A black and white logo

Description automatically generated

Final Report – Elements of Software Construction 50.003

Cohort 4 Group 1

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| Ong Jung Yi | 1006655 |
| Luvyn Sequeira | 1007081 |
| Stavya Sharma | 1006933 |
| Zhuang Yang Kun | 1006933 |
| Yee Jia Zhen | 1006969 |
| Lee Jun Hui Ryan | 1006652 |
| Luong Hung | 1007160 |

Contents

[1 Changes made since PM3 3](#_Toc174312863)

[2 Requirement and Design 3](#_Toc174312864)

[2.1 Use Case Diagram 3](#_Toc174312865)

[2.1.1 Use Case Description (With Respective Sequence Diagram) 4](#_Toc174312866)

[HBS\_UC\_1 Search Hotel 4](#_Toc174312867)

[HBS\_UC\_2 Book Hotel 5](#_Toc174312868)

[HBS\_UC\_3 Send Booking Confirmation 7](#_Toc174312869)

[HBS\_UC\_4 Cancel Booking 8](#_Toc174312870)

[HBS\_UC\_5 Handle Refunds 9](#_Toc174312871)

[HBS\_UC\_6 Send Cancellation Confirmation 10](#_Toc174312872)

[CAS\_UC\_1 Login 11](#_Toc174312873)

[CAS\_UC\_2 Register 12](#_Toc174312874)

[CAS\_UC\_3 Update Account Details 13](#_Toc174312875)

[CAS\_UC\_4 View Booking 14](#_Toc174312876)

[2.2 Domain Class Diagram 15](#_Toc174312877)

[2.3 Solution Class Diagram 16](#_Toc174312878)

[3 Implementation Challenges 17](#_Toc174312879)

[3.1 Algorithmic Challenges 17](#_Toc174312880)

[3.2 Engineering Challenges 17](#_Toc174312881)

[3.3 Testing Challenges 18](#_Toc174312882)

[4 Testing 19](#_Toc174312883)

[4.1 Unit Testing 19](#_Toc174312884)

[4.1.1 Backend 19](#_Toc174312885)

[4.2 Frontend Testing 26](#_Toc174312886)

[4.3 Integration Testing 32](#_Toc174312887)

[4.4 System Testing 37](#_Toc174312888)

[4.5 Robustness Testing (Fuzzing) 37](#_Toc174312889)

[5 Contributions to United Nation SDG 38](#_Toc174312890)

[6 Diversity and Inclusivity 38](#_Toc174312891)

# Changes made since PM3

* Updated use case diagram to match use case names in use case description.
  + “Manage Account” changed to “Update Account Details”
  + “Manage Booking” changed to “View Booking”
* Refined domain class diagram and solution class diagram with new attributes and operations used in the final product.
* Updated test case documentation to associate each test case with its corresponding use case.
* Implemented more comprehensive test cases for the backend.
* Implemented integration testing and system (end-to-end testing).

