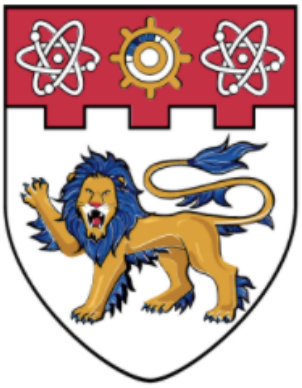


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering, SCSE

SC2006 Software Engineering

Project: MoveInLo!

SCSX Group 3 Code Crafters

Lab Instructor : Zheng Long Tao

Done by :	**Matriculation Number**
SCSE EUGENE WEE JUN LIN	U2120698A
SCSE IAIN RODERICK TAY RONG YU	U2221382K
SCSE JOSEPH TEO CHEE HOE	U2223041J
SCSE JIANG YIFEI	U2201092F
SCSE KIM HYUN BIN	U2020316D

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Table of Content

1. Requirements **4**
- **1.1. Introduction** **4**
  - 1.1.1. Purpose/ Mission Statement 4
  - 1.1.2. Document Conventions 4
  - 1.1.3. Intended Audience and Reading Suggestions 5
  - 1.1.4. Product Scope 6
- **1.2. Overall Description** **6**
  - 1.2.1. Project Perspective 6
  - 1.2.2. Product Functions 6
  - 1.2.3. User Classes and Characteristics 7
  - 1.2.4. Operating Environment 8
  - 1.2.5. Design & Implementation Constraints 8
  - 1.2.6. User Documentation 8
  - 1.2.7. Assumptions and Dependencies 9
- **1.3. External Interface Requirements** **9**
  - 1.3.1. User Interfaces 9
  - 1.3.2. Hardware Interfaces 11
  - 1.3.3. Software Interfaces 11
  - 1.3.4. Communications Interfaces 11
- **1.4. Functional Requirements** **11**
  - 1.4.1. User Authentication 11
  - 1.4.2. Student Moving Services 12
  - 1.4.3. Job Listings 14
- **1.5. Non-functional Requirements** **15**
  - 1.5.1. System Responsiveness 15
  - 1.5.2. Capacity 15
  - 1.5.3. System Applicable Platforms 16
  - 1.5.4. Safety Requirements 16
  - 1.5.5. Security Requirements 16
  - 1.5.6. Software Quality Attributes 17
  - 1.5.7. Business Rules 17
- **1.6. Use Case Diagram** **19**
- **1.7. Use Case Descriptions** **20**
- **1.8. Data Dictionary** **46**
- **1.9. UI Mockups** **48**
  - 1.9.1. Landing Page & PDPA Notice 48
  - 1.9.2. Login Page & Sign up Page 49
  - 1.9.3. Schedule Moving Service 50
  - 1.9.4. Job Seeker Home Page & Progress Tracker 51
  - 1.9.5. Job Listings & View Job 52

1.9.6. Registered Job & Withdraw Job 53
1.9.7. Registered Job Information & Registered Job Lists 54
2. System Design 55
2.1. Class Diagrams 55
2.2. Sequence Diagrams 57
2.3. System Architecture 60
2.3.1. Model-View-Controller (MVC) 60
2.3.2. Client-Server Architecture 60
2.4. Design Patterns 61
2.4.1. Creational Pattern 61
2.4.2. Structural Pattern 62
2.4.3. Behavioural Pattern 62
3. Implementation 65
3.1. Application Skeleton 65
3.2. Source Code 65
3.3. Demo Script 65
3.4. Demo Video 67
4. Testing 68
4.1. Overview 68
4.2. Black-Box Testing 69
4.3. White-Box Testing 71
5. References 73
6. Appendix 74

1. Requirements

1.1. Introduction

1.1.1. Purpose/ Mission Statement

This documents the software requirements and specifications for MoveInLo. MoveInLo is a mobile application that seeks to smoothen the process of moving in and out of university halls/ hostels for new and existing students in Singapore. MoveInLo achieves this by providing a seamless experience for moving in/ out as well as a new avenue for students to seek temporary jobs.

This app can be installed from the Android or IOS App store. This document will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system.

1.1.2. Document Conventions

For documentation purposes, we will abide by the following conventions:

a. **Font**: Times New Roman
b. **Font Size**s:
    i. **Heading 1 (18px Bold)**
    ii. **Heading 2 (14px, Bold)**
    iii. **Heading 3 (12px, Bold)**
    iv. Body (12 px, Normal)
c. **Line Height**: 1.5 spacing
    i. **Spacing:** Before and after each heading

1.1.3. Intended Audience and Reading Suggestions

The document is intended for all the stakeholder customers and developers — designers, coders, testers and maintainers. This document need not be read sequentially; users are encouraged to jump to any section they find relevant. Below is a brief overview for each part of the document.

Part 1 (Requirements)

This section offers a summary of the MoveInLo project, including goals, objectives, project scope, general system details and key stakeholders.

Part 2 (Design)

This section describes MoveInLo system class by class, including interface details, class hierarchies, performance/ design constraints and process details.

Part 3 (Implementation)

This section covers all of the details related to the structure of the graphical user interface (GUI), including some preliminary mockups of our MoveInLo mobile application. Readers can view this section for a tentative glimpse of what the final product will look like.

Part 4 (Testing)

This section offers a list of test cases, testing and expected output, separated into black box and white box testing. Readers can use this section to evaluate the completeness and reliability of the product.

Part 5 (References)

This section includes all the documentation and resources referenced and used in our project MoveInLo.

Part 6 (Appendices)

This section includes all UML Diagrams, links to mockups, source code, demo video and other deliverables.

1.1.4. Product Scope

MoveInLo is a mobile application which serves to ease the process of moving in/ out of hall residents for students in Singapore. It aims to achieve this through this key features:

a. Seamless moving services/ temporary storage facility of hall-related items.
b. Temporary job opportunities as logistics deliverers for students.

1.2. Overall Description

1.2.1. Project Perspective

MoveInLo was inspired by one of the biggest woes of university students during the hall life. The hassle of moving in/ out of halls during the start/ end of each academic term.

The inspiration for the name "MoveInLo" was drawn from the commonly-used term in Singapore national service, "ORDLo" which signifies the end of a significant milestone, yet start of a new journey.

1.2.2. Product Functions

1.2.2.1. Moving In/ Out Services

Users can schedule for moving services during any of the given periods indicated below:

a. **Moving In**: At the start of the academic term, items will be collected and delivered to the indicated locations within the same day.
b. **Moving Out**: At the end of the academic term. Users can opt for these options:

i. Temporary Storage: Items will be stored in our storage facility, and delivered back to their hall room, at the start of the academic term.

ii. Home Delivery: Items will be delivered back to their preferred address.

Storage boxes will be provided to users to store their items for collection & delivery.

1.2.2.2. Part-Time Job Opportunities

This mobile application will serve as a platform for university students to find temporary job opportunities as logistics movers (i.e. delivery, driver) to earn additional cash, to aid them ease the financial burdens of university studies.

1.2.3. User Classes and Characteristics

This section summarises the key user classes and their respective characteristics that will use our mobile application. User classes indicated with red asterisk (*) represents our important manpower personnels, which need to be satisfied in order for the operation to commence.

a. Hall Residents (Customer)

This includes university students who have been allocated housing within their campus, and are currently staying in their assigned housing.

b. Student Job Seekers *

It includes university students who want to earn additional cash, by assisting with the following roles:

i. Drivers. Requires a valid Class 3/ 3A driving licence.

ii. Logistics Carriers. Able to handle and carry heavy loads (at least 15kg).

Individuals in this section will mainly be operating with 2.2.1 Moving In/Out Services.

1.2.4. Operating Environment

Our mobile application will operate on the IOS and Android platforms, and will require location services.

a. For Android platforms, the OS version must be at least Android 6 (Marshmallow).
b. For IOS platforms, the OS version must be at least IOS 9.
c. For location services, the device must have built-in support of Global Positioning System.

1.2.5. Design & Implementation Constraints

This section specifies the standards/ constraints which we abide to when designing and implementing our mobile application:

a. **Frontend Interface (User Interface, UI)**: React Native version 0.72
b. **Backend**: Golang/ ExpressJS
c. **API**: GoogleMapsAPI
d. **Android**: at least Android 6 (Marshmallow)
e. **IOS**: at least IOS 9

1.2.6. User Documentation

Our mobile application comes with the following user documentation guide to help new users get started:

a. A YouTube video showcasing the core functionalities of the mobile application.
b. A Quick Start guide documenting the steps to get started using the application.

1.2.7. Assumptions and Dependencies

This section goes in-depth on the assumed factors and dependencies from third-party components which the mobile application has used.

a. GoogleMaps API requests is limited to
    i. Geolocation: 6000 queries per minute
    ii. Static Maps: 30,000 queries per min
b. The user's device has in-built support for the Global Positioning System.

1.3. External Interface Requirements

1.3.1. User Interfaces

1.3.1.1. UI Design

The design of the user interface should have a pop up to allow users to select the date and time for the service they have selected.

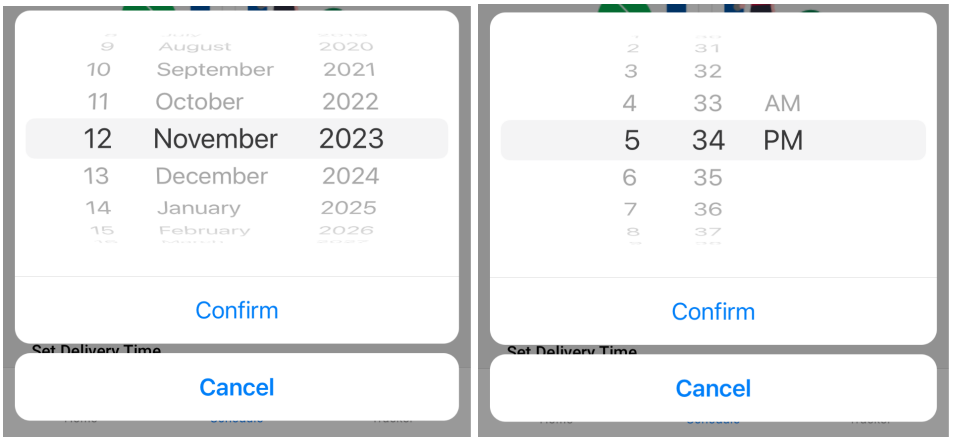


Figure 1: User Selection UI

1.3.1.2. Top Navigation Bar

The top navigation bar should have a back button to allow the user to go to the previous screen upon clicking on the button.

Figure 2: Top navigation bar

1.3.1.3. Bottom Navigation Bar

The application must have a navigation bar at the bottom of the screen once the user is logged in to allow them to switch between different services in the mobile application.

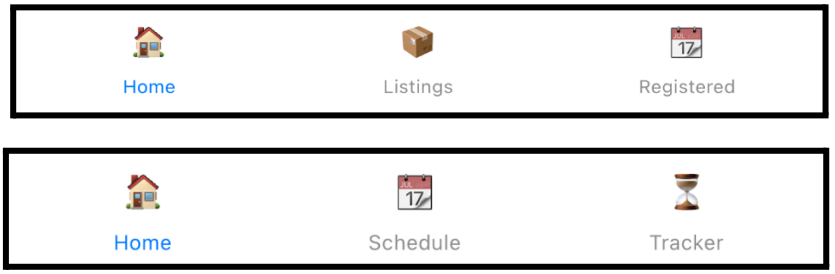


Figure 3: Bottom navigation bar

1.3.1.4. Invalid input with error message

The application must display an error message when there is an invalid input provided by the user.

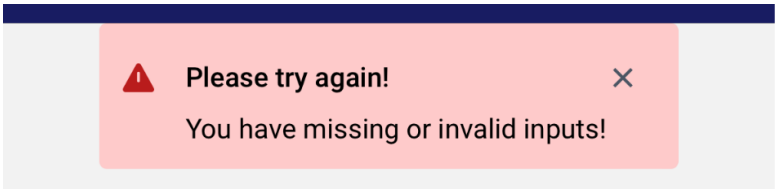


Figure 3: Error message

1.3.2. Hardware Interfaces

a. The device used to open this application must support Global Positioning System for the tracking of current user's location.

