



# SOFTWARE PROJECT REPORT

## **PREPARED BY**

Yashwanth Manivannan(1801202)

Ravindra Singh Sisodiya(1801138)



# EXECUTIVE SUMMARY

With the increase in digitization of society in general from digital marketing to digital payments, the world has moved ahead in terms of technology. Even with such abundance, technology is still seen as a tool of the affluent. The average business still hesitates to invest in technology to increase the efficiency of their businesses and does not understand the usefulness of data-driven analytics.

We wanted to create a product that can be operated with relative ease to keep track of business. The simple modular nature of our proposed product means it can be adjusted to every need.

It used to be that the key to reaping the benefits of an effective property management software system was to select the right one for your property. It was critical that you knew exactly what the technology was, and why it was important for you to implement it at your business. These days, every person can find different options for hotel reservation software free on the internet; however, one has to judge the solutions with the quantum and quality of features being provided.

Solutions that are able to scale themselves to fit and adapt to your business are the new way forward, and we built Innkeep in that spirit!

# Project Overview

The purpose of developing this adaptable variant of a “Property Management System” is to reduce the workload and simplify day-to-day tasks of the business administrators and improve efficiency and coordination. It also aims to eliminate completely the paperwork that has become the primary form of record-keeping in many small-scale businesses, which is not feasible in the long run.

It will allow administrators to monitor and keep track of day-to-day issues raised by inhabitants and address them reliably and efficiently by providing a comprehensive dashboard with an integrated task-assigning module. In addition, we provide automated logging in of data for existing manual paper entries.

## Product Context

The concept behind Innkeep is to build an all-purpose software that can be used to manage businesses or property alike. Although the initial idea was to build a hostel-specific software, we decided to instead make it as general-purpose as possible. This software can be adapted to manage any business that lends out a resource as a service, such as restaurants, hotels or car rentals.

The function of Innkeep is to keep a dynamic overview of a business. It allows for constant monitoring of resources and dynamic responses to customers to improve and increase the volume of customer relations. The swiftness with which issues can be resolved allows the business to increase customer satisfaction, which generates potential for an increase in future engagements for the business.

This product aims to bring together all possible features, giving flexibility and facilitating the management process. Innkeep’s main focus is to be robust in operation and user-friendly in usage.

1. Platforms are available: a web application and a mobile application which is compatible with the Android operating system and further will be extended to iOS.
2. Sign in options: username & password; or email account.

## User Characteristics

### 1. Admin

- the owner or manager of the business running this product
- Is able to see booked resources, can adjust price, modify reservations

- Is able to Add/Remove resources or possible users of the system
- Is able to observe Statistics - Inventory

## 2. Guest

- Can make reservations and check rates
- During the stay, can view if the room is ready by janitorial staff
- Can modify their requests and edit their credentials

## 3. Staff

- Will receive tasks allotted by the admin
- Can update the progress of the task

## Assumptions

The following assumptions have been made while making this project:

1. All users have a basic knowledge of English language
2. All users have a basic knowledge of computer and smartphone usage
3. Admin of the hotel has a basic knowledge of how to use the system due to previous experience with other systems
4. Client's business op is equipped with PC/Laptop/Tablet or a mobile phone
5. Client must have internet connection all the time

## Dependencies

As it is a web-based application, it is always dependent on an internet connection. A connection is required to send commands and receive answers, usually in the form of a JSON object.

# Requirements

## Functional Requirements

Functional requirements are tabulated as follows accompanied with a priority level.

