2/29/2020

HAPPY HIPPOS

SE – ASSESSED EXERCISE – HAPPY HIPPOS

Liam Krupej – 2026288K

Julia Saari – 2207399S

Gareth Sears - 2493194S

Hugh Winchester – 2494047W

Contents

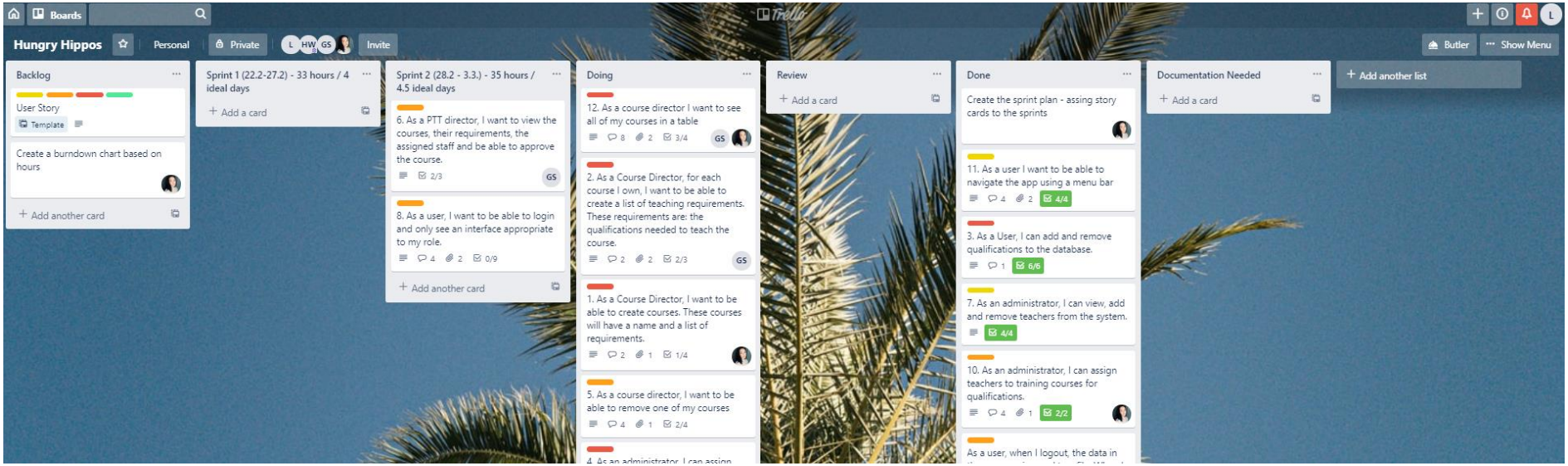
Members' Contributions

Names	Details of Contributions
Gareth Sears	Table Package, Database Package, Repository Setup, POM file, TeacherTableModel, Code Reviews, Sequence Diagrams
Hugh Winchester	Sequence Diagrams, AppMenu, AppController, AppView, HomePage, Logger, User (ENUM), Code Reviews
Liam Krupej	CourseTableModel, Sequence Diagrams, Report writing, User Stories , Code Reviews
Julia Saari	Course, Qualification, Training, Teacher, TrainingTableModel, QualificationTableModel, Sequence Diagrams, Code Reviews

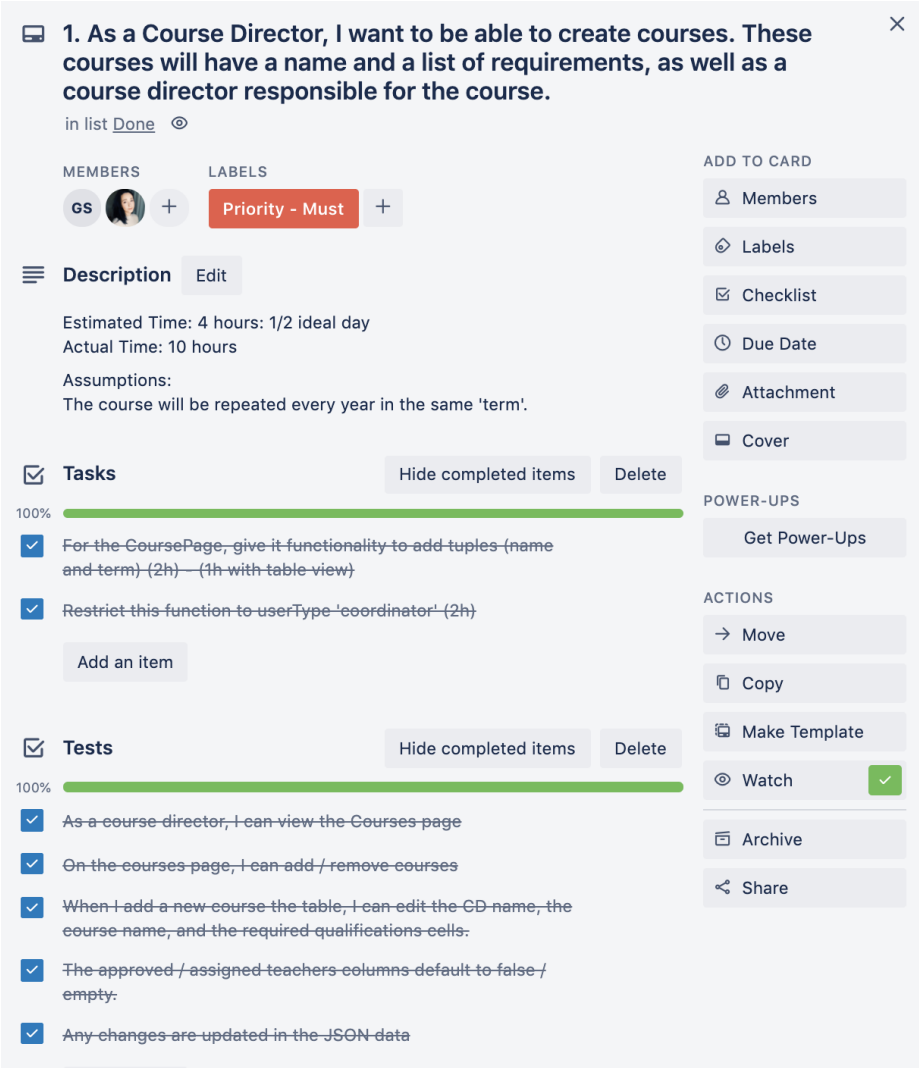
User Stories

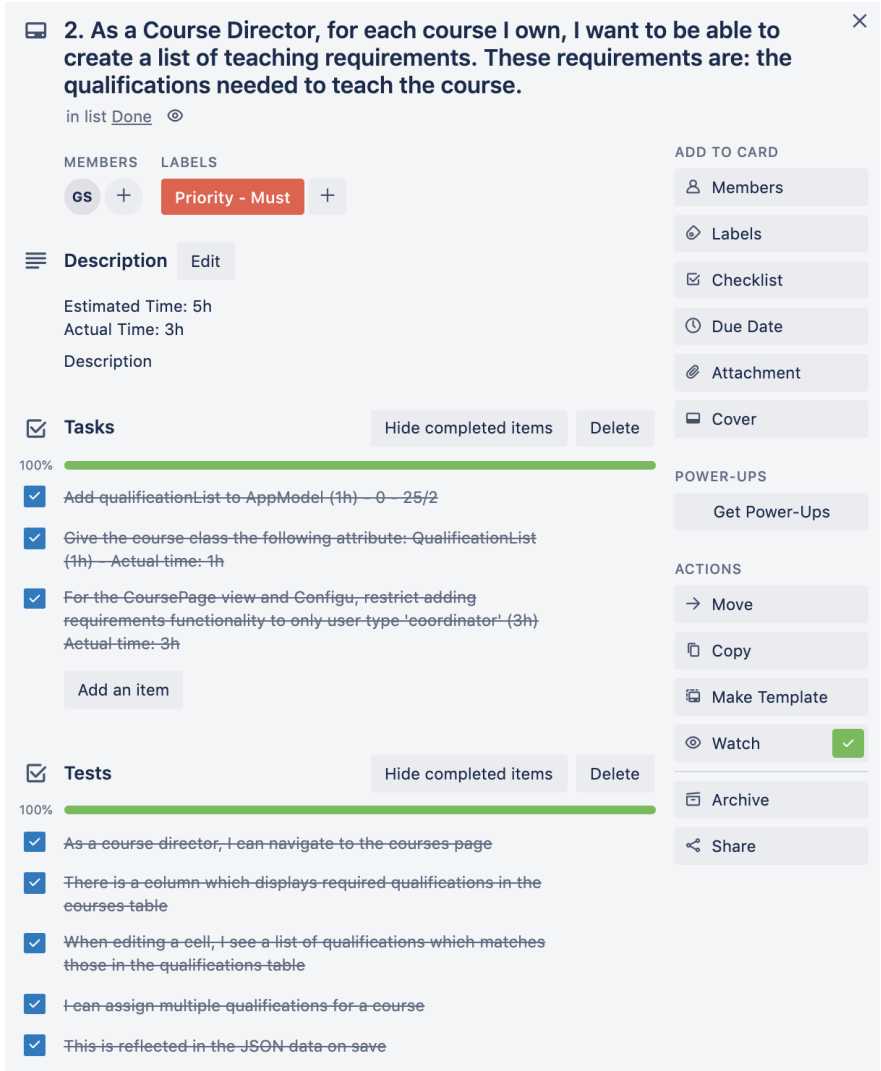
Below is an example of our Trello board towards the end of the project (displayed to show our choice of stages and how our tasks have progressed). We used seven categories to track the progress of our tasks:

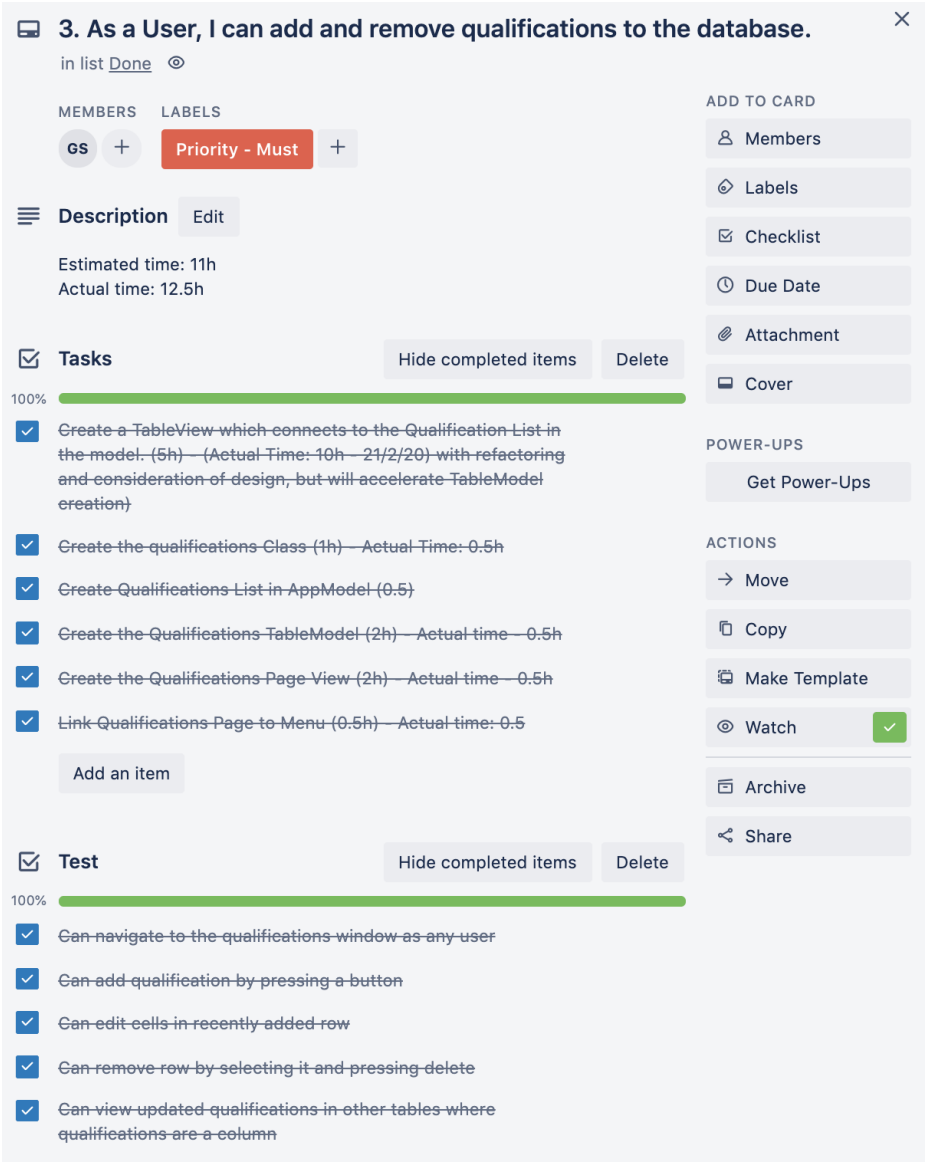
1. Backlog
2. Sprint 1
3. Sprint 2
4. Doing
5. Review
6. Done
7. Documentation Needed



The rest of our story cards are laid out below. Each card includes a header; a priority; an estimated time; a checklist of tasks necessary to fulfil the requirements of the story; attached diagrams; a set of tests; an owner, or multiple owners; and a conversation at the bottom.


1. As a Course Director, I want to be able to create courses. These courses will have a name and a list of requirements, as well as a course director responsible for the course.
in list Done
MEMBERS
LABELS
GS
Priority - Must
Description
Edit
Estimated Time: 4 hours: 1/2 ideal day
Actual Time: 10 hours
Assumptions:
The course will be repeated every year in the same 'term'.
Tasks
Hide completed items
Delete
100%
~~For the CoursePage, give it functionality to add tuples (name and term) (2h) - (1h with table view)~~
~~Restrict this function to userType 'coordinator' (2h)~~
Add an item
Tests
Hide completed items
Delete
100%
~~As a course director, I can view the Courses page~~
~~On the courses page, I can add / remove courses~~
~~When I add a new course the table, I can edit the CD name, the course name, and the required qualifications cells.~~
~~The approved / assigned teachers columns default to false / empty.~~
~~Any changes are updated in the JSON data~~
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share


2. As a Course Director, for each course I own, I want to be able to create a list of teaching requirements. These requirements are: the qualifications needed to teach the course.
in list Done
MEMBERS
LABELS
GS
Priority - Must
Description
Edit
Estimated Time: 5h
Actual Time: 3h
Description
Tasks
Hide completed items
Delete
100%
~~Add qualificationList to AppModel (1h) - 0 - 25/2~~
~~Give the course class the following attribute: QualificationList (1h) - Actual time: 1h~~
~~For the CoursePage view and Configu, restrict adding requirements functionality to only user type 'coordinator' (3h) Actual time: 3h~~
Add an item
Tests
Hide completed items
Delete
100%
~~As a course director, I can navigate to the courses page~~
~~There is a column which displays required qualifications in the courses table~~
~~When editing a cell, I see a list of qualifications which matches those in the qualifications table~~
~~I can assign multiple qualifications for a course~~
~~This is reflected in the JSON data on save~~
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share



3. As a User, I can add and remove qualifications to the database.

in list Done

MEMBERS GS + LABELS Priority - Must +

Description Edit

Estimated time: 11h
Actual time: 12.5h

Tasks Hide completed items Delete

100%

- ~~Create a TableView which connects to the Qualification List in the model. (5h) - (Actual Time: 10h - 21/2/20) with refactoring and consideration of design, but will accelerate TableModel creation)~~
- ~~Create the qualifications Class (1h) - Actual Time: 0.5h~~
- ~~Create Qualifications List in AppModel (0.5)~~
- ~~Create the Qualifications TableModel (2h) - Actual time - 0.5h~~
- ~~Create the Qualifications Page View (2h) - Actual time - 0.5h~~
- ~~Link Qualifications Page to Menu (0.5h) - Actual time: 0.5~~

