NUMBER ONE
BARBIER

BTM 495 - TEAM 2

PROJECT FINAL DOCUMENTATION

NUMBER ONE BARBERSHOP

CONCORDIA UNIVERSITY
JOHN MOLSON SCHOOL OF BUSINESS

Executive Summary

Number One Barbershop Booking and Scheduling System is an innovative solution designed to manage the barbershop's operations and improve customer experiences. By combining real-time booking, advanced scheduling, and customer relationship management (CRM), the system simplifies appointment management for customers as well as the barbers and the store's management.

For customers, the system offers an interface to browse available services, select the barbers that they want, and book appointments easily. Automated email and SMS confirmations, combined with timely reminders, ensure convenience and reduce missed appointments. Additionally, customers can manage their own bookings with the flexibility to reschedule or cancel as needed.

Barbers and store managers benefit from numerous tools designed to streamline daily operations. With role-specific dashboards, barbers can monitor their schedules, adjust availability, and reconcile appointments easily. Managers, on the other hand, gain important insights into business performance, including analytics on booking patterns, staff utilization, and customer retention rates. These features enable better decision-making and resource allocation for the store.

The platform's architecture is designed with scalability at its core, ensuring it can adapt to increasing demands as the barbershop grows its customer base and service offerings. Using Python (Django), React, PostgreSQL, and deployed on AWS Lightsail, the system ensures reliability and adaptability. With features like role-based dashboards, automated notifications, and data-driven insights, this platform enhances operational efficiency, reduces scheduling errors, and creates stronger customer relationships, making it an essential tool for modern barbershops aiming to thrive in a competitive market.

This system minimizes scheduling errors, optimizes resource utilization, and fosters customer loyalty. By automating appointment management and reducing administrative workload, the barbershop can enhance profitability while focusing on delivering high-quality services. To conclude, this is not just a tool for managing appointments; it is a strategic asset that drives operational excellence and elevates the overall customer experience.

Table of Content

Team Project Proposal

1. Project Sponsor:

Number One Barbershop
1228 Mackay St, Montreal, Quebec H3G 2H4

2. Project Scope:

Objective:
Design, develop and implement a booking and scheduling system with a CRM functionality that is intuitive and easy to use.

Features:

- Real-time Booking and Availability Management.
- Automated Client Relationship Management (CRM) Integration and Control Over the Database.
- Appointment Management and Payment Tracking System.
- User-friendly Dashboard.

3. Functional Requirements:

Real-time Booking and Availability Management

- The system must allow users to view available time slots in real-time and make bookings.
- The system must automatically update and sync booking availability to prevent double-booking.
- Users must be able to cancel, modify, or reschedule bookings.
- The system must send automated booking confirmations and reminders to users via email or SMS.
- Users must be able to select appointments by service type, by a service provider and then by “earliest time available”.

Automated Client Relationship Management (CRM) Integration and Control Over the Database

- The system must store and manage client information, including contact details, preferences, and appointment history.
- The system must allow employees to add, update, and delete client records securely.
- The system must allow users to update and change their contact information securely.
- The CRM system must have search and filter capabilities to access customer information quickly.
- Provide data import/export functionalities to integrate with external CRM or marketing platforms.

Appointment Management and Payment Tracking System

- The system must allow employees to track the status of appointments (e.g., scheduled, in progress, completed, canceled).

- The system must allow employees to update appointment statuses as needed.
- The system must provide functionality to input payment details such as payment method (credit card, cash, etc.) and payment amount (service amount and tip amount).
- Employees must be able to view customer appointment history, including payments made.

User-friendly Dashboard with Analytics

- The system must provide a dashboard that displays key metrics such as the number of bookings, cancellations, and appointment status.
- The system must allow users to view and generate reports based on data (e.g., bookings, payments, customer retention).
- The system must provide role-based access to ensure only authorized users can view specific data.

General System Requirements:

- The system must be dynamic and accessible through both web and mobile devices.
- The system must support multiple user roles with varying levels of access (e.g., admin, employee, customer).
- The booking system must have a confirmation layer using SMS.
- The system must provide data backup and recovery mechanisms.
- The system must be scalable to accommodate an increasing number of users and appointments.

4. Business Value

The business value in the new to-be system is significant and impacting the operational efficiency and customer experience.

- **Error Reduction**: By syncing booking availability and payment tracking, the system prevents double-bookings, scheduling errors, and payment processing mistakes.
- **Increased Booking Capacity**: With real-time booking and multi-service provider scheduling, businesses can optimize appointment slots, reducing downtime and maximizing booking capacity.
- **Enhanced Customer Experience**: real time access and flexibility improving user satisfaction.
- **Automated Processes**: Reduce the need for manual appointment verification and the tally of payment, tips and commission breakdown calculation, savings time for the employee and manager.

5. Special Issues or Constraints:

The to-be system will handle booking of clients and scheduling. It will also track the payment made by the customer. The project excludes the development of a POS System. Number One Barber Shop will have to procure a 3rd party vendor in order to accept card and contactless payments. Given the limited IT budget, the number of features will be limited to what has been outlined above.

Analysis Modeling

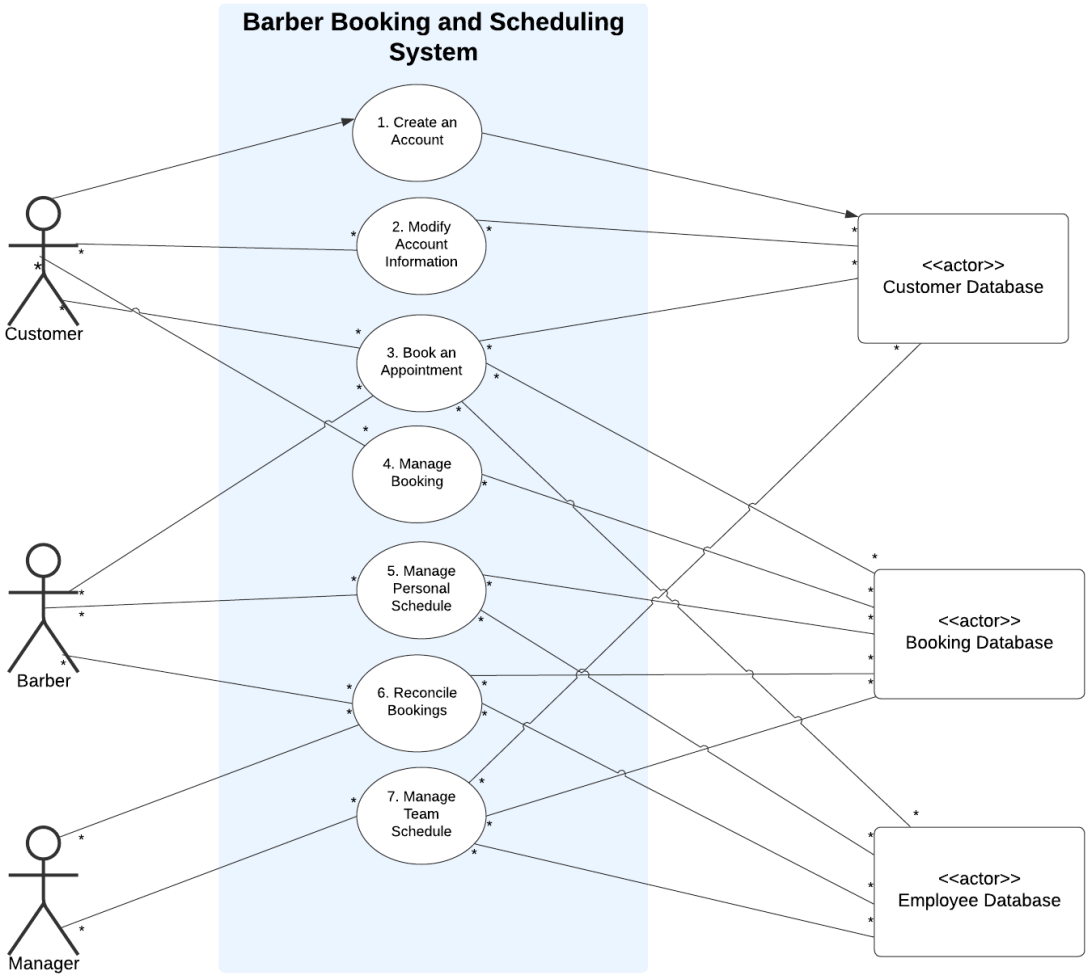
Functional Modeling

Use case diagrams and activity diagrams play a crucial role in illustrating the functionality of our information system. The use case diagrams provide an overview of the system within its environment, highlighting how users interact with it without delving into internal mechanisms. Additionally, the use case descriptions help prioritize the most critical use cases for development.

In contrast, activity diagrams offer a detailed depiction of the system's workflows, showing how processes unfold and how data moves throughout. This approach effectively communicates to users what the system will do without overwhelming them with technical details, while also helping to define the foundational components of the system's design.

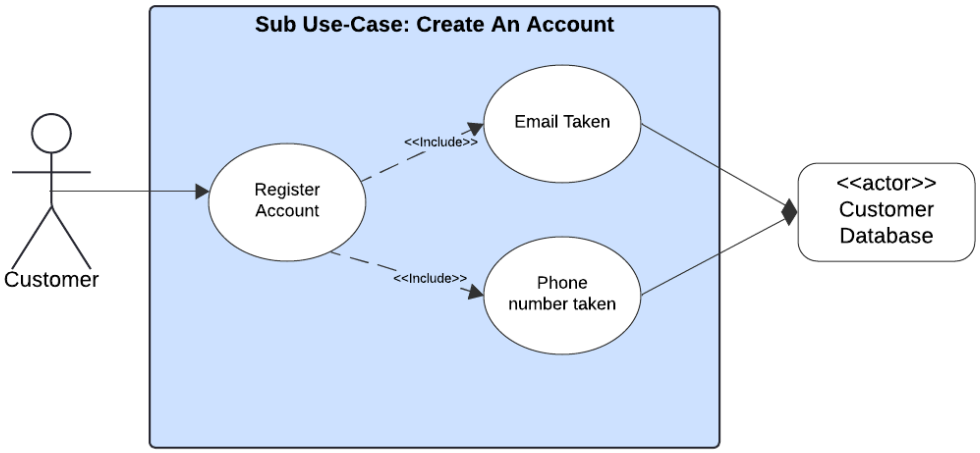
Below, you can find our system's use case diagram and seven functional models for each of our main functions in the system, including a use-case diagram, use-case description, and activity diagram.

System Use Case Diagram: Barber Booking And Scheduling System

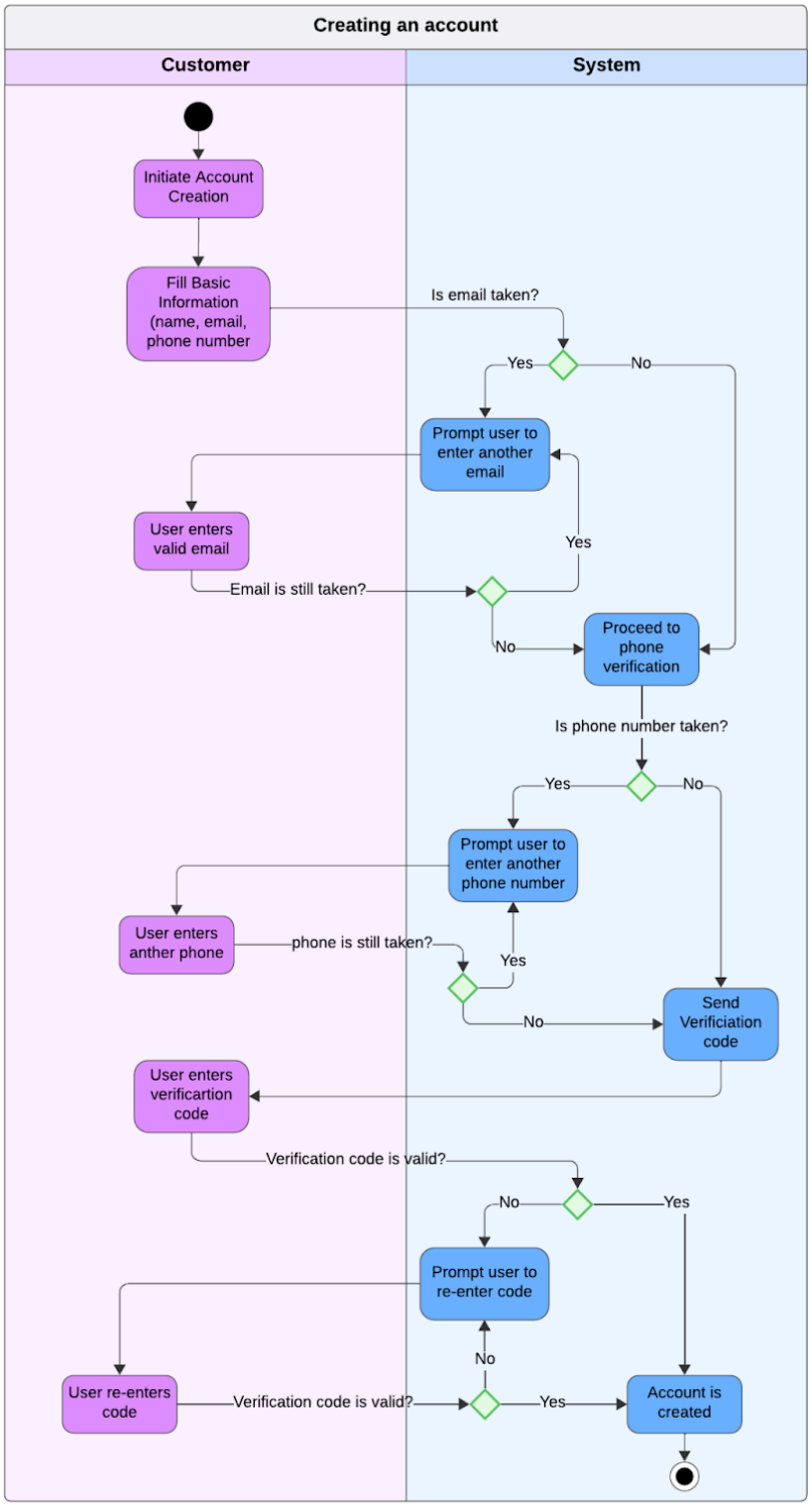


Account Creation

Use-Case Diagram 1



Activity Diagram Use-Case 1

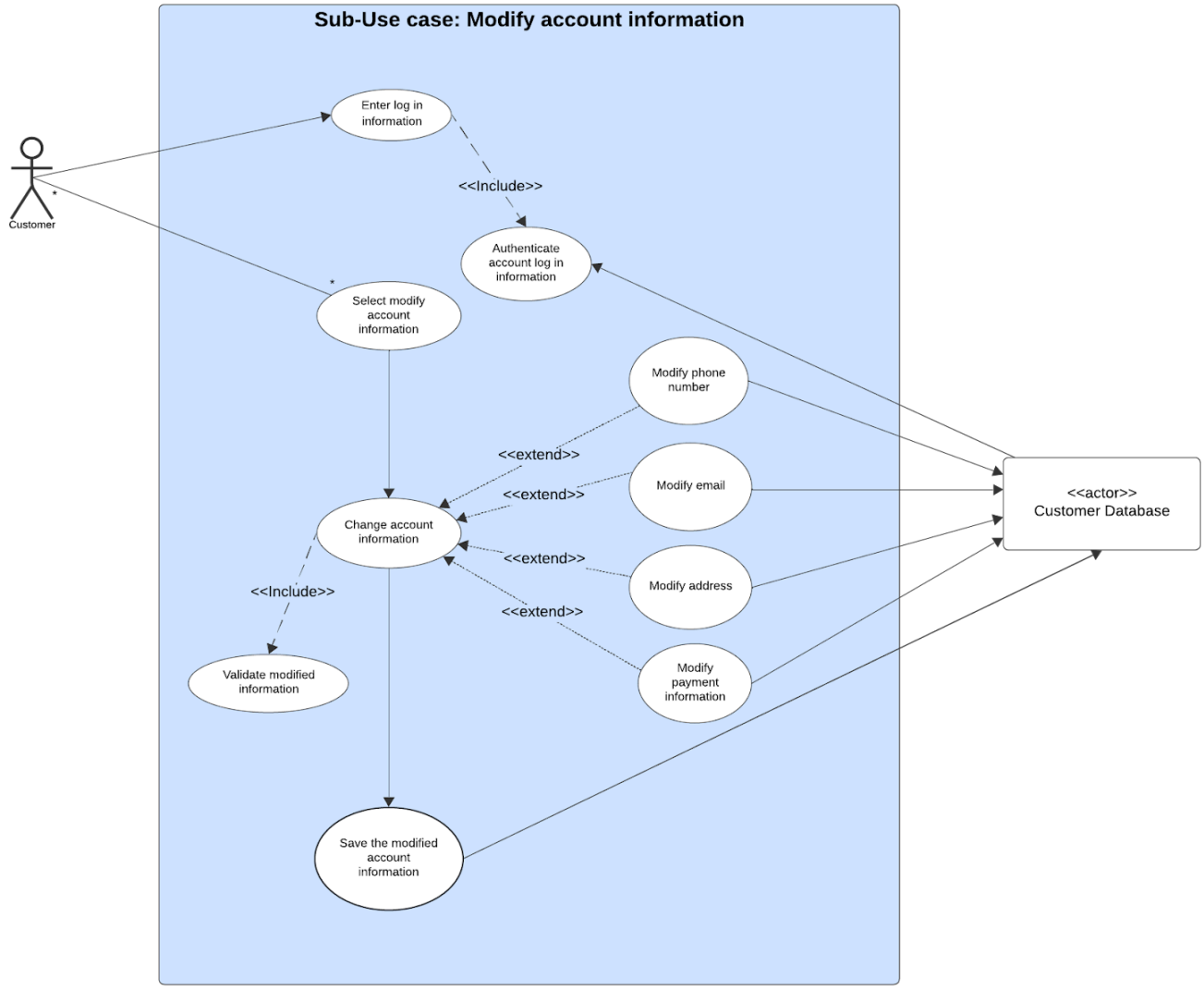


Use-Case Description Use-Case 1

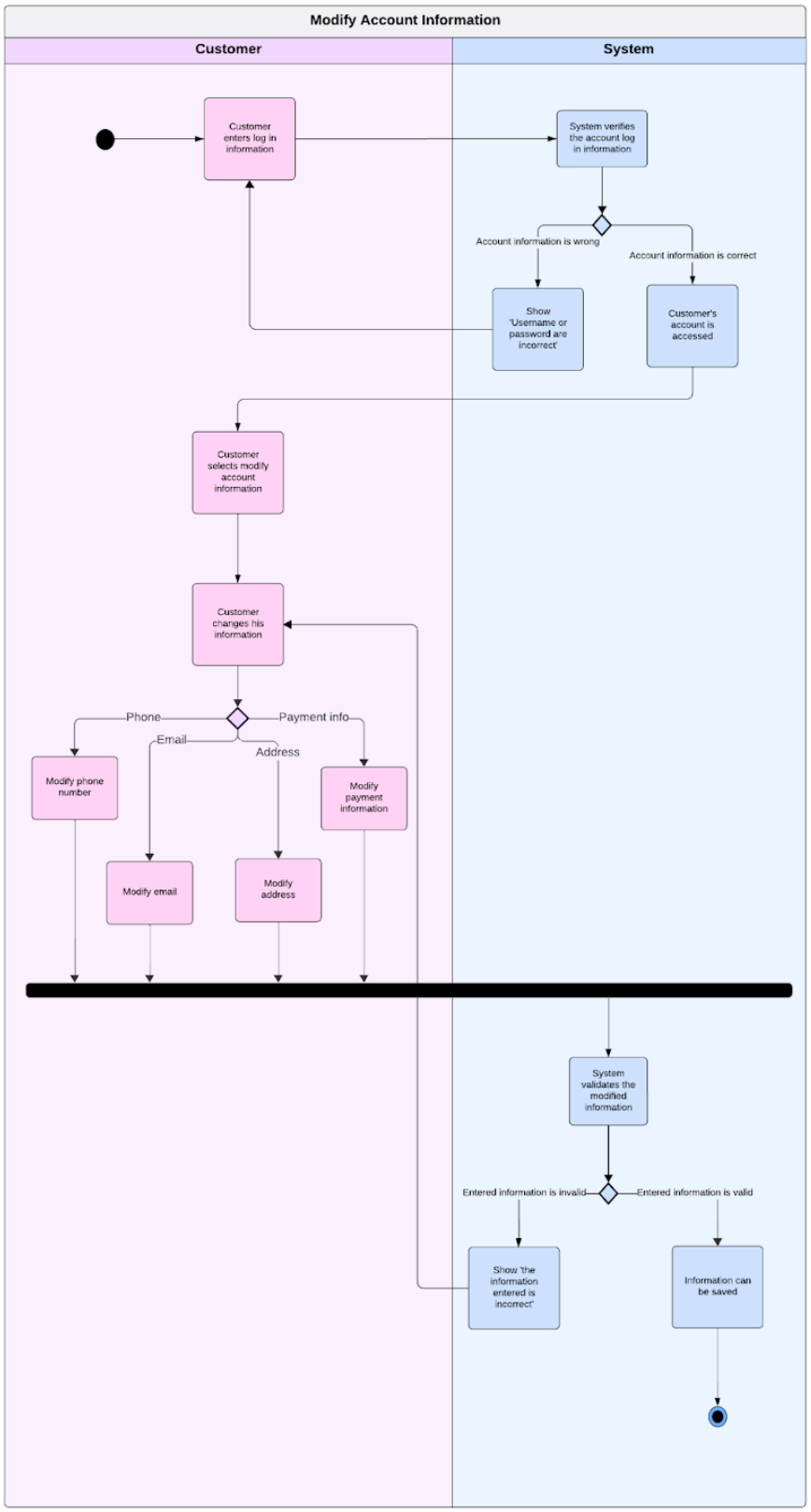
Use Case Name: Create an account	ID: 1	Importance Level: High
Primary Actor: Customer	Use Case Type: Detailed, real	
Stakeholders and Interests: Customer - wants to create an account		
Brief Description: This use-case identifies the main steps for the customers to create an account		
Trigger: Customer wants to create an account Type: Customer - External		
Relationships: Association: Customer Include: Phone verification code, email Extend: Generalization:		
Normal Flow of Events: 1. Customer initiate account creation 2. They fill in basic information (name, email, telephone, etc) 3. System verify if email is unique a. If email is taken Subflow S3.A b. If email is not taken proceed to step 4 4. System verify phone number a. If phone number is taken Subflow S4.A b. If phone number is not taken proceed to step 5 5. System send a verification code to phone 6. User is prompted to enter the verification code a. If the verification code is incorrect Subflow S6.A b. If verification code is correct proceed to next step 7. Account is created		
SubFlows: S3.A 1. Prompt user to enter another valid email 2. User enter a different email 3. If email is still taken Subflow S3.A restarts S4.A 1. Prompt user to enter another valid phone number 2. User enter a different phone number 3. If phone number is still taken Subflow S4.A restarts S6.A 1 Prompt user to re-enter the verification code 2 User re-enter the verification code 3 If code still wrong subflow S6.A restart		
Alternate/Exceptional Flows:		

Account Modification

Use-Case Diagram 2



Activity Diagram Use-Case 2

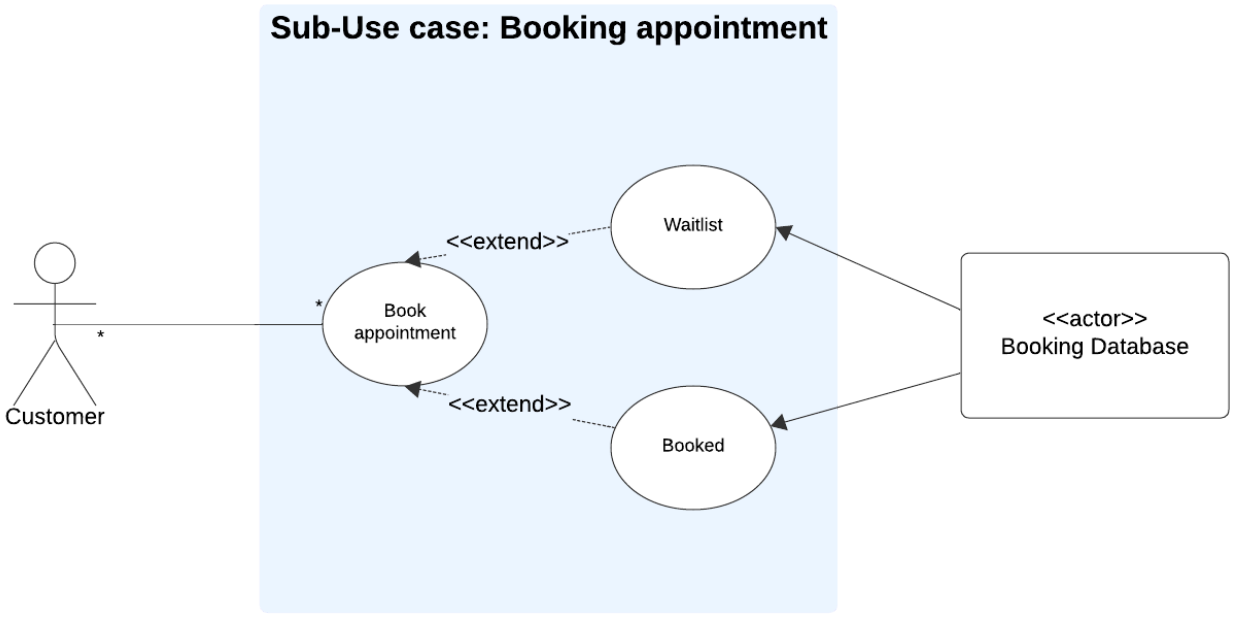


Use-Case Description Use-Case 2

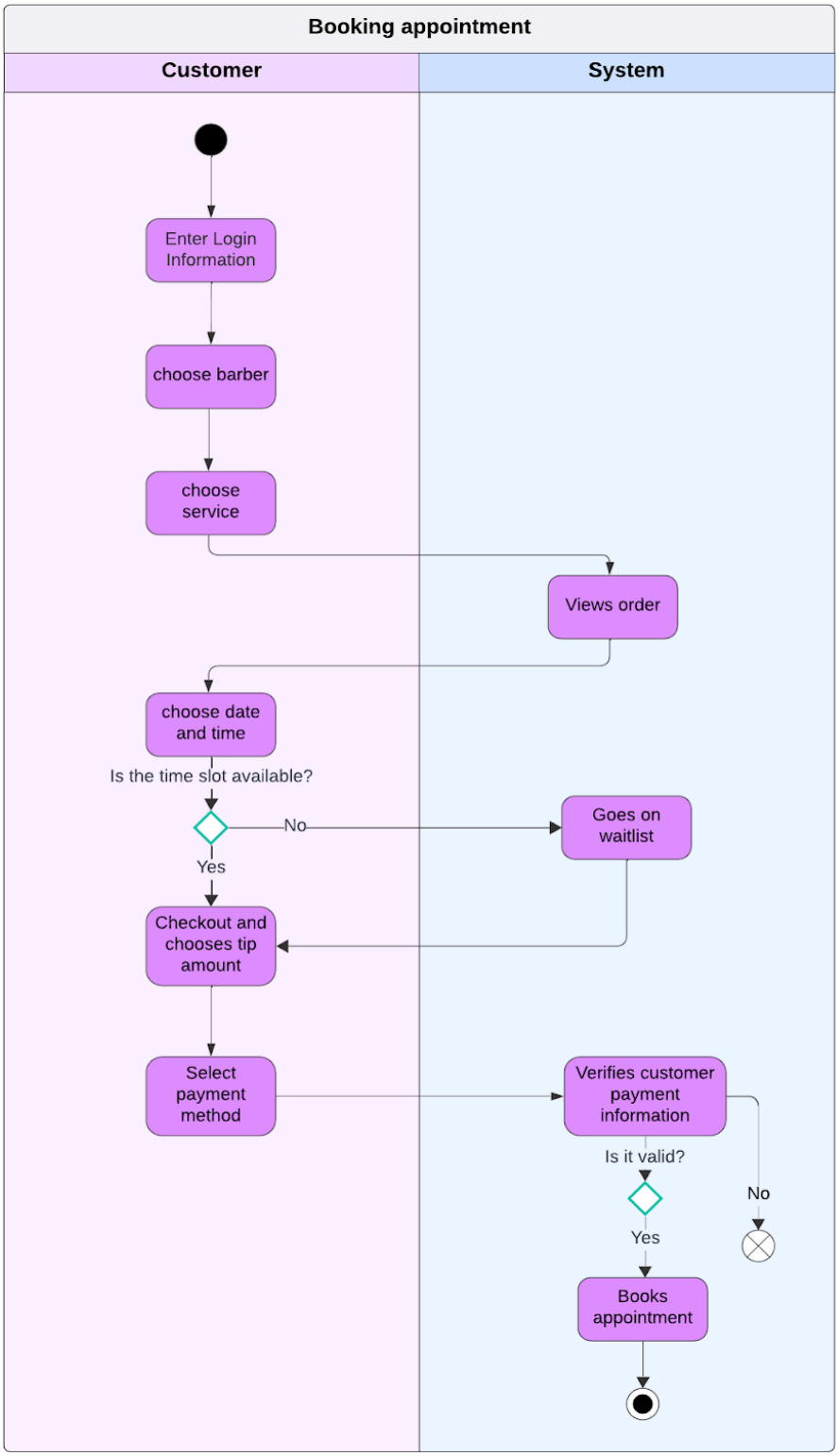
Use Case Name: Modify account information	ID: 2	Importance Level: High
Primary Actor: Customer	Use Case Type: Detail, real	
Stakeholders and Interests: Customer who wants to modify his account information. Barber who wants to update client's information, when receiving a phone call or a client informing personal information change.		
Brief Description: This use case allows customers and barbers to modify the customers' account information		
Trigger: customer or barber want to modify account information Type: ● Internal barber ● External customer		
Relationships: Association: Customer Include: Verify log in information and system validates the modified information Extend: Generalization:		
Normal Flow of Events: 2.1 Customer enters log in information 2.2 The system authenticates the account log in information 2.3 Customer selects modify account information 2.4 Customer changes his information 2.5 System validates the modified information 2.6 Customer saves the new information		
SubFlows (extend): 2.4.1 Modify phone number 2.4.2 Modify email 2.4.3 Modify address 2.4.4 Modify payment information		
Alternate/Exceptional Flows (include): 2.2.1. if the account information is wrong, the system shows "Username or password are incorrect" 2.2.2 if the account information is correct, the customer's account is accessed 2.5.1 if the entered information is wrong, the system shows "the information entered is incorrect" 2.5.2 if the entered information is correct, the information can be saved		

