Assignment 4: Sri Tel - New Customer Experience

Service Oriented Computing (CSC 4222)

Group Members:
Member 1 (Index)
Member 2 (Index)
Member 3 (Index)

January 8, 2026

Contents

1 Introduction

Sri Tel Ltd (STL), a telecommunication provider offering GSM/3G/4G/LTE services, aims to increase market share by improving customer care and experience. The current initiative involves deploying a state-of-the-art "Sri-Care" solution comprising a web portal and smartphone applications (iOS/Android).

The primary objective of this solution is to allow existing customers to manage their accounts, pay bills, and activate services without manual intervention from support staff. The solution must integrate with an existing Provisioning System and an external Payment Gateway while ensuring scalability, security, and high availability for notification delivery.

2 Proposed Architectures

To address the requirements of scalability, multi-client support (Web & Mobile), and integration with external systems, we analyzed two distinct architectural approaches.

2.1 Alternative Architecture 1: Service-Oriented Architecture (SOA) with ESB

The first approach utilizes a traditional Service-Oriented Architecture centered around an Enterprise Service Bus (ESB).

Description: In this model, an ESB acts as the central backbone. All client requests are routed through the ESB, which orchestrates calls to various backend services. The ESB handles message transformation, protocol bridging, and routing.

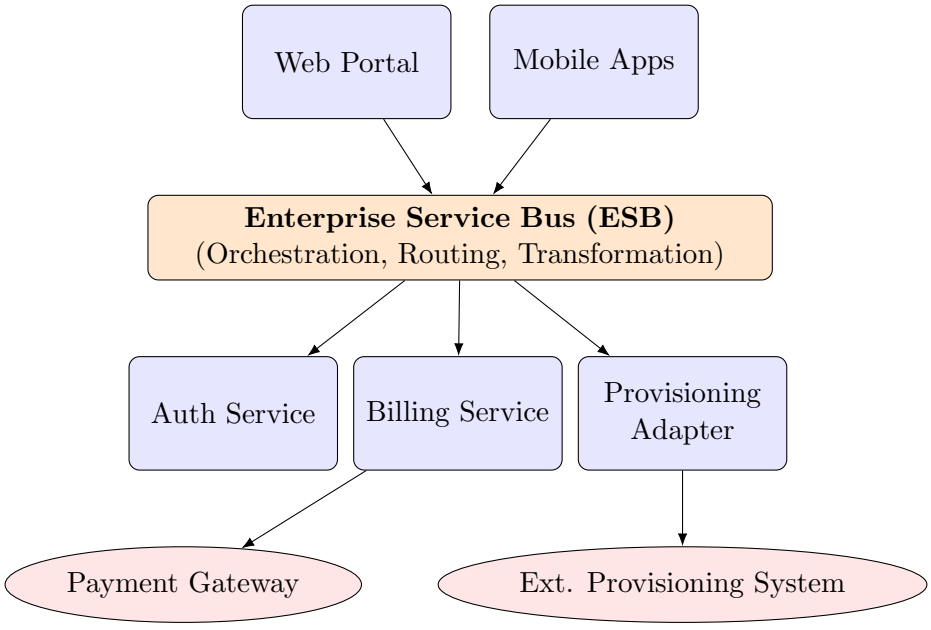

Figure 1: Alternative 1: SOA with ESB

Pros and Cons:

- **Pros:** Strong integration capabilities; centralized management of policies; mature technology for protocol mediation.
- **Cons:** The ESB can become a single point of failure and a performance bottleneck; logic often bleeds into the ESB (smart pipes), making it harder to scale individual components; heavy-weight for simple mobile app backends.

2.2 Alternative Architecture 2: Microservices Architecture (Selected)

The second and selected approach is a Microservices Architecture utilizing an API Gateway and Event-Driven asynchronous communication for notifications.

Description: The application is broken down into small, independent services (User, Bill, Notification, Provisioning). An API Gateway sits in front of the services, handling routing, authentication, and rate limiting. Inter-service communication is primarily RESTful for synchronous operations and Message-based (Pub/Sub) for asynchronous tasks like notifications.

