**Rental & Service Management System**

**Software Design**

**CSCI-P465/565 (Software Engineering I)**

**Project Team - 19**

**Ashutosh Gupta**

**Bharath Veldi**

**Carter Novak**

**Darrion Shack**

**Nithin Madhusudhana**

**1. Introduction**

Our project approach integrates the MERN stack and Agile methodology. We prioritize in-house development to maintain control over customization and scalability while closely aligning with the unique needs of off-campus students at IUB. Phased delivery ensures essential features are deployed early, and prototypes facilitate user feedback. We remain adaptable by evaluating emerging technologies and deciding on their integration based on alignment with project goals. This approach guarantees a responsive and tailored Rental & Services Management System for international students, enhancing their experience during property rentals and moving-in processes.

**1.1 System Description**

The Rental & Services Management System project aims to ease the moving-in process for international students (and in-state students), inspired by the founder's personal experience. Focused on apartment rentals and associated services, the project addresses critical gaps in existing rental platforms. Specifically designed for students attending the Indiana University of Bloomington, the system offers detailed property listings with essential information like proximity to supermarkets and campus facilities. By centralizing this data, it streamlines the apartment search, eliminating the need for students to navigate multiple platforms. The project also empowers students with tools for informed decision-making, including map views, multimedia features, and user reviews. Additionally, it simplifies access to moving-in services such as transportation and utility setup. Through features like messaging and chat, the system fosters a sense of community, enhancing the overall experience for international students during their crucial transition period.

**1.2 Design Evolution**

This section is intended to document the rationale behind the selected design solution.

**1.2.1 Design Issues**

Desired Operating Environment: The system must operate seamlessly in web environments to cater to users accessing it from different devices and platforms. It must also

Interface Protocols: The system needs to offer a user-friendly web interface with features like search filters, maps, multimedia elements, and interactive communication tools. These features work to provide a better experience for the users to quickly perform the tasks they desire, and with easy understanding of how to do so.

Remote Accessibility: Given the various locations of international students, the system must provide reliable remote accessibility with consistent performance. Map integration must still successfully operate from a distance far from Bloomington.

**1.2.2 Candidate Design Solutions**

Responsive Web Application: Develop a responsive web application using modern web development frameworks like React with backend elements in line with the MERN stack and standards. Backend application may vary slightly to incorporate the most effective and efficient system for this project.

API Integration: Integrate with mapping and location-based services like Google Maps to provide accurate property locations and proximity information.

**1.2.3 Design Solution Rationale**

Responsive Web Application: A responsive web application ensures cross-platform accessibility, allowing users to access the system from various devices. This approach aligns with our goal of making the system accessible to all users.

API Integration: Integrating with mapping services enhances the user experience by providing accurate property locations and proximity information. Having a map is crucial for apartment rentals and other vital services important to living in a campus environment.

**1.3 Design Approach**

**1.3.1 Methods**

Object-Oriented Design: Object-oriented design principles will be followed to model the system components efficiently.

Design Patterns: Common design patterns, such as MVC (Model-View-Controller) and Singleton, will be used to ensure maintainability and extensibility of the codebase. Single-page patterns are also suitable for web applications to avoid waiting for separate pages to load in between actions.

User-Centered Design: User research and usability testing will prioritize the needs and preferences of the end-users, especially international students. By analyzing common designs and implementations for rental services and map integration, we can provide an ideal interface for users to navigate easily.

**1.3.2 Standards**

Web Standards: The web application will conform to web standards, including HTML5, CSS3, and JavaScript best practices.

Data Security Standards: Data encryption and secure authentication mechanisms will be implemented to protect user data. Two-Factor Authentication is implemented to provide a secure login to a user’s account and provide unwanted access.

User Interface Guidelines: The user interface will follow established design principles and usability standards to ensure a consistent and user-friendly experience.

**1.3.3 Tools**

Development Environment: Visual Studio Code and web development tools will be utilized for coding and debugging.

Version Control: Git and GitHub will facilitate version control and collaboration among the team.

Mapping Services: Google Maps or another chosen API will be integrated for mapping and location-based services.

**2. System Architecture**

**2.1 System Design**

The high-level design of our apartment rental service is based on the MERN (MongoDB, Express.js, React, Node.js) stack, integrated with Firebase for secure authentication, including Multi-Factor Authentication (MFA). Here's an overview:

**Client-side** (Frontend):

Developed using React, it provides the user interface for interacting with the application.