Appointment Booking

Use-Case Diagram 3



Activity Diagram For Use-Case 3

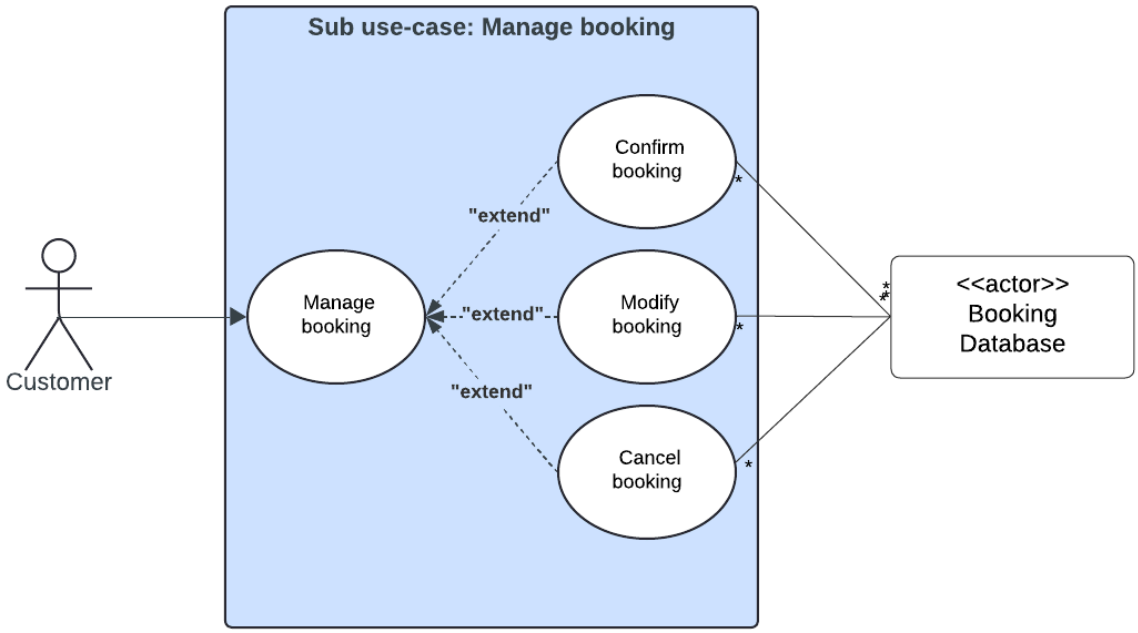


Use-Case Description For Use-Case 3

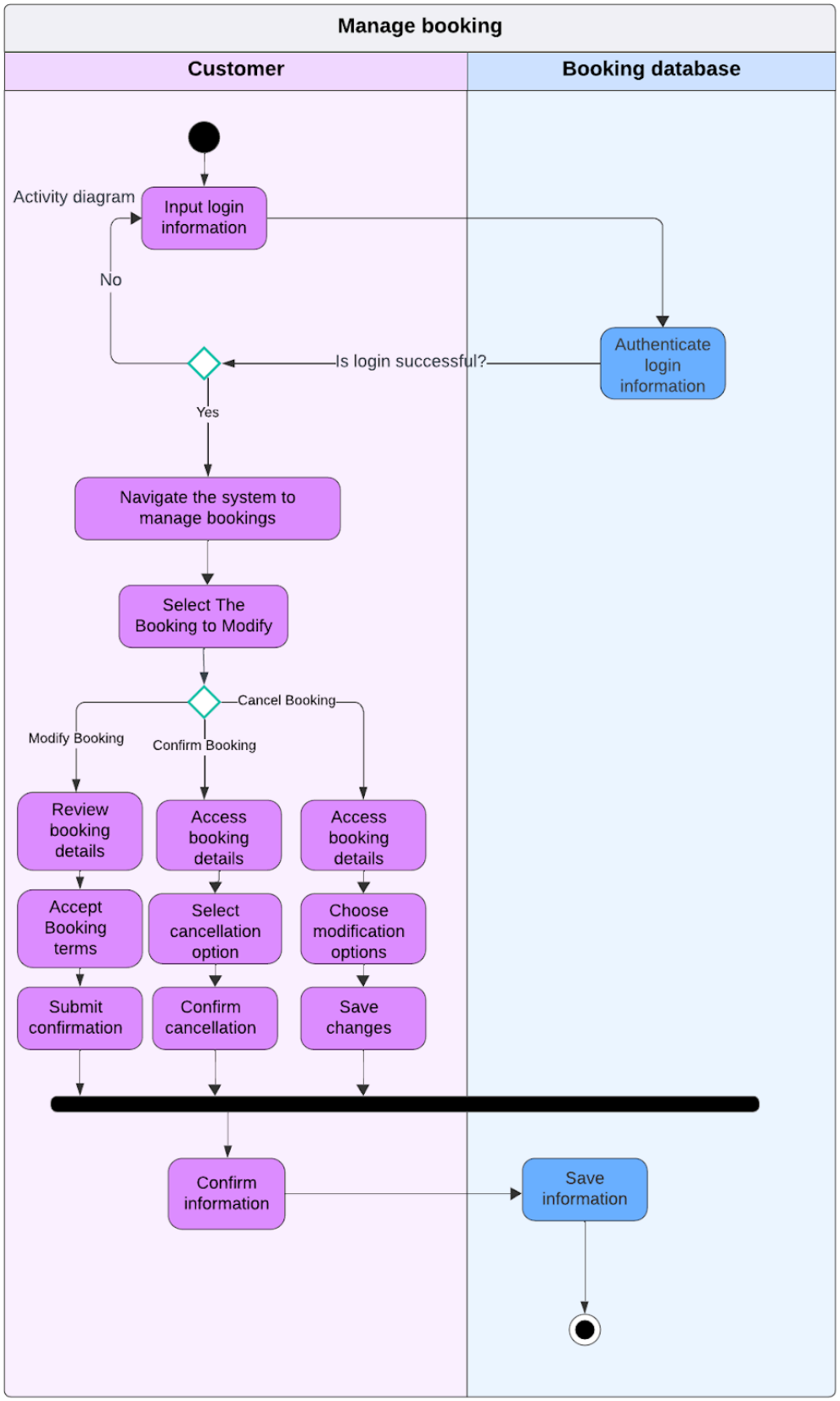
Use Case Name: Book an appointment	ID: 3	Importance Level: High
Primary Actor: Customer	Use Case Type: Detailed, real	
Stakeholders and Interests: Customer, wants to book an appointment		
Brief Description: Customer goes on the website to book an appointment and chooses the date and time for it.		
Trigger: Customer wants to book an appointment Type: External		
Relationships: Association: Customer Include: verify payment information Extend: check for time slot availability Generalization:		
Normal Flow of Events: 1. Customer enters login information 2. Customer chooses barber 3. Customer chooses the service 4. Customer views order 5. Customer chooses a day and time 6. Customer goes to checkout and chooses tip amount 7. Customer chooses payment method and fills in information 8. Customer books appointment		
SubFlows: 5S.1 Day and time that customer wants is already booked 5S.2 Customer goes on waitlist		
Alternate/Exceptional Flows: 7.1 Customer's payment information is valid 7.2 Customer's payment information is declined		

Manage Booking

Use-Case Diagram 4



Activity Diagram Use-Case 4

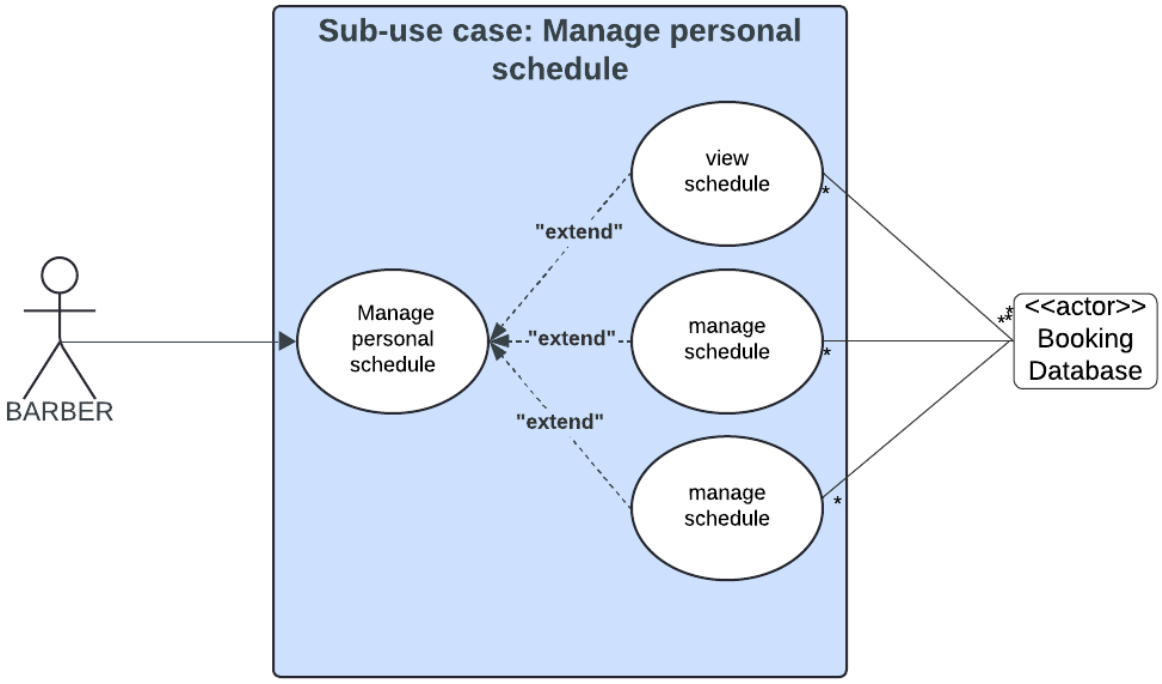


Use-Case Description For Use-Case 4

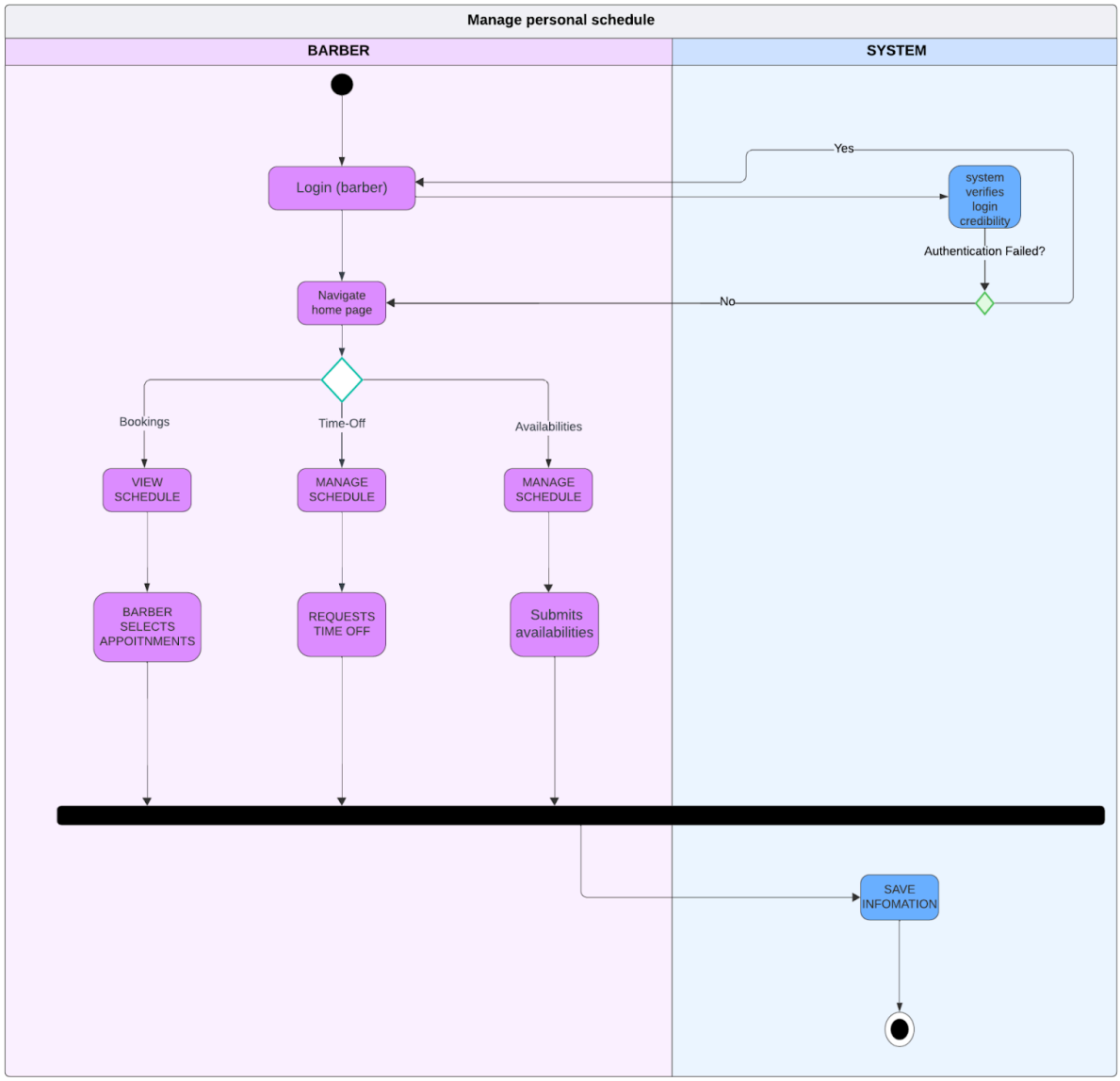
Use Case Name: Manage Booking.	ID: 4	Importance Level: High
Primary Actor: Customer.	Use Case Type: Detailed, Real.	
Stakeholders and Interests: Customer wants to manage his booking effectively.		
Brief Description: This use-case identifies the main steps customers have to take to adjust their existing booking.		
Trigger: Customer decides to manage his booking information. Type: External		
Relationships: Association: Customer Include: - Extend: Confirm, cancel, modify. Generalization: -		
Normal Flow of Events: 1. Customer navigates the appointment system. 2. Customer inputs login information. 3. System authenticates login information. 4. Customer navigates the system to the management section. 5. Select booking to modify. 6. Customer selects an action (modify, confirm, cancel) for their booking. If the customer selects confirm booking, sub-flow 3S.1 is completed. If the customer selects cancel booking, sub-flow 3S.2 is completed. If the customer selects modify booking, sub-flow 3S.3 is completed. 7. Customer confirms their choice.		
Sub Flows: 3S.1 Customer confirms his booking. - Review booking details. - Accept terms and conditions. - Submit confirmation. 3S.2 Customer cancels his booking. - Access booking details. - Select cancellation option. - Confirm cancellation. 3S.3 Customer modifies booking. - Access booking details. - Choose modification options. - Save changes.		
Alternate/Exceptional Flows:		

Barber Managing Personal Schedule:

Use-Case Diagram 5



Activity Diagram For Use-Case 5



Use-Case Description For Use-Case 5

Use Case Name: Manage personal schedule	ID: 5	Importance Level: High
Primary Actor: Barber	Use Case Type: Detailed, real	

Stakeholders and Interests:
Barber wants to manage bookings
Barbershop needs to verify barber availabilities

Brief Description:

This use case identifies the main steps the barbers need to manage their personal schedule

Trigger: Barber wants to manage/change their schedule or view

Type: Internal

Relationships:
Association: Barber
Include: -
Extend: Change schedule, Request time off, Verify personal bookings
Generalization: -

Normal Flow of Events:
1. Barber logs in to scheduling system
2. System verifies login information
3. System opens Home page
4. Barber selects view schedule
5. Barber selects home
6. Barber selects manage schedule
7. Barber submits availabilities
8. Barber selects manage schedule
9. Barber requests time off

SubFlows:

2S1. Barber selects view schedule
2S2. Barber selects appointments

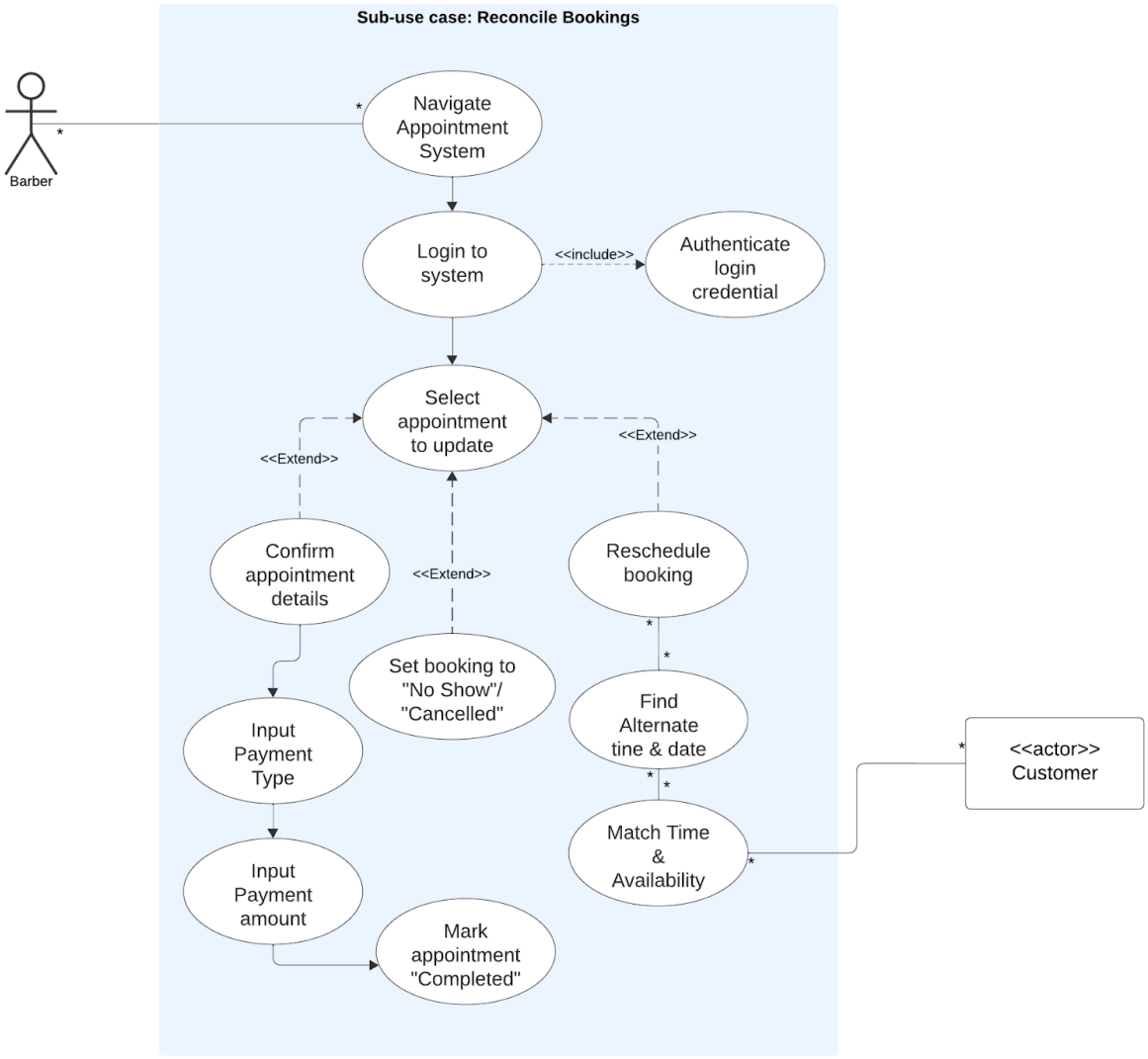
4S1. Barber selects manage schedule
4S2. Barber submits availabilities

6S1. Barber selects manage schedule
6S2.Barber selects requested time off

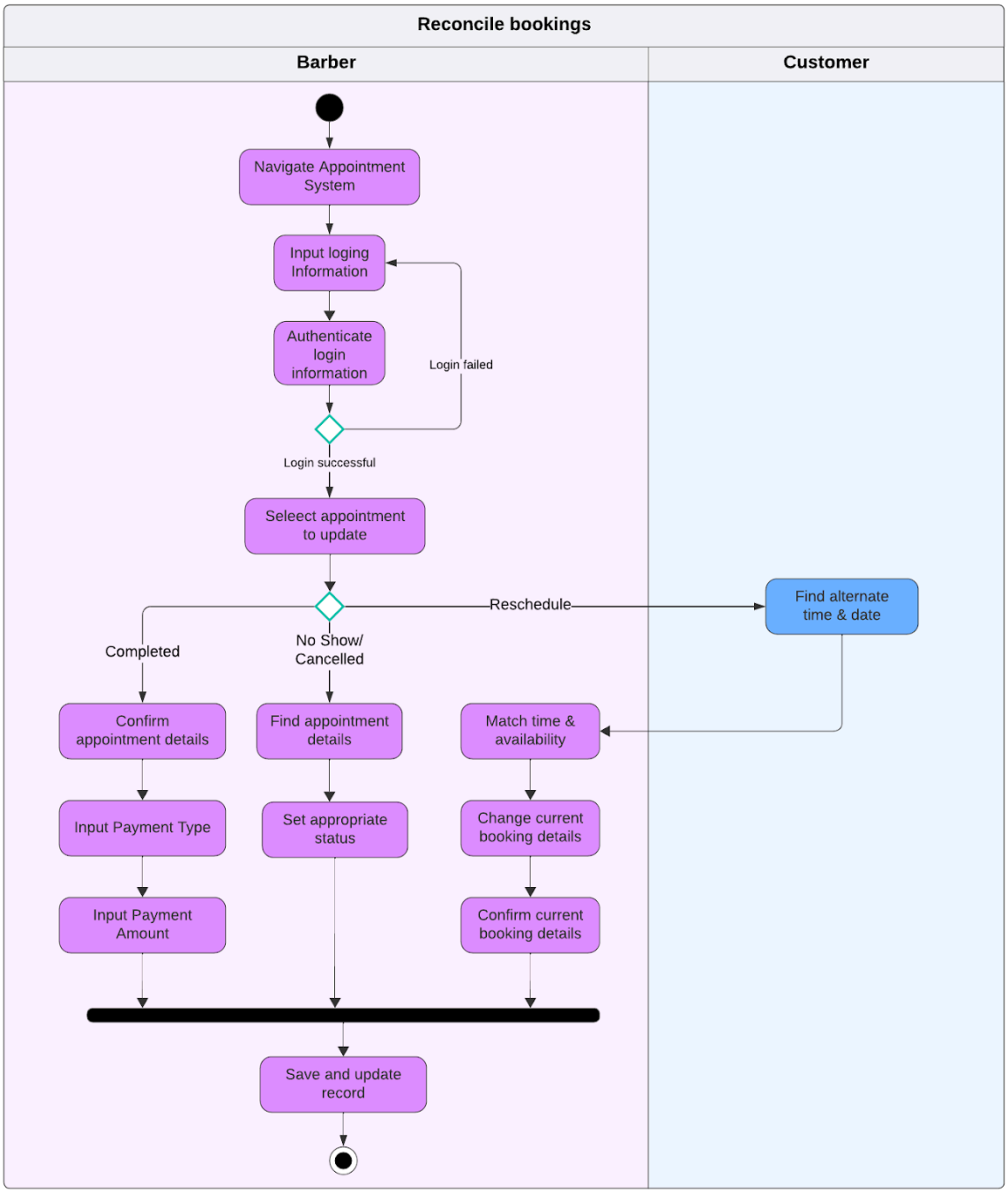
Alternate/Exceptional Flows:

Barber Updating Appointment & Reconciliation

Use-Case Diagram 6



Activity Diagram Use-Case 6

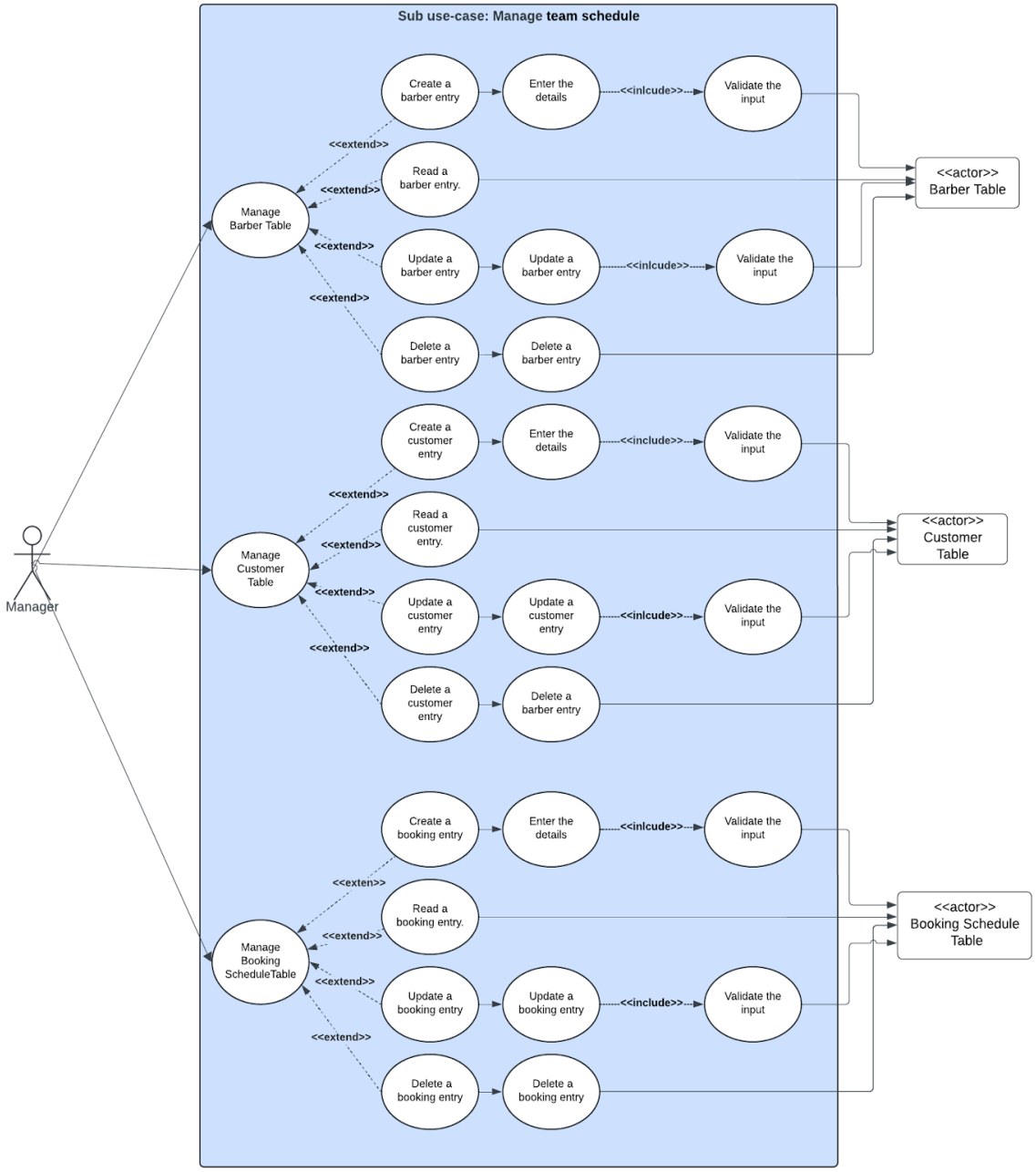


Use-Case Description For Use-Case 6

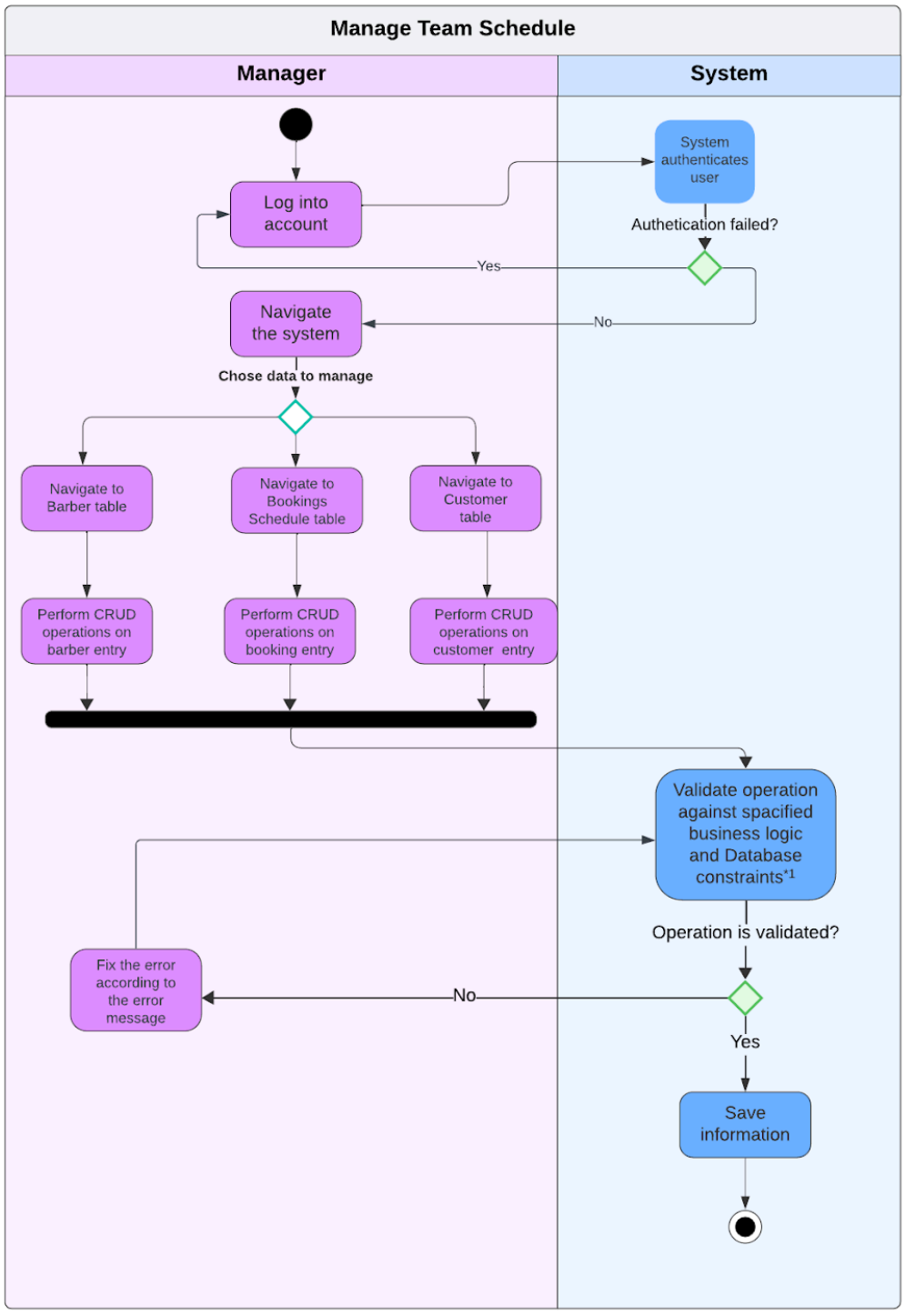
Use Case Name: Reconcile Bookings	ID: 6	Importance Level: High
Primary Actor: Barber	Use Case Type: Detailed, real	
Stakeholders and Interests: Barber - keeping track of their appointment and recording payment amount, type and tip breakdown.		
Brief Description: This use-case identifies the main steps barber have to go to update the status of their booked appointment to from scheduled to completed, canceled and no show, additionally record the payment type, amount and the breakdown of tip amount and service amount.		
Trigger: Barber updating the status of their appointments Type: Internal		
Relationships: Association: Barber Include: Verify login credential Extend: Sub-flow S1, Sub-flow S2, Sub-flow S3 Generalization:		
Normal Flow of Events: 6.1 Barber navigates the appointment system 6.2 Barber input login information 6.3 System authenticates login information 6.4 Select appointment to update 6.5 Update appointment status If the appointment is "Completed" sub-flow S1 is completed. If the appointment is "Rescheduled" sub-flow S2 is completed. If the appointment is "No Show" or "Canceled" sub-flow S3 is completed. 6.6 Save & Update Record		
Sub-Flows: S1: "Completed" 1. Barber to confirm appointment details 2. Barber Complete payment in cash or in card. 3. Barber input payment type in system 4. Barber input the breakdown of tip and service amount S2: "Rescheduled" 1. Barber asks the customer for possible appointment times 2. The Barber matches the client desired appointment times with available dates and times and schedules the new appointment. S3: "No Show" or "Canceled" 1. Barber find the appointment details in system 2. Baber set the appropriate status of the appointment		
Alternate/Exceptional Flows:		

Barber Managing Team's Schedule

Use-Case Diagram 7



Activity Diagram For Use-Case 7



Use-Case Description For Use-Case 7

Use Case Name: Manage Team Schedule / Manage The System	ID: 7	Importance Level: High
Primary Actor: Owner/Manager	Use Case Type: Detailed, essential	

Stakeholders and Interests:
Manager who wants to add/remove employees accounts from the system.
Manager who wants to see how busy the shop is.
Manager who wants to manage employees' schedules.

Brief Description:
This use-case identifies the main functionalities a manager could utilize in the system to manage it and improve business operations.

Trigger: Manager wants to manage.
Type: Internal

Relationships:
Association: Manager
Include: User Authentication
Extend:
Generalization:

Normal Flow of Events:
1. Manager logs in to the system.
2. Manager sees access to Users, Barbers, and Booking entries in the db.
3. Manager decides to manage a barber.
4. Manager decides to manage the Customer db table.
5. Manager decides to manage booking schedule
6. System validates entered details against the specified constraints.
7. System saves modified entries.

SubFlows:
1S.1 System authorizes and authenticates the user.

3S.1 Manager navigates to the Barber table.
3S.2 Manager needs to CRUD a barber entry.
3S.3 Manager finds an entry that requires a CRUD action.
3S.4 Manager fills out the details.
3S.5 Manager saves the change.

4S.1 Manager navigates to the Customer table.
4S.2 Manager needs to CRUD a customer entry.
4S.3 Manager finds an entry that requires a CRUD action.
4S.4 Manager fills out the details.
4S.5 Manager saves the change.

5S.1 Manager navigates to the Booking Schedule table.
5S.2 Manager needs to CRUD a booking schedule entry.
5S.3 Manager finds an entry that requires a CRUD action.
5S.4 Manager fills out the details.
5S.5 Manager saves the change.

Alternate/Exceptional Flows:
1E.1 System fails to authorize/authenticate the user.
1E.2 System returns the error message.
6E.1 System fails to validate user input against specified constraints.
6E.2 System displays an error message prompting the user to correct the error.

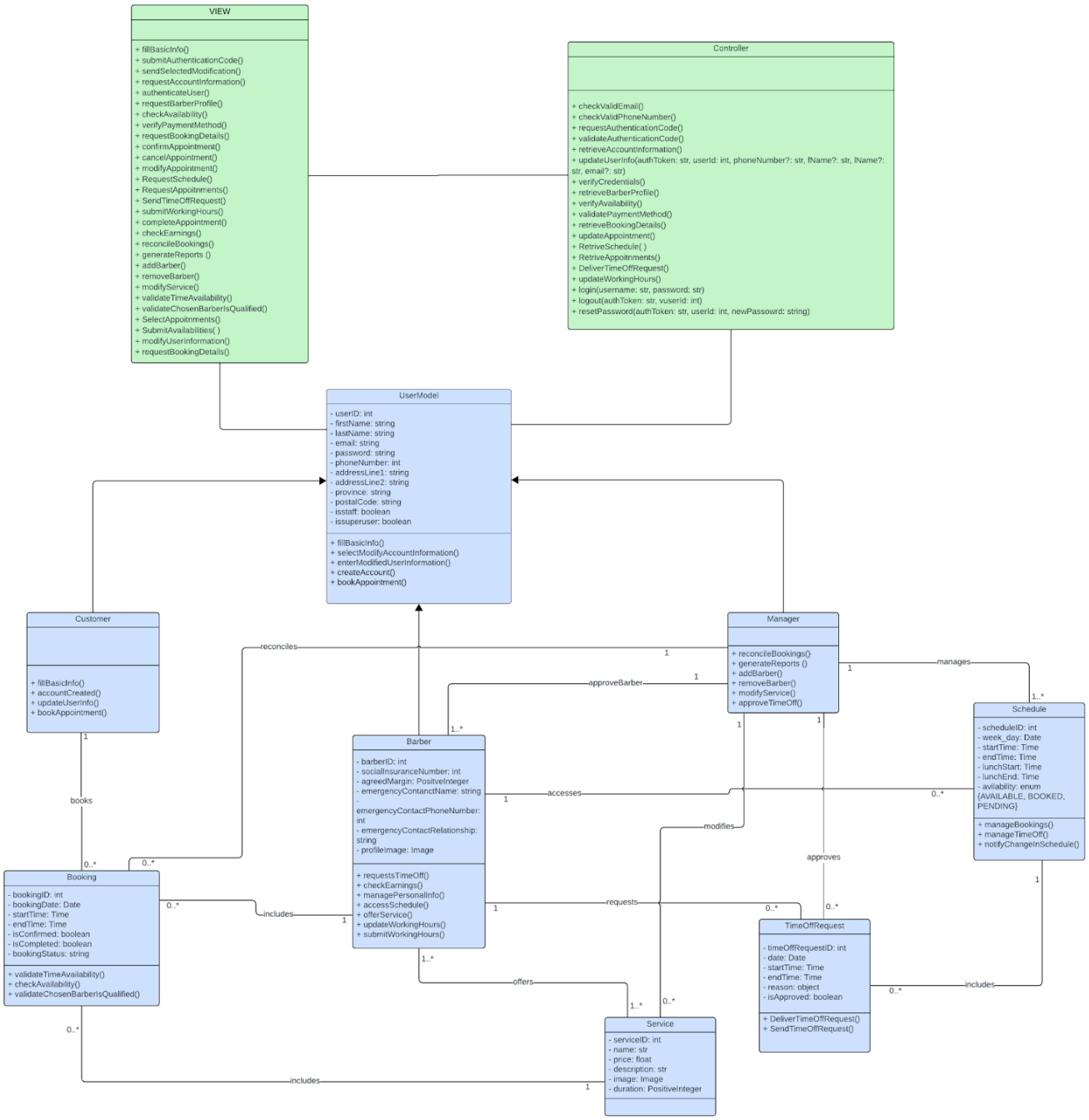
Structural Modeling

Structural modeling in our project is represented by a class diagram, which serves as a blueprint for the system's design. After determining what the system needs to do, the class diagram helps organize the objects that underpin the behavior modeled in the use case and activity diagrams. This organization aids in visualizing how the system's components interact and supports the implementation phase by providing a clear framework for development.

Additionally, CRC (Class-Responsibility-Collaborator) cards complement the class diagram by breaking down each class into its responsibilities and identifying its interactions with other classes. This approach encourages a more detailed understanding of the system and ensures that each class is well-defined and cohesive.

Below, you can find our system’s class diagram and 5 CRC cards for some of our important classes.

Class Diagram of The Sub-System



CRC Class-Responsibility-Collaboration Card

1. UserModel CRC Card

Front side:

Class Name: UserModel	**ID:** 1	**Type:** Domain class
Description: An individual that has or is creating an account.		**Associated Use Cases:** 1. Create an account (register) 2. Modify account information (updateUserInfo) 3. Book an appointment 4. Manage booking (confirm, modify, cancel)
Responsibilities Create an account login the account logout the account Modify account information Book an appointment Manage booking		**Collaborators** View Controller Customer Barber Booking

Back side:

Attributes: userID: int firstName: string lastName: string email: string password: string phoneNumber: int addressLine1: string addressLine2: string province: string postalCode: string isstaff: boolean issuperuser: boolean
Relationships: **Generalization (a-kind-of):** Barber, Customer **Aggregation (has-parts):** **Other Associations:** View, Controller, Booking

2. Barber CRC Card

Front side:

Class Name: Barber	**ID:** 2	**Type:** Concrete, Domain
Description: A barber is responsible for providing services to customers.		**Associated Use Cases:** 3. Book an Appointment. 5. Manage Personal Schedule. 6. Reconcile Bookings
Responsibilities Responsibility 1: Manage personal information. Responsibility 2: Request time off Responsibility 3: Time availability Responsibility 4: Check Earnings		**Collaborators** Collaborator1: UserModel Collaborator 2: TimeOffRequest Collaborator 3: Schedule Collaborator 4: Booking Collaborator 5: Service Collaborator 6: Manager

Back side:

Attributes: socialInsuranceNumber (int) agreedMargin (PositveInteger) emergencyContanctName (string) emergencyContactPhoneNumber (int) emergencyContactRelationship (string) profileImage (Image)
Relationships: **Generalization (a-kind-of):** UserModel **Aggregation (has-parts):** **Other Associations:** Schedule, TimeOffRequest, Service, Booking

3. Booking CRC Card

Front side:

Class Name: Booking	**ID:** 3	**Type:** Domain Class
Description: Booking is part of a booking system where each booking instance represents an appointment or service reservation made by the customers with an associated barber for a service.		**Associated Use Cases:** 3. Book an Appointment 4. Manage Booking 5. Manage personal schedule 6. Reconcile Bookings 7. Manage team schedule
Responsibilities Responsibility 1: Validate time availability for appointments. Responsibility 2: Verify barber qualifications for requested services.		**Collaborators** Collaborator 1: Customer Collaborator 2: Barber Collaborator 3: Service Collaborator 4: Manager

Back side:

Attributes: bookingID (int) bookingDate (Date) startTime (Time) endTime (Time) isConfirmed (boolean) isCompleted (boolean) bookingStatus (string)
Relationships: **Generalization (a-kind-of):** None **Aggregation (has-parts):** None **Other Associations:** Customer, Barber, Manager , Service

4. Schedule CRC Card

Front side:

Class Name: Schedule	**ID:** 4	**Type:** Domain Class
Description: A schedule represents time slots for bookings and availability for barbers and the manager.		**Associated Use Cases:** 3. Book an Appointment 4. Manage Booking 5. Manage Personal Schedule 6. Reconcile Bookings 7. Manage Team Schedule
Responsibilities Responsibility 1: Manage bookings Responsibility 2: Manage Time Off Responsibility 3: Notify change in schedule through bookings to customers		**Collaborators** Collaborator 1: TimeOffRequest Collaborator 2: Barber Collaborator 3: Manager

Back side:

Attributes: - scheduleID: int - week_day: Date - startTime: Time - endTime: Time - lunchStart: Time - lunchEnd: Time - avilability: enum {AVAILABLE, BOOKED, PENDING}
Relationships: **Generalization (a-kind-of):** None **Aggregation (has-parts):** None **Other Associations:** Barber, Manager, TimeOffRequest

5. Service CRC Card

Front side:

Class Name: Service	**ID:** 5	**Type:** Domain class
Description: Represents a service offered by the barbershop.		**Associated Use Cases:** 3. Booking appointment
Responsibilities		**Collaborators** Collaborator 1: Booking Collaborator 2: Barber Collaborator 3: Manager

Back side:

Attributes: serviceID (int) name (str) price (float) description (str) image (Image) duration (PositiveInteger)
Relationships: **Generalization (a-kind-of):** None **Aggregation (has-parts):** None **Other Associations:** Manager, Barber, Booking

Behavioral Modeling

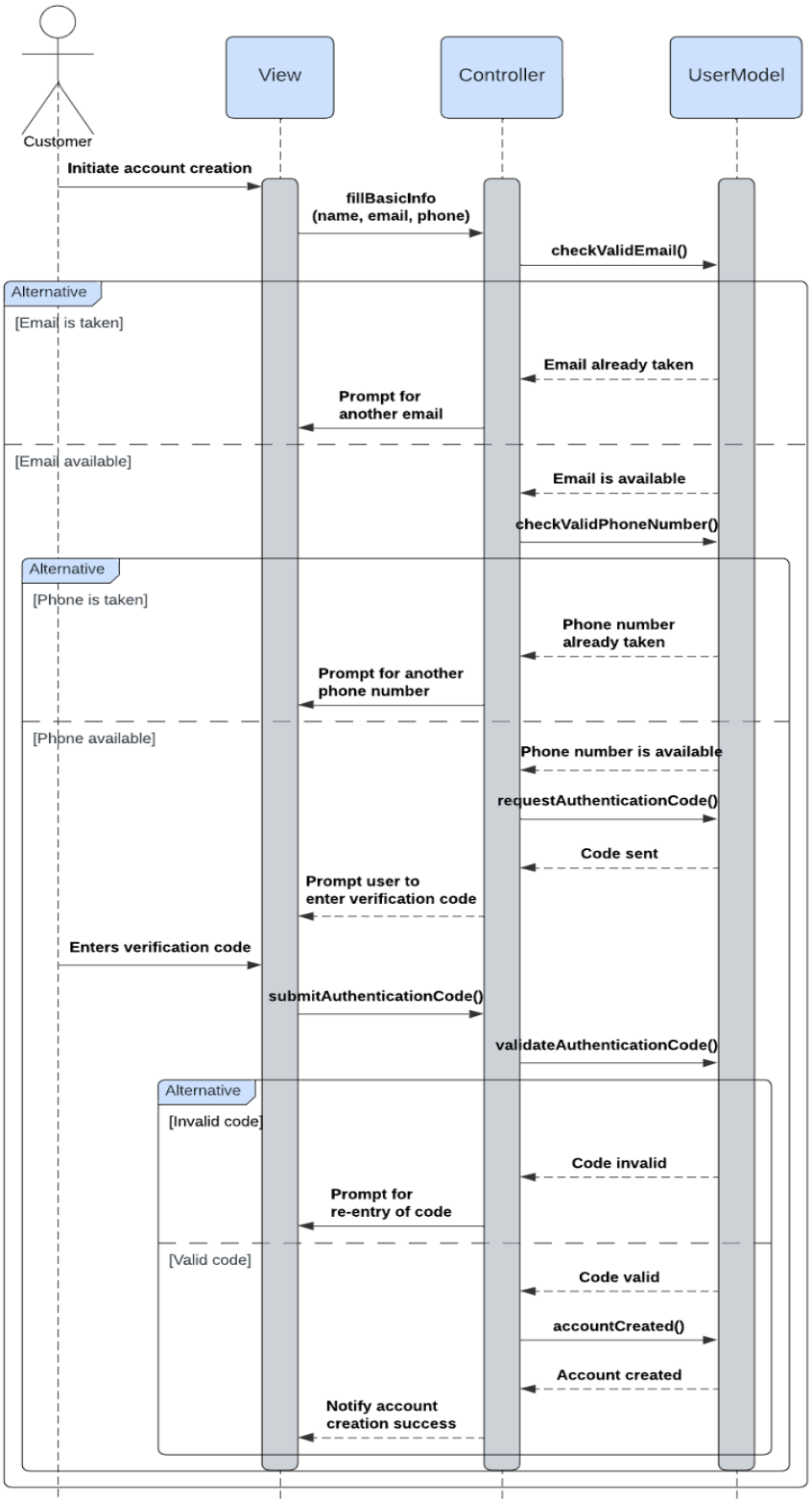
Behavioral modeling illustrates the internal dynamics of the system, focusing on how actors and objects within the problem domain interact over time to support each use case function. It provides a detailed understanding of how the system behaves in various scenarios, ensuring that the identified requirements of each use case are met effectively.

Two key tools for behavioral modeling are sequence diagrams and behavioral state machine diagrams. Sequence diagrams detail the flow of messages or interactions between objects in a specific scenario, focusing on the chronological order of events. Behavioral state machine diagrams, on the other hand, illustrate the various states an object can occupy and the transitions between these states in reaction to specific events.

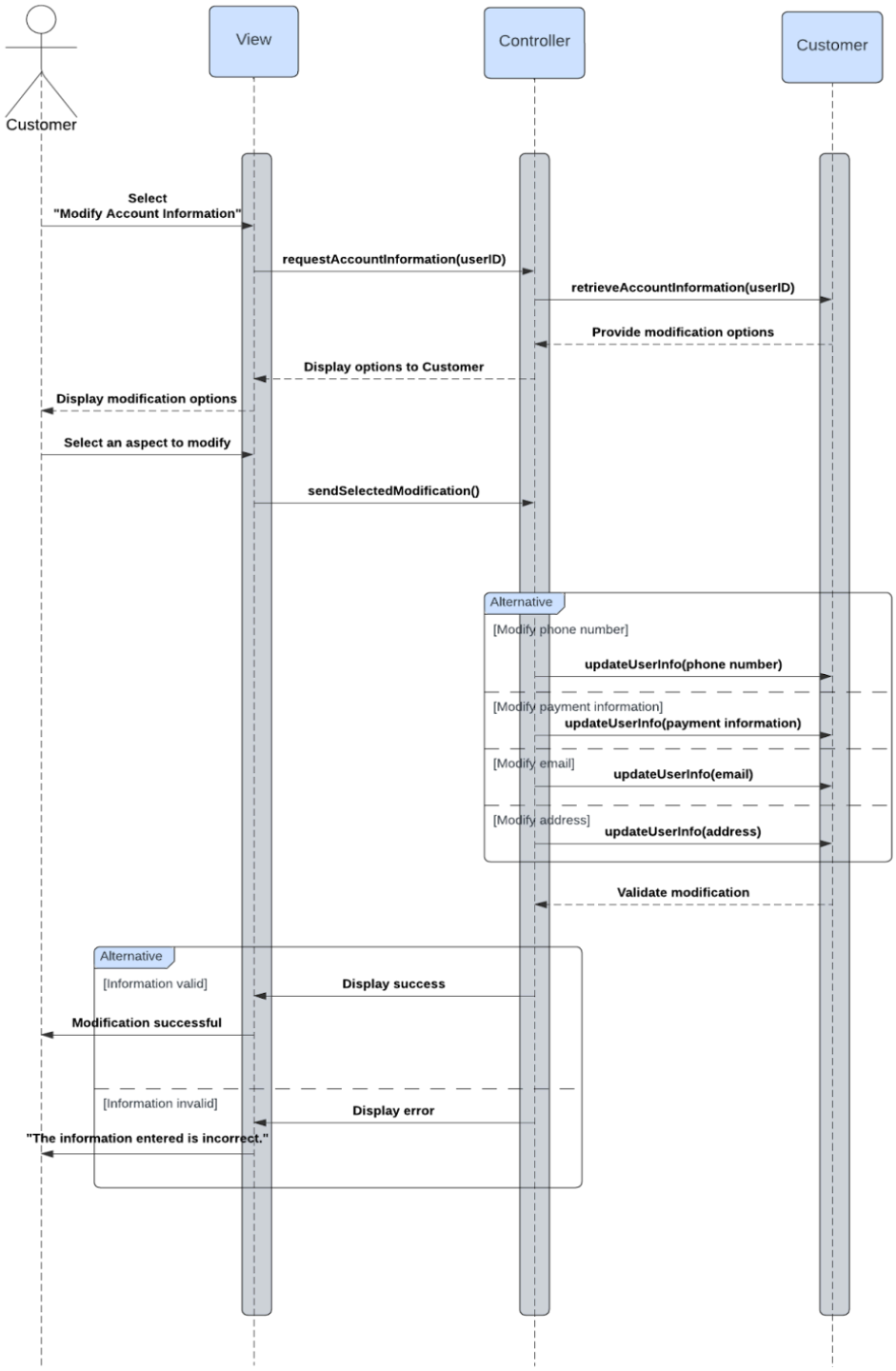
For this project, we have created seven sequence diagrams corresponding to each of our use cases and two behavioral state machine diagrams for the *TimeOffRequest* and *Booking* states, which represent the various stages a user progresses through in these processes.

