TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report

On

The Dorm Den: A MERN Stack Hostel Web Application

Submitted By:

Arahanta Pokhrel (THA076BCT009)

Bikrant Bidari (THA076BCT013)

Sameer Shrestha (THA076BCT039)

Sudeep Kaucha (THA076BCT044)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March 2023

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report

On

The Dorm Den: A MERN Stack Hostel Web Application

Submitted By:

Arahanta Pokhrel (THA076BCT009)

Bikrant Bidari (THA076BCT013)

Sameer Shrestha (THA076BCT039)

Sudeep Kaucha (THA076BCT044)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor’s Degree in Computer Technology.

Under the Supervision of

Er. Praches Acharya

March, 2023

DECLARATION

We hereby declare that the report of the project entitled **The Dorm Den** which is being submitted to the **Department of Electronics and Computer Engineering IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Engineering**, is a bonafide report of the work conducted by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Arahanta Pokhrel (THA076/BCT/009) ______________________

Bikrant Bidari (THA076/BCT/013) ______________________

Sameer Shrestha (THA076BCT039) ______________________

Sudeep Kaucha (THA076CT044) ______________________

Date: March, 2023

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to **the Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a minor project work entitled "**The Dorm Den**" submitted by **Arahanta Pokhrel, Bikrant Bidari, Sameer Shrestha** and **Sudeep Kaucha** in partial fulfillment for the award of Bachelor's degree in Computer Engineering. The Project was conducted under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfillment of Bachelor's degree of Computer Engineering .

Project Supervisor
Er. Praches Acharya
Department of Electronics and Computer Engineering, Thapathali Campus

External Examiner

Project Coordinator
Er. Dinesh Baniya Kshatri
Department of Electronics and Computer Engineering, Thapathali Campus

Head of Department
Er. Kiran Chandra Dahal
Department if Electronics and Computer Engineering, Thapathali Campus

March, 2023

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to the Institute of Engineering, Tribhuvan University for the inclusion of minor project during Bachelor's in Computer Technology.

We would like to extend our sincerest gratitude to Mr. Praches Acharya for unwavering support and guidance throughout our project. His invaluable insights and direction have been instrumental in helping us do the project successfully. We are also thankful to the department for their guidance and support throughout this project. Their valuable insights and feedback were instrumental in helping us to complete this report to the best of our abilities.

Finally, we would like to acknowledge the support and encouragement of our family and friends, who have always believed in us and supported us in some tough decisions along the way.

Thank you all for your contributions to this project.

Sincerely,

Arahanta Pokhrel (THA076BCT009)

Bikrant Bidari (THA076BCT0013)

Sameer Shrestha (THA076BCT039)

Sudeep Kaucha (THA076BCT044)

ABSTRACT

Every year there is a huge influx of students in Kathmandu valley from all over the country for their higher studies, as such, most of them are seeking a place to accommodate. For students, hostels or dormitory are the best choice as they can get a place to stay as well as daily foods. So, hostels are one of the basic infrastructures for our students who are the backbone and future of our country Even after all these years, finding a proper and clean hostel is hassle for students so our project The Dorm Den seeks to be a one stop solution to find a suitable hostel for all the students. The Genuity of information and reviews will be guaranteed in our application since only verified owners can add info and only verified students will be able to leave a review (thus frauds and fake reviews can be avoided). We will be using MERN stack for the development of our web application, which will have an efficient, user-friendly and attractive interface to interact with the data. Our project will not only be a way for students to find a good place to stay but also a good portal for hostel owners to highlight their business.

Keywords: *Express, Hostel, MongoDB, NodeJS, REACT.*

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BDUF	Big Design Upfront
ERD	Entity Relation Diagram
HCI	Human Computer Interaction
MERN	MongoDB Express React Node
RDMS	Relational Database Management System
SSPL	Server-Side Public License
UCD	User-centered Design
UI	User Interface

1 INTRODUCTION

1.1 Background

Altogether there are 1,440 campuses and 460,826 students currently in Nepal as per data collected in year 2077/78 [1] where majority of students prefer to choose to go away from home to study at a larger university as large universities often offer a wider range of academic programs and extracurricular activities than smaller colleges, which can provide students with more opportunities to explore their interests and develop new skills. Attending a large and well-known university can be a prestigious accomplishment, and may open future opportunities such as scholarships, internships, and job offers.

Living in a hostel can be a good option for these students who are looking for a cost-effective, social, and convenient place to live while studying rather than renting a place, room, or dorm.

Our application, a hostel booking app, aims to help users to find hostels in their destination and book a bed or a room quickly and easily, without the need to call or visit the hostel in person. This can save students time and effort and make their stay more comfortable and manageable. On top of that, we are also aiming to provide hostels with a convenient way to manage their reservations and occupancy, and potentially attract more guests through our application.

1.2 Motivation

Motivation for our application comes from several factors, one possible motivation is the need to make it easier for people to find and book accommodations without the need to call or visit the hostel in person. The Hostel Searching Web Application aims to simplify the process of finding a suitable hostel for students by providing them with a user-friendly interface that allows them to search for hostels based on various parameters such as location, price, amenities, and reviews. It provides a convenient, trustworthy, and efficient way for people to find suitable accommodation while also helping hostel owners to find suitable tenants. This project is focused on providing a solution for both hostel seekers and hostel owners.

1.3 Problem Definition

When booking a hostel offline, it can be difficult to access detailed information about the hostel, such as its location, amenities, and policies. This can make it challenging for students to compare different hostels and make an informed decision about where to stay.

Hostels can fill up quickly, especially during peak seasons. When booking offline, it may be difficult to determine the availability of beds or rooms at the hostel, and students may have to call or visit the hostel in person to check availability and make a reservation. An online hostel booking application like ours can help alleviate these challenges by providing users with easy access to detailed information about hostels and their availability and allowing them to make reservations quickly and easily.

1.4 Objectives

- To develop a web-based hostel application
- To offer detailed information about hostels, including their location, amenities, policies, and availability
- To offer hostels with a simple and effective way to manage their reservations and occupancy.

1.5 Project Scope and Application

A Hostel Booking application like ours can be used by students for faster and efficient finding and booking hostels in various locations, and by hostels to manage their reservations and occupancy.

The application provides detailed information about hostels, including their location, amenities, policies, and availability. Users could search for hostels based on their location, budget, and other criteria, and compare different options to find the best hostel for their needs.

Hostels could use the application to manage their reservations and occupancy, and to update information about their hostel, such as the availability of beds or rooms, policies, and amenities. This could help hostels attract more guests and manage their reservations more efficiently.

The scope and application of a hostel booking application would be to provide students and hostels with an easy and convenient way to find and book accommodations.

1.6 Report Organization

The contents of this report are organized into 9 chapters. The first chapter is the introduction where the introduction details of the project like what this project is trying to achieve, its objectives, scope, and application are detailed. The second chapter contains a brief description of the source materials and APIs we studied for this project. Third chapter is the analysis of the requirements, detailing various software, libraries used and operational feasibility. The fourth chapter explains about the architecture of our system, how the system is built, how its various components are connected and how they function together as a system. Fifth chapter is about the implementation details, enumerating how each individual component is implemented in the system. The chapter describes how the components communicate with each other, and what measures were applied for the efficient working of the system. The sixth chapter is dedicated to the results achieved in the project. The seventh chapter focuses on the possible enhancements that can be done in the future. The eighth chapter will have the conclusion of the project, entailing our success and hindrances while completing the project. Finally, the appendix will contain budget details of the project and the timeline of the development of the project.

2 LITERATURE REVIEW

2.1 Previous works

There have not been many previous works done in the field of hostel portal for students in Nepal, but something like our work can be seen in many Properties sales site or Hotel booking websites.

Previous work on the MERN stack has primarily focused on creating scalable and efficient web applications with a user-friendly interface. One such project is a hotel booking website developed using the MERN stack (from official React examples), which provides a platform for booking hotel rooms online. The website was developed using MongoDB for database management, Express for server-side routing, React for the front-end interface, and Node.js for server-side scripting. The website was well received by users, who appreciated its user-friendly interface, fast loading times, and the ability to book rooms quickly and easily. ("Hotel Booking App using the MERN Stack with TypeScript & Redux," ReactJSExample.

We found the best match possible to our project with Nepal Homes a large-scale property sales site from PropTech Nepal Private Limited, it is very user friendly, good optimization for site loading and proper data management. It also provided location search facilities and sorting on based on price, which we had to replicate in our project too. They seem to be using React as frontend, but we could not find out about their entire stack as it was not publicly available and we did not get any response from them either. But we could learn about proper system flow through their websites.

