Software Architecture Documentation for Smart Home Application System

SE 4352.001 Team 10:

Abdurrehman Zulfiqar, Jon Grimes, Sharmin Gaziani, Rikhab Yusuf, and Omar Hussain

Prepared for:

Professor Pushpa Kumar

Table of Contents

1. Introduction

1.1 Purpose

The project's objective involves developing and executing a sophisticated smart home solution specifically designed for rental properties that are not co-located. This initiative covers multiple facets of smart home innovations, concentrating on improving the visual attractiveness, security, customization, automation, and overall user experience.

1.2 Scope

The purpose of this project is to design and implement an innovative smart home system for non co-located rental properties. The application should provide secure access control while being pet and kid friendly as well as provide a personalized experience for the customer.

1.3 Overview

The Software Architecture Document contains the following subsections:

Section 2: description of the architecture, product features, context

Section 3: describes the requirements of the architecture including functional and non-functional

Section 4: describes the architecture of the system, views & behavior, and diagrams

Section 5: describes why an alternative architecture was not used

Section 6: describes a detailed analysis, quality attributes, risks, sensitivity points, trade offs, and a utility tree

Section 7: describes the size and performance aspects of the architecture

Section 8: describes the benefits of the chosen architecture

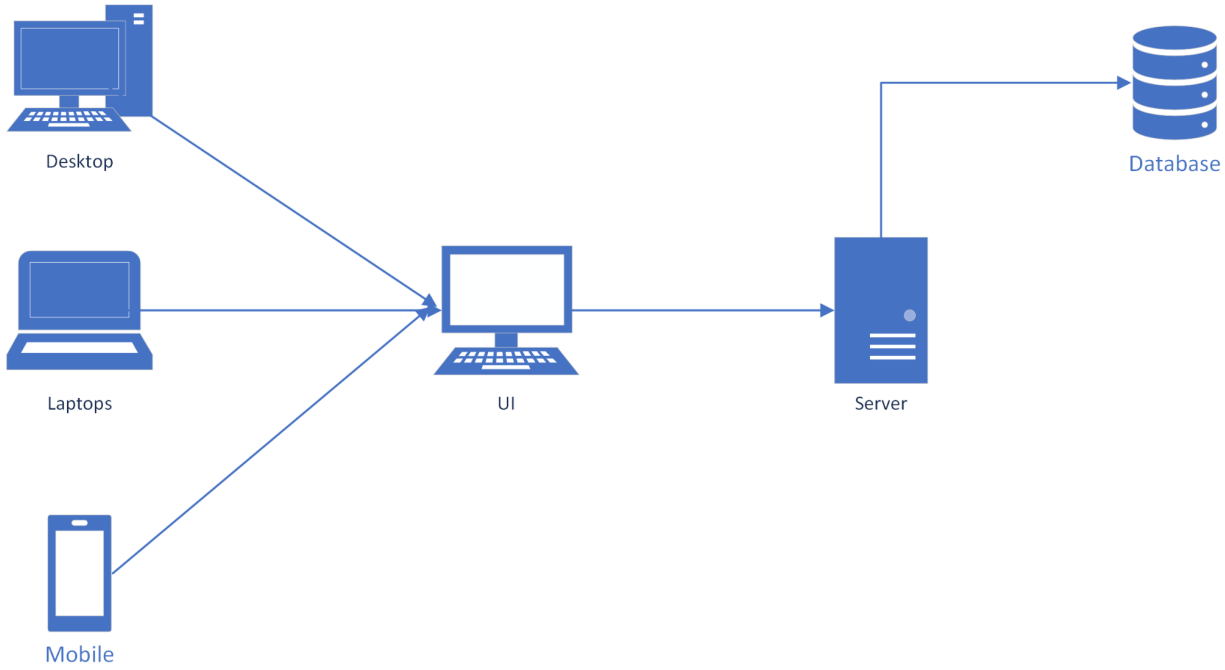
Section 9: references

Section 10: appendix

2. Architecture Description

2.1 Architecture Used

Our system is designed using a client server architecture in order to properly accommodate relaying customer requests to the property owner.



2.2 Platforms supported

Our system will support both desktop and mobile platforms. By using SwiftUI's declarative framework we are able to support both platforms with the Apple ecosystem with only one version of source code.

- ➢ This provides lower overhead cost for property owners.
- ➢ This also makes maintaining and growing the system less costly.

2.3 Product Features

The system will have a portal for both renter and rentee, providing ample functionality to both parties in order to provide the most seamless smart home experience.

