Requirement Analysis

Architecturally Significant Requirements Document

Group-1

Ahmed (Jimmy) Abdalla
Aditya Singh Gaharwar
Rhea Sera Rodrigues
Reuben Joslin Coutinho
Reshad Mohsini

Table of Contents

Architecturally Significant Requirements

Introduction

In working to create a cloud-based car repair service, it's crucial to focus on Architecturally Significant Requirements (ASRs) from the start. These are more than just basic needs; they're the foundation of the entire project's structure. Unlike regular requirements, ASRs add essential qualities to the system's design, such as scalability, reliability, security, and ease of use. These qualities aren't just nice to have—they're necessary for the service to work well and meet its goals, influencing key design decisions throughout the project.

ASRs encompass a broad spectrum of quality attributes that the system must demonstrably manifest, including but not limited to, adept performance under varying operational loads, unassailable security of customer data, and the system's adeptness at seamless integration with a multitude of external entities, such as payment gateways and vehicular diagnostic apparatus. Each architectural stratagem, from the selection of the technological stack to the formulation of user interaction paradigms, necessitates an informed foundation in these significant requisites. Neglect or misapprehension of ASRs could culminate in a system that is deficient in meeting the essential user needs or one that is rigid and unadaptable to the evolving paradigms of automotive service and technology.

The process of ASR identification transcends a mere enumeration of desired functionalities; it mandates a comprehensive and insightful engagement with the underpinning needs of the service, the extant challenges endemic to the automotive repair industry, and the anticipatory expectations of vehicle proprietors. This entails a proactive engagement with stakeholders, a thorough analysis of the competitive milieu, and a forward-looking perspective on vehicular technology and service delivery modalities. By anchoring the architectural decisions in a foundation of well-articulated ASRs, the project is poised not only to cater to the immediate exigencies of vehicle owners and service providers but also to position itself advantageously for future expansion and innovation within the domain of cloud-based automotive repair services.

Identification of ASRs.

In the formulation of a cloud-based automotive repair service, the identification of Architecturally Significant Requirements (ASRs) is instrumental in delineating the system's architectural blueprint. This process, informed by extensive stakeholder engagement and an in-depth analysis of project documentation, has elucidated a set of pivotal ASRs, each bearing a profound influence on the system's architectural design and operational efficacy:

1. **Scalability:** The system's architecture must be inherently scalable, facilitating seamless adaptation to fluctuating service demands. This scalability is crucial to managing diverse service requests ranging from routine maintenance to emergency repairs, ensuring consistent performance and user satisfaction.

2. **Reliability and Availability:** Given the critical nature of automotive repair services, the system must exhibit unparalleled reliability and availability. This entails robust failover mechanisms and redundancy to guarantee service continuity, particularly for breakdown services that necessitate immediate response.

3. **Security:** The protection of sensitive user data, encompassing personal information and transactional details, is paramount. The architectural design must incorporate advanced security protocols, including data encryption and secure data storage, to uphold user trust and comply with stringent data protection regulations.

4. **Interoperability:** The system's capacity to integrate seamlessly with a myriad of external systems, such as payment gateways, GPS for real-time tracking, and vehicle diagnostic tools, is fundamental. This interoperability enhances the service's comprehensiveness and enriches the user experience.

5. **Usability:** The user interface, serving as the primary interaction point for users, must be intuitively designed to cater to a diverse user base. This encompasses ease of service requests, clarity in service status updates, and straightforward navigation, facilitating an engaging and user-friendly experience.

6. **Compliance and Standards:** The system must adhere to automotive service regulations and environmental standards, necessitating an architectural framework that supports compliance and facilitates regular updates to align with evolving standards.

7. **Data Management and Privacy:** Effective management of user-profiles and service records is crucial for personalized service delivery and operational transparency. The system must ensure data integrity, privacy, and accessible service history for users, fostering trust and long-term user engagement.

