SCS3203 Middleware Architecture

University of Colombo School of Computing

Sri-Tel Ltd (STL)

Solution Documentation

Group — NoWhere

Name	Index Number
WIJEGUNAWARDANE I.A.P.P.	20002157
WICKRAMASINGHE W.M.O.E.	20002149
WALGAMA R.R.	20001959
PERERA M.A.I.A.	20001304
DE SILVA T.P.M.	20000375
HARISCHANDRA L.I.L.	20000715
SAMALIARACHCHI H.U.	20001551

November 5, 2023

Contents

List of Figures

1 Introduction to the Solution

Sri Tel Ltd (STL), a Sri Lankan-based telecommunication company, aims to enhance its market share by focusing on improving customer care and customer experience. As part of this initiative, STL plans to introduce a state-of-the-art Internet-based Customer Care Web Portal which is implemented using a React web app that can be scalable with platform-independent solutions such as Android and iOS with provided microservices-based APIs. The solution empowers customers to configure and pay for their services independently through a payment gateway. Key functionalities include account creation, bill payments, service activations, bill viewing, notifications, and online chat support. This report contains a comprehensive analysis of the architecture of this requirement and provides the implementation of a Spring Boot application with a scalable architecture.

2 Architecture of the Solution

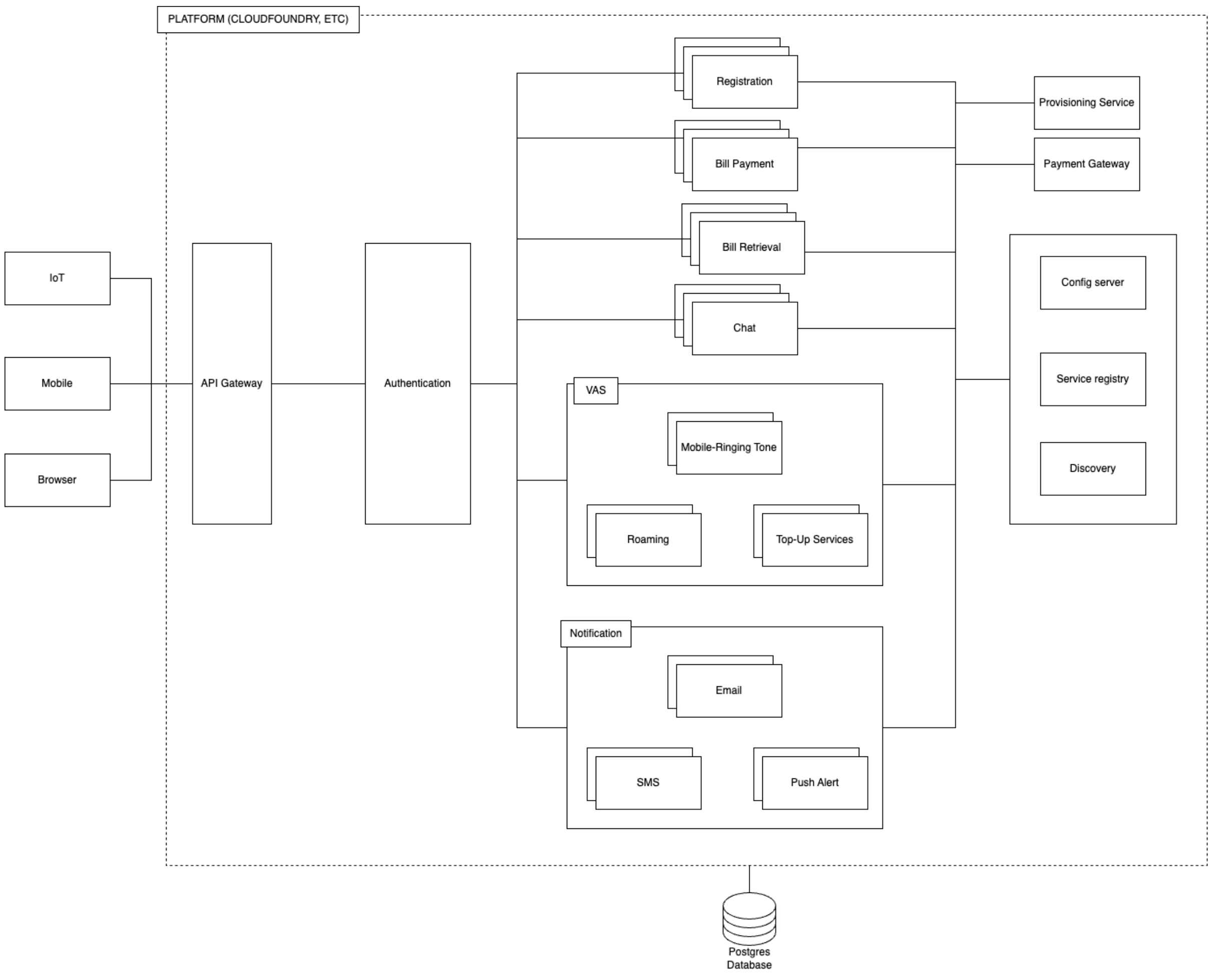


Figure 1: Proposed Microservices Architecture.

The Sri-Tel Ltd solution encompasses the following components:

- **User Management**: This component is responsible for user account creation, and email-password-based login. It should be simple and secure, allowing users to create accounts without manual assistance.
- **Billing and Payment**: Manages bill generation, bill viewing, and online payment processing using credit and debit cards.
- **Service Activation/Deactivation**: Interfaces with the existing Provisioning System to activate or deactivate Telco services, including Value Added Services (VAS), voice, and data services.
- **Notification Engine**: Delivers email, SMS, and push alerts to customers based on bill generation, service issues, disconnections, and other relevant events. It should not hinder the primary functions of the solution.
- **Online Chat Support**: Facilitates real-time chat between customers and customer care agents for issue resolution.

