

---

# **Lab 4**

**for**

**SpotOn**

**Version 1.0**

**Prepared By**

**Ng Zi Yuan**

**Pang Yee Leong**

**Lam Wai Jun**

**Stephanie Heather Zaw**

**Tan Ying Xuan**

**Nanyang Technological University**

**7/4/2025**

## **Deliverables**

- 1. Working application prototype**
- 2. Source code**
- 3. Test Cases and Testing Results**
- 4. Demo Script**

# Test Cases and Testing Results

## 1. Black Box Testing for UserControl Class

All inputs in Register and Login interfaces are case-sensitive.

### 1.1 Register Interface

#### 1.1.1 Equivalence Class Testing for Password

<b>Valid EC</b>	<ul style="list-style-type: none"><li>• A valid password with length of 8 and above, at least 1 uppercase character and at least 1 special character E.g. Pw12345!</li></ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"><li>• Password missing one uppercase character E.g. pw12345!</li><li>• Password missing one special character</li><li>• E.g. Pw123456</li></ul>

#### 1.1.2 Equivalence Class Testing for Re-enter Password

<b>Valid EC</b>	<ul style="list-style-type: none"><li>• Re-entered password is exactly identical to valid password E.g. Valid Password: Pw12345! Re-entered Password: Pw12345!</li></ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"><li>• Re-entered password is different from the valid password E.g. Valid Password: Pw12345! Re-entered Password: Pw12344! E.g. Valid Password: Pw12345! Re-entered Password: Pw12345!-</li></ul>

#### 1.1.3 Equivalence Class Testing for Email

<b>Valid EC</b>	<ul style="list-style-type: none"><li>• A valid email address with '@***.com' E.g. test123@gmail.com</li></ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"><li>• Email address missing username E.g. @gmail.com</li><li>• Email address with invalid domain (does not follow ***.*** format) E.g. NewUser@gmailcom</li><li>• Email address missing '@' symbol E.g. NewUser.gmail.com</li></ul>

#### 1.1.4 Equivalence Class Testing for Existing Username

Assume that there is already one registered username, 'ExistingUser1',

<b>Valid EC</b>	<ul style="list-style-type: none"><li>Unused username that does not exist in database E.g. NewUser123</li></ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"><li>Used username that exists in database E.g. ExistingUser1</li></ul>

#### 1.1.5 Equivalence Class Testing for Existing Email

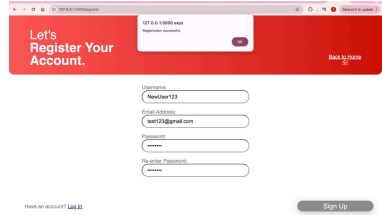
Assume that there is already one registered email, 'ExistingUser1@gmail.com'

<b>Valid EC</b>	<ul style="list-style-type: none"><li>Unused valid email that does not exist in database E.g. NewUser123@gmail.com</li></ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"><li>Used, valid email that exists in database E.g. ExistingUser1@gmail.com</li></ul>

#### 1.1.6 Test Valid Inputs

Assume that there is already one registered account that with:

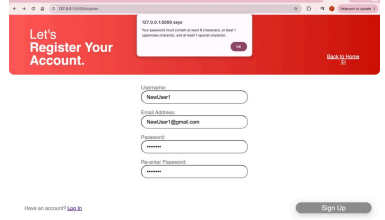
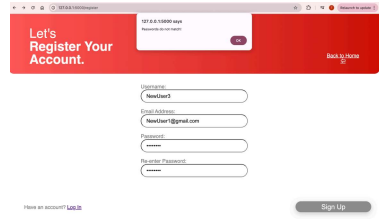
- Username: ExistingUser1
- Email: [ExistingUser1@gmail.com](mailto:ExistingUser1@gmail.com)

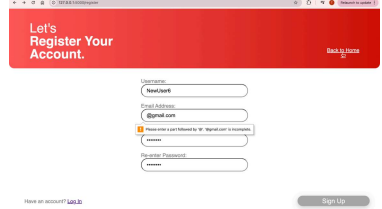
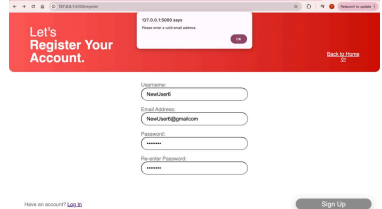
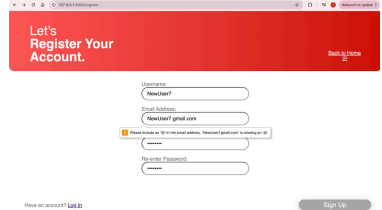
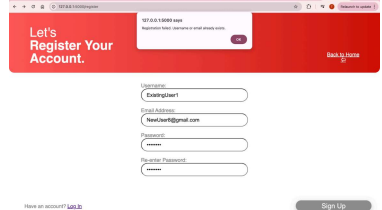
Username	Email	Password	Re-enter password	Expected Result	Actual Result
NewUser123	test123@gmail.com	Pw12345!	Pw12345!	Account is created successfully; System displays "Registration successful"; user is directed to Login Interface	Account is created successfully; System displays "Registration successful"; user is directed to Login Interface 

