## *Architectural Katas 2023*

*By O'Reilly Media*

# *The Road Warrior*

*Project Report*

*- Vibhu Mishra*

*- Subodh Kumar*

*- Madhvi Arora*

*- Amit Solanki*

# Table of Contents

# Introduction

The Road Warrior is a next generation online trip management dashboard to allow travelers to see all of their existing reservations organized by trip either online (web) or through their mobile device.

# Objective

The document discusses the architecture of the road warrior application which will help project owners and developers. The scope of this document is to describe the architecture of the road warrior project.

# Background

The Road Warrior empowers you to take control of your travel experiences , transforming the way you plan, manage and enjoy your trips . In a world filled with diverse travel options, from flights and accommodations to car rentals and activities, travelers often find themselves juggling a multitude of reservations across various platforms. Keeping track of flight details, hotel bookings, car rentals, and even restaurant reservations can be a logistical nightmare. With so much information scattered across emails, apps, and websites, the travel experience can quickly become overwhelming, leading to stress and potential mishaps along the way.

# Requirements

## Assumptions

Case study document says faster page load so we have assumed:

1. Time to Interactive should be less than 3 seconds.
2. LCP (Largest content paint) should be less than 2.5 seconds
3. CLS (Largest Contentful Paint) should be less than 0.1 seconds.
4. FCP (First Contentful Paint) should be less than 1.4 seconds.

By combining Agile principles with iterative development, the optimal approach allows for an adaptable, customer-centric, and efficient development process. This approach empowers the team to respond to changing requirements, deliver value incrementally, and build a successful and scalable web platform .

## Technical Constraints:

1. **Mobile Responsiveness:** The platform must be optimized for various devices, including smartphones and tablets, to provide a seamless application experience for customers on different screen sizes and resolutions.

2. **Browser Compatibility:** Application should be compatible with popular web browsers to ensure a consistent user experience across different browsers and versions.

3. **Geographic Distribution:** As the application aims to cater to a global customer base, the platform might require distributed server infrastructure to reduce latency and provide a better user experience for customers from different regions.

4. **Third-Party Integrations:** Integration with external services or third-party APIs must be carefully evaluated to ensure compatibility and security. Some third-party services might have limitations on the number of requests, data formats, or functionalities.

## Minimum Viable Product:

1. Application should be able to handle 2 million active users per month.
2. Application should support 15 million users accounts.
3. Application should have cross platform compatibility and backward compatibility .
4. Application should have email polling functionality for travel related emails with whitelisting and filtering features.
5. Application should be updated within 5 minutes of an update from an external agency.
6. Customers should be able to add, update, or delete existing reservations manually as well.
7. Items in the dashboard should be able to be grouped by trip, and once the trip is complete, the items should automatically be removed from the dashboard.
8. Users should also be able to share their trip information by interfacing with standard social media sites or allowing targeted people to view your trip.

# Quality Attributes

Based on the discussion among peers, going through the Vision document and considering existing similar products like Myntra, AJiO, NykaaFashion; we have charted out the following Quality Attribute Scenarios to help guide the architecture.

## Performance

- Application should load within 800 ms when the user accesses the website and mobile.
- Application should FCP under 1.4 seconds.

## Security

- The application should have WAF implemented to block malicious requests.
- Mobile devices should have tamper proofing implemented.
- Web Application should use security response headers to avoid SQL injection.
- PII data should be encrypted and the application should be VAPT compliant.

## Availability

- Application should be able to handle 2 million users per month and should be able to work as per performance limits.
- Application should have only up to 5 minutes of downtime per month.
- Application should implement a blue green deployment mechanism.
- Application servers should be Multi AZ and should have DR set up.

## Scalability

- We should be able to scale resources to handle increase in load.
- Application should have load balancer to distribute incoming traffic across multiple servers or instances.

## Extensibility

- Application should have extensible architecture that allows integration of third-party plugins or extensions to add new features or functionalities.

## Usability

- Application should have an intuitive and user friendly interface to accommodate users of all backgrounds.
- Application should be accessible to users having disabilities, complying with accessibility standards such as WCAG 2.1 .

## SEO

- Application time to interaction should be less than 800 ms .
- FCP should be less than 1.4 seconds.
- LCP should be less than 2.5 seconds .
- CLS should be less than 0.1 seconds.

## Cost Optimization

- Application should implement resource efficient coding practices and server utilization to reduce operational costs such as hosting and maintenance.

## Energy Consumption

- Application team should optimize software and hardware components to minimize energy consumption , especially for battery powered devices (Use Mobile specific technology over PWA and TWA ).