Add an item

Test Hide completed items Delete

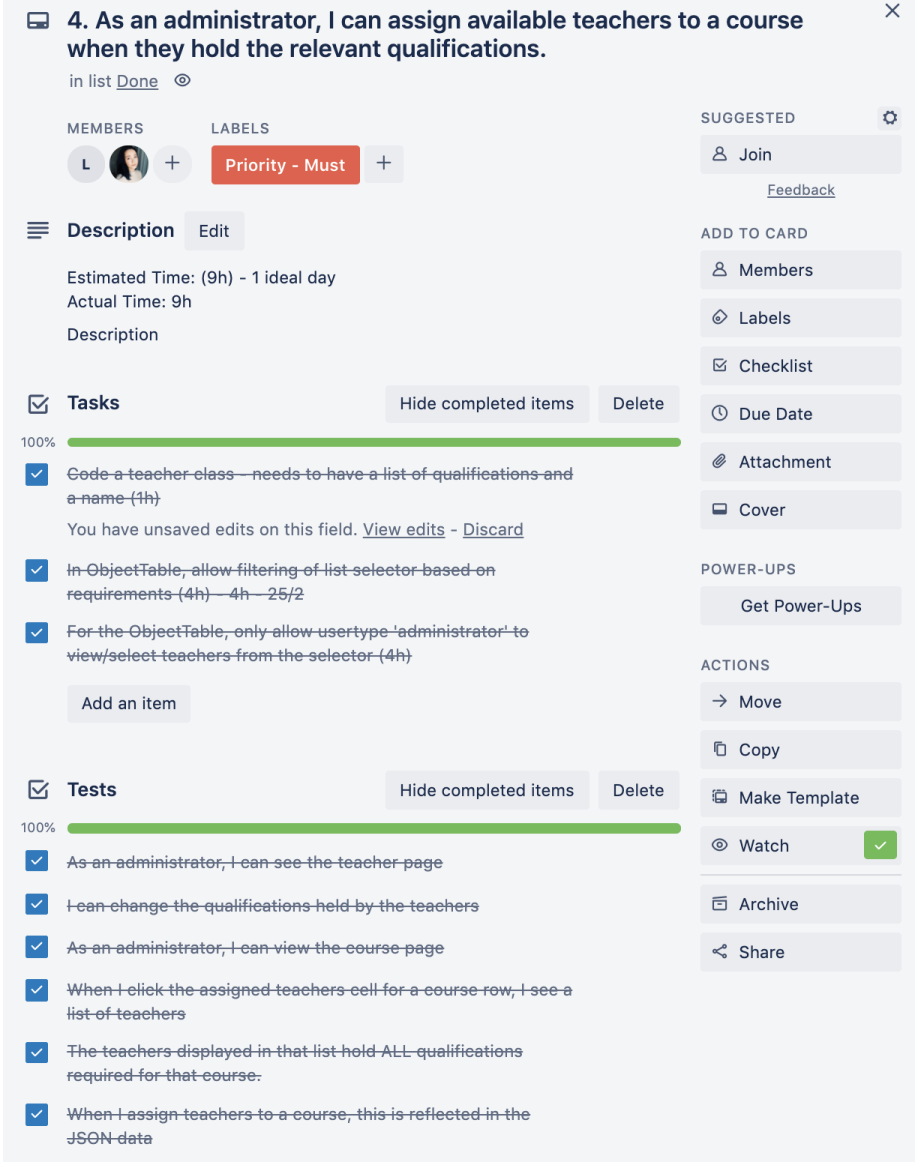
100%

- ~~Can navigate to the qualifications window as any user~~
- ~~Can add qualification by pressing a button~~
- ~~Can edit cells in recently added row~~
- ~~Can remove row by selecting it and pressing delete~~
- ~~Can view updated qualifications in other tables where qualifications are a column~~

ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover

POWER-UPS
Get Power-Ups

ACTIONS
Move
Copy
Make Template
Watch
Archive
Share



4. As an administrator, I can assign available teachers to a course when they hold the relevant qualifications.

in list Done

MEMBERS L + LABELS Priority - Must +

Description Edit

Estimated Time: (9h) - 1 ideal day
Actual Time: 9h

Description

Tasks Hide completed items Delete

100%

- ~~Code a teacher class - needs to have a list of qualifications and a name (1h)~~

You have unsaved edits on this field. View edits - Discard

- ~~In ObjectTable, allow filtering of list selector based on requirements (4h) - 4h - 25/2~~
- ~~For the ObjectTable, only allow usertype 'administrator' to view/select teachers from the selector (4h)~~

Add an item

Tests Hide completed items Delete

100%

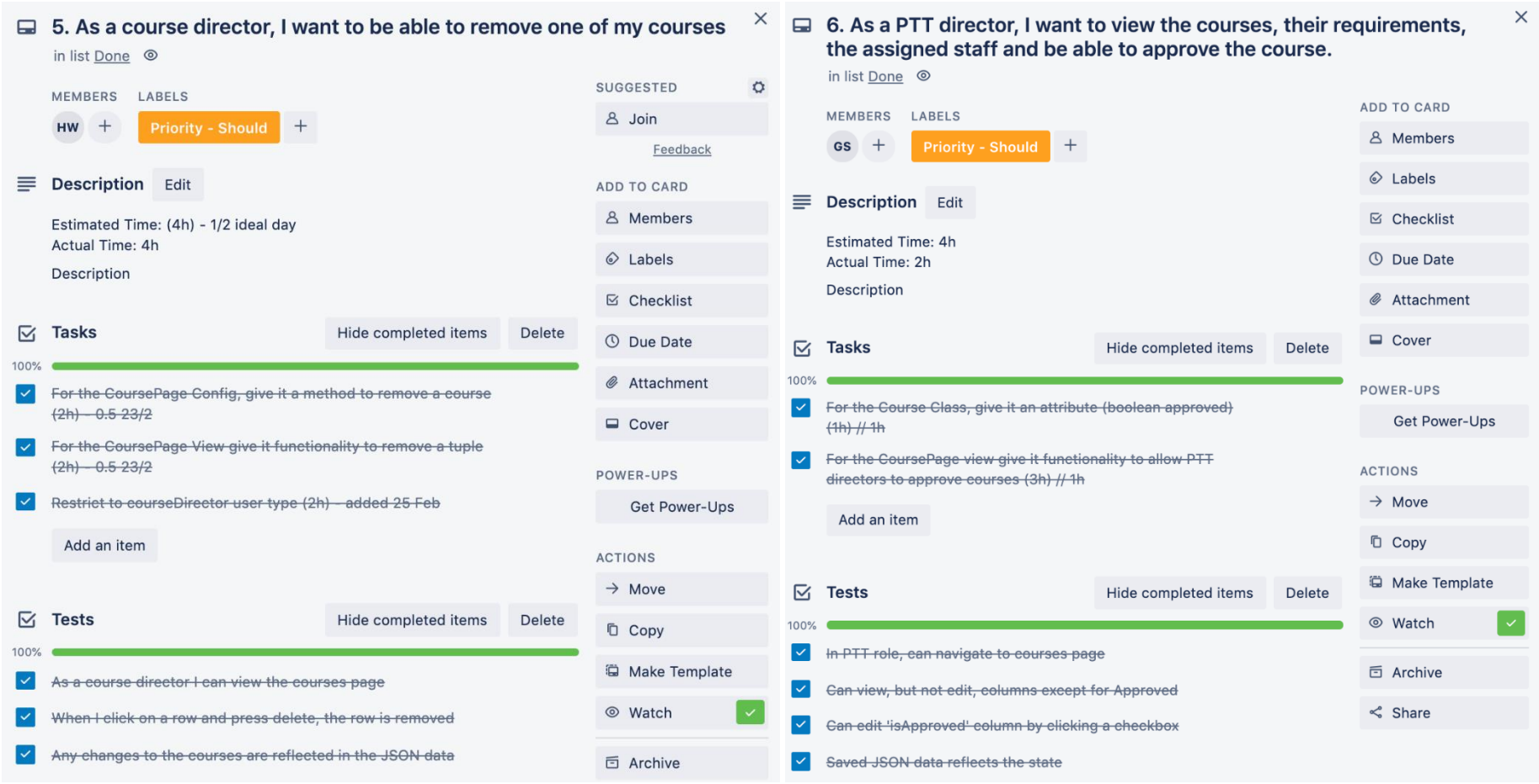
- ~~As an administrator, I can see the teacher page~~
- ~~I can change the qualifications held by the teachers~~
- ~~As an administrator, I can view the course page~~
- ~~When I click the assigned teachers cell for a course row, I see a list of teachers~~
- ~~The teachers displayed in that list hold ALL qualifications required for that course.~~
- ~~When I assign teachers to a course, this is reflected in the JSON data~~

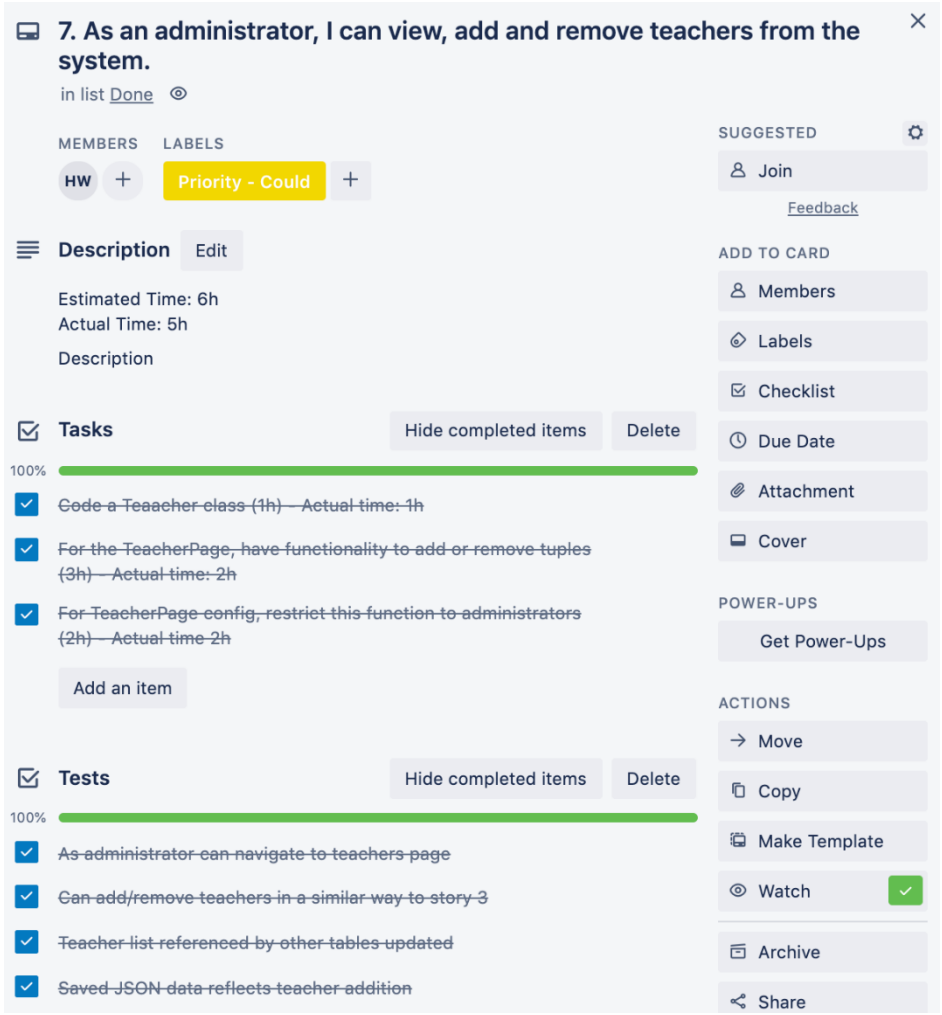
SUGGESTED
Join
Feedback

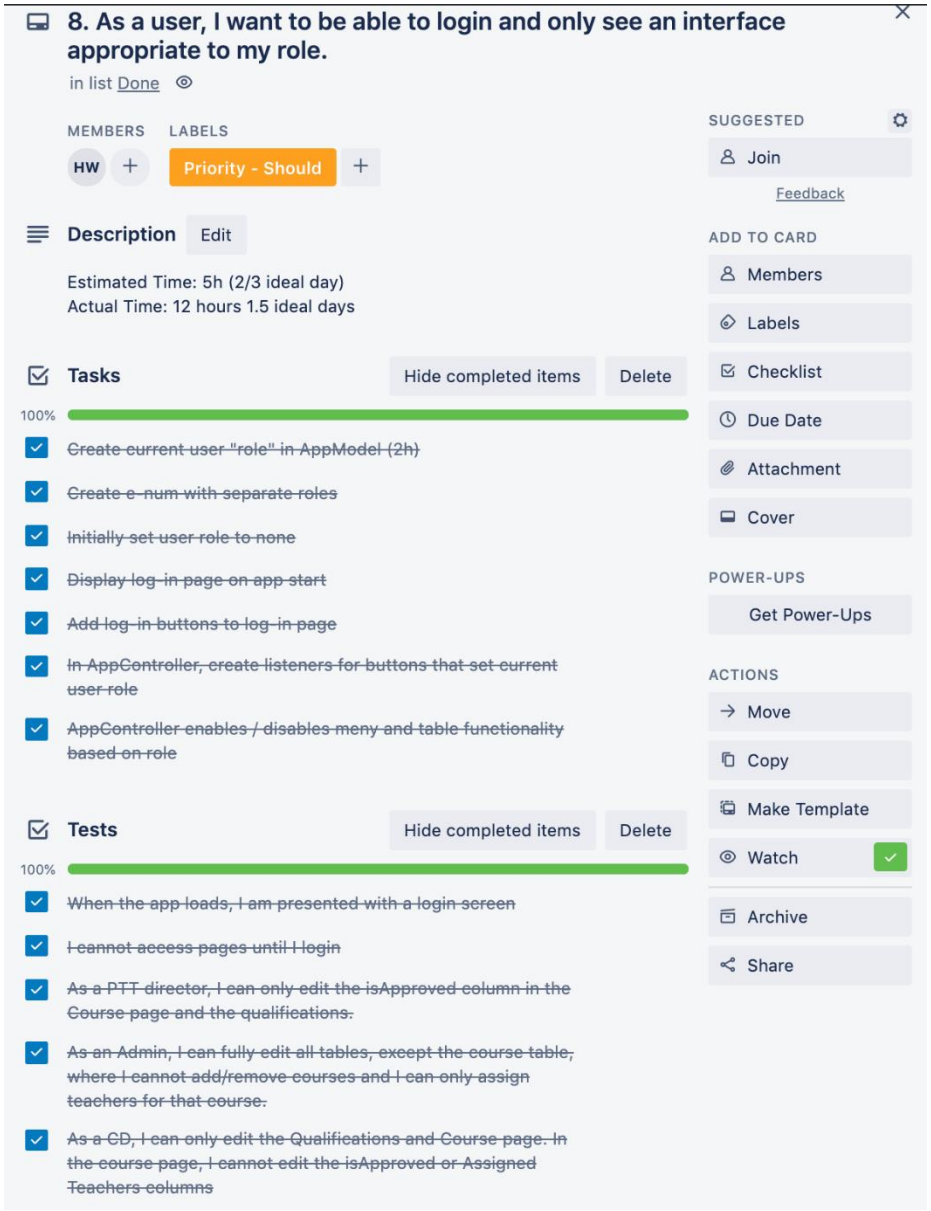
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover

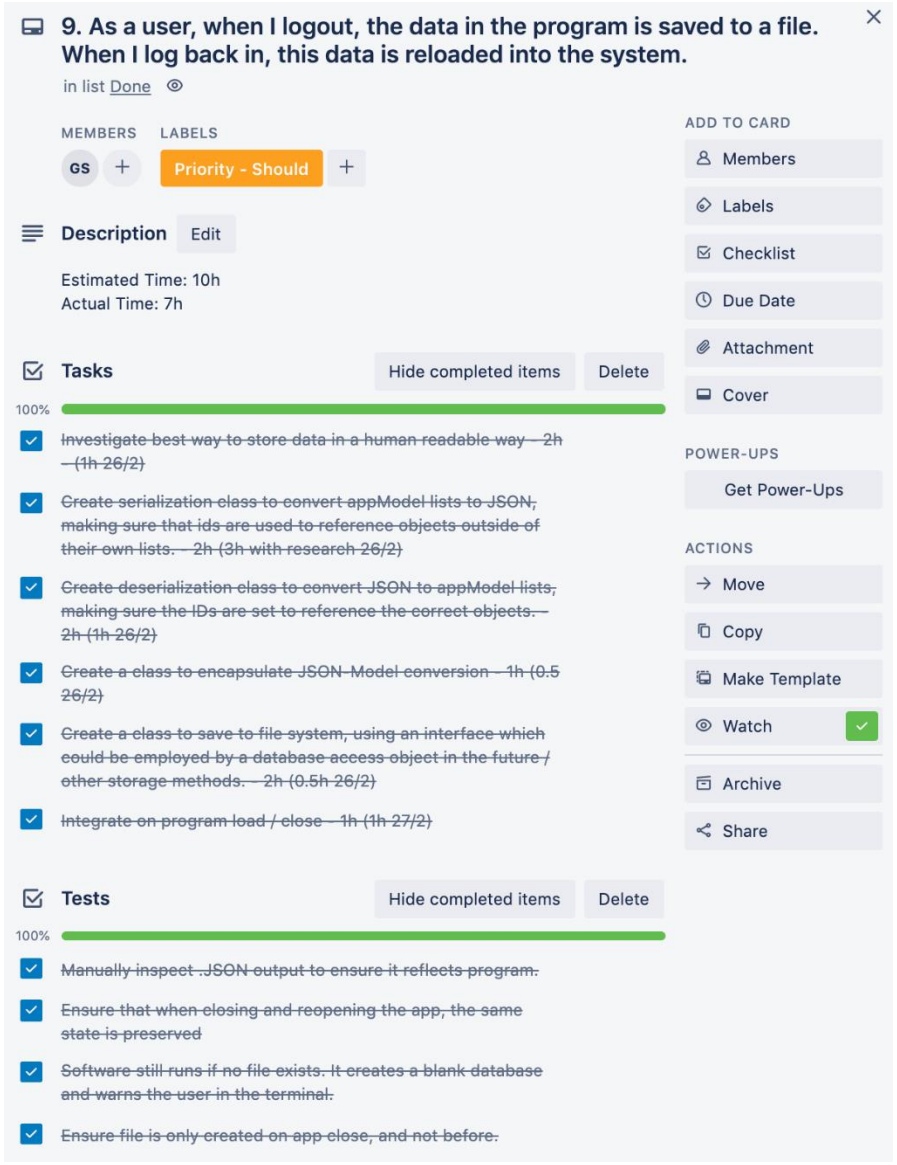
POWER-UPS
Get Power-Ups

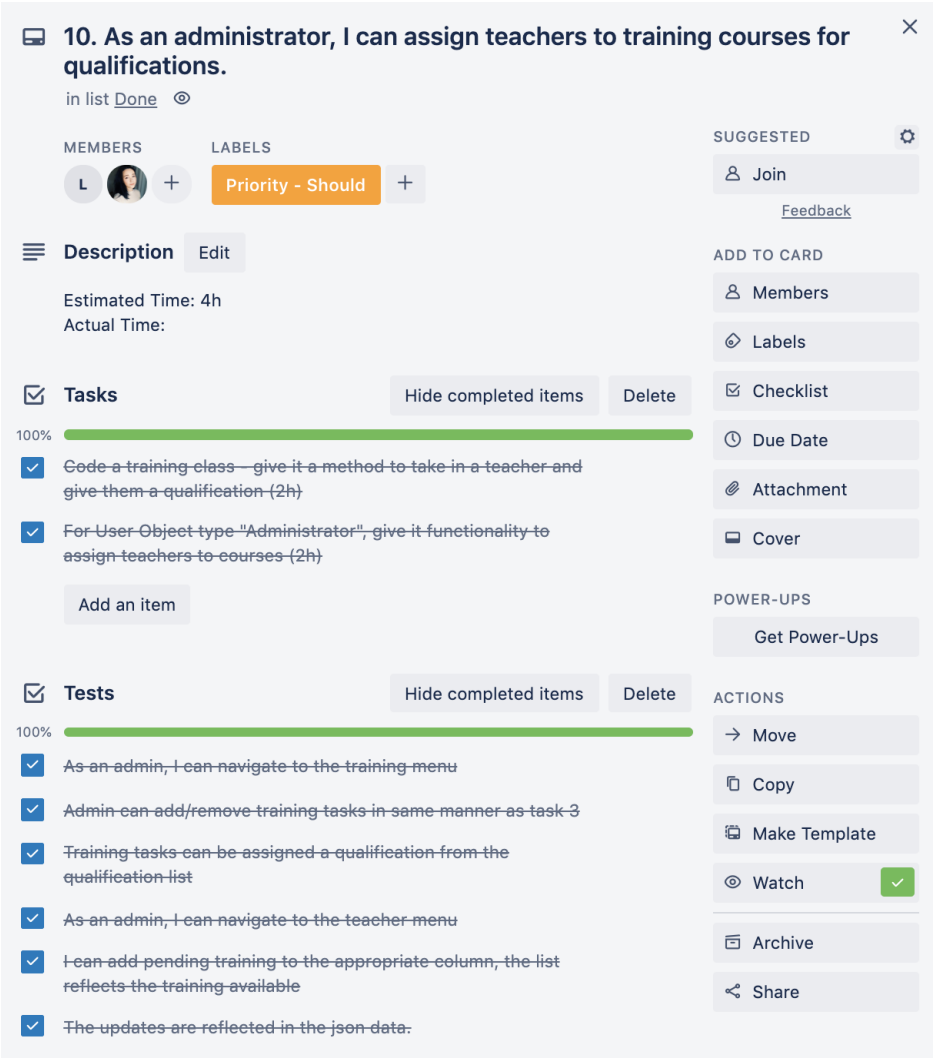
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share

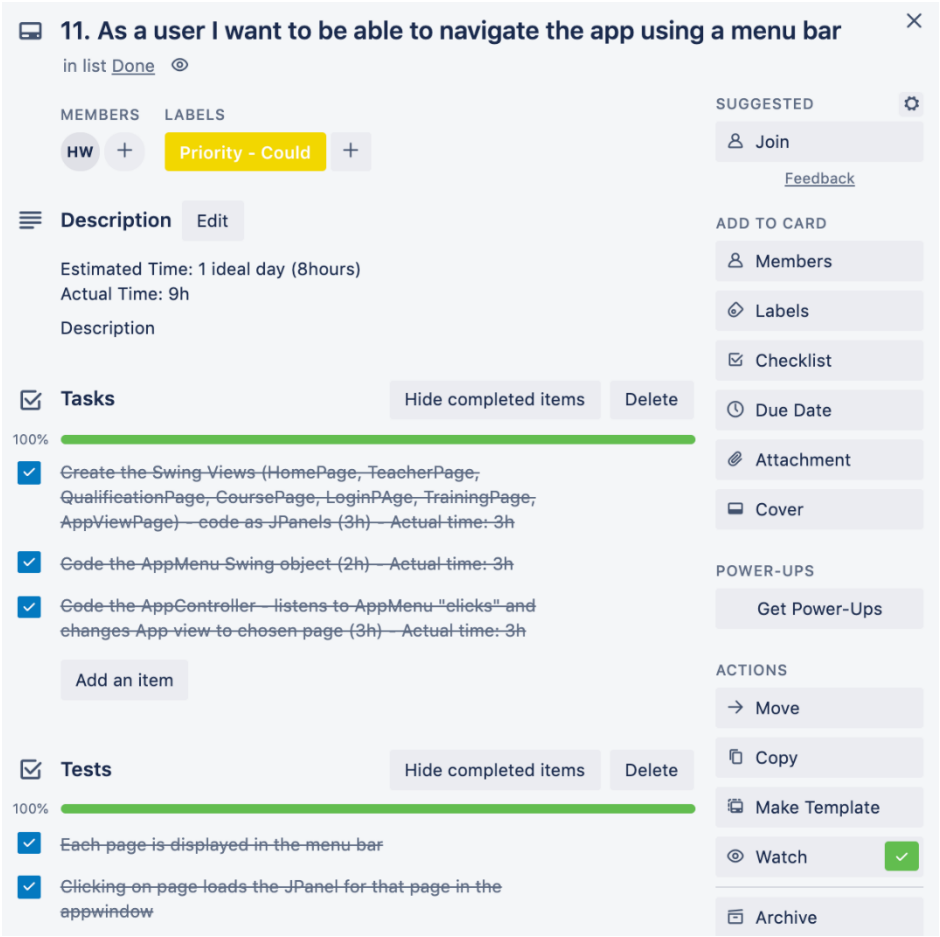

5. As a course director, I want to be able to remove one of my courses
in list Done
MEMBERS
LABELS
HW
Priority - Should
Description
Edit
Estimated Time: (4h) - 1/2 ideal day
Actual Time: 4h
Description
Tasks
Hide completed items
Delete
100%
For the CoursePage Config, give it a method to remove a course (2h) -- 0.5 23/2
For the CoursePage View give it functionality to remove a tuple (2h) -- 0.5 23/2
Restrict to courseDirector user type (2h) -- added 25 Feb
Add an item
Tests
Hide completed items
Delete
100%
As a course director I can view the courses page
When I click on a row and press delete, the row is removed
Any changes to the courses are reflected in the JSON data
SUGGESTED
Join
Feedback
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
6. As a PTT director, I want to view the courses, their requirements, the assigned staff and be able to approve the course.
in list Done
MEMBERS
LABELS
GS
Priority - Should
Description
Edit
Estimated Time: 4h
Actual Time: 2h
Description
Tasks
Hide completed items
Delete
100%
For the Course Class, give it an attribute (boolean approved) (1h) // 1h
For the CoursePage view give it functionality to allow PTT directors to approve courses (3h) // 1h
Add an item
Tests
Hide completed items
Delete
100%
In PTT role, can navigate to courses page
Can view, but not edit, columns except for Approved
Can edit 'isApproved' column by clicking a checkbox
Saved JSON data reflects the state
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share

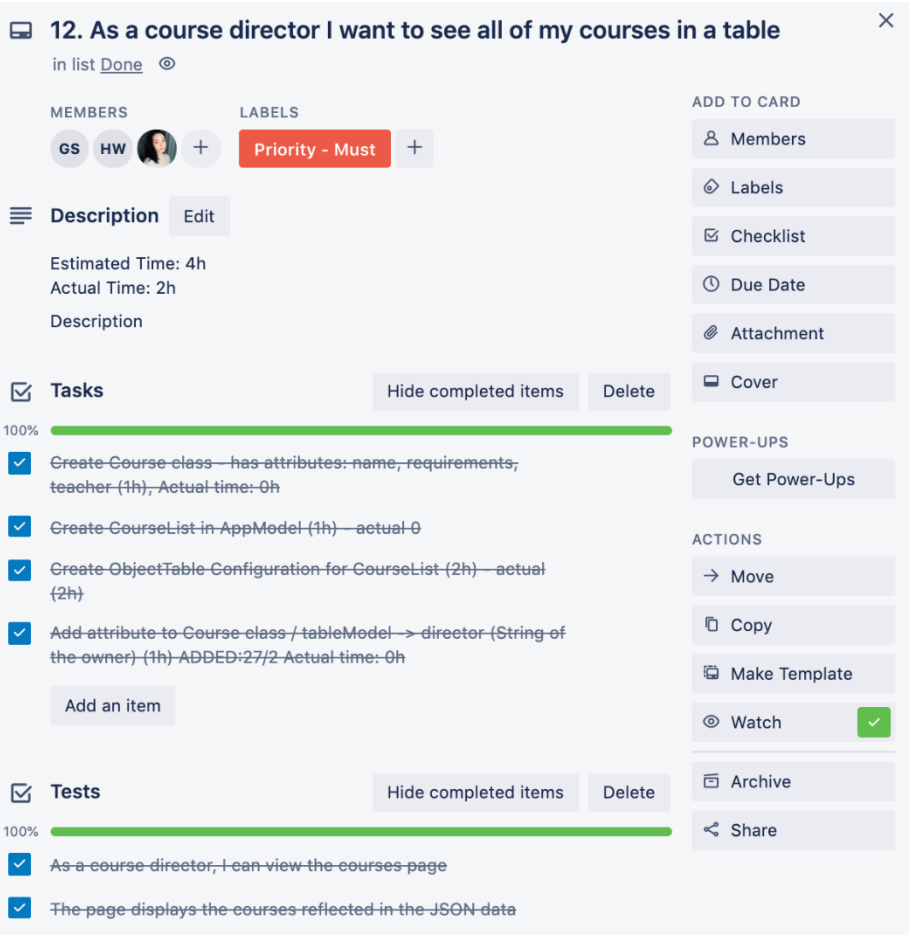

7. As an administrator, I can view, add and remove teachers from the system.
in list Done
MEMBERS
LABELS
HW
Priority - Could
Description
Edit
Estimated Time: 6h
Actual Time: 5h
Description
Tasks
Hide completed items
Delete
100%
Code a Teaacher class (1h) - Actual time: 1h
For the TeacherPage, have functionality to add or remove tuples (3h) - Actual time: 2h
For TeacherPage config, restrict this function to administrators (2h) - Actual time 2h
Add an item
Tests
Hide completed items
Delete
100%
As administrator can navigate to teachers page
Can add/remove teachers in a similar way to story 3
Teacher list referenced by other tables updated
Saved JSON data reflects teacher addition
SUGGESTED
Join
Feedback
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share


8. As a user, I want to be able to login and only see an interface appropriate to my role.
in list Done
MEMBERS
LABELS
HW
Priority - Should
Description
Edit
Estimated Time: 5h (2/3 ideal day)
Actual Time: 12 hours 1.5 ideal days
Tasks
Hide completed items
Delete
100%
Create current user "role" in AppModel (2h)
Create e-num with separate roles
Initially set user role to none
Display log-in page on app start
Add log-in buttons to log-in page
In AppController, create listeners for buttons that set current user role
AppController enables / disables meny and table functionality based on role
Tests
Hide completed items
Delete
100%
When the app loads, I am presented with a login screen
I cannot access pages until I login
As a PTT director, I can only edit the isApproved column in the Course page and the qualifications.
As an Admin, I can fully edit all tables, except the course table, where I cannot add/remove courses and I can only assign teachers for that course.
As a CD, I can only edit the Qualifications and Course page. In the course page, I cannot edit the isApproved or Assigned Teachers columns
SUGGESTED
Join
Feedback
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share


9. As a user, when I logout, the data in the program is saved to a file. When I log back in, this data is reloaded into the system.
in list Done
MEMBERS
LABELS
GS
Priority - Should
Description
Edit
Estimated Time: 10h
Actual Time: 7h
Tasks
Hide completed items
Delete
100%
Investigate best way to store data in a human readable way – 2h – (1h 26/2)
Create serialization class to convert appModel lists to JSON, making sure that ids are used to reference objects outside of their own lists. – 2h (3h with research 26/2)
Create deserialization class to convert JSON to appModel lists, making sure the IDs are set to reference the correct objects. – 2h (1h 26/2)
Create a class to encapsulate JSON-Model conversion – 1h (0.5 26/2)
Create a class to save to file system, using an interface which could be employed by a database access object in the future / other storage methods. – 2h (0.5h 26/2)
Integrate on program load / close – 1h (1h 27/2)
Tests
Hide completed items
Delete
100%
Manually inspect .JSON output to ensure it reflects program.
Ensure that when closing and reopening the app, the same state is preserved
Software still runs if no file exists. It creates a blank database and warns the user in the terminal.
Ensure file is only created on app close, and not before.
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share