2.3.1 Renter Portal Features

- **Personalization**

    - The customer shall be able to personalize their stay by adding information about themselves such as dietary preferences, music tastes, and sleep schedule and the home will tailor itself to best please the customer
- **Voice Command**
    - Customers shall be able to make commands such as “play [song title]”, “set timer for [duration]”, “start running shower at X% heat”, etc…
        - To prevent unauthorized users from making requests such as children, the user can submit voice recordings of themselves so the system shall only listen for their voice
        - For more privacy the customer is able to turn this feature off.
- **Pets**
    - The user shall be able to set a schedule for the house to set out food for their pets.
    - The home shall be equipped with lockable pet friendly doors in order to accommodate families’ pets and keep unwanted wild animals out.
- **Security**
    - The home shall be equipped with smart locks which are only unlockable via NFC sensor reading from an authorized customer's device.

2.3.2 Rentee Portal Features

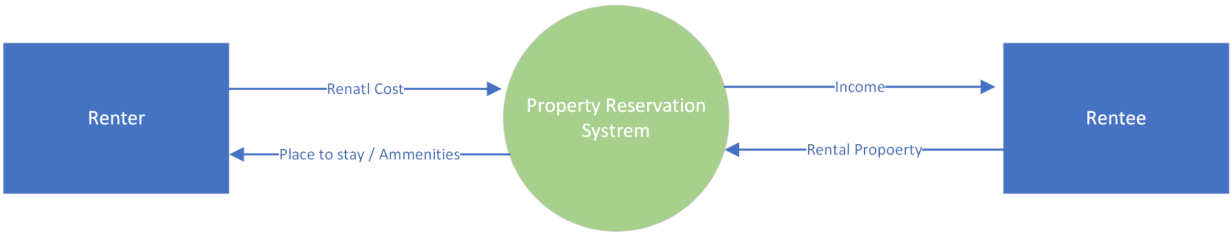
- **Billing**
    - The rentee shall be able to issue billing statements within the system
- **Property Management**
    - The system shall allow the rentee to add multiple properties.
    - The rentee shall be able to set up routines for things such as food restock, cleaning, and maintenance to occur before or after check-in or checkout.
    - The rentee shall be able to make certain features such as access to streaming services and higher quality food, such as wagyu steak, a premium feature in order to increase revenue.

2.3.3 General Features

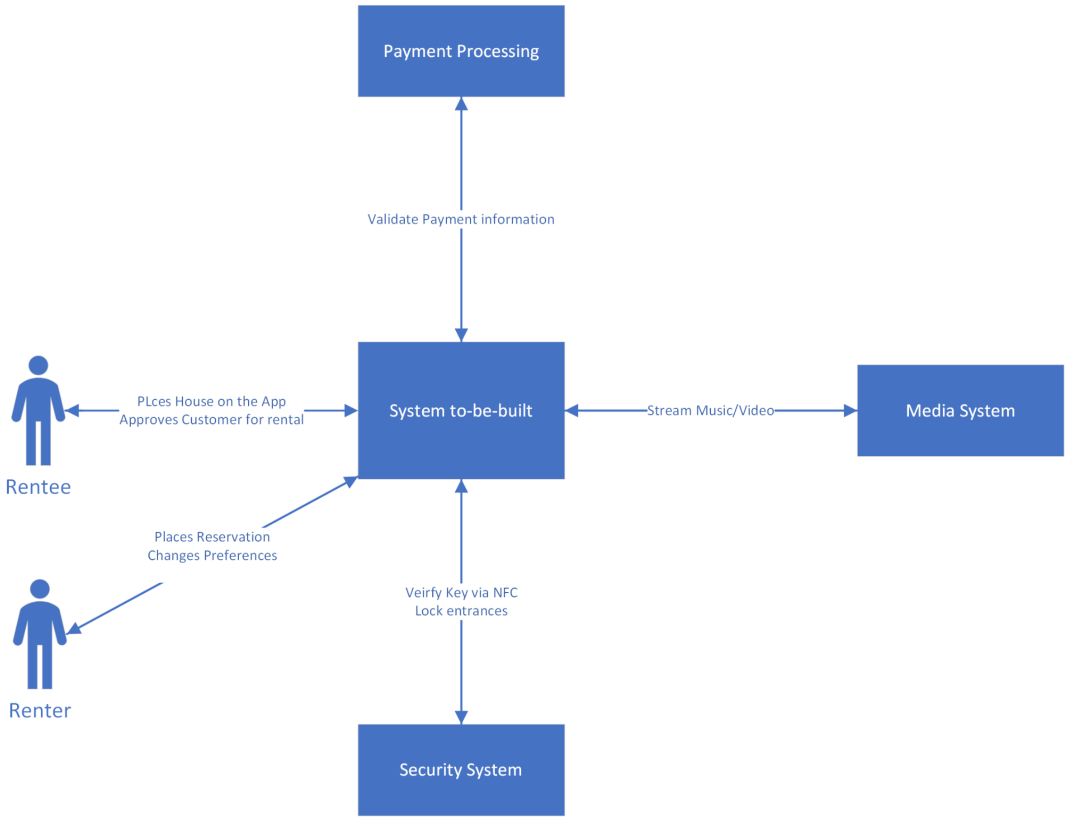
- **Data Persistence**
    - Data persistence shall be implemented using a relational database.
- **Data Privacy**
    - Sensitive data such as passwords and payment information shall be encrypted in line with AES encryption standards.

