SuiteSpot

Course Team Project, Team 5

Company Name

Meet the team

Nathan Sorensen

Riddhi Bhowmik

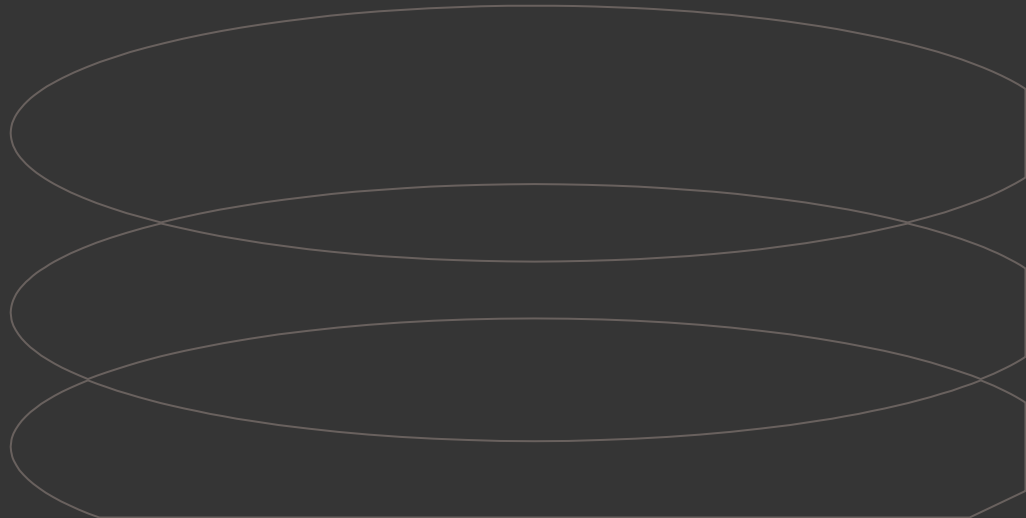
Koki Shin

Sergio Calvillo

Aeden Chandy

Eshwar Rajaputana

Roshan Katta



Project Objective

Our objective is to create a modern alternative to traditional vacation home rental websites like Airbnb and Vrbo. We will innovate beyond current norms by streamlining the reservation process, developing a responsive and appealing user environment, enabling precise searches with instant results, and providing an overall seamless booking experience.

Project Schedule

Justification: A three-month timeline allows for complete SDLC execution within the semester constraints.

Jan 20, 2025

Start Date
Requirements & Architecture

Feb - Mar

Development
Coding & Integration Phase

April 20, 2025

End Date
Testing & Final Deployment

Policy

4 Hours/Day
Weekdays Only (No Weekends)

Function Point Breakdown

Component	Type	Complexity	FP
User Registration/Login	External Input	Low	3
Create Rental Listing	External Input	Medium	4
Search/Filter Listings	External Input	Medium	4
Booking Property	External Input	Medium	4
Payment Processing	External Output	High	7
Messaging (Host/Guest)	External Interface	High	7
Admin Dashboard	External Output	Medium	5
Databases (Listings/Users)	Internal Logical File	High/Med	17
Total Adjusted FP	**(VAF 1.1)**	**—**	**72**