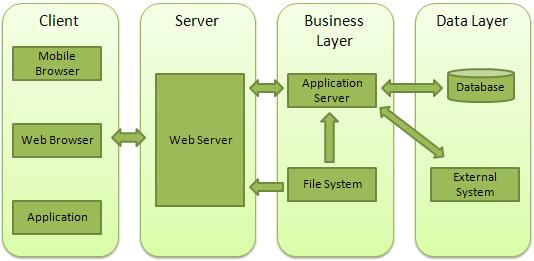
Utilizes various components for listing apartments, user authentication, and managing user profiles.

**Server-side** (Backend):

Built with Node.js and Express.js, it handles requests from the frontend, interacts with the database, and manages external authentication through Firebase.

**Creating Server :**

Using NodeJs and ExpressJS we created a server that handles requests from clients and EnRoute's to appropriate API to perform the action.



**Database:**

MongoDB is used to store apartment listings, user profiles, and transactional data.

Authentication and Security:

Firebase is integrated for user authentication, providing an added layer of security with Multi-Factor Authentication (MFA).

Schema Development and integration:

To store the data, firstly create database and connect it to Atlas to store data in cloud. Second, Create schema’s for the kind of collections we require. Using the schema store the data in backend. Create APIs for incoming requests to perform CRUD operations.

1. Community Schema and API creation.
2. Owners Schema and API’s creation.
3. Renters Schema and API’s creation.
4. Messages Schema and API’s creation.
5. Chats Schema and API’s creation.
6. Chat participants Schema and API’s creation.
7. Maintenance Schema and API’s creation.

A diagram of a process

Description automatically generated

**2.2 External Interfaces**

**Firebase Authentication**: Our system communicates with Firebase's authentication services via REST APIs. Data exchanged includes user credentials, tokens, and authentication states. Timing patterns are synchronous for login, but asynchronous for token refresh.

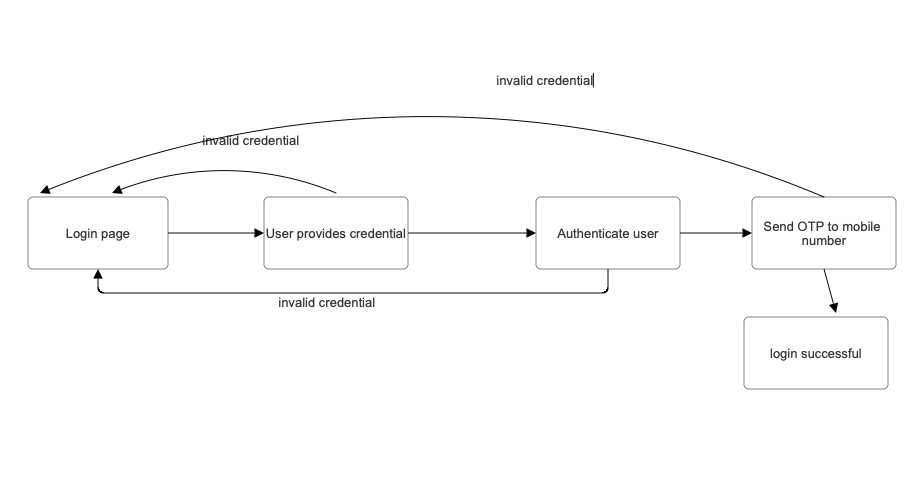
**Geolocation Service**: The system interfaces with a geolocation service to provide location-based search and mapping features. It exchanges data in the form of latitude and longitude coordinates, using REST APIs for location queries.

**Payment Gateway**: For handling payments related to apartment bookings, our system interfaces with a payment gateway service. This involves exchanging payment information in a secure manner using HTTPS and APIs provided by the payment service.

