Software Engineering

Final Project

Applying Software Engineering

Practices in Real-life Applications

MySpace Reservation System

Prepared by:

Name	ID
Youssef Samuel Nachaat Labib	6978
Youssef Amr Ismail Othman	6913
Arsany Mousa Fathy Rezk	6927

1. Table of Contents

2. Introduction

a. Purpose:

The Workspace Reservation System solves the common problem of complicated and time-consuming table reservations in workspaces. Currently, people struggle to find a table quickly and easily, leading to frustration. Workspace providers also face challenges in managing reservations across multiple branches efficiently.

The purpose of the proposed software is to design and implement a Workspace Reservation System that facilitates table reservations in workspaces in different locations. This system aims to enhance the overall user experience by providing a user-friendly platform for clients to reserve tables efficiently. This not only saves time for users but also helps workspace providers better organize reservations, leading to smoother operations overall.

b. Definitions:

- **Client/User:** Individuals who wish to reserve tables in workspaces. They must have user accounts for authentication.
- **Admin:** Administrators responsible for managing the system, confirming new branches, and updating table information.
- **Workspace:** Physical locations with tables available for reservation.
- **Tables:** Different table categories, number of seats and reservations.

3. Requirements

a. User Requirements

Functional Requirements

- **FR1:** Customers must be able to create accounts.
- **FR2:** Users must be able to log in securely.
- **FR3:** Customers must be able to search a table based on their preferences.
- **FR4:** Customers must be able to view available tables.
- **FR5:** Customers must be able to reserve an available table and confirm his reservation.

- **FR6:** Customers must receive confirmation of their reservation.
- **FR7:** Customers must be able to view their confirmed reservations.
- **FR8:** Admin must be able to view all reservations.
- **FR9:** Admin must be able to view status of tables on a specific day.
- **FR10:** Admin must be able to view system information.
- **FR11:** Admin must be able to get the daily payments of the branches.
- **FR12:** Admin must be able to add new tables.
- **FR13:** Admin must be able to update table price per hour.

Non-functional Requirements

- **NFR1:** Response time for system actions should be less than 2 seconds with automated user input speed.
- **NFR2:** Passwords of the users must be stored securely in the database.
- **NFR3:** The user interface should be intuitive and easy to navigate.
- **NFR4:** The system should provide alerts in case of any errors.

b. System Requirements

Functional Requirements:

- **FR1: Account Creation**

SR1.1: The system shall provide a user registration interface with fields for first name, last name, email, password, profession, and mobile number.
SR1.2: Emails must be unique, and the system shall perform real-time validation during registration.
SR1.3: Passwords must be re-written in a confirmation password field to ensure correctness.

- **FR2: Secure User Login**

SR2.1: The system shall match the user email and password with existing customer credentials in the database.
SR2.2: User's passwords must be invisible during the filling process to be unnoticeable.

- **FR3: Table Search based on preferences**

SR3.1: The system shall provide a search functionality allowing customers to filter tables.

SR3.2: Various criteria like availability (in terms of timing), capacity, branch location, category, and price shall be used for searching.

- **FR4: View Available Tables**

SR4.1: The system shall display a real-time list of available tables based on the customer's search criteria.
SR4.2: The table information should include details like capacity, location, category, and price.

- **FR5: Table Reservation and Confirmation**

SR5.1: Customers shall be able to select an available table and initiate a reservation transaction.
SR5.2: The system shall issue a detailed payment report to the customer before reservation confirmation.
SR5.3: The system shall confirm the reservation only if the customer paid for the reservation.
SR5.4: Reservation transactions must be atomic and support rollback in case of errors.

- **FR6: Confirmation Notification**

SR6.1: The system shall issue a notification after successful reservation.
SR6.2: New reservation shall be added to customer's reservations.

- **FR7: View Confirmed Reservations**

SR7.1: Customers shall have a dedicated section in their account to view and manage their confirmed reservations.

- **FR8: Admin View of All Reservations**

SR8.1: The admin interface shall display a list of all reservations made by customers.
SR8.2: Admins should be able to filter reservations based on various criteria, such as date and time.

- **FR9: Admin View of Table Status**

SR9.1: Admins shall have access to a dashboard displaying the current status of all tables on a specific day.
SR9.2: The status should include information on reserved, and available tables.

- **FR10: Admin Access to System Information**

SR10.1: The system shall provide an interface for admins to view all table reservations, including table details, customer names, and reservation status.
SR10.2: The system shall provide an interface for admins to access information on all customer reservations, displaying reservation details, customer information, and reservation status.
SR10.3: The system shall provide an interface for admins to view a list of all registered customers, including customer IDs, names, and contact details.
SR10.4: The system shall provide an interface for admins to access information on all tables, displaying table IDs, capacities, location, category, price, and current availability status.
SR10.5: The system shall provide an interface for admins to view a list of all branches, including branch IDs, and locations.

- **FR11: Daily Payments Report**

SR11.1: Admins shall be able to generate a report showing daily payments for each branch.
SR11.2: The report should include details such as branch name, and total revenue.

- **FR12: Add New Tables**

SR12.1: Admins shall have the capability to add new tables to the system, specifying details such as capacities, location, category, price.

- **FR13: Update Table Price per Hour**

SR13.1: Admins shall be able to update the price per hour for each table.
SR13.2: Price updates should be reflected in real-time and applied to future reservations.

Non-functional Requirements:

- **NFR1: Response Time**

SNR1.1: The system shall respond to user actions within 2 seconds under normal load conditions.
SNR1.2: Automated user input simulations should be conducted to validate response times.

- **NFR2: Password Security**

SNR2.1: Passwords shall be hashed using a strong cryptographic algorithm before storage.

- **NFR3: Intuitive User Interface**

SNR3.1: The user interface shall follow established design principles for usability and accessibility.
SNR3.2: User navigation paths shall be intuitive, with clear instructions and error messages.

- **NFR4: Error Handling and Alerts**

SNR4.1: The system shall issue errors messages after checking validity of user inputs.
SNR4.2: Users and admins shall receive real-time alerts in case of invalid input or an empty field that is required.

4. Software Process

a. Suggested Type of Software Process

For the development of the table reservation system described in the requirements, an Extreme Programming (XP) approach is recommended. XP is an agile methodology known for its emphasis on collaboration, customer involvement, and iterative development practices.

Reasoning:

- Test-Driven Development (TDD): XP places a strong emphasis on TDD, where developers write tests before writing the actual code. This ensures that the codebase is thoroughly tested and meets the specified requirements, supporting the reliability of a reservation system.
- Pair Programming: XP promotes pair programming, encouraging developers to work in pairs. This not only ensures the quality of code through constant review but also facilitates knowledge sharing and minimizes single points of failure.
- Customer Involvement: Similar to Agile, XP involves the customer throughout the development process. Regular customer interactions and continuous feedback loops are integral to XP, aligning well with the evolving requirements of a reservation system.
- Small Releases: XP promotes small, frequent releases. This aligns with the incremental development needed for a reservation system, allowing for the regular delivery of new features and improvements.

b. Division of Phases (In case of normal professional working process)

1. Exploration Phase:
- Project Kickoff: Establish project goals and introduce the development team to the project.
- Initial Customer Meetings: Engage with stakeholders to understand initial requirements and priorities.

2. Planning Phase:

- User Stories: Break down features into user stories that represent customer requirements.
- Release Planning: Plan for frequent releases based on the priority of user stories.

3. Execution Phase (Iterations):

- Test-Driven Development (TDD): Write tests before implementing features to ensure code reliability.
- Pair Programming: Developers work in pairs, with one writing code and the other reviewing and providing feedback.
- Continuous Integration: Integrate code changes frequently to maintain a stable codebase.
- Small Releases: Deploy small increments of the system regularly, ensuring a steady stream of functionality.
- Refactoring: Continuously refactor code to improve maintainability and adapt to changing requirements.

4. Feedback Loop Phase:

- Frequent Customer Feedback: Regularly demonstrate completed features to customers for feedback.
- Iterative Adjustments: Use customer feedback to make iterative adjustments to the system.

5. Monitoring and Maintenance Phase:

- Continuous Monitoring: Monitor system performance and user feedback continuously.
- Bug Fixes and Enhancements: Address bugs and implement enhancements based on ongoing user needs.

7. Closure Phase:

- Final Customer Review: Conduct a final review with the customer to ensure all requirements have been met.
- Documentation: Provide comprehensive documentation for future reference and maintenance.

Final Note:

It's important to note that while the outlined phases and practices align with recommended software development methodologies, the context of our project within a college setting may imply some adjustments. As a project undertaken for academic purposes within a classroom environment, certain deviations from standard industry practices are expected.

Limited Team Size:
Unlike professional settings with larger development teams, our project involves a smaller group of three students. This may influence the dynamics of activities such as pair programming and the frequency of interactions.

Academic Constraints:
The absence of real customers in a traditional sense, as well as the academic nature of the project, implies that certain aspects of customer involvement and feedback may be simulated or limited to interactions within the academic environment.

Educational Objectives:
The primary focus of this project is to enhance our understanding of software development practices, methodologies, and technologies. Therefore, certain modifications may be made to better align with the learning objectives of the course.

Simulated Customer Role:
In the absence of a real customer, the project may involve simulation of customer roles or the instructor playing a pivotal role in providing project requirements and feedback.

While our approach draws inspiration from industry-standard methodologies, these adaptations acknowledge the unique aspects of a college project. It is crucial to recognize that the adjustments made are intentional and aimed at optimizing the learning experience within the limits of our academic environment.

5. Architectural Design

a. Suggested System Architecture

Client-Server Architecture:

Client Side:

Web Application for Customers:

Client Interface: User interface for customers to interact with the system, search for tables, make reservations, and view profiles. This is the client-side component responsible for presenting information and handling user interactions.

Admin Dashboard:

Admin Client Interface: Interface for administrators to manage tables, view reservations, and perform administrative tasks. Similar to the web application for customers, this is the client-side component for administrators.

Server Side:

Application Server:

Backend Services: Implement the business logic, including reservation processing, customer management, and table management. This corresponds to the server-side logic responsible for processing requests and managing application-specific operations.

Database Server:

Relational Database: Store structured data, including customer information, reservations, table details, and admin credentials. This is the server-side component responsible for storing and retrieving data.

Benefits in a Client-Server Architecture:

Scalability:The client and server components can be scaled independently based on demand. For example, if there's a sudden increase in user activity, you can scale the server infrastructure to handle the load.

Security:The separation of concerns between the client and server enhances security. Sensible data and critical business logic are managed on the server side, reducing the risk of unauthorized access.

Flexibility:Modifications or enhancements to specific components (client or server) can be done independently without affecting the other. This makes the system more flexible and adaptable to changes.

b. Suggested Application Architecture

Layered Architecture:

Presentation Layer:

Web Application for Customers: User interface for customers to interact with the system, search for tables, make reservations, and view profiles.

Admin Dashboard: Interface for administrators to manage tables, view reservations, and perform administrative tasks.

Application Layer:

Backend Services: Implement the business logic, including reservation processing, customer management, and table management.

APIs: Define APIs for communication between the presentation layer and the backend services.

Data Layer:

Relational Database: Store structured data, including customer information, reservations, table details, and admin credentials.

Cache (optional): Implement caching for frequently accessed data to improve performance.

Benefits:

Separation of Concerns: Divides the application into distinct layers, making it modular and maintainable.

Scalability: Each layer can be scaled independently based on requirements.

Security: Provides a clear separation of concerns, enhancing security.

Flexibility: Allows for easier modifications or enhancements to specific layers without affecting others.

6. System Modelling

a. Use Case Diagram

The MySpace Reservation System, depicted in the use case diagram, provides a feature-rich environment for both Customers and Administrators.

Customer Use Cases:

Register/Login: Customers undergo the registration and login processes, with the system validating form entries, ensuring data accuracy.

Email Validation: The system enforces email validation during the registration process for added security.

