

CS993 Group Project
TimeKeep: Time Booking System
for Pulsion Technology Ltd.

Group Awe

**By Katie Nelson, Michael Flood, Barbara Pye, Vijayshree Joshi,
Luke Nolan**

Contents

1. Introduction	4
2. Requirements	4
2.1 Requirements introduction	4
2.2 Moscow prioritisation	4
2.3 Initial requirements document	5
2.3.1 Must-have requirements	5
2.3.2 Could-have requirements	5
2.4 Requirements clarification	5
2.4.1 Must-have updates	5
2.4.2 Should-have updates	6
2.5 Non-functional requirements	6
2.5.1 Must-have security requirements	6
2.5.2 Usability requirements	6
2.6 User stories	6
2.7 Backlog grooming	7
3. Project management and lifecycle	8
3.1 Introduction to project management	8
3.2 Gantt charts	8
3.3 Software lifecycle and Scrum	9
3.4 Tools used	10
4. Design	10
4.1 Introduction to design	10
4.2 High-level design and architecture	10
4.3 Low-level design & user interface	11
5. Construction	13
5.1 Construction introduction	13
5.2 Programming languages used	14
5.3 Framework used	15
5.4 Tools used	15
5.4.1 Version control tools	15
5.4.2 Build tools	15
5.5 Application Functionality	15
5.5.1 Sign up and login	15
5.5.2 Main screen	16
5.5.3 Add projects and tasks	16

5.5.4 View projects and tasks	17
5.5.5 Assign projects or tasks	17
5.5.6 Remove project or task	17
5.5.7 View list or user	17
5.5.8 Time booking	17
5.5.9 View schedule	18
6. Testing	18
6.1 Introduction to testing	18
6.2 Usability testing	18
6.3 Unit testing	19
6.3.1 Unit testing in Java	19
6.3.2 UI testing	19
6.4 Integration testing	19
6.5 Systems testing	19
6.6 Acceptance testing	20
7. Concluding remarks	20
Appendix A: CRC cards	21
Appendix B: Medium-fidelity mobile prototype (example)	23

1. Introduction

To overcome challenges related to an outdated paper-based time tracking routine in their organization, our client John McGuire from Pulsion Technology came to us with a set of requirements and requested us to develop a time-booking product to ease communication between an admin and a time booker. This would allow Pulsion to reduce the number of paper approvals around the office and between team members. The main advantage of this system is to make time booking quick, convenient, and easy.

Due to the nature of the project, we needed to make some assumptions. We have included them here for clarity. Firstly, we assumed that the simple nature of this system would make it permissible for us to add a few features outside of the requirements list that we felt would benefit the client. Additionally, we assumed that John would be available to act as the product owner in our Scrum group.

2. Requirements

2.1 Requirements introduction

Comprehensive and detailed requirements are vital for the success of a project. To achieve this, communication between the client and the company must be clear and concise to deliver the right result. A fixed set of 'absolute' requirements early on in a project can be restrictive. Instead, we gathered the 'must-have' features from the client with the remaining requirements presenting themselves throughout the lifecycle. Additional requirements which become apparent throughout the development process were communicated back to the client for approval.

Our team used Scrum, where face-to-face conversations are typically used to relay additional requirements back to the client in simple, non-technical language to all stakeholders. However, due to the pandemic, all meetings for this project were carried out in virtual environments (i.e. Zoom).

2.2 Moscow prioritisation

When working to a deadline it is imperative to prioritise the workload to ensure progress is steady and deadlines are met. To help with this, we used the Moscow prioritisation technique, outlined as follows.

- **Must Have:** These requirements are vital to the product. The product would be useless, unsafe, or illegal without these.
- **Should Have:** These requirements are important to the success of the project but not vital and should only be worked on once all 'must haves' are complete.

- **Could Have:** The extra desirable features set out by the client. These could be included when all other user stories are finished. They generally offer extra functionality and aesthetic.
- **Won't Have This Time:** These requirements will not be included by the initial deadline. It is important to identify these to ensure the focus is on must/should-have requirements.

2.3 Initial requirements document

The team was presented with a set of requirements from our client, John McGuire. These were vague and it became clear that extra thinking would be required to break them down. These initial requirements were grouped by their priority status as follows.

We want to develop a time booking system that can allow logged-in users to:

2.3.1 *Must-have requirements*

1. Register Projects and Tasks.
2. Have user roles (Admin or time booker).
3. Admins can assign users to projects or tasks.
4. Any assigned user can book time periods against the project or task with notes (on what was done) plus the start time and end time of the time booked.
5. Users should have to log in and only be allowed to book projects or tasks they have been assigned to.

2.3.2 *Could-have requirements*

1. The system should allow web and mobile access. A disconnected mobile app would be an interesting addition.

2.4 Requirements clarification

After receiving these requirements, a Zoom session was arranged with the client to provide more specific information. The additional requirements from this session were as follows:

2.4.1 *Must-have updates*

1. 'Project' is a parent of 'Task' and a 'Project' can have multiple Tasks.
2. Multiple administrators should be able to be involved in a project.
3. A 'Super-Administrator' should be created by the developers who will be able to allocate new administrators.
4. An administrator has the power to remove a time slot booked by a user.

2.4.2 Should-have updates

5. An administrator can book a user's time slot for them.
6. A user should get an error message when trying to book a slot without marking the time.

These additional requirements provided the much-needed detail required to move onto the next stage of the development process. The benefits of Scrum were apparent here, as by having a conversation with our client, the team were able to ask questions and extract useful information that was not apparent in our initial requirements.

2.5 Non-functional requirements

A non-functional requirement fulfils criteria not necessarily essential for a product's *primary* function. Our client identified certain security features as key non-functional requirements in additional meetings as follows:

2.5.1 Must-have security requirements

- Users should not be able to view the tasks which have been booked by other users.
- A need-to-know policy should be implemented for administrators.

The implementation of the above features is essential to protecting user privacy and to comply with GDPR. It would be a security breach if users were able to view the sensitive information of their counterparts. Also, the implementation of a need-to-know policy would ensure that any administrators were only able to access personal user information if it was necessary for their task, and only at the time of carrying out the task.

2.5.2 Usability requirements

In creating a user-focused system, usability was given a high priority. Our system implements an easy-to-use and intuitive GUI. This result was achieved by involving users throughout the development process by holding testing sessions (click tests, navigation tests) and focus groups. In gaining user insight, the chances of launching an unsuccessful system were greatly decreased.

2.6 User stories

One technique we used to refine requirements was writing user stories. A user story should describe the actions carried out to complete a task relevant to the project requirements from an end-user point of view. This was particularly useful for understanding the initial requirements document.

