Le Nid

Vacation rental platform

(mini-Airbnb)

Team members:

Ali Ravanbakhsh

Sarah Niazalizadeh Moghadam

Le Nid

MONTRÉAL • SEJOURS ACOGLANTS

Background of the project

The Market Need	The Solution
Need for cost-effective, Localized accommodation	P2P Sharing Economy Platform
Owner need to Monetize Unused Assets	

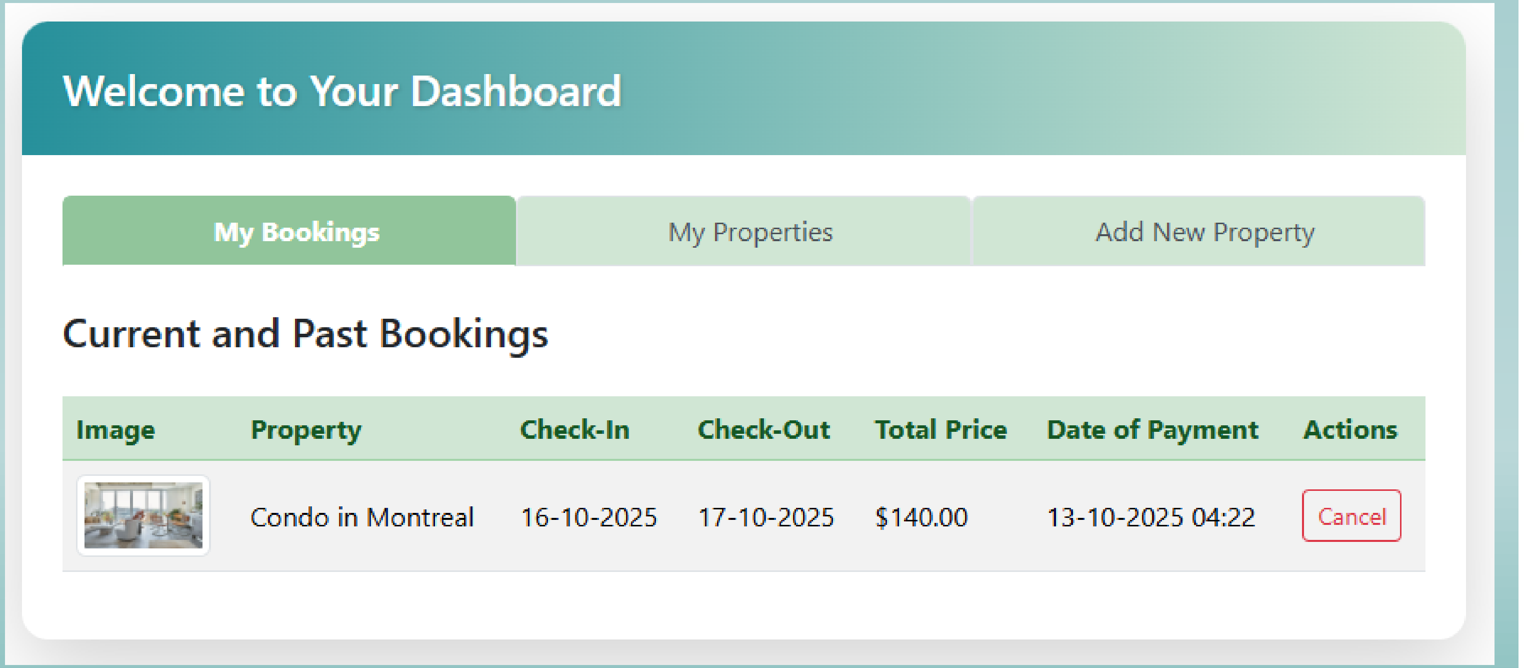
Core Value: Digital Intermediary

Key Function: Real-Time Matching by Location and Date

Result: Inventory Aggregation without Ownership

Solution Overview: Key User Journeys

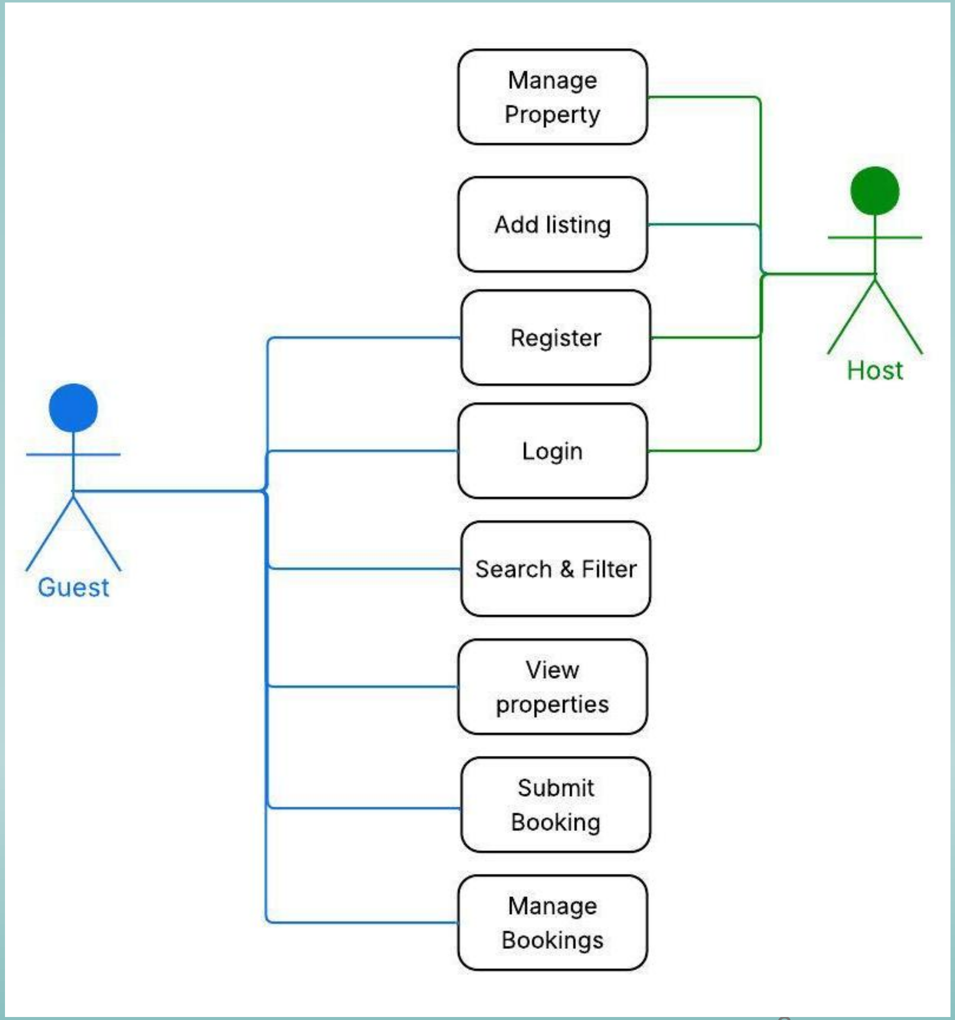
- Dual-role system: **Guest** and **Host.**


Welcome to Your Dashboard

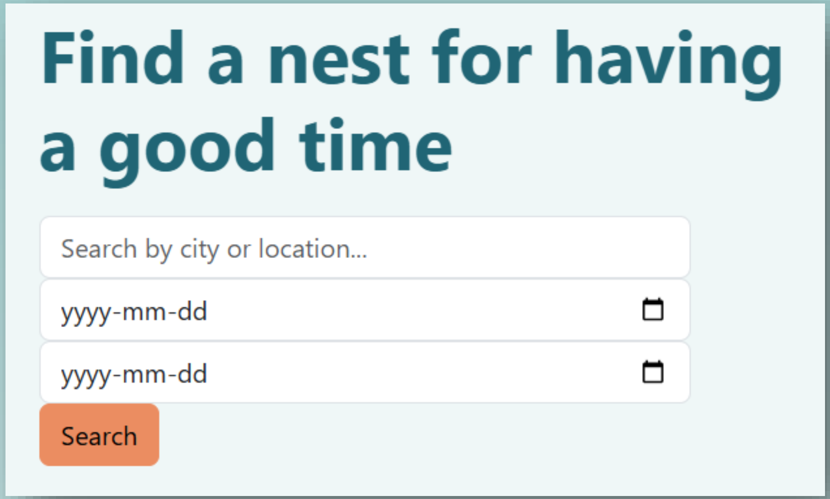
My Bookings | My Properties | Add New Property

Current and Past Bookings

Image	Property	Check-In	Check-Out	Total Price	Date of Payment	Actions
	Condo in Montreal	16-10-2025	17-10-2025	$140.00	13-10-2025 04:22	Cancel