Cost & Effort Summary

Total Function Points 72 FP

Estimated Effort (10hr/FP) 720 Hours

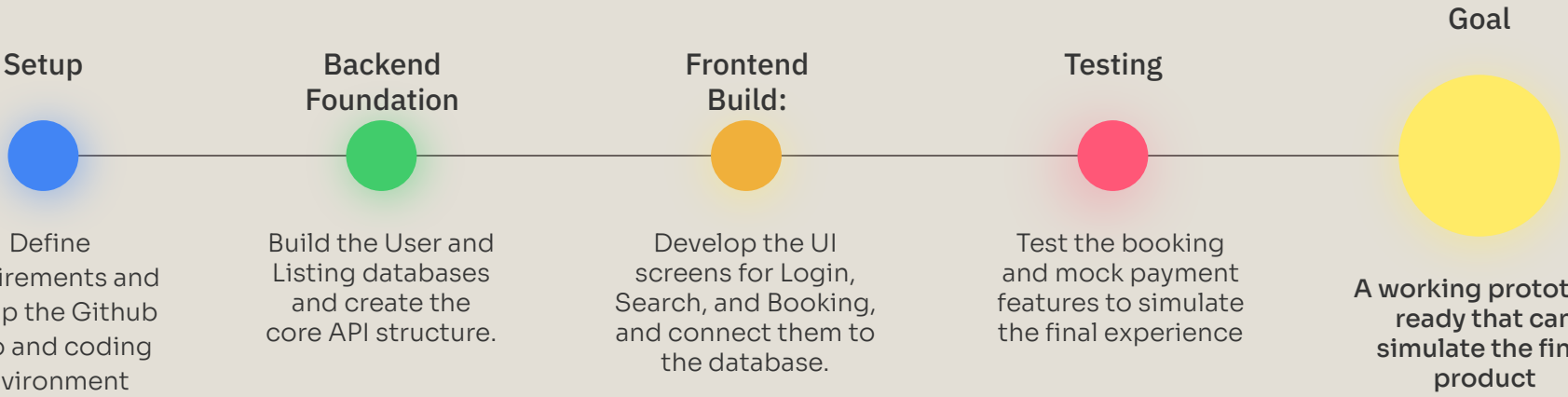
Labor Cost ($35/hr) $25,200

Hardware/Cloud $500

Software/Training $420

Estimated Final Price (with Overhead) = $28,980

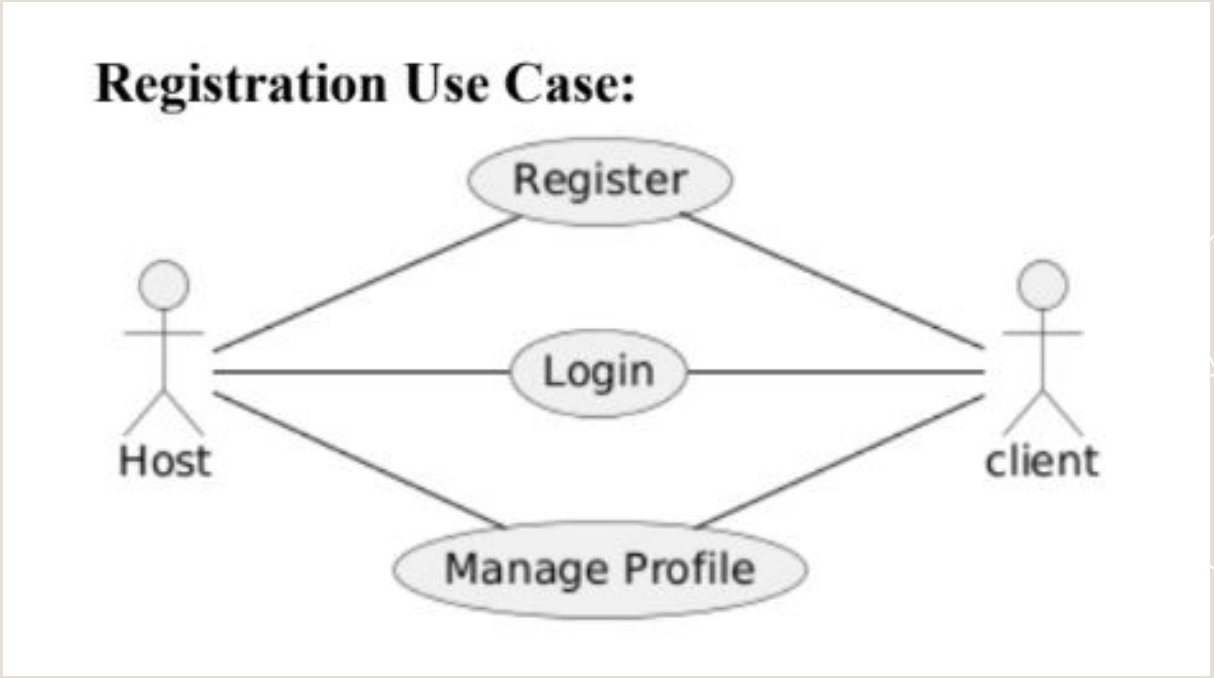
Project Timeline



Functional & Non-Functional Requirements

Functional Requirements	Non-Functional Requirements
Authentication: Secure Sign-up & Login. Search Engine: Filter by Location, Dates, & Guests. Booking System: Secure Payments & Email Confirmations. Listing Management: Hosts can Add, Edit, or Delete properties. Host Dashboard: View upcoming reservations. User Profile: Manage Wishlists & Saved Homes.	Technical & Quality Goals • Usability: Responsive design (Desktop & Mobile). • Performance: Search results load in < 2 seconds. • Reliability: Target 99% uptime during demo. • Portability: Compatible with modern browsers (Chrome/Edge). Tech Stack & Standards • Stack: MERN (MongoDB, Express, React, Node.js). • Workflow: GitHub for version control; REST API standards. • Code Quality: ESLint for code style and linting. Compliance & External • Security: Basic data privacy, mock integration for payments.

