[Ability Net UT Processes] Requirements Specification

Version 1.0

April 26, 2020

Table of Contents

Executive Summary

PROJECT DETAILS

Project Name	AbilityNet User Testing Process (UTP)
Project Type	***New Initiative***
Project Start Date	31st January 2020
Project End Date	19 June 2020
Project Manager	Neeraj Rai

VERSION CONTROL

Version	Date	Comment
0.0	08/02/2020	• Planned the structure of document and created styles for headings.
1.0	12/02/2020	• Briefing with stakeholders (1.3- 4.3) • Model of existing system (Plain English)
2.0	15/02/2020	• Use case diagram
3.0	22/02/2020	• Use case specification
4.0	23/02/2020	• Found conflicts with use cases after use case specification • Added and removed use cases. Found conflicts for findTesters in different use cases so it was separated onto findTester and pickTester. • Generalized requests. I.e. confirmRequest, deleteRequest, etc.
5.0	27/02/2020	• Use case prioritization
6.0	28/02/2020	• Analysis class diagram
7.0	29/02/2020	• Made changes to use case diagram after finding a more elegant solution on analysis class diagram.
8.0	1/03/2020	• Structure of document, presentation, appropriate language, diagrams, work plan added.
9.0	15/03/2020	• Made changes to requirement specifications after meeting with stakeholders.
10.0	22/03/2020	• Modelled design class diagram, ERD
11.0	29/03/2020	• Made changes to design class diagram after starting sequence diagram
12.0	05/04/2020	• Completed sequence diagram, state machines
13.0	17/04/2020	• Completed GUI modelling classes and site map and UI design

OVERVIEW

This document defines the high-level requirements AbilityNet UTP. It will be used as the basis for the following activities:

- Documenting project log entries.
- Creating system design.
- Developing test plans, scripts, and test cases.
- Demonstrating project completion with software in operation.
- Assessing project success with implementation report.

GLOSSARY OF TERMS

Term/Acronym	Definition
UTP	User Testing Process
UC	Use case
AN	Ability Net

PROJECT OVERVIEW

1.1 4.1 Project Overview and Background

1.1.1 Overview

The AbilityNet UTP is a project which provides an interface for ease of access to tester booking management, minimize manual tasks by automating processes and allowing consultants and planners to update the data dynamically to better reflect daily work. The entire user journey from registering as a tester to completing testing sessions to payments shall be automated.

If time enables, there is a scope to implement a survey functionality for companies to post requests for surveys which can be filled out by testers with payments authorized.

1.1.1 Current situation and challenges

The senior consultant members and planning team whom I spoke to had keen interest in this system being developed as much time is spent on admin planning, booking and delivering tasks. They experience difficulties in finding the right testers as consultants who do the testing cannot update the data and NetSuite does not have a search function which means that consultants and planners spend a lot of time to find people. This results in the same people on the top of the database being picked for testing - which leaves many testers not being fully utilized.

AbilityNet uses a business management software called NetSuite to centralize their business processes. They use this system to record and query user testing records. However, the suite is not optimized for such tasks. I have discussed the existing system with senior consultants and planners who believe that better a system is required.

I aim to develop an accessible user interface which captures the data and develop features such as querying, booking, updating notes and more, depending on feedback. The aim is to save the team's time and maximize productivity by cutting down on admin work.

The current system itself is far too complex (comprised of layered systems)- making it difficult for consultants to perform simple tasks, especially recording and querying for users with disabilities. Such a platform always poses another issue; that is, it is too general.

a. Consultants must manually view hundreds of records to find a user for testing.
b. Planners must manually sift through web submissions to see if data entered may be valid. Spelling mistakes, details about disabilities, etc.
    a. May have to query tester in case of mistakes.
c. Planners have to manually send reminder emails to testers.
d. Consultants cannot add meta data about a tester such as attendance/ reliability.
e. There are no opportunities to utilize the power of the database and scale the business. E.g. Survey services with users with disabilities.
f. Consultants cannot verify that testers came to testing sessions
g. Finance must manually send payments

The current system wastes a lot of time for consultants and planners, and the task itself is tedious.

1.1.2 Objectives

Overall aim: Lower costs, save time, better user experience/ service for planners and consultants.

The proposed system will make it much easier to manage user testing records and query them for scheduling testing dates. The system could be further extended to automate the phase of scheduling the dates by automatically finding free dates within schedule, also taking into account distance of testers. It will essentially find the cheapest tester available given constraints such as testing for certain disability or client, very quickly and with very little manual admin input.

User:

a. Signup form
b. Calculate expected pay from hours worked (secondary)

Admin:

c. Filtering people by postcode, disabilities, bank
d. Manual/ (Timed) reminder for tester's scheduled dates
e. Booking people on dates *Manually, account for meta data
    a. Calendar (secondary)
f. Manual verification on integrity of data (add to database button), on */*requests endpoint

Consultants:

g. Updating notes on user, on reliability or attendance.
h. Confirming people who turned up
    a. Send email to finance
    b. Potential for online payment processor (secondary)

Potential work:

i. Making an interface for companies to send private surveys to testers without disclosing tester's information.
j. Automated payment processor through Stripe API
k. Dashboard analytics of revenue from testers over time.
l. Algorithm for matching tester with a prospective client, based on previous history/ analytics.

1.2 4.2 Project Dependencies

- Express
- MongoDB, React, AWS,
- AbilityNet NetSuite DB for extraction of existing data,
- MapsAPI
- PayPal

4.3 Stakeholders

The following comprises the internal and external stakeholders whose requirements are represented by this document:

	Stakeholders
1.	Raphael Clegg-Vinell, Senior Consultant. Point of contact for requirement engineering and system design.
2.	Holly Rushton, Planning Manager, Point of contact for organising meetings.
3.	Louise Ashley, Project Manager & User Tester Admin, Point of contact for understanding Tester journey and developing better UX.

1.3 Initial briefing notes

Notes from the initial briefing

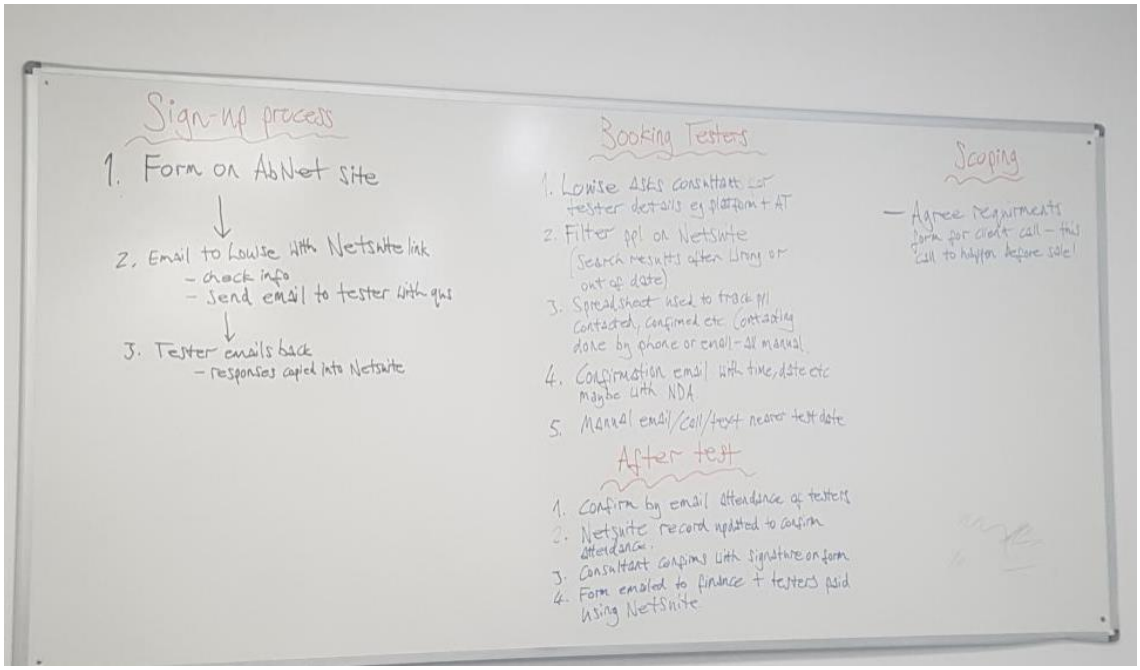


Figure 1 Initial system briefing on current AN UTP

Survey example which is sent to users with insufficient detail in their application.

Louise Ashley
Wed 12/02/2020 13:27
Neeraj Rai; Raphael Clegg-Vinell

Good Morning / Afternoon...,

Thank you for your application to become a user tester with AbilityNet. To complete your record, please answer the questions below in as much detail as possible.

1. How would you rate your tech skills?
a. Beginner
b. Intermediate
c. Expert

2. Which desktop and mobile operating systems do you use?

3. Do you use any Assistive Technology? If so, please specify which one(s).

4. How would you describe your proficiency with these?

5. Do you use any specialised hardware? If so, please specify which one(s).

6. Lab testing sessions are usually based in Central London. Which testing types are you available for?
a. Lab based testing
b. Remote testing
c. Both

7. Do you require assistance on the day of testing?
a. Wheelchair access
b. Sighted guide
c. Service animal
d. BSL interpreter
e. Other:

8. Is there any additional information you would like to share with us?

9. We require standby user testers in case of last-minute cancellations. Standby user testers receive £20, in addition to full payment if you are called in to attend the user testing session. Would you be happy to be a standby tester?

Figure 2 Example of how additional information is collected over email

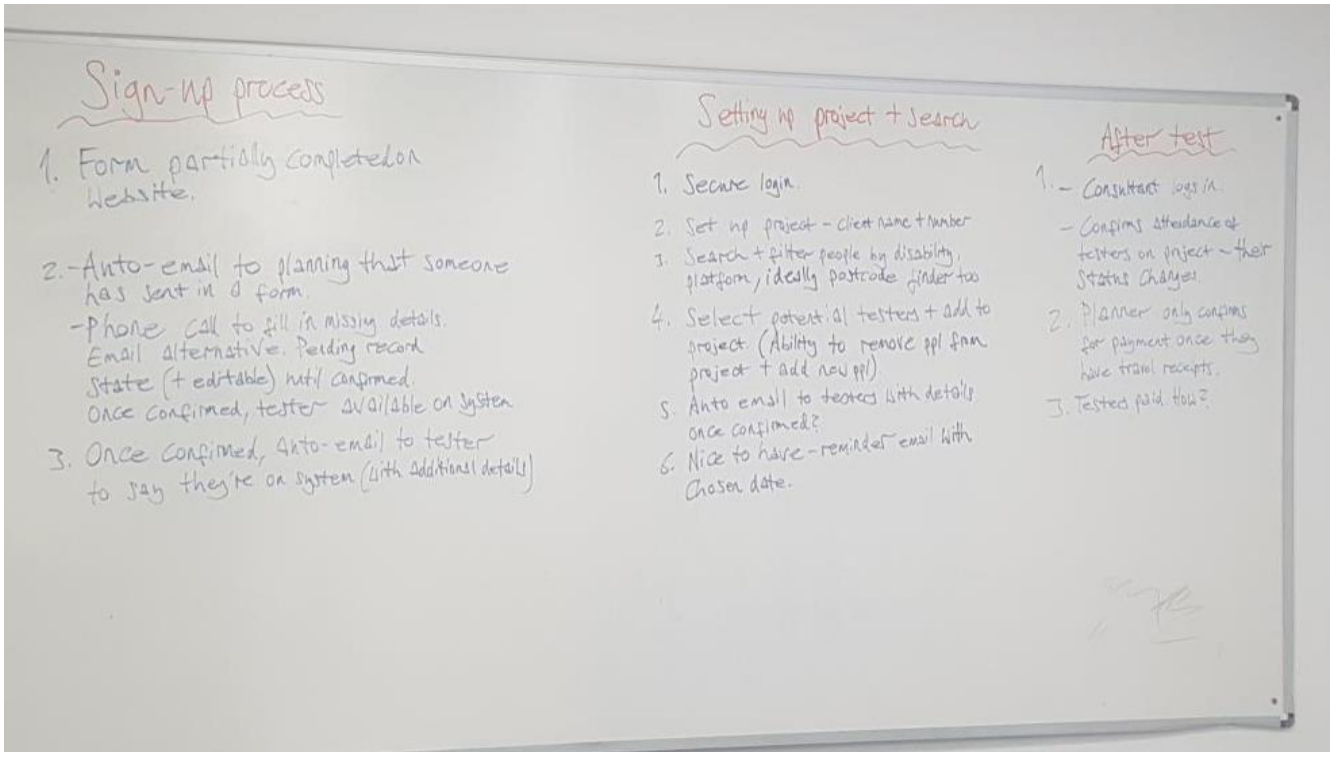


Figure 3 Proposed high-level overview of new AN UTP

2. Requirements

Priority Definitions

The following definitions are intended as a guideline to prioritize requirements.

- Priority High – The requirement is a "must have" as outlined by system requirements
- Priority Medium – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
- Priority Low – The requirement is a "nice to have" which may include new functionality.

The following table is a high-level overview for requirements and its priorities. Generalized use cases have not been mentioned- only the specialized one carrying functionality.

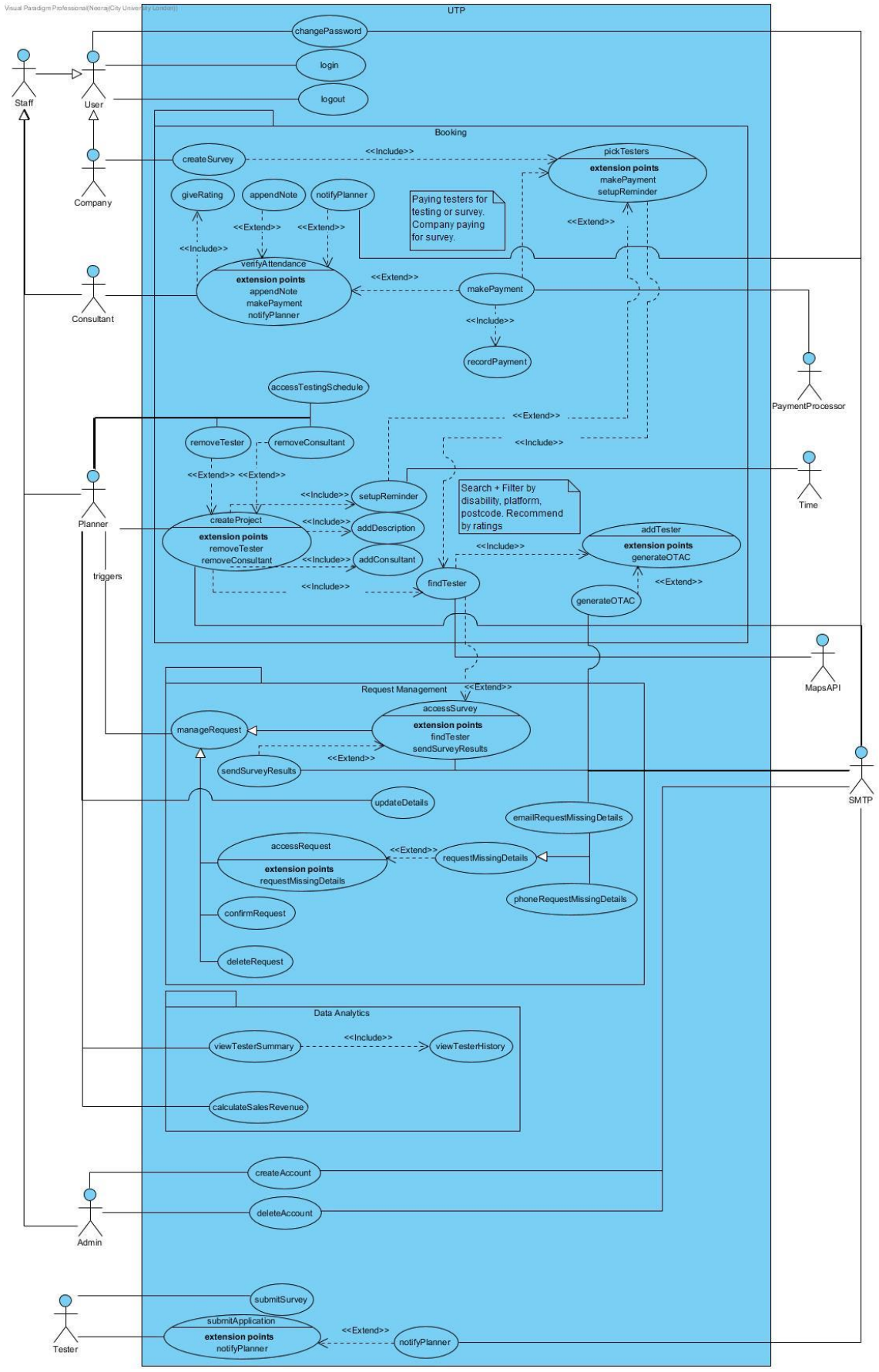
2.1 Use case prioritization

Use Case Name	Use Case ID	Use Case Priority	Use Case Description
login	UC01	High	Users can log into accounts with designated roles. Roles can allow access to different functionality within the implementation.
logout	UC02	High	Users can logout of accounts with designated roles. This is a security measure to prevent unauthorized access to accounts.
createSurvey	UC03	Medium	Brief description: Companies can create a survey request by adding or remove testers with certain disabilities to a 'survey' request record. Companies are not exposed to the users by simply a fragment of their description for testing such as disabilities or age. Company role will have a single page endpoint. (No data analytics may be done for scope of project).
verifyAttendance	UC04	Medium-High	Consultants can verify whether a tester showed up on a testing session
giveRating	UC05	Medium	The consultant can provide ratings for testers on completion of a testing session to keep an active record of the user. Ratings provide a basis for competencies with assistive technologies, general motivation for work and easiness.
appendNote	UC06	Medium	Consultants can add descriptions to testers proceeding a testing session to highlight any issues. Notes such as competencies with assistive technologies, general motivation for work and easiness.
notifyPlanner	UC07	Medium	The system sends an email message/ notification to the planner when a survey or testing project request has been submitted by users.
makePayment	UC08	High	The system can trigger make payment once a testing session has been completed and testers are paid money or

			payment is made to AbilityNet once companies pick testers for surveys.
recordPayment	UC09	High	The system records payment transaction carried out by online payment processor for payments made to testers for testing once a session is complete and for companies to pay for survey.
createProject	UC11	High	Planner can create a project, add or remove testers or consultants, change description and email testers.
accessTestingSchedule	UC12	Medium-High	The user can trigger an access testing schedule button which searches all the bookings and surveys due and generates a calendar for when testing is due.
addConsultant	UC13	Medium-High	The planner can add consultants from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. Without this feature, a project cannot be started as you need consultants to manage a project.
removeConsultant	UC14	Medium-High	The planner can remove consultants from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. Without this feature, a project cannot be started as you need consultants to manage a project.
addDescription	UC15	Medium	Planner can add and change change descriptions of projects.
setupReminder	UC16	Medium-Low	The system sends an email message/ notification to the planner when a survey or testing project request has been submitted by users.
findTester	UC17	High	Planners should be able to find testers from the database with specific disabilities, assistive technology competency, postcode, age, etc. The planners should then be able to
pickTester	UC18	Medium-Low	The user with role ‘company’ picks testers from a list of testers which are rendered after filtering.
addTester	UC19	Medium-High	The planner can add testers from a testing project record. This provides flexibility to the project as changes to testers can be made at any time. Without this feature, a project cannot be started as you need testers to manage a project.
generateOTAC	UC20	Medium-High	An OTAC webpage is generated which allows testers to submit their responses to surveys. The URL is sent to them through SMTP.
removeTester	UC21	Medium	The planner can remove testers from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. This prevents incorrect entries recorded in the database.

accessSurvey	UC23	Medium-Low	A planner can view survey requests sent by companies and confirm testers for the survey. Subsequently, email is sent to the testers about the survey and OTAC is generated for survey submission.
sendSurveyResults	UC24	Medium-Low	The planner sends survey results to the company email address. The status code for the survey is set to completed.
accessRequest	UC25	Low	The planner can access requests from people to become testers. User will be contacted via email or phone for any failure to capture information initially. There may be a button or link on the access request page which can be used to confirm or delete requests.
emailRequestMissingDetails	UC27	High	The planner triggers a button to send a document via email that aims to request missing details regarding user information.
phoneRequestMissingDetails	UC28	High	The planner phone calls a user a fill in missing or undetailed information regarding a user.
confirmRequest	UC29	High	The planner can confirm requests for users to become a tester. This is if user is suitable for the role.
deleteRequest	UC30	High	The planner can delete requests for users to become a tester. This is if a user is not suitable for the role.
updateDetails	UC31	High	Information confirmed through phone call or email, depends on user's preference. Such information can be updated by the planner.
viewTesterSummary	UC32	High	Planners should be able to view tester summary, which is a record containing tester data such as number of projects which the tester participated in, competencies ratings on those projects, who the project was initially led by, standard information about the tester, etc.
viewTesterHistory	UC33	High	Planners should be able to view tester history, which is a record containing *history of* tester data such as number of projects which the tester participated in, competencies ratings on those projects, who the project was initially led by, standard information about the tester- for all previous testing completed.
createAccount	UC34	High	The Admin creates a new user account for staff members.
changePassword	UC35	Medium	Users can request to change password and the system will change their password to their new password given that their initial password verification is correct.
notifyPlanner	UC36	High	The system sends an email message/ notification to the planner when a survey or testing project request has been submitted by users.

2.2 Use case diagram



2.3 Deferred Requirements

Requirements that have been deleted after approval or that may be delayed until future versions of the system. Also captures use cases that do not require a specification. I.e. generalized cases.

UC ReqID	Name	Comments	Priority	Date	Approval
10	recordCost, calculateSalesRevenue	'recordCost and recordPayment' initially provide the same functionality. Payment is how much is paid to testers and cost is amount paid by companies for testing. Not needed. Cost is a fixed constant.	Low	28/02/2020	AN SMT
22	manageRequest	Use case specification not need to write as it covers too many causes under requests. Use cases will be individually explored.	High	28/02/2020	AN SMT
26	requestMissing Details	Use case specification not need to write as it covers too many causes under requests. Use cases will be individually explored.	High	28/02/2020	AN SMT

2.4 Use case description – main flow

N.B. Is it to be assumed that the use case will begin on the first action on the main flow, right after precondition is made. For cases with **extend** or **include** reference to the start is made.

2.4.1 Create account

Use case ID: **35**	Use case: CreateAccount
Brief description: The Admin creates a new user account for staff members.	
Primary Actor: Admin	
Secondary Actor: None	
Preconditions: The actor is logged onto the system. The actor is on the admin panel where new accounts can be created.	
Main flow: 1. The system asks the actor to input details for a new account. 2. The actor inputs the login details such as name, email and role. 3. The system requires the actor to confirm input details. 4. The actor confirms input details. 3. The system generates a one-time password. 4. The system records new user account details in the database. 5. The system displays a message showing the outcome of the operation. 6. IF newUserAccount.role == staff 6.1 user account details which can be passed onto a staff member verbally. 7. ELSE IF newUserAccount.role == company 7.1 The system sends company email with company user login credentials.	

Postconditions: A new account is created for users with a role such as consultant, planner or company. The users can use the credentials to log into the system and have an active session.

2.4.2 Login

Use case ID: **01**	Use case: Login
Brief description: Users can log into accounts with designated roles. Roles can allow access to different functionality within the implementation.	
Primary Actor: User	
Secondary Actor: None	
Preconditions: The actor must be on the root/ login page. The actor must not be logged in to an active session.	
Main flow: 1. The actor enters in their username and password 2. The system creates a login session 3. The system verifies that the actor's input is valid or not 4. The system will determine the role of the user 5. According to the role the system identified, the system will display the menu accordingly 6. The system closes session	
Postconditions: The user is redirected to dashboard. Different containers will be rendered depending on user's role.	
Alternative flow: InvalidInputLogin, Cancel	

2.4.3 Create Project

Use case ID: **11**	Use case: createProject
Brief description: Planner can create a project, add or remove testers or consultants, change description and email testers.	
Primary Actor: Planner	
Secondary Actor: Time (include: setupReminder)	
Preconditions: An existing project name must not be used while creating a new record.	
Main flow: 1. The planner clicks on a 'Create Project' button. 2. Include(addDescription)	

3. The Planner specifies client name and project number and number of testers required. 4. Include(addConsultant) The system asks the planner to specify a consultant ID or name. 5. The planner specifies a consultant ID or name. 6. IF the entered id exists in database 6.1. Provide successful input visual cues 7. IF new consultant is required extend(removeConsultant) 8. WHILE the number of testers required is less than the numbers of testers selected 8.1 The system asks the Planner to select testers 8.2 The Planner filters potential testers which meet project requirements 8.3 Include(findTester) extend(removeTester) 9. The system awaits project confirmation and asks the user to specify reminder dates. 10. The planner specifies reminder dates and clicks on a button to confirm project. 11. The system creates a project record and reminders are sent to the testers through the SMTP. The system sets the project status to 'Booked'.
Postconditions: A new project record will be created with status 'Booked'. A reminder will be setup for testers and will be sent to them using email, or phone call.
Alternative flow: SessionTimeout, Cancel

2.4.4 Access request

Use case ID: **25**	Use case: accessRequest
Brief description: The planner can access requests from people to become testers. User will be contacted via email or phone for any failure to capture information initially. There may be a button or link on the access request page which can be used to confirm or delete requests.	
Primary Actor: Planner, Tester	
Secondary Actor: SMTP	
Preconditions: Planner must be logged in.	
Main flow: 1. This UC starts when the planner clicks on a view tester request link. 2. The system finds requests with status missing or waiting. 3. The planner can filter or sort the requests as wanted and renders the list of users. 4. The planner accesses a request record. Extend(requestMissingDetails)	
Postconditions: The system displays a list of request records to the planner after filtering or sorting. The planner accesses a record and is rendered.	
Alternative flow: DBConnectionError, sessionTimeout	

2.4.5 Verify attendance

Use case ID: **04**	Use case: verifyAttendance
Brief description: Consultants can verify whether a tester showed up on a testing session.	
Primary Actor: Consultant	
Secondary Actor: PaymentProcessor	
Preconditions: The consultant must be logged on. The consultant is on the root/ dashboard page.	
Main flow: 1. The system renders existing project records with status 'Booked'. 2. The consultant filters or sorts project and accesses a project record. 3. WHILE all the testers have not been confirmed. 3.1. The system asks the consultant to confirm a user's attendance. 3.2. The consultant confirms attendance and prevents the user from confirming the same tester again. 3.3. Include(giveRating) The system renders a form which asks the user to rate the tester's competencies today. 3.4. The consultant types into the form and submits it. 3.5. The system records, processes and updates tester records. extend(addNote) extend(notifyPlanner) 4. The system provides cues that attendance has been confirmed for all users. 5. The system sets the project's status to 'completed' Extend(makePayment)	
Postconditions: All the tester's attendance has been confirmed. Each of the attendants will have a rating given and may have some notes added. Project status is set to 'completed'.	
Alternative flow: DBConnectionError	

2.4.6 Find tester

Use case ID: **17**	Use case: findTester
Brief description: Planners should be able to find testers from the database with specific disabilities, assistive technology competency, postcode, age, etc. The planners should then be able to	
Primary Actor: Planner	
Secondary Actor: MapsAPI	
Preconditions: Journey 1: A user logged onto a company account should create a survey (createSurvey) Journey 2: A user logged onto a planner account created a project (createProject)	
Main flow: 1. The system renders a list of tests which are neither sorted nor filtered. Filter = [name, id, rating, assistive technology, age, postcode, etc.] 2. The system asks the planner to sort or filter testers by their meta data such as disabilities, assistive technology competency, postcode, age, etc.	