The team identified the following user stories:

1. As a user, I want to sign-up for a time booking system so I can use the system. **(Must Have)**
2. As a user, I want to log in so that I can register a Project. **(Must Have)**
3. As a user, I want to have an admin role so that I can manage the project. **(Must Have)**
4. As an admin, I want to log in so that I can book users on tasks. **(Must Have)**
5. As a time booker, I want to book a time slot for my project so that I can record my work, when I started and when I finished. **(Must Have.)**
6. As a user, I want to have a disconnected mobile app so that I can access the system from anywhere. **(Could Have)**
7. As a user, I want to have a meeting timetable so that I can keep track of my meetings. **(Could Have)**

User stories are what made up the main body of our project backlog, which contained the main aspects of both the business and the development process.

2.7 Backlog grooming

Backlog grooming is a powerful technique for agile teams. This process aims to prioritise the most relevant user stories to keep the product backlog organised and focused. Backlog grooming was performed regularly by the team throughout development. This exercise included:

- Removing irrelevant stories from the backlog
- Constructing new stories for new requirements
- Assigning estimates to new stories
- Reevaluating the relevance of stories

In doing this, we ensured that the backlog stayed focused on requirements. The backlog had an advantage over the 'requirements document' as it allowed for a degree of flexibility and kept us on the same page as our client. Our 'groomed' backlog always maintained the latest priorities and requirements throughout the process. For example, in regularly carrying out this activity, we identified the following 'extra' requirements to add to our backlog:

1. A 'Forgot Password' button should be included. **(Should Have)**
2. A user should be able to set a reminder for an upcoming task/project. **(Could Have)**

By keeping the client up to date with the newest aspects of the backlog throughout the development process we avoided unexpected outcomes and disastrous miscommunications.

3. Project management and lifecycle

3.1 Introduction to project management

Project management involves planning, organising and delegating a team towards an acquired goal. The goal usually has to be achieved within a specified time and has certain criteria which have to be met. The goal for Group Awe was to deliver a time booking system that meets all the users' requirements by 1st April 2021.

3.2 Gantt charts

Gantt charts are a graphical representation of tasks to be completed within a project. They are used predominantly in project management to distribute tasks within a team with specific timeframes and deadlines. They are normally created using Microsoft Project or Excel. Their main purpose is to ensure all tasks are completed within each time frame, to ensure things are running smoothly and to help the finished product to be ready in time for the product owner. Below is a Gantt chart produced of the distribution of tasks within this group along with timeframes/deadlines associated with each task.

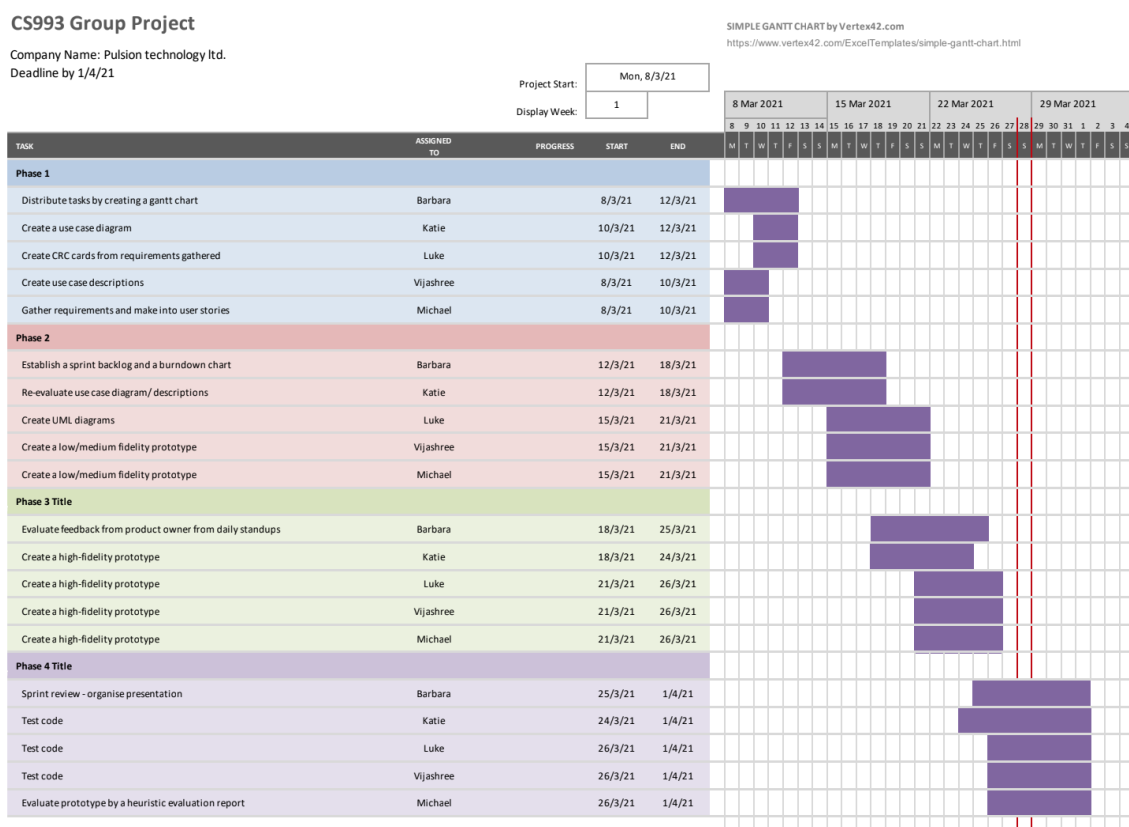


Figure 3.2

Initially, the Gantt chart was used as a rough guide to establishing timeframes and deadlines that had to be met for the product owner. But after phase 2 we realised that a lot more time

had to be focused on the design and construction of code after the requirements had been finalised, so the chart was changed to reflect this.

3.3 Software lifecycle and Scrum

The software development life cycle consists of four stages: Requirements, Design, Development and Testing. This can be followed by the use of the Waterfall model. The waterfall method is outdated and extends the period of deploying a potentially shippable product. For this reason, our group decided to use the agile methodology of Scrum instead.

The Scrum roles deployed are as follows:

- Product Owner - John McGuire
- Scrum Master - Barbara Pye
- Team - Katie Nelson, Luke Nolan, Michael Flood and Vijayshree Joshi

Scrum is an agile methodology that consists of the four key stages of the software lifecycle: plan, build, test and review in short sprints. Our sprints lasted between 1-3 weeks. These sprints allowed for all of the key stages to be worked through without each stage being held up by the last. Sprints allowed us to have a shippable product produced at the end of a sprint. In each sprint thereafter, we continued to add on and further improve on what had come before. We utilised sprints by prioritising the 'must-have' requirements first. This meant that we had the fundamentals of the system assembled and any extra, less critical requirements were added in the next sprint. This allowed us to produce a product in a very short time frame.