App Flow: Search and Filter



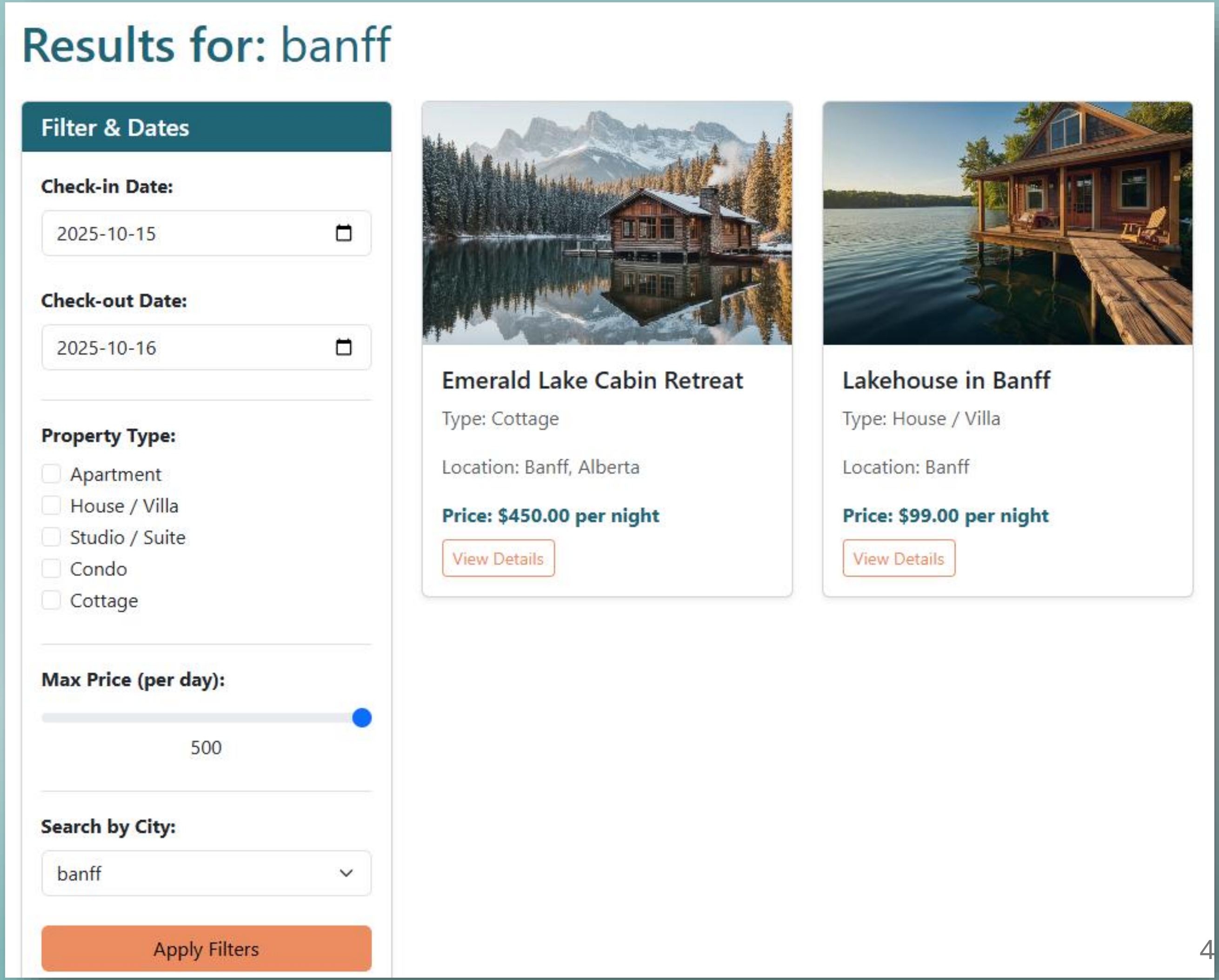
Find a nest for having a good time

Search by city or location...

yyyy-mm-dd

yyyy-mm-dd

Search



Results for: banff

Filter & Dates

Check-in Date: 2025-10-15

Check-out Date: 2025-10-16

Property Type:

- Apartment
- House / Villa
- Studio / Suite
- Condo
- Cottage

Max Price (per day): 500

Search by City: banff

Apply Filters

Emerald Lake Cabin Retreat

Type: Cottage

Location: Banff, Alberta

Price: $450.00 per night

View Details

Lakehouse in Banff

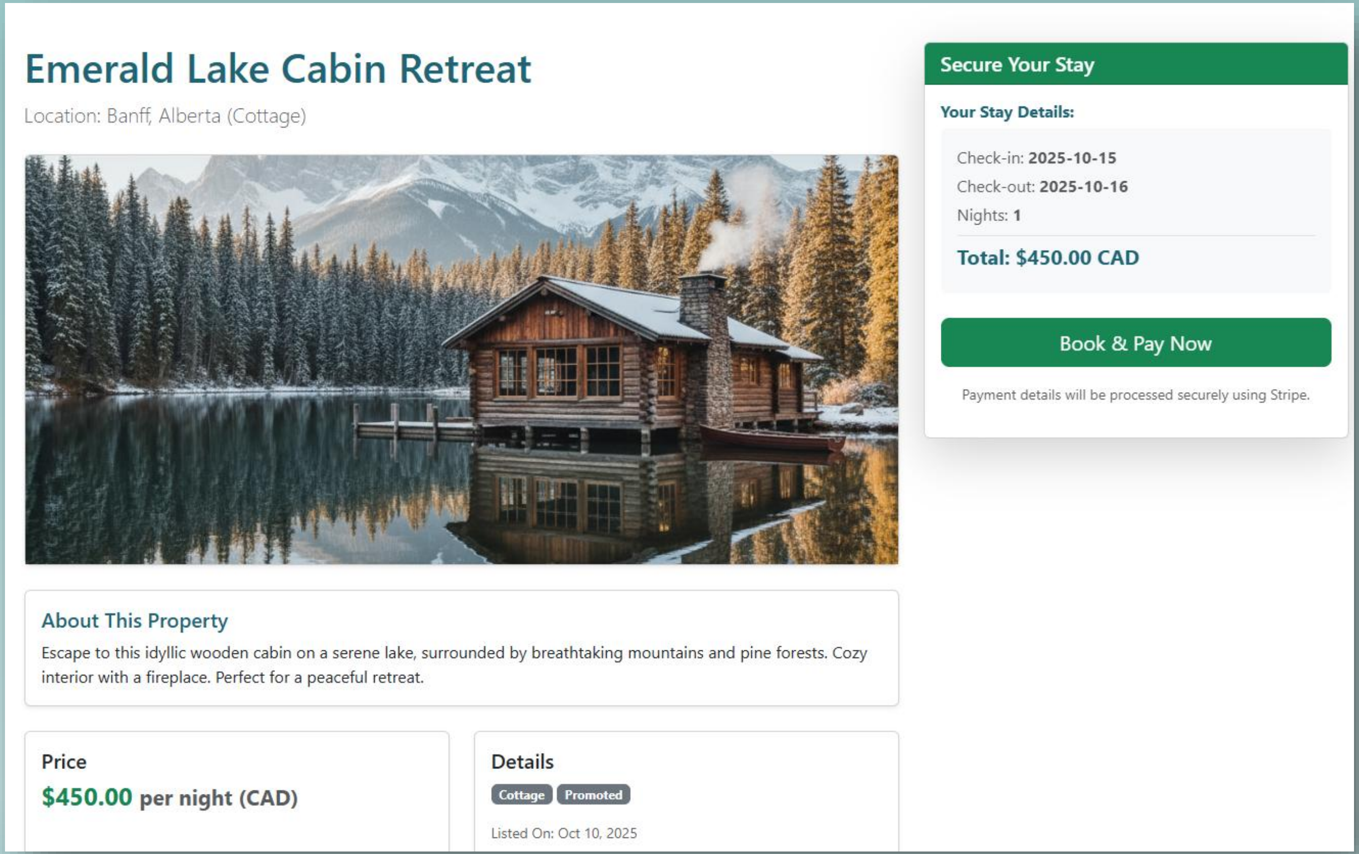
Type: House / Villa

Location: Banff

Price: $99.00 per night

View Details

App Flow: Book and manage booking



Emerald Lake Cabin Retreat

Location: Banff, Alberta (Cottage)

About This Property

Escape to this idyllic wooden cabin on a serene lake, surrounded by breathtaking mountains and pine forests. Cozy interior with a fireplace. Perfect for a peaceful retreat.

Price

$450.00 per night (CAD)

Details

Cottage Promoted

Listed On: Oct 10, 2025

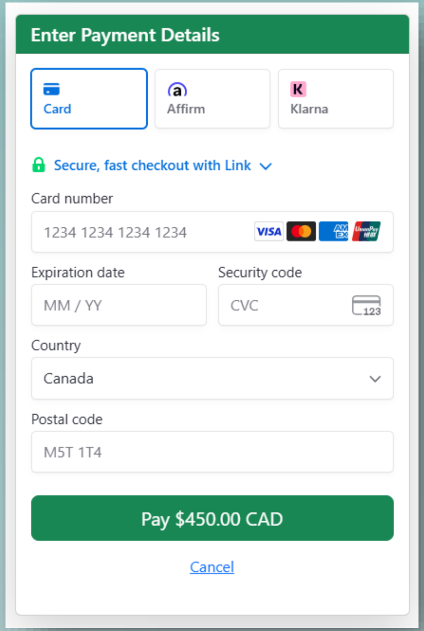
Secure Your Stay

Your Stay Details:

Check-in: **2025-10-15**
Check-out: **2025-10-16**
Nights: **1**

Total: $450.00 CAD

Payment details will be processed securely using Stripe.