3. The planner selects filters using checkboxes and inputs data within forms. 4. IF filter is 'name/id/assistive technology competency' The system will render all tester items with tester.filter == filter IF filter is 'rating' The system will sort the tester ratings by descending value. The system renders the list of sorted testers. The planner may trigger the button to sort by ascending value. IF filtering is 'age' The system renders a range input field to select lower and upper bounds. The planner inputs the range of age. The system renders the list of testers who pass the age requirement from the database. IF filter is 'postcode' The system will communicate with MapsAPI and generate a placeholder local map. The system will generate a form where planners can input postcode filters. The planner inputs postcode filters The system will send parameters from the form to the MapsAPI and persistent storage and request map which shows nearby testers. The system will process the response and render a data summary table, along with the map and recommendations for testers. The planner can further change form values and fine tune the data so that the ideal tester is found. 5. Include(addTester) The system renders a button next to each tester which the planner can trigger to add a tester to a project. 6. The user triggers buttons and add testers to the project or survey.
Postconditions: User picks testers and populates them into a project or survey.
Alternative flow: DBConnectionError, MapException, noActorFound

2.4.7 View tester summary

Use case ID: 32	Use case: viewTesterSummary
Brief description: Planners should be able to view tester summary, which is a record containing tester data such as number of projects which the tester participated in, competencies ratings on those projects, who the project was initially led by, standard information about the tester, etc.	
Primary Actor: Planner	
Secondary Actor: None	
Preconditions: 1. User must be logged onto a planner account. 2. Planner must create a project/ survey to render 'find a tester' form, or they must be on the dashboard -which shows overall data analytics on the testers.	
Main flow: 1. Planner views the summary information. 3. The system generates a form where the user can filter testers and view their data. 4. The user enters a tester ID or tester meta-data (such as disability or name) Extend(findTester) 5. The system generates a list of testers from the query and populates them into cards. 6. The user accesses individual tester records.	

7. The system renders individual tester record. 8. Include(viewTesterHistory) The system finds and generates each project/ survey a tester has participated in. 9. Data on testers must be populated for user to view a summary record. This is achieved through planners creating projects/ surveys and consultants 10. The system requests GraphQL API to generate graphs for data. 11. Graphs are rendered on the user's page.
Postconditions: 1. Planner views overall tester summary 2. Planner views specific tester summary 3. Data is visualized using GraphQL
Alternative flow: DBConnectionError

2.4.8 Access survey

Use case ID: **23**	Use case: accessSurvey
Brief description: A planner can view survey requests sent by companies and confirm testers for the survey. Subsequently, email is sent to the testers about the survey and OTAC is generated for survey submission.	
Primary Actor: Planner	
Secondary Actor: SMTP	
Preconditions: 1. User must be logged onto a planner account. 2. Planner must be on 'requests' page where requests are incoming.	
Main flow: 1. The system searches the database for survey records and renders them as a card. 2. Planner accesses survey record. 3. include(findTesters): use case written 4. extend(generateOTAC) 5. The system generates a webpage using an OTAC token. 6. The SMTP will send confirmation email to testers along with URL to OTAC session.	
Postconditions: 1. Planner accesses a survey request and assigns testers to survey. 2. SMTP will send confirmation emails to testers.	
Alternative flow: DBConnectionError	

2.4.9 Change password

Use case ID: **37**	Use case: changePassword
Brief description: Users can request to change password and the system will change their password to their new password given that their initial password verification is correct.	
Primary Actor: User	
Secondary Actor: SMTP	

Preconditions: The user must be logged onto an account. (request password for forgotten passwords will not be supported in this system scope). The user must be on the 'Profile page' where they can request to change password.
Main flow: 1. The system renders a change password form. 2. The user enters initial password and new password 3. The system verifies that the initial password is correct and stores the new password on the database. 4. The system will send a confirmation email to the user that their password has changed.
Postconditions: The user has their password changed.
Alternative flow: InvalidInputDetails

2.4.10 View tester history

Use case ID: 33	Use case: viewTesterHistory
Brief description: Planners should be able to view tester history, which is a record containing *history of* tester data such as number of projects which the tester participated in, competencies ratings on those projects, who the project was initially led by, standard information about the tester- for all previous testing completed.	
Primary Actor: Planner	
Secondary Actor: None	
Preconditions: 1. User must be logged onto a planner account. 2. Planner must create a project/ survey to render 'find a tester' form, or they must be on the dashboard -which shows overall data analytics on the testers.	
Main flow: 1. The user accesses individual tester records. 2. The system renders individual tester record. 3. Include(viewTesterHistory) The system finds and generates each project/ survey a tester has participated in. 4. Data on testers must be populated for user to view a summary record. This is achieved through planners creating projects/ surveys and consultants 5. The system requests GraphQL API to generate graphs for data. 6. Graphs are rendered on the user's page.	
Postconditions: 1. Planner views overall tester summary and triggers 'history' button. 2. System generates a list of all instances of a tester's testing history. 3. Data is visualized using GraphQL	
Alternative flow: DBConnectionError	

2.4.11 Logout

Use case ID: **02**	Use case: Logout
Brief description: Users can logout of accounts with designated roles. This is a security measure to prevent unauthorized access to accounts.	
Primary Actor: User	
Secondary Actor: None	
Preconditions: The actor must be logged onto an active session	
Main flow: 1. The user triggers logout button. 2. The system deletes the active session and redirects the user to the root page.	
Postconditions: The user is logged out of their session. The user is redirected to root.	
Alternative flow: DBConnectionError	

2.4.12 Create survey

Use case ID: **03**	Use case: createSurvey
Brief description: Companies can create a survey request by adding or remove testers with certain disabilities to a 'survey' request record. Companies are not exposed to the users by simply a fragment of their description for testing such as disabilities or age.Company role will have a single page endpoint. (No data analytics may be done for scope of project).	
Primary Actor: Company	
Secondary Actor:	
Preconditions: A user is logged onto an account with role 'Company'.	
Main flow: 1. The system renders a survey form with input fields. 2. The user inputs their survey questions into the input fields. 3. The system does a first pass validation test to prevent spam or unintended submissions. 4. The system confirms questions and stores them as a temporary record. 5. Include(pickTesters) The system awaits survey confirmation and asks the user to specify reminder dates. 6. The user specifies reminder dates and clicks on a button to confirm survey. 7. The system creates a survey record and reminders are sent to the testers and planner through the SMTP. The system sets the survey status to 'Booked'.	

Postconditions: A new survey record will be created with status 'Booked'. A reminder will be setup for testers and planners which will be sent to them using email.
Alternative flow: SessionTimeout, Cancel

2.4.13 Pick testers

Use case ID: **18**	Use case: pickTester
Brief description: The user with role 'company' picks testers from a list of testers which are rendered after filtering.	
Primary Actor: Company	
Secondary Actor: SMTP	
Preconditions: 1. The system renders a survey form with input fields. 2. The user inputs their survey questions into the input fields. 3. The system does a first pass validation test to prevent spam or unintended submissions. 4. The system confirms questions and stores them as a temporary record.	
Main flow: 1. The system renders a form for filtering testers. 2. The user inputs into the form what kind of tester they would want the survey done by. 3. The system uses 'findTesters' functionality to scan through the database for suitable testers and removes private information and stores only high-level information such as disability, age, etc. 4. The system renders the new high-level information on the possible testers. 5. The user picks most suitable testers for their needs and submits form. 6. Extend(makePayment) 7. Extend(setupReminder)	
Postconditions: Testers are picked for a survey by companies. A reminder will be setup for testers for survey and planners, which will be sent to them using email.	
Alternative flow: DBConnectionError, invalidPayment.	

2.4.14 Notify planner

Use case ID: **07**	Use case: notifyPlanner
Brief description: The system sends an email message/ notification to the planner when a survey or testing project request has been submitted by users.	
Primary Actor: Company, Tester	
Secondary Actor: SMTP	
Preconditions:	

1. A user must apply for becoming a tester or a company must submit a survey request.
Main flow: 1. The use case starts when the user submits tester request or company submits a survey request. 2. The system finds all the users in the database with role planner. 3. The system sends each of the planners an email to notify that a new request has been placed. 4. IF a planner has a session active 4.1 A visual cue and message is provided that a new request has been made. 4.2 The message contains a link which redirects the user to the request record.
Postconditions: Planner is made aware of any new requests made by testers or companies.
Alternative flow: DBConnectionError

2.4.15 Append note

Use case ID: **06**	Use case: appendNote
Brief description: Consultants can add descriptions to testers proceeding a testing session to highlight any issues. Notes such as competencies with assistive technologies, general motivation for work and easiness.	
Primary Actor: Consultant	
Secondary Actor:	
Preconditions: 1. The user must be logged onto a consultant account. 2. Testing session must be completed. Session='completed testing'.	
Main flow: 1. The use case starts when the testing is completed for a testing project. 2. The consultant triggers 'testing complete'. 3. FOR each of the testers 3.1. The system renders previous notes on a user 3.2. The system asks if the user wants to add a new note. 3.3. The user adds a note or triggers next button. 3.4. IF a note is added 3.4.1. Record the note and append it onto the tester's record. 3.5 include(giveRating) The system moves onto giveRating.	
Postconditions: The system offers the user to append a note to a tester record. The system appends any notes mentioned to the tester's record.	
Alternative flow: DBConnectionError	

2.4.16 Make payment

Use case ID: **08**	Use case: makePayment

Brief description: The system can trigger make payment once a testing session has been completed and testers are paid money or payment is made to AbilityNet once companies pick testers for surveys.
Primary Actor: PaymentProcessor
Secondary Actor:
Preconditions: 1. UC pickTesters -> extend(makePayment) (6) 1.1. The system uses 'findTesters' functionality to scan through the database for suitable testers and removes private information and stores only high-level information such as disability, age, etc. 1.2. The system renders the new high-level information on the possible testers. 1.3. The user picks most suitable testers for their needs and submits form 2. UC verifyAttendance -> (5) 2.1. The system confirms that attendance has been confirmed for all users. 2.2. The system sets the project's status to 'completed'
Main flow: 1. If UC starts at extend(pickTester) (6) 1.1. The system requests payment processor a payment transaction to be completed for surveys. The money is paid to AN. Amount is TBC. 1.2. The system creates a payment record and invoiced to planner. 1.3. The system creates a payment record with status 'awaitingAcceptance' 2. IF UC starts at extend(verifyAttendance) (5) 2.3. The system requests payment processor a payment transaction to be completed for bookings. The money is paid to testers at a 10 pound per person rate. 2.4. The system creates a payment record with status 'awaitingAcceptance' 3. The system receives a response from PaymentProcessor that payment has been accepted. 4. The system changes payment status to ='completed'.
Postconditions: Payment is processed by PaymentProcessor and transaction recorded.
Alternative flow: invalidPayment

2.4.17 Give rating

Use case ID: **05**	Use case: giveRating
Brief description: The consultant can provide ratings for testers on completion of a testing session to keep an active record of the user. Ratings provide a basis for competencies with assistive technologies, general motivation for work and easiness.	
Primary Actor: Consultant	
Secondary Actor:	
Preconditions: 1. The user must be logged onto a consultant account. 2. Testing session must be completed. Session='completed testing'.	
Main flow: 1. The use case starts when the testing is completed for a testing project. 2. The consultant triggers 'testing complete'.	

3. FOR each of the testers 3.1. The system renders previous ratings on a user 3.2. The system asks the user to provide a rating. 3.3 The user provides a rating 4. IF user has a rating 4.1. New rating = Average (new rating, initial rating) 5. ELSE 5.1 Rating = new rating. 6. The system records new rating in the tester's record.
Postconditions: A rating is provided for a tester by a consultant. A new rating average is calculated.
Alternative flow: DBConnectionError

2.4.18 Record submitted payment to database

Use case ID: **09**	Use case: recordPayment
Brief description: The system records payment transaction carried out by online payment processor for payments made to testers for testing once a session is complete and for companies to pay for survey.	
Primary Actor: PaymentProcessor	
Secondary Actor: SMTP	
Preconditions: 1. UC to starts at extend(pickTester) -> extend(makePayment), OR 2. UC to starts at extend(verifyAttendance) -> extend(makePayment)	
Main flow: 3. If UC starts at extend(pickTester) (6) 1.4. The system requests transaction ID from payment processor and stores it under a payment record 1.5. The system creates a 'Survey History' record and stores meta data on transaction 4. IF UC starts at extend(verifyAttendance) (5) 4.1. The system requests transaction ID from payment processor and stores it under a payment record 4.2. The system creates a 'Booking History' record and stores meta data on transaction 5. The system provides visual cues that the payment recording is complete 6. The system requests SMTP to send the planner that a new payment has been made. 7. The system requests SMTP to send the company/ consultant that a new payment has been made.	
Postconditions: 1. Payment transaction is recorded in the database 2. Email is sent to planners and testers that the transaction is successful, and payment is made.	
Alternative flow: DBConnectionError	

2.4.19 Access testing schedule

Use case ID: **12**	Use case: accessTestingSchedule
Brief description: The user can trigger an access testing schedule button which searches all the bookings and surveys due and generates a calendar for when testing is due.	
Primary Actor: Planner	
Secondary Actor: GraphQL	
Preconditions: 1. Projects must be scheduled 2. User must be logged onto a planner account	
Main flow: 1. The planner triggers access testing schedule button. 2. The system searches all records of bookings and surveys with status 'booked'. 3. The system parses the records, looking for references to date and time and project description. 4. The data and time value and description are used to form an event and events are assigned to each day slot. 5. Day slots are sent to an external API or internal process (undecided by AN) to generate a calendar. 6. The user can access the calendar and click on events to view more description. 7. The system requests information from the database regarding that triggered item. 8. The system generates a modal or card with more description.	
Postconditions: 1. The planner accesses a calendar which is populated by upcoming testing sessions.	
Alternative flow: mapException	

2.4.20 Setup reminder

Use case ID: **16**	Use case: setupReminder
Brief description: The system sends an email message/ notification to the planner when a survey or testing project request has been submitted by users.	
Primary Actor: Company, Tester	
Secondary Actor: SMTP	
Preconditions: 1. A user must apply for becoming a tester or a company must submit a survey request.	
Main flow: 1. The use case starts when the user submits tester request or company submits a survey request. 2. The system finds all the users in the database with role planner. 3. The system sends each of the planners an email to notify that a new request has been placed. 4. IF a planner has a session active 4.1 A visual cue and message are provided that a new request has been made. 4.2 The message contains a link which redirects the user to the request record.	

Postconditions: Planner is made aware of any new requests made by testers or companies through email reminders.
Alternative flow: DBConnectionError

2.4.21 Add description

Use case ID: **11**	Use case: addDescription
Brief description: Planner can add and change change descriptions of projects.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions: 1. User must be logged onto a planner account 2. A project must be created	
Main flow: 1. The use case starts at Include(addDescription) on createProject (2) The system asks the planner to specify a description project details. 2. The Planner specifies client name and project number and number of testers required.	
Postconditions: A project record's is assigned a description for the first time, or has existing description modified.	
Alternative flow: SessionTimeout, Cancel	

2.4.22 Remove consultant from project

Use case ID: **14**	Use case: removeConsultant
Brief description: The planner can remove consultants from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. This prevents incorrect entries recorded in the database.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions: 1. A project record must be created by a planner 2. Project record must have at least one consultant assigned to it	
Main flow: 1. The UC starts on extend(removeConsultant) of createProject (7) 2. The system renders a remove consultant button next to each consultant added. 3. The user triggers the button 4. The consultant is removed from the project record 5. The system provides visual cues that the removing of consultant has been successful.	

Postconditions: 1. The selected consultant is removed from a project record.
Alternative flow: DBConnectionError, noActorFound

2.4.23 Remove tester from project

Use case ID: **21**	Use case: removeTesterProject
Brief description: The planner can remove testers from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. This prevents incorrect entries recorded in the database.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions: 1. A project record must be created by a planner 2. Project record must have at least one tester assigned to it	
Main flow: 1. The UC starts on extend(removeTesterProject) of createProject (7) 2. The system renders a remove tester button next to each tester added. 3. The user triggers the button 4. The tester is removed from the project record 5. The system provides visual cues that the removing of tester has been successful.	
Postconditions: 1. The selected tester is removed from a project record.	
Alternative flow: DBConnectionError, noActorFound	

2.4.24 Add tester to project

Use case ID: **13**	Use case: addTester
Brief description: The planner can add testers from a testing project record. This provides flexibility to the project as changes to testers can be made at any time. Without this feature, a project cannot be started as you need testers to manage a project.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions: 1. A project record must be created by a planner 2. Project record must not have more than 5 testers assigned to it	
Main flow: 1. The UC starts on extend(addTester) of createProject (8) 2. The system renders a add testers button next an add consultant input field to 'Add more' testers. 3. WHILE testers in project <= 5 3.1. The user triggers the button, triggering findTester	

3.2. The system alerts the user to input staff name or id. 3.3. The user inputs tester name or id. 3.4. The system finds the staff 3.5. IF testers in project >= 5 3.5.1. The system should not render 'add more tester button. 4. The tester is added to the project record 5. The system provides visual cues that the adding of tester has been successful.
Postconditions: 1. A consultant is added to a project record.
Alternative flow: DBConnectionError, noActorFound

2.4.25 Generate OTAC web page for testing

Use case ID:	Use case: generateOTAC
Brief description: An OTAC webpage is generated which allows testers to submit their responses to surveys. The URL is sent to them through SMTP.	
Primary Actor: Planner	
Secondary Actor: SMTP	
Preconditions: 1. The planner must have created a project (survey or testing). 2. The planner finds testers and adds them onto a project. 3. Include(addTester) The system renders a button next to each tester which the planner can trigger to add a tester to a project. 4. The user triggers buttons and add testers to the project or survey.	
Main flow: 1. The UC starts on include(addTester) on findTester() (7) 2. An OTAC webpage token is generated unique to a project 3. The system populates each webpage with questions requested by companies. 4. The URL for the webpage is sent to the testers via SMTP	
Postconditions: 1. An OTAC URL is generated for testers 2. The URL is sent to testers	
Alternative flow: DBConnectionError	

2.4.26 Add consultant to project

Use case ID: **13**	Use case: addConsultant
Brief description: The planner can add consultants from a testing project record. This provides flexibility to the project as changes to consultants can be made at any time. Without this feature, a project cannot be started as you need consultants to manage a project.	
Primary Actor: Planner	
Secondary Actor:	

Preconditions: 1. A project record must be created by a planner 2. Project record must not have more than 3 consultants assigned to it
Main flow: 1. The UC starts on extend(addConsultant) of createProject (8) 2. The system renders a add consultant button next an add consultant input field to 'Add more' consultants. 3. The user triggers the button, triggering findTester implementation for staff members. 4. The system alerts the user to input staff name or id. 5. The user inputs staff name or id. 6. The system finds the staff 7. IF consultants in project >= 3 8.1. The system should not render 'add more consultant' button. 8. The consultant is added to the project record 9. The system provides visual cues that the adding of consultant has been successful.
Postconditions: 1. A consultant is added to a project record.
Alternative flow: noActorFound

2.4.27 Send results

Use case ID:	Use case: sendSurveyResults
Brief description: The planner sends survey results to the company email address. The status code for the survey is set to completed.	
Primary Actor: Planner	
Secondary Actor: SMTP	
Preconditions: 1. The survey must be completed by testers	
Main flow: 1. The UC starts after extend(sendSurveyResults) on accessSurvey (6). 2. The planner triggers 'send results' button. 3. The system sends email to company using SMTP. The email contains initial question and responses sent by users.	
Postconditions: 1. Survey response is complied and sent to company.	
Alternative flow: emailException	

2.4.28 Update tester information

Use case ID:	Use case: updateDetails
Brief description: Information confirmed through phone call or email, depends on user's preference. Such information can be updated by the planner.	
Primary Actor: Planner	

Secondary Actor:
Preconditions: 1. The user is in a phone call with a user 2. The user received an email from a user regarding more information
Main flow: 1. The system provides an interface to update information. 2. The system updates the information.
Postconditions: 1. The system records the missing information 2. Extend(confirmRequest)
Alternative flow: DBConnectionError, noActorFound

2.4.29 Email request missing details

Use case ID: **27**	Use case: emailRequestMissingDetails
Brief description: The planner triggers a button to send a document via email that aims to request missing details regarding user information.	
Primary Actor: Planner	
Secondary Actor: SMTP	
Preconditions: 1. Planner must be logged onto a planner account 2. A user must have sent an application to become a tester with insufficient detail	
Main flow: 1. The system generates a button to send document via email which explains more information is needed. 2. The planner triggers a button to send the email 3. SMTP request is processed, and email is sent to the user about more information.	
Postconditions: 1. SMTP request is processed, and email is sent to the user about more information.	
Alternative flow: emailException, DBConnectionError	

2.4.30 Phone request missing details

Use case ID: **28**	Use case: requestMissingDetails
Brief description: The planner phone calls a user a fill in missing or undetailed information regarding a user.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions:	

1. The UC starts on accessRequest (1) where user can access information about a user.
Main flow: 1. The planner calls the user and asks about missing information. 2. The system provides an interface to update information. 3. The system updates the information.
Postconditions: 3. Planner calls a user and fills missing information into a form. 4. The system records the missing information 5. Extend(confirmRequest)
Alternative flow: DBConnectionError

2.4.31 Confirm request

Use case ID:	Use case: confirmRequest
Brief description: The planner can confirm requests for users to become a tester. This is if user is suitable for the role.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions: 1. A user must have sent an application to become a tester in the form. 2. Planner must be logged into their account to view requests.	
Main flow: 1. The planner triggers confirm request button on the ‘requests page’ 2. The system asks the planner if the user should accept the user as a tester. 3. The planner confirms request 4. The temporary record is deleted, and user information is stored on the persistent database. 5. Email is sent to the user that their application has been successful.	
Postconditions: 1. Tester confirms request for a user to become a tester 2. Email is sent to the user that their application has been successful. 3. User account’s role is set to ‘Tester’.	
Alternative flow: DBConnectionError, noActorFound	

2.4.32 Delete requests

Use case ID:	Use case: deleteRequest
Brief description: The planner can delete requests for users to become a tester. This is if a user is not suitable for the role.	
Primary Actor: Planner	
Secondary Actor:	
Preconditions:	

1. A user must have sent an application to become a tester in the form. 2. Planner must be logged into their account to view requests.
Main flow: 1. The planner triggers delete request button on the 'requests page' 2. The temporary record is deleted. 3. The system provides visual cues that request has been deleted. 4. The system makes a record that person with their name had applied before. 5. The system requests SMTP to send email to the user that their application has been rejected.
Postconditions: 1. Tester deletes request for a user to become a tester. 2. Email is sent to the user that their application is rejected.
Alternative flow: noActorFound, DBConnectionError

2.5 Use case description – alternative flows

2.5.1 Alternative flow: DBConnectionError

Brief description: The system is unable to make a connection with the database.
Preconditions: The Admin user attempted to create a new user account with any role.
Alternative flow: 1. The alternative flow starts after flow 4.0 of the main flow. 2. The system displays a visual cue to indicate that the operation has failed 3. The actor can click on a button again to reattempt establishing a database connection and store new user data.
Postcondition: None

2.5.2 Alternative flow: InvalidInputLogin

Brief description: The actor enters invalid username or password.
Preconditions: The actor entered invalid username or password.
Alternative flow: The alternative flow starts after 3.0. 1) A message is displayed indicating that the details are invalid 2) The actor can click the appropriate button to dismiss the message.

Postcondition: User has to attempt to log in again

2.5.3 Alternative flow: SessionTimeout

Preconditions: Planner's login session has timed out.
Brief description: Security measures such as session timeout will be implemented. The planner will be logged out due to idling for too long.
Alternative flow: The flow starts at extension findTester (4) for example. 1) The user cannot create a new project as user.session = false 2) The system redirects the user to the root page to log in again.
Postcondition: User has to log in again to use website functionality.

2.5.4 Alternative flow: MapException

Preconditions: User requests MapsAPI for a map given some parameters.
Brief description: MapsAPI may be unavailable due to API downtime externally.
Alternative flow: 1) The MapsAPI returns a failure message 2) The system renders cannot render map, 'try again later' message.
Postcondition: The MapsAPI returns a failure message The system renders cannot render map, 'try again later' message.

2.5.5 Alternative flow: Cancel

Preconditions: The system is operation and the user triggers cancel button.
Brief description: User cancels flow by triggering cancel button
Alternative flow: 1) The flow may start at any step of the main flow. 2) The user aborts the form 3) The system halts the form submission from being recorded.
Postcondition: The user aborts the form The system halts the form submission from being recorded A different journey is rendered

2.5.6 Alternative flow: NoActorFound

Preconditions: The system is operation and the user searches for an actor using 'findTester'.

Brief description: No tester/ staff is found as the record does not exist.
Alternative flow: 1) The flow may start at any step of the main flow. 2) The user searches for a record of a user. 3) The system searches the database for an actor, but actor is not found 4) The system renders a visual cue saying actor is not found.
Postcondition: The system searches the database for an actor, but actor is not found The system renders a visual cue saying actor is not found.

