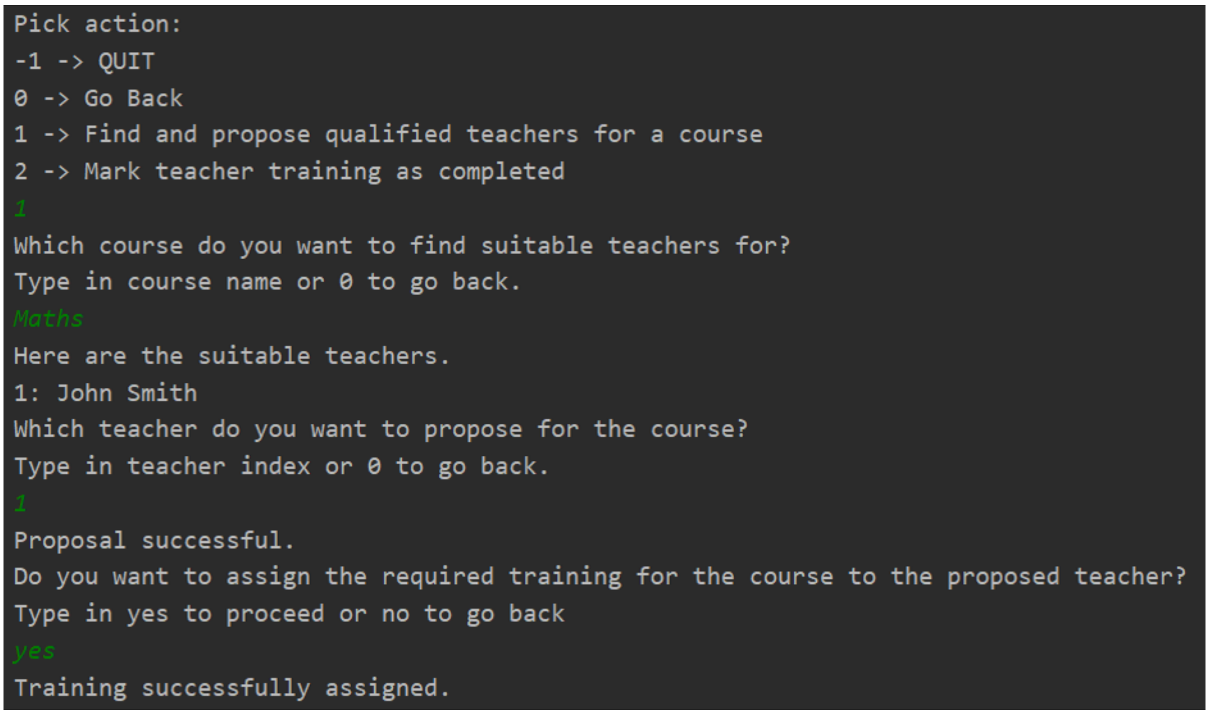
```
Pick action:
-1 -> QUIT
0 -> Go Back
1 -> Set course teaching requirements
2 -> Add teacher request
2
You have added a teach request for the course Maths
```

Figure 5.3: Course Director Submenu: 2. Adding a teach request functionality.

The Administrator Path

Following the selection of `option 1` from the main menu, the user is presented with the following options:

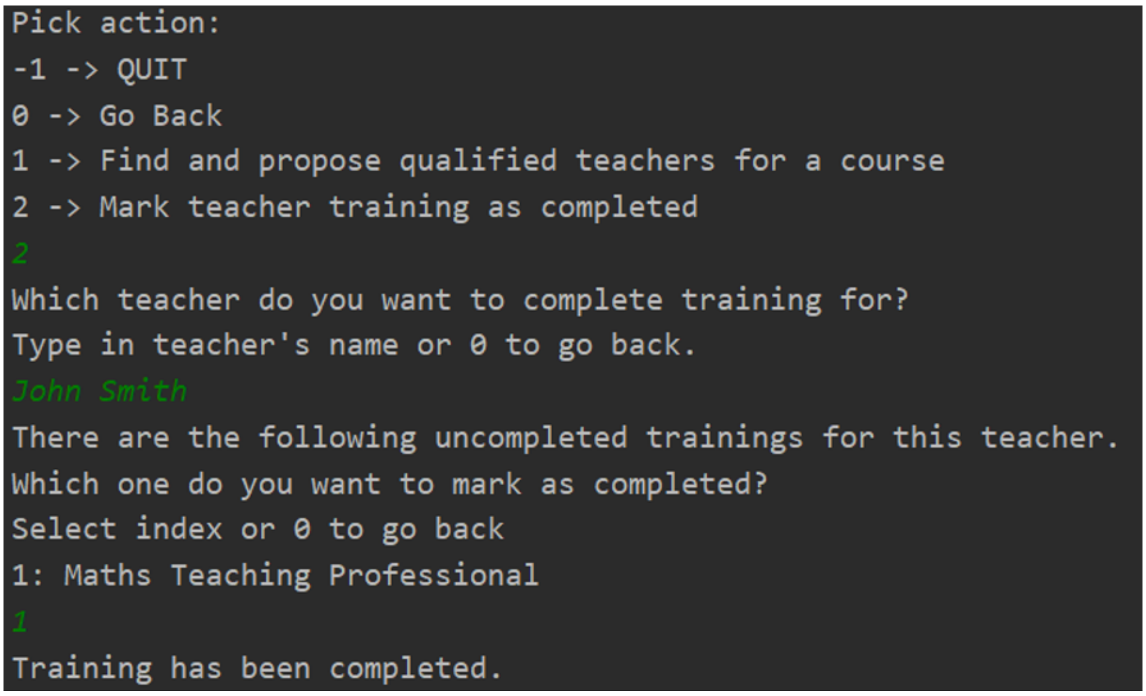
1. **Find and propose qualified teachers for a given course** by pressing 1 + Enter . The user will then be asked to type in the course name and after pressing Enter , the suitable teachers will be displayed, along with an index (See Figure 4.4 below). The program will then prompt for the selection of one of those teachers for the proposal by entering the index. Finally, the user is asked if they wish to assign the required training for the course to the proposed teacher. The user can either say “yes” or "no". If “yes” was inserted, the program will display a confirmation message notifying the user that the training has been assigned.



```
Pick action:
-1 -> QUIT
0 -> Go Back
1 -> Find and propose qualified teachers for a course
2 -> Mark teacher training as completed
1
Which course do you want to find suitable teachers for?
Type in course name or 0 to go back.
Maths
Here are the suitable teachers.
1: John Smith
Which teacher do you want to propose for the course?
Type in teacher index or 0 to go back.
1
Proposal successful.
Do you want to assign the required training for the course to the proposed teacher?
Type in yes to proceed or no to go back
yes
Training successfully assigned.
```

Figure 5.4: Administrator Submenu: 1. Finding and proposing qualified teachers functionality.

2. **Mark a teacher training as complete**. This is done by selecting 2 + Enter in the admin menu. The program will then ask the user to select a teacher which should be typed in, followed by pressing Enter . All uncompleted trainings will then be displayed, along with an index for selection which the user could use. Finally, a confirmation message will notify the user that the training has been completed.



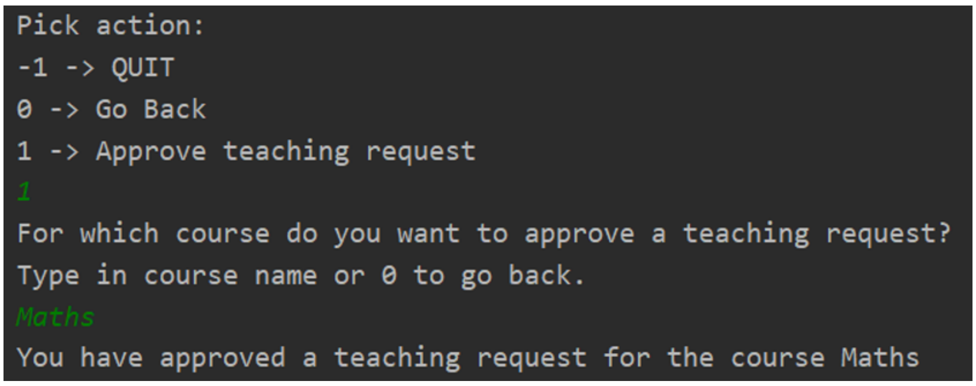
```
Pick action:
-1 -> QUIT
0 -> Go Back
1 -> Find and propose qualified teachers for a course
2 -> Mark teacher training as completed
2
Which teacher do you want to complete training for?
Type in teacher's name or 0 to go back.
John Smith
There are the following uncompleted trainings for this teacher.
Which one do you want to mark as completed?
Select index or 0 to go back
1: Maths Teaching Professional
1
Training has been completed.
```