Enter Payment Details

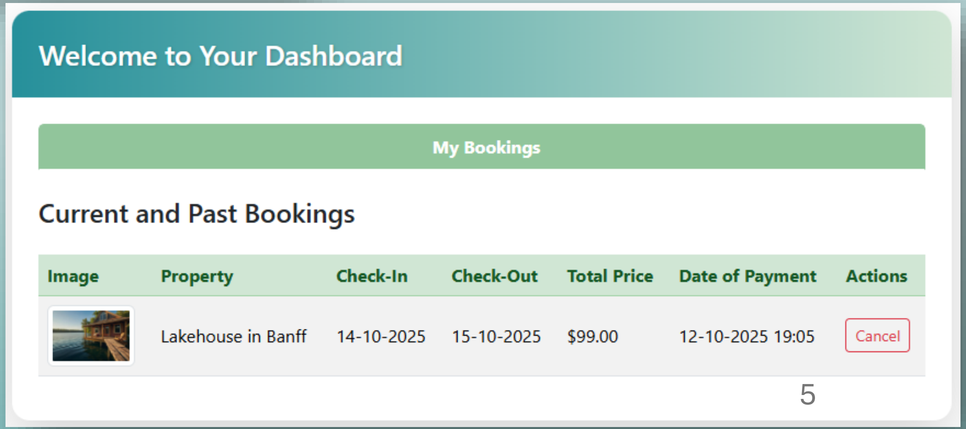
Card number: 1234 1234 1234 1234

Expiration date: MM / YY

Security code: CVC

Country: Canada

Postal code: M5T 1T4

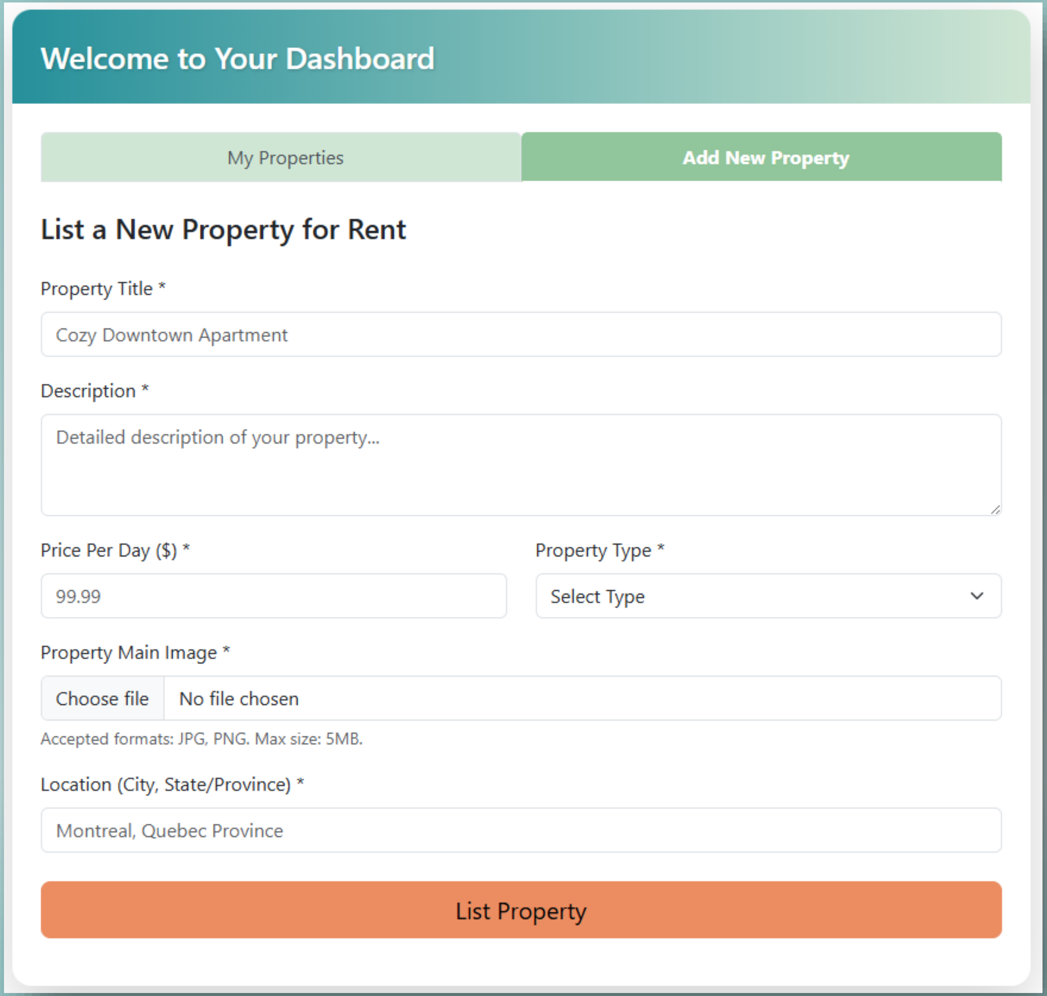


Welcome to Your Dashboard

Current and Past Bookings

Image	Property	Check-In	Check-Out	Total Price	Date of Payment	Actions
	Lakehouse in Banff	14-10-2025	15-10-2025	$99.00	12-10-2025 19:05	Cancel

App Flow: Add and manage properties



Welcome to Your Dashboard

My Properties | **Add New Property**

List a New Property for Rent

Property Title *

Cozy Downtown Apartment

Description *

Detailed description of your property...

Price Per Day ($) *

99.99

Property Type *

Select Type

Property Main Image *

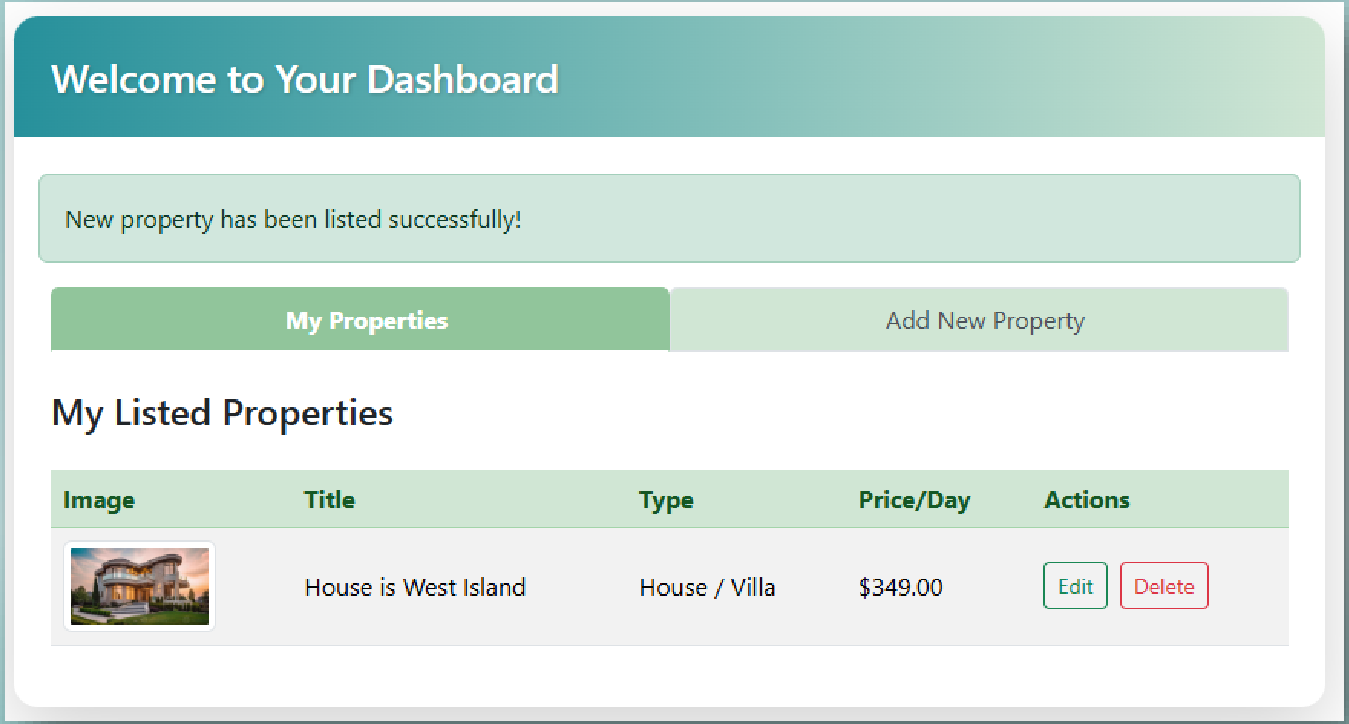
Choose file No file chosen

Accepted formats: JPG, PNG. Max size: 5MB.

Location (City, State/Province) *

Montreal, Quebec Province

List Property

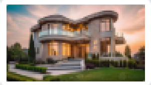


Welcome to Your Dashboard

New property has been listed successfully!

My Properties | Add New Property

My Listed Properties

Image	Title	Type	Price/Day	Actions
	House is West Island	House / Villa	$349.00	Edit Delete