Sequence Diagrams

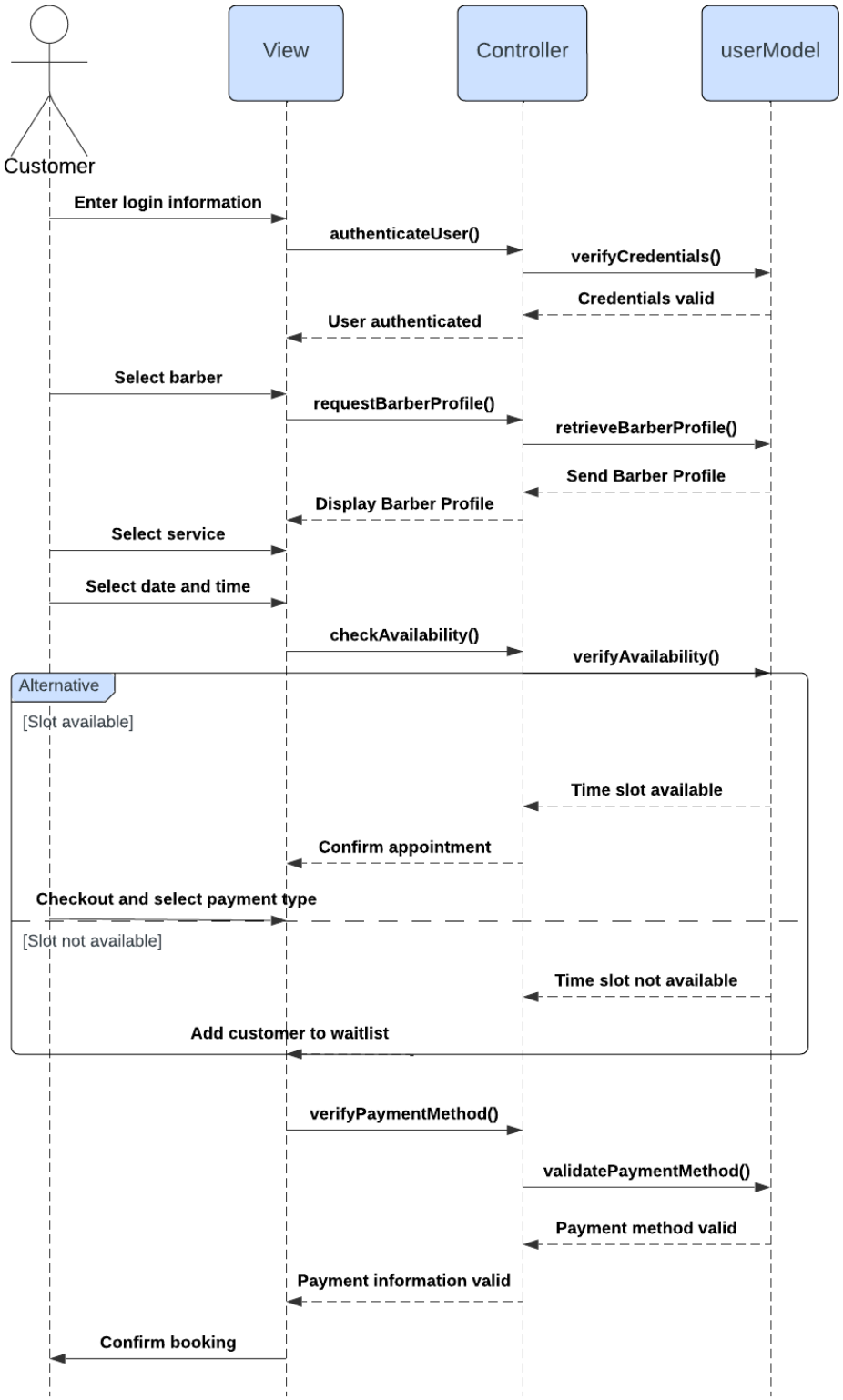
Sequence Diagram Use Case 1: Account Creation



Sequence Diagram Use Case 2: Modify Account Information

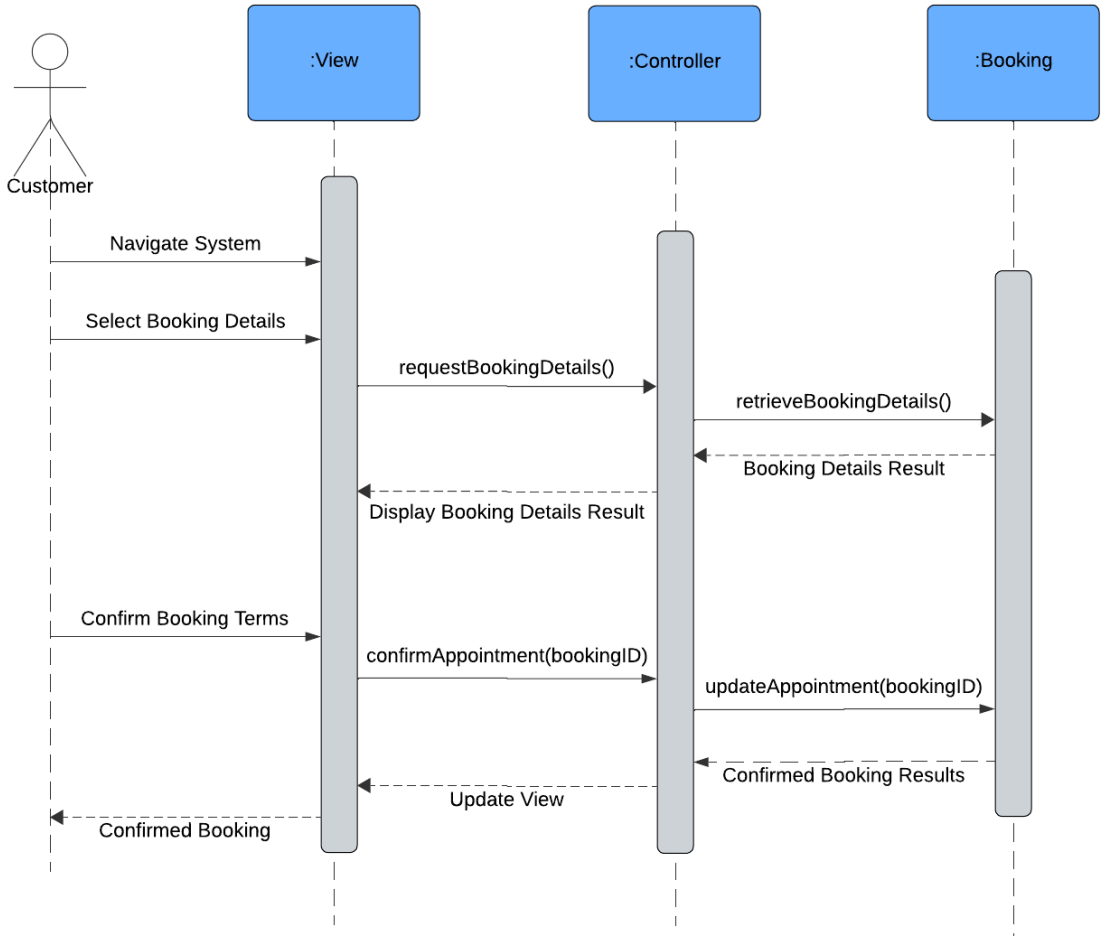


Sequence Diagram Use Case 3: Book an Appointment

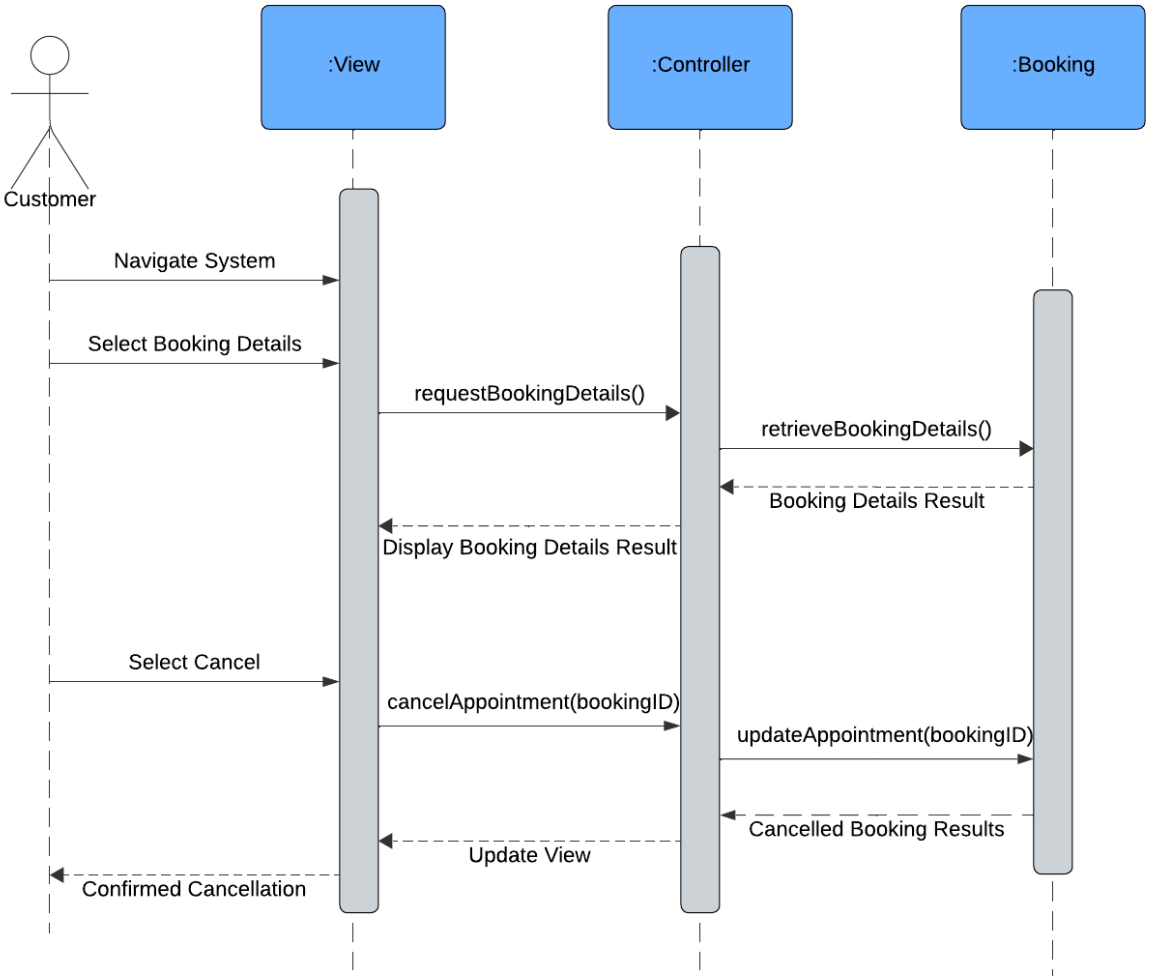


Sequence Diagrams Use Case 4: Manage Booking

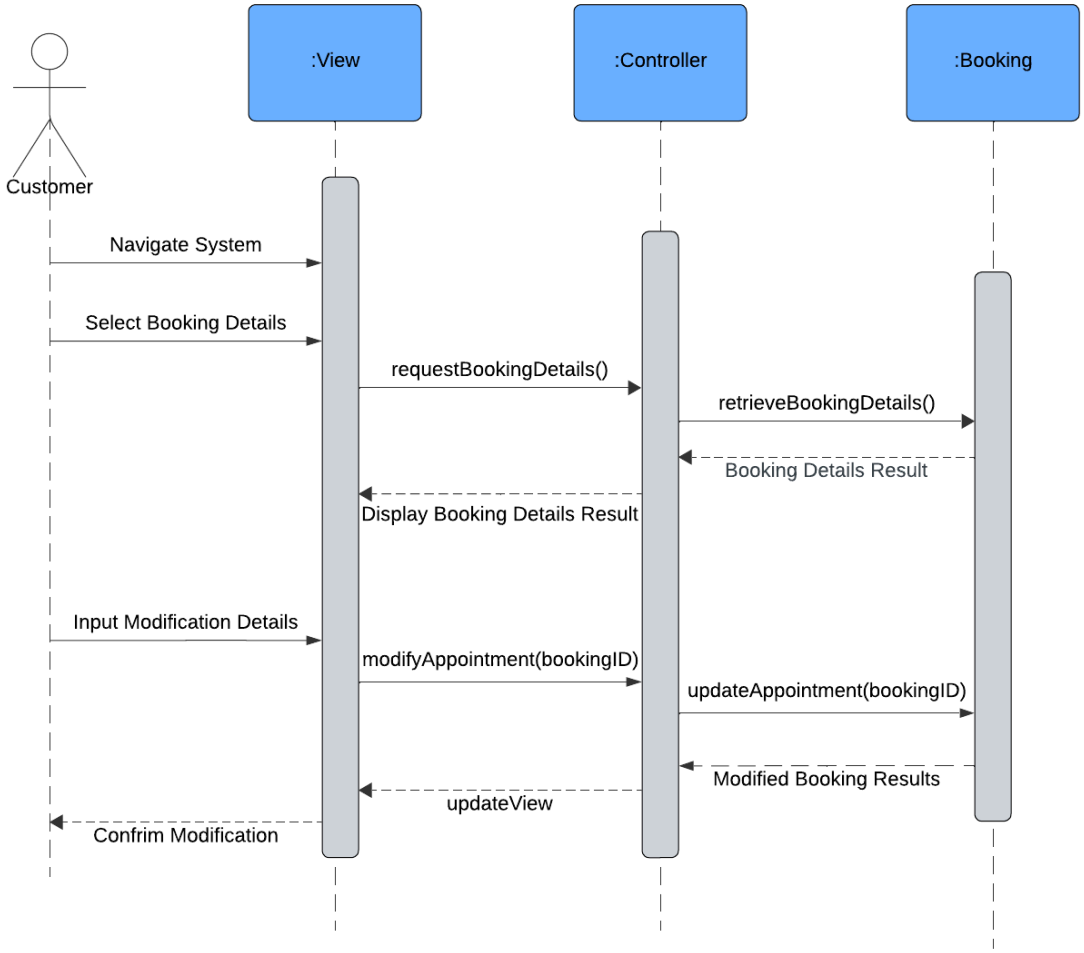
1. Confirm Booking:



2. Cancel Booking:

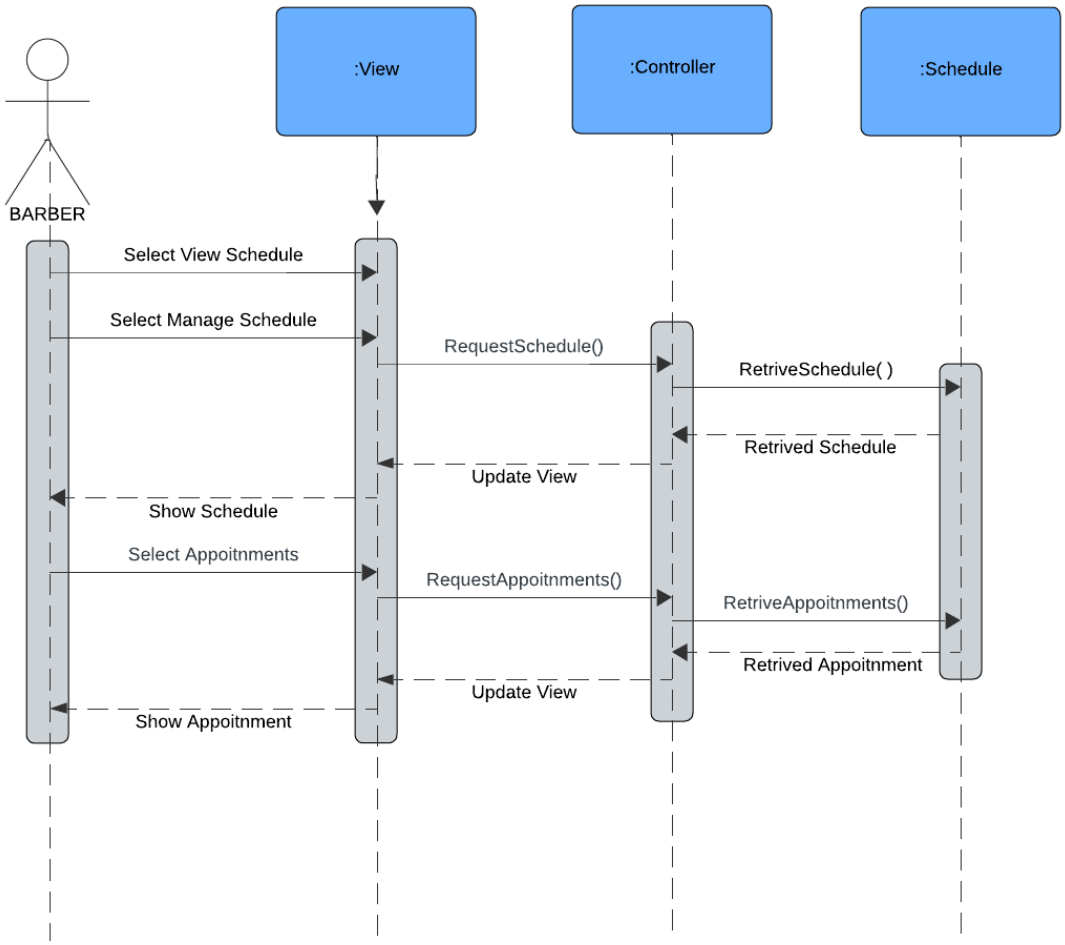


3. Modify Booking:

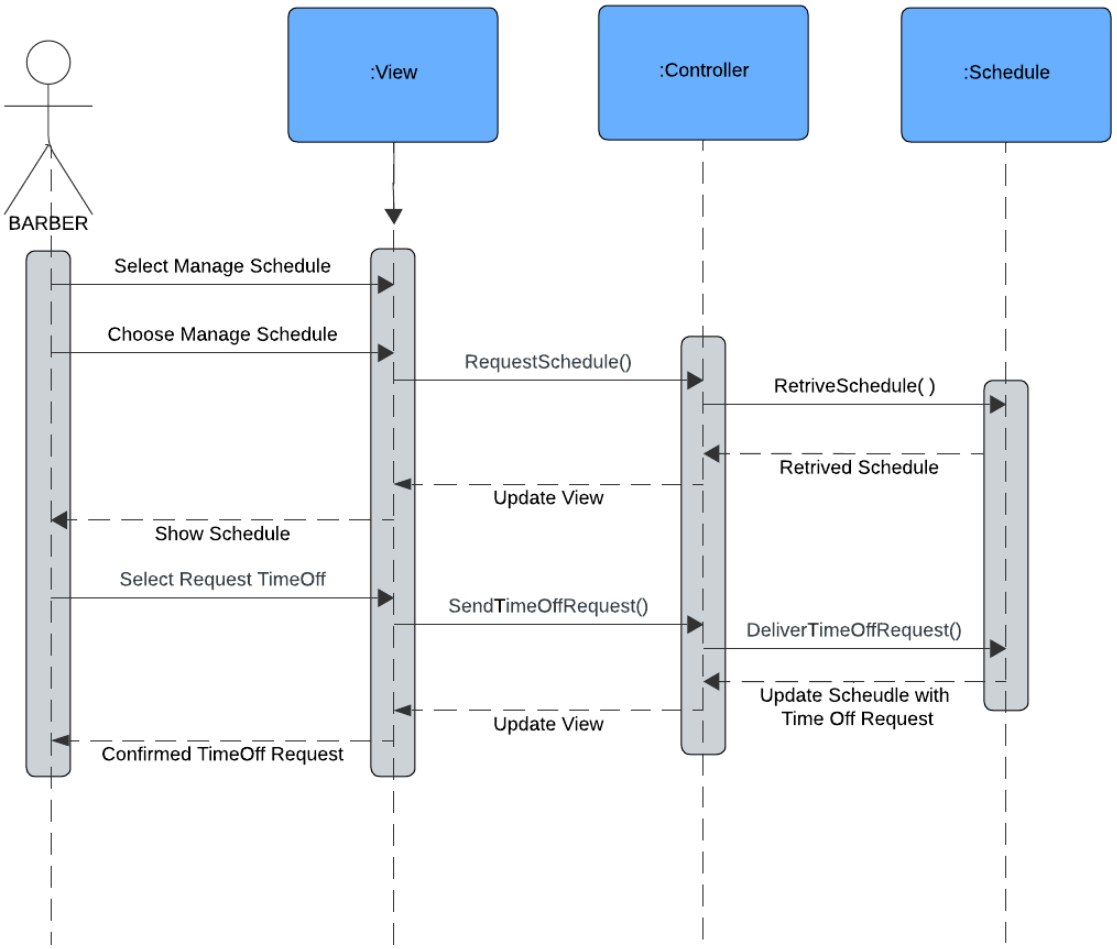


Sequence Diagrams Use Case 5: Manage Personal Schedule

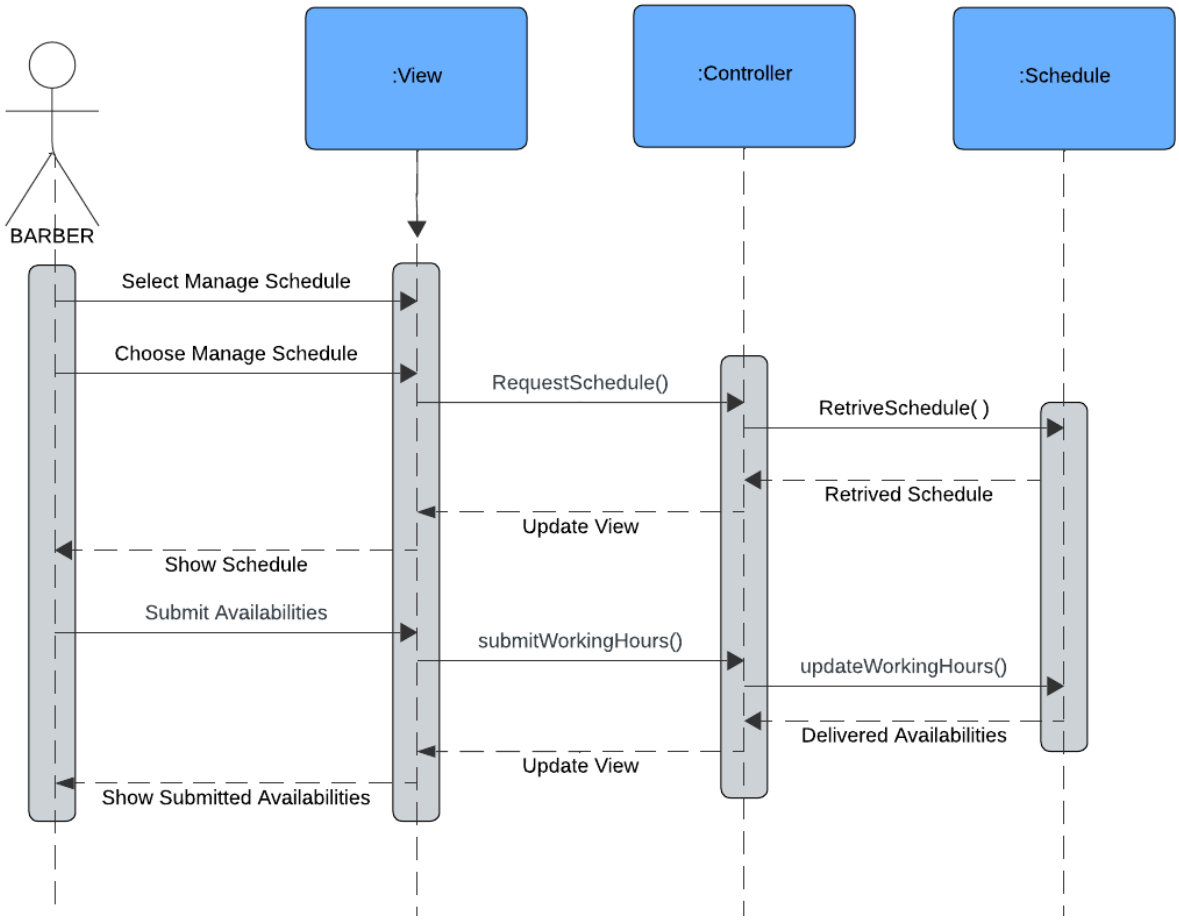
1. **Select Appointments:**



2. Time off Request:

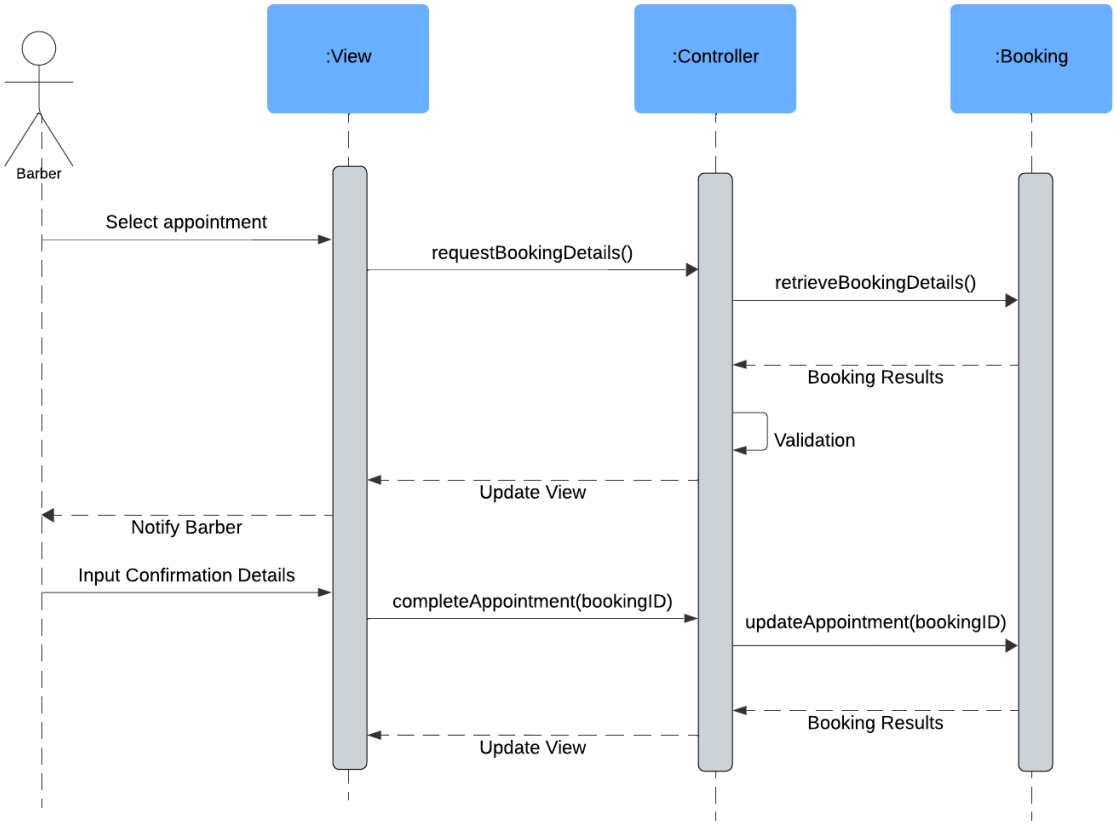


3. Submit Availabilities:

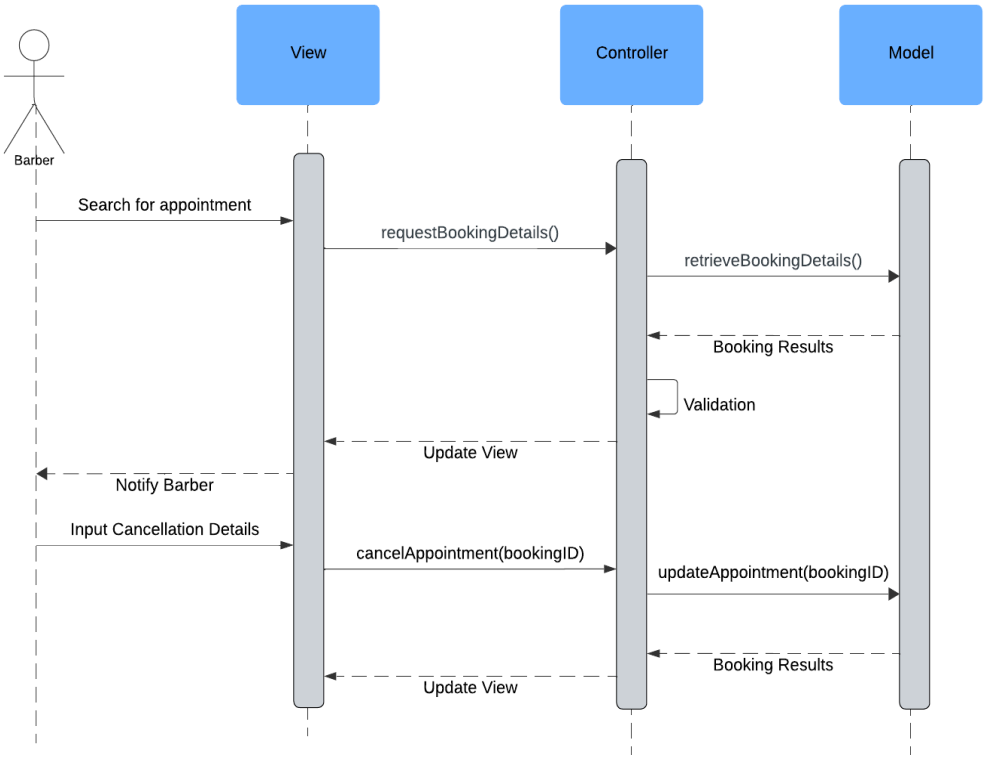


Sequence Diagrams Use Case 6: Reconcile Booking

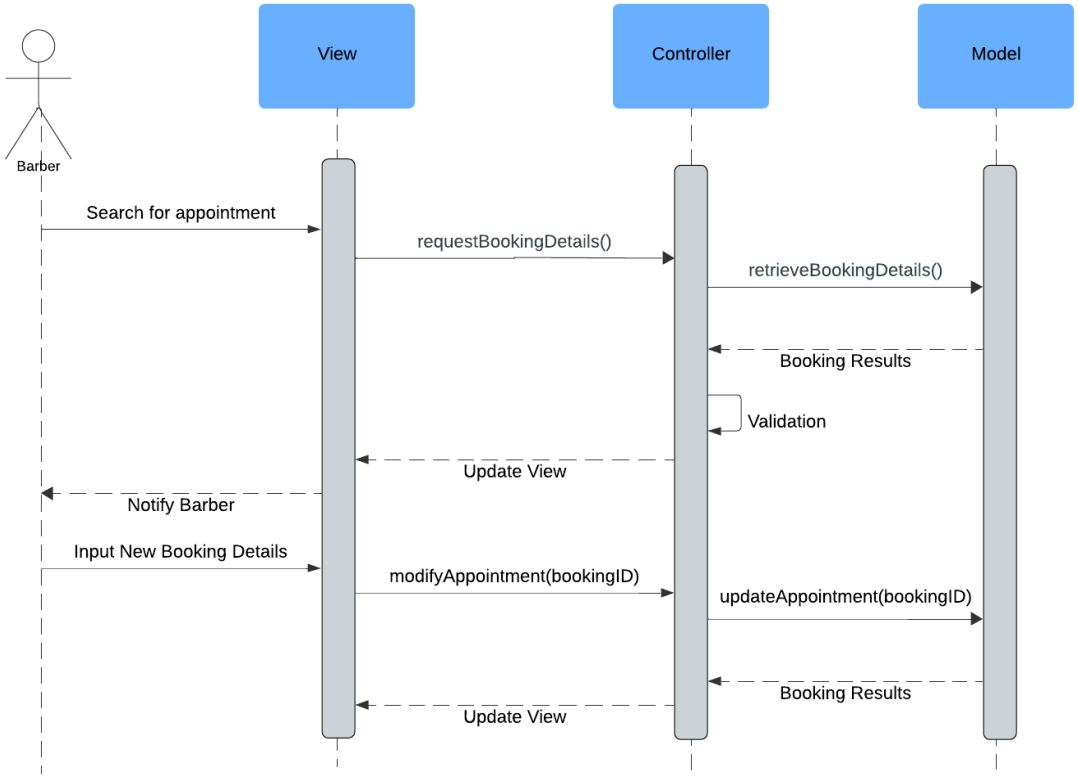
1. **Completed Booking (After client service):**



2. Update to No Show/Canceled Booking:

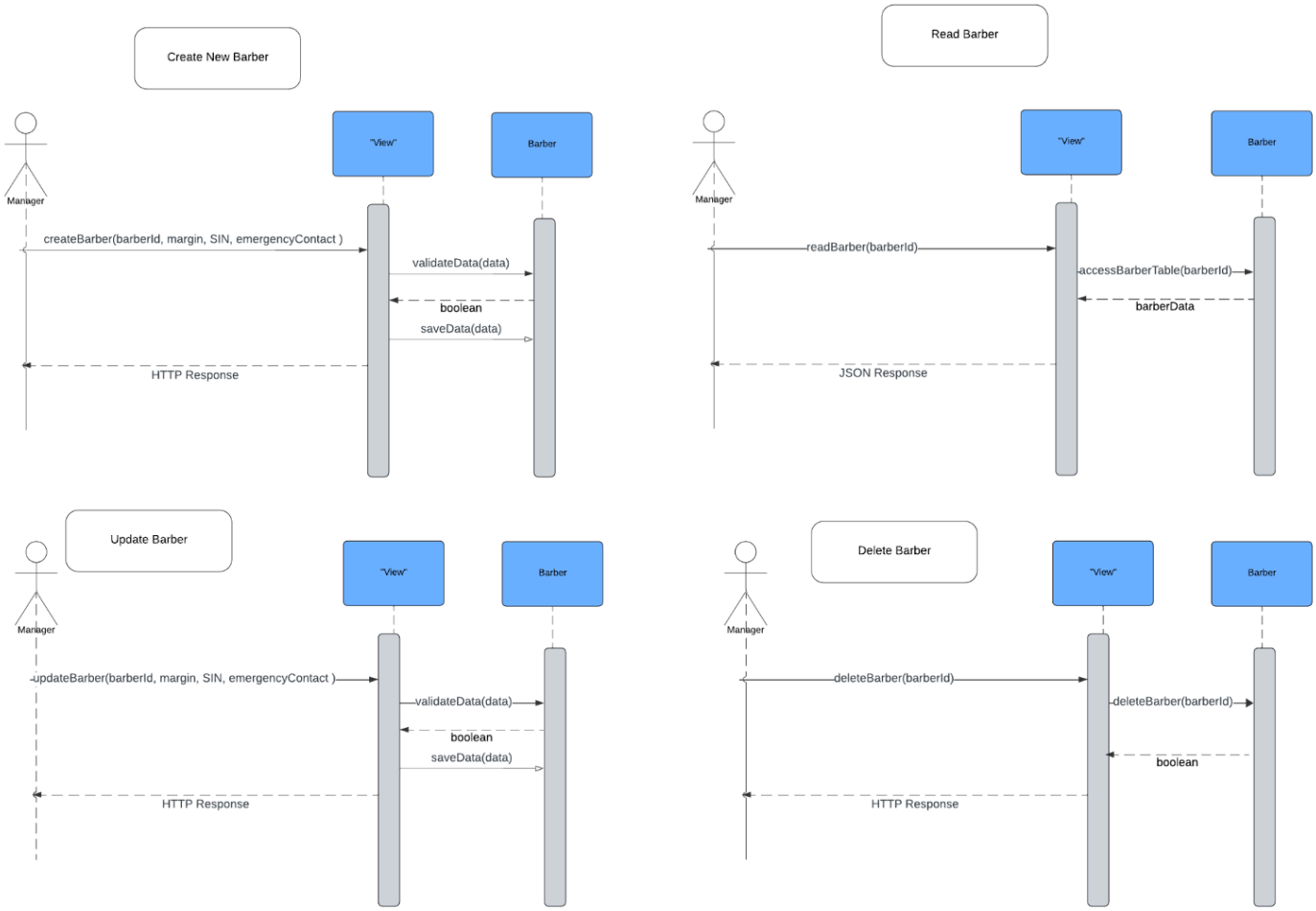


3. Reschedule Booking:

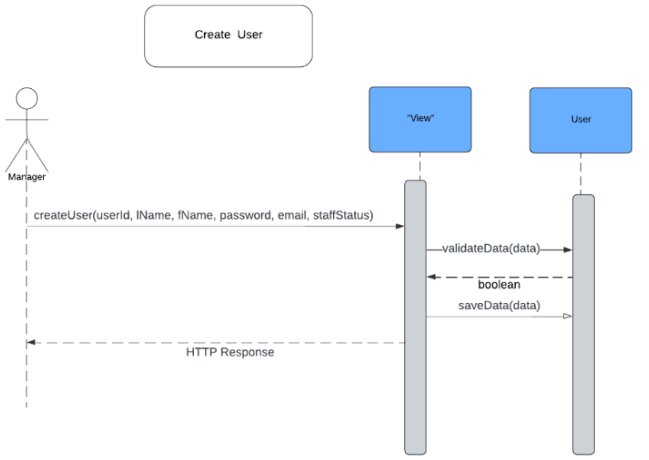


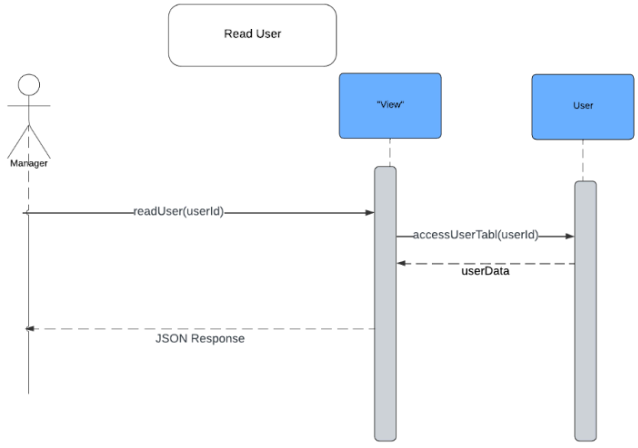
Sequence Diagrams Use Case 7: Manage Team Schedule

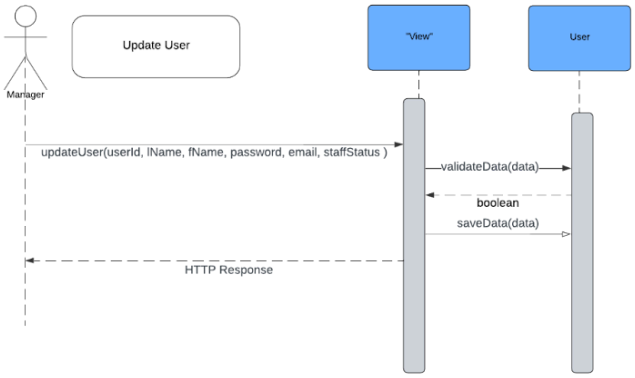
1. Manage Barber:

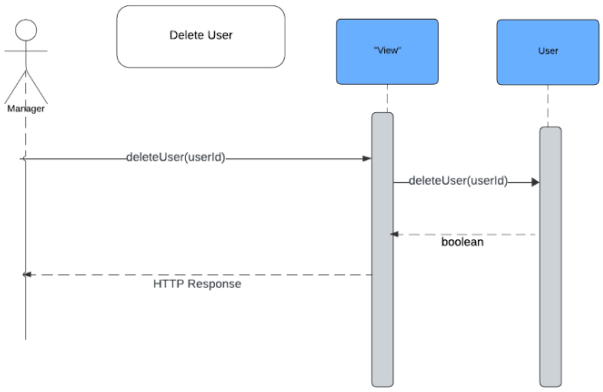


2. Manage User:

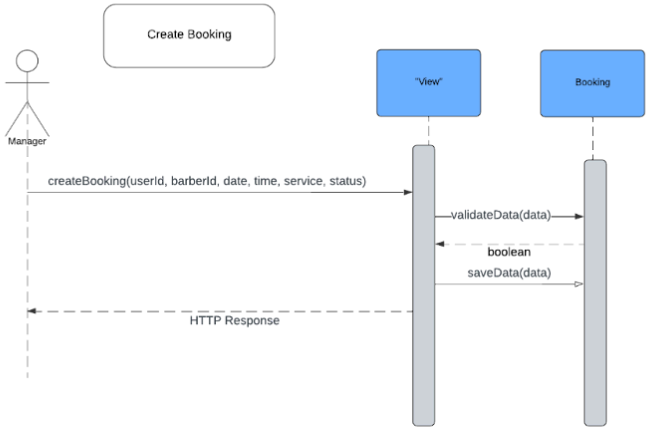


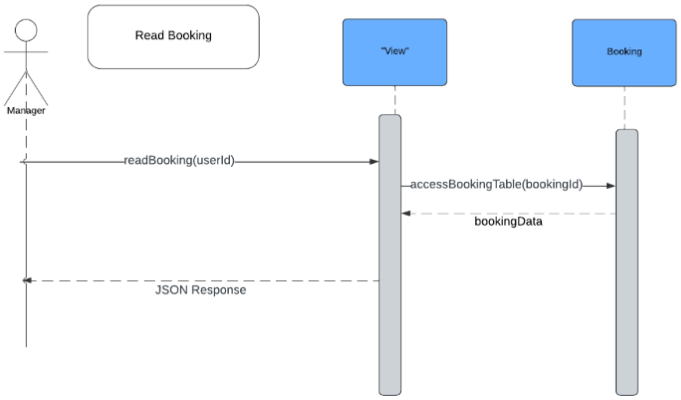


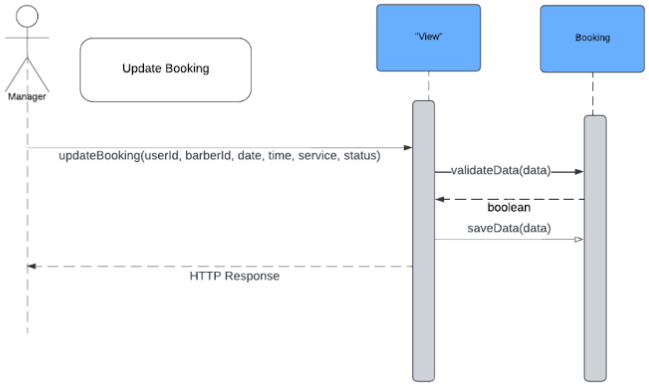


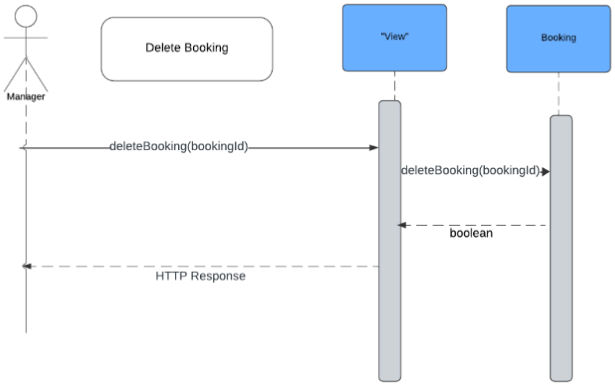


3. Manage Booking:



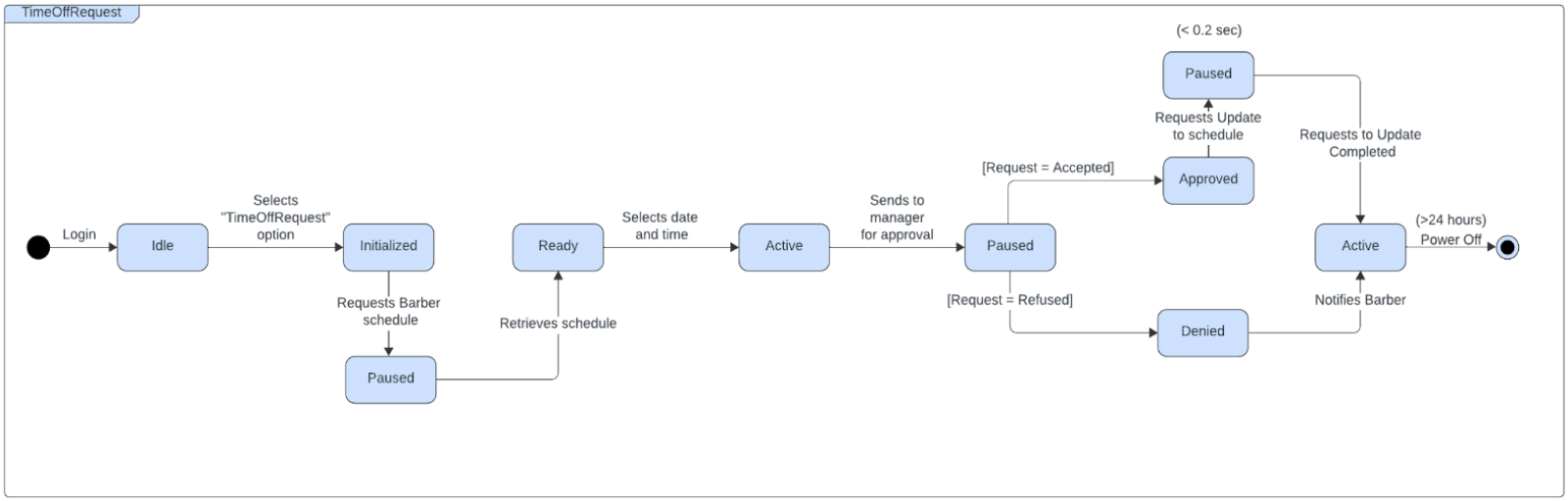




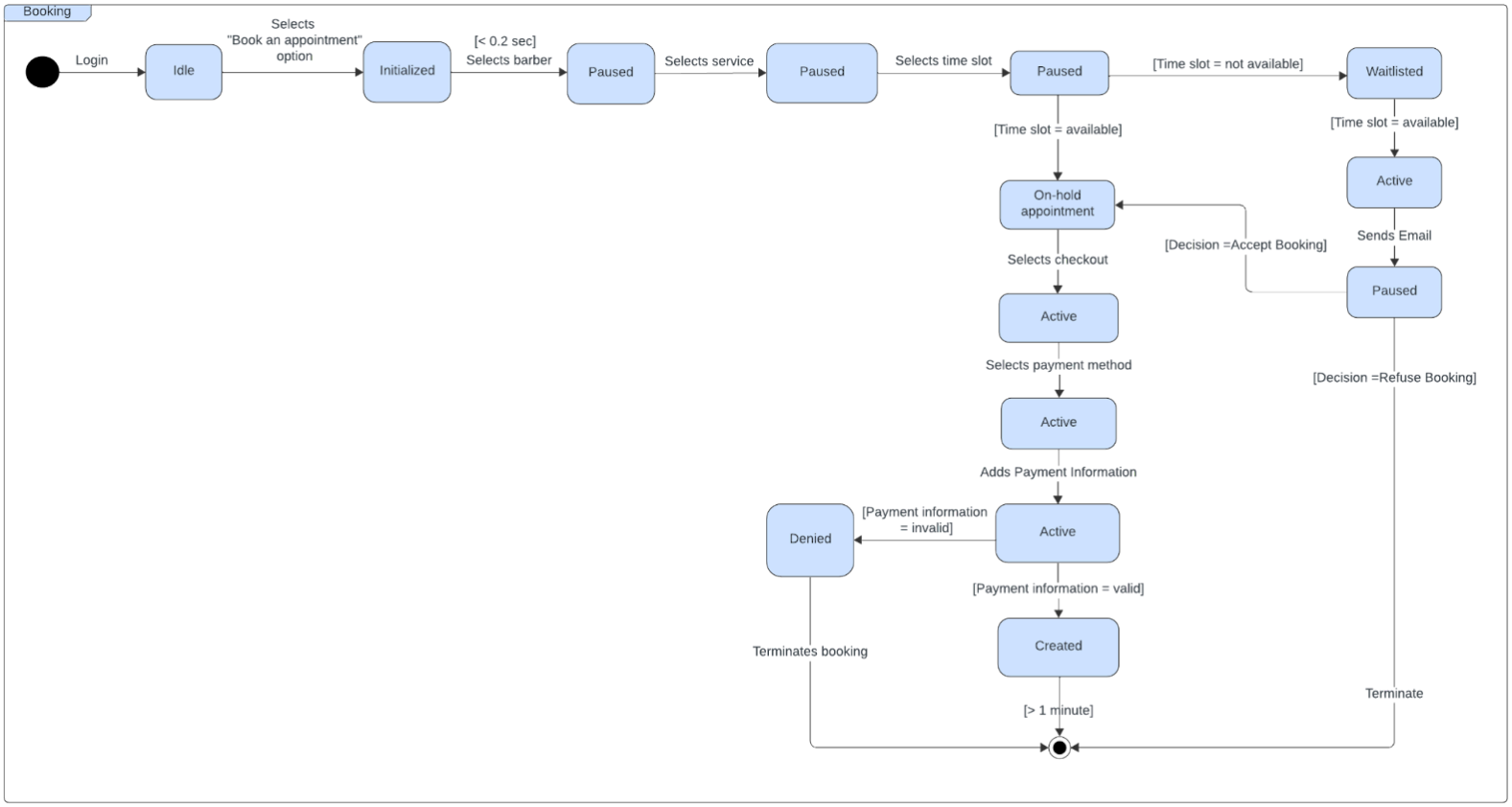


Behavioral State Machine Diagrams

1. TimeOffRequests

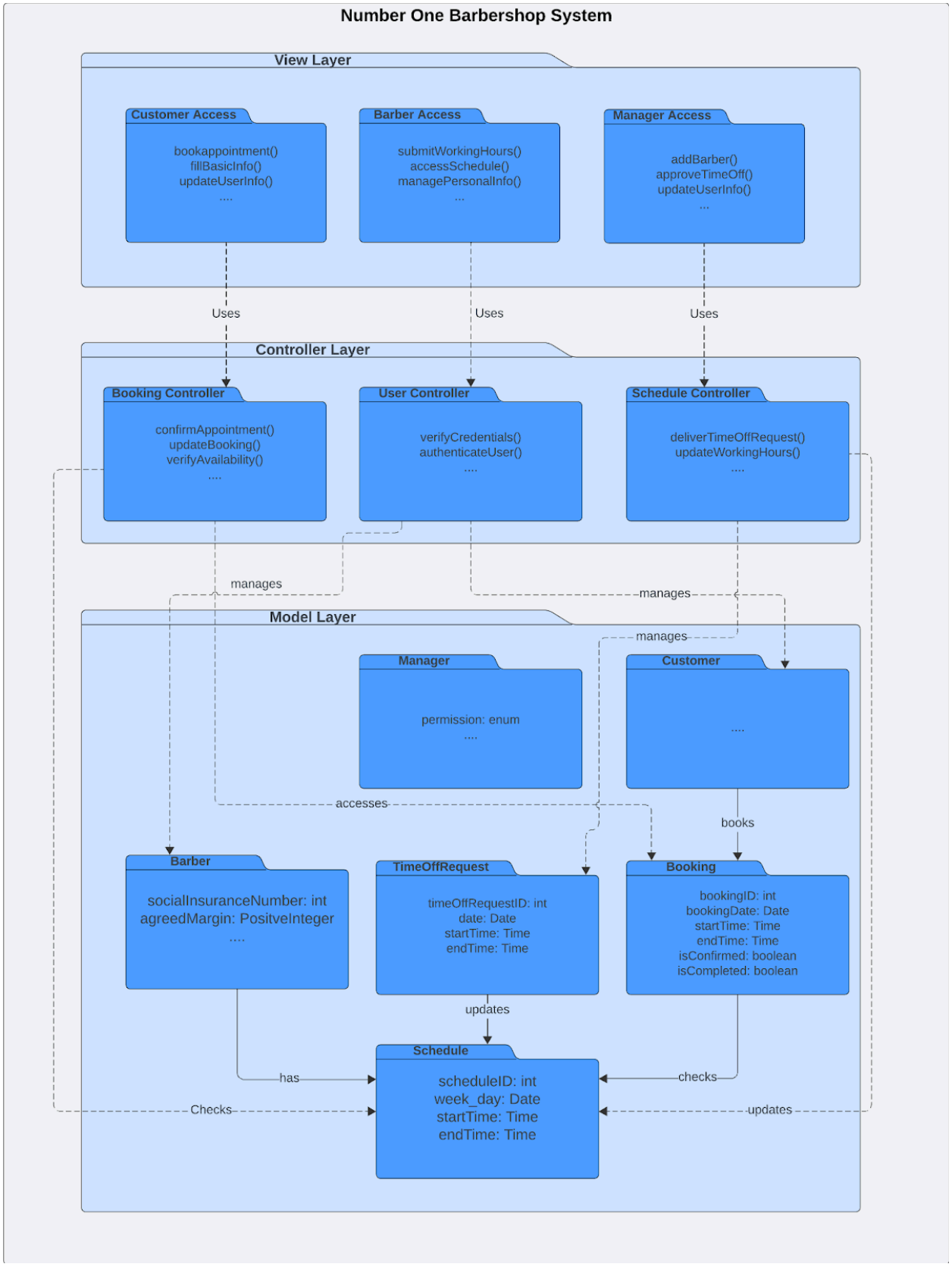


2. Booking


Booking
Login
Idle
Selects "Book an appointment" option
Initialized
[< 0.2 sec] Selects barber
Paused
Selects service
Paused
Selects time slot
Paused
[Time slot = not available]
Waitlisted
[Time slot = available]
Active
Sends Email
Paused
[Decision =Refuse Booking]
Terminate
[Time slot = available]
On-hold appointment
[Decision =Accept Booking]
Selects checkout
Active
Selects payment method
Active
Adds Payment Information
Active
[Payment information = invalid]
Denied
Terminates booking
[Payment information = valid]
Created
[> 1 minute]