# Requirement and Design

## Use Case Diagram

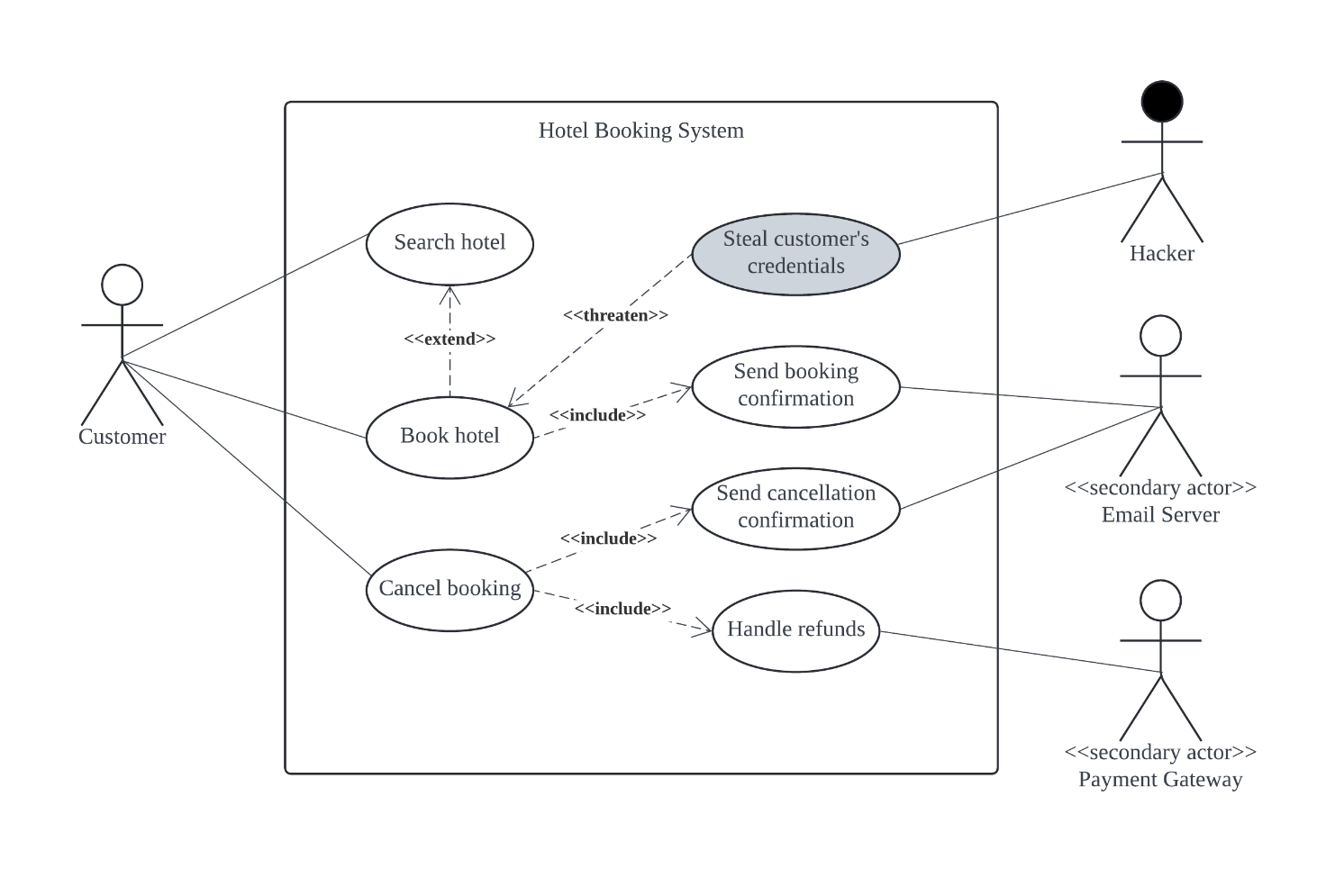


Figure : Hotel Booking System UC Diagram

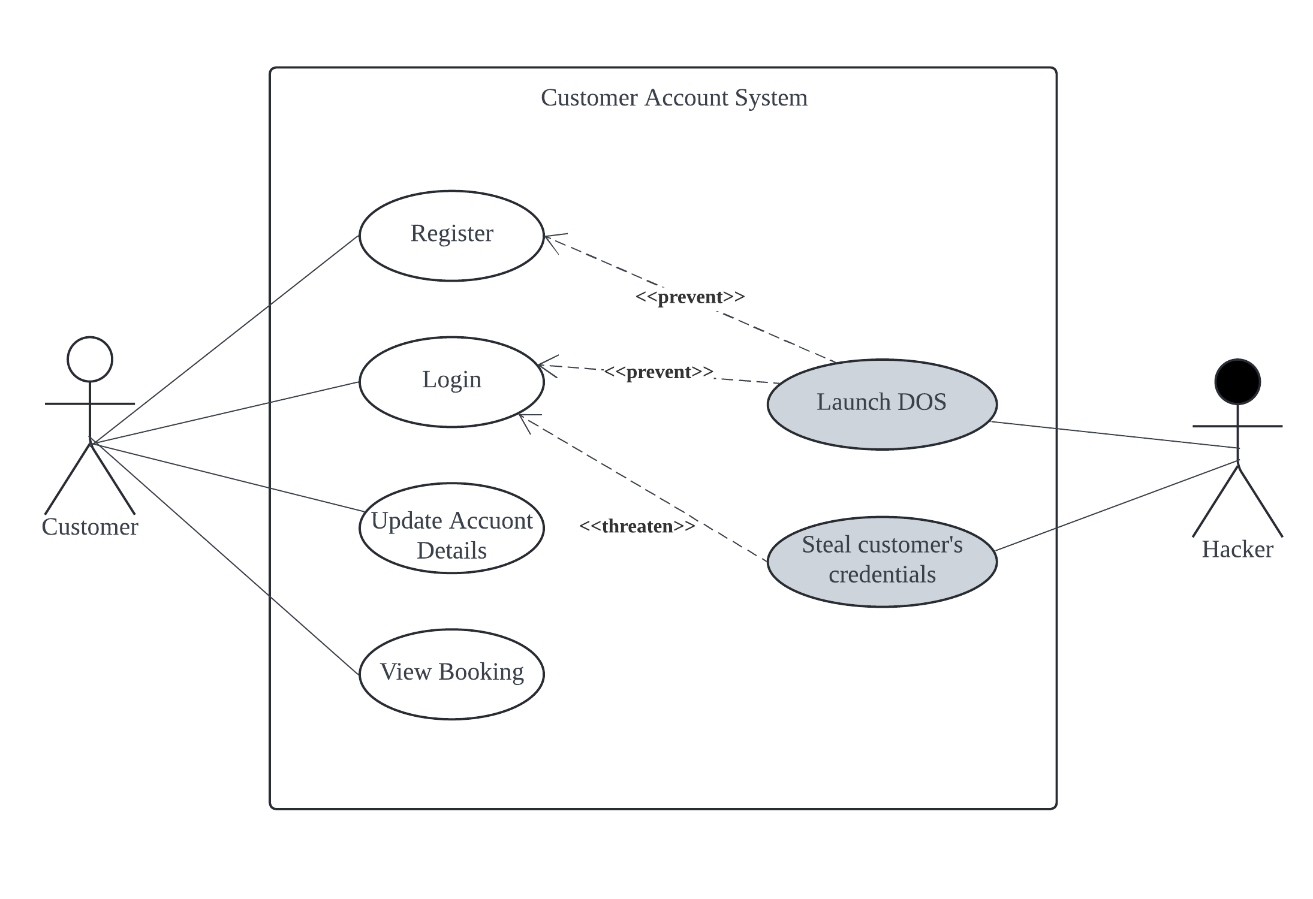


Figure : Customer Account System UC Diagram

### Use Case Description (With Respective Sequence Diagram)

#### HBS\_UC\_1 Search Hotel

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_1 |
| Use Case Name | Search Hotel |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case allows customers to search for hotels by destination, dates of stay, number of guests, and number of rooms. For each field, the customer selects input values from a predefined list. Upon submission, the system displays a list of matching hotels along with the cheapest room for each hotel. Customers can filter the list by star ratings, guest ratings, and price range. The customer can then select a specific hotel from the list to view a list of matching room types provided and any other additional information. |
| Primary Actor | Customer |
| Secondary Actor | Ascenda Hotel API |
| Precondition | 1. Hotel Listing API is available and functioning |
| Postconditions | 1. System displays details for specific hotel |
| Main Flow | 1. Customer inputs destination, date of stay, number of guests and number of rooms (select from list of valid input values) 2. Customer submits inputs 3. System validates inputs 4. System retrieves list of matching hotels 5. System displays list of matching hotels 6. Customer filters results by star ratings, guest ratings, and price range 7. System retrieves list of filtered matching hotels 8. System displays list of filtered matching hotels 9. Customer selects specific hotel from result list 10. System displays details of the selected hotel such as matching room details, ratings, and any other additional information 11. Extension point : customer books hotel (HBS\_UC\_2) 12. Use case ends |
| Alternate Flow | 3a. Missing mandatory search criterion/criteria   1. System prompts customer for the missing mandatory search fields 2. Use case resumes at main flow step 1 |
| Sequence Diagram | A diagram of a project  Description automatically generated |

#### HBS\_UC\_2 Book Hotel

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_2 |
| Use Case Name | Book Hotel |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case allows the customer to book a hotel room and make payment via payment gateway. To book a hotel room, the customer enters the following information: first name & last name, phone number & email address, special requests to hotel, payment information (credit card number, expiry date, CVV/CVC), and the billing address. If the customer is logged in, then certain fields are automatically filled in (depending on what information the customer has saved in their account). After payment is processed and booking is confirmed, the system will trigger the send booking confirmation use case (HBS\_UC\_3) in which the email server will send an email to the customer with booking confirmation and information. |
| Include Use Cases | 1. Send Booking Confirmation (HBS\_UC\_3) |
| Extends Use Cases: | 1. Search hotel (HBS\_UC\_1) |
| Primary Actor | Customer |
| Secondary Actor | Payment Gateway |
| Precondition | 1. Customer has selected (searched up) a hotel 2. Payment Gateway is available and functioning |
| Postconditions | 1. Hotel room is booked and customer receives a booking confirmation via email 2. Booking details are recorded in database (via Create Booking API) |