Password Matching: To enhance data integrity, the system ensures that passwords and confirmation passwords match.

Search for Tables: Customers can initiate searches for available tables based on specific criteria, facilitating the selection process.

Make Reservations: Customers have the ability to reserve tables, ensuring a personalized experience.

Make Payments: Customers can complete transactions securely through the system.

View Reservation Details: Access to detailed reservation information allows customers to stay informed.

Administrator Use Cases:

Login: Administrator undergo the login processes, with the system validating form entries, ensuring data accuracy.

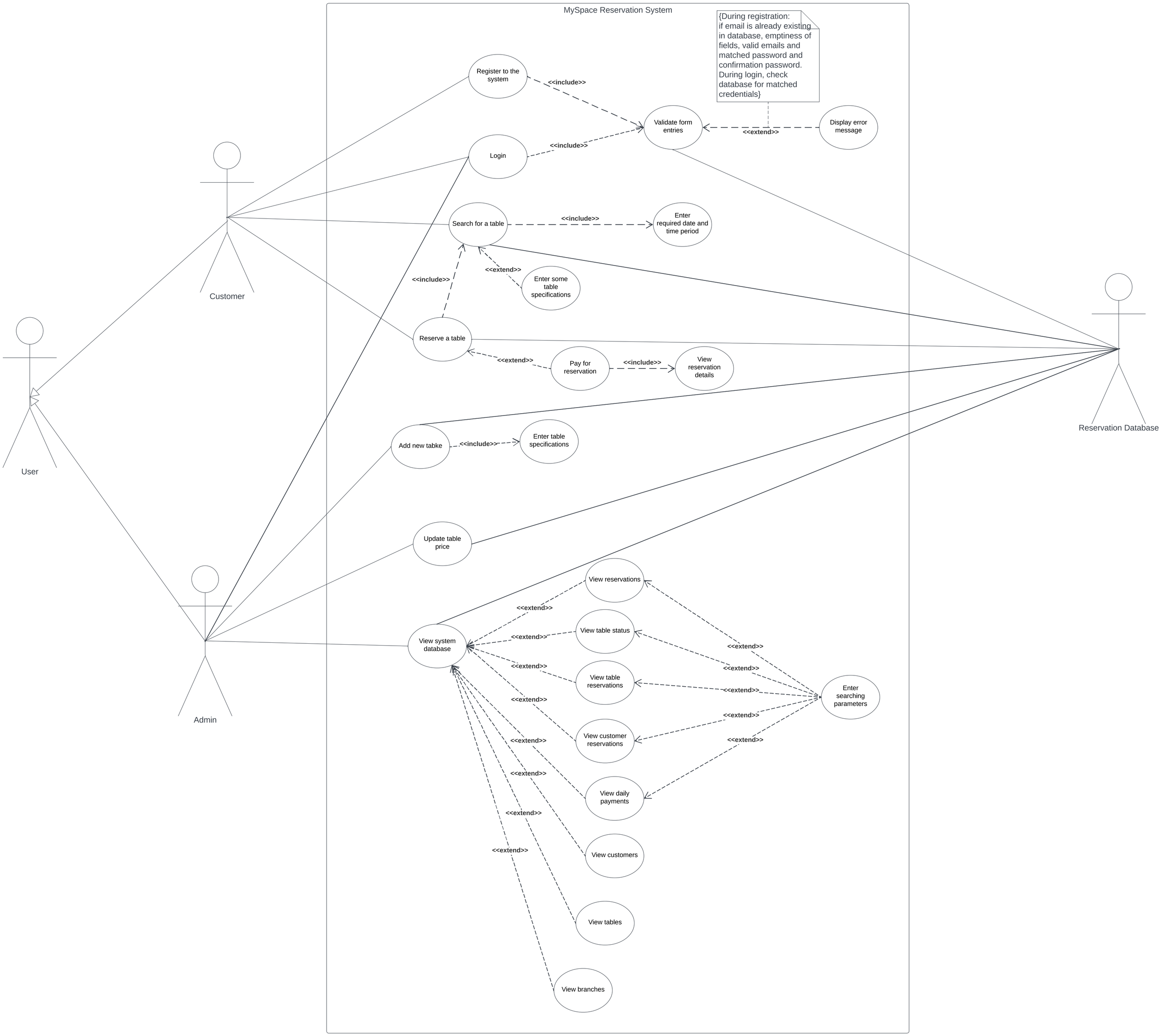
Add New Tables: The system allows administrators to add new tables, contributing to dynamic system scalability.

Update Table Prices: Administrators can access and modify table prices, maintaining pricing accuracy.

View System Database : Administrators have ability to view reporting for each system database component including reservations, table status, table reservations, customer reservation, daily payments, customers, tables, and branches.

All of the previous mentioned feature is also a use case too.

Diagram is shown in the next page


MySpace Reservation System
{During registration:
if email is already existing in database, emptiness of fields, valid emails and matched password and confirmation password.
During login, check database for matched credentials}
Register to the system
Login
Search for a table
Reserve a table
Add new tabke
Update table price
Validate form entries
Display error message
Enter required date and time period
Enter some table specifications
Pay for reservation
View reservation details
Enter table specifications
View system database
View reservations
View table status
View table reservations
View customer reservations
View daily payments
View customers
View tables
View branches
Enter searching parameters
<<include>>
<<extend>>
Customer
User
Admin
Reservation Database

b. Activity Diagram

The first Activity Diagram for the MySpace Reservation System's admin interactions illustrates the sequence of activities an administrator can perform within the system.

Start:

The activity begins with the initiation of the MySpace Reservation System by the admin.

Authenticate Admin Credentials:

The system prompts the admin to validate their credentials before granting access to the admin menu. This step ensures secure access to privileged functionalities.

Access Admin Menu:

Upon successful authentication, the admin gains access to the admin menu, providing a range of options for managing table reservations and accessing various system features.

Manage Table Reservations:

The admin has the capability to manage table reservations. This includes the ability to view and update reservations.

Add Tables:

Admins can add new tables to the system by entering specifications. The entered information is then stored in the system's database for future reference.

Update Table Prices:

Admins can modify table prices within the system, ensuring accurate and up-to-date pricing information.

View Table Status:

Admins can inquire about the status of tables for a specific day or across multiple days, providing valuable insights into table availability.

View Daily Payments:

Admins have the option to view daily payments for a specified period, aiding in financial tracking and reporting.

Show Reports:

The system allows admins to generate and view reports related to tables, customers, and branches. This feature facilitates data-driven decision-making.

Show Branches, Tables, and Customers:

Admins can access detailed information about branches, tables, and customers within the system.

End:

The activity diagram concludes, signifying the completion of admin interactions with the MySpace Reservation System.

The second Activity Diagram for Customer Interaction with the MySpace Reservation System illustrates the sequence of activities a customer can perform within the system:

Start:

The activity begins with the initiation of the MySpace Reservation System by the customer.

Login:

Customers have the option to log in to the system using their credentials. The system prompts them to validate their login information.

Validate Credentials:

The system validates the customer's login credentials. If successful, the customer gains access to reservation and payment functionalities. If validation fails, an error message is displayed.

Reserve Tables:

Authenticated customers can reserve tables from the available options. The system provides a list of tables for selection.

Registration (If Not Registered):

If a customer is not registered, they can fill out a registration form and submit it. This step ensures that all customers have a valid account for reservation and payment processes.

Confirm Payment:

After selecting and confirming their reservations, customers proceed to the payment confirmation step. The system verifies the payment and confirms it.

Redirect to Customer Page:

Upon successful payment confirmation, the system redirects the customer to their personalized page, displaying a reservation confirmation notification.

View Confirmed Reservations:

Customers can view a list of their confirmed reservations on their page. This feature provides transparency and allows customers to keep track of their reservations.

Enter New Reservation Requirements:

Customers have the option to enter new requirements for table reservations, giving them flexibility in making additional reservations.

Search for Tables:

Customers can initiate a search for available tables by clicking on the search button. This functionality enables them to filter and find tables based on specific criteria.

Redirect to Payment Page (For Another Transaction):

If a customer wishes to make another transaction, such as reserving additional tables, the system redirects them to the payment page.

The Two Diagrams are shown in the next pages

MySpace Reservation System

Admin

System

login

validate admin credentials

[incorrect admin credentials]

[correct]

show admin menu

select action

[add table]

add table

validate table specs

[else]

[valid]

show error message

store table in DB

enter table specs

[view reservations]

[update price]

update table price

validate input

[else]

[valid]

show error message

update price in DB

enter table specs

[table status]

[daily payments]

[reports]

view reservations

[for specific customer]

[for all]

[for specific table]

enter customer specs

validate input

[else]

[valid]

show error message

show reservations

enter table specs

enter specific period you search for

view table status

validate input

[else]

[valid]

show error message

show tables status at that day

enter specific day

view daily payments

enter specific period

validate input

[else]

[valid]

show error message

show daily payments at this period

show reports

[branches]

[tables]

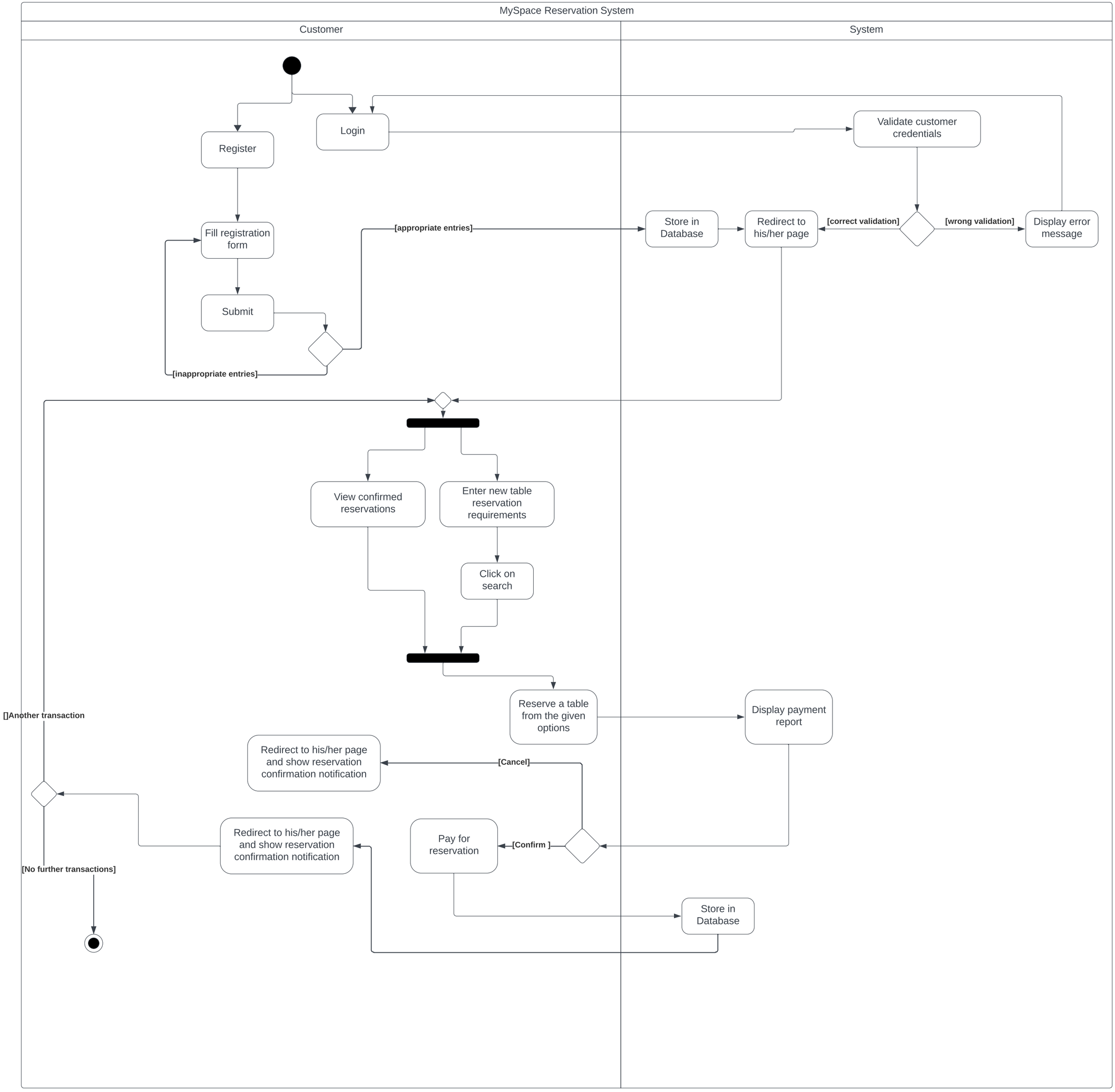
[customers]

show branches

show data required

show customers

show tables


MySpace Reservation System
Customer
System
Register
Login
Validate customer credentials
Fill registration form
[appropriate entries]
Store in Database
Redirect to his/her page
[correct validation]
[wrong validation]
Display error message
Submit
[inappropriate entries]
View confirmed reservations
Enter new table reservation requirements
Click on search
Reserve a table from the given options
Display payment report
[]Another transaction
Redirect to his/her page and show reservation confirmation notification
[Cancel]
Redirect to his/her page and show reservation confirmation notification
Pay for reservation
[Confirm]
[No further transactions]
Store in Database

c. State Machine Diagram

The first state machine diagram represents the lifecycle of a table reservation in the MySpace Reservation System. Here is a summary of the key states and transitions:

Initial State: "Available"

The table starts in the "Available" state, indicating that it is open for reservation.

