SMARTBRIDGE EXTERNSHIP PROGRAM

MODERN APPLICATION DEVELOPMENT (JAVA SPRING BOOT)

PROJECT REPORT

Topic: **HOUSE RENT WEB APPLICATION**

Submitted by:

Vaishnavi Sabarish (20BCE2230)
Abhishek S Mattam (20BCI0162)
Karthik Ajay (20BCI0179)
Arunima Mohanan (20BCE0357)

VIT Vellore

1. INTRODUCTION

1.1. OVERVIEW

The house rental web application is a user-friendly platform designed to simplify the process of finding and renting houses. It provides a comprehensive database of available rental properties, allowing users to search, filter, and compare listings based on their preferences and requirements. The application offers detailed property information, including photos, amenities, and pricing, ensuring users make informed decisions. With its intuitive interface and convenient features, the house rental web application streamlines the rental process, making it efficient and convenient for both tenants and landlords.

1.2. PURPOSE

The project has been developed to serve the following purposes:

- Enhance Business Processes: To be able to use internet technology to project the rental company to the global world instead of limiting their services to their local domain alone, thus increasing their return on investment.
- Online Booking: A tool through which customers can book available Rooms/House online prior to their date of using the house instead of walking around and asking for a vacant house.
- Customer's registration: A registration portal to hold customer's details, monitor their transaction and use the same to offer better and improve services to them and user accounts where he/she can view her/his details.

2. LITERATURE SURVEY

1. "Building a Property Management System using Spring Boot and Angular" by Ankit Kumar:

This article provides a step-by-step guide to building a property management system using Java Spring Boot on the backend and Angular on the frontend. While the focus is on property management, many concepts and techniques covered can be applied to a rent management system.

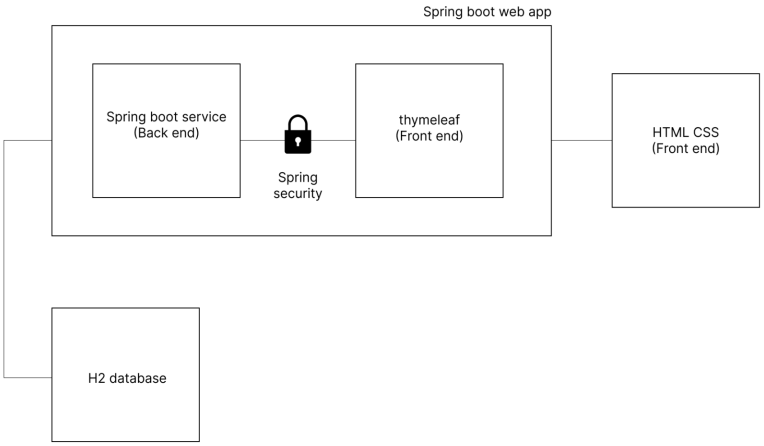
2. "Rental Property Management System using Spring Boot and React" by Siddharth Singh:
This tutorial demonstrates how to develop a rental property management system using Java Spring Boot on the backend and React on the frontend. Although it focuses on property management, the core functionalities, such as managing tenants, properties, leases, and payments, can be adapted for a rent management system.

3. "A Comparative Study of Rent Management Systems" by Laura Johnson:
This research paper compares and analyzes different rent management systems available in the market. It evaluates their features, usability, scalability, and performance. This study can provide insights into the existing solutions and help in identifying best practices and design considerations for developing a rent management system using Java Spring Boot.

4. "Design and Implementation of a Web-Based Property Management System" by John Doe et al.:
This academic paper presents the design and implementation of a web-based property management system. While the primary focus is on property management, it includes components like tenant management, lease agreements, and rent payment tracking.