1.3.3. Software Interfaces

a. For Android devices, the OS version must be at least Android 6 (Marshmallow).
b. For IOS devices, the OS version must be at least IOS 9.

1.3.4. Communications Interfaces

a. The application requires internet access.
b. The application requires access to Global Positioning System (GPS) for tracking the current user's location.

1.4. Functional Requirements

1.4.1. User Authentication

[REQ-1] Registration as Customer

The application must allow the current user to sign up for a customer account if there is no account associated with their email address.

[REQ-2] Registration as Job Seeker

The application must allow the current user to sign up for a job seeker account if there is no account associated with their email address.

[REQ-3] Login as Customer

The application must allow the current user to login to their account as a customer using their login credentials consisting of email address and password.

[REQ-4] Login as Job Seeker

The application must allow the current user to login to their account as a job seeker using their login credentials consisting of email address and password.

[REQ-5] Forget Password

The application must allow the current user to reset their password when they provide their email address.

[REQ-6] Successful Login

The application must allow the authenticated user to access their designated home page (i.e. Customer access Customer Home Page).

1.4.2. Student Moving Services

[REQ-7] Selecting type of moving services

The application must allow the customer to select between 2 options for the moving services, moving in or moving out.

[REQ-8] Scheduling Move In services

If the customer selects the "Move In" option, the application must allow the customer to schedule a service for moving in.

[REQ-9] Scheduling Move Out services

If the customer selects the "Move Out" option, the application must allow the customer to schedule a service for moving out.

[REQ-10] Providing location for item delivery

When the customer is scheduling a moving service, the application must request the customer to provide a valid 6-digit postal code, SXXXXX within Singapore for the item delivery.

[REQ-11] Providing date for item delivery

When the customer is scheduling a moving service, the application must request the customer to select an available date provided by our system.

[REQ-12] Providing time of item delivery

When the customer is scheduling a moving service, the application must request the customer to provide a date in HH:MM 24-hour format.

[REQ-13] Display Progress Tracker

When the customer has successfully scheduled a moving service, the application must display a progress tracker to the customer in the progress tracking page.

[REQ-14] Live Location Tracking

When the user is at the progress tracker, the application must display live location tracking using GoogleMapsAPI.

[REQ-15] Update Progress Tracker for Scheduled

When the customer has successfully scheduled the moving service, the application must update the progress tracker to display “Scheduled” in the progress tracking page.

[REQ-16] Update Progress Tracker for Delivering

When the job seeker has successfully collected the items for the moving service, the application must update the progress tracker to display “In Progress” in the progress tracking page.

[REQ-17] Update Progress Tracker for Delivered

When the job seeker has successfully delivered the items for the moving service, the application must update the progress tracker to display “Delivered” in the progress tracking page.

[REQ-18] Cancellation of Moving Service

If the current user has scheduled a moving service, the application must allow the user to cancel the service.

1.4.3. Job Listings

[REQ-19] Job Seeker Guide

If the current user logged in is a job seeker, the application must display a guide to the Job Seeker, showing step-by-step instructions.

[REQ-20] Available Job Listings

The application must display a list of available jobs to the current job seeker in the Job Listings page.

[REQ-21] Moving Service Job Listing Information

When a job seeker clicks on a moving service job listing in the Job Listings page, the application must display information on the date/ time/ location of pickup and delivery.

[REQ-22] Job Registration

The application must allow the current job seeker to register for an available job listed in the Job Listings page.

[REQ-23] Registered Jobs Listings

If the current job seeker has successfully registered for jobs, the application must display a list of registered jobs to the job seeker in their Registered Jobs page.

[REQ-24] Job Completion

If the current job seeker has successfully completed a job, the application must allow the job seeker to indicate its completion.

[REQ-25] Job Payment

When a job has been completed, the application must allow the user to indicate whether payment has been completed.

[REQ-26] Job Withdrawal

If the current job seeker has successfully registered for a job, the application must allow the job seeker to withdraw from the registered job.

1.5. Non-functional Requirements

1.5.1. System Responsiveness

[REQ-1] Registration and Login Responsiveness

All registration and login processes for customers and job seekers must have a response time of less than 2 seconds under normal system load to ensure a seamless user experience.

[REQ-3] Schedule Tracking Responsiveness

The schedule tracking page should load within 3 seconds to provide real-time updates to users without delays.

[REQ-4] Job Listing Retrieval Responsiveness

Retrieving and displaying job listings should take less than 2 seconds to ensure a smooth experience for job seekers.

[REQ-5] Job Registration and Withdrawal Responsiveness

Processes related to job registration and job withdrawal should have a response time of less than 3 seconds.

1.5.2. Capacity

[REQ-6] Users Capacity

The system must be able to support 1000 simultaneous users.

[REQ-7] Database Capacity

The database should be able to store up to 50,000 user activities, each mapped to a unique user.

1.5.3. System Applicable Platforms

[REQ-8] Available Download Platforms

This application can be installed from the Android or IOS App store.

[REQ-9] Available Running Operating Systems

This application can run on Android and IOS platforms.

1.5.4. Safety Requirements

1.5.4.1. Age Limit

[REQ-10] Age restriction

Users aged under 16 will not be able to register for an account.

1.5.5. Security Requirements

1.5.5.1. Data Privacy and Security

[REQ-11] Data Encryption

All user data must be encrypted during transmission using industry standard encryption protocols to prevent unauthorised access.

[REQ-12] Data privacy

The system must comply with data privacy regulations (e.g. PDPA) to protect user data, including personal information and transaction history. User consent for data handling must be obtained and documented.

[REQ-13] User Reminder for Security

The system must remind the users not to leak their personal information, username, and password.

1.5.5.2. User Account Policies

[REQ-14] Password Policies

The system should enforce the following password policies, including complexity requirements. The password must include the use of both upper-case and lower-case letters, one or more numerical digits and special characters, such as @, #, $.

[REQ-15] User Credentials

Users are required to login with their email and password.

1.5.6. Software Quality Attributes

1.5.6.1. Adaptability

[REQ-16] Adaptability Across Platforms

The system will be able to operate on both IOS and Android platforms.

1.5.6.2. Reliability

[REQ-17] System Reliability

The system must be reliable, with a target mean time between failures of at least 10,000 hours.

1.5.6.3. Maintainability

[REQ-18] Code maintainability

The system code must be provided with necessary comments by developers to make it easy to understand.

1.5.7. Business Rules

1.5.7.1. Developer

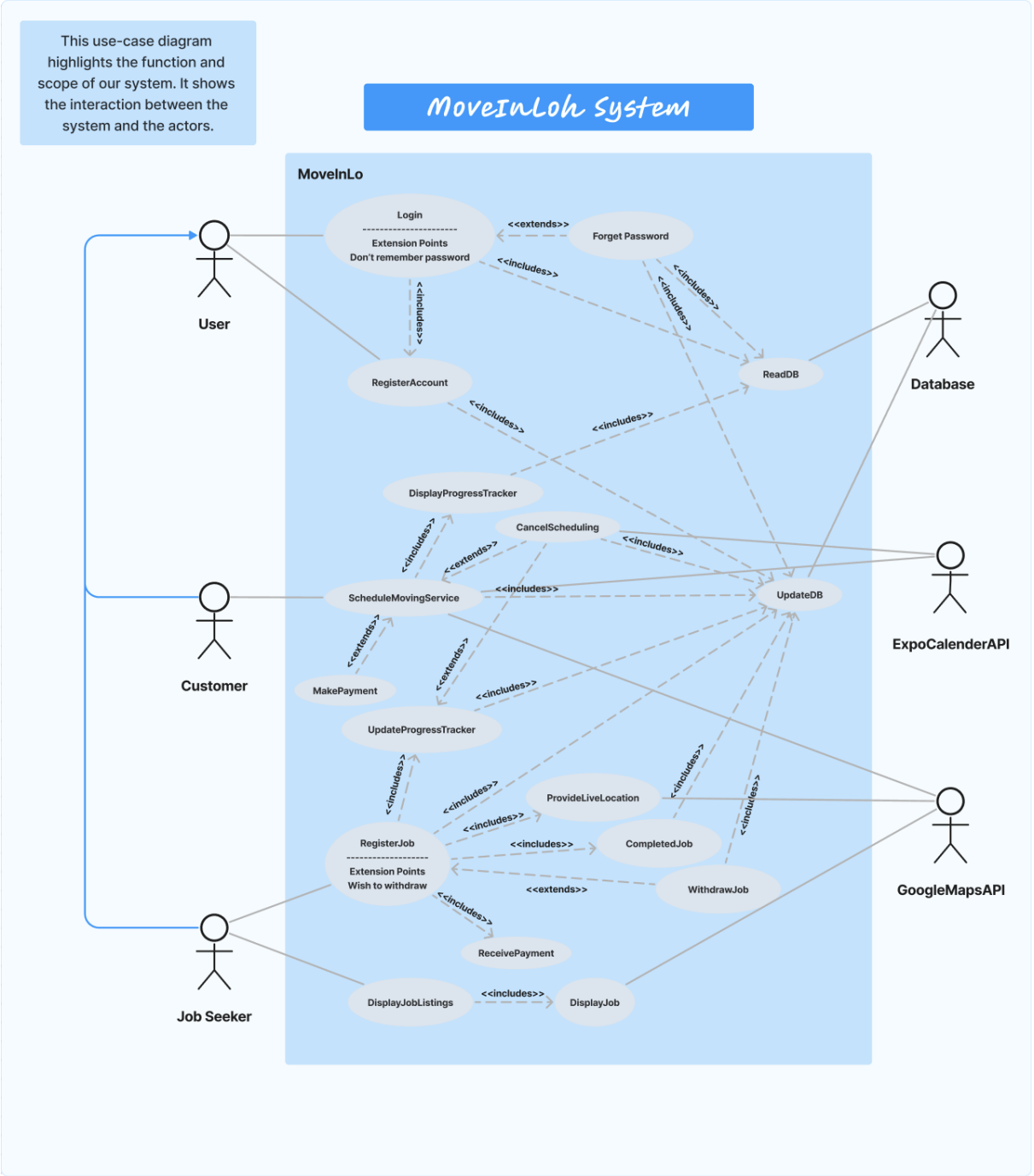
[REQ-19] User Removal

Developers can remove users not following the community guidelines.

[REQ-20] Activity Removal

Developers can remove activities not following the community guidelines.

1.6. Use Case Diagram



1.7. Use Case Descriptions

Use Case ID:	A0001		
Use Case Name:	Registration as Customer		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	User
Description:	The user creates a customer account by filling in an email address, username, account type, contact number, age, password.
Preconditions:	User clicks on Register in the Home Page.
Postconditions:	User's customer account successfully created.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The user presses the "Sign up" button in the landing page. 2. The user fills in the required information: email address, username, contact number, age, password. 3. The user indicates the account type as "Customer". 4. The user presses the "Sign up" button. 5. App validates the information and sends it to the database. 6. App redirects to the home page UI. 7. Database sends a response and the app displays the message "Account successfully created."
Alternative Flows:	4. App finds the information invalid. 5. Back to step 2.
Exceptions:	The user already has a customer account.
Includes:	-
Special Requirements:	App must be able to check information validity within 2 seconds
Assumptions:	-
Notes and Issues:	The user must input a valid email address that has '@' and '.com'. The user must input a valid password that has at least 8 characters, 1 special character, 1 capital letter, 1 numerical value.