| Main Flow | 1. Customer selects room to book 2. Customer enters required booking information (e.g. customer’s email) 3. Customer submits booking information 4. System validates booking information 5. System sends payment request to Payment Gateway 6. Customer enters payment details 7. Customer submits payment details 8. Payment Gateway validates payment details 9. Payment Gateway processes the payment and returns a confirmation 10. System confirms the booking and create booking in database 11. System triggers send booking confirmation use case (HBS\_UC\_3) 12. System displays “booking success” message 13. Use case ends |
| Alternate Flow | 2a. Customer is logged in   1. System auto fills form based on information saved to customer’s account 2. Customer fills in remaining fields 3. Use case resumes at main flow step 3   4a. Invalid or missing booking information   1. System prompts customer for missing/incorrect information 2. Use case resumes at main flow step 2   5a. Customer is logged in and has saved payment information   1. System auto fills payment details form based on payment information saved to customer’s account 2. Customer can modify the fields 3. Use case resumes at main flow step 7   6a. Customer is logged in and does not have payment information saved to account   1. System asks customer if they wish to save payment information to account 2. System says payment information to account if customer indicates to do so 3. Use case resumes at main flow step 8   7a. Invalid or missing payment details (that can be detected without Payment Gateway)   1. System prompts customer to fill in missing details or correct incorrect information 2. Use case resumes at main flow step 6   9a. Payment declined by Payment gateway   1. System informs customer that the payment was declined 2. a) Customer can retry the payment with different payment details by resuming use case at main flow step 6 b) Customer can cancel the booking process. Use case ends. |
| Sequence Diagram | A diagram of a diagram  Description automatically generated |

#### HBS\_UC\_3 Send Booking Confirmation

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_3 |
| Use Case Name | Send Booking Confirmation |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case allows the system to send a booking confirmation email to a customer (via the email server) after they have made a successful booking. This use case is included in the book hotel use case (HBS\_UC\_2) as it is part of the booking process, triggered when the customer’s booking is confirmed. The email should notify customer that their booking was successful and provide them all relevant information: destination ID, hotel ID, booking display information (number of nights, start date, end date, adults, children, message to hotel, room types), price, booking reference (booking ID), guest information (salutation, first name, last name), payee information (payment ID, payee ID). The email should also contain a unique URL that redirects the customer to a page that allows them to cancel that booking if they so wish. |
| Primary Actor |  |
| Secondary Actor | Email Server |
| Precondition | 1. Email server is available and functioning 2. Customer made successful booking (system triggers this use case) |
| Postconditions | 1. Customer receives a booking confirmation email with all relevant details |
| Main Flow | 1. System compiles customer’s booking details 2. System generates booking confirmation from compiled data 3. System composes an email with booking confirmation details 4. System sends composed email to the customer’s email address via the email server 5. Use case ends |
| Alternate Flow | 4a. Email server fails to send email   1. System retries sending the email via email server a specified number of times 2. If the email still cannot be sent, the system logs the failure and notifies system administrator 3. Use case ends |