3. THEORETICAL ANALYSIS

3.1. BLOCK DIAGRAM



3.2. HARDWARE/ SOFTWARE DESIGNING

Hardware Requirements:

Processor	Pentium II or higher
Processor Speed	533MHz
Hard Disk Space	20GB (min)
RAM memory	32MB

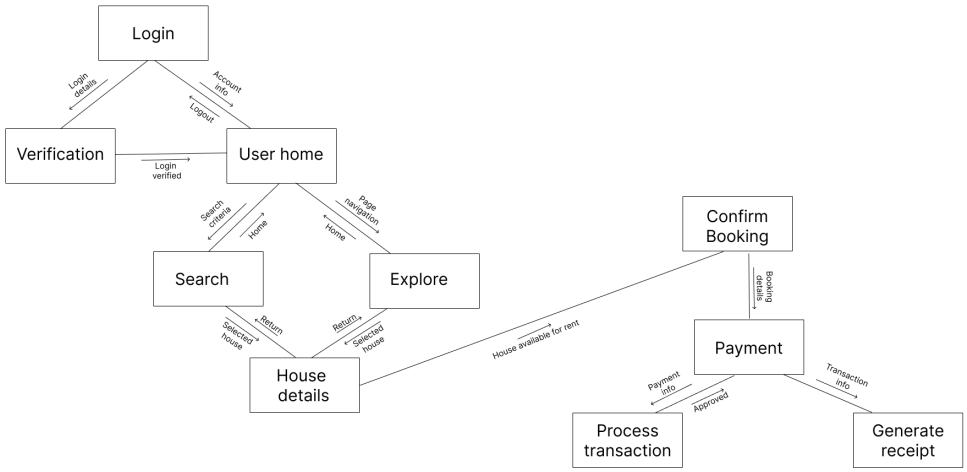
Software Requirements:

Operating System	Windows 10
Front end	VSCode
Back end	spring boot
Database server	MySQL

4. EXPERIMENTAL INVESTIGATIONS

The development of an online house rental web application involves several critical analyses and investigations. Market research is conducted to assess the existing landscape, competitors, and identify market opportunities. User needs analysis is performed through surveys, interviews, and feedback sessions to understand user requirements and pain points. Technical feasibility analysis helps determine the appropriate technologies and infrastructure required for development. Data analysis provides insights into rental property trends and informs decision-making. Security and privacy analysis ensures the protection of user data and identifies potential vulnerabilities. Payment integration investigations examine reliable and compatible options for secure transactions. User experience design involves usability testing to create an intuitive interface. Performance testing assesses the web application's ability to handle high traffic and maintain system stability. Legal and compliance investigations ensure adherence to relevant regulations. By conducting these analyses and investigations, a robust and user-centric online house rental web application that meets market demands, provides a secure experience, and complies with legal requirements has been developed.