In Scrum, there are three main artefacts deployed. First, we created a product backlog that consisted of user stories from the product owner. A user story is a method used to gather requirements from the product owner and helps to simplify and describe the requirements from an end user's perspective. User stories ask who the user is, what they want from the product and why. Second, we established a sprint backlog. This comprises the highest priority user stories along with a time estimate. Lastly, we created a burndown chart. This chart showed the progress of tasks as they were being completed in a sprint.

Scrum ceremonies were also a crucial part of our Scrum process. These involved sprint planning, daily standups and sprint reviews. In sprint planning, the product owner, Scrum master and the team all met up and discussed the user stories and their estimated sizes. This was conducted with John McGuire through Zoom in which we gathered the initial requirements of the time booking system and created user stories from these. Daily standups are progress meetings carried out by the Scrum master and the team. They involved discussing what had been completed since the previous meeting, what we planned to do next and any issues that had arisen. Our team conducted daily standups through communication via Discord. We discussed the team's progress, issues and next steps each

morning. We also held sprint reviews, which are demos conducted at the end of a sprint. These involved the team demonstrating the completed material to the product owner and discussing any issues or any improvements that can be made.

3.4 Tools used

We created a group chat on Discord which we used to communicate daily with one another. The daily meetings, i.e. daily standups were via Discord video call. We used it to delegate tasks, arrange future meetings and bring up any new issues. We also used it to distribute links for other work we had completed such as the CRC cards and the Google Drive.

Another tool we used was Trello. Trello is a project management tool used to help teams organise and collaborate ideas remotely. Trello enabled us to share the requirements we gathered from John McGuire's guest lecture and to add them to the product backlog by creating user stories.

For creating a project report, we decided to use Google word documents through Google Drive for writing the report. This enabled us to see each other's contributions with ease as there was no need for sending large word documents. It also made uploading and submitting the project much easier as any edits were made to everyone's copy simultaneously.

4. Design

4.1 Introduction to design

In any Software Engineering project, there are different aspects of design that need to be considered in planning and building. Of these aspects, the four main areas of design to consider are architecture, high-level design, low-level design and UI design. All of these areas require different approaches to plan and implement into a system.

4.2 High-level design and architecture

Architecture is the term that is used to describe the overall structure of an application (i.e. how the front and back end will communicate with each other). There are different types of possible architectures for web apps, but the path we chose to follow was that of Model View Controller architecture. In MVC, the system is split into three distinct sections; the model (this is the backend code and data in the application, i.e Java, SQL Database), the view (this is what the end-user sees on their device through a browser, i.e. HTML, CSS, JavaScript) and the controller (Controllers are pieces of code that link the model and view together and allow data and behaviour from the model to be displayed in the view). There are different ways in which MVC can be implemented. One example that is available for Java projects is

the Spring framework. The Spring framework is an open-source collection of tools that makes it easier to build Java web apps and implement structures such as MVC throughout a project.

High-level design focuses on how different parts of the code will interact with each other. Two popular ways of planning high-level design are class diagrams and CRC cards. For this project, we used CRC cards. CRC stands for Classes, Responsibilities and Collaborators and so a CRC card is used as a way of representing the main functionality of a class and the classes that it needs to interact with to perform this functionality. To create these CRC cards, we used an online tool (<https://echeung.me/crcmaker/>) which allowed all members of the group to see and interact with each card.

User	
Admin, TimeBooker	
Responsible for representing a User on the System	Task, Project
Has Fields: Name Email Username Password	
Can Do: User can register with Email and Password Can log in with Username and Password Can Create a Task/Project	

Figure 4.2

For example, the above image shows that a User of the system can create a project, register and log in to the app. This means that the ability of the user class would be to create projects and its collaborators needed to do this would be the project and task classes. In this project, a CRC card was made for every class to help paint the overall picture of the system and how it would work by showing how each class would interact with one another. Overall, using these CRC cards allowed the overall structure of the app to be viewed in a simpler manner and in turn made the development of the app easier to plan.

4.3 Low-level design & user interface

Another important aspect of the design of an app is low-level design. The low-level design focuses on how the high-level design should be implemented. There are many ways in which

code can be written so it implements the design. However, one of the most common ways of doing this through the implementation of design patterns.

Design patterns are reusable code patterns that are used to give code certain properties/strengths, e.g. reduce code complexity or separate different parts of the logic from one another. For example, in this project, a design pattern which we implemented was the decorator pattern. This pattern made it much easier to add additional functionality to an already existing class through the use of interfaces. Other design patterns that we used were the builder and factory patterns which both aim to separate some of the complex logic about constructing objects away from the default constructors.

The final aspect of design that we considered was the User Interface (UI). As this project focused on building both a mobile and web version of the application, this meant that there needed to be a different design for each with different functionality based on the device. We used rapid prototyping and gathered user feedback to improve the UI design over several iterations. An early version of the prototype is available in Appendix B for review.

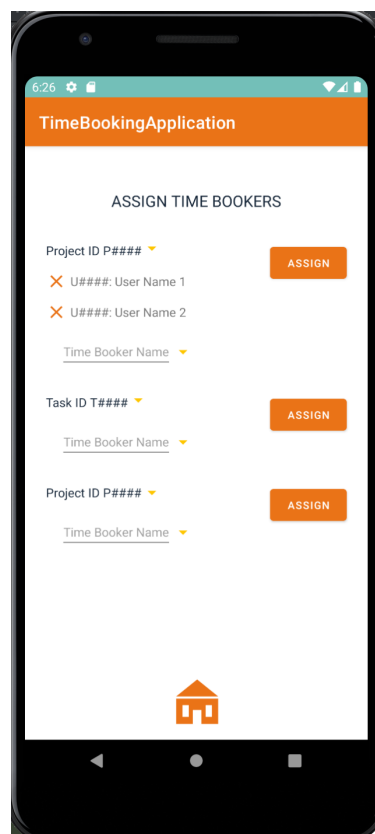


Figure 4.3 *Prototype screenshot*

The front end of a web application is generally composed of three main elements: HTML, CSS, and JavaScript (in this case by using React). Out of these three elements, CSS is the most important in defining how the app will look and JavaScript for how the app will feel. With CSS and JavaScript, we set different looks/functionality for the app based on which

type of device is accessing the app. This is known as responsive design and is important as we must consider that mobile and web apps will follow different conventions for their look (e.g. navbar vs drop-down menu). This was done by using the media properties of CSS combined with some Javascript so that smaller screens have the menu as a drop down while larger screens have it set as a navbar.

Generally, the CSS of many web pages follows the 12 column grid format where the page is divided into 12 equal columns and different amounts of column space can be allocated to different parts of the page. This is a good system as it makes it easy to divide up page space. Many different frameworks allow this design to be implemented, one of the most popular being Bootstrap. We used Bootstrap because it is free to use and provides many conventional styles for page elements making it easy for apps to follow established practices.