Figure 5.5: Administrator Submenu: 2. Marking a teacher training as complete functionality.

The PTT Director Path

Following the selection of `option 2` from the main menu, the user is presented with the following options:

1. **Approve a teaching request** by pressing 1 + Enter . The program will then ask which course to approve a teaching request for. The user can then insert the name of the course followed by pressing Enter . A confirmation message will then be displayed to confirm the approval.

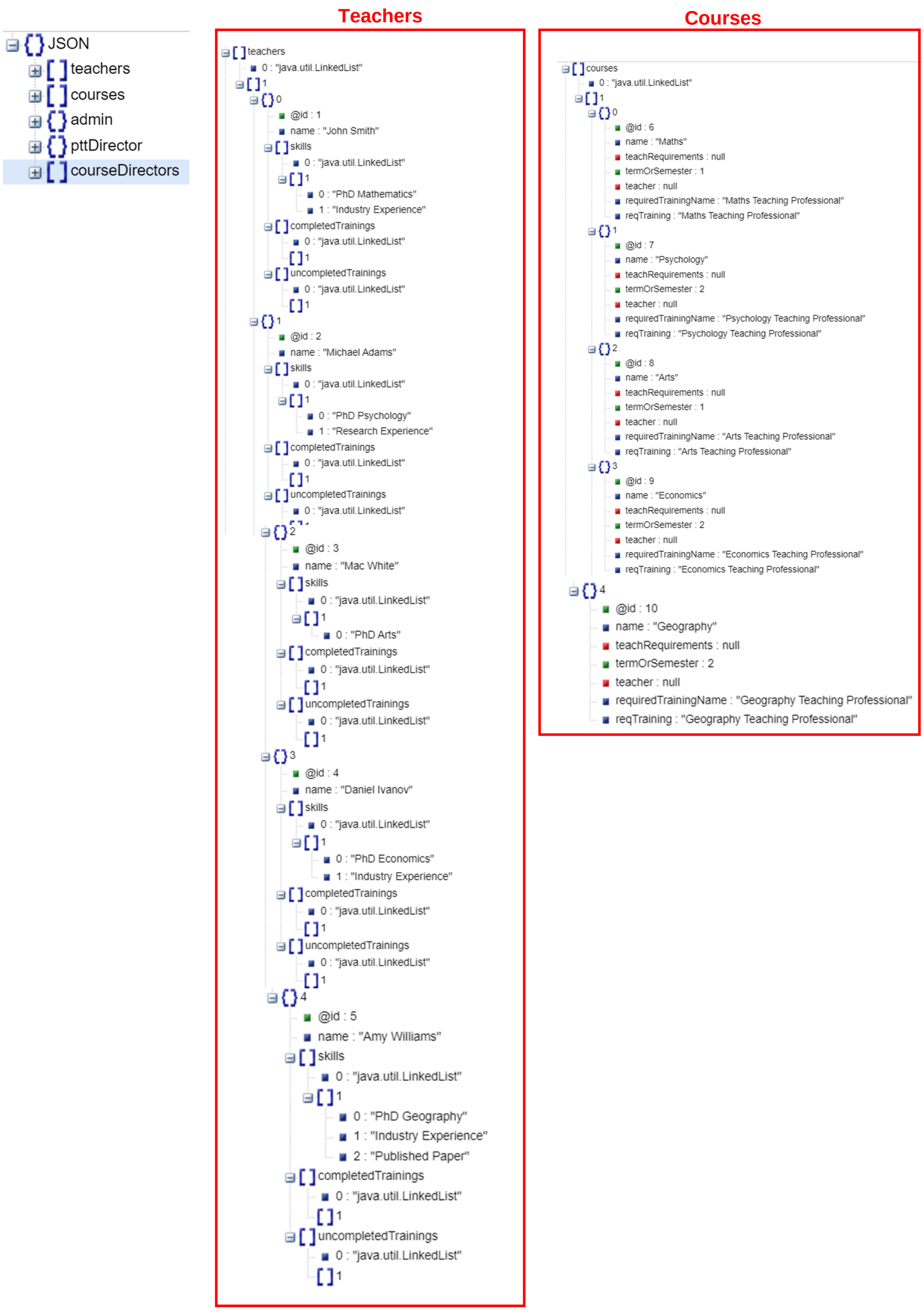


```
Pick action:
-1 -> QUIT
0 -> Go Back
1 -> Approve teaching request
1
For which course do you want to approve a teaching request?
Type in course name or 0 to go back.
Maths
You have approved a teaching request for the course Maths
```