#### HBS\_UC\_4 Cancel Booking

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_4 |
| Use Case Name | Cancel Booking |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case enables customers to cancel their previously made hotel booking. This use case includes the send handle refunds use case (HBS\_UC\_5) which triggers after the system processes cancellation, in which the system interacts with the payment gateway for any necessary refunds. Upon successful cancellation, the system will trigger the included send cancellation confirmation use case (HBS\_UC\_6), in which a cancellation confirmation email is sent to the customer. This use case is reached by customers via the special URL in their booking confirmation email, unique to that particular booking. |
| Include Use Cases | 1. Handle Refunds (HBS\_UC\_5) 2. Send Cancellation Confirmation (HBS\_UC\_6) |
| Primary Actor | Customer |
| Secondary Actor |  |
| Precondition | 1. Customer has existing booking that they wish to cancel 2. Payment gateway and email server are available and functioning |
| Postconditions | 1. Booking is cancelled 2. Customer receives a cancellation confirmation email 3. Booking status is updated to ‘cancelled’ in database |
| Main Flow | 1. Customer navigates to booking cancellation page unique to their booking, via URL in booking confirmation email 2. System prompts customer to confirm cancellation 3. Customer confirms booking cancellation 4. System processes cancellation request 5. System triggers handle refunds (HBS\_UC\_5) use case 6. System updates booking status to ‘cancelled’ in database 7. System triggers send cancellation confirmation (HBS\_UC\_6) use case 8. System displays “Booking cancelled successfully.” 9. Use case ends |
| Alternate Flow | 4a. Too late to cancel booking   1. System notifies customer that it is too late to cancel the booking 2. Use case ends   4b. Booking status is already cancelled   1. System notifies customer that the booking is already cancelled 2. Use case ends   5a. Refund request rejected by payment gateway   1. System informs customer that refund request has been rejected and thus booking cancellation request is also incomplete 2. Use case ends |
| Sequence Diagram | A diagram of a diagram of a square  Description automatically generated |

#### HBS\_UC\_5 Handle Refunds

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_5 |
| Use Case Name | Handle Refunds |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case enables the process of refunding the customer when they cancel a booking. This use case is included in the cancel booking use case (HBS\_UC\_4), triggered after the cancellation request is processed (refund request allowed by system). The system interacts with the payment gateway to process the refund request. |
| Primary Actor |  |
| Secondary Actor | Payment Gateway |
| Precondition | 1. Booking cancellation request is processed and confirmed by system 2. Payment gateway is available and functioning |
| Postconditions | 1. Payment gateway processes and confirms refund request transaction |
| Main Flow | 1. System fetches booking details needed for refund request 2. System sends a refund request to the payment gateway with the calculated amount and transaction details 3. The payment gateway processes the refund 4. The payment gateway returns confirmation of refund to the system 5. Use case ends |
| Alternate Flow | 3a. Refund request rejected by payment gateway   1. System informs customer that refund request has been rejected and thus booking cancellation request is also incomplete 2. Use case ends |

#### HBS\_UC\_6 Send Cancellation Confirmation

|  |  |
| --- | --- |
| Use Case ID | HBS\_UC\_6 |
| Use Case Name | Send Cancellation Confirmation |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case allows the system to send a cancellation confirmation email to a customer (via the email server) after they have successfully cancelled a booking. This use case is included in the cancel booking use case (HBS\_UC\_4), triggered when the booking cancellation is confirmed. The email should notify the customer that their booking cancellation was successful. |
| Primary Actor |  |
| Secondary Actor | Email Server |
| Precondition | 1. Email server is available and functioning 2. Customer successfully cancelled booking (refunded) |
| Postconditions | 1. Customer receives an email confirming their booking cancellation |
| Main Flow | 1. System composes an email notifying customer of successful booking cancellation 2. System sends composed email to the customer’s email address via the email server 3. Use case ends |
| Alternate Flow | 2a. Email server fails to send email   1. System retries sending the email via email server a specified number of times 2. If the email still cannot be sent, the system logs the failure and notifies system administrator 3. Use case ends |

#### CAS\_UC\_1 Login

|  |  |
| --- | --- |
| Use Case ID | CAS\_UC\_1 |
| Use Case Name | Login |
| Author | Ryan Lee |
| Date Created | 3/7/2024 |
| Description | This use case enables customers to login to their accounts before making a booking. While logging in is not required for booking, it provides additional convenience by autofilling the customer’s saved details and preferences when making a booking. Once a customer is logged in, they can proceed to fill in and save said details and preferences to their account. |
| Primary Actor | Customer |
| Secondary Actor |  |
| Precondition | 1. Customer has to have valid account |
| Postconditions | 1. System displays the relevant homepage with customer authenticated and logged in. |
| Main Flow | 1. Customer navigates to login page 2. Customer enters username and password 3. Customer submits username and password 4. System validates the username and password 5. System processes login 6. System verifies the username and password 7. System returns to homepage (logged in) 8. Use case ends |
| Alternate Flow | 4a. Missing username and/or password   1. System prompts customer for username and password 2. Use case resumes at main flow step 2   6a. Invalid username and/or password   1. System displays “Invalid username and/or password” message 2. System prompts customer for username and password 3. Use case resumes at main flow step 2   7a. Customer wishes to save booking details & preferences   1. Customer navigates to “Manage Account” page 2. Customer enters details & preferences they wish to save to their account (for autofill purposes) 3. Customer submits form 4. System saves information to database 5. Use case ends |
| Sequence Diagram | A diagram of a login diagram  Description automatically generated |

#### CAS\_UC\_2 Register

|  |  |
| --- | --- |
| Use Case ID | CAS\_UC\_2 |
| Use Case Name | Register |
| Author | Yee Jia Zhen |
| Date Created | 4 July 2024 |
| Description | This use case is for new users to create an account for the hotel booking web application. |
| Primary Actor | Customer |
| Secondary Actor |  |
| Precondition | 1. Customer is not an existing user |
| Postconditions | 1. Customer account created |
| Main Flow | 1. Customer navigates to the register account page 2. Customer fill in their details 3. Customer submit the register account form 4. System validates submitted details 5. System process registration 6. System adds customer to database 7. System displays registration successful message 8. System returns to log in page 9. Use case ends |
| Alternate Flow | 4a. Username or email taken   1. System prompts customer for new username or email 2. Use case resumes at main flow step 3   4b. Weak password   1. System prompts customer for a strong password 2. Use case resumes at main flow step 3 |
| Sequence Diagram | A diagram of a diagram  Description automatically generated |

#### CAS\_UC\_3 Update Account Details

|  |  |
| --- | --- |
| Use Case ID | CAS\_UC\_3 |
| Use Case Name | Update Account Details |
| Author | Yee Jia Zhen |
| Date Created | 4 July 2024 |
| Description | This use case allows customers to view and amend their personal information that they have provided during registration. |
| Primary Actor | Customer |
| Secondary Actor |  |
| Precondition | 1. Customer has a valid account |
| Postconditions | 1. Customer account details successfully updated |
| Main Flow | 1. Customer navigates to account page 2. Customer edit their personal information 3. Customer save their edits 4. System validates their personal information 5. System process account updates 6. System updates the database 7. System display new information 8. Use case ends |
| Alternate Flow | 4a. Username or email taken   1. System prompts customer for new username or email 2. Use case resumes at main flow step 3   4b. Weak password   1. System prompts customer for a strong password 2. Use case resumes at main flow step 3 |
| Sequence Diagram | A diagram of a diagram  Description automatically generated |