2.5.7 Alternative flow: invalidPayment

Preconditions: The system is operation and the user attempt a payment request through payment processor.
Brief description: A payment request is sent but either the payment details are rejected, or card is declined.
Alternative flow: 1) The flow may start at any step of the main flow. 2) The user attempts to make a payment request. 3) The system rejects payment request 4) The system renders a visual cue saying payment was rejected. 5) The system redirects the user to attempt payment again.
Postcondition: The system renders a visual cue saying payment was rejected. The system redirects the user to attempt payment again.

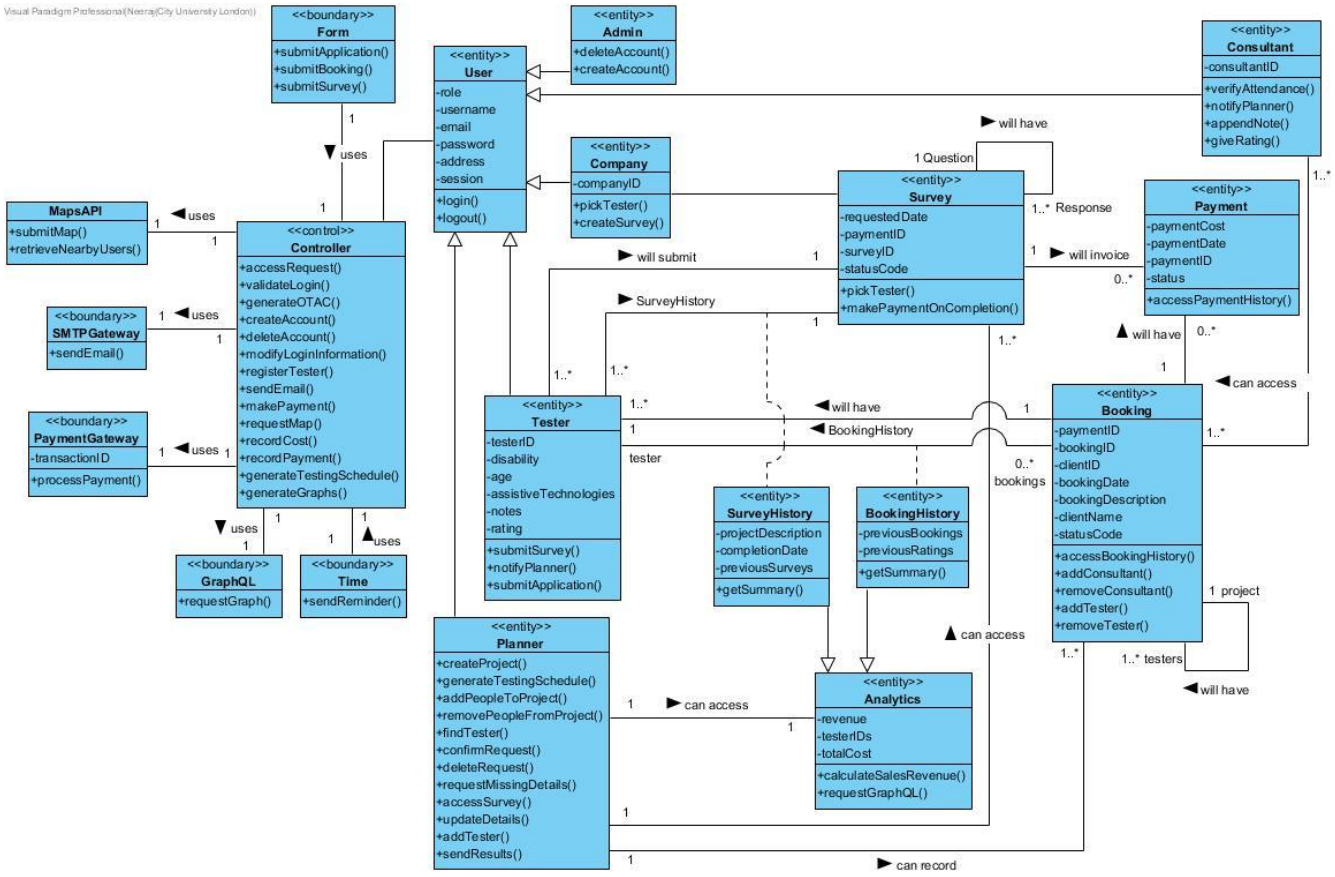
2.5.8 Alternative flow: emailException

Preconditions: User requests email SMTP for a send request given some parameters.
Brief description: Email SMTP may be unavailable due to downtime externally or invalid input into input headers.
Alternative flow: 1) The SMTP returns error message while sending an email. 2) The system renders failure message to the user. 3) The system asks the user if they want to attempt to send the email again.
Postcondition: The system renders failure message to the user. The system asks the user if they want to attempt to send the email again.

3. Class Diagram

Packages are not used as the analysis class diagram is quite simple and abstracting it may make it difficult to understand the bigger picture.

Packages will be used for the design class diagram, as implementation details will saturate the diagrams causing it to not fit in one page.



4. Design

Note: Version control is updated on the start of the document.

4.1 Introduction

4.2 Aims and objectives and summary

The aim of design is to create add implementation details on the problem domain entities to make transition to implementation easier. The content below shows the changes made to the working document after requirements engineering. Overall, not many functionalities from problem domain have been completely changed but mostly further developed with the exception being analytics use cases which were remodeled. Overall the design document models the system very well- from the static system to dynamic data interacting with external systems.

4.3 What has changed. How has it changed. Why it has changed.

N.B. Key functionality was not change so use case specifications are not changed, only the FindUser() specification has changed and new cases such as "showOngoingProjects" and "showTesterActivity" has been added.

The following below is a revised is a revised version of the use-case model. This relates to changes made from the initial use case modal due to disagreement with client on specific use case and to simplify the model.

The use case *accessTestingSchedule* initially was set to only be used by Planners. This use case now has been moved so that the Staff actor uses it, which means both consultants and planners have access to it. **See Figure 5**. This will ensure that all staff members can access testing schedules that is taking place and will take place in the future. The stakeholder *Raphael* had requested this change to be done after seeing the specification to let consultants see the future projects and plan.

Update: The use case accessTestingSchedule has been removed as AbilityNet already uses an internal system to manage scheduling, making this a redundant feature.

The use cases *emailRequestEmailDetails* and *phoneRequestEmailDetails* were generalized under *RequestMissingDetails*. This use case now has been modified by removing the generalization as it is not needed. **See Figure 7**. The use case *phoneRequestEmailDetails* does not store any data or have data interactions so there is no point of it being a use case.

Moreover, there were no mechanisms to record the emails once they were sent. **See Figure 6**. Now, a new use case *recordEmail* has been added to record the emails which have been sent to testers or companies. This was done on the request of *Louise* to keep track of emails sent.

Use cases from analytics package *viewTester* summary and *calculateSalesSummary* have been defined further to *showRevenueOverTime* – to show the revenue being generated by the testers, *showOngoingProjects* to show the state of all ongoing projects (booked, cancelled, on-going, awaiting, complete) and *showTesterActivity* to see which testers are most actively being used. GraphQL actor is also added on the right to represent the communication with external service. This would promote other testers to be used in more projects but will also contribute by showing which tests are more reliable derived from their frequency of bookings. This was finalized by *Louise* after a meeting as these graphs would provide the most helpful information.

4.4 History of Document

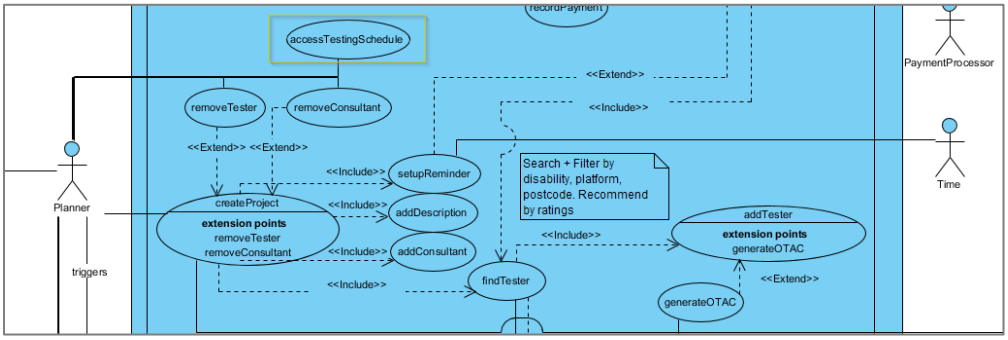


Figure 4 – Before: accessTestingSchedule could only be executed by Planners.

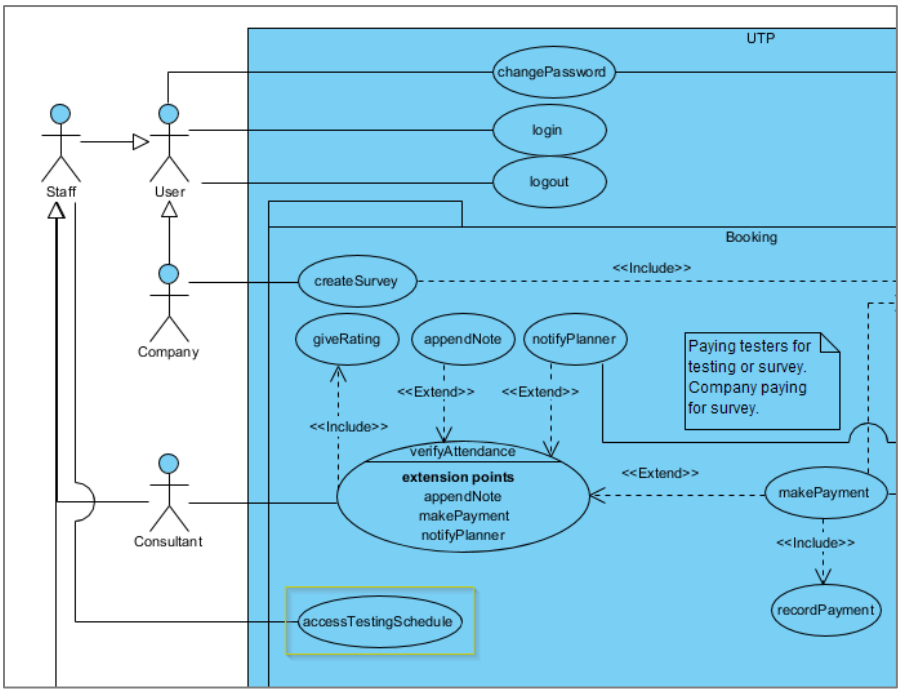


Figure 5 – After: accessTestingSchedule can be accessed by all staff members.

Note: accessTestingSchedule has been removed now as requested by Louise and planning team. See 4.3 for reference.

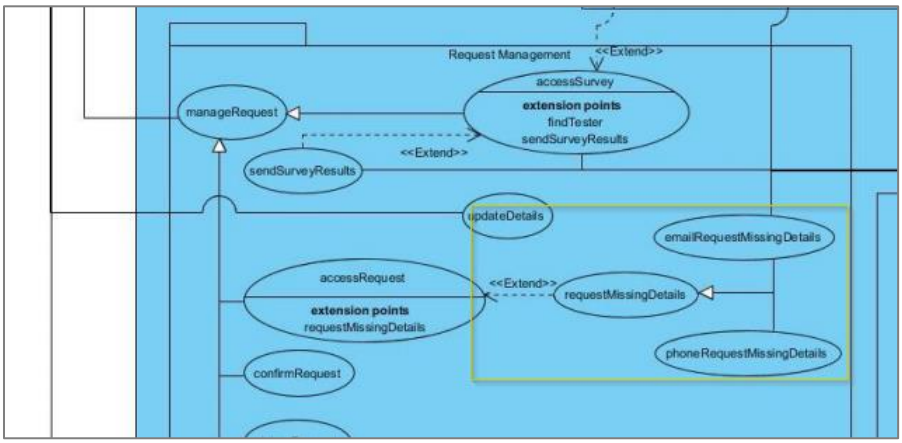


Figure 6 – Before: Use case generalized

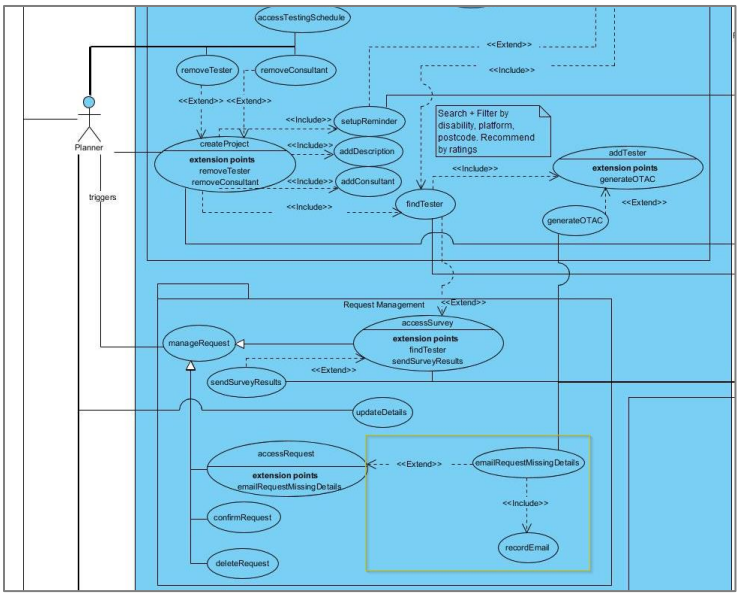


Figure 7 – After: Simplified use case. Added recordEmail use case.

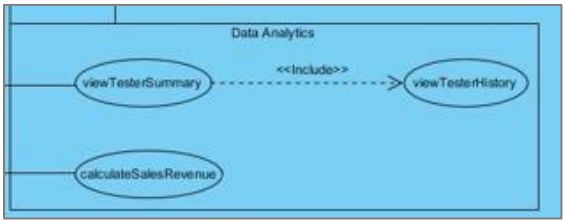


Figure 8 – Before: Use cases such as testerSummary not detailed.

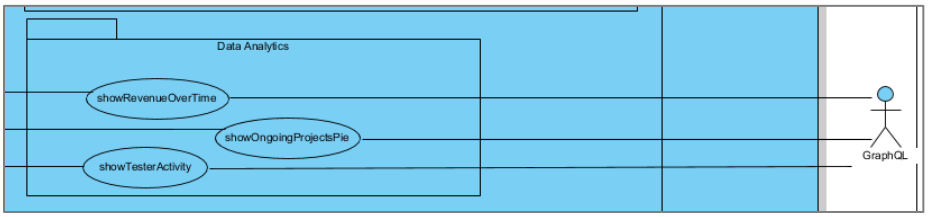


Figure 9 – After: Defined use cases more.

The use of findTester() only provides mechanism to search for testers and not other users such as consultants for a project which is important in bookings.

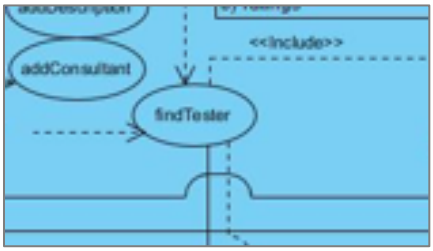


Figure 10 – Before: findTester() used .

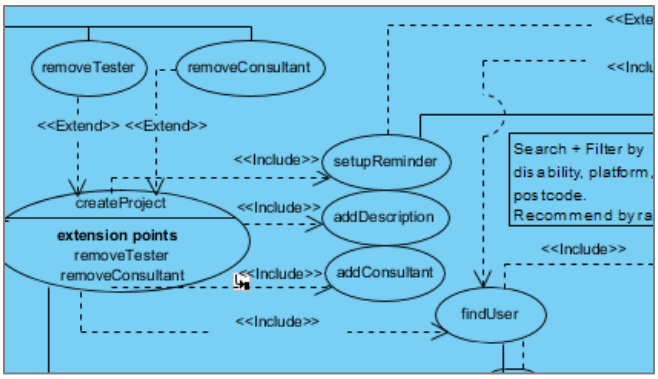


Figure 11 – After : findUser() used

4.4.1 Changes made to use case specifications

Use case ID: **17**	Use case: **findUser**
Brief description: Planners should be able to find users from the database given parameters such as username, userID or disability (if tester).	
Primary Actor: Planner	
Secondary Actor: MapsAPI	
Preconditions: Journey 1: A user logged onto a company account should create a survey (createSurvey) Journey 2: A user logged onto a planner account created a project (createProject)	
Main flow: 1. The system alerts user to enter search query into input field. 2. IF The user enters search query of type Consultant. (UserID =””). 2.1. The system displays a list of consultants who are available for testing. 2.2. The user picks a user for testing 2.3. The system stores the user onto the project record as consultant. 3. IF The user enters a search query of type tester. 3.1. The system renders a list of tests which are neither sorted nor filtered. 3.2. Filter = [name, id, rating, assistive technology, age, postcode, etc.] 3.3. The system asks the planner to sort or filter testers by their meta data such as disabilities, assistive technology competency, postcode, age, etc. 3.4. The planner selects filters using checkboxes and inputs data within forms. 3.5. IF filter is ‘name/id/assistive technology competency’ 3.5.1. The system will render all tester items with tester.filter == filter 3.5.2. IF filter is ‘ratingThe system will sort the tester ratings by descending value. 3.5.2.1. The system renders the list of sorted testers. 3.5.2.2. The planner may trigger the button to sort by ascending value. 3.5.2.2.1. IF filtering is ‘age’. 3.5.2.2.2. The system renders a range input field to select lower and upper bounds. 3.5.2.2.3. The planner inputs the range of age. 3.5.2.2.4. The system renders the list of testers who pass the age requirement from the database. 3.5.2.2.5. IF filter is ‘postcode’ 3.5.2.2.5.1. The system will communicate with MapsAPI and generate a placeholder local map. 3.5.2.2.5.2. The system will generate a form where planners can input postcode filters.	

3.5.2.2.5.3. The planner inputs postcode filtersThe system will send parameters from the form to the MapsAPI and persistent storage and request map which shows nearby testers. 3.5.2.2.5.4. The system will process the response and render a data summary table, along with the map and recommendations for testers. 3.5.2.2.5.5. The planner can further change form values and fine tune the data so that the ideal tester is found. 4. Include(addTester) The system renders a button next to each tester which the planner can trigger to add a tester to a project. 5. The user triggers buttons and add testers to the project or survey.
Postconditions: User picks testers and populates them into a project or survey.
Alternative flow: DBConnectionError, MapException, noActorFound

Use case ID: **34**	Use case: **showOngoingProjects**
Brief description: This use case aims to visualize on going projects using GraphQL external service by passing in values relating to Survey and Booking projects 's session Code as a pie chart to visualize what on going projects are in what state.	
Primary Actor: Planner	
Secondary Actor: MapsAPI	
Preconditions: Survey and Booking records must be populated with data relating to statusCode such as 'Booked'… 'Completed'… etc.	
Main flow: 1. The system generates a pie chart frame on page load. 2. The system finds all records of Survey and Bookings 3. For each of the records 3.1. Map each occurrence of a statusCode variable to a counter and status text. 4. The system requests GraphQL to generate a pie chart using the mapped data. 5. The system receives the graph instance from GraphQL and is assigned to the frame 6. The planner observes the new graph which is added.	
Postconditions: The planner observes the new graph which is added	
Alternative flow: DBConnectionError, GraphException, noActorFound	

Use case ID: 35	Use case: **showTesterActivity**
Brief description: This use case aims to visualize tester activity projects using GraphQL external service by passing in values relating to Survey and Booking projects. All of the tester's projects they worked on are counted which provides a mapping between tester and projects. A sortable bar graph will display the users with highest and lowest tester activity, which aims to reduce tester bias.	
Primary Actor: Planner	
Secondary Actor: MapsAPI	
Preconditions: Records for testers must exist.	
Main flow: 1. The system generates a pie chart frame on page load. 2. The system finds all records of testers. 3. For each of the records 3.1. Map each occurrence of Project or Survey variable to a counter and Tester. 4. The system requests GraphQL to generate a pie chart using the mapped data. 5. The system receives the graph instance from GraphQL and is assigned to the frame 6. The planner observes the new graph which is added.	
Postconditions: The planner observes the new graph which is added	
Alternative flow: DBConnectionError, GraphException, noActorFound	

4.4.2 Changelog after sequence diagrams

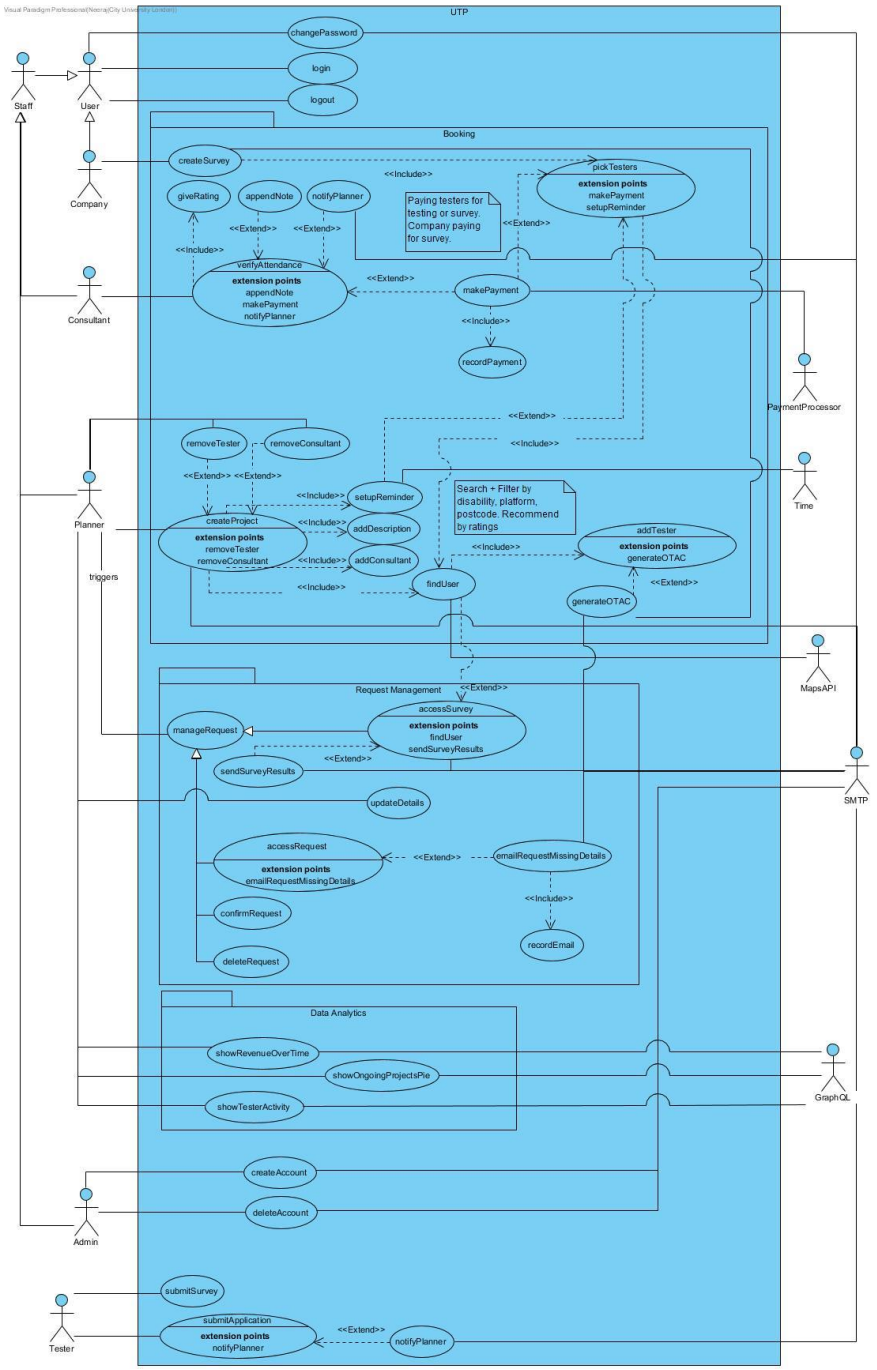
Implementation of sequence diagram introduces new attributes and operations which were initially not included in the diagrams.

1. Analytics added concrete class with two new classes, reports more detail.
2. Analytics for survey removed.
3. Upload button for booking class
4. Record email added as a use case and class diagram
5. age attribute from tester removed. cannot ask due to picking bias. Plus, extra complexity on increasing age
6. address attribute changed to street address... postcode...county, etc.
7. project tester reflexive association removed due to more data on tester needing to be updated.