Overall the key bonus of using this style of design is that it allowed us to set different views based on different screen sizes (i.e mobile/web). This meant that we were able to implement different styles easily and have an app with a custom view based on the device used while still adhering to different conventions surrounding this.

5. Construction

5.1 Construction introduction

After creating the design of our application, user evaluations, a medium-fidelity prototype and revised requirements, the construction began. We followed the test-driven development (TDD) technique. To do this, we wrote a unit test first for each method and class, before any code, expecting the test to fail. We used assertion to validate different case scenarios while writing tests for functionality.

TDD works as follows:

- Write a test
- The test should fail
- Write the code
- Run the test again
- The test should pass
- Refactor
- Repeat

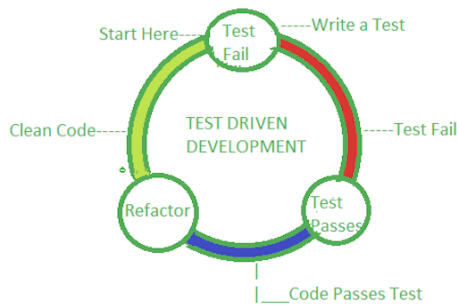


Figure 5.1

5.2 Programming languages used

In constructing the code, we used React Native for the UI so that our application works on both mobile and desktop. For the backend we used java. This backend java app interacts with the UI using RESTful API and storing data in Oracle. We have used JDBC to interact with the database from our Java backend app. This has been visualized in figure (5.2.1).

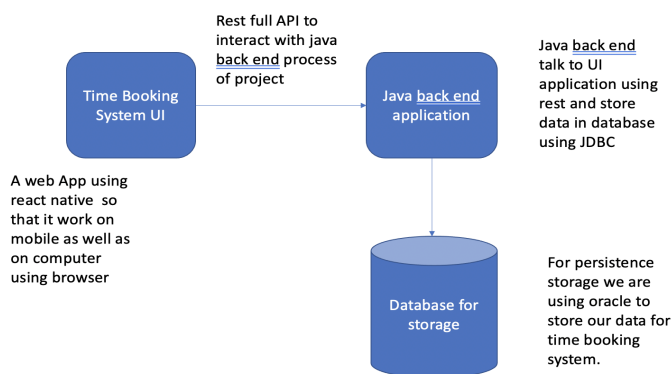


Figure 5.2.1

We used object-oriented programming (OOP) concepts while constructing our code. Inheritance (parent-child relationships) helped us to achieve reusability and reduce complexity. For example, we created one User class that included data and methods relevant to all users with the Admin and TimeBooker classes inheriting and adding data and functionality as needed. Encapsulation was achieved by creating proper classes to describe the property and behaviour of an object. Proper coding standards were used to define different classes, variables and methods. We also added comments as per java commenting standard and built JavaDoc using IntelliJ IDEA.

```

/**
 * Function to sign up for system.
 * A new member will be added to list of user/admin after validation.
 * @return sign up status message.
 */
public String signUp() {
    return "message";
}

```

Figure 5.2.2 *Commenting Style*

5.3 Framework used

For our Application construction we used the latest feature of the Spring framework, Spring Boot. This means that our application is production ready, and we don't need an additional server like Tomcat to deploy our app.

5.4 Tools used

5.4.1 Version control tools

To coordinate development between team members, we used GitHub for version control. The master branch kept a stable working copy of our code. Every developer was creating a feature branch to add code for specific features and creating a CR (change request). We performed peer reviews before merging a CR to develop the branch.

5.4.2 Build tools

We used MAVEN as a build tool, adding several dependencies, such as Spring Boot, JDBC, Jersey client, and Oracle driver for the corresponding API to be available to our code base. We also used the Maven build plugin to build our Jar artifact. For example:

```

<dependencies>
    <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-client</artifactId>
        <version>1.19.4</version>
    </dependency>
    .....
    .....
</dependencies>

```

5.5 Application Functionality

5.5.1 Sign up and login

The home screen of the application provides options to login and signup, which ensures that users or admin are registered. If the user/admin is already signed up, they can log in to the system using their ID and password. Otherwise, the signup button will navigate them to register as a time booker or admin. On the signup screen, application users need to provide details like name, address, user ID/email address, contact details and password. There is a mandatory field to choose user type. We have also implemented a 'forgot password' button to redirect users to reset their password. After each action, a pop-up message will display whether or not the action was successful.

When signing up, the details of the user are securely stored in the database and when logging in they will be validated against information stored.

5.5.2 Main screen

After logging in, the main screen displays different functionality based on user role. For each task, there is a button on the related window to navigate to the UI of the corresponding functionality.

Access control table for different functions

For Admin Role	For Time Booker Role
Add Project and Tasks	
View Projects and Task	View Project and Tasks assigned to user
Remove Project and Tasks	
Assign Project and Task to user	
View List of users	
Book time against Task or Project assigned.	Book time against Task or Project assigned.
View Schedule	View Schedule

Figure 5.5

5.5.3 Add projects and tasks

Only the admin can navigate to these screens. Here the admin has different UI components to add details about the project or task such as a description, start date, end date, etc. Tasks

can be assigned to a project from a drop-down list. When clicking 'add', details will be saved in a database with a unique project ID created by the system. There is also a profile drop-down icon on the header with options like view history (to see past activity) and a time table.

5.5.4 View projects and tasks

This function is available to both time bookers and admins, but an admin can see all of the projects and tasks in the system. Others can only see projects and tasks assigned to them. When clicking on a project or task, details about the project or task are displayed on another screen.

5.5.5 Assign projects or tasks

Only the admin can navigate to this screen. There is an option to choose a project or task from a list of projects/tasks provided in a drop-down component. Another drop down is provided to choose a user for assigning a project or task. The two buttons 'assign' and 'home' will assign users or go back to go to the home UI.

5.5.6 Remove project or task

On this screen, admins can select a project or task from a list and use the remove buttons to remove a project/task from the system. Before removing, a warning will be displayed if an admin tries to remove a project which is still in progress.

5.5.7 View list or user

This functionality is only for admins and will enable them to see how many users are in the system.

5.5.8 Time booking

This functionality is mainly for time bookers, though admins may also navigate here to input or remove time on others' behalf. A time booker will be brought to this screen when clicking on the 'add/submit' button from their main UI. On this screen, the user will only see the projects and tasks assigned to them by an admin. There is also a profile drop-down icon to help users check past projects/tasks, submission reports and offline timetables. On this screen, a calendar and clock are provided to book a time against the selected project/task. While booking, the system checks that this user is not booked on the same date/time for another project or task. Additional checks ensure that time bookers do not book a time before or after the planned dates for the project or task.