Transition: Customer Requests Reservation

When a customer requests a reservation, the system transitions the table state to "Not Available."

Concurrently, the system updates the table status in the database to reflect its unavailability during the specified time period.

Transition: Customer Cancels Payment

If a customer cancels the payment before confirmation, the system transitions the table state back to "Available."

The table status in the database is updated to make it available again during the previously reserved time period.

Transition: Customer Confirms Payment

Upon successful payment confirmation, the system transitions the table state to "Reserved."

Simultaneously, the table status in the database is updated to indicate its reservation during the specified time period.

Final State: "Reserved"

The table reaches the final state of "Reserved," signifying a confirmed reservation.

The customer is directed to their main page to view and manage their reservations.

The second state machine diagram represents the registration process for customers in the MySpace Reservation System. Here is a summary of the key states and transitions:

Initial State: "Unregistered"

- The customer starts in the "Unregistered" state, indicating that they are not yet registered in the system.

Transition: New User Requests Registration Page

- A new user in the "Unregistered" state requests the registration page and submits the registration form.

Transition: Registration Form Validation (Correct)

- If the registration form is correctly validated, the system transitions the customer to the "Registered" state.
- The system redirects the customer to their personalized page, showcasing options for table search and reservation.

Transition: Registration Form Validation (Incorrect)

- If the registration form validation is incorrect, the system prompts the customer to enter valid entries.
- Relevant error messages are displayed to guide the user.
- The system remains on the registration page, awaiting corrected input from the user.

Transition: New User Resubmits Registration Form

- If the user resubmits the registration form after receiving error messages, the system revalidates the entries.

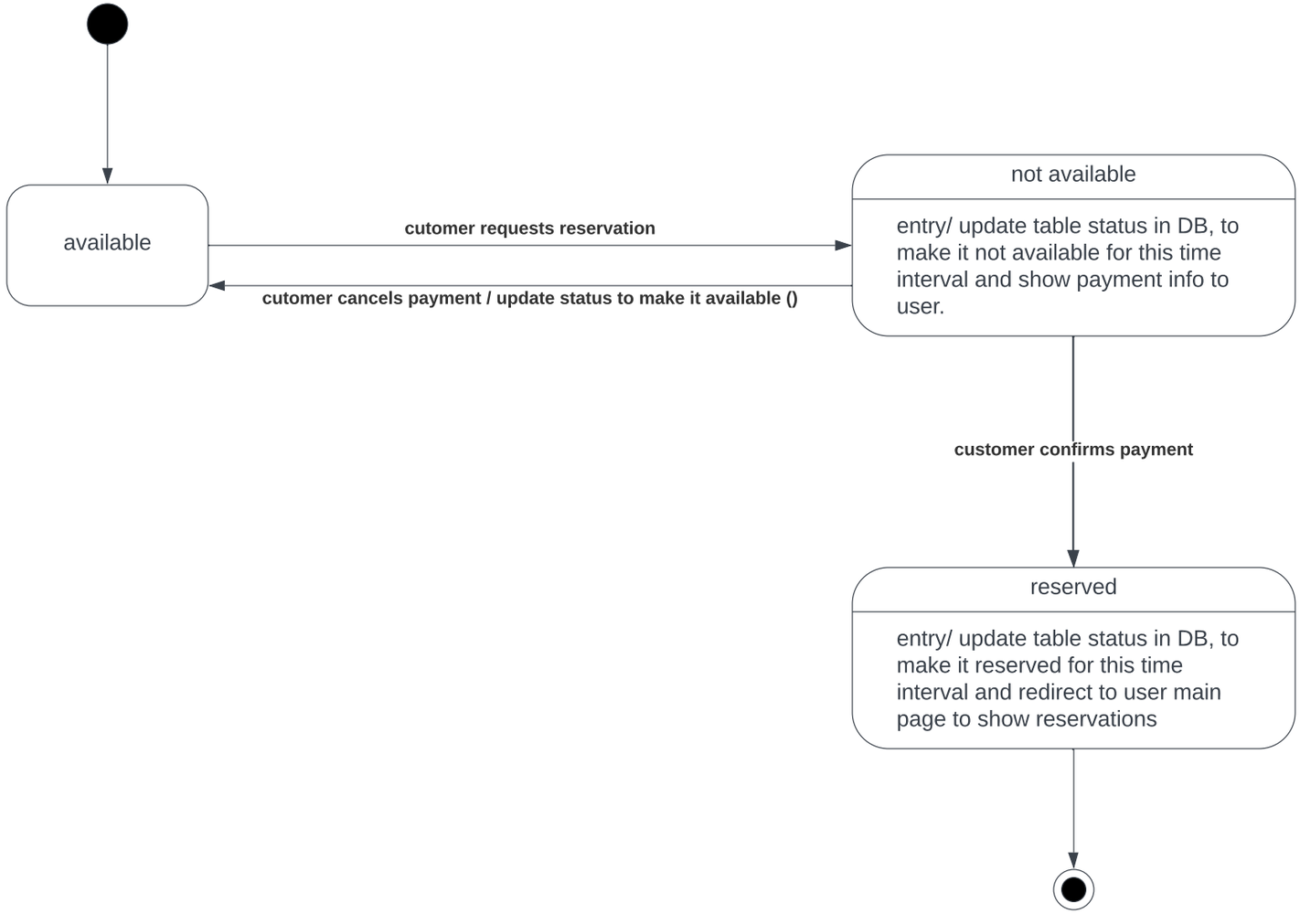
Transition: Registration Form Validation (Correct, Retry)

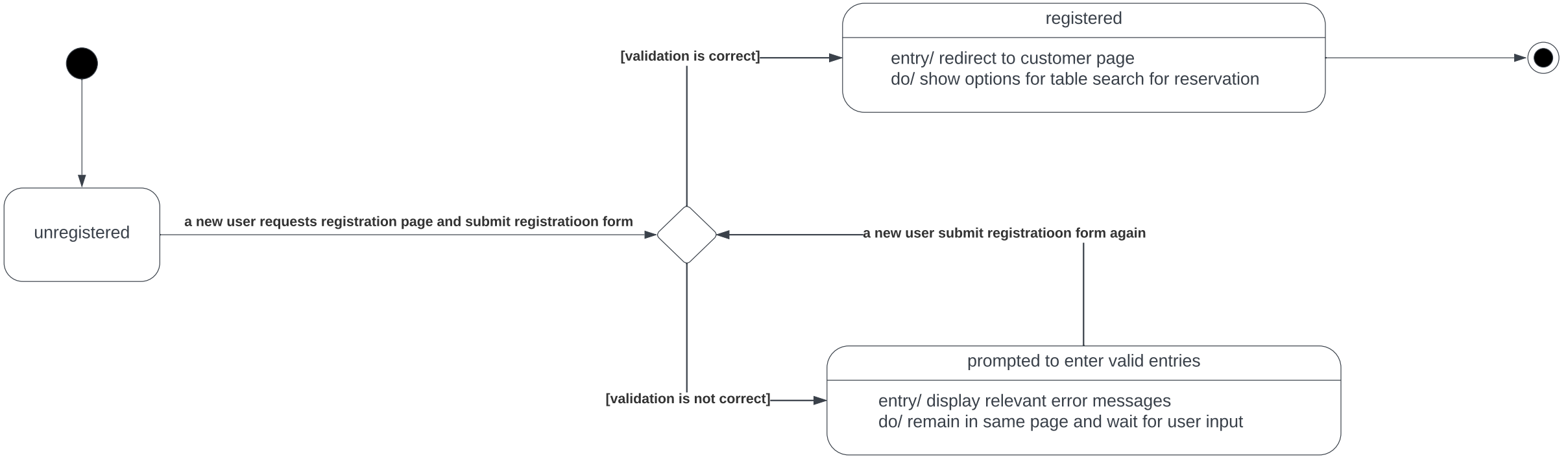
- If the resubmitted registration form is correctly validated, the system transitions the customer to the "Registered" state.
- The system redirects the customer to their personalized page, providing options for table search and reservation.

Final State: "Registered"

- The customer reaches the final state of "Registered," signifying successful registration in the system.
- The customer is redirected to their main page, where they can access various functionalities, including options for table search and reservation.

The Two Diagrams are shown in the next pages


available
cutomer requests reservation
cutomer cancels payment / update status to make it available ()
not available
entry/ update table status in DB, to make it not available for this time interval and show payment info to user.
customer confirms payment
reserved
entry/ update table status in DB, to make it reserved for this time interval and redirect to user main page to show reservations


unregistered
a new user requests registration page and submit registratioon form
[validation is correct]
registered
entry/ redirect to customer page
do/ show options for table search for reservation
a new user submit registratioon form again
[validation is not correct]
prompted to enter valid entries
entry/ display relevant error messages
do/ remain in same page and wait for user input

d. Sequence Diagram

The following are enumerated paragraphs in order of their existence in the report document.

1. The depicted sequence diagram explains the customer registration procedure within the workspace reservation system. It begins with the validation of form entries at the controller, which prompts users to correct any invalid inputs. Subsequently, the system inspects the user-provided email address, ensuring its uniqueness in the database. If the email already exists, the sequence transitions to the 'Email already exists' alternative path; otherwise, the input undergoes storing in the Customer table as a new customer entry in the database. This sequential flow ensures a thorough validation and registration process for customers, optimizing the accuracy and integrity of the workspace reservation system's data.

2. The presented sequence diagram describe the user credential validation process, initiated by the user inputting login details, subsequently verified by the Workspace Reservation Controller. In the event of an input format error, the controller prompts the user for reentry through an error message. Upon successful validation, the system queries the database for customer credentials, granting access if a match is found, and displaying an error message if not. For admin users, a separate validation process happens without direct database access, considering the singular existence of one admin. Successful validation provides admin access, while failure prompts an error message denoting an unrecognized admin account.

3. This sequence diagram illustrates the details of searching for workspace tables in the database. It starts with a customer initiating a reservation request, subject to validation for input errors. The reservation date and time are mandatory, while additional search criteria remain optional for further specification. In the event of errors or incomplete mandatory entries, an error message is generated. Upon successful validation, the system proceeds to query the database utilizing the collected form attributes. If no entries align with the search criteria, a message notifies the customer that no matching tables were found. Conversely, if entries are present, the system retrieves and displays the results, providing the customer with the relevant information for informed decision-making in the workspace reservation process.