Changes to class diagram

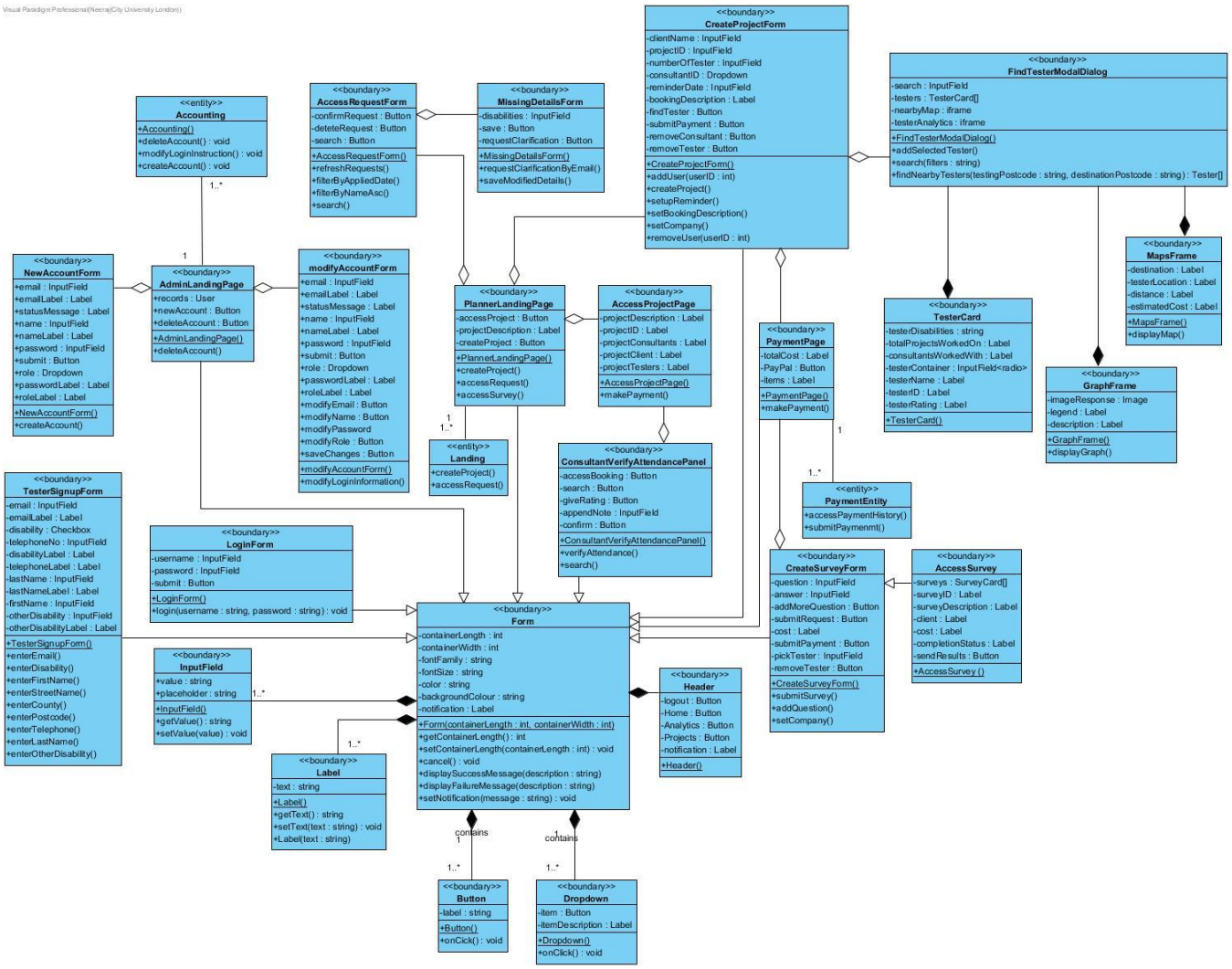
2. verifyLogin() used to verify login
3. getUser() to get user from database
4. displaySuccessMessage, displayFailureMessage to trigger messages to user.
5. addUser(userID : int) to add user to a project in creamProject form and ProjectController
6. setBookingDescription() to add booking description.
7. makeBooking() to make a booking in project controller.
9. setCompany() to set company in projectcontroller and project form
10. changed findTester to findUser for use case spec.
11. added decRemainingTester() to keep track of how many testers can be added more to a project
12. booking.status added
13. worked in what industry added for tester
14. Removed decRemainingTester() as length of testers array could be used to communicate same information
15. Added verified : Boolean in Tester entity to capture instances of testers in bookings if they are verified.
In practically, HashMap or any map could be used.

4.4.3 Revised use case model



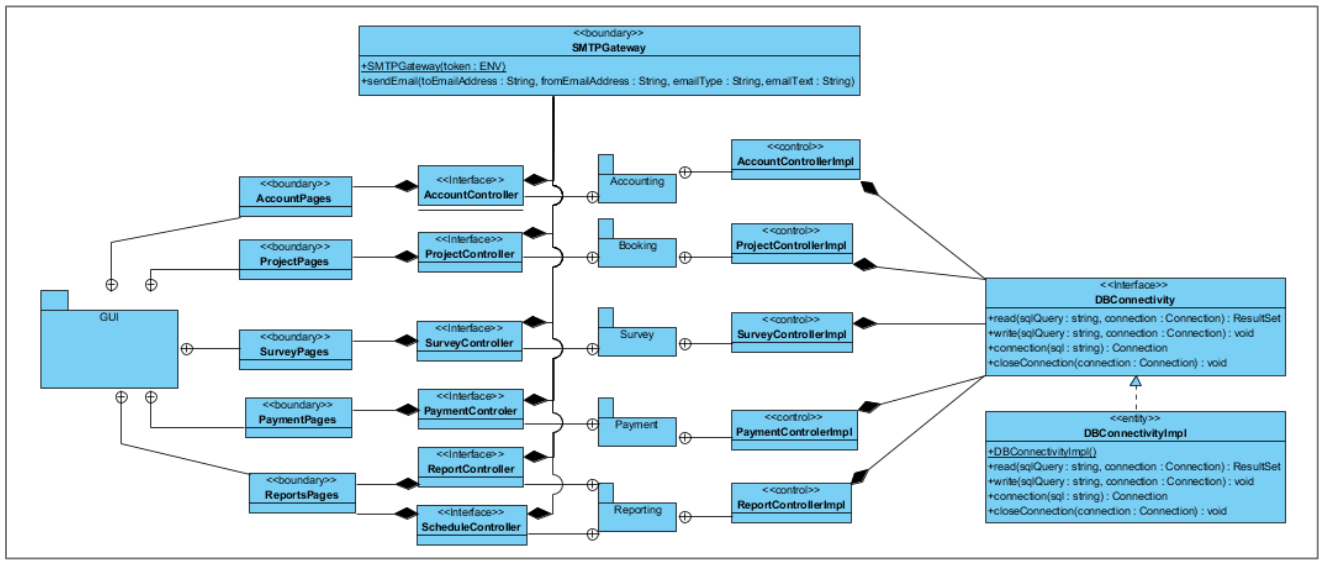
4.5 Design class diagram: GUI Diagram

GUI diagram gives overview of different pages within UTP system users can access and what form elements they are composed of.

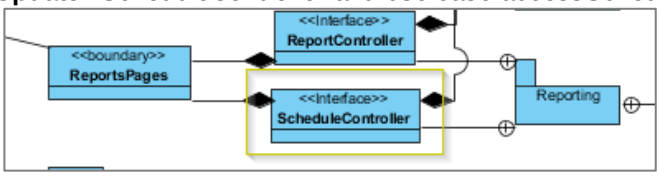


4.6 Design class package

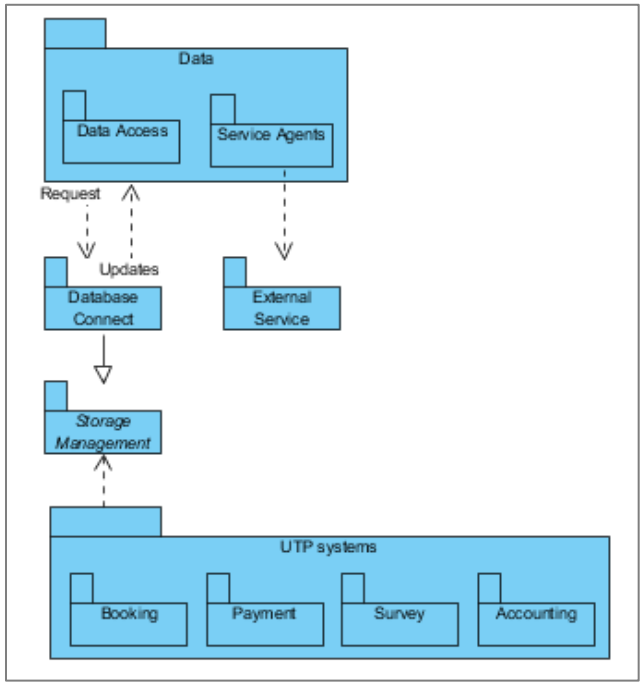
This diagram shows the abstract configuration of the AbilityNet User Testing Processes system. It shows where how data may be entered, processed and outputted to the user in a system and how it may be stored in a database. **See DBConnectivity.**



Update: ScheduleController and use case accessSchedule has been removed. See 4.3 for reference.

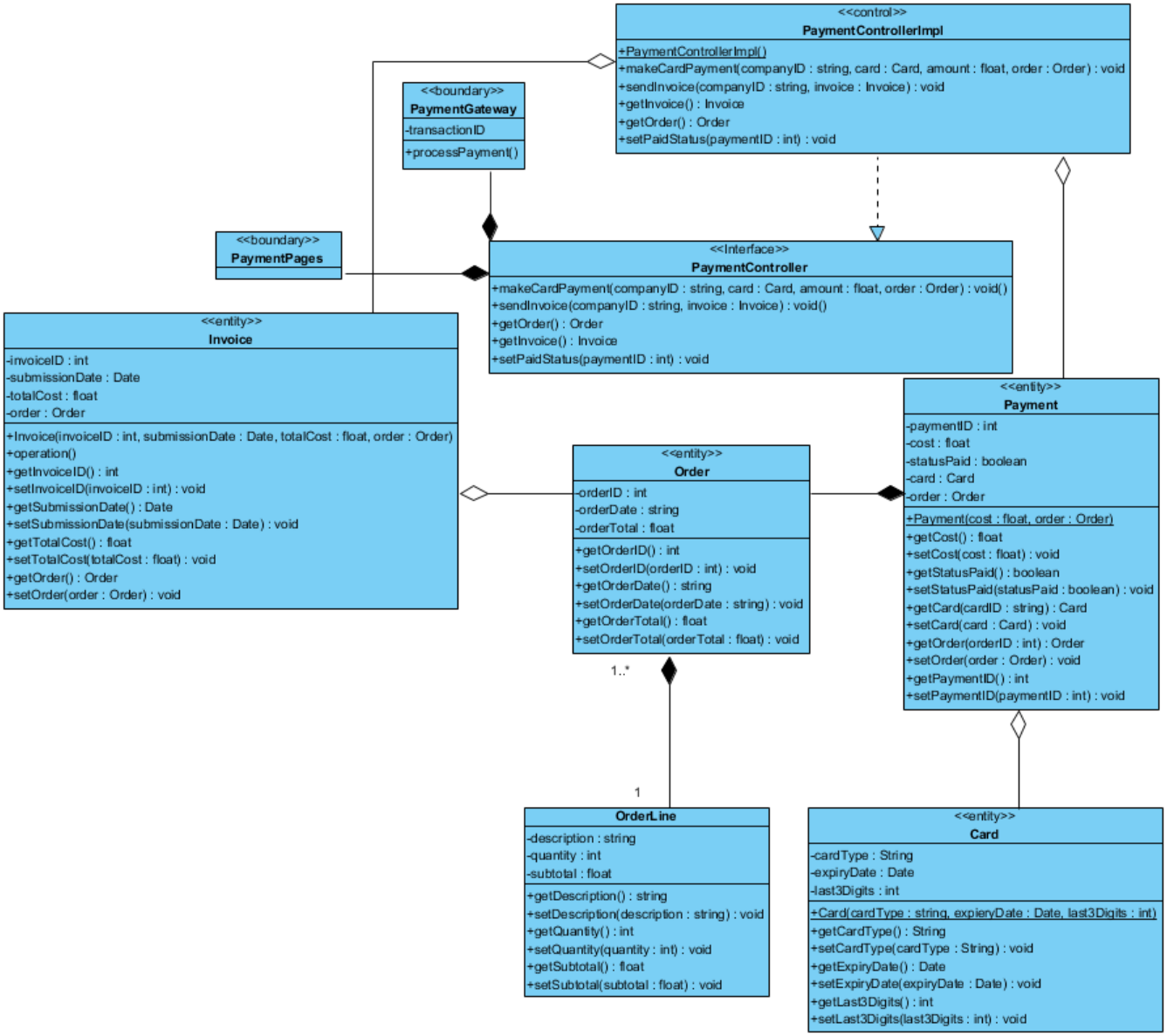


High level overview of system

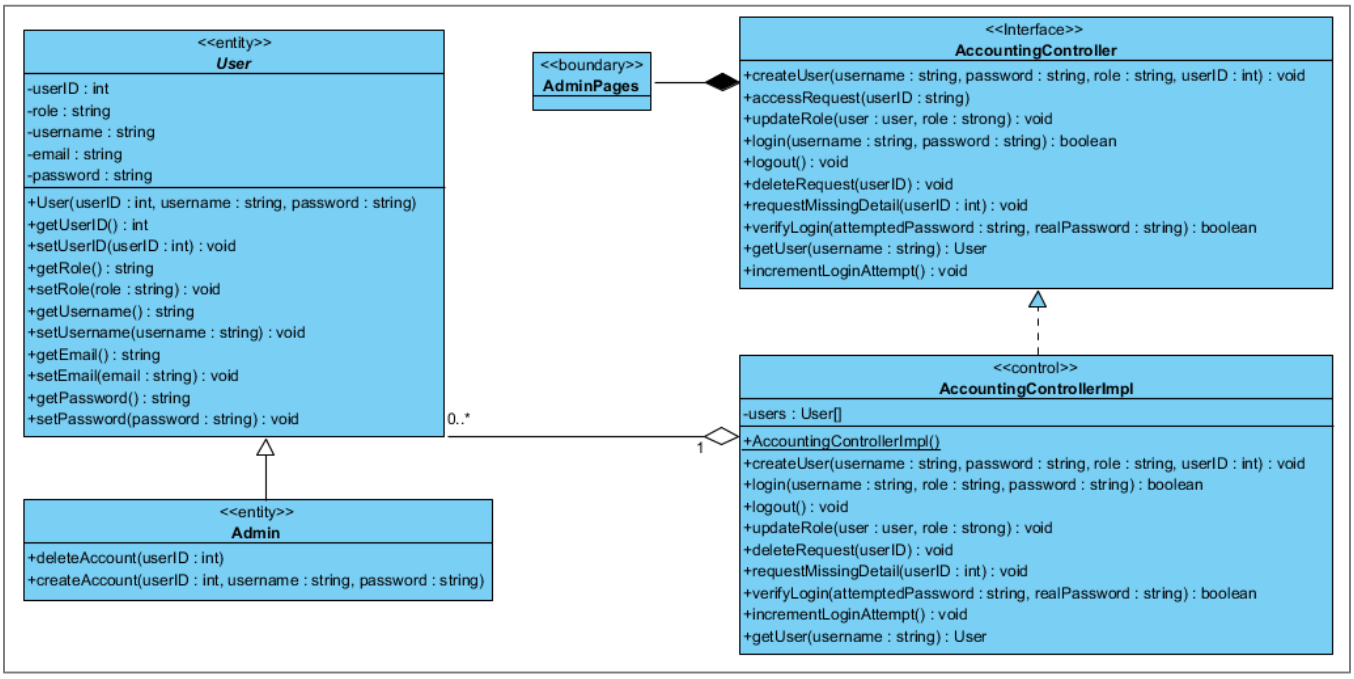


4.6.1 Booking package

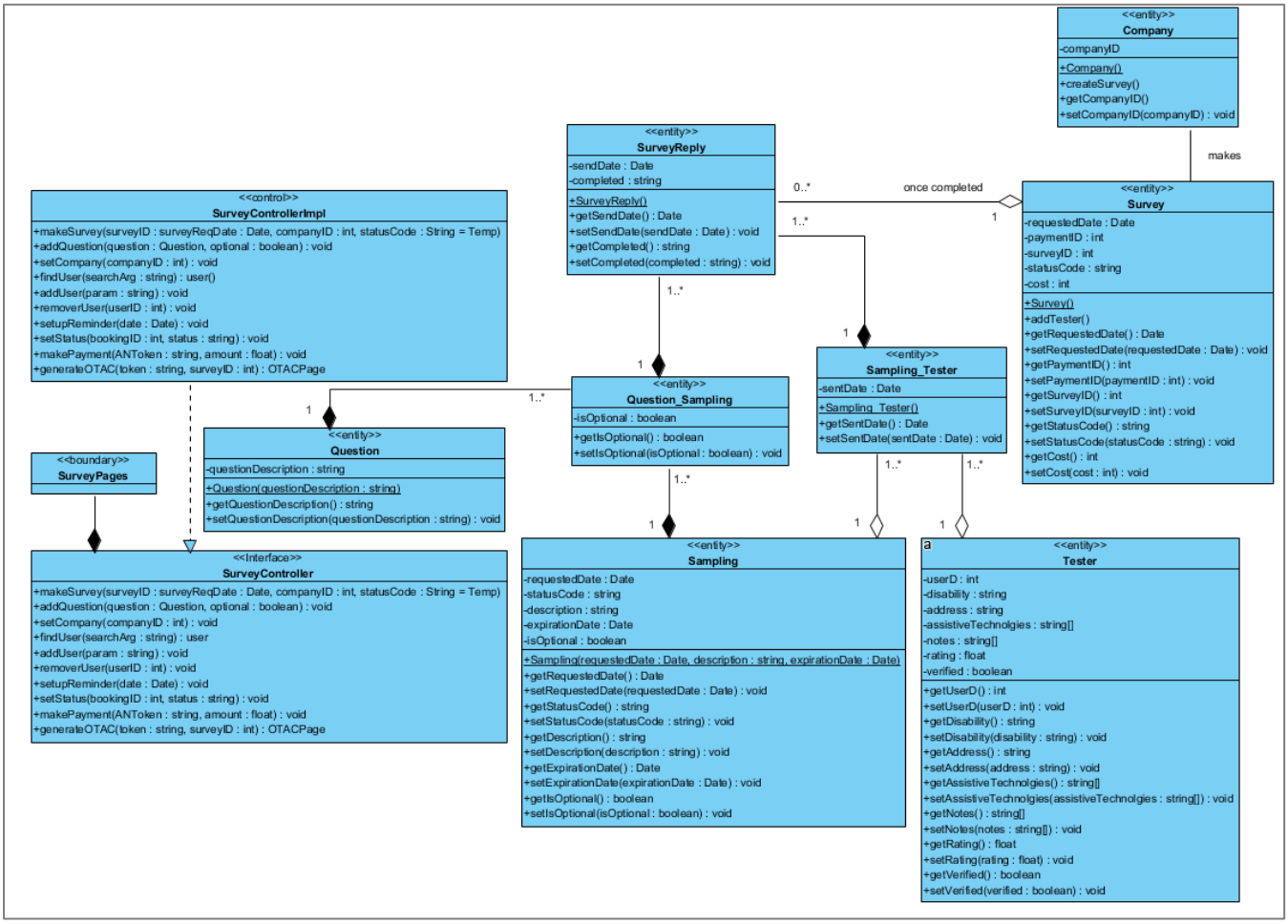
4.6.2 Payment package



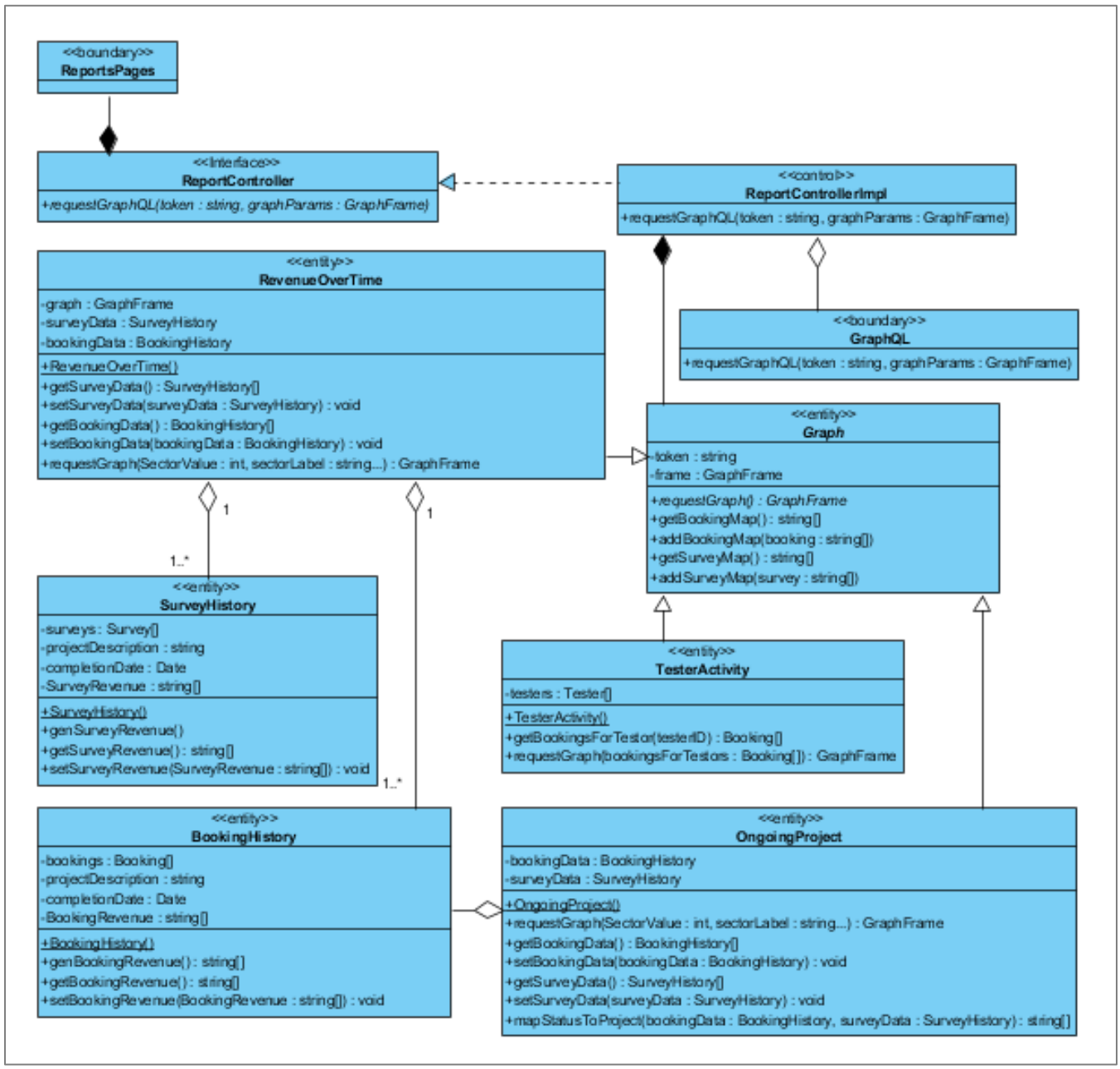
4.6.3 Accounting package



4.6.4 Survey package



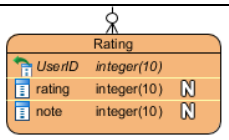
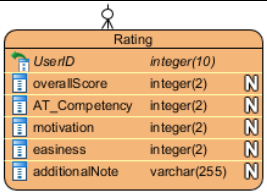
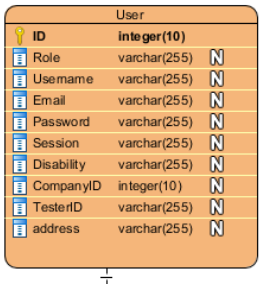
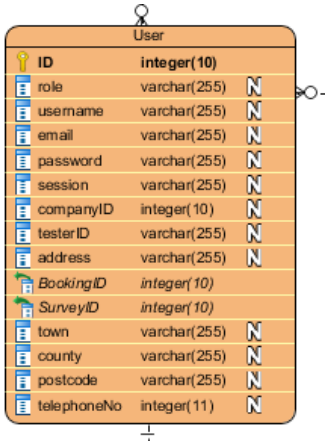
4.6.5 Reporting package



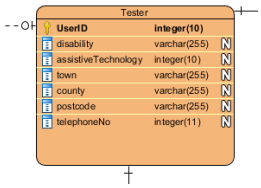
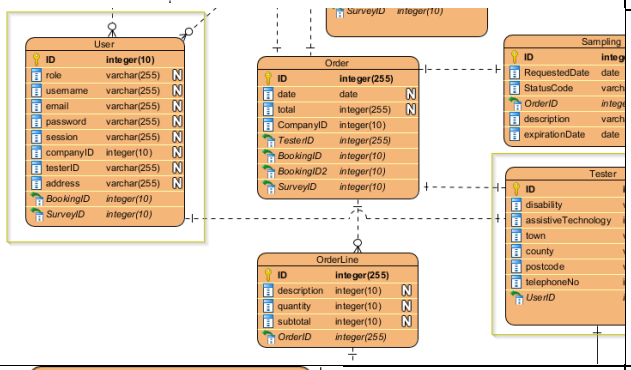
4.7 ERD

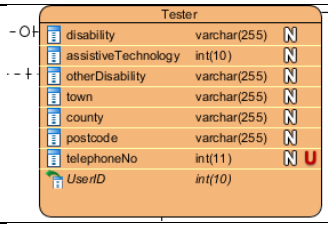
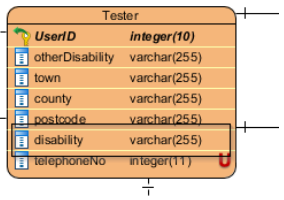
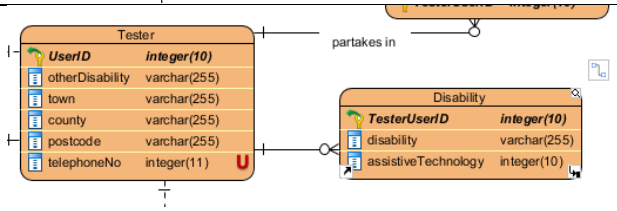
4.7.1 Changelogs for normalization (3rd normal form)

First normal form errors encountered and corrections. Corrections to design class diagram was made after this.

<table>
<tr><th>Initial</th><th>After</th><th>Reason</th></tr>
<tr><td> 
</td><td> 
</td><td>Rating column is too general. It may be perceived quantitively or qualitatively. It may be broken down into attributes of what is contributing to that rating.</td></tr>
<tr><td> 
</td><td> 

Update: The address information has been moved to Tester as Tester inherits User, and the address fields are only relevant for testers.</td><td>Address column is too general and can be broken down into street address, city, county and postcode.