5.5.9 View schedule

Here a dashboard is provided to a time booker to see the project/tasks assigned to them and their bookings against them. For an admin, this UI shows all of the projects and tasks they have created and assigned to a user against these tasks.

6. Testing

6.1 Introduction to testing

Testing cuts across the entire software lifecycle, but is mainly discussed here. When creating our requirements documents and putting together our design, we ensured that we only agreed to requirements we knew we could test. In the development phase, we used test-driven development (TDD). Because of this, when we reached what might be considered the formal testing phase, we already had a full suite of unit tests. These tests had been run many times (using Github actions) as part of a regression testing effort and throughout the construction of new features. Finally, as we move forward into maintenance, the existing test suite helps ensure that future additions and refactoring will not alter or break existing functionality.

In our project planning, personnel resources were allocated so that some unit and integration testing was carried out by someone other than the original code writer. This reduced bias in testing and allowed the tester to use more of a black-box approach (especially in the integration testing phase) as the tester would lack intimate knowledge of what was in a function or unit. Testing efforts paid particular attention to edge cases and proper exception handling was input to manage runtime errors not caught by the compiler.

6.2 Usability testing

Using the prototypes created during the design phase, our team held several rounds of rapid prototyping usability tests. We used a medium-fidelity, partially-clickable throwaway prototype that we built in Android Studio to go through several rounds of user interface testing with users from Pulsion in think-aloud style interviews. At each iteration, we updated the prototype according to user feedback. For instance, after the prototype we included in the index was created, user feedback encouraged us to incorporate a drop-down navigation feature for smaller screens and a bottom navbar for larger screens to improve user experience.

After the basic functionality was built out for the web and mobile apps, several rounds of alpha testing were also carried out across the team, running through various user-story-based scenarios to work out flaws in the UI and to double-check that core

functionality was being properly implemented. We also had a usability heuristics report drawn up and implemented fixes for all major to moderate violations.

6.3 Unit testing

6.3.1 Unit testing in Java

The vast majority of our unit test suite was built out during the construction phase. As JUnit is the gold standard of testing in Java, all of our unit tests were written in JUnit 5. We chose JUnit 5 as this software build is a new project, without a legacy testing suite already existing in JUnit4. Because of this, there was no reason not to use the latest tools available from JUnit. It was also an opportunity for the team to increase our familiarity with these newer tools.

We occasionally used stubs and doubles to test our units. For instance, when another connected unit was still under construction by another developer. We also used Mockito to create a mock time generator that allowed us to test all different times of day in the application. This helped us to make sure that the app would work properly even at times that would have been difficult to test in other ways, such as in the middle of the night, or at the turning of the new year.

6.3.2 UI testing

Some basic manual UI testing was carried out in an ongoing fashion during the construction phase to ensure UI tools were working. For UI testing of the mobile app, the team used a range of physical and emulated devices to test usability and visibility on different screen sizes. We also created an automated test suite for the UI using Selenium and Selendroid to get more frequent and robust data on our UI as we built the system.

6.4 Integration testing

In integration testing, we began to test our units together with a specific emphasis on data transfer between units. For example, for much of the early unit testing, we were using stubs for our Task class, as it was built after the Project class was built. This allowed us to adhere to TDD even when some related units were not yet built. In this phase of testing, we did away with our stubs and doubles to check that units were communicating properly. For this, we used the big-bang approach, chosen for simplicity due to the relatively small size of this system.

6.5 Systems testing

In systems testing, we frequently referred back to our requirements to ensure that all 'Must' and 'Should' requirements were fulfilled before turning any piece of software over to the

end users for acceptance testing. This allowed us to validate that we had created the right software according to the carefully-documented requirements. We also used a numbering system to track where each piece of required functionality was fulfilled (along with associated tests) to help us stay organised and validate that we had built the right thing.

6.6 Acceptance testing

When all of the above testing was complete for each sprint, we organised our user acceptance testing (UAT) by designing a document full of test cases that our UAT would cover. This document relied on the user stories that had been previously created for this project. We then coordinated with John and Pulsion to put together a testing team from the user base. This user group ran through the test cases and user stories and detailed notes were taken regarding the feedback. During the UAT for each sprint, bugs that needed to be added to the backlog for future sprints were noted. In the final round of UAT, the client did a final sign off once it was shown that we had met all requirements.

7. Concluding remarks

In the end, the software was presented promptly and our client John was satisfied with the quality of the work. The project has been put into maintenance where we will track bugs and work with the client and his team to prioritise bug fixes. All bugs will be tracked using Jira and fixed according to the level of priority. The agreement with the client is that the majority of support will be happening during normal business hours, with outside-hours contact information being made available for emergencies.

Appendix A: CRC cards

User	
Admin, TimeBooker	
Responsible for representing a User on the System	Task, Project
Has Fields: Name Email Username Password	
Can Do: User can register with Email and Password Can log in with Username and Password Can Create a Task/Project	

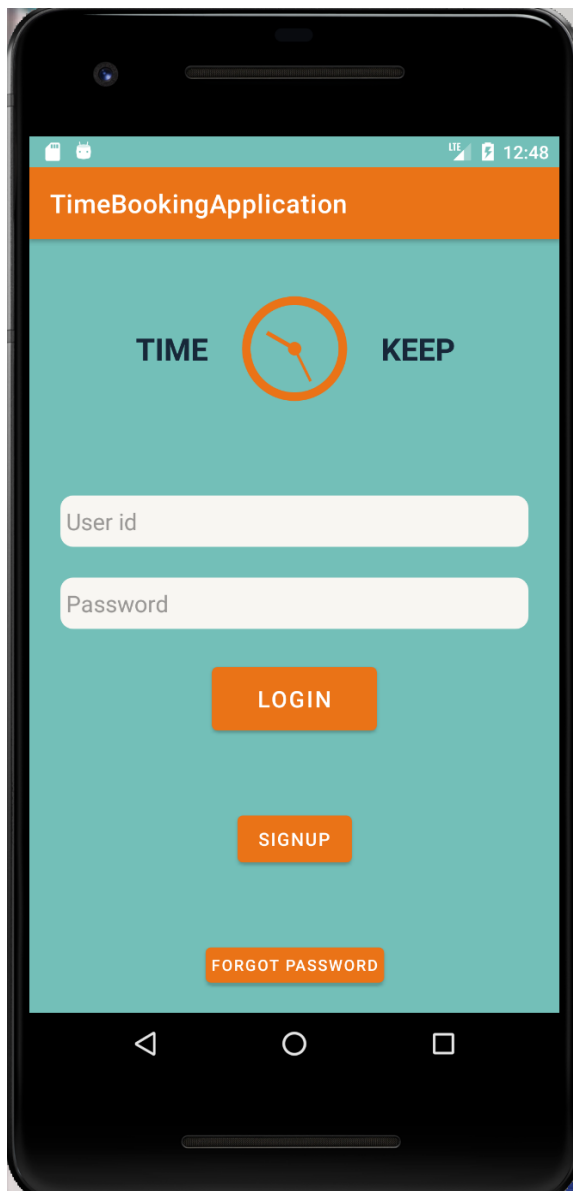
Admin	
User	
Represents a User who has the ability to create projects and tasks as well as assign TimeBookers	Task, Project, User, TimeBooker
Has Fields: Same as User	
Can Do: Can Add/Remove Users From Project Can set a User to be a TimeBooker Manage the Tasks on a Project	

