CS/SE/CE 3354 Software Engineering

Deliverable #2

SuiteSpot

URL of project repository: https://github.com/Eshwar1440/3354-team5

Note: For our java code, we used IntelliJ IDEA 2025.2.4 in order to compile, run, and test.

URL for slides:
https://docs.google.com/presentation/d/1NOTSajGyKqX3RDUjtiWFPpIVCL8mi1kWmnIjvjlGxkQ/edit?usp=sharing

1. Project Title

Our project title is SuiteSpot.

2. Delegated Tasks

Member Tasks:

- Riddhi Bhowmik: I will be expanding upon the search tool and adding more filtering options, such as preferred amenities, price range, and home types. I will also set it up so that once they set all their preferences, the app will show all such homes on one page that allows for infinite scrolling. Additionally, I will ensure that the frontend makes it easy for users to select these filters.

- Sergio Calvillo: I will be checking for any errors that might happen while implementing the payment actions and also look for features that can be included to make it reliable and consistent.

- Nathan Sorensen: I will create the means to and new rental listings as well as ways to book those rentals. I will also provide more functionality to lookup rentals based on the metadata.

- Eshwar Singh Rajaputana: I will deploy the application to a staging environment, manage environment variables and secrets, and ensure operation for testing and project demo.

- Roshan Katta: Adding onto the previous implementation, furthering the signup/login system and ensuring robust security. Begin work on reset password feature. Utilize basic password flow: email input, verification link, etc. Implement a password change page where users can reset their password using the token from the email.

- Koki Shin: I will perform QA testing on all implemented features to verify correctness and usability, document bugs, and improvement suggestions. I will also expand project documentation by adding usage guides and changelogs to ensure the system is easy to understand and maintain.
- Aedan Chandy: I will expand upon the listing database properties by adding specific things, such as titles, descriptions of the listing, and prices, among others. I will also handle the frontend and make sure that it can show and manage listings efficiently.

Group Tasks:

- Project scheduling, cost, effort, and pricing estimation, project duration, and staffing: Roshan Katta and Aeden Chandy
- Test Plan: Nathan Sorensen and Riddhi Bhowmik
- Comparison of work with similar designs: Koki Shin
- Conclusion: Riddhi Bhowmik
- References: Koki Shin
- Project Slides: Sergio Calvillo, Eshwar Rajaputana, Riddhi Bhowmik

3. Assumptions

We are making the following assumptions to simplify our implementation. Note that they may not be 100% accurate to real-world policies/rules.

1. All hosts that use SuiteSpot obtained the necessary permits or licensing required to list their homes on the rental marketplace
2. All users, including guests and hosts, need to pass an identity verification using a government-issued ID
3. Assume that all listings comply with basic safety requirements, such as smoke detectors, fire extinguishers, being cleaned properly, etc
4. SuiteSpot complies with global privacy laws
5. Hosts will be able to make their own cancellation policies, which would determine the latest date a guest could receive a full or partial refund on their booking, which will be shown to the guest when they view the property listing
6. Only verified users will be able to leave reviews, ensuring there is no spam or fraud 7.
Local taxes in certain regions will automatically be collected during payment, making sure SuiteSpot lines up with any local laws
8. All transactions will be processed securely through a third-party service, like Stripe, and payment

information will be encrypted, ensuring payment is secure and user data is protected

4. Addressing feedback

Here is the feedback we were given regarding our proposal:

> Well done. Elaborate on the proposed implementation.

In response to the feedback asking us to elaborate on our proposed implementation, we chose to outline our plan more thoroughly in a few key areas. This includes the core features we will implement the tech stack we plan to use, the structure of the application, and the personalized features we are currently thinking of adding.

In terms of core features, we decided that the things that are absolutely necessary in our project would be secure user authentication, a way for visitors to easily view and navigate property listings, a thorough and advanced search tool, and straightforward and reliable booking and payment systems.

Next, for the tech stack, we chose to use the MERN stack, which uses MongoDB for the database, Express.js for the backend, React for the frontend, and Node.js also for the backend. Starting off with the frontend, we will be using React in order to build a dynamic, modern, and interactive user interface. React allows us to reuse components, which will make development easier, which is a huge plus. Additionally, React will make it possible for us to quickly update and manage the interface as users interact with our site, guaranteeing a seamless experience every time. Next, for the backend, we will be using Node.js and Express.js. Node.js is a fast, scalable environment that works great for handling a lot of volume, which is important because in actual booking scenarios, our platform could get a lot of requests at once. Express.js will work with this in order to provide routing, handle requests, and overall structure the backend. It also makes building APIs a lot easier, which is how the frontend and backend communicate. Alongside this, Express.js makes it easy to add other functionalities, like checking if a user is logged in and handling errors, making it a great tool to use in a system like this. Then, for the database, we will be using MongoDB, which is a NoSQL database. MongoDB is known for being very flexible and highly scalable, so it is a perfect tool for us to use in a database that is projected to grow constantly. It is also great for storing the type of data we need, which includes things like property listings, user information, and bookings, along with any other data requirements that may appear later. Next, for unit testing, we plan to use Jest, which is a tool that easily runs unit tests with code written in JavaScript or its codebases. Finally, in order to make sure our API endpoints work properly, we will use a tool such as Playwright, which allows us to put in code samples directly from our Node.js backend and test to make sure it is properly linked with the frontend.

Next, for structuring, our frontend will be run in a user’s web browser and it will show the

user interface and any relevant data to users, as well as handle user inputs and backend interaction with RESTful API calling. On the backend, our Node.js/Express.js application will be in charge of dealing with all business logic, processing bookings, and interacting with the database. It will also expose any API endpoints that the frontend can call in order to smoothly communicate with each other. Here, we will use the MVC Architecture pattern, with the Model being where our data structures are defined, the View being the React components in the frontend that the user can see and interact with, and the Controller, which handles business logic in the backend, process requests, interact with the database, and send requests back to the frontend.
Finally, for personalized changes, we plan to add a wishlist feature where users can save a property for later. This will notify users of price changes and dates that the property is already booked. Another feature we want to add is a public profile page for hosts, which will allow them to add a bio so that guests can get to know them before booking. We will also let hosts use the map to highlight some local spots that they would suggest to visitors, which will make their experiences more personal.