#### 1.1.7 Test Invalid Inputs

Assume that there is already one registered account that with:

- Username: ExistingUser1
- Email: [ExistingUser1@gmail.com](mailto:ExistingUser1@gmail.com)

Username	Email	Password	Re-enter password	Expected Output	Actual Output
NewUser1	NewUser1@gmail.com	pw12345!	pw12345!	Account is not created; System displays "Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character."; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character."; user is directed to same Register Interface with entered details that can be amended 
NewUser2	NewUser2@gmail.com	Pw12345	Pw12345	Account is not created; System displays "Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character."; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character."; user is directed to same Register Interface with entered details that can be amended
NewUser3	NewUser3@gmail.com	Pw12345!	Pw12344!	Account is not created; System displays "Passwords do not match!"; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Passwords do not match!"; user is directed to same Register Interface with entered details that can be amended 
NewUser4	NewUser4@gmail.com	Pw12345!	Pw12345!-	Account is not created; System displays "Passwords do not match!"; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Passwords do not match!"; user is directed to same Register Interface with entered details that can be amended
NewUser5	@gmail.co	Pw12345!	Pw12345!	Account is not	Account is not created; System

	m			created; System displays "Please enter a part followed by '@'. '@gmail.com' is incomplete"; user is directed to same Register Interface with entered details that can be amended	displays "Please enter a part followed by '@'. '@gmail.com' is incomplete"; user is directed to same Register Interface with entered details that can be amended 
NewUser6	NewUser6@gmail.com	Pw12345!	Pw12345!	Account is not created; System displays "Please enter a valid email address."; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Please enter a valid email address."; user is directed to same Register Interface with entered details that can be amended 
NewUser7	NewUser7.gmail.com	Pw12345!	Pw12345!	Account is not created; System displays "Please include an '@' in the email address. 'NewUser7.gmail.com' is missing an '@.'"; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Please include an '@' in the email address. 'NewUser7.gmail.com' is missing an '@.'"; user is directed to same Register Interface with entered details that can be amended 
ExistingUser1	NewUser8@gmail.com	Pw12345!	Pw12345!	Account is not created; System displays "Registration failed. Username or email already exists."; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Registration failed. Username or email already exists."; user is directed to same Register Interface with entered details that can be amended 

NewUser9	ExistingUser1@gmail.com	Pw12345!	Pw12345!	Account is not created; System displays "Registration failed. Username or email already exists."; user is directed to same Register Interface with entered details that can be amended	Account is not created; System displays "Registration failed. Username or email already exists."; user is directed to same Register Interface with entered details that can be amended
----------	-------------------------	----------	----------	--	--

## 1.2 Login Interface

### 1.2.1. Equivalence Class Testing for Incorrect Username

Assume the user wants to log in with their registered username as 'ExistingUser1'.

<b>Valid EC</b>	<ul style="list-style-type: none"> <li>Registered username E.g. ExistingUser1</li> </ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"> <li>Incorrect username E.g. existinguser1</li> </ul>

### 1.2.2. Equivalence Class Testing for Incorrect Password

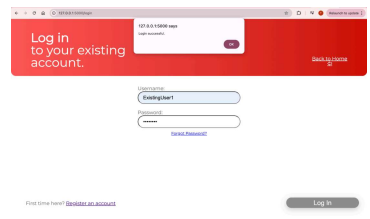
Assume the user wants to log in with their registered password as 'Pw12345!'.

<b>Valid EC</b>	<ul style="list-style-type: none"> <li>Registered password E.g. Pw12345!</li> </ul>
<b>Invalid EC</b>	<ul style="list-style-type: none"> <li>Incorrect password E.g. pw12345!</li> </ul>

### 1.2.3 Test Valid Inputs

Assume that there is already one registered account that with:

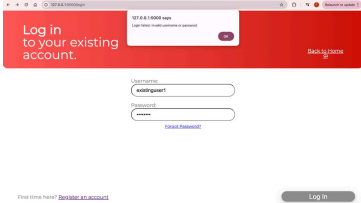
- Username: ExistingUser1
- Password: Pw12345!