4. The sequence diagram describes the process of reserving a workspace table, starting by the customer initiating a reservation request managed by the Reservation Controller. Upon the user's selection of a table, it is promptly marked as 'Not available' in the database, a preemptive measure for transaction atomicity, ensuring exclusive reservation rights during the transaction period. Following payment confirmation, the table status transitions to 'Reserved' in the database, and a corresponding entry is appended to the Reservations table. Database modifications are executed, and the customer is redirected to their homepage, where a confirmation of successful reservation is displayed. In the event of unconfirmed payment, a transaction rollback occurs, reverting the table status to 'Not available.' The customer is then redirected to their homepage, informed that no reservation has been made. These comprehensive interactions ensures a secure and seamless workspace table reservation process, prioritizing transaction integrity and user communication.

5. The sequence diagram explains the process of modifying table attributes within a workspace reservation system, initiated by an admin user. The admin enters the required modifications through a form, initiating a validation step to ensure the integrity of the input. In the event of invalid entries, the system seamlessly loops back to the form entry stage, accompanied by a detailed error message to guide the user in correcting the issues. Upon successful validation, the system proceeds to update the table attributes in the database. The sequence terminates with a confirmation message, reassuring the admin user that the modification process was executed successfully. This streamlined flow ensures a systematic and user-friendly approach to table attribute modification, enhancing the efficiency and accuracy of the workspace reservation system.

6. This sequence diagram describes the comprehensive process of validating input and querying a database for workspace reservations. Initiated by the admin, the sequence begins with the collection of form attributes and subsequent database query. Input validation is enforced, with the system promptly prompting the admin to correct any invalid entries through elaborate error messages. Following successful validation, the query is transmitted for execution in the database, and the subsequent results are retrieved. Notably versatile, this sequence diagram accommodates the diverse scenarios of searching reservations, whether in the general use case, customer reservation, or table reservation use case. Its clarity and

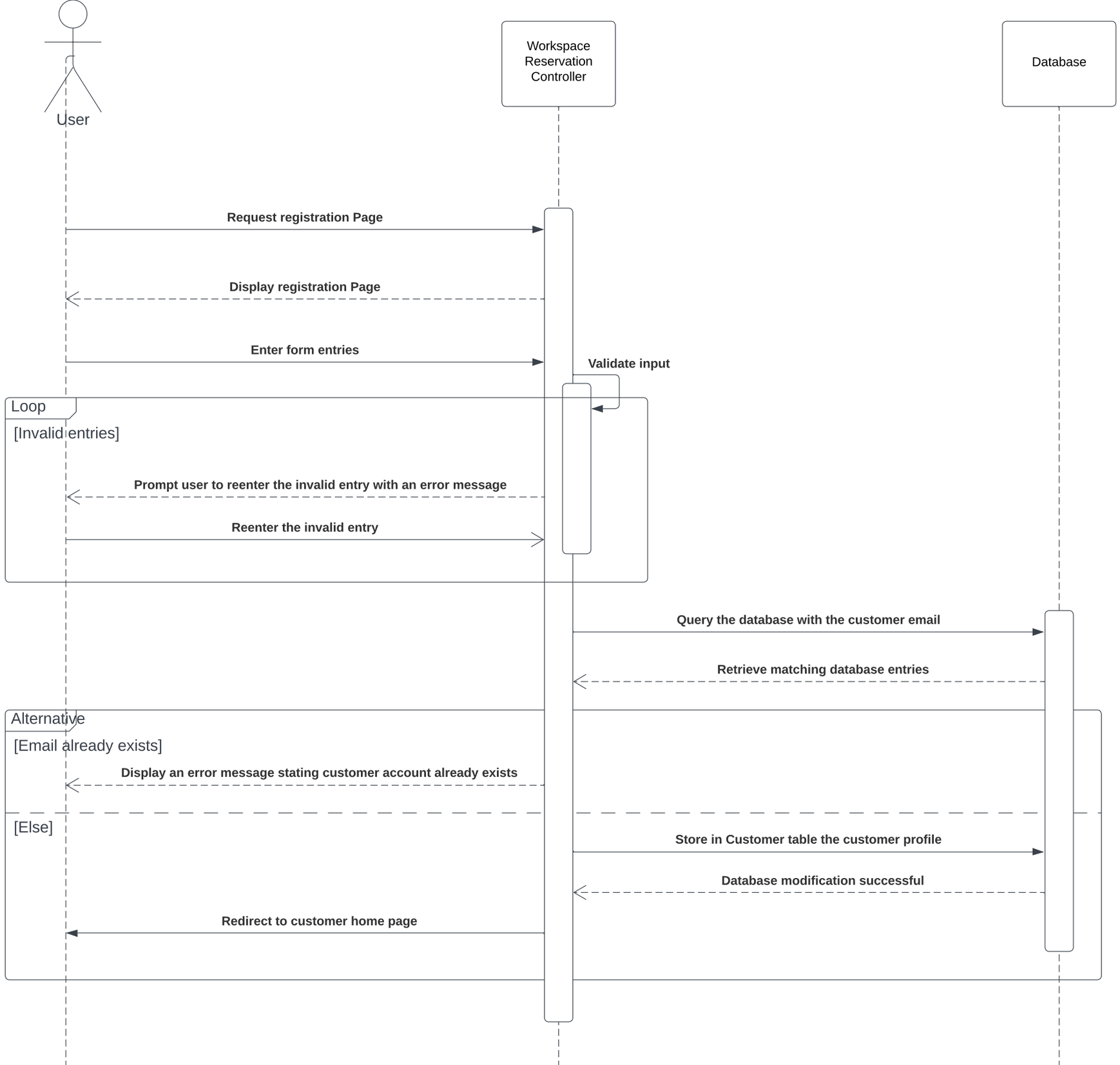
conciseness make it an effective tool for understanding the details of validating input and querying the database in the context of workspace reservations.

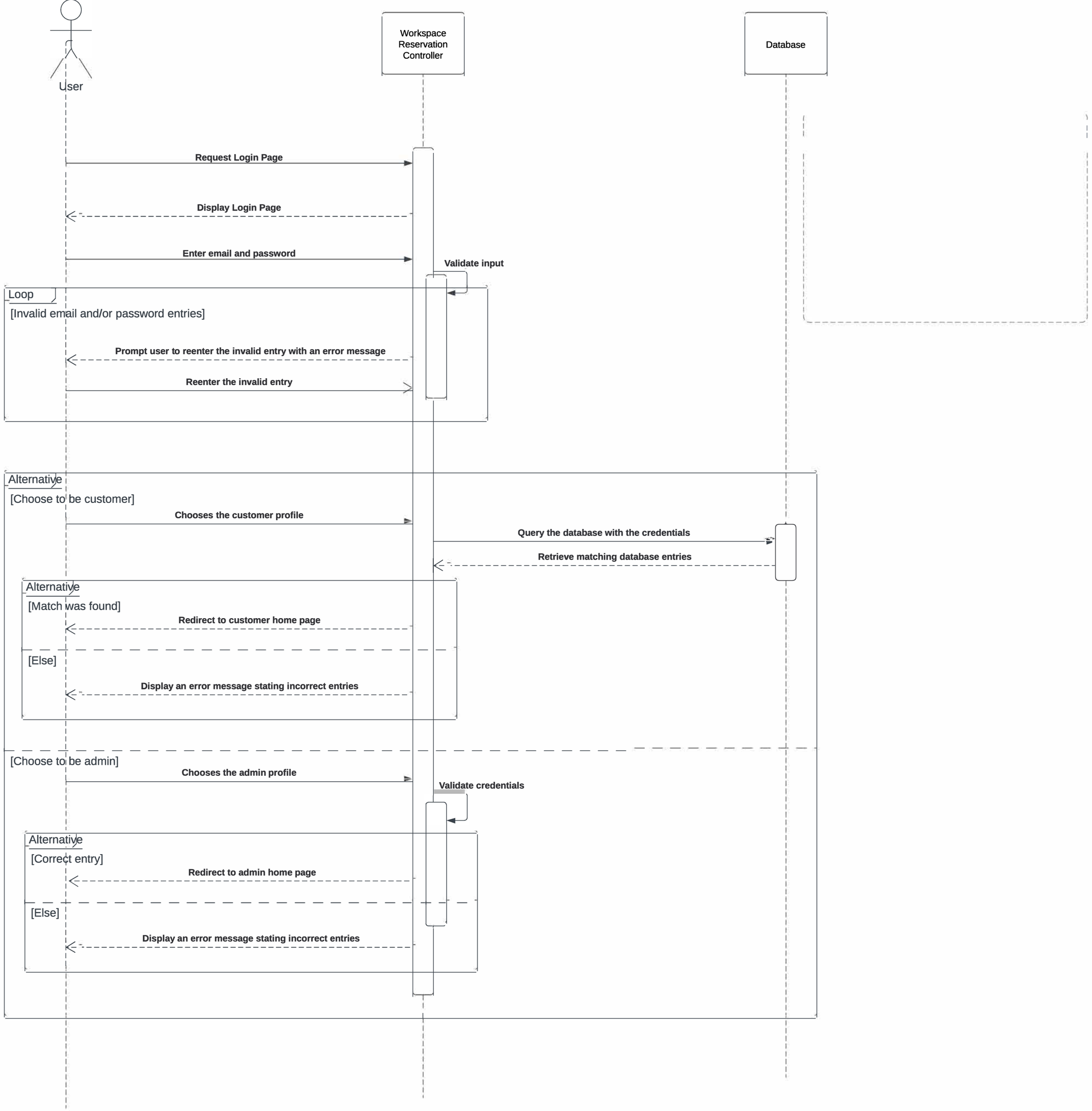
7. This sequence diagram illustrates the process through which an admin views the table statuses for a specified time period within the workspace reservation system. Initiated by the admin inputting a date for availability checking, the input undergoes validation, and in the case of invalid entries, the admin is guided to input a valid date through a detailed error message. Upon successful validation, the controller proceeds to query the database using the collected date. The database, in turn, retrieves the query results and transmits them back to the controller. Subsequently, the controller takes charge of displaying the table statuses to the admin, providing a streamlined and efficient mechanism for assessing table availability within the designated time frame.