2.2 Recommendation of hostels

2.3 Places autocomplete and Geo encoding

For projects like this where (work is done based on location) place autocomplete and Geo encoding is done to get proper data to work on.

Location autocomplete is an API (Application Programming Interface) service that suggests users location based on real time input of the users, this will help to correct any spelling mistakes and provide the correct location to the application.

And for the location-based search we need exact coordinates of the searched location, for that Geo encoding was used in previous projects. This takes the name of street, city, state, or country and returns a coordinate for that spot, which is then passed to the geospatial library so that it can filter all the objects using this location as a reference point.

2.4 Geospatial Information Processing in MongoDB database

The amount of personal location data is forecast to increase by 20% every year, and location-aware information occupies a substantial proportion of the data generated every day: 2.5 quintillion bytes [2]. How to store, manage and query geospatial data has been the focus of research and problems that must be solved.

New geospatial application requires more flexible data schema, a faster query response time, and more elastic scalability than traditional spatial relational database currently have. The most communal problem occurrence has been seen when streaming requests from client to servers suddenly increase, which might cause response delay or even server unavailability. To solve this very scalability problem, a scalable framework was proposed back in MongoDB to implement elastic deployment for geospatial information sharing with the client users growing in number [3]. In this framework, MongoDB is chosen because it is a distributed database and supports a flexible storage schema suitable for massive map tile storage.

Several studies have found that RDBMS have some disadvantages in terms of big data storage and query in some specific areas, such as in streaming request or large data access environment in geospatial applications [4].

MongoDB provides two types of geospatial index types: geohash for 2d sphere and 2d. Within the 2d sphere relevant queries are implemented through the calculation of index on Earth-like 2d sphere. Within the 2d index, queries are calculated through a calculation of geometries on a two-dimensional plane. Four topological query operations are provided in MongoDB for geospatial data: $geoIntersects, $geoWithin, $near, and $nearSphere.

In a quantitative comparison of geospatial big data processing between the PostGIS and MongoDB databases, MongoDB had some advantages with its “within” and "intersection" queries and in terms of its response time for loading big geospatial data [5].

3 REQUIREMENT ANALYSIS

3.1 Project Requirements

3.1.1 Hardware Requirements

The hardware requirements for a hostel searching website are minimal and can easily be met by a standard laptop or desktop computer. A reliable internet connection is necessary, but otherwise the website does not require any specialized hardware.

A computer (laptop or desktop) with sufficient processing power, memory, and storage to run the web browser and support the application can be used.

3.1.2 Software Requirements

3.1.2.1 Figma:

Figma is a cloud-based design and prototyping tool that allows designers and developers to create interactive designs and collaborate in real-time. It is a versatile tool that can be used to create anything from UI/UX designs to wireframes, high-fidelity prototypes, and even graphic design projects. Figma offers a range of features such as vector editing tools, customizable design components, and design libraries that make the design process more efficient and effective. It also allows all the team members to work together in real time and removes the requirement of file sharing among members. It makes it easier for team members for communication and make changes based on feedback. Its range of resources and tutorials is accessible to all, which helps designers and developers to work efficiently.

3.1.2.2 Postman:

Postman is a popular API development tool that allows developers to create, test, and manage APIs more efficiently. It provides a range of features such as request and response logging, automated testing, and collaborative features that make API development more efficient and effective. Postman is available as a desktop application and a web-based tool.so, user can work in preferred working environment. It allows developers to write and run automated tests for APIs which helps to identify and address any issues or errors in the API quickly. It also provides a range of testing tools such as assertion libraries, which helps developers to write more robust and effective tests for their APIs. It lets us share collections, test results, and documentation, and works with other tools like GitHub to make collaboration easier.

3.1.2.3 MongoDB compass:

MongoDB Compass is a graphical user interface (GUI) tool for MongoDB which is a popular document-oriented NoSQL database. It provides database administrators with a visual representation of their data, making it easier to manage, analyze and optimize their MongoDB databases. It offers users the chance to explore and manipulate data using an intuitive interface, providing a range of features such as data visualization, data filtering, and schema exploration. MongoDB Compass also integrates with other MongoDB tools, such as MongoDB Atlas, a cloud-hosted MongoDB service, making it easier to manage and monitor databases in the cloud It allows users to monitor and analyze their MongoDB databases in real-time, providing instant feedback on data changes and performance metrics. This helps to optimize the database for better performance, scalability and security.

3.1.2.4 React developer tools:

React Developer Tools is a free and open-source tool, browser extension for Chrome, Firefox, and Edge that allows developers to inspect, and debug React applications. It provides a range of features such as component trees, props and state inspection and performance profiling that make it easier to identify and fix issues in React applications. It makes it easier to debug and optimize React applications. It allows us to view the structure of their application's components, making it easier to identify where issues may be occurring. React Developer Tools also allows developers to inspect the props and state of each component, providing valuable insights into how the application works. It provides a range of metrics such as render time, update time, and time spent in lifecycle methods, making it easier to identify performance and optimize the application for better performance.

3.1.2.5 Node JS:

Node.js is an open-source, cross-platform, server-side JavaScript runtime environment. It enables developers to create scalable network applications with ease. Node.js uses an event-driven, non-blocking I/O model, making it efficient and lightweight. It has a vast library of pre-built modules that can be easily integrated into applications, saving developers time and effort. Node.js is widely used for developing web servers, real-time applications, chat applications, and streaming applications. Its popularity is also due to its ability to handle a large number of connections simultaneously, making it suitable for high-traffic websites. Node.js is supported by a large community of developers, who contribute to its growth by creating new modules and tools and providing support to fellow developers. With its flexibility and versatility, Node.js has become a popular choice for modern web application development.

3.2 Feasibility Analysis

3.2.1 Financial feasibility:

All the libraries we need for the completion of this project are open source. So, the expenditure is minimal for this project. We have to pay for server hosting and acquiring domain names. Also, we have all hardware requirements already, so we do not have to buy any additional hardware equipment for the completion of our project.

3.2.2 Technical feasibility:

Currently we have a laptop with the following specifications:

- Processor: Intel Core i5-8250U @ 1.6GHz
- RAM:8 GB
- SSD:256 GB
- HDD:1 TB
- Graphics: NVIDIA, MX130 ,2GB
- Network: We have a fast and stable internet connection in our laptop.

For software requirements we have installed all the necessary packages required for the completion of our project.

3.2.3 Operational feasibility:

- Security: Measures to protect sensitive data and prevent unauthorized access to the application.

- User Acceptance: User acceptance of the application based on ease of use, performance, and functional requirements.
- Maintenance: Cost and effort required to maintain and upgrade the application over time.
- Integration: Ability to integrate with existing systems and data sources.
- Scalability: Ability to handle increased demand for the application
- Reliability: Ability of the application to perform consistently and without failures.

