FACULTY OF ENGINEERING AND COMPUTING

School of Computing

MSc. DEGREE

IN

Software Engineering

Name: Chathura Samarasinghe

ID Number: K2166813

Project Title: Business Support System Framework For Provisioning Of Telecom Value Added Services

Supervisor: Prof. Ruvan Abeysekara

Date: 2022-September-05

Abstract

Value-added services (VAS) stand for non-core telecommunication services offered by telecom companies to their subscribers. These services usually do not align with the main services provided by telecom such as voice calls. VASs are a collection of advanced optional services that are designed for promoting telecom providers' primary business. VASs can be intelligently used to increase telecom providers' revenue by providing the best services to the subscribers and keeping them busy on their mobile devices.

For the telecom operators, it is critical to maintaining high availability of the VAS software system. Also, it is crucial to serve every VAS request of subscribers without getting blocked and at a minimum response time. Currently, the in-use software system of the client telecom company does not have the capacity to serve every request and keep up with high loads. It often goes offline due to technical issues of this legacy software system. Also, it takes a considerable amount of time to create and deploy new VASs to the subscribers.

This product focuses on a new approach to replace the current monolithic software system. The proposed solution introduces a micro-services based distributed software platform deployed on Kubernetes network. A platform with all features complemented towards telecom provider's effort to stay ahead of competitors with rapid value-added service design, creation, test, and deployment.

Table of Content

Glossary of Terms

Microservice architecture	A sub variant of service oriented structural architecture type
Message Queue	A message queue is a queue of messages passes between services
SMS Gateway	This enables software to send and receive SMS messages
WAP	Wireless Application Protocol is a technology that allows your mobile phone to browse the Web. It's a protocol for the transmission of data over low bandwidth wireless networks. [1]
OCS	Online Charging System
PCRF	Policy and Charging Rules Function
API	Application Programming Interface
VAS	Value Added Service
Server node	A physical server computer acting as a single unit of a server cluster
Subscriber	Telecom users
MSISDN	Identifier of subscribers' mobile number
JWT	JSON Web Token
BMS	Bare Metal Server

1. Introduction & Background

1.1 Introduction

In today's competitive world, it is critical for telecom providers to get into the market as soon as possible and capitalize their subscribers by providing valuable services in addition to their core telecom services such as voice calls, internet access, and text/multimedia messaging. One strategy to provide such services to the subscribers is the use of Value Added Services (VAS).

These services are provided either for free or for a limited subscription and they fall under a wide variety of service categories, namely, e-commerce, gaming, news, music, TV, and more…

VAS is often provided as short-term and long-term sustainable services. Creating VASs depends on multiple factors such as market needs and competitor activities.

VAS is used by telco service providers to;

- Increase profit by selling add-on services.
- Attract more customers.
- Create comfortable service packages and provide more control to the customer over the services [2].

Currently, the client of this proposed system uses a traditional monolithic software platform to manage these VASs.

1.2 Aim and Objectives

1.2.1 Aim

Develop a BSS (Business Support System) framework to manage and control Value Added Services.

1.2.2 Objectives

1. To provide a formal framework for VAS request handling.
2. To provide a web application to VAS provisioning.
3. To carry out research on technologies available under JavaEE/Spring ecosystem and low-code technologies.
4. To build a graphical low-code workflow builder.
5. To build a processing engine to execute workflows.
6. To provide a dynamic REST API deployment platform.

1.3 Background and Motivation

The process of VAS request processing is; customer sends an SMS or a USSD query containing command keywords requesting a specific service, then an SMSC gateway will receive this message and forward it to the VAS system. The VAS system will then do all the processing by going through related services and sending an SMS as a response back to SMSC and then to the subscriber number.

In order to serve customer requests of VAS, the VAS management platform interacts with a number of in-house built software systems (APIs, web services), payment gateways, and vendor-specific platforms such as Huawei OCS for charging, Huawei PCRF for rule-based charging, etc… The process of serving VAS requests is typically complicated due to the nature of internal system infrastructure, monolithic architectures, and the practices being followed by the provider when it comes to software development. This is common to every telco provider in the market.

There are key challenges that exist in the current business environment.

- High time to market.
- Creation and deployment take time to wrap up.
- By the time changes are deployed, requirements have expired in the market.
- Difficulty to keep up with the new technology stack due to customizations of core services.
- No standardization leads to an integration mess

A modern software solution is needed to mitigate those key challenges of the business.

1.4 Problem in brief

Due to the complexities that were pointed out in section 1.2, VAS management platforms suffer from a number of issues, namely, maintenance and operations, monitoring, downtimes, request blockings, etc… Usual approach to build and provision services is writing new code and integrating it with the available system after series of critical user verification. This involves multiple stakeholders such as business users, software developers, and testers. Developers write the code according to the software requirements specification. This development involves core service integrations, data streaming, and database and event generation system integrations along with API integrations through internal API gateway. These integrations are usually done through repeating already written codes in the new codebase as there are no standard best practices followed. This introduces bugs and unprecedented problems in actual environments. When the business requires to make changes to an existing service it would take a few days or weeks to build a patch, test, and deploy it to production. By the time the new changes are made available to the consumers, the market value requirements may have expired.

Another concern is the availability of the services. Current VAS system is a monolithic application and the company has not put backup and recovery strategies in place in-case of disastrous events. This application is highly vulnerable to single point of failures. If the service goes down due to high load or hardware

failures, the maintenance team will manually re-deploy it to the production environment again to serve the customers.

In a competitive market, those issues often lead to negative impacts on business and decreased customer satisfaction. It requires a modern strategy to overcome those concerns and get into the market early as possible.

1.5 Proposed Solution

There are two main sub-system in the proposed solution to overcome the problems mentioned previously.

- **VAS provisioning stack.**
  The proposed solution is built following a hybrid system architecture of microservice and event-driven architectures to ensure the availability and integrity of the system. There are three microservices in this VAS provisioning stack.
- **Low-code REST API builder.**
  Also the proposed system consists of an interactive low-code REST API builder to design, build, test, and deploy REST APIs as underlying logical representations of services.

This will completely replace the software system which is currently being used and provide a much more simplified user interface to design, develop, test, and deploy VASs to the production so that the subscribers can use them.

The implementation will result in the following essentials.

- Performance Enhancement.
- Technology Upgrade.
- Ability to configure services through a user interface.
- Minimized complexity.

By using this platform, the business can effectively deploy new services, change them when required, and monitor the system without a hassle.

1.6 Product Scope

The scope of this product is to provide the telecom provider a newly implemented Business Support System to provide functions of design, development, testing, and deployment of value added services. This enables the telecom provider to provision VASs easily and change them effectively when the business requirements and the market demand are changed over time. This consists of a web user interface to provide the required functionalities and leverages low-code approach to introduce a low-code graphical workflow API builder. The system will use a messaging service for the inter-service communication and a database system to store the system data and usage statistics.

Primary responsibilities of the system,

- Allow users to create/modify/import/export VASs.
- Allow users to create/modify and map SMS keyword commands.
- Allow users to design/develop/modify/test/deploy dynamic service workflow APIs as REST APIs.
- Allow users to view system usage statistics.
- Provides SMS routing and keyword mapping to appropriate services.
- Allow users to build workflows using a low-code graphical interface.
- Provides a processing engine to execute workflow API definitions and provide outputs.
- Allow users rollback services within the system and migrate between staging and production environments.

Apart from these primary responsibilities and functions, the system has many sub functions and they will be explored in later sections.

2. Requirements Specification

2.1 Product functions

This section specifies a list of functions and features that should be provided by the system.

- User login.
- Shows VASs as a pageable list.
- Shows details of selected VAS and lets users modify them.
- Shows details of actions declared for each VAS and lets users modify and create them.
- Shows details of APIs declared under each action.
- Shows details of keyword regexes and lets users modify and create them.
- Shows details of SMSs configured under each API and lets users modify and create them.
- Shows APIs/workflows as a pageable list.
- Show and modify details of each API.
- Show API change history.
- Show API usage statistics.
- Deploy and modify selected API versions.
- Design and develop new APIs and foked APIs.
- Build and save developed API
- Let users create new VAS, APIs, and import/export services.
- View SMS history, filter by contact number or transaction ID.
- SMS command/keyword set mapping.
- VAS discovery.
- API execution and deployment.
- View recent actions log.
- Show infrastructure/platform service status.
- Show inbound, outgoing SMS statistics.

2.2 Operating environment

There are two main environments; staging and production. The platform deployed in the staging environment is to design and build VASs and perform testing, quality assurance, and UATs of the service to be released. Once all the actions are completed and the business decides to go live with the service, it should be exported/migrated into the platform that exists in the production environment.

Both these server infrastructures are Kubernetes server clusters deployed in on-premise BMSs (Bare Metal Servers). Each service will be containerized with Docker. The Linux CentOS operating system works as the Linux kernel for the Docker application containers and each Docker container is based on an Alpine Linux file system. Alpine Linux is a minimal linux operating system that has all the necessary features to run an application. CentOS is a complete linux operating system which is known for high

stability and recommended for production usage. Company's DevOps team manages these physical environments.

2.3 Design and Implementation Constraints

This section specifies design constraints that should be followed while developing the system, deployment, and maintenance to keep the integrity of the system.

- The platform is to be deployed in an on-premise Kubernates server cluster, so that the service components must be built to support this environment.
- The platform is built as an elastic service, which means when it receives too much load these requests must be handled efficiently with minimum or no losses. If it is necessary services may be scaled up automatically on available hardware.
- All the data communications within services should be encrypted using valid SSL certificates from a trusted authority.
- Libraries, frameworks, and other related third party software components must be regularly scanned for security vulnerabilities and they should be kept updated.

2.4 Assumptions

The system assumes the following mentioned assumptions are satisfied prior to using the system and while using the system in the production and staging environments.

- The users must have an active internet connection and be connected to the internal network through the provided APN.
- Users have installed and use a modern web browser to access the system (preferably using a web browser with webkit engine).
- Maintenance staff have prior knowledge of operating the overall product.

2.5 External Interfaces

2.5.1 Software Interfaces

The system must interact with following software interfaces,

- MySQL 8.0 database cluster to access the data of the system through the port 3306.
- All the core microservices are deployed in Apache Tomcat 9 servlet container servers. Docker containers expose port 443 to http connections and this port is mapped into configurable ports by Docker and expose those ports to outside of the server cluster.
- Front end web UI is deployed in an Nginx 1.21.6 HTTP web server.
- Platform services must communicate with the RabbitMQ messaging service, which is run in port 5672.

2.5.1 Communication Interfaces

This section specifies how data should be transferred and communications protocols.

- Frontend web UI and backend REST API communicates through HTTPS and exchanges JSON messages.
- REST API and the Data Model component is connected with the MySQL database through JDBC protocol of Java.
- Static resources of the web application are served as plaintext, HTML, CSS, Images, and Fonts by the web server through HTTPS.
- Core microservices and RabbitMQ messaging service communicates through AMQP and exchanges JSON messages.
- Communication between SMSC and the platform is done by HTTP and JSON messages.

2.6 Functional Requirements

This section lists down all the functional requirements that are critical to the system, each requirement is uniquely identified by a requirement ID.

- Priority
  This specifies the order in which requirements should be implemented. Possible values are HIGH, MEDIUM, LOW.
- Risk
  This specifies the risk of not developing the requirement. Possible values are CRITICAL, HIGH.
    - HIGH - not implementing this requirement may break some functions, but the other parts of the system remain intact.
    - CRITICAL - not implementing this requirement will break the system.

User login

Use case name	User login
Requirement ID	REQ-01
Description	Users must be logged in to the system in order to continue.
Priority	High
Risk	C
Basic flow	• User indicates that they want to login. • The system asks for username and password. • The user provides username and password. • The system validates the credentials. • The system generates an access token (JWT) with user details and.

	• The system redirects to the home page.
Alternate flows	if the credentials do not match, goto step 2 and show an error.
Preconditions	The user is registered
Postconditions	The user can browse the system for further actions.

Table 2.6.1 - requirement user login

View service status

Use case name	Service status
Requirement ID	REQ-02
Description	Users must be able to view status of platform services
Priority	Medium
Risk	H
Basic flow	• User visits the dashboard • The dashboard shows the services status in separate tiles
Alternate flows	
Preconditions	The user is logged in
Postconditions	

Table 2.6.2 - requirement service status

View service usage statistics

Use case name	Service usage statistics
Requirement ID	REQ-03
Description	Users must be able to view usage statistics (inbound SMS count, outgoing SMS count)
Priority	Medium
Risk	H
Basic flow	• User visits the dashboard • The dashboard shows the usage statistics in separate tiles

Alternate flows	
Preconditions	The user is logged in
Postconditions	

Table 2.6.3 - requirement usage statistics

View recent user actions

Use case name	Recent user actions
Requirement ID	REQ-04
Description	Users must be able to view recent action log of other users in the system
Priority	Medium
Risk	H
Basic flow	• User visits the dashboard • The dashboard shows a list of recent actions log limited to 10 entries
Alternate flows	
Preconditions	The user is logged in
Postconditions	

Table 2.6.4 - requirement view recent user actions

View value added services

Use case name	View VASs
Requirement ID	REQ-05
Description	Users must be able to view VASs as tiles and displays the status active or inactive
Priority	High
Risk	C
Basic flow	• User visits the services page • The services page shows services
Alternate flows	

Preconditions	The user is logged in
Postconditions	

Table 2.6.5 - requirement view VASs

View VAS details

Use case name	View VAS details
Requirement ID	REQ-06
Description	Users must be able to view details of selected VAS
Priority	High
Risk	C
Basic flow	• User clicks on a VAS tile • The web application brings the user to the VAS details page • VAS details page shows the service information
Alternate flows	
Preconditions	The user is logged in
Postconditions	

Table 2.6.6 - requirement view VAS details

Modify VAS details

Use case name	Modify VAS details
Requirement ID	REQ-07
Description	Users must be able to modify basic details of selected VAS
Priority	High
Risk	C
Basic flow	• User clicks on edit button • The application shows edit pane along with current details • The user change details and clicks on submit button • The application sends API request to the server • The server responds with operation status • On successful modification, the edit pane automatically closes

	and refreshes the details section.
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The service edit pane is open
Postconditions	Service details section must be refreshed to show the changes

Table 2.6.7 - requirement modify VAS

View service actions

Use case name	Show service actions
Requirement ID	REQ-08
Description	Users must be able view actions under a selected service
Priority	High
Risk	C
Basic flow	• The service details page shows currently available actions of the selected service
Alternate flows	
Preconditions	User is on service details page
Postconditions	

Table 2.6.8 - requirement show service actions

Create actions

Use case name	Create actions
Requirement ID	REQ-09
Description	Users must be able create actions under a selected service
Priority	High
Risk	C
Basic flow	• User clicks on add action button • Application shows add action pane • User fills action details and selects an API • User clicks on submit button

	• Application makes a API request to the server • The server responds with a operation status • On successful status, add action pane closes and refreshes actions panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The add action pane is open
Postconditions	Actions list must be refreshed to show the changes

Table 2.6.9 - requirement create actions

Modify actions

Use case name	Modify actions
Requirement ID	REQ-10
Description	Users must be able modify selected action
Priority	High
Risk	C
Basic flow	• User clicks on modify action button of an action • Application shows modify action pane • User changes action details • User clicks on submit button • Application makes an update API request to the server • The server responds with a operation status • On successful status, modify action pane closes and refreshes actions panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The modify action pane is open
Postconditions	Actions list must be refreshed to show the changes

Table 2.6.10 - requirement modify actions

Delete actions

Use case name	Delete actions
Requirement ID	REQ-11

Description	Users must be able delete selected action
Priority	High
Risk	C
Basic flow	• User clicks on modify action button of an action • Application shows modify action pane • Clicks on delete button • Application makes a delete API request to the server • The server responds with a operation status • On successful status, modify action pane closes and refreshes actions panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The modify action pane is open
Postconditions	Actions list must be refreshed to show the changes

Table 2.6.11 - requirement delete actions

View keywords

Use case name	View actions
Requirement ID	REQ-12
Description	Users must be able view keyword details of a selected action
Priority	High
Risk	C
Basic flow	• User selects an action • Application shows a list keywords defined for the selected action
Alternate flows	If the server response has an error, application shows an error message
Preconditions	An action is selected
Postconditions	Keyword list must be refreshed to show the changes

Table 2.6.12 - requirement view actions

Create keyword set

Use case name	Create keywords set
Requirement ID	REQ-13
Description	Users must be able create keywords set under a selected action
Priority	High
Risk	C
Basic flow	• User clicks on add keywords set button • Application shows add keywords set pane • User fills keywords set details • User clicks on submit button • Application makes a API request to the server • The server responds with a operation status • On successful status, add keywords set pane closes and refreshes keywords set panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	An action is selected
Postconditions	Keywords set list must be refreshed to show the changes

Table 2.6.13 - requirement create keywords set

Modify keywords set

Use case name	Modify keywords set
Requirement ID	REQ-14
Description	Users must be able modify selected keywords set
Priority	High
Risk	C
Basic flow	• User clicks on modify keywords set button of a keywords set • Application shows modify keywords set pane • User changes keywords set details • User clicks on submit button • Application makes an update API request to the server • The server responds with a operation status • On successful status, modify keywords set pane closes and refreshes keywords set panel
Alternate flows	If the server response has an error, application shows an error message

Preconditions	The modify keywords set pane is open
Postconditions	Keywords set list must be refreshed to show the changes

Table 2.6.14 - requirement modify keywords set

Delete keywords set

Use case name	Delete keywords set
Requirement ID	REQ-15
Description	Users must be able delete selected keywords set
Priority	High
Risk	C
Basic flow	• User clicks on modify keywords set button of a keywords set • Application shows modify keywords set pane • User clicks on delete button • Application makes a delete API request to the server • The server responds with a operation status • On successful status, modify keywords set pane closes and refreshes keywords set panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The modify keywords set pane is open
Postconditions	Keywords set list must be refreshed to show the changes

Table 2.6.15 - requirement delete keywords set

View service SMSs

Use case name	View service SMSs
Requirement ID	REQ-16
Description	Users must be able view service SMSs details of a selected action
Priority	High
Risk	C
Basic flow	• User selects an action • Application shows a list of service SMSs defined for the selected action
Alternate flows	If the server response has an error, application shows an error message
Preconditions	An action is selected
Postconditions	Service SMSs list must be refreshed to show the changes

Table 2.6.16 - requirement view service SMSs

Create service SMSs

Use case name	Create service SMSs
Requirement ID	REQ-17
Description	Users must be able create service SMSs set under a selected action
Priority	High
Risk	C
Basic flow	• User clicks on add service SMSs set button • Application shows add service SMSs pane • User fills service SMSs details • User clicks on submit button • Application makes a API request to the server • The server responds with a operation status • On successful status, add service SMSs pane closes and refreshes service SMSs panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	An action is selected

Postconditions	Service SMSs list must be refreshed to show the changes

Table 2.6.17 - requirement reate service SMSs

Modify service SMSs

Use case name	Modify service SMSs
Requirement ID	REQ-18
Description	Users must be able modify selected service SMSs
Priority	High
Risk	C
Basic flow	• User clicks on modify service SMSs button of a service SMS • Application shows modify service SMSs pane • User changes service SMSs details • User clicks on submit button • Application makes an update API request to the server • The server responds with a operation status • On successful status, modify service SMSs pane closes and refreshes service SMSs panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The modify service SMSs pane is open
Postconditions	Service SMSs list must be refreshed to show the changes

Table 2.6.18 - requirement modify service SMSs

Delete service SMSs

Use case name	Delete service SMSs
Requirement ID	REQ-19
Description	Users must be able delete selected service SMSs
Priority	High
Risk	C
Basic flow	• User clicks on modify service SMSs button of a service SMS • Application shows modify service SMSs pane • User clicks on delete button

	• Application makes a delete API request to the server • The server responds with a operation status • On successful status, modify service SMSs pane closes and refreshes service SMSs panel
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The modify service SMSs pane is open
Postconditions	Service SMSs set list must be refreshed to show the changes

Table 2.6.19 - requirement delete service SMSs

Export VAS

Use case name	Export VAS
Requirement ID	REQ-20
Description	Users must be able to export selected VAS
Priority	High
Risk	C
Basic flow	• User click on export button • Application makes an API request to the server • The server generates an export data file along with all the service details as an xml file • The server responds with the generated file • Application downloads the file
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The user is in service page
Postconditions	

Table 2.6.20 - requirement export VAS

View API workflows

Use case name	View API workflows
Requirement ID	REQ-21
Description	Users must be able to view created API workflows
Priority	High
Risk	C
Basic flow	• User navigates to API workflows page • The applications shows a list of API workflows
Alternate flows	
Preconditions	The user is logged in
Postconditions	

Table 2.6.21 - requirement view API workflows

View API workflow details

Use case name	View API workflow details
Requirement ID	REQ-22
Description	Users must be able to view details of an a selected API workflow
Priority	High
Risk	C
Basic flow	• User clicks on an API workflow • The applications redirects the page to API workflow details page • The application shows details of the selected API workflow
Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.22 - requirement view API workflow details

Modify API workflow details

Use case name	Modify VAS details
Requirement ID	REQ-23
Description	Users must be able to modify basic details of selected API workflow
Priority	High
Risk	C
Basic flow	• User clicks on edit button • The application shows edit pane along with current details • The user change details and clicks on submit button • The application sends API request to the server • The server responds with operation status • On successful modification, the edit pane automatically closes and refreshes the details section.
Alternate flows	If the server response has an error, application shows an error message
Preconditions	The API workflow edit pane is open
Postconditions	API workflow details section must be refreshed to show the changes