Use Case ID:	A0002		
Use Case Name:	Registration as Job Seeker		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	User
Description:	The user creates a job seeker account by filling in information for email address and password.
Preconditions:	-
Postconditions:	User's job seeker account successfully created.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The user presses the "Sign Up" button. 2. The user fills in the required information: email address, username, contact number, age, password. 3. The user indicates the account type as "Job Seeker". 4. The user presses the "Sign up" button. 5. App validates the information and sends it to the database. 6. App redirects to the home page UI. 7. Database sends a response and the app displays the message "Account successfully created."
Alternative Flows:	4. App finds the information invalid. 5. Back to step 2.
Exceptions:	The user already has a job seeker account.
Includes:	-
Special Requirements:	App must be able to check information validity within 2 seconds
Assumptions:	-
Notes and Issues:	The user must input a valid email address that has '@' and '.com'. The user must input a valid password that has at least 8 characters, 1 special character, 1 capital letter, 1 numerical value.

Use Case ID:	A0003		
Use Case Name:	Customer Login		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer logs into the app using their registered email and password.
Preconditions:	Valid email and password.
Postconditions:	The customer's account is successfully logged in.
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer clicks the "Login" button on the Landing Page. 2. The customer fills in their registered email and password. 3. The customer selects "Customer" for account type. 4. The customer presses the "Login" button. 5. App authenticates the email and password. 6. The customer logs on to their account's home page.
Alternative Flows:	7. App finds credentials are invalid. 8. App displays the error message. 9. Back to step 2.
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to check information validity within 2 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	A0004		
Use Case Name:	Job Seeker Login		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker logs into the app using their registered email and password.
Preconditions:	Valid email and password.
Postconditions:	The job seeker's account is successfully logged in.
Priority:	-
Frequency of Use:	By the intention of the job seeker.
Flow of Events:	1. The customer clicks the "Login" button on the Landing Page. 2. The customer fills in their registered email and password. 3. The customer selects "Job Seeker" for account type. 4. The customer presses the "Login" button. 5. App authenticates the email and password. 6. The customer logs on to their account's home page.
Alternative Flows:	7. App finds credentials are invalid. 8. App displays the error message. 9. Back to step 2.
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to check information validity within 2 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	A0005		
Use Case Name:	Forget Password		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	User
Description:	On the login page, if the user does not remember their password, they can reset it through email.
Preconditions:	The user has an account.
Postconditions:	The user's current password is changed to the new password.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The user presses the "Forget Password" button. 2. App verifies email and the user presses on "Change Password" button. 3. The user is prompted to enter a new password. 4. The user enters a new password. 5. App checks if the new password is valid. 6. App replaces the old password with the new one in the database. 7. Password is reset.
Alternative Flows:	7. App checks that the password is invalid. 8. Back to step 5.
Exceptions:	1. The user's email is not linked to any valid account. 2. The user does not have access to his/her own email.
Includes:	-
Special Requirements:	-
Assumptions:	The email belongs to the user's account.
Notes and Issues:	-

Use Case ID:	A0006		
Use Case Name:	Successful Login		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	12th Sep 2023	Date Last Updated:	12 Sep 2023

Actor:	User
Description:	The application must allow the authenticated user to access their designated home page
Preconditions:	The user has been authenticated during the login.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The user is authenticated during successful login or registration. 2. If the user logs in as “Customer”, the user is redirected to the Customer Home Page.
Alternative Flows:	2. If the user logs in as “Job Seeker”, the user is redirected to the Job Seeker Home Page.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	The user account has successfully been authenticated.
Notes and Issues:	-

Use Case ID:	M0007		
Use Case Name:	Selection of moving services		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to select two options of moving services.
Preconditions:	The customer has logged in.
Postconditions:	The customer has selected what moving service he/she wants.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The customer presses the “Schedule” button. 2. The customer presses the “Moving In” if he/she wants to move in and the “Moving Out” if he/she wants to move out.
Alternative Flows:	3. The customer presses the “Back” button as he/she changes his/her mind.
Exceptions:	-
Includes:	-
Special Requirements:	App is able to show the moving services within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0008		
Use Case Name:	Schedule move in services		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to schedule a service to move in.
Preconditions:	The customer has selected a move in service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer presses the “Move In” button. 2. The customer is able to select the date and timing of the moving services.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to schedule the move in service within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0009		
Use Case Name:	Schedule move out services		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to schedule a service to move out.
Preconditions:	The customer has selected a move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	3. The customer presses the “Move Out” button. 4. The customer is able to select the date and timing of the moving services.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to schedule the move out service within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0010		
Use Case Name:	Providing location for delivery		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to provide a location for delivery
Preconditions:	The customer has selected a move in/move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer presses the “Provide Location” button. 2. The customer is able to enter the address of the delivery with a postal code.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to schedule the location within 2 seconds.
Assumptions:	-
Notes and Issues:	App must request the customer to provide a valid 6-digit postal code, SXXXXX within Singapore for the item delivery.

Use Case ID:	M0011		
Use Case Name:	Providing date for delivery		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to provide a date for delivery
Preconditions:	The customer has selected a move in/move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer presses the “Provide Date” button. 2. The customer is able to enter the date of the delivery.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to schedule the date within 2 seconds.
Assumptions:	-
Notes and Issues:	App must request the customer to provide a date in DD/MM/YYYY format.

Use Case ID:	M0012		
Use Case Name:	Providing time for delivery		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to provide a time for delivery
Preconditions:	The customer has selected a move in/move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer presses the “Provide Time” button. 2. The customer is able to enter the time of the delivery.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to schedule the time within 2 seconds.
Assumptions:	-
Notes and Issues:	App must request the customer to provide a date in HH:MM 24-hour format.

Use Case ID:	M0013		
Use Case Name:	Display progress tracker		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to track the progress of the delivery.
Preconditions:	The customer has selected a move in/move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer.
Flow of Events:	1. The customer presses the “Track delivery” button. 2. The customer is able to see the progress of the delivery.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to update the tracker within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0014		
Use Case Name:	Update Progress Tracker for Collected		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker is able to update the tracker to collected.
Preconditions:	The job seeker has scheduled a move in/move out service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the job seeker..
Flow of Events:	1. The job seeker collects the product from the location. 2. The job seeker updates the progress tracker to “Collected”.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to update the tracker within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0015		
Use Case Name:	Update Progress Tracker for Delivering		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker is able to update the tracker to delivering.
Preconditions:	The progress tracker of the product is collected.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the job seeker..
Flow of Events:	1. The job seeker collects the product from the location. 2. The job seeker updates the progress tracker to “In Progress”.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to update the tracker within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0016		
Use Case Name:	Update Progress Tracker for Delivered		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker is able to update the tracker to delivered.
Preconditions:	The progress tracker of the product is in progress.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the job seeker..
Flow of Events:	1. The job seeker delivers the product from the moving service. 2. The job seeker updates the progress tracker to “Delivered”.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to update the tracker within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	M0017		
Use Case Name:	Cancellation of Moving Service		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Customer
Description:	The customer is able to cancel the scheduled service.
Preconditions:	The customer has a scheduled service.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the customer..
Flow of Events:	1. The job seeker delivers the product from the moving service. 2. The job seeker updates the progress tracker to “Delivered”.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App is able to update the tracker within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	J0018		
Use Case Name:	Display of job seeker homepage		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job seeker
Description:	When the job seeker successfully logs in, the job seeker homepage is displayed.
Preconditions:	The job seeker manages to successfully log in.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the job seeker.
Flow of Events:	Job seeker homepage is shown.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to check information validity within 2 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	J0019		
Use Case Name:	Available Job Listings		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job seeker
Description:	The job seeker is able to view the available list of products.
Preconditions:	The job seeker has logged in.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The job seeker presses the “View products” button. 2. The job seeker is able to view the list of products available.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to display listing of products within 2 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	J0020		
Use Case Name:	Display of moving service job information		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker is able to view the information of the delivery.
Preconditions:	The job seeker has registered for a moving service job.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The job seeker presses the “View Information” button. 2. The app displays the date, time and location of the delivery.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to display information within 2 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	J0021		
Use Case Name:	Display of group buying job information		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	The job seeker is able to view the information of the delivery.
Preconditions:	The job seeker has registered for a group buying job.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The job seeker presses the “View Information” button. 2. The app displays the date, time and location of the delivery.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to display information within 2 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	J0022		
Use Case Name:	Job Registration		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	Job seekers caan register for available jobs.
Preconditions:	Job seekers have logged in.
Postconditions:	
Priority:	
Frequency of Use:	By the intention of the user.
Flow of Events:	1. Job seeker clicks register for an available job listed in Job Listings page. 2. The job is registered successfully.
Alternative Flows:	If Job Seeker has already registered for 5 jobs, show “You have registered the maximum number of jobs!” Then refuse the Job Seeker
Exceptions:	
Includes:	
Special Requirements:	Processes of job registration should have a response time of less than 3 seconds.
Assumptions:	
Notes and Issues:	

Use Case ID:	J0023		
Use Case Name:	Registered Jobs Listings		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	Job seeker can view their jobs on their Registered Jobs page
Preconditions:	Job seekers have logged in.
Postconditions:	
Priority:	
Frequency of Use:	By the intention of the user.
Flow of Events:	1. Job seeker go to their Registered Jobs page 2. Job seeker can see a list of jobs registered by the current job seeker
Alternative Flows:	If the current job seeker has registered no jobs, then show “No job registered.”
Exceptions:	
Includes:	
Special Requirements:	Displaying job listings should take less than 2 seconds.
Assumptions:	
Notes and Issues:	

Use Case ID:	J0024		
Use Case Name:	Completion of job		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	Job seeker can indicate he/she has completed the job.
Preconditions:	Job seekers have a registered job.
Postconditions:	
Priority:	
Frequency of Use:	By the intention of the user.
Flow of Events:	1. Job seeker has completed the job. 2. Job seeker can change the job status to “Completed”.
Alternative Flows:	If the current job seeker has registered no jobs, then show “No job registered.”
Exceptions:	
Includes:	
Special Requirements:	Displaying job listings should take less than 2 seconds.
Assumptions:	
Notes and Issues:	

Use Case ID:	J0025		
Use Case Name:	Job Payment Completed		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job seekers
Description:	Job seeker will know when the payment is completed.
Preconditions:	1. Job seeker has logged in. 2. Job seeker has completed a registered job. 3. Payment has been paid out by customers.
Postconditions:	
Priority:	
Frequency of Use:	By the intention of the user.
Flow of Events:	1. The job seeker goes to Registered Jobs page. 2. The display of the completed job shows “Payment completed”
Alternative Flows:	
Exceptions:	
Includes:	
Special Requirements:	Displaying job listings should take less than 2 seconds.
Assumptions:	
Notes and Issues:	

Use Case ID:	J0026		
Use Case Name:	Job Withdrawal		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023

Actor:	Job Seeker
Description:	Job Seeker can withdraw a job that the current job seeker successfully registered.
Preconditions:	1. Job seekers have logged in. 2. Job seeker has successfully registered for a job.
Postconditions:	
Priority:	
Frequency of Use:	By the intention of the user.
Flow of Events:	3. Job seeker clicks withdrawal for a certain job in the page of job registered.. 4. Job seeker click “Yes” on the confirmation page. 5. The job is withdrawn successfully.
Alternative Flows:	If the job seeker clicks “No” on the confirmation page, then the job will not be withdrawn, and the application gets back to the job registered page.
Exceptions:	
Includes:	
Special Requirements:	Processes of job withdrawal should have a response time of less than 3 seconds.
Assumptions:	
Notes and Issues:	

1.8. Data Dictionary

a. Customer
    i. A user which engages in moving-related services, and is associated with an account, granting access to the Moving Scheduler feature of the application.
b. Job Seeker
    i. A user which takes up paid moving-related jobs, and is associated with an account, granting access to the Job Schedule feature of the application.
c. Database
    i. A database is a structured collection of data organised into tables, where data can be efficiently created, retrieved, updated, and deleted. It serves as a central repository for storing and managing information.
d. API
    i. An API (Application Programming Interface) is a set of rules, protocols, and tools that allows different software applications to communicate and interact with each other. It defines the methods and data formats that applications can use to request and exchange information.
e. Moving Service
    i. A Moving Service is a service performed by a Job Seeker to transport items from the designated collection point to the designated delivery point. It includes both “Move In” and “Move Out” services.
f. Payment
    i. Payment is the process when the Customer pay money to the Job Seekers. The activities that need Payment service include Moving Service and Group Buying.
g. Job Scheduler

i. A Job Schedule is a system in the application designed to manage and display available jobs to Job Seekers. It serves to help match job opportunities to potential Job Seekers.

h. Progress Tracker
   i. A Progress Tracker is a feature within the application that monitors and records the progress of the moving-related task. It provides real-time updates and visual representations to help users track the progress of the movement. It contains “Scheduled”, “In Progress” and “Delivered”

i. Available Job
   i. A job is a job listing in the database that has yet been assigned to a job seeker. It represents a job opportunity for the Job Seeker to register for.

j. Registered Job
   i. A Registered Job is a job listing in a database which has been allocated to a job seeker and is actively stored in the job listing database.

k. Withdrawn Job
   i. A Withdrawn Job refers to a job listing in a database that was previously available or registered but has been removed by the previously assigned job seeker.

l. User Interface (UI)
   i. A User Interface is the visual aspect of the software application that allows users to interact and control it. It features components that accept user inputs and which accesses the functionality of the application.

m. Landing Page
   i. The initial page which is rendered to the user when they open up the application.