4 SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 System Block Diagram

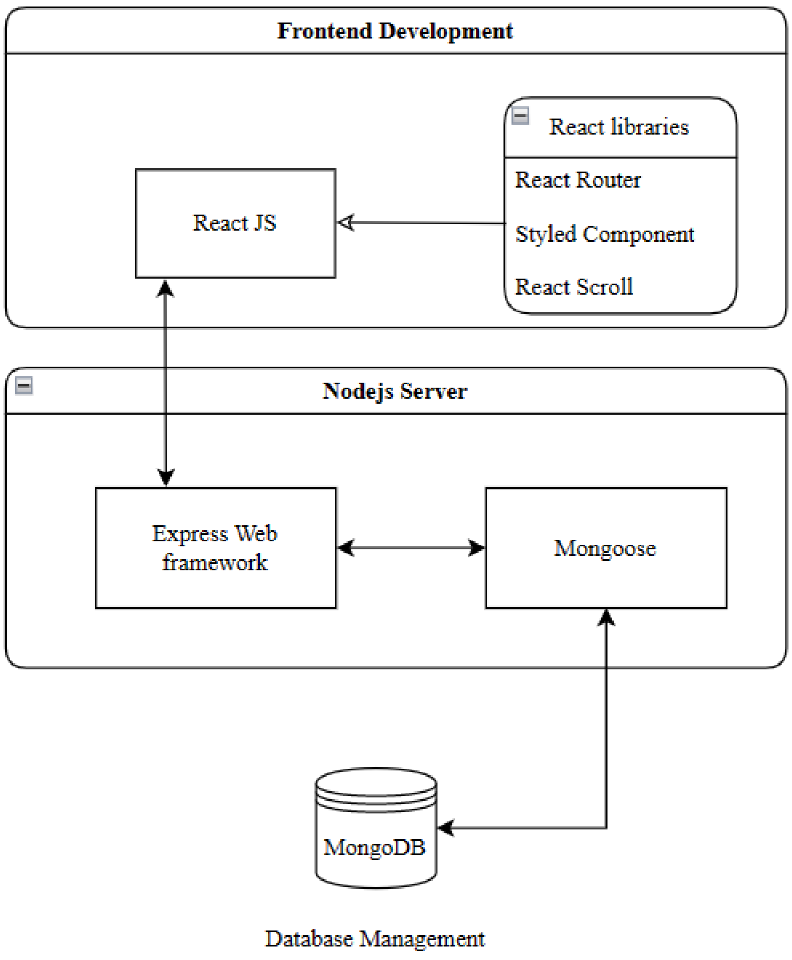
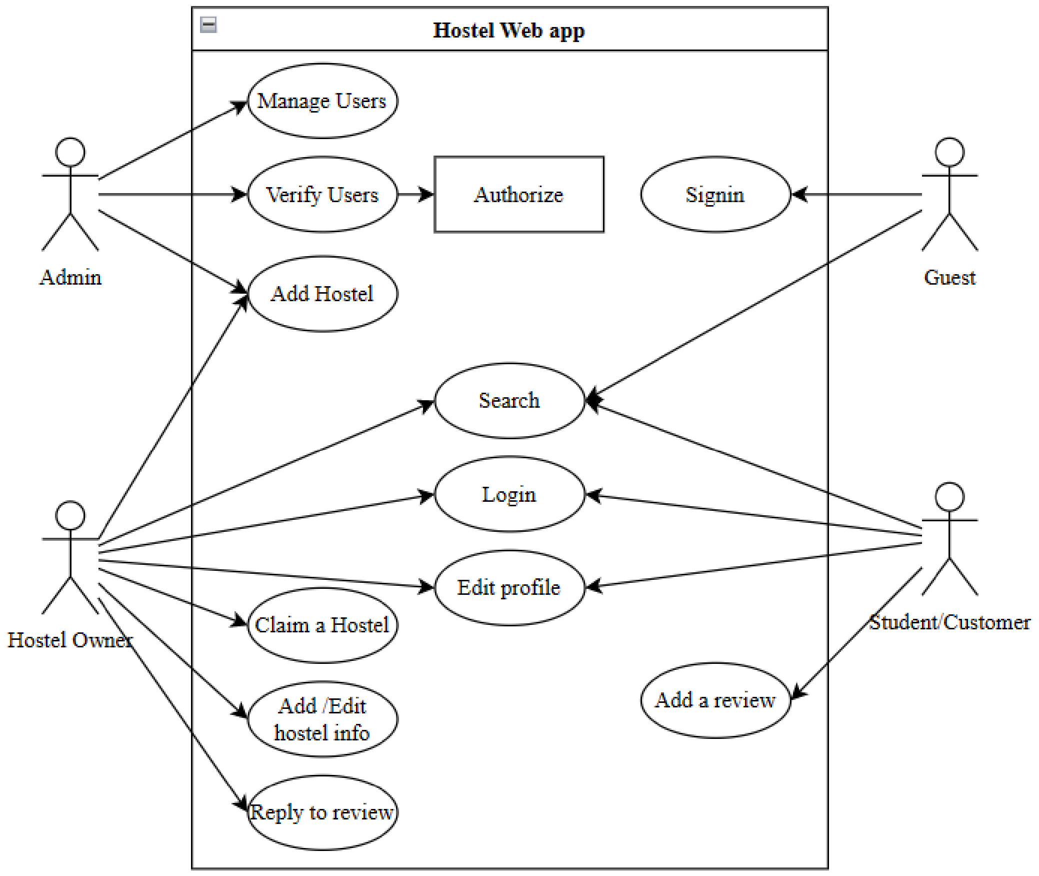


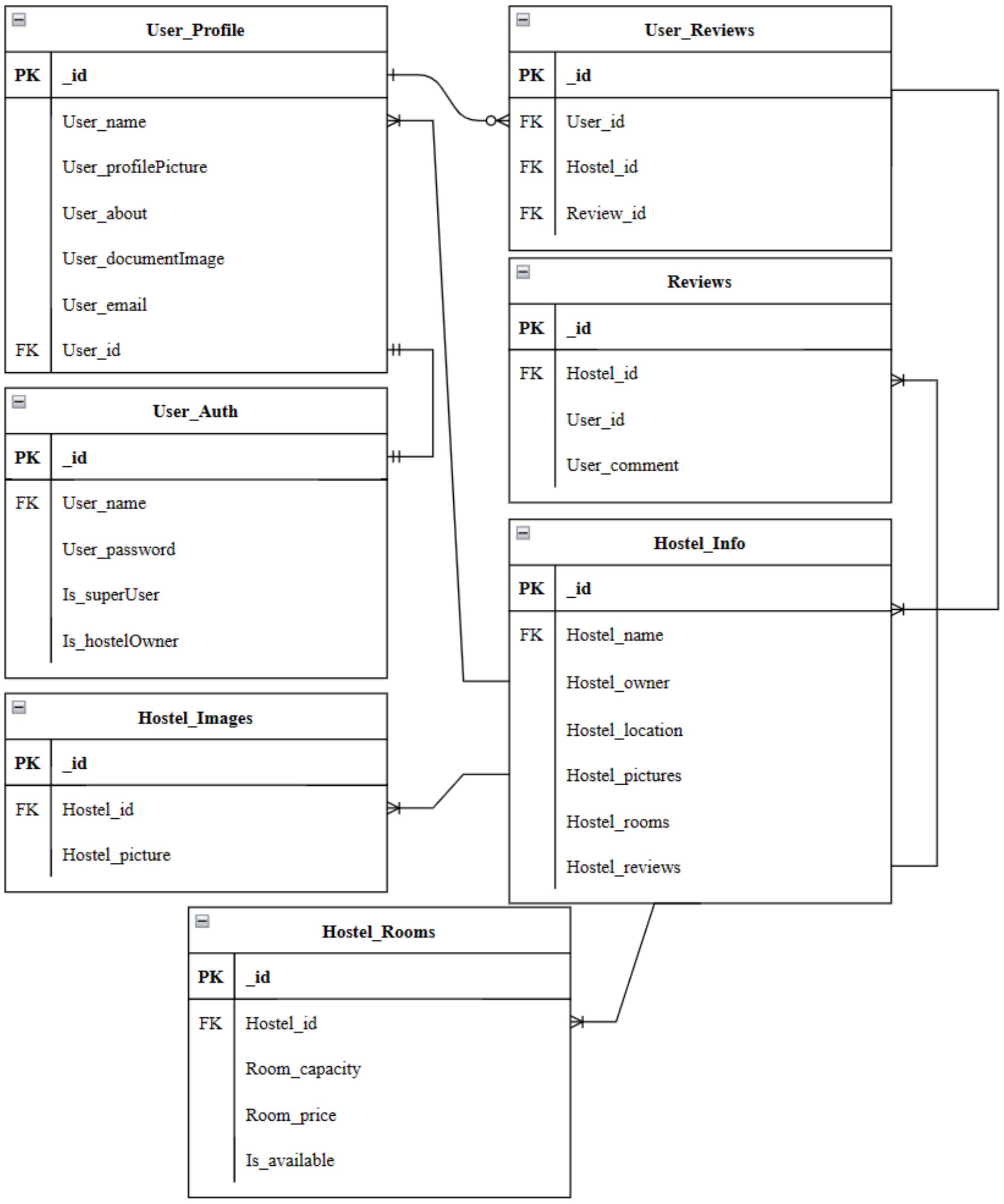
Figure 4.1 System Block Diagram

4.2 Use Case Diagram



The system's use case diagram, which represents the actors and the system, is depicted in the diagram above. The system's service to the actors is depicted in this diagram. In the system, there are four types of actors: Admins, Guest, Hostel owners and Registered Users. A registered user can search for hostels, write reviews, rate them, and use the recommendation and suggestion services. The user can also make changes to their account. The system administrator oversees Verifying user documents and moderating any contents as well as add new hostels. Hostel owners can request to add a new hostel or claim one with proper documentation. Guest users are only allowed to search hostels and watch their reviews but cannot give reviews of their own.