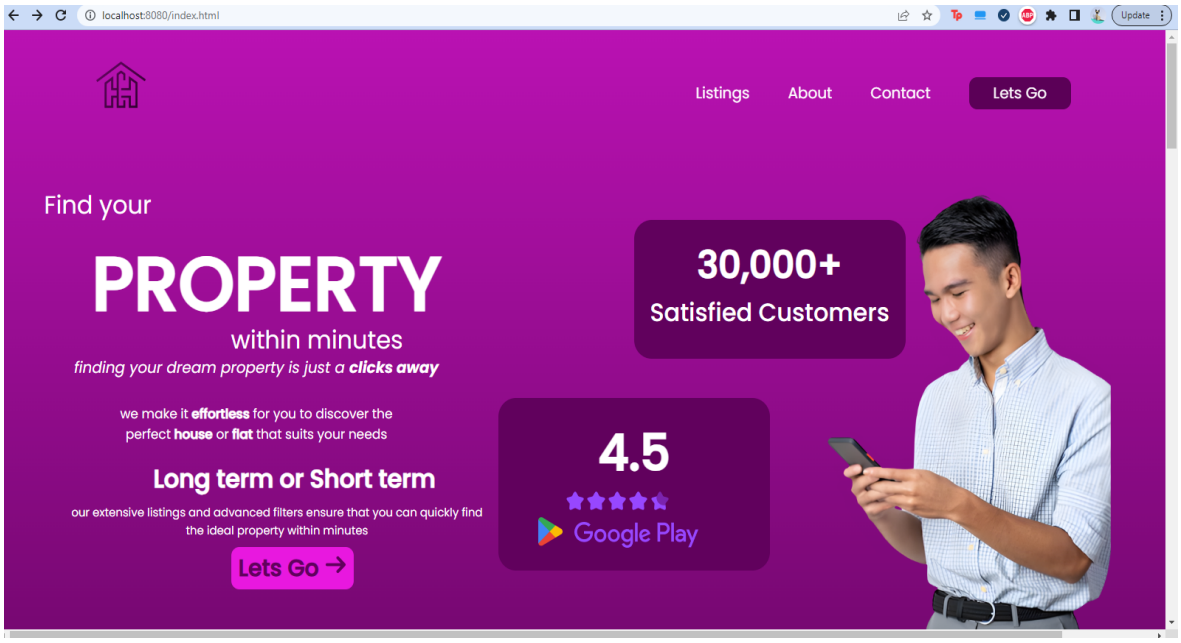
5. FLOWCHART

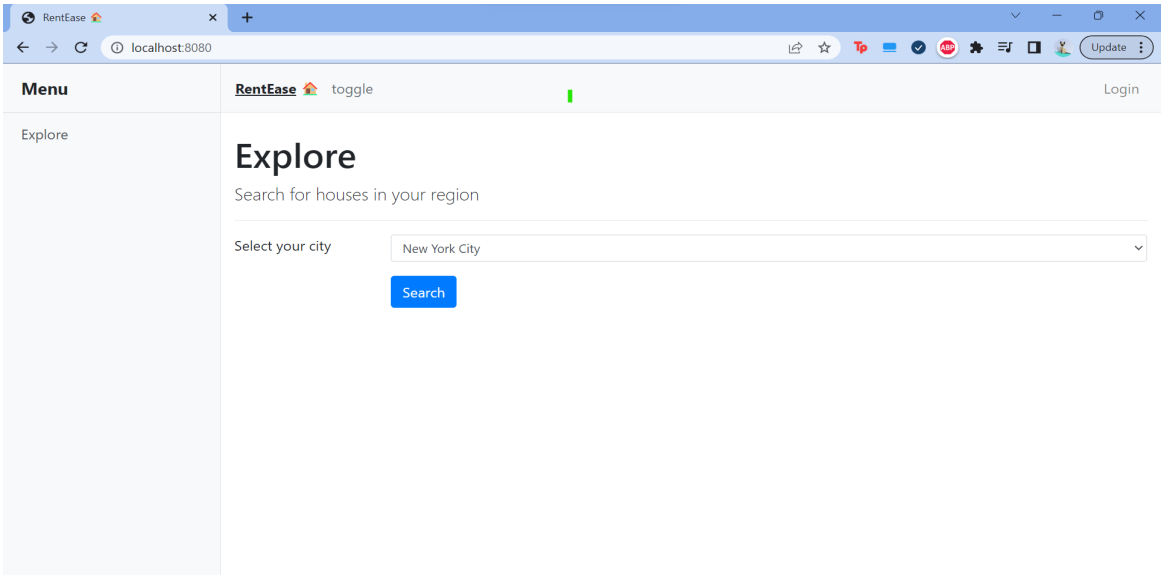


6. RESULT

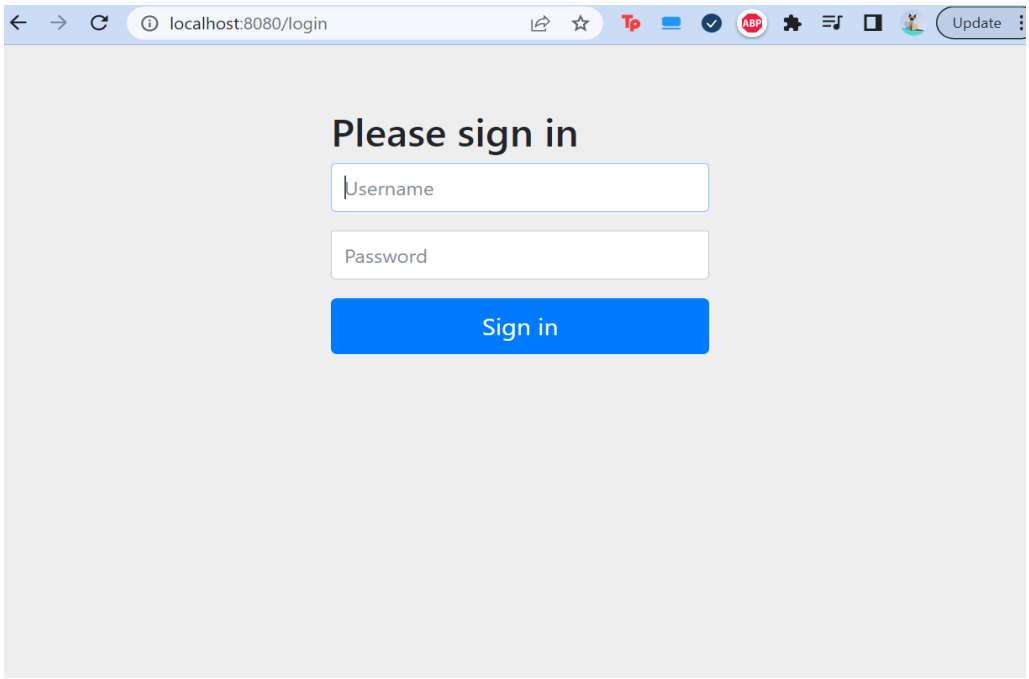
Successfully developed House-Rental application using java spring-boot and thymeleaf and H2 database.

Screenshots of the web application developed :