5. Software Process Model

For our project, we decided that we would use the Incremental Process Model, which combines linear and parallel process flows, allowing us to develop deliverable increments at the same time, but in parts. In this project, we each have certain responsibilities with the app, so we split up half of our work for the first deliverable and half for the second. For example, with the search feature, we will first make sure it works as a bare bones search tool with some enhancements, like guest count, location, and time frame for the first deliverable. Then, for the second deliverable, we will expand on the search tool by adding more filtering tools, like amenities, price range, and home types and all listings appear on one page with infinite scrolling.

We chose this model because it has some notable advantages. The main one is the fact that it allows for parallel processing, which means that we can have multiple members of our team working on different parts of the program simultaneously, which significantly speeds up our timeline and makes it possible for multiple features to be implemented and improved upon in each deliverable. Another reason is because incremental development allows us to accommodate changing requirements with not a lot of documentation needed to support these changes, which is very important because certain parts of our application are subject to change, like frontend design and specific personalized features. This also makes it easy to get feedback on what we have done, which is important for us because we can submit each increment to the professor and TAs and receive feedback, which we can use to improve upon our project.

6. Software Requirements

Functional Requirements:

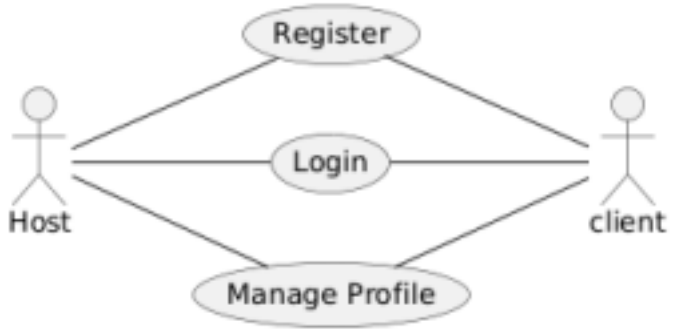
1. The system shall allow users to create an account and log in using secure authentication. 2. The system shall allow users to search for rental properties by location, dates, and number of guests.
3. The system shall allow users to book a selected property and process payments securely.
4. The system shall allow hosts to create, edit, and delete property listings. 5. The system shall allow users to view, edit, and manage their saved wishlist. 6. The system shall display booking confirmations and send email notifications. 7. The system shall provide hosts with the ability to view upcoming reservations.

Non-Functional Requirements:

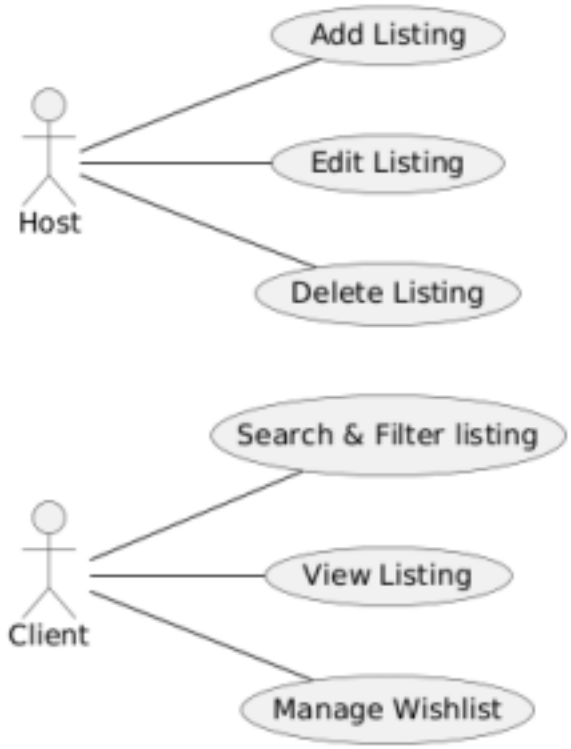
- Product Requirements:
  - Usability: The interface shall be intuitive and responsive across desktop and mobile devices.
  - Performance: Search results shall load within 2 seconds under normal network conditions.
  - Reliability: The system shall maintain 99% uptime during active deployment. ○ Portability: The web application shall run on all major browsers (Chrome, Edge, Safari, Firefox).
- Organizational Requirements:
  - Implementation: The project shall be developed using React, Node.js/Express, and MongoDB.
  - Standards: The development shall follow PEP 8 and ESLint conventions. API endpoint shall follow REST naming conventions and be documented.
  - Delivery: Source code and documentation shall be version-controlled in the GitHub repository.
- External Requirements:
  - Ethical: User data shall be used only for booking-related purposes.
  - Legislative: The system shall follow basic privacy and security expectations. ○ Interoperability: Payment services shall integrate securely with third-party APIs (Stripe or Paypal).