These ASRs are derived from a meticulous synthesis of stakeholder feedback, functional and technical requirements, and the overarching business and user needs identified through the project's requirement analysis phase. Their integration into the architectural design is pivotal in realizing a cloud-based automotive repair service that is not only technologically robust but also resonant with the expectations and values of its diverse user base.

Prioritizing ASRs through a Utility Tree Analysis:

In the architectural design process, the identification and prioritization of Architecturally Significant Requirements (ASRs) are crucial. To systematically address this, we employ Utility Tree Analysis – a methodical approach that supports the refinement of quality attributes and the mapping of specific ASRs against them.

This technique commences with the conceptualization of utility as the root node – the overarching effectiveness of the system. From this node, we extend branches representing the major quality attributes required by the system, such as performance, security, and usability. Each attribute is further refined into more granular sub-attributes, for which we delineate specific scenarios that embody the ASRs.

The scenarios at the leaves of the utility tree are then assessed against two pivotal criteria: the intrinsic business value they hold and the technical risk involved in their implementation. These are scored on a simplified scale of high (H), medium (M), and low (L), providing a clear indication of which ASRs are critical and must be addressed as a priority for the system's success. The (X, Y) notation is for the business value and the architecture value

The utility tree table, as presented below, is the culmination of this analysis. It illustrates the prioritized ASRs, providing a strategic direction for architectural decisions and ensuring that the design aligns with the business goals and technical realities of the project.

Quality Attribute	Attribute Refinement	ASR Scenario
Performance	Response Time	The system processes a service request and matches with a service center in under 25 seconds during peak loads. (H, M)
	Throughput	At peak load, the system is able to handle multiple simultaneous service requests per minute. (M, M)
Usability	User Interface Simplicity	A user with no prior experience is able to initiate

		a service request within 3 minutes of using the app. (H, L)
	Accessibility	The system provides equally efficient functionality across different platforms and devices. (H, M)
Security	Data Protection	Personal and vehicle information is encrypted in transit and at rest, complying with data protection standards. (H, M)
	Intrusion Detection	Unauthorized access attempts are detected and alerts are sent to administrators within 60 seconds. (H, M)
Availability	System Uptime	The platform is operational 24/7, with maintenance windows scheduled during off-peak hours and communicated in advance. (H, L)
	Redundancy	The system automatically fails over to a backup service center in case the primary is unavailable, without user intervention. (H, M)
Interoperability	External System Integration	The system integrates with third-party payment gateways and automotive diagnostic tools with minimal latency. (H, M)
	API Flexibility	The system's APIs support integration with new service centers or towing service providers with a two-week onboarding process. (M, L)
Maintainability	Ease of Modification	Updates to the service offerings can be made in the system without the downtime and are reflected

		immediately. (M, L)
	Diagnostic and Repair Tools Support	Integration with new diagnostic tools or software can be completed within one sprint cycle. (M, L)
Regulatory Compliance	Legal Adherence	The system is updated to adhere to new automotive service regulations within one month of enactment. (H, M)
	Environmental Standards	The system includes features to ensure service centers comply with environmental standards for vehicle maintenance. (M, M)

The preceding utility tree analysis represents a structured approach to prioritizing the Architecturally Significant Requirements for our cloud-based automotive repair service. This methodology not only identifies critical system qualities but also quantifies the value and technical feasibility associated with each requirement.

This prioritization informs our architectural strategy, guiding the allocation of resources towards requirements that deliver the most significant impact on the system's performance and stability while managing risks effectively. It ensures that we remain focused on delivering a robust, scalable, and secure platform that aligns with our business objectives and provides an optimal user experience.

As we advance the architectural design, the insights from this utility tree will serve as reference points for decision-making and design justification. It will also provide a clear communication tool to align stakeholder expectations and maintain strategic focus as the project progresses.

With this analysis, we solidify our commitment to an architecture that supports our mission to transform the automotive repair experience through innovative technology solutions, setting a strong foundation for the service's current and future capabilities.