1.9. UI Mockups

1.9.1. Landing Page & PDPA Notice

The landing page UI allows users to choose between logging in and signing up upon entering the page. The Personal Data Protection Act (PDPA) UI ensures users that their personal details will be kept confidential under PDPA confidentiality.

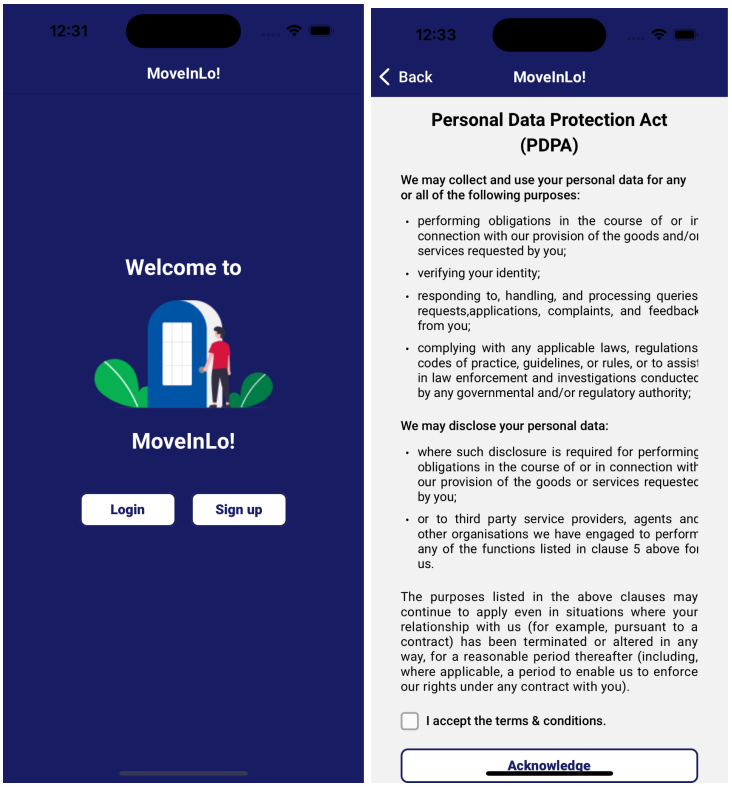


Figure 4: Landing Page & PDPA Notice

1.9.2. Login Page & Sign up Page

The login page UI allows users to log in to their respective account type. The sign up page UI allows users to create an account if they do not have an existing one.

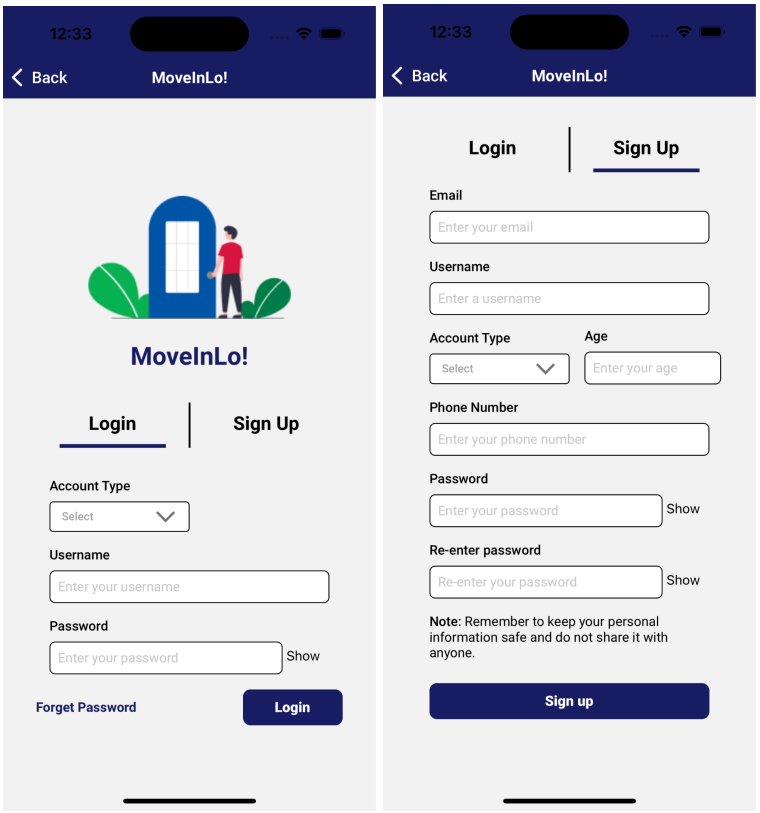


Figure 5: Log In Page & Sign Up Page

1.9.3. Schedule Moving Service

The scheduler page UI allows customers to schedule a moving service. They are required to input their pick up and delivery date, as well as their pick up and delivery location.

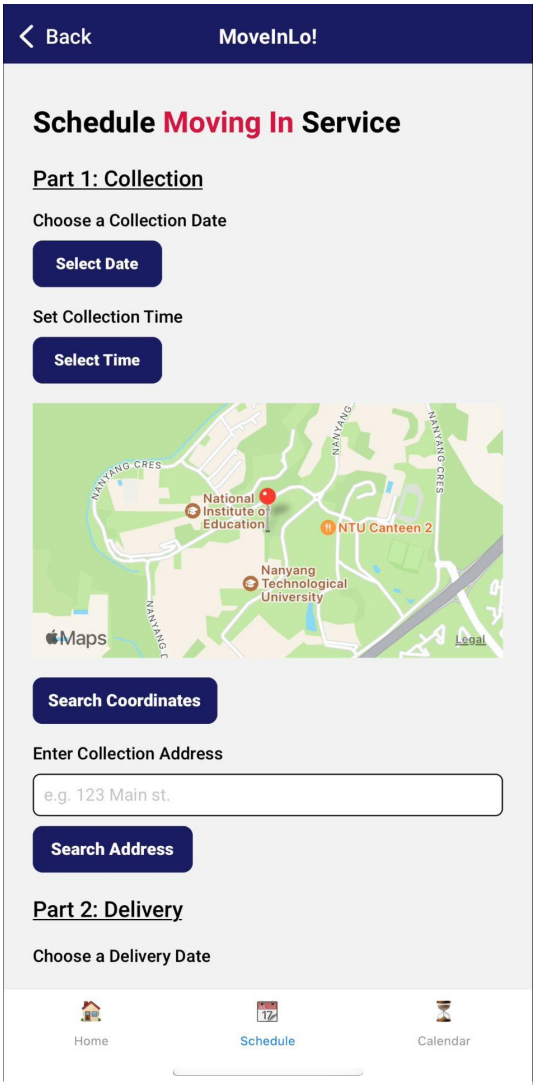


Figure 6: Scheduler Page UI

1.9.4. Job Seeker Home Page & Progress Tracker

The job seeker home page provides detailed instructions on how to accept job requests. The Progress Tracker UI allows Job seekers to track their progress on their delivery.

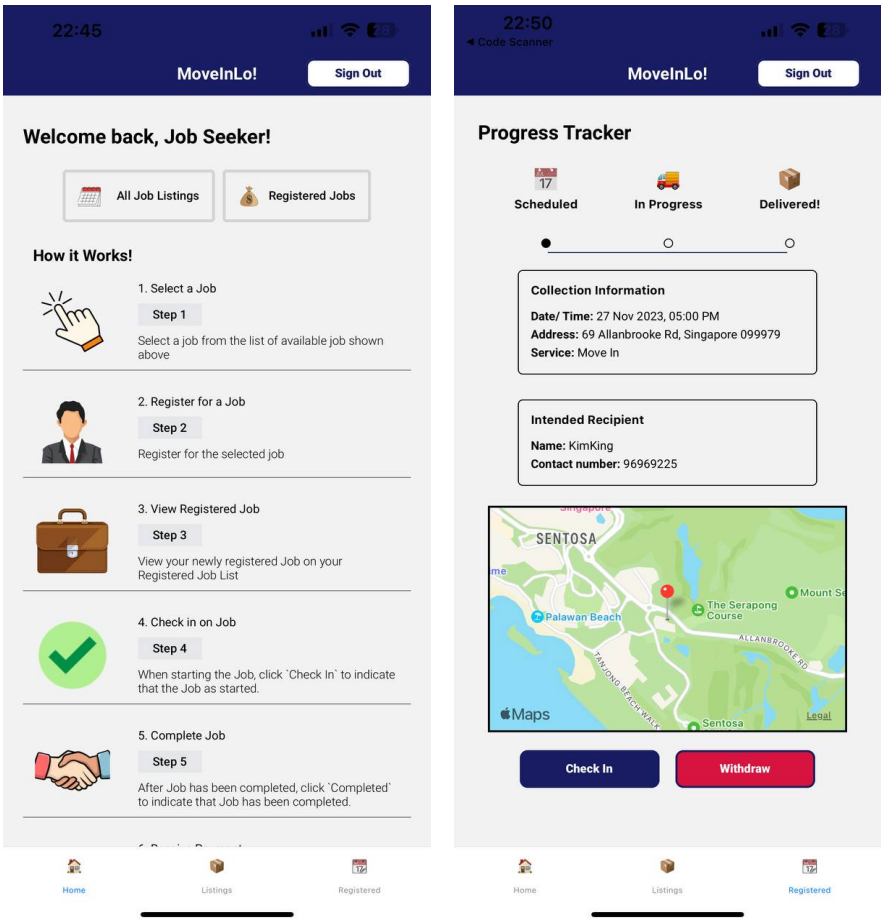


Figure 7: Job Seeker Home Page & Progress Tracker UI

1.9.5. Job Listings & View Job

The job listings UI allows job seekers to view the job requests available to them. The view job UI allows job seekers to view more information on a certain job and accept the job request.

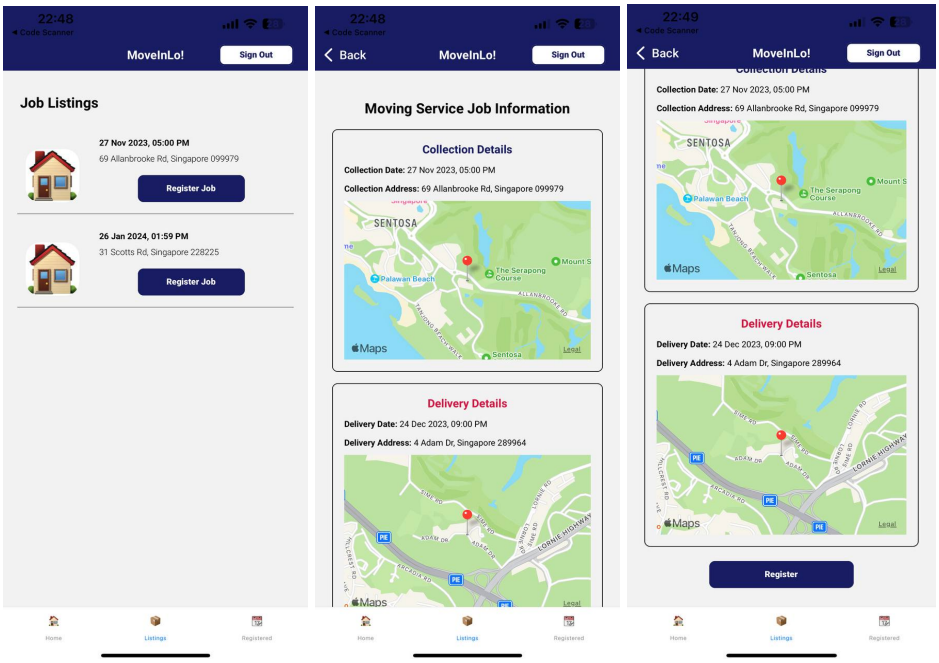


Figure 8: Job Listings UI & View Job UI

1.9.6. Registered Job & Withdraw Job

The registered job UI confirms with the user on their selected job they have chosen. The withdraw job UI allows users to withdraw from the job.