Use Case Diagram

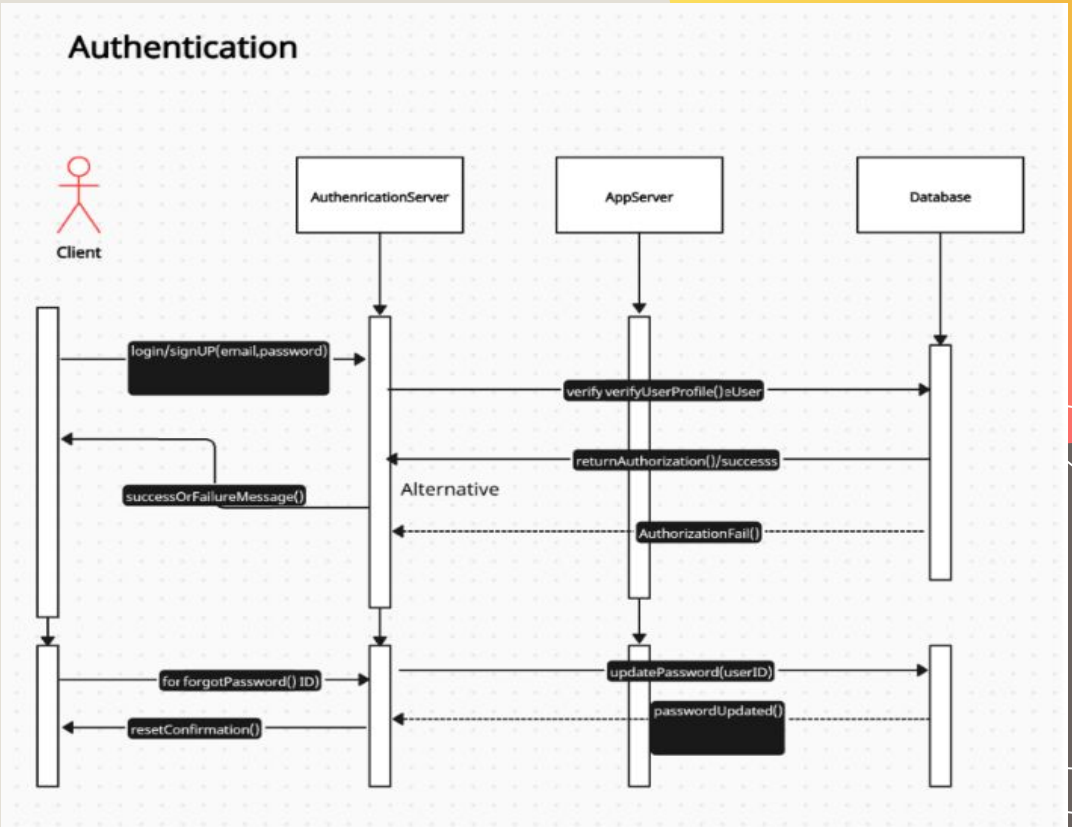


Sequence Diagram

The Client sends the email and password to the Authentication Server. The Server doesn't store passwords locally; it queries the Database to verify the profile.

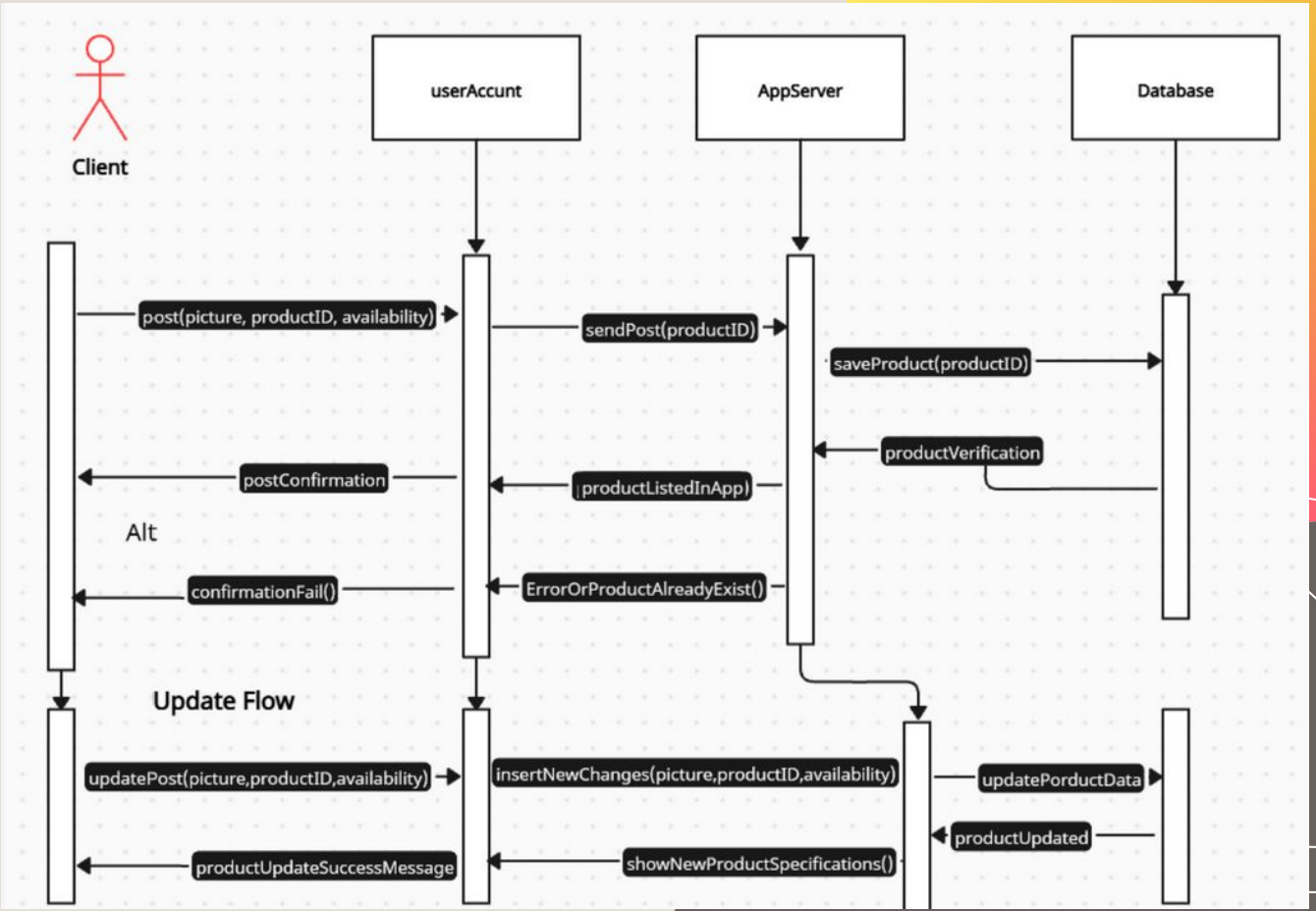
If the database returns a match, we send a success token. If not (the dashed arrow you see here), we return a failure message to the UI so the user knows to retry.