High-Level Architecture Overview:

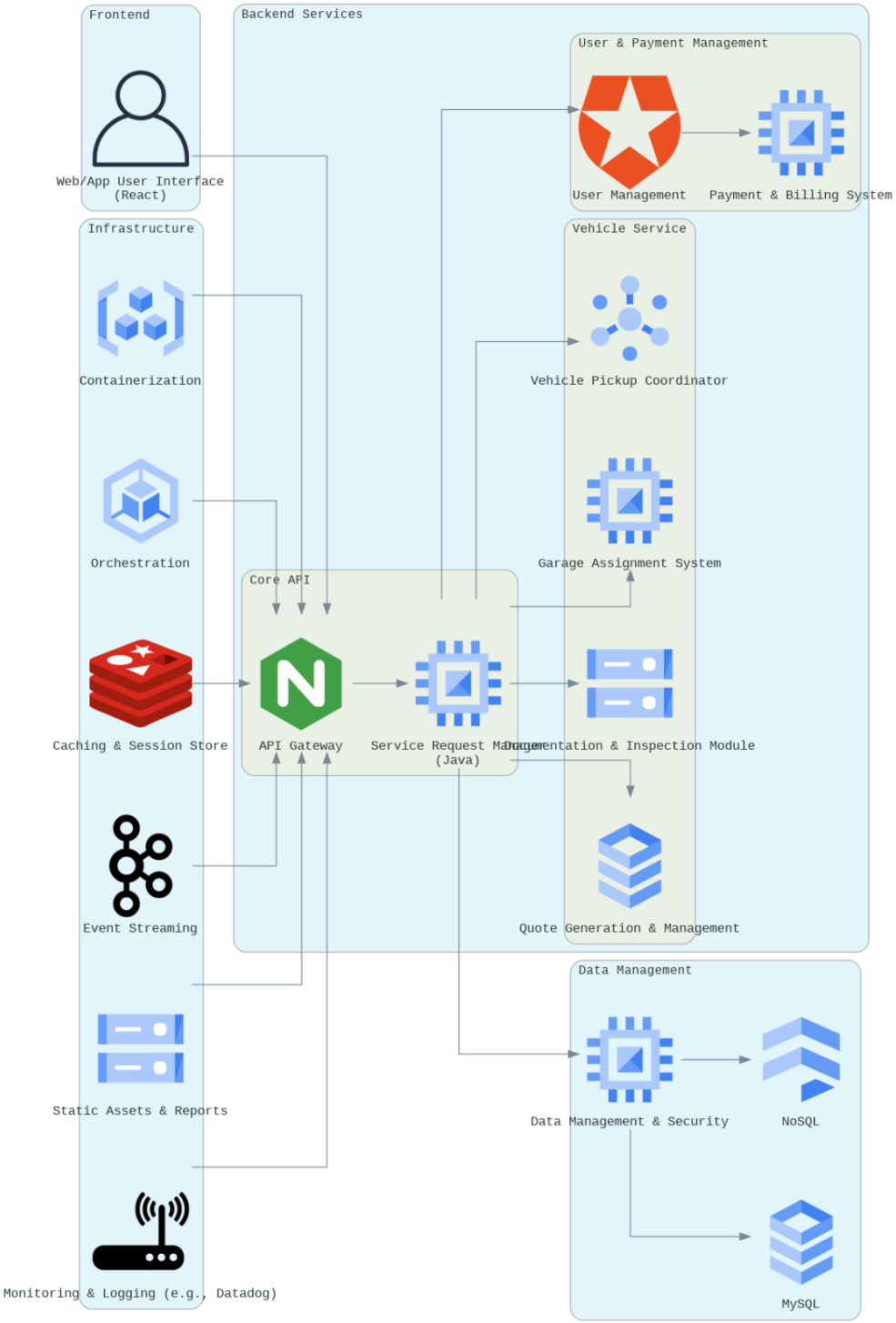
This section presents the high-level architecture for our cloud-based automotive repair service, detailing a comprehensive system design that embodies the architecturally significant requirements aspects (scalability, security, and reliability). The architecture encapsulates a front-end user interface crafted with React, a back-end processing core powered by Java (performance), and a suite of auxiliary services that ensure a seamless (interoperability), end-to-end service delivery for our customers.