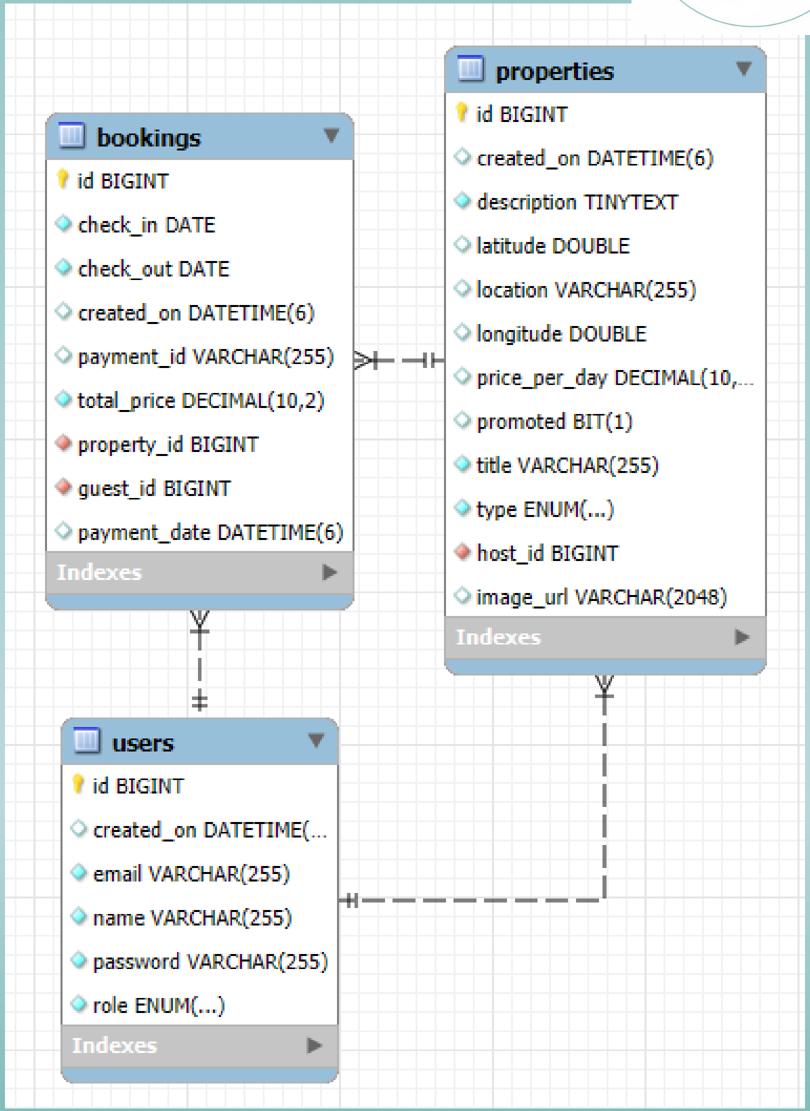
Technology Stack & Project Management

- **Backend:** Java Spring Boot
- **Frontend:** Thymeleaf, JavaScript, AJAX
- **Database/Storage:** MySQL on AWS RDS, AWS S3
- **Integration:** Stripe Payment API
- **Hosting:** Heroku
- **Code Management:** GitHub
- **Project Management:** Trello and Daily Scrum

Database Structure: MySQL on AWS RDS

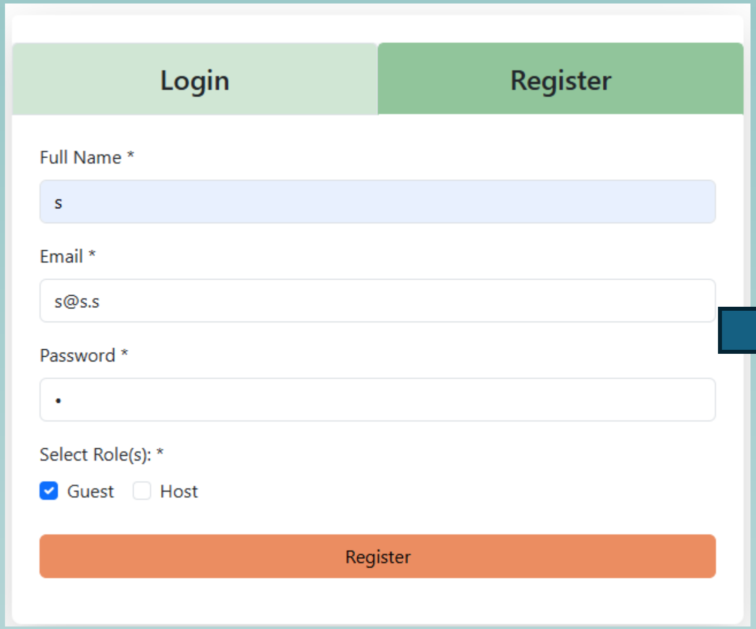
- Relational model with key tables for:
  - **Users**,
  - **Properties**, and
  - **Bookings**.

- Photos stored on **AWS S3**



```
bookings
id BIGINT
check_in DATE
check_out DATE
created_on DATETIME(6)
payment_id VARCHAR(255)
total_price DECIMAL(10,2)
property_id BIGINT
guest_id BIGINT
payment_date DATETIME(6)
Indexes

properties
id BIGINT
created_on DATETIME(6)
description TINYTEXT
latitude DOUBLE
location VARCHAR(255)
longitude DOUBLE
price_per_day DECIMAL(10,...
promoted BIT(1)
title VARCHAR(255)
type ENUM(...)
host_id BIGINT
image_url VARCHAR(2048)
Indexes

users
id BIGINT
created_on DATETIME(...
email VARCHAR(255)
name VARCHAR(255)
password VARCHAR(255)
role ENUM(...)
Indexes
```

Challenge #1- Registration Validation


Login | Register

Full Name *

s

Email *

s@s.s

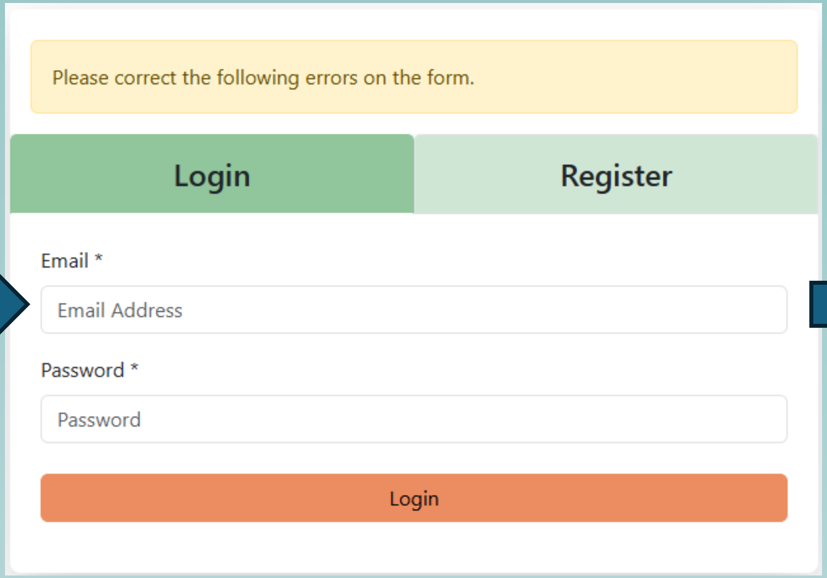
Password *

•

Select Role(s): *

Guest Host

Register


Please correct the following errors on the form.

Login | Register

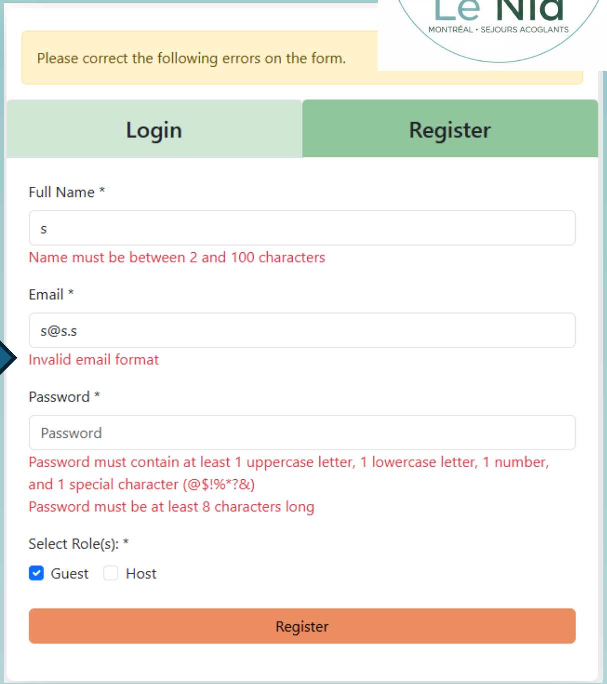
Email *

Email Address

Password *

Password

Login


Please correct the following errors on the form.

Login | Register

Full Name *

s

Name must be between 2 and 100 characters

Email *

s@s.s

Invalid email format

Password *

Password

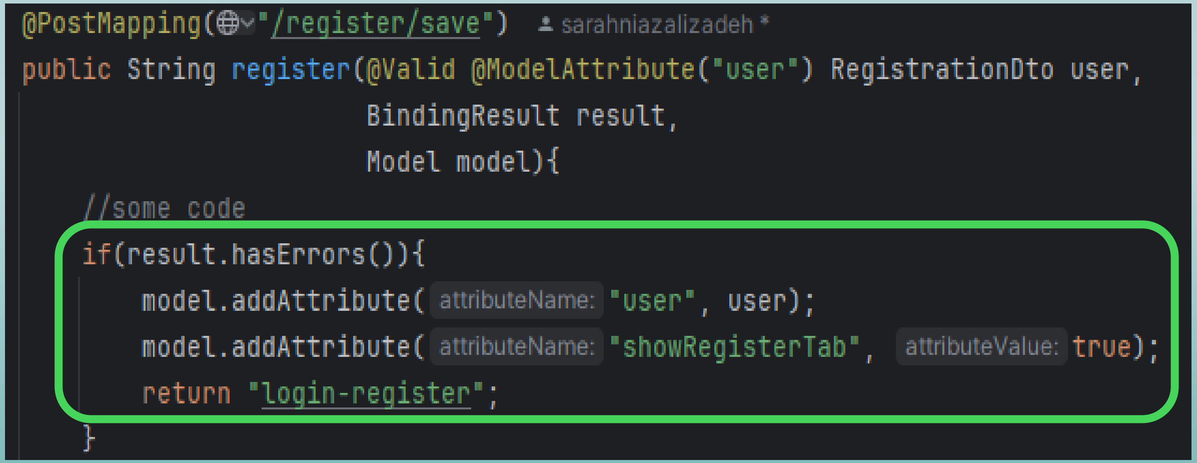
Password must contain at least 1 uppercase letter, 1 lowercase letter, 1 number, and 1 special character (@$!%*?&)