## 6 Part Analysis of Quality Attribute Scenarios

1. The application should load within 2 sec when the user accesses the website/mobile given the internet speed is at least 1mbps.
    a. Stimulus: User access the application
    b. Source of stimulus: ApplicationUser
    c. Environment: Application
    d. Artifact stimulated: Application system
    e. Response: Get website loaded
    f. Response measure: Should load within 2 sec

2. Users should be able to manage reservations only if they are authorized successfully via a login and credentials or using federated authentication.
    a. Stimulus: User managing the reservations
    b. Source of stimulus: User

c. Environment: Road warrior system
d. Artifact stimulated: Road Warrior reservation management system
e. Response: able to manage the reservation
f. Response measure: if the users have successfully logged in and are authorized

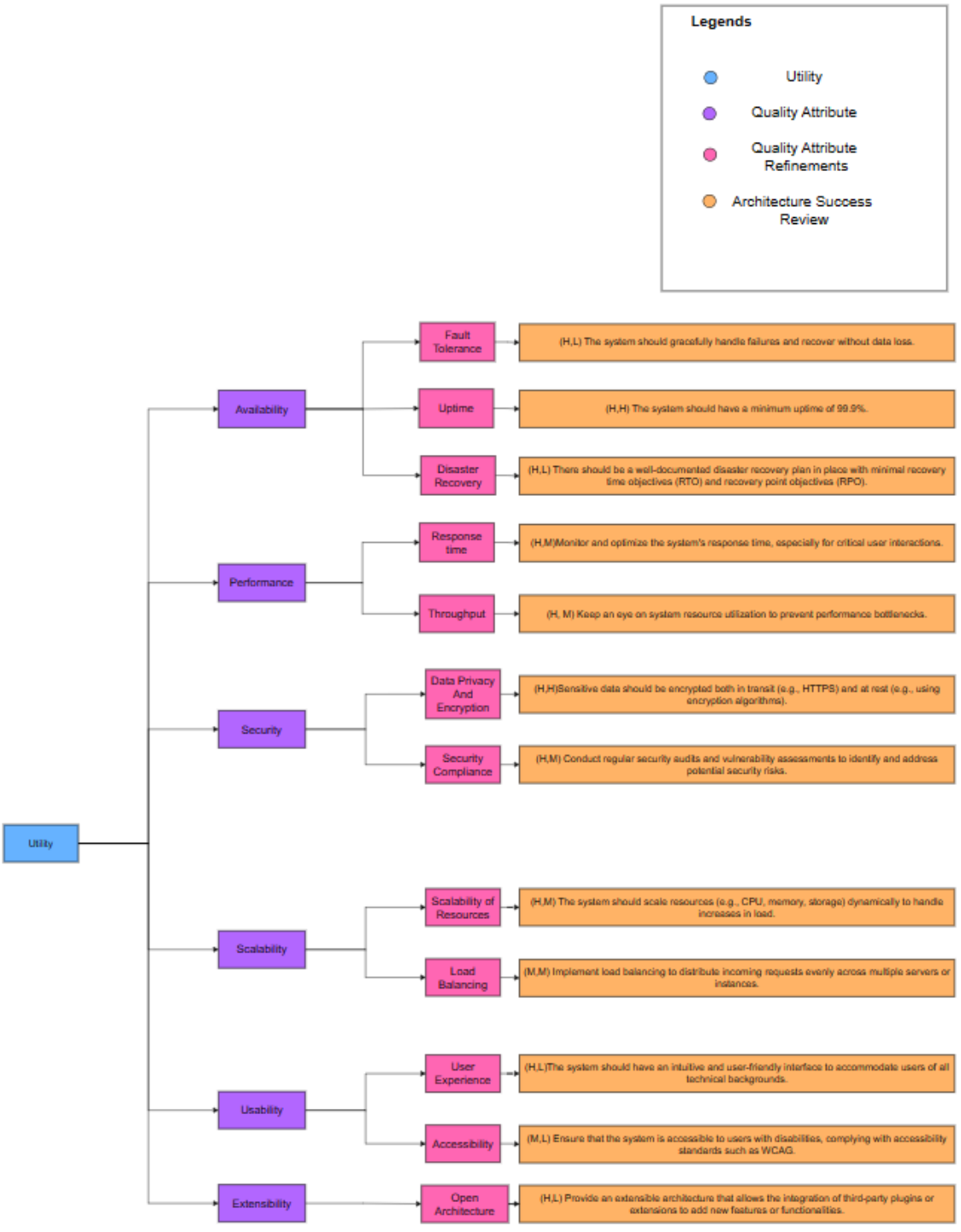
3. Application should be able to handle 2 million active users per week and should be able to work as per performance limits.
    a. Stimulus: 2 million active user per week
    b. Source of stimulus: Application users
    c. Environment: Road Warrior application
    d. Artifact stimulated: Road warrior system
    e. Response: Should able to load and response properly
    f. Response measure: should meet performance limit.