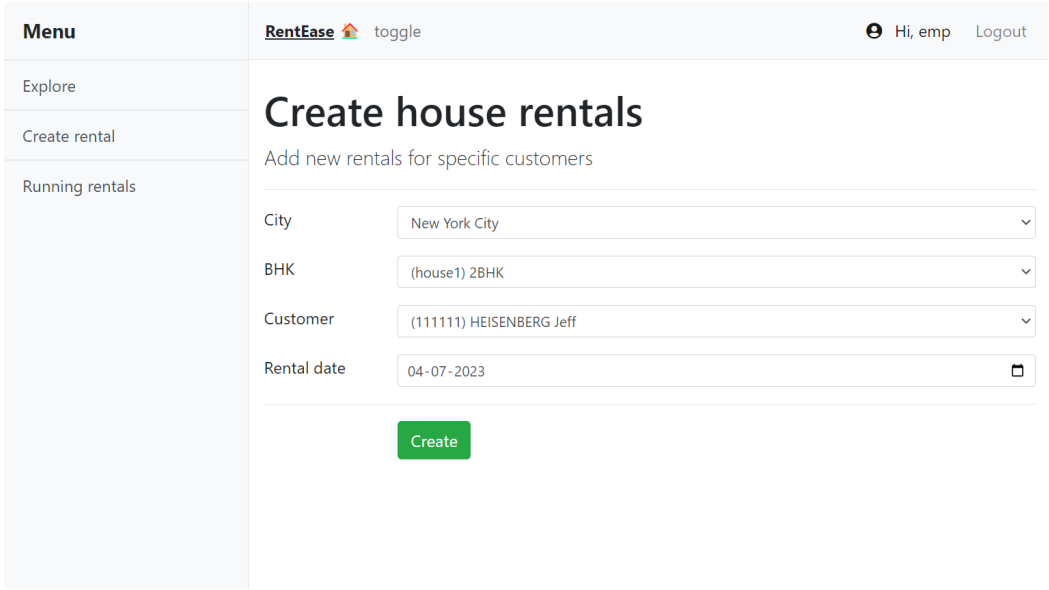




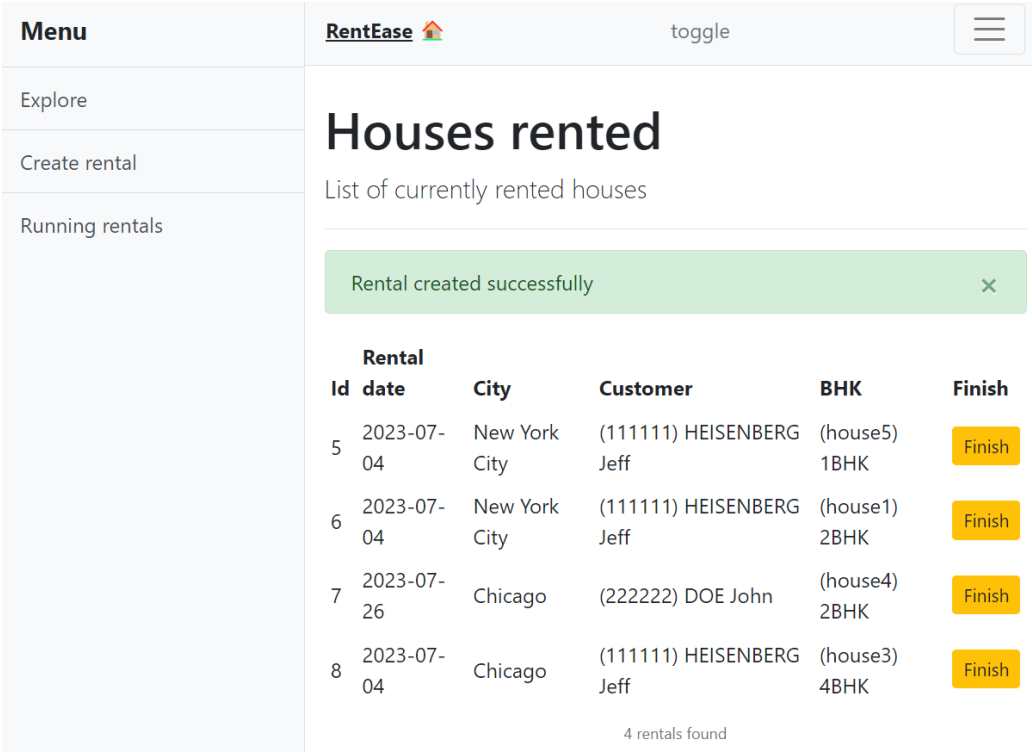
Explore page of the house rental application