4.3 Entity Relation Diagram

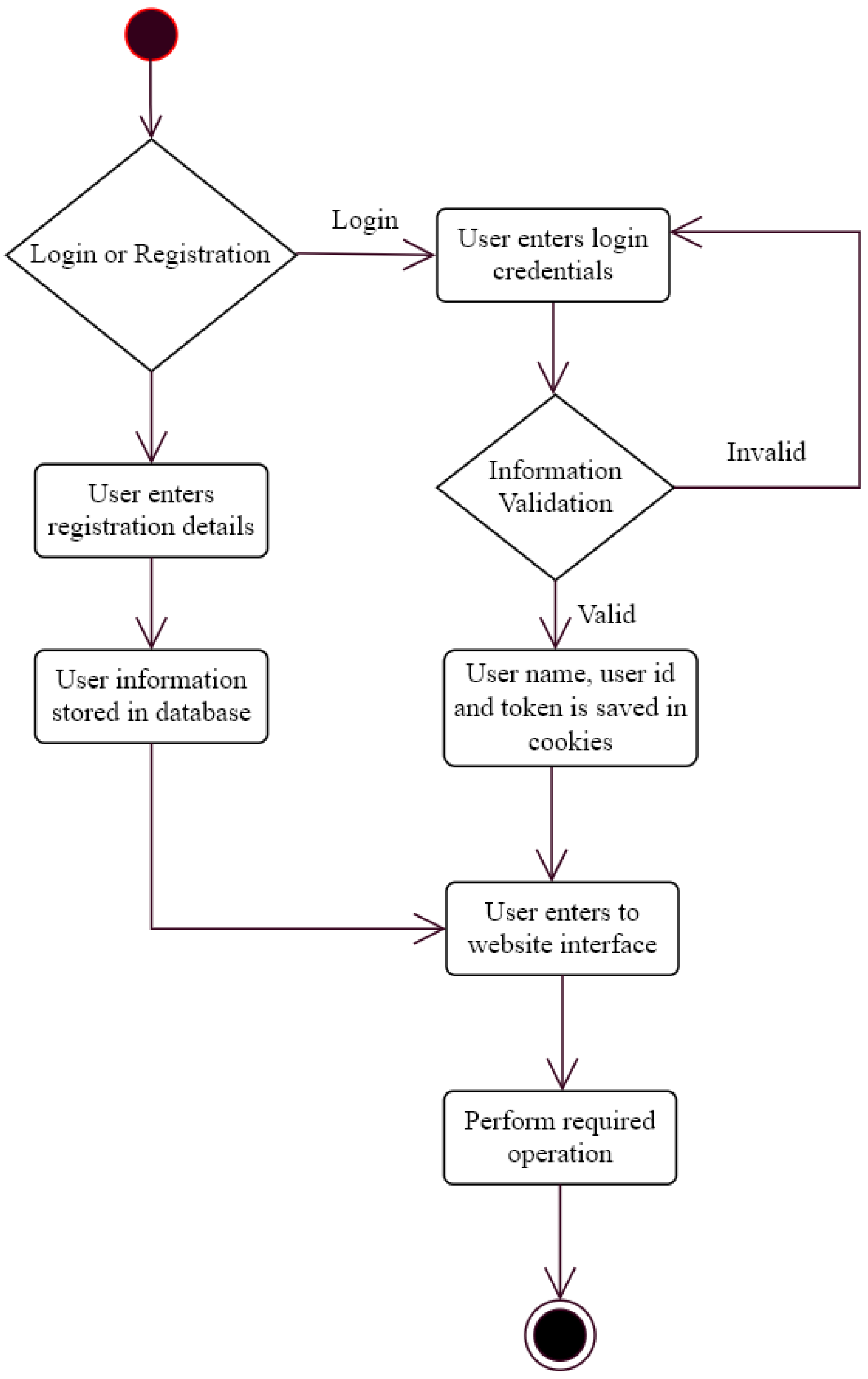


ER diagram (Entity-Relationship diagram) is a diagram which shows entities, attributes of entities and relationship between them. ER diagram of our system is as shown above.

4.4 Activity diagram:

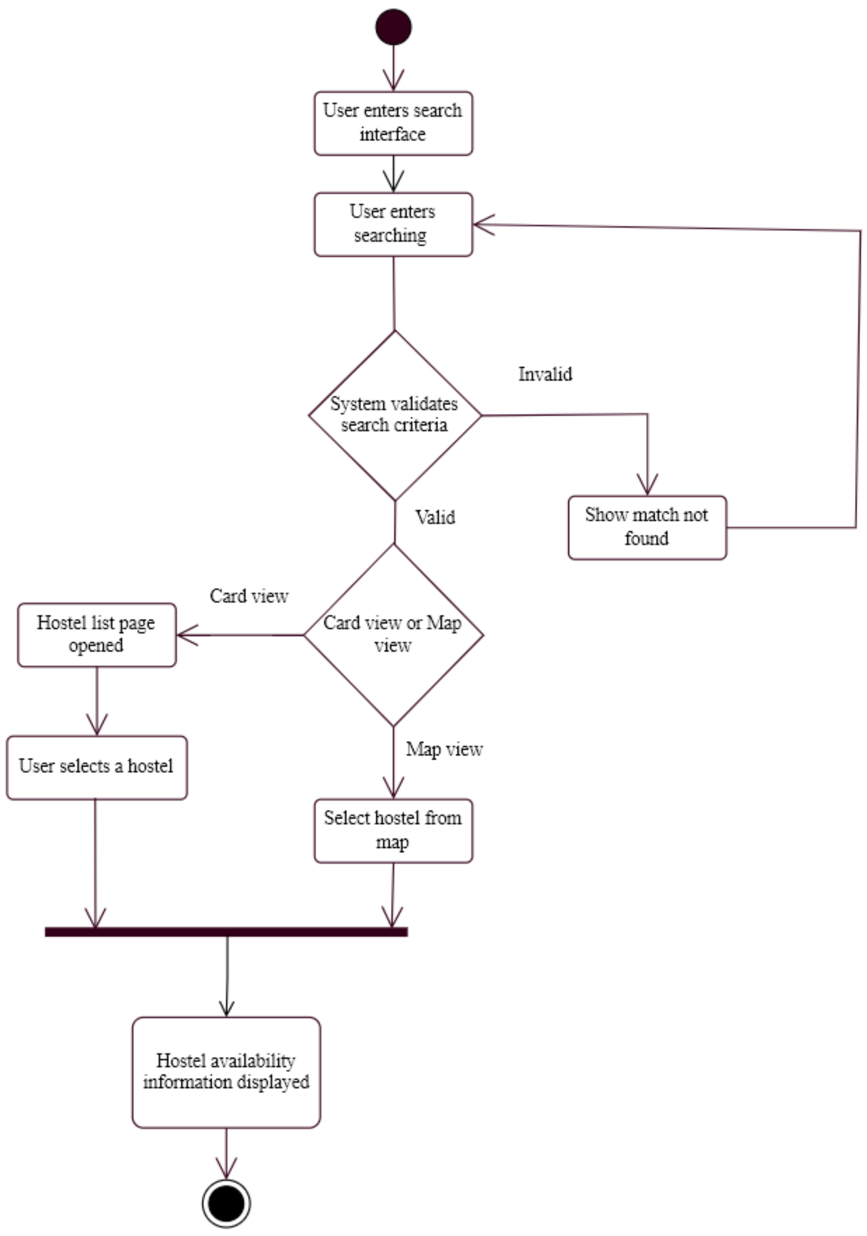
Our website involves various processes such as login and registration, searching for hostels, registering a hostel, editing hostel and user information. These activities help ensure a smooth and secure experience for both users and hostel owners, making the website an effective tool for searching hostels. The activity diagrams for these processes involve multiple steps, which is shown below:

4.4.1 Login and registration:


Login or Registration
Login
User enters login credentials
Information Validation
Invalid
Valid
User enters registration details
User information stored in database
User name, user id and token is saved in cookies
User enters to website interface
Perform required operation