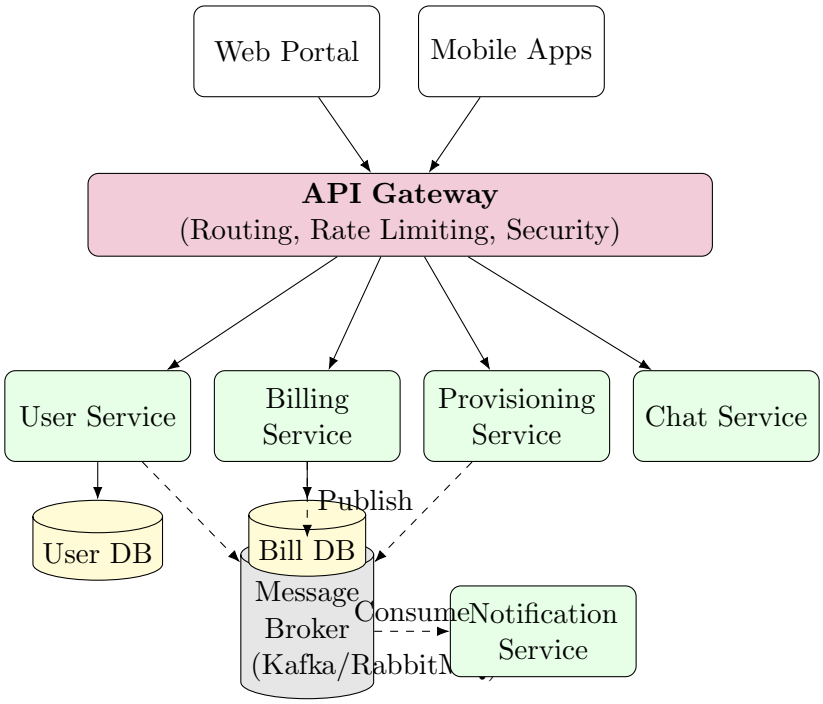

Figure 2: Selected Architecture: Microservices with Event-Driven Notifications

2.3 Rationale for Selection

We selected the **Microservices Architecture** for the Sri-Care solution for the following reasons:

- **Scalability:** The notification service is expected to handle high volumes ("best effort"). Microservices allow us to scale the Notification Service and Message Broker independently of the User or Billing services.
- **Agility:** The "Sri-Care" portal and apps require frequent updates. Independent deployment of microservices allows faster iteration without redeploying the entire monolith.
- **Technology Heterogeneity:** We can use Node.js for the I/O-bound Chat service and Java/Spring Boot for the transaction-heavy Billing service.
- **Fault Isolation:** A failure in the Chat service will not impact the critical Billing or Provisioning functionalities.

3 Architectural and Integration Patterns

Referencing the course notes on Microservices and Integration Patterns [5, 4], the following patterns are implemented:

3.1 Architectural Patterns

- **API Gateway Pattern:** Single entry point for all clients (Web and Mobile). It abstracts the underlying microservices, aggregates responses (reducing round trips for mobile users), and enforces security policies (SSL termination, Authentication).
- **Database per Service:** Each microservice (User, Billing) owns its database schema to ensure loose coupling. Changes to the Billing database do not impact the User service.
- **Backend for Frontend (BFF):** Although a single Gateway is proposed, logically we treat the APIs as catering to specific needs of the mobile/web clients, allowing optimized data payloads.

3.2 Integration Patterns

- **Publish-Subscribe Channel (Messaging):** Used for the Notification requirement. When a bill is generated or a service activated, the Billing Service publishes an event to the Message Broker (e.g., RabbitMQ). The Notification Service subscribes to this channel. This ensures the "best effort" delivery and prevents the main user flow from blocking while waiting for email/SMS sending.
- **Adapter Pattern:** The solution includes a "Provisioning Adapter" (or Anti-Corruption Layer) that translates internal REST domain calls into the specific format required by the external Telco Provisioning System.

4 Information Security Considerations

- **Authentication & Authorization (OAuth 2.0 / OIDC):** Using an Identity Provider (IdP) like Keycloak. The Mobile App and Web Portal authenticate against the IdP and receive an Access Token (JWT). The API Gateway validates this token before forwarding requests.
- **Transport Security (TLS/HTTPS):** All communication between clients and the Gateway, and between internal services, is encrypted.
- **Network Isolation:** The Microservices and Databases reside in a private subnet, not directly accessible from the internet. Only the API Gateway and Load Balancer are exposed publicly.
- **Secure Configuration:** Secrets (DB passwords, API keys for Payment Gateway) are managed via a Vault or Config Server, not hardcoded.

5 Chat Module Architecture

Although the chat implementation is optional, the architecture supports it via **WebSockets**.

- The Chat Service is a dedicated microservice keeping persistent WebSocket connections with the User App and the Agent Console.
- It uses a lightweight, event-driven runtime (e.g., Node.js or Go) to handle thousands of concurrent open connections.
- Redis is used as a Pub/Sub mechanism for syncing messages if the Chat Service is scaled horizontally across multiple instances.

6 Conclusion

The proposed Microservices solution effectively addresses Sri Tel's needs for a modern, scalable customer care platform. By leveraging the API Gateway for unified access and an Event-Driven architecture for high-volume notifications, the system ensures robustness and a superior user experience.

References

[1] MA-CR-01-Intro-Distributed-Systems.pdf, *Introduction to Distributed Systems Course Notes.*

[2] MA-CR-02-Intro-Middleware.pdf, *Introduction to Middleware Course Notes.*

[3] MA-CR-03-Architectural-Patterns.pdf, *Architectural Patterns Course Notes.*

[4] MA-CR-06-ESB-Integration-Patterns.pdf, *ESB Integration Patterns Course Notes.*

[5] MA-CR-07-MicroServices-1.pdf, *Microservices Part 1 Course Notes.*

[6] MA-CR-08-MicroServices-2.pdf, *Microservices Part 2 Course Notes.*