Table 2.6.23 - requirement modify VAS details

View API workflow change history

Use case name	View API workflow change history details
Requirement ID	REQ-24
Description	Users must be able to view change history details of an a selected API workflow
Priority	High
Risk	C
Basic flow	• User navigates to API workflow details page • Application shows a list of API workflow change history
Alternate flows	
Preconditions	The user selected an API workflow

Postconditions	

Table 2.6.24 - requirement view API workflow change history details

View API insights

Use case name	View API insights
Requirement ID	REQ-25
Description	Users must be able to view API usage insights
Priority	High
Risk	C
Basic flow	• User navigates to API workflow details page • Application shows API insights as average response time, requests count, and error rate
Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.25 - requirement view API insights

Deploy API

Use case name	Deploy API
Requirement ID	REQ-26
Description	Users must be able to deploy APIs
Priority	High
Risk	C
Basic flow	• User selects an API • The application shows a popup with API actions • User selects deploy option • The application marks the selected API version as the active API version • The application refreshes the page to set the default API (deployed API)

Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.26 - requirement deploy API

Checkout API

Use case name	Checkout API
Requirement ID	REQ-27
Description	Users must be able to checkout APIs to edit them
Priority	High
Risk	C
Basic flow	• User selects an API • The application shows a popup with API actions • User selects checkout option • The application marks the selected API version as the default API version in the page • The application refreshes the page to set the default API
Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.27 - requirement checkout API

Design API

Use case name	Design API
Requirement ID	REQ-28
Description	Users must be able to design APIs
Priority	High
Risk	C
Basic flow	• User selects an API • User navigates to Design tab of API page

	• The application shows initial graph with available nodes
Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.28 - requirement design API

Add nodes to the API

Use case name	Add nodes to the API
Requirement ID	REQ-29
Description	Users must be able to Add nodes to the API design
Priority	High
Risk	C
Basic flow	• User drags and drops API nodes on to the graph to design the workflow • The graph shows dropped nodes with the labels and edges
Alternate flows	
Preconditions	The user selected an API workflow
Postconditions	

Table 2.6.29 - requirement add nodes to the API

Edit nodes in the graph

Use case name	Edit nodes in the graph
Requirement ID	REQ-30
Description	Users must be able to edit nodes in the graph
Priority	High
Risk	C
Basic flow	• User double click on the node to be edited • Application shows a pane with current node details • User makes changes to the node details

	• User saves the changes by clicking on the submit button
Alternate flows	
Preconditions	The user selected a node
Postconditions	

Table 2.6.30 - requirement dit nodes in the graph

Save graph as API

Use case name	Save graph as API
Requirement ID	REQ-31
Description	Users must be able to Save graph as API
Priority	High
Risk	C
Basic flow	• User clicks on the save button in the graph • The applications asks for an API version and the change message • User provides these information and clicks on submit button • Application saves the API as a new version
Alternate flows	
Preconditions	The user created/modified a graph
Postconditions	

Table 2.6.31 - requirement save graph as API

Import service

Use case name	Import services
Requirement ID	REQ-32
Description	Users must be able to import already created service
Priority	High
Risk	C
Basic flow	• User clicks on import service tile

	• Application asks for the file to be uploaded • User selects a previously generated xml file • Application reads the file and reverse build the service details and saves them in the database
Alternate flows	
Preconditions	The user created/modified a graph
Postconditions	

Table 2.6.32 - requirement import services

View SMS history

Use case name	View SMS history
Requirement ID	REQ-33
Description	Users must be able to view SMS history
Priority	High
Risk	C
Basic flow	• User navigates to the SMS history page • User provides the contact number to check the SMS history • Application shows a history of SMSs
Alternate flows	
Preconditions	The user is on view SMS history page
Postconditions	

Table 2.6.33 - requirement View SMS history

2.7 Non-Functional Requirements

2.7.1 Performance requirements

- The system should be able to process 200 concurrent SMSs.
- The system should be effectively able to handle 3 concurrent users.
- The reasonable time to display the main webpage over a WIFI internet connection should not exceed 4 seconds.
- The average response time of the management API should not exceed 300 milliseconds.

2.7.2 Safety Requirements

- The system should only be accessed through an internal APN.
- The database should be backed up once an hour.

3. Technology Review

This section focuses on carrying out research on currently available production grade Java based technologies, tools, libraries, and frameworks that could possibly be used in the software development process of this project. Choosing correct tools with appropriate justification is critical to a successful implementation as it increases productivity, efficiency, and quality of the product.

3.1 Technology Stack

For the development of the proposed system, Java is used as the language to write the code. Java is chosen as the staff of the client company is confident with the language and it will be easy for them to use the system and carry out maintenance tasks. Java has a wide variety of supportive libraries and most of the modern frameworks are based on Java. in particular, Java 1.8 has been used to develop the platform services.

3.1.1 Version Controlling

Version controlling is the practice that a special system records changes to a file or multiple files in the process of software development (Git-scm.com, 2019). This way developers can recall specific versions or releases of the source code at any given time in the future and track the progress. Github is used as the version control system of this product. There are multiple version control systems and vendors are available. Among them, GIT is chosen as the version control system and Gitlab is chosen as the vendor platform to host the code. Gitlab is open-source code hosting service, any organization can securely implement an instance of GItlab in a server environment of their choice and use it to host the source code.

3.1.2 Quality and Testing

As per ANSI/IEEE 1059, software testing is the process of verifying a software to find whether the current product meets the required conditions or not. (Hamilton, 2019). There are numerous software testing techniques and tools are available for use. Unit testing, integration testing, stress testing, blackbox testing, and alpha/beta testing are used to evaluate this product.

JUnit is used as the main testing library along with MockMVC. JUnit is a unit testing library and it can be used to test each part of the code separately. MockMVC is used as an integration testing tool to test end to end APIs and endpoints.

JMeter is employed to test the performance of the application by simulating a load higher than the non-functional requirements state. This enables developers to check, understand, and implement the system to withstand large volumes of traffic.
Snyk online service and IDE tool is used to check the code is up to date with latest security updates. Blackbox and alpha/beta testing will be conducted upon the initial staging deployment with real users to make sure the product behaves correctly in the real environment.

SonarQube works as the static code analyser to evaluate source code level quality and get insights of the code and implement enhanced codes. This provides metrics to understand how well the source code is written. SonarQube detects security loopholes, code testing coverage, and code optimizations. Adhering to these quality metrics will result in a quality product.

3.1.3 Build and Library Management

Building is the process of transforming the source code into a runnable/deployable single entity of the program. In this product, Apache Maven is used as the build configuration management and runtime, compile time, and test library management. Maven comes with extensive features to automate internal dependency management and build. Maven uses a central dependency repository to provide mapped versions of dependencies and their inner dependencies. Also Maven is integrated with many IDEs to increase productivity of the development process.

3.1.4 Logging

Logging is the process of maintaining a plaintext file to write information about a transaction which was carried out by the application for the purpose of debugging and auditing. Summarized details of everything that happens in the system should go into the log files so that later they can be explored to understand events in case of errors, failures, security incidents, or audits.

Latest version of Log4j2 library is integrated with each service to get functionality of logging as this library is a production-ready utility. There are a lot of features provided by this such as log file rolling, triggering policies, rollover strategies, logging pattern customization, multiple log files management, etc…

3.2 Spring Framework

Spring framework is used as the underlying web MVC framework to develop the product on top of. Spring is a feature-rich web application development framework developed to enhance capabilities of JavaEE. Spring Boot is built on top of Spring MVC conventional framework which is optimized at the configuration level to build and deliver REST APIs and Microservices effectively. Spring ecosystem provides an extensive library and module set that enables writing scalable web/microservices. According to the framework specification, it provides below functionalities and features that are related to the development of the product,

- Microservices

Support for building production-grade, independently evolvable microservices.

- Reactivity
  Provides asynchronous, non-blocking architecture to write applications to get more from computing resources.
- Cloud support
  Integrate and scale Spring applications with any cloud provider.
- Web apps
  Libraries to build fast, secure, responsive web applications.
- Event driven
  React to business events or data streams.
- Batch processing
  Automated and offline processing of large amounts of data.
  (Spring.io, 2019)

These high level features are made possible by libraries and modules that come with the framework. These modules can be independently integrated with any Spring application with a library/build management tool of choice (in this case Apache Maven). Following Spring modules are explored more as they will be used to develop the product.

- Spring Web
  Spring web provides the core of the application as it deals with the frameworks' internal IoC (Inversion of Control) as dependency injection and interaction with JavaEE features. This enables writing an overall Spring web application to handle web requests and process them.
- Spring Data JPA
  This is to add data access capabilities to the system. Data JPA leverages Java persistence and Hibernate as the ORM framework. This also provides built-in ready to use interfaces to access databases seamlessly. Data JPA internally uses HikariCP for the database connection pooling. HikariCP is a high-performance connection pooling library to access databases through pooled connections (Wooldridge, 2022). Data JPA also provides a powerful query building, result set transformation mechanisms to interact with the data.
- Spring Actuator
  Every application should be monitored to keep in touch with the performance and usage. Spring Actuator is a module that provides monitoring capabilities to read telemetry data of the application. Internally it uses Micrometer to listen, collect and distribute these data to external monitoring services such as Prometheus and Grafana.
- Spring test
  This module provides built-in support to JUnit testing and MockMVC testing utilities to write and execute test cases, as well as to get test reports.
- Spring cloud streaming

3.3 Messaging and Service Communication

Services must handle user requests from the application. Additionally, services frequently work together to handle such requests. For this to work, they must employ an inter-process communication mechanism. Synchronous communication might result in tight runtime coupling, both client and service must be available in the lifecycle of the request. Since this platform has a few microservices working together and they are supposed to handle large volumes of requests load, there has to be a way to efficiently handle all these requests and provide responses with a minimum latency in an asynchronous way.

The solution is to use an asynchronous messaging system for inter-service communication. Services in the platform will communicate by exchanging messages over couple message channels.

There are few asynchronous messaging patterns can be identified,

- Request/response
  A service sends a request to another service and expects a response back from the second service promptly.
- Notifications
  A service sends a request to a second service but does not expect a response.
- Request/async-response
  A service sends a message to a recipient and expects a response eventually.
- Publish/subscribe
  A service (publisher) sends messages to zero or more recipients.
- Publish/async-response
  A service (publisher) sends messages to one or more recipients and some of the recipients replies back. (microservices.io, n.d.)

Core microservices of the system communicate through messaging services and services work as publishers and subscribers. RabbitMQ is used as the message broker to publish and keep messages. Each service works as both consumer and producer. JSON messages are used as standard messaging format. RabbitMQ is one of the most used message brokers available. This is lightweight and easy to deploy on premise and in the cloud (Rabbitmq.com, 2019).

Resulting integration has a number of benefits and a few drawbacks.

- This decouples the sender (publisher) and receiver (consumer)
- Availability is improved as all the messages are buffered until a consumer service processes them.
- Supports previously mentioned communication patterns.

As a drawback, this adds more complexity to the system as the services are highly dependent on the messages from the messaging service, it must be available all the time.

4. System Design

4.1 Introduction

This section describes all the developed software artifacts/components of the system in detail to provide an overall idea of their usage and how they work together to achieve the aim and goals of this project. This section also focuses on a discussion of how the specified software architectural styles are adapted to formulate the system.

4.2 Architectural Overview

4.2.1 System Architecture

This section discusses the system architecture of the project and how standard architectural styles are adapted as well as a justification on why chosen architecture is the best fit for the implementation of the project.

Currently there are a lot of architectural styles available for development of software systems. A well defined standard software architecture is needed to build a scalable software. Following points are to be considered in order to formulate a scalable system architecture for the project.

- The system must be easy to understand and modify.
- Continuous deployment.
- Handle large amounts of data.
- Run multiple instances in order to satisfy scalability and availability.
- Take advantage of emerging technologies (frameworks and languages)

There are four major aspects of this proposed system,

- SMS routing
- Service execution
- Response handling
- Service provisioning

Service execution part should have the highest processing power than the other two services as it processes the logical workflows defined in service definitions. So in a monolithic architecture this can be achieved through threads. A thread shares the same memory as its parent process. Even if the application used an executor thread pool to keep some amount of reusable threads, this requires more hardware power to process requests. This approach is also vulnerable to a single point of failure as one system handles all the functionalities.

To overcome mentioned problems is it a good practice to adapt a distributed system architecture to build the application. Use of distributed architecture helps to modularize the application into smaller services and make them work as independent service components. Each service can communicate using a data communication protocol such as HTTP. For this to work each service must expose REST API endpoints. Considering mentioned points, this approach is the suitable architectural style for the development of this application. There are a few distributed software architectures available.

Next concern is service communication. As mentioned before each service needs to expose REST API endpoints for the communication. By its nature, this system needs to handle a large volume of service requests. If there are lags between service calls, the application servers will drop requests and cause request errors in the system. When a service is busy processing requests other consumer and producer services will have to wait or drop requests until the fed up service becomes available. It is critical for the business to serve every request without any technical failures as much. To overcome this data communication issue it is a good practice to adapt and event driven architecture. Event driven processing involves services to work on availability of data.

By adapting a distributed architecture, it is easy and effective to modify and upgrade each service component separately in the system with minimum effort. This enables continuous integration and deployment to effectively be applied.

As for the scaling of the system, each service can be separately scaled in horizontally and vertically. “Horizontal scaling (aka scaling out) refers to adding additional nodes or machines to the infrastructure to cope with new demands. When an application runs on a server and finds that it no longer has the capacity or capabilities to handle traffic, adding a server may be the solution” (CloudZero, n.d.). Since the service components are to be deployed in a Kibernates server cluster, it is possible to distribute each service component in different physical server nodes. Handling of traffic is governed by the master node of Kubernates.

“Vertical scaling (aka scaling up) describes adding additional resources to a system so that it meets demand” (CloudZero, n.d.). It is also viable to add more hardware resources to each server node to increase traffic handling and processing capacity of the server.

Considering above mentioned issues and possible solutions to overcome them, it is identified to leverage a hybrid distributed architecture of microservice and event-driven architectures.

The resulting architecture has following advantages and strengths,

- Highly maintainable and testable.
- Loosely coupled.
- Can be deployed independently.
- Each service is relatively small, easier to understand and modify in the future.
- Scalability is higher.
- Large volumes of data can be effectively processed.

Following diagram presents a high level system design and service distribution of the proposed solution.

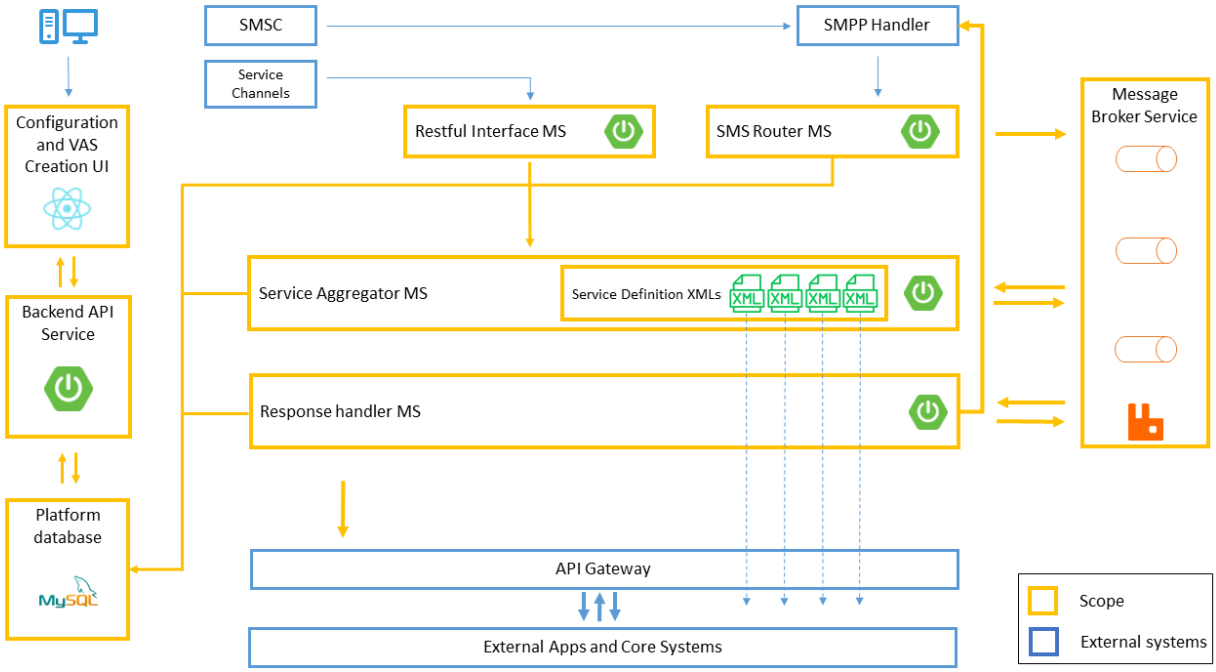


Diagram 4.2.1.1 - High level system architecture

Each microservice is built according to MVC architectural specification. Spring framework enforces the use of MVC to form a web service. Here, all the incoming requests (both message queue feed and HTTP requests) are handled in the controller layer of the application. Services are defined as models. Repository layer handles all the database related work. Since all of these services are APIs, there is no need for a presentation/view layer. This is handled in the configuration UI separately.

4.2.2 Logging And Tracking Strategy

This section describes how each VAS request is tracked in the system for the purposes of logging, monitoring, and debugging.

Since there are multiple microservices in the platform, a message is processed in multiple services deployed in multiple servers. Request tracking is essential to understand how each request is processed in this chain of services. Usual way to achieve this is by using log files. In a monolithic application system, there is a single log containing all the details of processing, since this is a distributed application system, there are multiple log files within the system.

Each service has its own log file. This log file is stored in a directory in the same docker container where the service runs. The Docker runtime syncs this log file with the logging directory in the physical server. The operations staff access log files that exist in the physical server. There could be multiple log files for

the same service with the same name and increment ID as a suffix of the name. This happens when the size of the log file exceeds 30 megabytes. At midnight, when a new day begins, the service creates a new log file with the date in the name.

In this platform, there is at least one log file for each service which records internal processing details. Each log file exists in different physical servers. To track a single request, it is required to manually visit each server and check each log file. This is a time-consuming task and maintenance staff needs to have controlled access to physical servers and file directories. Also, the details of a single transaction have spanned across multiple files which makes it harder to filter out transaction details.

To overcome the above mentioned problems, it is necessary to have a strategy that makes it easier to read log files in an effective way. The below solution is formulated to achieve this.

When a message is received by SMS router, it generates a unique ID based on numerical representation of the MSISDN and timestamp. This generated ID is used as the transaction ID, and it is logged along with the logging details. The outgoing JSON message contains the transaction ID and it is used by all other services for logging. This ID is available in the database as the summary of the request. If a request needs to be tracked, the system user can find the transaction ID and filter all the logs using it.

Still the system user must check all the log files manually by retrieving them. To overcome this problem, the below strategy is used.

The idea is to push all the log files into a central file storage server where system users can read them in one place. A Bash script runs on the storage server, and it syncs logs from all other servers with the log storage server . This is done in an interval. There is another Bash which can be used to filter logs by timestamp, MSISDN, original SMS content, and transaction ID.

As for the future enhancements, it is recommended to use an Elasticsearch server to pull the logs and process them in an interactive way.

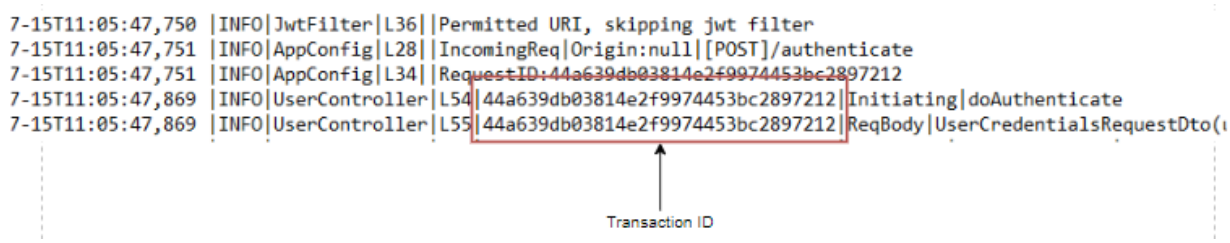


Diagram 4.2.2.1 - Logs format

4.3 Service Workflow

This section describes the workflow of the system and how they relate to each other. The system has below major functionalities,

- **SMS content validation**

This involves validating the content of the SMS against a predefined set of regexes to filter out if the SMS contains a valid service command set.

- **SMS service mapping**
  SMS service mapping involves identification of appropriate service by the SMS commands and matched regex.
- **Service execution**
  Once a service is identified the application will execute the service logic defined in an XML to provide the service that the subscriber asked for. This includes a series of internal and external service calls.
- **Response generation and delivery**
  After finalizing the service request processing, the service will generate an SMS or get a predefined response SMS to be sent to the subscriber to let them know the status of the service request.