Username	Password	Expected Output	Actual Output
ExistingUser1	Pw12345!	System displays "Login successful."; user is directed to InputDestination Interface	<p>System displays "Login successful."; user is directed to InputDestination Interface</p> 

### 1.2.3 Test Invalid Inputs

Assume that there is already one registered account that with:

- Username: ExistingUser1
- Password: Pw12345!

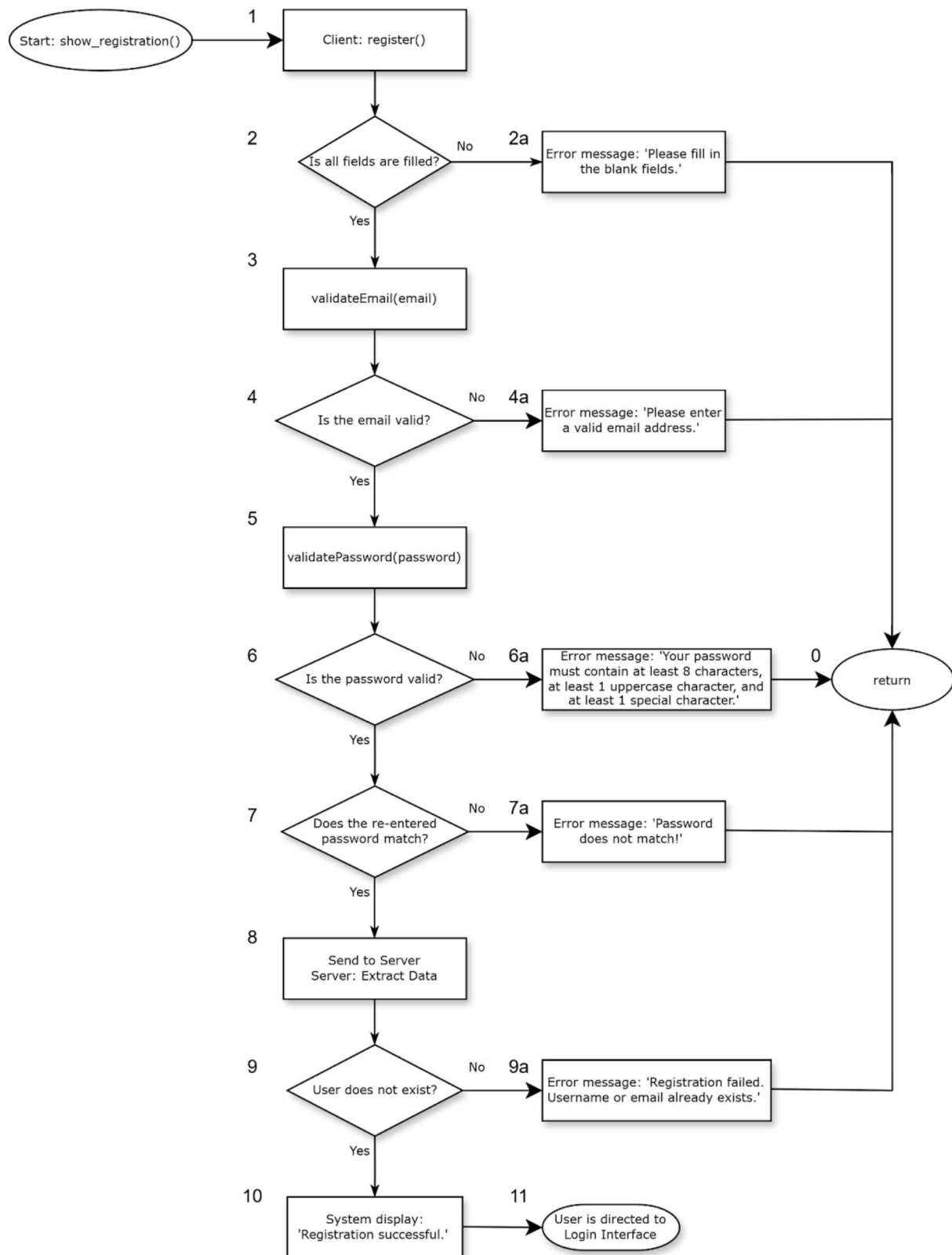
Username	Password	Expected Output	Actual Output
existinguser1	Pw12345!	System displays "Login failed: Invalid username or password"; user is directed to same Login Interface with entered details that can be amended	System displays "Login failed: Invalid username or password"; user is directed to same Login Interface with entered details that can be amended 
ExistingUser1	pw12345!	System displays "Login failed: Invalid username or password"; user is directed to same Login Interface with entered details that can be amended	System displays "Login failed: Invalid username or password"; user is directed to same Login Interface with entered details that can be amended