- Priority 1 – The requirement is a “must have” as outlined by policy/law
- Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
- Priority 3 – The requirement is a “nice to have” which may include new functionality

| Functional Requirement  | Priority |
|---|----------|
| Administration can create rooms/flats.  | 1        |
| Every room will have a unique ID  | 1        |
| Administration can check in/out residents for a corresponding room/flat.  | 2        |
| Administration can list the services available for the residents.   | 1        |
| Admin can also create a level of services based on their availability in a particular room/flat.                | 1        |
| Residents can raise requests for certain tasks (services) such as cleaning, repair, equipment malfunction, etc. | 1        |
| Residents can be notified of any warnings or precautions to be taken within the property.                       | 3        |
| Residents can safely receive courier/post from the security desk without risk of theft.                         | 3        |
| Administrators can manage property easily from a single dashboard.  | 1        |
| Records leave applications and waiver applications, and tracks the status of such applications.                 | 3        |
| Access to the database to details of residents to be contacted, checked as per administration's need.           | 3        |
| Root admin can track all the services/transactions/room occupancy.  | 1        |

|   |   |
|---|---|
| Residents can raise complaints.   | 1 |
| Every complaint can be tracked and updated.   | 1 |
| Services can be assigned; the assignee will be notified about the service through a notification. | 1 |
| Admin/resident can also upload the required documents at the time of check in.                    | 3 |
| Every feature will be anchored by a room number.  | 2 |
| Records transaction of key submissions quickly and neatly   | 3 |

## Nonfunctional Requirements

### UI Requirements

- A simple, responsive and fast system
- Web app, consistent in all interfacing screens or devices
- Details of anything will be displayed quickly
- Flexible navigation to and from displayed panels or pages

### Usability Requirements

The most important commands for each type of interface will be visible at first view so the software can be easily accessed. It will be designed in a way that allows modifications to be made, as it has to be updated time after time to fulfill management requirements and to manage possible error occurrences.

### Learnability

- Our product is user-friendly—everyone can easily learn the commands following the guidelines provided by us.
- The software is in English. It can be understood by someone with basic knowledge of English since every functionality will be graphically shown and minimal words are required to understand the working.

## **Availability**

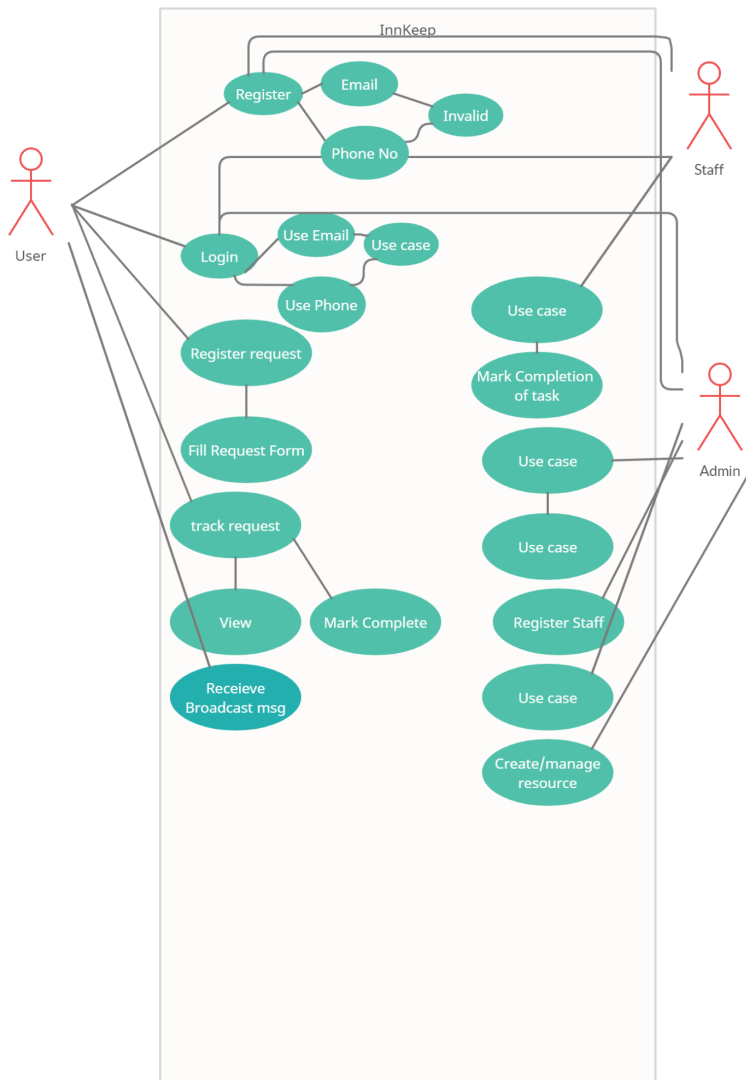
- The website will need to be available all the time, everyday, 24/7. It will have a high availability to achieve the highest possible percentage of up-time
- Even though it is in English, the system can be used world-wide as it is a web application
- Our product will have a downtime as minimal as possible as long as the software will be used with reliable web browsers
- The mobile application will be available for both Android and iOS.