We also accounted for password resets. The client sends a request, and the server updates the record in the database.
We separated the App Server from the Auth Server to ensure our application is modular and secure



Listing Management sequence diagram

The Client submits the picture and availability.
Our AppServer acts as a gatekeeper—it doesn't just save blindly. It first validates the data.
If the listing already exists or the data is corrupt, the Database rejects it, and we bubble that error up to the user immediately.
If valid, the database generates a unique ID, and we confirm the listing is live.



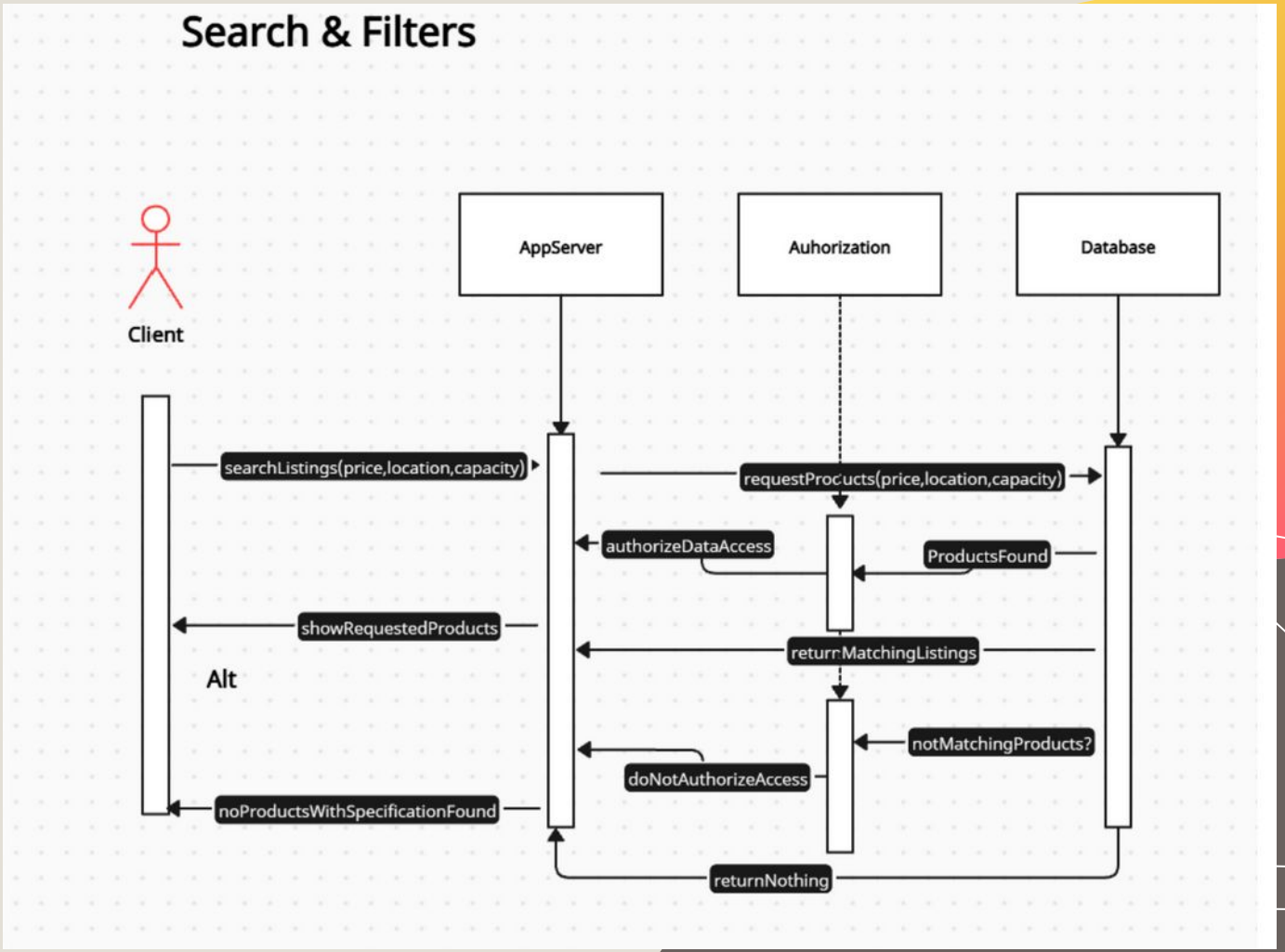
Search & Filters

The Client sends the filter parameters price, location, and capacity to the App Server.

The App Server constructs a query and sends it to the Database.

If matches found the database returns a list of objects and the app server processes these and returns them to the user.

If no matches it returns an empty list and the app server then triggers a specific UI response prompting the user to adjust their filters.

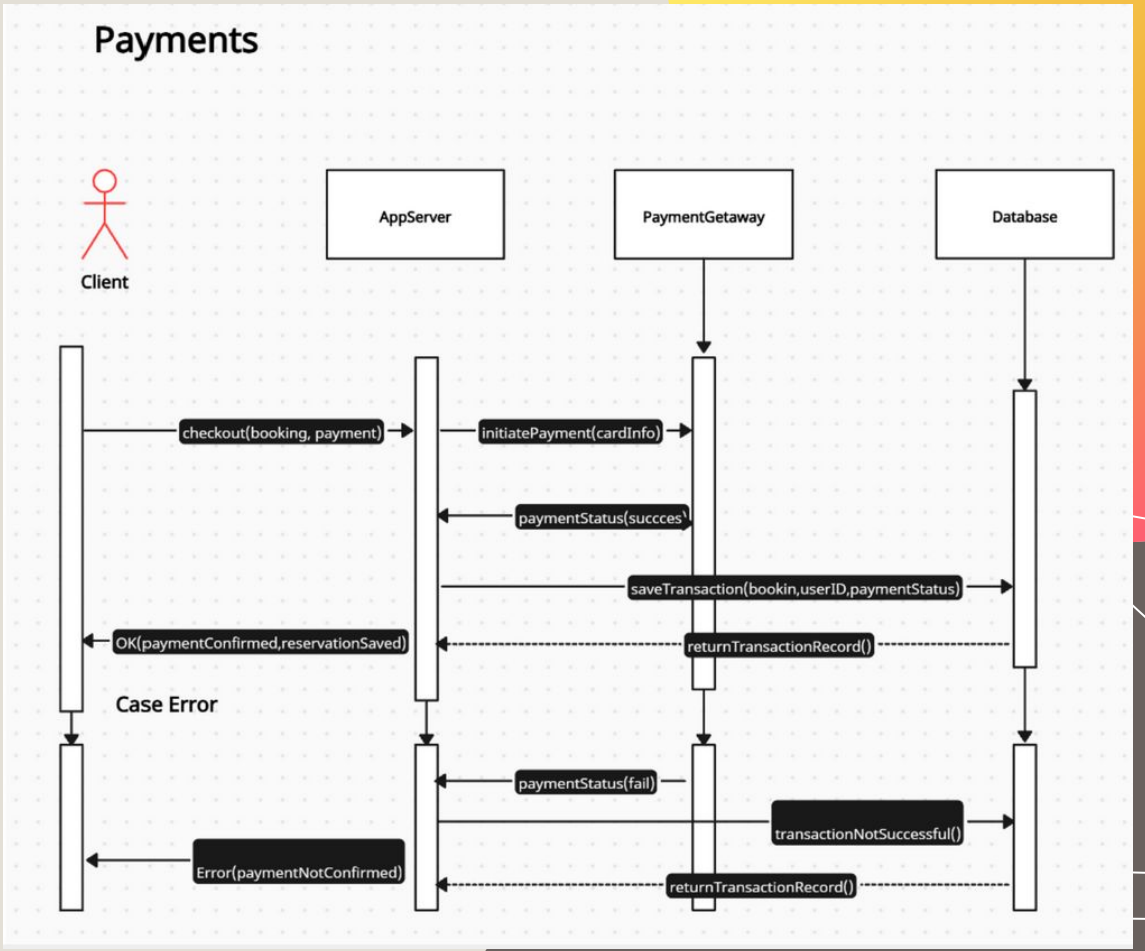


The Client sends the booking details and payment credentials.