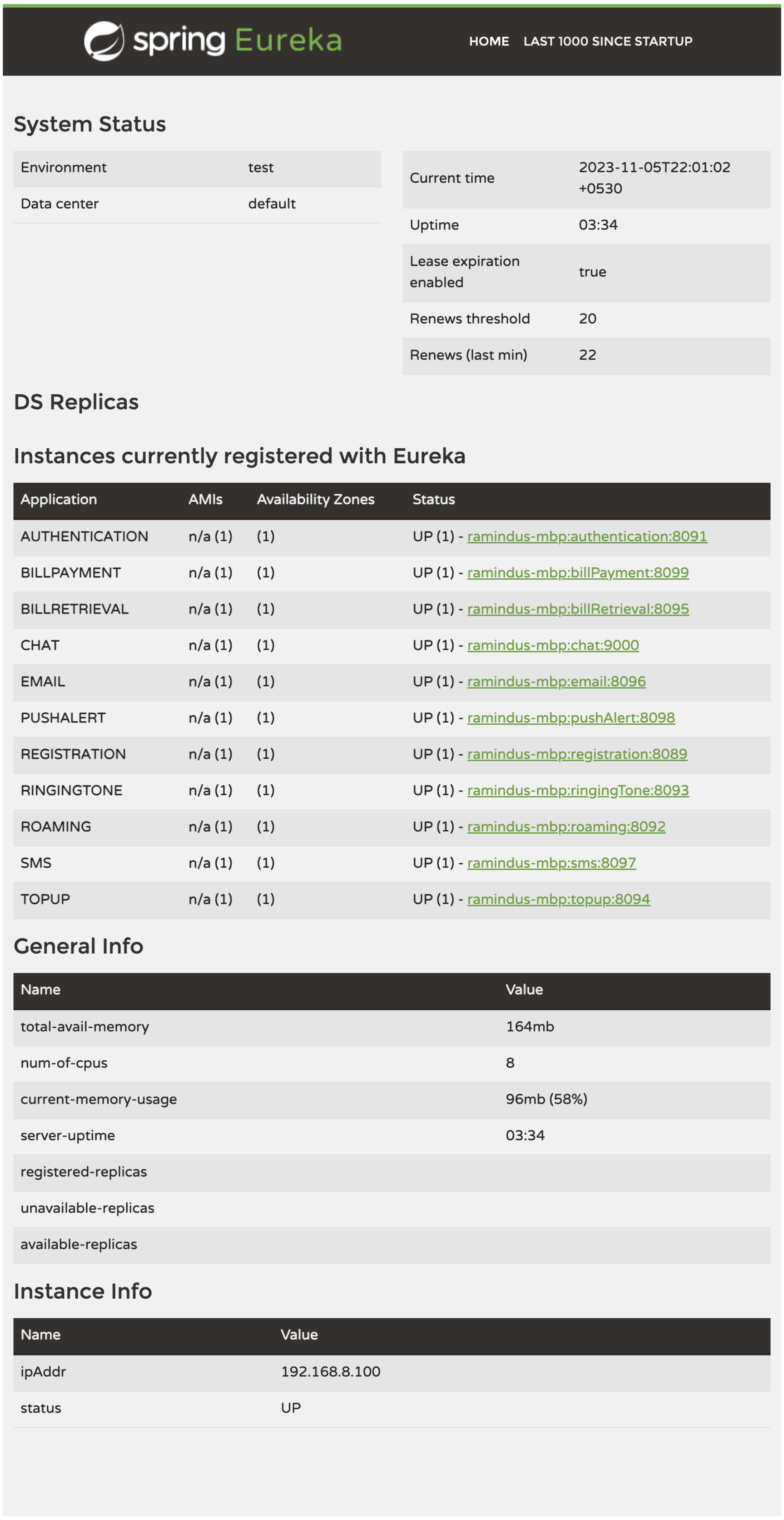
The proposed solution architecture is based on the Eureka server implemented using Spring Boot. In a Spring Boot microservices architecture, an API Gateway acts as a single entry point for all client requests, routing them to the appropriate microservice. Users have to first login in which is implemented as a service.

The proposed system consists of a configuration server and a service registry to look up and register services which are connected to all the available services. Hence it should not be a single point of failure; should be highly available. The service discovery is responsible for discovering new services and handling them.

The VAS is implemented as a Mobile Ringing Tone with Roaming features and Top-up services where users can add them as they wish using the web interface. Notifications service is described as Email, SMS, and Push Alert notifications. This will be communicated to the user through the application where the user can select each from the dashboard.

The provided API is configurable with a mobile, IoT, or any other application since the provided implementation is done using REST-full API. The implementation has been done only on the web for demonstration easiness.