2.4 Business & System Context Diagrams

- **Business Context**



- **System Context**



3. Requirements

3.1 Functional Requirements

- **Smart Home Appearance**
  - The software should enhance the appearance of the home
  - "Smart home but better"
    - Incorporate cutting edge technology to amplify the aesthetic appeal
  - The home should be Internet-connected
  - There must be a "monitor" feature to safeguard young users
    - Example: avoiding inappropriate sites for kids
- **Security**
  - The system should provide secure access control.
    - Every access request must be authenticated, authorized, and encrypted prior to the access being granted.
  - The system should record video and audio as needed for security purposes.
    - Team discretion on how much privacy a guest will need

        - Should not be invasive.
        - Potential idea: cameras should only be in entry points of home for security reasons
    - Only authorized people should be able to come in and out of the property
        - Guests and home pets
- **Pet & Kid Friendly Features**
    - The system should prevent unauthorized access through pet doors.
    - Home should not allow kids to make unnecessary requests
        - Example: children booking flights
- **Personalization**
    - The software should aim to provide a highly personalized experience.
    - It should allow customization to make different guests feel comfortable in their home.
        - Examples include respecting dietary restrictions for others
    - Customer trend analysis:
        - Features like reminders, calendar integration, and trend analysis should be incorporated.
        - Temperature control when guest comes home
        - Stereo control based on environment
    - Ensure that amenities are used only when someone is present.
        - Downtime is only acceptable if the system is not being used
        - Potentially be scheduled
- **Voice Command and Preferences**
    - The system should be able to interpret and act upon voice commands related to user preferences.
    - User preferences should be updated on a regular cycle
    - The level of privacy regarding voice commands should be configurable by the user.
        - Privacy can be team discretion
- **Automation**
    - Automatic repair to system
    - The system should automate routine tasks without requiring explicit programming.
        - Adjusting lights
        - Automatic pet feeding
- **Alerts for Residents**
    - The system should provide context-sensitive alerts to residents, e.g., for property rental appointments.
        - Owner should have as little responsibility as possible
    - Consider integrating scheduling capabilities for this purpose.
- **Consistent Home Layouts**
    - The software should support consistent home layouts across multiple houses.
    - Consider scalability and expansion requirements for future homes.

3.2 Non-functional Requirements

- **Budget**
  - The project should aim to accomplish more with less budget.
- **Privacy and Security**
  - Security should be robust to prevent unauthorized access and key duplication.
    - Potential idea: facial recognition or software-based key for enhanced security
    - Asymmetric encryption can be used where a public key can encrypt data and that data can only be decrypted with a private key.
    - Key escrow can be used as a key backup mechanism for encrypted keys.
  - Privacy considerations should be implemented as needed.
  - Consider how much privacy users would expect in various interactions.
  - Consider audio and visual surveillance and take certain rooms into context
- **Performance**
  - The system should respond promptly to user commands and automate tasks efficiently.
- **Scalability**
  - Ensure that the system can accommodate expansion to multiple houses with consistent layouts.
- **Reliability**
  - The system should reliably perform tasks and alerts
- **User Experience**
  - Prioritize a seamless and pleasant user experience.
  - The smart home should not only be functional but also aesthetically pleasing.
  - The software should be designed with an intuitive interface to reduce the need for explicit programming.
  - Guests should not have to program certain situations into their home.

4. System Architecture

4.1 Views and Behaviors

Our system is built on a client-server architecture designed to serve both property owners and guests/rentees. The user-interface seamlessly adapts to both desktop and mobile platforms, ensuring consistent and intuitive designs. As highlighted in the previous section, our system will leverage the SwiftUI's framework to establish a unified codebase within the Apple ecosystem, minimizing development overhead for property owners.

4.1.1 Renter Portal Views and Behaviors

- **Personalization**

    - The Renter Portal showcases a user-friendly dashboard enabling customers to input their personal preferences. Our system dynamically updates their home environment based on these preferences to create an enjoyable, personalized experience.
- **Voice Command**
    - With a voice command interface integrated into the Renter Portal, guests can navigate their system without a graphical user-interface. The system responds only to authorized users to ensure privacy protection and maintain their personalized experience.
- **Pets**
    - The pet management section offers a visual representation of the user's pet-related settings. Guests can automate routine activities with our system while ensuring that unwanted animals will not have entry access.
- **Security**
    - The Renter Portal includes a dedicated security hub. Guests have control over smart locks, authorized devices, and overall home security features.