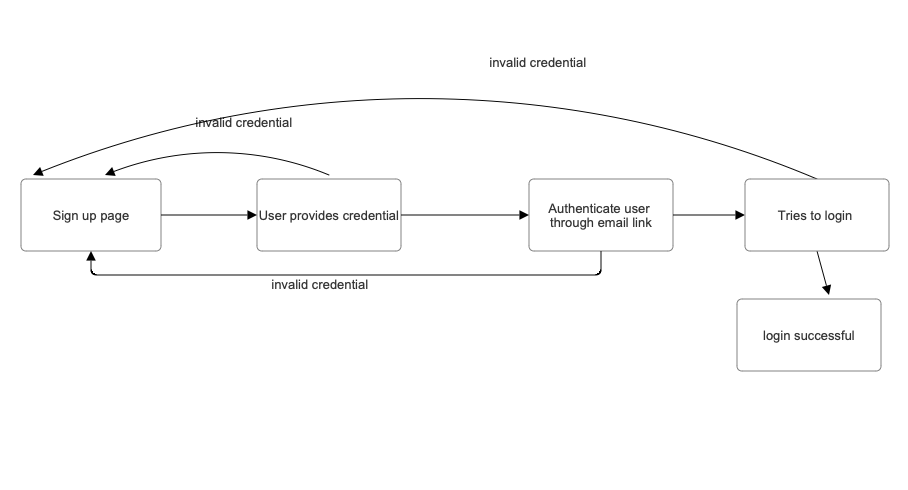
**Email Service**: To send email notifications, the system interfaces with an email service. This includes using SMTP or a similar protocol to transmit emails for booking confirmations, updates, and communication between users and the platform.

Each of these external interfaces follows industry-standard protocols for communication, ensuring secure data exchange and reliable interactions with our apartment rental service. Timing and handshaking patterns may vary depending on the specific external service but are designed to be robust and responsive to user actions within the system.

**3. Component Design**

* **Component Name: Login**
* Component Description:
* The Login component is responsible for authenticating users who wish to access the system. It verifies the user's identity based on their provided credentials (username and password) and grants access if the credentials are valid. If the credentials are invalid, appropriate error messages are generated and returned to the user interface. Upon login MFA is included so that dual security will be provided to the users.
* Responsible Development Team Member: Novak Carter
* Component Diagram:   
  
* Component User Interface:
* - Login Screen: This screen allows users to input their username and password.
* - Error Messages: The component may generate error messages such asInvalid username or password" or "invalid otp" in case of incorrect input or other issues.
* Component Objects:
* - User class:
* - Data Members:
* - `username` (string): Stores the user's username.
* - `password` (string): Stores the user's password.
* - Methods:
* - `authenticate(username, password)`: Verifies the user's credentials by comparing them with stored data.
* - `verifyOTP()`: verify otp input from the user.
* Component Interfaces (Internal and External):
* - External Interface:
* - The component receives user login requests through the user interface.
* - It communicates with a database component to retrieve user credentials.
* - Internal Interface:
* - The component interacts with the User class to perform authentication.
* - It may make operating system calls for tasks like locking user accounts.

Component Error Handling:

* - Error Case 1: Invalid Credentials
* - If the entered username or password is incorrect, the component will return an "Invalid username or password" error message to the user interface.
* - Error Case 2: Invalid otp
* - After a predefined number of failed otp attempts, the component cannot login back and need to start from scratch.
* **Component Name: Sign Up**
* Component Description:The Sign Up component allows users to create new accounts in the system. Users provide their personal information and create a username and password. The component verifies the provided information, creates a new account, and stores it in the database.
* Responsible Development Team Member:Nithin Shastry Madhusudhana
* Component Diagram:
* 
* Component User Interface:
* - Sign-Up Screen:This screen allows users to input their personal information, username, and password.
* - Success Message: Upon successful account creation, a message like "Account created successfully and verify link via email" is displayed.