The resulting architecture is a combination of multiple architectures, each built to respect an individual ASR to its maximum potential. Each architecture is built on a “bottom-up” approach. Where the goals (ASRs) were put first, and architectural components were put in place to facilitate each respective ASR. Admittedly, expediting a single ASR intensely will result in the architecture (**relatively**) exhibiting other ASRs poorly. Hence, as an architect, leveraging software technologies to cover otherwise seemingly hardware ground is essential. An example of this would be an architecture that’s built with security as a top priority, hence segregating all components in numerous geographical locations. An architecture of that manner will downgrade the architecture’s capability to perform highly and hinder scalability. As such, leveraging software functionalities such as a virtual private cloud (VPC) offers industry-level security while tying components together to negate the setbacks of complete isolation. As such, implementing multiple architectures widens the architect’s eyes on aspects that could possibly be lacking in architecture. The architectures are then faithfully put together to build the best possible architecture.

Our system is built on a foundation of containerization with Docker and orchestrated by Kubernetes (scalability), ensuring that our services are robust and can scale to meet demand. The integration of both SQL and NoSQL databases demonstrates our commitment to flexibility and performance in data management storing both transactional data from requests and media-based data from the service centers.

The architecture is designed with the foresight to accommodate future enhancements and to ensure adherence to regulatory standards, which are critical to the automotive service industry. Each component of the architecture is purpose-built to contribute to a cohesive, user-centered experience, from vehicle pickup coordination to quote management and service delivery.

The architecture is outlined in a diagrammatic representation:



Cloud-based Automotive Repair Service Architecture - GCP

Architecture Components:

1. **Front-End-React (UI & Experience):** This represents the user interface built with React, a popular JavaScript library for building user interfaces, which aligns with the usability and accessibility requirements.

2. **Back-End-Java (Service Request Manager):** Java is used for the back-end services, which is a robust choice for handling logic and data processing. The Service Request Manager would be responsible for handling the initial processing of service requests, which is a fundamental requirement of the project.

3. **Vehicle Pickup Coordinator:** This component is responsible for coordinating the logistics of vehicle pickup, which is a crucial aspect of the service delivery process mentioned in the document.

4. **Garage Assignment System:** Aligns with the automated service center matching requirement, ensuring that vehicles are matched with the appropriate service centers based on various criteria.

5. **Documentation & Inspection Module:** This is in line with the digital documentation requirement of the project, providing transparency through photos and videos documenting the vehicle's condition.

6. **Quote Generation & Management:** Reflects the need for generating and managing repair quotes, a key feature that allows users to review and approve or reject service quotes.

7. **Payment & Billing System:** Corresponds with the secure payment processing integration, supporting various payment methods and billing functionalities, including the no-charge policy for rejected services.

8. **Data Management & Security:** Essential for managing user profiles and service records securely, ensuring data protection and privacy as per regulatory compliance.

9. **Maintenance & Support:** This represents the operational requirement for a maintenance plan for the system and a support structure for users, as mentioned in the document.

10. **Containerization-Docker:** Dockerizing application components, which is a modern practice for deploying applications in a scalable and isolated environment to ensure security.

11. **Orchestration-Kubernetes:** Kubernetes technology is used for orchestrating the containers, managing their lifecycle, scaling, and ensuring high availability.

12. **Databases (MySQL, FireStore):** The inclusion of both SQL and NoSQL databases (MySQL and FireStore) reflects that the system will handle a variety of data types, from structured data to more flexible, document-oriented structures.