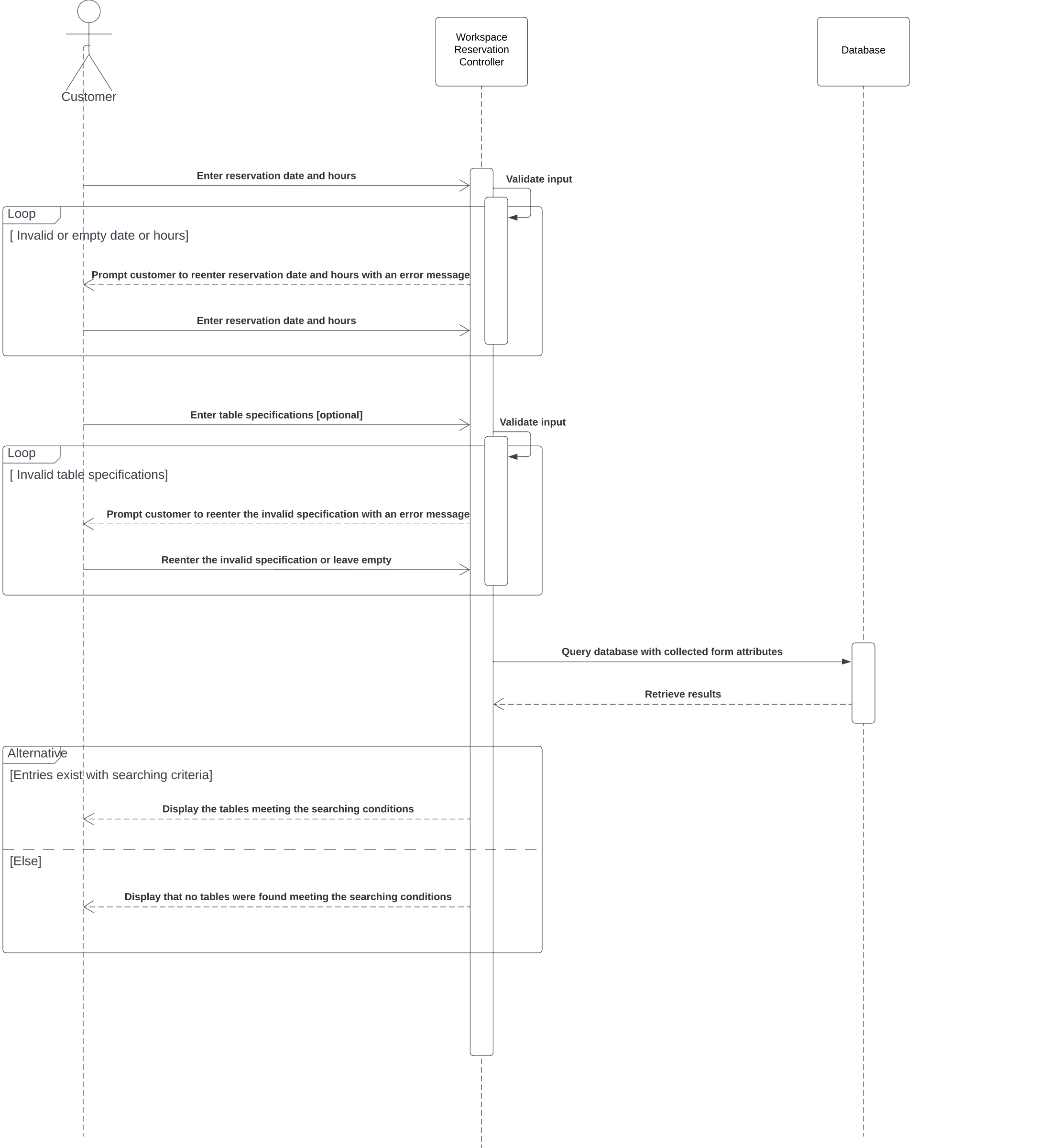
8. This sequence diagram describes the process by which an admin reviews daily payments received for a specified time period in the workspace reservation system. The admin initiates the operation by inputting a date, with the option to define a date interval or leave it empty for a comprehensive view of historical and upcoming reservations. Input validation is enforced, and in the case of invalid entries, the admin receives guidance to input a valid date through a detailed error message. Upon successful validation, the controller proceeds to query the database with the collected date, facilitating the retrieval of payment information. The database transmits the query results back to the controller, which subsequently displays the payment details to the admin. This sequence ensures a systematic and user-friendly approach for admins to assess daily payments within specified time frames, enhancing the efficiency of the workspace reservation system.

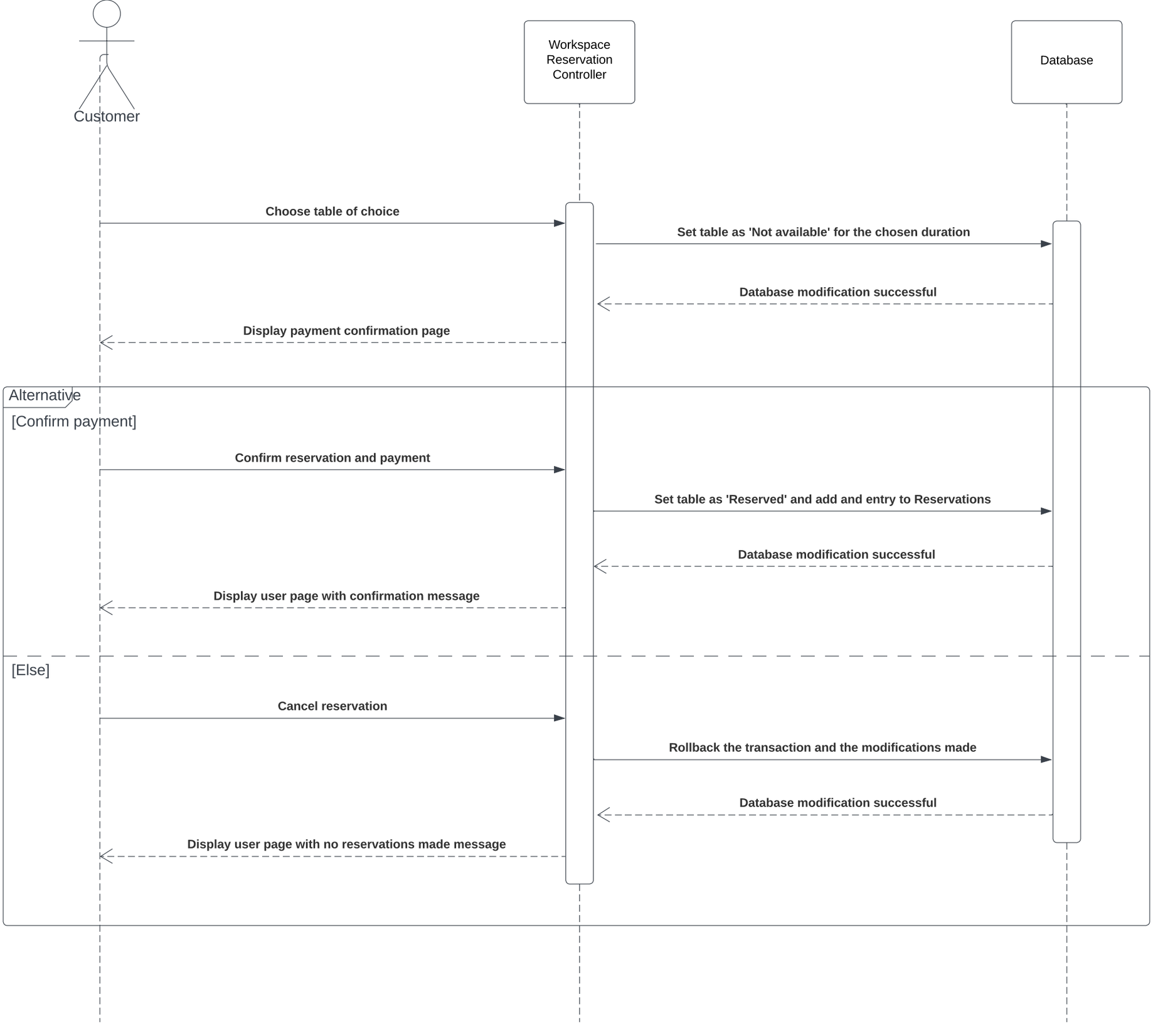
9. This sequence diagram illustrates the interaction between the Workspace Reservation Controller and the Admin in the process of querying the Database to retrieve data. The Admin, in this scenario, has the flexibility to choose from three distinct options: viewing customers, tables, or branches. The diagram delineates optional paths corresponding to each of these choices, accommodating three distinct use cases. Whether the Admin opts to explore customer data, table information, or branch details, the sequence diagram provides a comprehensive representation of the interactions and optional paths associated with each use case.

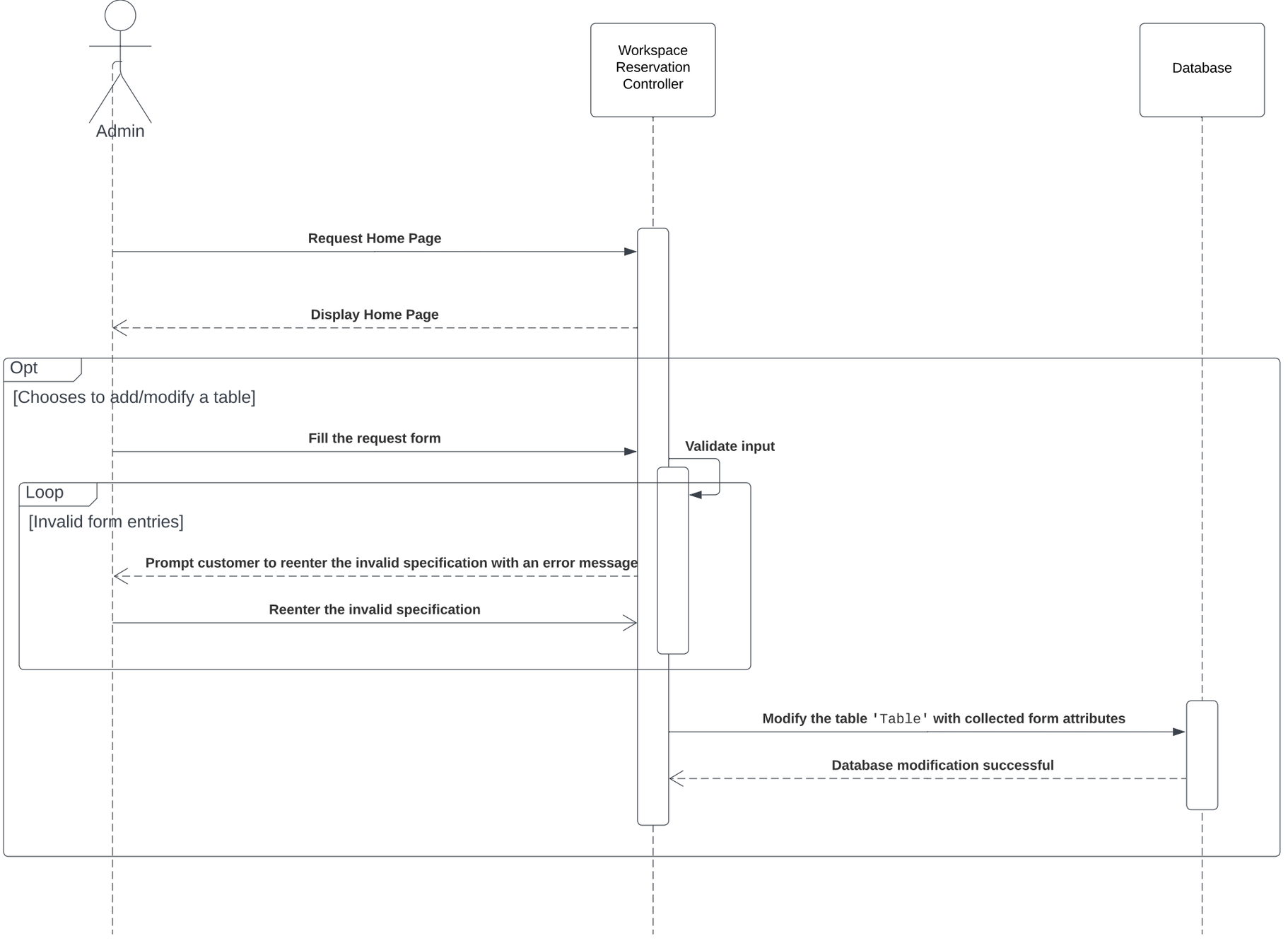
The Nine Diagrams are shown in the next pages