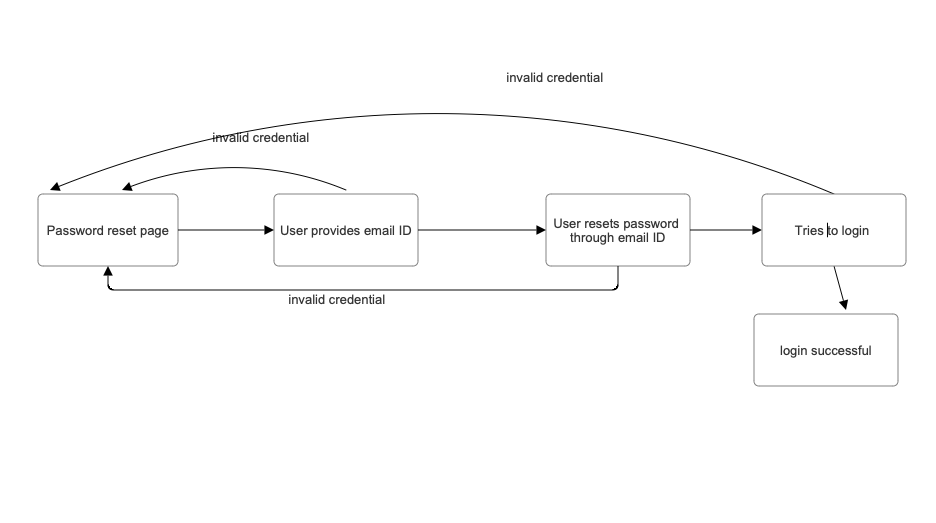
* Component Objects:
* - User class:
* - Data Members:
* - `username` (string): Stores the user's username.
* - `password` (string): Stores the user's password.
* - Methods:
* - `createAccount(username, password)`: Creates a new user account and stores it in the database.

Component Interfaces (Internal and External):

* - External Interface:
* - The component receives sign-up requests through the user interface.
* - It communicates with a database component to store user account information.
* - Internal Interface:
* - The component interacts with the User class to create and store user accounts.

* Component Error Handling:
* - Error Case 1: Existing email id
* - If the chosen email already exists, the component will return an error message like "Email already taken."

* - Error Case 2: Invalid Input
* - If any of the provided information is invalid or incomplete, the component will return an error message asking the user to correct the input.

* **Component Name: Reset Password**
* Component Description:
* The Reset Password component allows users to reset their forgotten passwords. Users provide their username and receive instructions for resetting their password, either through email or other means.Responsible
* Development Team Member: Nithin Shastry Madhusudhana
* Component Diagram:
* 
* Component User Interface:
* - Reset Password Screen: This screen allows users to input their user email.
* - Instructions: After entering the user email, the component provides instructions for resetting the password, which will involve sending an email with a reset link.
* Component Objects:
* - User class:
* - Methods:
* - `resetPassword(username)`: Initiates the password reset process and sends instructions to the user.
* Component Interfaces (Internal and External):
* - External Interface:
* - The component receives reset password requests through the user interface.
* - It may communicate with an email system to send reset instructions.
* - Internal Interface:
* - The component interacts with the User class to initiate the password reset process.
* Component Error Handling:
* - Error Case 1: Communication Failure
* - If there is a failure in sending reset instructions (e.g., email server issues), the component will return an error message indicating the communication problem.