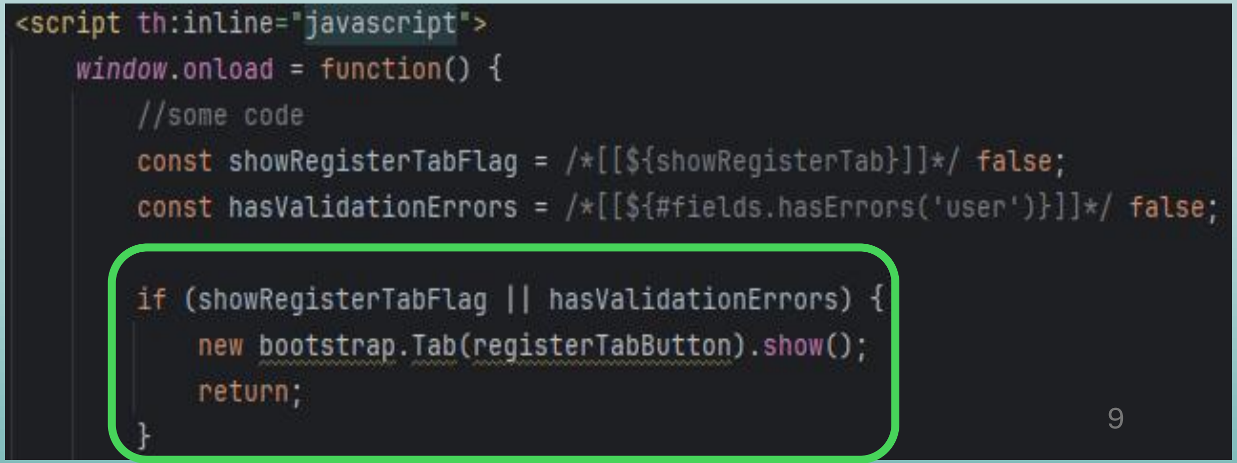
Password must be at least 8 characters long

Select Role(s): *

Guest Host

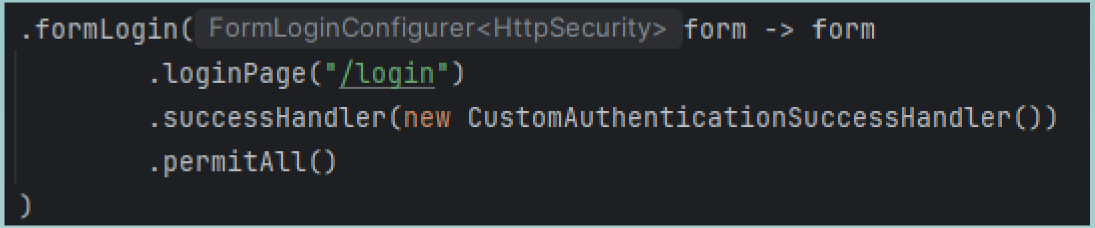
Register


```java
@PostMapping("/register/save")  sarahniazalizadeh *
public String register(@Valid @ModelAttribute("user") RegistrationDto user,
                       BindingResult result,
                       Model model){
    //some code
    if(result.hasErrors()){
        model.addAttribute( attributeName: "user", user);
        model.addAttribute( attributeName: "showRegisterTab",  attributeValue: true);
        return "login-register";
    }
```


```javascript
<script th:inline="javascript">
    window.onload = function() {
        //some code
        const showRegisterTabFlag = /*[[${showRegisterTab}]]*/ false;
        const hasValidationErrors = /*[[${#fields.hasErrors('user')}]]*/ false;

        if (showRegisterTabFlag || hasValidationErrors) {
            new bootstrap.Tab(registerTabButton).show();
            return;
        }
```

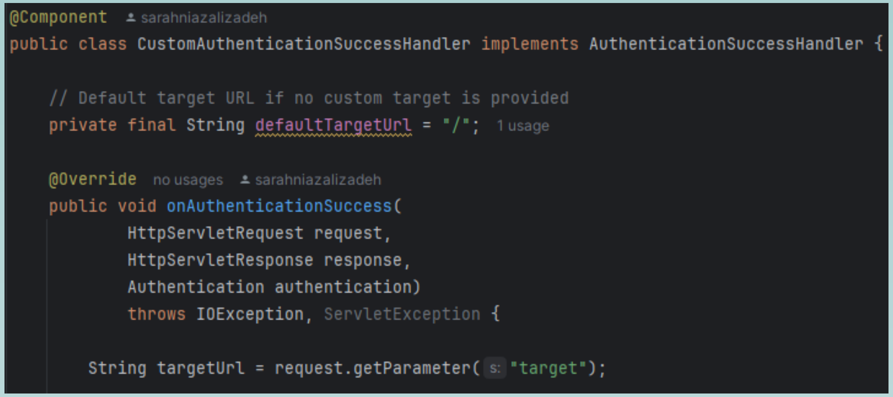
Challenge #2- Ajax login redirect to index

Security Config:



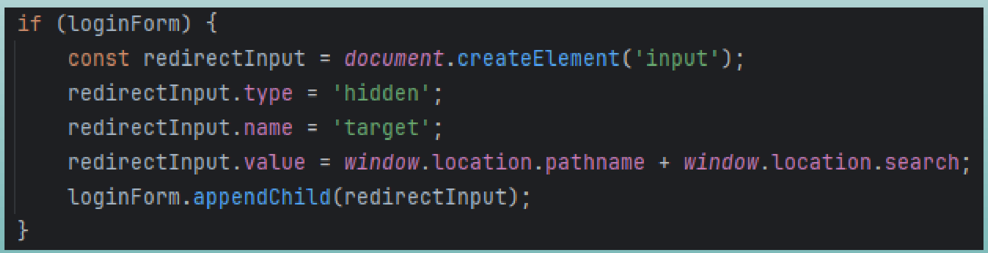
```
.formLogin( FormLoginConfigurer<HttpSecurity> form -> form
        .loginPage("/login")
        .successHandler(new CustomAuthenticationSuccessHandler())
        .permitAll()
)
```

Custom Authentication Success Handler:

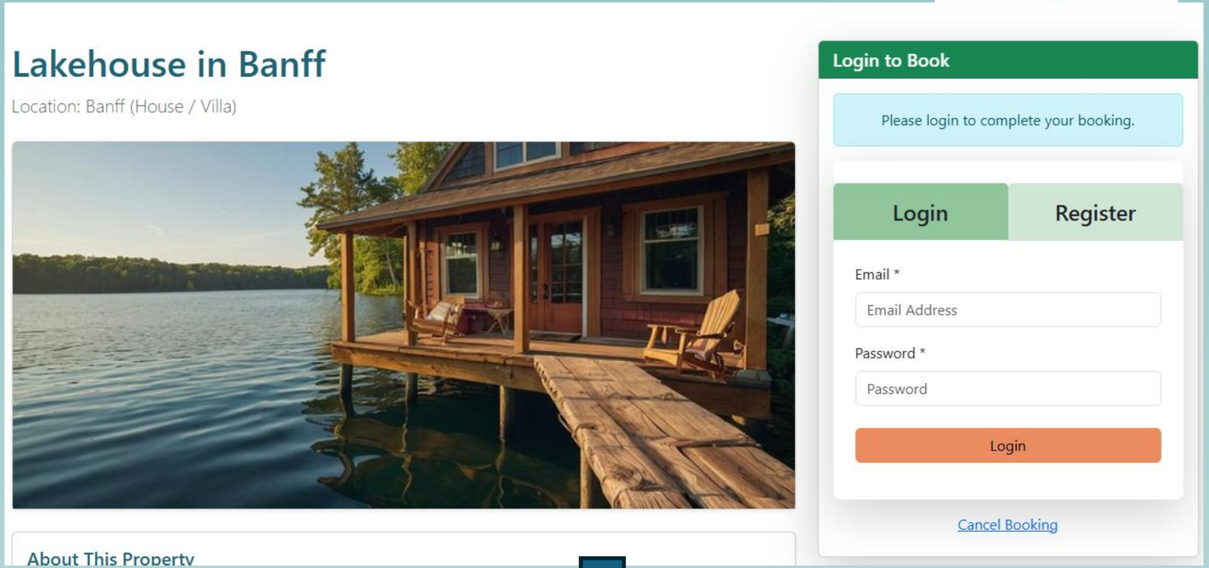


```
@Component  sarahniazalizadeh
public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {

    // Default target URL if no custom target is provided
    private final String defaultTargetUrl = "/";  1 usage

    @Override  no usages  sarahniazalizadeh
    public void onAuthenticationSuccess(
            HttpServletRequest request,
            HttpServletResponse response,
            Authentication authentication)
            throws IOException, ServletException {

        String targetUrl = request.getParameter( s: "target");
```

Property-details.html:



```
if (loginForm) {
    const redirectInput = document.createElement('input');
    redirectInput.type = 'hidden';
    redirectInput.name = 'target';
    redirectInput.value = window.location.pathname + window.location.search;
    loginForm.appendChild(redirectInput);
}
```