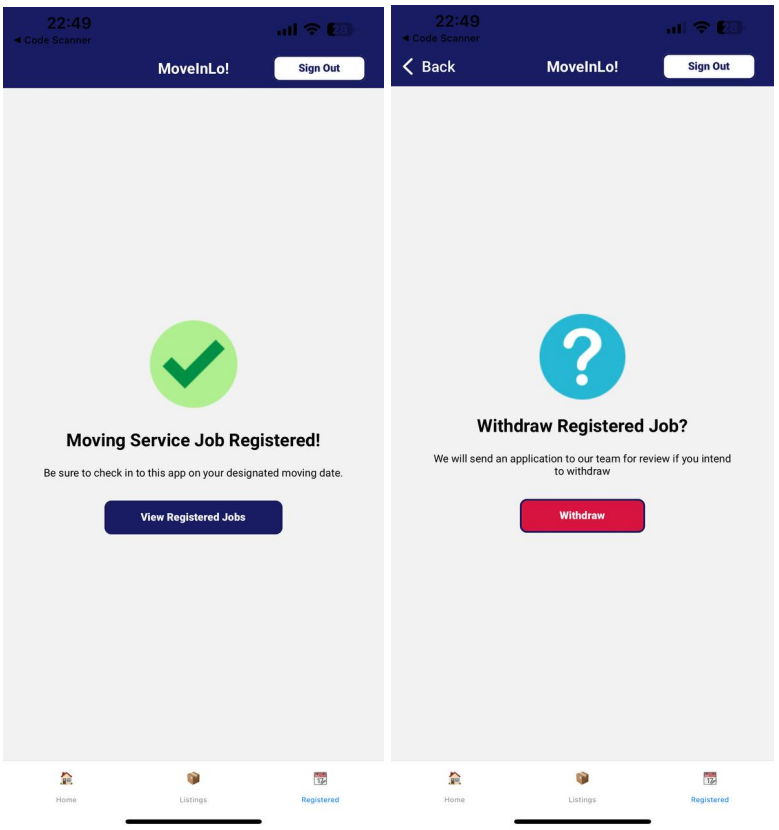


Figure 9: Registered Job UI & Withdraw Job UI

1.9.7. Registered Job Information & Registered Job Lists

The **registered job tracker page** shows the details of the job the user has registered for and the **registered job lists page** shows the lists of jobs the user has registered for.

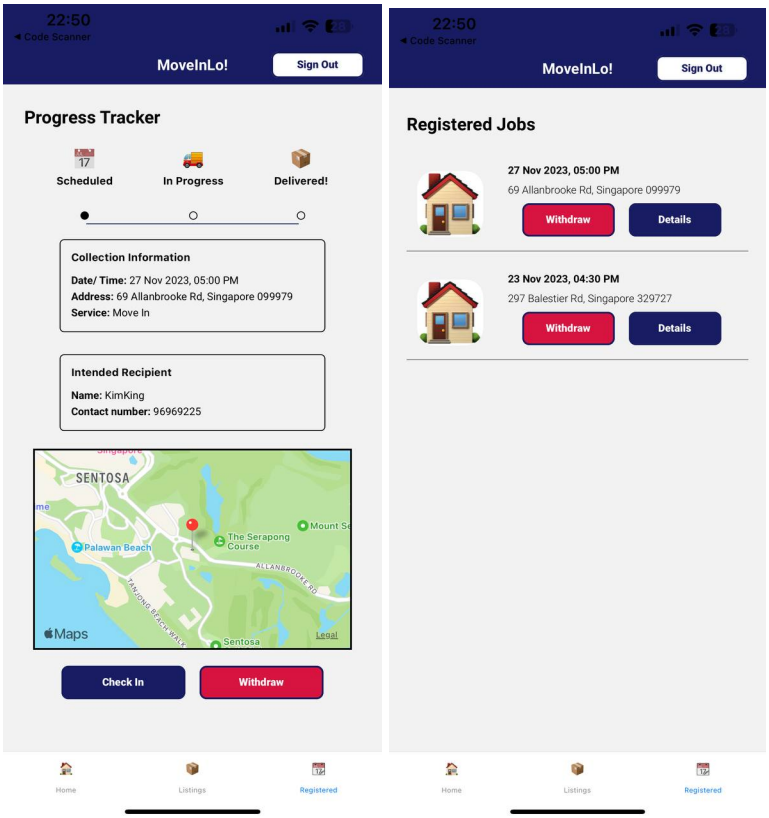


Figure 10: Registered Job Information UI & Registered Job Lists UI

2. System Design

2.1. Class Diagrams

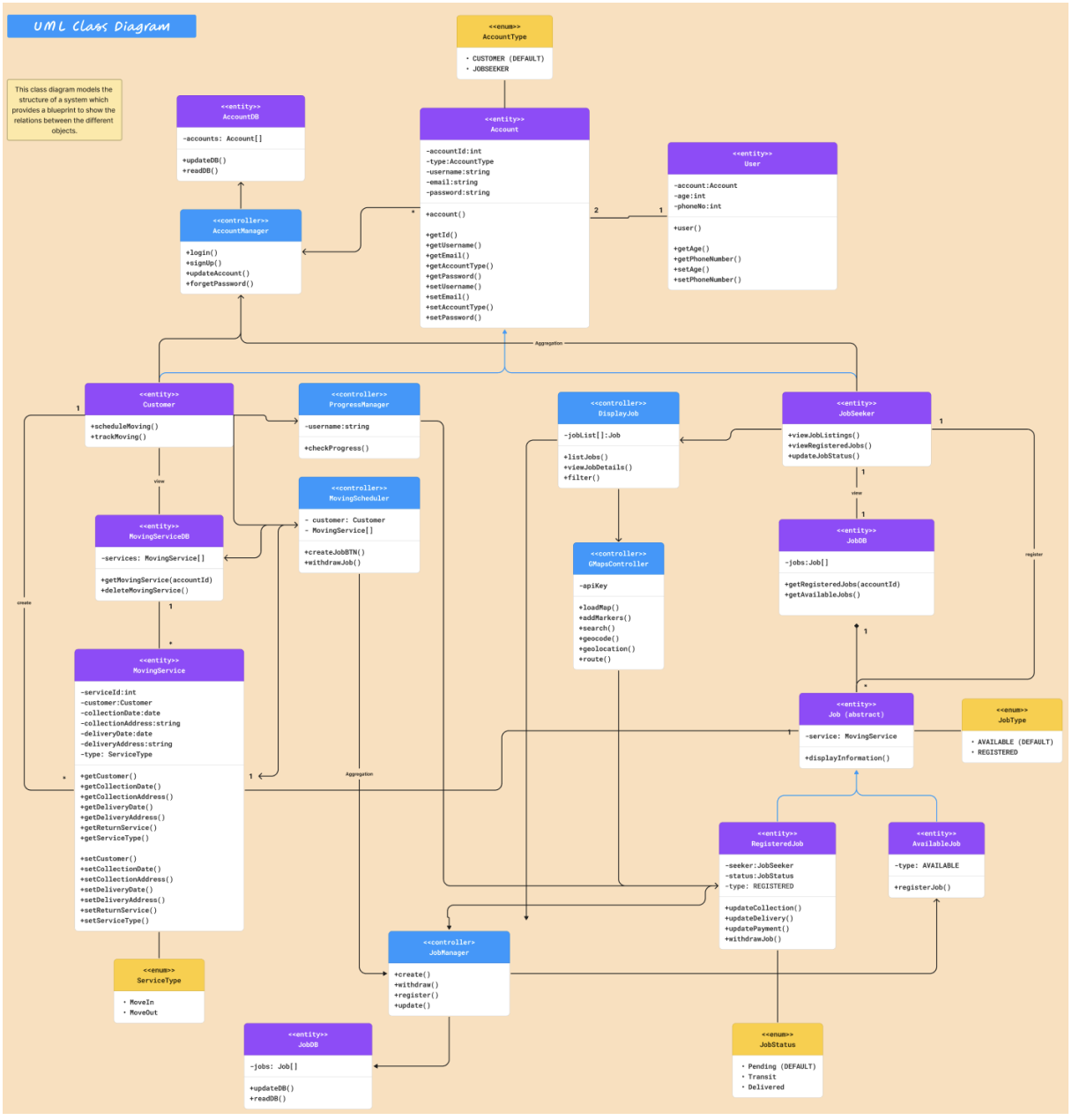


Figure 11: UML Class Diagram

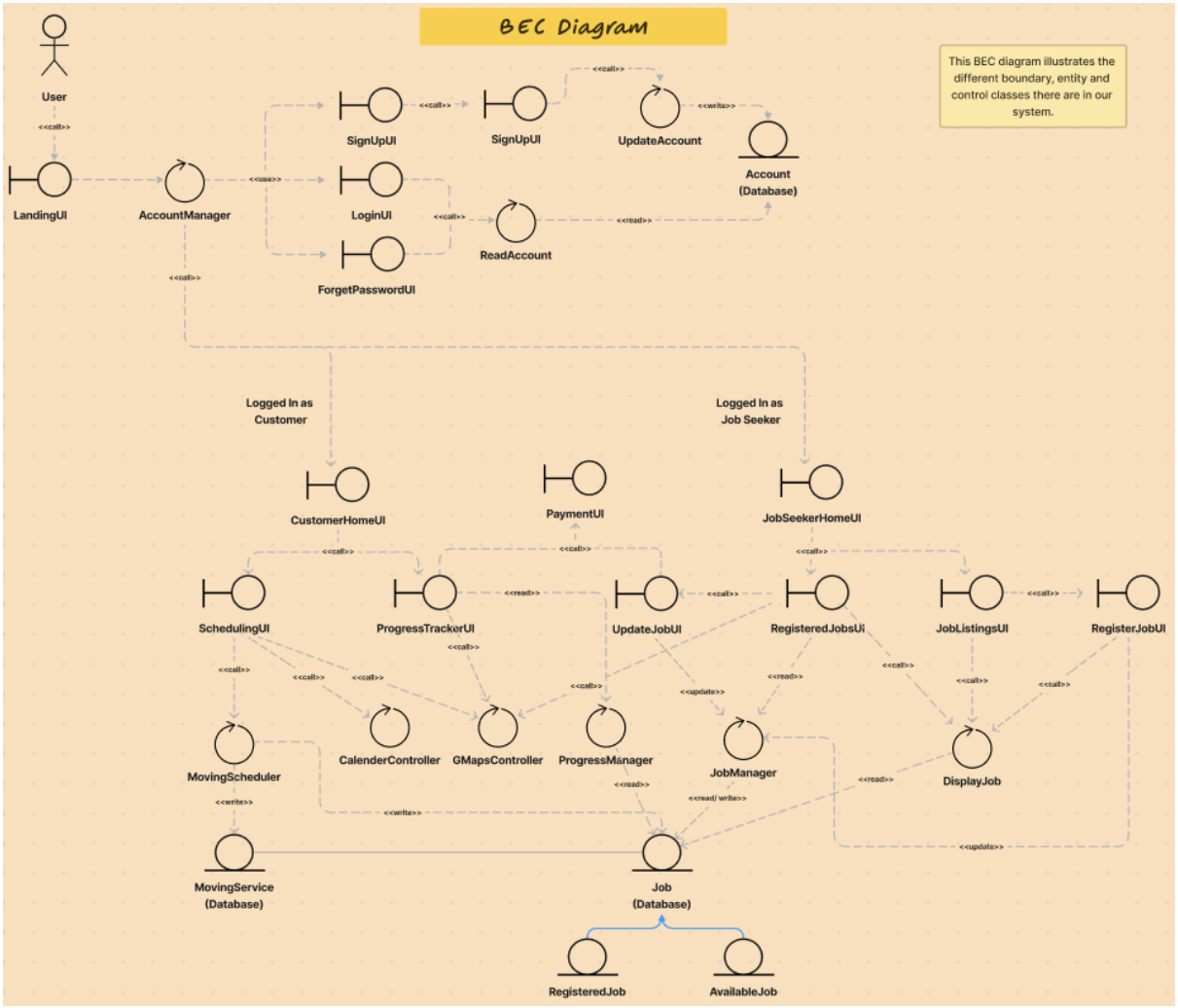


Figure 12: BEC Diagram(Dynamic Class Diagram)