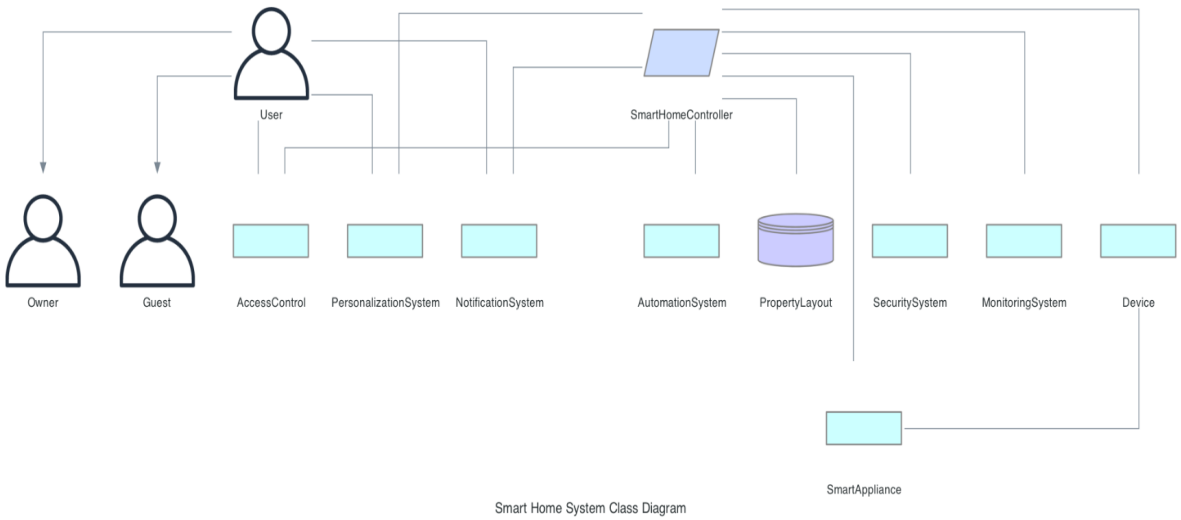
4.1.2 Rentee Portal Views and Behaviors

- **Billing**
    - The Rentee Portal displays an intuitive billing dashboard. Renters can generate billing statements within the system and any other financial documentation.
- **Property Management**
    - An interactive property management interface allows renters to oversee multiple properties seamlessly. Renters can automate routines such as restocking, cleaning, and any other maintenance to ensure a smooth experience for both renters and rentees.

4.1.3 General Features

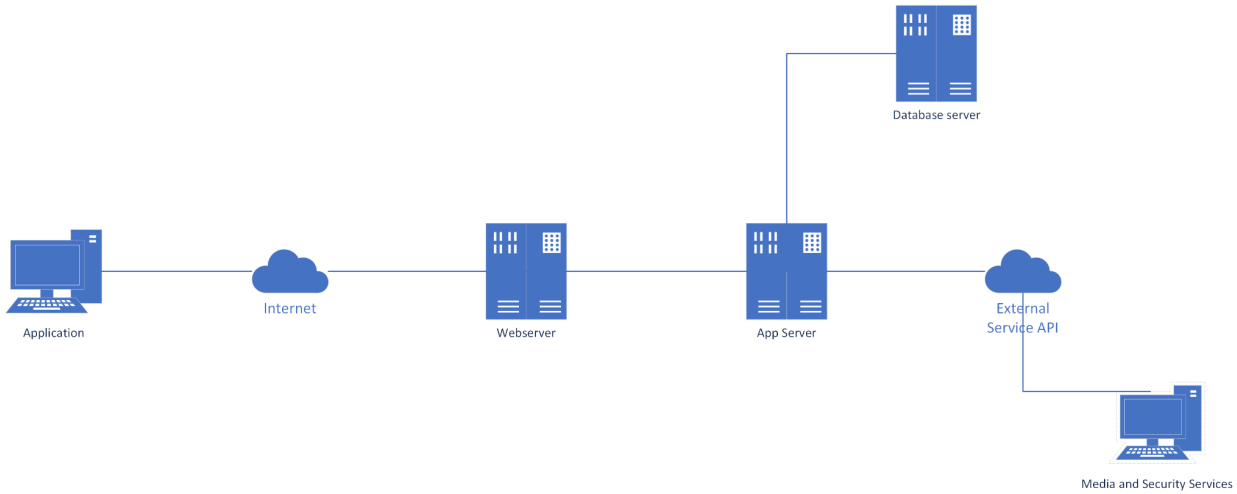
- **Data Persistence**
    - The system's database provides a clear visualization of stored data, including customer preferences and property details, utilizing a relational database to ensure reliability.
- **Data Privacy**
    - Our system offers a dedicated security settings page where users can edit sensitive information, safeguarding user data from unauthorized access using AES encryption standards.