The described high-level architecture provides a strategic blueprint for our cloud-based automotive repair service. The solution is not only technologically advanced but also attuned to the needs of our customers and service providers.

As the project progresses, refining this architecture will not demand a dramatic overhaul, ensuring that it remains aligned with evolving business objectives, stakeholder expectations, and industry standards. Our vision is to deliver an unparalleled automotive repair experience through innovation and customer-centric design, setting a new benchmark in the industry.

Architectural Components and their value in achieving ASR goals:

Scalability

- **Google Kubernetes Engine (GKE)** and **Compute Engine**: These components are fundamental to achieving scalability. GKE allows for the orchestration of containerized applications, automatically managing scaling based on load, which is crucial for handling varying service demands. Compute Engine instances can be scaled up or down in response to demand, ensuring that the system can adapt to both routine maintenance requests and emergency repairs without degradation of performance.
- **Pub/Sub:** Facilitates asynchronous messaging that decouples services, allowing for easier scaling of individual components of the system.

Reliability and Availability:

- **Load Balancing:** Distributes incoming traffic across multiple Compute Engine instances, improving reliability and ensuring high availability by preventing any single point of failure.
- **Firestore and SQL Database:** The use of both NoSQL and SQL databases ensures that the system can reliably handle different types of data with redundancy and automatic failover capabilities, ensuring data is always available even in the event of a component failure.

Security

- **Identity Services (Auth0):** Manages authentication and authorization, ensuring that only authorized users can access their data. This component is critical for protecting sensitive user information.
- **Data Encryption & Secure Data Storage:** While specific components for encryption are not listed, GCP services like Storage, SQL, and Firestore inherently provide options for data encryption at rest and in transit, aligning with the security ASR.

Interoperability

- **API Gateway (Nginx):** Serves as the entry point for all requests into the backend services, enabling the system to integrate with external systems like payment gateways and GPS tracking services. This component is key to achieving interoperability.
- **Pub/Sub:** Offers a messaging backbone that facilitates communication between different parts of the system and external services, enhancing the system's ability to integrate with various tools and services seamlessly.

Usability

- **User Interface (React):** The choice of React for building the web/app user interface supports the need for an intuitive, responsive design that can cater to a diverse user base, providing a seamless and engaging user experience.

Compliance and Standards

- **All GCP Components:** Google Cloud's infrastructure is designed to comply with a wide range of standards and regulations. Using GCP services ensures that the application is built on a foundation that adheres to industry standards and can facilitate compliance with automotive service regulations and environmental standards.

Data Management and Privacy

- **Firestore and SQL Database:** By employing both NoSQL and SQL databases, the system ensures the efficient management of user-profiles and service records, supporting data integrity and privacy. Firestore provides scalable, flexible storage for user profiles, while SQL databases can manage structured data, such as service records, with robust access controls.

- **Data Security Components (Compute Engine):** While not explicitly mentioned, the use of Compute Engine for services like the Data Management & Security module implies the implementation of security best practices, including network security, data encryption, and access controls, which are essential for maintaining data privacy and integrity.

This architectural overview illustrates how the chosen components and services contribute to fulfilling the ASRs, ensuring the system is scalable, reliable, secure, interoperable, and user-friendly, while also adhering to necessary compliance standards and effectively managing data privacy.

References:

1. Wiegers, K., & Beatty, J. (2013). Software Requirements. Microsoft Press.
2. Anish, P.R., Balasubramaniam, B., Cleland-Huang, J., Wieringa, R., Daneva, M. and Ghaisas, S., 2015, May. Identifying architecturally significant functional requirements. In *2015 IEEE/ACM 5th International Workshop on the Twin Peaks of Requirements and Architecture* (pp. 3-8). IEEE.
3. Grünbacher, Paul, Alexander Egyed, and Nenad Medvidovic. "Reconciling software requirements and architectures with intermediate models." *Software & Systems Modeling* 3 (2004): 235-253.