2019

PRCS252 Report

GROUP J

REECE DRAGE: 10339605
ADAM EDWARDS: 10554466
EUGENIO GUARINO: 10592152
GABRYEL MASON-WILLIAMS: 10574626

1. Project Management and Agile Framework

For the development of this project an application of the Agile development methodology was attempted, based on the 12 Agile development principles from the Agile Manifesto *(Hasan, 2017)*. While this attempt had many successes, with regular sprint meetings and reviews keeping a consistent flow of progress, reflection and improvement, not all aspects of Agile development were effectively applied.

Communication at the beginning of the project was limited, with the GitHub discussion board being unused early-on. Usage of the discussion board grew as the project progressed, though a lack of feedback and communication on many posts resulted in a slower application of changes for some user stories. Face-to-face communication was relatively high at many points in the project, however, with several meetings per-week. The frequency of these face-to-face meetings improved the quality of group communication and aided product development *(Cockburn, 2002),* allowing individual members to clearly discuss their intended solutions and produce more effective software due to the collaboration. These meetings were used primarily for planning and adjusting the product backlog, discussing user requirements and solutions to technical problems.

Despite the limited communication, the production of group profiles (See profiles directory on the GitHub repository) for each member aided interactions. Each member of the group had a profile containing preferred working hours and communication times, allowing group members to consider arrangements around the personal schedules of other members. This eased communication between group members and allowed them to collaborate more effectively, being able to consider relative strengths and weaknesses and using them to adjust task assignments.

Adaptability became incredibly important very early in the project lifespan. The early development of the API was problematic, with inexperience and technical problems resulting in a heavily slowed development in the initial sprint. This resulted in a re-estimation and adjustment of the product backlog, as basic login functionality was harder to produce than initially believed.

The production of a Minimum Viable Product (MVP) was kept in mind throughout the project, with assigned user requirements for all sprints being based around the development of a usable, business-focused piece of software *(Hasan, 2017)*. Issues with the task assignments did arise during later sprints. Some user requirements were reliant on other requirements, notably the production of a booking/ticket table on the database for one requirement directly impacted and slowed the development of a front-end function that utilised it.

Some user story assignments were problematic due to technical limitations for some members of the group. Difficulties with connecting and modifying the API resulted in some members being unable to completely implement some functionality, having to implement the front-end application without the underlying API and database interactions. Overall, cooperation on several user stories became a key focus, with the API and the front-end application being developed simultaneously by different group members.

Communication again became problematic towards the end of the project. Changes made to the database resulted in issues with front-end functionality and were eventually reverted to restore functionality. This resulted in lost development time that could be allocated to other tasks. This was later resolved by communicating issues between members and ensuring changes were not made without prior communication to other group members.

Difficulties arose with members failing to consider the MVP, resulting in additional functionalities being developed despite them not fitting into the current sprint plan for the fifth sprint. This resulted in development time and resources not being allocated to other development tasks during this sprint.

Overall, despite issues faced, the project was a success. The MVP was reliably produced at the end of every sprint and a functionality was continuously produced and implemented in the agile development framework correctly. The majority of issues outside of communication involved conflicts in coding styles and ideologies. The group lacked a unified vision of the expected outcome, which at times impacted the functionality produced.

1.1. Risk Assessment

Risk	Occurrence	Responsible Parties	Grade Impact	Time Impact	Response
Not turning up to meetings	Mid	Adam/Gabryel	Low	Low	Contact staff after three consecutive missed pre-planned meetings
Conflict between group members	Low	Everyone	High	High	Hold a team meeting to resolve conflict issues
Weather conditions resulting in difficulty attending meetings	Low	Reece	Mid	Mid	Post on the GitHub discussion board in the event of inability to attend.
Equipment fault	Mid	Everyone	High	High	Contact staff to resolve issues.
Network issues	Mid	Everyone/Adam	Mid	Mid	Communicate when issues arise to allow other members to compensate
Oversight of task difficulty	High	Everyone	Low	High	Re-evaluate task assignments.
Family issues	Mid	Everyone	High	High	Inform group when issues arise. Take into consideration when assignments are made.
Misunderstanding with assigned tasks.	Mid	Everyone	High	High	Refer back to task assignments on project backlog. Communicate and clarify tasks to other group members.
Extension of tasks / “Gold Plating”	Mid	Everyone/ Eugenio	Low	High	Consult the backlog before the additional of additional features.
Changes to requirements	Mid	Everyone	Low	High	Discuss as a group before any changes to the requirements model.

Overall the risk assessment was effective for this project. A wide range of potential problems were anticipated, and appropriate responses were considered. While many of the risks never arose, the specific issues with equipment and network failures were a consistent issue, but the impacts were mitigated by the planned responses. In particular the network issues caused a lot of adjustment of sprint plans, moving API functionality to group members with more consistent success connecting to and modifying the API.

Misunderstandings and task difficulty oversight were also problems that arose, though to a lesser degree than the technical faults. These were also appropriately dealt with by consistently referring to and re-evaluating the project backlog to accommodate changes.

The “Extension of tasks” risk was not handled as effectively. The 5th sprint involved the production of some un-planned features for the minimum viable product, resulting in resources and time being pulled from other necessary functionality. The response to this risk was not sufficient to prevent the additional features from being developed. A better solution would have been to ensure consistent reference to the project backlog and repeated communication and discussion took place.

1.2. Sprint Reviews

Sprint Review Document	
Sprint Number: 1	Review Date: 04/03/2019

Overall Progress:
General overestimation of group workload resulted in less progress and a tighter deadline for the first sprint. API and network connectivity functionality proved more complex and harder to implement than the team initially predicted. POST request functions were a particular point of difficulty and were not fully implemented by the conclusion of the first sprint.

Issues:

- API creation and functionality was initially difficult to build.
    - Production of specialised SQL Queries was difficult. None of the queries built were kept in the final committed version.
    - Incorporation of database-end functions and procedures took longer than expected, though was eventually completed.
    - Server issues with the SOCEM server prevented hosting for the first sprint, instead local hosting was used for much of the functionality testing.
- POST requests to the API were difficult to implement, having been left incomplete at the end of the first sprint.
    - Errors with the request body when sent from the desktop application caused issues with the log-in function. Eventually GET was decided on as a temporary measure.
    - The mobile application had functioning POST requests due to the usage of the OKHTTP library.

Changes to be Made:

- Desktop login functionality to be improved with POST request at a later point.
- Sprints to be re-planned to accommodate heavier workload. Completion dates to be re-estimated.
- SOCEM issues to be discussed with product owner.

Sprint Review Document

Sprint Number: 2 | Review Date: 18/03/2019

Overall Progress:
Progress was more effective than the initial sprint due to more appropriate planning of task allocation and awareness of tasks to be completed. Problems still arose with API connectivity issues, but the timetable functionality planned was mostly completed. Web application functionality was not fully completed due to the aforementioned connectivity issues.

Issues:

- Lack of awareness or experience with android development slowed the production of a functional timetable for the android application, though it was completed on time.
    - Difficulty with multithreading caused several exceptions to be thrown at runtime.
    - Unfamiliarity with android UI caused timetable list development to be slow.
- API connectivity issues with Adam resulted in slowed progress on the web application.
    - Adam was unable to generate the controllers or model classes needed on the API.
    - CORS errors due to a mismatch between the API and the web interface security requirements produced issues with making requests from the web application.
    - Previously completed web application login functionality was broken by CORS errors.
- Junit tests on desktop application will not build and run

Changes to be Made:

- Adam will need to work from the university systems until the connectivity issue can be fixed.
- CORS issues with the API need to be investigated & fixed.
    - Discuss with Shirley.
- Resolve issues with Junit tests.
    - Possibly need to discuss with Shirley.

Sprint Review Document

Sprint Number: 3 | Review Date: 03/04/2019

Overall Progress:
Progress was restricted by the assignment of functional requirements that relied on other requirements being completed, resulting in an incomplete minimum viable product at the end of the sprint. Testing processes need to be reviewed to ensure a thorough selection of browsers for the web application, as issues went un-noticed for the current committed version.

Issues:

- Oversight on planning lead to some tasks being reliant on other tasks within the same sprint.
    - Gabryel and Adam had tasks reliant on a booking/ticket table that was completed at the end of the sprint, so these tasks were not able to be completed in time.
- Poor testing for browser compatibility with the web application meant that the completed functionality was not operating correctly on Mozilla Firefox.

Changes to be Made:

- Ensure future planning involves entirely independent requirements or functional requirements that rely on previously completed tasks.
- Ensure future testing and review processes involve thorough tests of different browsers and devices.

Sprint Review Document	
Sprint Number: 4	Review Date: 17/04/2019
Overall Progress: Progress was smooth with few issues. Task assignments were appropriate given the technical limitations Adam struggled with, resulting in effective development and production.	
Issues: • Small issues with some functions not removing tables with foreign keys before attempting to remove the primary record.	
Changes to be Made: • Ensure future development considers all linked tables as opposed to a single table with primary records.	

Sprint Review Document	
Sprint Number: 5	Review Date: 01/05/2019
Overall Progress: Issues with development of unassigned functionality and communication problems caused conflicts and lost progress. The project backlog needs to be consulted to ensure no time is dedicated to unnecessary functionality in the future.	
Issues: • Poor communication lead to lost changes as modifications to the database caused errors in existing functionality. The changes were later reverted to remove the issues. • Development of unassigned functionality removed resources from other aspects of development and lead to time being lost on functionality that was not part of the minimum viable product.	
Changes to be Made: • In the future, the project backlog and sprint plans need to be followed more closely and regularly used as reference. This will prevent the creation of features that aren’t defined in the minimum viable product. • Communication between group members needs to be improved to prevent conflict and prevent wasted time and lost progress.	

2.1. Project Backlog (Incomplete Tasks)

- Update a station record
  - System Admin
  - Priority: High
- QR code scanner
  - Guard
  - Priority: Low
- 16 digits REFERENCE_NO on tickets
  - Customer/Admin
  - Priority: Low
- Checkout as guest
  - Customer
  - Priority: Low
- Show timetable record
  - Controller
  - Priority: High
- Assign a guard to a specific station
  - Controller
  - Priority: High
- Assign a member of staff to a specific train
  - Controller
  - Priority: High
- Assign a specific train to a specific journey
  - Controller
  - Priority: High
- Create a new account for staff members
  - System Admin
  - Priority: High
- View how many people should be on the train
  - Driver
  - Priority: High
- Receive Notifications (Changes to personal timetable)
  - Driver
  - Priority: High
- Receive Notifications (Route Changes)
  - Driver
  - Priority: High

2.2. Completed Project Tasks

2.2.1. Sprint One

- Log-in with specific account details
  - Driver
  - Assigned To: Reece
  - Code Review Performed By: Gabryel
  - Estimated Sprint Completion: Sprint 1
  - Actual Sprint Completion: Sprint 1
  - Priority: High

- Log-in with specific account details
    - Guard
    - Assigned To: Everyone
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 1
    - Actual Sprint Completion: Sprint 1
    - Priority: High
- Log-in Securely
    - Cabin Crew
    - Assigned To: Adam
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 1
    - Actual Sprint Completion: Sprint 2
    - Priority: High
- Login by using the user's information
    - Customer
    - Assigned To: Eugenio
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 1
    - Actual Sprint Completion: Sprint 1
    - Priority: High
- Login with specific account details
    - Admin and Controller
    - Assigned To: Gabryel
    - Code Review Performed By: Eugenio
    - Estimated Sprint Completion: Sprint 1
    - Actual Sprint Completion: Sprint 1
    - Priority: High

2.2.2. Sprint Two

- Securely log-in with specific account details
    - System Admin and Controller
    - Assigned To: Gabryel
    - Code Review Performed By: n/a
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 2
    - Priority: High
- Remove a route record for a cancelled route
    - System Admin
    - Assigned To: Adam
    - Code Review Performed By: n/a
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 3
    - Priority: High

- Add a new route record
    - System Admin
    - Assigned To: Adam
    - Code Review Performed By: Eugenio
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 3
    - Priority: High
- Show Train Timetable
    - Staff
    - Assigned To: Eugenio
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 2
    - Priority: High
- Create Hash function for Customer table
    - Customer
    - Assigned To: Gabryel
    - Code Review Performed By: Reece
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 2
    - Priority: Mid
- Show Journeys
    - Customer
    - Assigned To: Reece
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 2
    - Actual Sprint Completion: Sprint 2
    - Priority: High

2.2.3. Sprint Three

- Associate a reference code to every purchased journey
    - Customer
    - Assigned To: Everyone
    - Code Review Performed By: Everyone
    - Estimated Sprint Completion: Sprint 3
    - Actual Sprint Completion: Sprint 3
    - Priority: High
- Validate Ticket
    - Cabin Crew
    - Assigned To: Gabryel
    - Code Review Performed By: Adam
    - Estimated Sprint Completion: Sprint 3
    - Actual Sprint Completion: Sprint 3
    - Priority: High

- Allow the users to search for a journey
    - Customer
    - Assigned To: Eugenio & Reece
    - Code Review Performed By: Eugenio & Reece
    - Estimated Sprint Completion: Sprint 3
    - Actual Sprint Completion: Sprint 3
    - Priority: High
- Buy tickets
    - Customer
    - Assigned To: Adam
    - Code Review Performed By: Eugenio
    - Estimated Sprint Completion: Sprint 3
    - Actual Sprint Completion: Sprint 4
    - Priority: High

2.2.4. Sprint Four

- Manage/modify existing route records
    - Controller
    - Assigned To: Gabryel
    - Code Review Performed By: Eugenio
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 5
    - Priority: High
- Add a new train record
    - System Admin
    - Assigned To: Reece
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: High
- Create sessions for webpages so pages are hidden before log-in
    - Assigned To: Gabryel
    - Code Review Performed By: Adam
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: Mid
- Add a new station record
    - System Admin
    - Assigned To: Eugenio
    - Code Review Performed By: Reece
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: High
- Remove a train record for a train
    - System Admin
    - Assigned To: Reece
    - Code Review Performed By: Gabryel
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: High

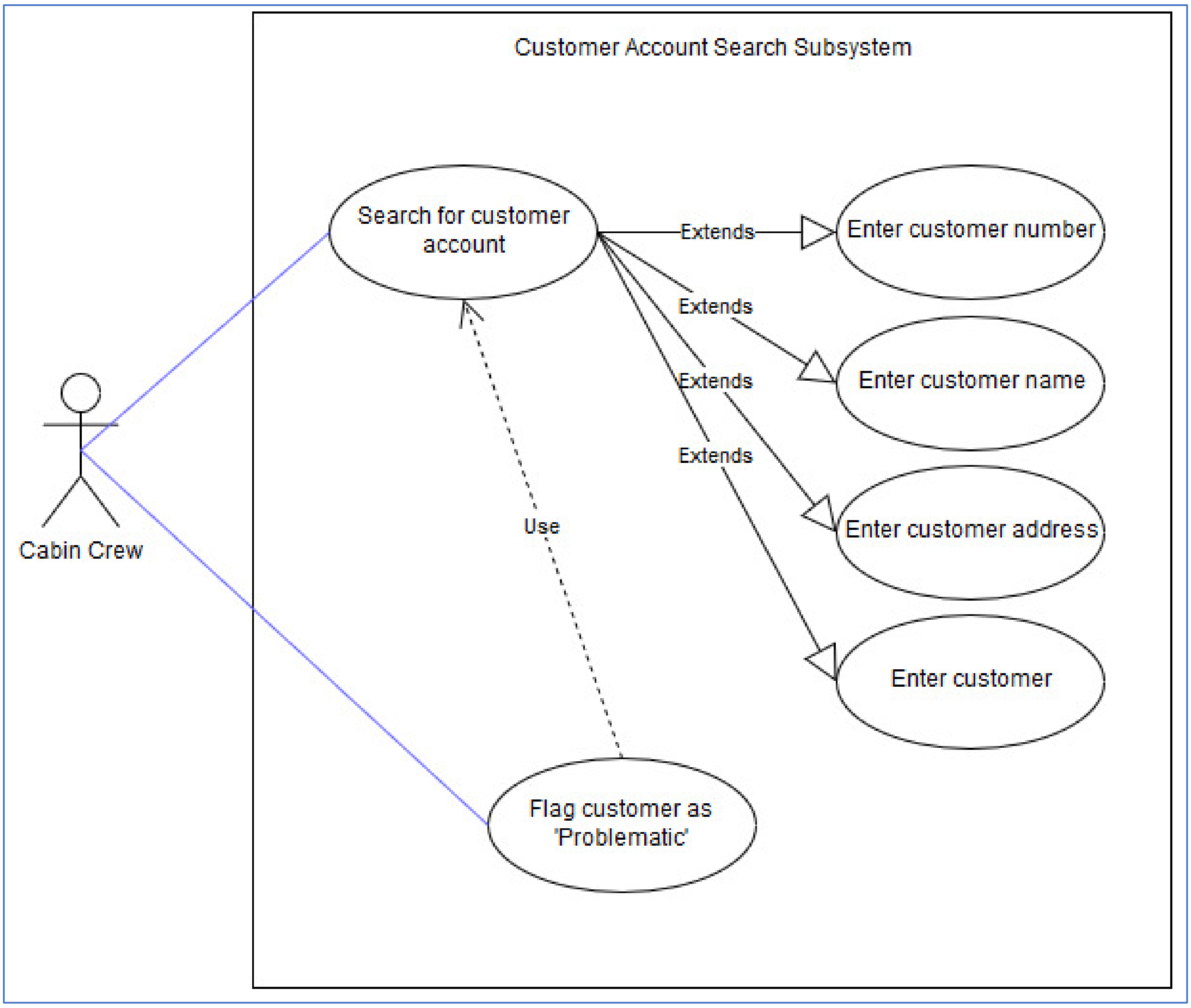
- Conform Website to HCI principles
    - Assigned To: Eugenio
    - Code Review Performed By: Everyone
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: Mid
- Update a train record
    - System Admin
    - Assigned To: Gabryel
    - Code Review Performed By: Reece
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: High
- Remove a station record
    - System Admin
    - Assigned To: Eugenio
    - Code Review Performed By: Reece
    - Estimated Sprint Completion: Sprint 4
    - Actual Sprint Completion: Sprint 4
    - Priority: High

2.2.5. **Sprint Five**

- Create logout function
    - Web application
    - Assigned To: Gabryel
    - Code Review Performed By: Eugenio
    - Estimated Sprint Completion: Sprint 5
    - Actual Sprint Completion: Sprint 5
    - Priority: Mid
- Integrate PayPal sandbox into mobile app
    - Customer
    - Assigned To: Adam
    - Code Review Performed By: Everyone
    - Estimated Sprint Completion: Sprint 5
    - Actual Sprint Completion: Sprint 5
    - Priority: Mid
- Mobile application improvements
    - Customer
    - Assigned To: Eugenio
    - Code Review Performed By: Reece
    - Estimated Sprint Completion: Sprint 5
    - Actual Sprint Completion: Sprint 5
    - Priority: Mid
- Show personal timetable
    - Staff
    - Assigned To: Gabryel & Reece
    - Code Review Performed By: Reece & Gabryel
    - Estimated Sprint Completion: Sprint 5
    - Actual Sprint Completion: Sprint 5
    - Priority: High

2.3. Use Case Model

2.3.1. Desktop Application Functionality (With interlinking functionality)

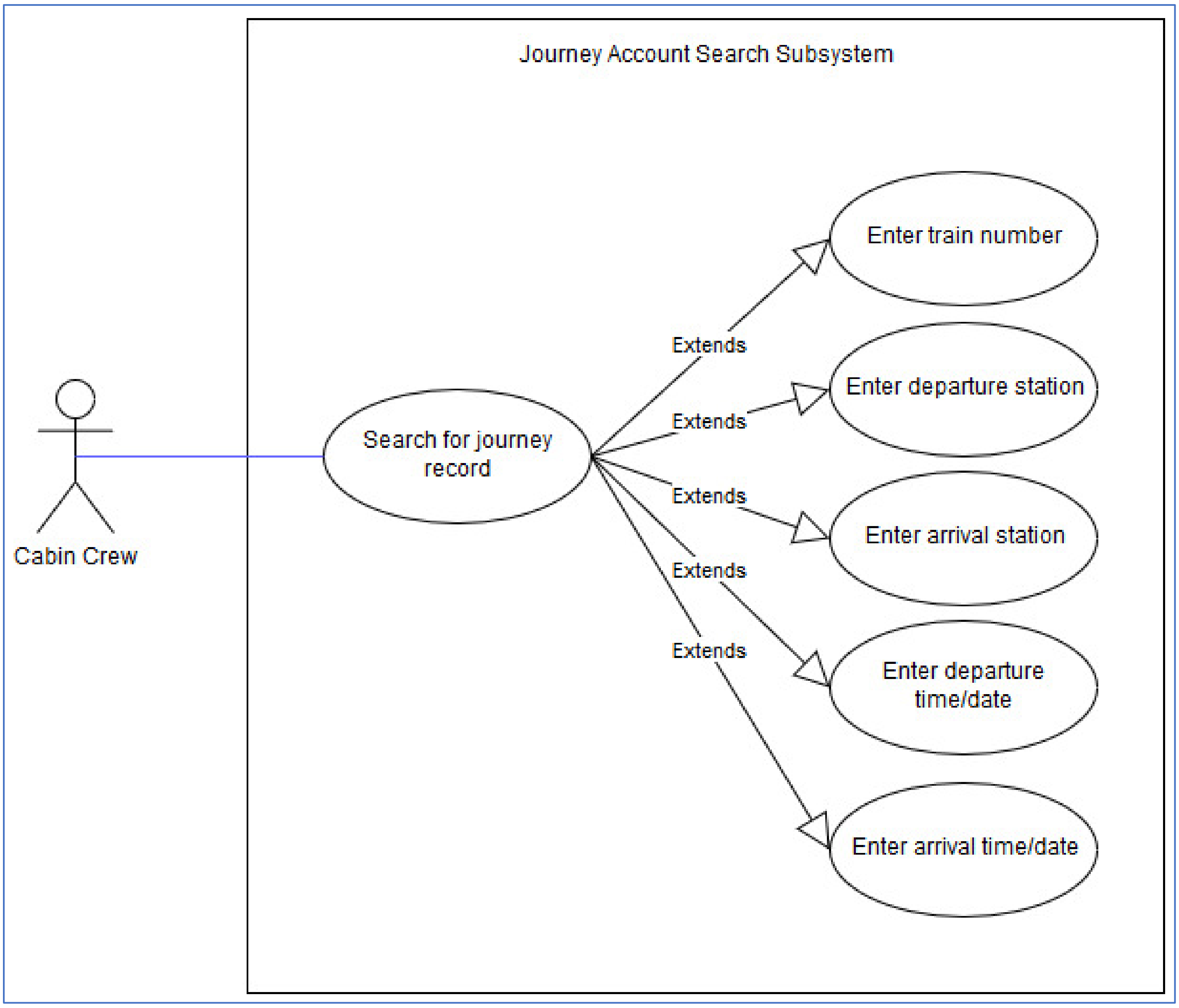


Use case description for "Search for customer account".