TimeBooker		User
Represents a User who has the ability to book times for tasks Has Fields: Same as User Can Do: Book a Time against a Task		Task, Project, User

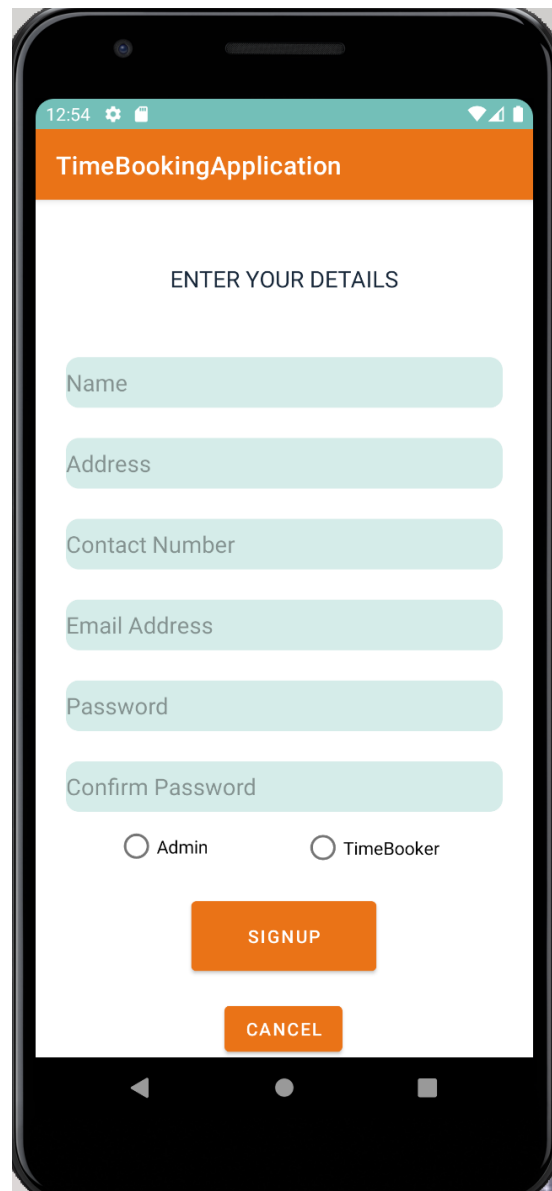
Project	
Represents overall Project in the system Fields: List Of Tasks Start/End Date List of Users working on a project Project Admin Title References Can Do:	Tasks, Users, Project

Task	
Represents an individual task within a project set by User or Admin Fields: Start/End Time Users Name Importance Can Do:	User, Admin, Project, TimeBooker

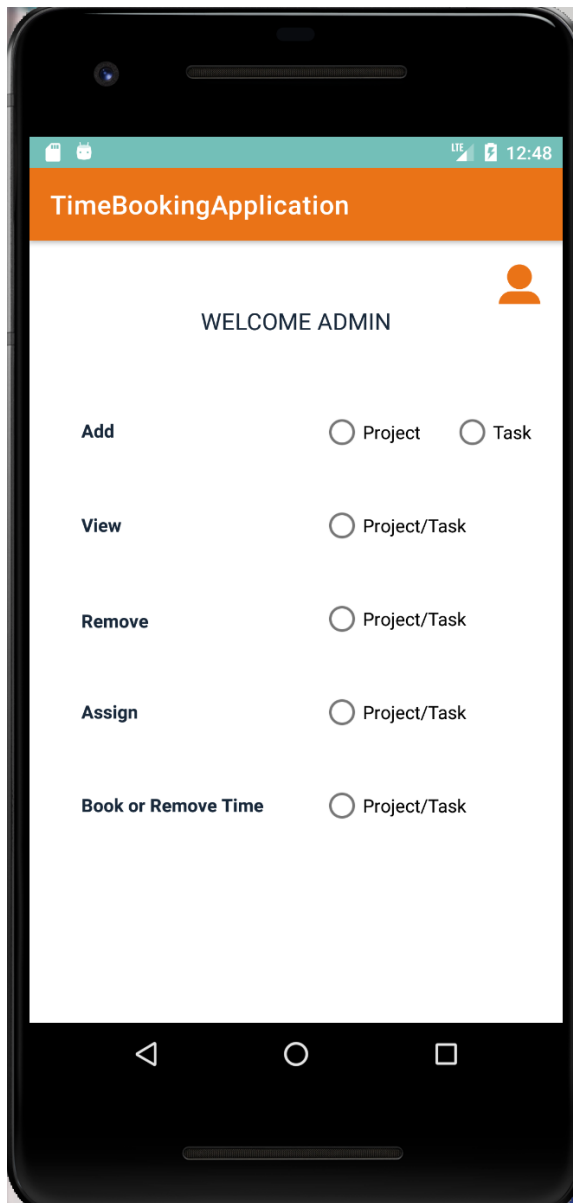
Appendix B: Medium-fidelity mobile prototype (example)