4. Application should have only up to 5 minutes of downtime per month in case of different types of failures such as hardware failure, software crash, etc.
    a. Stimulus: occurrence of downtime
    b. Source of stimulus: different types of failures as hardware failure, software crash, etc
    c. Environment: Road Warrior application
    d. Artifact stimulated:Road Warrior application
    e. Response: minimize and ensure that the downtime remains within the specified limit of 5 minutes per month
    f. Response measure: Should be less than 5 minutes per month.

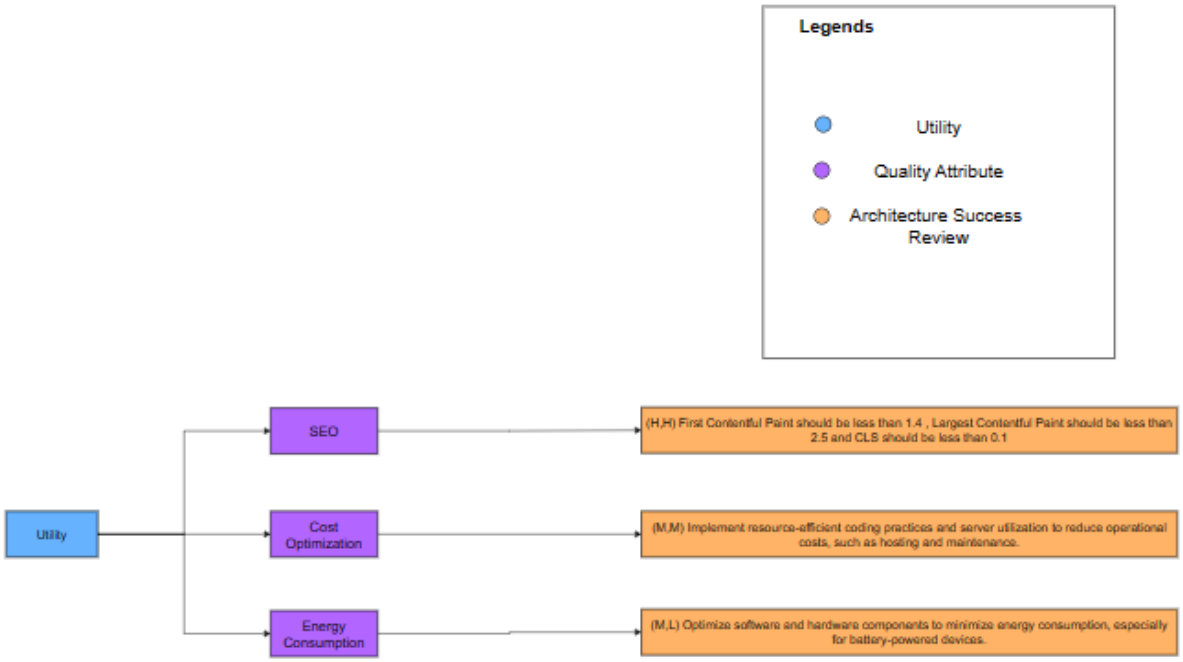
| Quality Attribute | Priority | Trade-offs |
|---|---|---|
| Performance | High | Cost using a combination of CMS and frontend technology which may increase server cost .<br>Maintainability : Using complex solution to achieve best performance |
| Availability | High | Cost : Achieving 100% uptime vs. cost and complexity trade-offs. Having MultiAZ and DR setup is expensive |
| Security | High | Latency : Implementing strong security measures without compromising with extra WAF layer which will increase latency in application. |
| Usability | Medium | Maintainability: Ensuring usability doesn't compromise ease of maintenance. |
| Scalability | Medium | Cost : Infrastructure cost will increase . |
| Extensibility | Medium | Maintainability : Maintaining code can become tough in order to get better extensibility and configurability. |

In this trade-off matrix, the quality attributes are prioritized based on their importance to the project. High priority attributes include Performance, Availability, and Security. Trade-offs are identified for each attribute, where the decisions made in one area may have impacts on other attributes or project considerations. Medium and Low priority attributes have less significant trade-offs, but they still require thoughtful consideration in the development process to achieve the best overall outcome for the application project.