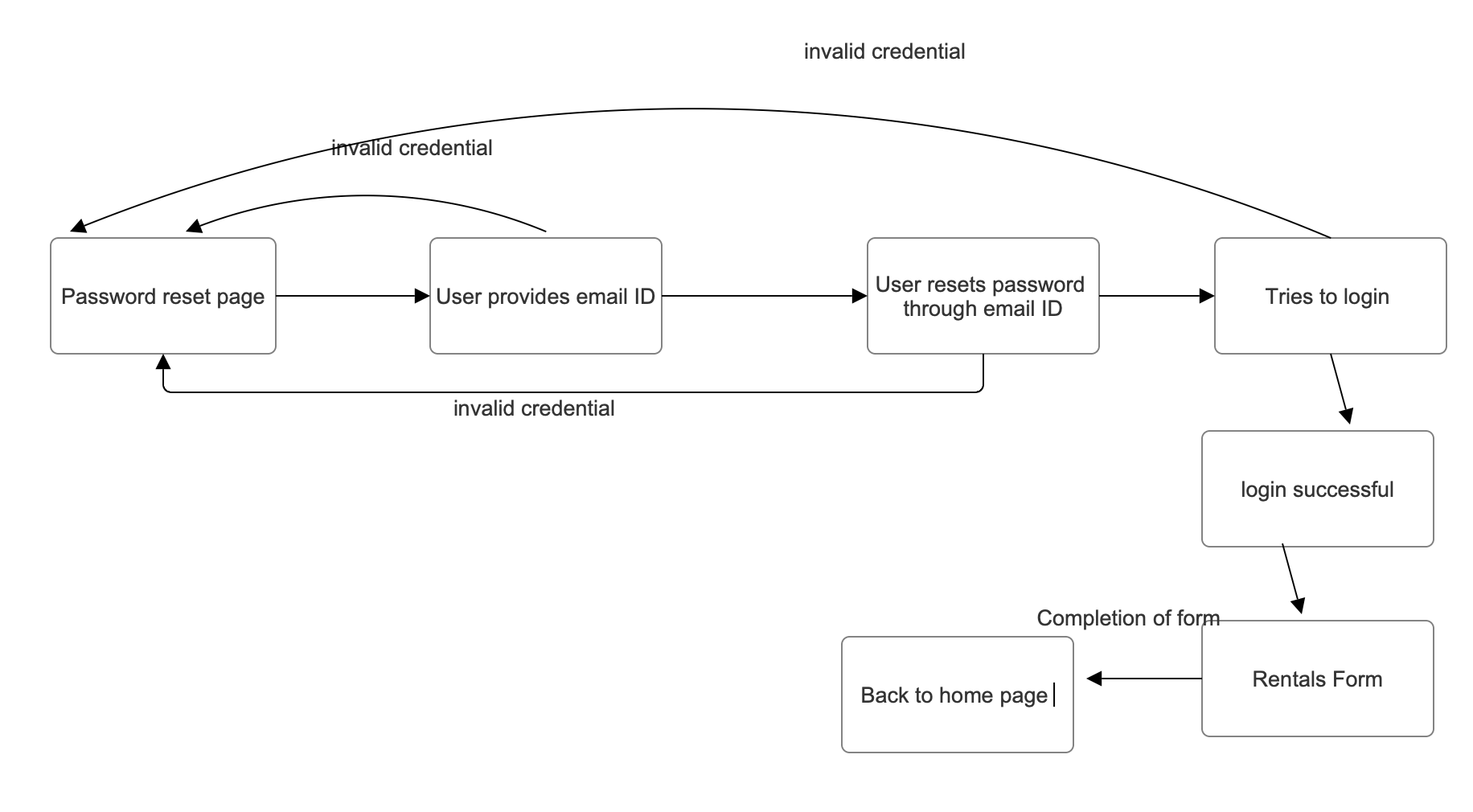
**Component Name: RentalsForm**

Component Description:

The RentalsForm component is responsible for capturing and validating user input related to property rentals. It provides a user interface for users to input details such as full name, email address, age, street address, home type, number of bedrooms (bed), number of bathrooms (bath), land area, apartment condition, home tour link, and contact number. This component ensures the collected data is accurate, and it may interact with other components or services to process and store this information.

Responsible Development Team Member:Nithin Shastry

Component Diagram:



Component User Interface:

- Rental Information Form:This is the user interface that allows users to input information regarding rental properties.

- Validation Messages: If there are any validation errors or issues with the user's input, the component should display appropriate validation messages.

Component Objects:

- RentalForm class:

- Data Members:

- `fullName` (string): Stores the user's full name.

- `email` (string): Stores the user's email address.

- `age` (int): Stores the user's age.

- `streetAddress` (string): Stores the address of the rental property.

- `homeType` (string): Specifies the type of home (e.g., apartment, house).

- `bed` (int): Specifies the number of bedrooms.

- `bath` (int): Specifies the number of bathrooms.

- `landArea` (float): Stores the land area of the property.

- `apartmentCondition` (string): Describes the condition of the apartment.

- `homeTourLink` (string): Stores the link to the virtual home tour.

- `contactNumber` (string): Stores the user's contact number.

- Methods:

- `validateData()`: Validates the user's input to ensure it meets specified criteria.

- `submitData()`: Sends the validated data to a storage or processing component.

Component Interfaces (Internal and External):

- External Interface:

- The component receives user input through the user interface.

- It may communicate with a database or external data storage service to store rental property information.

- Internal Interface:

- The component interacts with the RentalForm class to validate and submit data.

- It may make operating system calls for data storage and retrieval.

Component Error Handling:

- Error Case 1: Validation Failures

- If the user's input does not meet the validation criteria (e.g., invalid email format, missing required fields), the component should display appropriate validation messages.

- Error Case 2: Data Submission Failure

- If there are issues with storing the rental property information (e.g., database connection problems), the component should handle and report the error appropriately.