2.2. Sequence Diagrams

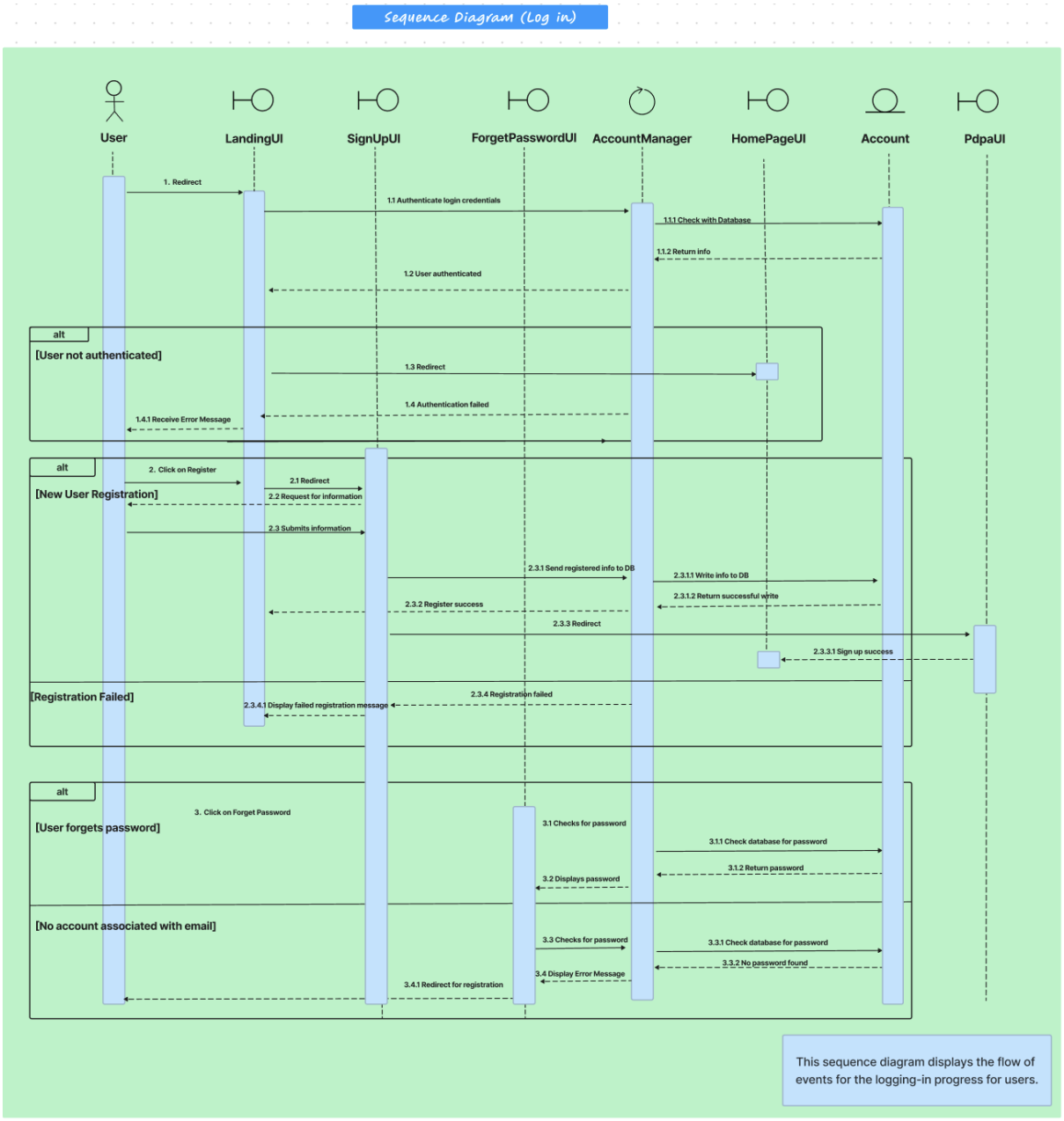


Figure 13: Sequence Diagram(Authentication)

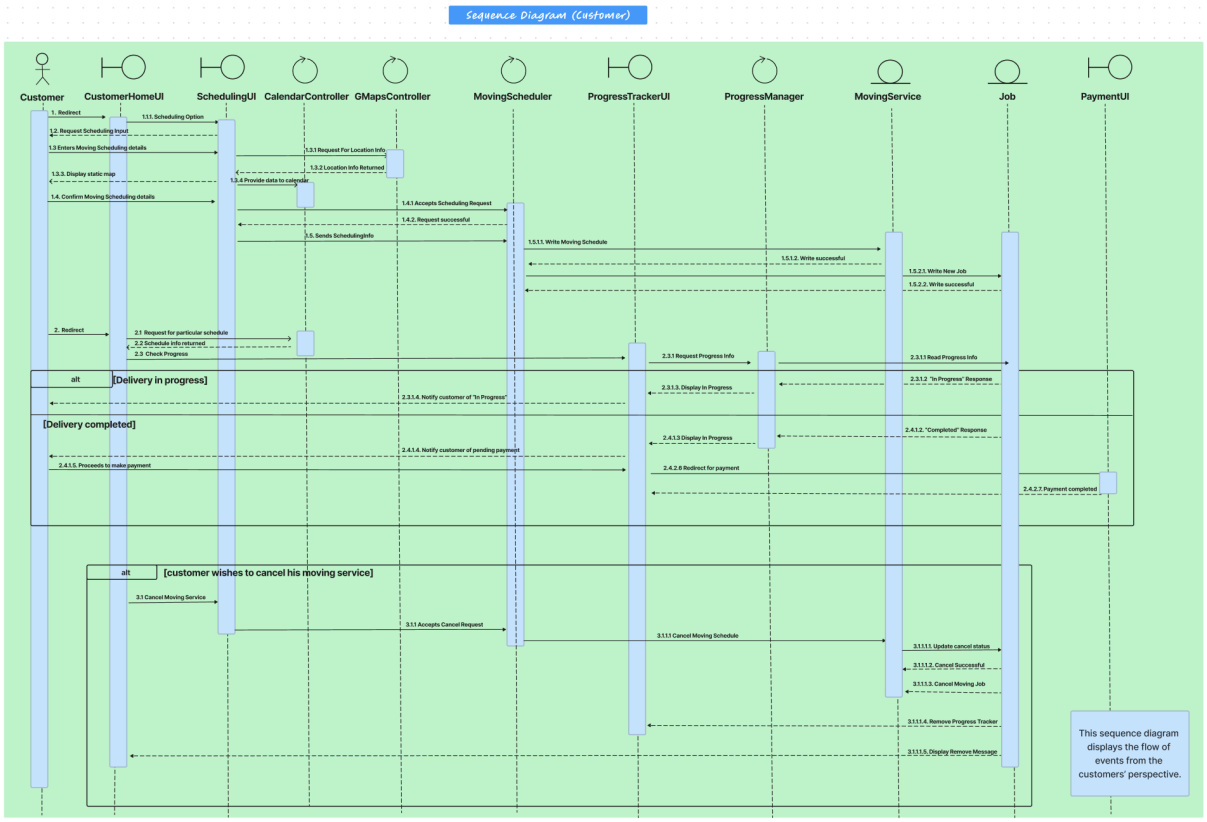


Figure 14: Sequence Diagram(Customer)

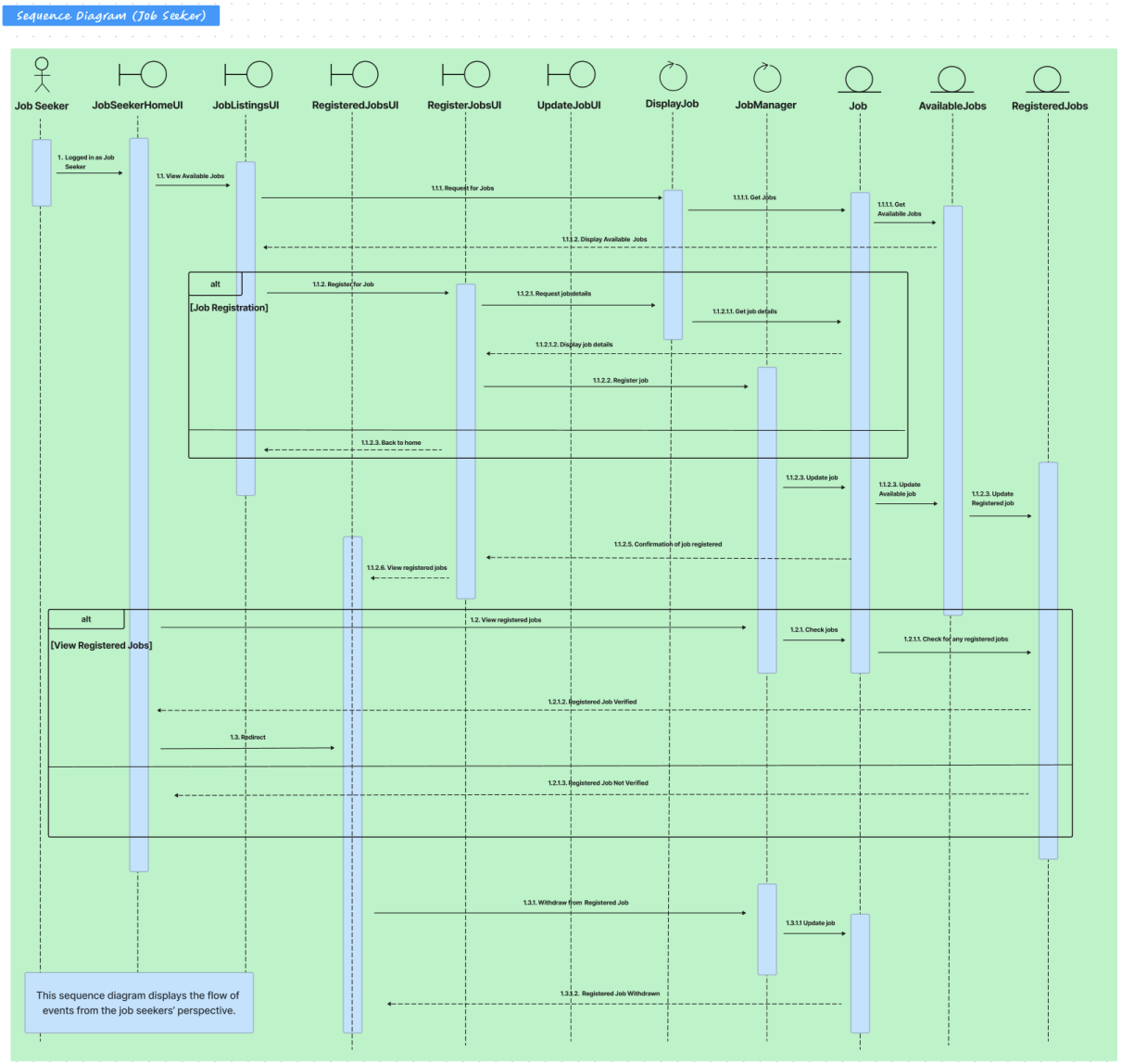


Figure 15: Sequence Diagram(Job Seeker)

2.3. System Architecture

2.3.1. Model-View-Controller (MVC)

To decouple the user interface (View), application business logic (Controller) and data layers (Model), we utilised the Model-View-Controller architecture for better code organisation, and supported faster planning and execution of development work.