Our app server immediately forwards this data to the external payment gateway.We do not process the money ourselves.

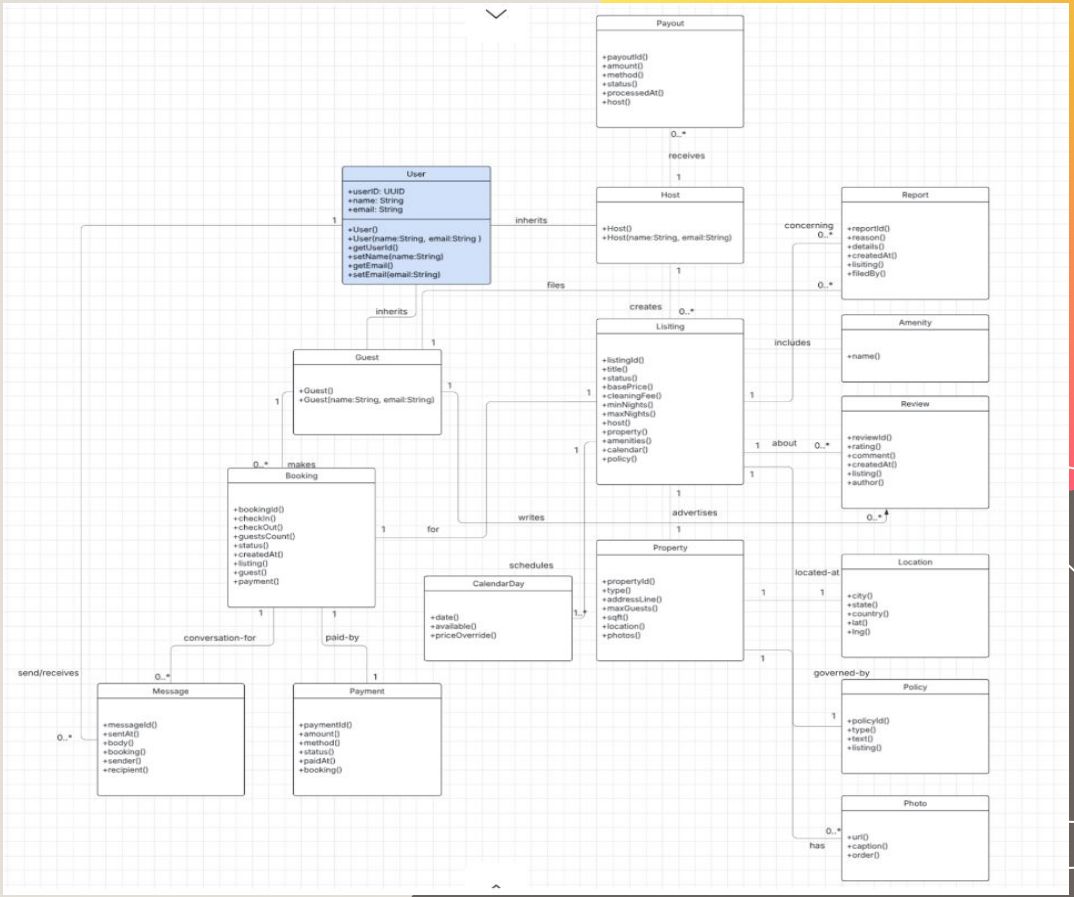
If the Gateway returns 'Success', we commit the transaction to our Database and confirm the booking to the user.

If the bank declined the card, we still talk to the Database. We log the failed attempt to create an audit trail, ensuring we can debug why transactions are failing.