## **Portability**

The web application can be accessed in any platform and browser as long as user has an Internet connection.

# Software analysis and Design

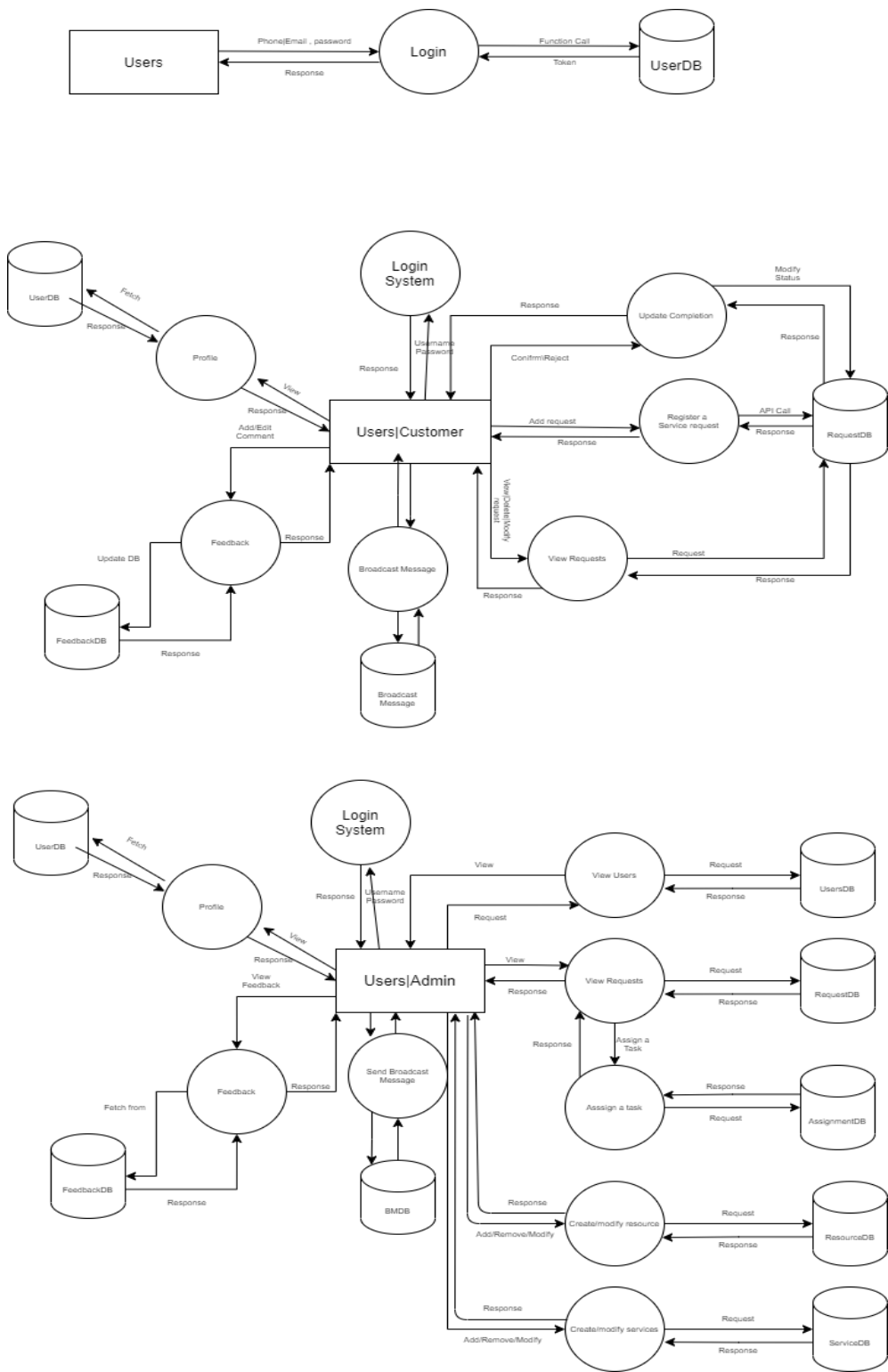
## Use-Case Diagram



The diagram demonstrates the possible use cases of the application between the Guest, Admin and Staff. It helped us keep in mind what scenarios we need to design the software around and to keep everything coherent.

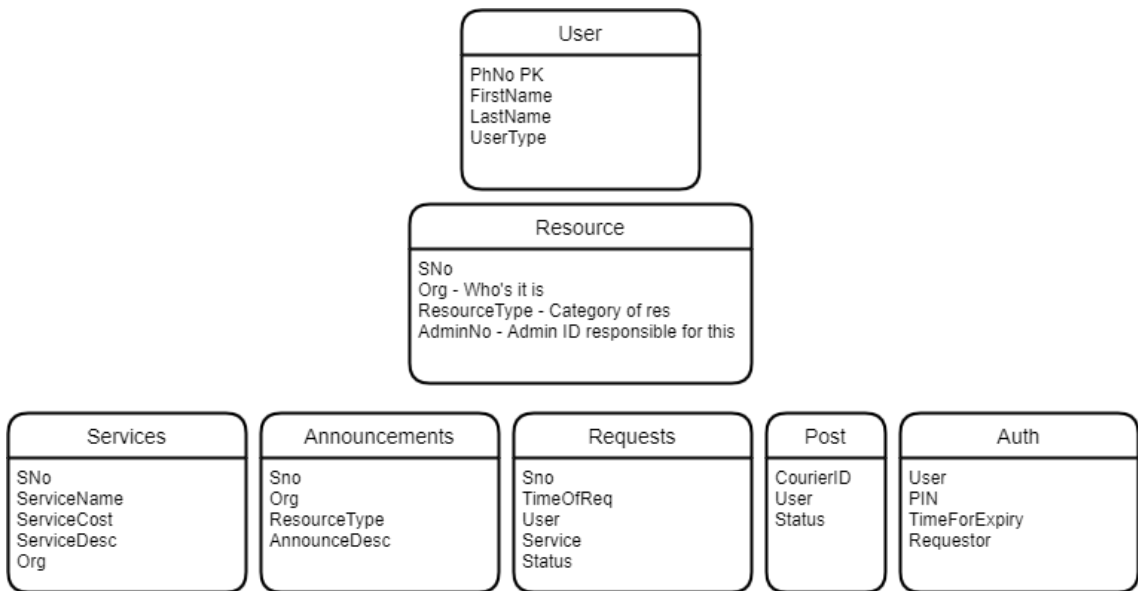


Data-Flow Diagram



ER Diagram

Even though we used a NoSQL database for our backend operation, we thought it would prove useful to plan out what data we were going to store, and how, beforehand. This was useful while creating data models and nothing to figure it out while building the application.



# Implementation

## Project Setup

A basic app was set up and pushed into GitHub to use as version control, so both developers could work on the project and collaborate.

After creating a basic app, we hosted it on Heroku. Then, we set up GitHub actions for continuous deployment of the application to the Heroku server so that the application auto deployed with every commit to the repository. After that, Google's Firebase service was integrated to serve many of the application's backend needs.

We integrated ESLint, a linting tool to check all code being pushed into the repository; it is standard and ensures the quality of the code.

The application is built on the Vue.js JavaScript framework. We chose Vue.js for its quick-form coding. The framework was built to enable quick prototyping and testing which enabled us to build our modules quickly. We chose this framework over more popular frameworks like Angular or React due to our internal assessment of the projected size of the project and our requirement to have it also double as a mobile application.