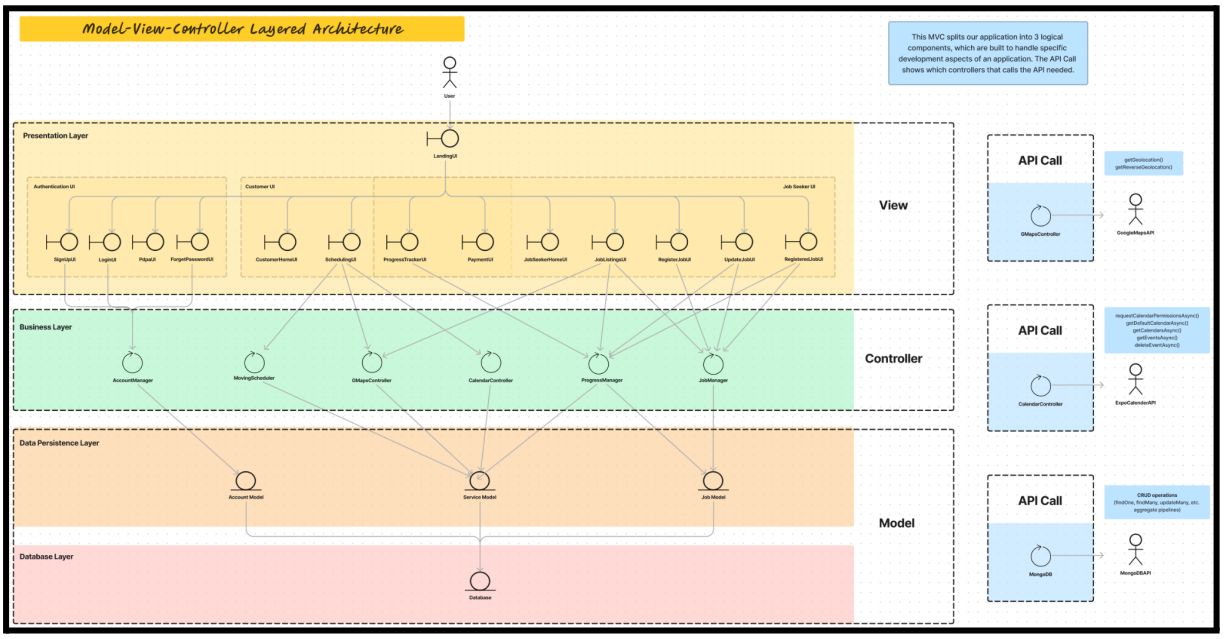


Figure 16: MVC Architecture

2.3.2. Client-Server Architecture

By dividing our codebase into client and server side applications, we can enable **concurrent frontend and backend development** work without much dependencies across both teams. There is **better separation of concerns** enabling **faster time-to-market development** and allows for different features to be developed independently.

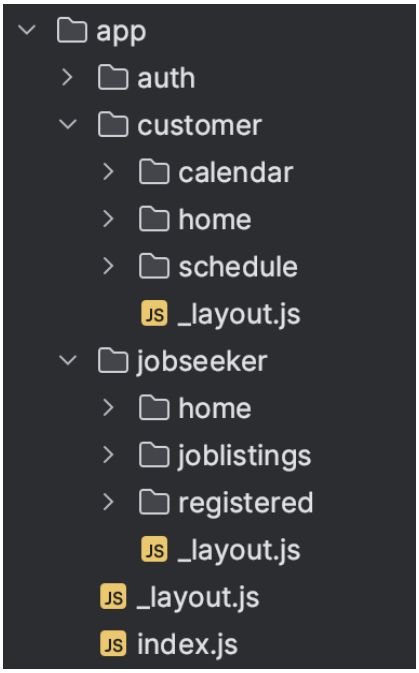


Figure 17: Client-side Layout

Narrowing our focus to the client-side application, since our project uses Expo framework, it uses a **directory structure** to handle page-routing within the application.

Related pages can be grouped together using Stack (i.e. RegisterUI, RegisterSuccessUI) in the same directory, while Tab can allow **different pages to be toggled** using the footer buttons (i.e. Job Seeker can toggle between home page, job listings & registered jobs).

Therefore, we can encapsulate different groups of pages, **enabling better readability** within the client-side codebase.

2.4. Design Patterns

In this section, we discuss the various design patterns we used during our development.

2.4.1. Creational Pattern

Singleton Pattern (Server side)

In our server side application, we are required to establish a database connection to MongoDB Atlas and persist that connection throughout the server. We can achieve this using the Singleton pattern, which ensures that there is **only one global access point to that single database instance**, enabling a central point of control for all database requests.

2.4.2. Structural Pattern

Higher Order Component Pattern (Client side)

In React Native, we can design a Higher Order Component (HOC) which provides the same logic to multiple components, while ensuring that **all the logic remains in a single component**.

For instance, when building reusable components for our client side application (i.e. BaseButton), it contains common, standardised styles and functionalities in the base component, while granting developers the **flexibility to extend its functionalities** and design by passing context-specific props during its implementation. This **reduces redundant code,** improves **maintainability** for developers and also ensures **consistency** in components used throughout the application.

2.4.3. Behavioural Pattern

Chain of Responsibility Pattern (Client & Server side)

For our **client side** application, due to the nature of various inter-linked interactions between different components, we could implement the chain of responsibility design pattern which utilises handlers to encapsulate the implementation logic, as well as to catch any thrown errors. This allows for better readability within the codebase as well as enables better error-control management within the application.

```
useEffect( effect: () : void => {
  1 usage  JosephT631
  async function prepare() : Promise<void> {
    try {
      await fetchAccount(notes);
      await fetchAllData(notes);
    } catch (e) {
      console.warn(e);
    } finally {
      // Tell the application to render
      setAppIsReady( value: true);
    }
  }
  if (!appIsReady) {
    prepare();
  }
}, deps: [idStore]);
```

Figure 18: Pre-fetching data

From Figure xx, we can observe the chain of responsibility pattern being implemented. The asynchronous function, prepare() encapsulates the data fetching logic, where **account-related information is first retrieved** and updates the relevant states before subsequently **fetching all other data** (i.e. service and progress information), ensuring that data is valid before proceeding to the next stage.

This similar pattern was also used for our **server side** application, where through the use of API handlers, we can abstract the validation of APIs calls from the business logic implementation present in controllers. This also ensures that any errors are caught and handled at their respective stages, enabling developers to be better able to spot and debug on the spot.

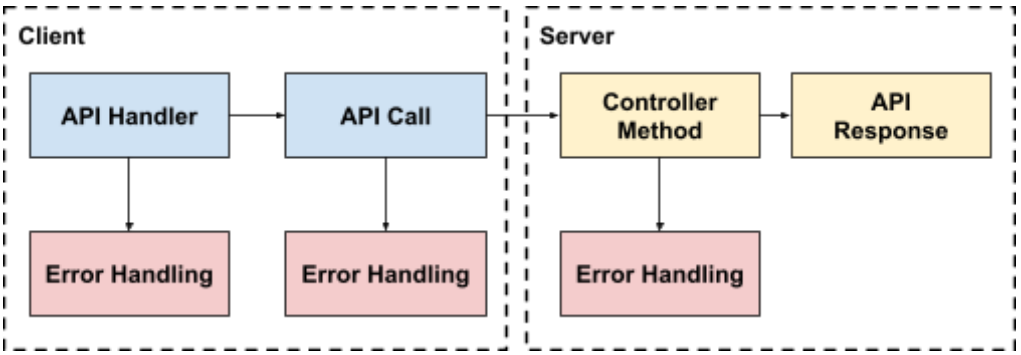


Figure 19: Use of API Handlers

Command Pattern (Client side)

Since form inputs generally originate from various sources (i.e. text inputs, date/time picker, API calls), we implemented the command design pattern using an inputHandler() which encapsulates the logic of updating the state of

the user inputs. This handler can also manage updates to other states encapsulated within this function, reducing coupling between GUI and business logic layers.

```
const inputHandler = (input, field) : void  => {
  setInfo( value: (prevState : {...} ) => ({ ...prevState, [field]: input }));
};
```

Figure 20 : Input Handler implementation

Therefore, this allows for separation of concerns since each respective component only needs to trigger the input handler without needing to worry about the implementation of the updating of state.

3. Implementation

3.1. Application Skeleton

The complete application skeleton can be found at the following GitHub link: https://github.com/softwarelab3/2006-SCSX-Codecrafters/tree/main/Lab%205%20Final%20Deliverables/Implementation/appskeleton

3.2. Source Code

The complete source code can be found at the following GitHub link: https://github.com/softwarelab3/2006-SCSX-Codecrafters/tree/main/MoveInLo

3.3. Demo Script

3.3.1. Registration

a. Click on “Sign up”
b. *Redirected to Sign Up UI*
c. Insert Email: jeff@gmail.com
d. Insert Username: Jeff
e. Insert Account Type: Customer
f. Insert Age: 22
g. Insert Phone Number: 87654321
h. Insert Password: Password123!
i. Insert Password again: Insert: Password123!
j. Click on Sign Up
k. *Redirected to PDPA UI*

3.3.2. Accept PDPA

a. Click on “I accept terms and condition”
b. Click on “Acknowledge”
c. *Redirect to customer home page*

3.3.3. Schedule a move in service (Customer)

a. Click on Schedule
b. Click on Move In service
c. *Redirected to Scheduler UI*
d. Select Pick Up Date: 23 November 2023
e. Select Pick Up Time: 4:30 PM
f. Select Pick Up Location: 288 Balestier Rd
g. Select Delivery Date: 24 November 2023
h. Select Delivery Time: 4:30 PM
i. Select Delivery Location: Hall of Residence 3
j. Click on Schedule

3.3.4. Track Delivery (Part 1) (Customer)

a. Click “Track Delivery”
b. Click “View Details

3.3.5. Logging in (Job Seeker)

a. Click on “Login”
b. Select “Job Seeker” account type
c. Insert Username: John
d. Insert Username: Password123!
e. Click on “Login”

3.3.6. Accepting a Job and Viewing registered Job (Job Seeker)

a. Click on Job Listings
b. Select Job Listing that was created at 3.3.3 by clicking on “Register Job”
c. Click on “Register”
d. Click on “View Registered Job”
e. *Redirected to Registered Job list*
f. Click on “Details” of the job just registered (3.3.3.)

3.3.7. Update Tracker (Part 1) (Job Seeker)

a. Click on “Check in”

3.3.8. Track Delivery (Part 2) (Customer)

a. Click “View Details”

b. *Redirected to Tracker UI*

3.3.9. Update Tracker (Part 2) (Job Seeker)

a. Click on “Completed”

3.3.10. Make Payment (Customer)

a. Click on “Proceed to Payment”

b. *Redirected to QR Code*

c. Click on “Payment made”

3.3.11. Receive Payment (Job Seeker)

a. Click on “Payment Received”

3.4. Demo Video

The complete demo video can be found at the following Youtube link: https://youtu.be/cP5oUKL51ws

4. Testing

4.1. Overview

For a complete testing environment, we utilised **both black-box and white-box testing** frameworks for our application. Our rationale was to **avoid duplicate testing**, yet provide a **comprehensive testing framework** which ensures overall functionality of the application.

For cases where there are **multiple error-control checks**, we utilise **black-box testing** to ensure that each test scenario is accounted for (i.e. input validations), which are boundary values.