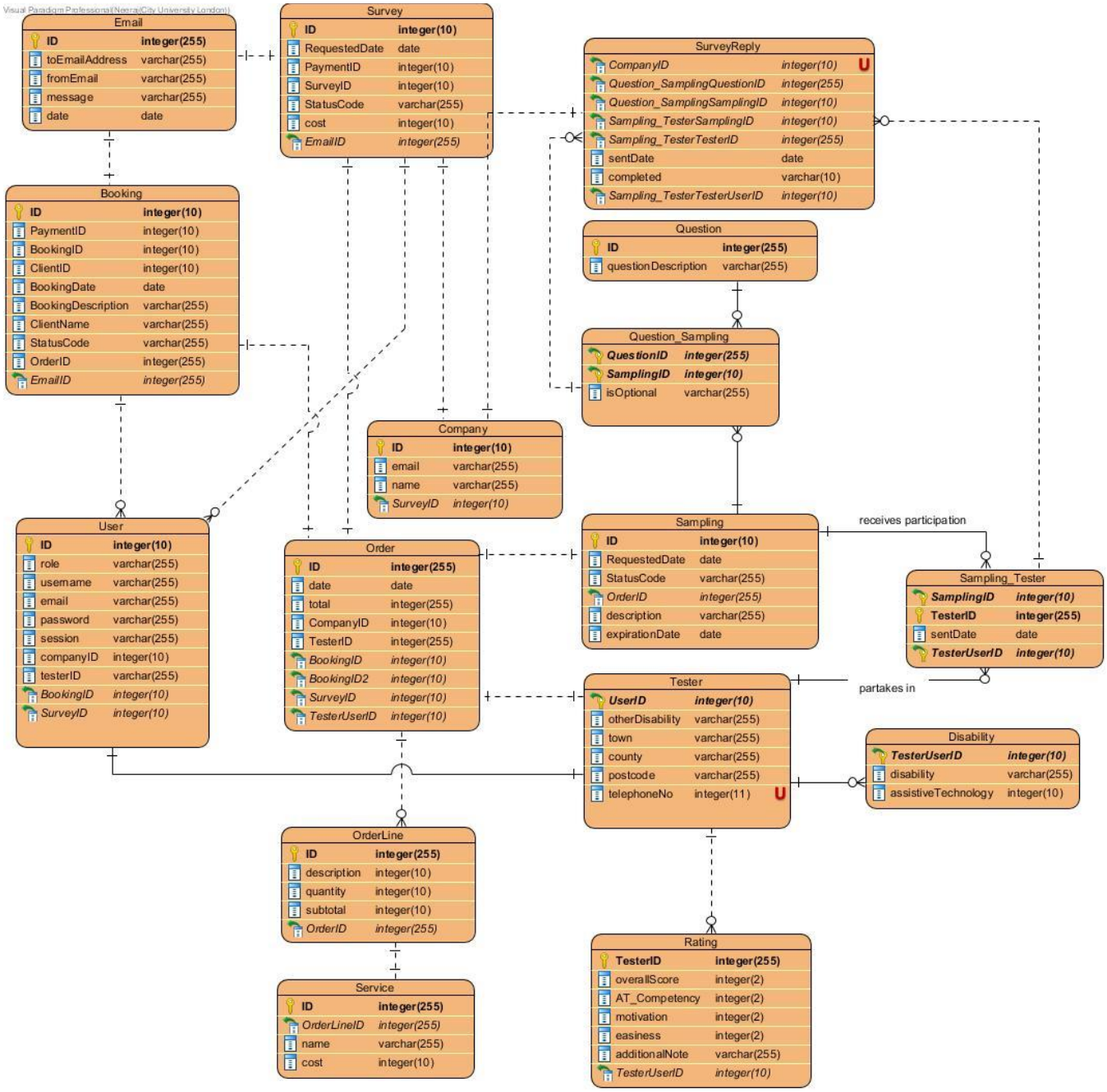
Disability column applies to tester users specifically.</td></tr>
<tr><td> 
</td><td> 
</td><td>3rd normal form problem.

UserID was used as an indirect mapping within testerID to identify a certain user. Foreign key userID with one to one associated is used now with unique TesterID primary key.</td></tr>
<tr><td>TelephoneNo is not set as a contraint.</td><td> 
</td><td>TelephoneNo is set as a constraint now. telephoneNo with UserID can act as a composite key for searching.</td></tr>
<tr><td> 
</td><td> 
</td><td>Disability record has been made into an own table as there many be many disabilities a tester may have.</td></tr>
</table>

4.7.2 Diagram

Complex logic for Survey. Other tables are straight forward.

- A Tester may register in 0…* Survey.
- A Survey may be for 0…* Testers.
- A Survey will have 0…* Questions.
- A Question may be in 0…* Surveys.
- A Question may have 0…* Reply (Survey_Reply).
- A Reply must be sent by exactly one Tester for a given Company's Survey request.
- A Reply must be only accessed by the Company.



4.8 ERD SQL

4.8.1 Tables – MySQL Dialect

4.8.1.1 Booking

```
CREATE TABLE Booking (
  ID                int(10) NOT NULL AUTO_INCREMENT,
  PaymentID         int(10),
  BookingID         int(10),
  ClientID          int(10),
  BookingDate       date,
  BookingDescription varchar(255),
  ClientName        varchar(255),
  StatusCode        varchar(255),
  OrderID           int(255) NOT NULL,
  EmailID           int(255) NOT NULL,
  PRIMARY KEY (ID));
```

4.8.1.2 Company

```
CREATE TABLE Company (
  ID       int(10) NOT NULL AUTO_INCREMENT,
  email    varchar(255),
  name     varchar(255),
  SurveyID int(10) NOT NULL,
  PRIMARY KEY (ID));
```

4.8.1.3 Email

```
CREATE TABLE Email (
  ID             int(255) NOT NULL AUTO_INCREMENT,
  toEmailAddress varchar(255),
  fromEmail      varchar(255),
  message        varchar(255),
  dateRecord         date,
  PRIMARY KEY (ID));
```

4.8.1.4 Order

```
CREATE TABLE Order (
  ID         int(255) NOT NULL AUTO_INCREMENT,
  date     date,
  total      int(255),
  CompanyID  int(10) NOT NULL,
  TesterID   int(255) NOT NULL,
  BookingID  int(10) NOT NULL,
  BookingID2 int(10) NOT NULL,
  SurveyID   int(10) NOT NULL,
  PRIMARY KEY (ID));
```

4.8.1.5 Order Line

```
CREATE TABLE OrderLine (
  ID          int(255) NOT NULL AUTO_INCREMENT,
  description int(10),
  quantity    int(10),
  subtotal    int(10),
  OrderID     int(255) NOT NULL,
  PRIMARY KEY (ID));
```

4.8.1.6 Question

```
CREATE TABLE Question (
  ID                  int(255) NOT NULL AUTO_INCREMENT,
  questionDescription varchar(255),
  PRIMARY KEY (ID));
CREATE TABLE Question_Sampling (
  QuestionID int(255) NOT NULL,
  SamplingID int(10) NOT NULL,
  isOptional varchar(255),
  PRIMARY KEY (QuestionID,
  SamplingID));
```

4.8.1.7 Rating

```
CREATE TABLE Rating (
  TesterID       int(255) NOT NULL AUTO_INCREMENT,
  overallScore   int(2),
  AT_Competency  int(2),
  motivation     int(2),
  easiness       int(2),
  additionalNote varchar(255),
  PRIMARY KEY (TesterID));
```

4.8.1.8 Sampling

```
CREATE TABLE Sampling (
  ID             int(10) NOT NULL AUTO_INCREMENT,
  RequestedDate  date,
  StatusCode     varchar(255),
  OrderID        int(255) NOT NULL,
  description    varchar(255),
  expirationDate date,
  PRIMARY KEY (ID));
```

4.8.1.9 Sampling_Tester

```
CREATE TABLE Sampling_Tester (
  SamplingID int(10) NOT NULL,
  TesterID   int(255) NOT NULL,
  sentDate   date,
  PRIMARY KEY (SamplingID,
  TesterID));
```

4.8.1.10 Service

```
CREATE TABLE Service (
  ID          int(255) NOT NULL AUTO_INCREMENT,
  OrderLineID int(255) NOT NULL,
  name        varchar(255),
  cost        int(10),
  PRIMARY KEY (ID));
```

4.8.1.11 Survey

```
92  CREATE TABLE Survey (
93    ID            int(10) NOT NULL AUTO_INCREMENT,
94    RequestedDate date,
95    PaymentID     int(10),
96    SurveyID      int(10),
97    StatusCode    varchar(255),
98    cost          int(10),
99    EmailID       int(255) NOT NULL,
100   PRIMARY KEY (ID));
```

4.8.1.12 SurveyReply

```
102  CREATE TABLE SurveyReply (
103    CompanyID                    int(10) NOT NULL,
104    Question_SamplingQuestionID  int(255) NOT NULL,
105    Question_SamplingSamplingID  int(10) NOT NULL,
106    Sampling_TesterSamplingID    int(10) NOT NULL,
107    Sampling_TesterTesterID      int(255) NOT NULL,
108    sentDate                     date,
109    completed                    varchar(10));
110
```

4.8.1.13 Tester

```
1   CREATE TABLE Tester (
2     disability          varchar(255),
3     assistiveTechnology int(10),
4     otherDisability     varchar(255),
5     town                varchar(255),
6     county              varchar(255),
7     postcode            varchar(255),
8     telephoneNo         int(11),
9     UserID              int(10) NOT NULL,
10    CONSTRAINT telephone
11      UNIQUE (telephoneNo));
12
```

4.8.1.14 User

```
121  CREATE TABLE User (
122    ID        int(10) NOT NULL AUTO_INCREMENT,
123    role      varchar(255),
124    username  varchar(255),
125    email     varchar(255),
126    password  varchar(255),
127    session   varchar(255),
128    companyID int(10),
129    testerID  varchar(255),
130    BookingID int(10) NOT NULL,
131    SurveyID  int(10) NOT NULL,
132    PRIMARY KEY (ID));
133
```

4.8.1.15 Disability

```
15  CREATE TABLE Disability (
16    TesterUserID        integer(10) NOT NULL,
17    disability          varchar(255) NOT NULL,
18    assistiveTechnology integer(10) NOT NULL,
19    PRIMARY KEY (TesterUserID));
20
```

4.8.2 Key specification w/ functional dependencies

```
ALTER TABLE Booking ADD CONSTRAINT FKBooking48070 FOREIGN KEY (EmailID) REFERENCES Email (ID);
ALTER TABLE Survey ADD CONSTRAINT FKSurvey477897 FOREIGN KEY (EmailID) REFERENCES Email (ID);
ALTER TABLE Tester ADD CONSTRAINT FKTester150299 FOREIGN KEY (UserID) REFERENCES `User` (ID);
ALTER TABLE `Order` ADD CONSTRAINT FKOrder185564 FOREIGN KEY (SurveyID) REFERENCES Survey (ID);
ALTER TABLE `Order` ADD CONSTRAINT FKOrder666274 FOREIGN KEY (BookingID) REFERENCES Booking (ID);
ALTER TABLE Company ADD CONSTRAINT FKCompany99793 FOREIGN KEY (SurveyID) REFERENCES Survey (ID);
ALTER TABLE `User` ADD CONSTRAINT FKUser377807 FOREIGN KEY (SurveyID) REFERENCES Survey (ID);
ALTER TABLE `User` ADD CONSTRAINT FKUser858517 FOREIGN KEY (BookingID) REFERENCES Booking (ID);
ALTER TABLE SurveyReply ADD CONSTRAINT FKSurveyRepl940439 FOREIGN KEY (Sampling_TesterSamplingID, Sampling_TesterTesterID) REFERENCES Sampling_Tester (SamplingID, TesterID);
ALTER TABLE SurveyReply ADD CONSTRAINT FKSurveyRepl443169 FOREIGN KEY (Question_SamplingQuestionID, Question_SamplingSamplingID) REFERENCES Question_Sampling (QuestionID,
SamplingID);
ALTER TABLE Question_Sampling ADD CONSTRAINT FKQuestion_S727215 FOREIGN KEY (SamplingID) REFERENCES Sampling (ID);
ALTER TABLE Question_Sampling ADD CONSTRAINT FKQuestion_S477253 FOREIGN KEY (QuestionID) REFERENCES Question (ID);
ALTER TABLE SurveyReply ADD CONSTRAINT FKSurveyRepl317447 FOREIGN KEY (CompanyID) REFERENCES Company (ID);
ALTER TABLE Sampling ADD CONSTRAINT FKSampling755615 FOREIGN KEY (OrderID) REFERENCES `Order` (ID);
ALTER TABLE Service ADD CONSTRAINT FKService217570 FOREIGN KEY (OrderLineID) REFERENCES OrderLine (ID);
ALTER TABLE OrderLine ADD CONSTRAINT FKOrderLine150838 FOREIGN KEY (OrderID) REFERENCES `Order` (ID);
ALTER TABLE `Order` ADD CONSTRAINT FKOrder850343 FOREIGN KEY () REFERENCES Tester ();
ALTER TABLE Rating ADD CONSTRAINT FKRating430435 FOREIGN KEY () REFERENCES Tester ();
ALTER TABLE Sampling_Tester ADD CONSTRAINT `partakes in ` FOREIGN KEY () REFERENCES Tester ();
ALTER TABLE Sampling_Tester ADD CONSTRAINT `receives participation` FOREIGN KEY (SamplingID) REFERENCES Sampling (ID);
```

4.8.3 Representative set of operational SQL statement

4.8.3.1 Select

The following can be used to fetch booking records for certain ID. E.g. A visual card may show many records and this SQL query could be used to fetch the information.

```
SELECT * FROM Booking
WHERE ID=’12’;
```

The following can be used to fetch records with Rating overall score of more than 3.

```
SELECT * FROM Rating
WHERE overallScore > 3;

```

This may be referenced to another table via join using IN (SELECT * FROM Testers) WHERE firstName =”” to search for certain user with over a certain overall score.

The following may be used to find survey records which have not been completed and can be sent to the testers to complete.

```
SELECT * FROM SurveyReply
WHERE completed=’false’
```

4.8.3.2 Update

The following can be used to change a tester’s telephone number when requested.

```
UPDATE Tester SET telephoneNo = '01632960701'
WHERE UserID = '255';
```

The following can be used to set the role of a user to tester if they their application has been accepted.

```
UPDATE User SET role = 'Tester'
WHERE username = 'Abi;
```

The following can be used to change the quantity during a purchase if the user changes their mind.

```
UPDATE OrderLine SET quantity = '1',
WHERE ID = '142';
```

4.8.3.3 Delete

The following can be used to delete a tester's account if they requested (due to GDPR)

```
DELETE FROM Tester
WHERE UserID = '231';
```

The following can be used to delete a user order if they cancel their purchase before transaction is complete.

```
DELETE FROM Order
  WHERE ID = '345';
```

The following can be used to delete older email records if they are no longer needed. Could be done routinely or manually to conserve space.

```
DELETE FROM Email
WHERE ID = '100';
```

4.8.3.4 Insert

The following can be used to create a new booking record. Relevant values can be passed as parameters into VALUES to create a record accordingly.

```

INSERT INTO Booking(ID, PaymentID, BookingID, ClientID, BookingDate, BookingDescription, ClientName, StatusCode, OrderID, EmailID)
VALUES ('1','1','1','1','2020-11-11','UT Component review','NHS','booked','1', 'EmailID');

```

The following can be used to create a new User record. E.g. creating an admin account.

```

INSERT INTO User(ID, role, username, email, password)
VALUES ('1', 'Admin','root','JohnDoe@root.com','root');

```

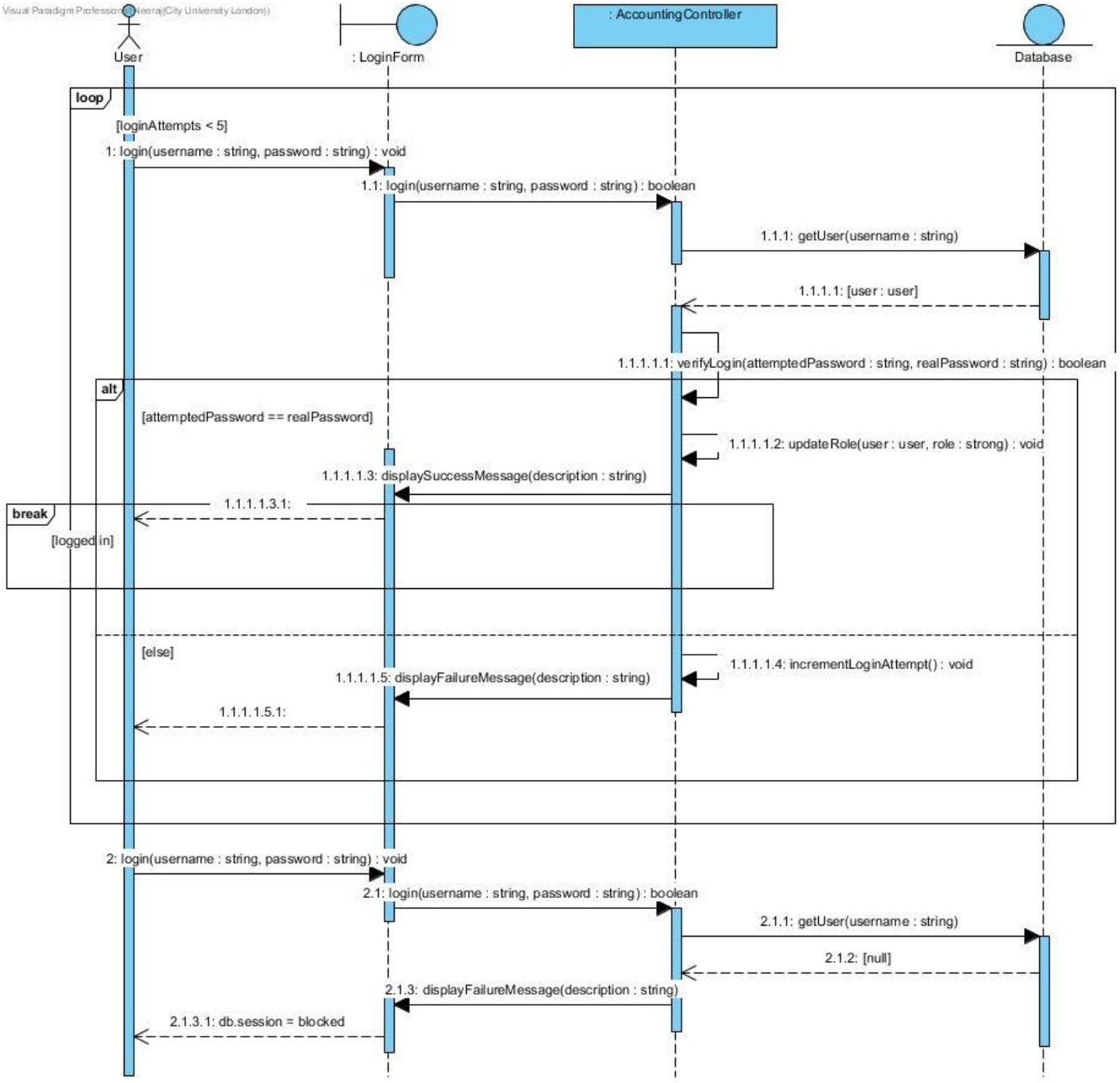
The following can be used to create order lines and set items within it for an order.

```

INSERT INTO OrderLine(ID, description, quantity, subtotal, OrderID)
VALUES ('1', 'Tester (hearing)','1','100','255');

```

4.9 Sequence diagram

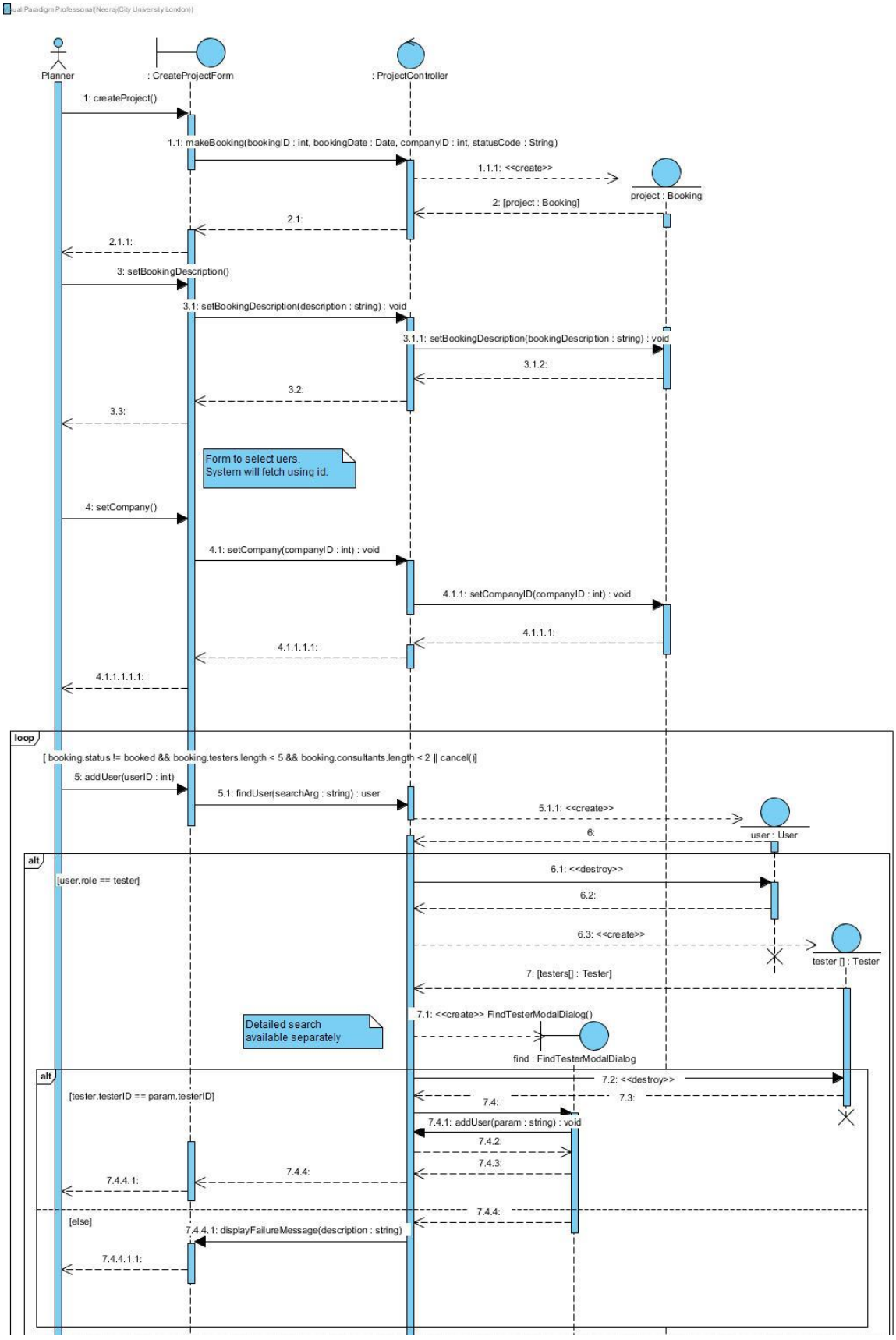
A representational sample of sequence diagrams were illustrated to describe the order and flow of data between groups of **important** classes. Key features were also included in the illustrations to ensure correctness during implementation phase. The diagrams which have sequence diagrams have been selected for their relative importance to client and overall complexity in terms of data exchange. Diagrams which contained similar logic have been removed. E.g. Log out, Adding/ Removing users from projects, adding description to project, etc. A diagram may contain multiple use cases where specified.

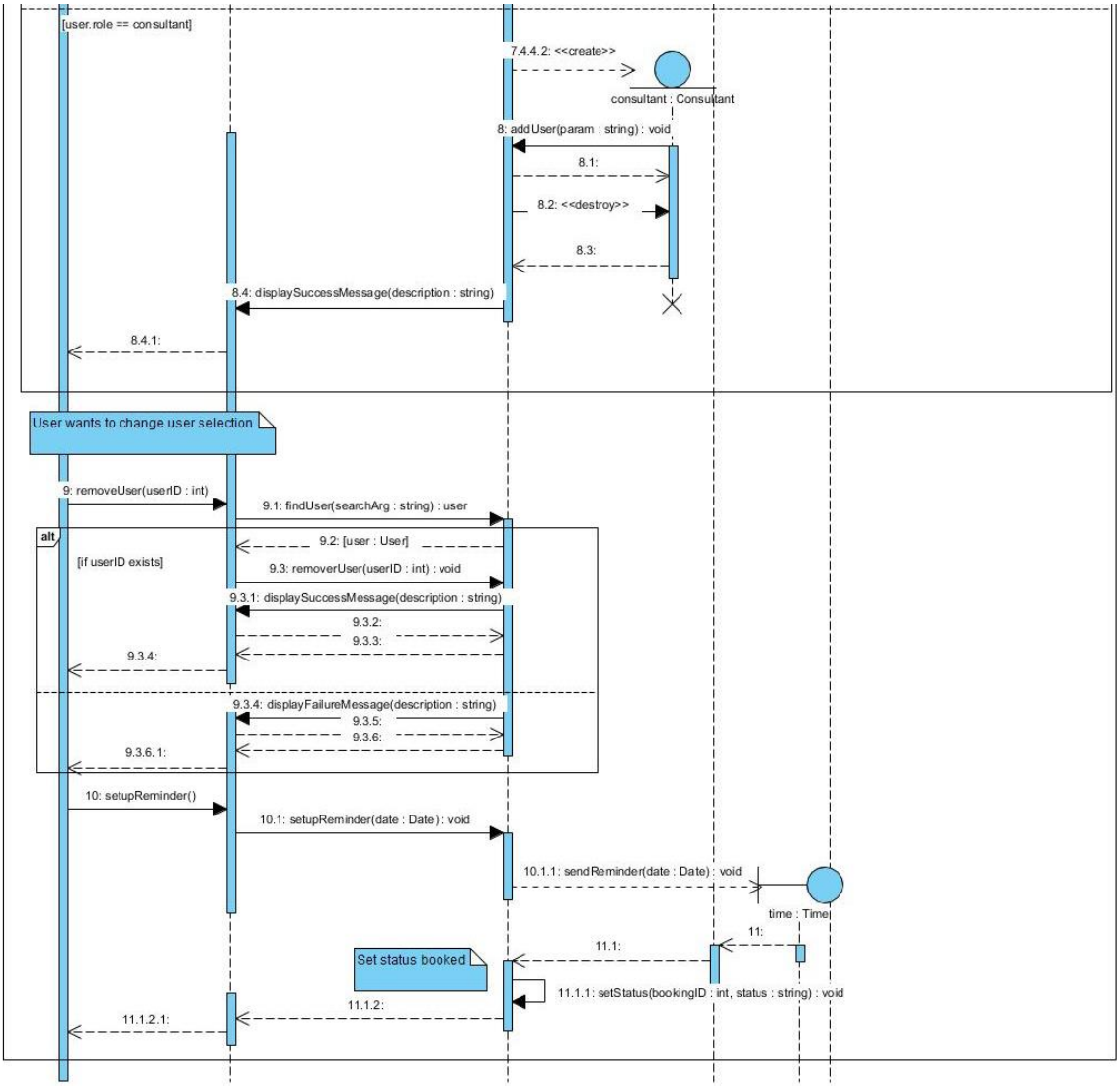
4.9.1 Login



4.9.2 Create Project

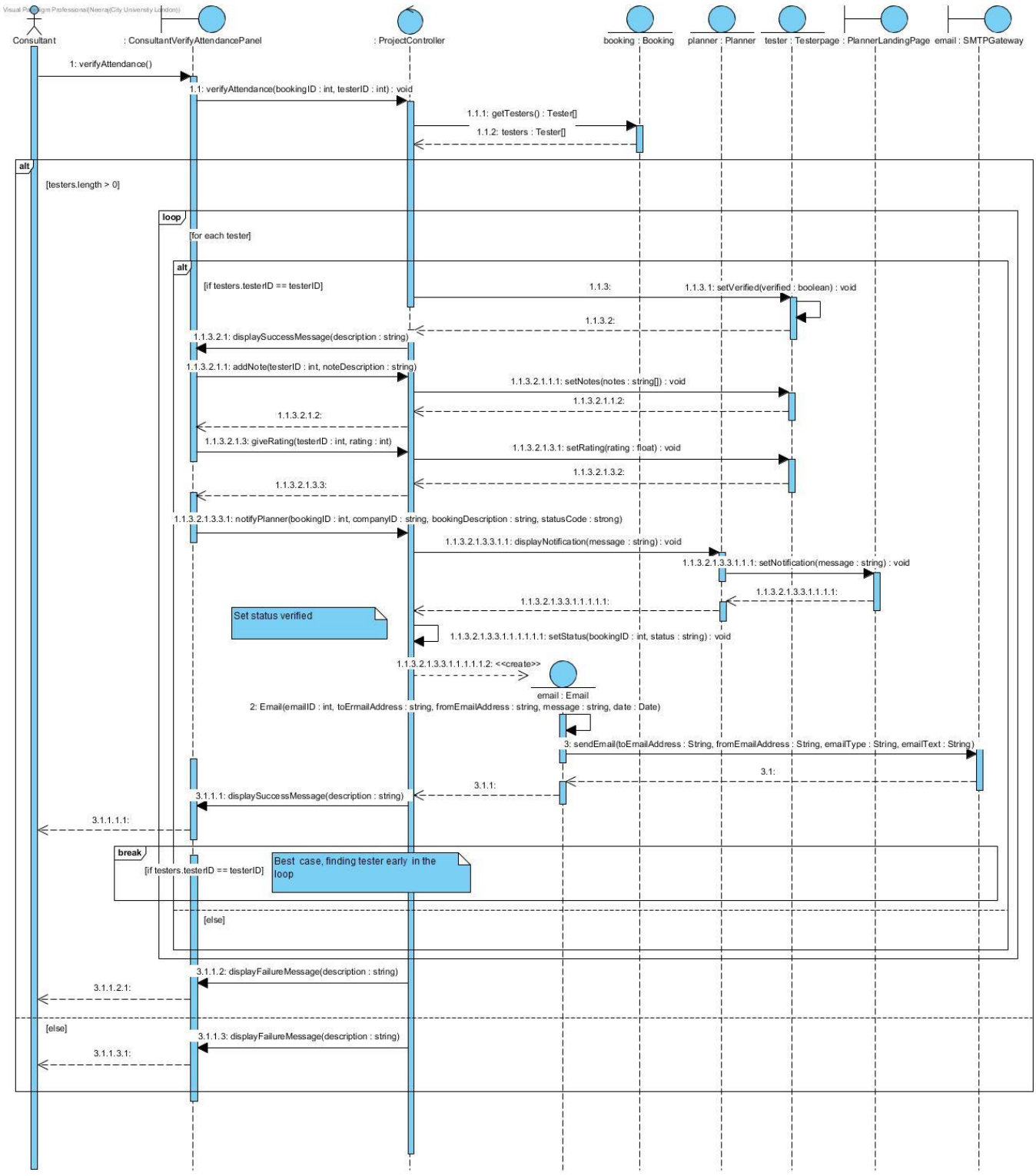
4.9.2.1 setBookingDescription(), setCompany(), addUser(), findTester(), removeUser(), setupReminder()