## Utility Table/Tree




Traditional Quality Attribute Utility Tree




Non Traditional Quality Attribute

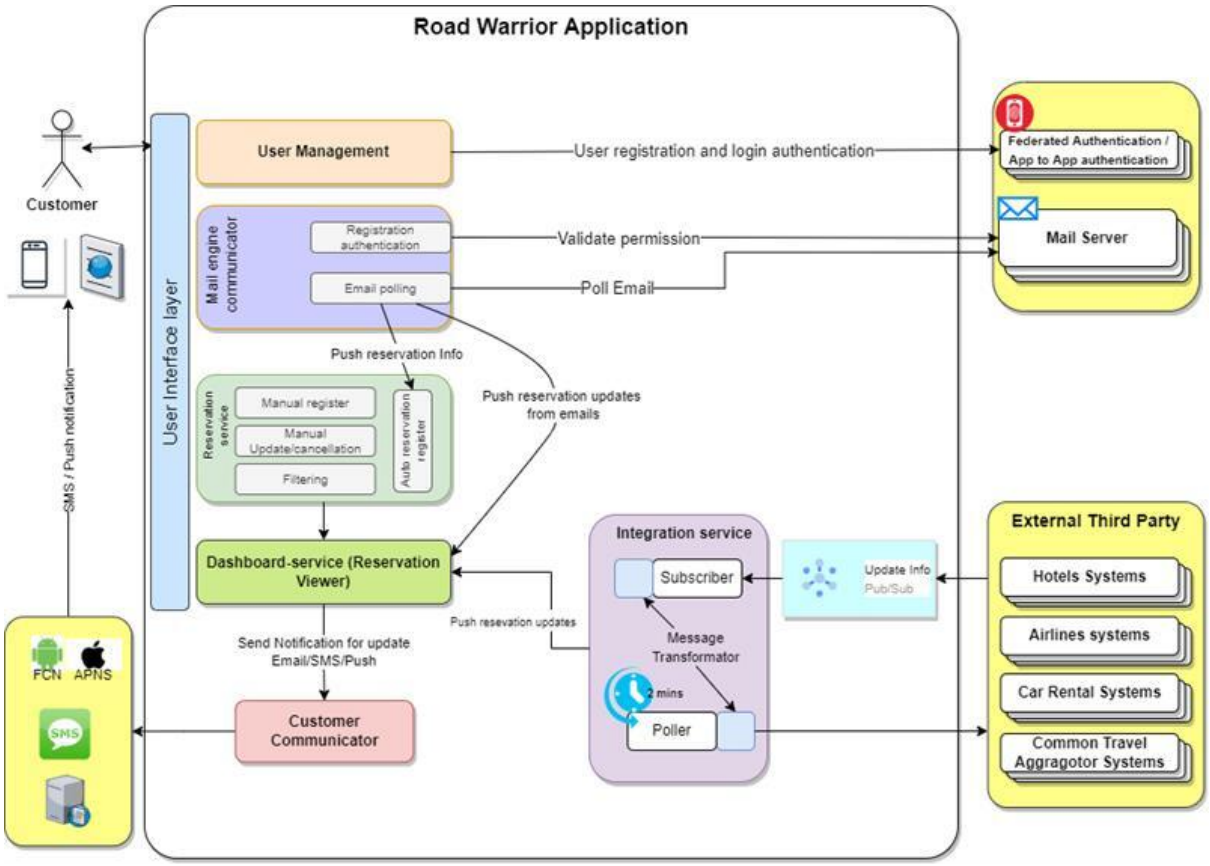
## Architecture Styles

We have analyzed various possible architectural styles for building the application keeping in mind the architectural drivers in terms of Quality Attributes and Technical and Business Constraints.

- **Microservices Architecture**: Breaking down the system into small, independent microservices allows for flexibility and scalability. Each microservice can handle a specific aspect of trip management, such as reservations, notifications, or user authentication, and can be developed, deployed, and scaled independently.
- **Client-Server Architecture**: The application will be using client -server architectural style where clients (web browsers and mobile devices) communicate with a central server or a cluster of servers. The backend server will manage data storage, business logic, and authentication, while clients handle the user interface and user experience.
- **REST Architectural Style:** We will be using RESTful web services for developing backend services at it gives simplicity, statelessness, scalability, interoperability, resource-centric, cacheability, flexibility, incremental development, self-descriptive messages, global reach, maturity and industry adoption