## 2. White Box Testing

### 2.1 Register Interface

#### 2.1.1 Control Flow Graph



### 2.1.2 Basis Path Testing

**Cyclomatic complexity:** | Decision Points | + 1 = 6

#### Basis Paths

1. Baseline Path: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
2. Basis Path 2: 1, 2, 2a, 0
3. Basis Path 3: 1, 2, 3, 4, 4a, 0
4. Basis Path 4: 1, 2, 3, 4, 5, 6, 6a, 0
5. Basis Path 5: 1, 2, 3, 4, 5, 6, 7, 7a, 0
6. Basis Path 6: 1, 2, 3, 4, 5, 6, 7, 8, 9, 9a, 0

#### Test Cases

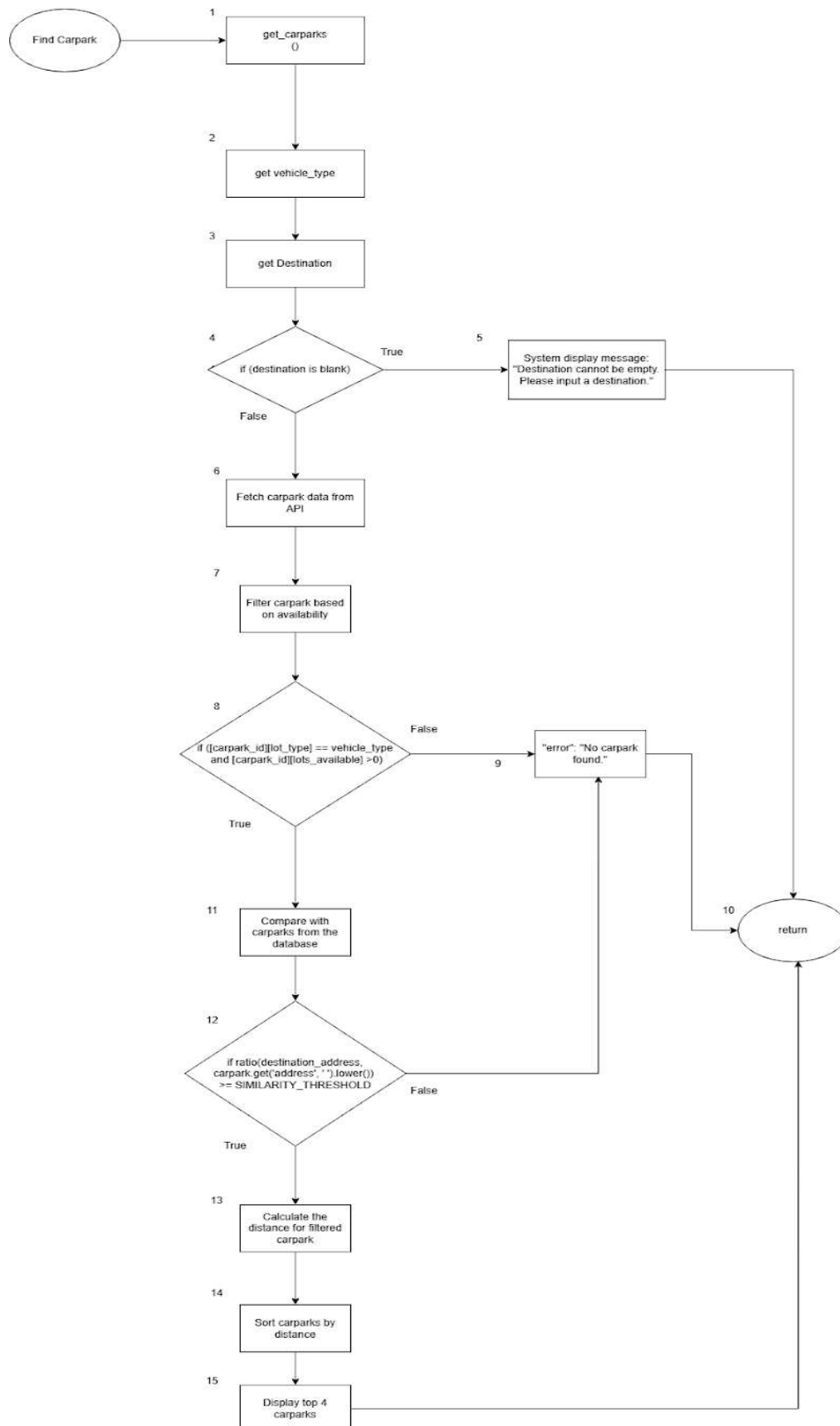
Assume that there is already one registered account that has:

- Username: ExistingUser1
- Email: [ExistingUser1@gmail.com](mailto:ExistingUser1@gmail.com)

Basis Path	Username	Email	Password	Confirm Password	Expected Output	Actual Output
1	NewUser1	NewUser1@gmail.com	Pw12345!	Pw12345!	System display: 'Registration successful,' and user is directed to Login Interface	System display: 'Registration successful,' and user is directed to Login Interface
2	(blank)	NewUser1@gmail.com	Pw12345!	Pw12345!	Error message: 'Please fill in the blank fields.'	Error message: 'Please fill in the blank fields.'
3	NewUser1	@gmail.com	Pw12345!	Pw12345!	Error message: 'Please enter a valid email address.'	Error message: 'Please enter a valid email address.'
4	NewUser1	NewUser1@gmail.com	pw12345!	pw12345!	Error message: 'Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character.'	Error message: 'Your password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character.'
5	NewUser1	NewUser1@gmail.com	Pw12345!	Pw12344!	Error message: 'Password does not match!'	Error message: 'Password does not match!'
6	ExistingUser1	ExistingUser1@gmail.com	Pw12345!	Pw12345!	Error message: 'Registration failed. Username or email already exists.'	Error message: 'Registration failed. Username or email already exists.'

## 2.2 get\_carparks() Method

### 2.2.1 Control Flow Graphic



### 2.2.2 Basis Path Testing

**Cyclomatic complexity:** | Decision Points | + 1 = 3 + 1 = 4

#### **Basis Paths**

1. Baseline Path: 1, 2, 3, 4, 6, 7, 8, 11, 12, 13, 14, 15, 10
2. Basis Path 2: 1, 2, 3, 4, 5, 10
3. Basis Path 3: 1, 2, 3, 4, 6, 7, 8, 9, 10
4. Basis Path 4: 1, 2, 3, 4, 6, 7, 8, 11, 12, 9, 10

Test Input	Expected Output	Actual Output
Vehicle_type = Car/Van  Destination = CALTEX YISHUN, 1, YISHUN STREET 11, CALTEX YISHUN, 1 YISHUN STREET 11 CALTEX YISHUN SINGAPORE 768642, 768642	System displays message: "Search Successful" and show the display top 4 carparks near the destination.	System displays message: "Search Successful" and show the display top 4 carparks near the destination
Vehicle_type = Car/Van or Motorcycle or Heavy  Destination = (blank)	System displays message: "Destination cannot be empty. Please input a destination."	System displays message: "Destination cannot be empty. Please input a destination."
Vehicle_type = Motorcycle  Destination =CALTEX YISHUN, 1, YISHUN STREET 11, CALTEX YISHUN, 1 YISHUN STREET 11 CALTEX YISHUN SINGAPORE 768642, 768642	System displays message: "No carpark found."	System displays message: "No carpark found."
Vehicle_type = Motorcycle  Destination =CALTEX YISHUN, 1, YISHUN STREET 11, CALTEX YISHUN, 1 YISHUN STREET 11 CALTEX YISHUN SINGAPORE 768642, 768642	System displays message: "Search Successful" and display the top 4 carparks near the destination	System displays message: "Search Successful" and display the top 4 carparks near the destination

# Demo Script

## 1. Introduction

Hello everyone. We are Team A6 from SCSC. In this presentation, we will introduce our car park finding web application, SpotOn, and conduct a live demonstration to illustrate how it works.