We used Vue-CLI's development server for development on local, which is fast compared to the node server since we didn't need to build every time for production.

Hot module replacement (HMR) was enabled using Vue-CLI which watches the live changes in the files and updates the application automatically.

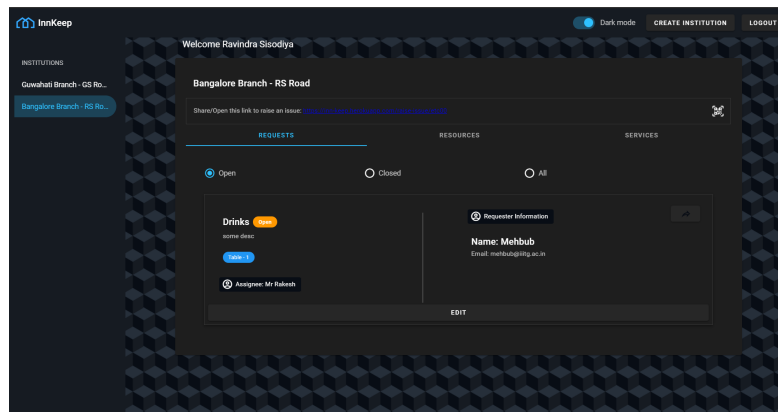
In addition, we used Vuex store (which uses FLUX architecture internally) for centralized data management, and Vue-router for managing the dynamic application routes.

## First Steps

The first point of contact for the customer would be the authentication page. We implemented an email-password sign in and sign up page. The authentication was handled by FirebaseAuth, which encrypted passwords with multiple encryption algorithms ensuring data safety.

## Dashboard

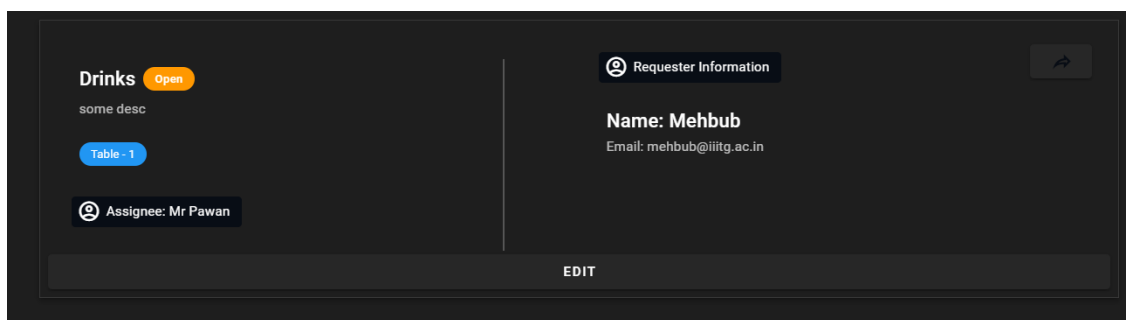
This is the crux of the application. Several actions can be done from here.



As seen from the screenshot, the admin can navigate through the application from a single screen and access all the details they need. They can add, remove and modify any elements or aspect of their business with a few easy clicks.

## Requests Tab

The requests tab keeps a real-time list of requests generated by guests. It allows for the admin to update the request when it's done. They can further assign an employee to address the request. They can also see who has requested it and this data is stored on record indefinitely.



Each item is furnished with details that might be relevant to carrying out the request and tracing the history.

## Services Tab

The services tab is used to create new services—in this case, a menu. Guests can choose strictly only out of the service listed in this. Adding an element or removing one immediately changes the menu dynamically.