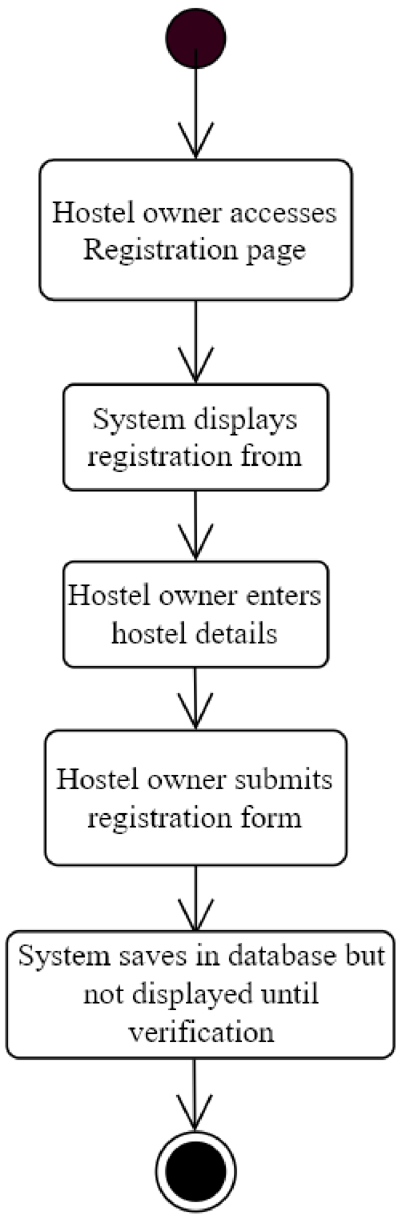
This activity diagram represents the steps involved in user registration and login process. It shows the flow of events that occur when a user registers for an account and logs in to the system. The user may be Student, admin or Hostel owner. The user registration and login process typically involves several steps. First, the user must register for an account by providing their registration information, such as their name, email address, Phone number and password. Once the user has registered, user id, user name and token are saved in cookies. For login, they must enter their login credentials. The system then validates the information provided by the user and checks whether the user's login credentials are valid. If the information is valid and the credentials are correct, the user is logged in and taken to the page, where they can access the features and functionality of the system that are available to them based on their account type and permissions. If the information or credentials are invalid, the system displays an error message, and the user is prompted to correct the invalid information or enter valid login credentials. Once the user successfully logs in, they can use the system's features and functionality, and their information is stored in the system for future use.

4.4.2 Search process:



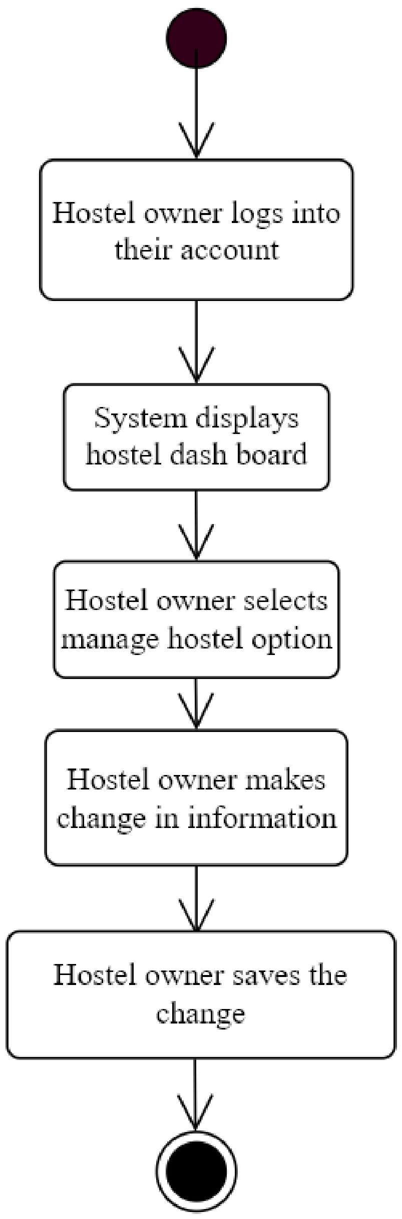
This activity diagram for the search operation in our website involves the user entering search interface, the system validating and retrieving a list of hostels that match the criteria, displaying the list to the user, allowing them to select a hostel, and displaying details and availability or information of the selected hostel. The user can select according to location, seater, hostel name. If the user entered invalid search information then system shows match not found. The user can select hostel in card mode or view mode.

4.4.3 Hostel Registration:


Hostel owner accesses Registration page
System displays registration from
Hostel owner enters hostel details
Hostel owner submits registration form
System saves in database but not displayed until verification

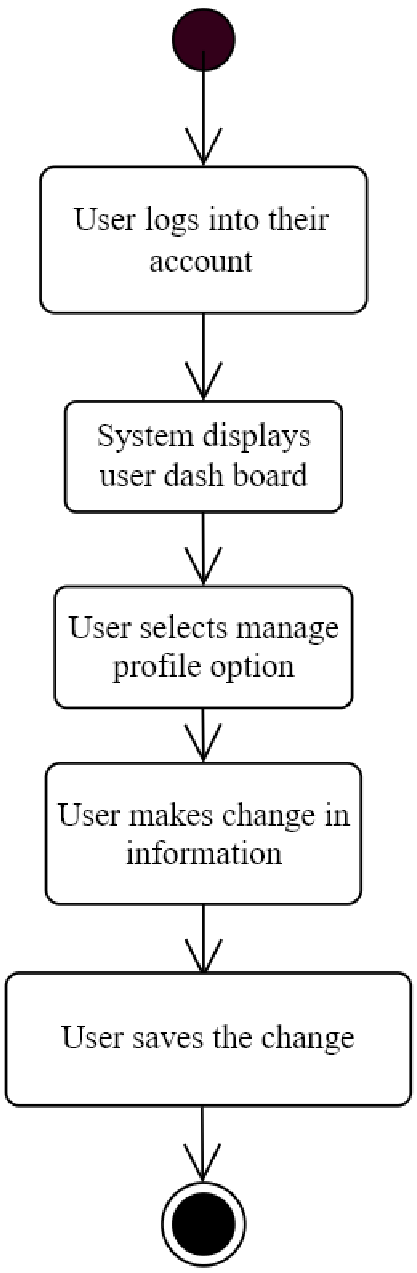
This activity diagram for hostel registration involves the hostel owner accessing the registration page, and entering the hostel details such as name, address, amenities, room types(seater) and pricing, hostel photos etc. The system saves the entered details of registration form in database and only shows the information in website after validation. The activity diagram helps ensure a smooth and secure registration process for hostel owners, enabling them to showcase their hostels and attract potential guests through the website.

4.4.4 Edit hostel information:


Hostel owner logs into their account
System displays hostel dash board
Hostel owner selects manage hostel option
Hostel owner makes change in information
Hostel owner saves the change

This activity diagram involves the hostel owner logging into their account, selecting the hostel they want to edit and choosing the information they want to modify, such as amenities, pricing, availability etc. The Hostel owner selects the manage hostel option in their dashboard to make any changes in the hostel information. After successful change they saves the changes. This process enables hostel owners to keep their hostel information up-to-date, providing accurate details to potential guests. It ensures a secure and efficient editing and updating process on the website.

4.4.5 Edit user information:

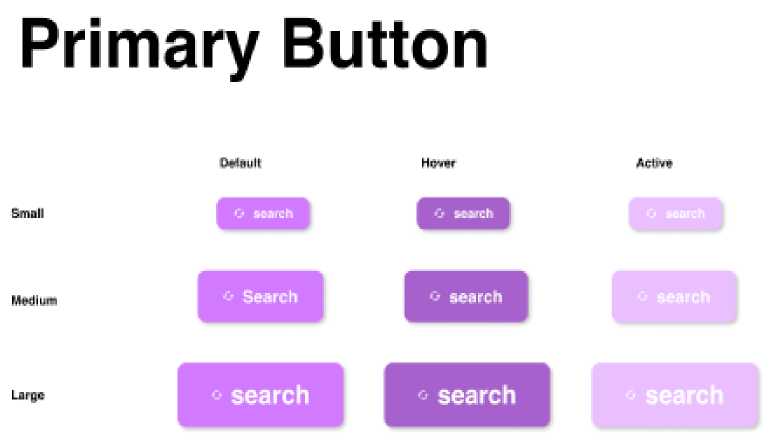

User logs into their account
System displays user dash board
User selects manage profile option
User makes change in information
User saves the change

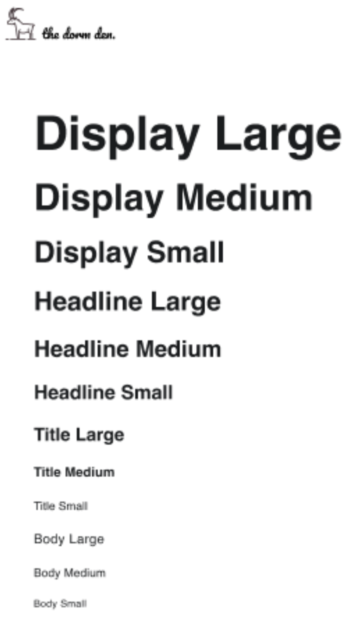
The activity diagram for editing user information requires the user logging into their account, accessing their profile, and selecting the information they want to modify, such as personal details. The system displays the selected information for editing, and the user makes changes and saves them in database. Later admin validates and verifies changes for ensuring the security of user information and protecting against fraud. This activity diagram enables users to keep their information up-to-date. It also provides accurate details and ensures a smooth and secure website using experience.