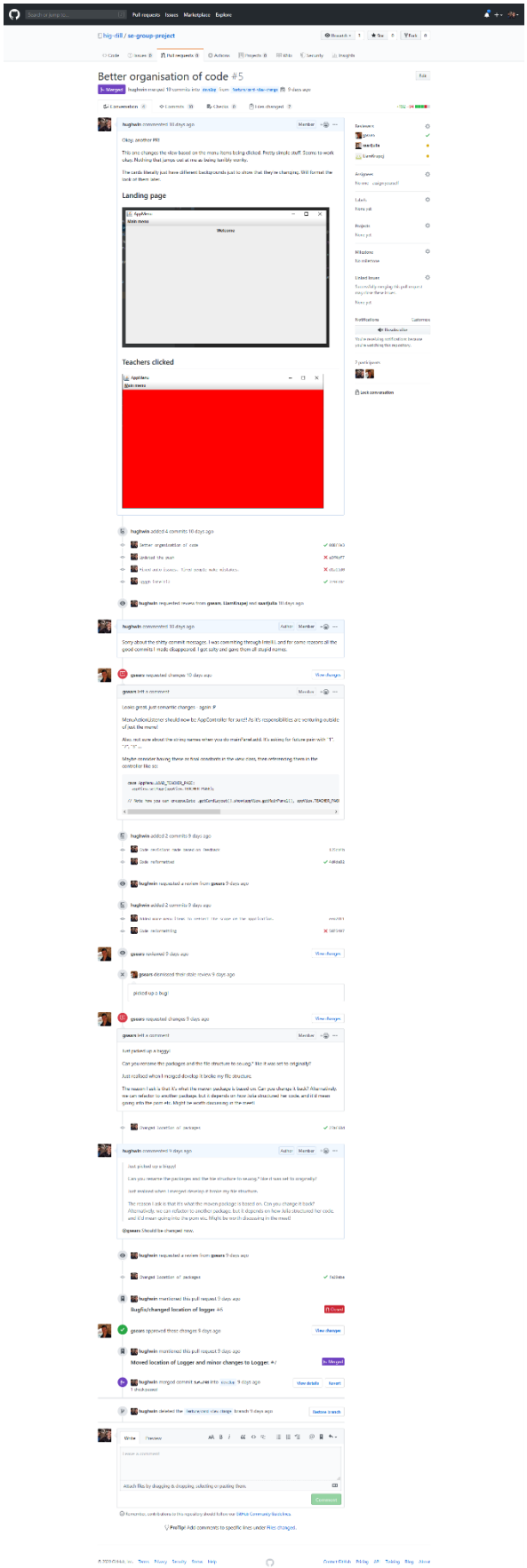

10. As an administrator, I can assign teachers to training courses for qualifications.
in list Done
MEMBERS
LABELS
L
Priority - Should
Description
Edit
Estimated Time: 4h
Actual Time:
Tasks
Hide completed items
Delete
100%
Code a training class – give it a method to take in a teacher and give them a qualification (2h)
For User Object type "Administrator", give it functionality to assign teachers to courses (2h)
Add an item
Tests
Hide completed items
Delete
100%
As an admin, I can navigate to the training menu
Admin can add/remove training tasks in same manner as task 3
Training tasks can be assigned a qualification from the qualification list
As an admin, I can navigate to the teacher menu
I can add pending training to the appropriate column, the list reflects the training available
The updates are reflected in the json data.
SUGGESTED
Join
Feedback
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share


11. As a user I want to be able to navigate the app using a menu bar
in list Done
MEMBERS
LABELS
HW
Priority - Could
Description
Edit
Estimated Time: 1 ideal day (8hours)
Actual Time: 9h
Description
Tasks
Hide completed items
Delete
100%
Create the Swing Views (HomePage, TeacherPage, QualificationPage, CoursePage, LoginPAge, TrainingPage, AppViewPage) - code as JPanels (3h) - Actual time: 3h
Code the AppMenu Swing object (2h) - Actual time: 3h
Code the AppController - listens to AppMenu "clicks" and changes App view to chosen page (3h) - Actual time: 3h
Add an item
Tests
Hide completed items
Delete
100%
Each page is displayed in the menu bar
Clicking on page loads the JPanel for that page in the appwindow
SUGGESTED
Join
Feedback
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive

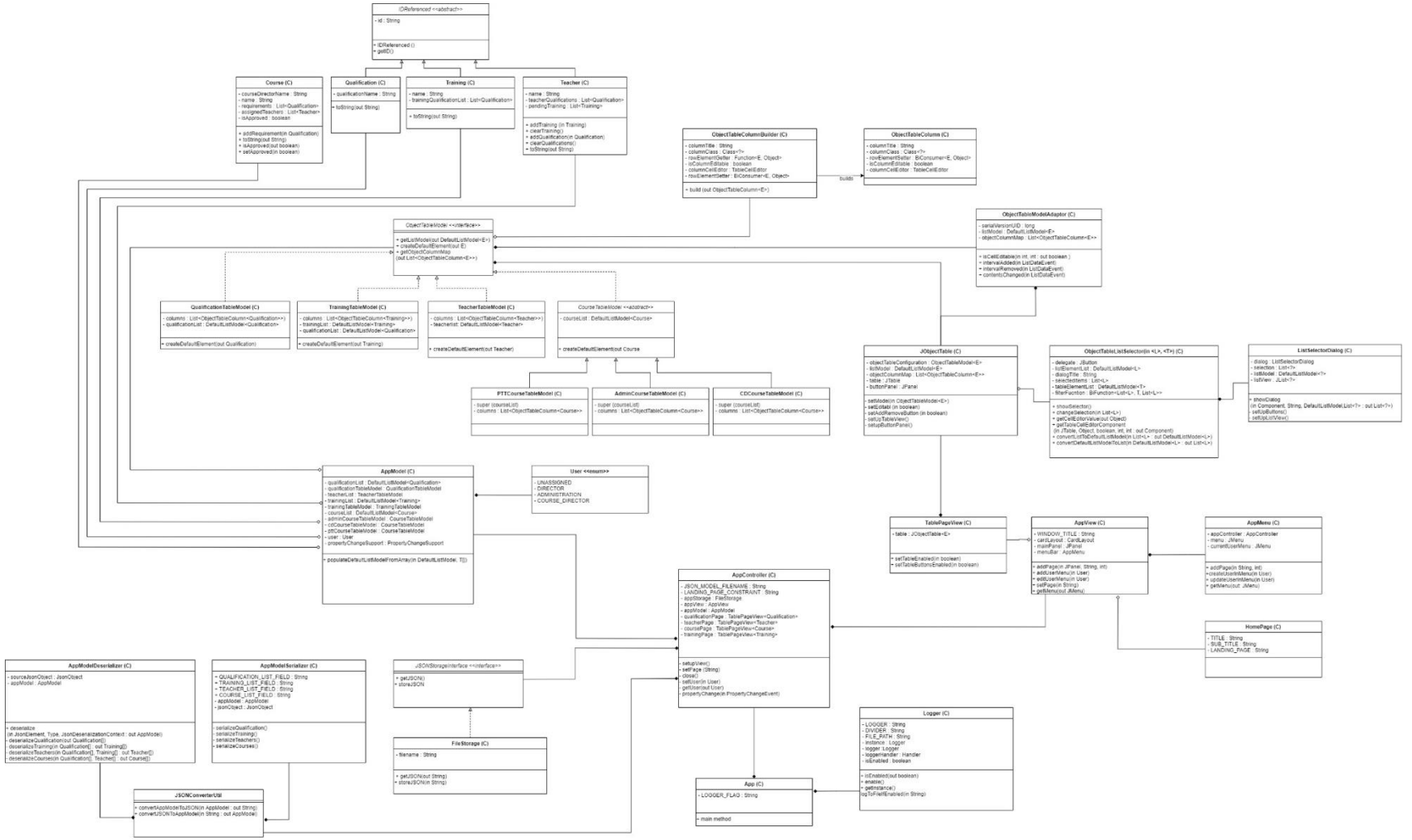

12. As a course director I want to see all of my courses in a table
in list Done
MEMBERS
LABELS
GS
HW
Priority - Must
Description
Edit
Estimated Time: 4h
Actual Time: 2h
Description
Tasks
Hide completed items
Delete
100%
Create Course class - has attributes: name, requirements, teacher (1h), Actual time: 0h
Create CourseList in AppModel (1h) - actual 0
Create ObjectTable Configuration for CourseList (2h) - actual (2h)
Add attribute to Course class / tableModel -> director (String of the owner) (1h) ADDED:27/2 Actual time: 0h
Add an item
Tests
Hide completed items
Delete
100%
As a course director, I can view the courses page
The page displays the courses reflected in the JSON data
ADD TO CARD
Members
Labels
Checklist
Due Date
Attachment
Cover
POWER-UPS
Get Power-Ups
ACTIONS
Move
Copy
Make Template
Watch
Archive
Share

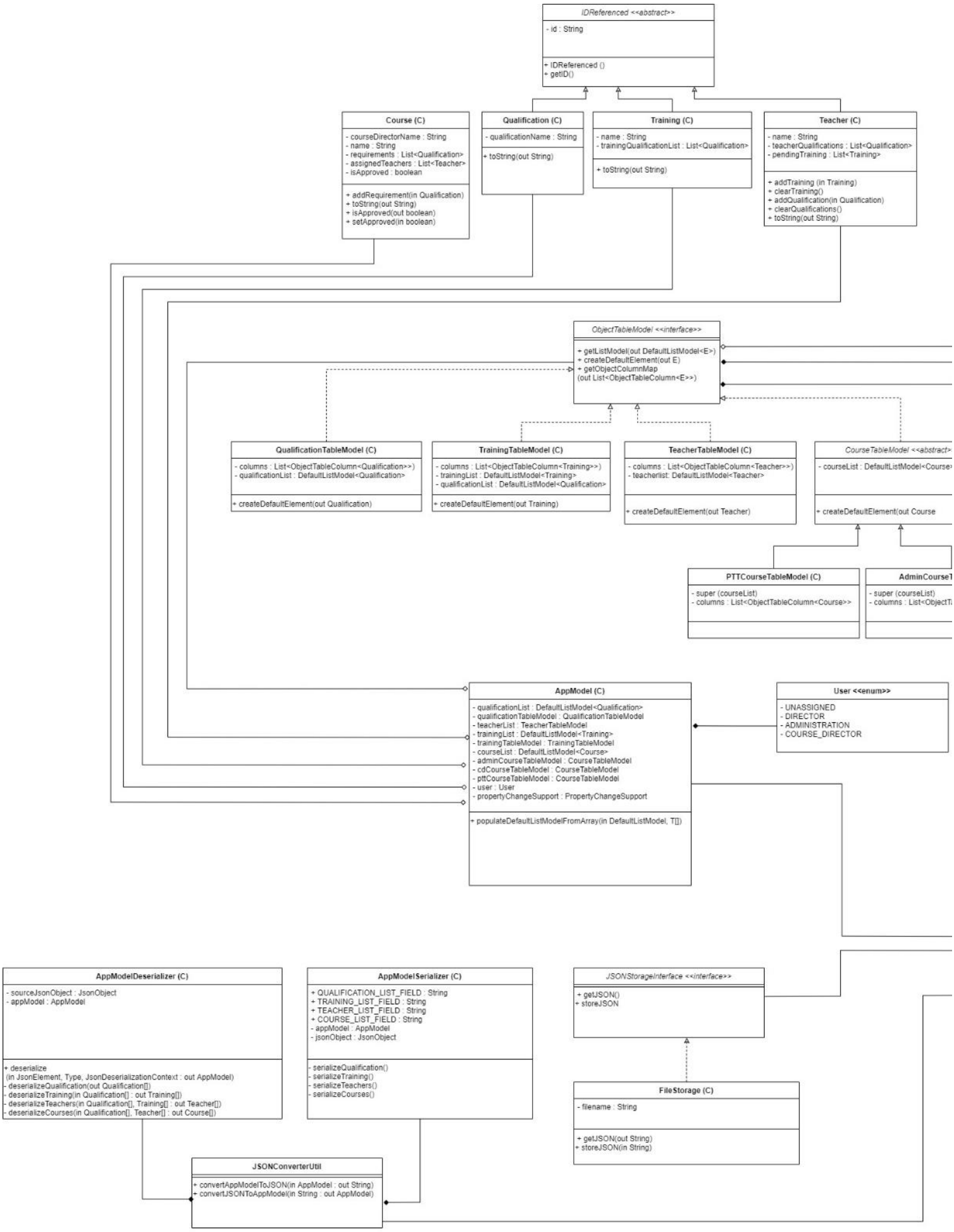
Conversations

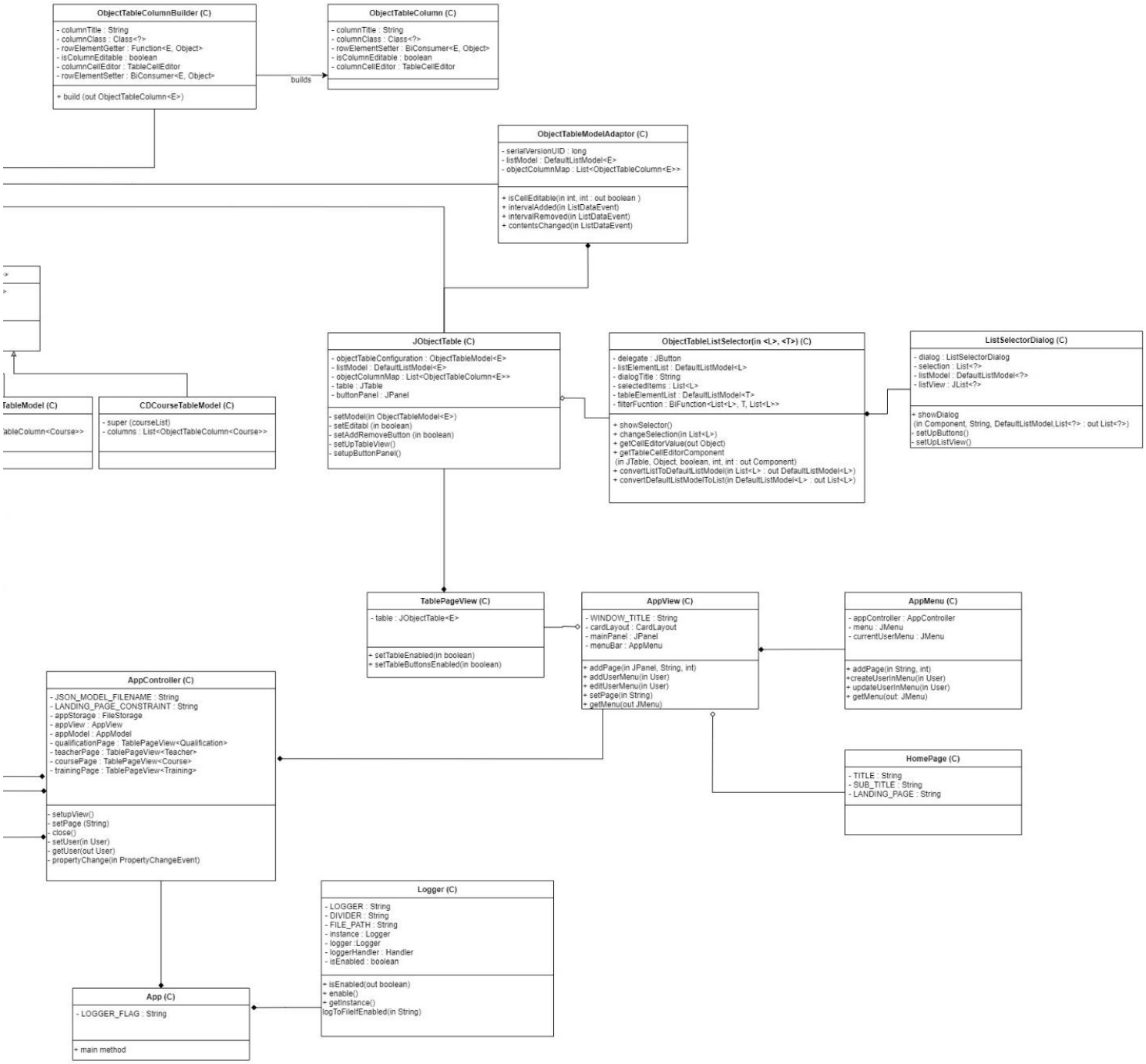
Most of our conversations weren't centralised on a single tool, and were relatively free flowing across Trello, GitHub and WhatsApp reflecting the decentralised and remote method of working we employed. As the conversation couldn't be accurately reproduced here, they have been included in full in the submitted zip file.