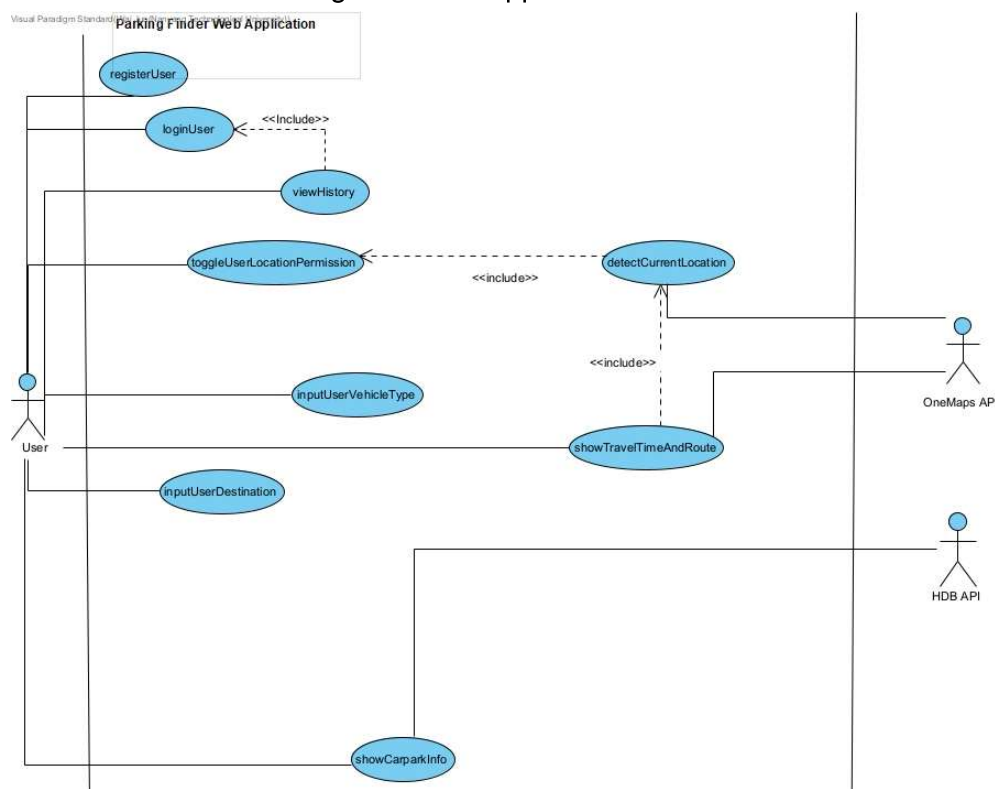
### 1.1 Problem Statement

In this fast-paced Singaporean society, convenience and efficiency are top priorities for many Singaporeans. Meanwhile, Singapore's vehicle population exceeded 1 million for the first time in July 2024. Finding an available parking spot in Singapore during peak times is a significant challenge, leading to wasted time and money.

As a result, we identified a key problem: the need for a reliable, user-friendly system that helps drivers find available parking spaces. SpotOn solves this issue for Singaporeans driving cars by offering real-time car park availability updates and other relevant information about nearby car parks from their destination, helping them make better decisions in choosing the most spot on and optimal parking space.

### 1.2 Use Case Diagram

This is the Use Case Diagram of our application.



### 1.3 Main Functions of Our App

Let me walk you through how a user using SpotOn would find their ideal parking spot near their destination.

Firstly, on the landing page, a user has three options: Register, Login or enter the app as a guest. The register function allows the user to create an account with their unique username and password, using their email. The login function allows existing users to log into the app smoothly. If the user forgets password, the user can click on 'forget password' to reset their password. The guest option allows users to use the app without having an account, however their past carpark search histories will not be logged into our database.

Next, let's say the user is past the landing page and is ready to find that parking spot. The user will be directed to the InputDestination page where they can choose their vehicle type from three options, Car, Motorcycle, or Heavy vehicle. Afterwards, the user will key in their destination in this search box that will smartly recommend the actual address name of the destination they had in mind. These actual addresses are recognised by Singapore's OneMap API, which will aid in locating car parks later.

After both vehicle types and destinations are chosen, the user will be directed to an InputCarpark page showing a list of nearby car parks and a car park finding map.

In the list of car parks the user can choose from, each car parks' address, lot availability, distance from destination, parking rates and times, gantry height and so on are explicitly shown. This gives the user a clear list of nearby car parks and each of their attributes so that users can make an informed decision. The map below shows the location pins of: the real time current location of the user, car parks within a close radius from chosen destination, and the chosen destination keyed in earlier. On the map, users can click on the car park's location pin they wish to visit, and a routing line will appear alongside step-by-step directions, such as "Turn right onto Road A." Users can hover and click their cursor over each direction to view its position along the suggested route.

After each successful search, the search history function will save the search history of the query a registered user has submitted. Registered users can view their search history by clicking on the history icon on the top right or left corner of SpotOn's InputDestination or InputCarpark page. Search history information includes the carpark chosen and time the search was done.

#### **1.4 External APIs used**

Some of the external APIs we used include:

##### OneMap API

OneMap is Singapore's authoritative national map. We use the OneMap API to perform location-based services, such as geocoding user input addresses to coordinates and routing. OneMap integrates well with local systems and provides detailed information on addresses, landmarks, and infrastructure, enhancing the overall user experience and making it highly relevant for Singaporean users finding car parks in Singapore.

(Source: <https://www.onemap.gov.sg/docs/>)

##### HDB Carpark Information API

We use the HDB Carpark Availability API from data.gov.sg to retrieve live carpark lot availability information. This ensures that our users always have access to the latest parking data across Singapore. This API retrieves the latest car park information every minute.