Postgres database is used as the database with a single database to mimic the data source and it is running on the default port 5432. Below is shown the Eureka server with all the service statuses when all the services are running.



System Status

Environment	test
Data center	default

Current time	2023-11-05T22:01:02 +0530
Uptime	03:34
Lease expiration enabled	true
Renews threshold	20
Renews (last min)	22

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AUTHENTICATION	n/a (1)	(1)	UP (1) - ramindus-mbp:authentication:8091
BILLPAYMENT	n/a (1)	(1)	UP (1) - ramindus-mbp:billPayment:8099
BILLRETRIEVAL	n/a (1)	(1)	UP (1) - ramindus-mbp:billRetrieval:8095
CHAT	n/a (1)	(1)	UP (1) - ramindus-mbp:chat:9000
EMAIL	n/a (1)	(1)	UP (1) - ramindus-mbp:email:8096
PUSHALERT	n/a (1)	(1)	UP (1) - ramindus-mbp:pushAlert:8098
REGISTRATION	n/a (1)	(1)	UP (1) - ramindus-mbp:registration:8089
RINGINGTONE	n/a (1)	(1)	UP (1) - ramindus-mbp:ringingTone:8093
ROAMING	n/a (1)	(1)	UP (1) - ramindus-mbp:roaming:8092
SMS	n/a (1)	(1)	UP (1) - ramindus-mbp:sms:8097
TOPUP	n/a (1)	(1)	UP (1) - ramindus-mbp:topup:8094