Figure 5.6: PTT Director Submenu: 1. Approving a teach request functionality.

6 | JSON File

The following figures illustrate the initial information written in the `JSON` file. The sample data from the section above was used to populate the file. `Teachers` and `Courses` are further expanded to show the details needed in order to use the program.



```
JSON
  teachers
  courses
  admin
  pttDirector
  courseDirectors
```

Teachers

```
teachers
  0 : "java.util.LinkedList"
  1
    0
      @id : 1
      name : "John Smith"
      skills
        0 : "java.util.LinkedList"
        1
          0 : "PhD Mathematics"
          1 : "Industry Experience"
      completedTrainings
        0 : "java.util.LinkedList"
        1
      uncompletedTrainings
        0 : "java.util.LinkedList"
        1
    1
      @id : 2
      name : "Michael Adams"
      skills
        0 : "java.util.LinkedList"
        1
          0 : "PhD Psychology"
          1 : "Research Experience"
      completedTrainings
        0 : "java.util.LinkedList"
        1
      uncompletedTrainings
        0 : "java.util.LinkedList"
    2
      @id : 3
      name : "Mac White"
      skills
        0 : "java.util.LinkedList"
        1
          0 : "PhD Arts"
      completedTrainings
        0 : "java.util.LinkedList"
        1
      uncompletedTrainings
        0 : "java.util.LinkedList"
        1
    3
      @id : 4
      name : "Daniel Ivanov"
      skills
        0 : "java.util.LinkedList"
        1
          0 : "PhD Economics"
          1 : "Industry Experience"
      completedTrainings
        0 : "java.util.LinkedList"
        1
      uncompletedTrainings
        0 : "java.util.LinkedList"
        1
    4
      @id : 5
      name : "Amy Williams"
      skills
        0 : "java.util.LinkedList"
        1
          0 : "PhD Geography"
          1 : "Industry Experience"
          2 : "Published Paper"
      completedTrainings
        0 : "java.util.LinkedList"
        1
      uncompletedTrainings
        0 : "java.util.LinkedList"
        1
```

Courses

```
courses
  0 : "java.util.LinkedList"
  1
    0
      @id : 6
      name : "Maths"
      teachRequirements : null
      termOrSemester : 1
      teacher : null
      requiredTrainingName : "Maths Teaching Professional"
      reqTraining : "Maths Teaching Professional"
    1
      @id : 7
      name : "Psychology"
      teachRequirements : null
      termOrSemester : 2
      teacher : null
      requiredTrainingName : "Psychology Teaching Professional"
      reqTraining : "Psychology Teaching Professional"
    2
      @id : 8
      name : "Arts"
      teachRequirements : null
      termOrSemester : 1
      teacher : null
      requiredTrainingName : "Arts Teaching Professional"
      reqTraining : "Arts Teaching Professional"
    3
      @id : 9
      name : "Economics"
      teachRequirements : null
      termOrSemester : 2
      teacher : null
      requiredTrainingName : "Economics Teaching Professional"
      reqTraining : "Economics Teaching Professional"
    4
      @id : 10
      name : "Geography"
      teachRequirements : null
      termOrSemester : 2
      teacher : null
      requiredTrainingName : "Geography Teaching Professional"
      reqTraining : "Geography Teaching Professional"
```