Actor Action	System Response
1. Actor enters Customer details in search fields.	2. Get customers with matching details in completed search fields. 3. Display relevant customer records.

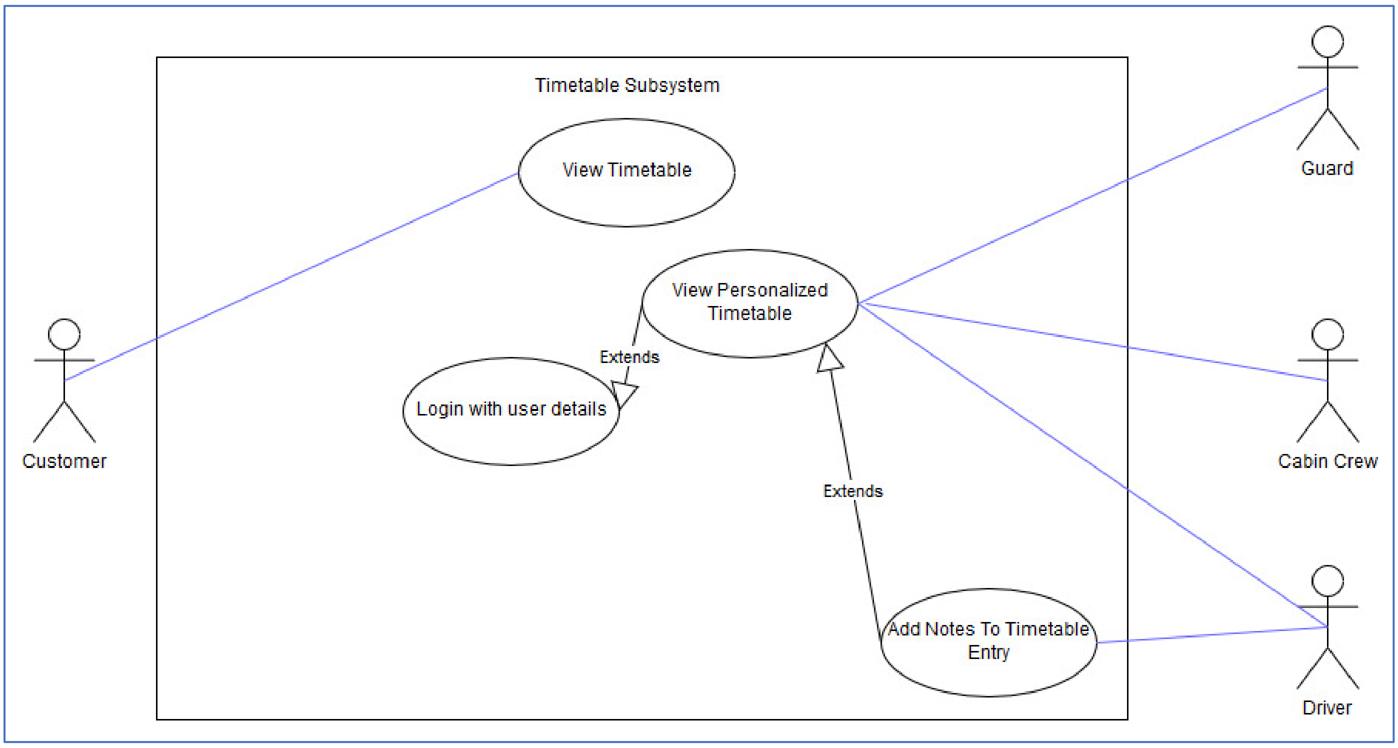
Use case description for "Flag customer as 'Problematic'".

Actor Action	System Response
1. Actor enters Customer details in search fields. 4. Actor selects specific customer to flag. 6. Actor chooses a reason to flag the customer. 7. Actor selects 'Submit'.	2. Get customers with matching details in completed search fields. 3. Display relevant customer records. 5. Display relevant customer details for chosen record. 8. Changes are committed to the system. 9. Confirmation message is displayed to the actor.



Use case description for "Search for journey record".

Actor Action	System Response
1. Actor enters journey details in search fields. 4. Select specific journey record from list.	2. Get journey records with matching details in completed search fields. 3. Display relevant journey records. 5. Display journey details for the chosen record.

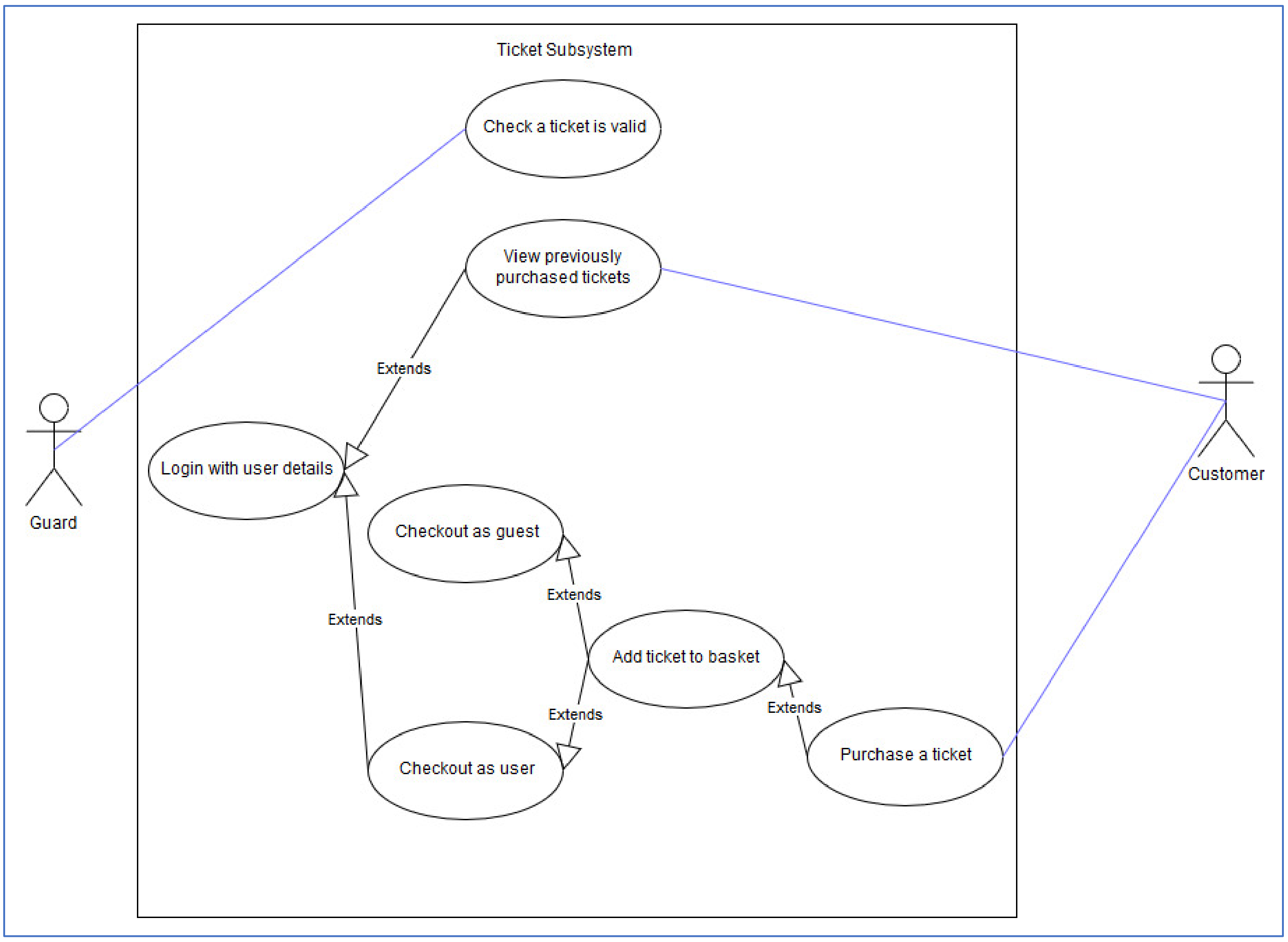


Use case description for "View timetable".

Actor Action	System Response
1. Actor logs into system.	2. Gets current Date. 3. Gets actor user ID number. 4. Gets journey records that the actor is assigned to based on the ID number. 5. Displays timetable with journey records for the current week.

Use case description for "Login with user details".

Actor Action	System Response
1. Actor Enters Username and Password.	2. Check users Details against database for user validation. 3. Retrieves actor user ID number. 4. Display feedback message. 5. Display Home screen.



Use case description for "Check a ticket is valid".

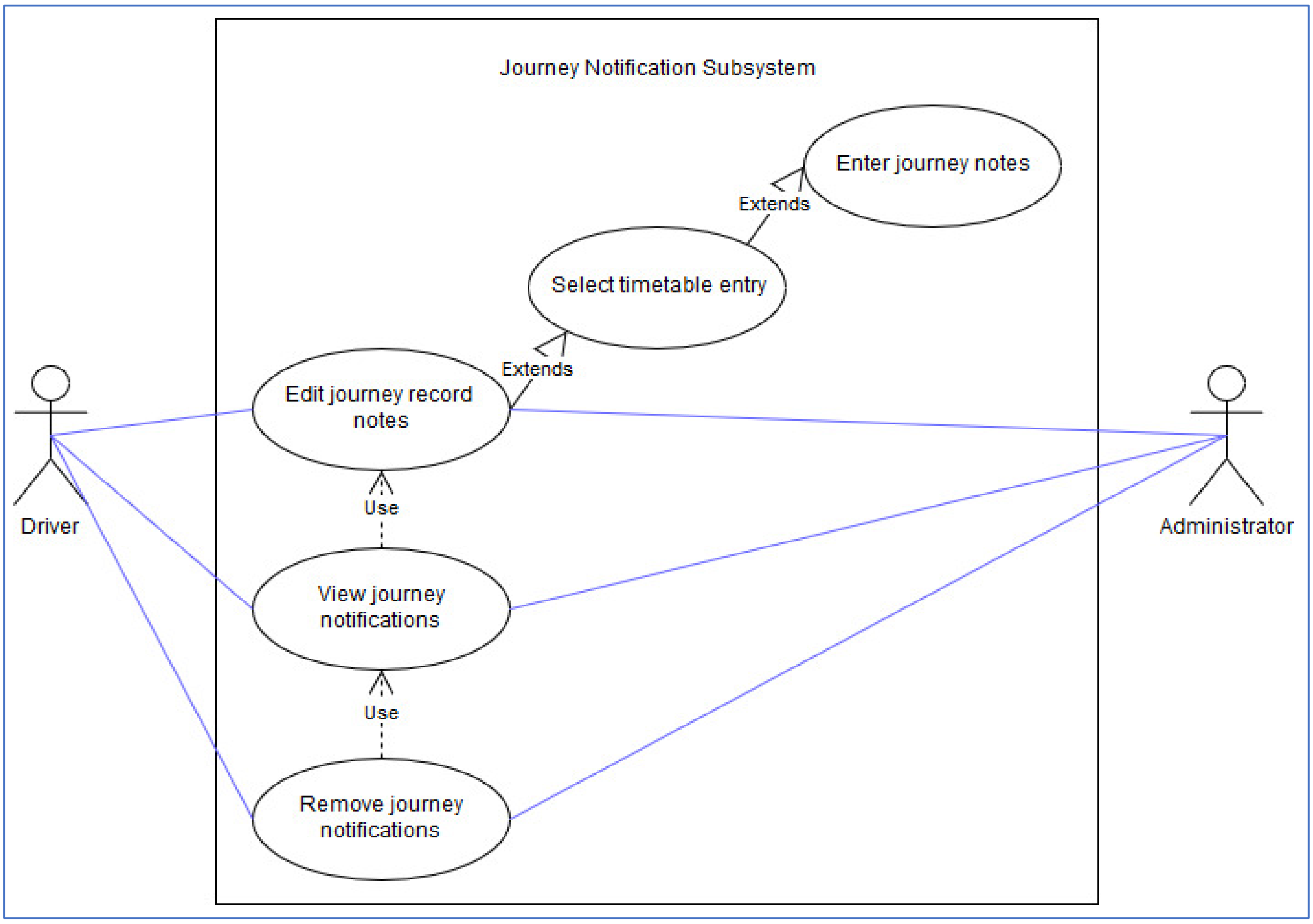
Actor Action	System Response
1. Actor Enters ticket ID	2. Locate Ticket in database 3. Get validation property 4. Display Valid/invalid feedback

Use case description for "Purchase a ticket".

Actor Action	System Response
1. Actor selects a journey. 2. Actor selects 'Purchase'. 4. Actor confirms purchase. 6. Actor enters payment details.	3. Displays confirmation message. 5. Prompts actor for payment details. 7. Processes payment details. 8. Returns message confirming that the journey has been purchased.

Use case description for "View previously purchased tickets".

Actor Action	System Response
1. Actor logs into system. 2. Actor selects "Ticket history" option.	3. Gets journey records with a booking record linked to the actor user ID. 4. Display journey records as a list.


Journey Notification Subsystem
Enter journey notes
Extends
Select timetable entry
Extends
Edit journey record notes
Use
View journey notifications
Use
Remove journey notifications
Driver
Administrator

Use case description for "Edit journey record notes".

Actor Action	System Response
1. Actor selects the "Edit" option on a specific journey. 3. Actor adds/modifies notes for the journey. 4. Actor selects the "Submit" button. 6. Actor confirms the modified journey notes.	2. Displays record details for the chosen journey. 5. System prompts actor for confirmation. 7. Notifies Controllers of change through addition of the note to a notification feed. 8. Presents a confirmation message that note has been added.

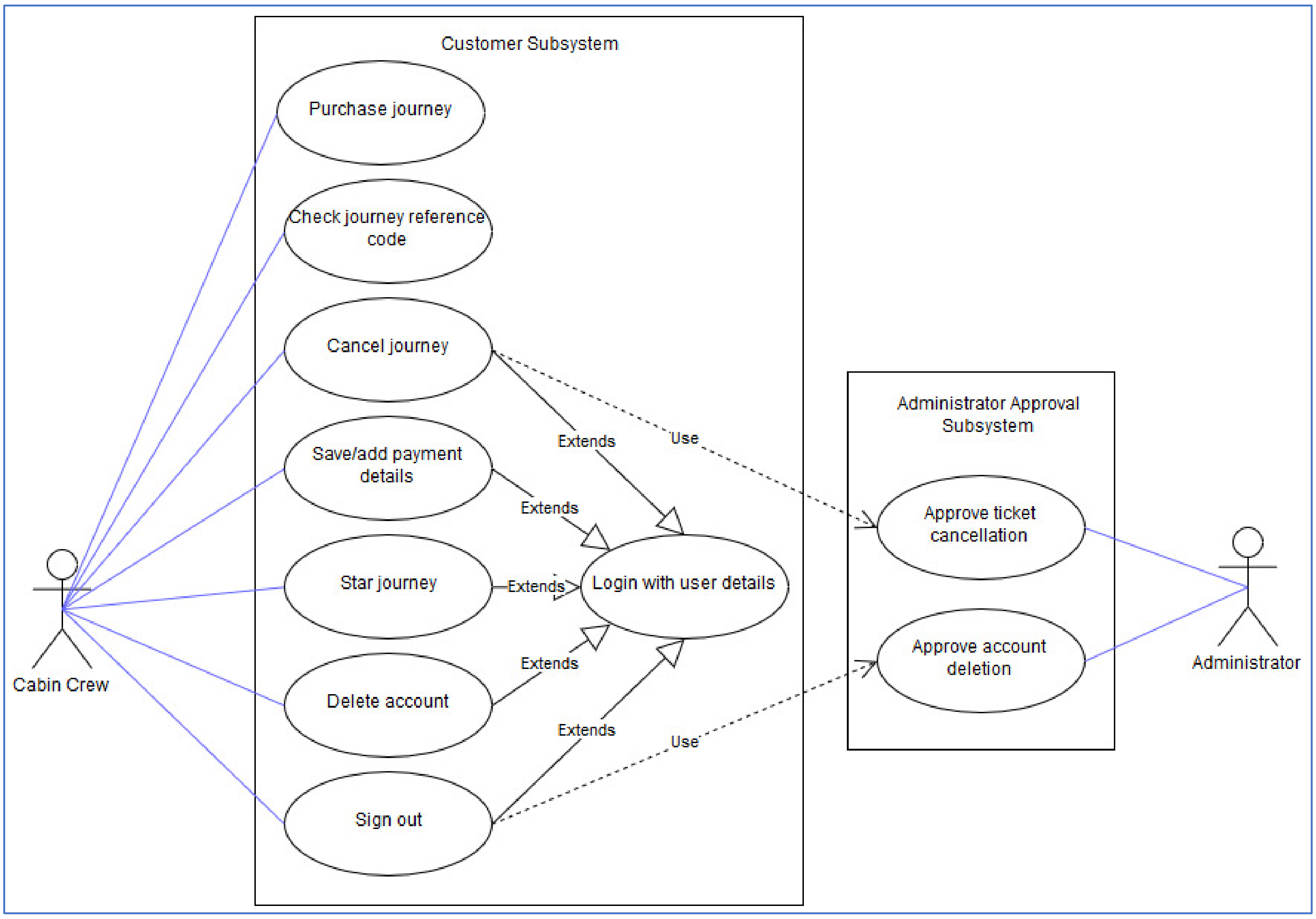
Use case description for "View journey notifications".

Actor Action	System Response
1. Actor logs in to system.	2. Gets notification records. 3. Displays notification records.

Use case description for "Remove journey notifications".

Actor Action	System Response
1. Actor selects "Remove" button on specific record 3. Actor confirms removal of record.	2. Prompts user to confirm notification removal. 4. Commits removal of record. 5. Displays a confirmation message that the notification has been removed.

2.1.2. Mobile Application Functionality (With interlinking functionality)



Use case description for "Purchase journey".

Actor Action	System Response
1. Actor selects a journey. 2. Actor selects 'Purchase'. 4. Actor confirms purchase. 6. Actor enters payment details.	3. Displays confirmation message. 5. Prompts actor for payment details. 7. Processes payment details. 8. Returns message confirming that the journey has been purchased.

Use case description for "Check journey reference code".

Actor Action	System Response
1. Navigate to the "Tickets" section. 3. Select a specific journey record.	2. Displays active and expired journeys. 4. Displays specific journey details for the selected record (Includes the journey reference code).

Use case description for “Cancel a journey”.

Actor Action	System Response
1. Navigate to the “Tickets” section. 3. Select a specific journey record to cancel. 5. Selects the “Cancel” button. 7. Confirms the action.	2. Displays active and expired journeys. 4. Displays journey’s description screen. 6. Displays confirmation message. 8. Sends cancellation request. 9. Returns message stating that the request has been sent and is waiting for approval.

Use case description for “Save/add payment details”.

Actor Action	System Response
1. Navigate to the “Profile” section. 2. Selects the “Add payment details” button. 4. Inputs payment information. 5. Selects “Submit”.	3. Displays payment details creation interface. 5. Returns message confirming that the details have been added.

Use case description for “Star a journey”.

Actor Action	System Response
1. Navigate to the “Tickets” section. 3. Selects the star icon situated on the journey record.	2. Displays active and expired journeys. 4. Adds journey to starred journeys. 5. Returns message confirming that the journey has been starred.

Use case description for “Delete account”.

Actor Action	System Response
1. Navigate to the “Profile” section. 2. Selects the “Delete Account” button. 4. Confirms action.	3. Displays confirmation message 5. Sends cancellation request. 6. Displays success message.

Use case description for “Sign out”.

Actor Action	System Response
1. Navigate to the “Profile” section. 2. Selects the “Sign out” button.	3. Redirects user to home screen. 4. Displays success message.

Use case description for “Login with user details”.

Actor Action	System Response
1. Navigate to “Login” section. 2. Inputs username and password. 3. Selects “Login” button.	4. Checks username and password with records on database. 5. Redirects user to home screen. 6. Displays success message.