(Source:

[https://data.gov.sg/datasets/d\\_ca933a644e55d34fe21f28b8052fac63/view#tag/default/GET/transport/carpark-availability](https://data.gov.sg/datasets/d_ca933a644e55d34fe21f28b8052fac63/view#tag/default/GET/transport/carpark-availability))

## 2. Live Demo

### Registration Interface

This is the landing page of our website. As a new user, they would have to register for an account or continue as a guest. Registration can be done by clicking on the “Register” button. I will now register an account using some invalid test cases.

Pres Registration

Username: (BLANK)

Email Address: NewUser1@gmail.com

Password: Pw12345!

Re-enter Password: Pw12345!

Press Sign Up

If there are missing fields, the system will prompt the user to fill in the blank fields.

Username: NewUser1

Email Address: @gmail.com or NewUser1@gmail

Press Sign Up

The system also checks if the email is valid. As shown, if the email is invalid, the system will prompt the user to enter a valid email address.

Email Address: NewUser1@gmail.com

Password: pw12345!

Re-enter Password: pw12345!

Press Sign Up

The password here is missing an uppercase character. Hence the password does not meet the requirements, and the system will display an error showing that user’s password must contain at least 8 characters, at least 1 uppercase character, and at least 1 special character.

Password: Pw12345!

Re-enter Password: pw12344!

Press Sign Up

If the password and the re-entered password does not match, the system will inform the user that the password does not match.

Before this, I created an account using the following username and email address:

Username: ExistingUser1  
Email Address: ExistingUser1@gmail.com  
Press Sign Up

The registration failed and the system will inform the user that the username or email already exists.

Username: NewUser1  
Email Address: NewUser1@gmail.com  
Press Sign Up

If everything is valid, the system will display 'Registration successful' and bring us to the Login page.

### Login Interface

Username: newuser1  
Password: Pw12345!  
Press Login

We first try with an invalid username. As we can see, the system will display 'Login failed: Invalid username or password.'

Username: NewUser1  
Password: pw12345!  
Press Login

The same error message will be shown if the password is incorrect.

Password: Pw12345!  
Press Login

Only when both username and password are correct, the system will display 'Login successful' and direct the user to the InputDestination Interface.

### History Interface



Before searching for the carpark, we can see that the history page shows an empty record, as the user hasn't gotten a route to any carpark yet.

#### InputDestination Interface

vehicle\_type: Car/Van or Motorcycle or Heavy  
Destination : (blank)

When the user clicks on the Find Parking! button without inputting a destination, the system will display "Destination cannot be empty. Please input a destination."

vehicle\_type: Car/Van  
Destination : CALTEX YISHUN, 1, YISHUN STREET 11, CALTEX YISHUN, 1 YISHUN STREET 11 CALTEX YISHUN SINGAPORE 768642, 768642

After selecting the vehicle type and the destination, the user will be able to find some nearby car parks from the destination.

The user can select "show route" from any car park listed. After that, the system will display the calculated distance and travel time from the current user location to the designated car park. On the map, users can click on the car park's location pin they wish to visit, and a routing line will appear alongside step-by-step directions. Users can hover and click their cursor over each direction to view its position along the suggested route.

#### History Interface

Now, the history interface will record the details of the carpark and our visit time.

### **3. Good SWE Practices**

Next, we will share some good SWE practices that our team has adopted.

#### **3.1 Documentation code comments & Pydoc**

We ensure the system is well documented by writing clear code comments and docstrings that detail the purpose, attributes and return type of each function and class. This ensures code readability for collaboration as the comments and docstrings help our team better understand what the code does quickly, allowing us to work together more efficiently.

The docstrings are then used to generate Pydoc, so descriptions of our classes, functions, their parameters and return types can be viewed quickly in a structured, consistent format. This documentation helps our team easily understand how to interact with the code without needing to dive deep into the code itself, which allows for future extensibility by making it easier to understand and build upon the existing codebase.

#### **3.2 Test-driven development**

To guide the design and implementation of our critical components, we ensure to apply test-driven development. By writing tests before implementation, we clarify requirements

early and build a safety net that supports refactoring. For example, when developing our user registration flow, we first created comprehensive test cases that covered all the validation steps shown in our control flow diagram. First, we wrote tests for each validation path which defines the expected behavior. Before writing any implementation code, our tests clearly specified what should happen when a user submits registration information that is valid or invalid. Our Implementation then followed the test cases, with continuous validation to ensure we have not broken existing functionality.