5 IMPLEMENTATION DETAILS

5.1 Design:

UI/UX is a very important step in development of web applications, it can greatly impact the user's perception of a product as well as enhances the user retention rate. A well-designed website can increase efficiency as well as improve overall user experience. With the understanding of the needs and behavior of users, we have created a very easy to use and attractive design for the Dorm den site using Figma tools.




the dorm den.

Display Large

Display Medium

Display Small

Headline Large

Headline Medium

Headline Small

Title Large

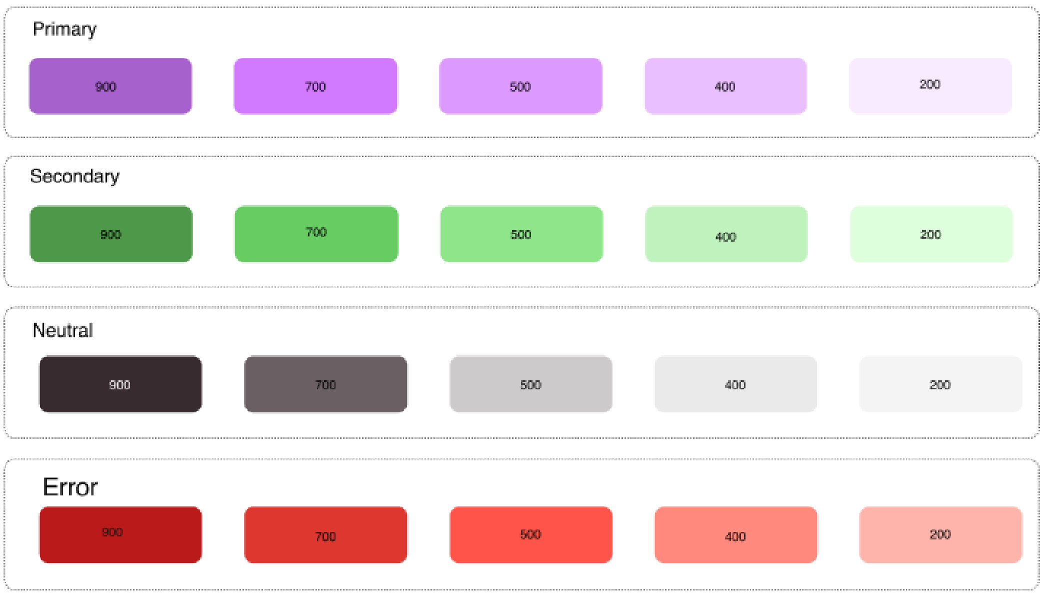
Title Medium

Title Small

Body Large

Body Medium

Body Small



5.2 JWT:

JSON Web Token (JWT) is a standard for securely transmitting information between parties as a JSON object. JWT is used in our project as a means of authentication and authorization. When a user logs in, they receive a JWT that contains information about their identity and permissions. The token is then sent with every subsequent request, allowing the server to verify the user's identity and ensure they have the necessary permissions to perform the requested action. Here’s a code snippet of how user id, user name and their type is first encrypted using a secret key and then send to user through the means of cookies:

```
const userForToken = {
        username : user.username,
        id : user._id,
        isAdmin : user.usertype.typeof_user === “admin” ? true : false
};
const token = jwt.sign(userForToken, process.env.SECRET);
res.status(200)..cookie(“token”,token,options).send({token,
username:user.username,id:user._id});
```

5.3 Nodemon:

Nodemon is a development tools that helps by auto restarting the node server whenever a change is saved in node, thereby removing the tedious and manual process of starting the server after each change in the file. You don’t have to make any additional changes to the code or method of development, to use nodemon just replace the “node” with nodemon on command line while writing the script. Here’s the code snippet of our package.json where we used nodemon:

```
"scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
// When starting the server run the command:
> npm run dev
```

5.4 Nodemail:

Nodemailer is a Node.js module used for sending email messages from a Node.js application. It provides a simple API for sending email with attachments and html contents. In our project we used nodemailer to send a token to user email in case they forgot their password and want to reset them. A random hex value is generated (stored in user database) and its encrypted form is sent to the user in the mail as a recovery link.

```
const createResetPasswordToken = async () => {
    const resetToken = crypto.randomBytes(20).toString('hex');
    const reset_password_token = crypto
        .createHash('sha256')
        .update(resetToken)
        .digest('hex');
    const reset_token_expires = Date.now() + 10 * 60 * 1000;
    return { resetToken, reset_password_token, reset_token_expires };
}
```

5.5 Pigeon Map:

Pigeonmap is a JavaScript library that provides a simple and intuitive way to display maps and markers on a web page. The API integrates with popular mapping services such as OpenStreetMap and Leaflet to provide a user-friendly interface for adding and customizing markers on a map.

We have used pigeonMap in our project to show the hostels on a map using their coordinated, this will make the user interaction even better since they will have a visual representation apart from just textual location.

```
<Map
        center={center}
        zoom={zoom}
        onClick={()=>{setOverlayStatus(false)} }
        >
            {
            hostels.map(hostel => (
            <Marker
            key={hostel._id}
            anchor={[hostel.location.coordinates[1], hostel.location.coordinates[0]]}
            color="#f40d0d"
            onClick={() => {
                setOverlayStatus(true)
                setOverlayContent(hostel)}}
            />
      ))}
 </Map>
```

5.6 React router dom:

React Router DOM is a popular routing library for React applications that allows developers to manage navigation within a single page application. It provides a simple API for creating routes, handling redirects, and passing parameters between components. This package also supports dynamic and nested routing. We have created different routes for different pages so that users can get a seamless experience. A small code snippet about the use of our react-routing is given here:

```
import {Routes, Route } from 'react-router-dom';
<Routes>
     <Route path="/" element={<HomePage/>} />
     <Route path="/hostels" element={<HostelSearchResultPage />} />
     <Route path="/hostels/:id" element={<HostelIndividualPage />} />
     <Route path="/login" element={<LoginPage />} />
     <Route path="/register/user" element={<UserRegistrationPage/>} />
     <Route path="/register/hostel" element={<HostelRegistrationPage/>} />
     <Route path="/admin" element={<AdminDashboardPage/>} >
       <Route path="userverification" element={<UserVerificationDashboard />} />
       <Route path="users" element={<AdminDashboardUserComponent/>} />
       <Route path="hostels" element={<AdminDashboardHostelComponent/>} />
     </Route>
     <Route path="/user" element={<UserDashboardPage/>} >
       <Route                                          path="changepassword"
element={<UserDashboardPasswordComponent/>} />
     </Route>
   </Routes>
```

5.7 Styled Component:

Styled-components is a react library which allows developers to write CSS code within their JSX components. Its main advantage is creating styled components which can be reused throughout the application. We have used Styled Components in our project to create custom styled components that are easy to modify and extend. With Styled Components, we can define styles directly within our components, making it easier to keep track of CSS rules and reducing the potential for naming conflicts.

```
import styled from 'styled-components';
const Input = styled.input`
          height: 2rem;
          width: 100%;
```

```
        padding: 0.5rem;
        border: 1px solid #D179FF;
        border-radius: 6px;
        &:focus {
                outline: none;
                border: 1px dashed #D179FF;
                background-color: #e9e9e940;
  }
`
```

5.8 React Icons:

React Icons is a popular library that provides a collection of customizable icons for use in React applications. It includes icons from popular icon sets such as Font Awesome, Material Icons, and Feather Icons. One of the key features of react icons is that developers can easily use icons without individually downloading each SVG and pictures. The icons provided by react icons are customizable. We have used React Icons in our project to improve the user interface and provide an intuitive experience for users.

5.9 Unbiased rating system:

For unbiased recommendation of hostels in our project we will be using reviews of authenticated students' feedback (authentication through user ID) but doing so we would face a major hurdle i.e., sparse number of reviews/feedbacks in the beginning.