4.2 Class Diagram



Smart Home System Class Diagram

4.3 Deployment Diagram



5. Alternative Architecture

5.1 Why not choose other architectures?

We designed our project with the client-server architecture in mind - that is to say, there is to be a centralized server that fulfills the requests of a client at the property location. In developing the groundwork and core considerations for the system, we had also discussed two other architectures that may have supported our vision for our system design. In the end, we opted to choose the client-server architecture for a few reasons, namely that it can be made more secure with additional servers with the added benefit of exceptional performance.

5.1.1 The Layered Architecture

Originally, we had considered the layered architecture for our application due to its low coupling and high cohesion. This would make it an efficient and highly performant application with clear separation of concerns. The drawback to this design, however, is that it would have to be a strictly layered system where the layer above can only interact with the layer directly beneath it - this would impact the efficiency of the application. A solution could be designed in which a layer is allowed to access all layers below it, leading to improved efficiency; this, however, is harder to implement and we wanted our design to be simple and to perform without sacrificing low coupling and high cohesion. Moreover, the layers in this design add complexity to a system and contribute to lower performance.

5.1.2 The Pipe-and-Filter Architecture

We also considered the pipe-and-filter architecture for this application because we thought that it would serve the system well - different pipes could be connected to different parts of the house where the user requests different things, which are then routed to different filters that fulfill those requests. This would give our application concurrency and independence, as well as simplicity. However, there are still problems with this design that prevented us from choosing it; for one, it is impossible in this design to exchange complex data structures. This is a very fatal flaw in the system, as many of our requests may be encapsulated using hashmaps and linked lists for security and for efficiency. Moreover, excessive use of multiple independent filters results in low performance, as this gives the system a large amount of overhead. As this is a real-time system, we desire our application to be as fast as possible to fulfill the needs and requests of the user immediately, with as little delay as possible.

6. Detailed Analysis

6.1 Risks

Privacy Concerns

The system's audio and visual surveillance capabilities raises privacy concerns for the end users. We must implement specific privacy controls, clearing user consent mechanisms, and adhering to legal frameworks will be pivotal in addressing this risk.

Security

Integrating security features, such as facial recognition and encryption, poses a potential risk of vulnerabilities. Ongoing threat assessments and regular security audits will be crucial to mitigate this risk.

6.2 Sensitivity Points

User Data

Personalizing user experiences through voice commands and trend analysis requires handling sensitive user data. Our system will implement safeguarding tools such as robust encryption standards, secure storage, and strict access controls to protect user information.

Automating Tasks

The system's automation features such as automatic repairs and maintenance, carry the risk of unintended consequences. Continuous monitoring and fail-safes will be implemented to ensure that automated tasks do not disrupt the user experience or compromise security.

Video and Audio Surveillance

With the collection of audio and video recordings, a sensitivity point is raised with obeying privacy regulations. It is crucial our system implements limited storage retention and user-configurable privacy settings to manage the data responsibly.

6.3 Quality Attributes

- **Security**
    - The system prioritizes robust security measures, including encryption, secure access controls, and facial recognition.
    - Regular security audits and continuous monitoring must be enforced to maintain an adequate level of protection against unauthorized access.
- **Reliability**

    - This is fundamental to our system to ensure routine tasks, alerts, and the security features consistently perform as intended.
    - Regular testing and failover mechanisms will be implemented to enhance overall system reliability.
- **Scalability**
    - This is a key quality attribute to support the system's expansion to multiple houses.
    - The client-server architecture is designed to accommodate multiple properties while maintaining consistency and performance.
- **Privacy**
    - With the amount of user data being stored in our system, privacy considerations are embedded in the system design.
    - Configurable privacy settings and adhering to legal frameworks will ensure user data is protected and the surveillance features prioritize privacy without compromising security.
- **User Experience**
    - The intuitive interface minimizes the need for explicit programming, ensuring both renters and rentees can easily navigate and interact with the smart home system.
- **Performance**
    - The system must respond promptly to user commands while minimizing downtime.
    - Performance testing and optimization will be ongoing to enhance the overall responsiveness of the system.
- **Maintainability**
    - Ensuring long-term sustainability and adaptability of the software is pivotal for the system.
    - Our system aims to minimize technical debt, reduce the cost of future enhancements, and enhance overall responsiveness of our smart home system.