Better organisation of code #5
Landing page
Teachers clicked

Class Diagram – For a higher resolution version, please view ProjectUML.png in the project source code.

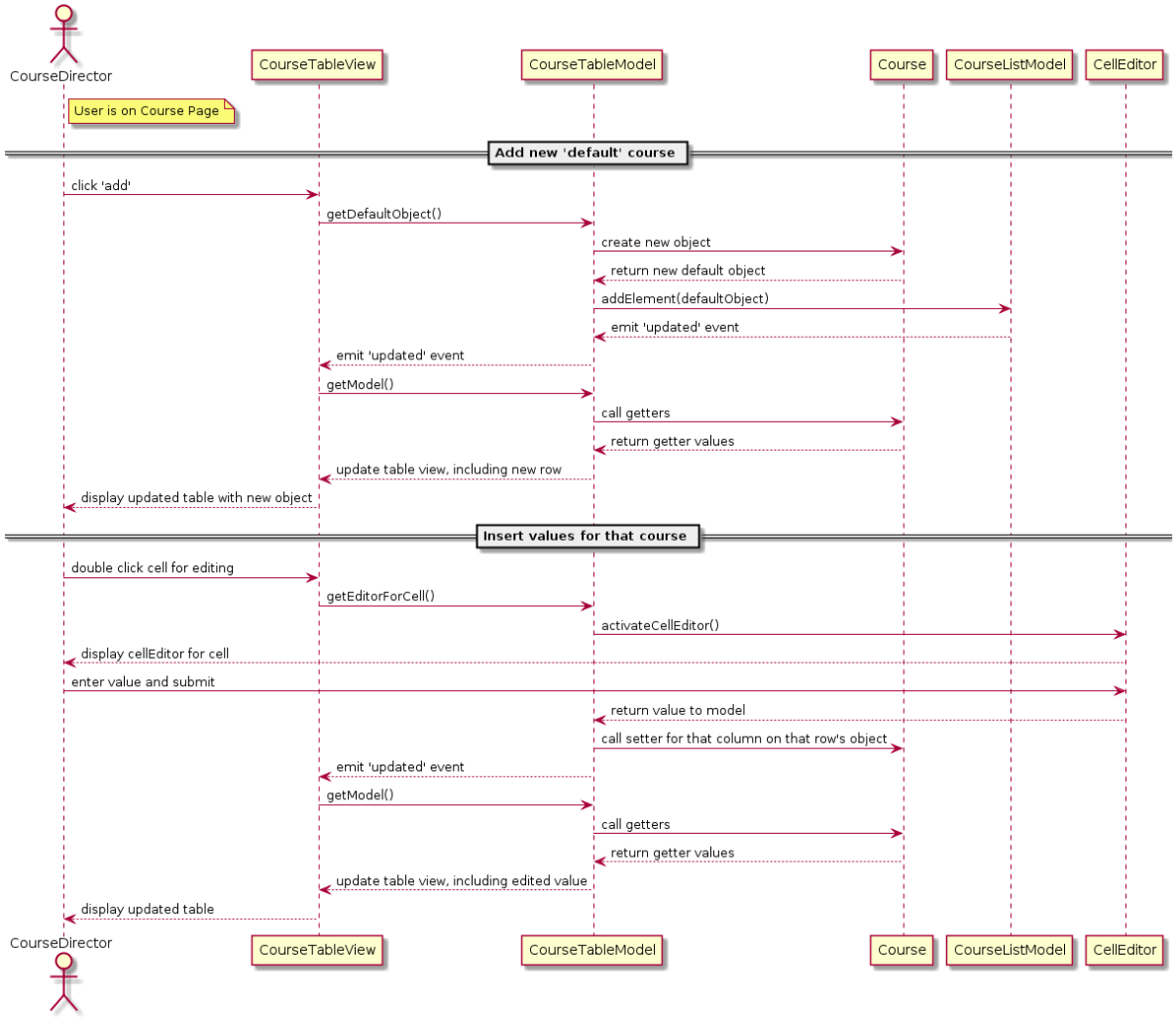



IDReferenced <<abstract>>
- id : String
+ IDReferenced ()
+ getID()
Course (C)
- courseDirectorName : String
- name : String
- requirements : List<Qualification>
- assignedTeachers : List<Teacher>
- isApproved : boolean
+ addRequirement(in Qualification)
+ toString(out String)
+ isApproved(out boolean)
+ setApproved(in boolean)
Qualification (C)
- qualificationName : String
+ toString(out String)
Training (C)
- name : String
- trainingQualificationList : List<Qualification>
+ toString(out String)
Teacher (C)
- name : String
- teacherQualifications : List<Qualification>
- pendingTraining : List<Training>
+ addTraining (in Training)
+ clearTraining()
+ addQualification(in Qualification)
+ clearQualifications()
+ toString(out String)
ObjectTableModel <<interface>>
+ getListModel(out DefaultListModel<E>)
+ createDefaultElement(out E)
+ getObjectColumnMap
(out List<ObjectTableColumn<E>>)
QualificationTableModel (C)
- columns : List<ObjectTableColumn<Qualification>>)
- qualificationList : DefaultListModel<Qualification>
+ createDefaultElement(out Qualification)
TrainingTableModel (C)
- columns : List<ObjectTableColumn<Training>>)
- trainingList : DefaultListModel<Training>
- qualificationList : DefaultListModel<Qualification>
+ createDefaultElement(out Training)
TeacherTableModel (C)
- columns : List<ObjectTableColumn<Teacher>>)
- teacherlist: DefaultListModel<Teacher>
+ createDefaultElement(out Teacher)
CourseTableModel <<abstract>
- courseList : DefaultListModel<Course>
+ createDefaultElement(out Course
PTTCourseTableModel (C)
- super (courseList)
- columns : List<ObjectTableColumn<Course>>
AdminCourseT
- super (courseList)
- columns : List<ObjectT
AppModel (C)
- qualificationList : DefaultListModel<Qualification>
- qualificationTableModel : QualificationTableModel
- teacherList : TeacherTableModel
- trainingList : DefaultListModel<Training>
- trainingTableModel : TrainingTableModel
- courseList : DefaultListModel<Course>
- adminCourseTableModel : CourseTableModel
- cdCourseTableModel : CourseTableModel
- pttCourseTableModel : CourseTableModel
- user : User
- propertyChangeSupport : PropertyChangeSupport
+ populateDefaultListModelFromArray(in DefaultListModel, T[])
User <<enum>>
- UNASSIGNED
- DIRECTOR
- ADMINISTRATION
- COURSE_DIRECTOR
AppModelDeserializer (C)
- sourceJsonObject : JsonObject
- appModel : AppModel
+ deserialize
(in JsonElement, Type, JsonDeserializationContext : out AppModel)
- deserializeQualification(out Qualification[])
- deserializeTraining(in Qualification[] : out Training[])
- deserializeTeachers(in Qualification[], Training[] : out Teacher[])
- deserializeCourses(in Qualification[], Teacher[] : out Course[])
AppModelSerializer (C)
+ QUALIFICATION_LIST_FIELD : String
+ TRAINING_LIST_FIELD : String
+ TEACHER_LIST_FIELD : String
+ COURSE_LIST_FIELD : String
- appModel : AppModel
- jsonObject : JsonObject
- serializeQualification()
- serializeTraining()
- serializeTeachers()
- serializeCourses()
JSONStorageInterface <<interface>>
+ getJSON()
+ storeJSON
FileStorage (C)
- filename : String
+ getJSON(out String)
+ storeJSON(in String)
JSONConverterUtil
+ convertAppModelToJSON(in AppModel : out String)
+ convertJSONToAppModel(in String : out AppModel)

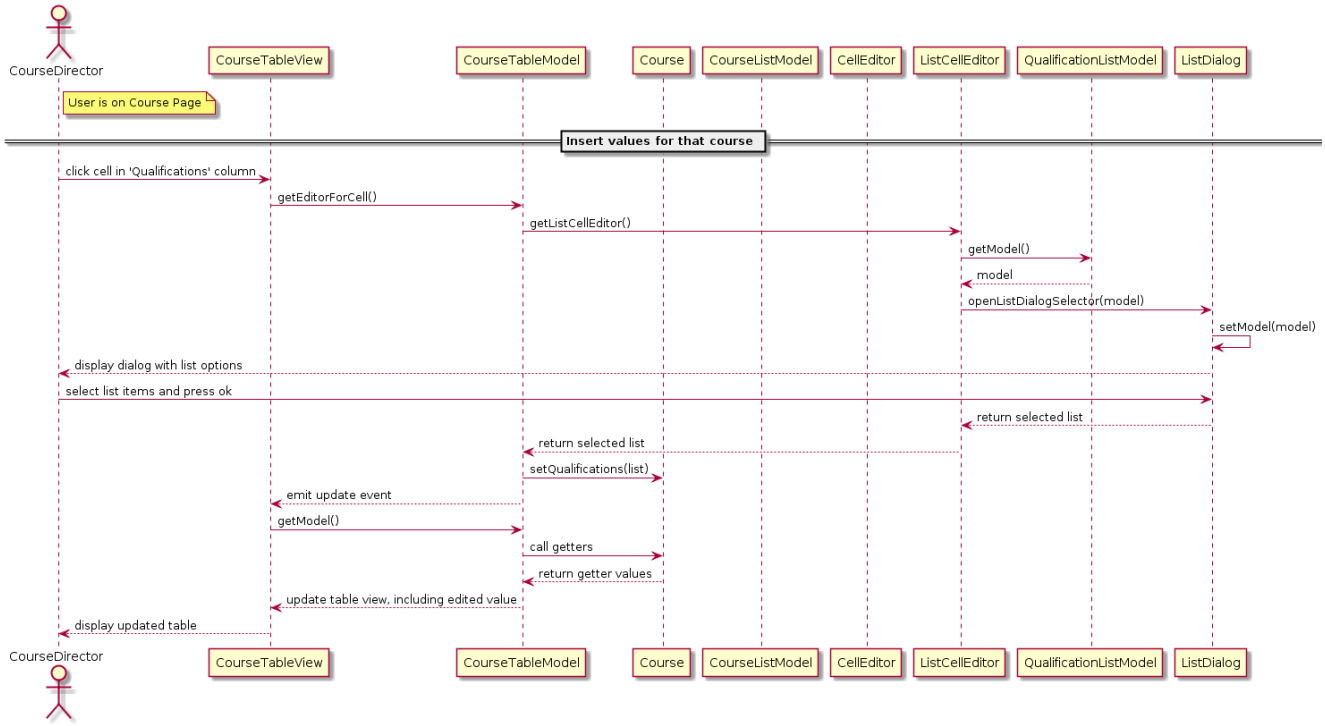

ObjectTableColumnBuilder (C)
- columnTitle : String
- columnClass : Class<?>
- rowElementGetter : Function<E, Object>
- isColumnEditable : boolean
- columnCellEditor : TableCellEditor
- rowElementSetter : BiConsumer<E, Object>
+ build (out ObjectTableColumn<E>)
builds
ObjectTableColumn (C)
- columnTitle : String
- columnClass : Class<?>
- rowElementSetter : BiConsumer<E, Object>
- isColumnEditable : boolean
- columnCellEditor : TableCellEditor
ObjectTableModelAdaptor (C)
- serialVersionUID : long
- listModel : DefaultListModel<E>
- objectColumnMap : List<ObjectTableColumn<E>>
+ isCellEditable(in int, int : out boolean)
+ intervalAdded(in ListDataEvent)
+ intervalRemoved(in ListDataEvent)
+ contentsChanged(in ListDataEvent)
JObjectTable (C)
- objectTableConfiguration : ObjectTableModel<E>
- listModel : DefaultListModel<E>
- objectColumnMap : List<ObjectTableColumn<E>>
- table : JTable
- buttonPanel : JPanel
- setModel(in ObjectTableModel<E>)
- setEditabl (in boolean)
- setAddRemoveButton (in boolean)
- setUpTableView()
- setupButtonPanel()
ObjectTableListSelector(in <L>, <T>) (C)
- delegate : JButton
- listElementList : DefaultListModel<L>
- dialogTitle : String
- selectedItems : List<L>
- tableElementList : DefaultListModel<T>
- filterFucntion : BiFunction<List<L>, T, List<L>>
+ showSelector()
+ changeSelection(in List<L>)
+ getCellEditorValue(out Object)
+ getTableCellEditorComponent
(in JTable, Object, boolean, int, int : out Component)
+ convertListToDefaultListModel(in List<L> : out DefaultListModel<L>)
+ convertDefaultListModelToList(in DefaultListModel<L> : out List<L>)
ListSelectorDialog (C)
- dialog : ListSelectorDialog
- selection : List<?>
- listModel : DefaultListModel<?>
- listView : JList<?>
+ showDialog
(in Component, String, DefaultListModel.List<?> : out List<?>)
- setUpButtons()
- setUpListView()
CDCourseTableModel (C)
- super (courseList)
- columns : List<ObjectTableColumn<Course>>
TablePageView (C)
- table : JObjectTable<E>
+ setTableEnabled(in boolean)
+ setTableButtonsEnabled(in boolean)
AppView (C)
- WINDOW_TITLE : String
- cardLayout : CardLayout
- mainPanel : JPanel
- menuBar : AppMenu
+ addPage(in JPanel, String, int)
+ addUserMenu(in User)
+ editUserMenu(in User)
+ setPage(in String)
+ getMenu(out JMenu)
AppMenu (C)
- appController : AppController
- menu : JMenu
- currentUserMenu : JMenu
+ addPage(in String, int)
+createUserInMenu(in User)
+ updateUserInMenu(in User)
+ getMenu(out: JMenu)
AppController (C)
- JSON_MODEL_FILENAME : String
- LANDING_PAGE_CONSTRAINT : String
- appStorage : FileStorage
- appView : AppView
- appModel : AppModel
- qualificationPage : TablePageView<Qualification>
- teacherPage : TablePageView<Teacher>
- coursePage : TablePageView<Course>
- trainingPage : TablePageView<Training>
- setupView()
- setPage (String)
- close()
- setUser(in User)
- getUser(out User)
- propertyChange(in PropertyChangeEvent)
HomePage (C)
- TITLE : String
- SUB_TITLE : String
- LANDING_PAGE : String
Logger (C)
- LOGGER : String
- DIVIDER : String
- FILE_PATH : String
- instance : Logger
- logger :Logger
- loggerHandler : Handler
- isEnabled : boolean
+ isEnabled(out boolean)
+ enable()
+ getInstance()
logToFileIfEnabled(in String)
App (C)
- LOGGER_FLAG : String
+ main method