Figure 6.1: jsonSpecs

7 | Instructions for Testing

Knowledge of the names and skills of the teachers is necessary to test the program. The given teachers have no trainings set(completed or uncompleted). Trainings are added by the `Admin menu` when the user is proposing a teacher for an existing teaching request from a given list of suitable teachers.

Knowledge of the names and teaching requirements of the courses is also necessary to test the program. The given courses have no teaching requirements initially set, this can be done via the `Course Director menu`. The teaching requirements need to be matched by a subset of a teacher's skills so that they can be considered suitable for a course. The given courses also have no teacher initially set. This can be done by using the `Course Director menu` to set teaching requirements and make a teaching request. Then by using the `Admin menu` to find suitable staff, propose a teacher among them, add the required training and complete it. Finally, the teaching request must be approved by the `PTTDirector menu`.

8 | Retrospective

Overall, the objectives of this project were achieved with a full implementation of all User Stories and no known bugs or errors.

Following the specification of the app, the team was able to design a class structure in which the separate classes interact with each other and the overall design fulfills the requirements. The recommended tools of agile development for the analysis and design were used for the development of the software. Going through this process also helped the team to undergo a full life-cycle of a software engineering project, from initial gathering and analysis of requirements, through testing and development, to produce a final product.

The planning stage of the project focused on the class architecture and overall skeleton of the project. This ensured the implementation of the app was smooth without any major setbacks. As for the team management, the self-organisation was excellent, with frequent meetings attendance, great work ethic and overall equal contribution towards the final project. Furthermore, Trello allowed the team to stay organized and the agile approach and User Stories helped the overall team management.

As described in more detail in corresponding chapter, the testing part of the app development cycle went very well with both JUnit and user tests used. By discovering a number of bugs, testing with overall wide coverage helped to achieve a program with full functionality.

There were some challenges the team had to overcome. Firstly, the open and general definition of the project's scope meant that the team had to put extra effort to ensure a shared understanding of requirements and be able to set an end goal. Secondly, during the project, the JSON file reading was returning a null object value since the initial class design had a circular reference between `Teacher` and `Course`. That is why some design changes in the class structure had to be made during the project, which resulted in a time delay and more actual effort put into completing the `InputOutput` and `DataWrapper` classes. It is worth noting that the structure of the project was inspired by the `Model-View-Controller` pattern, but as the project only used a command line output, the `View` class was omitted for practical reasons. If the program is changed in the future to a `Swing` version, this can easily be adapted with a new view class. The same goes for incorporating database functionality which at the moment is not present, but could easily be added.

To conclude, this project allowed the team to experience and practice Agile development tools and techniques as well as to gain bigger insight and knowledge in class architecture design, patterns and file reading and writing.

9 | Team Roles and Contributions

The work on the app development and documentation was distributed equally between all members of the Slavlings team in the following way:

- Nadezhda Dimitrova
  - Class Employee (task 1)
  - Class Teacher (tasks 2-6)
  - Class TeacherTest
  - Class InputOutput (task 21)
  - Report - UML, Sequence Diagrams
  - refractoring
- Ventsislav Antov
  - Class Admin (tasks 11-14)
  - Class AdminTest
  - Class DataWrapper (task 21)
  - task 25 (classDirectorMenu) in Controller
  - Report - User Manual, Testing Instructions, JSON file, Retrospective
  - testing
- Tereza Buckova
  - Class Course (tasks 7-8)
  - Class CourseTest
  - Class CourseDirector (tasks 9-10)
  - Class Controller (tasks are split between all members)
  - task 24 (pttDirectorMenu) in Controller
  - Report - User Stories, Ideal/Actual effort
- Filip Marinov
  - Class PTT Director (task 15)
  - Class TeachRequest (task 16-17)
  - Class TeachRequestTest
  - Class TeachRequestMap (18-19)
  - Class TeachRequestMapTest
  - Associated Exceptions
  - task 23 (adminMenu) in Controller

  - testing
  - Report - Retrospective, Instructions for Testing, User Manual

The full description of all tasks can be found in Chapter 3.