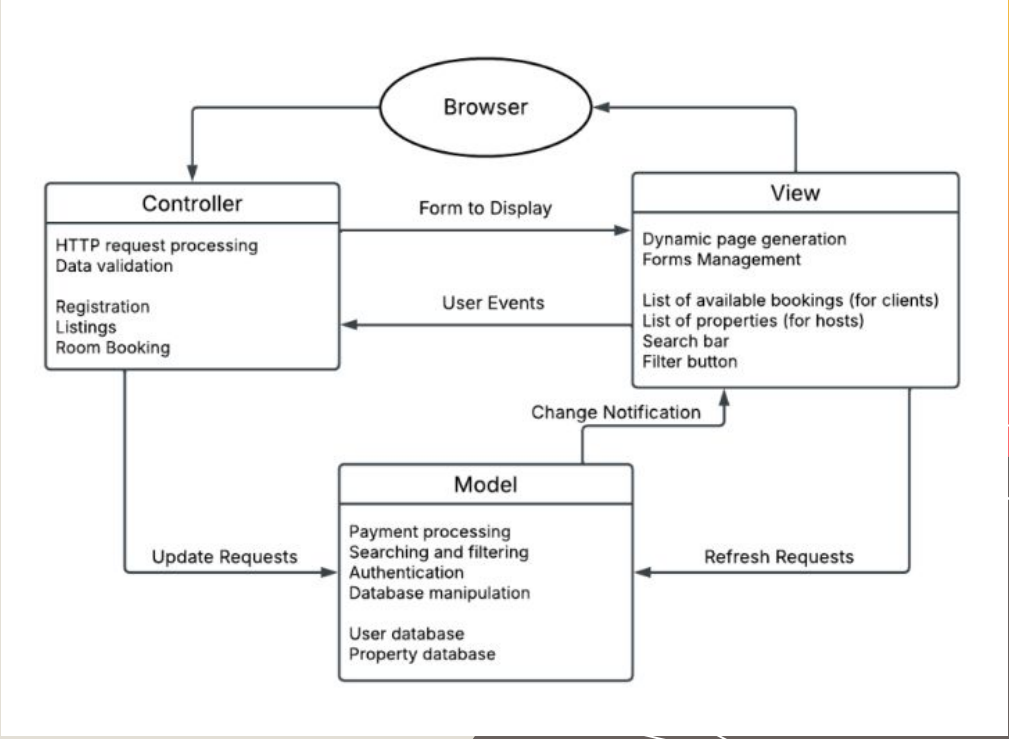
Class Diagram

This UML diagram outlines our online home rental marketplace application's key classes and their relationships. At its core, the diagram models 3 user types: User that has the specialized classes for Host & Guest. The Host class is able to create Listings that advertise Properties at a specific Location. Additionally, each Listing may have multiple Photos, Amenities, and CalendarDays, and is governed by a Policy. Guests are able to interact with Listings by making Bookings, which are associated with Payments and Messages. Bookings may also trigger Reviews that are written by the Guests and Reports filed against Listings. Hosts will receive Payouts based on their Listings' activities. The model captures the structuring of the marketplace, from user interaction to property management and transaction flows.



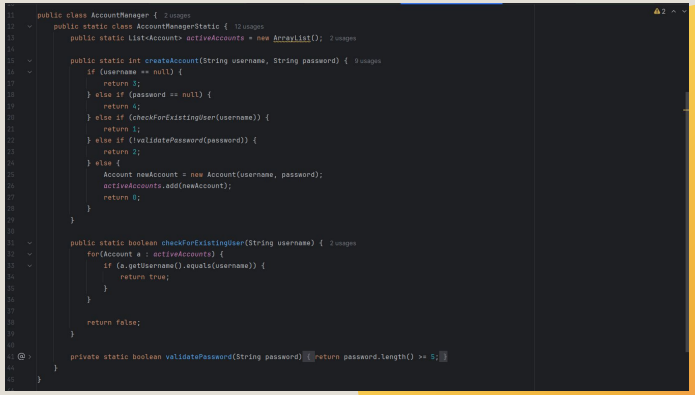
Architectural Design

We're choosing to use the MVC architectural pattern. It allows us to provide different views of the data to hosts and clients in order to fit each of their needs. It also gives us the flexibility to adapt to future requirements that we aren't currently aware of. Since our system is a web-based service application, the architectural model builds off of the web-based application model.