7. Use Case Diagrams

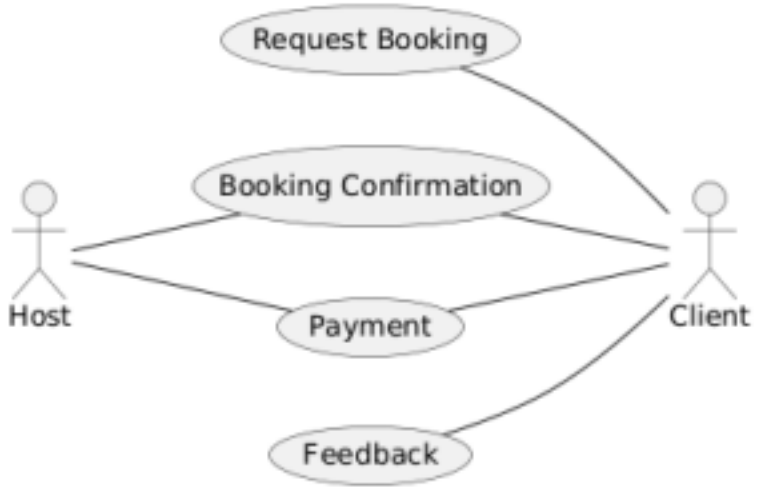
Registration Use Case:



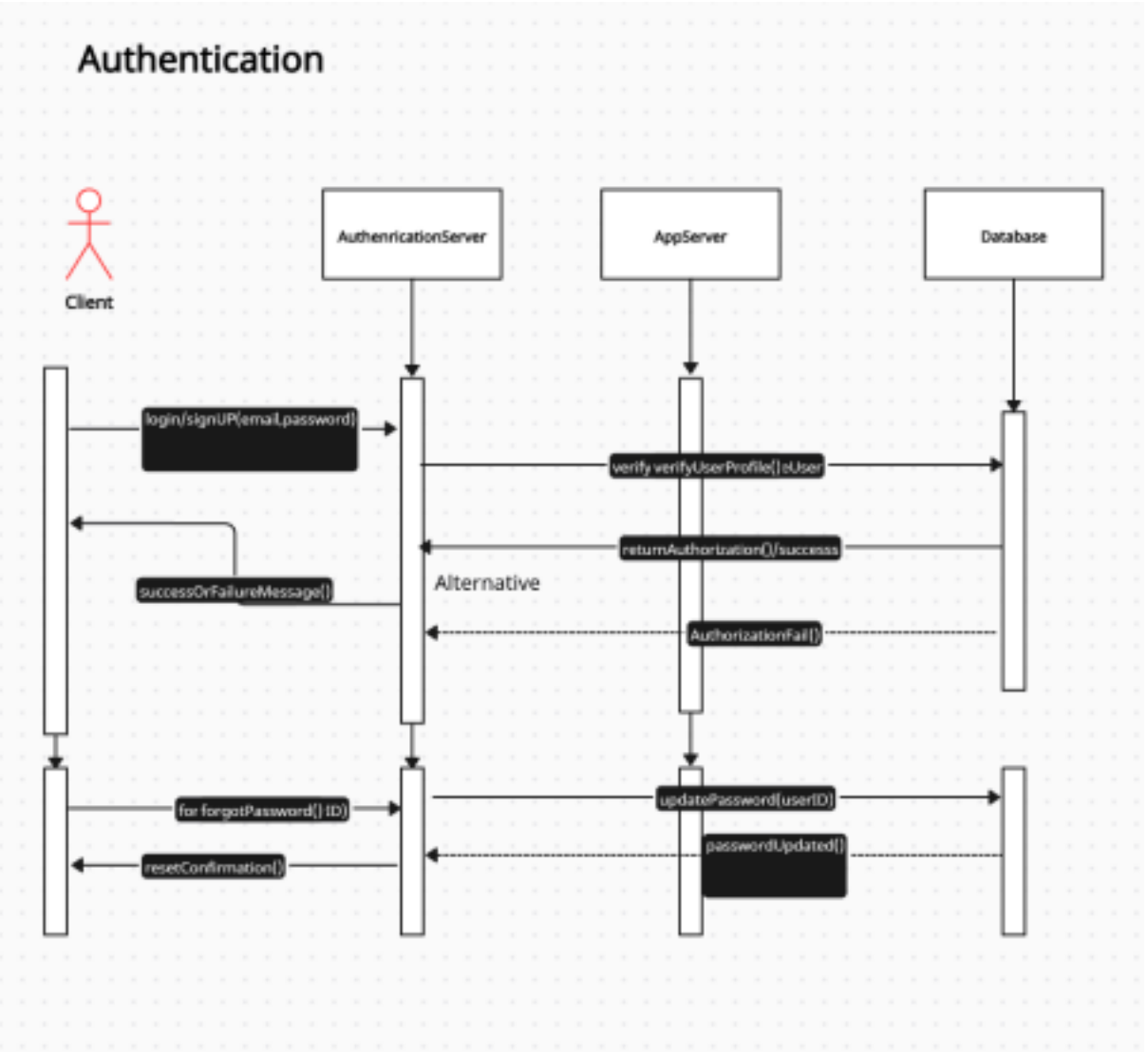
Listing Use Case:

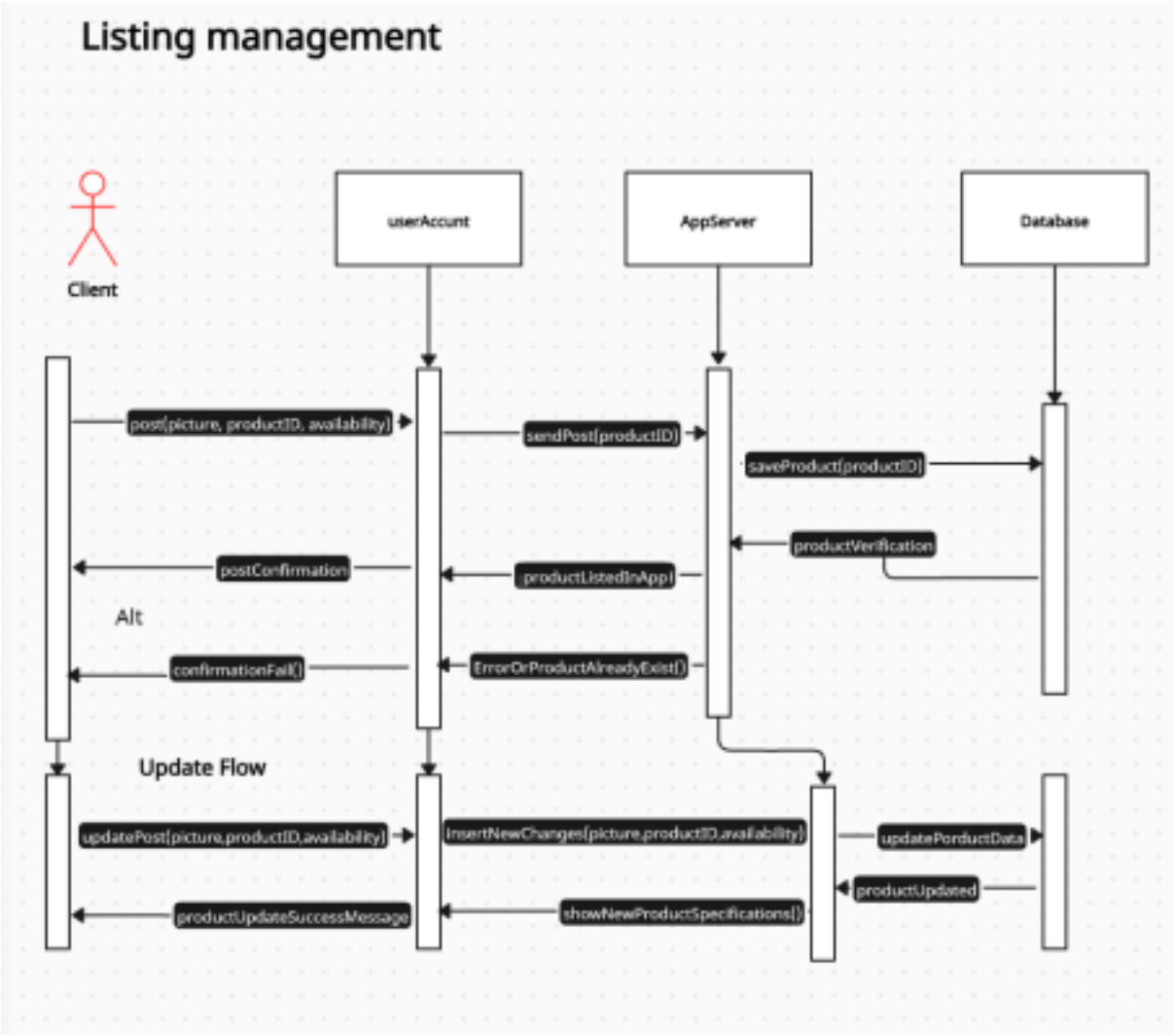


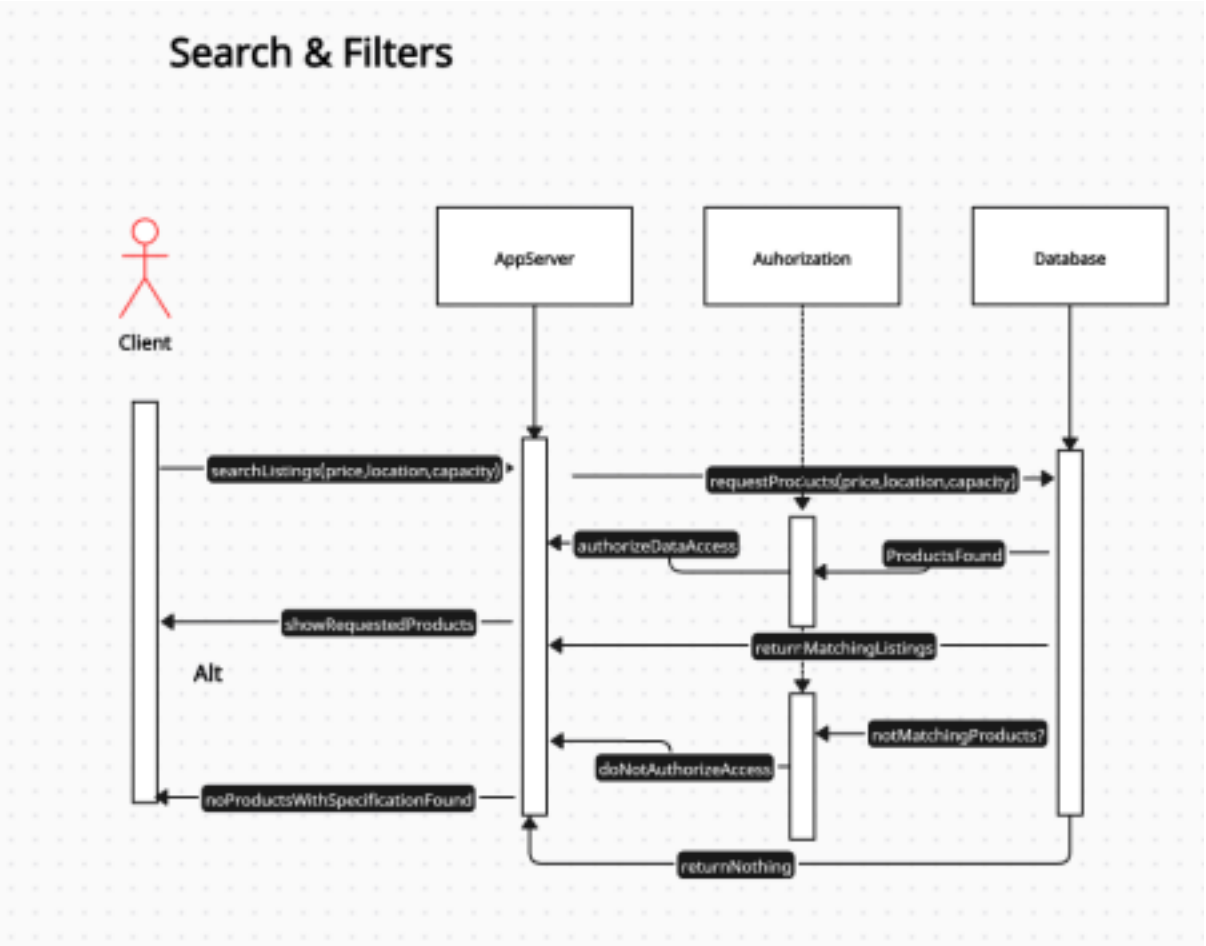
Room Booking Use Case:

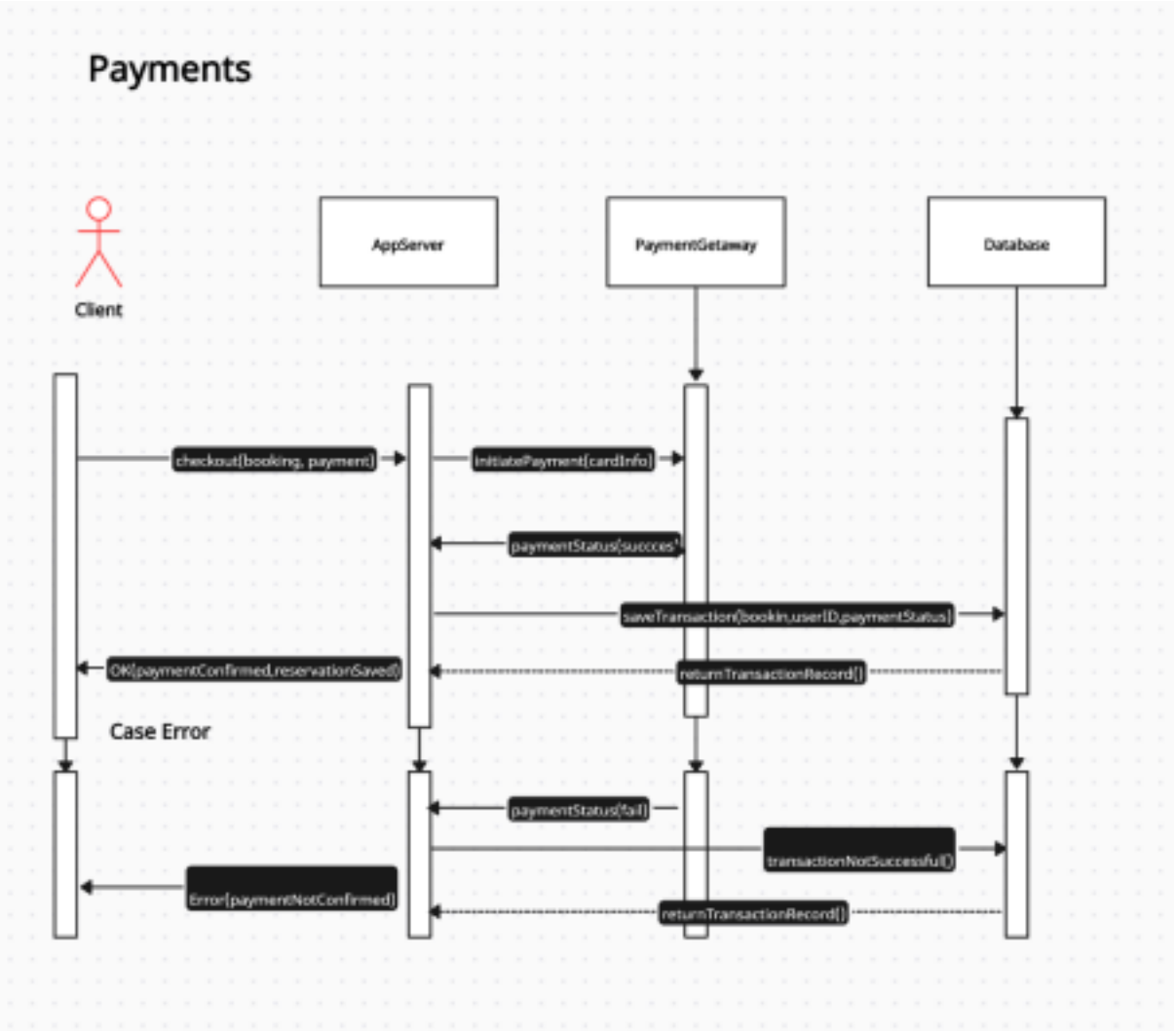


8. Sequence Diagrams


Authentication
Client
AuthenricationServer
AppServer
Database
login/signUP(email,password)
verify verifyUserProfile():User
returnAuthorization()/success
successOrFailureMessage()
Alternative
AuthorizationFail()
for forgotPassword() ID)
updatePassword(userID)
resetConfirmation()
passwordUpdated()