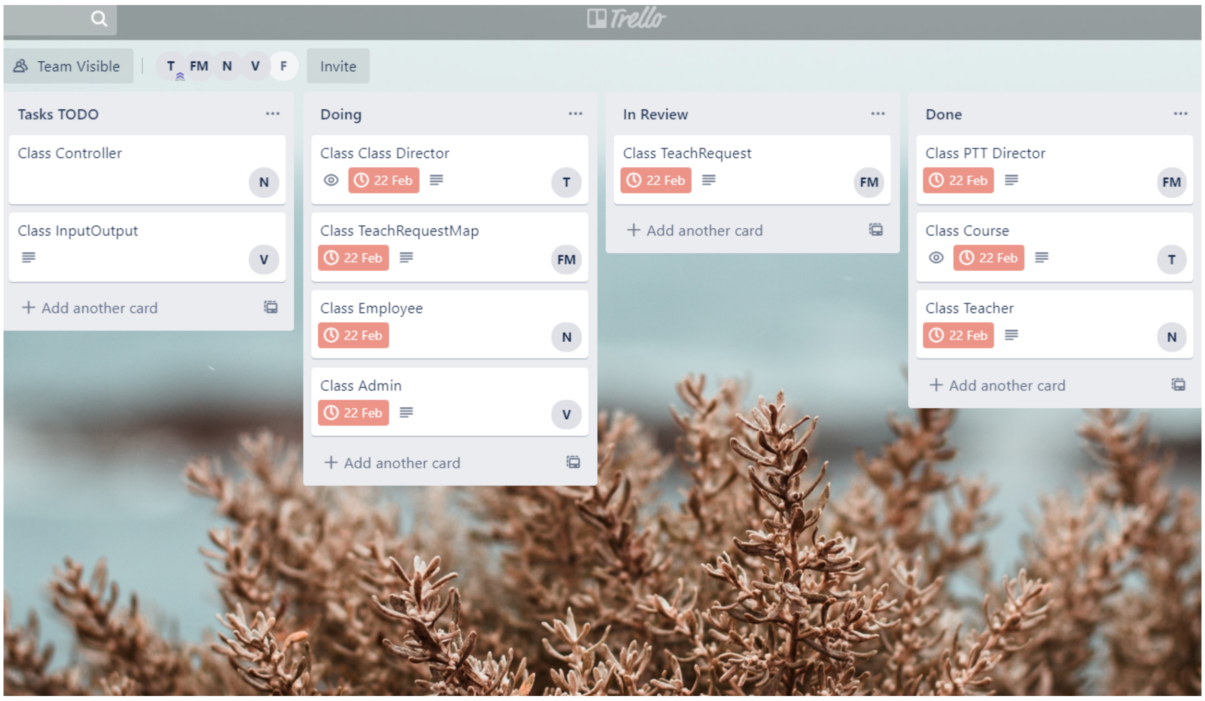
10 | Conclusion

The team was able to successfully complete the project within the given timeframe and to produce a working solution given the project requirements. The project utilised the Agile methodology and was able to plan out the project and to adapt accordingly in order to maximise efficiency.

The team encountered some difficulties, but managed to solve them all and left no known bugs in the final version of the project. The most significant obstacle was the `JSON` file reading which was returning a null object value.

If given more time, the project could expand into building a `Swing` interface instead of the current command line version. There are also other possible additions in terms of functionality such as a log-in function that could be added, but not part of the project requirements.