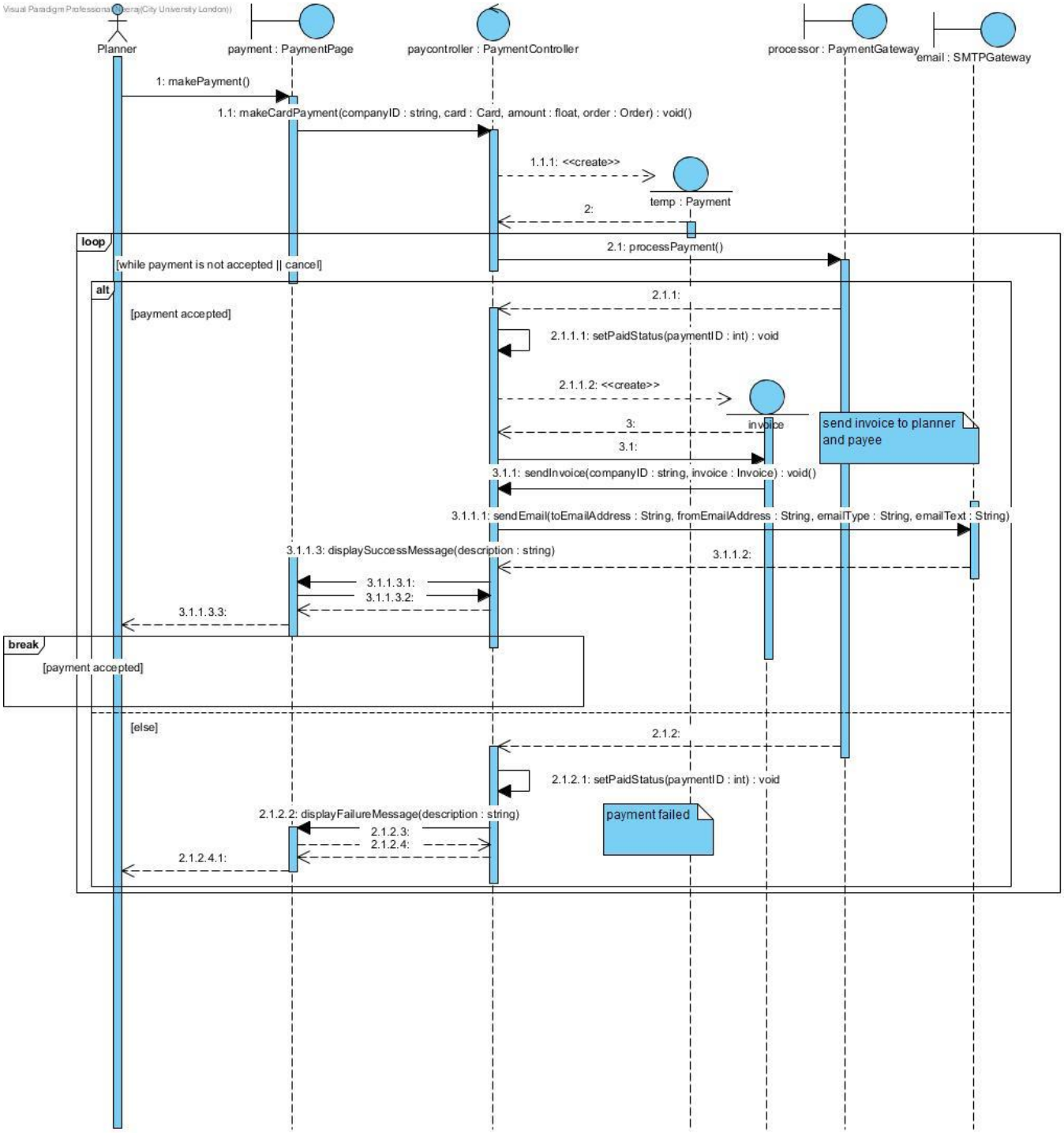

[user.role == consultant]
7.4.4.2: <<create>>
consultant : Consultant
8: addUser(param : string) : void
8.1:
8.2: <<destroy>>
8.3:
8.4: displaySuccessMessage(description : string)
8.4.1:
User wants to change user selection
9: removeUser(userID : int)
9.1: findUser(searchArg : string) : user
alt
[if userID exists]
9.2: [user : User]
9.3: removerUser(userID : int) : void
9.3.1: displaySuccessMessage(description : string)
9.3.2:
9.3.3:
9.3.4:
9.3.4: displayFailureMessage(description : string)
9.3.5:
9.3.6:
9.3.6.1:
10: setupReminder()
10.1: setupReminder(date : Date) : void
10.1.1: sendReminder(date : Date) : void
time : Time
11:
11.1:
Set status booked
11.1.1: setStatus(bookingID : int, status : string) : void
11.1.2:
11.1.2.1:

4.9.3 Verify Attendance

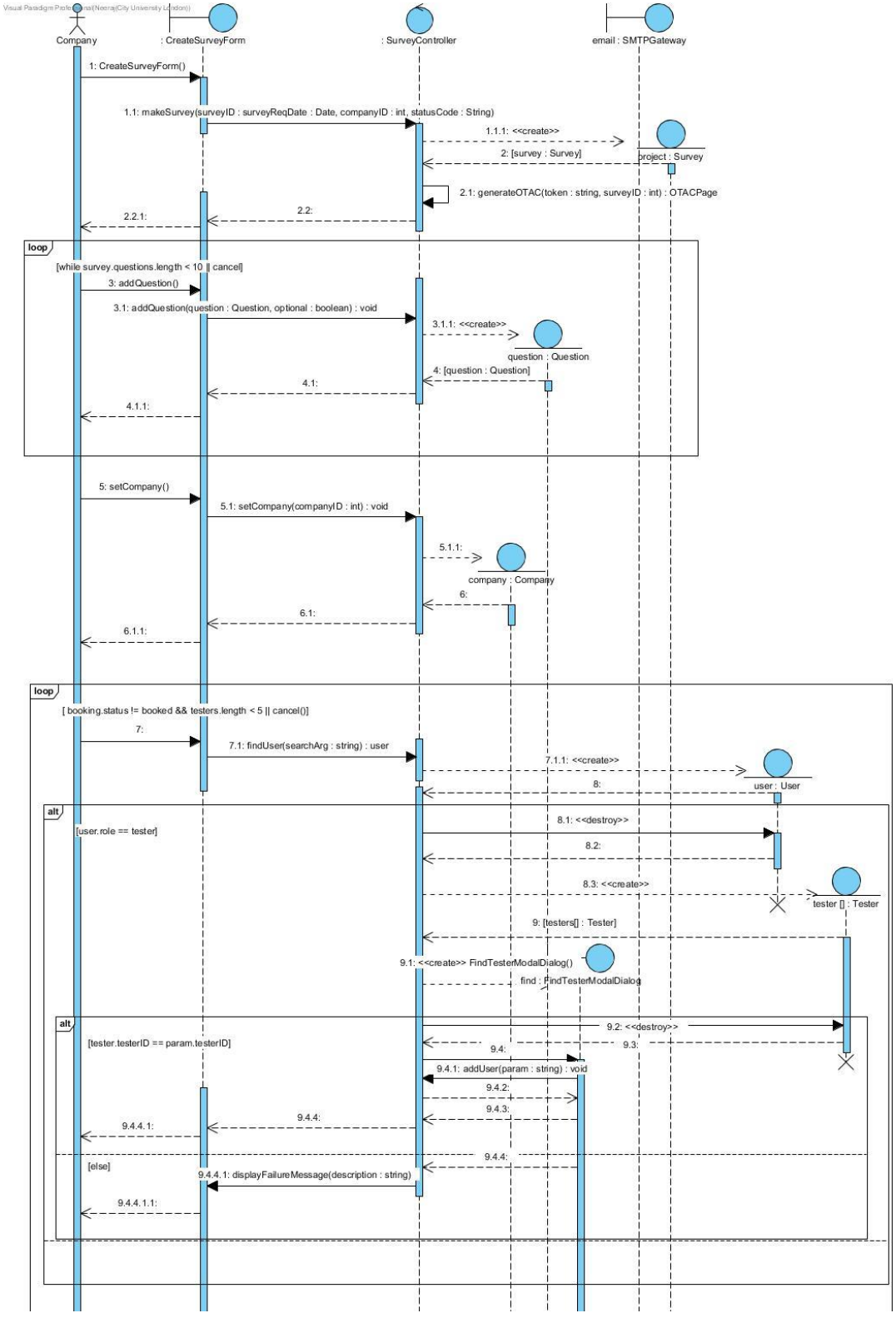
4.9.3.1 addNote(), giveRating(), notifyPlanner()

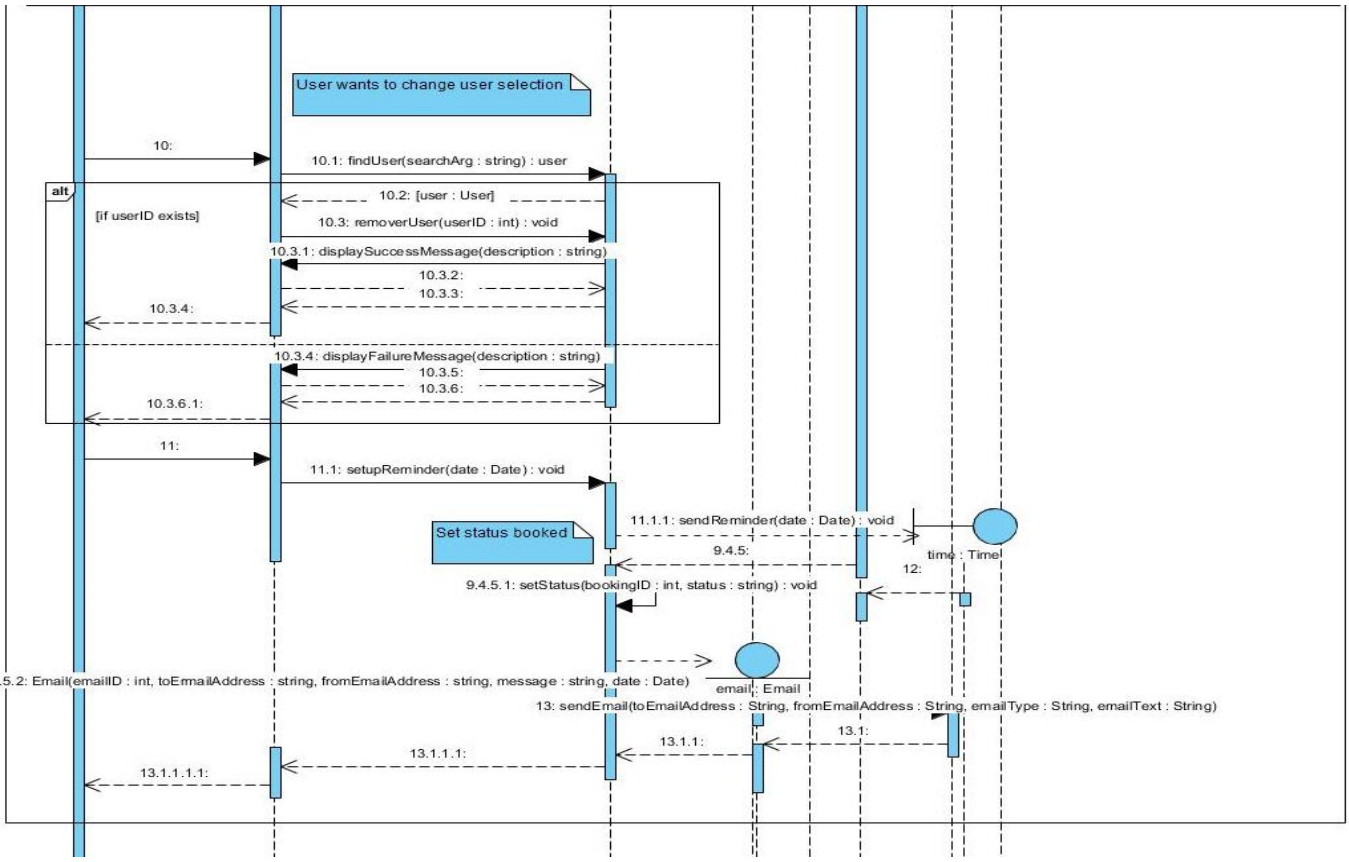


4.9.4 Make Payment

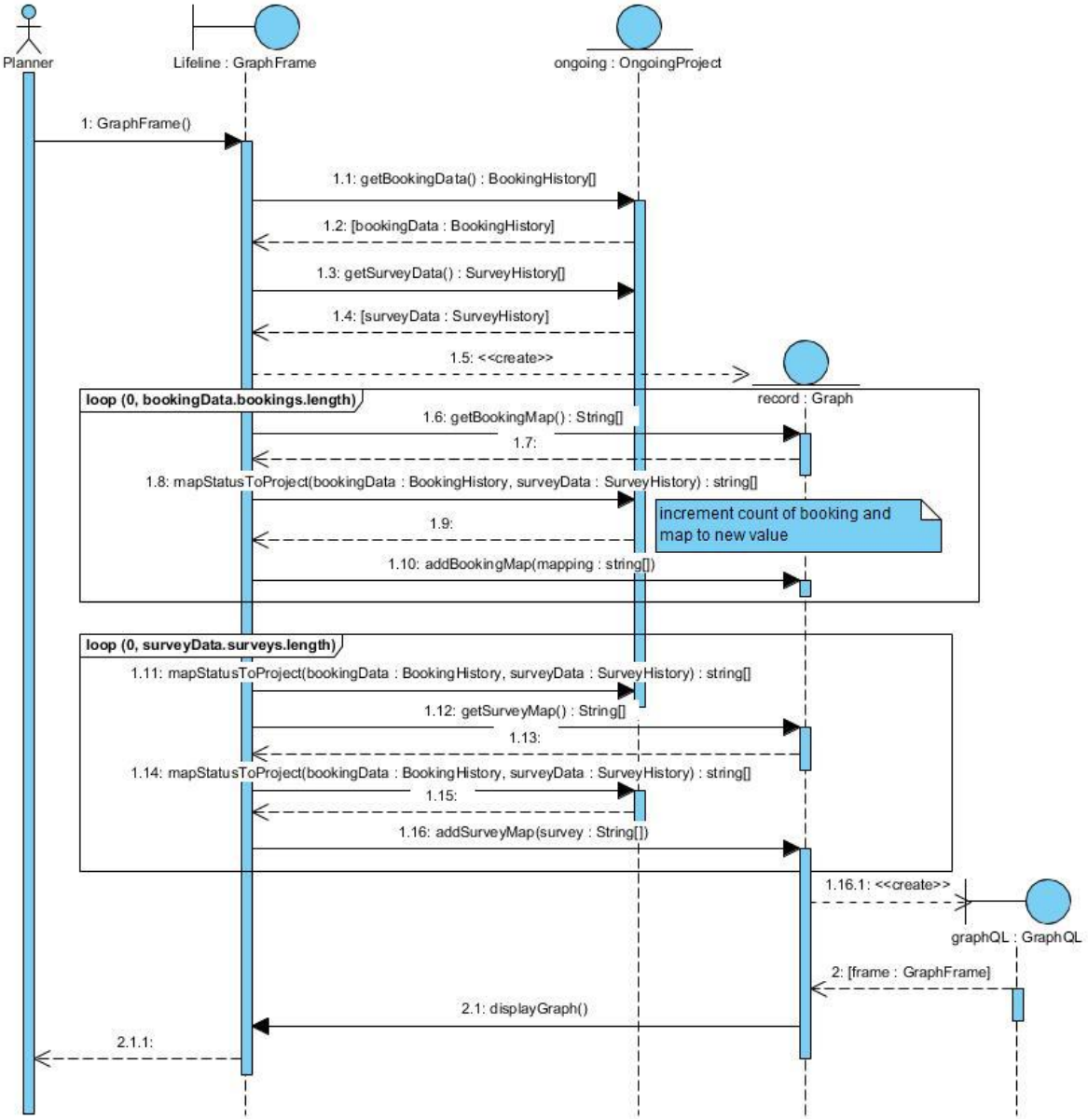


4.9.5 Create Survey




User wants to change user selection
10:
10.1: findUser(searchArg : string) : user
alt
[if userID exists]
10.2: [user : User]
10.3: removerUser(userID : int) : void
10.3.1: displaySuccessMessage(description : string)
10.3.2:
10.3.3:
10.3.4:
10.3.4: displayFailureMessage(description : string)
10.3.5:
10.3.6:
10.3.6.1:
11:
11.1: setupReminder(date : Date) : void
11.1.1: sendReminder(date : Date) : void
time : Time
Set status booked
9.4.5:
12:
9.4.5.1: setStatus(bookingID : int, status : string) : void
.4.5.2: Email(emailID : int, toEmailAddress : string, fromEmailAddress : string, message : string, date : Date)
email : Email
13: sendEmail(toEmailAddress : String, fromEmailAddress : String, emailType : String, emailText : String)
13.1:
13.1.1:
13.1.1.1:
13.1.1.1.1:

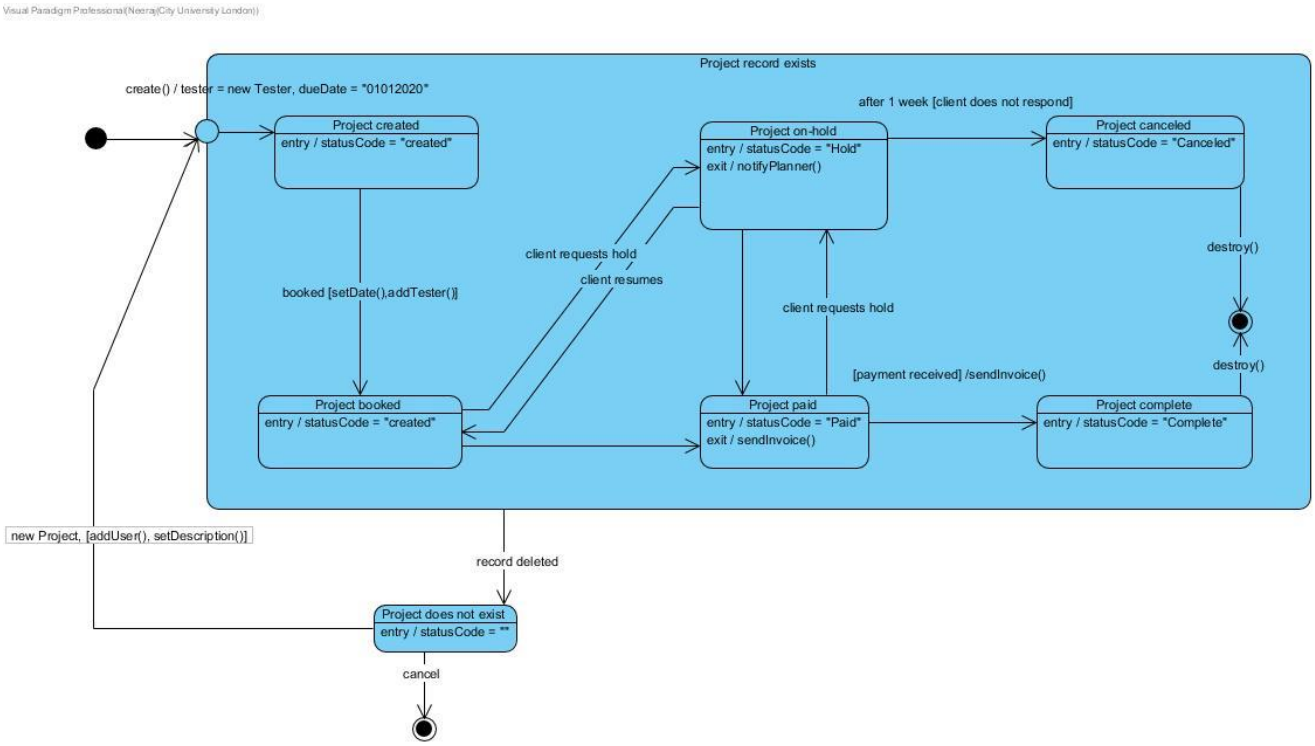
4.9.6 Analytics – Show Ongoing projects (pie chart)



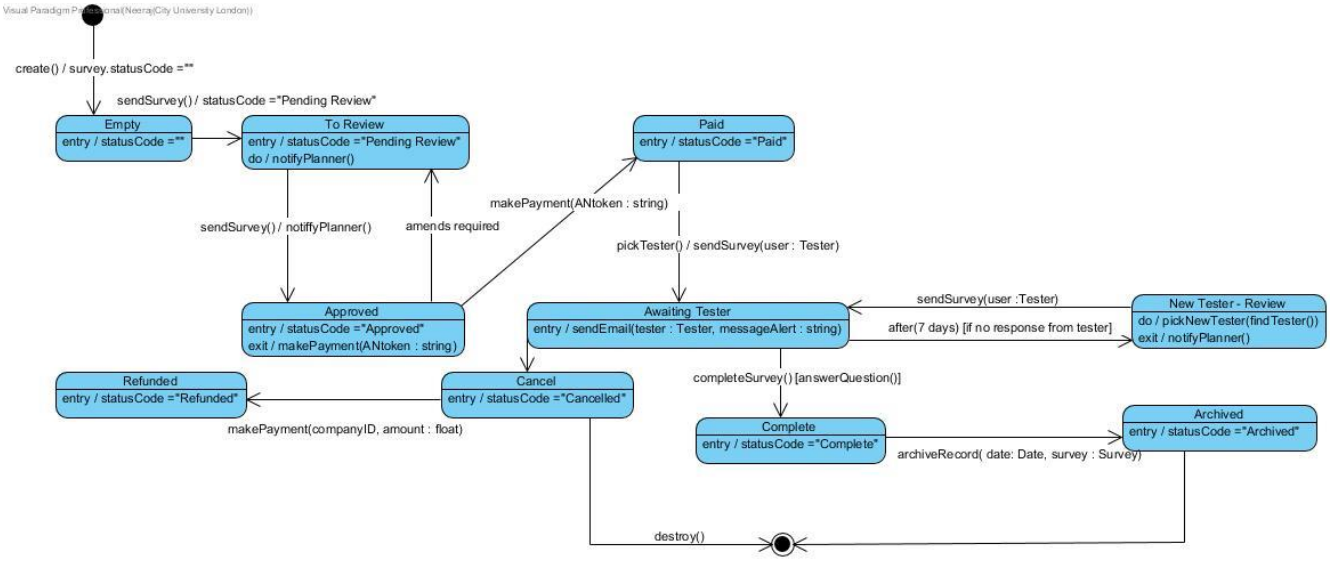
4.10 State diagrams

These diagrams describe the dynamic nature of entities.