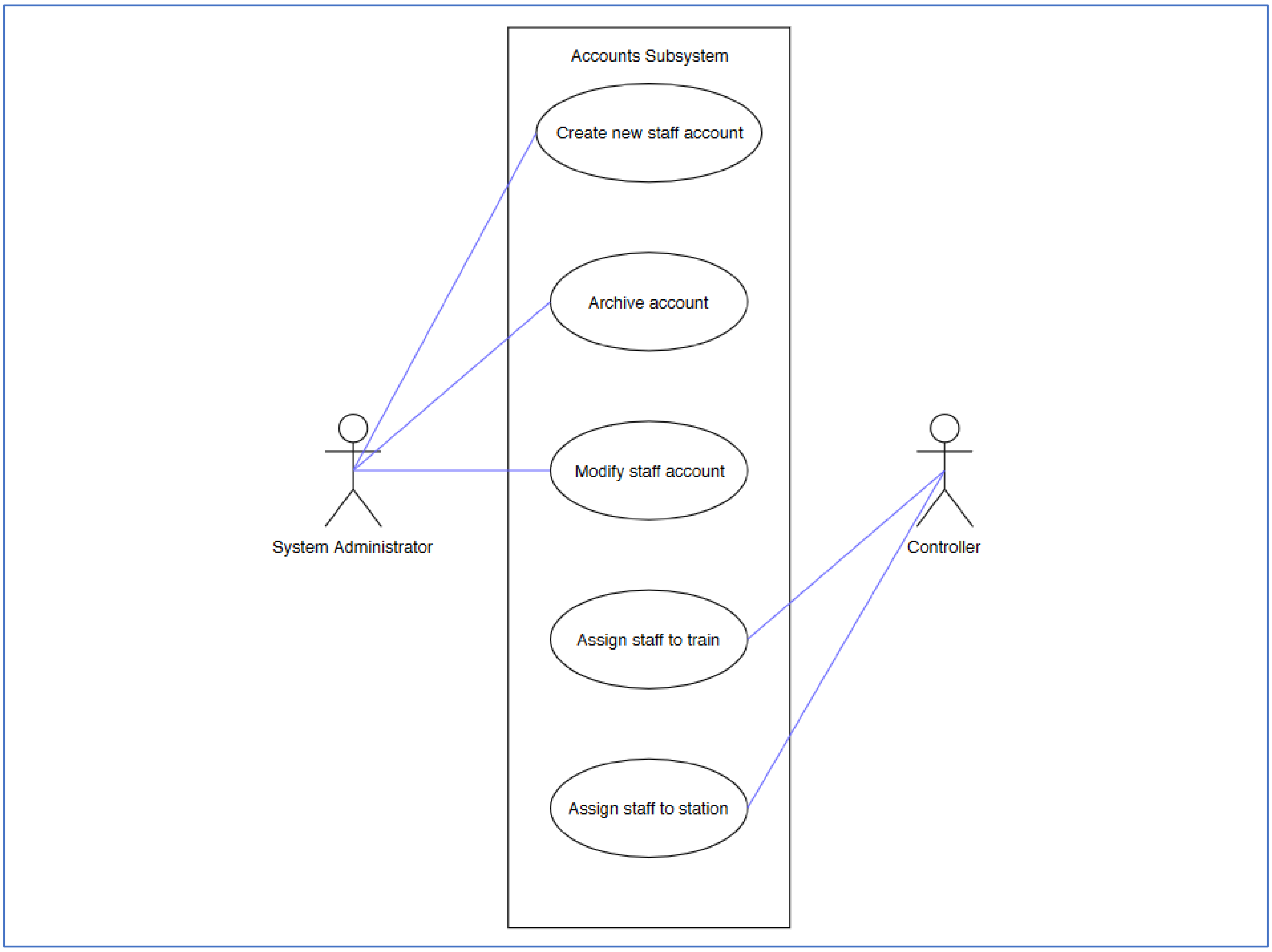
Use case description for “Approve ticket cancellation”.

Actor Action	System Response
1. Navigates to “Notification Feed”. 4. Actor selects either “Approve” or “Decline”. 6. Actor confirms action.	2. Adds notification of a ticket cancellation request to the notification feed. 3. Displays notification feed. 5. Displays confirmation message. 7. System authorizes or declines a cancellation request and commits it to the database. 8. Displays success message.

Use case description for “Approve account deletion”.

Actor Action	System Response
1. Navigates to “Notification Feed”. 4. Actor selects either “Approve” or “Decline”. 6. Actor confirms action.	2. Adds notification of an account deletion request to the notification feed. 3. Displays notification feed. 5. Displays confirmation message. 7. System authorizes or declines a deletion request and commits it to the database. 8. Displays success message.

2.1.3. Web application functionality (With interlinking functionality)



Use case description for "Create new staff account".

Actor Action	System Response
1. Actor enters account details. 2. Actor selects "Submit" button. 4. Actor confirms new staff account.	3. Displays confirmation message for new account creation 5. Returns message confirming that a new staff account has been created

Use case description for "Archive account".

Actor Action	System Response
1. Actor selects a user account 3. Actor selects "Archive Record" button 5. Actor confirms user account archival	2. Displays user account details 4. Displays confirmation message for user account archival 6. Returns message confirming that the account has been archived

Use case description for “Modify staff account”.

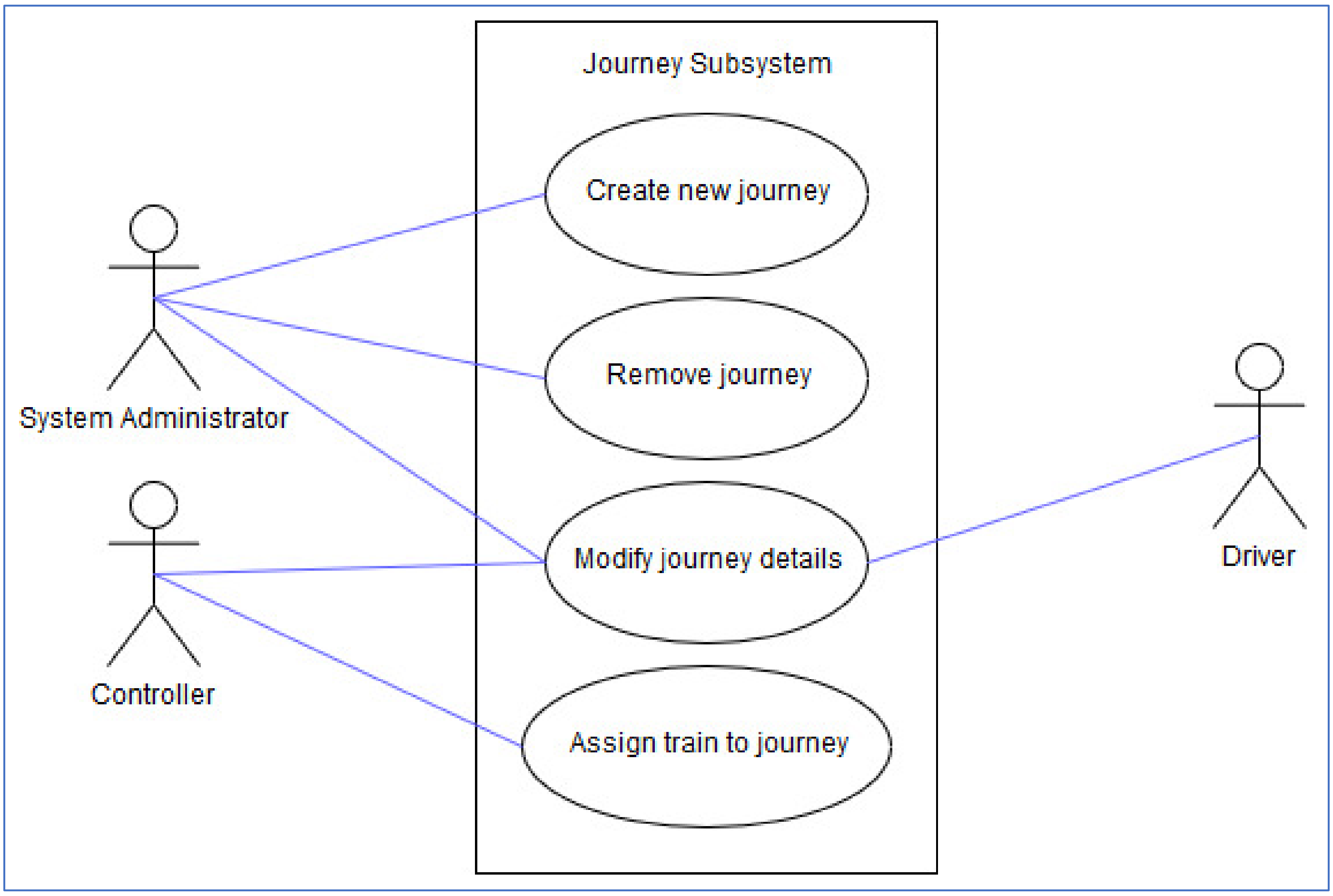
Actor Action	System Response
1. Actor selects a train record 3. Actor enters modified train details 5. Actor confirms train record modification	2. Displays train record details 4. Displays confirmation message for train record modification 6. Returns message confirming that the train record has been modified

Use case description for “Assign staff to train”.

Actor Action	System Response
1. Actor selects a train record. 3. Actor selects “Assign Crew” button. 5. Actor selects a staff account record. 7. Actor selects “Assign to Train” button. 9. Actor confirms Crew assignment.	2. Displays train record details. 4. Displays list of staff account records. 6. Displays staff account record details. 8. Displays confirmation message for crew assignment. 10. Returns message confirming that the staff member has been assigned to the specified train.

Use case description for “Assign staff to station”.

Actor Action	System Response
1. Actor selects a station record. 3. Actor selects “Assign Crew” button. 5. Actor selects a guard account record. 7. Actor selects “Assign to Station” button. 9. Actor confirms Crew assignment.	2. Displays station record details. 4. Displays list of guard account records. 6. Displays guard account record details. 8. Displays confirmation message for crew assignment. 10. Returns message confirming that the staff member has been assigned to the specified train.



Use case description for "Create new journey".

Actor Action	System Response
1. Actor enters journey details. 2. Actor selects "Submit" button. 4. Actor confirms new journey.	3. Displays confirmation message for new journey record creation 5. Returns message confirming that a new journey has been created

Use case description for "Remove journey".

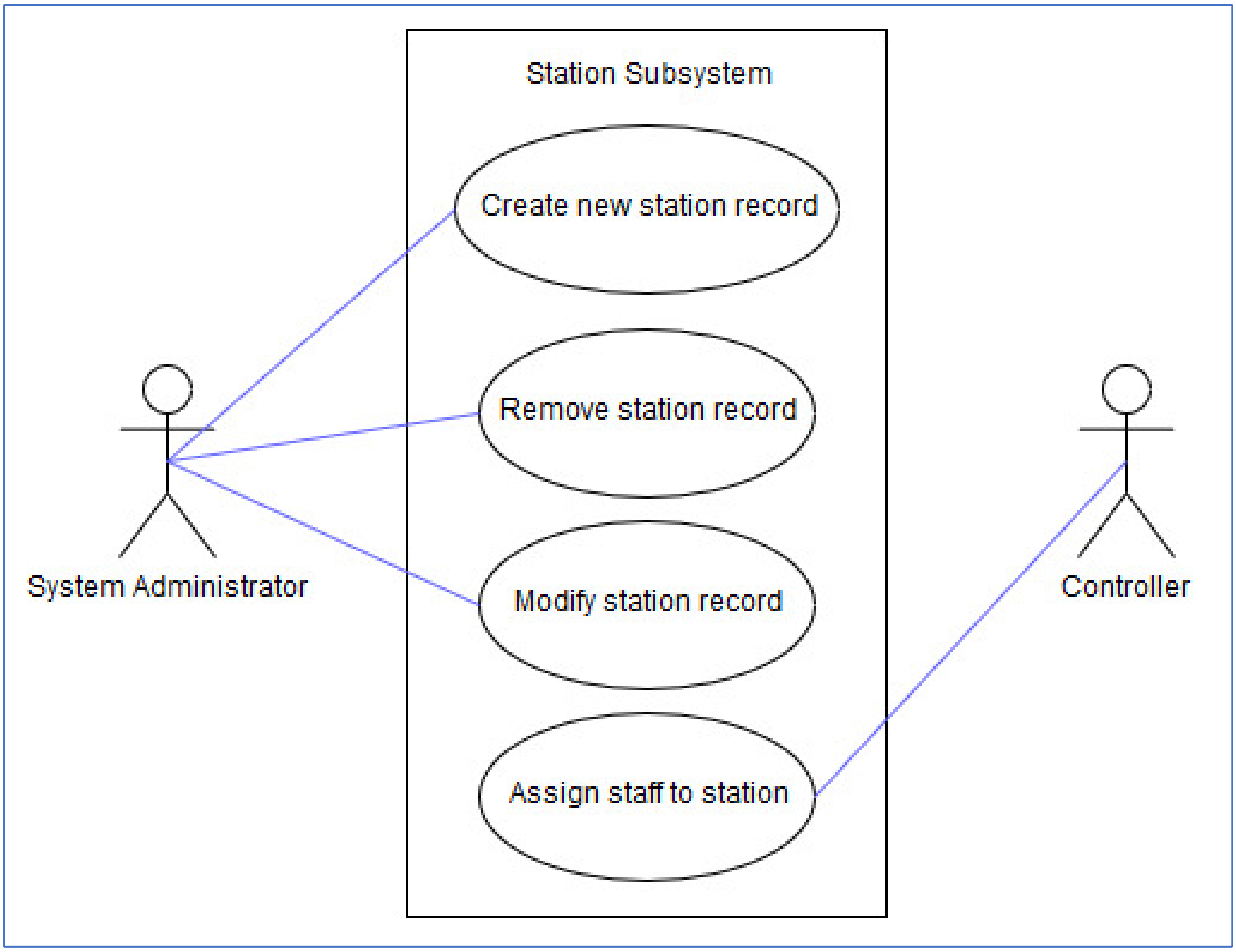
Actor Action	System Response
1. Actor selects a journey record 3. Actor selects "Delete Record" button 5. Actor confirms journey record deletion	2. Displays journey record details 4. Displays confirmation message for journey record deletion 6. Returns message confirming that the journey has been deleted

Use case description for "Modify journey details".

Actor Action	System Response
1. Actor selects a journey record 3. Actor enters modified journey details 5. Actor confirms Journey modification	2. Displays journey record details 4. Displays confirmation message for journey record modification 6. Returns message confirming that the journey has been modified

Use case description for “Assign train to journey”.

Actor Action	System Response
1. Actor selects a journey record. 3. Actor selects “Assign Train” button. 5. Actor selects a train record. 7. Actor selects “Assign to Journey” button. 9. Actor confirms train assignment.	2. Displays journey record details. 4. Displays list of train records. 6. Displays train record details. 8. Displays confirmation message for train assignment. 10. Returns message confirming that the train has been assigned to the specified journey.



Use case description for “Create new station record”.

Actor Action	System Response
1. Actor enters station details. 2. Actor selects “Submit” button. 4. Actor confirms new station record.	3. Displays confirmation message for new station record creation 5. Returns message confirming that a new station record has been created

Use case description for “Remove station record”.

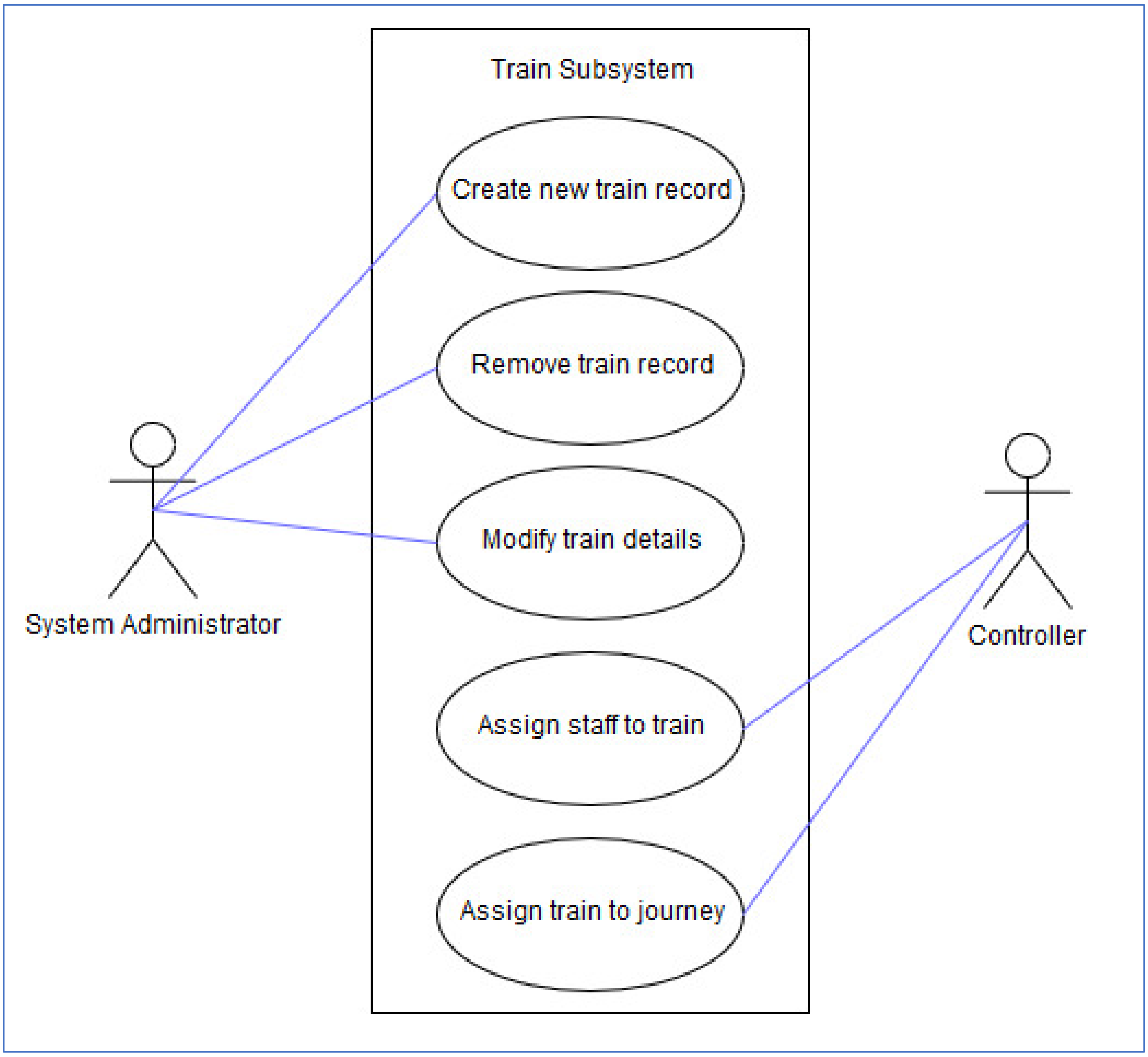
Actor Action	System Response
1. Actor selects a station record 3. Actor selects “Delete Record” button 5. Actor confirms station record deletion	2. Displays station record details 4. Displays confirmation message for station record deletion 6. Returns message confirming that the station has been deleted

Use case description for “Modify station record”.

Actor Action	System Response
1. Actor selects a station record 3. Actor enters modified station details 5. Actor confirms station record modification	2. Displays station record details 4. Displays confirmation message for station record modification 6. Returns message confirming that the station record has been modified

Use case description for “Assign staff to station”.

Actor Action	System Response
1. Actor selects a station record. 3. Actor selects “Assign Crew” button. 5. Actor selects a guard account record. 7. Actor selects “Assign to Station” button. 9. Actor confirms Crew assignment.	2. Displays station record details. 4. Displays list of guard account records. 6. Displays guard account record details. 8. Displays confirmation message for crew assignment. 10. Returns message confirming that the staff member has been assigned to the specified train.



Use case description for "Create new train record".

Actor Action	System Response
1. Actor enters train details. 2. Actor selects "Submit" button. 4. Actor confirms new train record.	3. Displays confirmation message for new train record creation 5. Returns message confirming that a new train record has been created

Use case description for "Remove train record".

Actor Action	System Response
1. Actor selects a train record 3. Actor selects "Delete Record" button 5. Actor confirms train record deletion	2. Displays train record details 4. Displays confirmation message for train record deletion 6. Returns message confirming that the train has been deleted

Use case description for “Modify train record”.

Actor Action	System Response
1. Actor selects a train record 3. Actor enters modified train details 5. Actor confirms train record modification	2. Displays train record details 4. Displays confirmation message for train record modification 6. Returns message confirming that the train record has been modified

Use case description for “Assign staff to train”.

Actor Action	System Response
1. Actor selects a train record. 3. Actor selects “Assign Crew” button. 5. Actor selects a staff account record. 7. Actor selects “Assign to Train” button. 9. Actor confirms Crew assignment.	2. Displays train record details. 4. Displays list of staff account records. 6. Displays staff account record details. 8. Displays confirmation message for crew assignment. 10. Returns message confirming that the staff member has been assigned to the specified train.

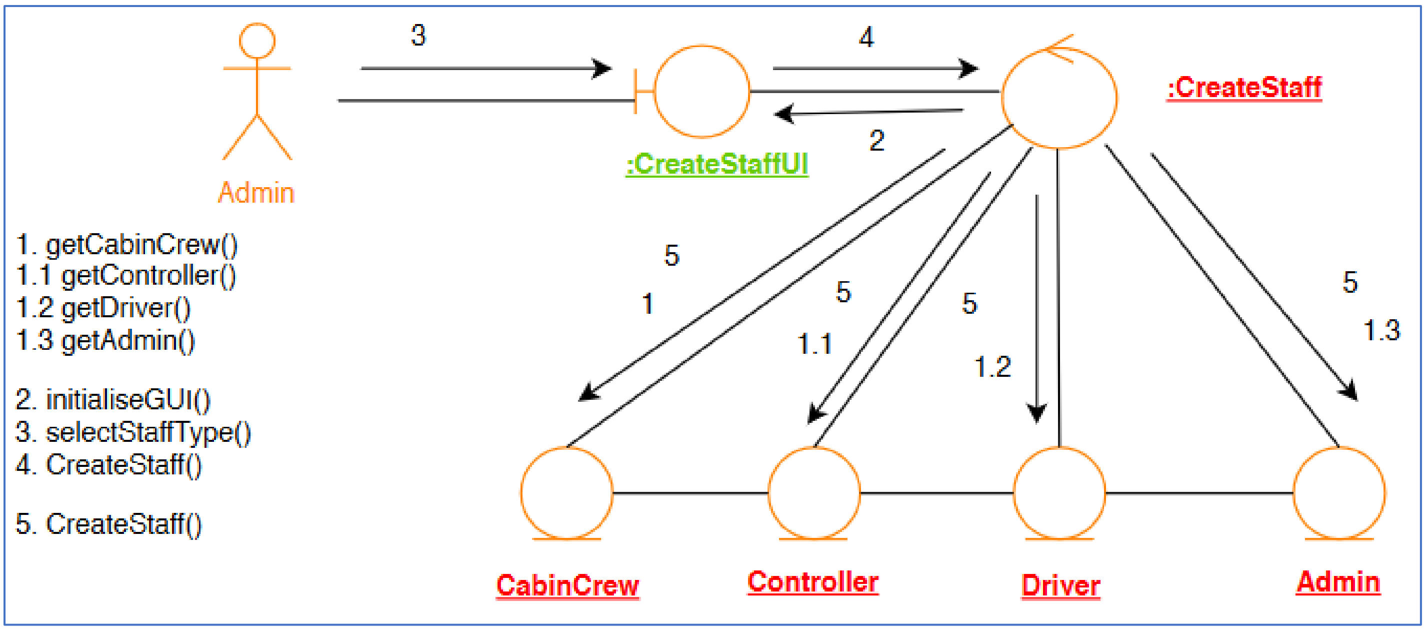
Use case description for “Assign train to journey”.

Actor Action	System Response
1. Actor selects a journey record. 3. Actor selects “Assign Train” button. 5. Actor selects a train record. 7. Actor selects “Assign to Journey” button. 9. Actor confirms train assignment.	2. Displays journey record details. 4. Displays list of train records. 6. Displays train record details. 8. Displays confirmation message for train assignment. 10. Returns message confirming that the train has been assigned to the specified journey.