- **Event-Driven Architecture:** When reservations change, flights are delayed, or other travel-related events occur, the system can notify users in real-time through events and messaging systems. An event-driven architecture can be used to handle real-time updates and notifications.
- **Caching:** Implementing caching mechanisms, such as Content Delivery Networks (CDNs) or in-memory caching, maintaining browser cache for static pages can enhance the system's performance by reducing the load on backend services and speeding up data retrieval.This will help in search engine optimization by decreasing FCP, LCP, Time to Interactive and CLS.
- **Containerization And Orchestration:** We will use containerization using technologies like Docker and orchestration with Kubernetes can simplify deployment and scaling of microservices. Containers provide consistency across development and production environments, while orchestration tools manage container deployment and scaling.
- **Authentication and Authorization:** We will use federated authentication (app to app , login via gmail , facebook,etc.) and OAuth 2.0 and authorization can enhance security and allow users to access their trip data securely.
- **Load Balancing:** We will use load balancers to distribute incoming traffic across multiple servers or instances to ensure high availability and good system performance.
- **Monitoring And Logging:** We will use robust monitoring and logging systems to track system health, performance, and user activities. (tools like Prometheus and ELK Stack can be useful for this purpose )
- **Scalability and Elasticity:** Design the system to be horizontally scalable so that it can handle increased loads during peak travel seasons. Cloud services like AWS Auto Scaling can help automatically adjust resources as needed.
- **Security Layers:** Application should have multiple layers of security, including encryption of data in transit (HTTPS) and at rest, role-based access control, WAF layer and regular security audits to protect user data and system integrity.
- **API First Design Approach :** Application should follow API first approach in case of third party integrations.

# Application Component Diagram




Component & Connector Diagram

The above diagram depicts the components of the system and their integration and connections.

**User Interface Layer :** Customers can use mobile as well as web applications to access The Road Warrior.

**User Management :** This component takes care of user registration and login using federated authentication.

**Email Polling Service :** Once the user has logged in , email polling service polls the email from the server using either POP3 or IMAP protocol.

**Reservation Service :** Email Polling Service pushes the notification to reservation service and Dashboard service.Using Reservation Service either reservations are auto register or user can add/update/cancel/filter the reservation.

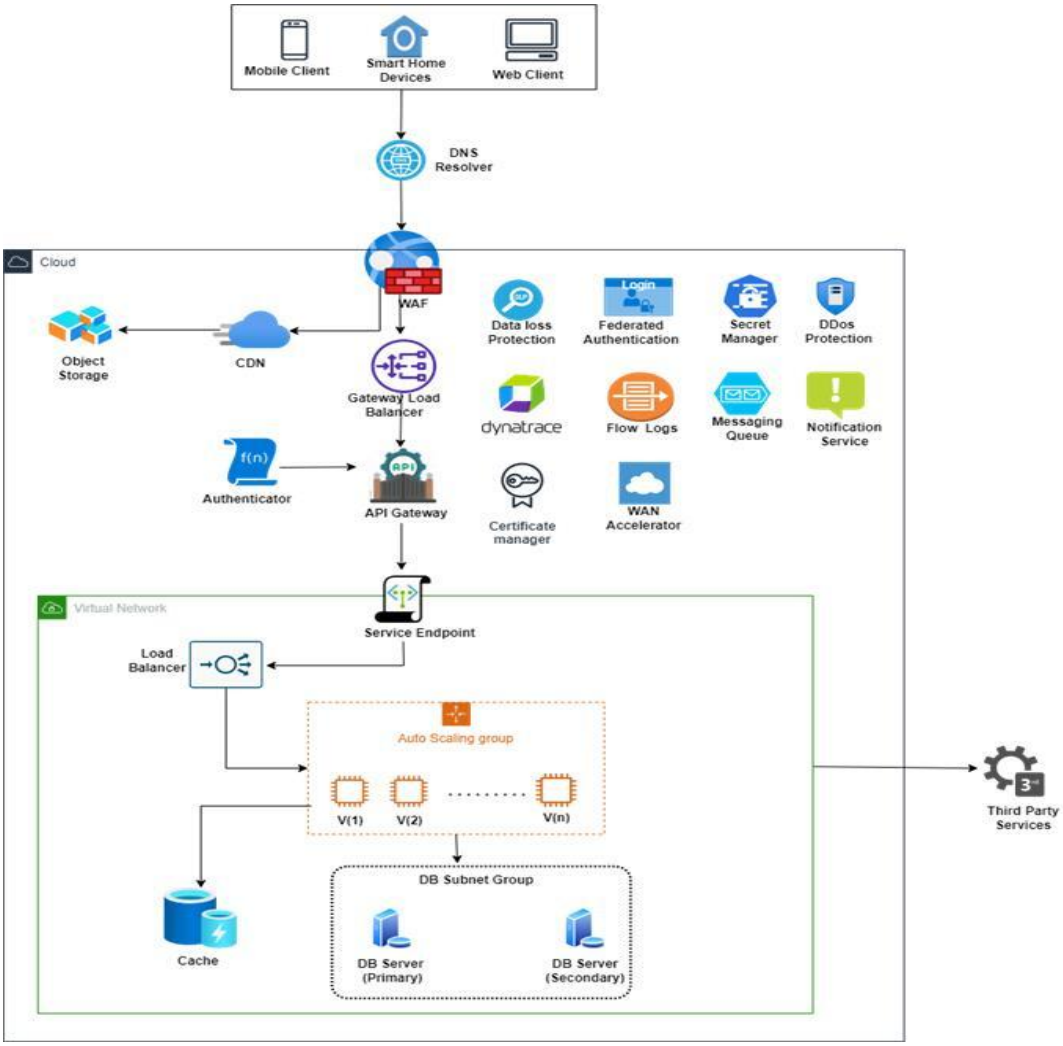
**Dashboard Service :** On dashboard service gets the notification it displays the information on the dashboard screen and sends the update to customer communicator component via email,sms and push notification.