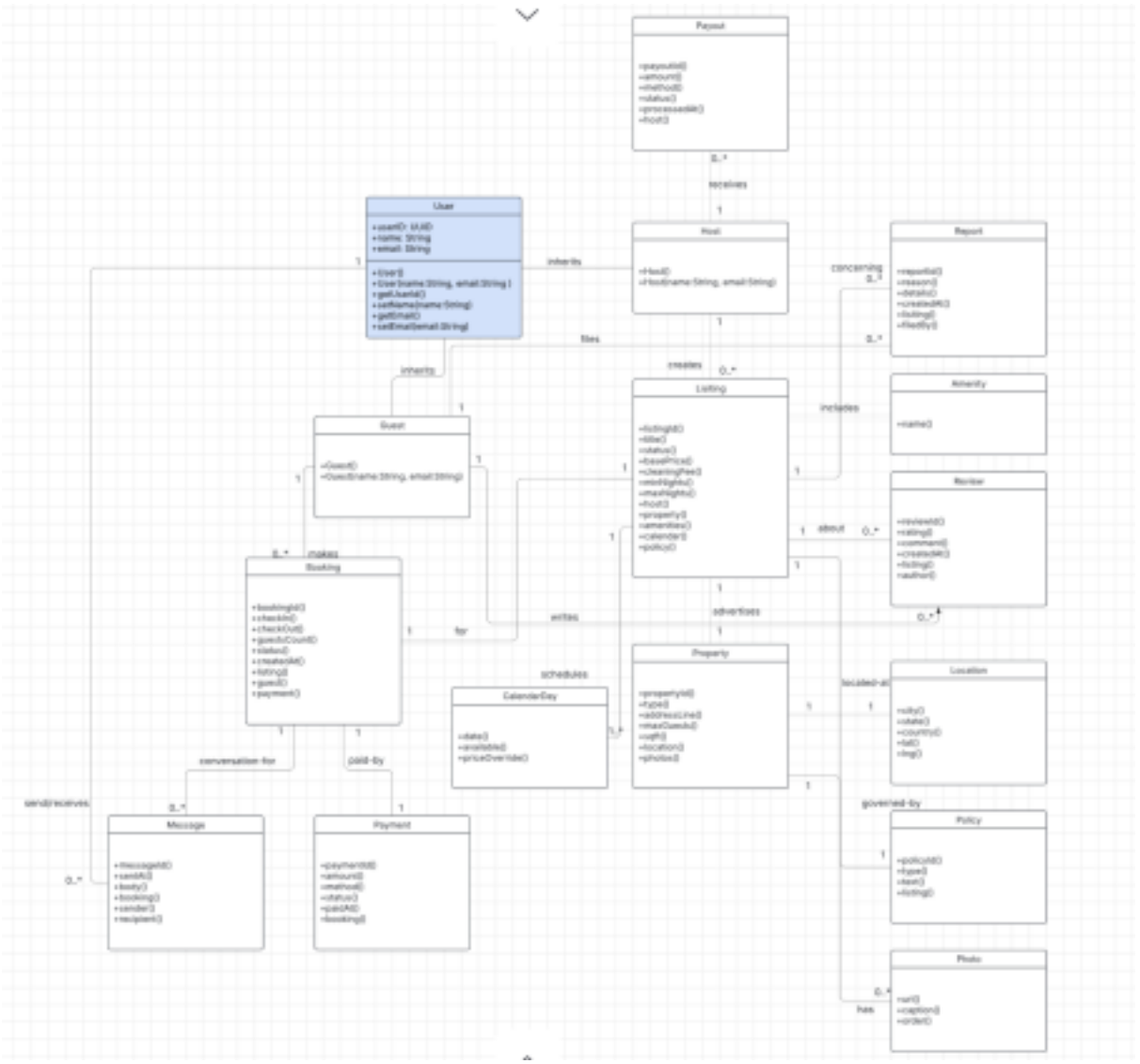

Listing management
Client
userAccount
AppServer
Database
post(picture, productID, availability)
sendPost(productID)
saveProduct(productID)
productVerification
postConfirmation
productListedInApp()
Alt
confirmationFail()
ErrorOrProductAlreadyExist()
Update Flow
updatePost(picture,productID,availability)
InsertNewChanges(picture,productID,availability)
updatePorductData
productUpdated
productUpdateSuccessMessage
showNewProductSpecifications()


Search & Filters
Client
AppServer
Authorization
Database
searchListings(price,location,capacity)
requestProducts(price,location,capacity)
authorizeDataAccess
ProductsFound
showRequestedProducts
returnMatchingListings
Alt
notMatchingProducts?
doNotAuthorizeAccess
noProductsWithSpecificationFound
returnNothing



9. Class Diagram

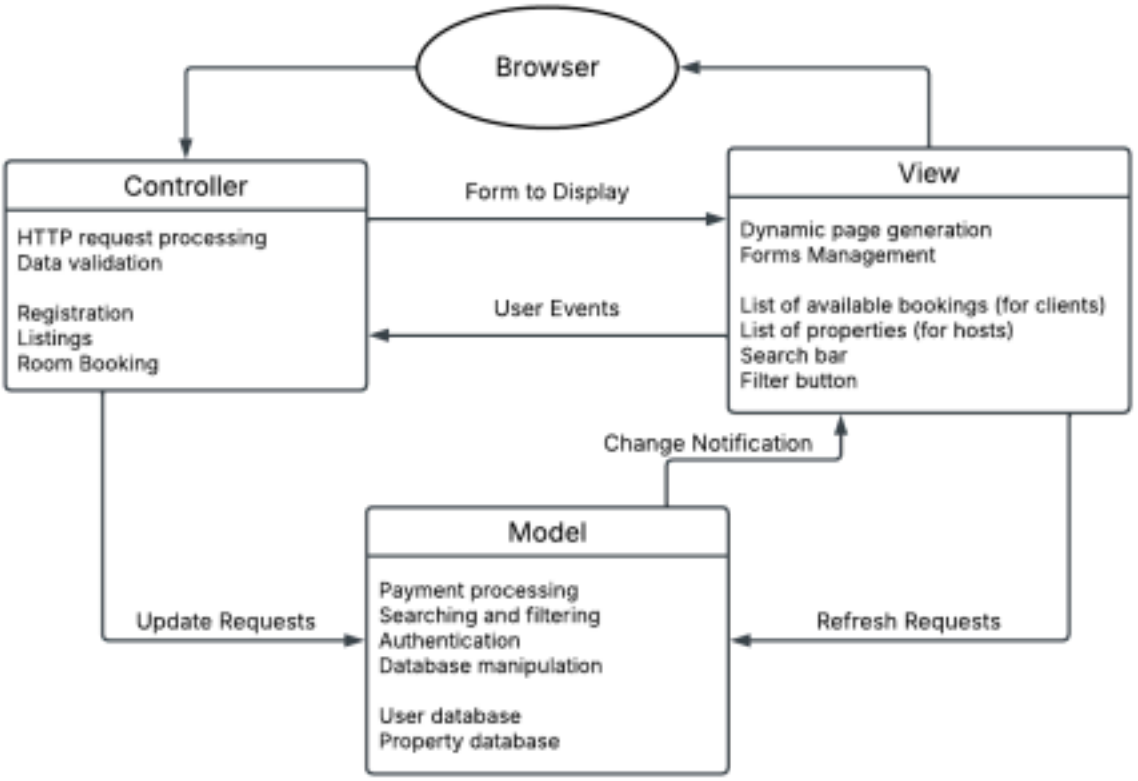
This UML diagram outlines our online home rental marketplace application's key classes and their relationships. At its core, the diagram models 3 user types: **User** that has the specialized classes for **Host** & **Guest**. The Host class is able to create **Listings** that advertise **Properties** 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.



