ParkMe (Smart Parking System) - Software Architecture

Architectural Design Strategy

Strategy: Design based on quality requirements.

The design based on quality requirements architectural style focuses on ensuring that the architecture of a system meets specific quality attributes or non-functional requirements The idea is to prioritize these attributes early in the design process to guide architectural decisions and trade-offs.

Quality Requirements

1. **Scalability**
   Design the system to handle a large number of concurrent users and parking lots. Since the application will be used by many users, it should be able to handle a large amount of requests effectively and efficiently.

2. **Security**
   Implement robust authentication, encryption, and access controls. Since the system will contain highly sensitive information such as payment methods, car registration, user parking preferences, and vehicle location, the system should

be highly secure and ensure no breach of sensitive user data.

3. **Performance**
   Optimize response times for real-time updates. The system should have fast and reliable response and update speeds since users will need real-time updates on available parking slots and changes in parking slot availability. Furthermore, since the system makes use of high-performance services such as Google Maps, machine learning, and third-party payment services, it should have fast response time, which will mainly be achieved by incorporating caching mechanisms to store frequently accessed data, reduce server load through load balancing, and implement asynchronous processing where possible.

4. **Usability**
   The system should be easy to use and simple to understand with a user-friendly interface with a low learning curve. Since the system will have a large user base consisting of various types of individuals, it should be easily accessible and easy to understand for all types of users with a clear understanding of the different features offered by the application as well as help features to aid in its usability.

5. **Maintainability**
   The system should plan for regular maintenance and future enhancements. Due to the real-time nature of the application, it should be regularly maintained to remain up to

date with the latest changes in parking locations and Google locations so that the users are not operating on out-of-date data.

Architectural Styles

1. **Model-View-Controller (MVC):**
   - Separation of concerns
   - Maintainability
   - Reusability

MVC (Model-View-Controller) architecture is a design pattern that divides our application into three interconnected components: the Model, the View, and the Controller. The Model represents the application's data and business logic, managing the data and rules of our application. The View is responsible for displaying the data to the user and handling user interactions through the user interface. The Controller acts as an intermediary, processing user input received from the View, updating the Model, and ensuring the View displays the updated data. This separation of concerns promotes organized code, making the application easier to manage, test, and scale, while allowing for independent development and maintenance of each component

2. **Service-Oriented Architecture (SOA):**
    - Security
    - Reliability
    - Flexibility

Service-oriented architecture (SOA) is a design pattern that will structure our application as a collection of loosely coupled services, each providing a specific business functionality, allowing for interoperability between different systems and technologies. SOA promotes reusability, scalability, and flexibility by enabling services to be developed, deployed, and maintained independently. It also facilitates easier integration of diverse systems, making it ideal for complex, distributed applications that require frequent updates and scalability making our application easier to update and maintain.

3. **Layered Architecture:**
    - Scalability
    - Usability
    - Modularity

Layered Architecture is a software design pattern that organizes our application into distinct layers, each with specific responsibilities and roles. Common layers include the Presentation Layer, which handles user interface and interactions; the Application Layer, which orchestrates business logic and workflow; the Business Logic Layer, which implements core business rules; the Data Access Layer, which manages data retrieval and storage; and the Database Layer, which handles data persistence. This separation of concerns enhances modularity, maintainability, and scalability, allowing each layer to be developed, tested, and updated independently, thus simplifying our overall system management.

Architectural Constraints

1. **Scalability:** Design the system to handle a large number of concurrent users and parking lots.
2. **Security:** Implement robust authentication, encryption, and access controls.
3. **Performance:** Optimize response times for real-time updates.
4. **Redundancy:** Ensure system availability even during hardware failures.
5. **Cloud Integration:** Consider cloud-based hosting for scalability and reliability.
6. **Maintenance and Upgrades:** Plan for regular maintenance and future enhancements.