Bangalore Branch - RS Road

Share/Open this link to raise an issue: <https://inn-keep.herokuapp.com/raise-issue/etc00>

REQUESTS RESOURCES SERVICES

ADD NEW SERVICE

**New Service**

Title\*

Description

Food  
Price 300

Drinks  
Price 270

CLOSE SAVE

## Filing a request

**Add Request**

Resource\*

Service\*

Name

Email

Request Description

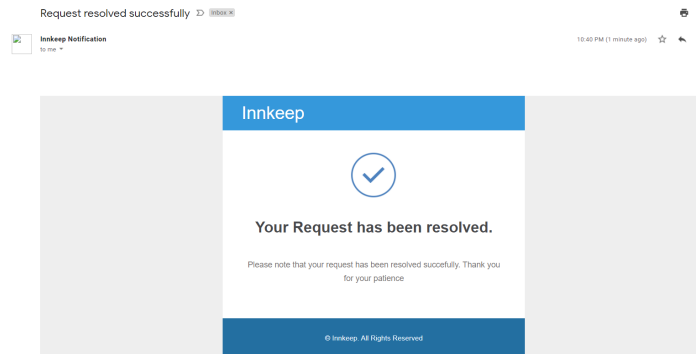
0

SUBMIT

Guests are presented with this form. They can access this through a static QR code that is presented to the Admin when they create a new institution. By filling out this form, they will receive a tracking code to the person's email, through which they can track the status of their requests. The mailing mechanism will be explained in its own section.

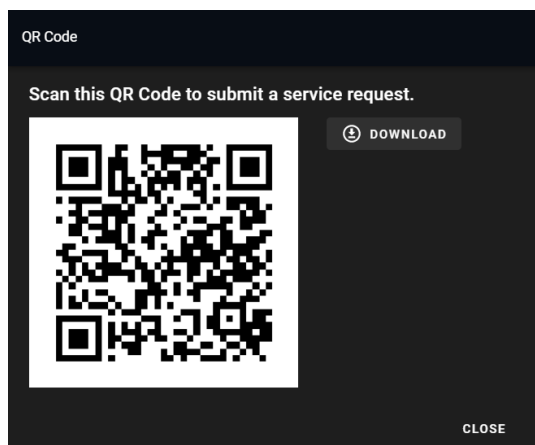
## Mailing System

To send email to guests/users, we built our very own mailing framework through which we can send HTML-styled email to users in real time. By taking advantage of GMail's pre-existing mailing infrastructure, we can send mails with HTML templates to our users with minimal delay.



The API is used twice: when creating a request and when updating a request. We wrote and styled our very own email template to make our mails look aesthetic. In the case of an undelivered email, the failure is processed safely by rejecting the promise and then retrying after a small delay.

## QR Code




The digital QR code is generated out of a unique link for each occurrence of an institution. This was done to greatly increase ease of use for the customer. They can scan this code with their mobile phones and immediately be redirected to the Request filing form.

# Testing

## Unit Testing

This unit test case snippet is used to validate if the text fields are taking in valid inputs. It runs them against a large set of inputs and checks the outputs to make sure they are consistent.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a JavaScript unit test snippet using Jest and Vue Test Utils. The code defines a test case 'should emit the updated value', sets a dummy name, creates a wrapper, sets the input value, and asserts that an 'onInput' event has been emitted.

```
test('should emit the updated value', async function ()
{ let name = 'DUMMY NAME'
  createWrapper()

  // set input value
  await findNameInput().setValue(name)
  await wrapper.vm.$nextTick()

  // assert event has been emitted
  expect(onInput).toHaveBeenCalled()
});
```

This unit tests a Vuex action which, makes a Firebase API call (an async action). The unit is run against a set of positive and negative test cases (payloads). It also tests the different kinds of responses returned by the call. The dummy responses include a 200 status response and a 500 status response. MockAdapter is used to mock the async request, so that the test doesn't make any network request while executing.