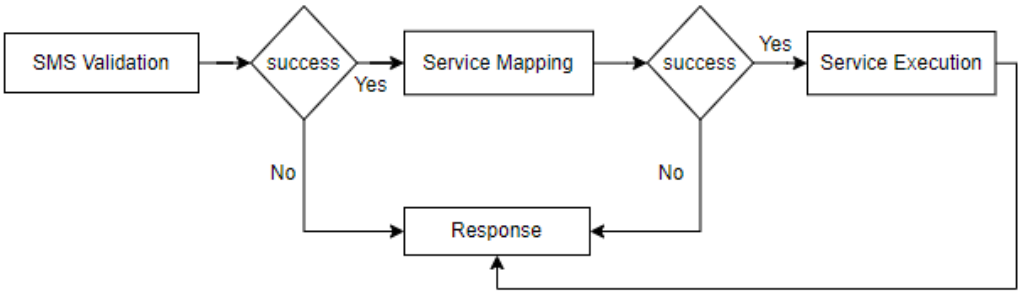


Diagram 4.3.1 - High level service workflow - I

Provision of VASs service includes below workflow,

- **Service flow API management**
  API management is one of the most critical functionalities of the VAS provisioning. This involves creation, testing, and deployment of underlying logical service API. The low code service API builder is used to form the logic. It is also possible to manually develop the service definition and deploy it.
- **Keyword and SMS commands management**
  This deals with regexes. SMS command mapping is totally dependent on pattern matching. When the service requests for a valid pattern match, the system matches the provided commands/keywords set against a predefined list of regexes.
- **Service mapping**
  Each SMS command has its own service to be invoked in order to provide subscribers with the VAS that they are requested.
- **Service action mapping**
  Each service may or may not have an action to differentiate the request based on the command list.
- **SMS response mapping**

Each service action has its own set of predefined subscriber SMS responses. It is possible to have dynamically generated SMS by the service flow API.

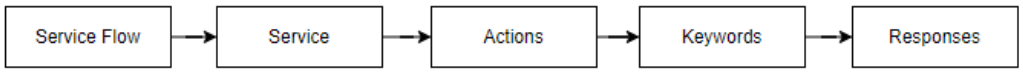


Diagram 4.3.2 - High level service workflow - II

4.4 Software Artifacts

The proposed system contains a total of five newly developed software artifacts as services. Each service can work independently to serve each request.

4.4.1 SMS Router Service

SMS router works as the entrypoint of the platform. Once an SMS is received through SMSC (Short Message service center) it will be published to a specific RabbitMQ topic as a JSON message which contains request information such as correlation ID, received timestamp, sender's MSISDN (contact number). All these messages are stored in a specific queue in RabbitMQ server until they are consumed by the SMS router service. The SMS router service consumes incoming messages through the topic which they are published on.

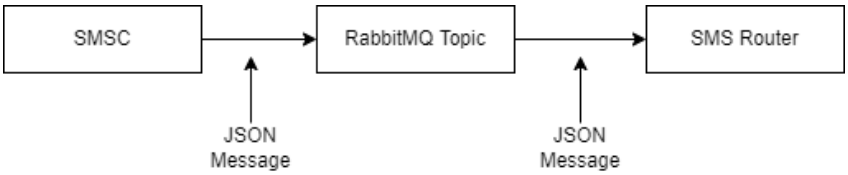


Diagram 4.4.1.1 - SMS router message exchange

Once an SMS is received by the SMS router service, it will fire up a new thread in the application and process the message. A new thread will acquire using an executor thread pool of size of 200 ready to use threads. If all the threads are in use by the service, the new request will wait until a thread becomes available.

Message processing part consists of three processes.

- Validate the SMS content against predefined Regexes.

  The database of the platform contains many regexes to identify the SMS content and the defined service for it. At time of service initialization on the application container, the SMS router service will preload all the service definitions and keyword regexes to the memory so that there will not be any database calls to load them each time an SMS is received. This cache needs to be refreshed

if a system user makes any changes to the database. As a future enhancement it is possible to use a Redis database to use as a database cache to keep this data cached and read them instead of in-memory data of the service. These SMS may contain multiple keywords.

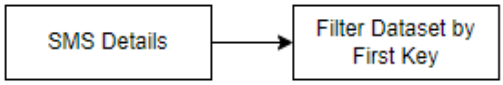


Diagram 4.4.1.2 - SMS discovery - I

Once an SMS is received the service will first go through the dataset to find an appropriate keyword set using the first key of the SMS content. If one is found, it will go through the list of possible keyword sets to get an exact match using regexes. Once a match is found the service will publish the message into another topic which is being consumed by the next service in the chain (Service integrator). This newly published JSON message contains service details such as service ID and name.

- Find the matching service definition.

  Once the service finds a list of possible keyword matches, it will then go through each of these elements while matching the SMS content against regexes. There could be a lot of regexes to match for a single SMS content. Once an exact match is found, it then retrieves service details that are mapped with the found keyword set. The service details object contains service ID, name, and other related information. For the purpose of identifying service, only the service ID is needed at this point.

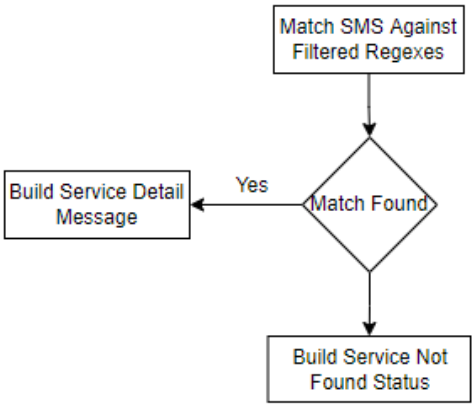


Diagram 4.4.1.2 - Service discovery

When a service is identified, the service will then build a JSON message with the service ID. After that it will be merged with the data of the initial JSON message body.

- Publish SMS and service details into another RabbitMQ topic.

The finalized JSON message body will be published to the next RabbitMQ topic for further message processing. There, the published message waits in a queue until consumed by the Service integrator.

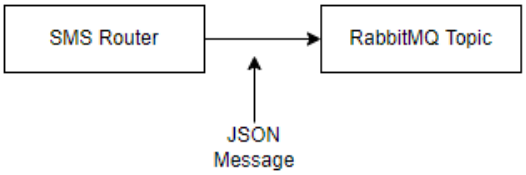


Diagram 4.4.1.3 - SMS router outbound message exchange

4.4.2 Service Integrator

Service integrator works as a container and processing engine of service flows. Service flow is a logical workflow defined as an XML which contains details of how the service should work to provide an output.

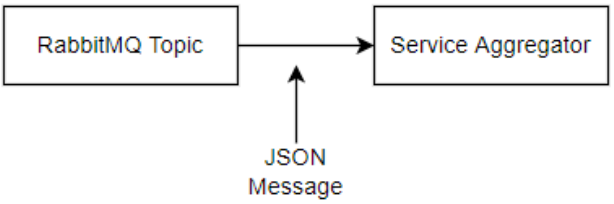


Diagram 4.4.2.1 - Service integrator message exchange

Initially the service integrator takes messages from a specific incoming RabbitMQ topic. These messages contain all the details of identified service and original message details.

Once a message is pushed into a thread to process, first it goes through the cached service details in Service integrator . When the Service integrator is deployed and started in the Docker container, it reads the database and loads all the service details along with XML definitions to an in-memory cache. This in-memory cache will work as the data store for reading required service. This cache should be refreshed if the system user makes any changes to XML service definitions, otherwise older versions of the service will be invoked. If a service is available with a given service ID, it retrieves the XML and executes it with given input parameters.

The service flow is executed by the service flow engine. This is a processing engine which reads XML and goes through block by block as defined in the flow logic. This triggers REST API calls, database calls, and/or any other service integration logic and provides an output as a JSON, XML, or plain text based on the return type definition. Each of these services can be directly invoked through an HTTP API call of the REST API interface. If a service is invoked through the REST API, it will format the output as a JSON and return to the client as a JSON response.

The service flow processing engine has the ability to connect with the API gateway, payment gateway, SMSC gateway, internal databases through mediator REST APIs, and other telecom services. These connectivity's are made possible by using plugins. Creation of these plugins involves writing code and it is not included in the scope of this project.

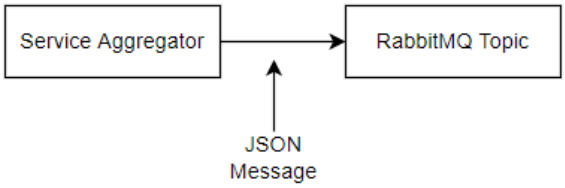


Diagram 4.4.2.2 - Service integrator outbound message exchange

Once the service flow processing is completed, a new JSON message is created with the content of the initial JSON message from the SMS router service. This new JSON message includes the response data of the service flow along with a response ID to identify the SMS to be sent by the Response Handler Service. After the creation of the JSON message, it will be pushed to the next RabbitMQ topic to be consumed by the Response Handler Service. This message will be stored in a RabbitMQ queue which is bound to the same topic and stays there until consumed.

4.4.3 Response Handler Service

Response Handler Service responsible for the creation of the response SMS. Initially this service consumes messages from a specific RabbitMQ topic. These messages are coming from either SMS Router Service or Service integrator Service. Almost all the SMSs coming from SMS Router are error messages and they are handled here accordingly.

Response Handler uses an executor thread pool to process messages asynchronously. There are 200 threads in the thread pool and available threads carry out the message processing. If all the threads are currently in use, then the queue polling function waits until a thread becomes available in the pool. This size of the thread pool is not expected to be exceeded at any given time as the system expects 10 TPS (Transactions per second). Once the processing is completed the thread will return back to the pool to be reused.

Response Handler Service has two main tasks to perform.

- Build the response SMS and send it to the subscriber.

  If the message object contains an error status, then an error SMS will be sent to the subscriber and message details will be sent to the database. Otherwise, an SMS will be picked from the database by querying using response SMS ID extracted from the incoming JSON message from the RabbitMQ topic.

There are two types of SMSs. Plain text SMSs and parameterized SMSs. Plain text SMSs are just simple SMSs, and they can be directly sent to the subscriber number without doing any preprocessing to the message content. Parameterized SMSs are messages that have variables in them. They are not to be sent directly as plain text SMSs. They should be preprocessed to resolve variable values that are extracted from the incoming JSON message. Once a plain text message is built it will be handed over to the SMSC gateway through a REST API request to the Kannel server which is working as an SMPP handler.

- Push the response SMS to the database.

  Second task is to save the transaction summary details along with the transaction ID, original SMS, SMS received time, response time, and response SMS in the database provided by the company's CRM division. This service will not make any database connections directly, instead a REST API is used to publish transaction details. The API will then perform appropriate database operations.

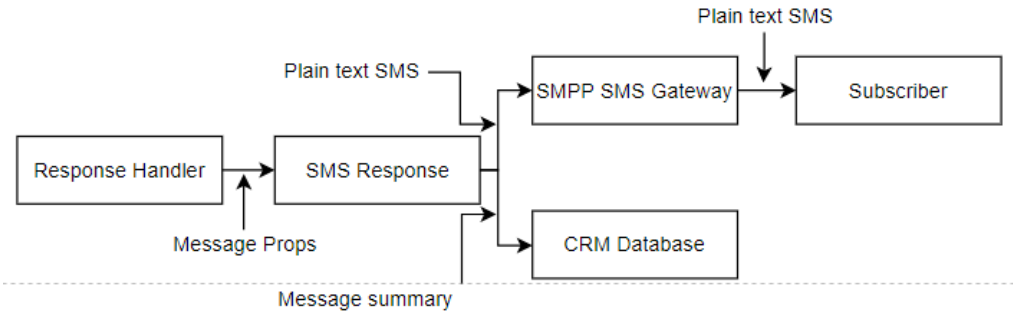


Diagram 4.4.3.1 - SMS response handling

4.4.4 Data Model

The data model is a shared physical component of the platform. This handles all the database operations and contains ORM (Object Relational Mapping) modules of the data access layer. This component is used as a separate library in all other microservices. Data access layer of the system is built as a separate component as the entire system uses the same centralized database and any changes that could be made to the database will result in modifications of all other services. To minimize this impact data access layer is developed as a separate reusable component.

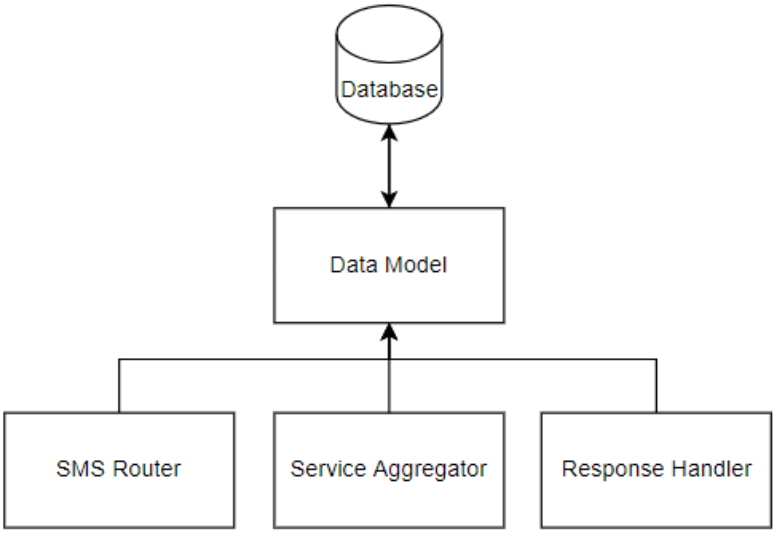


Diagram 4.4.4.1 - Data model integration

There are only two layers in this component, Repository Layer and Entity Layer. Repository layer contains database queries, object transformations, and JPA/Hibernate utility integrations. Entity layer contains ORMs of data objects and related database tables.

For the development of this component JavaEE, JPA/Hibernate, and Spring Boot Data module are used. Spring Boot Data is prebuilt with essential data access functions such as dynamic query building, exception translation, and data transformation. Also it provides utility interfaces to interact with the database in an effective way such as JPA and CRUD repositories.

In the future if there are changes that need to be made to the database, all the changes are reflected in this data model and it should be released as a new version of the repository. Other services can then use the updated latest version of the library and make service level changes accordingly.

This service component is developed as a Java Archive or library which can be integrated with any Spring Boot web service component using Maven or any other build configuration management tool.

4.4.5 Management Web Application

Management web application is the front face of the system. The system users access the system using this web interface. This web interface provides all the necessary functionalities to provision VASs in an interactive way. This application can be accessed through any modern web browser and users must log in to get access. Web application provides all the required functionalities for creation, testing, designing, monitoring, and maintenance of VASs. This also provides a low-code graphical service builder to design, build, test, and deploy REST APIs to designated environments. This user interface also provides versioned access to workflow APIs, usage and performance statistics of each workflow API and service.

The application has two major functions,

- VAS provisioning.

  This includes creation of new VASs, test, and deploy them to staging and production environments. Users can make changes to existing services through the user interface and regulate them.

- Build service flows as APIs.

  Each VASs should have a service flow API to function as the core logic. This service can be built using the web interface. The web application has a feature called API designer. The API designer is a low-code interactive service builder. Users can use this to design and create new services. The application provides all the necessary features as nodes to form a logical flow and interact with external systems. Users can also build their custom features as plugins and publish them in the platform. These plugins can be used in other services to build APIs.

  For the development of the web application JavaScript and React JS is used. React JS is used as a user interface development library and for the UI elements, Microsoft's Fluent UI design and components are used. An NGinx server will serve the finalized application as a web server.

4.4.6 Backend API of the Web Application

Backend REST API is responsible for interacting with the web user interface and support operations carried out by the web application.

This includes below mentioned major functions.

- Authentication and Authorization

  Both authentication and authorization are built with Spring Security module and JWT (JSON Web Tokens). Authentication is the process of recognizing a user's identity (The Economic Times, n.d.). This associates a request with identifying credentials. When a user is going to log in to the system, they should provide their username and password. These credentials then verified against the data found on the database. If the user credentials are matched the user has successfully authenticated with the system.

  Authorization is to check if the user has the access rights to access protected API endpoints of the REST API. At the authentication step, once a user has successfully authenticated the system will generate an access token as a JWT and send it back to the client web application. This access token includes the scope of the user. User scope is a set of access levels specially granted to that user.

Once a request is received by a protected endpoint of the REST API, it will look for authorization HTTP header in the request headers list. This header must contain a valid access token. The security service of the API will validate the token against expected claims and then it checks whether the user has permission to access the requested resource based on the scope of the token. This is explained in more detail in the security section.

- Service provisioning.

  This includes creation and regulation of VASs, keyword mapping, service mapping, service flow API definitions management, SMS mapping, and regex building.

This is built using JavaEE and Spring Boot as the core web framework. The resulting web service is a JSON REST API which is entirely independent from the web user interface.

4.4.7 Low-Code Processing Engine

This is one of the most critical artifacts of the platform. Low code processing engine responsible for creation, evaluation, and execution of REST API service definitions stored in the database and provide API endpoints to access these services.

The low-code graphical user interface is used to design APIs and form the logical flow of the service provisioning. This is done by using directed graphs of various nodes along with the data bound to each node and edges that contain relationships between each node. Once the graph is built, it will generate an XML which contains service workflow. This XML is then minified to save space and remove unwanted white spaces from the XML, then it will be stored with a commit ID and a version number in the database. The API registry maintains a number of API workflows for each API differentiated by this commit ID and version number. Commit ID is a unique ID created by calculating the MD5 hash of the appropriate XML.

The REST API workflows also has its own version control mechanism. This is provided to maintain multiple versions of workflows of the same API. Each API can be independently deployed or tested. Once an API is deployed it will override the currently in-use API to take place of it. APIs can be uniquely tested in the staging environment. The API to be tested should be deployed in the staging environment prior to the test. Each version can be updated by foking it to the low-code builder UI. Once the changes are made it will be saved with a new version number and a commit ID.

Every XML workflow is loaded to the memory from the database by the processing engine once they are requested. For the purpose of speeding up the execution, these invoked XMLs are stored in an in-memory cache of the service so that there will not be any database round trips to load XMLs. if the cache contains the requested workflow it will be executed, otherwise it will be loaded from the database. Once a new patch of a workflow is deployed, the processing engine will refresh the cache to reload the changed workflow.

Workflow execution is carried out as a sequence of node executions. There are multiple core nodes and function nodes to support the workflow. Each node contains information of the node before it and the node after it as in a linked list. Also each node contains the information of graphical position in the graph, node type, inner function type, display labels, as well as properties of nodes such as expressions, variables, and function properties.

All the nodes are categorized under two sections,

- Core nodes
  Core nodes are the ones that are mandatory to build a workflow. These nodes contain primary functionalities to form a logical flow of nodes such as variable declarations, conditional routing, and looping. These are hardcoded with the system.
- Function nodes
  These types of nodes provide functions to extend features of the workflows. These are single purpose configurable functions defined to carry out next level operations such as external system integrations.

Below list contains the details of nodes and their functionalities.

- Assign
  Assign node is to declare and keep variables bound to the workflow. These variables have dynamic types at runtime. It is possible to define integers, floating point variables, strings, characters, booleans, and lists. This also supports JavaScript functions that return variables as row types and objects. These JavaScript functions will be executed with the given arguments by the processing engine.
- Branch
  Branch is there to control access to each node by providing conditional expressions. In programming this can be compared with IF-ELSE and SWITCH/CASE. A branch should contain at least one CASE block and only one default block. There can be many CASE blocks.
- Case
  A Case block is associated with a Branch. Each case block has an expression/condition which is executed at runtime to evaluate the flow control. Case blocks can contain variables as in the Assign block. This works exactly as IF part of an IF-ELSE or Case of a Switch-Case.
- Default
  The Default block is the part that gets executed if any of the Cases were not evaluated to true. Variables can be declared here as in Case block.
- HTTP
  This is a type of a function node. An HTTP node can be used to make HTTP requests to designated web resources. Mainly this can be used to call REST APIs of external systems and call API workflows defined in this platform through HTTP. The response is expected to be in a valid JSON format; it will be bound with the workflow session so that the response can be read later in the workflow.
- Invoke

Invoke is another type of function node which can be used to invoke another API workflow defined in the system through the current workflow.

- Email
  This node sends emails to provided recipients.

Once the processing of the XML is completed it will generate a JSON formatted output and send it back to the REST client or push it to the service queue for further processing of the message. This response typically contains all the variables and responses from functions unless otherwise specified.

4.4.8 Database

The database of the system is a centralized MySQL database. All the data are distributed among logical tables. The database is backed up once an hour to make sure that the data is safe in case of a disastrous event. Some of the data is cached in memory of services to avoid expensive database round trips. When new changes take place, services will trigger a refresh event to load the data again to the service memory so that the updated data will be processed. Structure of the database schema is explained under the implementation section in more detail.

4.4.9 Message Queue Service

RabbitMQ works as the underlying event processing and message sharing service. This messaging server has the capacity to handle large volumes of data. Each of the core services of the platform (SMS Router, Service integrator , and Response Handler) are dependent on data streams of RabbitMQ message queues. Those services push and poll messages from specific queues asynchronouesly.

4.5 System Security

This section walks through how standard security practices are adapted and implemented in this system to make sure that the system provides maximum security to end to end operations and its data.

4.5.1 Authentication

User/request authentication is a critical part of the proposed system, this makes sures only securely validated users can continually use the system. For authentication, users should use their username and password to login and the system will check those credentials. If the credentials are successfully validated, then the particular user can continue to work. Otherwise the system will reject the request to authenticate until valid credentials are provided.

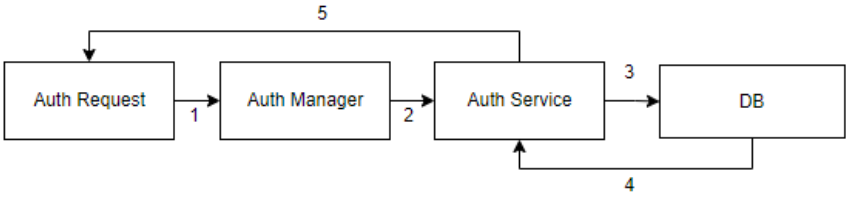


Diagram 4.5.1.1 - Authentication

From the technical perspective, system authentication is built with the features provided in the Spring Security module. This involves an in-built database authentication manager. User credentials are stored in a MySQL database. Once a user requests a credential check/login, the authentication manager will check provided credentials against matched records in the database. Passwords are stored in the database as an encrypted hash.