4.10.1 Booking state machine

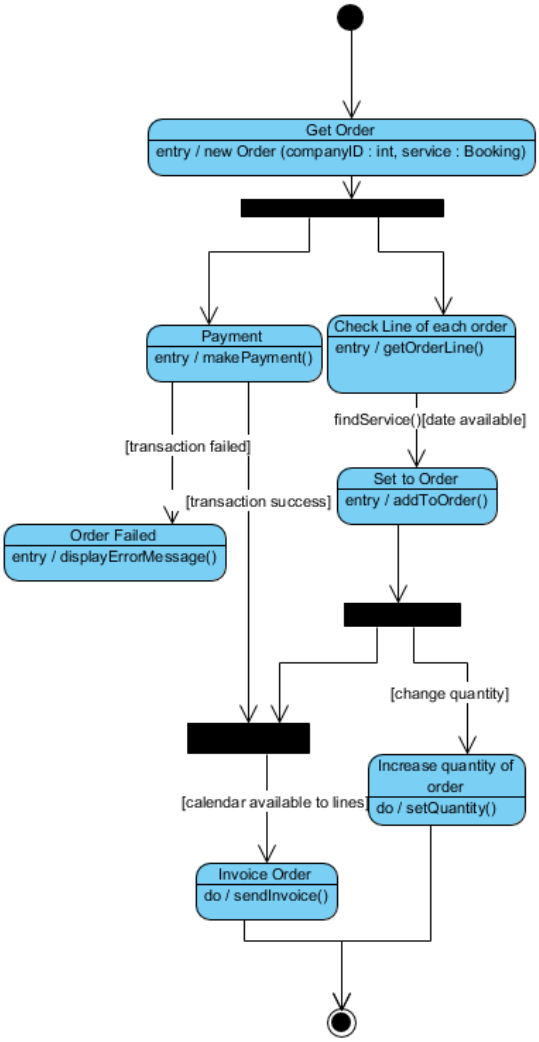


4.10.2 Survey state machine



4.10.3 Order state chart/ Activity diagram

Order is best described by activity diagram due to its nature of state transition in means of activities rather than simply changing states.

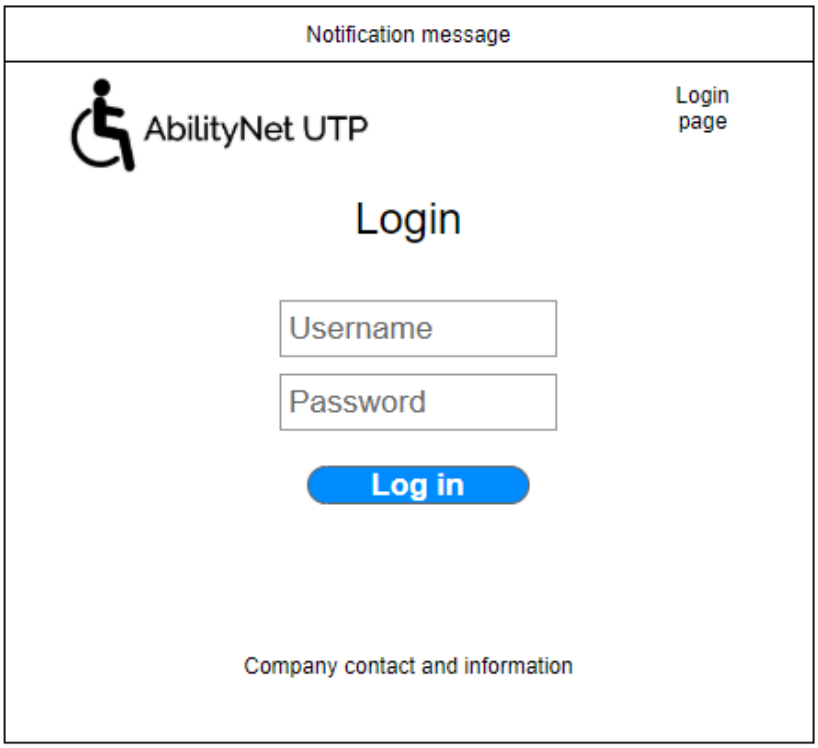


4.11 GUI

These GUI diagrams give a high-level overview of boundary entities on the page which the user can use to interact with the UTP system.

4.11.1 Login

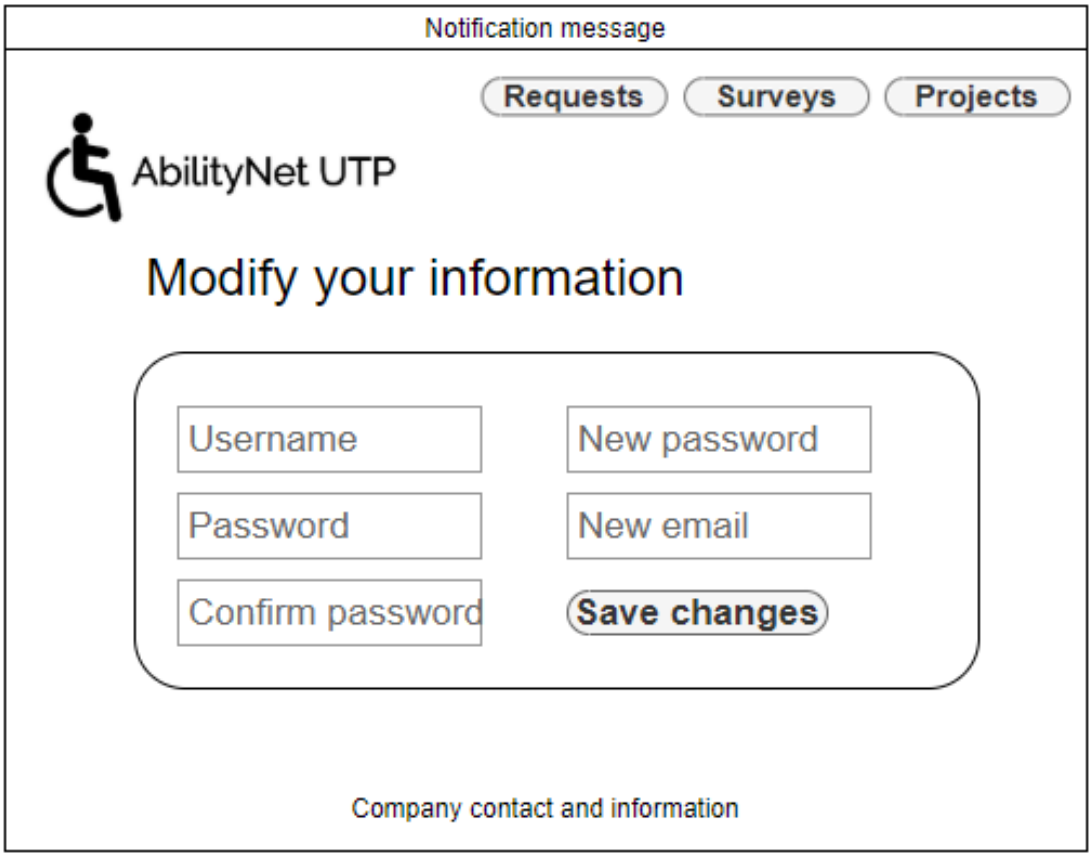
Boundary class	How to navigate
Input field: Username, Password Button: Login	Users can log into the UTP systems from the AbilityNet current website after triggering a button. That button will render this page where users may enter their credentials into the input field to gain access to their account in the system.



4.11.2 Modify details (Staff)

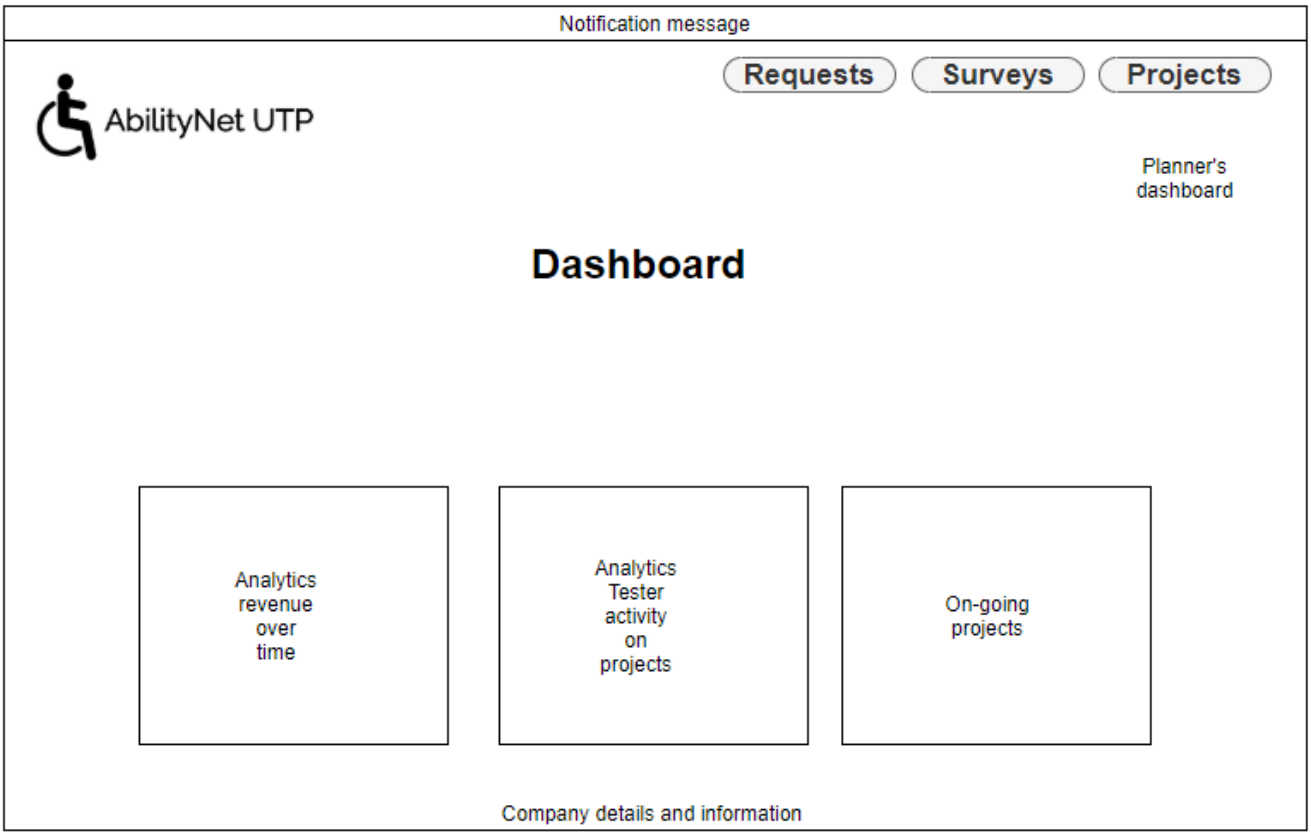
Testers must request a password reset via email.

Boundary class	How to navigate
Input field: Username, Password, confirm password, New password, New email Button: Save changes	The modify your information page allows the user to update their existing details or reset them if they have been forgotten. The user would require inputting the Username and password first in order to change their details and enter the password ensuring extra safety for the data of the user. After the security details have been confirmed, the user would just need to input the detail he wishes to update (password or email) and save the changes. By saving the changes the user's details will be updated and saved in the system.



4.11.3 Dashboard - Planner

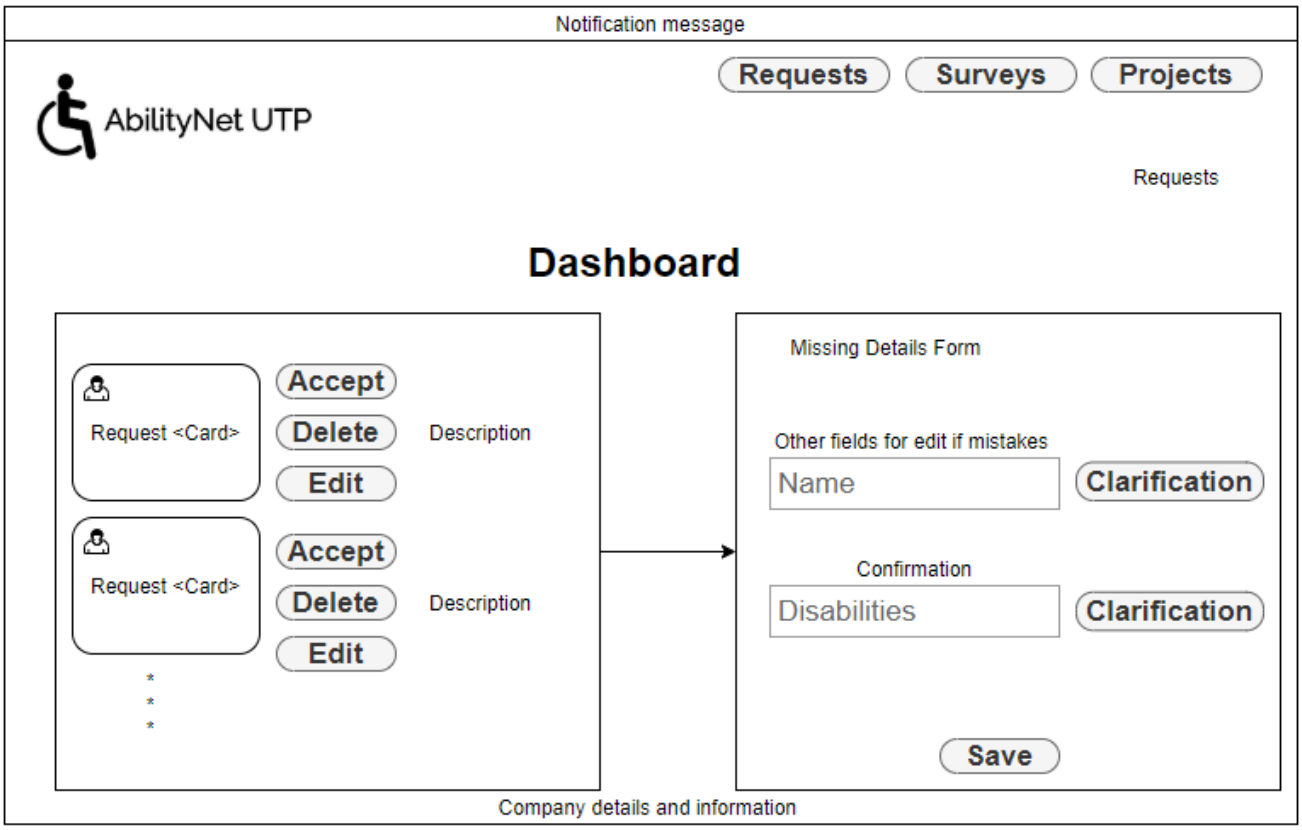
Boundary class	How to navigate
Buttons: Request, Survey, Project. Interactive GraphQL elements.	Planner logs in and is redirected to /dashboard. There, 3 interactive graphs are displayed which shows analytics on the different on going UTP projects such as user testing bookings and surveys. There are navigation buttons on the top which the user can trigger and go to different pages where they may view requests, surveys and projects.



4.11.4 Access Requests

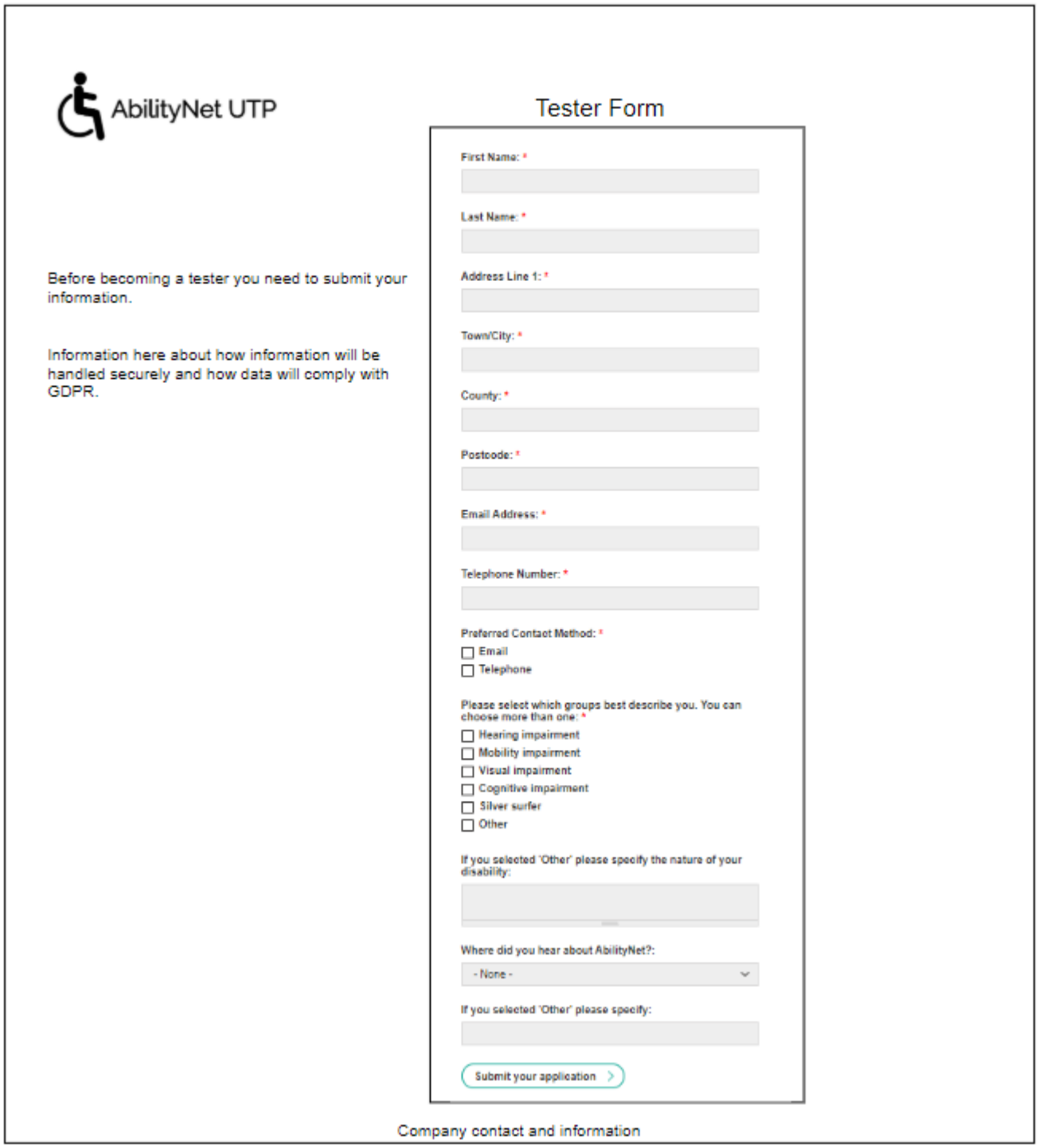
4.11.4.1 Missing details form

Boundary class	How to navigate
Button: Accept, Delete, Edit Input Field: Name, Disabilities	The main functionality is for planners to view different requests made by users to become testers. The planner can accept their request, delete from database or edit, which allows the planner to modify relevant input fields which may require further clarification- which can be done via phone call or email. The planner can then save the changes and accept the request.



4.11.5 Sign up form

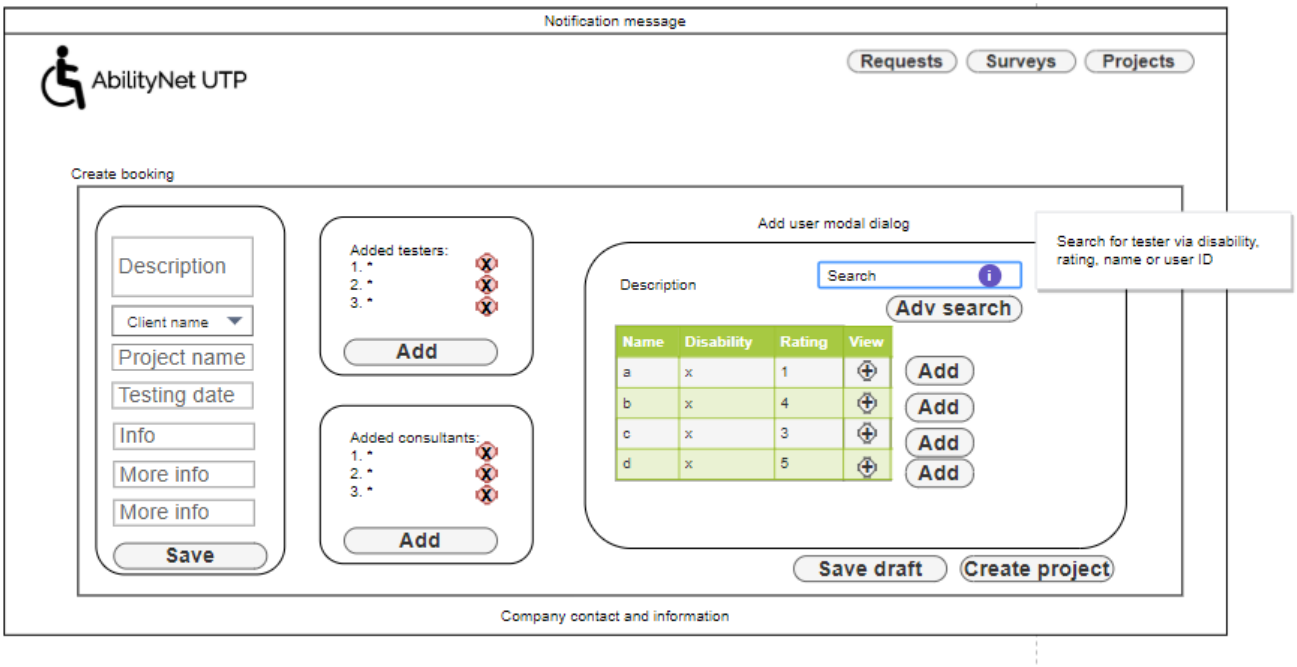
Boundary class	How to navigate
Input Field: First name, Last name, Address Line 1, Town/City… *Type=checkbox:* Preferred contact method Button: Submit application	Users can enter relevant information into the input fields and submit to submit a request to become a tester.



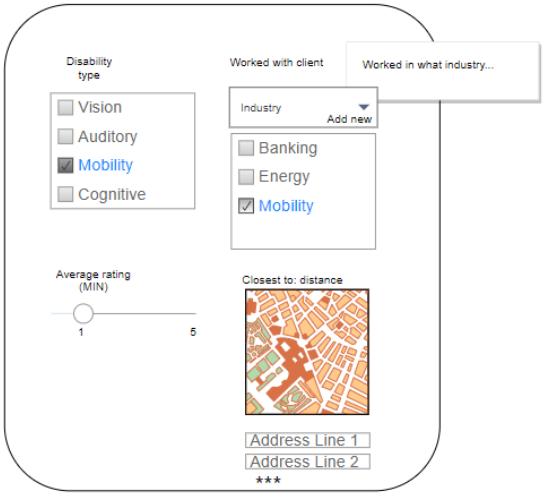
4.11.6 Create new project

4.11.6.1 Find user modal

Boundary class	How to navigate
Input field: Search field, input project details. Button: Add tester, save button, remove testers.	The planner can create a new project by entering project details on the input fields and saving it. Then, they can add and remove testers and consultants until the limit is reached. Add user dialog is triggered when add button is triggered and allows users to make queries for testers and consultants. Advanced search button triggers another modal which provides more detailed search, allowing users to be chosen based on their location, expertise, ranking, etc.

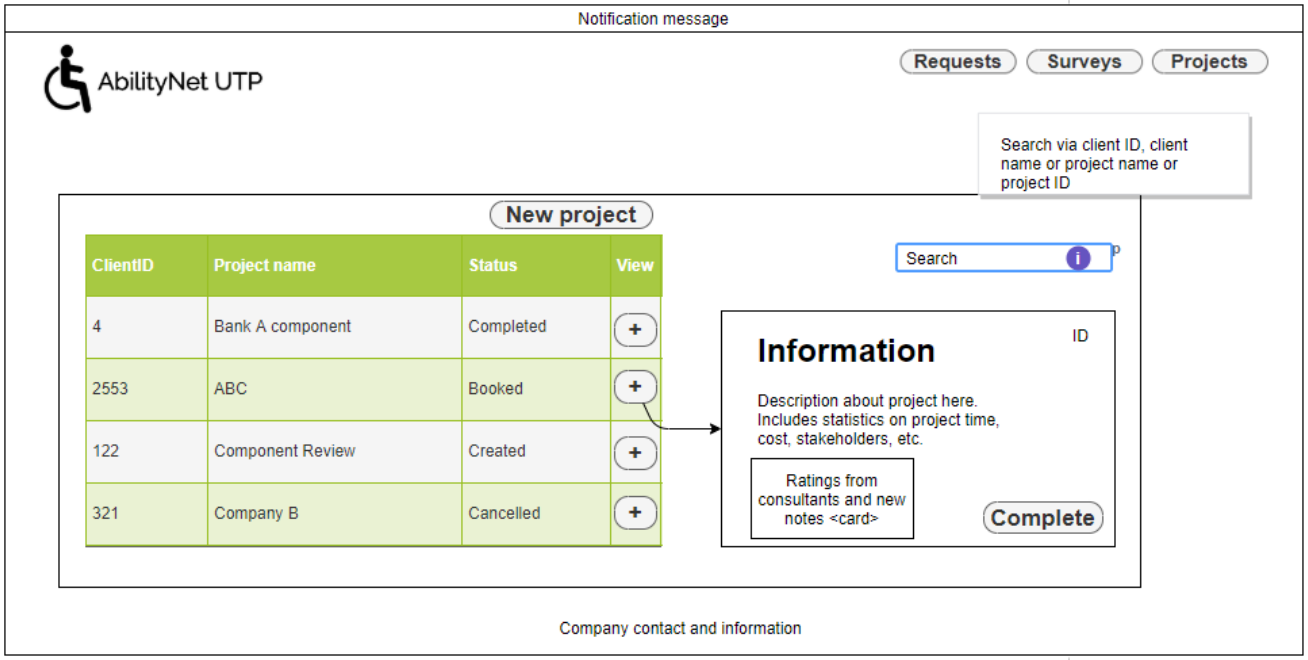


4.11.6.2 Advanced search modal



4.11.7 Access projects

Boundary class	How to navigate
Button: View project record to view records. Input field: Search	Planners can view project records created in the UTP. They can trigger the view button to display an overlay which provides more information on the record. For projects which are **not** complete (status != completed), the planners can click on the complete button to sign off the project and submit a request for payment to tester.