On the other hand, we utilised **white-box testing**

4.2. Black-Box Testing

Authentication

Logging in

Input	Expected Output	Actual Output
Existing User & Valid Password	Successful Authentication	Successful Authentication
Non-existing User	Error Message	Error Message: Invalid username and/or type
Invalid Password	Error Message	Error Message: Invalid password

Registration

Email	UserName	Account Type	Age	Phone Number	Password	Re-enter Password	Expected Output	Actual Output
Valid Email	Valid username	Valid Type	Valid EC 16 < x	Valid number	Valid password	Same as password	Success	Success
Valid Email	Valid username	Valid Type	Invalid EC 15	Valid number	Valid password	Same as password	Error	Error: Users aged 16 and below cannot register
Valid Email	Valid username	Valid Type	Valid EC 16 < x	Valid number	Valid password	Different from password	Error	Error: Please enter a valid email
Invalid Email (without @)	Valid username	Valid Type	Valid EC 16 < x	Valid number	Valid password	Same as password	Error	Error: Both passwords must match
Valid Email	Existing username	Valid Type	Valid EC 16 < x	Valid number	Valid password	Same as password	Error	Error: Username exists

Customer

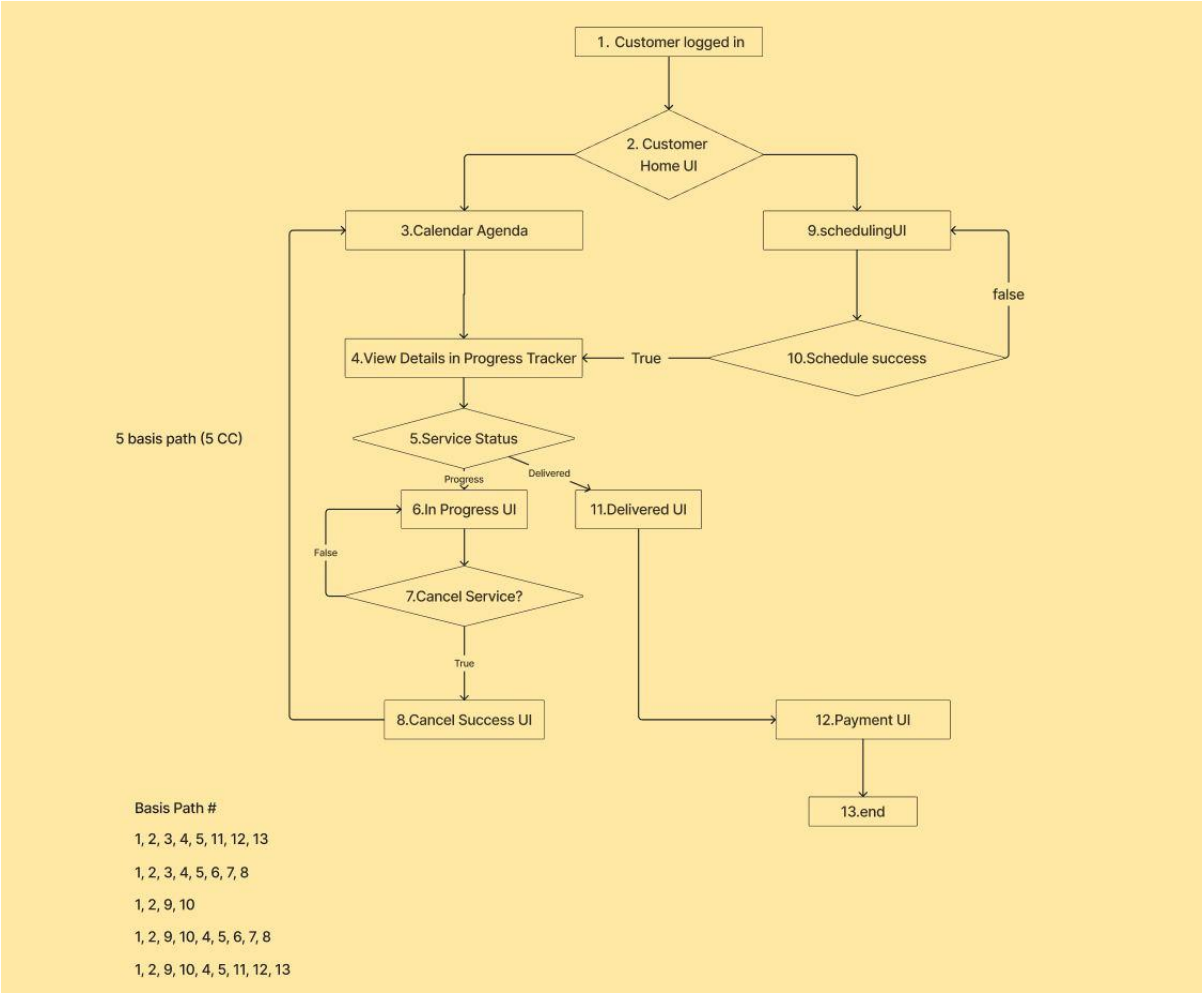
Moving in Scheduler

- When a customer wants to schedule

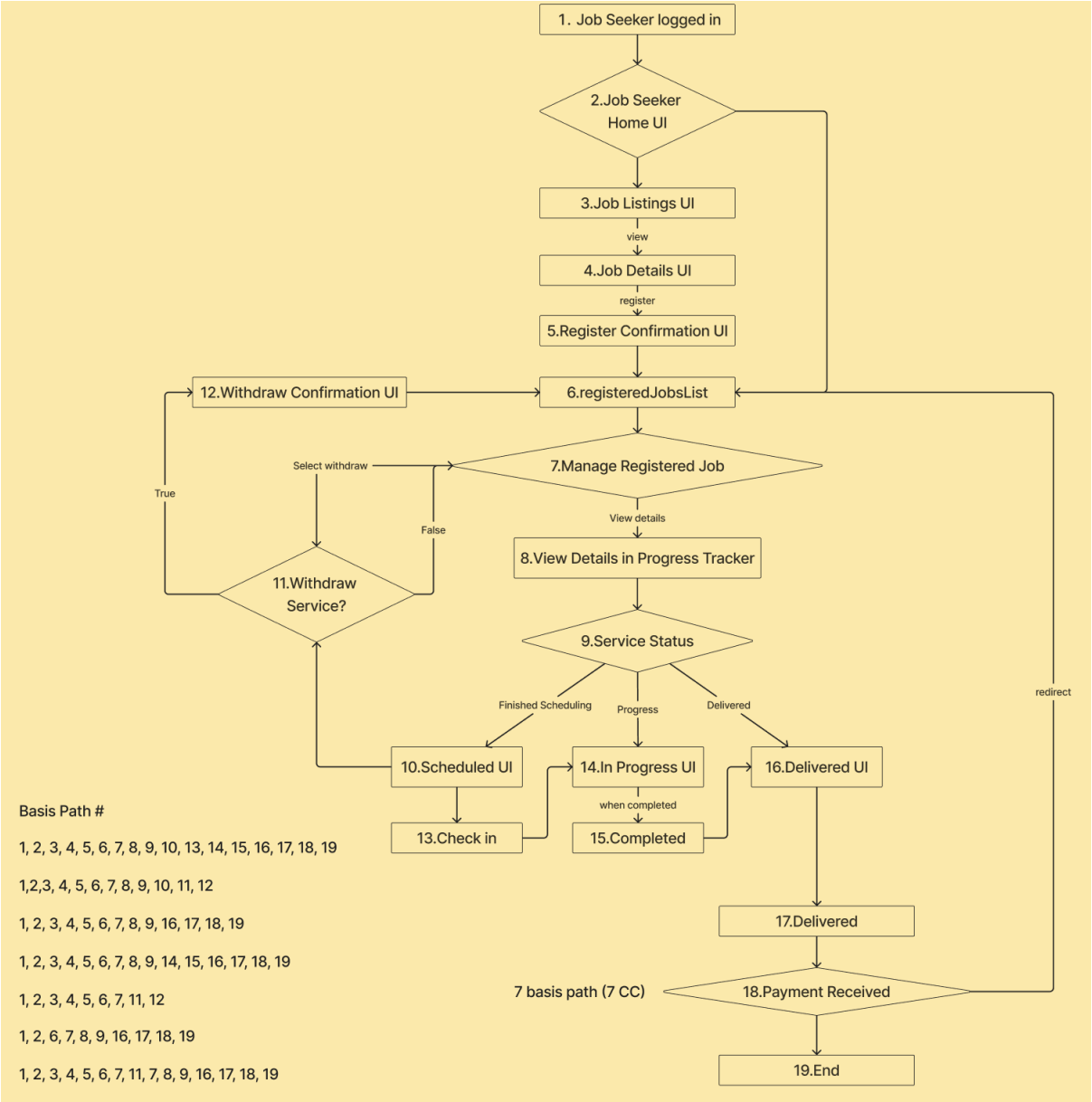
Pick up date	Pick up time	Pick up address	Delivery date	Delivery time	Deliver address	Expected output	Actual output
>= current date	> current date	Valid address	> pick up date	N.A.	Valid address	Success	Success
< current date	> current date	Valid address	> pick up date	N.A.	Valid address	Error	Error: Collection Date has to be later than today's date!
>= current date	< current date	Valid address	> pick up date	N.A.	Valid address	Error	Error: Collection Time should be later than current date!
>= current date	> current date	Invalid address	> pick up date	N.A.	Valid address	Error	Error: No results found, try providing more information
>= current date	> current date	Valid address	< pick up date	N.A.	Valid address	Error	Error: Delivery time has to be later than collection date!
>= current date	> current date	Valid address	> pick up date	N.A.	Invalid address	Error	Error: No results found, try providing more information

4.3. White-Box Testing

Customer



JobSeeker


1. Job Seeker logged in
2.Job Seeker Home UI
3.Job Listings UI
view
4.Job Details UI
register
5.Register Confirmation UI
6.registeredJobsList
12.Withdraw Confirmation UI
7.Manage Registered Job
Select withdraw
True
False
View details
8.View Details in Progress Tracker
11.Withdraw Service?
9.Service Status
Finished Scheduling
Progress
Delivered
redirect
10.Scheduled UI
14.In Progress UI
16.Delivered UI
when completed
13.Check in
15.Completed
17.Delivered
18.Payment Received
19.End
Basis Path #
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19
1,2,3, 4, 5, 6, 7, 8, 9, 10, 11, 12
1, 2, 3, 4, 5, 6, 7, 8, 9, 16, 17, 18, 19
1, 2, 3, 4, 5, 6, 7, 8, 9, 14, 15, 16, 17, 18, 19
1, 2, 3, 4, 5, 6, 7, 11, 12
1, 2, 6, 7, 8, 9, 16, 17, 18, 19
1, 2, 3, 4, 5, 6, 7, 11, 7, 8, 9, 16, 17, 18, 19
7 basis path (7 CC)

5. References

Refactoring Guru (n.d.). Chain of Responsibility Design Pattern. Retrieved on 11 November 2023, https://refactoring.guru/design-patterns/chain-of-responsibility

Google. (n.d.). *Google Maps API Platform*. Google Maps Platform FAQ | google for developers. https://developers.google.com/maps/faq#setup_billing

NTU Quick Brand Guide 2018 - corporate NTU. (n.d.). https://www3.ntu.edu.sg/CorpComms2/NTU%20Quick%20Brand%20Guide%202018.pdf

Bruegge, & Dutoit, A. H. (2010). Object-oriented software engineering : using UML, patterns, and Java (3rd ed.). Prentice Hall.

Sommerville, I. (2016). Software Engineering. 10th Edition, Pearson Education Limited, Boston.

6. Appendix

This section includes all UML Diagrams, Links to mockups, Source code and other deliverables.

1. Use Case Diagram
2. UML Class Diagram
3. BEC Diagram
4. Sequence Diagram(Auth)
5. Sequence Diagram(Customer)
6. Sequence Diagram(Jobseeker)
7. DialogMap
8. System Architecture Diagram
9. Initial UI Mockups
10. Application Skeleton
11. Source Code
12. Demo Video
13. GitHub Repository
14. Video Demo