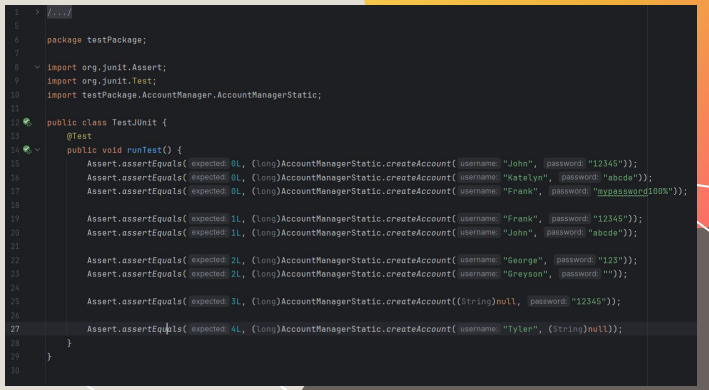


Unit Testing

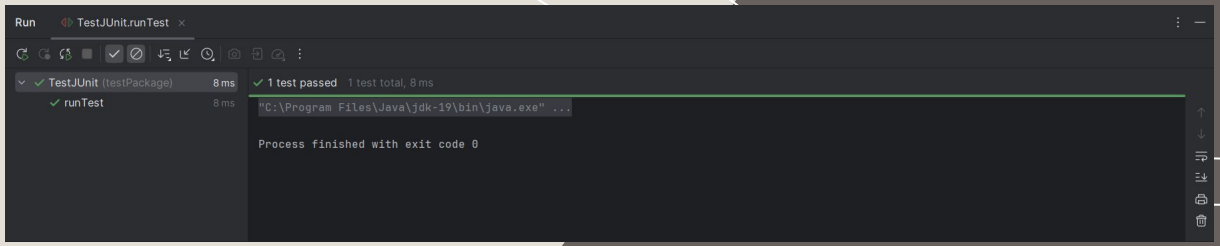
As shown here, we did some simple unit testing, where we attempted to make new accounts with valid credentials, then try to create a duplicate user, then try to set an invalid password, and finally attempt to create an account that is missing either a username or a password. As seen, the test passes, since the error codes returned by each output align with how we defined them in our AccountManager file.



```
public class AccountManager {
    public static class AccountManagerStatic {
        public static List<Account> activeAccounts = new ArrayList();

        public static int createAccount(String username, String password) {
            if (username == null) {
                return 3;
            } else if (password == null) {
                return 4;
            } else if (checkForExistingUser(username)) {
                return 1;
            } else if (!validatePassword(password)) {
                return 2;
            } else {
                Account newAccount = new Account(username, password);
                activeAccounts.add(newAccount);
                return 0;
            }
        }

        public static boolean checkForExistingUser(String username) {
            for(Account a : activeAccounts) {
                if (a.getUsername().equals(username)) {
                    return true;
                }
            }

            return false;
        }

        private static boolean validatePassword(String password) { return password.length() >= 5; }
    }
}
```



```
/.../

package testPackage;

import org.junit.Assert;
import org.junit.Test;
import testPackage.AccountManager.AccountManagerStatic;

public class TestJUnit {
    @Test
    public void runTest() {
        Assert.assertEquals( expected: 0L, (long)AccountManagerStatic.createAccount( username: "John", password: "12345"));
        Assert.assertEquals( expected: 0L, (long)AccountManagerStatic.createAccount( username: "Katelyn", password: "abcde"));
        Assert.assertEquals( expected: 0L, (long)AccountManagerStatic.createAccount( username: "Frank", password: "mypassword100%"));

        Assert.assertEquals( expected: 1L, (long)AccountManagerStatic.createAccount( username: "Frank", password: "12345"));
        Assert.assertEquals( expected: 1L, (long)AccountManagerStatic.createAccount( username: "John", password: "abcde"));

        Assert.assertEquals( expected: 2L, (long)AccountManagerStatic.createAccount( username: "George", password: "123"));
        Assert.assertEquals( expected: 2L, (long)AccountManagerStatic.createAccount( username: "Greyson", password: ""));

        Assert.assertEquals( expected: 3L, (long)AccountManagerStatic.createAccount((String)null, password: "12345"));

        Assert.assertEquals( expected: 4L, (long)AccountManagerStatic.createAccount( username: "Tyler", (String)null));
    }
}
```



```
Run    TestJUnit.runTest

TestJUnit (testPackage)  8 ms
    runTest              8 ms

1 test passed  1 test total, 8 ms

"C:\Program Files\Java\jdk-19\bin\java.exe" ...

Process finished with exit code 0
```

Comparison with Similar Designs

Feature	Vrbo	Airbnb	SuiteSpot
Accommodation Type	**Entire homes only.** Focused on traditional vacation spots (family/groups).	**Diverse**. Rooms, shared spaces, entire homes, unique stays	**Curated Entire Homes:** Vetted, high-quality properties with consistent, professional service.
Search/Filtering	Good, dedicated filters. Filters are primary.	Uses filters and **Categories** for discovery.	**Semantic Search (AI-Driven):** Search by experience (e.g., "quiet cabin with hot tub"). Instant, precise results.
Pricing Display	Guest service fees **added at checkout.**	Guest service fees **added late in the process** (the "Surprise Fee").	**All-Inclusive Upfront Pricing:** Total, final price displayed in search results for complete transparency.
Booking Process	Often requires a **booking request** (host approval). Less instant.	**Mostly Instant Book**, but still relies on host interaction/messaging.	**Seamless & Instant:** Automated, one-click booking with digital check-in integration.
Guest Communication	Less frequent, more formal (email-based messaging).	In-app messaging (host-to-guest). Support is often through the host.	**Digital Concierge:** Automated in-app support for common needs + 24/7 centralized human support.
UI/UX Approach	More traditional/functional; filter-heavy.	Visually appealing, dynamic, and mobile-friendly; discovery-focused.	Mobile-first, responsive design focusing on streamlined workflow and fewer clicks to confirmation.
Core Value	Privacy, space, and family-focused vacation stays.	Variety, budget options, and unique local experiences.	Seamless Experience, Guaranteed Quality, and Personalized Search.

Any questions?
Ask away!

Thank you