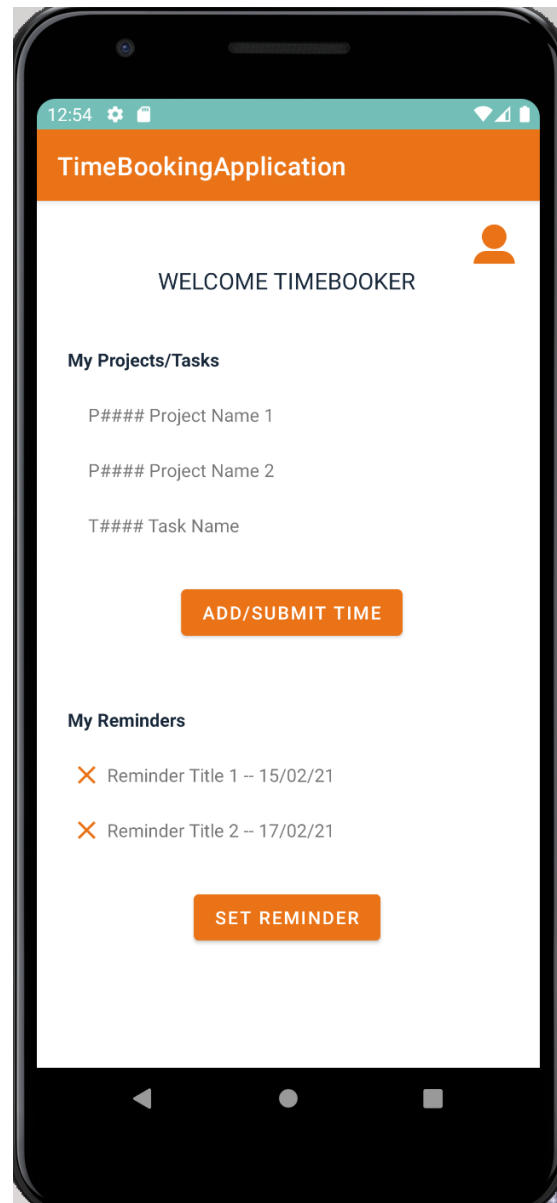
Login page



Sign up page



Main screen (Admin)



Main screen (Time booker)