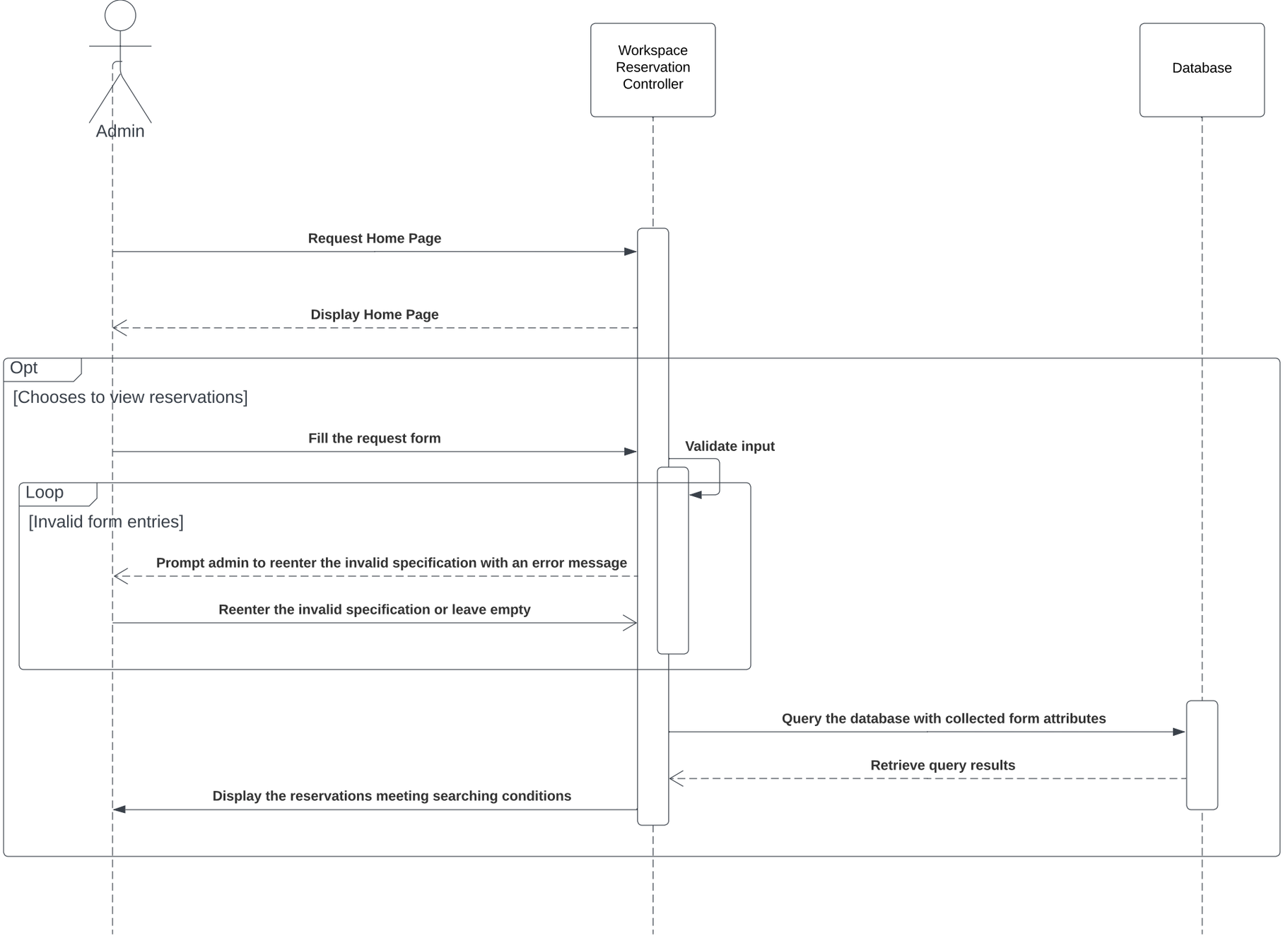

User
Workspace Reservation Controller
Database
Request registration Page
Display registration Page
Enter form entries
Validate input
Loop
[Invalid entries]
Prompt user to reenter the invalid entry with an error message
Reenter the invalid entry
Query the database with the customer email
Retrieve matching database entries
Alternative
[Email already exists]
Display an error message stating customer account already exists
[Else]
Store in Customer table the customer profile
Database modification successful
Redirect to customer home page

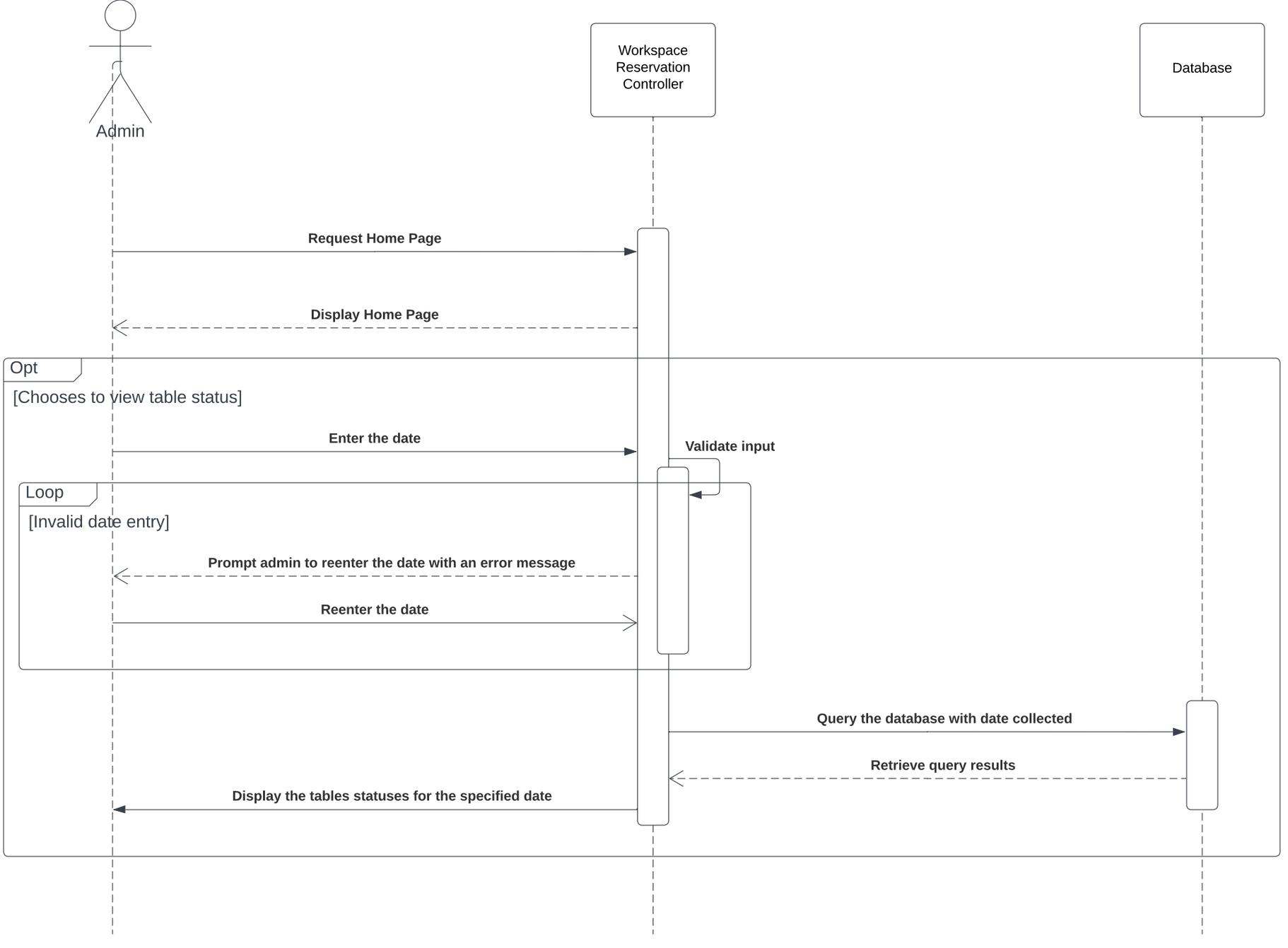

User
Workspace Reservation Controller
Database
Request Login Page
Display Login Page
Enter email and password
Validate input
Loop
[Invalid email and/or password entries]
Prompt user to reenter the invalid entry with an error message
Reenter the invalid entry
Alternative
[Choose to be customer]
Chooses the customer profile
Query the database with the credentials
Retrieve matching database entries
Alternative
[Match was found]
Redirect to customer home page
[Else]
Display an error message stating incorrect entries
[Choose to be admin]
Chooses the admin profile
Validate credentials
Alternative
[Correct entry]
Redirect to admin home page
[Else]
Display an error message stating incorrect entries

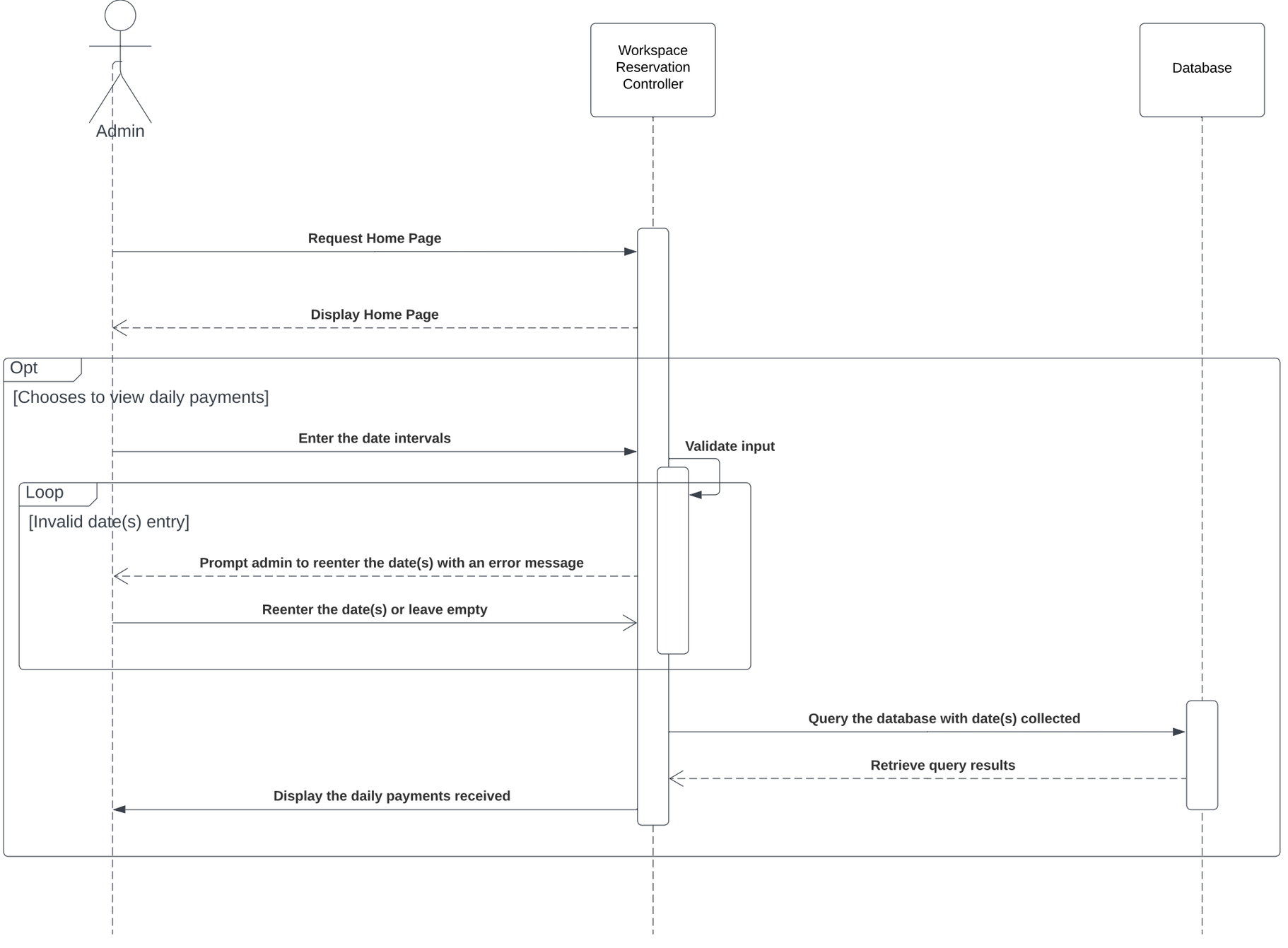

Customer
Workspace Reservation Controller
Database
Enter reservation date and hours
Validate input
Loop
[Invalid or empty date or hours]
Prompt customer to reenter reservation date and hours with an error message
Enter reservation date and hours
Enter table specifications [optional]
Validate input
Loop
[Invalid table specifications]
Prompt customer to reenter the invalid specification with an error message
Reenter the invalid specification or leave empty
Query database with collected form attributes
Retrieve results
Alternative
[Entries exist with searching criteria]
Display the tables meeting the searching conditions
[Else]
Display that no tables were found meeting the searching conditions