Sequence Diagrams Per User Story

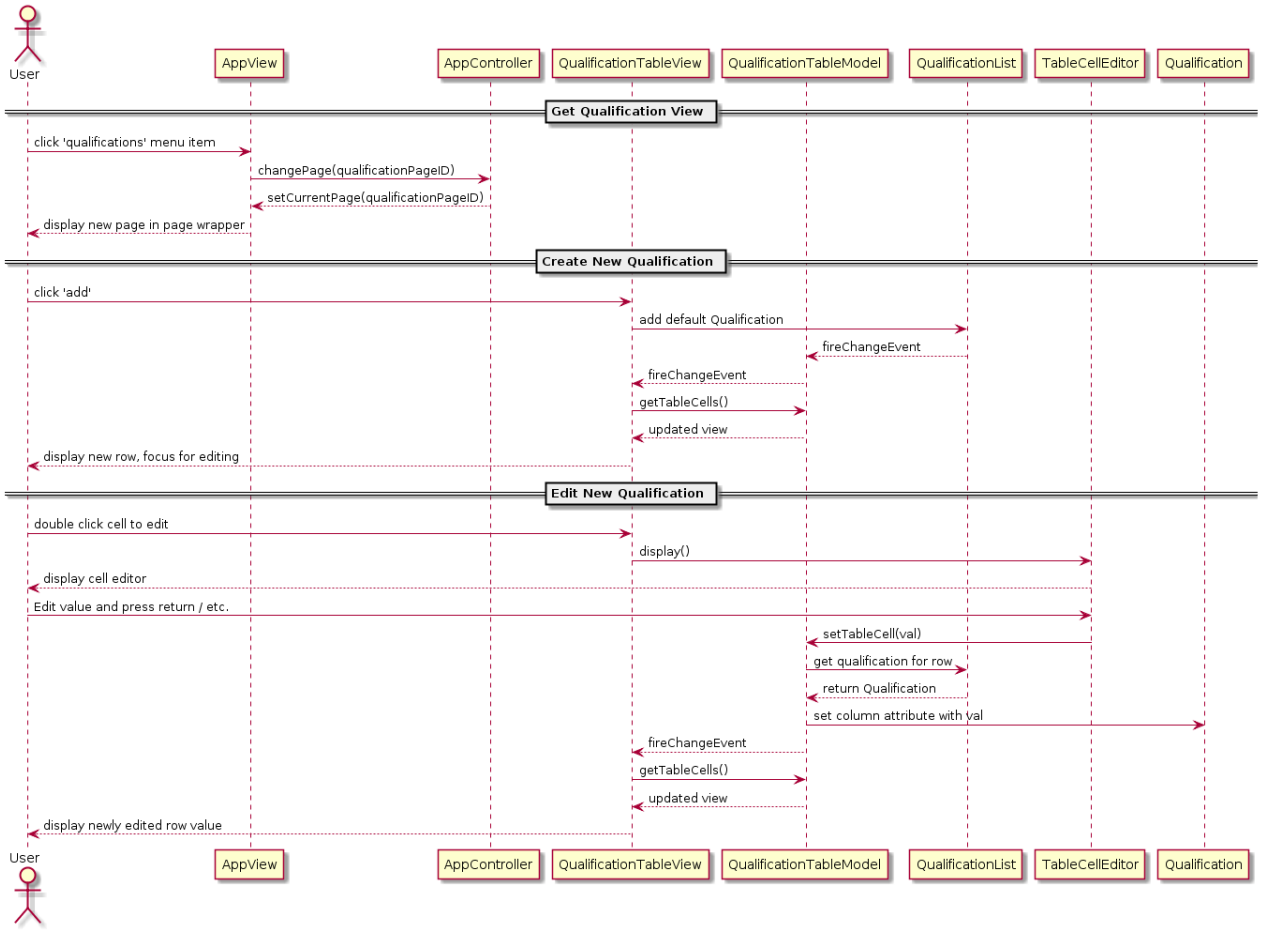
1. As a Course Director, I want to be able to create courses. These courses will have a name and a list of requirements:



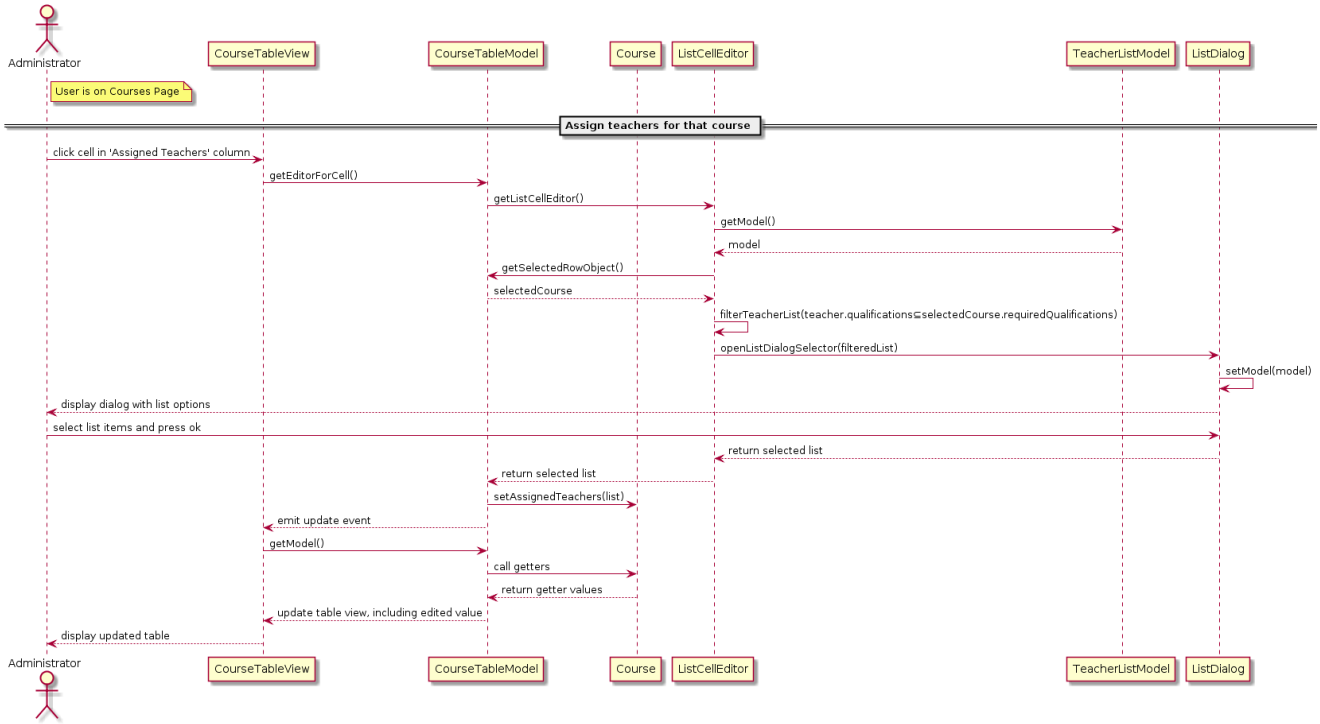
2. As a Course Director, for each course I own, I want to be able to create a list of teaching requirements. These requirements are: the qualifications needed to teach the course:



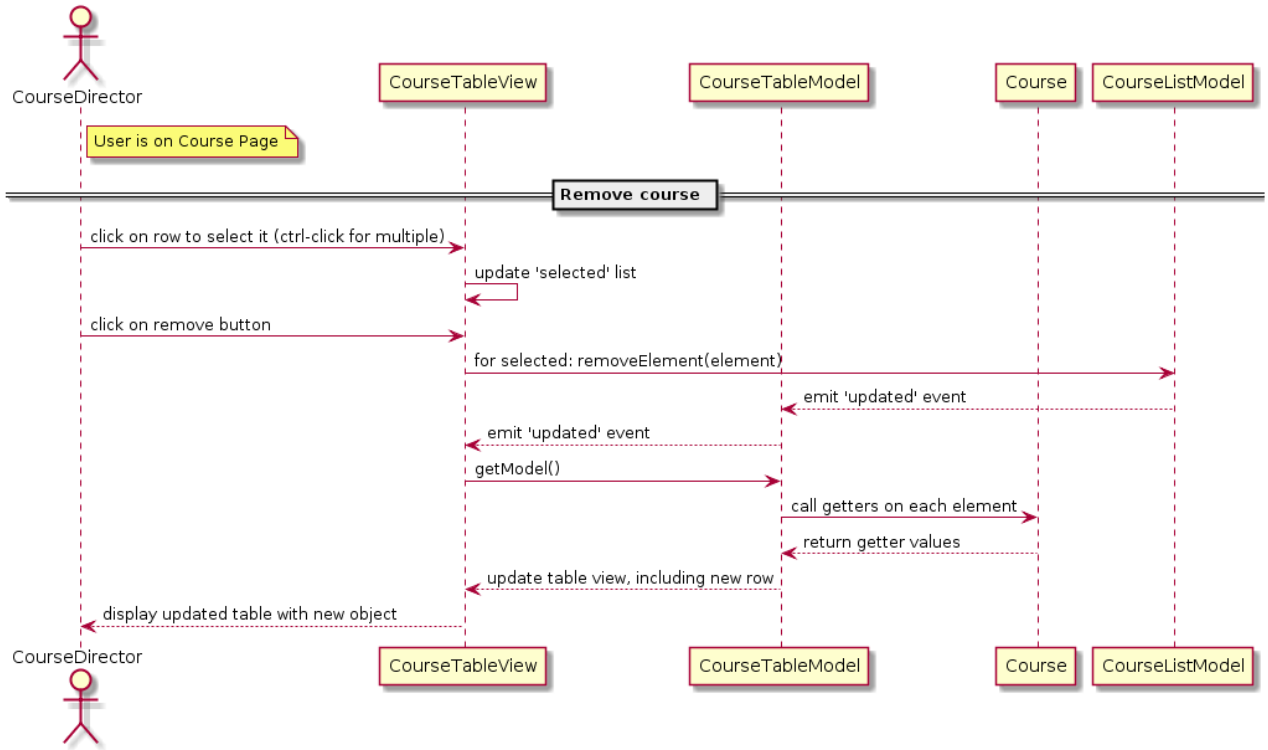
3. As a User, I can add and remove qualifications to the database.



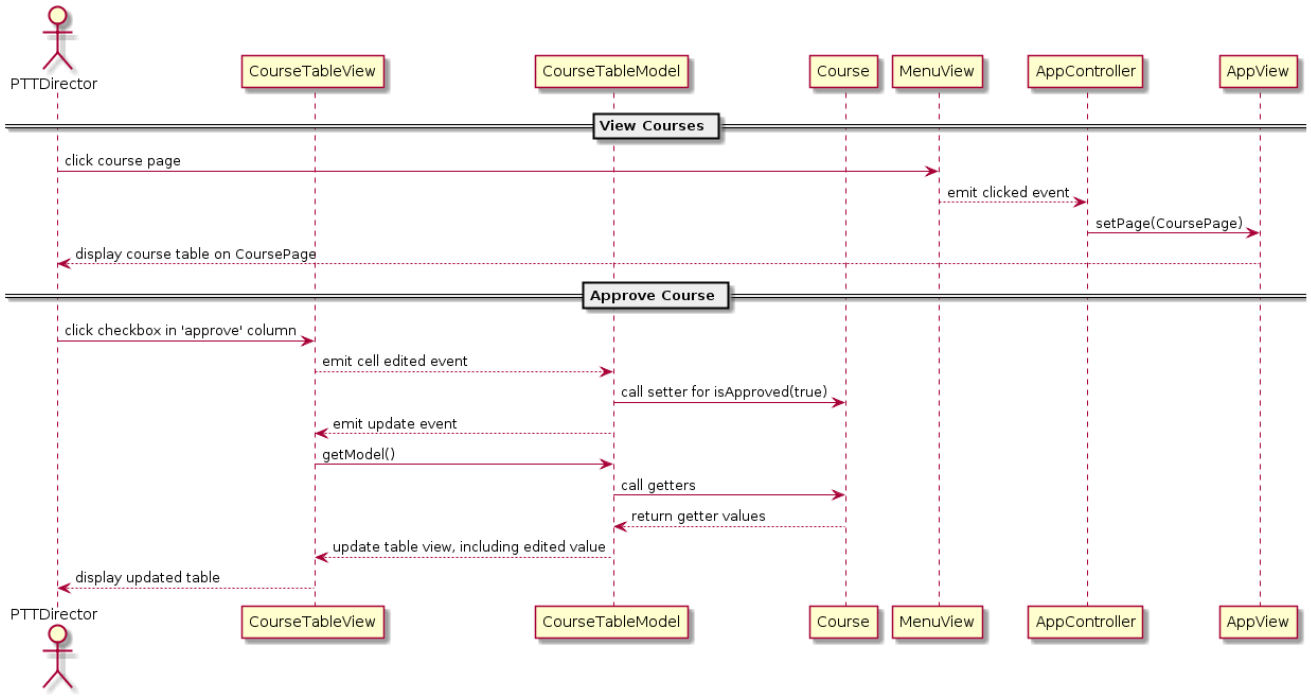
4. As an administrator, I can assign available teachers to a course when they hold the relevant qualifications.



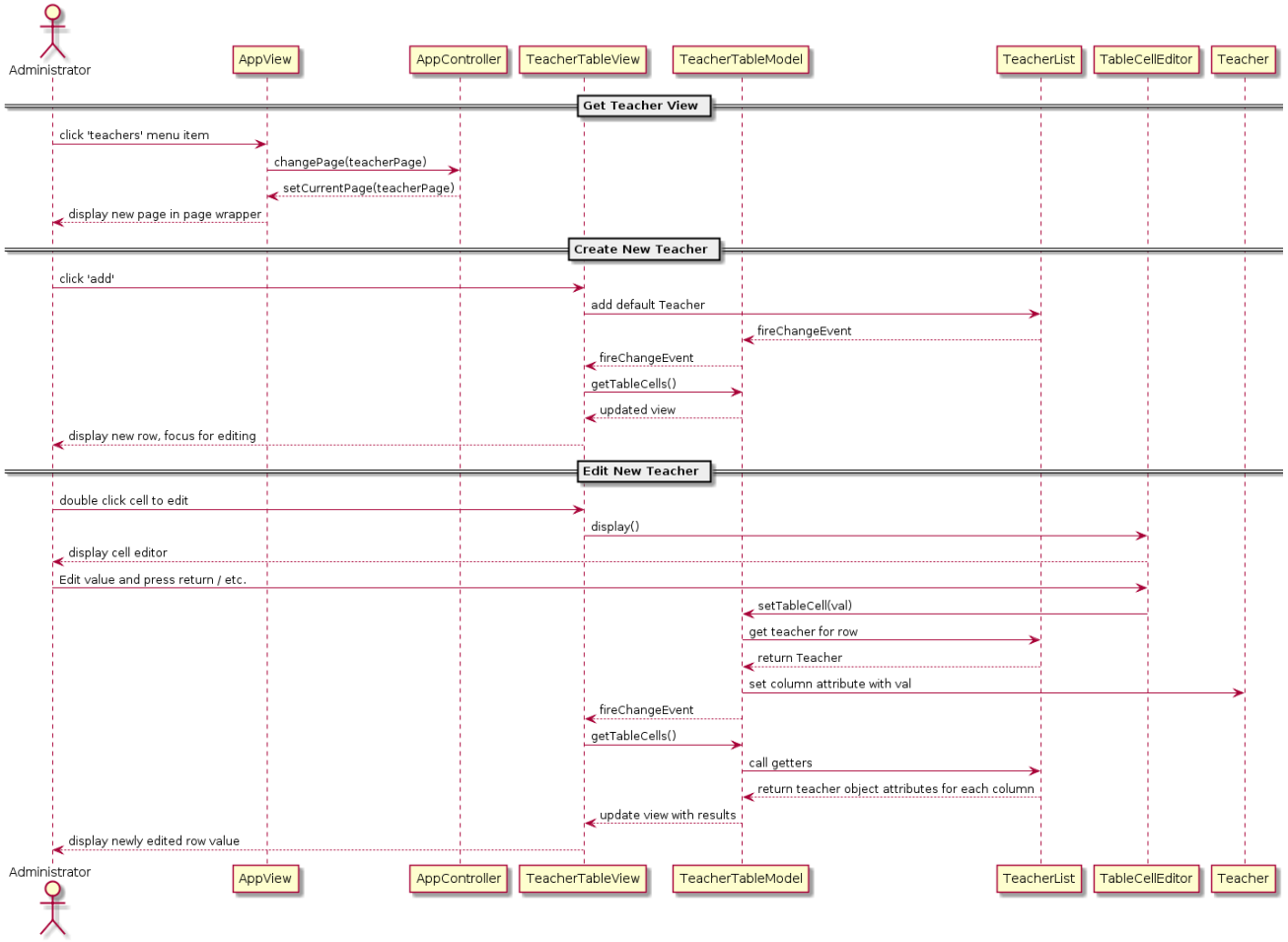
5. As a course director, I want to be able to remove one of my courses



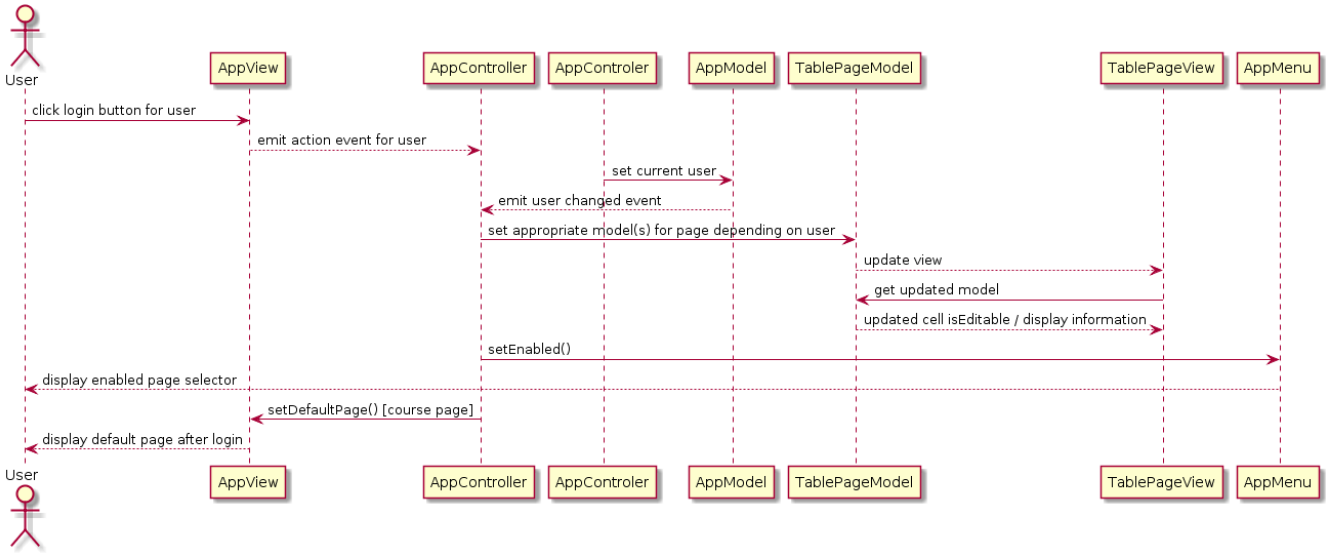
6. As a PTT director, I want to view the courses, their requirements, the assigned staff and be able to approve the course.