12:52 12:52 12:52

TimeBookingApplication

ADD PROJECT

Project Id: P####

Project Name


Due Date

Client

Description

ADD

CANCEL



Add projects

12:52 12:52 12:52

TimeBookingApplication

ADD TASK

Task ID: T####

Task Name


Due Date

Description

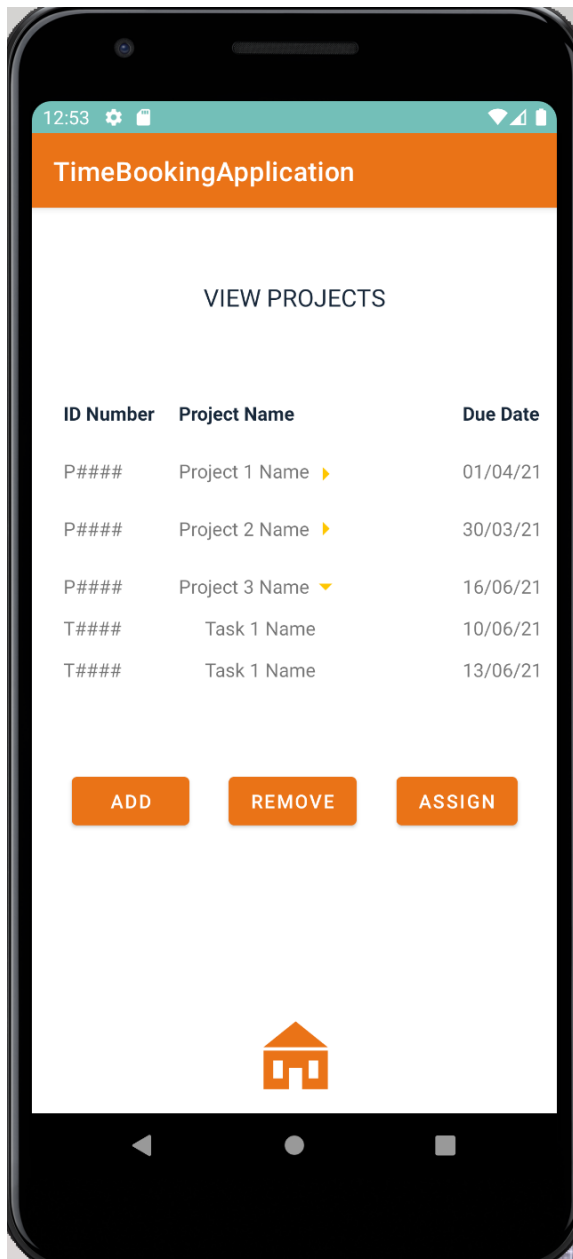
Parent Project

ADD

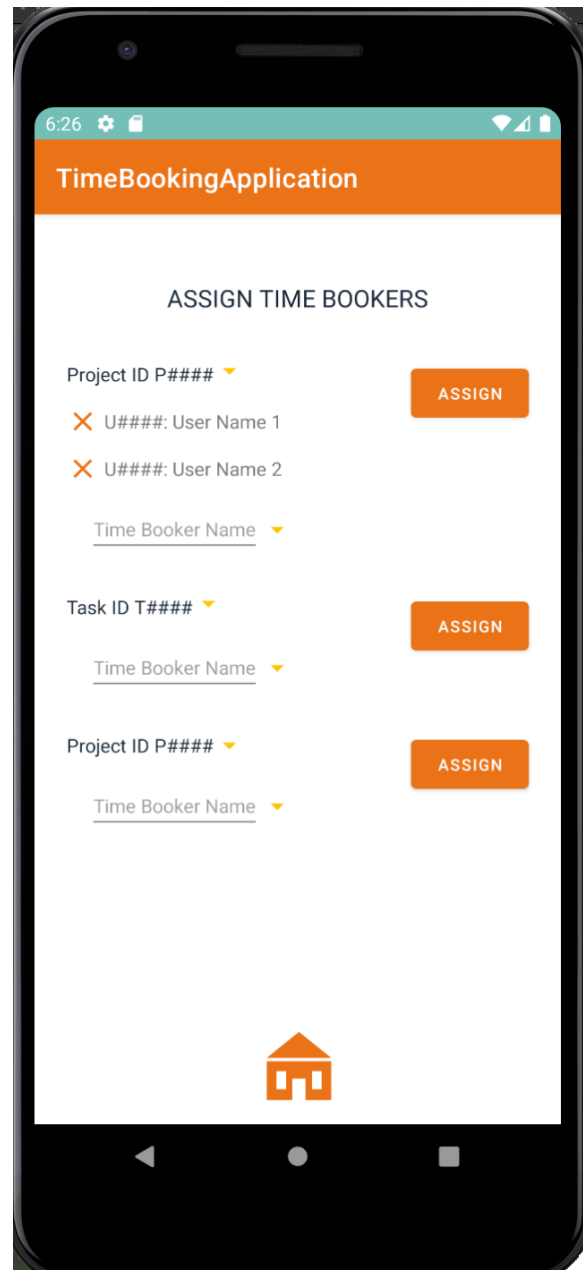
CANCEL



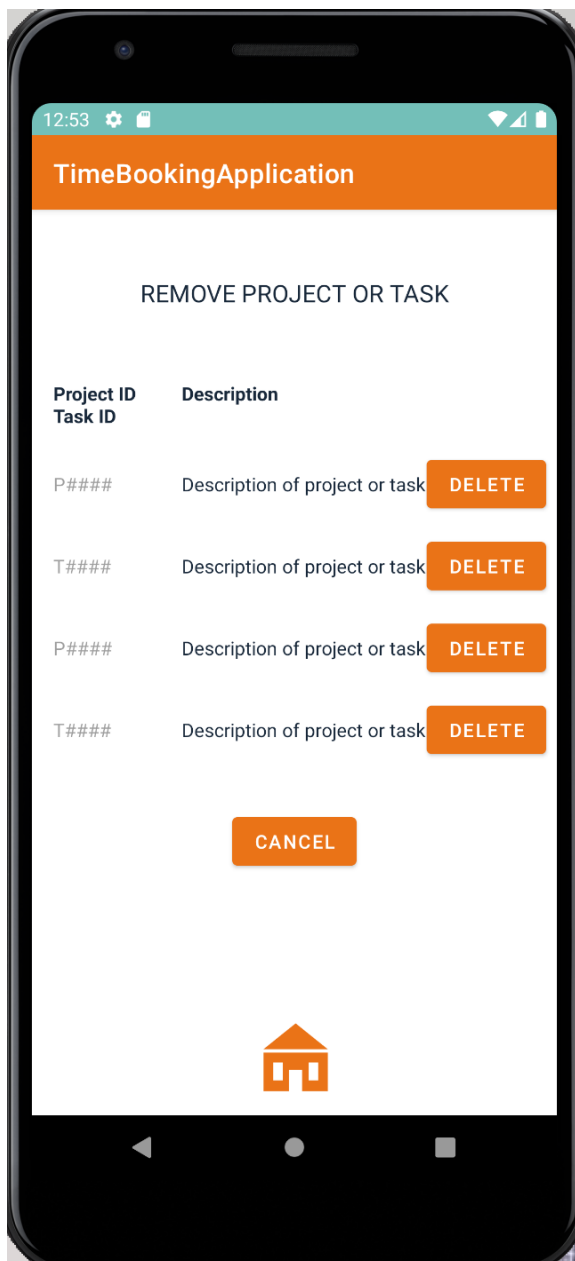
Add tasks



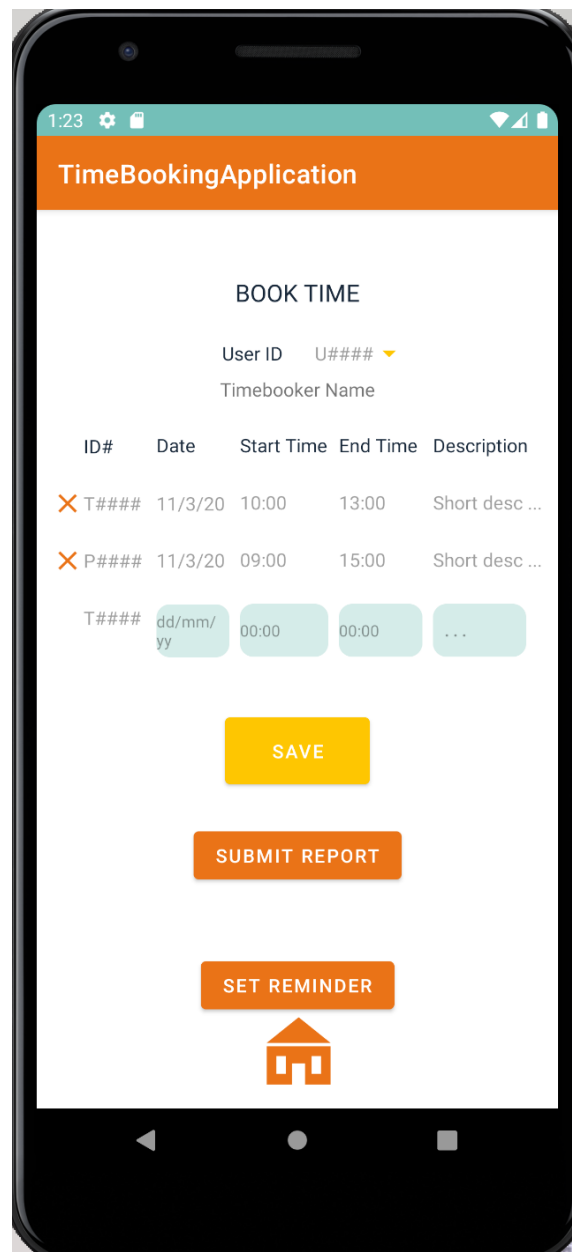
View projects/tasks



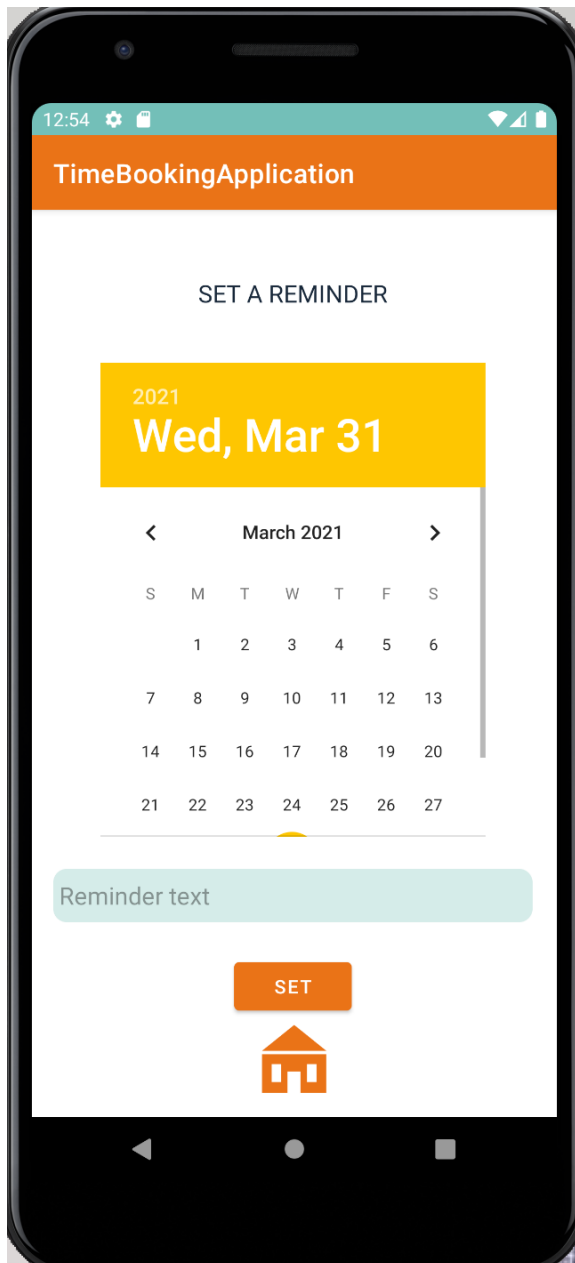
Assign projects/tasks



Remove project/task



Time booking



Set reminder