For example, if hostel A had 100 reviews, among them 90 are 5 stars and the remaining are 4 stars (making their average less than 5) and hostel B had 5 reviews, all being a 5-star rating (making their average exactly 5 stars). This would pose a huge problem when sorting based on reviews.

To overcome this, we will be using true Bayesian estimate, which has been previously used by IMBD to make their top 250 movies list.

$$Weighted\ rating(WR) = \left(\frac{v}{v+m}\right) * R + \left(\frac{m}{v+m}\right) * c$$

which can be mathematically simplified to:

$Rating = (R * v + C * m)/(v + m)$

Where the meaning of symbols is,

R = average for the movie (mean)

v = number of votes for the movie

m = minimum votes required to be listed in the Top N (This can be thought as tuning parameter)

C = the mean vote across the whole report (The average rating across the whole database)

Using this we would create a ranking list for every hostel in the database and use that for unbiased recommendations.

```
hostelSchema.methods.compute_ranking = async function() {
const m = 15;
   const R = this.hostel_rating;
   const v = this.number_of_reviews;
   reviews = await Review.find({})
   let C = 0;
   reviews.forEach(review => {
      C += review.overall_rating;
   })
   C /= reviews.length;
   this.ranking = (R * v + C * m) / (v + m);
}
```

5.10 Address Autocomplete API:

We used Address Autocomplete API for address autocomplete inputs. When user enters free-form addresses, the API returns verified and standardized addresses. The API returns address suggestions as user types. So, if user types "Thapathali", the API will return "Thapathali, Kathmandu, Bagmati Pradesh, Nepal" and more address suggestions to choose from.

In our react component, we fetch the response from Address Autocomplete API using axios in JSON format. As user types the name of place, the input value is saved in a state variable “location” and the API will be queried with the name of the place. The response of the API includes the list of the places it matches to in JSON format.

```
axios
.get(`https://api.geoapify.com/v1/geocode/autocomplete?
text=${location}&apiKey=${API_KEY}`
)
.then((res) => {
    setResponse(res.data)
}
```

The API_KEY in request URL is the key obtained from Geoapify which is needed to fetch the list of matching places. The response from API is then stored in a state variable “response” as an array of formatted address.

```
name: "Thapathali"
country: "Nepal"
country_code: "np"
state: "Bagmati Pradesh"
county: "Kathmandu"
city: "Kathmandu"
postcode: "00779"
suburb: "Thapathali"
lon: 85.3201449
lat: 27.6915948
formatted: "Thapathali, Kathmandu, Bagmati Pradesh, Nepal"
address_line1: "Thapathali"
address_line2: "Kathmandu, Bagmati Pradesh, Nepal"
```

As user types location, a small modal box listing the formatted address fetched using API pops up, from where user can choose an address. The chosen formatted address consists of “lon” and “lat” property which is stored as a state variable “longitude” and “latitude”. In this way, Address Autocomplete API makes user’s search even more accurate. Now when a user types in the location input field, the API will be queried and the returned suggestions will be stored in the response state variable, which can be used to display a list of suggested address to the user.

5.11 Geo Spatial datatype

A geospatial data type is a type of data that represents a location on the earth's surface. In Mongoose, the two main geospatial data types are "Point" and "MultiPoint", which are special GeoJSON data types that represents a point or multiple points on the earth's surface, respectively.

We need geospatial data types to enable us to store and query location data in our MongoDB databases. By using geospatial data types, we can easily perform complex location-based queries using Mongoose's built-in geospatial query operator.

```
location: {
type: {
type: String,
enum: ['Point'],
required: true
},
coordinates: {
type: [Number], // longitude and latitude format !extremely important
require: true
}
}
```

First, we defined a Mongoose schema for our location data that includes a field for the location coordinates. We used "mongoose.Schema.Types.Point" data type for this field, which is a special GeoJSON data type that represents a point on the earth's surface. The coordinate of hostel is stored in longitude and latitude format in the database.

```
Hostel.find({
location: {
$near: {
$geometry: {
type: "Point",
coordinates: [longitude, latitude]
},
$maxDistance: 10000
}
}
})
```

Finally, `Hostel.find()` method is used to query the database for locations within a certain radius of a given location. Here’s an example query that finds all locations within 10km of a point with coordinates `[longitude, latitude]` where longitude and latitude variables as passed as query from the client side of our application.

We used `$near` operator in the query to find documents with `location` fields that are near the `Point`. We also set the `$maxDistance` option to 10,000 meters to limit the results of hostel to those within 10km.

5.12 Cookie-Parser

When a user logs in to our application, we need to set a cookie that identifies them. Cookies are used to maintain state across HTTP requests, which is important for tracking the user’s session and authentication status. “Cookie-parser" is a middleware that makes it easy to parse cookies from incoming requests, so we can verify a user and grant or deny access to the protected route based on the authentication status.

```
const UserToken = {
username: user.username,
id: user._id,
isAdmin: user.usertype.typeof_user === "admin" ? true : false
};
const token = jwt.sign(UserToken, process.env.SECRET);
res.status(200).cookie("token", token, options).send({ token });
```

When a user sends login credential to backend server for verification, server sends JWT signed token as response if the user is verified. When you sign a JWT, you create a header, a payload, and a signature. The payload contains information such as username, user_id, and boolean data of whether the user is admin or not. The resulting token is sent to the client and is used to authenticate future requests.

```
const cookieParser = require('cookie-parser');

app.use(cookieParser());

const token = req.cookies.token;
if (!token) return next(CreateError("You are not authenticated!", 401))
try {
const user = jwt.verify(token, process.env.SECRET)
req.user = user
next()
} catch (error) {
next(CreateError("Token is not valid!", 401))
}
```

In order to implement cookie-parse we use it as a middleware in our application. Next, we create our own token verification middleware as “verifyToken” method which we pass to our routes. Using cookie-parser as middleware we can access the cookies sent by the client side of our application. The middleware will check for a JWT token in the cookie named ‘token’. If the token exists and is valid, it will set the decoded user object on the request object and call the next middleware in the chain. Otherwise, it will return a 401 Unauthorized error.

```
const express = require('express');
const router = express.Router();

router.route('/delete/:id').delete(verifyUser, delete_user);
```

This route will only be accessible to users who have a valid JWT token in their cookie. The user object from the JWT token will be available on the request object as “req.user”. With these steps, our backend application can parse cookies and verify JWT tokens for authenticated routes.

5.13 Bcrypt

Passwords are sensitive data that should not be stored in plain text in a database or transmitted over the network without encryption. If an attacker gains access to the password database or intercepts the network traffic, they could easily read the passwords and use them to gain unauthorized access to user accounts.

Hashing is a technique that allows us to store passwords securely without revealing the original password. When we hash a password, we convert it into a fixed-length string of characters that is unique to that password. This hash can then be stored in a database or transmitted over the network without revealing the original password.

Bcrypt is a popular and secure hashing algorithm that is designed to be slow and computationally expensive. It uses a technique called key stretching to make it more difficult for an attacker to guess the original password from the hash. Bcrypt also automatically generates a random salt for each password, which makes it more difficult for an attacker to use precomputed tables (known as rainbow tables) to crack the hashes.

```
const bcrypt = require("bcrypt");
const saltRounds = 10;
const passwordHash = bcrypt.hashSync(body.password, saltRounds);
```

we first define a “saltRounds” variable to determine the number of salt rounds to use when hashing passwords. Generally, the higher the number of salt rounds, the more secure the hash will be, but it will also take longer to compute. The “hashSync” method computes the hash using the salt and the password which is stored in “passwordHash” and saved to database used to authenticate user on login.

```
const bcrypt = require("bcrypt");
bcrypt.compare(password, user.passwordHash);
```

“compare” method of bcrypt takes a plain text password and a hash as input, and returns true if the password matches the hash, and false otherwise. If the method returns true, we then send token and success information with status 200 to client-side else status 401 with unauthorized message is sent. In this way, bcrypt is used to hash password in our application that protects user accounts from unauthorized access.

5.14 Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a way to define data schemas and interact with MongoDB databases using JavaScript syntax.