#### CAS\_UC\_4 View Booking

|  |  |
| --- | --- |
| Use Case ID | CAS\_UC\_4 |
| Use Case Name | View Booking |
| Author | Yee Jia Zhen |
| Date Created | 4 July 2024 |
| Description | This use case allows customers to view their hotel booking(s) information and gives them the option to cancel their booking(s). |
| Primary Actor | Customer |
| Secondary Actor |  |
| Precondition | 1. Customer has a valid account |
| Postconditions | 1. Customer’s booking(s) successfully cancelled 2. Customer booking(s) successfully changed. |
| Main Flow | 1. Customer navigates to bookings page 2. System displays customer’s booking(s) 3. Use case ends |
| Alternate Flow | 2a. Customer wishes to cancel booking   1. Customer selects cancel booking option, triggering cancel booking use case (HBS\_UC\_4) 2. Use case ends |
| Sequence Diagram | A diagram of a diagram of a product  Description automatically generated with medium confidence |

## A screenshot of a computer screen Description automatically generatedDomain Class Diagram

## Solution Class Diagram

The diagram is hard to see in this report and there is watermark. You may view it here (you will need to register for a Visual Paradigm Online account): <https://online.visual-paradigm.com/share.jsp?id=333531383937322d31> A close-up of a computer screen

Description automatically generated

# Implementation Challenges

## Algorithmic Challenges

**Challenge 1: Slow API Response Times When Fetching Hotel Listings**

Fetching the list of hotels from Ascenda’s API can be time-consuming, leading to delays and a poor user experience, especially when customers perform multiple searches for the same destination.

**Solution: Implemented a Caching System**

To address this issue, we implemented a caching system. When a customer searches for the same destination more than once, the hotel listings are retrieved from the cache rather than making a new API call. This significantly reduces the time required to retrieve hotel listings, making the application more responsive and improving the overall user experience.

**Challenge 2: Slow Loading of Hotel Details Impacting User Experience**

Hotel details, including images, names, and prices, are taking a significant amount of time to load, leading to poor user experience. This delay is particularly noticeable when displaying a large number of search results.

**Solution: Implemented Front-End Pagination**

To improve the user experience, we implemented front-end pagination. The application now loads only ten hotel results at a time, and only load the next ten when the customer scrolls all the way down the list of hotels. Instead of making multiple requests, a single request is made to the server's API to retrieve the entire list of hotels matching the customer’s search criteria. This data is stored locally, enabling quick and seamless navigation between pages without additional server requests.

## Engineering Challenges

**Challenge 1**: Integration of Login and Register Functionality  
Integrating login and registration functionality presented challenges, particularly related to the validation process and ensuring data privacy.  
  
**Solution:** Front-End Validation and Backend Security  
To address the slow validation at scale, we implemented front-end validation to ensure input is in the correct format before it is sent to the backend database. This reduces the load on the backend and speeds up the validation process. Additionally, we implemented hashing of passwords and ensured that the logic related to data privacy and security is handled on the server side, which is not accessible to users. This ensures that sensitive information is protected and enhances the overall security of the application.   
  
**Challenge 2:** Data Privacy and Security  
Ensuring the security and privacy of user data, especially during the login and registration processes, was a significant challenge.  
  
**Solution**: Implemented Hashing and Secure Protocols  
We addressed this by implementing strong hashing algorithms for passwords stored in the database. Additionally, we incorporated server-side logic for handling authentication and authorization, ensuring that these processes are not exposed to the client side.  
  
**Challenge 3**: Scalability of the Backend Infrastructure  
As the number of users increases, the backend infrastructure needs to handle a larger volume of requests efficiently.  
  
**Solution**: Implemented MVC Architecture  
To handle scalability, we adopted the Model-View-Controller (MVC) architecture for our backend infrastructure. This design pattern helps in separating the concerns of the application, making it more manageable and scalable. The MVC architecture divides the application into three interconnected components:  
  
    Model: Manages the data and business logic.  
    View: Handles the presentation layer and user interface.  
    Controller: Acts as an intermediary between Model and View, processing user input and interacting with the Model to update the View.

**Challenge 4:** Integrating with external APIs

Particularly the Ascenda Hotel API, presented challenges such as handling rate limits imposed by the API provider. These limits restricted the number of requests that could be made within a certain time frame, which was particularly problematic during testing and high-traffic scenarios.

**Solution:** To mitigate this, we implemented a request queueing system that managed and spaced-out API calls, ensuring compliance with rate limits while maintaining application performance. Additionally, a caching mechanism was introduced to store and reuse data from previous API responses, reducing the number of redundant calls.

## Testing Challenges

**Challenge 1: Jest configuration**

Jest uses CommonJS whereas our express app (server) uses ES6. As such, we had to configure such that ES6 is converted to CommonJS for the test cases to run.

**Solution**: **Look for other methods to convert CommonJS to ES6**

This process is challenging as we had to go through several methods for the conversion as the Jest mocking is not working as expected, with other errors such as “Jest is not defined” when Jest is already imported.

**Challenge 2: Jest conflicts**

Jest had some conflicts with our MySQL database connection code (db.js) that we use for our actual model functions.

**Solution: Create script for database connection meant for testing**

As a workaround, we created new code for a connection pool meant for test cases only (dbTest.js). At the same time, we ensured that a separate table is created and used for testing, to isolate them from the actual database.

# Testing

## Unit Testing

### Backend

Some notes:

* Some of our model functions use and make calls to Ascenda API. For those functions, we simply take in the inputs and pass it directly to the API for processing. Two reasons: 1) input formatting and basic validation already done by frontend before being passed to backend function and 2) API is a blackbox and we do not know how they process the inputs (we don’t know what range of values the API accepts and doesn’t). For instance, a set of parameter values might work on one day (results can be found) and not on another day (no results found, API returns 422). Thus for these functions, we are more so focused on testing whether they can handle the various responses from the API (e.g. 422 unprocessable) by mocking them.
* Models like the Booking model rely on our MySQL database. For the purposes of unit testing, we mock the response from the MySQL database.
  + Ensuring the actual MySQL database integrates together with the rest of the system is done with the integration testing (actual MySQL connection used, rather than mocked response).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| models/hotel.test.js  Testing Hotel model functionality | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| HM-001 | Hotel class constructor should create a Hotel instance correctly | None | Hotel data (id, name, latitude, longitude, address, rating, categories, description, amenities, image\_details) | Hotel instance with correct properties | Hotel instance exists |
| HM-002 | fetchHotelsByDestination should fetch hotels and return a Map of Hotel instances (success) | Ascenda API (via axios) mocked to return hotel data (2 hotels) | Destination id ‘destination1’ | Map where key is hotel ID and value is hotel instance  Map of size 2 | Map containing 2 Hotel instances with correct data exists |
| HM-003 | fetchHotelsByDestination should throw an error and log it if the request fails (failure) | Ascenda API (via axios) mocked to throw network error | Destination id ‘destination1’ | Thrown error with message ‘Error fetching hotels by destination’ | Error logged to console |
| HM-004 | fetchHotel should fetch a single hotel and return a Hotel instance (success) | Ascenda API (via axios) mocked to return single hotel data | Hotel id ‘1’ | Hotel instance with correct properties (matching mock data) | Hotel instance exist with correct data |
| HM-005 | fetchHotel should throw an error and log it if the request fails | Ascenda API (via axios) mocked to throw network error | Hotel id ‘1’ | Thrown error with message ‘Error fetching hotel by id’ | Error logged to console |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| models/hotelPrice.test.js  Testing HotelPrice model functionality | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| HP-001 | HotelPrice class constructor should create a HotelPrice instance correctly | None | HotelPrice data (id, searchRank, price, market\_rates) | HotelPrice instance with correct properties | HotelPrice instance exists |
| HP-002 | fetchHotelPricesByDestination should fetch hotels and return a Map of HotelPrice instances (success) | Ascenda API (via axios) mocked to return hotel price data (2 hotels) | Destination: 'destination1', Check-in: '2024-08-01', Check-out: '2024-08-05', Language: 'en', Currency: 'USD', Guests: 2 | Map where key is hotel ID (mocked as ‘T9cE’ and ‘obxM’) and value is HotelPrice instance  Map of size 2 | Map containing 2 HotelPrice instances with correct data exists |
| HP-003 | fetchHotelPricesByDestination should poll API multiple times until completion (success) | Ascenda API (via axios) mocked to return incomplete response twice, then complete response | Same as HP-002 | Map containing 2 HotelPrice instances | Console log shows ‘API polled 3 time(s)’ |
| HP-004 | fetchHotelPricesByDestination should throw an error and log it if the request fails (failure) | Ascenda API (via axios) mocked to throw network error | Same as HP-002 | Thrown error with message ‘Error fetching hotel prices by destination’ | Error logged to console |
| HP-005 | fetchHotelPricesByDestination handles exceeding API poll limit | Ascenda API (via axios) mocked to always return incomplete response | Same as HP-002 | Empty Map | Console log shows ‘Exceeded API poll limit’ |
| HP-006 | fetchHotelPricesByDestination handles Ascendas API 422 status (unprocessable inputs) | Ascenda API (via axios) mocked to reject with 422 status | Same as HP-002 | Empty Map | None |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| models/room.test.js  Testing Room model functionality | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| RM-001 | Room class constructor should create a Room instance correctly | None | Room data (hotelId, key, roomNormalizedDescription, free\_cancellation, roomDescription, long\_description, images, amenities, price, market\_rates) | HotelPrice instance with correct properties | HotelPrice instance exists |
| RM-002 | fetchRoomPrices should fetch room prices and return an array of Room instances (success) | Ascenda API (via axios) mocked to return room data (2 rooms) | hotelId: 'hotel1', destinationId: 'dest1', checkin: '2023-06-01', checkout: '2023-06-05', lang: 'en', currency: 'USD', guests: '2' | Array of 2 Room instances with correct properties | Console log shows "API polled 1 time(s)" |
| RM-003 | fetchRoomPrices should poll API multiple times until completion (success) | Ascenda API (via axios) mocked to return incomplete response twice, then complete response | Same as RM-002 | Map containing 2 HotelPrice instances | Console log shows "API polled 3 time(s)" |
| RM-004 | fetchRoomPrices should handle API poll limit exceeded | Ascenda API (via axios) mocked to always return incomplete response | Same as RM-002 | Empty Array | Console log shows ‘Exceeded API poll limit’ |
| RM-005 | fetchRoomPrices should handle Ascendas API 422 status | Ascenda API (via axios) mocked to reject with 422 status | Same as RM-002 | Empty Array | None |
| RM-006 | fetchRoomPrices should throw an error if the request fails | Ascenda API (via axios) mocked to throw a network error | Same as RM-002 | Thrown error with message ‘Error fetching room prices of given hotel’ | Error logged to console |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| models/booking.test.js  Testing Booking model functionality | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| BM-001 | Booking class constructor should create a Booking instance correctly | None | Booking data (bookingId, status, destinationId, hotelName, hotelId, roomKey, customerId, numberOfNights, startDate, endDate, numAdults, numChildren, msgToHotel, roomTypes, price, guestSalutation, guestFirstName, guestLastName, paymentId, payeeId) | Booking instance with correct properties | Booking instance exists |
| BM-002 | insertBooking should insert a booking and return the booking ID (insertId) | Database (via db.promisedConnectionPool) mocked to return successful insert result | Valid Booking object | Returned insertId (1) | Query called with correct SQL and parameters |
| BM-003 | insertBooking should throw an error when insertion fails | Database mocked to reject with an error | Valid Booking object | Thrown error with message 'Insertion failed' | Error logged: 'Database connection failed: Error: Insertion failed' |
| BM-004 | findBookingByBookingId should return a booking when found | Database mocked to return a single booking row | bookingId: 1 | Booking instance with correct properties | Query called with correct SQL and parameters |
| BM-005 | findBookingByBookingId should return null when booking is not found | Database mocked to return empty result | bookingId: 999 | null | Query called with correct SQL and parameters |
| BM-006 | findBookingByBookingId should throw an error when query fails | Database mocked to reject with an error | bookingId: 1 | Thrown error with message 'Query failed' | Error logged: 'Database query failed: Error: Query failed' |
| BM-007 | findBookingByCustomerId should return an array of bookings when found | Database mocked to return multiple booking rows | customerId: 123 | Array of Booking instances | Query called with correct SQL and parameters |
| BM-008 | findBookingByCustomerId should return null when no bookings are found | Database mocked to return empty result | customerId: 999 (non-existent customer) | null | Query called with correct SQL and parameters |
| BM-009 | findBookingByCustomerId should throw an error when query fails | Database mocked to reject with an error | customerId: 123 | Thrown error with message 'Query failed' | Error logged: 'Database query failed: Error: Query failed' |
| BM-010 | updateBookingStatus should update the booking status | Database mocked to return successful update result | bookingId: 1, status: 'cancelled' | Object with affectedRows: 1 | Query called with correct SQL and parameters |
| BM-011 | updateBookingStatus should return null when no booking is found to update | Database mocked to return result with no affected rows | bookingId: 999 (non-existent booking), status: 'cancelled' | null |  |
| BM-012 | updateBookingStatus should throw an error when update fails | Database mocked to reject with an error | bookingId: 1, status: 'cancelled' | Thrown error with message 'Update failed' | Query called with correct SQL and parameters |
| BM-013 | removeBooking should remove a booking | Database mocked to return successful delete result | bookingId: 1 | Object with affectedRows: 1 | Query called with correct SQL and parameters |
| BM-014 | removeBooking should throw an error when booking does not exist | Database mocked to return result with no affected rows | bookingId: 999 (non-existent booking) | Thrown error with message 'Booking with ID 999 does not exist.' | No specific post condition |
| BM-015 | removeBooking should throw an error when deletion fails | Database mocked to reject with an error | bookingId: 1 | Thrown error with message 'Deletion failed' | Error logged: 'Failed to remove booking: Error: Deletion failed' |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| models/customer.test.js  Testing Customer (i.e. user) model functionality | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| CM-001 | Customer constructor should create a valid Customer object without reset password fields | None | customerId: 1, username: 'mock\_customer', email: '[mock@example.com](mailto:mock@example.com)', password: 'password123', hp: '999-0009' | Customer instance with correct properties and null reset password fields | Customer instance exists |
| CM-002 | Customer constructor should create a valid Customer object with reset password fields | None | customerId: 1, username: 'mock\_customer', email: '[mock@example.com](mailto:mock@example.com)', password: 'password123', hp: '999-0009', resetPasswordToken: 'exampleToken', resetPasswordExpires: '2023-08-15' | Customer instance with all properties including reset password fields | Customer instance exists |
| CM-003 | insertCustomer should create a Customer entry and return the insertId | Database mocked to return successful insert result | Customer object with username: 'mock\_customer', email: '[mock@example.com](mailto:mock@example.com)', password: 'password123', hp: '999-0009' | Returned insertId (1) | Query called with correct SQL and parameters |
| CM-004 | insertCustomer should throw an error when insertion fails | Database mocked to reject with an error | Valid Customer object | Thrown error with message 'Insertion failed' | Error logged: 'Database connection failed: Error: Insertion failed' |
| CM-005 | findOneByCustomerId should return a Customer instance when found | Database mocked to return a single customer row | customerId: 1 | Customer instance with customerId: 1, username: 'mock\_customer' | Query called with correct SQL and parameters |
| CM-006 | findOneByCustomerId should return null when customer is not found | Database mocked to return empty result | customerId: 999 (representing non-existent customer) | null | Query called with correct SQL and parameters |
| CM-007 | findOneByCustomerId should throw an error when query fails | Database mocked to reject with an error | customerId: 1 | Thrown error with message 'Query failed' | Error logged: 'Database query failed: Error: Query failed' |
| CM-008 | findUserByEmail should return a Customer instance when found | Database mocked to return a single customer row | email: '[mock@example.com](mailto:mock@example.com)' | Customer instance with email: '[mock@example.com](mailto:mock@example.com)', customerId: 1, username: 'mock\_customer' | Query called with correct SQL and parameters |
| CM-009 | findUserByEmail should return null when no matching customers found | Database mocked to return empty result | email: '[abcdefghijklmnop@yahoo.sg](mailto:abcdefghijklmnop@yahoo.sg)' (representing email that does not have matching customer) | null | Query called with correct SQL and parameters |
| CM-010 | findUserByEmail should throw an error when query fails | Database mocked to reject with an error | email: '[mock@example.com](mailto:mock@example.com)' | Thrown error with message 'Query failed' | Error logged: 'Database query failed: Error: Query failed' |
| CM-011 | updateUser should return true if 1 or more entries affected | Database mocked to return result with affectedRows: 1 | customerId: 1, updateData: {email: "[new@new.com](mailto:new@new.com)"} | true | Query called with correct SQL and parameters |
| CM-012 | updateUser should return false if 0 entries affected | Database mocked to return result with affectedRows: 0 | customerId: 999, updateData: {email: "[new@new.com](mailto:new@new.com)"} | false | Query called with correct SQL and parameters |
| CM-013 | updateUser should throw an error when query fails | Database mocked to reject with an error | customerId: 1, updateData: {email: "[new@new.com](mailto:new@new.com)"} | Thrown error with message 'Query failed' | Error logged: 'Database update failed: Error: Query failed' |
| CM-014 | findUserByToken should return a Customer instance when found | Database mocked to return a single customer row with reset password fields | token: 'exampleToken' | Customer instance with resetPasswordToken: 'exampleToken', resetPasswordExpires: '2023-08-15', customerId: 1 | Query called with correct SQL and parameters |
| CM-015 | findUserByToken should return null when no matching customers found | Database mocked to return empty result | token: 'badToken' | null | Query called with correct SQL and parameters |
| CM-016 | findUserByToken should throw an error when query fails | Database mocked to reject with an error | token: 'exampleToken' | Thrown error with message 'Query failed' | Error logged: 'Database query failed: Error: Query failed' |