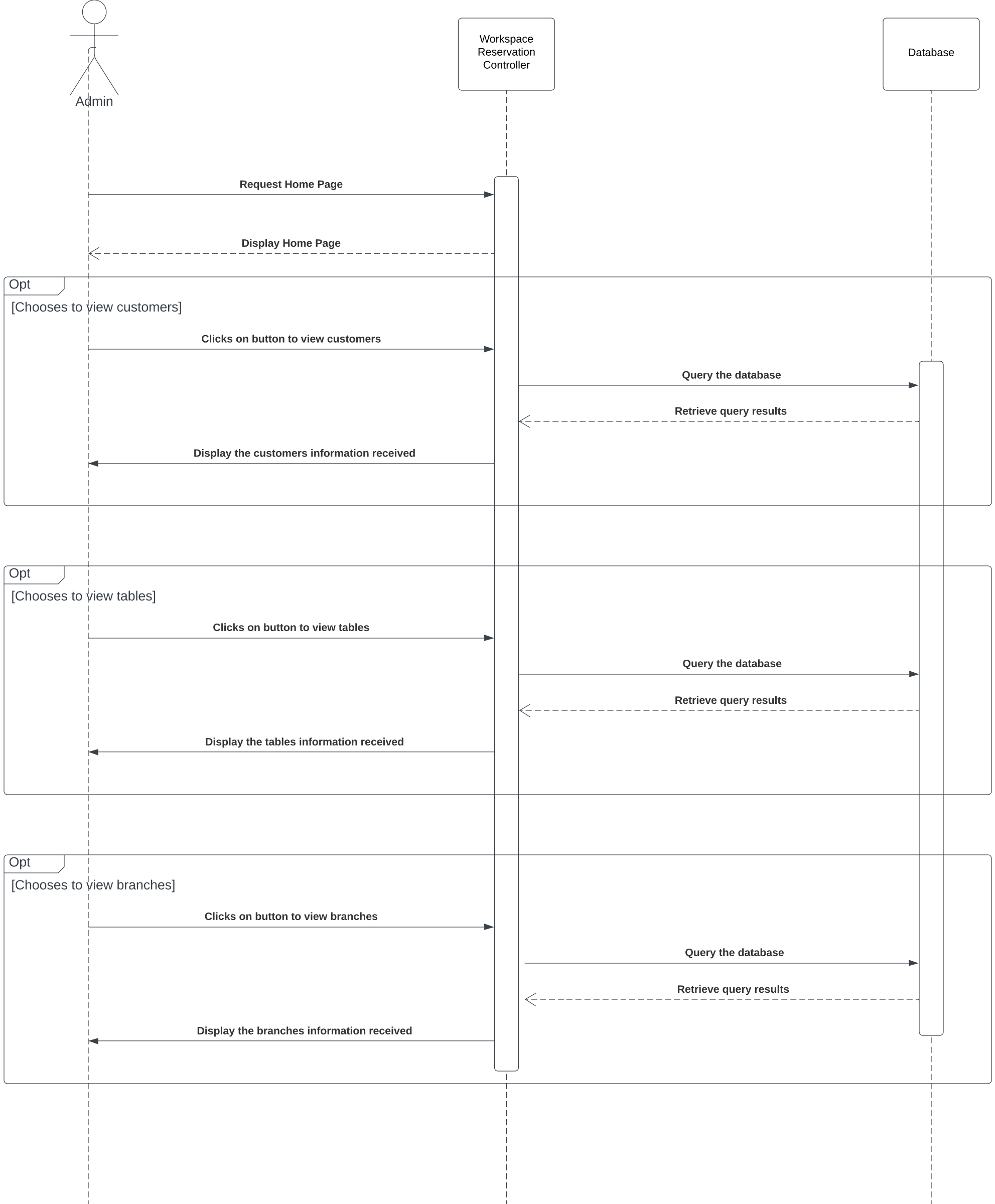

Customer
Workspace Reservation Controller
Database
Choose table of choice
Set table as 'Not available' for the chosen duration
Database modification successful
Display payment confirmation page
Alternative
[Confirm payment]
Confirm reservation and payment
Set table as 'Reserved' and add and entry to Reservations
Database modification successful
Display user page with confirmation message
[Else]
Cancel reservation
Rollback the transaction and the modifications made
Database modification successful
Display user page with no reservations made message


Admin
Workspace Reservation Controller
Database
Request Home Page
Display Home Page
Opt
[Chooses to add/modify a table]
Fill the request form
Validate input
Loop
[Invalid form entries]
Prompt customer to reenter the invalid specification with an error message
Reenter the invalid specification
Modify the table 'Table' with collected form attributes
Database modification successful


Admin
Workspace Reservation Controller
Database
Request Home Page
Display Home Page
Opt
[Chooses to view reservations]
Fill the request form
Validate input
Loop
[Invalid form entries]
Prompt admin to reenter the invalid specification with an error message
Reenter the invalid specification or leave empty
Query the database with collected form attributes
Retrieve query results
Display the reservations meeting searching conditions


Admin
Workspace Reservation Controller
Database
Request Home Page
Display Home Page
Opt
[Chooses to view table status]
Enter the date
Validate input
Loop
[Invalid date entry]
Prompt admin to reenter the date with an error message
Reenter the date
Query the database with date collected
Retrieve query results
Display the tables statuses for the specified date

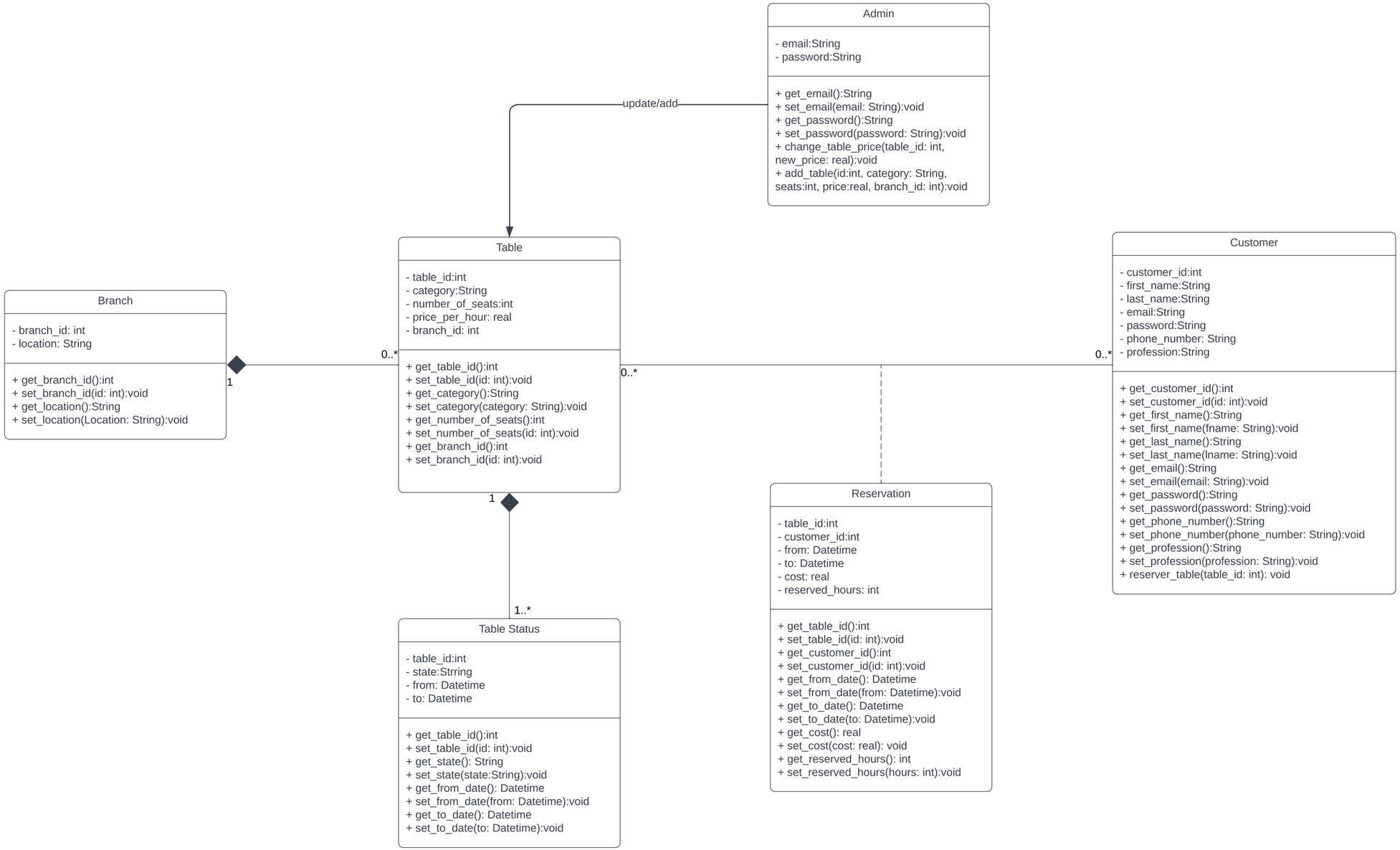

Admin
Workspace Reservation Controller
Database
Request Home Page
Display Home Page
Opt
[Chooses to view daily payments]
Enter the date intervals
Validate input
Loop
[Invalid date(s) entry]
Prompt admin to reenter the date(s) with an error message
Reenter the date(s) or leave empty
Query the database with date(s) collected
Retrieve query results
Display the daily payments received


Admin
Workspace Reservation Controller
Database
Request Home Page
Display Home Page
Opt
[Chooses to view customers]
Clicks on button to view customers
Query the database
Retrieve query results
Display the customers information received
Opt
[Chooses to view tables]
Clicks on button to view tables
Query the database
Retrieve query results
Display the tables information received
Opt
[Chooses to view branches]
Clicks on button to view branches
Query the database
Retrieve query results
Display the branches information received

e. Class Diagram

The class diagram presents a comprehensive data model for a reservation system, encompassing key entities such as Reservation, Branch, Table, Table Status, Customer, and Admin. Each entity is defined with its relevant attributes and relationships. A Reservation includes essential details like table_id, customer_id, dates, cost, and reserved hours. The Branch entity is characterized by a branch_id and location. Tables, categorized by table_id, also include information on seats, price per hour, and their association with a specific branch. The Table Status entity captures the state of a table at different times. Customer entities encompass attributes such as customer_id, name, email, password, phone_number, and profession. Admins, identified by email and password, have the capability to modify table prices and add new tables. Relationships between entities are established through associations, compositions, and associations with an association class. For instance, the composition relationship between Branch and Table signifies that a branch can have multiple tables, each existing within that branch. Similarly, the composition between Table and Table Status highlights that each table can have multiple statuses, with the initial status being 'available.' The association between Admin and Table reflects the admin's authority to add or modify table attributes. Lastly, the association class Reservation links Customer and Table, allowing a customer to reserve tables over various time periods and the table being reserved by any number of customers, creating a robust and interconnected data model for the reservation system.