## 4. System Design and Architecture

Next, we will share more about our system design

### 4.1 System Architecture

We have adopted a layered monolithic architecture for our system, separating the data access layer from the business logic. The communication goes top-down, from the presentation layer, through the application logic layer, and to the persistent data layer. This enhances the modularity, maintainability, and extensibility of the system as each layer is responsible for a specific set of functionalities, and can evolve independently without affecting the others. For instance, if we need to change the way data is stored or add a new database feature, we can do so in the persistent data layer without impacting the presentation layer or business logic.

While a monolithic architecture provides simplicity, ease of development coordination, and lower operational overhead during early-stage development, it is also **future-proofed** to transition into a **microservices architecture**. If specific features eventually require independent scalability or deployment, they can be decoupled and evolved as separate services.

### 4.2 Class Diagram

Our class diagram also showcases this layered architecture.

### 4.3 Design Principles

We integrated key design principles into our system to enhance maintainability and scalability.

First, our code follows a good **Separation of Concerns**, where we separate business logic from database logic, connected by an appropriate connector. For example, our User class handles high-level update logic and manages its own attribute, but does not write SQL queries directly. All database interactions are delegated to userQueries, instantiated through DBFactory. This allows for easier modification of components, like if our database changes, we only need to update the database layer and not our entire user layer.

Secondly, our classes are well-**modularised** and adheres to the **Single Responsibility Principle**. Our classes each have a clear single responsibility, and focuses only on a single

task or functionality. For instance, User class is only focused on user-related operations like login, logout, and updating attributes.

Thirdly, we apply the **Open-Closed Principle**. By using **Factory** and **Façade** patterns, we use abstraction to keep our system open for extension but closed for modification. For example, if new user objects are created, we can extend DBFactory to produce them without modifying existing logic.

#### 4.4 Design Pattern

For our design pattern, first, we have implemented the **Model-View-Controller (MVC)** design pattern, where there is a clear separation between presentation (View), application logic (Controller), and data (Model). Separating UI from core data Model and using Controller to link between them allows the frontend and backend to be managed in smaller, separate components, making our code more scalable and maintainable.

Secondly, our code adheres to **Factory** pattern, which simplifies database management and abstracts the creation logic by centralizing query creation through a single DBFactory class. Each entity only needs to reference their respective DatabaseQueries to handle database interactions, which keeps our system clean and reduces dependencies. This approach makes it easy to add new query types in the future without disrupting the existing code. This ensures efficient and organized database handling.

Thirdly, we implemented the **Façade** pattern for most of our control classes. Our controller classes provide simplified interfaces to the complex subsystem of data queries. An example is our CarparkCtrl class. The CarparkCtrl class provides a simplified interface to the API, the Carpark entity, and Carpark database access methods. This abstracts complexity so the caller does not need to deal with raw API responses, parsing, or database mechanics and can just call the CarparkCtrl class. This approach allows for encapsulation, maintainability, and reusability.

## 5. Traceability

For traceability, we will look into the implementation of the registration use case.

### 5.1 Use-case Description

The main flow of this use case is for the user to register an account in SpotOn. The user must use a valid username, email, and password to register an account.

### 5.2 Sequence diagram

From the sequence diagram, we can see how a user can register an account. This process is handled across three types of classes in the system: Boundary, Control, and Entity classes

### 5.3 Class diagram

From that, we can see how the use case and sequence diagram combine to form the class diagram with the different boundary, entity, and control classes.

### 5.4. Good Design Principles

We have implemented some good design principles in our system.

Solid Responsibility Principle

Each of classes has a specific responsibility. This improve the changeability of the code, with simple and independent units interacting through well defined interface

Open-Closed Principle

UserCtrl class is open for extension but is closed for modification. If there is a new validation check method, it is possible to add into it without modified the code in UserCtrl class.

### 5.5. Testing

We conducted white box testing for the register() function. This allows us to examine the internal code and structure of software, ensuring thorough coverage of all possible execution paths.

In this testing, we identify 5 binary decision points hence the cyclomatic complexity is 6 and there are 6 basis paths.

The first path occurs when all the decision points are true, which concludes the user passes through all the validation checks and registers an account successfully. While other 5 paths means the user did not go through the validation checks such as path 3 and path 4 are invalid email address and password format.

By doing the testing, this helps improve the robustness and reliability of the registration feature.

## **6. Novelty of system**

There are 2 novelty system in our application. First is Localised Route Planning. The system focuses specifically on Singapore's road network, offering precise, relevant routing within the local context.

The second is the Car Park Recommendation near the Destination. The system will automatically suggest nearby car parks after selecting a destination, reducing the hassle of manual searching and helping users plan their journey more efficiently.

That's all from our presentation. Thank you.