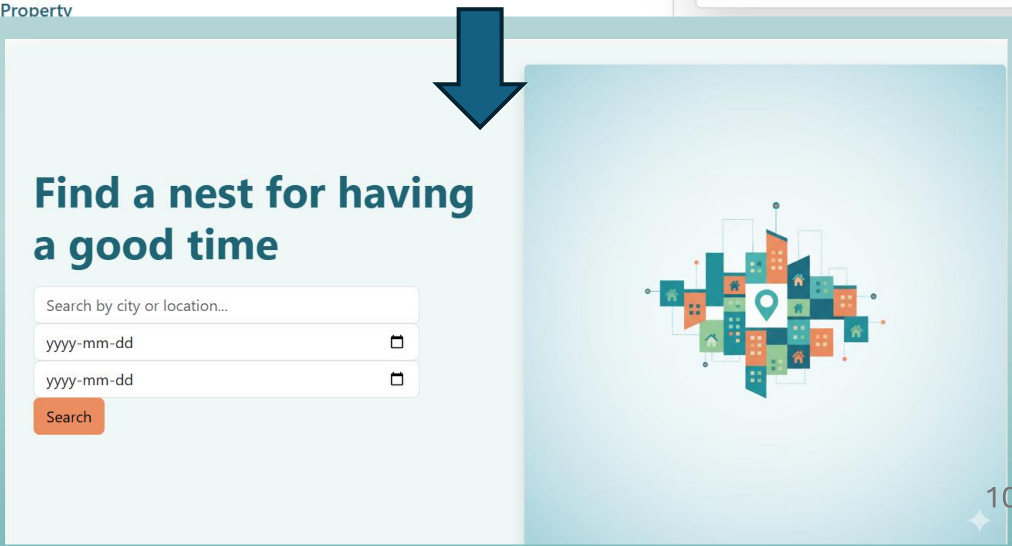
Lakehouse in Banff

Location: Banff (House / Villa)

About This Property

Login to Book

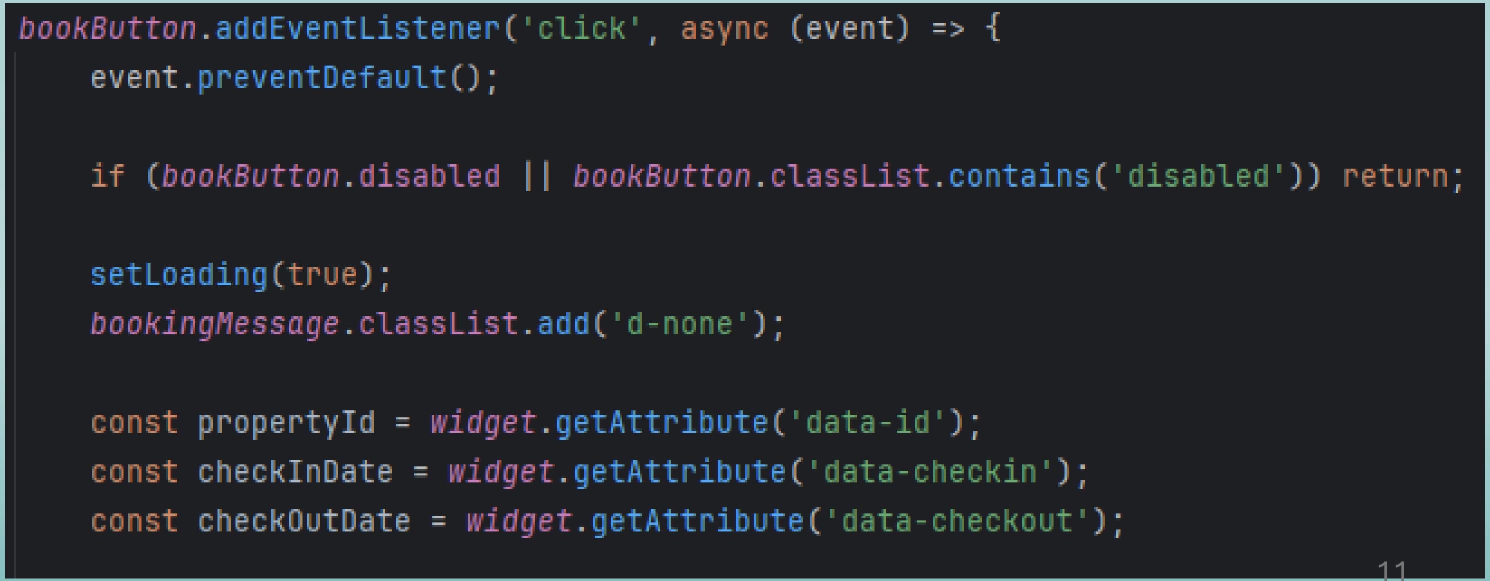
Please login to complete your booking.



Find a nest for having a good time

Teaching Point 1: AJAX

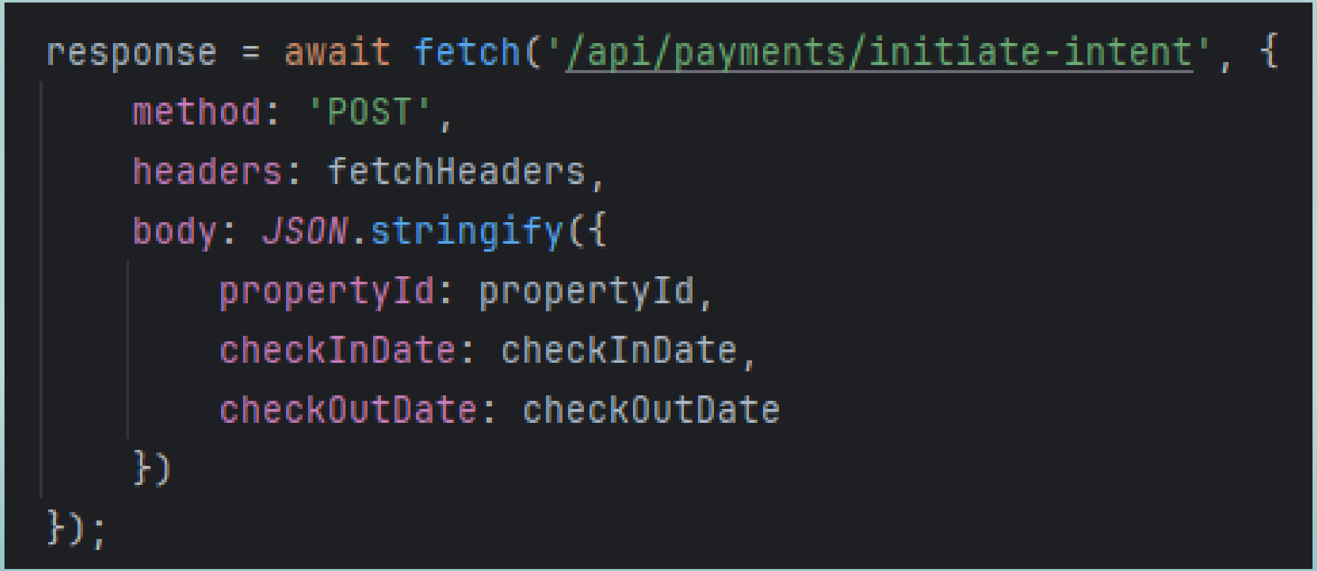
- AJAX (Asynchronous JavaScript and JSON)
- Communicate with the server in the background
- Prevents full page reloads
- Usage example: loading the payment form



```
bookButton.addEventListener('click', async (event) => {
    event.preventDefault();

    if (bookButton.disabled || bookButton.classList.contains('disabled')) return;

    setLoading(true);
    bookingMessage.classList.add('d-none');

    const propertyId = widget.getAttribute('data-id');
    const checkInDate = widget.getAttribute('data-checkin');
    const checkOutDate = widget.getAttribute('data-checkout');
```

Teaching Point 1: AJAX - Continue

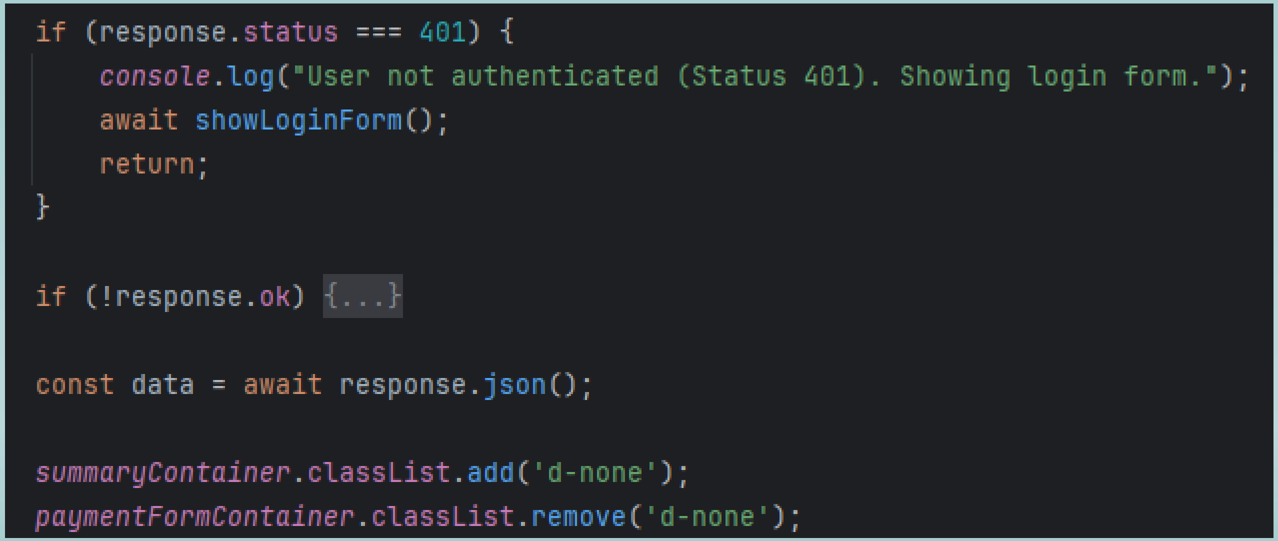
- JavaScript fetch API to send data
- Request is asynchronous with a loading spinner
- Must include CSRF token



```
response = await fetch('/api/payments/initiate-intent', {
    method: 'POST',
    headers: fetchHeaders,
    body: JSON.stringify({
        propertyId: propertyId,
        checkInDate: checkInDate,
        checkOutDate: checkOutDate
    })
});
```