-Error Case 3: Invalid Data Types

- If the data received from the user interface is not of the expected data types, the component should validate and handle this case by displaying error messages.

**Component Name: Search**

Component Description:

The Search component provides users with the ability to search for specific keywords or phrases within the system's data. It enables users to filter and locate relevant apartment cards based on their search criteria. The Search component enhances the user's experience by helping them quickly find apartments that match their preferences.

Responsible Development Team Member: Nithin

Component User Interface:

- Search Input: Users can enter keywords or phrases in a search input field to define their search criteria.

- Search Filters: Users can apply filters to narrow down search results, such as location, price range, number of bedrooms, and more.

- Search Results: The component displays a list of apartment cards that match the search criteria.

Component Objects:

- SearchService class:

- Data Members:

- `searchQuery` (string): Stores the user's search query.

- `filters` (object): Stores the selected filters for the search.

- `searchResults` (array): Holds the matching apartment cards.

- Methods:

- `performSearch(query, filters)`: Initiates the search based on the provided query and filters.

- `applyFilters(filters)`: Filters the search results based on user-selected criteria.

- `resetFilters()`: Resets all filters to their default values.

- `displayResults()`: Renders the search results on the user interface.

Component Interfaces (Internal and External):

- External Interface:

- The component interacts with the user interface, allowing users to input search queries and select filters.

- It communicates with the database component to retrieve apartment data and filter results.

- Internal Interface:

- The component interacts with the SearchService class to perform search operations.

- It may utilize browser APIs for efficient client-side search functionality.

Component Error Handling:

- Error Case 1: No Search Results

- If the search query and filters do not yield any matching apartment cards, the component will return a message like "No results found. Try refining your search criteria."

- Error Case 2: Invalid Filter Selection

- If users select filters that do not correspond to available options, the component will provide an error message such as "Invalid filter selection."

- Error Case 3: Database Connectivity Issues

- If there are problems connecting to the database to retrieve apartment data, the component will display an error message like "Database connection error. Please try again later."

- Error Case 4: Unexpected Errors

- In the event of unexpected errors or issues, the component will return a generic error message and log the error for debugging purposes.

- Error Case 5: Empty Search Query

- If users attempt a search without entering any search criteria, the component will prompt them with a message like "Please enter a search query."

Component Usage:

- Users can initiate a search by entering keywords, phrases, and selecting filters to find the ideal apartment.

- The Search component plays a crucial role in improving the user experience by streamlining the apartment search process.

**Component Name: Utilities**

Component Description:

The Utilities component is a comprehensive information hub that offers users insights into various financial and commuting options. It empowers users with knowledge about bank account choices, credit card options, and commuting alternatives. The component acts as a valuable resource, guiding users to make informed decisions regarding their finances and transportation.

Responsible Development Team Member: nithin

Component User Interface:

- Bank Accounts: Users can access information about different types of bank accounts, their features, and eligibility criteria.

- Credit Cards: Users can explore a range of credit card offerings, their benefits, and application requirements.

- Commuting Options: Users can discover various transportation alternatives, including buses, ridesharing services, and bike programs.

Component Objects:

- UtilitiesService class:

- Data Members:

- `bankAccounts` (array): Stores information about available bank accounts.

- `creditCards` (array): Contains details on different credit card options.

- `commutingOptions` (array): Provides information on various commuting alternatives.

- Methods:

- `fetchBankAccounts()`: Retrieves data about available bank accounts from the database.

- `fetchCreditCards()`: Retrieves information about credit card options from the database.

- `fetchCommutingOptions()`: Fetches data on commuting choices from external sources or APIs.

- `displayDetails()`: Renders the information on the user interface for users to access.

Component Interfaces (Internal and External):

- External Interface:

- The component communicates with external data sources to retrieve information about bank accounts, credit cards, and commuting options.

- It interacts with the user interface, providing users with easy access to these details.

- Internal Interface:

- The component utilizes the UtilitiesService class to fetch, organize, and display data.

- It may employ data caching mechanisms to improve data retrieval efficiency.

Component Error Handling:

- Error Case 1: Data Retrieval Failure

- If there is a problem retrieving data from external sources or the database, the component will display an error message like "Data retrieval failed. Please try again later."

- Error Case 2: Empty Data

- If no information is available for a specific category (e.g., no bank account details found), the component will inform users with a message such as "No data available at the moment."