User provided password is encrypted to get the hash of plain text prior to matching against the value in the database. After that both hashed passwords are compared to check if they match. Encryption method of passwords is described in section 4.4.3 in detail. The system authenticates the user credentials only after both username and password are correctly matched with a record that exists in the database.

4.5.2 Authorization

For the authorization, the system uses functionalities provided with Spring Security module and RBAC (Role Based Access Control) along with JWT (JSON Web Token). The database contains a set of user roles and each user has one or many roles. At the time of login/authentication, registered user roles are queried from the database. Once a user has successfully authenticated with the system, the authentication service generates an access token as a JWT and sends it back to the web client. This JWT contains user ID, scope, and native claims such as subject, issuer, expiration, and generated time. User ID is to identify the user of the token and scope is to identify access levels of the authenticated user. This includes a list of user roles as granted authorities.

The web application stores the initial JWT in the local storage of the web browser and sends the token with every request that is sent to the API as a request header (Authorization header). The API extracts the token from the authorization header and checks the validity. If this validation succeeds, the authorization manager checks if the token has the permission to access the requested resource. This is done by evaluating the token scope and roles associated with the user stored in the database. Once it resolves the permission levels, the API will continue with the processing, otherwise it will reject the request with the HTTP 403 error. Each generated JWT has a maximum of 24 hour validity period. When this time expires the server will reject all the requests that contain the same expired token. The API uses a securely stored HS256 key to sign JWTs and this is subject to change.

4.5.3 Encryption

Encryption involves transforming data into a format that only the authorized parties can understand. This converts human-readable plain text into incomprehensible text (Cipher text) (Cloudflare, n.d.).

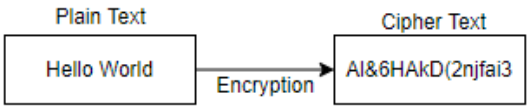


Diagram 4.5.3.1 - Encryption

This platform uses encryption to convert plain text passwords into incomprehensible text and store them in the database for the security of them. BCrypt is used by default as the encryption algorithm. “BCrypt was designed for password hashing hence it is a slow algorithm. This is good for password hashing as it reduces the number of passwords by second an attacker could hash when crafting a dictionary attack” (Dan Arias, 2018). Therefore, all the passwords are hashed using BCrypt hashing algorithm in this platform to maximize password security.

All the passwords are hashed and stored at the time of user creation/registration and password change. Upon user login, these stored credentials are acquired and cross checked against credentials provided by the user.

4.5.4 Accessibility

The system is deployed in an inhouse server infrastructure so that only employees of the company can access the system. Users must log in to a remote windows server in order to get access to internally deployed application interfaces. They can not directly access their computing devices. This imposes another layer of security as users must be logged into a remote server prior to accessing the VAS system. Users will have to use their active directory credentials to login to the remote server and use application specific credentials to access the system.

4.5.4 Workflow Authenticity

API workflows are stored in the database as plain text XMLs. Since these XMLs are critical for the service execution, they must be highly secured as they are executed directly by the processing engine. if the services contain malicious parts, it will cause unprecedented security issues. To overcome this issue the system uses a hash to evaluate the authenticity of the XML. Each stored XML is associated with a MD5 hash code. This code is generated at the time of saving the XML in the database. Each time any service is invoked the processing engine will calculate the MD5 hash of the XML and compare it with the stored hashed value. The workflow will only be executed if this hash comparison is successfully completed. Otherwise a fallback mechanism is activated and reported as a security issue.

5. Implementation

This section provides details on how the system is implemented using mentioned technologies and frameworks. For illustration purposes, UML diagrams and custom high level design diagrams are used to depict each part.

5.1. Components

For the development of the backend services including REST API and core microservices, Spring Boot framework and Java 8 were used. Below UML component diagram visualizes how each component is interrelated with each other.

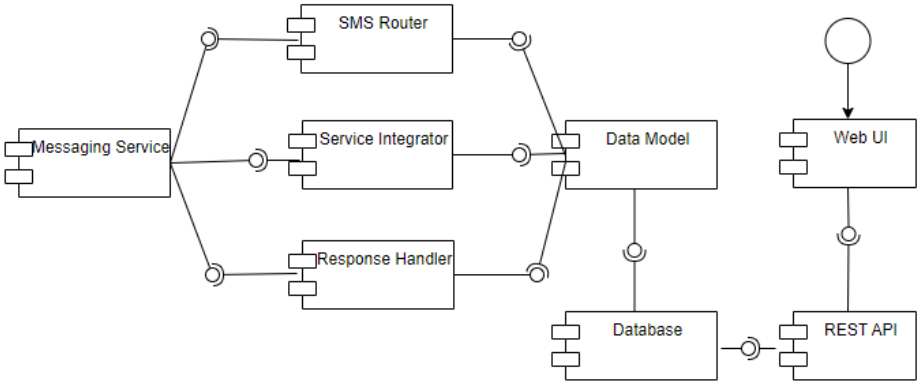


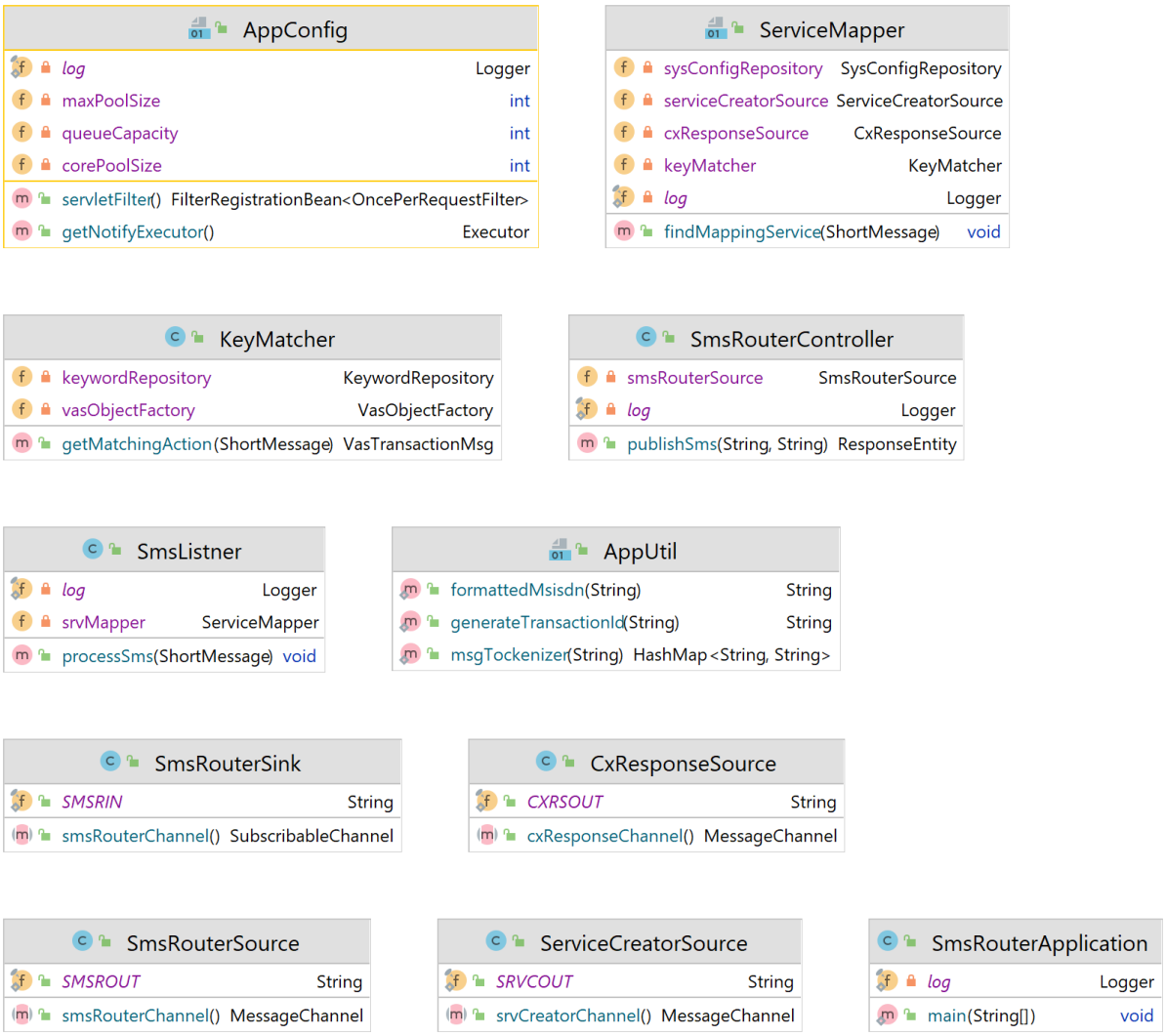
Diagram 5.1.1 - Components distribution

Each core microservice (SMS Router, Service Integrator, and Response Handler) is connected to RabbitMQ messaging service to push and pull messages in consumer/producer pattern. At the same time these services are connected to the Data Model component to communicate with the database to get service related data. Data Model is built as an independent library which contains all the data access related functionalities and all these three core microservice have it integrated.

User interacts with the Web UI and the REST API is there to support the Web UI functionalities. This REST API and Web UI communicate with each other using HTTP. REST API exposes a number of protected and public REST endpoints to provide data. REST API directly communicates with the database.

5.1.1. SMS Router

For the development of this service JavaEE and Spring Boot web MVC framework is used along with Spring Cloud Streaming module to interact with RabbitMQ message queue. A REST API endpoint is defined to receive inbound SMSs from the SMSC.

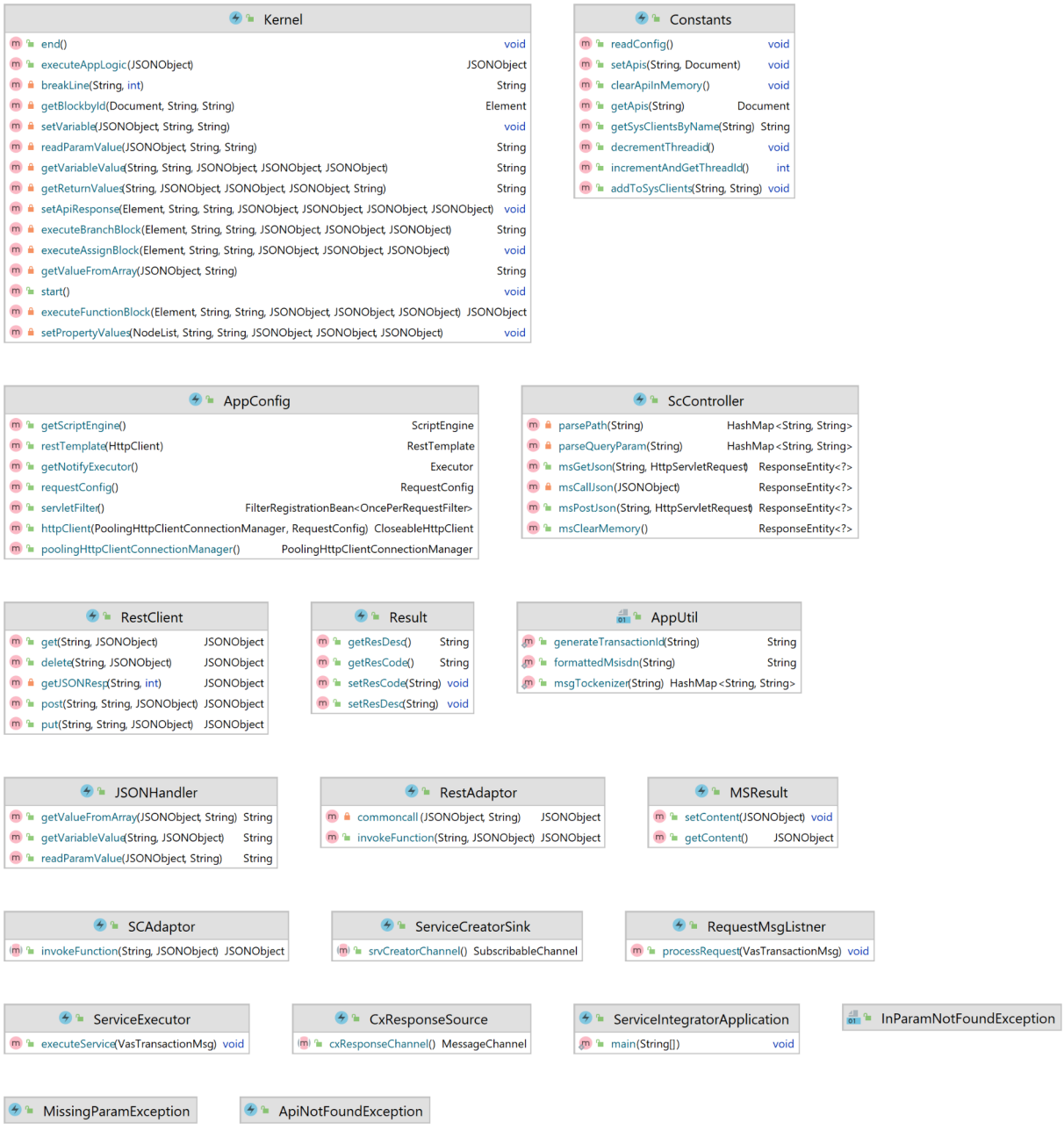


5.1.1.1 Diagram - SMS router class structure

5.1.2. Service Integrator

For the development of this service JavaEE and Spring Boot web MVC framework is used along with Spring Cloud Streaming module to interact with RabbitMQ message queue. A REST API endpoint is defined to receive direct API calls for the deployed service workflow APIs.

Once a message is received from a queue a new thread is spawned to process the message asynchronously. The thread is acquired from an executor thread pool same as in the SMS Router Service. There are 200 threads available in this thread pool.



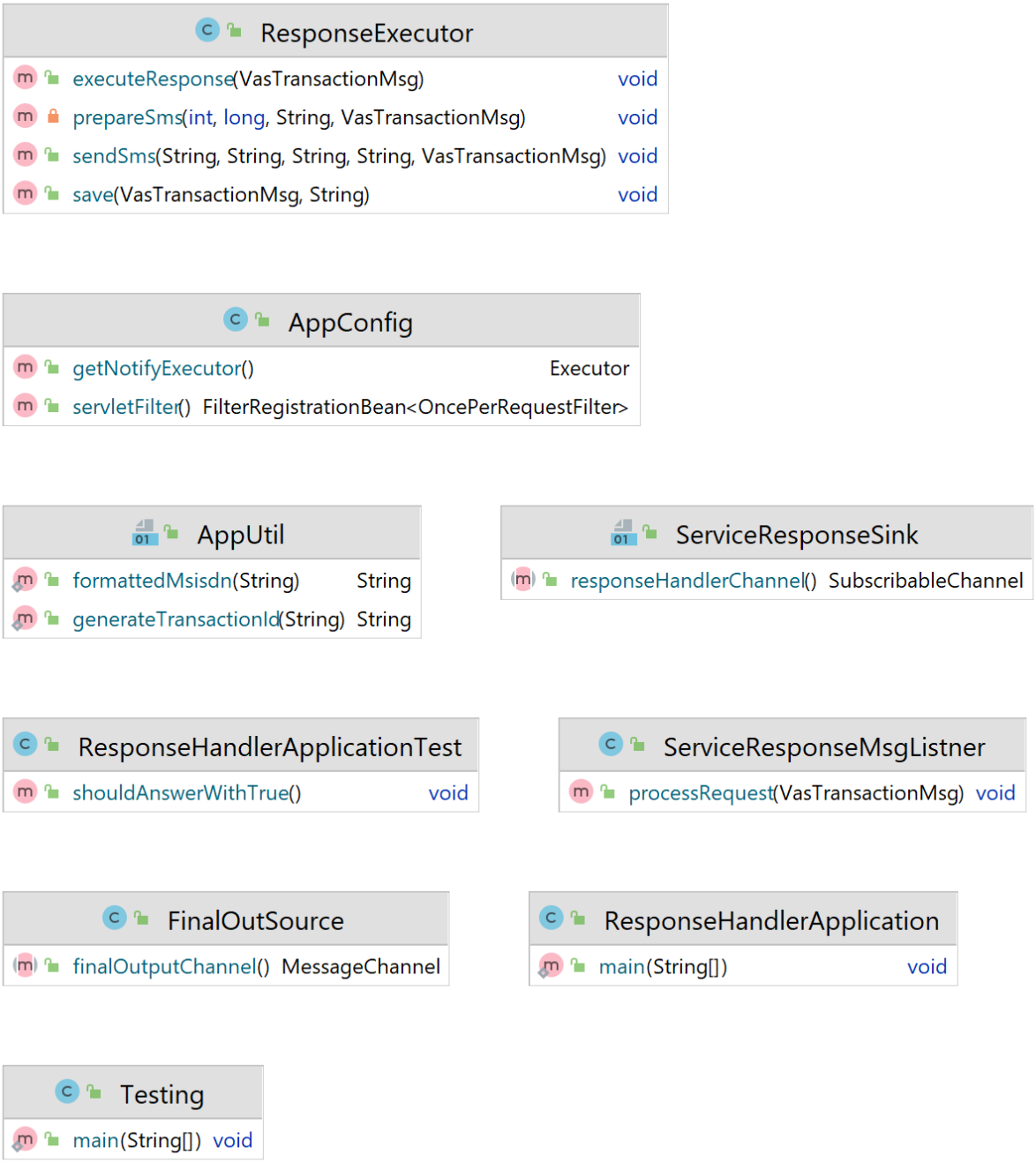
5.1.2.1 Diagram - Service integrator class structure

```
@Bean("serviceExecutorPool")
public Executor getNotifyExecutor() {
    ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
    executor.setCorePoolSize(corePoolSize);
    executor.setMaxPoolSize(maxPoolSize);
    executor.setQueueCapacity(queueCapacity);
    executor.setThreadNamePrefix("sc-srv-exec-");
    executor.initialize();
    return executor;
}
```

5.1.2.2 Code sample - Executor thread pool

5.1.3. Response Handler Service

For the development of this service JavaEE and Spring Boot web MVC framework is used along with Spring Cloud Streaming module to interact with RabbitMQ message queue.



5.1.3.1 Diagram - Response handler class structure

5.1.4. Data Model

For the development of the data model component Spring Boot framework used along with Spring Data JPA to access the database.

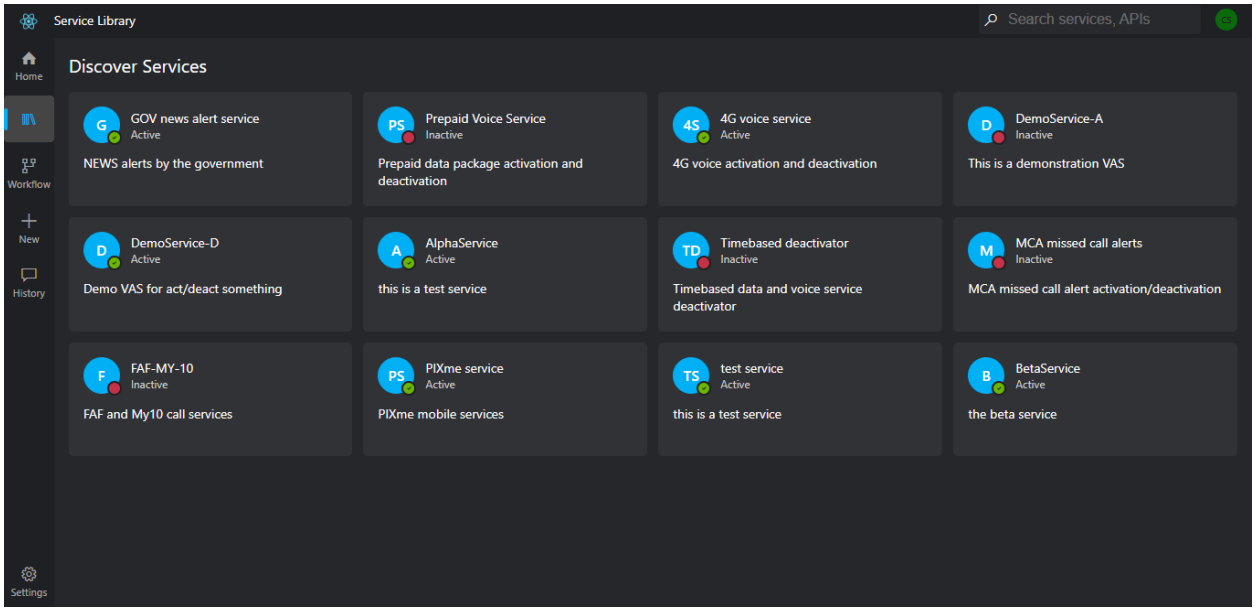
```
public interface ApiHistoryRepository extends JpaRepository<ApiHistoryEntity, Long> {

    @Query("select his " +
            "from ApiEntity api, ApiHistoryEntity his " +
            "where " +
            "api.id = his.api.id and api.name = :name and his.isActive = true")
    Optional<ApiHistoryEntity> findActiveApiByName(@Param("name") String name);

    @Modifying
    @Query("update " +
            "ApiHistoryEntity his " +
            "set " +
            "his.totalRequestsCount = his.totalRequestsCount + 1, " +
            "his.avgResTime = :avgResTime + his.avgResTime, " +
            "his.errorCount = :errorCount + his.errorCount " +
            "where " +
            "his.api.id = (select api.id from ApiEntity api where api.name =:apiName) " +
            "and " +
            "his.isActive = true"
    )
    void updateStats(
            @Param("apiName") String apiName,
            @Param("avgResTime") Long avgResTime,
            @Param("errorCount") Long errorCount
    );
}
```

5.1.4.1 - Sample code - Data access layer

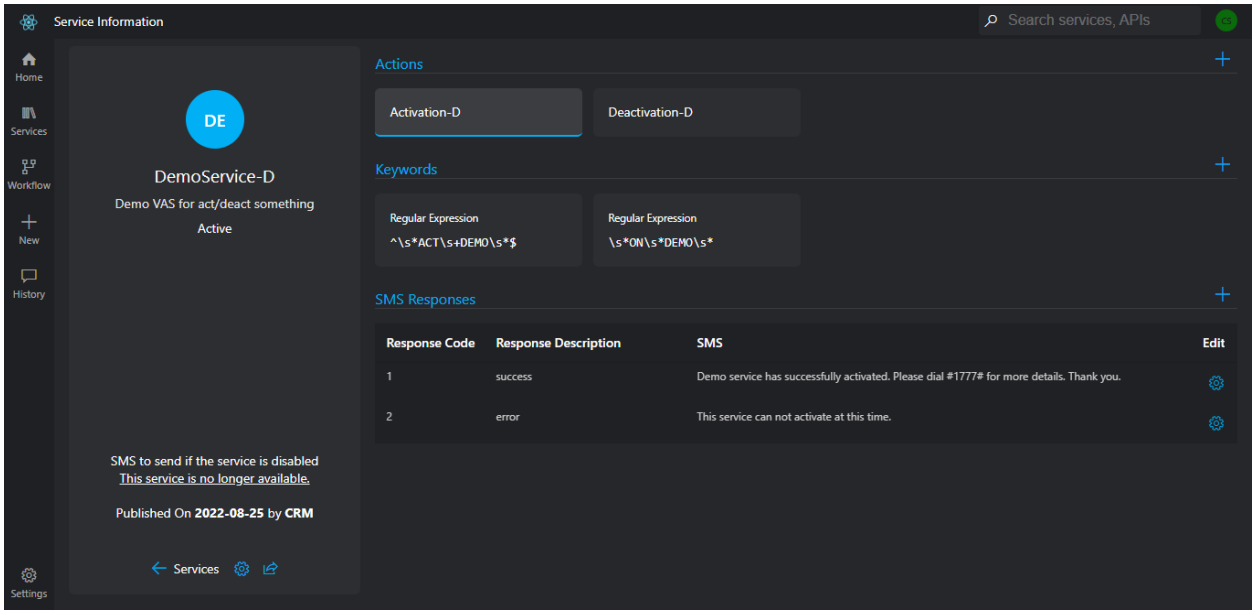
5.1.5. Management Web Application

This web application is built with React JS. as the frontend user interface component library, Microsofts' Fluent UI library and UI Fabric styling guidelines are used. The application is built with Webpack, NPM, and NodeJS. This will be deployed into an NginX web server.