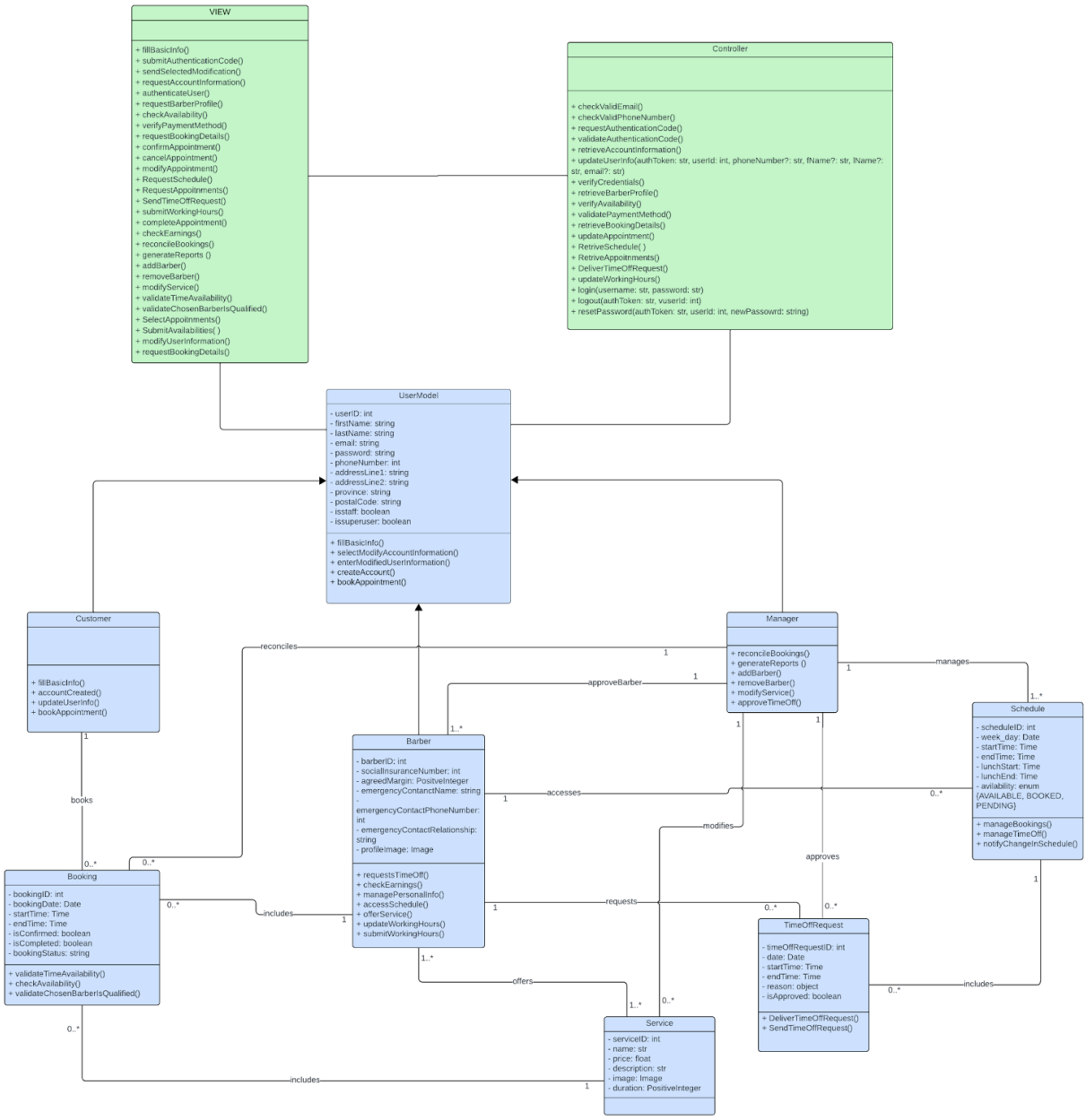
Package Diagram

The following package diagram illustrates the architectural design of the "Number One Barbershop System," developed using the Model-View-Controller (MVC) paradigm. The diagram provides a high-level representation of the system's organization, highlighting the modular separation of concerns among the three layers: Model, View, and Controller. The Model layer encapsulates the core business logic and data entities such as customers, barbers, appointments, and schedules. The View layer focuses on user interfaces tailored to different roles: customers, barbers, and managers. The roles ensure a role-specific functionality. The Controller layer mediates interactions between the Model and View, implementing key operations like booking management, user authentication, and schedule handling. We chose to leverage the MVC architecture, to promote modularity in the system, provide ease of maintenance, and prioritizing scalability, as reflected in the clearly defined relationships and dependencies between the packages.


Number One Barbershop System
View Layer
Customer Access
bookappointment()
fillBasicInfo()
updateUserInfo()
....
Barber Access
submitWorkingHours()
accessSchedule()
managePersonalInfo()
...
Manager Access
addBarber()
approveTimeOff()
updateUserInfo()
...
Uses
Uses
Uses
Controller Layer
Booking Controller
confirmAppointment()
updateBooking()
verifyAvailability()
....
User Controller
verifyCredentials()
authenticateUser()
....
Schedule Controller
deliverTimeOffRequest()
updateWorkingHours()
....
manages
manages
manages
Model Layer
Manager
permission: enum
....
Customer
....
accesses
books
Barber
socialInsuranceNumber: int
agreedMargin: PositveInteger
....
TimeOffRequest
timeOffRequestID: int
date: Date
startTime: Time
endTime: Time
Booking
bookingID: int
bookingDate: Date
startTime: Time
endTime: Time
isConfirmed: boolean
isCompleted: boolean
updates
Schedule
scheduleID: int
week_day: Date
startTime: Time
endTime: Time
has
checks
Checks
updates

Class and Method Design

Class Diagram



Method Contracts and Specifications

Method contracts and specifications are tools used to illustrate and specify how functions should operate. They define the expectations, responsibilities and outcomes to help users and system developers understand what each method is supposed to do.

Method contracts contain the Description of Responsibilities, preconditions that have to be true for the method to be called and postconditions that have to be true after the method is completed. Method specifications provide further information about how a method should operate.

We have identified method contracts and specifications for 4 of our methods: createAccount(), bookAppointment(), manageBooking() and modifyAccount().

1. createAccount()

Contract:

Method Name: createAccount()	Class Name: UserModel	ID: 1
Clients (Consumers): Customer		
Associated Use Cases: Create an account		
Description of Responsibilities: Responsible for handling the registration of a new user account.		
Arguments Received: anAccount: Account		
Type of Value Returned: Void		
Pre-Conditions: !email.includes(anAccount) - The email is not already in use !phonenumber.includes(anAccount) - The phone number is not already in use anAccount.phonenumber.isValid() - The number follows the expected format		
Post-Conditions: sendVerificationSMS(anAccount)		

Specifications:

Method Name: createAccount()	Class Name: UserModel	ID: 1
Contract ID: 1	Programmer: George	Date Due: 11/30/24

Programming Language:
• Python • TypeScript

Triggers/Events:
This method is triggered when a user submits a registration form to create a new account

Arguments Received: Data Type:	Notes:
Account	The customer's new account

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
Controller.requestAuthenticationCode()	boolean	Requests an authentication code
View.submitAuthenticationCode()	int	User submits an authentication code
Controller.validateAuthenticationCode()	int	Validates the authentication code
Controller.checkValidEmail()	String	Checks if email is already in use
Controller.checkValidPhoneNumber()	String	Checks if phone number is already in use

Argument Returned: Data Type:	Notes:
boolean	Returns true if the account is successfully created, otherwise false.

Algorithm Specification:

1) Validate the inputs:
   IF email or phoneNumber already exists in the system THEN
   - return false (account creation is failed due to duplication of information)
2) Verification Code
   Send a verification code to the user through their phoneNumber
   User submits verification code to the system
   IF verification code is correct THEN
   - return true

   ELSE
   - return false (user must re-enter code or request another one)
3) Return Result
   IF all steps are successfully completed THEN
   - return true (the account is completed)

   ELSE
   - return false (there is an error in previous steps)

Misc.Notes: N/A

2. bookAppointment()

Contract:

Method Name: bookAppointment()	Class Name: UserModel	ID: 2
Clients (Consumers): Appointment		
Associated Use Cases: Booking Appointment		
Description of Responsibilities: Implement the necessary behavior to create a new booking for a user. The method ensures that the barber and services selected are valid, the time slot is available, and the booking details are stored in the system.		
Arguments Received: 'serviceID','barberId','bookingDate','startTime','endTime','userID'		
Type of Value Returned: HTTPResponse, 201 for success, 400 for fail – bad request		
Pre-Conditions: To call bookAppointment you need to pass(('serviceID'),('barberId'),('date'),('timeSlot'),('userId'))		
Post-Conditions: N/A		

Specifications:

Method Name: bookAppointment()	**Class Name:** UserModel	**ID:** 2
Contract ID: 2	**Programmer:** George	**Date Due:** 11/30/23
Programming Language: • Python • TypeScript		
Triggers/Events: Customer Books an Appointment		

Arguments Received: **Data Type:**	**Notes:**
HTTPRequest	The Customer's New Appointment

Messages Sent & Arguments Passed: **ClassName.MethodName:**		
Request.query_params.get('serviceID')		
Request.query_params.get('barberId')		
Request.query_params.get('date')		
Request.query_params.get('timeSlot')		
Request.query_params.get('userId')		
Barber.createObject([serviceId, barberId, date, timeSlot, userId])		

Argument Returned: Data Type:	Notes:
HTTPResponse	201 for success 400 for fail – bad request

Algorithm Specification:
Retrieve query params from query string
Validate each parameter
if Validation:{
  Booking.createObject(
    pass retrieved query params
)
   return HTTPRequest(2001, 'Success: Appointment Created')
} else: {
  return HTTPResponse(400, 'Bad Request: validation failed')
}

Misc.Notes:
bookAppointment() function is an exposed endpoint that takes POST http requests to create an appointment and it interacts with Booking model class to create the appointment entry inside the database table Booking.

3. manageBooking()

Contract:

Method Name: manageBooking()	Class Name: UserModel	ID: 3
Clients (Consumers): Booking		
Associated Use Cases: Manage Booking.		
Description of Responsibilities: Implement the necessary behavior to manage the customers booking. The method ensures the possibility of confirming.		
Arguments Received: • authToken: String - Authentication token for the user managing the booking. • username: String - The username of the user making changes to the appointment. • aBooking: Booking - The existing booking to be managed. • newDetails: BookingDetails - The updated details for the booking.		
Type of Value Returned: Void		
Pre-Conditions: bookings.includes(aBooking) This indicates that the method can only be executed if a Booking already exists in the list of bookings.		
Post-Conditions:		

Specifications:

<table>
<tr><td>Method Name: manageBooking()</td><td>Class Name: UserModel</td><td>ID: 3</td></tr>
<tr><td>Contract ID: 3</td><td>Programmer: George</td><td>Date Due: 11/30/24</td></tr>
<tr><td colspan="3">Programming Language:
• Python • TypeScript</td></tr>
<tr><td colspan="3">Triggers/Events:
This method is triggered when a user attempts to modify or update an existing booking.</td></tr>
<tr><td colspan="3"><table>
<tr><td>Arguments Received:
Data Type:</td><td>Notes:</td></tr>
<tr><td>Booking</td><td>The customer’s new updated booking.</td></tr>
</table></td></tr>
<tr><td colspan="3"><table>
<tr><td>Messages Sent & Arguments Passed:
ClassName.MethodName:</td><td>Argument
Data Type:</td><td>Notes:</td></tr>
<tr><td>Authentication.validate()</td><td>string</td><td>Validates the user using the authentication token.</td></tr>
<tr><td>BookingList.get()</td><td>int</td><td>Gets booking from the db if it exists.</td></tr>
<tr><td>aBooking.update()</td><td>Booking</td><td>Update and save the booking with the new details.</td></tr>
</table></td></tr>
<tr><td colspan="3"><table>
<tr><td>Argument Returned:
Data Type:</td><td>Notes:</td></tr>
<tr><td>HTTPResponse</td><td>201 for success
400 for fail</td></tr>
</table></td></tr>
<tr><td colspan="3">Algorithm Specification:

Retrieve query params from query string
Validate each parameter
if Validation:{
Booking.manageObject(
pass retrieved query params
)
return HTTPRequest(201, ‘Success: Booking Managed Successfully’)
} else: {
return HTTPResponse(400, ‘Bad Request: validation failed’)
}</td></tr>
<tr><td colspan="3">Misc.Notes:
manageBooking() function is an exposed endpoint that takes POST http requests to manage bookings. It interacts with Booking model class to update or cancel entries in the Booking database table.</td></tr>
</table>

4. modifyAccount()

Contract:

Method Name: modifyAccountInfo()	Class Name: UserModel	ID: 4
Clients (Consumers): Account		
Associated Use Cases: Use case #2: Modify Account Information		
Description of Responsibilities: This method allows customers to modify their account information, such as phone number, email, address, payment information, by accessing the account.		
Arguments Received: - userID: int - firstName: string - lastName: string - email: string - phoneNumber: int - addressLine1: string - addressLine2: string		
Type of Value Returned: Void		
Pre-Conditions: accounts.includes(account) To call modifyAccountInfo the user needs to have an account.		
Post-Conditions: None		

Specifications:

<table>
<tr><td>Method Name:
modifyAccountInfo()</td><td>Class Name: UserModel</td><td>ID: 4</td></tr>
<tr><td>Contract ID: 4</td><td>Programmer: George</td><td>Date Due: 11/30/24</td></tr>
<tr><td colspan="3">Programming Language:
• Python • TypeScript</td></tr>
<tr><td colspan="3">Triggers/Events:
Customer modifies his account information</td></tr>
<tr><td>Arguments Received:
Data Type:</td><td colspan="2">Notes:</td></tr>
<tr><td>HTTPRequest</td><td colspan="2">The customer’s updated account information</td></tr>
<tr><td>Messages Sent & Arguments Passed:
ClassName.MethodName:</td><td>Argument
Data Type:</td><td>Notes:</td></tr>
<tr><td>Account.modifyObject</td><td>string</td><td>Holds validated account info, such as username, email and phone number.</td></tr>
<tr><td>Validator.validateEmail</td><td>string</td><td>Verify accurate email format</td></tr>
<tr><td>Argument Returned:
Data Type:</td><td colspan="2">Notes:</td></tr>
<tr><td>HTTPResponse</td><td colspan="2">201 for success
400 for fail – bad request</td></tr>
<tr><td colspan="3">Algorithm Specification:

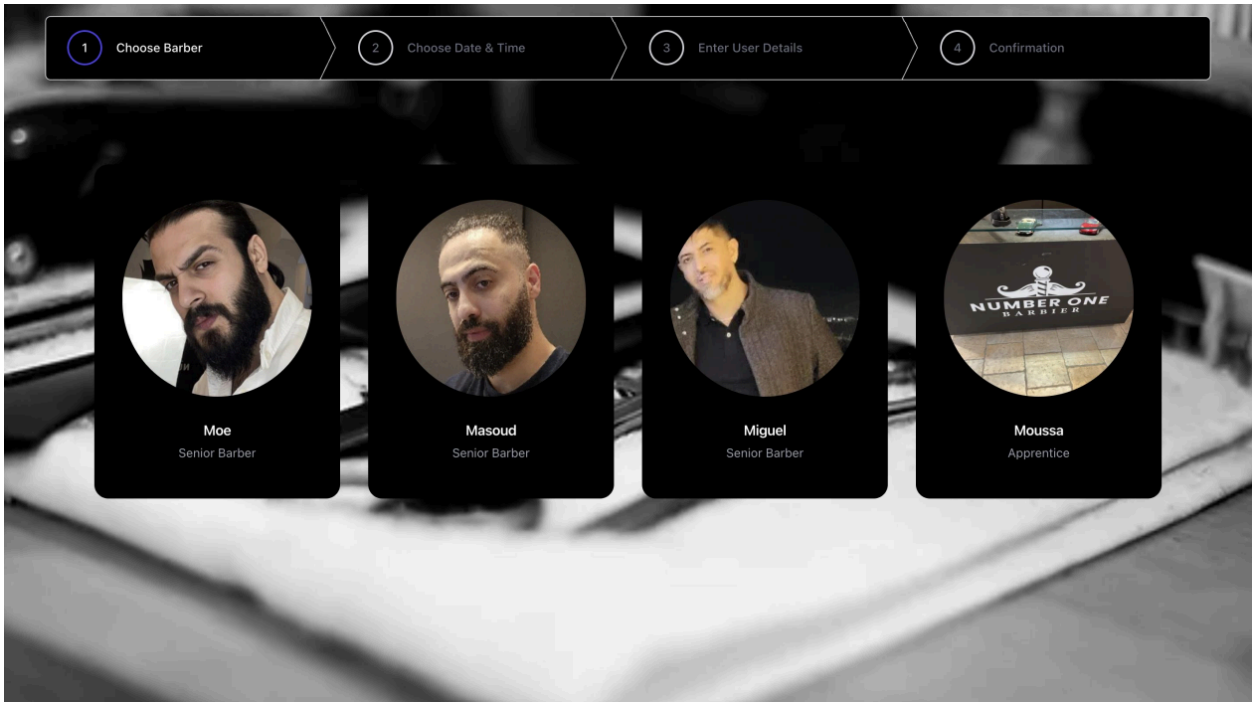
Retrieve query params from query string
Validate each parameter
if Validation is successful:{
Account.modifyObject(
pass retrieved query params
)
return HTTPRequest(201, ‘Success: Account Information Updated’)
} else: {
return HTTPResponse(400, ‘Bad Request: Invalid Account Information’)
}</td></tr>
<tr><td colspan="3">Misc.Notes:</td></tr>
</table>

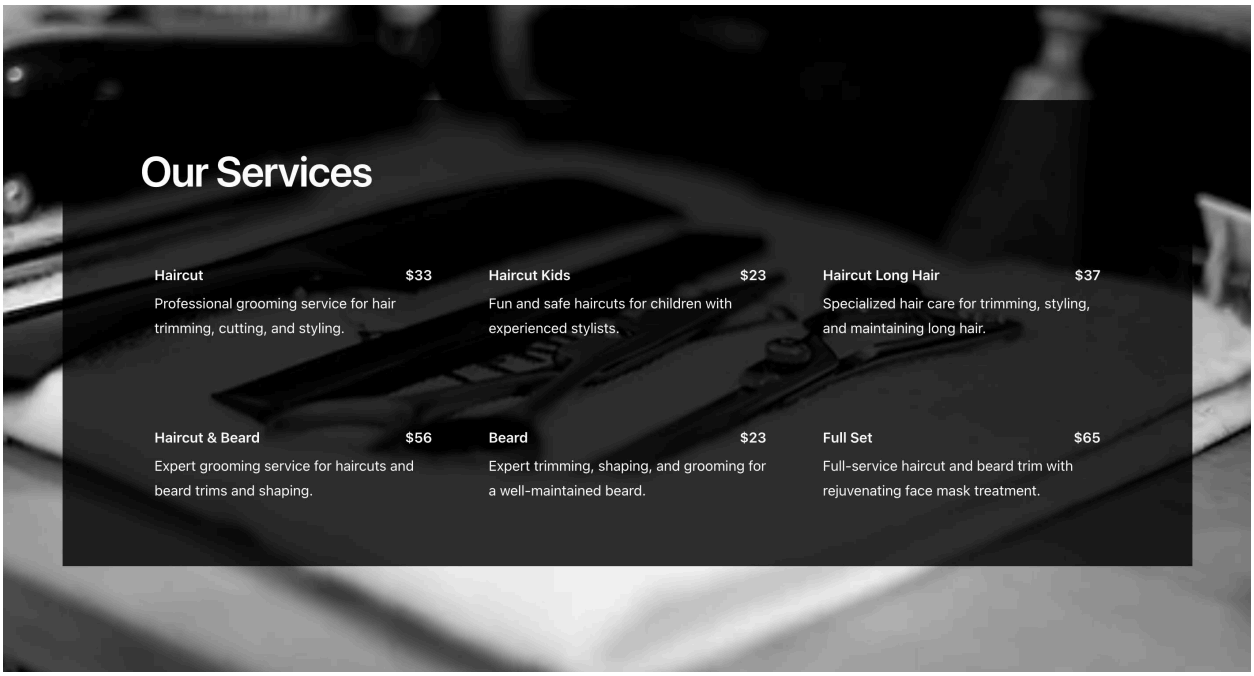
HCI Design

1. User Books an Appointment

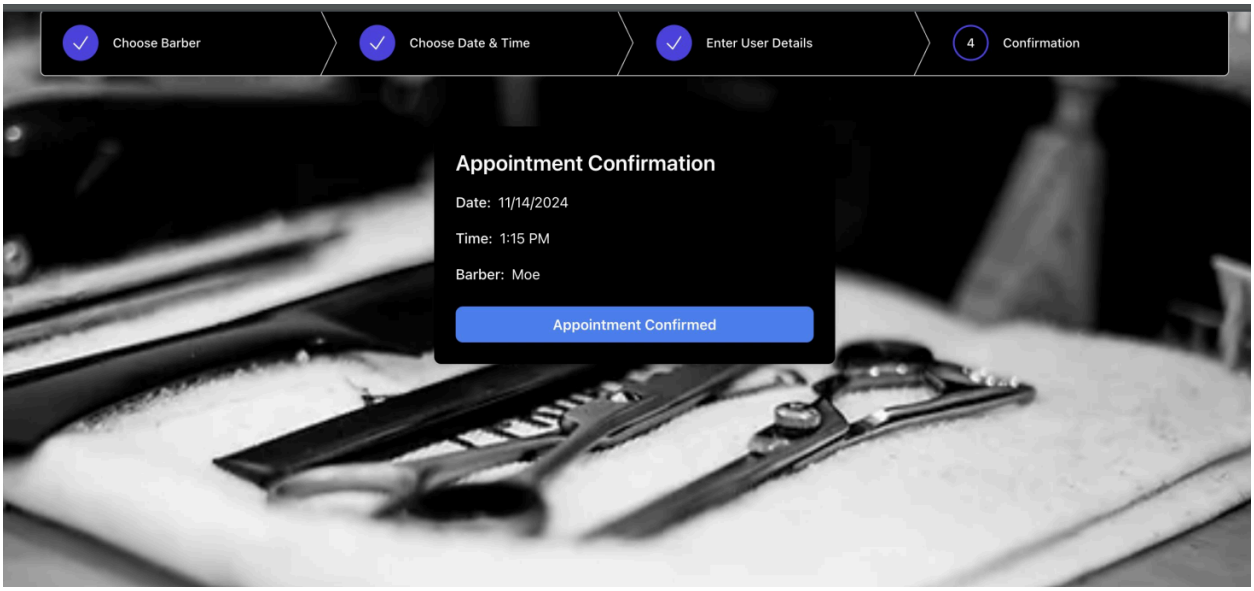
Use Scenario Description: User selects a service, barber and the date and time and receives a confirmation
Step 1: The user opens the website and scrolls to “Our Services”
Step 2: The user selects the type of service
Step 3: The system displays the different barbers available who provide the service
Step 4: The user selects the barber
Step 5: The system displays time slots based on barber’s availabilities
Step 6: The user selects date and time
Step 7: The system confirms availability
Step 8: The user receives a confirmation on the appointment

Navigation Diagram


1 Choose Barber
2 Choose Date & Time
3 Enter User Details
4 Confirmation
Moe
Senior Barber
Masoud
Senior Barber
Miguel
Senior Barber
Moussa
Apprentice
NUMBER ONE
BARBIER


Our Services
Haircut $33
Professional grooming service for hair trimming, cutting, and styling.
Haircut Kids $23
Fun and safe haircuts for children with experienced stylists.
Haircut Long Hair $37
Specialized hair care for trimming, styling, and maintaining long hair.
Haircut & Beard $56
Expert grooming service for haircuts and beard trims and shaping.
Beard $23
Expert trimming, shaping, and grooming for a well-maintained beard.
Full Set $65
Full-service haircut and beard trim with rejuvenating face mask treatment.