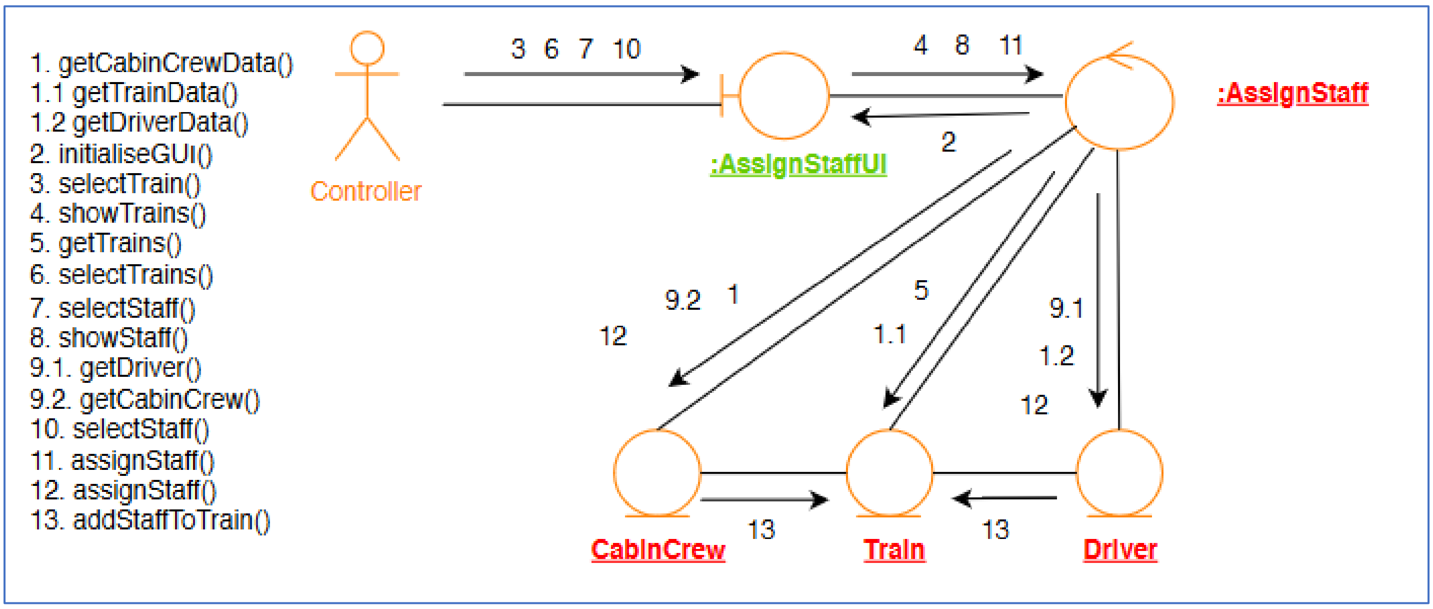
2.2. Use case Communication Diagrams

Use case communication Diagrams were created for 3 of the more complex functional requirements/user stories. These highlighted the general process to be carried out by each function during runtime.

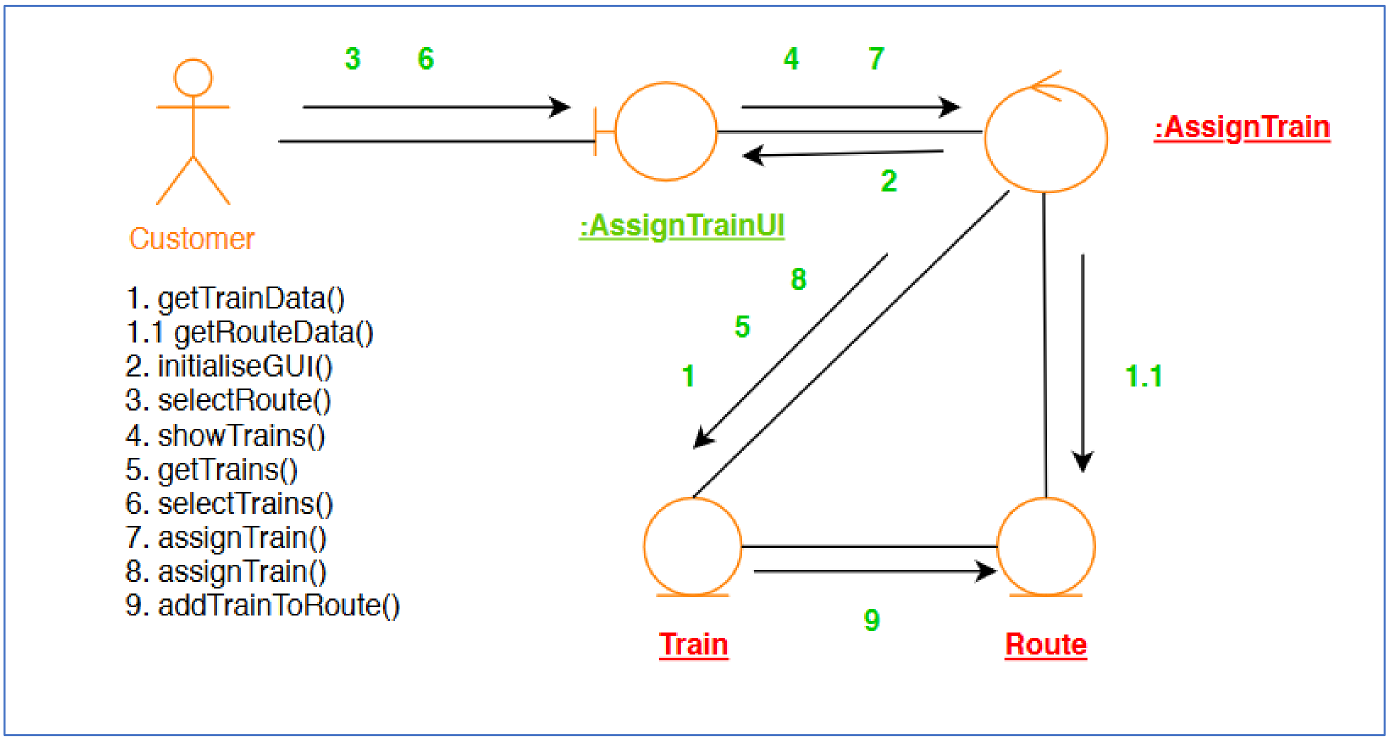
Create staff user account:



Assign staff to a train:

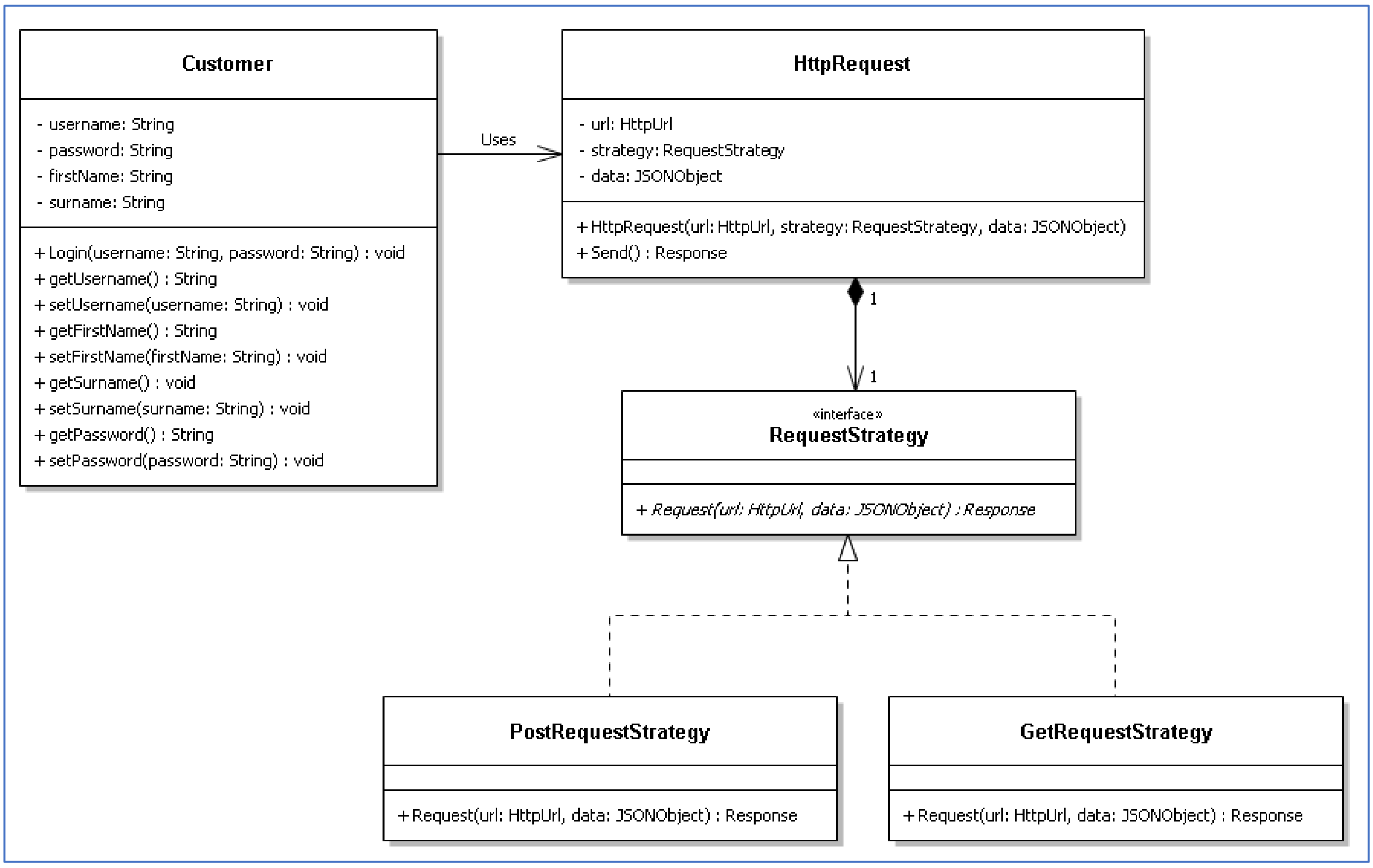


Assign train to journey:

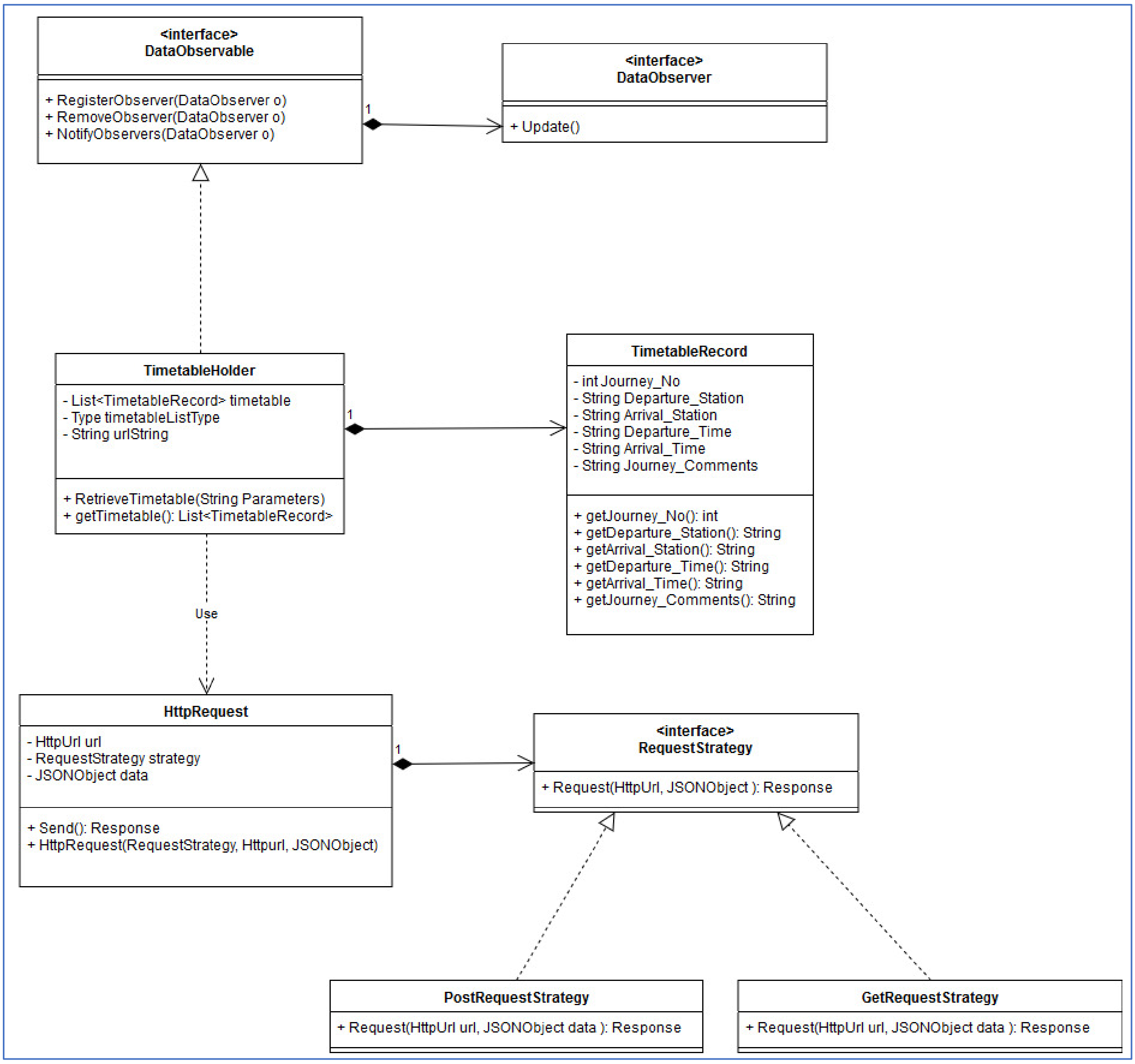


2.3. UML Class Diagrams

2.3.1. Login with Customer Details

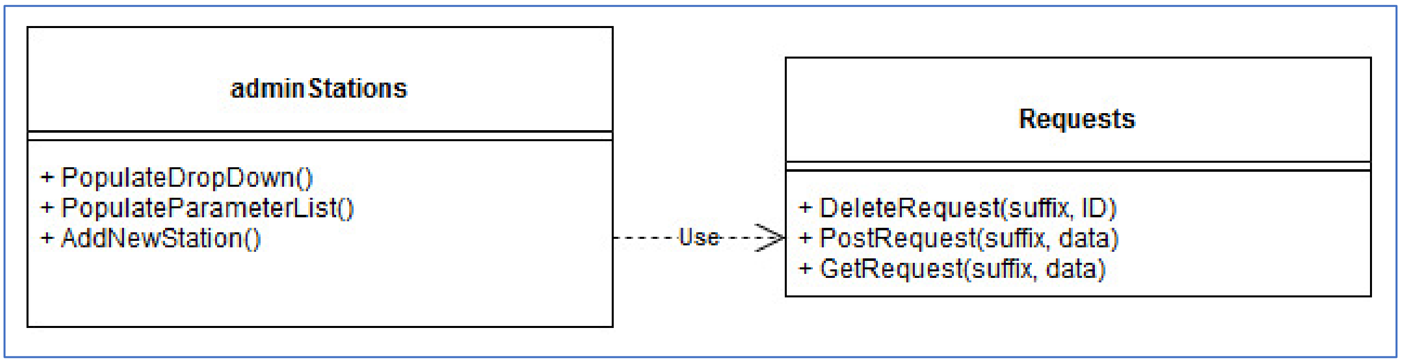

Customer
- username: String
- password: String
- firstName: String
- surname: String
+ Login(username: String, password: String) : void
+ getUsername() : String
+ setUsername(username: String) : void
+ getFirstName() : String
+ setFirstName(firstName: String) : void
+ getSurname() : void
+ setSurname(surname: String) : void
+ getPassword() : String
+ setPassword(password: String) : void
Uses
HttpRequest
- url: HttpUrl
- strategy: RequestStrategy
- data: JSONObject
+ HttpRequest(url: HttpUrl, strategy: RequestStrategy, data: JSONObject)
+ Send() : Response
1
1
«interface»
RequestStrategy
+ Request(url: HttpUrl, data: JSONObject) : Response
PostRequestStrategy
+ Request(url: HttpUrl, data: JSONObject) : Response
GetRequestStrategy
+ Request(url: HttpUrl, data: JSONObject) : Response

2.3.2. Retrieve Timetable Information

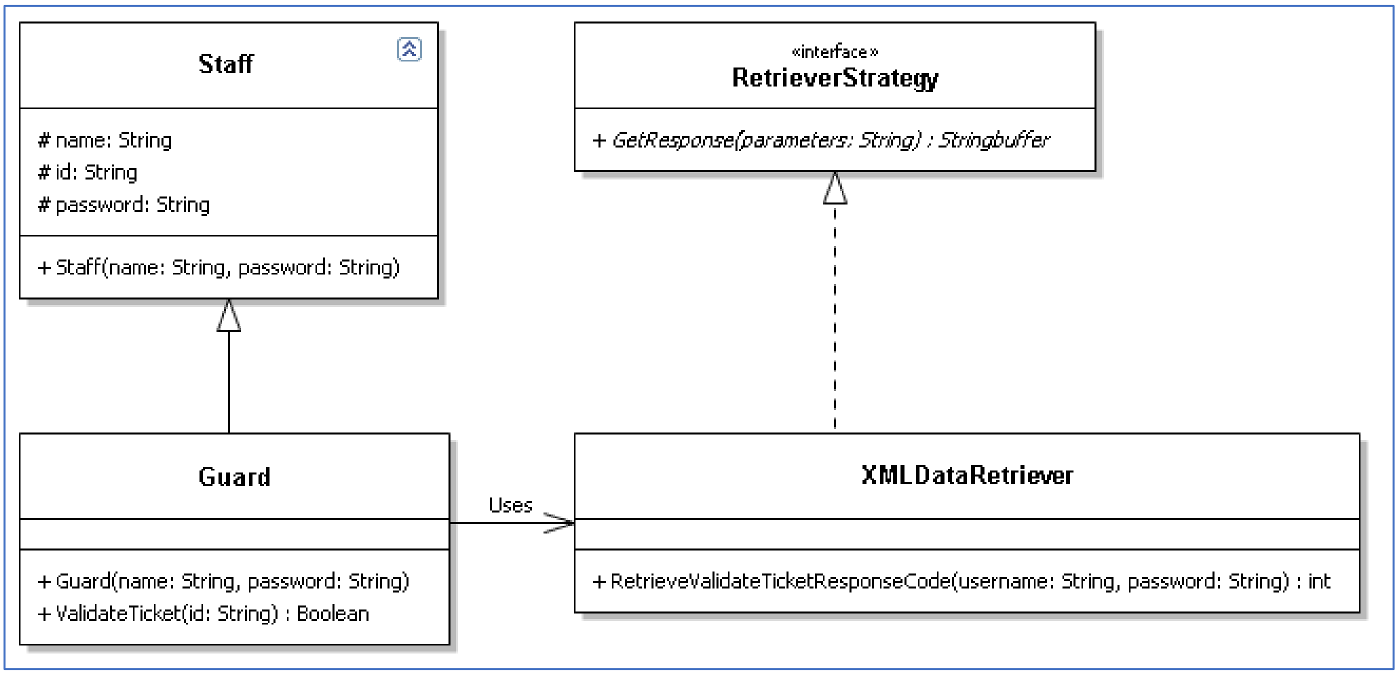

<interface>
DataObservable
+ RegisterObserver(DataObserver o)
+ RemoveObserver(DataObserver o)
+ NotifyObservers(DataObserver o)
1
<interface>
DataObserver
+ Update()
TimetableHolder
- List<TimetableRecord> timetable
- Type timetableListType
- String urlString
+ RetrieveTimetable(String Parameters)
+ getTimetable(): List<TimetableRecord>
1
TimetableRecord
- int Journey_No
- String Departure_Station
- String Arrival_Station
- String Departure_Time
- String Arrival_Time
- String Journey_Comments
+ getJourney_No(): int
+ getDeparture_Station(): String
+ getArrival_Station(): String
+ getDeparture_Time(): String
+ getArrival_Time(): String
+ getJourney_Comments(): String
Use
HttpRequest
- HttpUrl url
- RequestStrategy strategy
- JSONObject data
+ Send(): Response
+ HttpRequest(RequestStrategy, Httpurl, JSONObject)
1
<interface>
RequestStrategy
+ Request(HttpUrl, JSONObject): Response
PostRequestStrategy
+ Request(HttpUrl url, JSONObject data): Response
GetRequestStrategy
+ Request(HttpUrl url, JSONObject data): Response

The view classes implement DataObserver, which is used to monitor changes in the TimetableHolder class for the purpose of updating list elements in the UI when new TimetableRecord objects are retrieved.

2.3.3. Add Station Record


adminStations
+ PopulateDropDown()
+ PopulateParameterList()
+ AddNewStation()
Use
Requests
+ DeleteRequest(suffix, ID)
+ PostRequest(suffix, data)
+ GetRequest(suffix, data)

Validate Ticket


Staff
# name: String
# id: String
# password: String
+ Staff(name: String, password: String)
«interface»
RetrieverStrategy
+ GetResponse(parameters: String) : Stringbuffer
Guard
+ Guard(name: String, password: String)
+ ValidateTicket(id: String) : Boolean
Uses
XMLDataRetriever
+ RetrieveValidateTicketResponseCode(username: String, password: String) : int

2.4. Design Changes During Implementation

The design was changed over time as lacking functionality and missing requirements were identified and altered. Some notable additions were made to the project backlog in response to members of the group comparing the design against rival applications on the market.

QR code functionality was added in response to the Plymouth Citybus application on Android (2019), which includes a QR code for every ticket purchased. The QR can be scanned to validate the ticket and allow transport. Requirements to hold tickets in a shopping cart system and checkout without registering an account were added later in development. These functions were both added to bring the application design in line with more conventional online stores.