5.1.5.1 - UI - Dashboard



5.1.5.1 - UI - Service library

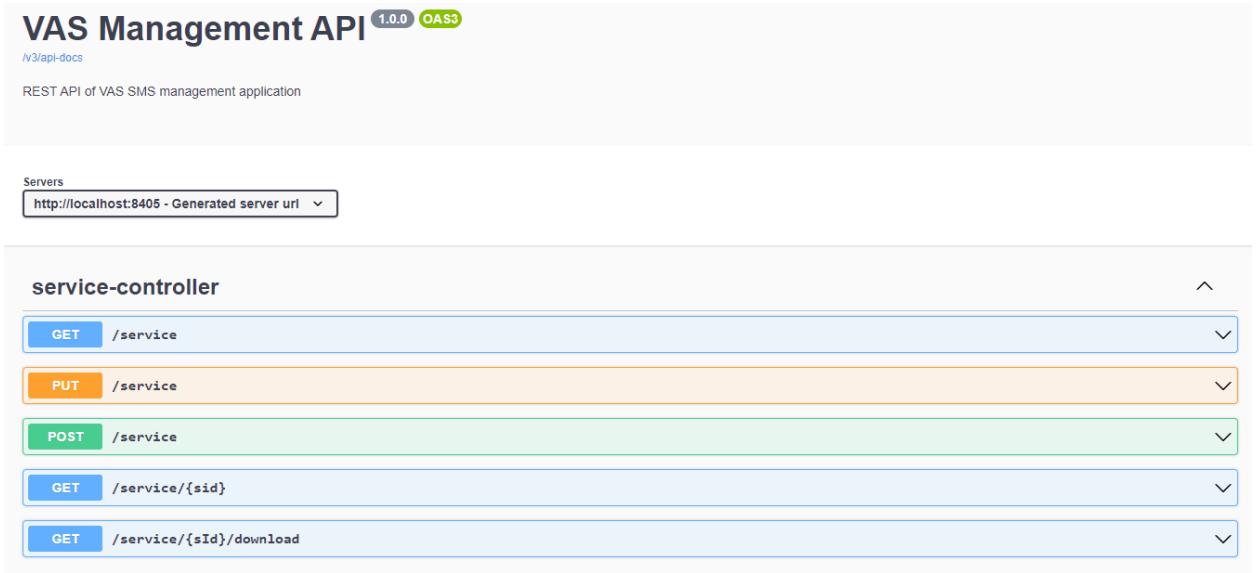


5.1.5.1 - UI - Service information

More user interface designs are included in Appendix A.

5.1.6. Backend API of the Web Application

For the development of this service JavaEE and Spring Boot web MVC framework is used. Backend API is developed as a REST API which exchanges JSON messages with the web application. Below images show Swagger API definitions of the REST API.



keyword-controller

- PUT /keyword
- POST /keyword
- GET /action/{aid}/keyword

cx-response-controller

- PUT /cx-response
- POST /cx-response
- GET /api/{api_id}/cx-responses

api-controller

- GET /api
- PUT /api
- POST /api
- POST /api/{id}/deploy/{commitId}
- POST /api/commit
- GET /api/{id}
- GET /api/{id}/versions

action-controller

- PUT /action
- POST /action
- GET /service/{sid}/action

user-controller

- POST /user/create
- POST /authenticate
- PATCH /user/act-deact/{userId}
- GET /user

sms-history-controller

- GET /history

config-controller

- GET /config/sms/stats
- GET /config/health
- GET /config/actions

5.1.6.1 - Swagger API definitions of REST API

5.1.7. Low-Code Processing Engine

This engine is developed as a part of the service integrator. This is a pure Java 8 based XML processing program which processes service workflow APIs.

XML implementation details are shown below.

A sample of Assign block XML is shown below.

```
<block id="1" type="assign" nodeCType="customNode" pos="176.453125,225" icon="Variable">
    <variable name="inKey">'%ip.tokens.token1%'</variable>
    <variable name="inKey2">'%ip.tokens.token2%'</variable>
    <variable name="msisdn">'%ip.msisdn%'</variable>
    <next-node>2</next-node>
</block>
```

Below sample XML code snippet shows an Assign block with JavaScript function that returns a value.

```
<block id="7" type="assign" nodeCType="customNode" pos="845.453125,164" icon="Variable">
    <variable name="resCode">1</variable>
    <variable name="jsVar">
        function jsVar(x, y) {
            var cx = x, cy = y;
            return cx * cy;
        }
        jsVar(2, 5);
    </variable>
    <next-node>END</next-node>
</block>
```

A sample of Branch block XML is shown below.

```
<block id="2" type="branch" nodeCType="customNode" pos="340.453125,224" icon="BranchFork2">
    <case id="3" type="case" nodeCType="customNode" pos="506.453125,165">
        <expression>'%ses.inKey%' == 'ACT' && 1 == 1</expression>
        <variable name="action">"ACTIVATION"</variable>
        <next-node>6</next-node>
    </case>
    <case id="5" type="case" nodeCType="customNode" pos="508.453125,264">
        <expression>'%ses.inKey%' == 'DEACT'</expression>
        <variable name="action">'DEACTIVATION'</variable>
        <variable name="resCode">2</variable>
        <next-node>END</next-node>
    </case>
    <default id="4" type="default" nodeCType="customNode" pos="501.453125,344">
        <variable name="action">"NULL"</variable>
        <variable name="resCode">3</variable>
        <next-node>END</next-node>
    </default>
</block>
```

Below XML snippet shows internal details of a Case block and how it is constructed.

```
<case id="3" type="case" nodeCType="customNode" pos="506.453125,165">
    <expression>'%ses.inKey%' == 'ACT' && 1 == 1</expression>
    <variable name="action">"ACTIVATION"</variable>
    <next-node>6</next-node>
</case>
```

A sample of the HTTP node XML definition is provided below.

```
<block id="6" type="func" nodeCType="customNode" pos="695.453125,165" icon="Remote">
    <functionType>http</functionType>
    <method>get</method>
    <class>rest</class>
    <param name="url">https://run.mocky.io/v3/15f6d1bf-4b96-4261-b2e7-d26cc41bf140</param>
    <param name="body">{}</param>
    <next-node>7</next-node>
</block>
```

Below XML snippet shows internal details of a Default block and how it is constructed.

```
<default id="4" type="default" nodeCType="customNode" pos="501.453125,344">
    <variable name="action">"NULL"</variable>
    <variable name="resCode">3</variable>
    <next-node>END</next-node>
</default>
```

Complete sample of a generated XML service definition is presented below.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name='TestApi'>
  <block id="0" type="assign" nodeCType="startNode" pos="25,225">
    <next-node>1</next-node>
  </block>
  <block id="1" type="assign" nodeCType="customNode" pos="176.453125,225" icon="Variable">
    <variable name="inKey">'%ip.tokens.token1%'</variable>
    <variable name="inKey2">'%ip.tokens.token2%'</variable>
    <variable name="msisdn">'%ip.msisdn%'</variable>
    <next-node>2</next-node>
  </block>
  <block id="6" type="func" nodeCType="customNode" pos="695.453125,165" icon="Remote">
    <functionType>http</functionType>
    <method>get</method>
    <class>rest</class>
    <param name="url">https://run.mocky.io/v3/15f6d1bf-4b96-4261-b2e7-d26cc41bf140</param>
    <param name="body">{}</param>
    <next-node>7</next-node>
  </block>
  <block id="7" type="assign" nodeCType="customNode" pos="845.453125,164" icon="Variable">
    <variable name="resCode">1</variable>
```