Teaching Point 1: AJAX - Continue

- We analyze the server's HTTP response status
- Success (200): Display payment form
- Failure (401): Dynamically load the login form using showLoginForm()
- Goal: Seamlessly switch the widget state based on security status



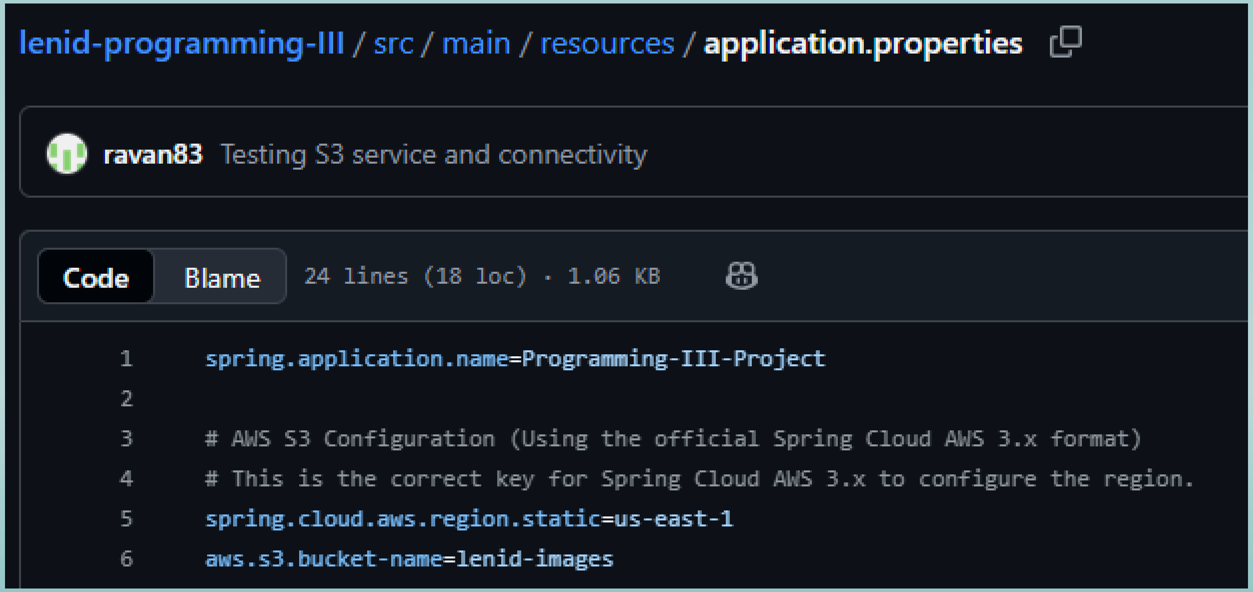
```
if (response.status === 401) {
    console.log("User not authenticated (Status 401). Showing login form.");
    await showLoginForm();
    return;
}

if (!response.ok) {...}

const data = await response.json();

summaryContainer.classList.add('d-none');
paymentFormContainer.classList.remove('d-none');
```

Teaching Point 2: AWS S3 Testing - 1

Step 1 : Configuration

(application.properties)

- Externalizes S3 connection details.
- Clean, flexible, and environment-agnostic code.



lenid-programming-III / src / main / resources / application.properties

ravan83 Testing S3 service and connectivity

```
spring.application.name=Programming-III-Project

# AWS S3 Configuration (Using the official Spring Cloud AWS 3.x format)
# This is the correct key for Spring Cloud AWS 3.x to configure the region.
spring.cloud.aws.region.static=us-east-1
aws.s3.bucket-name=lenid-images
```

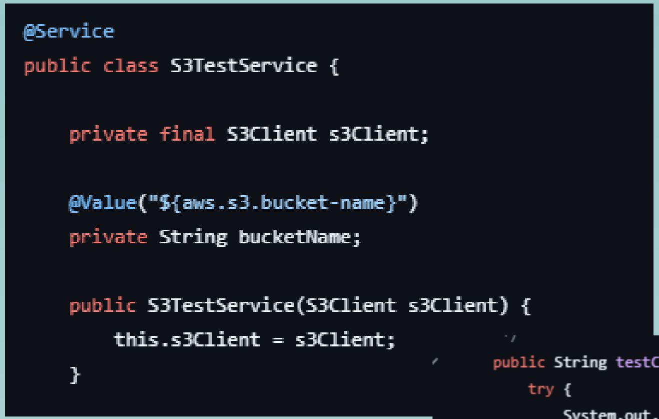
Teaching Point 2: AWS S3 Testing - 2

Step 2: The Service

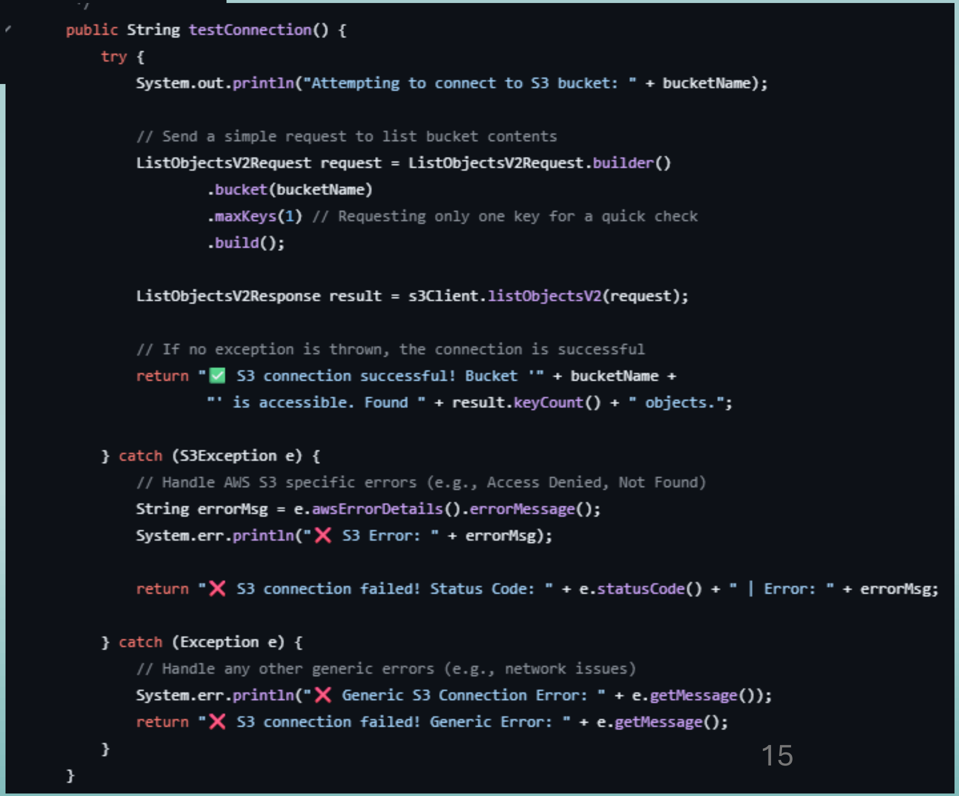
- Spring @Service to contain the logic.
- Auto-injected S3Client from Spring Cloud AWS.
- @Value annotation to read properties.

Step 3: The Logic& Error Handling testConnection() Method

- Action: A simple listObjects API call.
- Success: try block completes.
- Failure: catch blocks identify the error source.
  S3Exception: AWS permission/ config error.
  Exception: Network or other generic error



```java
@Service
public class S3TestService {

    private final S3Client s3Client;

    @Value("${aws.s3.bucket-name}")
    private String bucketName;

    public S3TestService(S3Client s3Client) {
        this.s3Client = s3Client;
    }
```



```java
    public String testConnection() {
        try {
            System.out.println("Attempting to connect to S3 bucket: " + bucketName);

            // Send a simple request to list bucket contents
            ListObjectsV2Request request = ListObjectsV2Request.builder()
                    .bucket(bucketName)
                    .maxKeys(1) // Requesting only one key for a quick check
                    .build();

            ListObjectsV2Response result = s3Client.listObjectsV2(request);

            // If no exception is thrown, the connection is successful
            return "✅ S3 connection successful! Bucket '" + bucketName +
                    "' is accessible. Found " + result.keyCount() + " objects.";

        } catch (S3Exception e) {
            // Handle AWS S3 specific errors (e.g., Access Denied, Not Found)
            String errorMsg = e.awsErrorDetails().errorMessage();
            System.err.println("❌ S3 Error: " + errorMsg);

            return "❌ S3 connection failed! Status Code: " + e.statusCode() + " | Error: " + errorMsg;

        } catch (Exception e) {
            // Handle any other generic errors (e.g., network issues)
            System.err.println("❌ Generic S3 Connection Error: " + e.getMessage());
            return "❌ S3 connection failed! Generic Error: " + e.getMessage();
        }
    }
```