November 2024
Su Mo Tu We Th Fr Sa
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
12:00 AM
12:15 AM
12:30 AM
12:45 AM
1:00 PM
1:15 PM
1:30 PM
1:45 PM
2:00 PM
2:15 PM
2:30 PM
2:45 PM
3:00 PM
3:15 PM
3:30 PM
3:45 PM
4:00 PM
4:15 PM
4:30 PM
4:45 PM

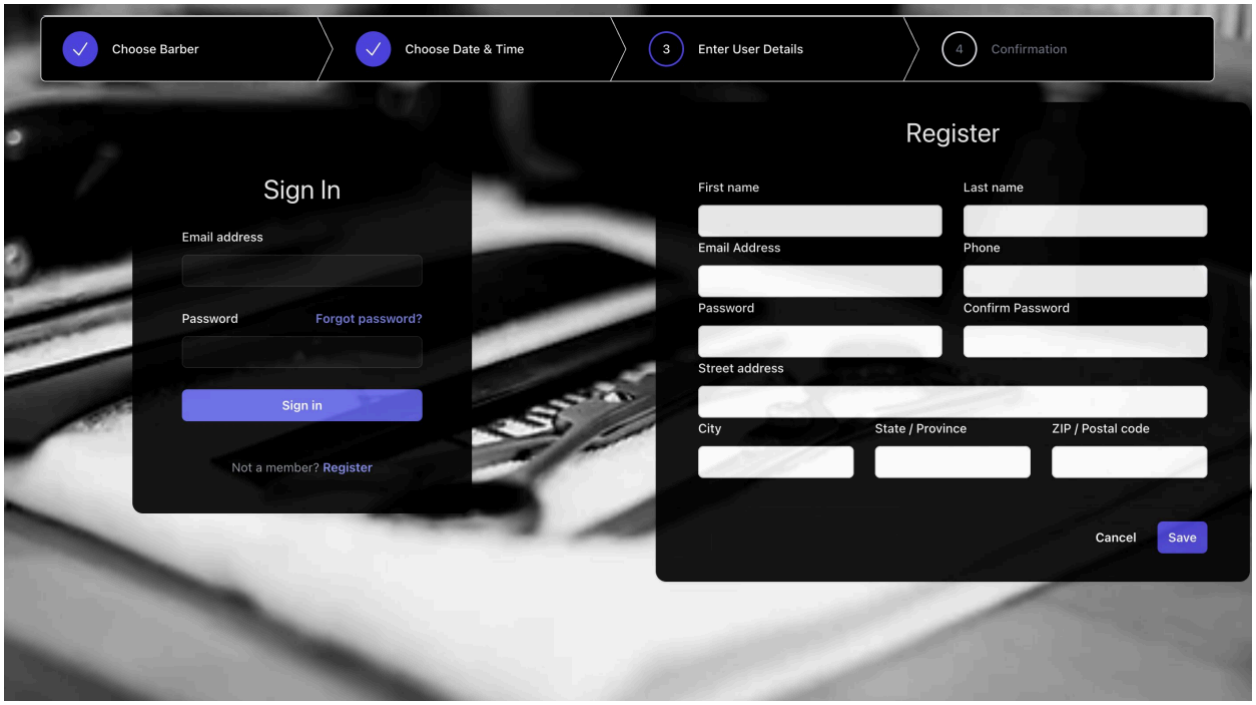

Choose Barber
Choose Date & Time
Enter User Details
4
Confirmation
Appointment Confirmation
Date: 11/14/2024
Time: 1:15 PM
Barber: Moe
Appointment Confirmed

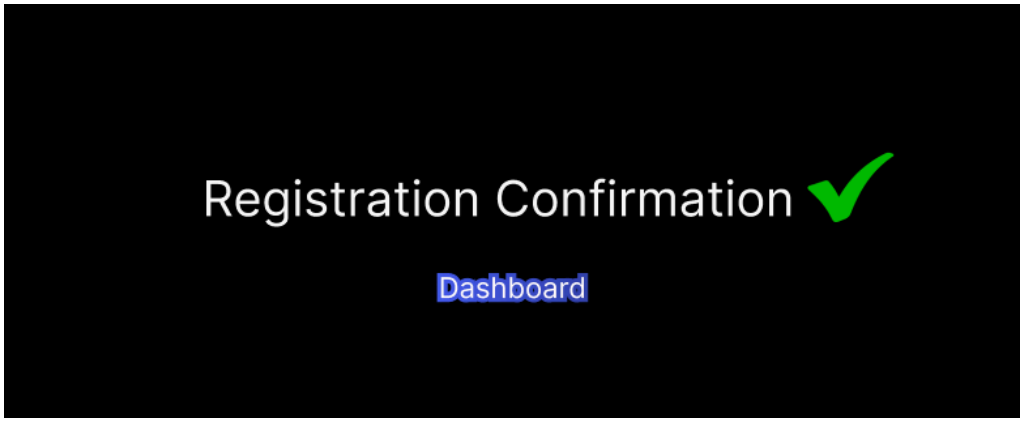
Service Selection Page (Input) : A grid displaying available services that a user can select. Once selected, it brings them to the Barber Selection Page
Barber Selection Page (Input): A list of available barbers with their names and photos that a user can select. Once selected, it brings them to the Date and Time Page
Date and Time Selection Page (Input): A calendar view of dates that a user can pick and then time slots for the barber's availability. Once selected the user enters their information and the system brings them to the Confirmation Page.
Confirmation Page (Output): A confirmation message that displays the date, time and barber name to the user.

2. User Creates New Account

Use Scenario Description: User creates a new account by providing necessary information such as their name, email, phone number, etc.
Step 1: Once the user has selected the service, baber, date and time, the system displays the "Enter User Details" Page.
Step 2: The system displays the form "Register"
Step 3: The user is required to fill in basic information such as their name, email, phone number, password, and address.
Step 3: The system performs a validity checks
Step 4: The user submits the form by clicking the "Save" button
Step 5: User receives confirmation

Navigation Diagram



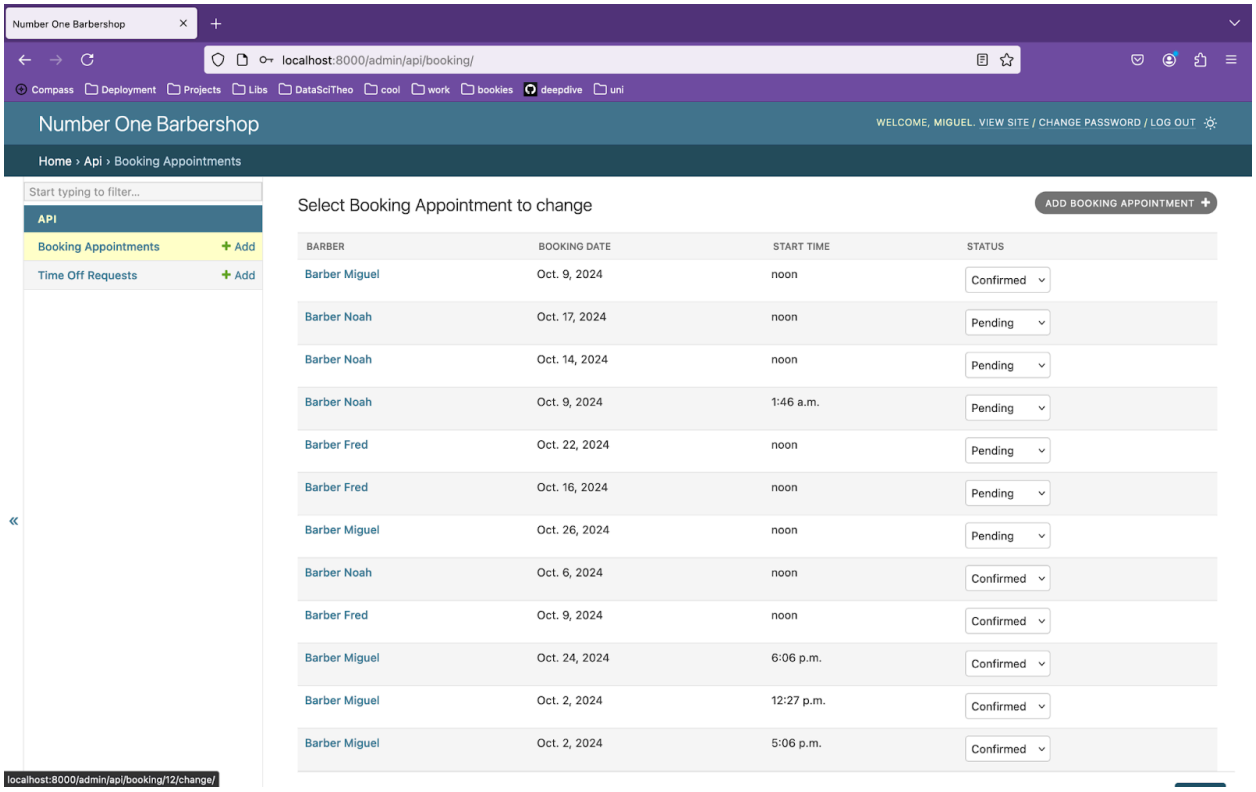


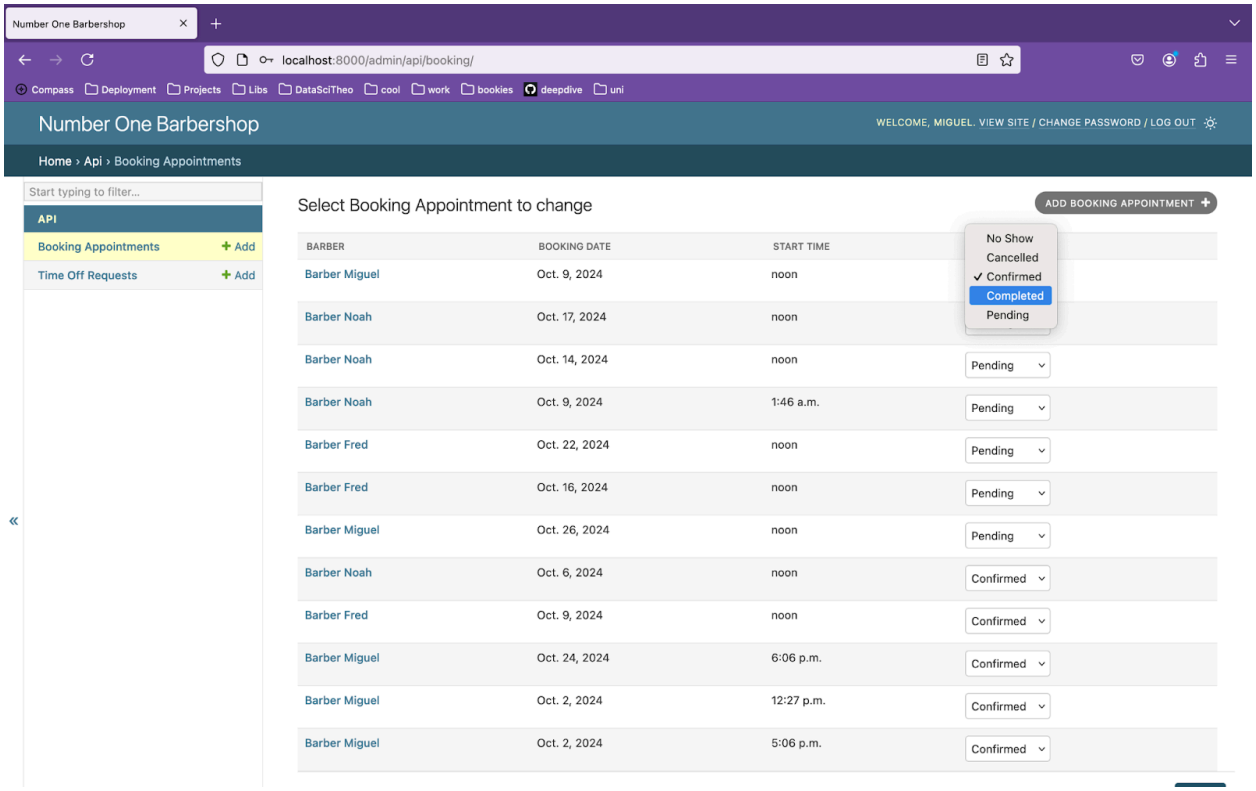
Enter User Details Page (Input): A form where the user enters their details and clicks save that brings them to the Account Creation Confirmation Page.
Account Creation Confirmation Page (Output): A confirmation message that displays that registration is complete and invites you to your dashboard to access details on your appointments.

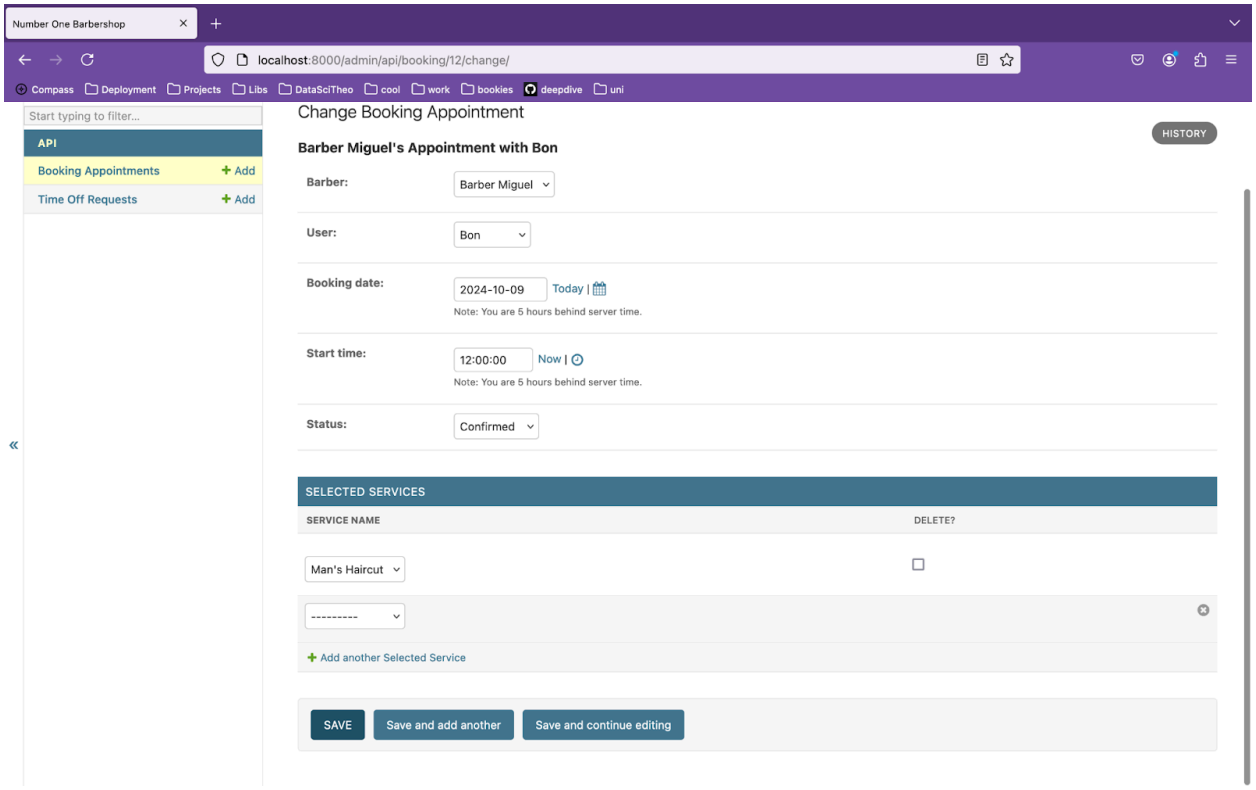
3. Manage Personal Schedule:

Use Scenario Description: Barber accesses their personal schedule to make any changes to their appointments or to their time off request
Step 1: Barber logins in and navigates to Booking Appointments
Step 2: The system displays their upcoming appointments
Step 3: Barber can change the status of his appointments
Step 4: Barber navigates to Time Off Request
Step 5: The system displays their time off requests
Step 6: Barber can modify their request
Step 7: The system validates to ensure there is no conflicts
Step 8: Barber submits changes
Step 9: The system saves the updates

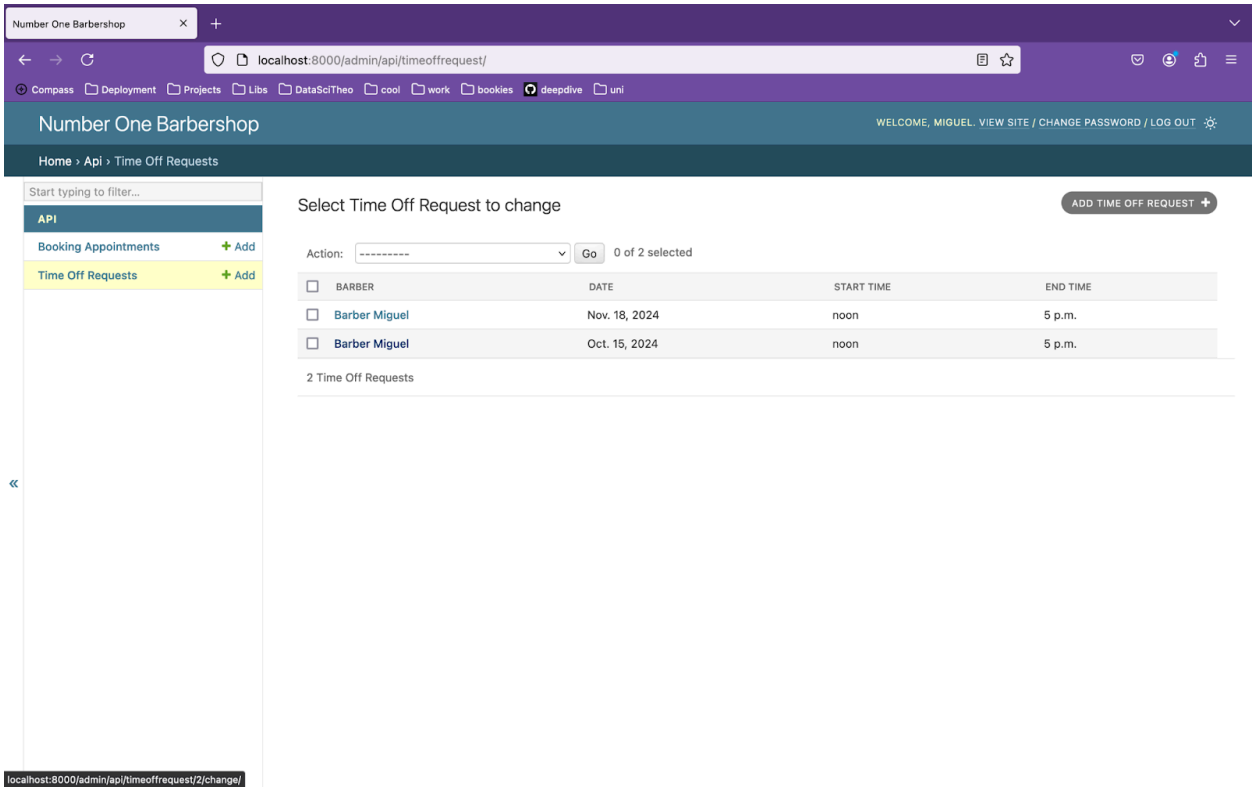
Navigation Diagram

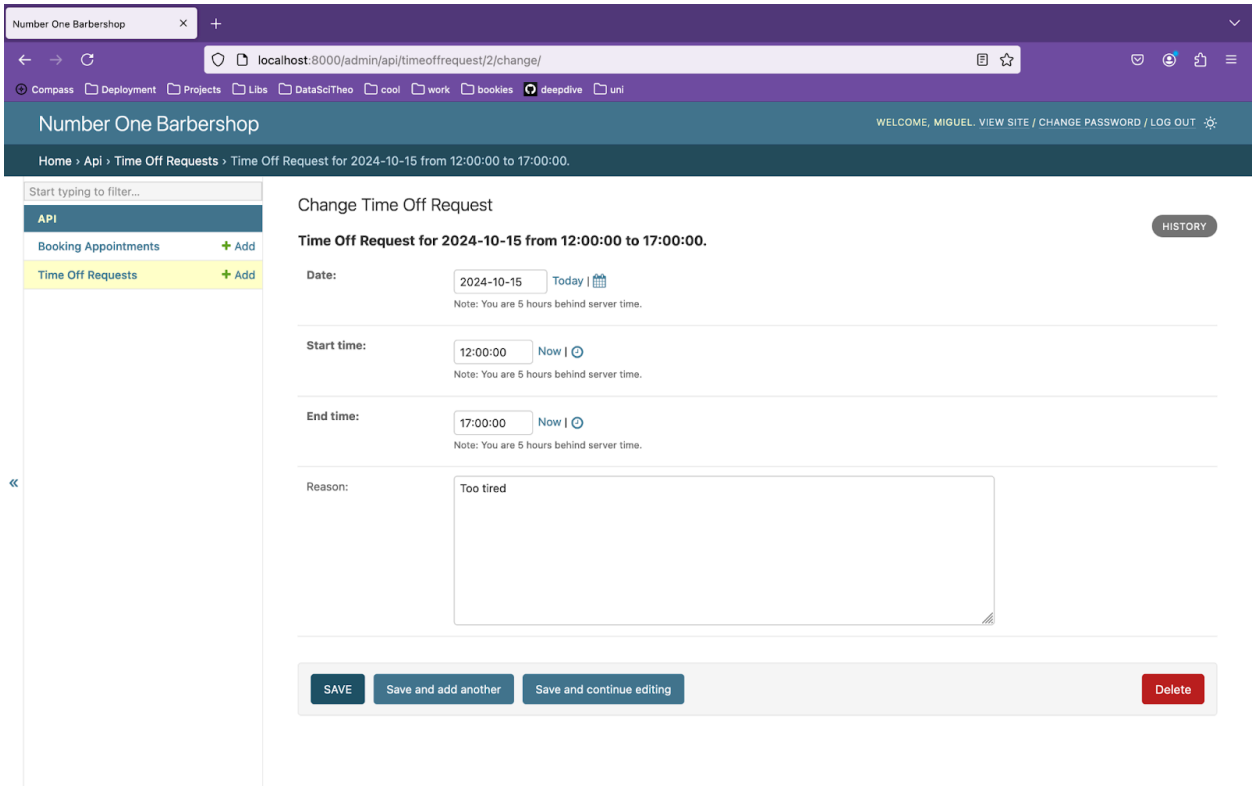





Number One Barbershop
localhost:8000/admin/api/booking/12/change/
Compass Deployment Projects Libs DataSciTheo cool work bookies deepdive uni
Start typing to filter...
API
Booking Appointments + Add
Time Off Requests + Add
Change Booking Appointment
HISTORY
Barber Miguel's Appointment with Bon
Barber: Barber Miguel
User: Bon
Booking date: 2024-10-09 Today |
Note: You are 5 hours behind server time.
Start time: 12:00:00 Now |
Note: You are 5 hours behind server time.
Status: Confirmed
SELECTED SERVICES
SERVICE NAME
DELETE?
Man's Haircut

+ Add another Selected Service
SAVE
Save and add another
Save and continue editing


Number One Barbershop
localhost:8000/admin/api/timeoffrequest/
Compass Deployment Projects Libs DataSciTheo cool work bookies deepdive uni
Number One Barbershop
WELCOME, MIGUEL. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Api › Time Off Requests
Start typing to filter...
API
Booking Appointments + Add
Time Off Requests + Add
Select Time Off Request to change
ADD TIME OFF REQUEST +
Action: --------- Go 0 of 2 selected
BARBER
DATE
START TIME
END TIME
Barber Miguel
Nov. 18, 2024
noon
5 p.m.
Barber Miguel
Oct. 15, 2024
noon
5 p.m.
2 Time Off Requests
localhost:8000/admin/api/timeoffrequest/2/change/



Booking Appointment Screen (Input): The barber can select the drop down menu to change the status of the appointment to either No Show, Canceled, Confirmed, Completed or Pending.

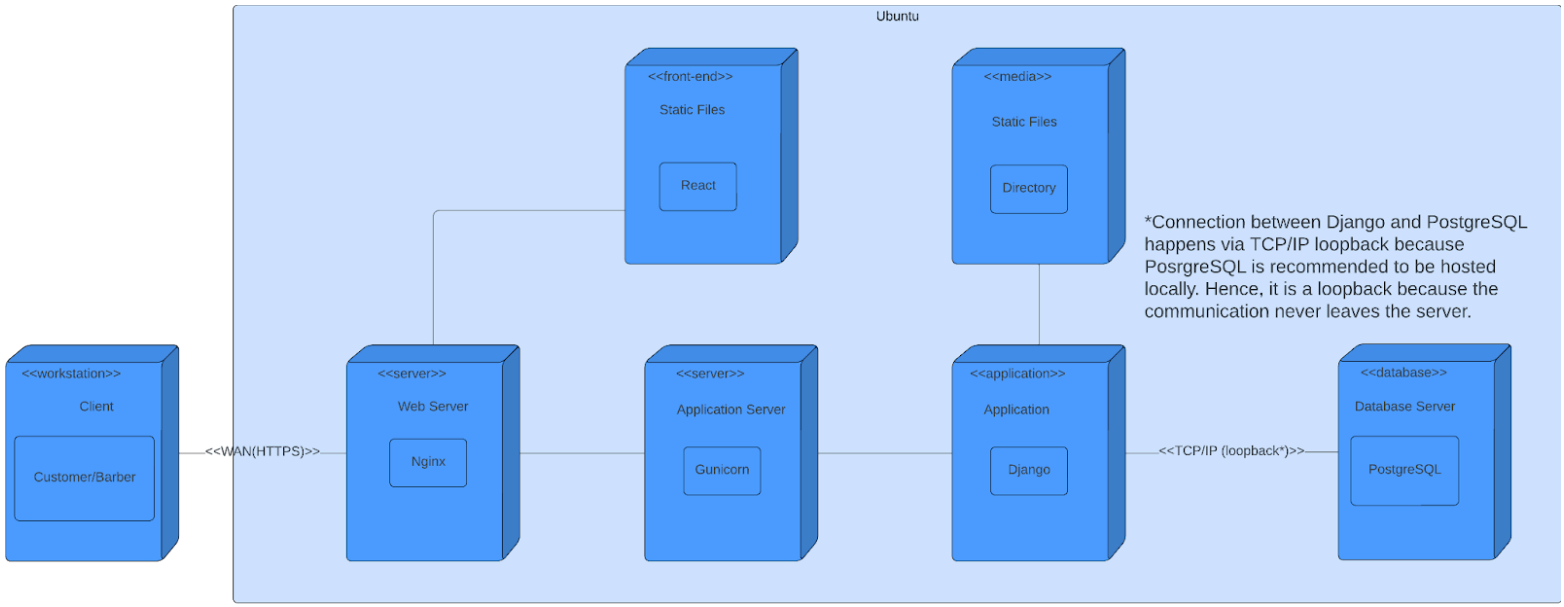
Booking Appointment Screen (Output): Once a change has been made, Barber clicks the “Save” button and gets notified that the change has been made at the bottom of the screen.

Time Off Request Screen (Input): The barber can view their time off request and select which request they want to modify. Once selected, they have the option of selecting a different date through a calendar view, the start time, the end time and reason for their request.

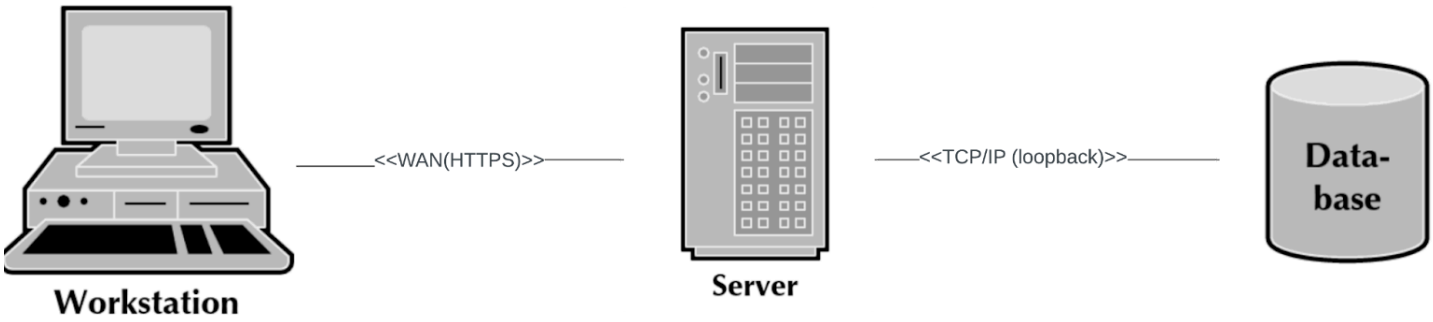
Time Off Request Screen (Output): Once the change has been made, Barber clicks the “Save” button and gets notified that the change has been made at the bottom of the screen.