- Error Case 3: User Interface Error

- In case of unexpected errors within the user interface, the component will provide a generic error message and log the issue for further investigation.

Component Usage:

- Users can explore bank account options, compare credit cards, and discover convenient commuting choices.

- The Utilities component empowers users to make informed decisions regarding their financial and transportation needs.

- It serves as a one-stop information center, providing valuable insights for users to optimize their financial and commuting experiences.

**Component Name: User Profile**

* Component Description:
  + The UserProfile component is a React-based user profile page that displays and allows users to manage their personal information, including their name, email, phone number, address, date of birth, and password. It also provides access to lease information and allows users to view and manage maintenance requests. Users can switch between viewing and editing their personal information, and they can post new maintenance requests.
* Responsible Development Team Member: Ashutosh
* Component User Interface:
  + User Profile Display: Users can view their profile information, including name, email, phone number, address, date of birth, lease details, and maintenance requests.
  + User Profile Editing: Users can edit their profile information, including name, email, phone number, address, date of birth, and lease details.
  + Maintenance Requests: Users can view a list of maintenance requests, including headings, descriptions, and statuses.
  + Adding Maintenance Requests: Users can add new maintenance requests, providing a heading and description.
  + Password Management: Users can change their password by entering their current password and a new password.

* Component Objects:
  + userProfile: A state object that holds the user's profile data, including personal information, lease details, and maintenance requests.
  + isEditing: A state variable that determines whether the user is in edit mode for personal information.
  + isCreatingTicket: A state variable for controlling the addition of new maintenance requests.
  + newTicket: A state object for storing data about a new maintenance request.
  + isAddingTicket: A state variable for controlling the addition of new maintenance requests.
  + isEditingPassword: A state variable that controls whether the user is in edit mode for changing their password.
  + newPassword: A state variable that holds the new password entered by the user.
  + currentPassword: A state variable that holds the current password entered by the user.
  + passwordMatch: A state variable that checks whether the current password matches the stored password.

* Component Interfaces (Internal and External):

* + Internal Interface: The UserProfile component interacts internally with state variables and functions to manage user data and interface state. It allows users to edit personal information and change their password.
  + External Interface: The UserProfile component can be used as part of a larger application or website. It may receive data for a user's profile as props and communicate changes to the parent component or server when the user saves edits or adds maintenance requests.

* Component Error Handling:
  + Invalid Email:
    - Error Handling: When the user attempts to save changes with an invalid email format, an alert is shown.
    - Error Case: If the user provides an email address that doesn't match the required email format (e.g., "example.com" instead of "[user@example.com](mailto:user@example.com)"), an error alert is displayed.
  + Invalid Phone Number:
    - Error Handling: Similar to the email validation, when the user attempts to save changes with an invalid phone number format, an alert is displayed.
    - Error Case: If the user enters a phone number that doesn't match the expected format (e.g., "1234" instead of "123-456-7890"), an error alert is shown.
  + Incorrect Current Password:
    - Error Handling: This case is not yet implemented in the component, but it can be added as a new error case.
    - Error Case: If the user attempts to change their password and provides an incorrect current password, an error message should be displayed, indicating that the current password is incorrect.
  + Empty Fields on Maintenance Request:
    - Error Handling: The component does not currently have error handling for empty fields when adding a new maintenance request. It should validate and alert the user if they try to post a maintenance request with empty fields.
    - Error Case: If the user tries to post a maintenance request without specifying the heading or description, an error message should be shown, prompting them to fill in these fields.

* Component Usage:
  + The UserProfile component is used within an application or website to provide users with a dedicated profile page.
  + It can be integrated into property management or tenant-facing applications to allow users to view and edit their personal information and manage maintenance requests.
  + The component can be used in a secured environment where user data is managed and validated appropriately.

**Component Name: Chat**

* Component Description:
  + The Chat component is a React-based user chat page that displays and allows users to message and chat with other users on the website. Users can switch between multiple chats.
* Responsible Development Team Member: Darrion
* Component User Interface:
  + Chats: Users can see other chats they have previously engaged in, allowing for quickly following up on previous messages.
  + Search: Users can search for other users to chat with or find an old chat.
  + Chat Box: User can send messages to the currently selected chat user in real-time.
* Component Objects:
  + user: State object that holds the current user and user uid.
  + chat: A state object that holds the currently selected chat.
  + chats: A state object that holds a history of user’s chats.
  + messages: State object with all messages sent in current chat.
* Component Interfaces (Internal and External):

* + Internal Interface: The Chat component interacts internally with state variables and functions to manage user data and interface state. It allows users to send messages.
  + External Interface: The Chat component can be used as part of a larger application or website. It may receive data for a user's profile and communicate changes to the firebase firestore when the user sends or receives messages.