General Info

Name	Value
total-avail-memory	164mb
num-of-cpus	8
current-memory-usage	96mb (58%)
server-uptime	03:34
registered-replicas	
unavailable-replicas	
available-replicas	

Instance Info

Name	Value
ipAddr	192.168.8.100
status	UP

Figure 2: Eureka Server with all the services running.

3 Implementation Architectures

The following are two alternative architectures for implementing the solution:

3.1 Alternative Architecture 1: Monolithic Architecture

In this alternative architecture, the Sri-Tel solution is implemented as a single, tightly integrated monolithic application. All functionalities, including user management, service management, billing, chat support, and external integrations, are bundled into a single codebase and deployed as one application.

Monolithic architectures are easier to develop and maintain, as there's no need to manage multiple microservices. In some cases, a monolithic architecture may result in lower latency as there are no network calls between microservices.

But, Monolithic applications can be challenging to scale horizontally because all components are tightly coupled and as the application grows, maintaining and updating the monolith becomes more complex. Also Changes in one part of the application may affect other parts, making it less modular. Because of these drawbacks the micro services architecture was selected for the implementation.

3.2 Alternative Architecture 2: Service-Oriented Architecture (SOA)

In the SOA architecture, the Sri-Tel solution is designed as a collection of loosely coupled, independent services. Each service focuses on a specific set of functionalities, such as user management, billing, chat support, and external integrations. These services communicate with each other through standardized interfaces, such as APIs or message queues.

SOA allows for loose coupling between services, making it easier to update and maintain individual components without impacting the entire system. Also The services can be reused across different applications or platforms, improving efficiency and reducing development time. SOA supports horizontal scalability as services can be deployed independently based on demand.

But in SOA architecture, managing a large number of services and their interactions can become complex and Depending on the implementation, there might be some latency in service-to-service communication.Also SOA requires strong governance to ensure consistency and compatibility among services. As a result of these drawbacks, the microservices architecture was used for the implementation.

4 Architectural Patterns and Integration Patterns

The Sri Tel solution is based on a Microservices Architecture implemented using Spring-Boot. The Microservice Architecture divides the application into smaller, loosely coupled services that can be developed, deployed, and maintained independently as the development team of this project follows. It aligns with the requirement to provide distinct functionalities to Sri-Tel users while maintaining agility and scalability as mentioned in the previous section.

4.1 Integration Patterns

- **Message-Oriented Integration** This pattern is foundational to the scenario as it involves using messages for communication and integration between different parts of the system. Messages are crucial for sending notifications, alerts, and communication between components.
- **Request-Reply Pattern** This pattern is applied in cases where synchronous interactions are necessary, in requesting bill information and verifying payment status with the Payment Gateway.
- **Multiple Consumers** This pattern may apply when multiple consumers (customers) receive and process messages, alerts, or chat messages.
- **Return Address** Return addresses are used to specify where response messages should be sent in request-reply interactions, ensuring that responses are routed back to the correct recipients.
- **Multiple Service Providers** In the context of the Sri-Tel solution, Provisioning System and Payment Gateway, are involved in providing services and integrating with the system.
- **Content-Based Router** This pattern is employed to route messages and notifications based on their content or attributes, ensuring that the right messages reach the right recipients.

4.2 Information Security Considerations

During the design and implementation of the solution, various information security considerations have been addressed.

- Authentication and Authorization
- Password Management
- Payment Security through Payment
- Notifications are only accessed through user accounts
- Microservices are independent

5 Source Code

GitHub: Sritel-Microservice