6.4 Trade-offs

Privacy and Security

With balancing extensive surveillance capabilities, there may be a trade-off in terms of the depth of the surveillance. Our system strives to find the right balance between user privacy and home safety. Limiting storage retention and offering configurable privacy settings, users gain greater control over their personal information. This empowers users to customize their privacy preferences to ensure security is met at the proper standard, without compromising individual privacy rights.

Scalability

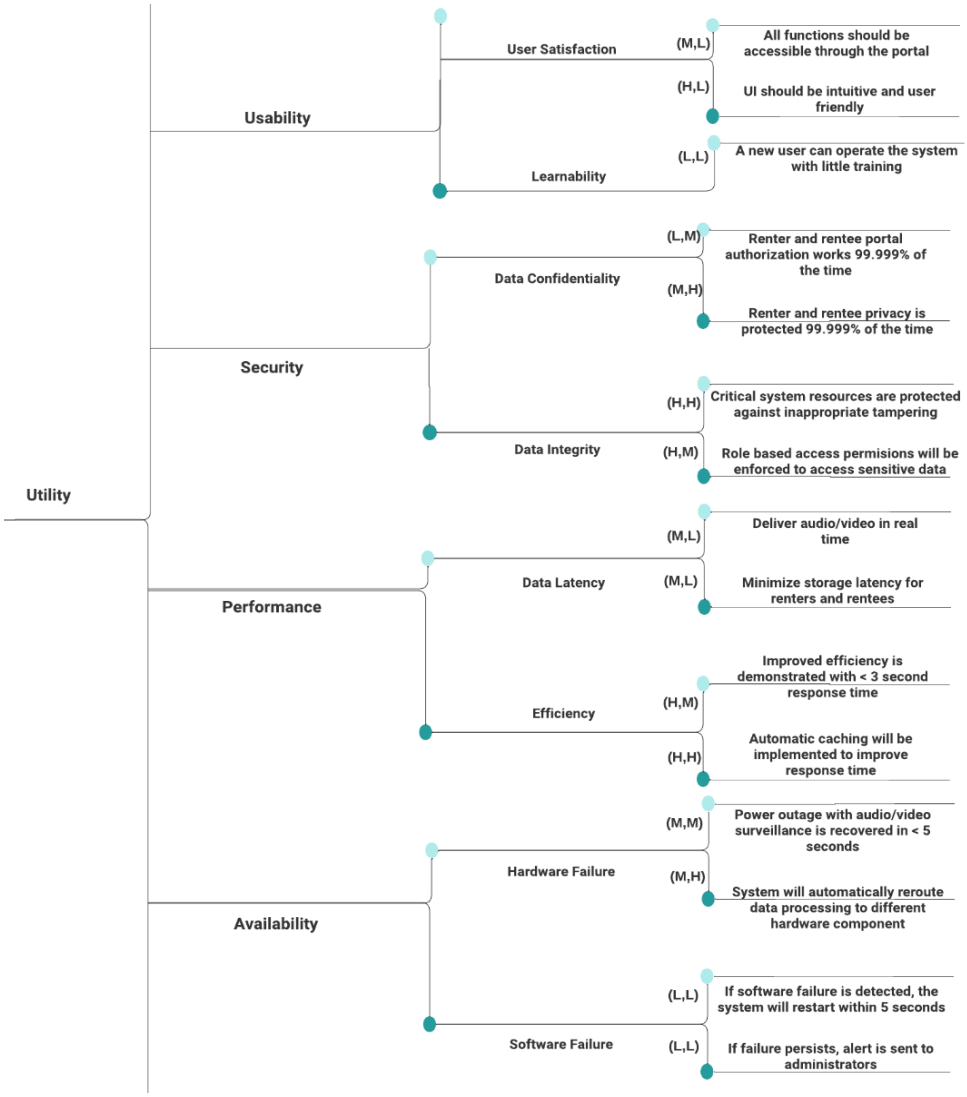
While designing for scalability, there is potential for a trade-off between immediate performance and the potential overhead of a highly scalable system. The objective is to

efficiently expand the system without introducing any compromises to the current performance standard.

User Experience

Our system places a high priority on delivering an intuitive user experience in a way that navigates the trade-off between minimizing the need for explicit programming but maintaining system flexibility. Simultaneously, we aim to create a layered experience for sophisticated users, providing advanced customization options for a more in-depth personalized interaction with the system.

6.5 Utility Tree



7. Size and Performance

7.1 Performance Capabilities

- **Response Time**
  - The system will be able to respond promptly to customer commands quickly, without any noticeable delay.
- **Processing Time**
  - The system will be able to process customer requests quickly and efficiently, without any noticeable lag.
- **Availability**
  - The system will be available at all times for the customer, even during peak usage periods.
- **Task Execution**
  - The system will have the ability to efficiently automate tasks related to controlling and automating various aspects of a smart home.

7.2 Size Capabilities

- **Load Balancing**
  - The system will be able to balance the load across multiple servers to handle increasing and varying volumes of traffic while the customer is using it.
    - As the system expands to multiple houses, additional servers can be added to handle the increased traffic and distribute the load, ensuring a consistent and responsive system.
- **Data Management**
  - The system will be able to manage and store large amounts of data efficiently for the customer.
    - This allows for efficient storage, retrieval, and manipulation of data related to smart home configurations and settings.
    - As the system scales, the server can handle larger amounts of data, ensuring consistent and reliable access to information across multiple houses.
- **Resource Management**
  - The system will be able to manage several resources efficiently for the customer such as memory, data processing, and storage.
    - This division of labor allows for efficient utilization of resources and ensures that the system can handle the increased demands of multiple houses without compromising performance.

8. Architectural Benefits

8.1 Client-Server Architecture

The client-server architecture is the best fit for the Smart Home Application System due to its numerous benefits in terms of performance, scalability, and user experience among others. This architecture divides the system into two main components: the client, which is the user interface or front-end, and the server, which is the back-end responsible for processing user requests and managing data.

8.1.1 Client-Server Architecture Benefits

- Improved Performance
    - With the client-server architecture, the workload is distributed between the client and the server. The server takes care of the complex processing tasks and data management, while the client handles the user interface and user interactions.
        - This allows for optimized performance as the server can leverage its computational power and resources to efficiently process and handle requests. As a result, clients experience faster response times and smoother interactions with the system.
- Effective Scalability
    - The client-server architecture provides excellent scalability, which is crucial for smart home systems that may expand to include multiple houses with consistent layouts. By separating the client and server components, the system can handle an increasing number of clients and their requests without sacrificing performance.
        - If the system experiences a surge in usage, additional servers can be added to distribute the workload and maintain responsiveness.
        - This scalability ensures that the smart home application system can accommodate the growth and demands of a larger user base.
- Centralized Data Management
    - In the client-server architecture, the server acts as a centralized hub for data management. Clients can access and update their smart home configurations and settings from anywhere, and the server ensures the consistency and integrity of the data.
        - This centralized approach simplifies data management, reduces data redundancy, and enhances data security.
- Enhanced User Experience
    - The client-server architecture provides a seamless and intuitive user experience for clients. The client component allows users to interact with the smart home system effortlessly. Clients can control various aspects of their smart homes, such as lighting, temperature, security, and entertainment systems, with just a few taps or clicks. The server component handles the processing behind the scenes, ensuring that the clients' commands are executed promptly and accurately.
        - This smooth user experience contributes to the overall satisfaction and usability of the smart home application system.

- Fortified Security
    - The client-server architecture can address any security concerns for the customer. By centralizing data storage and management on the server, it becomes easier to implement security measures. The server can enforce access control policies, authenticate users, and encrypt sensitive data.
        - This helps protect the client's data from unauthorized access or tampering.
- Improved Maintenance and Updates
    - With the client-server architecture, system maintenance and updates become more manageable. Any changes or updates to the system can be implemented centrally. Clients can continue using their existing client applications without requiring any updates, as long as the server remains compatible.
        - This centralized approach simplifies version control, reduces compatibility issues, and makes it easier to roll out new features and improvements to the smart home application system.
- Cross-Platform Compatibility
    - The client-server architecture enables the development of client applications that can run on different platforms and devices. Clients can access the smart home application system from desktop computers, smartphones, tablets, or any other device with network connectivity.
        - This flexibility enhances the user experience and allows clients to access the system from their preferred devices.

8.2 Model 2 Architecture connection with Client-Server Architecture

The Model 2 architecture, also known as the MVC architecture, offers several benefits for a smart home application system such as separation of concerns among many others and it connects seamlessly with the client-server architecture. It connects with the client-server architecture by dividing the application into three main components: the model which represents the data and business logic of the application, the view which is responsible for presenting the data to the user and handling the user interface, and the controller which acts as an intermediary between the model and the view.

8.2.1 Model 2 Architecture Benefits

- Separation of Concerns
    - The Model 2 architecture provides a clear separation of concerns between the data management, user interface, and application logic. The Model represents the data and functionality of the smart home system, the View handles the user interface, and the Controller manages the interactions between the Model and the View. Each component has a specific role, ensuring a clear separation of concerns.
        - This separation allows for better organization and maintainability of the system.

- Flexibility and Extensibility
    - The Model 2 architecture provides flexibility and extensibility, making it easier to add new features or integrate with other systems. For a smart home application system, this means that new devices, protocols, or automation rules can be integrated without affecting the existing codebase.
        - This allows for independent development and modification of the model, view, and controller components, ensuring that the system can adapt to evolving technology trends and user requirements.
- Consistent User Interface
    - The Model 2 architecture promotes the reuse of code and components, which means a consistent user interface can be created across multiple devices and platforms.
        - This means that the client can access the application from different devices and platforms without having to learn a new user interface each time.
- Lower Bandwidth Usage
    - The Model 2 architecture reduces the amount of data that needs to be transferred between the server and the client, resulting in lower bandwidth usage.
        - This means that the client can access the application from slower internet connections without experiencing significant delays or performance issues.
- Better Accessibility
    - The Model 2 architecture allows for multiple views of the same data, each tailored to a specific device or platform.
        - This means that the client can access the application from different devices and platforms, including assistive technologies, making the application more accessible to a wider range of users.
- Efficient Modularity
    - The Model 2 architecture allows for modular development, where each component can be developed independently. This means that updates or modifications to one component can be made without affecting the others.
        - As a result, the client can benefit from faster updates and bug fixes, ensuring a more reliable and up-to-date smart home application.

9. References

[1] J. McGovern, S. Tyagi, M. Stevens, and S. Mathew, “Component-Based Service Development,” *Java Web Services Architecture*, Sept. 2007. Available: https://www.sciencedirect.com/science/article/abs/pii/B9781558609006500063.

[2] B. Zhang, P.-L. Rau, and G. Salvendy, “Design and evaluation of smart home user interface: effects of age, tasks and intelligence level,” *Behaviour & Information Technology*, Jul. 2007. Available: https://web-s-ebscohost-com.libproxy.utdallas.edu/ehost/pdfviewer/pdfviewer?vid=0&sid=1be54bc8-740d-466b-9d4e-c9cf083d7ae5%40redis.

[3] D. Santos *et al.*, “A client–server architecture for remotely controlling a robot using a closed-loop system with a biological neuroprocessor,” *Robotics and Autonomous Systems*, Sep. 16, 2010. Available: https://www-sciencedirect-com.libproxy.utdallas.edu/science/article/pii/S0921889010001569

[4] S. A. Hussain, M. Fatima, A. Saeed, I. Raza, and R. K. Shahzad, “Multilevel classification of security concerns in cloud computing,” *Applied Computing and Informatics*, vol. 13, no. 1, pp. 57–65, Jan. 2017, Available: https://doi.org/10.1016/j.aci.2016.03.001.

[5] S. Devi and D. K. Harika, "AES encryption and decryption standards," *Journal of Physics: Conference Series,* vol. 1228, *(1),* 2019. Available: http://libproxy.utdallas.edu/login?url=https://www.proquest.com/scholarly-journals/aes-encryption-decryption-standards/docview/2566185786/se-2.

[6] Douha, N.Y.-R., Renaud, K., Taenaka, Y. and Kadobayashi, Y. , "Smart home cybersecurity awareness and behavioral incentives", *Information and Computer Security*, Vol. 31 No. 5, pp. 545-575, June 2023, Available: https://doi-org.libproxy.utdallas.edu/10.1108/ICS-03-2023-0032.

10. Appendix

Appendix 1: Definitions, Acronyms, and Abbreviations

AES - Advanced Encryption Standard

NFC - Near Field Communication

MVC - Model View Controller pattern

Relational Database - A database that organizes data into one or more tables where each table consists of a set of rows and columns. Columns represent the attributes of the data and rows represent instances of the data.

Low Coupling - A design principle that aims to minimize the dependencies between different components/modules of a system. Changes made to one module have minimal impact on other modules.

High Cohesion - A design principle that refers to how closely the responsibilities and functionality of a module/component are related. The elements within a module work together to perform a specific, well-defined task or function.

SwiftUI - A user interface framework developed by Apple across their platforms such as iOS and macOS. It is designed to work seamlessly with the Swift programming language and integrates with the Xcode development environment.

Version Control - A system used to manage and track changes to source code and other files over time. It is a critical tool for maintaining a history of changes made to the codebase.

Apple Ecosystem - Refers to the collection of devices, software, services, and technologies developed by Apple that work together to create a seamless and integrated user experience. Key components of the ecosystem include hardware, operating systems, services, and software applications.

Separation of Concerns - A principle in software development that separates a software system into distinct and independent parts, each of which are responsible for a specific functionality. It can be achieved through techniques such as abstraction and encapsulation.

Data Redundancy - A situation that occurs when the same data is repeated or duplicated in multiple parts of a database or information system.

Concurrency - Refers to the ability of a system/program to execute multiple tasks/processes simultaneously. It helps improve performance and enhances user experience.