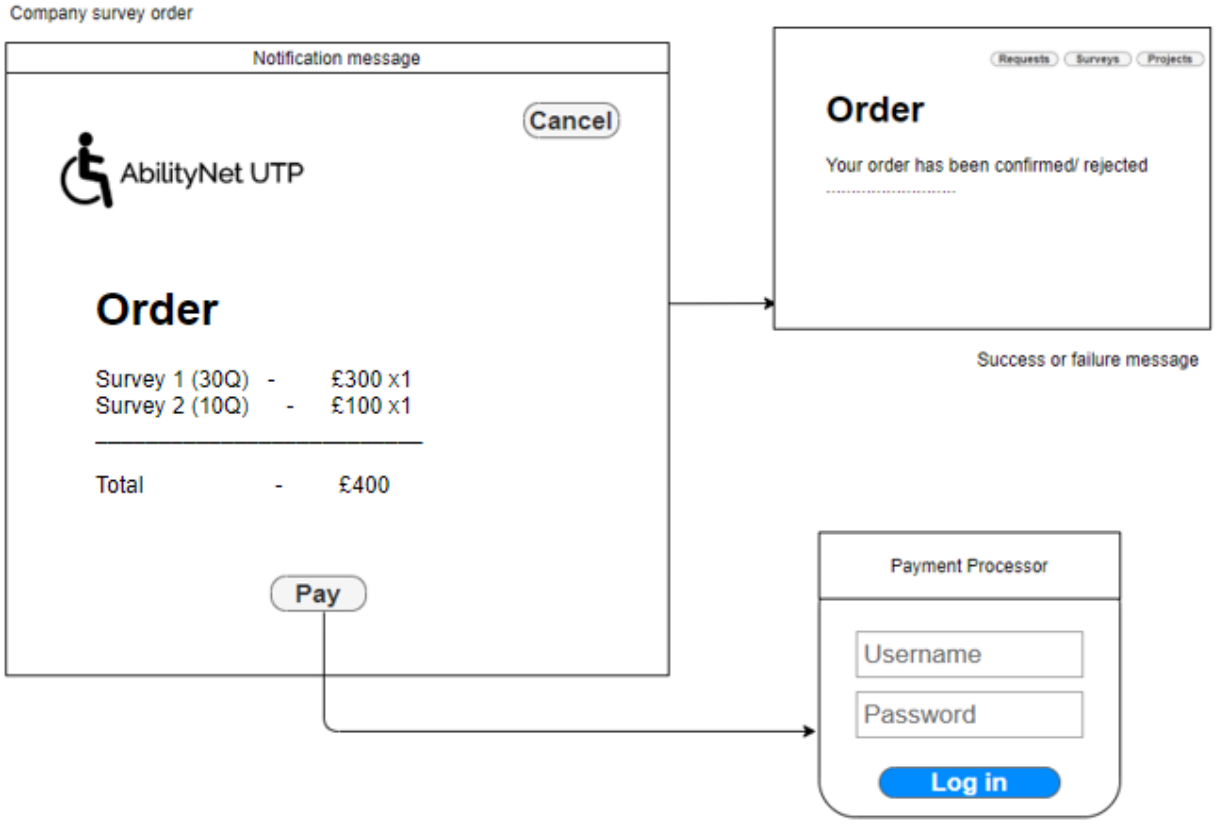


4.11.8 Payment – Survey Dashboard (Pay Tester)

4.11.8.1 Order success/ failure status updates.

N.B. For implementation. Ensure that dynamic updates are also communicated to assistive technology users using **aria-live** attribute.

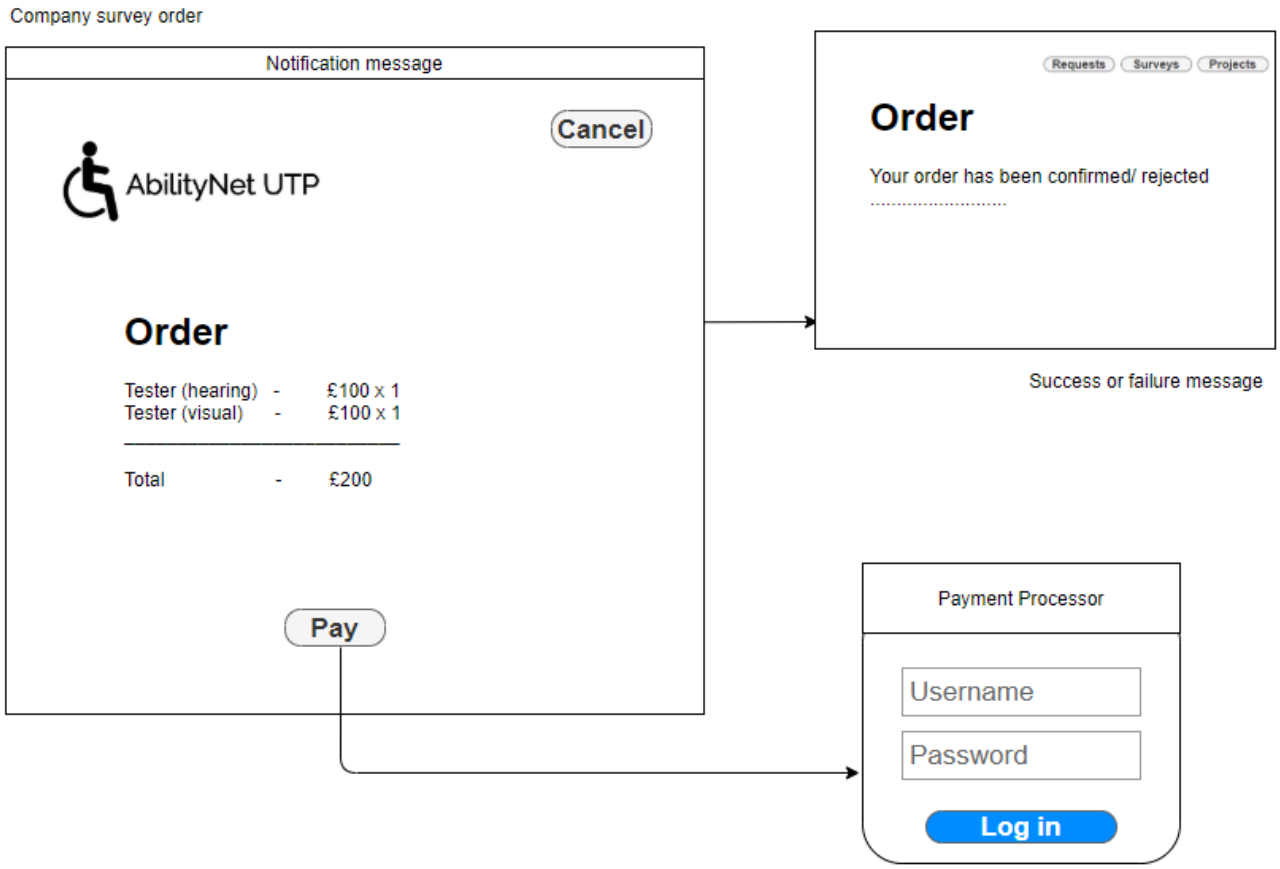
Boundary class	How to navigate
Input field: Payment processor username and password. Button: Pay button to initiate log in and log in for payment processor.	A list of all the surveys and their total question answered will be displayed to ensure that the tester is paying for the correct order. The total price would be displayed at the end and to pay the fee the company user would just have to press the pay button. This will redirect the company to a login page for their payment processor and as soon as the payment has been processed a notification of the order status will be displayed. If the payment transaction fails, it will be rejected, and error message will show. If the payment was successful, the order status will be displayed as confirmed.



4.11.9 Payment – Project (Planner)

4.11.9.1 Order success/ failure status updates.

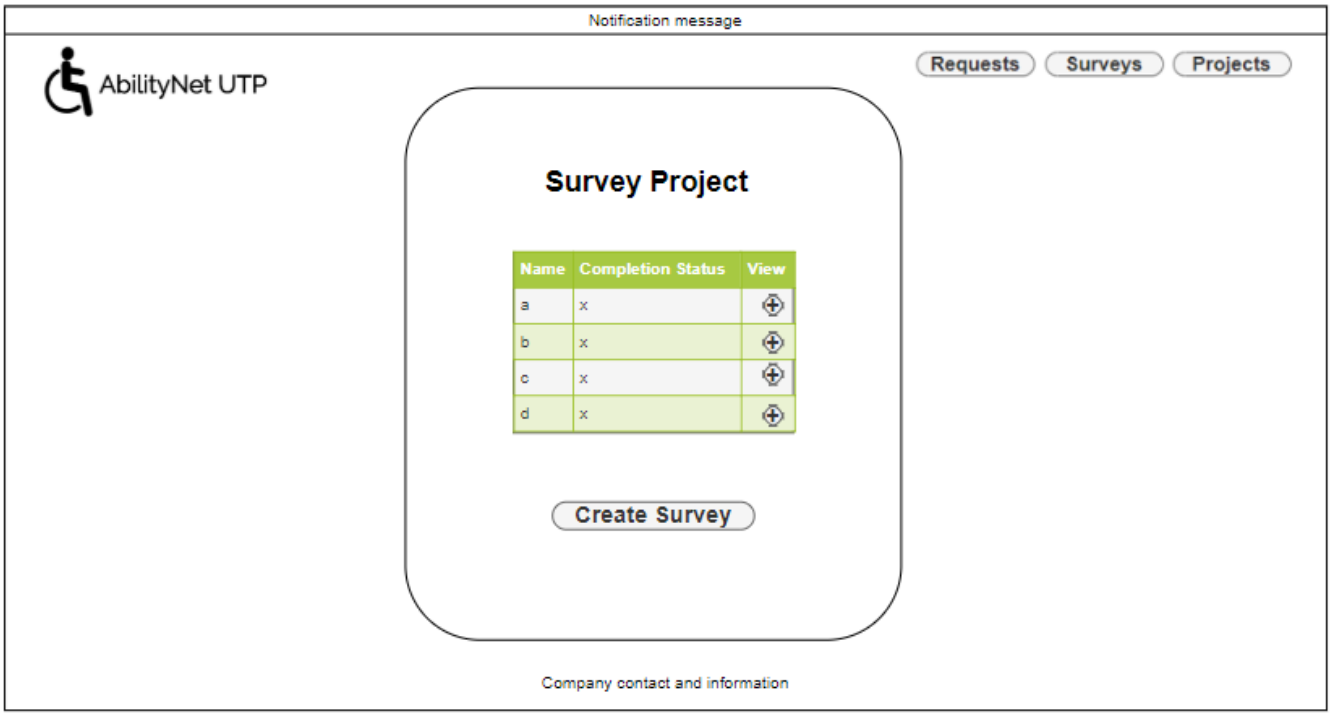
Boundary class	How to navigate
Input field: Username and password input fields in payment processor Button: Cancel button and pay button, and log in button for payment processor.	The order will display the name of each tester and their disability with the quantity of surveys they have completed and the line total they require. At the bottom, the total cost will be calculated and to proceed with the payment the user would just need to press the pay button. The user will need to provide the username and password for the payment processor. If the payment transaction fails the order will display the payment as rejected, otherwise it will present an order confirmed page.



4.11.10 Survey projects with creative survey feature

4.11.10.1 aaa

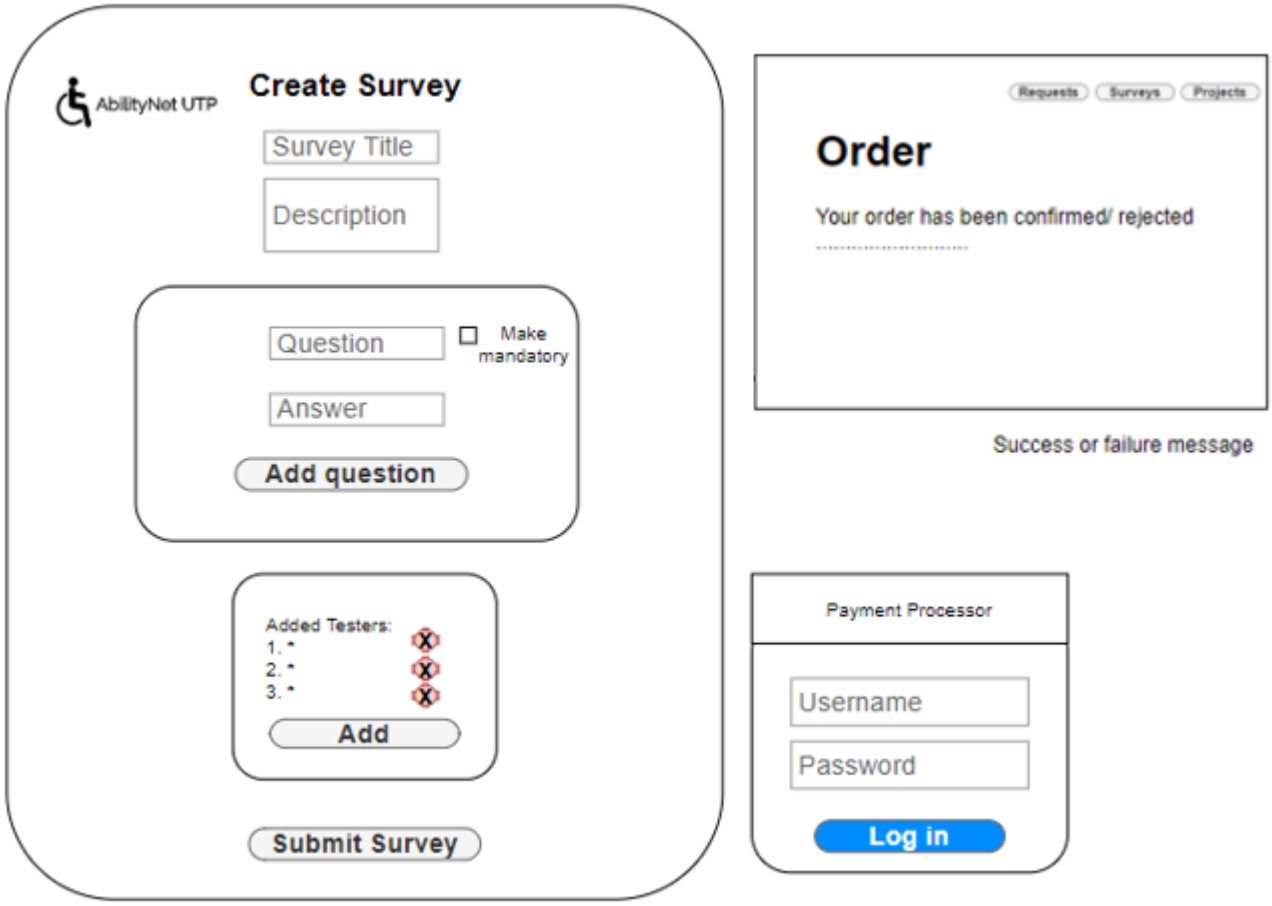
Boundary class	How to navigate
Input field: Button: Create survey button	Planner logins and goes to /survey. Shows how many people have completed surveys and they can view the survey by triggering the button. They can also create a new survey by triggering create survey button.



4.11.11 Create Survey (Pay AN by Company)

4.11.11.1 Payment API included

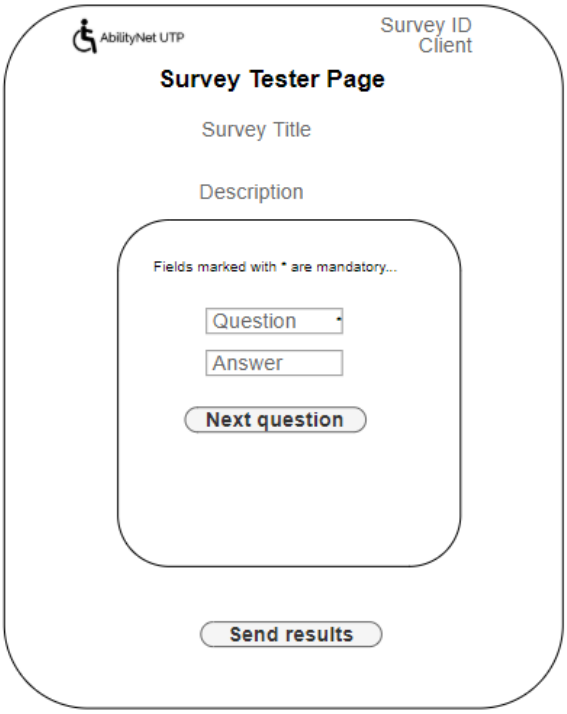
Boundary class	How to navigate
Input field: Survey title, description, question, answer Button: Pay button, submit survey button.	Companies enter data into the relevant input fields and can add questions which testers should answer. The questions can be made mandatory by selected the checkbox. The user can add a question which allows for new questions to be added. They can submit the survey once enough questions are added and payment is complete.



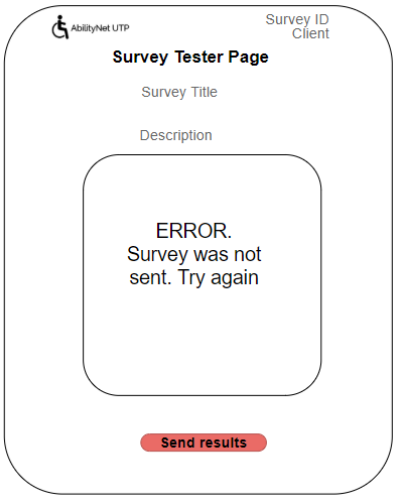
4.11.12 Access survey (Tester triggers link) OTAC

4.11.12.1 Success and Failure message

Boundary class	How to navigate
Input field: Question, Answer Button: Next Question, Send results	Testers click on OTAC links which are sent to them on email. The users enter their responses to the questions and send results. The button becomes disabled once done and message appears upon success or failure.



Success message

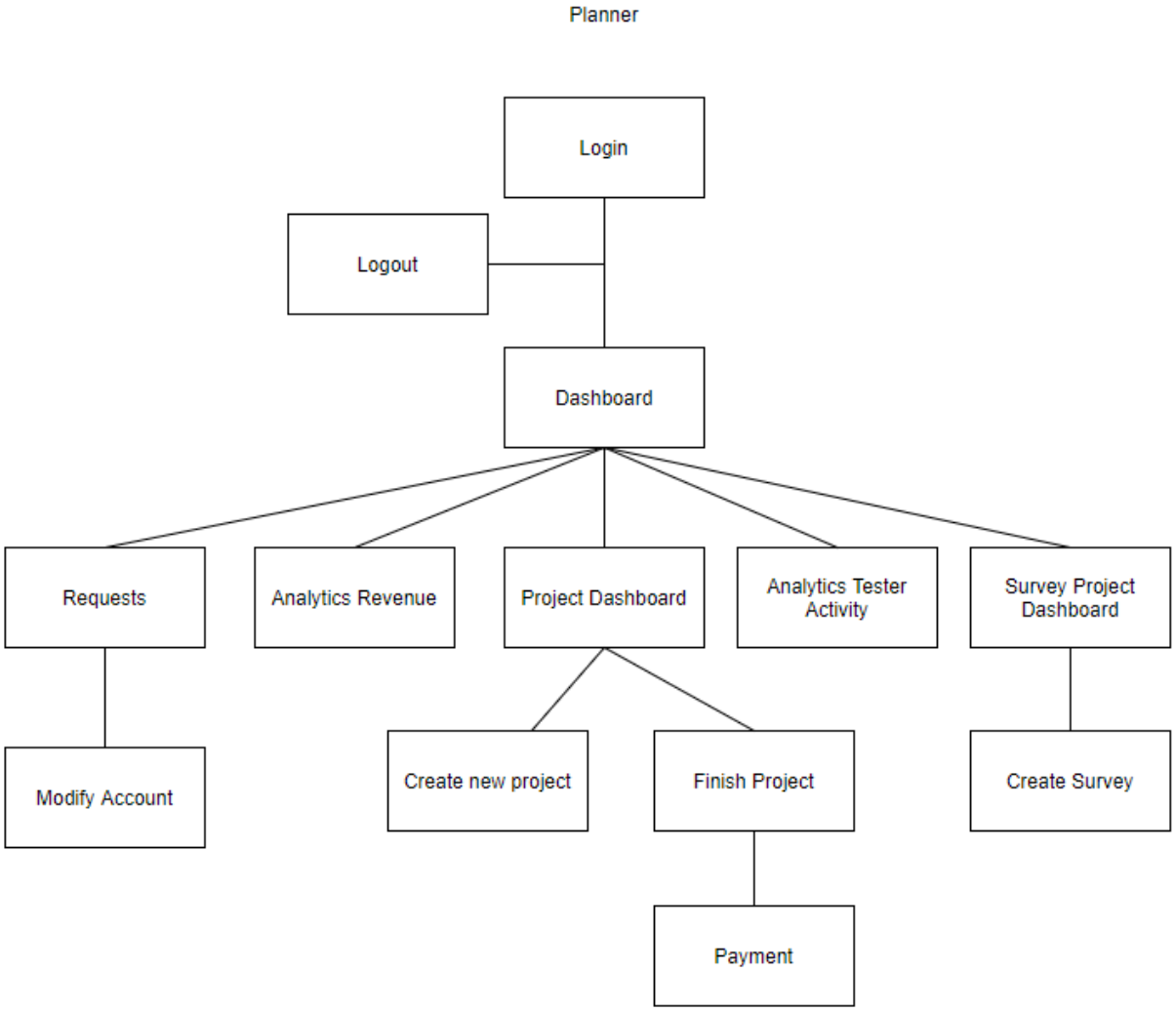


Failure message

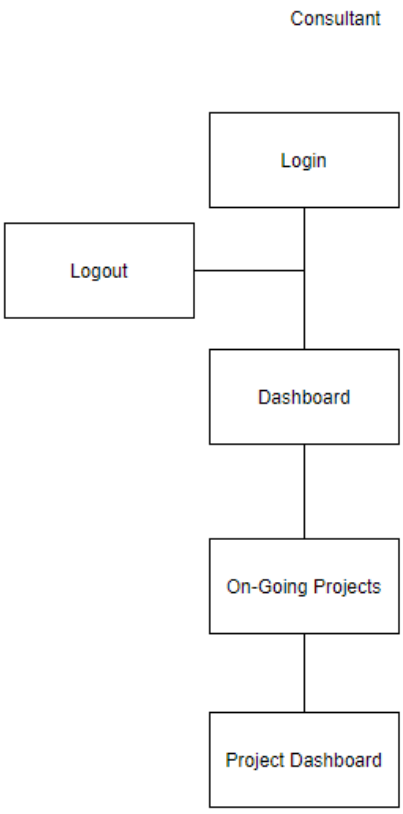
4.12 SITEMAP

The sitemap provides a high-level overview of the different views and operations accessible by different user roles.

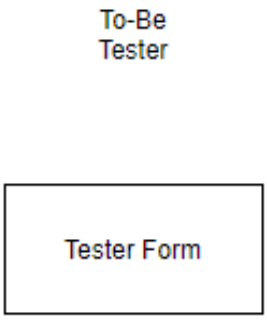
4.12.1 Planner



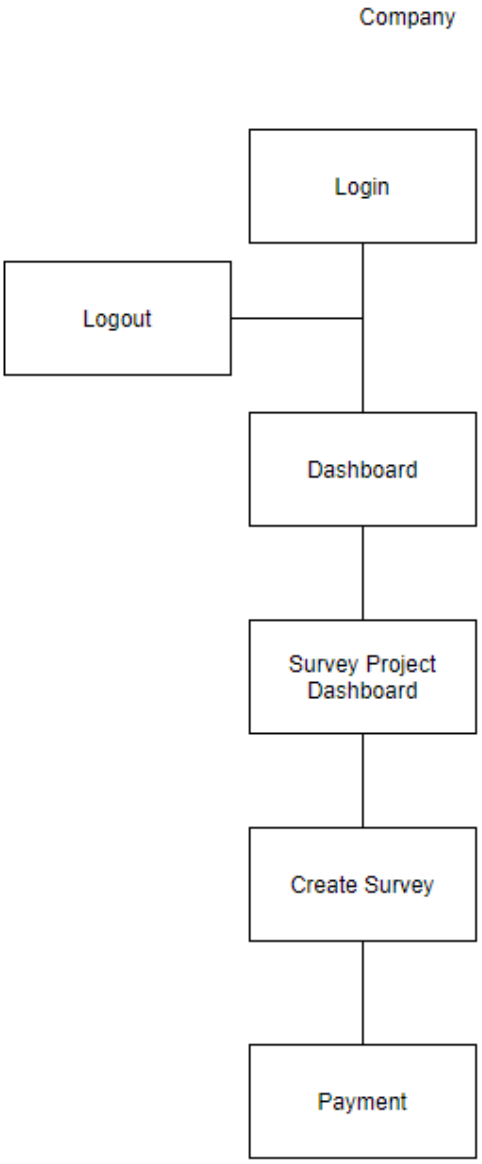
4.12.2 Consultant



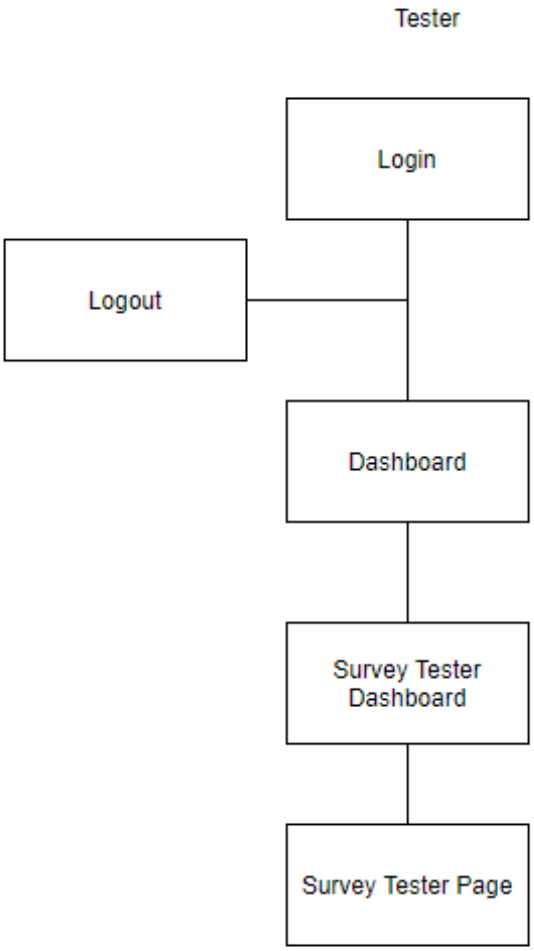
4.12.3 User (Submit Application Form)



4.12.4 Company



4.12.5 Tester


Tester
Login
Logout
Dashboard
Survey Tester Dashboard
Survey Tester Page