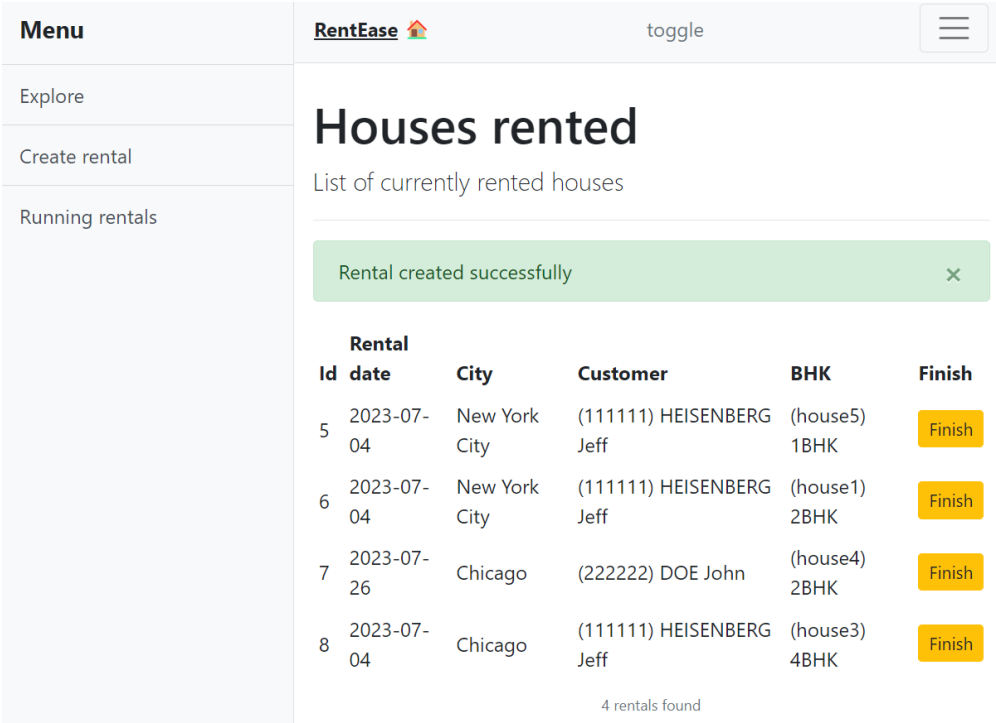
Sign in page



Employee side

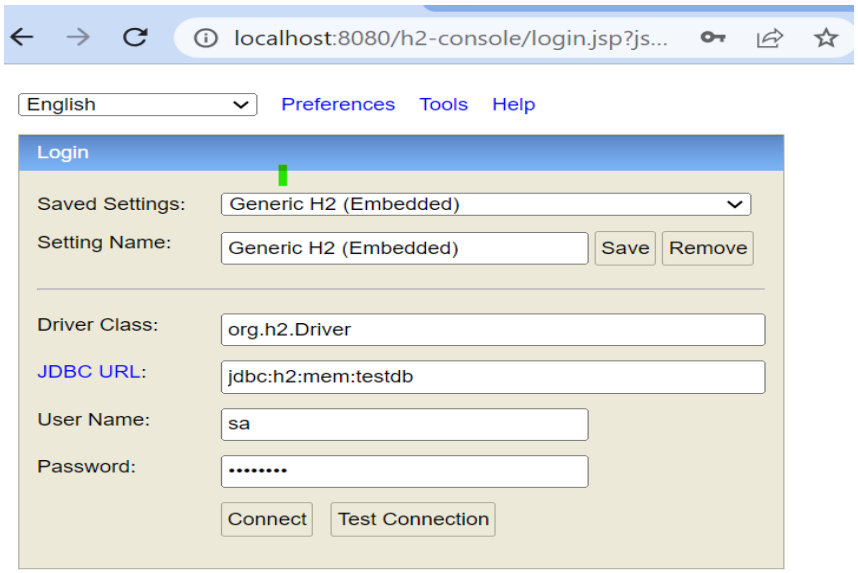


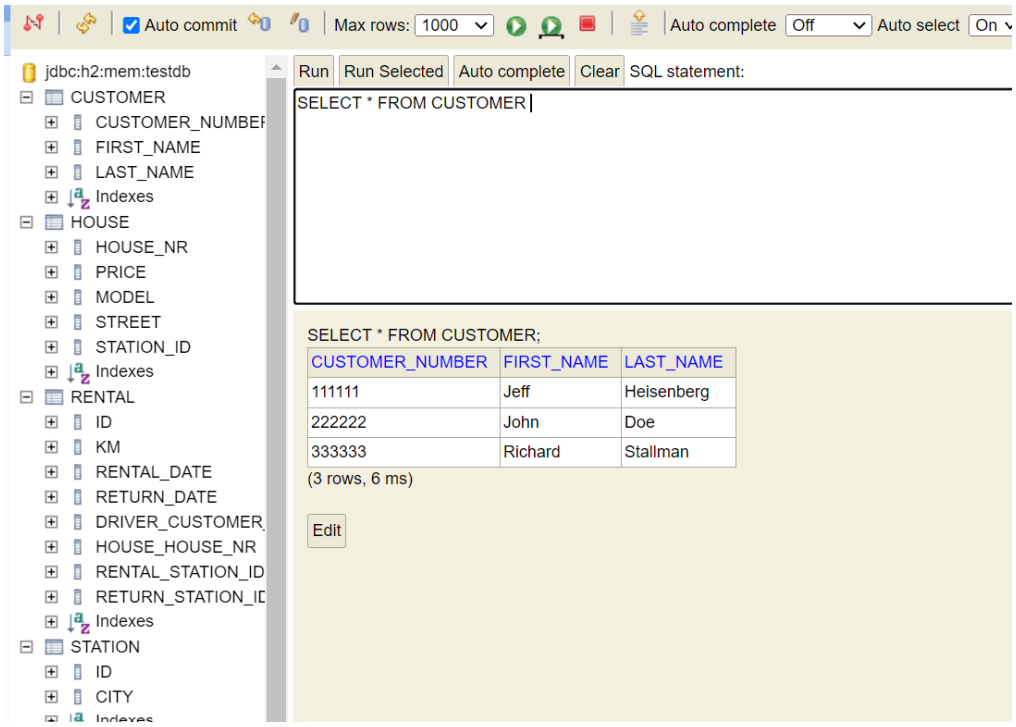
Creating a rental



Running rentals under the employee

Database:





H2 database console showing the tables with sql query

7. ADVANTAGES AND DISADVANTAGES

Advantages:

- Time and Cost Savings: The application streamlines the rental process by automating tasks such as property searching, online applications, document uploads, and online payments. This saves time for both landlords and tenants, reducing administrative burdens and paperwork. Additionally, it eliminates the need for brokers or intermediaries, reducing associated costs.
- Security and Verification: The application implements user verification processes and security measures to ensure the authenticity of users and protect personal information. This enhances trust and reduces the risk of fraudulent activities.
- Easy Access and Convenience: Users can access the application from anywhere and at any time, making it convenient for both landlords to list their properties and tenants to search for rentals. It eliminates the need for physical visits to rental agencies or property showings.
- Detailed Property Information: Online house rental applications provide detailed property information, including high-quality photos, floor plans, virtual tours, and amenities. This helps tenants make informed decisions without having to physically visit each property.

Disadvantages:

- Limited Physical Inspection: One major drawback is the limited ability for tenants to physically inspect the property before renting. While virtual tours and photos provide some insight, they may not always accurately represent the true condition or ambiance of the property.
- Lack of Personalized Assistance: Online applications may lack the personalized assistance and guidance that tenants would receive from a real estate agent or rental agency. Some individuals may prefer the guidance and expertise provided by a human intermediary during the rental process.
- Reliance on Technology and Internet Access: An online rental application heavily relies on technology and internet access. If tenants

or landlords encounter technical issues or do not have reliable internet connections, it can hinder their ability to use the application effectively.
- Potential Security Risks: While online applications often implement security measures, there is always a potential risk of data breaches, hacking, or unauthorized access to personal and financial information.

8. APPLICATIONS

An online house rental web application has various applications and can be used in several scenarios. Here are some key applications:

- Tenant House Search: The primary application is to provide tenants with a platform to search and find rental properties based on their specific requirements, such as location, budget, size, amenities, and other preferences.
- Property Listing and Management: Landlords can use the web application to list and manage their rental properties. They can create property profiles, upload photos, provide detailed descriptions, set rental prices, update availability, and communicate with potential tenants.
- Rental Payments: Online house rental application integrates secure payment gateways, allowing tenants to make rental payments conveniently and securely. This eliminates the need for physical checks or cash transactions and provides a transparent record of payment history.
- Booking and Reservation: The web application enables tenants to book and reserve rental properties directly through the platform. It facilitates the secure exchange of rental agreements, payment

processing, and confirmation of bookings, streamlining the rental transaction process for both parties.

9. CONCLUSION

In conclusion, an online house rental web application project brings significant advantages to both tenants and landlords. The convenience and accessibility offered by the application simplify the house rental process, allowing tenants to easily search for properties that meet their needs and landlords to efficiently manage their rental listings. The platform provides a wide range of options, detailed property information, and user reviews, empowering tenants to make informed decisions. It saves time and reduces costs by automating tasks such as property searching, communication, and payment processing. For landlords, the application expands their reach, increases visibility for their rental properties, and streamlines the tenant screening and booking process. The platform enhances transparency and trust by providing a secure environment for rental transactions, document exchange, and user feedback.