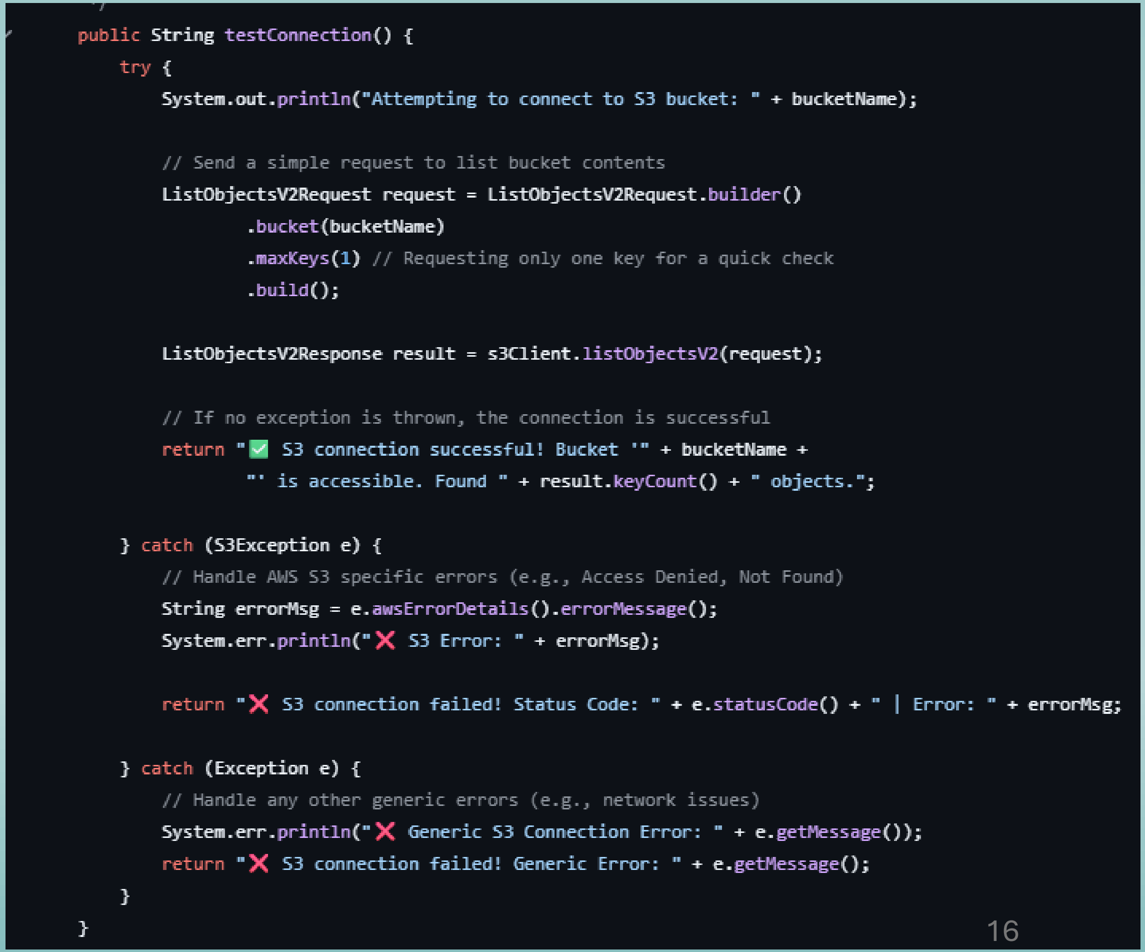
Teaching Point 2: AWS S3 Testing - 3

Step 4: The Logic& Error Handling

testConnection() Method

- Action: A simple listObjects API call.
- Success: try block completes.
- Failure: catch blocks identify the error source.

- S3Exception: AWS permission/config error.
- Exception: Network or other generic error.



```
public String testConnection() {
    try {
        System.out.println("Attempting to connect to S3 bucket: " + bucketName);

        // Send a simple request to list bucket contents
        ListObjectsV2Request request = ListObjectsV2Request.builder()
                .bucket(bucketName)
                .maxKeys(1) // Requesting only one key for a quick check
                .build();

        ListObjectsV2Response result = s3Client.listObjectsV2(request);

        // If no exception is thrown, the connection is successful
        return "✅ S3 connection successful! Bucket '" + bucketName +
                "' is accessible. Found " + result.keyCount() + " objects.";

    } catch (S3Exception e) {
        // Handle AWS S3 specific errors (e.g., Access Denied, Not Found)
        String errorMsg = e.awsErrorDetails().errorMessage();
        System.err.println("❌ S3 Error: " + errorMsg);

        return "❌ S3 connection failed! Status Code: " + e.statusCode() + " | Error: " + errorMsg;

    } catch (Exception e) {
        // Handle any other generic errors (e.g., network issues)
        System.err.println("❌ Generic S3 Connection Error: " + e.getMessage());
        return "❌ S3 connection failed! Generic Error: " + e.getMessage();
    }
}
```

Teaching Point 2: AWS S3 Testing - 4

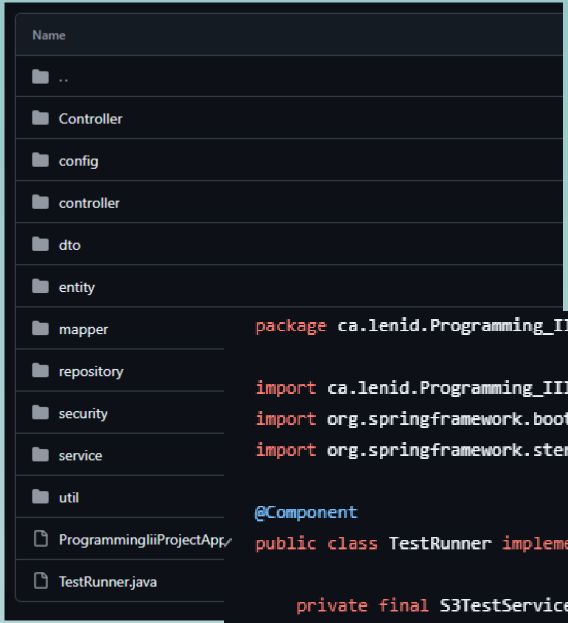
Step 5: Automating the Test on Startup

TestRunner.java

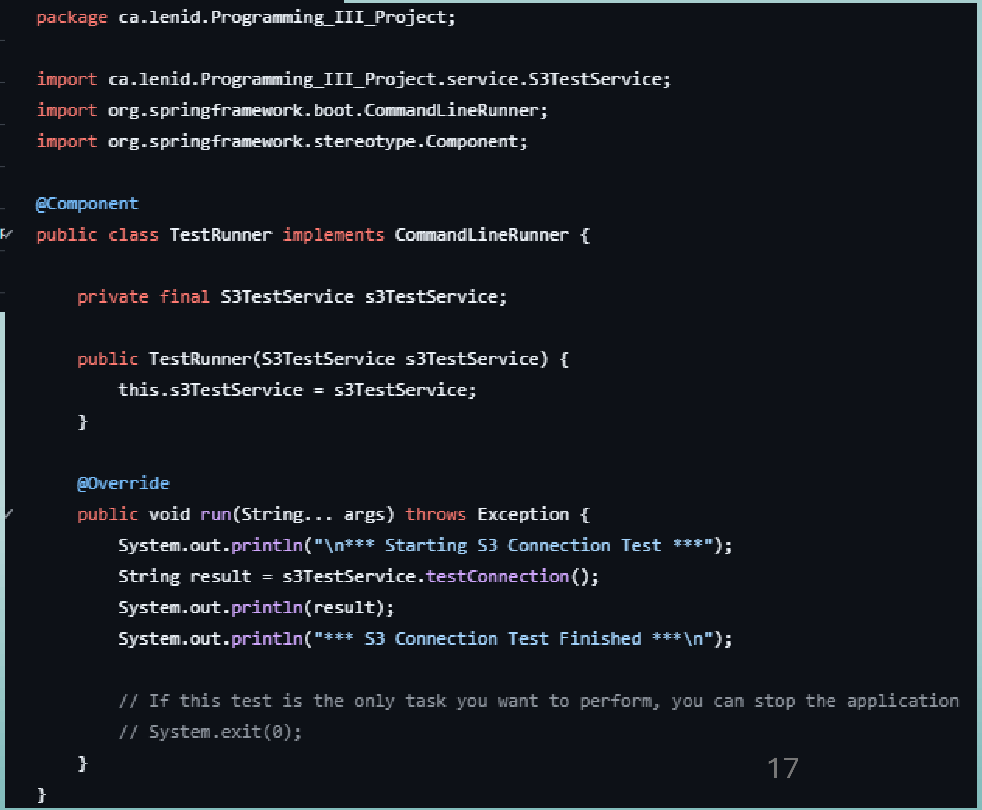
- Concept: Uses CommandLineRunner interface

What it does:

- A Spring Boot feature to run code immediately after the application starts.
- It automatically calls our S3TestService.
- Prints the connection status to the console on every startup.



```
Name
..
Controller
config
controller
dto
entity
mapper
repository
security
service
util
ProgrammingIiiProjectApp
TestRunner.java
```



```java
package ca.lenid.Programming_III_Project;

import ca.lenid.Programming_III_Project.service.S3TestService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class TestRunner implements CommandLineRunner {

    private final S3TestService s3TestService;

    public TestRunner(S3TestService s3TestService) {
        this.s3TestService = s3TestService;
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println("\n*** Starting S3 Connection Test ***");
        String result = s3TestService.testConnection();
        System.out.println(result);
        System.out.println("*** S3 Connection Test Finished ***\n");

        // If this test is the only task you want to perform, you can stop the application
        // System.exit(0);
    }
}
```

Future Work

- Implement a dedicated Host Dashboard for booking management and approval.
- Introduce an Admin Role for centralized user account and property management
- Host can promote a property

Summary

- Functional P2P rental platform.
- Secure role management & real-time date validation.
- Seamless Stripe payment integration using AJAX.
- Deployed application on Heroku & AWS RDS/S3.
- Result: A complete, secure, and scalable solution.

Ali	Sarah
AWS S3 bucket & RDS	AWS S3 bucket & RDS
FE foundation (bootstrap, JS, fragments)	Database (entities)
Login & Register frontend	Login & Register backend with validation
Security Config	Search-result backend
Index frontend & backend	Property-detail frontend & backend
Search-result frontend	Payment backend
Dashboard frontend & backend	Move date pickers to index
S3 image uploading	Deployed on Heroku
Edit property frontend & backend	

Le Nid

MONTRÉAL • SEJOURS ACOGLANTS

THANK YOU!