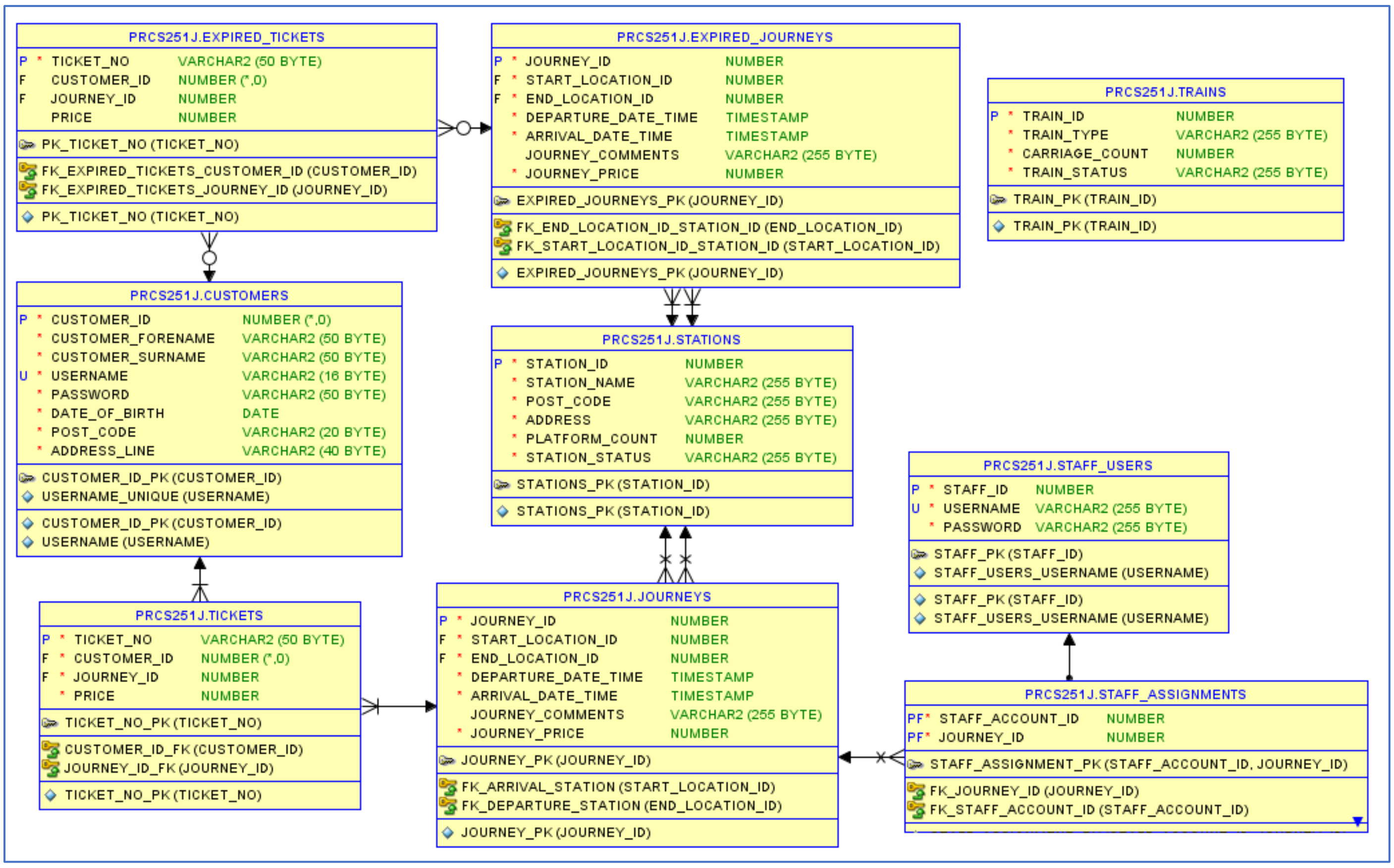
Increasing awareness of security during the implementation of elements on the database lead to changes to the overall project design. When initial testing of the API showed that passwords stored in plain-text were easily accessible through a GET request, the decision to hash passwords before storage was made. A hashing function was added to the product backlog and treated as a high priority task. As well as this, the production of a cookie-based security system for the web application was added to the project backlog, which would prevent unauthorized access to the administrator functionality.

Asynchronous functionality was planned later in development. The mobile and desktop applications needed to be restarted to refresh the stored records, which was ineffective in terms of design and usability. As a result, we planned for a timer system to refresh and retrieve data at set intervals.

In terms of database design, the archive tables EXPIRED_JOURNEYS and EXPIRED_TICKETS were added in response to concerns that querying the JOURNEYS and TICKETS tables would have considerable performance issues if large numbers of records were stored.

3. Database Design

3.1. Database Entity Relationship Diagram (Generated by SQL Developer)


PRCS251J.EXPIRED_TICKETS
P * TICKET_NO VARCHAR2 (50 BYTE)
F CUSTOMER_ID NUMBER (*,0)
F JOURNEY_ID NUMBER
PRICE NUMBER
PK_TICKET_NO (TICKET_NO)
FK_EXPIRED_TICKETS_CUSTOMER_ID (CUSTOMER_ID)
FK_EXPIRED_TICKETS_JOURNEY_ID (JOURNEY_ID)
PK_TICKET_NO (TICKET_NO)
PRCS251J.EXPIRED_JOURNEYS
P * JOURNEY_ID NUMBER
F * START_LOCATION_ID NUMBER
F * END_LOCATION_ID NUMBER
* DEPARTURE_DATE_TIME TIMESTAMP
* ARRIVAL_DATE_TIME TIMESTAMP
JOURNEY_COMMENTS VARCHAR2 (255 BYTE)
* JOURNEY_PRICE NUMBER
EXPIRED_JOURNEYS_PK (JOURNEY_ID)
FK_END_LOCATION_ID_STATION_ID (END_LOCATION_ID)
FK_START_LOCATION_ID_STATION_ID (START_LOCATION_ID)
EXPIRED_JOURNEYS_PK (JOURNEY_ID)
PRCS251J.TRAINS
P * TRAIN_ID NUMBER
* TRAIN_TYPE VARCHAR2 (255 BYTE)
* CARRIAGE_COUNT NUMBER
* TRAIN_STATUS VARCHAR2 (255 BYTE)
TRAIN_PK (TRAIN_ID)
TRAIN_PK (TRAIN_ID)
PRCS251J.CUSTOMERS
P * CUSTOMER_ID NUMBER (*,0)
* CUSTOMER_FORENAME VARCHAR2 (50 BYTE)
* CUSTOMER_SURNAME VARCHAR2 (50 BYTE)
U * USERNAME VARCHAR2 (16 BYTE)
* PASSWORD VARCHAR2 (50 BYTE)
* DATE_OF_BIRTH DATE
* POST_CODE VARCHAR2 (20 BYTE)
* ADDRESS_LINE VARCHAR2 (40 BYTE)
CUSTOMER_ID_PK (CUSTOMER_ID)
USERNAME_UNIQUE (USERNAME)
CUSTOMER_ID_PK (CUSTOMER_ID)
USERNAME (USERNAME)
PRCS251J.STATIONS
P * STATION_ID NUMBER
* STATION_NAME VARCHAR2 (255 BYTE)
* POST_CODE VARCHAR2 (255 BYTE)
* ADDRESS VARCHAR2 (255 BYTE)
* PLATFORM_COUNT NUMBER
* STATION_STATUS VARCHAR2 (255 BYTE)
STATIONS_PK (STATION_ID)
STATIONS_PK (STATION_ID)
PRCS251J.STAFF_USERS
P * STAFF_ID NUMBER
U * USERNAME VARCHAR2 (255 BYTE)
* PASSWORD VARCHAR2 (255 BYTE)
STAFF_PK (STAFF_ID)
STAFF_USERS_USERNAME (USERNAME)
STAFF_PK (STAFF_ID)
STAFF_USERS_USERNAME (USERNAME)
PRCS251J.TICKETS
P * TICKET_NO VARCHAR2 (50 BYTE)
F * CUSTOMER_ID NUMBER (*,0)
F * JOURNEY_ID NUMBER
* PRICE NUMBER
TICKET_NO_PK (TICKET_NO)
CUSTOMER_ID_FK (CUSTOMER_ID)
JOURNEY_ID_FK (JOURNEY_ID)
TICKET_NO_PK (TICKET_NO)
PRCS251J.JOURNEYS
P * JOURNEY_ID NUMBER
F * START_LOCATION_ID NUMBER
F * END_LOCATION_ID NUMBER
* DEPARTURE_DATE_TIME TIMESTAMP
* ARRIVAL_DATE_TIME TIMESTAMP
JOURNEY_COMMENTS VARCHAR2 (255 BYTE)
* JOURNEY_PRICE NUMBER
JOURNEY_PK (JOURNEY_ID)
FK_ARRIVAL_STATION (START_LOCATION_ID)
FK_DEPARTURE_STATION (END_LOCATION_ID)
JOURNEY_PK (JOURNEY_ID)
PRCS251J.STAFF_ASSIGNMENTS
PF* STAFF_ACCOUNT_ID NUMBER
PF* JOURNEY_ID NUMBER
STAFF_ASSIGNMENT_PK (STAFF_ACCOUNT_ID, JOURNEY_ID)
FK_JOURNEY_ID (JOURNEY_ID)
FK_STAFF_ACCOUNT_ID (STAFF_ACCOUNT_ID)

3.2. Database Object Descriptions

3.2.1. STAFF_USERS Table

The STAFF_USERS table contains the details necessary for a member of staff (Driver, cabin-crew, guard, administrator, controller) to log-in and access the system. STAFF_ID is a primary key for this table. New values for the STAFF_ID field are generated by the sequence STAFF_USERS_ID_SEQ. USERNAME is a unique, non-null field, with the constraint STAFF_USERS_USERNAME ensuring this. PASSWORD is non-unique, non-null and is hashed before storage using the STAFF_USER_SECURITY.GET_HASH function.

3.2.2. STAFF_USERS_ID_SEQ Sequence

STAFF_USERS_ID_SEQ is the sequence used to fetch the next value for the STAFF_ID primary key in the STAFF_USERS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.3. TRG_STAFF Trigger

The trigger TRG_STAFF is used to insert a value taken from STAFF_USERS_ID_SEQ upon insert or update of a row in the STAFF_USERS table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.4. STATIONS Table

The STATIONS table stores records for the stations used as start and end-points for train routes. STATION_ID is the primary key for the table. New values for STATION_ID are generated by the sequence STATION_ID_SEQ and are inserted by the trigger TRG_STATION_ID in the event of a null value. The other fields, STATION_NAME, POST_CODE, ADDRESS, PLATFORM_COUNT and STATION_STATUS are all properties of the station and are all non-null fields.

3.2.5. STATION_ID_SEQ Sequence

STATION_ID_SEQ is the sequence used to fetch the next value for the STATION_ID primary key in the STATIONS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.6. TRG_STATION_ID Trigger

The trigger TRG_STATION_ID is used to insert a value taken from STATION_ID_SEQ upon insert or update of a row in the STATIONS table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.7. CUSTOMERS Table

The CUSTOMERS table stores records of customer account details for customers to log-in on the mobile application, as well as for staff members to identify customers in search functions. CUSTOMER_ID is the primary key for this table. New values for CUSTOMER_ID are generated by the sequence CUSTOMER_ID_SEQ and inserted by the trigger TRG_CUSTOMER in the event of a null value. The fields CUSTOMER_FORENAME and CUSTOMER_SURNAME are non-null fields used for identification by staff. The fields USERNAME and PASSWORD are non-null fields used for log-in functionality. USERNAME is a unique field with the constraint “USERNAME_UNIQUE” preventing identical values.

3.2.8. CUSTOMER_ID_SEQ Sequence

CUSTOMER_ID_SEQ is the sequence used to fetch the next value for the CUSTOMER_ID primary key in the CUSTOMERS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.9. TRG_CUSTOMER Trigger

The trigger TRG_CUSTOMER_ID is used to insert a value taken from CUSTOMER_ID_SEQ upon insert or update of a row in the CUSTOMER table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.10. JOURNEYS Table

The JOURNEYS table holds information about current routes available for travel. JOURNEY_ID is the primary key for this table. New values for JOURNEY_ID are generated by the sequence JOURNEY_ID_SEQ and inserted by the trigger TRG_JOURNEYS in the event of a null value. START_LOCATION_ID and END_LOCATION_ID are foreign key columns, containing references to the STATION_ID column in the STATIONS table. These foreign keys are defined with the constraints FK_ARRIVAL_STATION and FK_DEPARTURE_STATION. DEPARTURE_DATE_TIME, ARRIVAL_DATE_TIME and JOURNEY_PRICE are all non-null columns that contain information about the journey. JOURNEY_COMMENTS is a nullable column that contains optional additional notes about a particular route.

3.2.11. EXPIRED_JOURNEYS Table

The EXPIRED_JOURNEYS table holds information about expired routes previously available for travel. JOURNEY_ID is the primary key for this table. START_LOCATION_ID and END_LOCATION_ID are foreign key columns, containing references to the STATION_ID column in the STATIONS table. These foreign keys are defined with the constraints FK_END_LOCATION_ID_STATION_ID and FK_START_LOCATION_ID_STATION_ID. DEPARTURE_DATE_TIME, ARRIVAL_DATE_TIME and JOURNEY_PRICE are all non-null columns that contain information about the journey. JOURNEY_COMMENTS is a nullable column that contains optional additional notes about a particular route.

3.2.12. JOURNEY_ID_SEQ Sequence

The JOURNEY_ID_SEQ is the sequence used to fetch the next value for the JOURNEY_ID primary key in the JOURNEYS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.13. TRG_JOURNEYS Trigger

The trigger TRG_JOURNEYS is used to insert a value taken from JOURNEY_ID_SEQ upon insert or update of a row in the JOURNEYS table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.14. TRAINS Table

The TRAINS table holds information about the trains available for the controller to assign to a specific journey. TRAIN_ID is the primary key for this table. New values for TRAIN_ID are generated by the sequence TRAIN_ID_SEQ and inserted by the trigger TRG_TRAIN_ID in the event of a null value. TRAIN_TYPE, CARRIAGE_COUNT and TRAIN_STATUS are all non-null fields that contain information about the train. TRAIN_STATUS is filled by the TRG_TRAIN_STATUS trigger if the inserted data is null, defaulting to ‘Operational’.

3.2.15. TRAIN_ID_SEQ Sequence

The TRAIN_ID_SEQ is the sequence used to fetch the next value for the TRAIN_ID primary key in the TRAINS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.16. TRG_TRAIN_ID Trigger

The trigger TRG_TRAIN_ID is used to insert a value taken from TRAIN_ID_SEQ upon insert or update of a row in the TRAINS table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.17. TRG_TRAIN_STATUS Trigger

The trigger TRG_TRAIN_STATUS is used to insert the VARCHAR value of ‘Operational’ into the TRAIN_STATUS field when inserting or updating a record with a null value for that field.

3.2.18. STAFF_ASSIGNMENTS Table

The STAFF_ASSIGNMENTS table is used as a link table to store staff assignments to journey records. The primary key for this table is a compound key of the STAFF_ACCOUNT_ID and the JOURNEY_ID fields. STAFF_ACCOUNT_ID is a foreign key that references STAFF_ID from the STAFF_USERS table, based on the constraint FK_STAFF_ACCOUNT_ID. JOURNEY_ID is a foreign key that references JOURNEY_ID in the JOURNEYS table, based on the constraint FK_JOURNEY_ID.

3.2.19. TICKETS Table

The TICKETS table stores customer ID numbers, journey ID numbers and the price at the time of purchase. It acts as a receipt for customer purchases as well as for purchase validation and authorization of travel. TICKET_NO is the primary key for this table. New values for TICKET_NO are generated by the sequence TICKET_NO_SEQ and inserted by the trigger TRG_TICKETS in the event of a null value. PRICE is a non-null field that contains the price of the ticket at the time of purchase, acting as a receipt. CUSTOMER_ID is a foreign key that references CUSTOMER_ID in the CUSTOMERS table, based on the constraint FK_CUSTOMERS. JOURNEY_ID is a foreign key that references JOURNEY_ID in the JOURNEYS table, based on the constraint FK_JOURNEYS.

3.2.20. EXPIRED_TICKETS Table

The EXPIRED_TICKETS table stores customer ID numbers, journey ID numbers and the price at the time of purchase. It acts as a receipt for customer purchases as well as for purchase validation and authorization of travel. TICKET_NO is the primary key for this table. PRICE is a non-null field that contains the price of the ticket at the time of purchase, acting as a receipt. CUSTOMER_ID is a foreign key that references CUSTOMER_ID in the CUSTOMERS table, based on the constraint FK_CUSTOMERS. JOURNEY_ID is a foreign key that references JOURNEY_ID in the JOURNEYS table, based on the constraint FK_JOURNEYS.

3.2.21. TICKET_NO_SEQ Sequence

The TICKET_NO_SEQ is the sequence used to fetch the next value for the TICKET_NO primary key in the TICKETS table. It has an increment of 1, a minimum value of 1, and a maximum value of 9999999999999999999999999999.

3.2.22. TRG_TICKETS Trigger

The trigger TRG_TICKETS is used to insert a value taken from TICKET_NO_SEQ upon insert or update of a row in the TICKETS table in the event that the inserted record has a value of null or 0 for the primary key.

3.2.23. TIMETABLE_RECORDS View

The view TIMETABLE_RECORDS is a view for customers and staff to access a full timetable with station names at the start and end-points of the journey in a simplified query. JOURNEY_NO, DEPARTURE_TIME, ARRIVAL_TIME and JOURNEY_COMMENTS refer to JOURNEY_NO, DEPARTURE_DATE_TIME, ARRIVAL_DATE_TIME and JOURNEY_COMMENTS in the JOURNEYS table, respectively. DEPARTURE_STATION and ARRIVAL_STATION refer to the STATION_NAME field in the STATIONS table, retrieved through an inner join using the STATION_ID field in STATIONS and the START_LOCATION_ID and END_LOCATION_ID fields in JOURNEYS.

3.2.24. HISTORY_RECORDS View

The view HISTORY_RECORDS is a view for customers and staff to access a full historical timetable with station names at the start and end-points of the journey in a simplified query. JOURNEY_NO, DEPARTURE_TIME, ARRIVAL_TIME and JOURNEY_COMMENTS refer to JOURNEY_NO, DEPARTURE_DATE_TIME, ARRIVAL_DATE_TIME and JOURNEY_COMMENTS in the EXPIRED_JOURNEYS table, respectively. DEPARTURE_STATION and ARRIVAL_STATION refer to the STATION_NAME field in the STATIONS table, retrieved through an inner join using the STATION_ID field in STATIONS and the START_LOCATION_ID and END_LOCATION_ID fields in EXPIRED_JOURNEYS.

3.2.25. VALIDATE_USER Procedure

VALIDATE_USER is a procedure that takes in a username, a password and an output value as parameters. This procedure will be called on an attempted log-in. The procedure queries the USERNAME field of either the CUSTOMERS or the STAFF_USERS table, depending on the prefix of the username, and sets the output value to ‘1’ if the PASSWORD field associated with the username matches a hashed version of the input username and password. This returned value of ‘1’ will be read on the API to feed-back to the user whether the log-in attempt was successful or not.

3.2.26. ADD_STAFF_USER Procedure

ADD_STAFF_USER is a procedure that inserts a new user into the STAFF_USERS table, to be used by the administrators to add new accounts for new members of staff. The procedure takes a username and a password as parameters. The procedure will insert the input username parameter into the USERNAME field, call the GET_HASH function with both the username and password as parameters to create a new hashed string to insert into the PASSWORD field, and then retrieve a new value from the STAFF_USERS_ID_SEQ to insert into the STAFF_ID field.

3.2.27. CHANGE_PASSWORD Procedure

CHANGE_PASSWORD is a procedure that replaces a password value in the STAFF_USERS table. The procedure takes in a username string, as well as an old and new password string. The procedure retrieves a record with a USERNAME field that matches the input username parameter, check the PASSWORD field is equal to the hash value returned from GET_HASH with the username and old password parameters as parameters, then insert the hash value from the username and the new password if the username and old password matched the value already stored.

3.2.28. GET_HASH Function

GET_HASH returns a hashed value based on the input username and password string parameters. The GET_HASH procedure uses the DBMS_OBFUSCATION_TOOLKIT.MD5 function and a salt value to generate the returned value.

3.2.29. ARCHIVE_EXPIRED_JOURNEYS Procedure

ARCHIVE_EXPIRED_JOURNEYS searches through the TICKETS and JOURNEYS tables to select records that have departure dates before the current date and inserts them into EXPIRED_TICKETS and EXPIRED_JOURNEYS, respectively.

3.3. SQL Create Statements

3.3.1. STAFF_USERS Table

```
CREATE TABLE STAFF_USERS
(
    STAFF_ID NUMBER,
    USERNAME VARCHAR(255) NOT NULL,
    PASSWORD VARCHAR(255) NOT NULL,

    CONSTRAINT STAFF_PK PRIMARY KEY (STAFF_ID),
    CONSTRAINT STAFF_USERS_USERNAME UNIQUE (USERNAME)
);
```

3.3.2. STAFF_USERS_ID_SEQ Sequence

```
CREATE SEQUENCE STAFF_USERS_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.3. TRG_STAFF Trigger

```
CREATE OR REPLACE TRIGGER PRCS251J.TRG_STAFF BEFORE INSERT OR
UPDATE OF STAFF_ID, USERNAME, PASSWORD ON STAFF_USERS FOR EACH ROW

BEGIN
    IF INSERTING THEN
        IF :NEW.STAFF_ID IS NULL OR :NEW.STAFF_ID = 0 THEN
            SELECT STAFF_USERS_ID_SEQ.NEXTVAL
            INTO :NEW.STAFF_ID
            FROM SYS.DUAL;
        END IF;
    END IF;
 END;
```

3.3.4. STATIONS Table

```
CREATE TABLE STATIONS
(
    STATION_ID NUMBER,
    STATION_NAME VARCHAR(255) NOT NULL,
    POST_CODE VARCHAR(255) NOT NULL,
    ADDRESS VARCHAR(255) NOT NULL,
    PLATFORM_COUNT NUMBER NOT NULL,
    STATION_STATUS VARCHAR(255) NOT NULL,

    CONSTRAINT STATIONS_PK PRIMARY KEY (STATION_ID)
);
```

3.3.5. STATION_ID_SEQ Sequence

```
CREATE SEQUENCE STATION_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.6. TRG_STATION_ID Trigger

```
CREATE OR REPLACE TRIGGER TRG_STATION_ID
    BEFORE INSERT OR UPDATE OF
    STATION_ID
    ON STATIONS FOR EACH ROW
    BEGIN
    IF INSERTING THEN
        SELECT STATION_ID_SEQ.NEXTVAL
        INTO :NEW.STATION_ID
        FROM SYS.DUAL;
    END IF;
 END;
```

3.3.7. CUSTOMERS Table

```
CREATE TABLE PRCS251J.CUSTOMERS
    (CUSTOMER_ID INT,
        CONSTRAINT CUSTOMER_ID_PK
        PRIMARY KEY (CUSTOMER_ID),
    CUSTOMER_FORENAME VARCHAR2(50)
        CONSTRAINT CUSTOMER_FORENAME_NN NOT NULL ENABLE,
            CHECK(CUSTOMER_FORENAME = INITCAP(CUSTOMER_FORENAME)),
     CUSTOMER_SURNAME VARCHAR2(50)
        CONSTRAINT CUSTOMER_SURNAME_NN NOT NULL ENABLE,
            CHECK(CUSTOMER_SURNAME = INITCAP(CUSTOMER_SURNAME)),
    USERNAME VARCHAR2(16)
        CONSTRAINT USERNAME UNIQUE
        CONSTRAINT USERNAME_NN NOT NULL ENABLE
            CHECK(USERNAME = LOWER(USERNAME))
);
```

3.3.8. CUSTOMER_ID_SEQ Sequence

```
CREATE SEQUENCE CUSTOMER_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.9. TRG_CUSTOMER Trigger

```
CREATE OR REPLACE TRIGGER PRCS251J.TRG_CUSTOMER BEFORE INSERT OR
    UPDATE OF CUSTOMER_FORENAME, CUSTOMER_SURNAME , USERNAME ON CUSTOMERS FOR
EACH ROW

BEGIN
    IF INSERTING THEN
        IF :NEW.CUSTOMER_ID IS NULL THEN
            SELECT CUSTOMER_ID_SEQ.NEXTVAL
            INTO :NEW.CUSTOMER_ID
            FROM SYS.DUAL;
        END IF;

        :NEW.CUSTOMER_FORENAME := INITCAP(:NEW.CUSTOMER_FORENAME);
        :NEW.CUSTOMER_SURNAME := INITCAP(:NEW.CUSTOMER_SURNAME);
        END IF;
 END;
```

3.3.10. JOURNEYS Table

```
CREATE TABLE JOURNEYS
(
    JOURNEY_ID NUMBER,
    START_LOCATION_ID NUMBER NOT NULL,
    END_LOCATION_ID NUMBER NOT NULL,
    DEPARTURE_DATE_TIME TIMESTAMP NOT NULL,
    ARRIVAL_DATE_TIME TIMESTAMP NOT NULL,
    JOURNEY_COMMENTS VARCHAR(255),

    CONSTRAINT JOURNEY_PK PRIMARY KEY (JOURNEY_ID),
    CONSTRAINT FK_DEPARTURE_STATION
    FOREIGN KEY (START_LOCATION_ID)
    REFERENCES STATIONS(STATION_ID) ON DELETE CASCADE,
    CONSTRAINT FK_ARRIVAL_STATION
    FOREIGN KEY (END_LOCATION_ID)
    REFERENCES STATIONS(STATION_ID) ON DELETE CASCADE
);
```

3.3.11. EXPIRED_JOURNEYS Table

```
CREATE TABLE EXPIRED_JOURNEYS
(
    JOURNEY_ID NUMBER,
    START_LOCATION_ID NUMBER NOT NULL,
    END_LOCATION_ID NUMBER NOT NULL,
    DEPARTURE_DATE_TIME TIMESTAMP NOT NULL,
    ARRIVAL_DATE_TIME TIMESTAMP NOT NULL,
    JOURNEY_COMMENTS VARCHAR(255),

    CONSTRAINT EXPIRED_JOURNEY_PK PRIMARY KEY (JOURNEY_ID),
    CONSTRAINT FK_START_LOCATION_ID_STATION_ID
    FOREIGN KEY (START_LOCATION_ID)
    REFERENCES STATIONS(STATION_ID) ON DELETE CASCADE,
    CONSTRAINT FK_END_LOCATION_ID_STATION_ID
    FOREIGN KEY (END_LOCATION_ID)
    REFERENCES STATIONS(STATION_ID) ON DELETE CASCADE
);
```

3.3.12. JOURNEY_ID_SEQ Sequence

```
CREATE SEQUENCE JOURNEY_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.13. TRG_JOURNEYS Trigger

```
CREATE OR REPLACE TRIGGER PRCS251J.TRG_CUSTOMER BEFORE INSERT OR
    UPDATE OF CUSTOMER_FORENAME, CUSTOMER_SURNAME , USERNAME ON CUSTOMERS FOR
EACH ROW

BEGIN
    IF INSERTING THEN
        IF :NEW.CUSTOMER_ID IS NULL THEN
            SELECT CUSTOMER_ID_SEQ.NEXTVAL
            INTO :NEW.CUSTOMER_ID
            FROM SYS.DUAL;
        END IF;

        :NEW.CUSTOMER_FORENAME := INITCAP(:NEW.CUSTOMER_FORENAME);
        :NEW.CUSTOMER_SURNAME := INITCAP(:NEW.CUSTOMER_SURNAME);
        END IF;
 END;
```

3.3.14. TRAINS Table

```
CREATE TABLE TRAINS
(     TRAIN_ID NUMBER,
      TRAIN_TYPE VARCHAR2(255) NOT NULL,
      CARRIAGE_COUNT NUMBER NOT NULL,
      TRAIN_STATUS VARCHAR2(255) NOT NULL,
      CONSTRAINT TRAIN_PK PRIMARY KEY (TRAIN_ID)
);
```

3.3.15. TRAIN_ID_SEQ Sequence

```
CREATE SEQUENCE TRAIN_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.16. TRG_TRAIN_ID Trigger

```
CREATE OR REPLACE TRIGGER TRG_TRAIN_ID
    BEFORE INSERT OR UPDATE OF
    TRAIN_ID
    ON TRAINS FOR EACH ROW
    BEGIN
    IF INSERTING THEN
        IF :NEW.TRAIN_ID IS NULL THEN
            SELECT TRAIN_ID_SEQ.NEXTVAL
            INTO :NEW.TRAIN_ID
            FROM SYS.DUAL;
        END IF;
    END IF;
 END;
```

3.3.17. TRG_TRAIN_STATUS Trigger

```
CREATE OR REPLACE TRIGGER TRG_TRAIN_STATUS
    BEFORE INSERT OR UPDATE OF
    TRAIN_STATUS
    ON TRAINS FOR EACH ROW
    BEGIN
    IF INSERTING THEN
        IF :NEW.TRAIN_STATUS IS NULL OR :NEW.TRAIN_STATUS = '' THEN
            SELECT 'OPERATIONAL'
            INTO :NEW.TRAIN_STATUS
            FROM SYS.DUAL;
        END IF;
    END IF;
 END;
```

3.3.18. STAFF_ASSIGNMENTS Table

```
CREATE TABLE STAFF_ASSIGNMENTS
(
     STAFF_ACCOUNT_ID NUMBER,
     JOURNEY_ID NUMBER,

     CONSTRAINT STAFF_ASSIGNMENT_PK PRIMARY KEY
     (STAFF_ACCOUNT_ID, JOURNEY_ID)

     CONSTRAINT FK_STAFF_ACCOUNT_ID FOREIGN KEY (STAFF_ACCOUNT_ID)
     REFERENCES STAFF_USERS (STAFF_ID) ON DELETE CASCADE,
     CONSTRAINT FK_JOURNEY_ID FOREIGN KEY (JOURNEY_ID)
     REFERENCES JOURNEYS (JOURNEY_ID) ON DELETE CASCADE
);
```

3.3.19. TICKETS Table

```
CREATE TABLE PRCS251J.TICKETS
    (TICKET_NO VARCHAR2(50),
        CONSTRAINT TICKET_NO_PK
        PRIMARY KEY (TICKET_NO),
    CUSTOMER_ID NUMBER
    CONSTRAINT FK_CUSTOMERS
      REFERENCES CUSTOMERS(CUSTOMER_ID)
        ON DELETE CASCADE
        NOT NULL,
    JOURNEY_ID NUMBER
    CONSTRAINT FK_JOURNEYS
        REFERENCES JOURNEYS(JOURNEY_ID)
        ON DELETE CASCADE
        NOT NULL,
    PRICE NUMBER
        CONSTRAINT PRICE_NN NOT NULL
);
```

3.3.20. EXPIRED_TICKETS Table

```
CREATE TABLE EXPIRED_TICKETS
(
      TICKET_NO VARCHAR2(50),
      CONSTRAINT PK_TICKET_NO
      PRIMARY KEY (TICKET_NO),
      CUSTOMER_ID NUMBER
      JOURNEY_ID NUMBER
      PRICE NUMBER

      CONSTRAINT FK_EXPIRED_TICKETS_JOURNEY_ID (JOURNEY_ID)
      REFERENCES EXPIRED_JOURNEYS (JOURNEY_ID) ON DELETE CASCADE

      CONSTRAINT FK_EXPIRED_TICKETS_CUSTOMER_ID (CUSTOMER_ID)
      REFERENCES CUSTOMERS (CUSTOMER_ID) ON DELETE CASCADE
);
```

3.3.21. TICKET_NO_SEQ Sequence

```
CREATE SEQUENCE TICKET_NO_SEQ
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

3.3.22. TRG_TICKETS Trigger

```
CREATE OR REPLACE TRIGGER PRCS251J.TRG_TICKETS
    BEFORE INSERT OR UPDATE OF
    TICKET_NO
    ON TICKETS FOR EACH ROW
    BEGIN
    IF INSERTING THEN
        IF :NEW.TICKET_NO IS NULL THEN
            SELECT TICKET_NO_SEQ.NEXTVAL
            INTO :NEW.TICKET_NO
            FROM SYS.DUAL;
        END IF;
    END IF;
 END;
```

3.3.23. TIMETABLE_RECORDS View

```
CREATE OR REPLACE VIEW TIMETABLE_RECORDS
(
     "JOURNEY_NO",
     "DEPARTURE_STATION",
     "ARRIVAL_STATION",
     "DEPARTURE_TIME",
     "ARRIVAL_TIME",
     "JOURNEY_COMMENTS"
)
AS SELECT
     JOURNEYS.JOURNEY_ID AS "JOURNEY_NO",
     S1.STATION_NAME AS "DEPARTURE_STATION",
     S2.STATION_NAME AS "ARRIVAL_STATION",
     JOURNEYS.DEPARTURE_DATE_TIME AS "DEPARTURE_TIME",
     JOURNEYS.ARRIVAL_DATE_TIME AS "ARRIVAL_TIME",
     JOURNEYS.JOURNEY_COMMENTS AS "JOURNEY_COMMENTS"
FROM JOURNEYS
INNER JOIN STATIONS S1 ON JOURNEYS.START_LOCATION_ID = S1.STATION_ID
INNER JOIN STATIONS S2 ON JOURNEYS.END_LOCATION_ID = S2.STATION_ID;
```

3.3.24. HISTORY_RECORDS View

```
   CREATE OR REPLACE VIEW HISTORY_RECORDS
   (
     "JOURNEY_NO",
     "DEPARTURE_STATION",
     "ARRIVAL_STATION",
     "DEPARTURE_TIME",
     "ARRIVAL_TIME",
     "JOURNEY_COMMENTS",
     "JOURNEY_PRICE"
   )
   AS SELECT
     EXPIRED_JOURNEYS.journey_id AS "JOURNEY_NO",
     s1.STATION_NAME AS "DEPARTURE_STATION",
     s2.STATION_NAME AS "ARRIVAL_STATION",
     EXPIRED_JOURNEYS.DEPARTURE_DATE_TIME AS "DEPARTURE_TIME",
     EXPIRED_JOURNEYS.ARRIVAL_DATE_TIME AS "ARRIVAL_TIME",
     EXPIRED_JOURNEYS.JOURNEY_COMMENTS AS "JOURNEY_COMMENTS",
     EXPIRED_JOURNEYS.JOURNEY_PRICE AS "JOURNEY_PRICE"
   FROM EXPIRED_JOURNEYS
   INNER JOIN STATIONS s1 ON EXPIRED_JOURNEYS.START_LOCATION_ID = s1.STATION_ID
   INNER JOIN STATIONS s2 ON EXPIRED_JOURNEYS.END_LOCATION_ID = s2.STATION_ID;
```

3.3.25. VALIDATE_USER Procedure

```
CREATE OR REPLACE PROCEDURE VALIDATE_USER
(
      P_USERNAME  IN  VARCHAR2,
      P_PASSWORD  IN  VARCHAR2,
      R_VALUE OUT VARCHAR2
)
IS
BEGIN

/* COMPARE THE FIRST TWO CHARACTERS OF THE USERNAME INPUT AGAINST THE STRING
'CU', THE PREFIX FOR A CUSTOMER ACCOUNT */
      IF UPPER(SUBSTR(P_USERNAME, 1, 2)) = 'CU' THEN
            SELECT '1' INTO R_VALUE
            FROM CUSTOMERS
            WHERE USERNAME = UPPER(P_USERNAME)
            AND PASSWORD = STAFF_USER_SECURITY.GET_HASH(P_USERNAME, P_PASSWORD);

/* IF ACCOUNT IS NOT A CUSTOMER ACCOUNT, SEARCH THE STAFF_USERS TABLE */
      ELSE
            SELECT '1' INTO R_VALUE
            FROM STAFF_USERS
            WHERE USERNAME = UPPER(P_USERNAME)
            AND PASSWORD = STAFF_USER_SECURITY.GET_HASH(P_USERNAME, P_PASSWORD);
      END IF;

EXCEPTION

/* IF NO DATA IS FOUND, RETURN R_VALUE = 0 */
WHEN NO_DATA_FOUND THEN
      R_VALUE := '0';

END;
```

3.3.26. GET_HASH Function (Part of the STAFF_USER_SECURITY PLSQL package)

```
FUNCTION GET_HASH
(
      P_USERNAME IN VARCHAR2,
      P_PASSWORD IN VARCHAR2
)
RETURN VARCHAR2 AS
      L_SALT VARCHAR2(30) := 'PUTYOURSALTHERE';
BEGIN
      RETURN DBMS_OBFUSCATION_TOOLKIT.MD5(
      INPUT_STRING => UPPER(P_USERNAME) || L_SALT || UPPER(P_PASSWORD));
END;
```

3.3.27. ADD_STAFF_USER Procedure (Part of the STAFF_USER_SECURITY PLSQL package body)

```
PROCEDURE ADD_STAFF_USER
(
      P_USERNAME  IN  VARCHAR2,
      P_PASSWORD  IN  VARCHAR2
)
AS
BEGIN
      INSERT INTO STAFF_USERS
      (
            STAFF_ID,
            USERNAME,
            PASSWORD
      )
      VALUES
      (
            STAFF_USERS_ID_SEQ.NEXTVAL,
            UPPER(P_USERNAME),
            GET_HASH(P_USERNAME, P_PASSWORD)
      );

COMMIT;
END;
```

3.3.28. CHANGE_PASSWORD Procedure (Part of the STAFF_USER_SECURITY PLSQL package body)

```
PROCEDURE CHANGE_PASSWORD
(
      P_USERNAME IN VARCHAR2,
      P_OLD_PASSWORD IN VARCHAR2,
      P_NEW_PASSWORD IN VARCHAR2
)
AS
      V_ROWID ROWID;
BEGIN
      SELECT ROWID
      INTO V_ROWID
      FROM STAFF_USERS
      WHERE USERNAME = UPPER(P_USERNAME)
      AND PASSWORD = GET_HASH(P_USERNAME, P_OLD_PASSWORD)
      FOR UPDATE;

      UPDATE STAFF_USERS
      SET PASSWORD = GET_HASH(P_USERNAME, P_NEW_PASSWORD)
      WHERE ROWID = V_ROWID;

COMMIT;
EXCEPTION
WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20000, 'Invalid username/password.');
END;
```

3.3.30. ARCHIVE_EXPIRED_JOURNEYS Procedure

```
CREATE OR REPLACE PROCEDURE ARCHIVE_EXPIRED_JOURNEYS
AS
BEGIN
    INSERT INTO PRCS251J.EXPIRED_JOURNEYS SELECT *
    FROM PRCS251J.JOURNEYS WHERE(JOURNEYS.ARRIVAL_DATE_TIME < SYSDATE);

    INSERT INTO
    EXPIRED_TICKETS(customer_id , journey_id , price , ticket_no )
    SELECT CUSTOMER_ID,JOURNEY_ID,PRICE,TICKET_NO
    FROM TICKETS
    WHERE TICKETS.JOURNEY_ID IN(
        SELECT JOURNEYS.JOURNEY_ID FROM PRCS251J.JOURNEYS
        WHERE(JOURNEYS.ARRIVAL_DATE_TIME < SYSDATE)
        );

    DELETE FROM PRCS251J.TICKETS
    WHERE TICKETS.JOURNEY_ID IN(
        SELECT JOURNEYS.JOURNEY_ID FROM "PRCS251J".JOURNEYS
        WHERE(JOURNEYS.ARRIVAL_DATE_TIME < SYSDATE)
        );

    DELETE FROM PRCS251J.JOURNEYS WHERE(JOURNEYS.ARRIVAL_DATE_TIME < SYSDATE);
END;
```

3.3.31. STAFF_USER_SECURITY Package

```
CREATE OR REPLACE PACKAGE STAFF_USER_SECURITY
AS

FUNCTION GET_HASH
(
     P_USERNAME IN VARCHAR2,
     P_PASSWORD IN VARCHAR2
)
RETURN VARCHAR2;

PROCEDURE ADD_STAFF_USER
(
     P_USERNAME IN VARCHAR2,
     P_PASSWORD IN VARCHAR2
);

PROCEDURE CHANGE_PASSWORD
(
     P_USERNAME IN VARCHAR2,
     P_OLD_PASSWORD IN VARCHAR2,
     P_NEW_PASSWORD IN VARCHAR2
);
END;
```

3.3. Normalised Tables

STAFF_USERS
+ STAFF_ID
+ POSITION
+ FIRST_NAME
+ LAST_NAME
+ DATE_OF_BIRTH
+ POST_CODE
+ ADDRESS
+ USERNAME
+ PASSWORD

CUSTOMERS
+ CUSTOMER_ID
+ CUSTOMER_FORENAME
+ CUSTOMER_SURNAME
+ DATE_OF_BIRTH
+ POST_CODE
+ ADDRESS
+ USERNAME
+ PASSWORD

JOURNEYS
+ JOURNEY_ID
+ START_LOCATION_ID
+ END_LOCATION_ID
+ DEPARTURE_DATE_TIME
+ ARRIVAL_DATE_TIME
+ JOURNEY_COMMENTS
+ JOURNEY_PRICE

EXPIRED_JOURNEYS
+ JOURNEY_ID
+ START_LOCATION_ID
+ END_LOCATION_ID
+ DEPARTURE_DATE_TIME
+ ARRIVAL_DATE_TIME
+ JOURNEY_COMMENTS
+ JOURNEY_PRICE

STATIONS
+ STATION_ID
+ STATION_NAME
+ POST_CODE
+ ADDRESS
+ PLATFORM_COUNT
+ STATION_STATUS

TRAINS
+ TRAIN_ID
+ TRAIN_TYPE
+ CARRIAGE_COUNT
+ TRAIN_STATUS

TICKETS
+ TICKET_NO
+ CUSTOMER_ID
+ JOURNEY_ID
+ PRICE

STAFF_ASSIGNMENTS
+ STAFF_ACCOUNT_ID
+ JOURNEY_ID

3.4. Database Overview

3.4.1. Design Assumptions

- It was assumed that administrator and controller staff members were infallible and did not make mistakes during data entry.
- It was assumed that every station had a guard that would validate every ticket for travel. No ticket validation would be carried out during travel.

3.4.2. Design Shortcomings

Lacking constraints in some database table result in the ability to enter incorrect or invalid data. The EXPIRED_JOURNEYS and JOURNEYS tables both lacked validation on the DEPARTURE_DATE_TIME and ARRIVAL_DATE_TIME columns, with EXPIRED_JOURNEYS allowing data in these columns to be entered that was set after the current date/time. JOURNEYS likewise had this problem, allowing new journey records to be entered into the database that have a DEPARTURE_DATE_TIME set before the current date/time. For both tables it is also possible to set the ARRIVAL_DATE_TIME to a date/time before the ARRIVAL_DATE_TIME value. The STATIONS table likewise lacks crucial constraints. The PLATFORM_COUNT column can be set to a value of 0, allowing new station records to lack any platforms for a train to arrive at.

Technical issues with Entity Framework in the Restful API lead to conflicts with stored functions and procedures on the database. Because of the API issues less functions were used than initially planned, with much of the functionality instead relying on SQL query strings in the API itself.

While initially planned, the production of an automated system for moving JOURNEY records to the EXPIRED_JOURNEYS table upon expiry was not produced. This is due to lacking permissions on the database system. Currently, the JOURNEY records need to be moved manually.

3.4.3. Design Advantages

PLSQL triggers have been used to automatically retrieve a value from sequences and assign a new ID to a record before insertion into the database, in the event of a primary key value of ‘0’ or null. This adds a level of resilience to the database, as insertion of a null primary key would otherwise return an error and fail.

A PLSQL trigger is used to hash the PASSWORD field of a new record in the CUSTOMERS table. This triggers upon insertion or update of the values. This allows developers to make use of the pre-generated API POST function without having to call a stored procedure for customer generation, while still maintaining database security and integrity.

The HISTORY_RECORDS and TIMETABLE_RECORDS views are used to acquire values from different columns in distinct tables in a single readable view. This simplifies the process from the API, which can make a simple GET request and retrieve numerous values without a custom SQL query.

The use of PLSQL sequences automatically generates a unique primary key value for most tables on the database. This stops users from accidentally violating UNIQUE constraints on primary key columns.

4. Usability

4.1. Progressive list of prioritized essential and desirable changes required

Desktop Application	
More feedback upon actions	Critical
The interface needs bigger elements that can be easily clicked with a touchscreen	Serious
Wording needs to be improved to match the User Mental Model	Serious
Aesthetic improvements	Minor

Web Application	
More feedback upon actions	Critical
Changed colour scheme to a darker tone to reduce eye strain	Serious

Mobile Application	
More feedback upon actions	Critical
Wording needs to be improved to match the user mental model	Serious
Larger font	Serious
Dashboard	Minor
Matching icons in navigation bar	Minor

4.2. Usability Report

During the project, the design has been treated as an iterative process. At every sprint of the project, all platforms underwent many improvements and tweaks, due to the Cognitive Walkthrough and Usability tests. Regular user feedback during the development paved the way for better and more intuitive applications.

Every testing session lasted from 10 to 15 minutes. During the tests, the users were asked to think aloud whilst carrying out the tasks to explain what they did and why. The tester asked the end-users question when tasks were ambiguous or when they appeared to struggle. Most of the tests took place in a public space as it is a real-life scenario and it may even be possible to find the test users on the spot. The testing team mainly handled two roles: the facilitator and the log keeper. However, in some occasions it was required to handle both roles by only one at the same time.

The facilitator had the main contact with the user, gave him the test tasks and made the questions. The log keeper wrote down what happened, where the user encountered problems and what the user believed about the system. Occasionally, the log keeper cautiously interfered with the user, for instance to clarify a problem or to hold the test user back while writing the notes. By following this methodology, the testers felt at ease working towards the requirements of the task scenarios.

The first prototype created was for Desktop Application and was paper based: it demonstrated to be a quick, simple, and cost-effective way to mock-up an idea of the application at its early stage; the mobile application had a good reception due to the fact that the visual prototype was created with Adobe XD, a specialised tool to create appealing and responsive visual prototypes. In fact, the mobile application had the most positive feedback at the earliest stage of the project; the prototype for the web application was the most realistic as it was developed by using HTML and CSS, the languages used in the end-product. This allowed the gathering of more reliable feedback and realistic reactions across all the testers.

Thanks to the result collected from the Cognitive Walkthroughs, it was acknowledged that changed needed to be made in order to follow the common User Mental Model (*Walker, 2018*). Therefore, subsequent changes were implemented on the interface across the applications, mainly repositioning and resizing of elements at this phase.

The following sprints required to move from prototyping, to implementation and the Cognitive Walkthrough were discarded at this stage. Usability tests on real implementations followed, carried out periodically and in a timely manner as the applications needed more frequent feedback since new features were introduced at each sprint.

During this phase of the development, very few criticisms were made to the applications. In fact, no problems were encountered in the layout, the font or the size of the elements. On the other hand, a lack of feedback was noticed across all applications, especially in the web application, which resulted in the users not knowing what their actions have led them to.

Negative feedback was reported in the Web Application as no actions confirmed success through dialogs, hence the users could not know if the actions were actually being carried out; the mobile and desktop application provided very limited feedback: only during the login for the mobile application and the log out for the Desktop platform.

To address this problem, sprint 3, 4 and 5 involved the implementation of dialog boxes in order to communicate to the users what their resulted in, which resulted in a boost of the information flow across all the platforms. Now the website implementation and now it is the most informative application, featuring not only dialogs to inform the users but also animations for the login and logout actions.

Colours turned out to be very important details for some users. In the web application, the brightness of the background was so high that the operators accused eye strains, which successively lead to a change in the colour scheme. Eventually it was decided to opt for a darker colour scheme which resulted in improved interactions with the platform during the following usability tests.

Wording was a common downside across the Mobile and Desktop apps as they did not reflect the common User mental model (e.g. departure date instead of leaving date or Search instead of Bookings). The users' feedback greatly enhanced the end-product by providing terms that are commonly expected by this sort of applications.

4.3. Strengths

The Gestalt Principles (Walker, 2018) of grouping were used extensively across the applications. The Proximity, Continuation, Closure and Similarity Principles are generally a good index of usability when it comes to usable applications because they boost information flows, making the utilisation easier and the scanning process nearly automatic (Figure 5-7, Appendix A).

For instance, the mobile application's dashboard features four squared buttons which benefits from the Similarity Principle and the Proximity principles as they are similar looking objects and very close to each other making them perceived as grouped (Figure 5, Appendix A). Furthermore, the Good Continuation synergises with Closure as the elements are laid out in continuous lines, and together, look like a bigger square.

On the other hand, the web application features a good example of the good continuation principle across its pages, specifically on the menu bar at the top which shows a group buttons on the same line (Figure 6, Appendix A). Not only it creates a distinct section but also improves geometric elegance.

The interfaces created within this project also made use of some of the Golden Rules of interface design. For instance, common icons were used to be consistent with the other applications in this business sector. Reason why, the mobile application's dashboard makes use of buttons showing skeuomorphic icons: for example, a calendar to represent the "Timetable" section of the app; a magnifying lens to point at the search feature; a gear to refer to the settings and so on. "By making a control or interface behave (and perhaps look) like a physical counterpart, users can draw on previous experience to understand how to use the UI" (Alan S., *In Defence of Skeuomorphism*).

On all parts the applications, dialogs are implemented to yield closure for every action (Figures 1-4, Appendix A). Informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief. Furthermore, the number of the elements presented (buttons, panels, information) within the screen is always seven plus or minus two to reduce the short memory load and the user doesn't need to remember more things than necessary (University of Maryland, 2016).

4.4. Weaknesses

Unfortunately, none of the applications allow the user any kind reversal of actions. For instance, the users can't undo delete actions in the web application, meaning that journeys, stations or trains deleted from the system are unrecoverable. Furthermore, the mobile application does not allow the user to cancel a booking and, therefore, no refunds can be issued for unused bookings.

No consistent style has been adopted across the applications. All applications are very different regarding the colour used, the dialogs, the fonts and the overall appearance. Consequently, the operators cannot use they knowledge from the other applications to use another one.

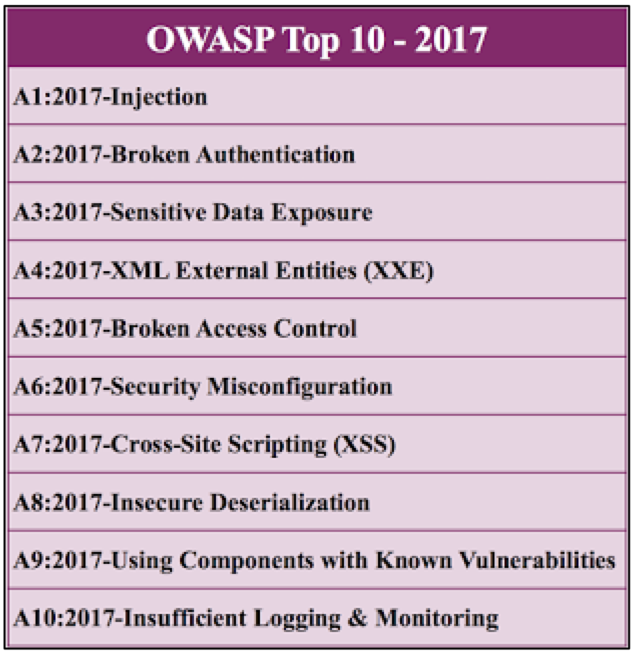
The software platforms do not feature any help section or way to get in touch. No aid is provided for the users if they are stuck in the applications which affects the user level of confidence, perhaps preventing from exploring unknown features within the applications.

Finally, the desktop application and web application does not provide any search feature to filter down the lists of journeys, meaning that on larger businesses, the operator might have to scroll down hundreds of journeys before getting to the desired entries. This not only will have a negative impact on the times of the staff, but also on the customers when they are asking information about the journeys to staff members.

Not all error messages pop on the web app due to unfamiliarity with the promise, resolve and reject system used in JavaScript for requests, however, success confirmation pop-ups always display.

5. Security

Across all applications, login functions are performed with POST requests: this way, the credential details are sent in the HTTP messages body rather than appending them in the URL with a GET request. In fact, GET requests would store the login link with password and username in the browser's history or they could also be accidentally captured by firewall logs. For the aforementioned reason, using POST requests made the system less vulnerable to attacks. When inserting data PUT requests where used to a similar effect to POST request.



OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

The OWASP Top 10 Security Risks (Cardoza and Cardoza, 2019)

Storing user passwords was a critical component whilst working on the database. The “hashing” function ensured a safer database in such a way that if the data is compromised, the user’s password will not get exposed. A “Hash” is a one-way function that generates a representation of the password. As the hash function is in the ORACLE database, whenever a user signs up, the password is stored as a generated hash, rather than the actual characters that the user typed in. However, an oversight in the hashing function causes all passwords to be capitalised, increasing potential hashing conflicts.

In order to make it more difficult to expose the hash (Arias, 2018), “Salting” was used. Essentially, it’s a unique value that can be added to the end of the password to create a different hash value. This adds an extra layer of security to the hashing process, because it adds more computational power in order to expose the hash. Hashing a password is not 100% secure as hashing alone is not that difficult to break.

XEE (XML External Entities) attacks were taken in account when deciding how to transmit data to the API. XEE is an attack against a web application that parses XML input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An ‘external entity’ in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker (Muscat, 2019). To prevent XEE attacks it was ensured all the applications use a less complex input, JSON, a type of simple, human-readable notation often used to transmit data over the internet.

The use of parameterised queries in the API functions not only improved the performance of the system, but also assures a good level of protection to the database from SQL injections. The SQL queries in the API functions make use of “bind variables” (or parameters) instead of inserting the values directly into the queries. Bind variables are considered as strings by the database. Which is why they prevent the back-end from running malicious queries that are harmful to the database: so, if the user entered 12345 or 1=1 as the input, the parameterized query would search in the table for a match with the entire string 12345 or 1=1.

Cookies where added to the website on the client side so pages are unable to be accessed without prior user validation preventing unauthorised users from accessing sensitive data and damage database security.

Penetration testing was performed using Pentest-Tools.com (Pentest-Tools.com, 2019) light version giving results in appendix B showing 1 medium and 2 low risk security flaws according to their metrics.

6. Software Engineering

6.1. Desktop Application (Driver/Cabin-Crew/Guard)

6.1.1. Assumptions Made

It was assumed that a specific job-role would be present for the validation of purchased tickets, the guard role. Guards were given a unique interface that provides the ability to validate tickets when a number is entered, containing no other UI elements. It was also assumed that guards would have no specific station assignment for the minimum viable product, with guards having no personalised timetable.

Another assumption was that the cabin-crew had no ability to validate tickets themselves. Assuming the guards were infallible and present at every station simplified the distribution of tasks for job roles.

It was assumed that the platforms of trains would be listed at the stations outside of the developed system.

6.1.2. Software Design Shortcomings

Difficulty with handling JSON data initially resulted in the production of the "XMLDataRetriever" and "XMLParser" classes. The XML classes were eventually phased out in favour of the "PostRequest" and "JsonDataRetriever" classes, which handle POST and GET requests for JSON format data and the GSON parser provided by Google *(2019)*. Despite the change from JSON to XML, some functionality in the desktop application is still reliant on XML due to the late implementation of the format changes, resulting in a level of redundancy that was not initially planned for.

The timetable functionality in the desktop application was originally implemented without consideration for design patterns and data, consisting of a "TimetableHolder" class that requested and filled a list with received timetable data, as well as a "TimetableEntry" class to hold the data itself. Later iterations modified the timetable functionality to include a singleton design pattern, with the "TimetableListSingleton" class retrieving timetable data and storing it in a singular instance of the "TimetableListSingleton".

Issues with the design of API-end functionality affected the desktop application negatively. The STAFF_ID value of the currently logged-on user is retrieved separately to the log-in query function call, despite the functionality for both being interlinked. A better implementation would have the STAFF_ID returned together with the HTTP response code, allowing the model to store it for later use.

6.1.3. Software Design Advantages

The design of the model classes aided extensibility, with additions to the model classes in the front-end applications and the database allowing for the addition of extra columns with minor alterations.

The "TimetableListSingleton" class adheres to the responsibility, creator and expert patterns of the GRASP design principles *(Rao, n.d.)*. "TimetableListSingleton" stores a list of "TimetableEntry" objects that are instantiated within the singleton class. The "TimetableListSingleton" object is singularly responsible for fetching, creating and managing the timetable data.

The personalised timetable provided to staff members was fulfilled by the PersonalTimetable class. This provides a separation of concerns, with the functionality being placed in a distinct class from the classes used for other timetable functionality.

Controller classes were used to manage interactions between the view classes and the model classes, adhering to correct Model-View-Controller (MVC) design principles, ensuring a separation of functionality.

6.2. Mobile Application

6.2.1. Assumptions Made

It was assumed the user would use a PayPal-style service for managing transactions, and would not directly process transactions through the application, or would not store additional payment information in the application.

It was assumed that users would know the name of the destination station, and could correctly enter it into the search field.

It was assumed that the platforms of trains would be listed at the stations outside of the developed system.

6.2.2. Software Design Shortcomings

Due to the UI recycler view element used to display the timetable records, a level of high coupling between specific classes was produced. The controller for the UI gets a list of "TimetableRecord" objects from the "TimetableHolder" instance and then obtains data directly from each "TimetableRecord". This is a high coupling design according to the GRASP design principles *(Rao, n.d.),* as the controller has an aggregation association with both elements.

A security flaw is present within the log-in functionality for the customer. The log-in process creates an object of the "Customer" class, which holds "username" and "password" properties that are retrieved from the entered data. These properties are used in the "Login" method but are stored within the object in plain-text. While the "Customer" object will likely be removed by the Java garbage collector upon going out of scope, it still presents a significant security risk.

6.2.3. Software Design Advantages

The mobile application makes use of an observer/observable pattern to ensure the timetable list can update asynchronously, with the controller class being set as an observer and the "TimetableHolder" class being set as an observable. Whenever the "TimetableHolder" class retrieves data from the API/database it triggers an update process in the recycler view UI element defined in the "Timetable" controller, adding the new records to the display.

A strategy pattern was applied for data retrieval, with the "RequestStrategy" interface being implemented by "GetRequestStrategy" and "PostRequestStrategy". This design aids expandability and modularity, allowing further additions to be implemented with minimal changes to existing code.

6.3. Web Application

6.3.1. Assumptions Made

It was assumed that a single staff role was responsible for maintaining the database records and assigning other staff to specific areas. Originally a "Controller" and an "Administrator" role was planned, though this was phased out in later iterations of the plan.

It was assumed that the platforms of trains would be listed at the stations outside of the developed system.

6.3.2. Software Design Shortcomings

The "Requests" file contains functions for making DELETE, POST and GET requests. In the final product it reads specific HTTP response codes, which in some cases don't exactly match up with the codes that are actually returned by the API. This can lead to issues where errors are displayed when the request was successful. This was planned for expansion and refinement later in the project, with a more object-oriented design process to allow the calling object to decide the accepted response codes.

The individual scripts for pages ("adminTrains", "adminRoutes" and "adminStations") are heavily coupled to their respective pages. All of the individual scripts access HTML components in the pages they're used, which can be problematic. In the event of a change to the HTML, the JavaScript would need alteration to match, reducing the ease of extension and modification.

6.3.3. Software Design Advantages

The production of individual JavaScript files for specific pages results in a level of separation of concerns, with tasks split by page. This aids code readability as the functions are segmented off into distinct groups.

Code re-use is significant in the web application, with all request functionality being provided by the Requests.js file, which is referenced whenever a HTTP request is needed. This reduces the amount of development time needed as repeated functionality can be used.

References

Cockburn, A. (2002). *Agile Software Development*. Boston, MA.: Pearson Education Inc., p.84.

Hasan, S. (2017). *12 Key Principles of Agile Project Management*. [online] TaskQue. Available at: https://blog.taskque.com/principles-agile-project-management/ [Accessed 15 Apr. 2019].

Plymouth Citybus. (2019). Newcastle upon Tyne: The Go-Ahead Group plc.

Arias, D. (2018). Adding Salt to Hashing: A Better Way to Store Passwords. [online] Auth0. Available at: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/ [Accessed 24 Apr. 2019]

Muscat, I. (2019). What Are XML External Entity (XXE) Attacks? [online] acunetix. Available at https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/ [Accessed 24 Apr. 2019]

Google (2019). *gson*. [online] GitHub/Google/gson. Available at: https://github.com/google/gson [Accessed 12 Mar. 2019].

Rao, D. (n.d.). *GRASP Design Principles*. [online] University of Colorado Boulder. Available at: https://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf [Accessed 1 May 2019].

Alan, S. (n.d). In defense of skeuomorphism [online]. Prototypr.io. Available at https://blog.prototypr.io/in-defense-of-skeuomorphism-2895308218ee/ [Accessed 5 May 2019]

University of Maryland (2016). The Eight Golden Rules of Interface Design [online]. University of Maryland. Available at https://www.cs.umd.edu/users/ben/goldenrules.html/ [Accessed 5 May 2019]

Walker D (2018) Lecture 4: Knowledge Representation. University of Plymouth. Pages 7-8 [Accessed 1 May 2019]

Walker D (2018) Lecture 2: Lecture 2: HCI – Perceiving Visual Information. University of Plymouth. Pages 6-22 [Accessed 1 May 2019]

Cardoza, C. and Cardoza, V. (2019). *OWASP releases the Top 10 2017 security risks - SD Times*. [online] SD Times. Available at: https://sdtimes.com/app-development/owasp-releases-top-10-2017-security-risks/ [Accessed 1 May 2019].

Pentest-Tools.com. (2019). *Online Penetration Testing and Ethical Hacking Tools*. [online] Available at: https://pentest-tools.com/home [Accessed 5 May 2019].

Appendix A

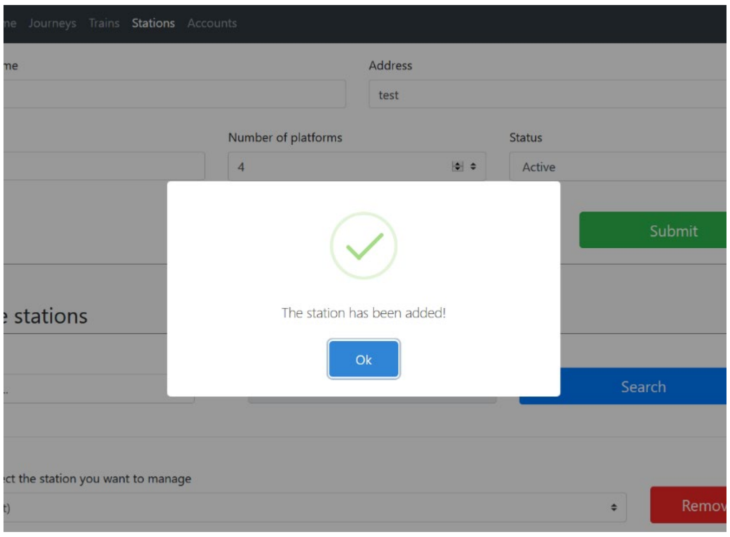


Figure 1 (Dialog box in the Web Application)

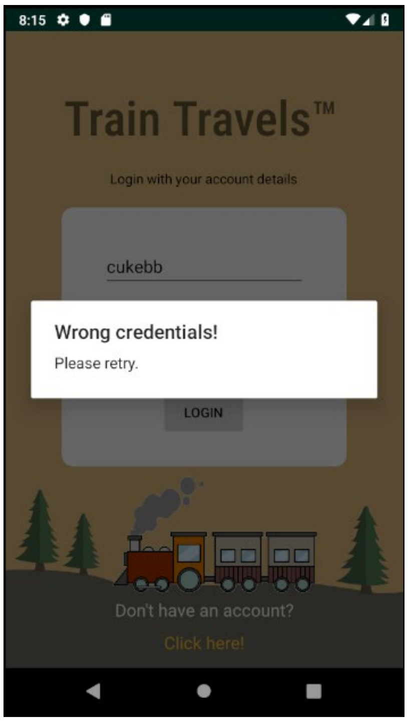


Figure 2 (Dialog box in the Android Application)

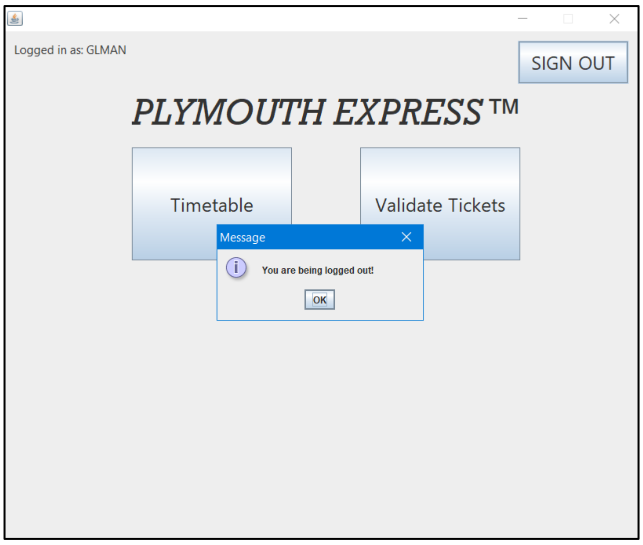


Figure 3 (Dialog box in the Desktop Application)

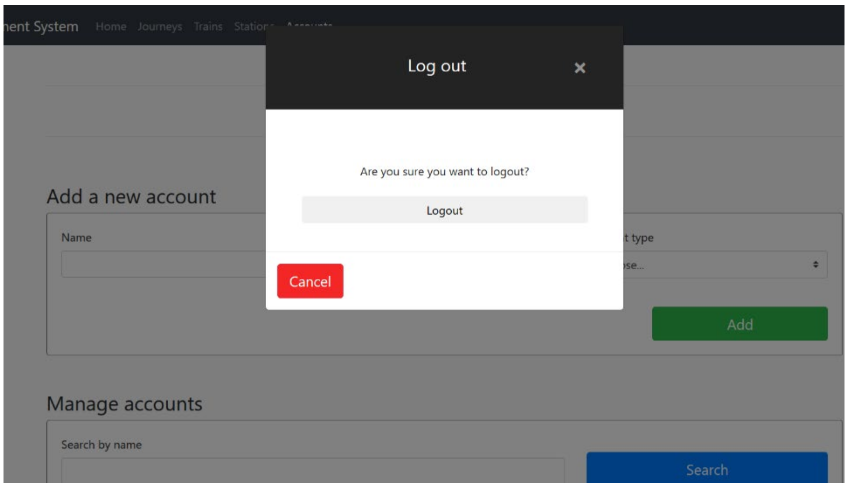


Figure 4 (Log out Dialog in the Web Application)