* Component Error Handling:
  + Invalid Search:
    - Error Handling: When the user attempts to search a user that doesnt exist, an error is shown.
    - Error Case: If the user provides an email address that doesn't match any emails in the firestore database

* Component Usage:
  + The Chat component is used within an application or website to provide users with a dedicated chat page.
  + It can be integrated into any page for chatting as the component handles everything it needs on its own.

Component Name: UserView

Component Description:

The UserView component is a React-based component that dynamically adjusts the user interface based on the role of the signed-in user. It provides different views and functionalities for property owners and regular users.

Responsible Development Team Member: Olivia

Component User Interface:

1. Property Owner View:

- Rental Form: Property owners have access to a rental form, allowing them to input and manage information related to their property for rent.

- Property Management: Property owners can view and manage their property listings.

2. User View:

- General User Dashboard: Regular users see a dashboard with features such as viewing available properties, messaging property owners, and managing their profile.

- Property Listing: Users can browse through available properties but do not have access to the rental form.

Component Objects:

- userRole:

- State object that holds information about the role of the currently signed-in user (e.g., "propertyOwner" or "user").

Component Interfaces (Internal and External):

- Internal Interface:

- The UserView component interacts internally with the state variable "userRole" to dynamically render the appropriate view based on the user's role.

- External Interface:

- The UserView component can be used within a larger application or website. It may receive user authentication data and communicate with other components to enable or disable certain features based on the user's role.

Component Error Handling:

- Invalid Role:

- Error Handling: If an invalid or unexpected role is provided, the component displays an error message.

- Error Case: If the user role is not recognized as either "propertyOwner" or "user."

Component Usage:

The UserView component is used within an application or website to customize the user experience based on the user's role. It ensures that property owners and regular users have access to the relevant features and views according to their designated roles.

Component Name: Payment Gateway

Component Description:

The Payment Gateway component facilitates secure and efficient payment processing for transactions within the system. It leverages the services provided by the Stripe payment gateway to handle payment operations securely, ensuring reliability and compliance with industry standards.

Responsible Development Team Member: [Insert Name]

Component User Interface:

- Payment Form Interface: Allows users to input payment details such as card information, billing address, and transaction amount.

- Confirmation Interface: Displays transaction confirmation messages or prompts in response to successful or failed transactions.

Component Objects:

- PaymentHandler Class:

- Data Members:

- `cardNumber` (string): Stores the user's card number for payment.

- `expiryDate` (string): Stores the expiry date of the card.

- `cvc` (string): Stores the Card Verification Code for validation.

- `billingAddress` (string): Stores the user's billing address for the transaction.

- `amount` (float): Stores the transaction amount.

- Methods:

- `processPayment(cardNumber, expiryDate, cvc, billingAddress, amount)`: Initiates the payment transaction by communicating with the Stripe API and processing the payment details.

Component Interfaces (Internal and External):

- External Interface:

- The component receives payment requests through the system's user interface or APIs.

- It communicates with the Stripe API to authorize and process payments securely.

- Internal Interface:

- The component interacts with the PaymentHandler class to orchestrate payment processing.

- It may utilize encryption or tokenization methods to secure sensitive payment data.

Component Error Handling:

- Error Case 1: Payment Authorization Failure:

- If the payment authorization fails due to invalid card details or insufficient funds, the component generates an error message and communicates it back to the user interface or system logs.

- Error Case 2: Network or Connectivity Issues:

- In case of network disruptions or communication errors with the Stripe API, the component logs the issue, retries the transaction, and notifies the user about the issue if needed.

Component Usage

Customers can utilize payment gateway to pay rent on regular basis based on the rent customers have.

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| Frontend Components | 10/02/2023 |  |
| Server and db creation with API’s | 10/16/2023 |  |
| Chat component | 10/30/23 | Added description for chat component |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |