Smart Parking System
DaVinci Code
Architectural Specifications

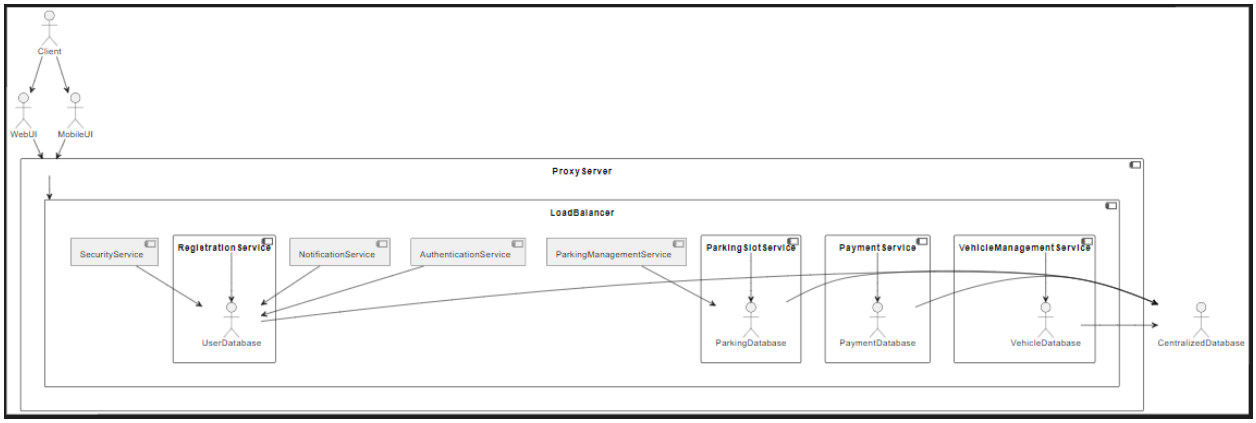
Quality Requirements

1. **Scalability:** Scalability in the Smart Parking System is crucial to ensure it can efficiently handle a growing number of users and parking facilities. The system must support extensive user registration, authentication, vehicle management, real-time parking slot searches, reservations, and bookings without performance degradation. It should process numerous concurrent transactions, including payment processing, dynamic pricing adjustments, and real-time data updates.
2. **Usability:** Usability in our Smart Parking System means providing an intuitive, user-friendly interface that ensures a seamless experience for users. Given that the system will be mostly used on mobile it will be important for the system to facilitate easy navigation through features like user registration, vehicle management, parking slot search, booking, and payment processing. It should also offer clear visual cues for slot availability and status, provide real-time updates, and enable efficient booking with minimal steps. Furthermore, an intuitive interface encourages broader adoption, including among less tech-savvy users, and reduces the learning curve for the average urban resident, making the system accessible with minimal support.
3. **Performance:** Performance is a critical architectural quality requirement for our Smart Parking System as we need to ensure the system operates smoothly and responsively under various conditions. The system must deliver real-time updates on parking slot availability, process search queries, and manage bookings with minimal latency. This focus on performance will ensure a seamless user experience, even during high demand, and supports the system's reliability and scalability.
4. **Security:** Security is a fundamental architectural quality requirement for our Smart Parking System, to ensure the protection of user data and system integrity. The system must implement robust authentication mechanisms, including multi-factor authentication (MFA), to verify user identities securely. Furthermore,

data encryption, both at rest and in transit, protects sensitive information such as personal details, payment data, and vehicle information from unauthorized access and breaches. Role-based access control (RBAC) ensures that only authorized users can access specific functionalities and data within the system. Regular security audits and vulnerability assessments are essential to identify and mitigate potential threats proactively.

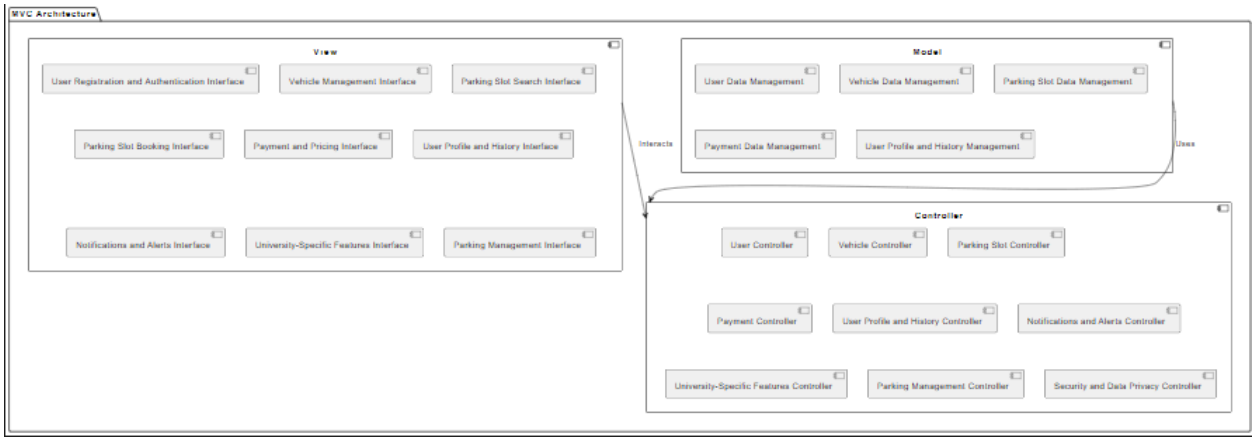
Software Architectural Patterns

1. **Service-Oriented Architecture (SOA)**:
   We decided to make use of the Service-Oriented Architecture (SOA) for our Smart Parking System as the SOA allows the system to be composed of modular services that can be developed, deployed, and scaled independently, enhancing flexibility and maintainability and allowing us to always keep our system up to date. Additionally, SOA facilitates better fault isolation, as issues in one service are less likely to impact the entire system allowing our service to stay up for users while we fix the issues. However, there are some disadvantages. SOA can introduce complexity due to the need for service orchestration and management, leading to higher overhead in terms of development and maintenance which can be resource and time-consuming.

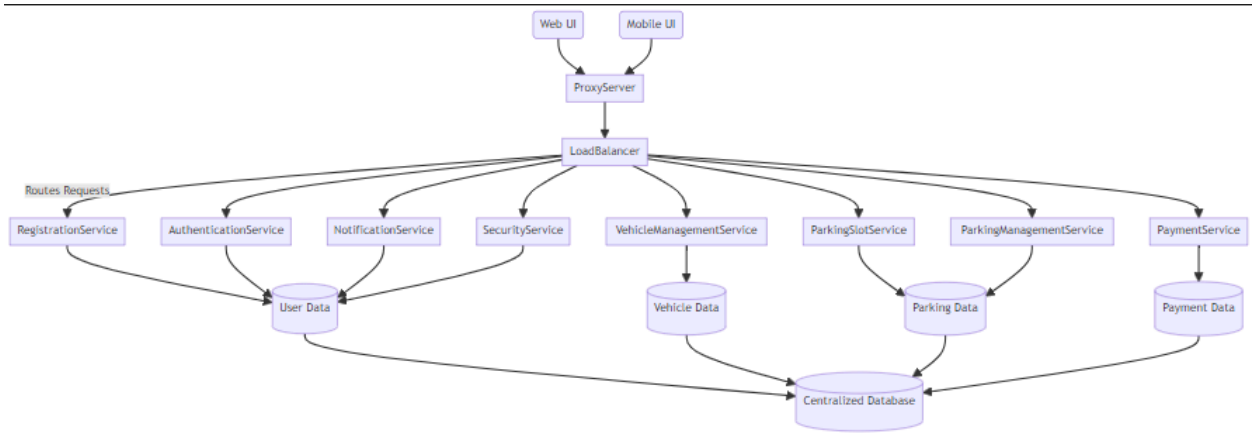


2. **Model-View-Controller (MVC)**:
   We also decided to make use of the Model-View-Controller (MVC) architecture for the Smart Parking System as MVC promotes a clear separation of concerns, dividing the system into three interconnected components: Model (data), View (UI), and Controller (logic). This separation enhances maintainability, allowing us to work on the UI, business logic, and data layers independently, facilitating parallel development and easier debugging. It also supports scalability, as each component can be scaled or modified without affecting the others. However, there are some disadvantages. MVC can introduce complexity as it requires a well-defined structure and more initial setup. Additionally, ensuring seamless

synchronization between the UI and the underlying data model can be challenging, requiring careful design and implementation.



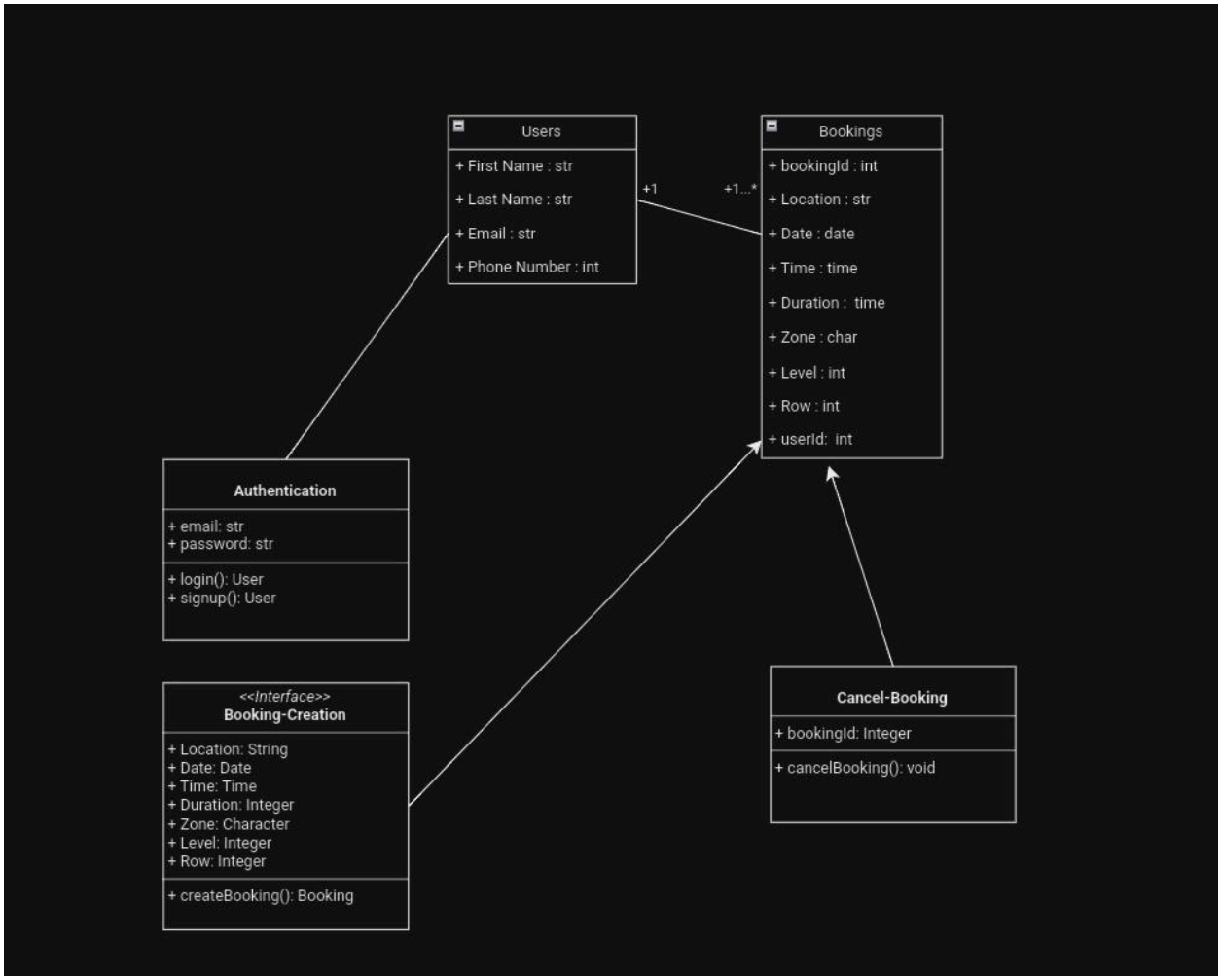
3. **Proxy Architecture:**
   We also agreed that employing the Proxy Architecture for the Smart Parking System offers several advantages. It introduces an additional layer of security by mediating requests between clients and backend services, allowing for request filtering, authentication, and encryption to protect sensitive user data. This will enhance the system's ability to prevent unauthorized access and mitigate security threats. Additionally, proxy servers can improve our system performance through caching and load balancing, optimizing resource utilization. However, there are also disadvantages. Implementing proxy architecture can introduce complexity in our system design and maintenance, requiring careful configuration and management of proxy servers. It can also introduce latency due to the additional step in request processing, potentially affecting response times. Furthermore, if the proxy server becomes a single point of failure, it can impact the availability and reliability of the entire system, necessitating robust failover mechanisms.



Constraints

1. Hardware Limitations:
   • Restricted Environments: In narrow spaces or specific topographies, camera placement may be limited.
   • Obstacles and Distances: The accuracy and distance of camera measurements depend on installation height and the presence of obstacles around the detection area.
2. Privacy and Security:
   • Protect user data (personal information) captured by cameras. Implement encryption and comply with data protection regulations.
   • Inform users about camera surveillance and obtain necessary consent.
   • Balance security with privacy rights

Class Diagram



Technology Requirements

1. **Figma:**
   We used Figma for the design and prototyping of the application. This step is essential to ensure that the final product aligns with the client's vision and meets all specified requirements. Additionally, Figma will be an invaluable reference for our development team, providing a clear guide for the application's style and expected functionality. It will essentially serve as a comprehensive blueprint that informs both the aesthetic elements and the interactive features of the front end.
   Advantages
   - Easy to prototype the UI.

- Real-time Collaboration
- Version Control

2. **Flutter**
   Flutter was used for developing the front end of the application. Flutter is a powerful UI toolkit that enables the creation of natively compiled applications for mobile, web, and desktop from a single codebase. Its rich set of pre-designed widgets and capabilities for custom designs will help us create a visually appealing and highly functional application.
   Advantages
   - Fast Development
   - Customizable Widgets
   - Single Codebase
3. **Node.js**
   Node.js will be used for the backend development of the application. It is a powerful, event-driven JavaScript runtime that enables the building of scalable and high-performance server-side applications. By using Node.js, we can handle multiple connections concurrently with high efficiency, making it an ideal choice for real-time applications such as the Smart Parking System. Node.js will be responsible for handling API requests, managing data transactions, and ensuring seamless communication between the front end and the database.
   Advantages
   - Scalability
   - Fast Performance
   - Single Programming Language
4. **MongoDB**
   MongoDB will be used as the database solution for the application. MongoDB is a NoSQL database that provides high performance, high availability, and easy scalability. It stores data in flexible, JSON-like documents, which means fields can vary from document to document, and data structure can be changed over time.
   Advantages
   - High Availability
   - Flexibility
   - Integration