10. 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 example given in Figure 6.6 the book:



11. Project Scheduling, Cost, Effort, and Pricing Estimation, Project Duration, and Staffing

Online Home-Rental Marketplace

a. Project Scheduling

Start date: January 20, 2025
End date: April 20, 2025

Justification: A three-month timeline allows the team to complete requirements gathering, architecture, development, testing, and deployment. This schedule also aligns well with a semester-based course project.

Weekend Policy: Weekends are not counted in the schedule.

Working Hours Per Day: 4 hours per team member per weekday.

b. Cost, Effort, and Pricing Estimation

Cost Estimation Method: The Function Point (FP) method is used for estimating system size, effort, and cost.

Function Point Breakdown:

Component	Type	Complexity	FP
User registration/login	External Input	Low	3
Create rental listing	External Input	Medium	4
Edit/delete listing	External Input	Medium	4
Search/filter listings	External Input	Medium	4
Booking a property	External Input	Medium	4
Payment processing	External Output	High	7
Messaging between host + guest	External Interface	High	7
Admin dashboard	External Output	Medium	5
Listings DB	Internal Logical File	High	10
Users DB	Internal Logical File	Medium	7

Reservations DB	Internal Logical File	High	10

Unadjusted FP Total: 65 FP

Adjusted FP (VAF 1.1): 72 FP

Effort Estimation: 72 FP × 10 hours/FP = 720 hours

Labor Cost: 720 hours × $35/hour = $25,200

Final Price with Overhead: Approximately $28,980

c. Estimated Hardware Costs

Cloud hosting, database instances, storage, and monitoring tools total approximately $400–$500.

d. Estimated Software Costs

Most development tools are free. Small costs for domain name and optional hosting tiers bring total to about $50–$70.

e. Estimated Personnel Costs

Total labor cost: Approximately $25,200.
Training costs: Approximately $350.
Total personnel cost: About $25,550.

12. Test Plan

a. For testing, we decided to focus on the unit of our application that deals with account creation, which is the public static int createAccount(String username, String password) located inside the AccountManagerStatic class, which is in AccountManager.java. We are going to be testing that this method does its job correctly, which is to either create a new user account when both the username and password are valid or to reject an account creation if a username has already been used or a password is invalid. This method is designed to return a code for each case of the testing. The first is the case where an account is successfully created and added to the database, which will return 0. Next, we check to see if a username is being duplicated, in which case we return 1. Then, we test to see if the password is of an acceptable length, which we defined as being 5 or more characters, and if not, we will return 2. Next we check to see if the username is missing entirely by comparing the inputted username to null and if so, we return 3. Finally, we test to see if the password string is null, which will return 4. In order to perform these tests, we will use JUnit, which is able to run automated tests. In the TestJunit file, we have a runTest method where we have a few tests set up on each of these 5 cases with assertEquals(expected, actual), which will automatically check to see if the values we give it are equal to the value we expect from it. If

all of the assertEqual checks pass, the test is successful. Overall, this plan guarantees that we check all of the outcomes of the createAccount method and prove that its logic is correct.

b. Screenshot of our testing output:

i. 

c.

i. Test case 1:
   1. Category: Success
   2. Input: (“John”, “12345”)
   3. Expected output: 0
   4. Meaning: Unique username, valid password of 5+ chars

ii. Test case 2:
   1. Category: Success
   2. Input: (“Katelyn”, “abcde”)
   3. Expected output: 0
   4. Meaning: Unique username, valid password of 5+ chars

iii. Test case 3:
   1. Category: Success
   2. Input: (“Frank”, “mypassword100%”)
   3. Expected output: 0
   4. Meaning: Unique username, valid password of 5+ chars

iv. Test case 4:
   1. Category: Duplicate user
   2. Input: (“Frank”, “12345”)
   3. Expected output: 1
   4. Meaning: The username “Frank” has already been used

v. Test case 5:
   1. Category: Duplicate user
   2. Input: (“John”, “abcde”)
   3. Expected output: 1
   4. Meaning: The username “John” has already been used

vi. Test case 6:
   1. Category: Invalid password
   2. Input: (“George”, “123”)
   3. Expected output: 2
   4. Meaning: The password “123” is less than the required 5 characters

vii. Test case 7:
   1. Category: Invalid password

2. Input: (“Greyson”, “”)
3. Expected output: 2
4. Meaning: The password “” is less than the required 5 characters

viii. Test case 8:

1. Category: Empty username
2. Input: (null, “12345)
3. Expected output: 3
4. Meaning: The username input is null

ix. Test case 9:

1. Category: Empty password
2. Input: (“Tyler”, null)
3. Expected output: 4
4. Meaning: The password input is null

13. Comparison of Work with Similar Designs

Feature	Vrbo	Airbnb	SuiteSpot
Accommodation Type	**Entire homes only.** [1] Focused on traditional vacation spots (family/groups).	**Diverse**. Rooms, shared spaces, entire homes, unique stays [1], [2].	**Curated Entire Homes:** Vetted, high-quality properties with consistent, professional service. [1]
Search/Filtering	Good, dedicated filters. Filters are primary. [2]	Uses filters and **Categories** for discovery. [4]	**Semantic Search (AI-Driven):** Search by experience (e.g., "quiet cabin with hot tub"). Instant, precise results. [4]
Pricing Display	Guest service fees **added at checkout.** [3]	Guest service fees **added late in the process** (the "Surprise Fee"). [3]	**All-Inclusive Upfront Pricing:** Total, final price displayed in search results for complete transparency. [3]
Booking Process	Often requires a **booking request** (host approval). [3] Less instant.	**Mostly Instant Book**, but still relies on host interaction/messaging. [2], [3]	**Seamless & Instant:** Automated, one-click booking with digital check-in integration. [5]
Guest Communication	Less frequent, more formal (email-based messaging). [2]	In-app messaging (host-to-guest). Support is often through the host. [3]	**Digital Concierge:** Automated in-app support for common needs + 24/7 centralized human support. [5]
UI/UX Approach	More traditional/functional; filter-heavy. [4]	Visually appealing, dynamic, and mobile-friendly; discovery-focused. [4]	Mobile-first, responsive design focusing on streamlined workflow and fewer clicks to confirmation. [5]

Core Value	Privacy, space, and family-focused vacation stays. [1]	Variety, budget options, and unique local experiences. [2]	Seamless Experience, Guaranteed Quality, and Personalized Search.

14. Conclusion

In conclusion, our team has successfully completed the overall tasks of designing and architecturally planning SuiteSpot, a vacation home rental application. We have delivered extensive artifacts, such as use case, sequence, and class diagrams, as well as established the full MVC architecture that will be required in order to have our system running correctly and efficiently.

As for our implementation, we chose to focus specifically on the account creation logic, which tests to see if a username and password combination are both valid, in which case it creates an account. We successfully developed a proof of concept for this logic in order to show that we stayed consistent with what we said in our use case and sequence diagrams. We also tested all of these algorithms with a variety of cases that would test invalid usernames and passwords to make sure that all aspects of the module were up to par with our expectations. Although our initial goal was to develop everything using the MERN stack, we chose to do the testing in Java so that we could leverage the knowledge of our team and quickly deliver a working prototype that proves our user management system is logically sound, so that we could eventually transition to the MERN stack.

While our architectural design covers everything that was required of us, our implementation and testing strategies did change in a couple ways. First, we originally planned to implement a fully working application, which would include creating user profiles and authenticating them, making a database to handle listings, creating a search system that would also allow for extensive filtering, create a database for bookings, and make a payment/checkout system. After consideration, we decided to prioritize making sure that all of our architectural design and documentation was thorough and fully completed before we tried our hand at any actual full-stack application development. This aligns with the specifications, since full implementation was optimal, so we chose to focus on overall quality of our software design artifacts instead of spending large amounts of time implementing the complex software features that would be needed to have this kind of application fully working, which is why in the end we only developed the account creation part of the software.

Another change we made was deciding to use Java and JUnit in order to perform unit tests. Although the initial plan based on our project architecture was to use the MERN stack and use Jest for testing the JavaScript modules, we decided to not set up the full Node.js environment. Our justification for this is that we were allowed to write the code in whatever language we wanted according to the specifications, so because our group was pretty proficient in Java already, we were able to quickly and effectively develop a prototype and run tests to validate our business logic for the account creation module without having to setup the full MERN environment. This made it possible for us to verify that our logic for our algorithm was correct and efficient while focusing on other parts of our deliverable to ensure that everything was executed to a high standard.

Finally, one more change we made was specifying certain features we wanted to add. In our proposal, we said that we wanted to include personalized changes to make our app more user friendly. So, we decided on two specific features, which are a wishlist feature so that users could make a list of properties they liked that would give them price and booking updates and public host profiles that would allow users to get to know them and also get local recommendations from them, making the experience feel more personal. We picked these features because we felt that just saying personalized changes was too vague and these would be some interesting ideas that we hadn't really seen on other major home rental platforms like Airbnb and Vrbo. This would give users more incentive to use our application over alternatives, since our changes serve the function of keeping users constantly updated and allowing them to build trust with property owners before booking.

Following the incremental process model, our next steps would be to actually take all of our diagrams and what we showed in them, as well as our tested Java logic and architectural design and put it into the MERN stack environment, so that we could eventually deploy a fully functioning application.

15. References

[1] S. Kemmis and E. Geller, "Airbnb vs. Vrbo: Which Is Better for Travelers?," *NerdWallet*, Dec. 21, 2020. [Online]. Available: https://www.nerdwallet.com/travel/learn/airbnb-vs-vrbo-which-is-better-for-travelers. [accessed Nov. 23, 2025].

[2] "VRBO vs Airbnb vs Booking Direct: What is the Difference?," *BVW Unsalted Vacations*, Jan. 27, 2023. [Online]. Available: https://unsaltedvacations.com/vrbo-vs-airbnb-vs-booking-direct. [Accessed Nov. 23, 2025].

[3] L. Griffin, "Airbnb vs. Vrbo: Key differences every vacation rental host should know," *Breezeway.io*, Oct. 24, 2024. [Online]. Available: https://www.breezeway.io/blog/airbnb-vs-vrbo. [Accessed Nov. 23, 2025].

[4] V. Nakum, "Rethinking User Experience of Airbnb for Similar App Development," *TRooTech*, Mar. 22, 2021. [Online]. Available: https://www.trootech.com/blog/rethinking-user-experience-of-airbnb-for-similar-app-development. [Accessed Nov. 23, 2025].

[5] F. Depino, "Vacation Rental Website Design - 50 Examples to Emulate," *Mediaboom*, Oct. 30, 2025. [Online]. Available: https://mediaboom.com/news/vacation-rental-website-design. [Accessed Nov. 23, 2025].