**Customer Communicator :** Customer communicator sends the update back to the customer.

**External Third Party :** External Party / Aggregator are the agencies specific to hotels , car rental systems and airlines which sends updates on the bookings. This information is published .

**Integration Service :** The integration service subscribes the published information from external systems and pushes it to the dashboard.

## Architecture-Diagrams




High Level Diagram

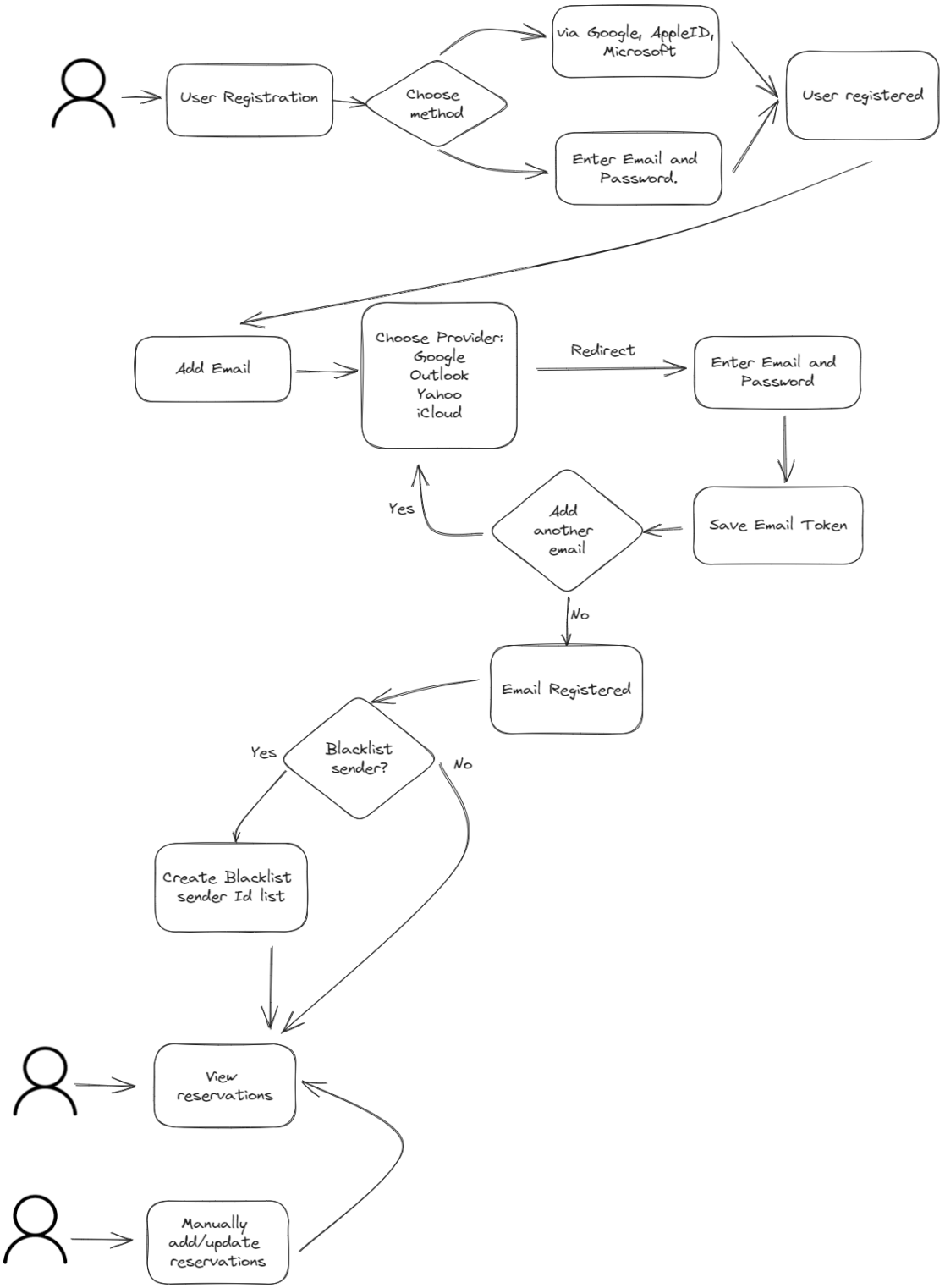
The style of the above HLD is a combination of a cloud-based and microservices-based deployment approach. It leverages various cloud services and microservices to create a