## Frontend Testing

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Client/src/searchBar/guestInput.js  Testing number of guests and rooms input functionality | | | | | | | | | | |
| Test Case ID | | Description | Preconditions | Input | Expected Output | | Post Condition | | | |
| GI-001 | | GuestInput Component renders input field with default values | None | Render the `GuestInput` component with `guests` as `{ adults: 1, children: 0, rooms: 1 }` | Input field displays '1 adults · 0 children · 1 room' | | Input field is correctly rendered | | | |
| GI-002 | GuestInput Component opens dropdown on input field click | | | | | None | | Click on the input field | Dropdown opens, displaying 'Adults', 'Children', and 'Rooms' text | Dropdown is visible |
| GI-003 | GuestInput Component increments and decrements adults count | | | | | Dropdown is open | | Click on increment (`+`) and decrement (`-`) buttons for adults count | `setGuests` function is called with updated `guests` values for adults | Adults count is updated |
| GI-004 | GuestInput Component increments and decrements children count | | | | | Dropdown is open | | Click on increment (`+`) and decrement (`-`) buttons for children count | `setGuests` function is called with updated `guests` values for children | Children count is updated |
| GI-005 | GuestInput Component increments and decrements rooms count | | | | | Dropdown is open | | Click on increment (`+`) and decrement (`-`) buttons for rooms count | `setGuests` function is called with updated `guests` values for rooms | Rooms count is updated |
| GI-006 | GuestInput Component closes dropdown on Done button click | | | | | Dropdown is open | | Click on 'Done' button | Dropdown closes, and no 'Adults', 'Children', or 'Rooms' text is visible | Dropdown is closed |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Client/src/searchBar/destinationSearch.js  Testing destination search functionality | | | | | |
| Test Case ID | Description | Preconditions | Input | Expected Output | Post Condition |
| DS-001 | DestinationSearch Component renders input field with placeholder | None | Render the `DestinationSearch` component with empty query | Input field with placeholder 'Where to?' is displayed | Input field is correctly rendered |
| DS-002 | DestinationSearch Component displays suggestions when input value matches destinations | None | Render the `DestinationSearch` component with query 'Ba', input value 'Ba' | Suggestions that match 'Ba' are displayed, including 'Bali, Indonesia' | Matching suggestions are visible |
| DS-003 | DestinationSearch Component updates query when a suggestion is clicked | Input value matches a destination | Render the `DestinationSearch` component with query 'Ba', click on 'Bali, Indonesia' suggestion | `setQueryMock` is called with 'Bali, Indonesia', `setDestinationIdMock` is called with 'WP3Z' | Query and Destination ID are updated |
| DS-004 | DestinationSearch Component shows no suggestions when input is empty | Input field is focused | Render the `DestinationSearch` component with empty query, focus on input field | No suggestions are displayed | No suggestions are visible |
| DS-005 | DestinationSearch Component shows suggestions when input is focused and query is not empty | Input field is focused and query is not empty | Render the `DestinationSearch` component with query 'Ro', focus on input field | Suggestions matching 'Ro' are displayed, including 'Rome, Italy' | Matching suggestions are visible |
| DS-006 | DestinationSearch Component filters suggestions based on input value | Input value matches a destination | Render the `DestinationSearch` component with query 'Ro', input value 'Ro' | Only suggestions matching 'Ro' are displayed, 'Rome, Italy' is present and 'Bali, Indonesia' is not | Filtered suggestions are visible |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Client/src/searchBar/customerDetail.js  Testing input handling for customerDetail form | | | | | |
| Test Case ID | Description | Preconditions | Input | Expected Output | Post Condition |
| CD-001 | CustomerDetail Component handles First Name input changes correctly | None | Simulate user input for the First Name field with value 'John' | handleChange is called with 'firstName', 'John' | First Name field updates correctly |
| CD-002 | CustomerDetail Component handles Last Name input changes correctly | None | Simulate user input for the Last Name field with value 'Doe' | handleChange is called with 'lastName', 'Doe' | Last Name field updates correctly |
| CD-003 | CustomerDetail Component handles Country input changes correctly | None | Simulate user input for the Country field with value 'USA' | handleChange is called with 'country', 'USA' | Country field updates correctly |
| CD-004 | CustomerDetail Component handles Telephone input changes correctly | None | Simulate user input for the Telephone field with value '1234567890' | handleChange is called with 'telephone', '1234567890' | Telephone field updates correctly |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| client/src/UserAuth/loginPage/loginPage.js | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| TC\_L  \_001 | Renders login form correctly | None | Render Login component | The login form with Email, Password fields, Next button, Forgot Password, and Create Account links should be visible. | Login form rendered |
| TC\_L  \_002 | Validates email and password fields | None | Enter invalid email (invalid-email) and short password (short) | Error messages: 'Please enter a valid email address.' and 'Password must be at least 8 characters long and include one lowercase letter, one uppercase letter, one digit, and one special character.' The Next button should be disabled. | Validation errors displayed, form invalid |
| TC\_L  \_003 | Handles successful login | Valid email and password | Enter valid email (test@example.com) and valid password (Valid123!) | Calls authService.login with email and password. Redirects to the homepage ('/') upon successful login. | User logged in, redirected to homepage |
| TC\_L  \_004 | Handles login errors | Valid email and password | Enter valid email (test@example.com) and invalid password (Valid123!) | Calls authService.login with email and password. Displays error message: 'Invalid credentials'. | Error message displayed |
| TC\_L  \_005 | Handles server errors | Valid email and password | Enter valid email (test@example.com) and valid password (Valid123!) | Calls authService.login with email and password. Displays error message: 'An error occurred. Please try again later.' | Error message displayed |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| client/src/UserAuth/forgotPasswordPage/forgotPasswordPage.js | | | | | |
| Test Case  ID | Description | Precondition | Input | Expected Output | Post Condition |
| TC\_FP  \_001 | Test rendering of the ForgotPassword component | None | None | Component renders with form elements and PageHeader | Component rendered |
| TC\_FP  \_002 | Test email validation error message | Email input is invalid | invalid email | Error message 'Please enter a valid email address.' is displayed | Error state updated |
| TC\_FP  \_003 | Test successful password reset request | Email input is valid | valid email | Success message 'Password reset link has been sent to your email.' is displayed | Success state updated |
| TC\_FP  \_004 | Test error message for password reset request | Email input is valid | valid email | Error message from the server is displayed | Error state updated |
| TC\_FP  \_005 | Test server error during password reset request | Email input is valid | valid email | Error message 'Error sending email.' is displayed | Error state updated |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| client/src/pageHeader/pageHeader.js | | | | | |
| Test  Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| TC\_PH  \_001 | Test header renders correctly | BrowserRouter is set up | Render PageHeader component | Header should render with logo, Sign In, Contact Us, and SGD elements | Header elements displayed |
| TC\_PH  \_002 | Test logo renders correctly | BrowserRouter is set up | Render PageHeader component | Logo image should be displayed | Logo displayed |
| TC\_PH  \_003 | Test Sign In link functionality | BrowserRouter is set up | Click Sign In link | Navigation to login page | Redirect to /login |
| TC\_PH  \_004 | Test Contact Us link functionality | BrowserRouter is set up | Click Contact Us link | URL hash should be set to #contactus | URL hash updated |
| TC\_PH  \_005 | Test currency button functionality | BrowserRouter is set up | Click SGD button | Currency dropdown should be displayed | Currency dropdown displayed |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| client/src/hotelSearchPage/filterSection/filterSection.js | | | | | |
| Test  Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| TC\_FS  \_001 | Check if 'Filter By' heading is rendered | FilterSection component is available | Render FilterSection component | 'Filter By' heading should be in the document | None |
| TC\_FS  \_002 | Check if 'Total Price' label is rendered | FilterSection component is available | Render FilterSection component | 'Total Price' label should be in the document | None |
| TC\_FS  \_003 | Check if 'Max' and 'Min' input fields are rendered | FilterSection component is available | Render FilterSection component | 'Max' and 'Min' input fields should be in the document | None |
| TC\_FS  \_004 | Check if 'Star Rating' label is rendered | FilterSection component is available | Render FilterSection component | 'Star Rating' label should be in the document | None |
| TC\_FS  \_005 | Check if star rating buttons (1 to 5) are rendered | FilterSection component is available | Render FilterSection component | Star rating buttons (1 to 5) should be in the document | None |
| TC\_FS  \_006 | Check if 'Guest Rating' label is rendered | FilterSection component is available | Render FilterSection component | 'Guest Rating' label should be in the document | None |
| TC\_FS  \_007 | Check if guest rating radio buttons are rendered | FilterSection component is available | Render FilterSection component | Guest rating radio buttons ('Any', 'Wonderful 9+', 'Very Good 8+', 'Good 7+') should be in the document | None |