A | Trello


Trello

Team Visible | T FM N V F | Invite

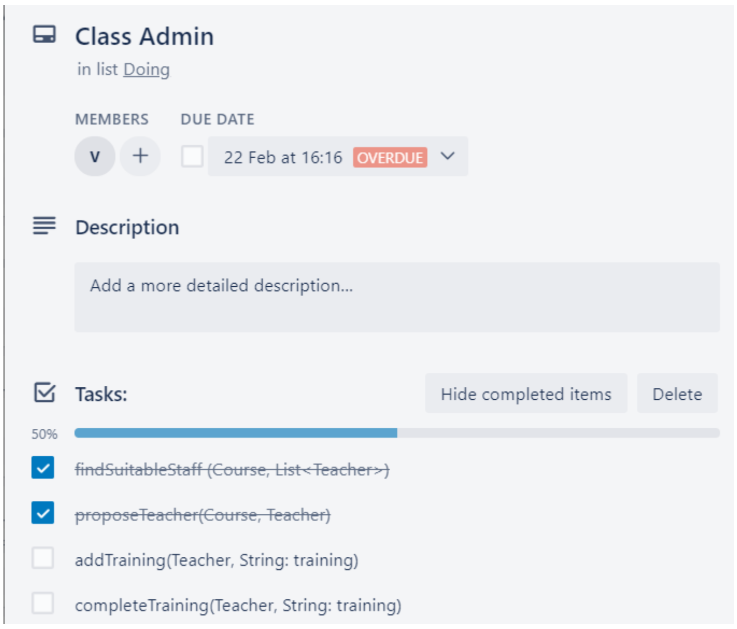
Tasks TODO
- Class Controller — N
- Class InputOutput — V
- + Add another card

Doing
- Class Class Director — 22 Feb — T
- Class TeachRequestMap — 22 Feb — FM
- Class Employee — 22 Feb — N
- Class Admin — 22 Feb — V
- + Add another card

In Review
- Class TeachRequest — 22 Feb — FM
- + Add another card

Done
- Class PTT Director — 22 Feb — FM
- Class Course — 22 Feb — T
- Class Teacher — 22 Feb — N
- + Add another card

Figure A.1: Tasks on Trello


Class Admin

in list Doing

MEMBERS: V +

DUE DATE: 22 Feb at 16:16 OVERDUE

Description

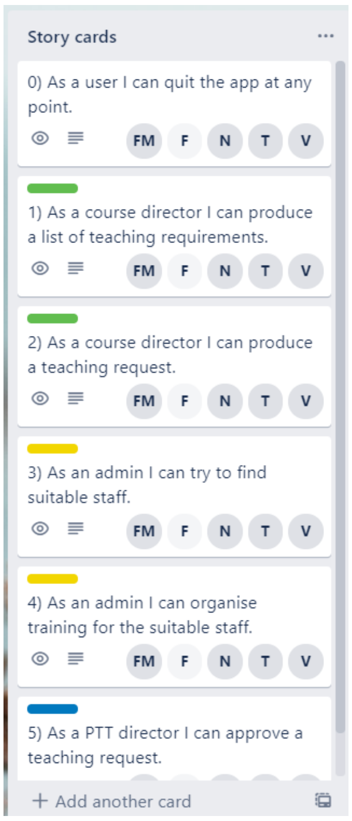
Add a more detailed description...

Tasks: Hide completed items | Delete

50%

- [x] ~~findSuitableStaff (Course, List<Teacher>)~~
- [x] ~~proposeTeacher(Course, Teacher)~~
- [ ] addTraining(Teacher, String: training)
- [ ] completeTraining(Teacher, String: training)

Figure A.2: Developer tasks for Admin



Story cards

0) As a user I can quit the app at any point.

FM F N T V

1) As a course director I can produce a list of teaching requirements.

FM F N T V

2) As a course director I can produce a teaching request.

FM F N T V

3) As an admin I can try to find suitable staff.

FM F N T V

4) As an admin I can organise training for the suitable staff.

FM F N T V

5) As a PTT director I can approve a teaching request.

+ Add another card

Figure A.3: User story cards on Trello