10. FUTURE SCOPE

There are several enhancements that can be made to further improve the application. Here are some potential areas for development:

- Enhanced Search Filters: Introduce advanced search filters to allow tenants to refine their search based on specific criteria such as property size, number of bedrooms, amenities, proximity to certain locations, and more. This would provide a more tailored and personalized search experience.
- Machine Learning and Recommendation Systems: Utilize machine learning algorithms to analyze user preferences, behavior, and historical data to provide personalized property recommendations.

This can help tenants discover properties that align with their preferences and increase overall user satisfaction.

- Expanded Payment Options: Integrate additional payment options, such as digital wallets, cryptocurrencies, or installment plans, to offer more flexibility for tenants and accommodate their preferred payment methods.
- Multilingual Support: Provide multilingual support to cater to a diverse user base, allowing tenants and landlords to access and use the platform in their preferred language, thereby enhancing inclusivity and user engagement.

11. BIBLIOGRAPHY

APPENDIX

Source code:

application.java

```
package at.htlstp.aslan.houserent;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HouseRentApplication {

    public static void main(String[] args) {
        SpringApplication.run(HouseRentApplication.class, args);
    }

}
```

Service.java file

```
package at.htlstp.aslan.houserent.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import at.htlstp.aslan.houserent.model.House;
```

```java
import at.htlstp.aslan.houserent.model.Station;
import at.htlstp.aslan.houserent.repository.HouseRepository;
import at.htlstp.aslan.houserent.util.MessagesBean;

import javax.persistence.EntityExistsException;
import javax.persistence.EntityNotFoundException;
import java.util.List;

@Service
public class HouseService {

    @Autowired
    private MessagesBean messages;

    @Autowired
    private HouseRepository houseRepository;

    @Autowired
    private StationService stationService;

    @Autowired
    private RentalService rentalService;

    /**
     * @param station the station to be searched for
     * @return all Houses that are part of the given station
     */
    public List<House> findByStation(Station station) {
        return houseRepository.findByStation(station);
    }

    /**
     * @param street the minimum street to be searched for
     * @return all Houses that have a street greater than the given one
     */
    // public List<House> findByStreetGreaterThan(String street) {
    // return houseRepository.findByStreetGreaterThan(street);
    // }

    /**
     * Saves the given House.
     *
     * @param House House to be saved
     * @return the saved House coming from the database (only when no exceptions
     *         occur)
     * @throws EntityNotFoundException  if the provided station of the House does
     *                                  not exist
     * @throws EntityExistsException    if the given primary key already belongs to
     *                                  an existing entity
     * @throws IllegalArgumentException if the station is null
     */
    public House create(House house) {
        if (house.getStation() == null) {
            throw new IllegalArgumentException(messages.get("HouseStationNotNull"));
        }
```

```java
        if (house.getStation().getId() == null ||
!stationService.existsById(house.getStation().getId())) {
            throw new EntityNotFoundException(messages.get("stationNotFound"));
        }
        if (houseRepository.existsById(house.getHouseNr())) {
            throw new EntityExistsException(messages.get("HouseAlreadyExists"));
        }
        return houseRepository.save(house);
    }

    /**
     * Deletes a House with the given primary key.
     * However, the House must be deletable. See {@link #canDelete(House)}
     *
     * @param houseNr the primary key to be searched for
     * @throws EntityNotFoundException  if the given primary key does not belong to
     *                                  any existing entity
     * @throws IllegalArgumentException if the House cannot be deleted
     */
    public void deleteById(String houseNr) {
        House house = houseRepository.findById(houseNr)
                .orElseThrow(() -> new EntityNotFoundException(messages.get("HouseNotFound")));
        if (!canDelete(house)) {
            throw new IllegalArgumentException(messages.get("HouseDeleteError"));
        }
        houseRepository.delete(house);
    }

    /**
     * Checks whether a given House can be deleted or not.
     * The House can be deleted, if, and only if, the station House is not in use
     * (station equals null) and the House was never used in any rental before.
     * @param House the House to be checked
     * @return true, if the House fulfills the mentioned criteria, false otherwise.
     */
    public boolean canDelete(House house) {
        return house.getStation() != null && rentalService.findByHouse(house).isEmpty();
    }
}
```