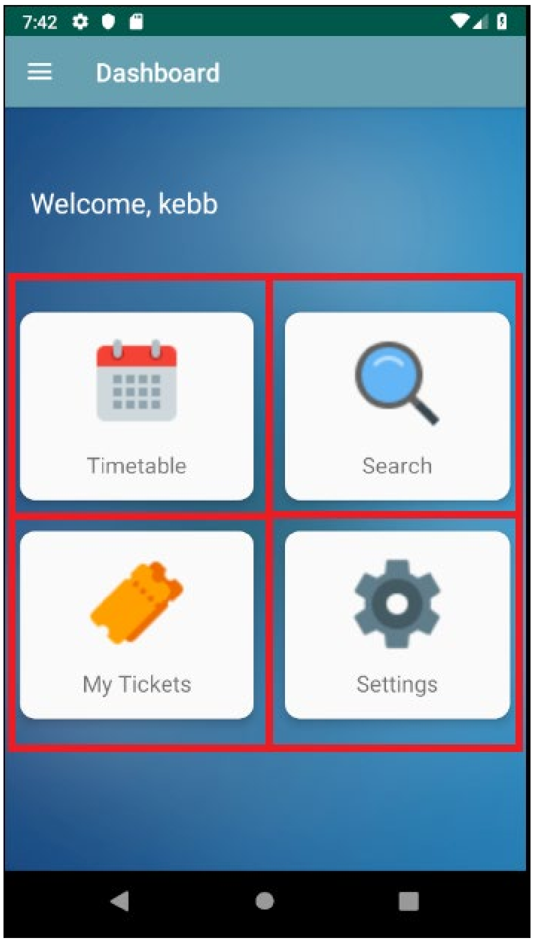


Figure 5 (Gestalt Principles in the Mobile Application)

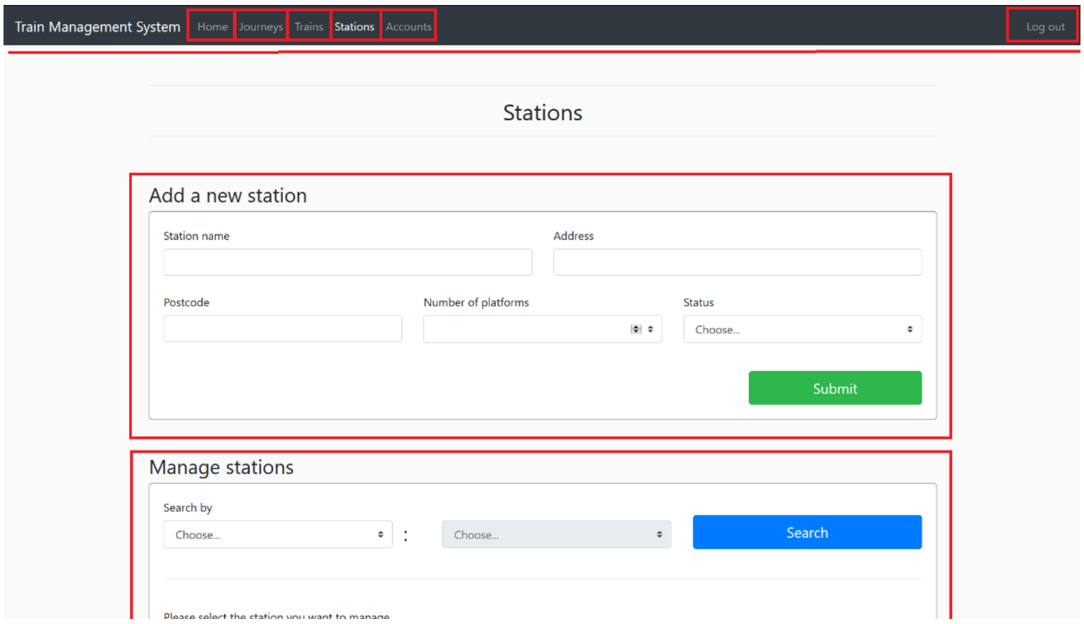
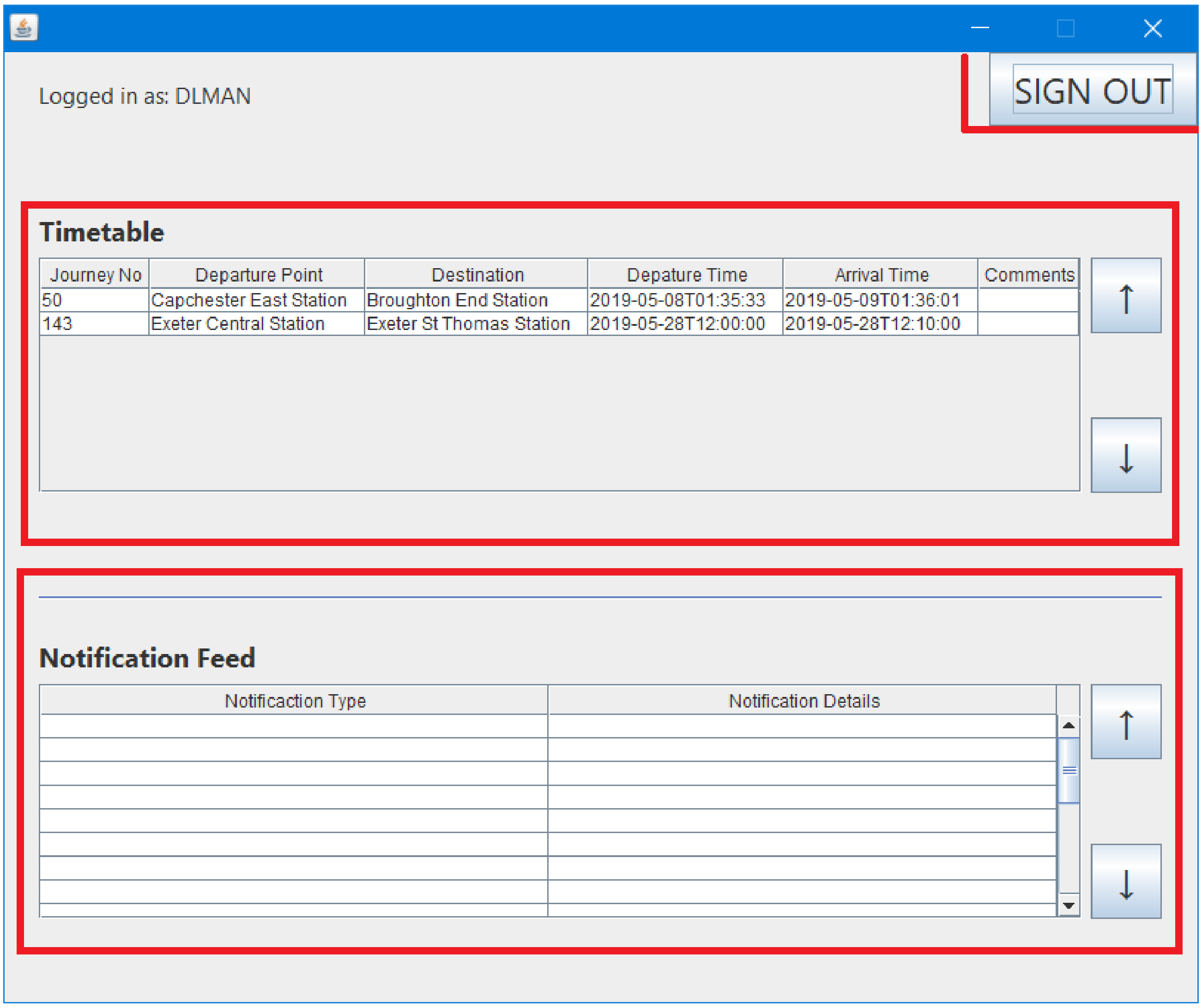


Figure 6 (Gestalt Principles in the Web Application)



Logged in as: DLMAN

Timetable

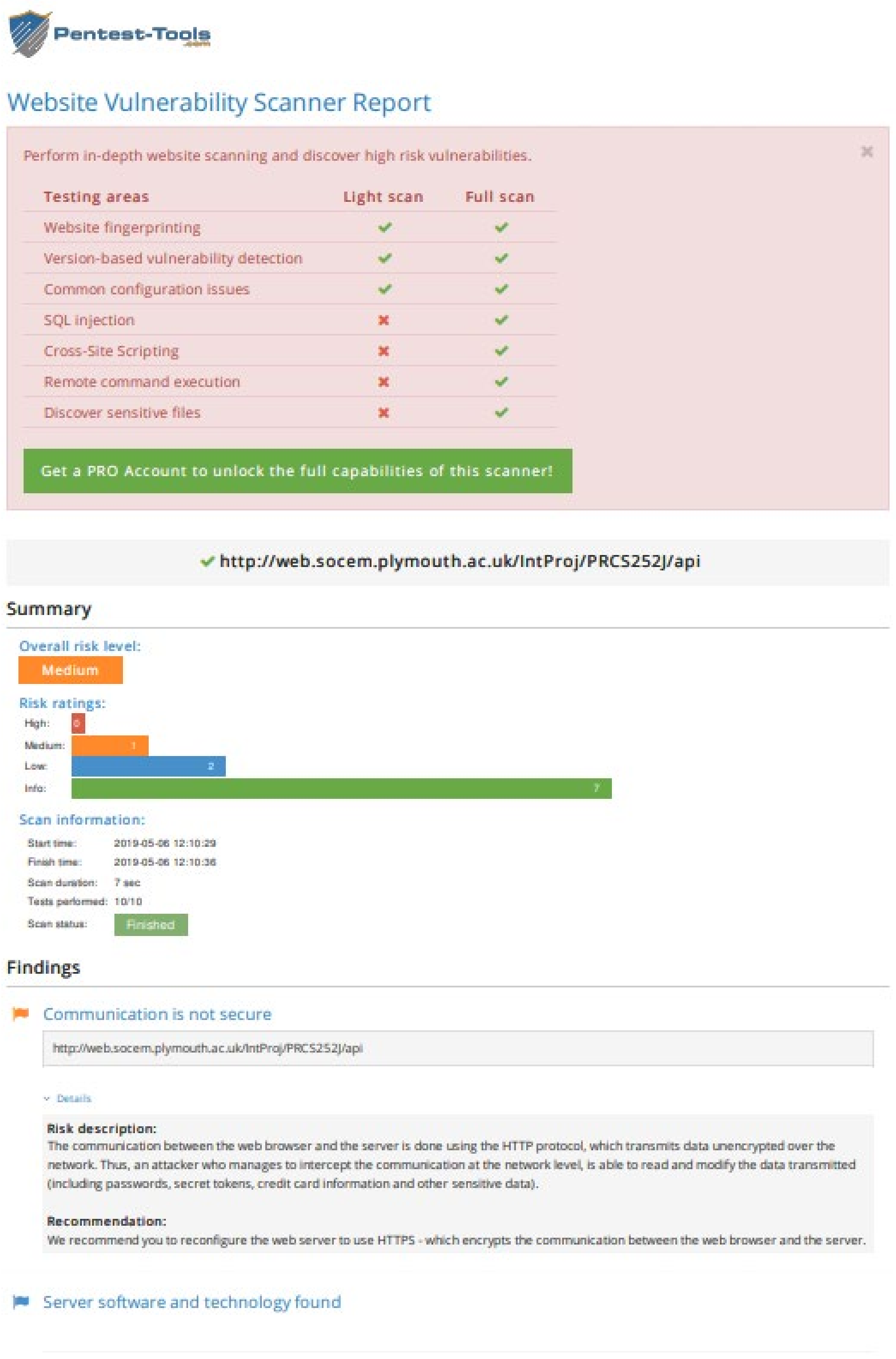
Journey No	Departure Point	Destination	Depature Time	Arrival Time	Comments
50	Capchester East Station	Broughton End Station	2019-05-08T01:35:33	2019-05-09T01:36:01	
143	Exeter Central Station	Exeter St Thomas Station	2019-05-28T12:00:00	2019-05-28T12:10:00	

Notification Feed

Notificaction Type	Notification Details

Figure 7 (Gestalt Principles in the Desktop Application)

Appendix B



Pentest-Tools .com

Website Vulnerability Scanner Report

Perform in-depth website scanning and discover high risk vulnerabilities.

Testing areas	Light scan	Full scan
Website fingerprinting	✓	✓
Version-based vulnerability detection	✓	✓
Common configuration issues	✓	✓
SQL injection	✗	✓
Cross-Site Scripting	✗	✓
Remote command execution	✗	✓
Discover sensitive files	✗	✓

Get a PRO Account to unlock the full capabilities of this scanner!

✓ http://web.socem.plymouth.ac.uk/IntProj/PRCS252J/api

Summary

Overall risk level:
Medium

Risk ratings:

High:	0
Medium:	1
Low:	2
Info:	7

Scan information:

Start time:	2019-05-06 12:10:29
Finish time:	2019-05-06 12:10:36
Scan duration:	7 sec
Tests performed:	10/10
Scan status:	Finished

Findings

Communication is not secure

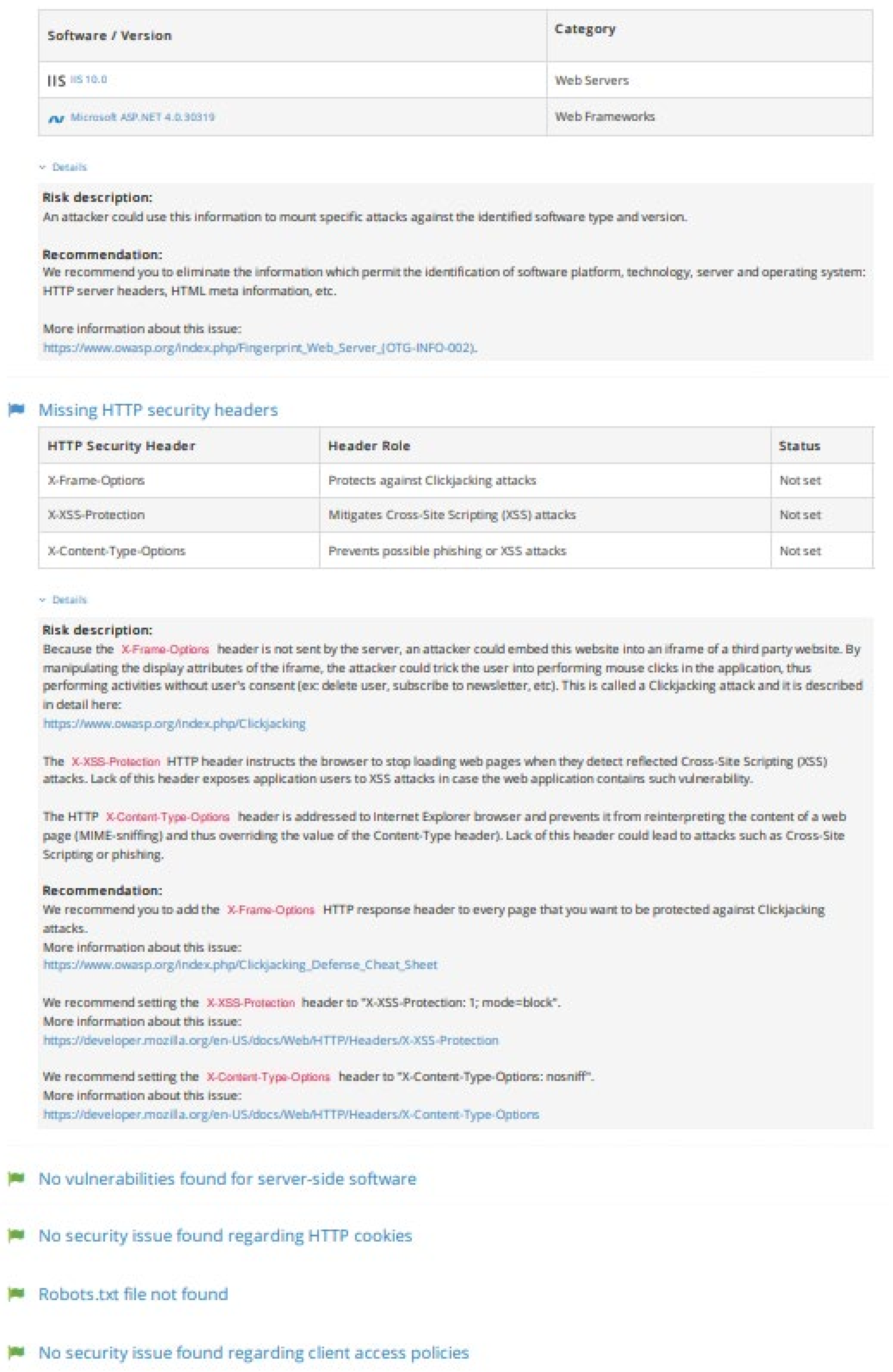
http://web.socem.plymouth.ac.uk/IntProj/PRCS252J/api

Details

Risk description:
The communication between the web browser and the server is done using the HTTP protocol, which transmits data unencrypted over the network. Thus, an attacker who manages to intercept the communication at the network level, is able to read and modify the data transmitted (including passwords, secret tokens, credit card information and other sensitive data).

Recommendation:
We recommend you to reconfigure the web server to use HTTPS - which encrypts the communication between the web browser and the server.

Server software and technology found



Software / Version	Category
IIS IIS 10.0	Web Servers
Microsoft ASP.NET 4.0.30319	Web Frameworks

Details

Risk description:
An attacker could use this information to mount specific attacks against the identified software type and version.

Recommendation:
We recommend you to eliminate the information which permit the identification of software platform, technology, server and operating system: HTTP server headers, HTML meta information, etc.

More information about this issue:
https://www.owasp.org/index.php/Fingerprint_Web_Server_(OTG-INFO-002).

Missing HTTP security headers

HTTP Security Header	Header Role	Status
X-Frame-Options	Protects against Clickjacking attacks	Not set
X-XSS-Protection	Mitigates Cross-Site Scripting (XSS) attacks	Not set
X-Content-Type-Options	Prevents possible phishing or XSS attacks	Not set

Details

Risk description:
Because the X-Frame-Options header is not sent by the server, an attacker could embed this website into an iframe of a third party website. By manipulating the display attributes of the iframe, the attacker could trick the user into performing mouse clicks in the application, thus performing activities without user's consent (ex: delete user, subscribe to newsletter, etc). This is called a Clickjacking attack and it is described in detail here:
https://www.owasp.org/index.php/Clickjacking

The X-XSS-Protection HTTP header instructs the browser to stop loading web pages when they detect reflected Cross-Site Scripting (XSS) attacks. Lack of this header exposes application users to XSS attacks in case the web application contains such vulnerability.

The HTTP X-Content-Type-Options header is addressed to Internet Explorer browser and prevents it from reinterpreting the content of a web page (MIME-sniffing) and thus overriding the value of the Content-Type header). Lack of this header could lead to attacks such as Cross-Site Scripting or phishing.

Recommendation:
We recommend you to add the X-Frame-Options HTTP response header to every page that you want to be protected against Clickjacking attacks.
More information about this issue:
https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

We recommend setting the X-XSS-Protection header to "X-XSS-Protection: 1; mode=block".
More information about this issue:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

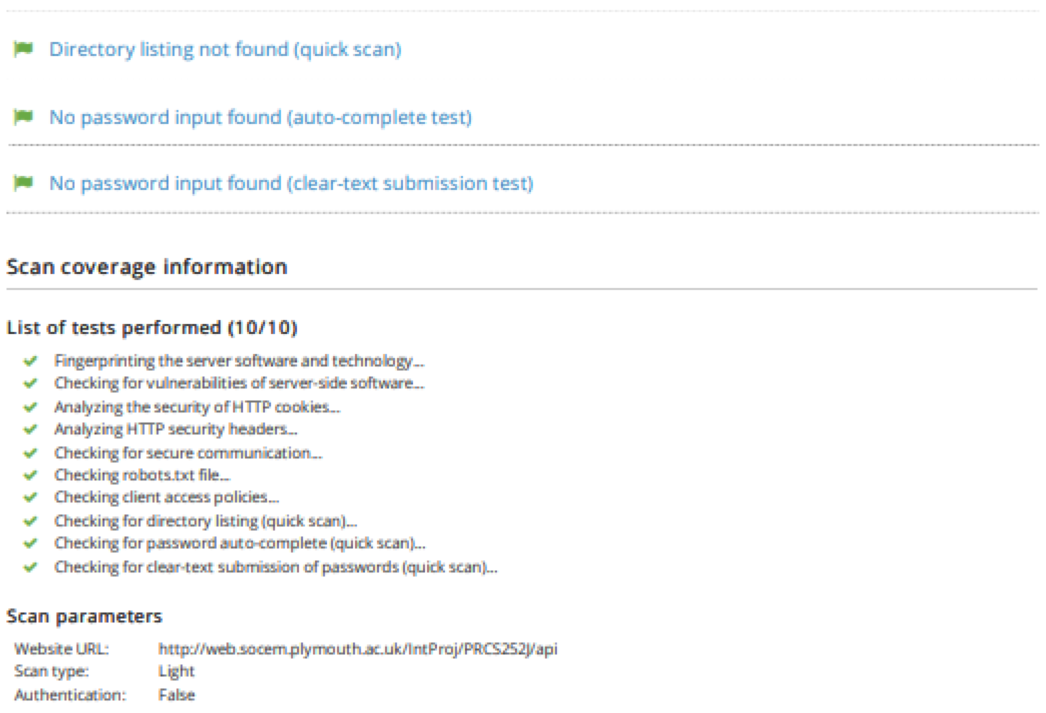
We recommend setting the X-Content-Type-Options header to "X-Content-Type-Options: nosniff".
More information about this issue:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

No vulnerabilities found for server-side software

No security issue found regarding HTTP cookies

Robots.txt file not found

No security issue found regarding client access policies



Directory listing not found (quick scan)

No password input found (auto-complete test)

No password input found (clear-text submission test)

Scan coverage information

List of tests performed (10/10)

- ✓ Fingerprinting the server software and technology...
- ✓ Checking for vulnerabilities of server-side software...
- ✓ Analyzing the security of HTTP cookies...
- ✓ Analyzing HTTP security headers...
- ✓ Checking for secure communication...
- ✓ Checking robots.txt file...
- ✓ Checking client access policies...
- ✓ Checking for directory listing (quick scan)...
- ✓ Checking for password auto-complete (quick scan)...
- ✓ Checking for clear-text submission of passwords (quick scan)...

Scan parameters

Website URL:	http://web.socem.plymouth.ac.uk/IntProj/PRCS252J/api
Scan type:	Light
Authentication:	False