Deployment Diagram

Deployment of the artifact to a node



Deployment of the artifact to a node with extended notation



Hardware and Software Specifications

Specification	Standard Client	Standard web Server	Standard Application Server	Standard Application	Standard Database Server
Operating system	Windows	Ubuntu 22.04 LTS (Jammy Jellyfish)	Ubuntu 22.04 LTS (Jammy Jellyfish)	Ubuntu 22.04 LTS (Jammy Jellyfish)	Ubuntu 22.04 LTS (Jammy Jellyfish)
Special Software	Microsoft 365 Adobe Reader	Nginx	Gunicorn	Django	PostgreSQL
Hardware	816 GB Memory 500 gig disk drive Intel Core i6 2- 22" Monitors	1-2 vCPUs 1 GB RAM SSD storage 10-20 GB	1-2 vCPUs 1 GB RAM SSD storage 10-20 GB	1-2 vCPUs 1 GB RAM SSD storage 10-20 GB	1-2 vCPUs 1 GB RAM SSD storage 10-20 GB
Network	WiFi	100 Mbps	100 Mbps	100 Mbps	100 Mbps

The Reasoning Behind The Deployment Decision

Operational requirements

Uptime	Ensure 99.99% uptime for the booking system
Maintenance Windows	Allow scheduled downtime for system updates during non-peak hours
Monitoring	Use tools from AWS CloudWatch to monitor server health and performance
Backup and Recovery	The system will perform automated nightly backups of the database and application files. Recovery of the system must be possible within 2 hours in the event of failure

Performance requirements

Response Time	Ensure that booking requests are processed in under 2 seconds.
Concurrent Users	Support up to 1,000 concurrent users during peak hours.
Scalability	Automatically scale resources during high demand periods using cloud infrastructure.

Security requirements

Data Encryption	Use HTTPS for secure client-server communication. Encrypt sensitive data in the database, like customer details and passwords.
Authentication and Authorization	Use Django's built-in authentication with role-based access control
Vulnerability Prevention	Regular security audits, including testing against SQL injection, cross-site scripting (XSS), and other common vulnerabilities
Multi-factor Authentication (MFA)	Admin users must authenticate using MFA, requiring a password and a time-based one-time password (TOTP) sent via a secure application like Google Authenticator

Cultural and Political requirements

Data Residency	Store data in compliance with local regulations for PIPEDA in Canada
Accessibility	Ensure the user interface is accessible for all users, including those with disabilities
Language Support	Provide multilingual options for users in diverse cultural settings
Accessibility Standards Compliance	The system must comply with accessibility standards such as WCAG 2.1 (Web Content Accessibility Guidelines) to ensure that individuals with disabilities can effectively use the platform

1. Installation and Operation

The installation process involves setting up the backend (Django), frontend (React), and database (PostgreSQL) in a production environment, ensuring a seamless integration of all components. Deployment requires configuring a production server using Ubuntu 22.04 LTS, installing essential dependencies, and running database migrations to establish the system's structure. Hardware requirements include 1–2 vCPUs, 1 GB RAM, and SSD storage of 10–20 GB for web, application, and database servers. Operationally, the system emphasizes high uptime through AWS CloudWatch monitoring and automated nightly backups for disaster recovery, with a targeted recovery time of under two hours in case of failures

2. User Testing and Training

User testing ensures that the system meets functional, usability, and performance standards through phases such as unit testing, integration testing, system testing, and user acceptance testing (UAT). End users, including barbers and customers, participate in UAT to simulate real-world tasks and provide feedback. Training focuses on enabling users to effectively operate the system. It includes comprehensive user manuals, interactive sessions, and mock scenarios for both administrative staff and barbers. Feedback from training sessions is used to refine the system and ensure smooth adoption with minimal disruption

Prototype

1. Github links

Frontend - https://github.com/georgi3/NOB_f
Backend - https://github.com/georgi3/bookingSystem

2. Data Management

The system is built on Django & React, Django a Python-based web framework, which serves as the platform for managing APIs and database interactions. Django leverages an Object-Relational Mapping (ORM) tool, enabling straightforward interaction with the database using Python code instead of raw SQL, thereby simplifying and streamlining database operations. For the development environment, the system uses SQLite, a lightweight and embedded database solution. SQLite is well-suited for development due to its ease of use, minimal setup, and seamless integration with Django, making it ideal for rapid prototyping and testing of database-related functionalities. For the production environment PostgreSQL is recommended to be used.

3. Physical Architecture

Deployment recommendations Hardware/Software

The system is recommended to be deployed using AWS Lightsail, which provides a simplified environment for hosting small-scale applications. While Lightsail does not offer pre-configured instances specifically for Django and React applications, it provides general-purpose virtual servers with operating systems like Ubuntu or Amazon Linux. To serve the application effectively, manual configuration of the environment is required.

Virtual Server – AWS Lightsail Instance

A Lightsail instance with the following specifications is recommended: 1-2 vCPUs, 1 GB RAM, and SSD storage 10-20 GB.

Virtual Server OS – Ubuntu 22.04 LTS (Jammy Jellyfish)

Ubuntu is a popular and stable Linux-based operating system that is widely used for server environments. For this deployment, a Long-Term Support (LTS) version of Ubuntu Jammy Jellyfish is recommended to ensure ongoing security updates and compatibility with commonly used software. Ubuntu provides a reliable base for installing and managing components like Nginx, Gunicorn, and Django, offering robust community support and extensive documentation.

Web Server – Nginx

Nginx is a high-performance reverse proxy web server with load balancing capabilities. It should be configured to serve static files, manage HTTPS connections through SSL certificates, forward client

requests to the application server, and handle redirects. For example, HTTP requests should be redirected to HTTPS. Additionally, Nginx can function as a load balancer to distribute incoming traffic across multiple backend application workers, ensuring better performance and reliability for high-traffic scenarios. While load balancing may not be immediately necessary, its capability makes Nginx a flexible and future-proof solution.

Application Server – Gunicorn

Green Unicorn (Gunicorn) is a Web Server Gateway Interface (WSGI) application server designed to serve Python applications such as Django. It acts as an intermediary between the Nginx web server and the Django application, processing HTTP requests and routing them to Django for backend processing. Gunicorn ensures efficient communication with Nginx and provides robust support for concurrent request handling.

Database – PostgreSQL

PostgreSQL is a robust relational database that is recommended for its scalability, reliability, and compatibility with Django. For this setup, the database is recommended to be hosted locally on the same Lightsail instance as the application because it is a small-scale application and the chosen virtual server has a large SSD of 10-20GB. This configuration simplifies network setup, reduces latency, and is cost-effective for a small-scale application. Access to the database should be restricted to the application via localhost, and strong authentication credentials should be enforced. Regular back-ups of the database should be implemented to protect data and ensure system reliability.

Static Files & Media – Local Server

It is recommended to handle static files and media directly on the server for the current deployment, leveraging Nginx to serve these assets efficiently. This approach simplifies the setup and is cost-effective for a small-scale application. However, in the case of scaling, it is advisable to offload static and media files to a storage service like AWS S3. Using S3 enhances scalability, reduces server load, and improves performance by enabling a content delivery network (CDN) like AWS CloudFront to deliver assets globally with low latency.

Version Control – Git

Git is an essential tool for efficient version control and managing code deployments on the server. It allows for streamlined updates, rollback capabilities in the case of any incidents. On the server, Git can be used to pull the latest changes directly from a remote repository, ensuring that the deployed application stays up-to-date with the latest features and bug fixes. By integrating Git into the deployment workflow, the risk of manual errors is minimized, and managing the codebase becomes more efficient. For additional security, SSH keys should be used to authenticate with the Git repository.

Domain Name System (DNS) Provider – Namecheap

Namecheap is a DNS provider that facilitates the purchase and registration of custom domain names. The DNS records of the domain have to be configured to point to the public IP address of the Lightsail instance to enable proper routing of traffic. Alternative providers, such as GoDaddy or Bluehost, can also be used. For integrated AWS solutions, **Amazon Route 53** is a suitable alternative for managing DNS.

Secure Sockets Layer (SSL) Certificate – Let's Encrypt

Let's Encrypt is a widely used certificate authority that provides free Transport Layer Security (TLS) certificates, enabling secure HTTPS connections. SSL certificates issued by Let's Encrypt must be installed and configured on the web server (Nginx). For automated provisioning and renewal of SSL certificates, AWS Certificate Manager can be used as an alternative.

Server Security – Uncomplicated Firewall (UFW) & Secure Shell (SSH)

SSH and UFW softwares have to be configured on the server to ensure its security. The UFW should be configured to allow only necessary traffic, such as HTTP (port 80), HTTPS (port 443), and SSH (port 22). All other ports should be blocked by default. Additionally, secure SSH configuration is essential: password-based root login should be disabled, and access should be restricted to authorized users via public/private key authentication. This ensures that only users with the correct private key can log in to the server.

Deployment Conclusion

The deployment recommendation for the barbershop scheduling application provides a cost-effective, secure, and efficient solution tailored for a small-scale system. By leveraging AWS Lightsail, the setup ensures simplified management and predictable costs. The integration of Ubuntu as the server OS, PostgreSQL for database management, and the combination of Nginx and Gunicorn for serving the application guarantees robust performance and scalability for the current needs. Security measures, such as configuring UFW and SSH, along with the use of Git for version control, further enhance reliability and safeguard the system. With these carefully considered recommendations, the application is well-positioned for reliable operation and future growth.

4. Construction of the System

The software tools and frameworks used in constructing the barbershop scheduling system were carefully selected to ensure modularity, maintainability, and functionality, leveraging well-established and reliable technologies.

Choice of Programming Languages – Python & TypeScript

Python was chosen for its simplicity, readability, and extensive ecosystem, making it ideal for backend development with Django. Its robust support for libraries, frameworks, and community-driven resources enables rapid development and scalability. Python's integration with tools like Pandas and Plotly enhances data processing and visualization capabilities, aligning well with the system's needs.

TypeScript, a superset of JavaScript, was selected for frontend development with React due to its strong typing capabilities, which help reduce runtime errors and improve code maintainability. Its compatibility with modern JavaScript ensures a smooth development experience while providing enhanced tooling and editor support. TypeScript's ability to manage complex UI interactions and state management makes it a suitable choice for building dynamic and interactive user interfaces.

Backend Framework – Django

Django is a robust backend framework used for API development and business logic. Its modular architecture and built-in features, such as ORM, authentication, and an admin panel, simplify backend operations and accelerate the development process. Django's capabilities make it particularly well-suited for small-scale applications like a barbershop booking system, providing efficiency and reliability.

Core Backend Libraries:

- **Django REST Framework (DRF):**
  Chosen for its ability to simplify the development of robust and scalable RESTful APIs, DRF integrates seamlessly with Django and provides extensive tools for building API endpoints. It ensures efficient and secure communication between the frontend and backend, making it ideal for handling the application's interactive and data-driven features.
- **Pandas & NumPy:**
  Selected for their powerful data processing and analysis capabilities, Pandas and NumPy are used to manage and manipulate data efficiently. Their versatility and performance make them particularly suitable for preparing complex datasets for visualizations.
- **Plotly:**
  Chosen for its ability to create dynamic and interactive data visualizations. Its integration with Django enables the generation of insightful visual representations directly within the admin panel, enhancing data accessibility and decision-making for system administrators.

Rest of the backend libraries used can be found here.

Database – SQLite

SQLite is a lightweight, embedded database chosen for the development environment due to its simplicity and ease of use. It does not require a separate server setup, making it an ideal choice for rapid prototyping and testing. As Django's default database engine, SQLite integrates seamlessly with the framework, allowing it to focus on building features without additional configuration complexity before transitioning to a production-grade database.

Primary Frontend Library – React

React is a JavaScript/TypeScript frontend library chosen for its ability to build dynamic and interactive user interfaces efficiently. Its component-based architecture promotes reusability, modularity, and maintainability, making it ideal for scalable and structured frontend development. React's virtual Document Object Model (DOM) enhances performance by enabling fast and efficient updates, which is

particularly beneficial for creating responsive and interactive Single Page Applications (SPAs). These features make React a powerful and flexible choice for modern web development.

Core Frontend Dependencies:

- **Tailwind CSS:**
  Chosen for its utility-first approach, offering a highly customizable and responsive styling framework. It streamlines the design process by providing pre-defined CSS classes, eliminating the need for custom CSS in many cases, and enabling rapid prototyping of modern, responsive user interfaces.
- **React Day Picker:**
  React Day Picker is a versatile component for building date pickers, calendars, and date inputs in React applications. It was selected for its ease of integration, flexibility, and ability to simplify the appointment booking flow by providing an intuitive and interactive date selection experience.
- **React Router DOM:**
  Chosen to implement declarative routing in the application. Its ability to manage navigation, handle dynamic routes, and support SPAs makes it essential for building seamless and user-friendly navigation within the application.

Rest of the frontend libraries used can be found here.

Integrated Development Environments (IDEs)

- **WebStorm:**
  Chosen for React development due to its advanced support for JavaScript, TypeScript, and CSS. Its intelligent code completion, powerful debugging tools, and seamless integration with modern frameworks streamline the development of dynamic and interactive frontend applications.
- **PyCharm:**
  Selected for Django development because of its comprehensive support for Python and Django-specific features. With tools like automatic code inspections, debugging, and built-in database management, PyCharm enhances productivity and simplifies backend development.

Package Managers & Virtual Environments

- **Conda:**
  Chosen for managing Python dependencies due to its robust environment management capabilities and ability to ensure compatibility between packages. It simplifies the installation and maintenance of backend tools.
- **Node Package Manager (NPM):**
  Selected for managing React dependencies and JavaScript/TypeScript packages for the frontend. Its extensive ecosystem and straightforward package management streamline the installation, versioning, and updating of libraries, ensuring a smooth and efficient frontend development process.

Version Control Tool

- **Git**:
  Chosen for its reliability and efficiency in version control and code management. It enables seamless collaboration by allowing multiple contributors to work on the same codebase without conflicts. Git's powerful features, such as branching, merging, and rollback capabilities, streamline the development process.

Design Patterns

- **MVC (Model-View-Controller):**
  The MVC pattern was chosen for its ability to separate concerns, which is implemented in Django. This design ensures that the application is scalable, maintainable, and organized by dividing the logic into three interconnected components: Model (data), View (UI), and Controller (business logic).
- **Component-Based Architecture:**
  React's component-based architecture was selected to create modular, reusable, and maintainable UI components. This pattern enables efficient management of UI complexity by breaking it into smaller, isolated parts that can be developed and tested independently.
- **React Document Object Model (DOM):**
  React DOM represents a pattern within React's ecosystem. It focuses on rendering React components into the DOM efficiently. This pattern separates the UI logic from the underlying rendering process, allowing for platform-specific optimizations and flexible UI updates in SPAs.

Programming Paradigms

- **Object-Oriented Programming (OOP)** – Was primarily used in the Django backend to design to ensure modularity, scalability, and maintainability of the backend logic. The following key components of the application were developed using OOP principles:
  - **Models:** Encapsulation of database entities into Python classes allowed for clear and reusable representations of data.
  - **Admin Customization:** Object-oriented techniques made it easy to extend and customize Django's admin interface for enhanced usability.
  - **Serialization:** The use of serializers in Django REST Framework (DRF) provided a structured and reusable way to convert complex data into JSON for API responses.
- **Functional Programming (FP)** – The functional approach was predominantly employed for:
  - **Declaring Endpoints:** Django views and API endpoints often leveraged function-based views for simplicity and readability in cases where advanced state management wasn't required.
  - **React Components:** Functional components in React were chosen over class-based components due to their simplicity and integration with React Hooks. Hooks allow for managing state and side effects directly within functional components, making the codebase cleaner and easier to understand. This paradigm also aligns well with React's declarative style, emphasizing how the UI should look at any given time rather than detailing how to achieve that state.

Other Tools

- **Developer Tools (DevTools):**
  DevTools were chosen for their indispensable role in debugging and optimizing both frontend and backend code during development. Browser-based DevTools, such as those in Chrome and Firefox, provide real-time inspection and debugging of UI components, network activity, and performance metrics, while backend tools allow for efficient logging and error tracking.
- **Postman:**
  Postman was selected for its efficiency in testing and validating API endpoints. Its user-friendly interface and powerful features, such as automated testing and environment management, make it an essential tool for ensuring the reliability, correctness, and performance of RESTful APIs.

Construction Conclusion

By leveraging a carefully selected set of tools, frameworks, design patterns, and programming paradigms, the construction of the system was both efficient and effective. Each component, from IDEs to design patterns, was chosen to streamline the development process, enhance maintainability, and ensure the application's long-term reliability and scalability. This cohesive approach has resulted in a robust and well-structured solution tailored to meet the application's needs.

Appendices

Team management documents

1. Team Meeting Agendas and Minutes:

Meeting # 1

Date:	September 11th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Rama
Scribe:	Lynn
Attendance:	Rama, Lynn, Thomas, Ryan, Glory, Nader, George

Agenda (insert as many rows as needed)

I. Select the project from 481
II. Fill out the MileStone 1 project proposal

Required Materials and files:

- Milestone 1 document, Final Report of 481 and Project Proposal part 1 document

Minutes

A. Introduction

- o Leader's opening remarks
- o Announcements

B. Issues/Items Discussed

- Discussed which project from 481 to continue to work on
- Identified strengths in knowledge area for each team member
- Filled out project proposal Part 1

C. Decisions

- Selected Number One Barbershop for the project

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch UML Fundamental videos and YuJa videos for next class.

F. Date, Time, and Location of the Next Meeting

- September 14th at 7:00 pm, Zoom

Meeting # 2

Date:	September 14th 2024
Start time:	7:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Lynn
Scribe:	Thomas
Attendance:	Rama, Lynn, Thomas, Ryan, Glory, Nader, George

Agenda (insert as many rows as needed)

I. Select a subsystem from 481
II. Discuss potential use cases (7 in total)

Required Materials and files:

- *Milestone 1 document and LucidChart*

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Selected a subsystem from Number One Barbershop
- Each member came up with one use case

C. Decisions

- Finalized the subsystem and 7 use cases

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch UML Fundamental videos and YuJa videos for next class.
- Nader: Submit milestone 1

F. Date, Time, and Location of the Next Meeting

- September 22nd at 6:00 pm, Zoom

Meeting # 3

Date:	September 22nd 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Thomas
Scribe:	Ryan
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Modifications on Milestone 1
II. Go over activity diagram, use case descriptions and use case

Required Materials and files:

- *LucidChart and Use Case Description Document*

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discuss the modifications that need to make to milestone 1
- Discussed activity diagram, use case and use case description

C. Decisions

- Made changes to milestone 1 to make it better

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Fix and finalize the activity diagram, use case, and use case descriptions. Watch UML Fundamental videos and YuJa videos for next class.
- Nader: Submit milestone 2

F. Date, Time, and Location of the Next Meeting

- September 29th at 6:00 pm, Zoom

Meeting # 4

Date:	September 29th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Ryan
Scribe:	Glory
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Structural Modelling - Class diagram
II. Prototype 1

Required Materials and files:

- *LucidChart, GitHub.com, WebStorm and PyCharm*

Minutes

A. Introduction

- o Leader's opening remarks
- o Announcements

B. Issues/Items Discussed

- Discussed the TMV model for class diagram
- Discussed what platform to do the prototype

C. Decisions

- Decided to learn more about TMV before modelling class diagrams and put George in charge of the prototype
- Meeting with prof on October 8th

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Understand how TMV works to work on the class diagram in the next meeting. Prepare any questions for the prof for the meeting next week.
- George: Start working on the prototype

F. Date, Time, and Location of the Next Meeting

- October 8th at 6:30 pm, Zoom

Meeting # 5

Date:	October 8th 2024
Start time:	6:30pm
End time:	9:00pm
Location:	Zoom
Leader:	Glory
Scribe:	Nader
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Structural Modelling - Class diagram
II. Prototype 1

Required Materials and files:

- LucidChart, GitHub.com, WebStorm and PyCharm

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed what we learnt about TMV model
- Discussed how we're going to do the class diagram

C. Decisions

- Decided to start making different classes for the class diagram

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Add attributes and operations to the class diagram.

- George: Continue working on the prototype

F. Date, Time, and Location of the Next Meeting

- October 20th at 6:00 pm, Zoom

Meeting # 6

Date:	October 20th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Nader
Scribe:	George
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Structural Modelling - Class diagram
II. CRC Cards

Required Materials and files:

- LucidChart and CRC Cards documents

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed on how to improve the class diagram
- Discussed CRC diagrams

C. Decisions

- Team members picked which CRC card to work on and based on CRC cards, Nader will make adjustments to our class diagram

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch UML Fundamental videos and YuJa videos on Behavioral Modelling.
- Rama: CRC card
- Lynn: CRC card
- Thomas: CRC card
- Ryan: CRC card
- Glory: CRC card
- Nader: Fix class diagram after CRC cards are complete
- George: Continue working on the prototype

F. Date, Time, and Location of the Next Meeting

- October 27th at 6:00 pm, Zoom

Meeting # 7

Date:	October 27th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	George
Scribe:	Rama
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Structural Modelling - Class diagram
II. CRC Cards
III. Prototype 1

Required Materials and files:

- LucidChart, CRC Cards documents, Milestone 3 document, GitHub.com, WebStorm and PyCharm

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed CRC cards and made adjustments to each
- Discussed on improving class diagram
- Discussed the prototype and how far it came

C. Decisions

- Changed and finalized the class diagram
- Finalized the CRC cards

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Submit peer review evaluations.
- Rama: Submit Milestone 3
- George: Submit prototype 1

F. Date, Time, and Location of the Next Meeting

- October 30th at 6:00 pm, Zoom

Meeting # 8

Date:	October 30th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom

Leader:	Rama
Scribe:	Lynn
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Sequence Diagram
II. Behavioural State Model

Required Materials and files:

- *Milestone 4, LucidChart*

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed sequence diagram
- Discussed behavioral state diagram

C. Decisions

- Decided to split up into 2 groups to do behavioral state diagram

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Finish up sequence diagram and put methods into the class diagram
- Sub teams: Each subteam does one behavioral state diagram
- George: Continue to work on the prototype 2

F. Date, Time, and Location of the Next Meeting

- November 3rd at 6:00 pm, Zoom

Meeting # 10

Date:	November 3rd 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Lynn
Scribe:	Thomas
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Sequence Diagram
II. Behavioural State Model
III. Team Feedback Conversation

Required Materials and files:

- *Milestone 4, LucidChart*

Minutes

A. Introduction

- o Leader's opening remarks
- o Announcements

B. Issues/Items Discussed

- Discussed sequence diagram
- Discussed behavioral state diagram
- Discussed feedback

C. Decisions

- Finalized both sequence diagram and behavioral state diagram
- Meeting with prof on November 5th

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch UML Fundamental videos and YuJa videos for next class. Finish the team feedback reflection. Prepare any questions for the prof for the meeting next week.

- George: Continue to work on the prototype 2

- Ryan: Submit Milestone 4

F. Date, Time, and Location of the Next Meeting

- November 13th at 6:00 pm, Zoom

Meeting # 10

Date:	November 13th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Thomas
Scribe:	Ryan
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Class and Methods Design
II. HCI Design
III. Prototype 2

Required Materials and files:

- *LucidChart, Figma, Milestone 5, Milestone 6, GitHub.com, WebStorm and PyCharm*

Minutes

A. Introduction

o Leader’s opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed the method contract and method specification
- Discussed how to use figma
- Discussed HCI Design and what methods are already implemented in our prototype

C. Decisions

- Divided method contract and method specifics amongst group members

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch UML Fundamental videos and YuJa videos for next class. Finish the method contract and method specification. Finish put HCI Design.
- George: Continue to work on the prototype 2

F. Date, Time, and Location of the Next Meeting

- November 17th at 6:00 pm, Zoom

Meeting # 11

Date:	November 17th 2024
Start time:	6:00pm
End time:	9:00pm
Location:	Zoom
Leader:	Ryan
Scribe:	Glory
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Class and Methods Design

II. HCI Design
III. Prototype 2

Required Materials and files:

- *LucidChart, Figma, Milestone 5, Milestone 6, GitHub.com, WebStorm and PyCharm*

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed everything done for Milestone 5 (method contract and method specifications)

- Discussed HCI design and the 3 methods used

C. Decisions

- Decided to come up with questions about the final report and presentation to ask the prof on November 19th

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch and YuJa videos on deployment diagrams for the last milestone. Come up with questions to ask the teacher for the last meeting

- George: Submit prototype 1

- Glory: Submit Milestone 5 and Milestone 6

F. Date, Time, and Location of the Next Meeting

- November 19th at 6:00 pm, Zoom

Meeting # 12

Date:	November 19th 2024
Start time:	6:00pm

End time:	9:00pm
Location:	Zoom
Leader:	Glory
Scribe:	Nader
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Final Report
II. Final Presentation
III. Deployment Diagram

Required Materials and files:

- *Final Report and Milestone 7*

Minutes

A. Introduction

o Leader's opening remarks

o Announcements

B. Issues/Items Discussed

- Discussed what needs to be done for the file report
- Discussed the deployment diagram briefly
- Discussed who is presenting for the final presentation

C. Decisions

- Decided who is going to complete what tasks for the final report
- Decided who is going to be presenting in front of the class

D. Adjournment of the meeting

The meeting adjourned at *9:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Watch and YuJa videos on deployment diagrams for the last milestone. Complete the task that was assigned to you for the final report.

- Ryan and Thomas: Work on deployment diagram

F. Date, Time, and Location of the Next Meeting

- November 23rd at 3:00 pm, Zoom

Meeting # 13

Date:	November 23rd 2024
Start time:	3:00pm
End time:	6:00pm
Location:	Zoom
Leader:	Nader
Scribe:	George
Attendance:	Rama, Lynn, Thomas, Ryan Glory, Nader, George

Agenda (insert as many rows as needed)

I. Final Report
II. Final Presentation
III. Deployment Diagram

Required Materials and files:

- *Final Report, Milestone 7, WebStorm and PyCharm*

Minutes

A. Introduction

o Leader's opening remarks
o Announcements

B. Issues/Items Discussed

- Discussed final report and final presentation
- Discussed deployment diagram

C. Decisions

- Lynn, Glory and Thomas work on report, finalize and submit
- Rama, Ryan, Nader and George work on presentation
- George works on final prototype

D. Adjournment of the meeting

The meeting adjourned at *6:00pm*, with the agreement of all members present.

E. Tasks Assigned

- All team members: Complete peer review
- Thomas: Submit Milestone 7
- George: Submit final prototype
- Lynn, Glory and Thomas: Finalize the report and submit
- Rama: Submit final presentation
- Rama, Nader, Ryan and George: Prepare for final presentation

2. Project Management Documents:

To effectively manage our team of 7, we decided to assign roles and responsibilities based on our strengths and areas of expertise which ensured that tasks were evenly distributed.

Weekly meetings through Zoom were how we collaborated together to check in, plan our tasks, and see how we are progressing. For each meeting, we rotated the role of the team leader, who managed the discussions, maintained the schedules and guided project tasks. We also had a scribe for each of the meetings who was responsible for organizing the agenda, taking notes, clarifying details and documenting everything in the Meeting Minutes Template. We started off each meeting with updates from the team and then moved on to planning our tasks and what needs to be accomplished. Additionally our team emphasized on setting and reviewing our deadlines, and making sure we were on track with our goals.

During our meetings, we made sure to gather feedback from everyone to get insights and figure out where we could improve. Midway through the semester, we filled out a Team Feedback form and then discussed it together in a separate meeting to see how we could work better as a team.

This approach helped us to stay organized, accountable and aligned throughout our project. By checking in regularly through meetings, sharing feedback and making adjustments when needed, we were able to stay focused and work effectively as a team to meet our goals.

3. Software used to support work:

Project Management: Google meets for weekly team meetings and Google Calendar to schedule and keep track of meetings
Modeling: To create all of our diagrams - use case, activity diagram, class diagram, sequence diagram, etc.
Programming: Webstorm for frontend development and PyCharm for back-end development.
Implementation: GitHub to collaborate on code.