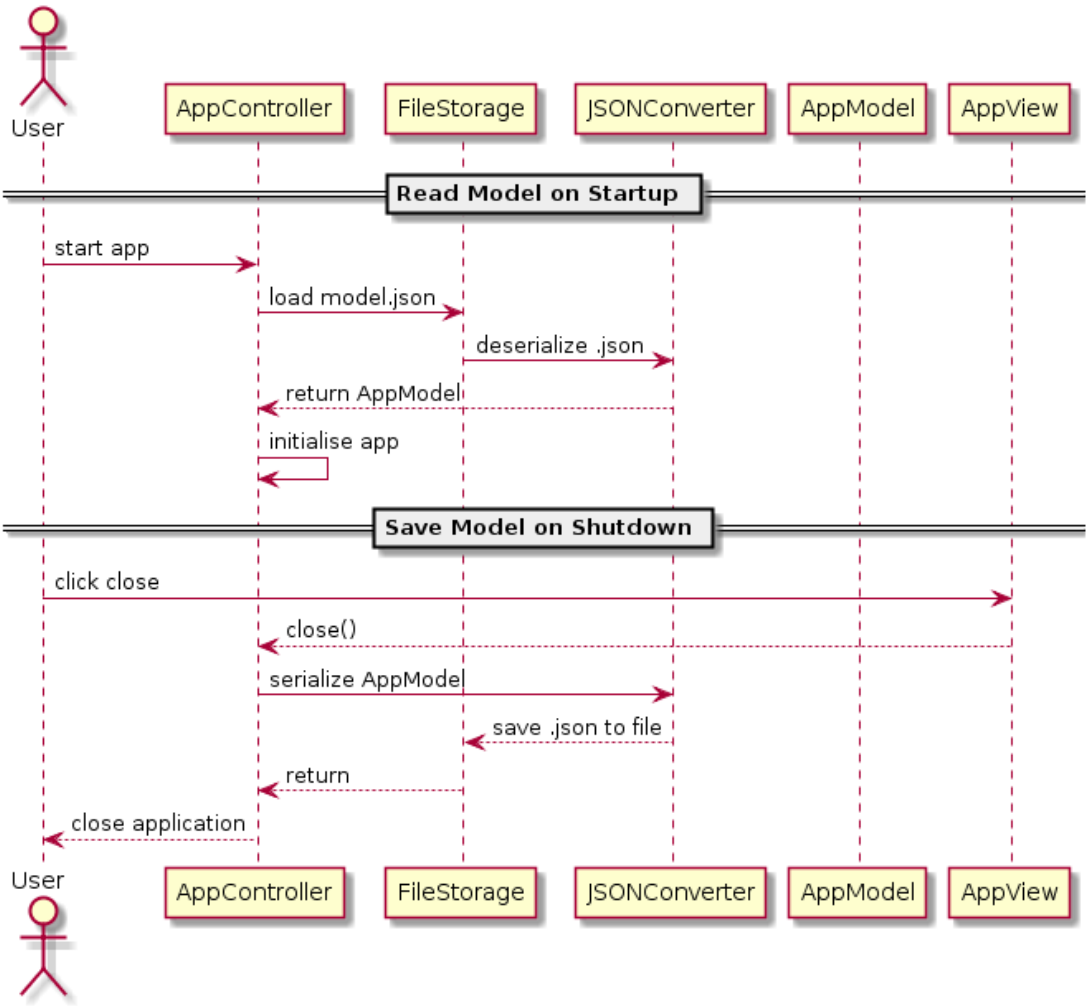
7. As an administrator, I can view, add and remove teachers from the system.



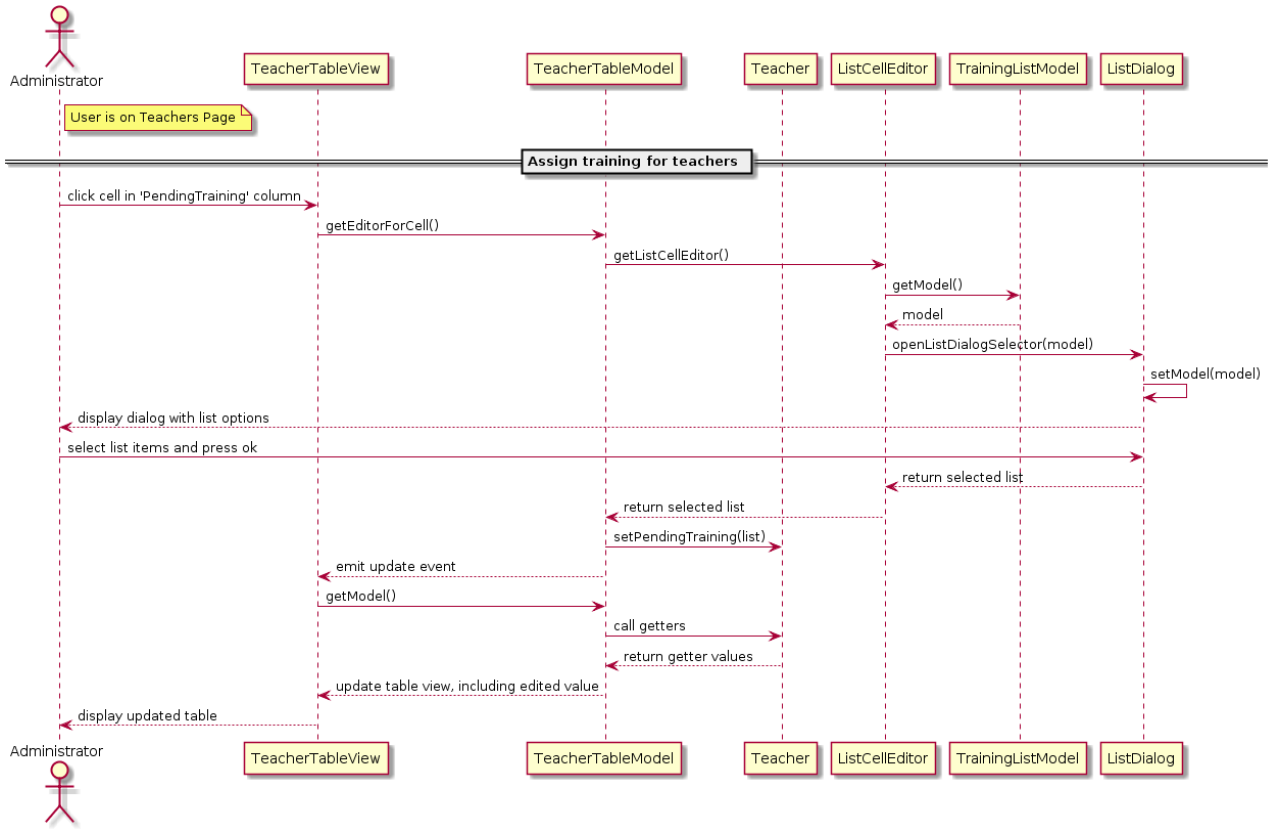
8. As a user, I want to be able to login and only see an interface appropriate to my role.



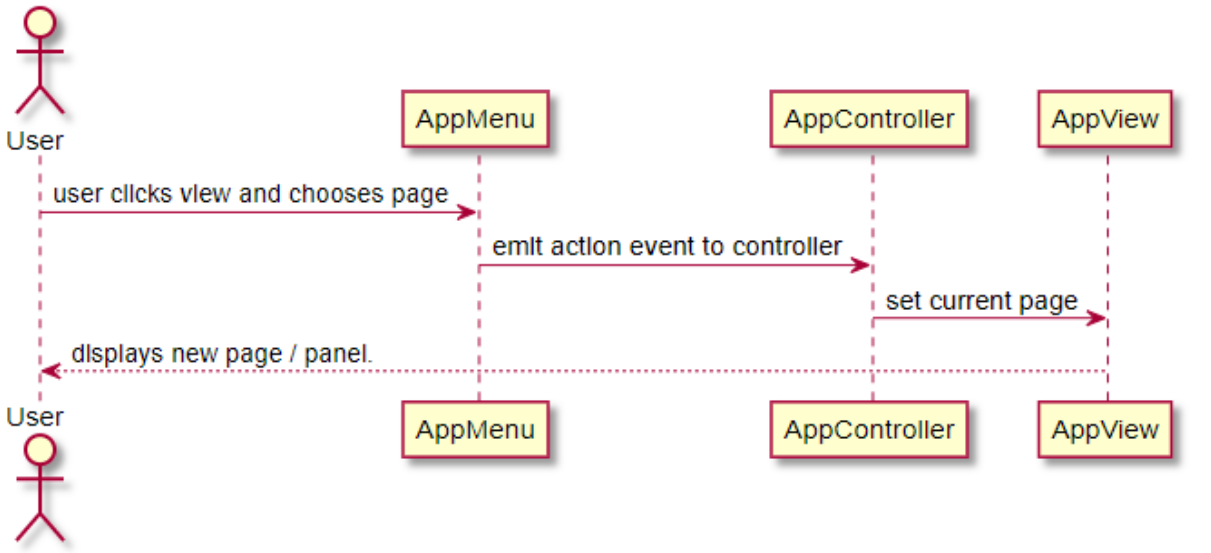
9. As a user, when I logout, the data in the program is saved to a file. When I log back in, this data is reloaded into the system.



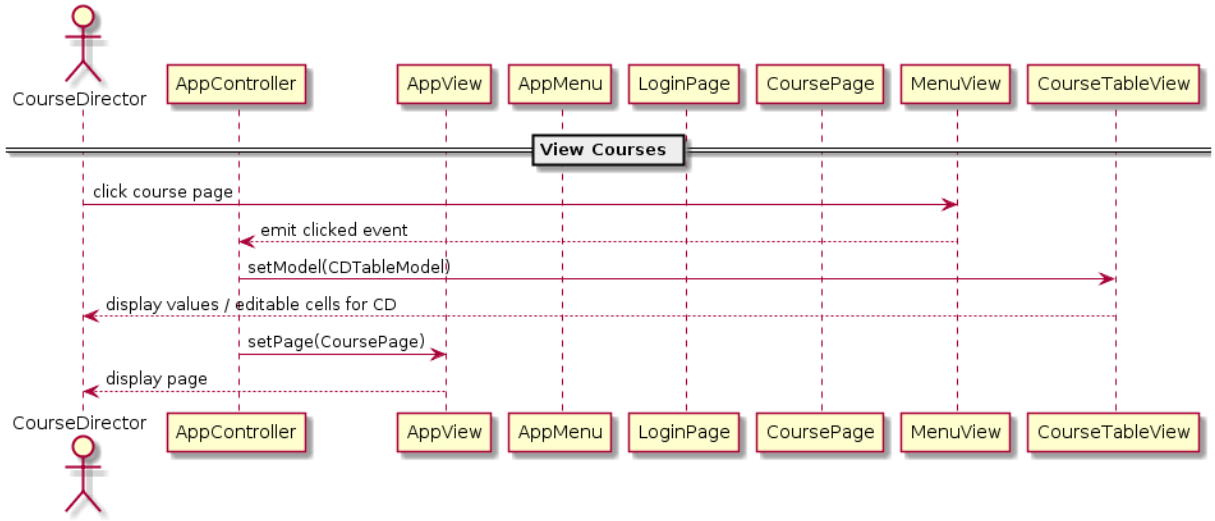
10. As an administrator, I can assign teachers to training courses for qualifications.



11. As a user I want to be able to navigate the app using a menu bar



12. As a course director I want to see all of my courses in a table



User Story Estimate vs Real Effort

User Story	Estimated Time	Actual Time
1. As a Course Director, I want to be able to create courses. These courses will have a name and a list of requirements.	4 Hours	10 Hours
2. As a Course Director, for each course I own, I want to be able to create a list of teaching requirements. These requirements are: the qualifications needed to teach the course.	5 Hours	3 Hours
3. As a User, I can add and remove qualifications to the database.	11 Hours	12.5 Hours
4. As an administrator, I can assign available teachers to a course when they hold the relevant qualifications.	9 Hours	9 Hours
5. As a course director, I want to be able to remove one of my courses	4 hours	4 Hours
6. As a PTT director, I want to view the courses, their requirements, the assigned staff and be able to approve the course.	4 Hours	2 Hours
7. As an administrator, I can view, add and remove teachers from the system.	6 Hours	5 Hours
8. As a user, I want to be able to login and only see an interface appropriate to my role.	5 Hours	12 Hours
9. As a user, when I logout, the data in the program is saved to a file. When I log back in, this data is reloaded into the system.	10 Hours	7 Hours
10. As an administrator, I can assign teachers to training courses for qualifications.	4 Hours	3 Hours
11. As a user I want to be able to navigate the app using a menu bar	8 Hours	9 Hours
12. As a course director I want to see all of my courses in a table	4 Hours	2 Hours

The time estimates vs actual time spent can also be seen on our Trello cards.

Screenshots of Program

Our full program has been submitted alongside this report for review; however, we will use this space to give a brief overview of our system's functionality. It has been implemented using Java Swing.

The program opens with a menu that allows the user to select their role in the organisation, as shown in Figure 1.

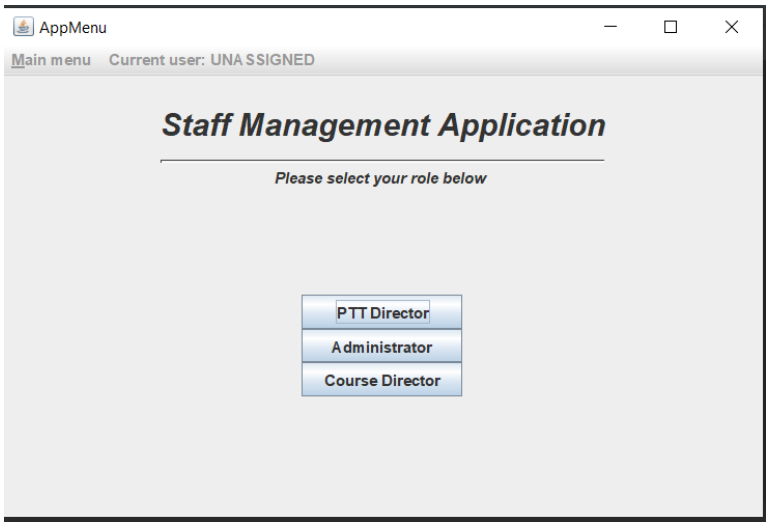


Figure 1 - The login screen

Once a role is selected, the user can only see and interact with parts of the system relevant to them. For example, when the Course Director role is selected the user sees the following panel, as shown in Figure 2.