## Integration Testing

* Integration testing for backend involves testing various controller functions, which are called directly when a certain route is which. Thus, the controllers are tested by making a request to our various server routes. The controllers handle important business logic and utilize the model functions which have already been unit tested. We initiate the call from the app-level via the routers.
* We perform **call-graph based** integration testing to ensure that the testing reflects the actual execution and function call relation (e.g. webhook may make a request to create-booking route, etc)
* We follow a **bottom-up** approach as we have already extensively unit tested the models (subunits) and can now test the parent components, with the actual implementations of the model functions. Because they have already been unit tested, we know that any faults will not be because of them.
* Mocking and stubbing:
  + As mentioned previously, we now use the actual implementations of the model functions and this includes using an actual database connection (rather than mocking database response as we did for unit test). A mock database connection (dbTest.js) along with a mock table is used to isolate the tests from the actual database tables being used.
  + However, other external dependencies like Ascendas API and Stripe are still mocked
* Robustness: tests include both happy path scenarios and error cases (e.g. testing the view booking endpoint with both existing and non-existing booking IDs).
* State verification: after operations that modify data (e.g., canceling a booking), beyond just checking expected return values, the tests verify the actual database state by making additional API calls.
* Environment setup:
  + Before each test suite, the test environment is set up, including table name configuration and console output mocking.
  + After all tests, there's a cleanup process to tear down the test booking table and close the database connection.
* Test Isolation: Jest’s clearAllMocks() is used before each test to ensure independence between test cases.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| controllers/bookHotelController.test.js | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| BI-001 | GET /booking/view/:id should return booking object with given booking ID | Test database populated with sample bookings | GET request to /booking/view/1 | 200 status code, Booking object with ID 1 | None |
| BI-002 | GET /booking/view/:id should return null for non-existent booking | Test database populated with sample bookings | GET request to /booking/view/900000000 | 200 status code, null response body | None |
| BI-003 | GET /booking/customer/:id should return bookings associated with customer id | Test database populated with sample bookings | GET request to /booking/customer/1 | 200 status code, Array of booking objects for customer 1 | None |
| BI-004 | GET /booking/customer/:id should return null for customer with no bookings | Test database populated with sample bookings | GET request to /booking/customer/900000000 | 200 status code, null response body | None |
| BI-005 | GET /booking/cancel/:id should set booking's status to cancelled | Test database with a 'confirmed' booking | GET request to /booking/cancel/1 | 200 status code, "Success: booking cancelled" message | Booking status changed to 'cancelled' in database |
| BI-006 | GET /booking/cancel/:id should return failure message for non-existent booking | Test database populated with sample bookings | GET request to /booking/cancel/900000000 | 200 status code, "Failure: booking cannot be found" message | None |
| BI-007 | POST /booking/create should create a new booking | Test database populated, Stripe and hotel API mocked | POST request to /booking/create with valid booking data | 200 status code, object with new bookingId | New booking added to database |
| BI-008 | POST /booking/checkout should create a Stripe checkout session | Stripe API mocked | POST request to /booking/checkout with booking information | 200 status code, mock Stripe session object | None |
| BI-009 | POST /booking/webhook should handle 'charge.succeeded' event | Stripe webhook mocked | POST request to /booking/webhook with 'charge.succeeded' event data | 200 status code, session ID logged | None |
| BI-010 | POST /booking/webhook should handle 'checkout.session.completed' event | Stripe webhook mocked | POST request to /booking/webhook with 'checkout.session.completed' event data | 200 status code, createBookingRoute called with correct params | createBookingRoute called to follow ‘create booking’ procedure |
| BI-011 | POST /booking/webhook should handle 'payment\_intent.created' event | Stripe webhook mocked | POST request to /booking/webhook with 'payment\_intent.created' event data | 200 status code, session logged with added receipt email | None |
| BI-012 | POST /booking/webhook should handle ‘refund.created’ event | Stripe webhook mocked | POST request to /booking/webhook with ‘refund.created' event data | 200 status code, session logged | None |
| BI-013 | POST /booking/webhook should handle ‘refund.updated’ event | Stripe webhook mocked | POST request to /booking/webhook with ‘refund.updated’ event data | 200 status code, session logged | None |
| BI-014 | POST /booking/refund/:bookingId/:paymentId should process refund successfully | Test database with a 'confirmed' booking, Stripe API mocked | POST request to /booking/refund/1/456 | 200 status code, {success: true} | Booking status changed to 'cancelled' in database |
| BI-015 | getBillingEmail function should return customer billing email | Stripe API mocked | paymentId '3' | "[customer@email.com](mailto:customer@email.com)" | None |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| controllers/searchHotelController.test.js | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| HS-001 | POST /search/destination/  should return a list of hotels for a given destination | Mocked Ascenda API returns hotel data | POST request to /search/destination/w0Xm with valid checkin, checkout, lang, currency, and guests | 200 status code, Array with one hotel, containing hotel and price data | None |
| HS-002 | POST /search/destination/  should handle empty hotel list data from API | Mocked Ascenda API returns empty hotel list | POST request to /search/destination/w0Xm with valid checkin, checkout, lang, currency, and guests | 200 status code, empty array | None |
| HS-003 | POST /search/hotel/  should return hotel details with room information | Mocked Ascenda API returns hotel and room data | POST request to /search/hotel/123 with valid destination\_id, checkin, checkout, lang, currency, and guests | 200 status code, object with hotel and room information | None |
| HS-004 | POST /search/hotel/  should handle hotel not found | Mocked Ascenda API returns empty hotel data | POST request to /search/hotel/nonexistent with valid destination\_id, checkin, checkout, lang, currency, and guests | 200 status code, object with empty hotel and room information | None |
| HS-005 | Endpoints should handle Ascenda API 422 errors (should not crash) | Mocked Ascenda API returns 422 error | POST request to /search/destination/w0Xm or /search/hotel/123 with valid input | 200 status code | None |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Server/source/controllers/authController.js  Integration Testing of Register, Login, Forgot password and Reset Password | | | | | |
| Test Case ID | Description | Precondition | Input | Expected Output | Post Condition |
| TC\_AC 1 | (Login) should return 404 if user not found | `findUserByEmail` returns `null` | { email: 'nonexistent@example.com', password: 'Password@123' } | Status 404, `{ success: false, message: 'User not found' }` | No user found in database |
| TC\_AC 2 | (Login) should return 400 if the password is incorrect | `findUserByEmail` returns a valid user | { email: '50.003.c4g1@gmail.com', password: 'wrongpassword' } | Status 400, `{ success: false, message: 'Invalid username or password' }` | Password does not match |
| TC\_AC 3 | (Login) should return 200 if login is successful | `findUserByEmail` returns a valid user, password matches | { email: 'test@example.com', password: 'Password@12345' } | Status 200, `{ success: true, token: 'fake-jwt-token', userId: 4, username: 'testuser' }` | User is authenticated, JWT token issued |
| TC\_AC 4 | (Login) should return 500 if server error occurs | `findUserByEmail` throws an error | { email: 'test@example.com', password: 'Password@12345' } | Status 500, `{ success: false, message: 'Server error' }` | Server error during login |
| TC\_AC 5 | (Register) should return 400 if user already exists | `findUserByEmail` returns a valid user | { email: '50.003.c4g1@gmail.com', password: 'Password@12345', username: 'testuser', hp: '12345678' } | Status 400, `{ success: false, message: 'User already exists' }` | User already exists in database |
| TC\_AC 6 | (Register) should return 201 if registration is successful | `findUserByEmail` returns `null`, `insertCustomer` succeeds | { email: '50.003.c4g1@gmail.com', password: 'Password@12345', username: 'testuser', hp: '12345678' } | Status 201, `{ success: true, token: 'fake-jwt-token', userId: 1, username: 'testuser' }` | User registered successfully, JWT token issued |
| TC\_AC 7 | (Register) should return 500 if server error occurs | `findUserByEmail` throws an error | { email: '50.003.c4g1@gmail.com', password: 'Password@12345', username: 'testuser', hp: '12345678' } | Status 500, `{ success: false, message: 'Server error' }` | Server error during registration |
| TC\_AC 8 | (Forgot Password) should return 404 if user not found | `findUserByEmail` returns `null` | { email: '50.003.c4g1@gmail.com' } | Status 404, `{ success: false, message: 'Email not found' }` | No user found in database |
| TC\_AC 9 | (Forgot Password) should return 200 if reset password email is sent successfully | `findUserByEmail` returns a valid user, `updateUser` succeeds | { email: '50.003.c4g1@gmail.com' } | Status 200, `{ success: true, message: 'Email sent' }` | Password reset email sent successfully |
| TC\_AC 10 | (Forgot Password) should return 500 if server error occurs | `findUserByEmail` throws an error | { email: '50.003.c4g1@gmail.com' } | Status 500, `{ success: false, message: 'Server error' }` | Server error during password reset email |
| TC\_AC  11 | (Reset Password) should return 404 if user not found | `findUserByToken` returns `null` | { token: 'fake-token', password: 'Password@12345' } | Status 400, `{ success: false, message: 'Token is invalid or has expired' }` | Invalid or expired token |
| TC\_AC  12 | (Reset Password) should return 200 if password is reset successfully | `findUserByToken` returns a valid user, `updateUser` succeeds | { token: 'fake-token', password: 'Password@12345' } | Status 200, `{ success: true, message: 'Password has been reset successfully' }` | Password reset successfully |
| TC\_AC  13 | (Reset Password) should return 500 if server error occurs | `findUserByToken` throws an error | { token: 'fake-token', password: 'Password@12345' } | Status 500, `{ success: false, message: 'Server error' }` | Server error during password reset |

## System Testing

We used Cypress for our system testing as it is beginner friendly requiring minimal setup and easy to create tests for our web application.

We have performed system testing for the following use cases:

* HBS\_UC\_1 Search Hotel
* HBS\_UC\_2 Book Hotel
* HBS\_UC\_4 Cancel Booking
* CAS\_UC\_1 Login
* CAS\_UC\_2 Register

## Robustness Testing (Fuzzing)

This fuzz testing suite is designed to continuously test the robustness of the CustomerDetail component in a React application by injecting random input strings into various input fields. The primary goal is to evaluate how the component handles unexpected or extreme inputs, which is essential for ensuring the application's stability and security.

**Testing Methodology**

**Random String Generation**: The test suite generates random strings using characters within the ASCII range 32 to 96. This includes a wide variety of characters such as spaces, punctuation, and other special characters. The lengths of these strings are also randomized to simulate various edge cases, such as very short or extremely long inputs.

* **Test Case Generation**: The fuzz test suite generates multiple test cases (in this instance, 100) by assigning each one a random string of varying length. These test cases are created to simulate user input that could potentially break or expose vulnerabilities in the component.
* **Input Fields Tested**: The CustomerDetail component is tested by injecting the generated random strings into the following input fields:
  + firstName
  + lastName
  + country
  + telephone
* **Continuous Testing Loop**: For each generated test case, the suite performs the following actions:
  + Renders the CustomerDetail component.
  + Simulates a user typing the random string into each of the input fields mentioned above.
  + Logs the test details, including the specific input string used, to a log file for later analysis.
* **Logging**: All actions performed during the tests, including the random strings injected and the fields tested, are logged to a file (fuzz\_test\_log.txt). This allows for detailed post-test analysis to identify any issues that arise during testing

# Contributions to United Nation SDG

Our hotel booking web application aims to contribute positively towards sustainability, diversity, and inclusion. The project aligns with several United Nations Sustainable Development Goals (SDGs), particularly:

**SDG 8: Decent Work and Economic Growth**

By providing a seamless hotel booking platform, we support the hospitality industry, which is a significant source of employment and economic growth worldwide. Our platform allows small and independent hotels to compete with larger chains by providing them with equal visibility to potential customers.

**SDG 9: Industry, Innovation, and Infrastructure**

Our implementation of efficient and scalable technology, including caching systems and front-end pagination, promotes sustainable industrialization. By optimizing resource use and improving performance, our project supports innovation and resilient infrastructure in the hospitality sector.

**SDG 10: Reduced Inequalities**

Our platform is designed to be inclusive and accessible to a diverse user base, regardless of their demographic background. We ensure that the user interface is intuitive and accessible to people with disabilities. By providing various language options and accommodating different payment methods, we strive to make travel accessible to all, thus reducing inequalities.

# Diversity and Inclusivity

**Payment Options**

The application supports various payment methods, including local payment systems, to accommodate users from different regions who may not have access to international payment options.

**Age Inclusivity**

The website is designed to be user-friendly for all age groups. We have incorporated features like larger fonts, clear navigation, and simplified booking processes to ensure that both younger and older users can comfortably use the platform.