The test asserts the promise payload return by the login action.

```
import Vue from 'vue'
import Vuex from "vuex";
import AuthStore from "@admin/modules/order/store/AuthStore";

Vue.use(Vuex)

export const MOCK_STORE = (
  mockState = {},
  mockGetters={},
  mockMutations={},
  mockActions={}
) => {
  return new Vuex.Store({
    modules: {
      OrderStore: Object.assign(AuthStore, {
        state: Object.assign(AuthStore.state, mockState),
        getters: Object.assign(AuthStore.getters, mockGetters),
        mutations: Object.assign(AuthStore.mutations, mockMutations),
        actions: Object.assign(AuthStore.actions, mockActions)
      })
    }
  })
}
```

Here is a test snippet which is testing the GreetingMessage.vue component. It shallow-mounts the component and provides the respective props which are then tested against some set of positive and negative tests. The tests assert the title rendered inside the component.



```

describe('Auth Actions', () => {
  let mockedState;
  let mock;

  beforeEach(() => {
    mockedState = initialState();
    mock = new MockAdapter();
  });

  describe('login', () => {
    it('success', (done) => {
      const data = {
        email: '',
        password: {}
      }
      mock($firebase.login).replyOnce(200, {})
      testAction( actions.login, data, mockedState, [], [], done);
    })

    it('failure', (done) => {
      const data = {
        email: '',
        password: {}
      }
      mock($firebase.login).replyOnce(500, {})
      testAction( actions.login, data, mockedState, [], [], done);
    })
  })
})

```

The below code snippet mocks the Vuex store which is used for testing the components using the \$store option. The created mock store can be integrated with a component while mounting and should be removed after destroying a component. The mock Vue store is being used throughout the unit tests of the entire application. It takes in the default state, getters, mutations and actions as the parameters, which should be passed based on the module where the mocked store is being used.

```

import GreetingMessage from "@/components/GreetingMessage.vue";
import {shallowMount} from "@vue/test-utils";

describe('GreetingMessage', () => {
  let wrapper = null;
  let props = {
    name: '__test__',
    time: "09:43:23",
  }
  const createWrapper = function(_props = props){
    wrapper = shallowMount(GreetingMessage, {
      propsData: _props
    })
  }
  afterEach(() => {
    if(wrapper){
      wrapper.destroy();
      wrapper = null;
    }
  })
  const getTitle = () => wrapper.find(".b-main-message")
  test('renders the greeting message', () => {
    createWrapper();
    expect(getTitle().exists()).toBe(true);
    expect(getTitle().text()).toBe(`${props.name} (${props.time})`)
  })
})

```

## Manual Testing

Two test account were created to test the entire flow of the application:

### Restaurant Account (BBQ)

- A dummy BBQ center - GS Road was created.
- 50 tables (resources) were registered on the application.
- A food menu was registered which includes the main course, complimentary drink, etc.
- The request submission link and a QR code was provided for submitting the requests, which could be tracked in the requests section in the dashboard.
- Some dummy requests were created, which were then assigned to the waiters responsible for the corresponding orders.
- The requests were updated to resolve after completion.

- Original mail IDs were added in the requests to test the mail feature, and the mails were working properly and were getting triggered on request creation and request updation.
- Tables were marked unavailable after a user submitted a request for that table. This was done to track available tables internally.
- The open URL to track the request was tested and the request status was getting updated properly on the URL provided in the mail.

### **A Residential Institute (IIIT Guwahati)**

- Two hostels were created: Boys' Hostel, Girls' hostel
- Rooms were registered for the boys' hostel and for the girls' hostel
- Services such as floor cleaning, washroom cleaning, etc., were added
- A link and the QR code were generated to register the requests
- Some dummy requests were registered
- The requests were assigned to the corresponding staff by the caretaker
- The requests were updated by the caretaker on resolution
- Mails were getting triggered on request creation and completion

Taking the account through the complete process from sign up to the testing every feature ensured that none of the features were breaking in production and that they were all working properly.

We also ran the tests in various devices and resolutions to make sure the UI did not vary incorrectly. This process was repeated with varying the inputs and corner case inputs to check for integrity of the application. We employed various techniques to test input validation, including bug checks.

## Future Scope and Work

The following features have been planned to be added to the project in time:

### Dashboard Analytics



We aim to include a dashboard analytic deck to the application to display useful information at the top. A primitive version has been implemented for now but needs to be improved.

### Secure Take-Away System

The app will also need a takeaway system that will monitor and authenticate physical transactions so that it can be logged in to the system and archived. It also plays a role in a sort of KYC verification so there aren't misplaced orders/transactions.