scalable and modular architecture for handling user requests in a website/mobile application.

- Explanation of High Level diagram :
    - The High-Level Design (HLD) of our proposed architecture outlines a robust and versatile framework designed to support a wide range of clients, including mobile devices, smart home devices, and web clients.
    - At its core, this architecture leverages the capabilities of multiple cloud providers to ensure both security and scalability.
    - The infrastructure is divided into several essential layers, starting with a DNS Resolver and Web Application Firewall for routing and protection.
    - The CDN Network with Object Storage and API Gateway Load Balancer enhances content delivery and application performance.
    - A critical aspect is the Scalable Instance Servers that can dynamically scale to meet varying workloads, backed by resilient Cache and NoSQL DB servers. Additionally, we incorporate third-party integrations with travel industry systems and deploy various additional components for data security, identity management, secret management, DDoS protection, log observability, messaging, notifications, certificate management, and WAN acceleration.
    - This comprehensive design aims to provide a secure, reliable, and highly scalable infrastructure for our mobile app, ensuring a seamless user experience across various client devices while leveraging the strengths of multiple cloud providers and third-party services.

      This architecture leverages specific cloud services and tools to create a resilient and versatile infrastructure, allowing you to deploy and manage your web/mobile app across various cloud providers while benefiting from their respective services and capabilities.

# User Journey Workflow

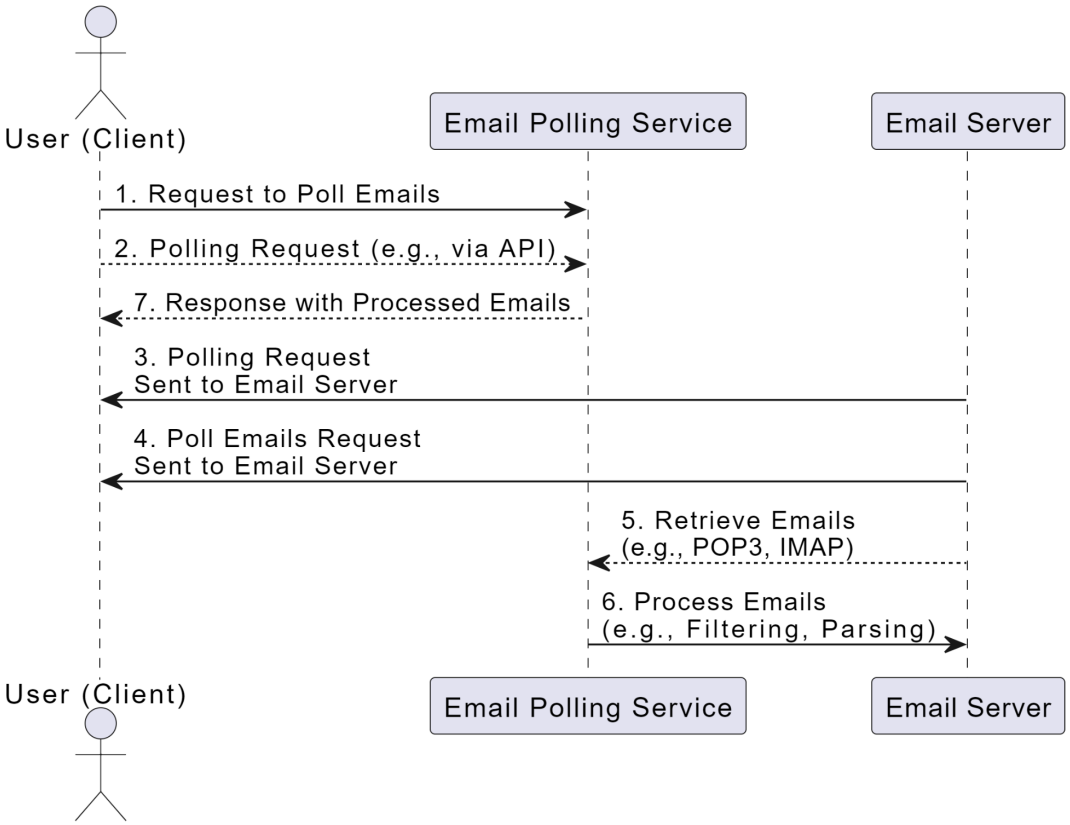



User Journey Workflow

The above diagram depicts the user journey flow.