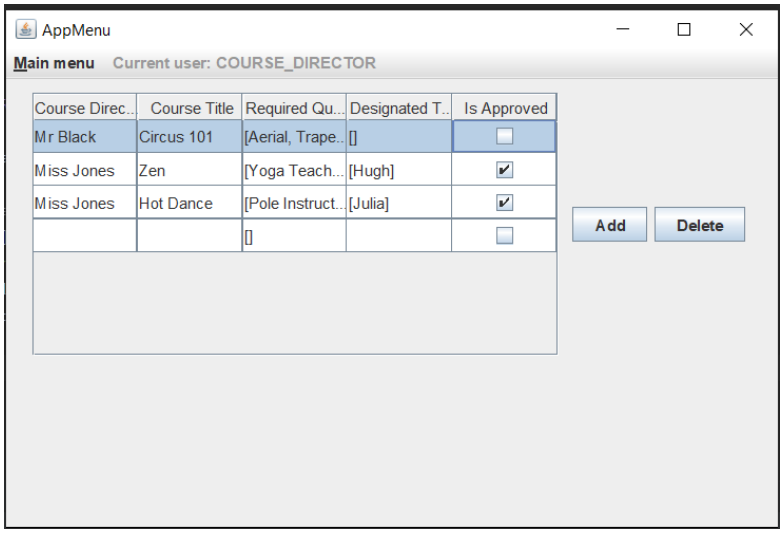


Figure 2 - The Course Table Editor

Here, the table contains the details of each course: the director, the title, the required qualifications, the designated teachers (which can only be changed by the Administrator), and whether it is approved (which can only be changed by the PTT Director). The Course Director has an extra functionality the other users do not: on the right hand side of the panel she can see 'Add' and

'Delete' buttons. This allows her to add or delete courses from the table. This creates a default 'Course' object, which can then be edited by editing its cell's row, usually by double clicking the cell to display the editor for that attribute.

When a cell represents a list selection from another list in the database, for example 'Required Qualifications' referencing the Qualification list, clicking this cell will provide a popup menu where a selection can be made from said list. This menu is synchronised with the Qualifications table, so when Qualifications are added, the dialog popup is updated too. This is shown in Figure 3. This list selection functionality is the same in other tables. A notable variant is 'Designated Teachers', where the list is filtered to only allow teachers holding the required qualifications to be selected, as shown in Figure 4.

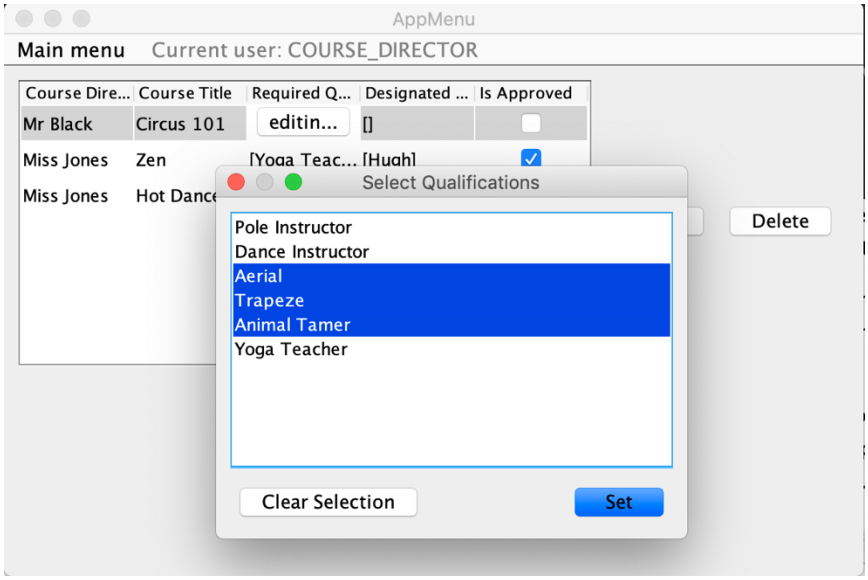


Figure 3 - A Dialog List Selector Synchronised with the Qualification Table

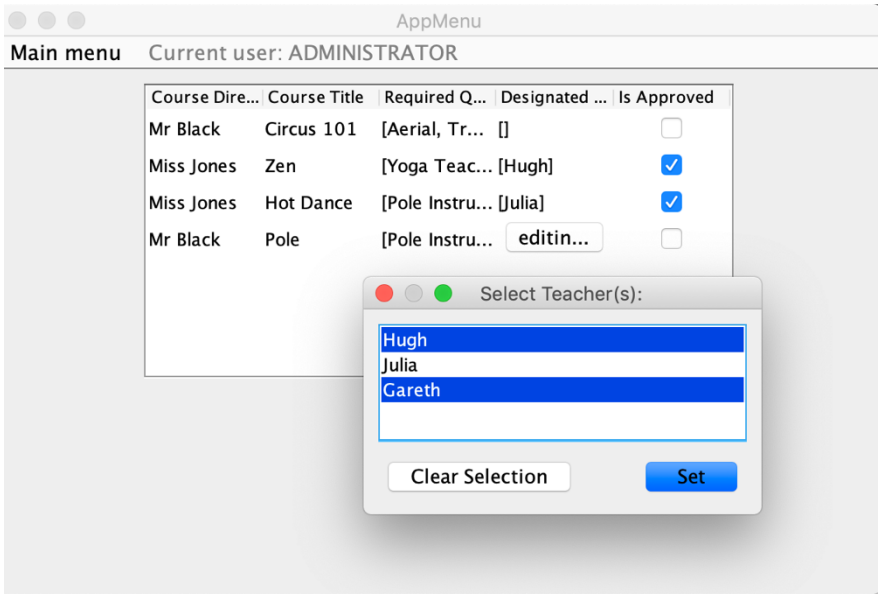


Figure 4 - A Filtered List for Teachers Holding the Pole Qualification

The Main Menu button at the top enables us to navigate through the rest of the system. The options available in the drop-down are: Home, Qualifications, Teachers, Courses and Training. Each of those views are shown below (again, their functionality is restricted depending on the user role, being either 'read only' or editable).

The Qualifications page shows all possible qualifications and includes functionality for adding or deleting, again, creating a new row which displays a default Qualification object which can be edited. This is shown in Figure 5.

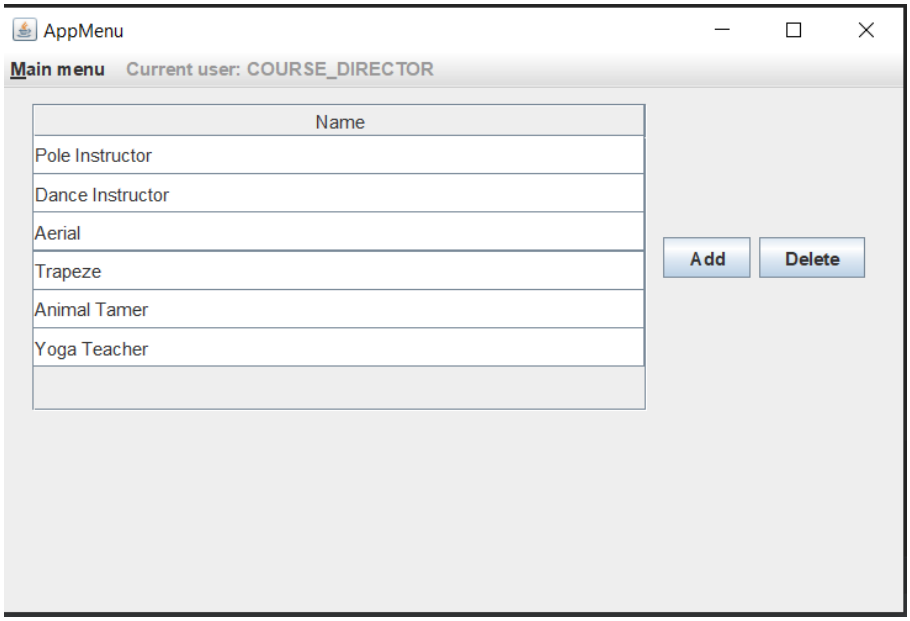


Figure 5 - The Qualification Table View

The Teachers page shows all teachers on the faculty, alongside their individual qualifications and their assigned training, as shown in Figure 6.

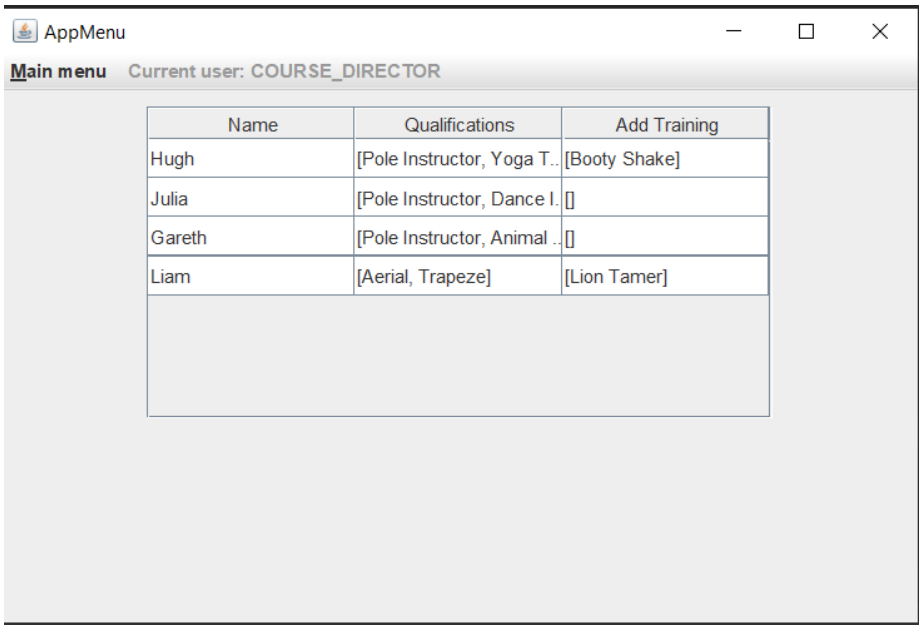


Figure 6 - The Teacher Table View, with Assigned Training and Qualifications

And finally the training view can be seen in Figure 7 which has “Add” and “Delete” options for the Administrator user.

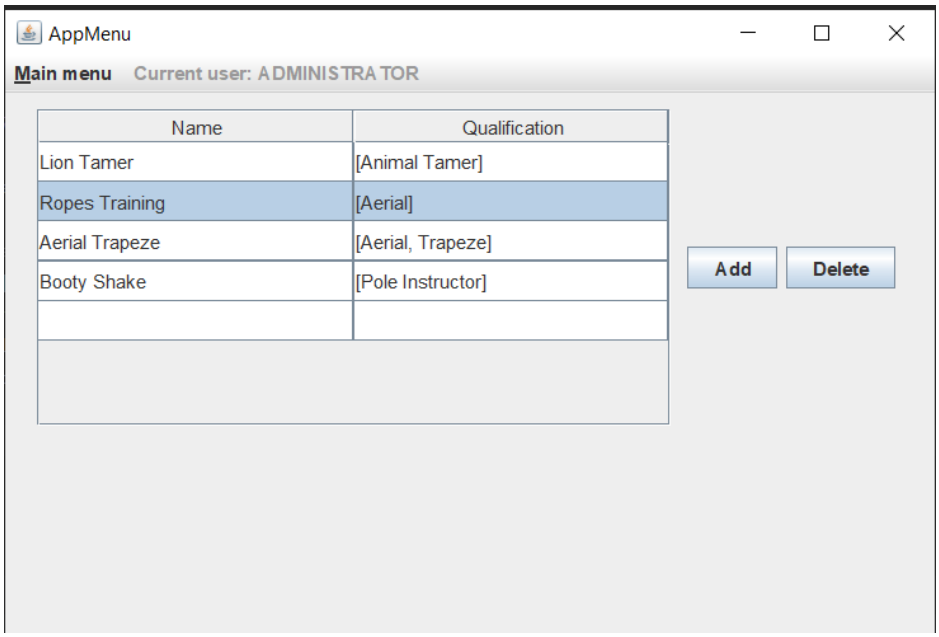


Figure 7 - The Training Table View (Editable for Administrator)

The full program has been submitted alongside this report. It is a Maven project which can be built using Maven, but for convenience, a runnable .jar file (‘happy-hippos-group-project.jar’) has been included which contains all the required dependencies as well as a sample ‘model.json’ with an example database. This ‘model.json’ needs to be in the same directory as the .jar when run.

Retrospective

During this assignment, we implemented Scrum principles in the design, planning and execution of the application. We began the assignment by holding thorough planning meetings, during which we decided on the user stories, application functionalities and a rough class structure.

The user stories that were written were divided into smaller tasks. We attempted to make the tasks represent coding functionalities and classes. This allowed us to divide up the tasks between team members. To ensure that people stayed within their assigned stories and the work wasn't duplicated, we assigned classes so that each person was responsible for developing within these stories.

By dividing the stories into tasks and giving each individual task a cost estimation, it was easier to obtain overall cost estimations for the user stories. We found that this process allowed us to make accurate cost estimations in terms of time.

We found it challenging to divide the tasks prior to beginning development, as many implementation decisions had yet to have been made. We found that as we progressed through the development process, we had to revisit the tasks to amend them to better represent our code structure.

As a part of scrum, we held regular stand-up meetings, during which we discussed the implementation of user stories and task delegations. We also used this time to discuss any problems we were facing and come up with solutions to these. The group also kept in contact using WhatsApp, Trello and GitHub so that any problems that were noticed outside of these meetings could be immediately remedied.

As we spent a significant amount of time planning and designing the application in the beginning, we found that the actual coding of the application came together quickly. We did not face many issues during development and did not have to undergo major code refactors. During this design phase, we found a number of patterns would be advantageous and allow us to reuse as much code as possible. Firstly, we took advantage of Swing's observer pattern in its DefaultListModels to synchronise our customised ObjectTable and ObjectTableListSelector with the lists in the model. The disadvantage of this was the that we had a mix of Swing's ListModel with the usual Java List interface in our code. A future refactor would work toward creating our own observer variant as a List, and removing the dependence on the more rigid Swing variant. Another pattern which was very useful was an Adaptor pattern, where we inherited from Swing's AbstractTableModel to create our own table model which could be used to edit objects and their attributes inside of the JTable view. This was the purpose of the ObjectTableModelAdaptor class. This was immensely powerful, and allowed us to refactor our code to highly reusable view components. A final pattern of note that we used was the Builder pattern. Initially, building the ObjectTableModels was a laborious task, as the column constructors required numerous variables, including potentially complex lambda functions. We used the Builder pattern to mitigate this, where the ObjectTableColumnBuilder allowed us to use useful defaults and created a less error prone implementation. Overall, the time spent in determining the best patterns to use and taking the time to implement them saved us many hours of work down the line and resulted in very clean code.

At the end of each sprint, we conducted a retrospective and discussed what had been going well, and what could be improved, both with our code and our workflow. This proved to be a beneficial

activity, as during the first retrospective, we identified that some of the code being used was repeated throughout the application. As this was a breach of the DRY principle, we decided to refactor so that this duplication was reduced.

The development of the app was planned to be done in two sprints; however, it was done in less time than anticipated and completed in only 1.5 weeks.

As part of our workflow we used GitHub. GitHub allowed us to forensically examine and review each other's code before it was merged with the remote; ensuring errors were removed. This approach also allowed more experienced developers to share best practice when reviewing less experienced group member's code.