The Diagram is shown in the next page


Admin
- email:String
- password:String
+ get_email():String
+ set_email(email: String):void
+ get_password():String
+ set_password(password: String):void
+ change_table_price(table_id: int, new_price: real):void
+ add_table(id:int, category: String, seats:int, price:real, branch_id: int):void
update/add
Branch
- branch_id: int
- location: String
+ get_branch_id():int
+ set_branch_id(id: int):void
+ get_location():String
+ set_location(Location: String):void
1
0..*
Table
- table_id:int
- category:String
- number_of_seats:int
- price_per_hour: real
- branch_id: int
+ get_table_id():int
+ set_table_id(id: int):void
+ get_category():String
+ set_category(category: String):void
+ get_number_of_seats():int
+ set_number_of_seats(id: int):void
+ get_branch_id():int
+ set_branch_id(id: int):void
1
1..*
Table Status
- table_id:int
- state:Strring
- from: Datetime
- to: Datetime
+ get_table_id():int
+ set_table_id(id: int):void
+ get_state(): String
+ set_state(state:String):void
+ get_from_date(): Datetime
+ set_from_date(from: Datetime):void
+ get_to_date(): Datetime
+ set_to_date(to: Datetime):void
0..*
0..*
Reservation
- table_id:int
- customer_id:int
- from: Datetime
- to: Datetime
- cost: real
- reserved_hours: int
+ get_table_id():int
+ set_table_id(id: int):void
+ get_customer_id():int
+ set_customer_id(id: int):void
+ get_from_date(): Datetime
+ set_from_date(from: Datetime):void
+ get_to_date(): Datetime
+ set_to_date(to: Datetime):void
+ get_cost(): real
+ set_cost(cost: real): void
+ get_reserved_hours(): int
+ set_reserved_hours(hours: int):void
0..*
Customer
- customer_id:int
- first_name:String
- last_name:String
- email:String
- password:String
- phone_number: String
- profession:String
+ get_customer_id():int
+ set_customer_id(id: int):void
+ get_first_name():String
+ set_first_name(fname: String):void
+ get_last_name():String
+ set_last_name(lname: String):void
+ get_email():String
+ set_email(email: String):void
+ get_password():String
+ set_password(password: String):void
+ get_phone_number():String
+ set_phone_number(phone_number: String):void
+ get_profession():String
+ set_profession(profession: String):void
+ reserver_table(table_id: int): void

7. Design and Implementation

a. Design Description

The design process of the workstation system included some activities that are mainly derived from the object-oriented design process. In the following section, we show how we started to design the system by identifying the context and modes of use, then specifying the system architecture, designing the database, identifying the design models and finally identifying the user interfaces.

a.1 System Context and Modes of Use of the System:

System Overview:

The Workspace Reservation System aims to simplify table reservations in workspaces, catering to the needs of both regular users and administrators. By providing a user-friendly platform, the system addresses the common challenges associated with finding and managing table reservations across multiple branches.

User Roles:

The system defines two primary user roles:

1. Regular Users:
    a. Responsibilities: Reserve tables, view reservations, login, and register.
    b. Interactions: Utilize the interface designed for normal user activities.
2. Administrators:
    a. Responsibilities: System administration, information updates, and addition of new tables.
    b. Interactions: Access the dedicated admin interface for managing system functionalities.

a.2. System Architecture

The implemented system adopts a client-server architecture to facilitate the functionalities of the Workspace Reservation System. This architecture effectively separates the concerns of the end-user, who may be a regular client seeking to reserve a table, or an admin overseeing system administration. The system is organized into distinct services, strategically designed to handle specific functions.

- Client: Represents the end-user, either a regular client or an admin. Clients interact with the system to perform actions such as table reservations and system administration.
- Server: The server component plays a pivotal role by handling the diverse functions required by clients. It serves as the centralized hub for processing reservation requests, managing system updates, and ensuring seamless communication between clients and the shared database.
- Services: The system functionalities are modularized into services, contributing to a more organized and maintainable architecture. These services can include, but are not limited to, user authentication, reservation processing, and system administration.
- Shared Database: A shared database serves as the backbone of the system, providing a unified storage space accessible by various services. This ensures consistency and coherence of information across the entire system.

The client-server architecture streamlines the interactions within the Workspace Reservation System. Clients, whether regular users or admins, direct their requests to the server, which, in turn, coordinates with the appropriate services and accesses the shared database to fulfill the requested functionalities.

a.3 Database Design

1. Tables:

CUSTOMER Table

- cust_id: Unique identifier for each customer (auto-incremented).
- fname: First name of the customer.
- lname: Last name of the customer.
- email: Email address of the customer.
- password: Password for customer authentication.
- phone_number: Phone number of the customer.
- profession: Profession of the customer.
- Primary Key: cust_id

BRANCH Table

- branch_id: Unique identifier for each branch (auto-incremented).
- location: The location of the branch.

- Primary Key: branch_id

TABLE Table

- table_id: Unique identifier for each table (auto-incremented).
- category: Category of the table (e.g., differentiating between types of tables).
- number_of_seats: The number of seats available at the table.
- price_per_hour: The cost per hour for reserving the table.
- branch_id: Foreign key referencing the branch to which the table belongs.
- Primary Key: table_id
- Foreign Key: branch_id references BRANCH(branch_id)

TABLE_STATUS Table

- table_id: Foreign key referencing the table.
- state: The state of the table (e.g., reserved, available).
- from: Start date and time of the table status.
- to: End date and time of the table status.
- Primary Key: table_id, state, from
- Foreign Key: table_id references TABLE(table_id)

RESERVATION Table

- cust_id: Foreign key referencing the customer.
- table_id: Foreign key referencing the table.
- from: Start date and time of the reservation.
- to: End date and time of the reservation.
- cost: Cost of the reservation.
- reserved_hours: Number of hours the table is reserved.
- Primary Key: cust_id, table_id, from
- Foreign Key: cust_id references CUSTOMER(cust_id); table_id references TABLE(table_id)

ADMIN Table

- admin_id: Unique identifier for each admin (auto-incremented).
- admin_email: Email address of the admin.
- admin_password: Password for admin authentication.
- Primary Key: admin_id

2. Constraints:

- **Foreign Key Constraints:**
    - STATTABLEFK: TABLE_STATUS(table_id) references TABLE(table_id)
    - BRATABFK: TABLE(branch_id) references BRANCH(branch_id)
    - RESTABFK: RESERVATION(cust_id) references CUSTOMER(cust_id)
    - RESTABFK2: RESERVATION(table_id) references TABLE(table_id)

These constraints ensure referential integrity, linking the tables together based on their relationships. For example, the RESERVATION table is linked to both the CUSTOMER and TABLE tables via foreign keys. The TABLE_STATUS table tracks the status of each table over time.

This database design supports the functionality of the Workspace Reservation System, allowing for the management of tables, branches, customers, reservations, and administrators.

a.4. Develop Design Models:

As shown in the report, design models were implemented to identify and describe all aspects of the system. These design models included:

- State machine diagrams: to show how the individual property of an object is changed in response to internal or external events
- Use case diagram: to show the interaction between the system and the environment.
- Sequence diagram: to model the sequence of interactions between objects.
- Activity diagram: to represent the flow of activities or actions within a system or a specific process.
- Class diagrams: to describe all the entities in the system, their attributes, methods, and relationships between them.

a.5 Designing the user interface

The user interface (UI) of the web application is thoughtfully designed to cater to two primary user roles: normal users and administrators.

- Normal User Interface:
    - Allows regular users to seamlessly navigate through the process of reserving a table, viewing their existing reservations, and managing their account through login and registration functionalities.
    - The interface prioritizes user-friendliness, ensuring a smooth and intuitive flow for users to reserve tables with ease.
    - Incorporates informative alerts to guide users and provide assistance in case of errors, enhancing the overall user experience.
- Admin Interface:
    - Provides administrators with a dedicated section featuring pertinent information about the system.
    - Enables admins to efficiently perform system updates, additions, and modifications, contributing to effective system administration.
    - The admin interface is designed to be informative, allowing administrators to access key insights about the system and take necessary actions.

By structuring the user interface in this manner, the system caters to the distinct needs of both regular users and administrators, enhancing accessibility and usability across the board. The design incorporates a user-centric approach, promoting an intuitive and enjoyable experience for all users of the Workspace Reservation System.

b. Implementation (Development environment & Coding)

b.1 Development Environment

The development of the Workspace Reservation System was carried out within a carefully chosen environment to ensure efficiency, maintainability, and optimal performance.

Programming Languages and Frameworks:

The core technologies utilized in the implementation include HTML, CSS, JavaScript, and PHP. These languages collectively provide a robust foundation for creating a dynamic and responsive web application. PHP serves as the server-side scripting language, seamlessly connecting the system to the database.

Web Server:

The Apache server was selected as the web server to host and serve the web application. Apache's reliability and versatility contributed to the system's overall stability.

Database Connectivity:

The system's interaction with the database was facilitated through PHP. PHP, being a server-side scripting language, allowed for seamless communication with the database, ensuring efficient data retrieval and manipulation.

Development Tools:

- **Integrated Development Environment (IDE):**
    - Visual Studio Code was the IDE of choice for its lightweight yet feature-rich interface. Its extensive support for various languages, debugging capabilities, and a vibrant extension ecosystem significantly contributed to the development workflow.

- **Version Control System:**
    - The development process was streamlined with the integration of a version control system. Git was employed to track changes, manage collaboration, and maintain a versioned history of the codebase.
- **Testing Frameworks:**
    - Two testing frameworks, Jest and Cypress, were instrumental in ensuring the reliability and functionality of the system. Jest, a testing framework for JavaScript, was utilized for unit testing, while Cypress, a JavaScript end-to-end testing framework, facilitated comprehensive testing of the entire application.

This carefully curated development environment and toolset played a pivotal role in the successful implementation of the Workspace Reservation System. The combination of robust programming languages, efficient web server, and powerful development tools contributed to the creation of a responsive, user-friendly, and functionally sound web application.

8. Testing

Introduction:

This section provides an overview of the testing strategies employed during the development and release phases of the workspace reservation system. The testing process involves both unit testing using Jest for development testing and end-to-end testing using Cypress for release testing.

a. Development Testing

Jest Unit Testing:

Development testing involves continuous testing during the development phase to identify and fix bugs and defects.

Automated Unit Testing with Jest: The JavaScript functions utilized within HTML forms for validation underwent automated unit testing using Jest. These tests are run without manual intervention, allowing for efficient regression testing. A comprehensive regression test suite was incrementally developed throughout the program's evolution.

Types of Unit Test Cases: The unit test cases were designed to cover two essential scenarios:
Normal Operation:
This type of test case ensures that the component functions correctly under standard conditions.

Sample Test Case:

```
test('validateCustomer with valid input', () => {
    expect(validateCustomer('John', 'Doe',
'john.doe@example.com')).toBe(true);
});
```

Abnormal Inputs:

This type of test case verifies that the component handles abnormal inputs appropriately without crashing.

Sample Test Case:

```
test('validateCustomer with invalid email format', () => {
    expect(validateCustomer('Jane', 'Smith',
'invalid_email')).toBe(false);
```

```
    expect(global.alert).toHaveBeenCalledWith('THE EMAIL YOU ENTERED
IS NOT A CORRECT ONE. IT MUST BE IN THE FORM OF
"EX12PLE@EX45PLE.COM"');
});
```

Guideline-Based Testing: Guideline-based testing was employed to select test cases, leveraging past experience to identify common programming errors. This approach ensured a robust testing suite that addressed potential pitfalls

b. Release Testing

Cypress End-to-End Testing:

Release testing involves validating the entire system before its release. Cypress, a comprehensive end-to-end testing framework, was employed for this purpose.

Functional Test Suites: Test suites were logically divided between the admin and user sections, considering dependencies between components. For example, a test suite for the customer involved testing functionalities such as logging in, viewing the user webpage, specifying a table reservation, selecting a table, confirming the reservation, and verifying its display.

Performance Analysis: Cypress was instrumental in evaluating the system's performance. The analysis demonstrated that the application met performance requirements even under automated user input conditions.

Conclusion:

The combined use of Jest for automated unit testing during development and Cypress for end-to-end testing during release ensured a comprehensive and robust testing strategy. This approach facilitated the early identification and resolution of bugs, contributing to the overall reliability and performance of the web application for workspace reservation.