- User registers using federated authentication (via Google , AppleID , Microsoft ).
- Users add his/her email and choose the provider for registration .
- The request is redirected to the provider app for authentication .
- After authentication is done , a token is generated which is stored for authentication in the Road Warrior application.
- A prompt box is displayed for the user to confirm if they want to register from another email ID.
- If the user wants to register from another email ID then the user is redirected to the second step.
- If the user wants to continue with the same email ID , then the user gets registered with the same.
- Backend Service checks whether the id is valid or blacklisted .
- If it is blacklisted , then it is stored as a blacklisted email ID.
- If it is a valid email ID then the user is redirected to view his/her reservations where the user can add or update the reservations manually.

# Email Polling Service




**Sequence Description:**

- The user (client) initiates a request to poll emails from their email account using the Email Polling Service.
- The client sends a polling request to the Email Polling Service, typically via an API or other communication mechanism.
- The Email Polling Service forwards the polling request to the Email Server responsible for the user's email account.
- The Email Server receives the polling request and processes it, retrieving the user's emails (e.g., using POP3 or IMAP).
- The Email Server sends the retrieved emails back to the Email Polling Service.
- The Email Polling Service processes the incoming emails, which may involve filtering, parsing, or other actions.
- The Email Polling Service responds to the client with the processed emails or relevant information.

# Architecture Decision Records

## Hosting On-Premises vs On-Cloud

### Status

The proposed approach of hosting application On-Cloud has been accepted.

### Context

The project needs to scale appropriately when regions are increased or during certain times of the year like holiday season. We are discussing whether to host the application on-premises or on-cloud

### Decision

We are proposing to host the application on cloud.

### Rationale

- **Scalability:** Cloud platforms offer the ability to scale resources up or down based on demand. This ensures that road-warrior can handle varying levels of user traffic during peak times, such as holiday sales, without the need for additional physical infrastructure.

- **Flexibility:** Cloud hosting provides flexibility in terms of resource allocation and configurations. This allows for quick adjustments to meet changing business needs or to accommodate potential growth in the future.

### Consequences

- Promotes scalability quality attribute. Since we are hosting on-cloud on AWS, it is easy to scale horizontally by adding more instances or vertically by using instances with more computation power.

## Using Headless CMS vs complete COTS like Wordpress, Drupal, etc

### Status

We finalized on using Headless CMS.

### Context

Using Headful CMS or COTS products would lead to compromise over performance by almost 50%.

#### Rationale

There are certain technologies like NEXT JS (front-end technology) with server side rendering and Strapi JS (Headless CMS) which offer better SEO benefits in combination .

#### Decision

We are proposing to use Headless CMS instead of Headful CMS

#### Consequences

- Performance benefits and SEO benefits are not well supported by using complete COTS. Using Headless CMS, we could leverage those.
- We can avoid scenarios where the system might be impacted due product decommissioning or shutdown of the product vendor.

### Using SQL vs No SQL

#### Status

The proposed approach for using NoSQL databases has been accepted.

#### Context

With the constraint of delivering the project within the scope of MVP, we have discussed if we should integrate No SQL or SQL or both.

#### Decision

We are proposing to use NoSQL instead of SQL .

#### Rationale

We are proposing to use NoSQL instead of SQL as with the email polling we will be getting unstructured data from different hotels, car rental systems and airline industries . So NO-SQL database is preferred over SQL database.

#### Consequences

- Given the scope of the project using these No SQL would help us to gather unstructured data from different systems .

## Future Steps

- We can use Google analytics to analyze the adoption of the application by the customers.

- We can plan for an external database for Headless CMS which will be a SQL database
- We can plan to set up a data lake to provide data analytics .

# References and Tools

## References:

1. We referred to Katas Challenge documents .
2. We referred to The Object Management Group (OMG) standard for UML conventions.
3. We referred to Documenting Software Architectures: Views and Beyond, Second Edition. Paul Clements et al. Addison Wesley 2011 for good documentation practices.
4. Existing web/mobile platforms like Makemytrip and Cleartrip for existing architectural references.

## Tools:

Draw.io and Excalidraw.io Confluence for architectural diagrams and exercises , Google Docs and Google Slides