Mongoose allows us to define data models for our MongoDB collections, which can then be used to create, read, update, and delete documents in the database. It also provides a number of useful features such as data validation, middleware, and query building.

```
const mongoose = require(“mongoose”)
mongoose.connect(config.MONGODB_URI)
.then(() => {
logger.info('connected to MongoDB');
})
.catch((error) => {
logger.error('error connecting to MongoDB: ', error.message)
})
```

In our backend server, we require ‘”mongoose” package and connect to a MongoDB database. We connect to a MongoDB database served in cloud through “MongoDB Atlas” service using the “mongoose.connect” method.

```
const hostelSchema = new mongoose.Schema({
name: {
type: String,
required: true
})

module.exports = mongoose.model('Hostel', hostelSchema);
```

Once we are connected to the database, we can define a data schema and model using Mongoose. We define a “hostelSchema” using the “mongoose.Schema” method. The schema defines the structure of the Hostel collection in the database, including the fields name. We also specify some validation rules for the fields using Mongoose's schema options. Likewise, we also mase schema for Review and User collection.

```
const hostel = await Hostel.findById(req.params.id);

const updatedHostel = await Hostel.findByIdAndUpdate(req.params.id, body, {
new: true,
runValidators: true,
useFindAndModify: false
});

const hostels = await Hostel.find({ 'verified': false }).populate('owner');

hostel.remove();
```

We then define a Hostel model using the “mongoose.model” method. The model represents the Hostel collection in the database and provides a way to interact with the collection using JavaScript syntax. We can then use the Hostel model to perform CRUD operations on the Hostel collection.

We find “Hostel” documents using “find” and “findById” method, update a “Hostel” document using “findByIdAndUpdate” method, and remove the particular “Hostel” document using “remove” method. “populate” method is used to load referenced documents from other collections, and it is used to resolve references between collections. It allows you to include references from other collections in a query result.

5.15 Dotenv

The .env file is a configuration file used in Node.js-based projects to store environment variables. These variables are key-value pairs that can be accessed by your Node.js application at runtime. We have saved sensitive informations, such as API keys for MongoDB, GeoAPI, Nodemail, cloudinary and password information for mail. We can easily change these variables without having to modify our code, and we can keep them private and secure.

```
require(‘dotenv’).config();
const MONGODB_URI = process.env.MONGODB_URI;
```

We create “.env” file in the root directory of our directory and define our enviromental variables in the “.env” file using the “KEY=VALUE” format. Then we access these environment variables in our code using the “process.env” object.

We add the .env file to your .gitignore file, so that it is not checked into version control and does not get exposed to unauthorized users. By using .env files in your Node.js project, we easily managed our configuration variables and kept sensitive information secure.

5.16 CORS

CORS stands for Cross-Origin Resource Sharing, which is a security feature implemented in web browsers that restricts web pages from making requests to a different domain than the one the page came from. This is done to prevent malicious scripts from stealing sensitive data or performing unauthorized actions on behalf of the user.

However, in our case, it is necessary for a web application to make cross-origin requests to fetch resources from a backend server serving in different port address. CORS provides a mechanism for a web application to inform the browser that it should allow requests from a different origin.

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors());
```

In your Express.js application, require the cors module and use it as middleware. This will allow all cross-origin requests.

5.17 useContext API

```
import React, { createContext, useReducer, useEffect } from "react"
import Cookies from "js-cookie"

const INITIAL_STATE = {
user_id: Cookies.get("user_id") || null ,
username: Cookies.get("username") || null,
token: Cookies.get("token") || null,
loading: false,
error: null
}
```

We are importing the necessary dependencies for creating a context and reducer in React, as well as a third-party library "js-cookie" that allows us to manipulate browser cookies.

We initialize the initial state of the authentication context. It retrieves the values of user_id, username, and token from cookies, which may have been previously set during a previous login session. If no values are found, the state is set to null. The loading and error properties are initially set to false and null respectively.

```
export const AuthContext = createContext(INITIAL_STATE)
```

We are create a new authentication context using createContext() API from React. This context is then exported for other components in your React application to use.

```
const AuthReducer = (state, action) => {
switch (action.type) {
case "LOGIN_START":
return {
        user_id: null ,
        username: null,
        token: null,
        loading: true,
        error: null
}
case "LOGIN_SUCCESS":
return {
        user_id: action.payload.user_id ,
        username: action.payload.username,
        token: action.payload.token,
        loading: false,
        error: null
}
case "LOGIN_FAILURE":
return {
        user_id: null,
        username: null,
        token: null,
        loading: false,
        error: action.payload
}
default:
        return state
}
}
```

We defined the reducer function for your authentication context. It takes two arguments, state and action, and returns the updated state based on the action dispatched. The reducer function updates the authentication state by switching on the action type and returning a new state object that reflects the updated authentication state.

6 RESULT AND ANALYSIS

6.1 Features and Functionality

6.1.1 Frontend

6.1.2 Backend

6.2 Performance Analysis

The response time of data retrieval from the backend is 200 to 300 ms, which is fast and efficient for the system. The page load time is efficient, meets industry standards and is consistently within acceptable limits, providing a positive user experience. The web-application can oversee a high volume of requests per second with no noticeable lag or delay. The results of resource usage testing indicate that the application uses a minimal amount of memory, CPU, and other resources, providing a fast and responsive user experience.

6.3 Security Analysis

The security of the system is ensured using JWT (JSON Web Token). The token is signed by a secret key on the server, ensuring that the contents of the token cannot be altered during transmission. The client sends the token in the Authorization header with each request, allowing the server to verify the authenticity of the request and identify the user associated with it, without having to perform a database lookup. JWT restricts unauthorized access to any data required and handles the data security issue of the system.

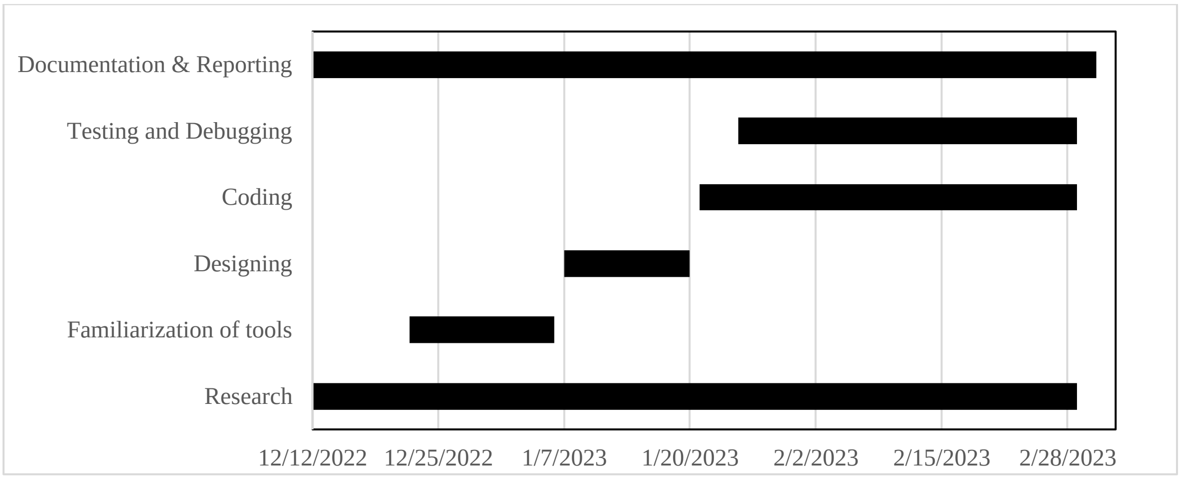
7 FUTURE ENHANCEMENT

8 CONCLUSION

9 APPENDIX

9.1 Project Schedule

Table 9.1 : Project Schedule



9.2 Project Budget

Table 9.2 : Project Budget

S.N.	Particulars	Price (Rs)	Quantity	Total Price (Rs)
1.	Server Hosting	2000 per month	1	2000
2.	Acquiring a domain name	1500 per year	1	1500
TOTAL				3500

REFERENCES

[1] "Annual Report," University Grants Commission, 2021.

[2] S. Agarwal and S. Rajan, "Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries," *Spatial Information Research,* vol. 24, no. 6, pp. 671-677, 2016.

[3] B. Cheng and X. Guan, "Design and evaluation of a high-concurrency web map tile service framework on a high," *International Journal of Grid and Distributed Computing,* vol. 9, no. 12, pp. 127-142, 2016.

[4] J. Ferreira, J. Noble and R. Biddle, "Agil. 2007," in *Agile Development Iterations and UI Design,* United States, IEEE Computer Society, 2007, pp. 50-58.

[5] E. Baralis, A. Valle, P. Garza, C. Rossi and F. Scullino, "SQL versus NoSQL Databases for Geospatial Applications," in *In Proceedings of the 2017 IEEE International Conference on Big Data,* Boston, 2017.