```
      <variable name="response">'%key%'</variable>
      <next-node>END</next-node>
    </block>
    <block id="2" type="branch" nodeCType="customNode" pos="340.453125,224"
icon="BranchFork2">
      <case id="3" type="case" nodeCType="customNode" pos="506.453125,165">
        <expression>'%ses.inKey%' == 'ACT' && 1 == 1</expression>
        <variable name="action">"ACTIVATION"</variable>
        <next-node>6</next-node>
      </case>
      <case id="5" type="case" nodeCType="customNode" pos="508.453125,264">
        <expression>'%ses.inKey%' == 'DEACT'</expression>
        <variable name="action">'DEACTIVATION'</variable>
        <variable name="resCode">2</variable>
        <next-node>END</next-node>
      </case>
      <default id="4" type="default" nodeCType="customNode" pos="501.453125,344">
        <variable name="action">"NULL"</variable>
        <variable name="resCode">3</variable>
        <next-node>END</next-node>
      </default>
    </block>
    <block id="return" type="return" nodeCType="returnNode" pos="1025,225" icon="undefined">
      <outputparams name="ses">-1</outputparams>
    </block>
</service>
```

5.2. Database

Below diagram shows the structure of the database of the system.

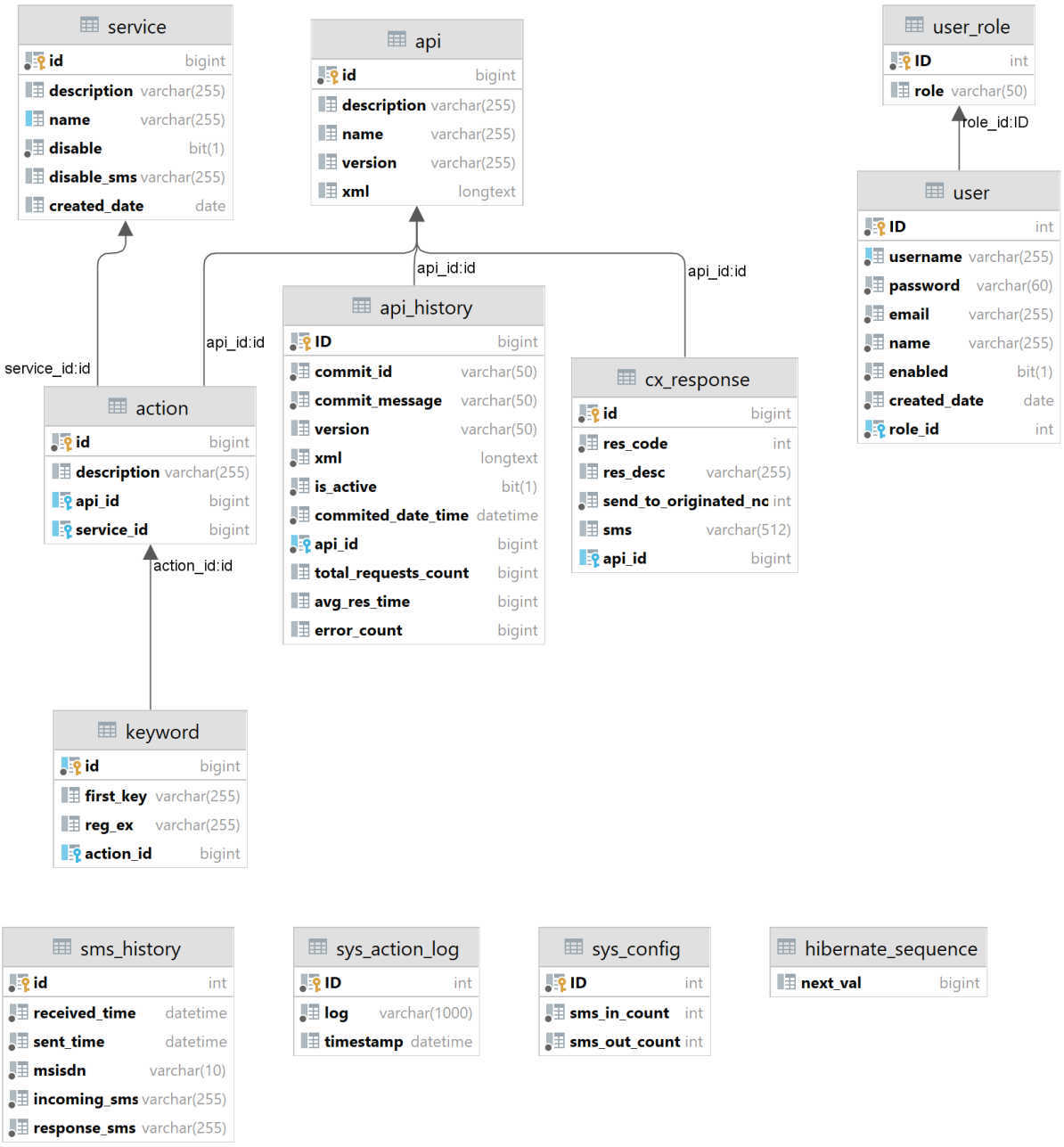
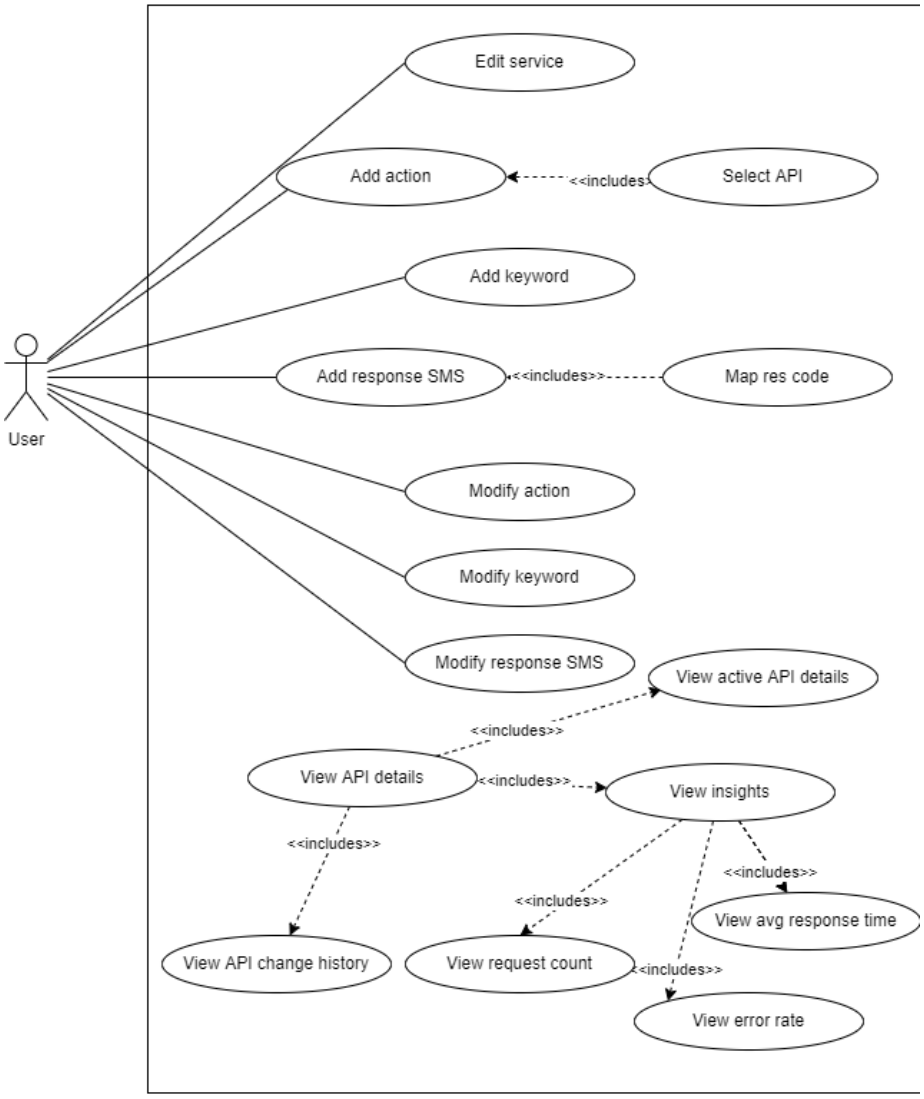


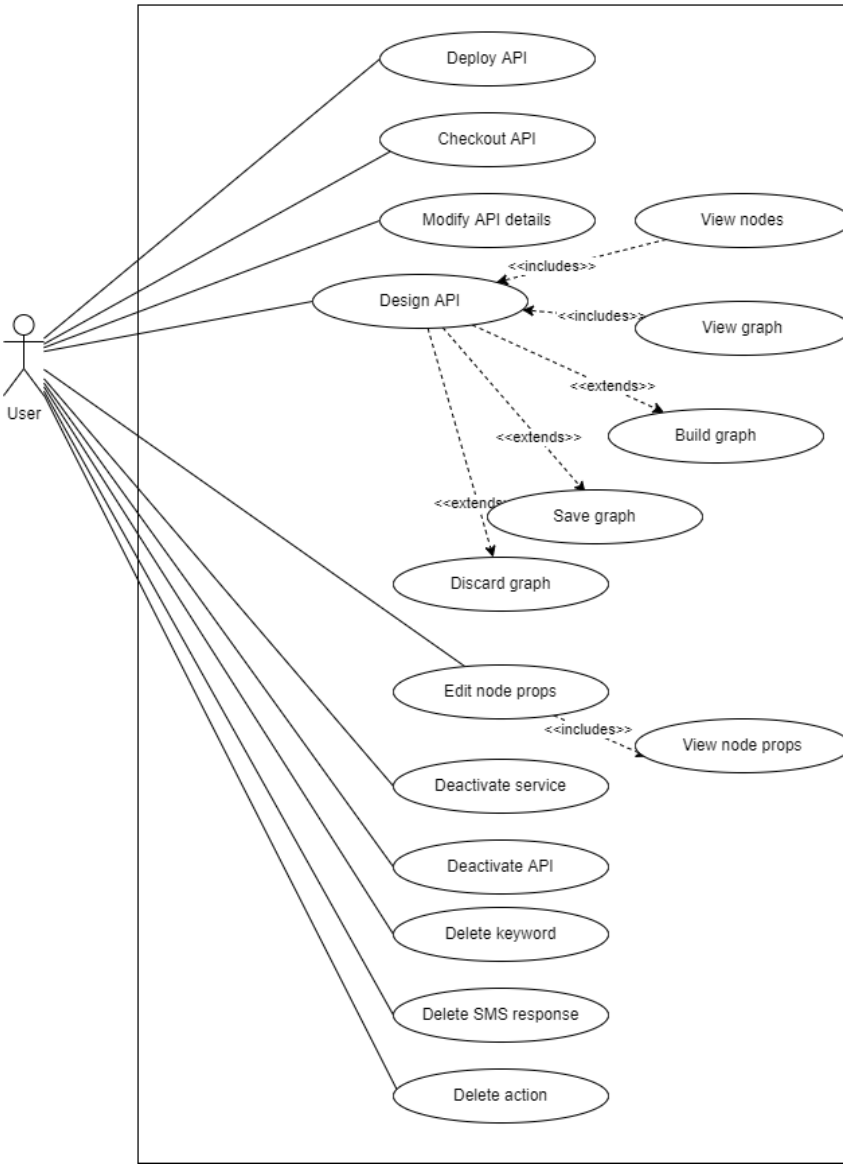
Diagram 5.2.1 - Database design

5.3. Use case diagram

5.3.1 - Use case diagram - I



5.3.2 - Use case diagram - II



5.3.3 - Use case diagram - III

5.4. Implementation Support

5.4.1. Hardware

This section specifies the hardware requirements for the implementation of the platform. Each service will be deployed separately on BMSs.

Name	Storage	RAM
SMS Router	SSD - 500GB	8GB
Service Integrator	SSD - 1TB	16GB

Response Handler	SSD - 120GB	8GB
Web Application	SSD - 120GB	8GB
REST API	SSD - 500GB	8GB

Table 5.4.1.1 - Hardware specs

5.4.2. Software

This section specifies required softwares to implement the product.

Name	Type	Version	License
Kubernetes	Deployment	1.23	Free
Docker	Deployment	4.6.1 (76265)	Personal (Initially)
NginX	Web server	2	Free
MySQL	DB	8.0	Enterprice
Linux Kernel	Operating System Kernel	5.15	Free
Redhat Linux	Operating System	7	Free
RabbitMQ	Message broker		Free

Table 5.4.2.1 - Software specs

6. Evaluation

This section provides information on how the system is evaluated in order to validate if the system satisfies the requirements of the system and the purpose.

6.1. Test cases

Test case ID	Category	Objective	Steps	Input	Expected result	Actual result	Status
TC01	Login	Verify user can see email and password fields in the screen	goto url goto login page		user can see email and password	As expected	Pass
TC02	Login	Verify the user can type username and password then submit	goto login form type into username type into password click login	username=demo password=alpha@123	user can type into username and password and submit button is disabled until response comes back	As expected	Pass
TC03	Login	Shows an error message when wrong username is provided	goto login form type into username type into password click login	username=demo1 password=alpha@123	shows an error message below the submit button	As expected	Pass
TC04	Login	Shows an error message when wrong password is provided	goto login form type into username type into password click login	username=demo password=alpha@1234	shows an error message below the submit button	As expected	Pass
TC05	Login	Redirects to home/dashboard page when conrrect credentials are used	goto login form type into username type into password click login	username=demo password=alpha@123	Redirects to homepage	As expected	Pass
TC06	Home	verify if the user can see service status	goto home page		homepage shows service stats as tiles	As expected	Pass
TC07	Home	verify if the user can see service usage status	goto home page		homepage shows service usage stats as tiles	As expected	Pass
TC08	Home	verify if the user can see recent action list	goto home page		homepage shows recent action list as items	As expected	Pass
TC09	Home	verify if the recent action log is limited to maximum 10 items	goto home page		recent action log only shows maximum 10 entries	As expected	Pass
TC10	Home	verify if service statistics panel shows loading animation before content load	goto home page refresh the page		service statictics panel shows loading animation	As expected	Pass
TC11	Home	verify if service usage statistics panel shows loading animation before content load	goto home page refresh the page		service usage stats panel shows loading animation	As expected	Pass
TC12	Home	verify if recent action log panel shows loading animation before content load	goto home page refresh the page		recent action log panel shows loading animation	As expected	Pass

TC13	Service library	verify if the service library page shows services	goto services page		page shows services as tiles	As expected	Pass
TC14	Service library	verify if the page contains four service tiles in one row	goto services page		page shows services as tiles, four items in a single row	As expected	Pass
TC15	Service library	verify if a service tile contains name of the service, status, and description	goto services page		tile shows mentioned details	As expected	Pass
TC16	Service library	verify if a service tile shows status as a tick along with the name	goto services page		tile shows red tick if the service is inactive, green tick otherwise	As expected	Pass
TC17	Service library	verify if the page shows a loading animation before the content is loaded	goto services page refresh the page		page shows a loading animation on page load	As expected	Pass
TC18	API library	verify if the API library page shows APIs	goto API/workflow page		page shows APIs as tiles	As expected	Pass
TC19	API library	verify if the page contains four service tiles in one row	goto API/workflow page		page shows APIs as tiles, four items in a single row	As expected	Pass
TC20	API library	verify if a API tile contains name of the API, version, commit ID, and description	goto API/workflow page		tile shows mentioned details	As expected	Pass
TC21	API library	verify if a API tile shows status as a tick along with the name	goto API/workflow page		tile shows red tick if the service is inactive, green tick otherwise	As expected	Pass
TC22	API library	verify if the page shows a loading animation before the content is loaded	goto API/workflow page		page shows a loading animation on page load	As expected	Pass
TC23	Service creator	verify if the page shows create service tile	goto service creator page		the page shows a tile named create service	As expected	Pass
TC24	Service creator	verify if the page shows create workflow tile	goto service creator page		the page shows a tile named create workflow	As expected	Pass
TC25	Service creator	verify if the page shows import service tile	goto service creator page		the page shows a tile named import service	As expected	Pass
TC26	Service creator	verify if the page shows service create form when click on create service	click on create service tile		the page shows create service form	As expected	Pass
TC27	Service creator	verify if the back button shows initial page	click on back button of service create form component		the page hides the form and shows initial page component	As expected	Pass
TC28	Service creator	verify if the cancel button shows initial page	click on cancel button of service create form component		the page hides the form and shows initial page component	As expected	Pass
TC29	Service creator	verify if the create service form is submitable	click on create service tile goto form fill the details click on submit button	name=test_service_01 description=test service disabled SMS=this service is no longer available	the form shows an animation when click on submit button	As expected	Pass
TC30	Service creator	verify if the service create form shows an error message when an error is occurred	click on create service tile goto form fill the details click on submit button	name=test_service_01 description=test service disabled SMS=this service is no longer available	form shows an error message below submit button	As expected	Pass
TC31	Service creator	verify the form does not allow empty form fields to submit	click on create service tile goto form fill the details click on submit button	name= description= disabled SMS=	form shows errors below each input field	As expected	Pass
TC32	Service creator	redirect to service explore page on successful creation	click on create service tile goto form fill the details click on submit button	name=test_service_01 description=test service disabled SMS=this service is no longer available	page redirects to service explore page	As expected	Pass
TC33	Service creator	verify if the page shows API create form when click on create workflow	click on create workflow tile		the page shows create workflow form	As expected	Pass

TC34	Service creator	verify if the back button shows initial page	click on back button of workflow create form component		the page hides the form and shows initial page component	As expected	Pass
TC35	Service creator	verify if the cancel button shows initial page	click on cancel button of workflow create form component		the page hides the form and shows initial page component	As expected	Pass
TC36	Service creator	verify if the create workflow form is submitable	click on create API tile goto form fill the details click on submit button	name=test_api_01 description=test api version=1.0.0	the form shows an animation when click on submit button	As expected	Pass
TC37	Service creator	verify if the workflow create form shows an error message when an error is occurred	click on create API tile goto form fill the details click on submit button	name=test_api_01 description=test api version=1.0.0	form shows an error message below submit button	As expected	Pass
TC38	Service creator	verify the form does not allow empty form fields to submit	click on create API tile goto form fill the details click on submit button	name= description= version=	form shows errors below each input field	As expected	Pass
TC39	Service creator	redirect to API page on successful creation	click on create API tile goto form fill the details click on submit button	name=test_api_01 description=test api version=1.0.0	page redirects to API page	As expected	Pass
TC40	Service creator	shows a file browser popup when clicks on import service tile	click on import service tile		page shows a file browser popup	As expected	Pass
TC41	SMS history viewe	verify if the page shows a text field to search by MSISDN	goto sms history page		page shows a search field on top of the page	As expected	Pass
TC42	SMS history viewe	verify if the page shows an empty table on page load and has MSISDN, Incoming SMS, Response SMS, Received time, Sent	goto sms history page		the page shows an empty table with mentioned colums	As expected	Pass
TC43	SMS history viewe	verify if the uyser can type MSISDN into search	goto sms history page type into search field		user can type values into search field	As expected	Pass
TC44	SMS history viewe	verify if the search is performed when user has entered an MSISDn and hit enter key	goto sms history page type into search field press enter key		the page performs search and shows an animation before load data	As expected	Pass
TC45	SMS history viewe	verify if the tables shows data after search	goto sms history page type into search field press enter key		the tables shows data after search	As expected	Pass
TC46	SMS history viewe	verify if the tables shows no data label when no data is found for the	goto sms history page type into search field press enter key		the table shows a label "No data" when data is not found	As expected	Pass
TC47	Service library	verify if the user is redirected to service information page when click on a service tile	goto service library page click on a service tile		tse user is redirected to service information page	As expected	Pass

TC48	service informatio	verify if the service imformation page shows service details in left side bar as service name, description, status, disabled sms, published data	click on a service tile goto service information page		the page shows service details in the left side bar	As expected	Pass
TC49	service informatio	verify if the page shows actions in right panel	goto service information page		the page shows service action details on top of the right side panel as tiles	As expected	Pass
TC50	service informatio	verify if the action tile is highlighted when get clicked on	goto service information page click on an action tile		selected action tile is highlighted	As expected	Pass
TC51	service informatio	verify if the page shows keywords in right panel	goto service information page		the page shows action keywords details right below actions panel in the right side panel as tiles	As expected	Pass
TC52	service informatio	verify if the keywords tile is highlighted when get clicked on	goto service information page click on a keywords tile		selected keywords tile is highlighted	As expected	Pass
TC53	service informatio	verify if the page shows SMS responses in right panel	goto service information page		the page shows sms responses right below keywords panel in the right side panel as a table	As expected	Pass
TC54	service informatio	verify keywords tile refreshed and shows new data when action is changed	click on each action tile		keyword tile should be changed when the action is changed	As expected	Pass
TC55	service informatio	verify sms response table refreshed and shows new data when action is changed	click on each action tile		sms response table should be changed when the action is changed	As expected	Pass
TC56	service informatio	verify if the sms response table has response code, description, sms, and			sms response table shows mentioned columns	As expected	Pass
TC57	service informatio	verify page redirects back to service page when click on service button in left sidebar	click on service button		webpage navigates to services page	As expected	Pass
TC58	service informatio	verify if the page downloads an xml file when click on export button in left sidebar	click on export button		page downloads an xml file to downloads directory	As expected	Pass
TC59	service informatio	verify if the page shows a modal sidebar when click on settings button In left sidebar of service information	click on settings button		page slides in a modal sidebar from the left	As expected	Pass
TC60	service informatio	verify if the page shows a modal sidebar when click on settings button in an action tile	navigate to an action tile click on settings button		page slides in a modal sidebar from the left	As expected	Pass

TC61	service informatio	verify if the page shows a modal sidebar when click on settings button In keyword tile	navigate to a keyword tile click on settings button		page slides in a modal sidebar from the left	As expected	Pass
TC62	service informatio	verify if the page shows a modal sidebar when click on settings button in a sms row	navigate to the sms table click on settings button		page slides in a modal sidebar from the left	As expected	Pass
TC63	service informatio	verify if the service info form has filled with data	click on settings button in service info panel goto form		the service edit form should display data	As expected	Pass
TC64	service informatio	verify if the user can change service details	click on settings button in service info panel goto form change details click on submit button	name=test_service_01 description=test service disabled SMS=this service is no longer available	on successful data submit the panel will be closed and service info panel is refreshed	As expected	Pass
TC65	service informatio	verify if the service can be activated/diactivated	click on toggle button to enable/disable service	name=test_service_01 description=test service disabled SMS=this service is no longer available	on successful data submit the panel will be closed and service info panel is refreshed	As expected	Pass
TC66	service informatio	verify if the service edit form cannot be submitted when the form is incomplete	fill the service edit form and submit	name= description=test service disabled SMS=this service is no longer available	form displays errors under empty input fields	As expected	Pass

TC67	service informatio	verify if the add new action panel opens when click on add action	click on add action button		page shows a modal sidebar with a form	As expected	Pass
TC68	service informatio	verify if the add new keyword panel opens when click on add keyword button	click on add keyword button		page shows a modal sidebar with a form	As expected	Pass
TC69	service informatio	verify if the add new sms panel opens when click on add sms button	click on add sms button		page shows a modal sidebar with a form	As expected	Pass
TC70	service informatio	verify if the add action form can be submitted when is filled	click on add action button fill the form click the submit button	action=test_action api=test_api_01	when success, the panel will be closed and refreshed the actions panel	As expected	Pass
TC71	service informatio	verify if the add action form cannot be submitted when the form is incomplete	click on add action button fill the form click the submit button	action=test_action api=	the form shows errors under each incomplete form field	As expected	Pass
TC72	service informatio	verify if the form closes when click on close	click on close button below the form		the form panel closes	As expected	Pass
TC73	service informatio	verify if the add keyword form can be submitted when is filled	click on add keyword button fill the form click the submit button	firstkey=ACT regex=^\s+ACT\s+DEMO \s+$	when success, the panel will be closed and refreshed the actions panel	As expected	Pass
TC74	service informatio	verify if the add keyword form cannot be submitted when the form is incomplete	click on add keyword button fill the form click the submit button	firstkey= regex=^\s+ACT\s+DEMO \s+$	the form shows errors under each incomplete form field	As expected	Pass

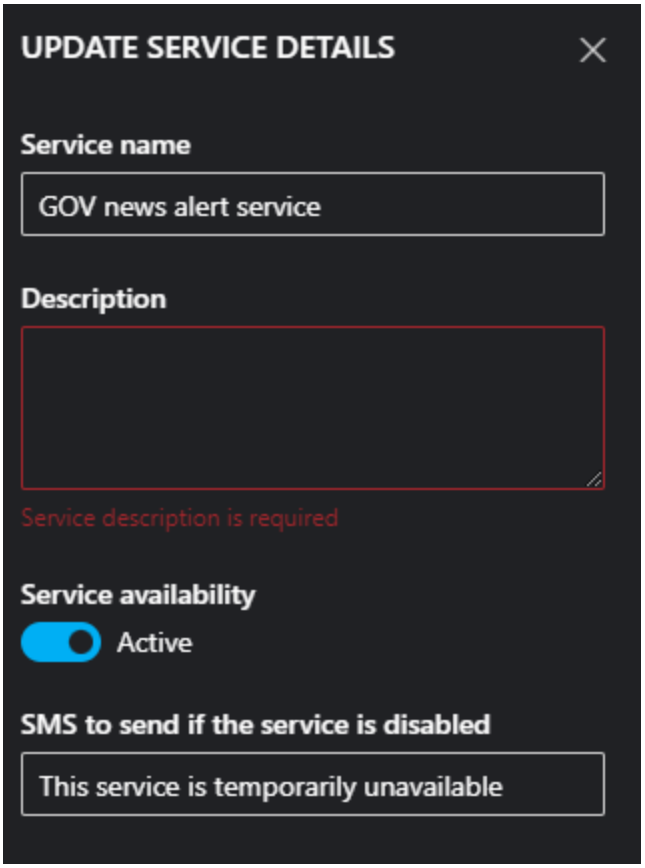
TC75	service informatio	verify if the add keyword form closes when click on close button	click on close button below the form		the form panel closes	As expected	Pass
TC76	service informatio	verify if the add sms form can be submitted when is filled	click on add sms button fill the form click the submit button	responseCode=1 desc=success sms=test sms	when success, the panel will be closed and refreshed the actions panel	As expected	Pass
TC77	service informatio	verify if the add sms form cannot be submitted when the form is	click on add sms button fill the form click the submit button	responseCode=1 desc= sms=test sms	the form shows errors under each incomplete form field	As expected	Pass
TC78	service informatio	verify if the add sms form closes when click on close button	click on close button below the form		the form panel closes	As expected	Pass
TC79	API details	verify if the page shows details in the left sidebar	navigate to api details page		the page shows api details in the left sidebar	As expected	Pass
TC80	API details	verify if the page shows api insights	navigate to api details page		the page shows api insights on top of the page right side	As expected	Pass
TC81	API details	verify if the page shows api change history	navigate to api details page		the page shows api change history right under the page right side	As expected	Pass
TC82	API details	verify if the page shows design tab	navigate to api details page		the page shows design tab on the top	As expected	Pass
TC83	API details	verify if the page shows details tab	navigate to api details page		the page shows details tab on the top	As expected	Pass
TC84		verify if the page shows api details edit panel	navigate to api details page		the page shows edit panel as a modal		
TC85	API details	verify if the edit panel closes when click close button	navigate to api details page click on settings button		the panel gets closed	As expected	Pass
TC86	API details	edit form does not allow to submit if the form is incomplete	navigate to api details page click on settings button fill the form	name= desc=	form shows errors owhen the form is incomplete	As expected	Pass
TC87	API details	edit form can be submitted when the details are ok	navigate to api details page click on settings button fill the form	name=abc desc=abc	form submits and closes the panel, refreshes the page	As expected	Pass
TC88	API details	verify if the api can be deployed	click on settings button on api version in api history table click on deploy button		api deployes and refresh the page to get deployed version to the editor, shows	As expected	Pass
TC89	API details	verify if the api can be checked out	click on settings button on api version in api history table click on checkout button		api checks out and refresh the page to get deployed version to the editor	As expected	Pass
TC90	API details	verify if the design page can be viewed	click on design tab		the page shows design view	As expected	Pass
TC91	API details	verify if the page shows available nodes	goto design tab		tha page shows nodes in the lefts idebar	As expected	Pass
TC92	API details	verify if the page shows the designer	goto design tab		the page shows graph designer	As expected	Pass
TC93	API details	verify if the nodes can be dragged into graph	goto design tab drag and drop node on to the graph		the graph shows dropped node	As expected	Pass

TC94	API details	verify if nodes can be connected	draw edges by draging a handle in a node and drop it on to a node		nodes are connected through edges	As expected	Pass
TC95	API details	verify if the edges can be removed	click delete button at the middle of a edge		the edge deletes	As expected	Pass
TC96	API details	verify if the nodes can be removed	select a node and press delete key		the node deletes	As expected	Pass
TC97	API details	verify if the properties form is shown when a node is double clicked	double click on a node		node properties panel shows	As expected	Pass
TC98	API details	verify if the node position can be changed	select and drad a node		node position updates along with edges	As expected	Pass
TC99	API details	verify if the graph shows api name, version, and commit ID			the graph shows mentioned details on top of the graph	As expected	Pass
TC100	API details	verify if the node details are shown when selected	select a node		the graph shows node details on top of the page, right	As expected	Pass
TC101	API details	verify if the graph lets you build it	click on build		graph shows popup with xml content	As expected	Pass
TC102	API details	verify if the graph can be saved	click on save fill he form click on submit		the form submits and new version checks out	As expected	Pass
TC103	API details	verify if the graph can be discarded	click on discard click on yes in opup		the graph is discarded	As expected	Pass
TC104	SMS mapping	verify regexes are match against keywords	send sms check log for matching details		if the matching is done, log shows transaction details	As expected	Pass
TC104	SMS mapping	verify regexes are match against keywords	send sms check log for matching details		if the matching is done, log shows transaction details	As expected	Pass
TC105		verify service is mapped	send sms check log for matching details		if the matching is done, log shows transaction details	As expected	Pass
TC106		verify response is sent	send sms check log for matching details		if the response is generated the service will sent an sms to the subscriber	As expected	Pass

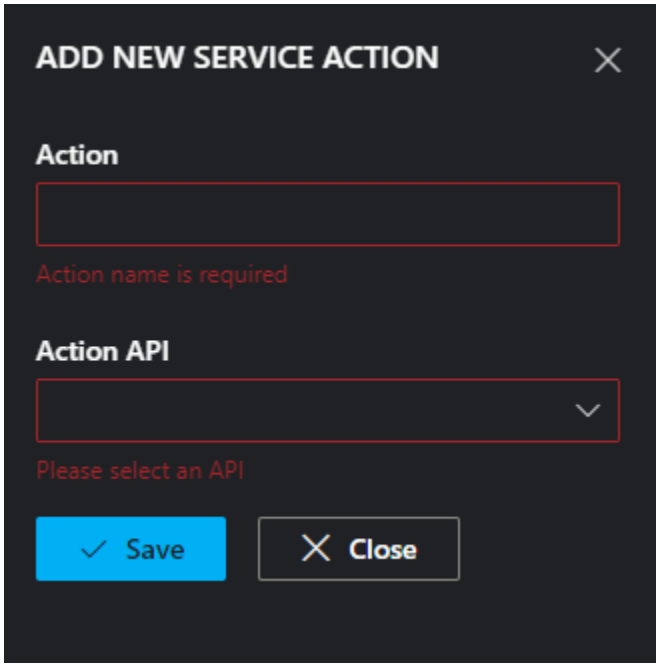
6.1.1 - Test cases and results

6.2. Unit testing

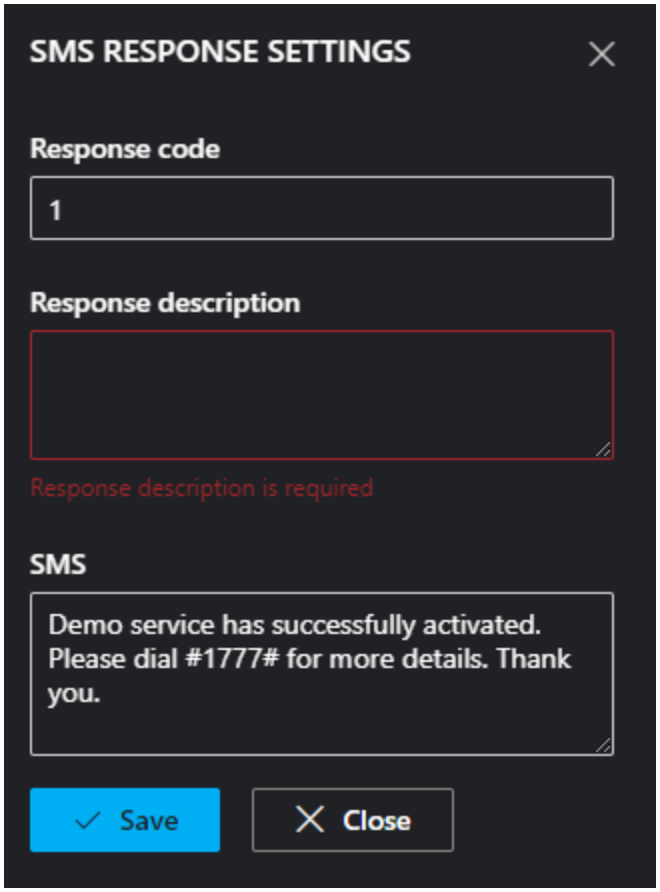
Below evidence shows screens of testings when forms are in invalid state.



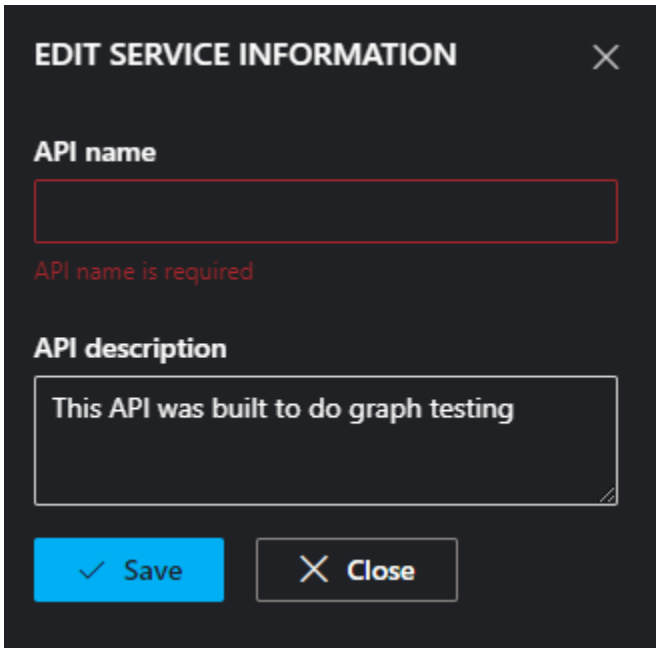
6.2.1 - Test case, should not be able to update when the provided form is in invalid state PASS



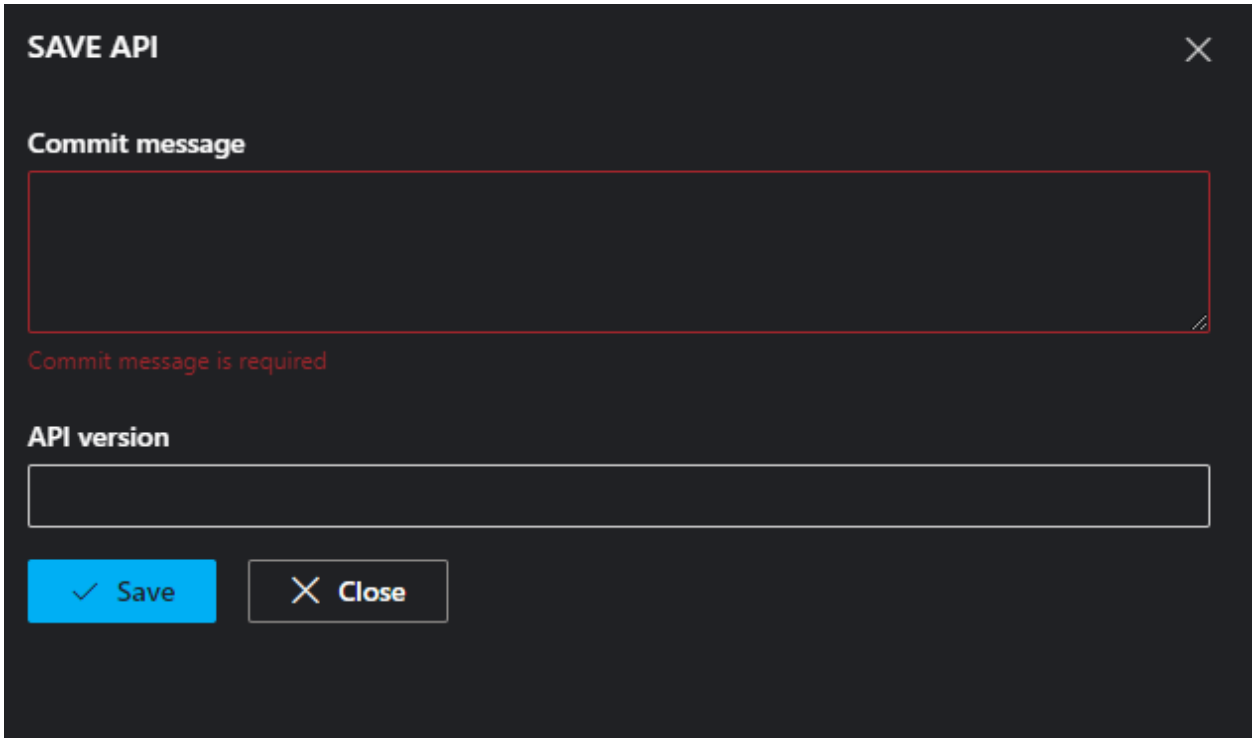
6.2.2 - Test case, should not be able to update when the provided form is in invalid state PASS



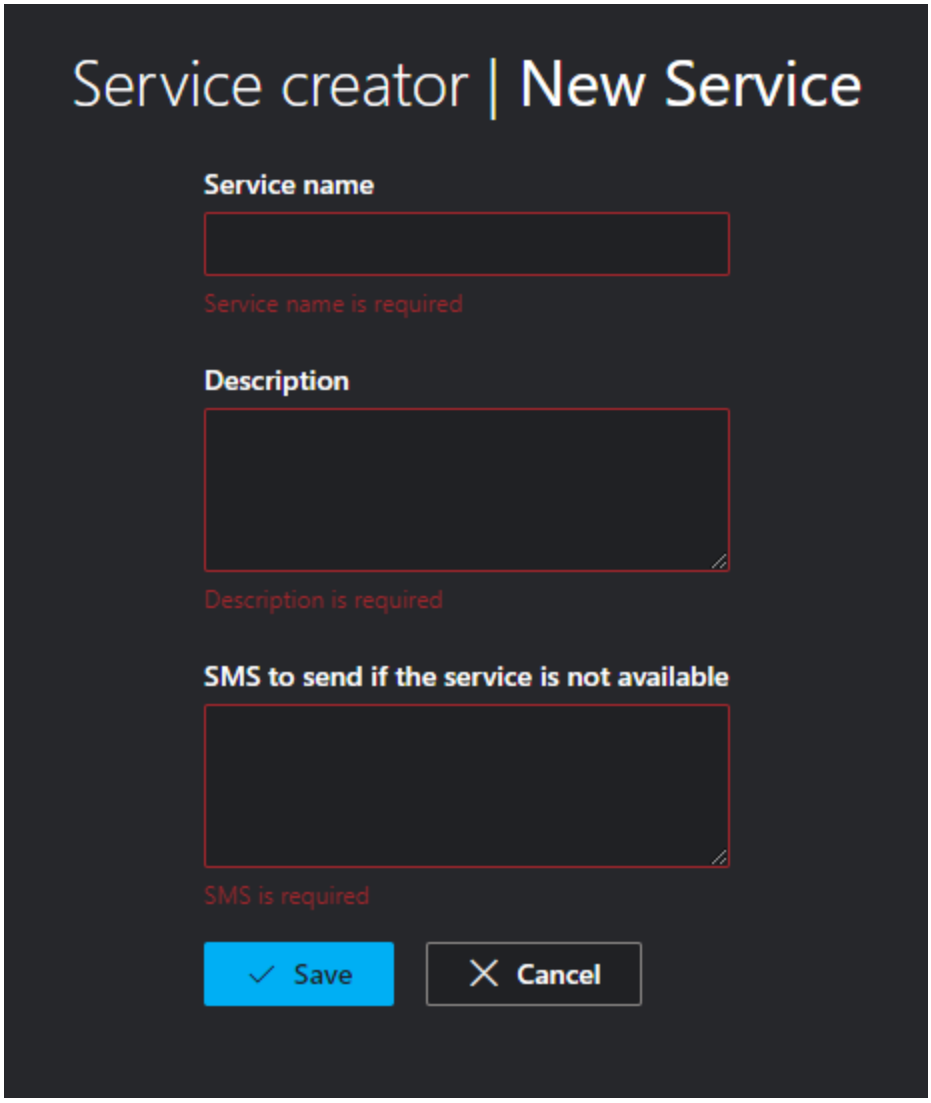
6.2.3 - Test case, should not be able to update when the provided form is in invalid state PASS



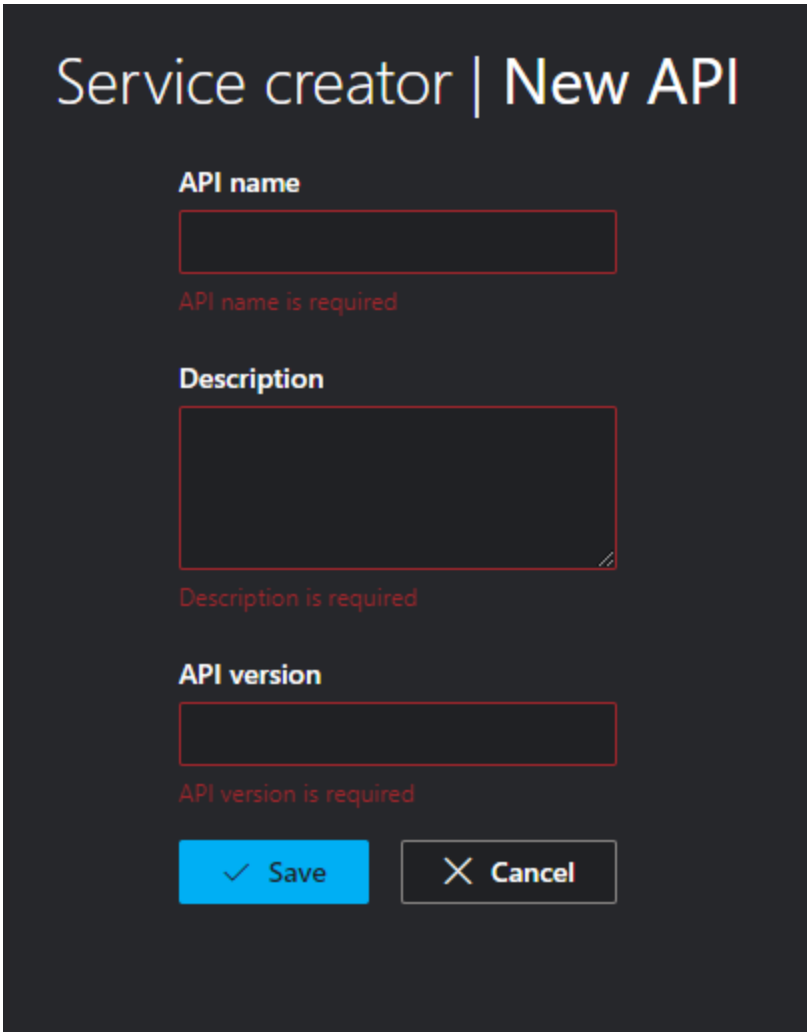
6.2.4 - Test case, should not be able to update when the provided form is in invalid state PASS



6.2.5 - Test case, should not be able to update when the provided form is in invalid state PASS



6.2.7 - Test case, should not be able to update when the provided form is in invalid state PASS



6.2.8 - Test case, should not be able to update when the provided form is in invalid state PASS

Results of JUnit unit testing are shown in the report below.

vas-management-api

Last Published: 2022-08-14 | Version: 1.0.0 vas-management-api

Project Documentation
- Project Information
- Project Reports
  - Surefire Report

Built by: maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
9	0	0	0	100%	3.853

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.sys.vas.management.service	9	0	0	0	100%	3.853

com.sys.vas.management.service

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	ActionServiceTest	2	0	0	0	100%	3.422
	ApiServiceTest	7	0	0	0	100%	0.431

Test Cases

[Summary] [Package List] [Test Cases]

ActionServiceTest

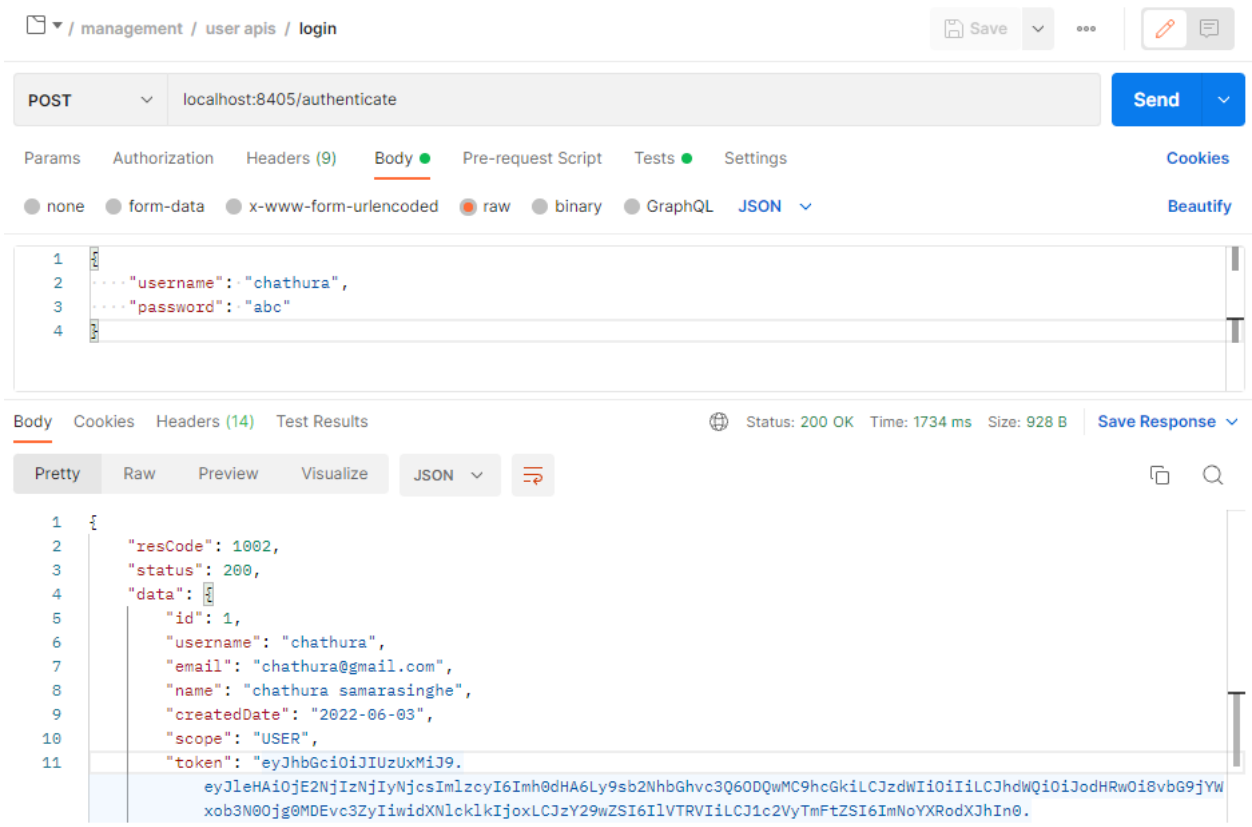
	should_throw_error_when_create_if_api_NotFound	0.096
	should_throw_error_when_id_null_on_update	0.019

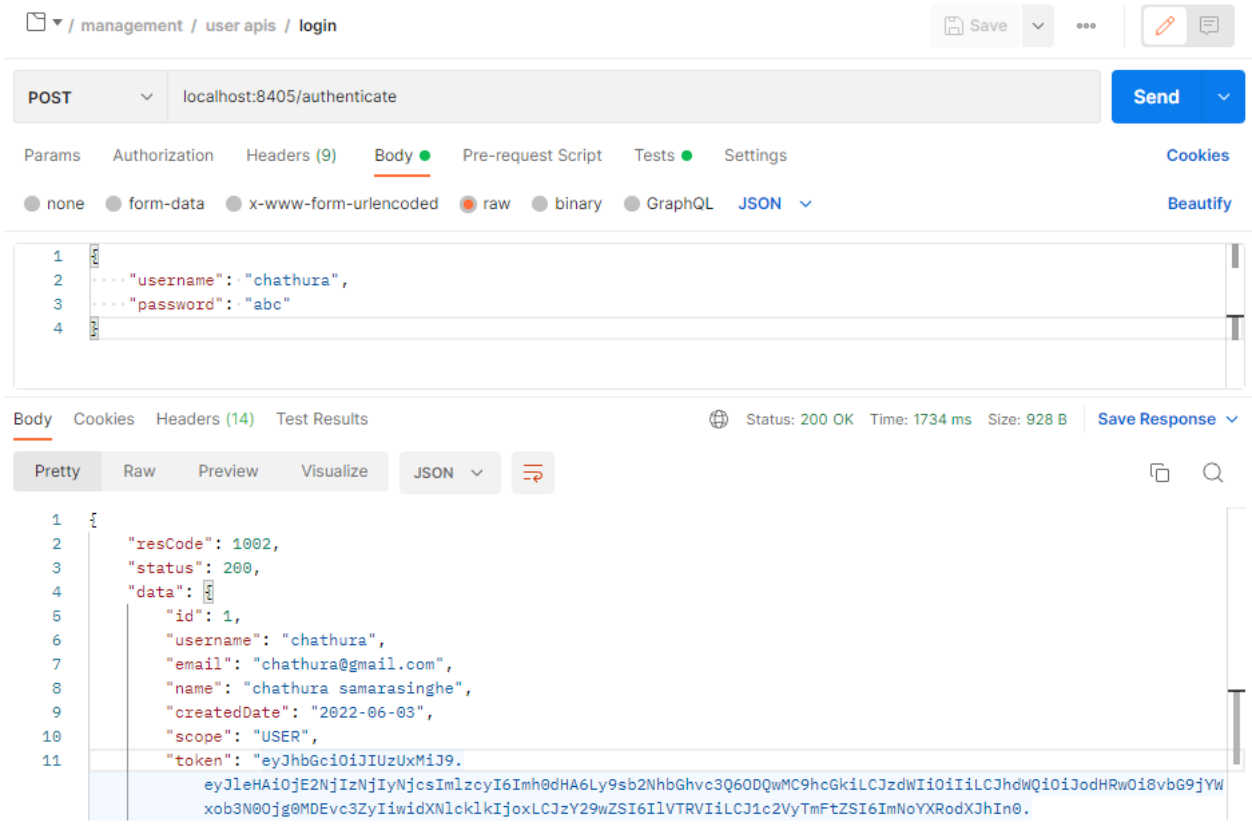
ApiServiceTest

	should_deploy_if_validId_and_commitID	0.025
	should_throw_error_when_create_if_api_NotFound	0.012
	should_return_list_if_data_exists	0.016
	should_throw_error_when_id_null_on_update	0.016
	should_create_when_valid_Api	0.016
	should_throw_error_if_apiid_notFound	0.017
	should_not_create_if_xml_is_null_or_empty	0.002

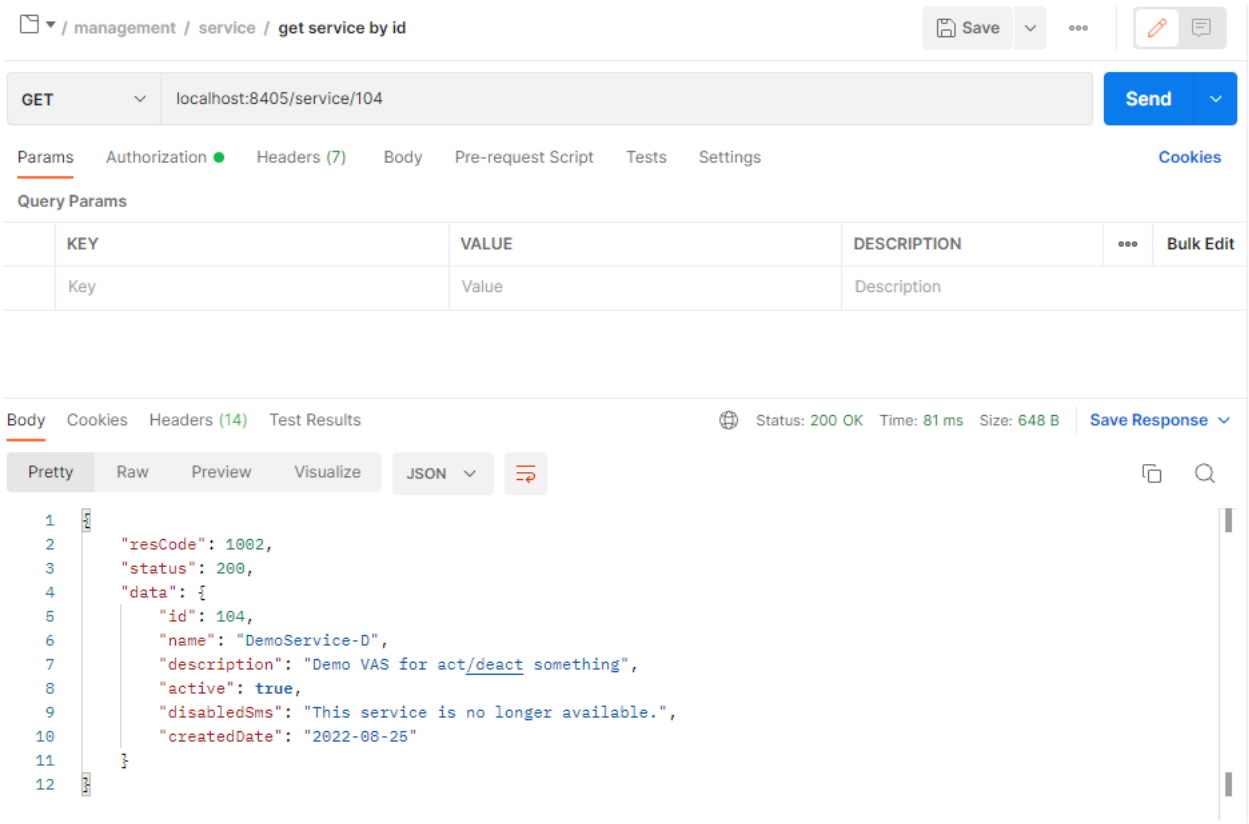
6.3. Integration testing

Integration testing has been carried out using the Postman tool to do full path testing of API endpoints in REST API.

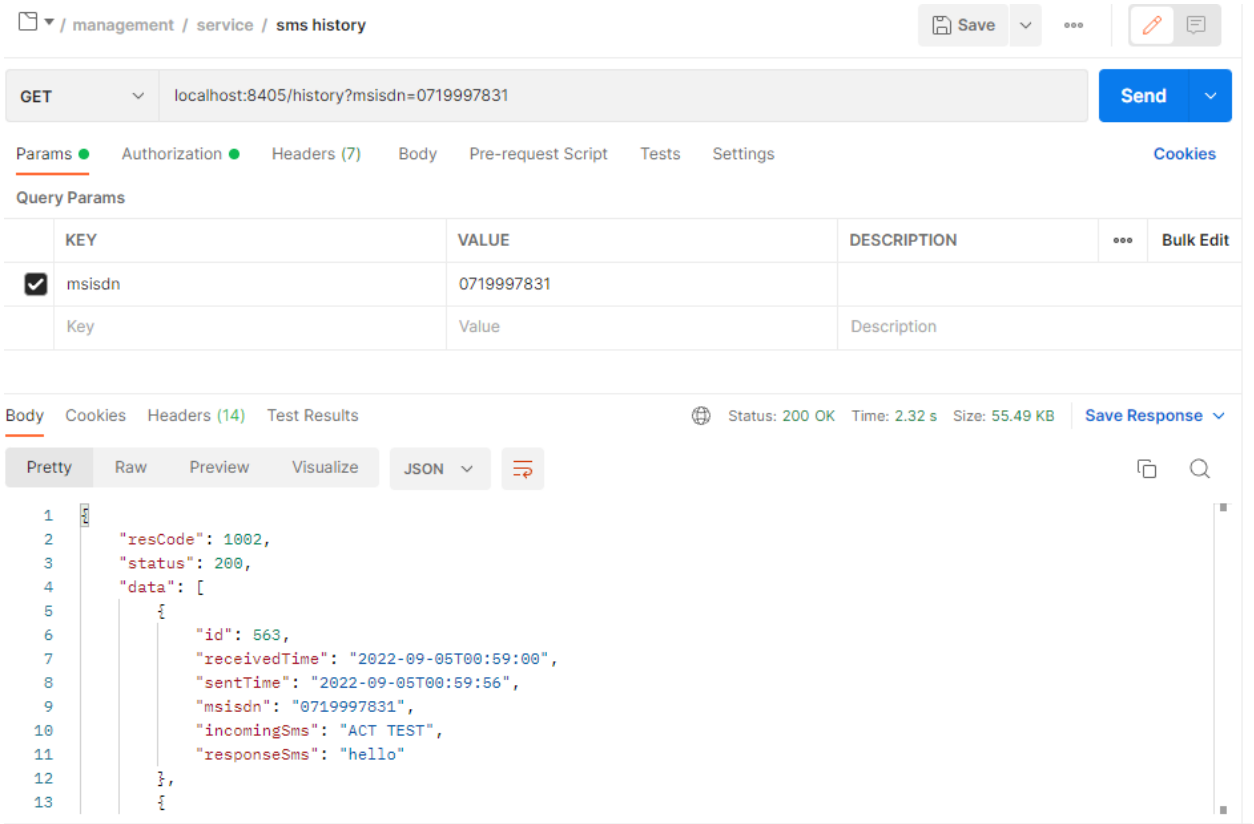

/ management / user apis / login
Save
POST
localhost:8405/authenticate
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username": "chathura",
"password": "abc"
}
Body
Cookies
Headers (14)
Test Results
Status: 200 OK
Time: 1734 ms
Size: 928 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"resCode": 1002,
"status": 200,
"data": {
"id": 1,
"username": "chathura",
"email": "chathura@gmail.com",
"name": "chathura samarasinghe",
"createdDate": "2022-06-03",
"scope": "USER",
"token": "eyJhbGciOiJIUzUxMiJ9.
eyJleHAiOjE2NjIzNjIyNjcsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6ODQwMC9hcGkiLCJzdWIiOiIiLCJhdWQiOiJodHRwOi8vbG9jYW
xob3N0Ojg0MDEvc3ZyIiwidXNlcklkIjoxLCJzY29wZSI6IlVTRVIiLCJ1c2VyTmFtZSI6ImNoYXRodXJhIn0.


/ management / user apis / login
Save
POST
localhost:8405/authenticate
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username": "chathura",
"password": "abc"
}
Body
Cookies
Headers (14)
Test Results
Status: 200 OK
Time: 1734 ms
Size: 928 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"resCode": 1002,
"status": 200,
"data": {
"id": 1,
"username": "chathura",
"email": "chathura@gmail.com",
"name": "chathura samarasinghe",
"createdDate": "2022-06-03",
"scope": "USER",
"token": "eyJhbGciOiJIUzUxMiJ9.
eyJleHAiOjE2NjIzNjIyNjcsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6ODQwMC9hcGkiLCJzdWIiOiIiLCJhdWQiOiJodHRwOi8vbG9jYW
xob3N0Ojg0MDEvc3ZyIiwidXNlcklkIjoxLCJzY29wZSI6IlVTRVIiLCJ1c2VyTmFtZSI6ImNoYXRodXJhIn0.

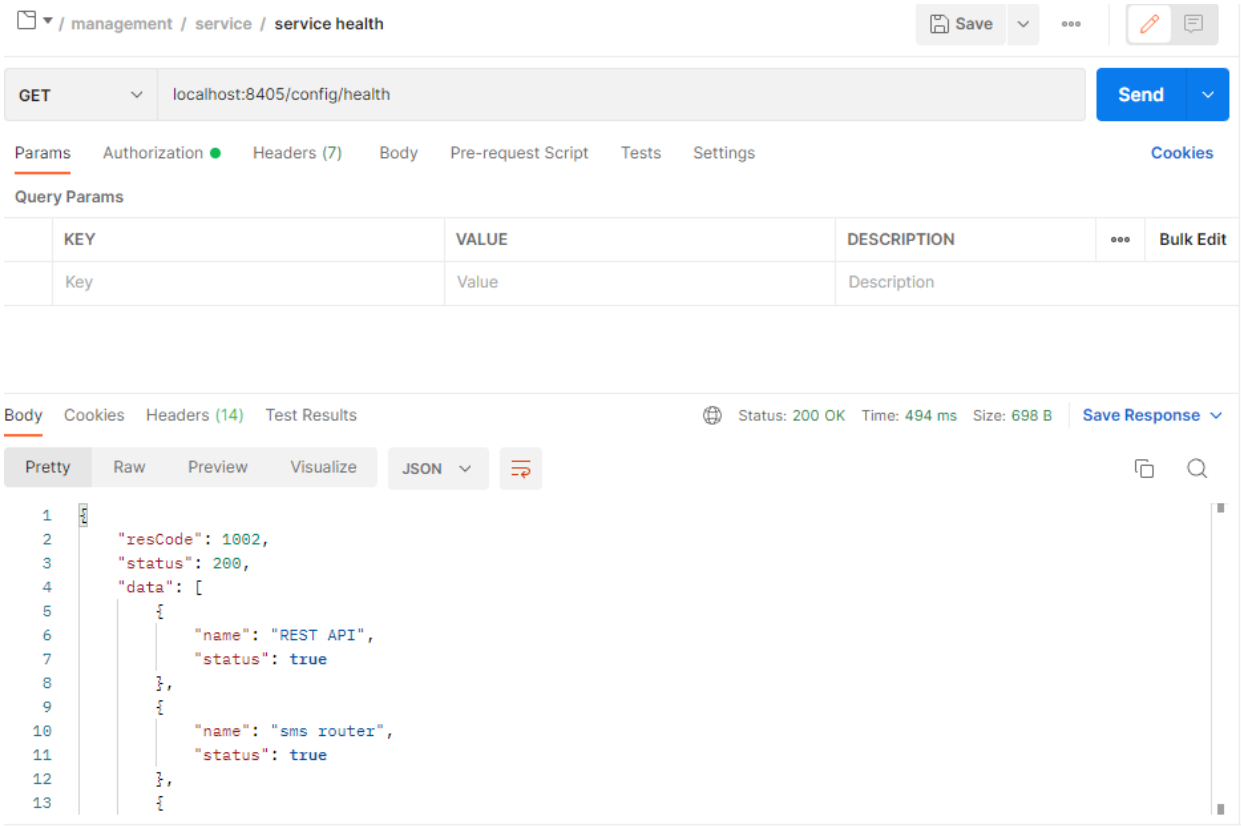
Test result 6.3.1 - Authenticate - Passed



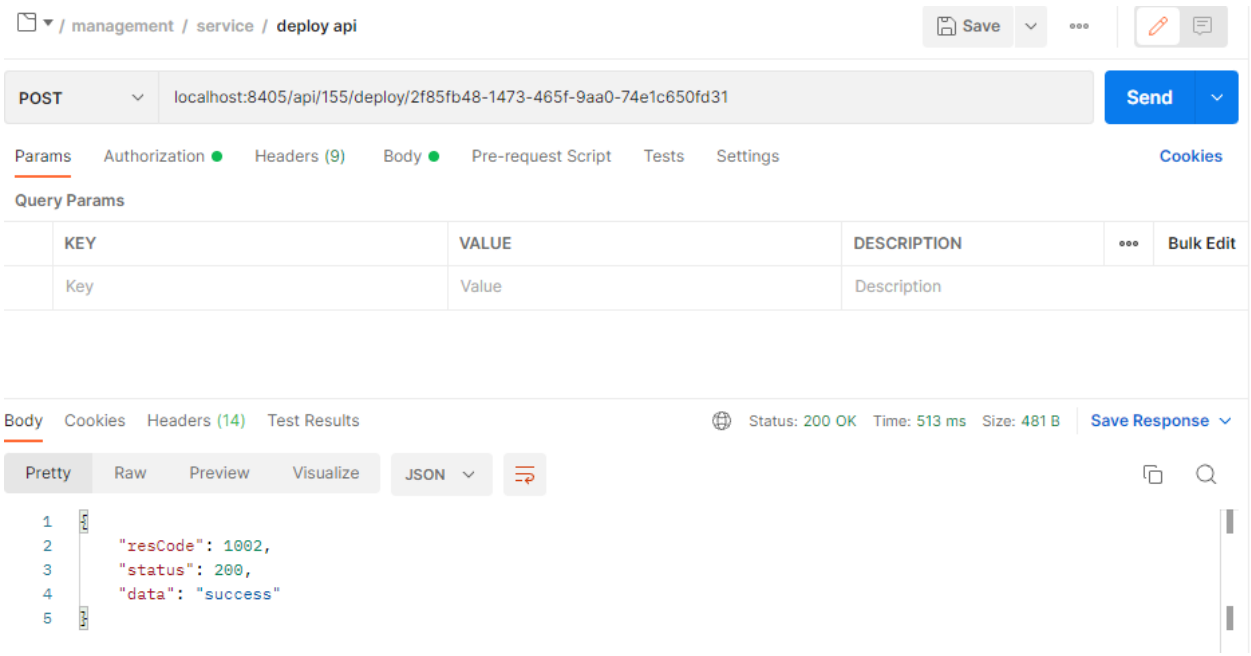
Test result 6.3.2 - Get service by ID - Passed



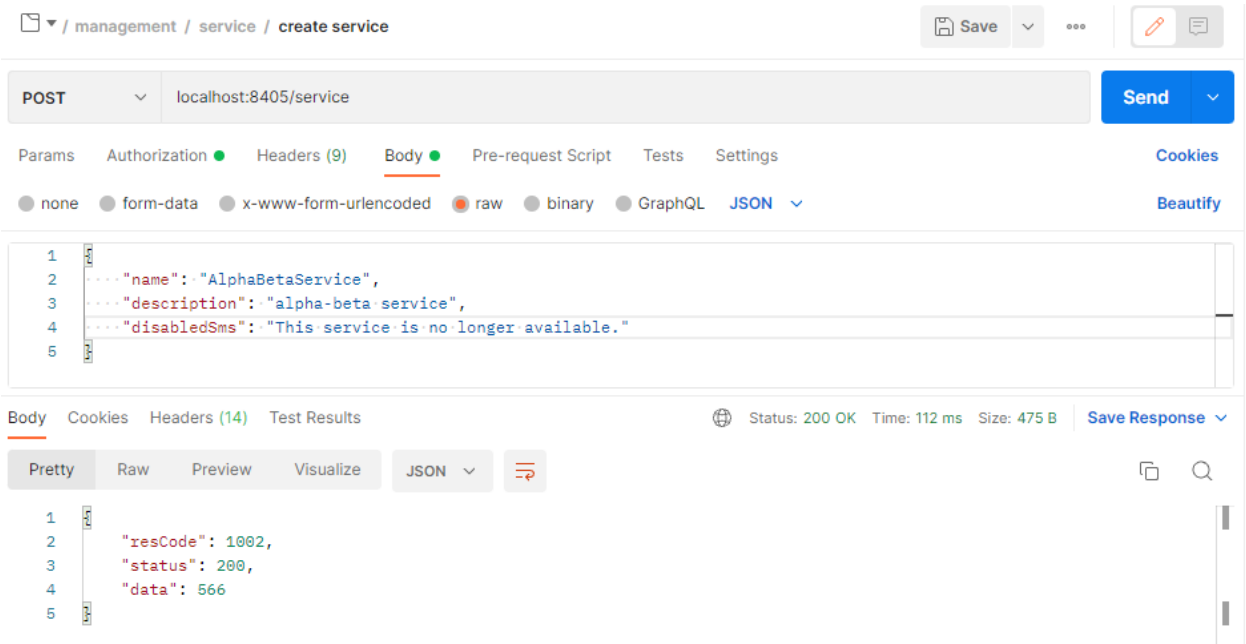
Test result 6.3.3 - SMS history search - Passed



Test result 6.3.4 - Service health - Passed



Test result 6.3.5 - Deploy API to the production - Passed

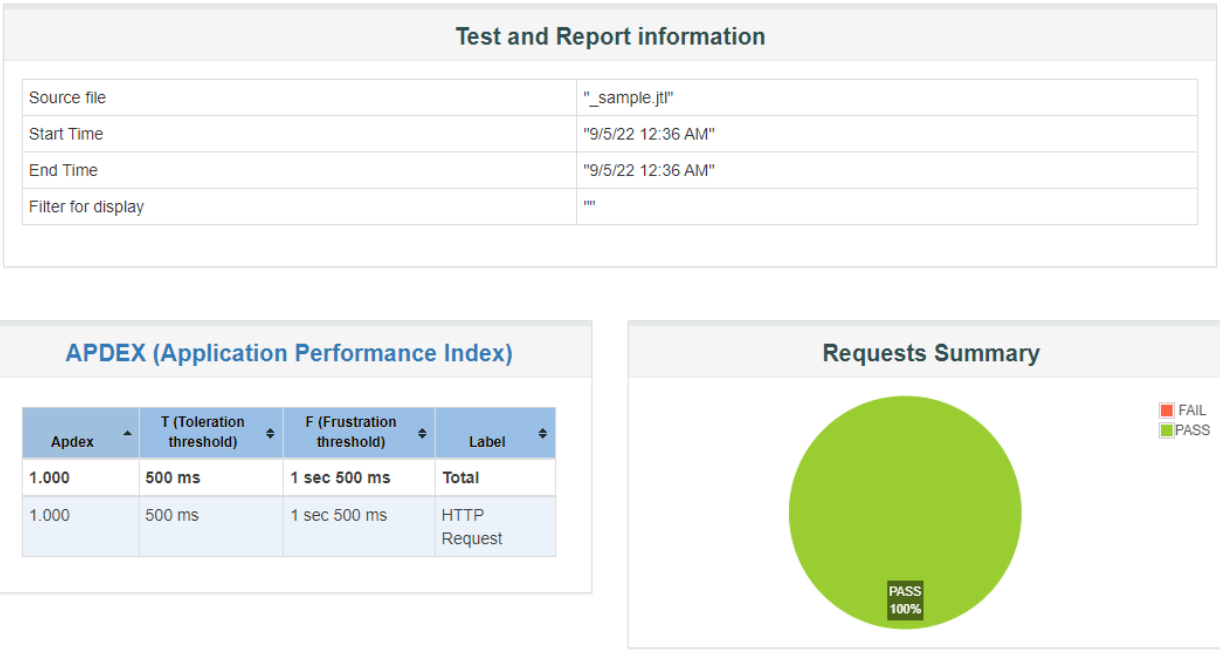


Test result 6.3.6 - Create service - Passed

6.4. Performance testing

For the performance and stress testing of the application JMeter is used. Below screens show the results of load testing.

Application Performance Index & Requests Summary.



Test and Report information

Source file	"_sample.jtl"
Start Time	"9/5/22 12:36 AM"
End Time	"9/5/22 12:36 AM"
Filter for display	""

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**500 ms**	**1 sec 500 ms**	**Total**
1.000	500 ms	1 sec 500 ms	HTTP Request

Requests Summary

6.4.1. Load test results of SMS router

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**100**	**0**	**0.00%**	**64.26**	**8**	**237**	**17.00**	**175.70**	**198.75**	**236.83**	**128.53**	**15.19**	**20.96**
HTTP Request	100	0	0.00%	64.26	8	237	17.00	175.70	198.75	236.83	128.53	15.19	20.96

Errors

Type of error	Number of errors	% in errors	% in all samples

Top 5 Errors by sampler

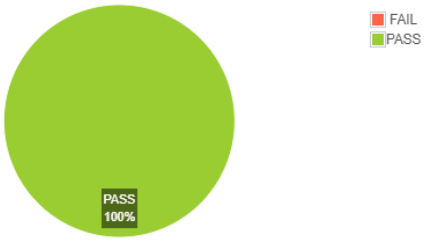
Sample	#Samples	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors
Total	**100**	**0**										

6.4.2. Load test results of SMS router

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.520	**500 ms**	**1 sec 500 ms**	**Total**
0.000	500 ms	1 sec 500 ms	authenticate
0.245	500 ms	1 sec 500 ms	get api
0.850	500 ms	1 sec 500 ms	get actions
0.985	500 ms	1 sec 500 ms	get api by commit

Requests Summary



6.4.3. Sample load test results of REST API

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**400**	**0**	**0.00%**	**2317.82**	**14**	**9887**	**633.50**	**6831.60**	**7863.70**	**9597.14**	**39.80**	**35.40**	**19.99**
authenticate	100	0	0.00%	6570.57	4350	9887	6222.00	8492.90	9401.95	9886.51	10.02	9.17	5.51
get actions	100	0	0.00%	450.81	15	3670	373.00	798.50	1239.20	3665.68	20.33	11.59	9.77
get api	100	0	0.00%	2105.36	20	4104	2215.50	3725.50	3992.00	4103.56	18.59	24.11	8.66
get api by commit	100	0	0.00%	144.54	14	783	54.50	378.10	472.35	781.19	78.93	61.12	40.47

Errors

Type of error	Number of errors	% in errors	% in all samples

Top 5 Errors by sampler

Sample	#Samples	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors
Total	400	0										

6.4.4. Sample load test results of REST API

7. Discussion

7.1 Security concerns

This system is deployed in an internal server environment of the clients' server infrastructure. Therefore, at the deployment and user acceptance test (UAT) phase, certain security best practices must be followed. Access control and privilege enforcement mechanisms must strictly be followed according to the guidelines provided by the internal security team in order to secure internal services from malicious, unwanted activities and unprecedented mistakes.

The system must provide a secure credential store to store service tokens. Service integrator microservice uses a credential store to keep access tokens, API keys, and passwords of certain external services such as API gateway, SMSC gateway, and payment gateway. These data must be stored in cryptographically secure storage. Also, in an event of a security breach or related concern, stored data must be invalidated and re-imported.

Personally identifiable data of customers/subscribers except the MSISDN should not append into logs files. Once the processing is completed, customers' data must be cleared from the session.
Since the proposed solution is a core framework for VAS management, it does not require authorizations and approvals from gateway systems at the development phase.

8. Conclusion and Future Work

In this project we have covered a few aspects such as modern software development tools and technologies, in-depth analysis of requirements, architectural analysis, and modern development trends. With the research carried out to identify appropriate design for the software, it is found that the best option to implement the system is to use the microservice architecture and event-driven approach to asynchronously communicate with each service.

For the development of the software Java based technologies are used, notably Spring framework to build each microservice. Technology stack review has been carried out to identify strengths of the Spring eco-system and Java based technologies.

Sub domain decomposition technique is used to identify different parts of the platform and develop them as individual services.

This system successfully introduced and demonstrated how low-code software building approaches can be adapted to build modern programs. One of the most important and critical parts of this system is the low-code support service API builder and deployment infrastructure. This can be used to build and deploy efficient REST APIs with ease and highly responsive for the changes.

Evaluation of the software has been performed to demonstrate that the software satisfies the requirements of the business. Various software testing techniques were used to carry out the test plan.

As for the future work, below mentioned points can be enhanced and introduced.

- Enhanced low-code builder to optimize the flow definitions.
- Add more plugins to the plugins library.
- Decouple service integrator to separate the low-code processing engine to build it as an independent service.
- Enhance UI features.
- Introduce more system monitoring tools.

References

Cloudflare (n.d.). What is Encryption? | Types of Encryption | Cloudflare. *Cloudflare*. [online] Available at: https://www.cloudflare.com/learning/ssl/what-is-encryption/.

CloudZero (n.d.). *Horizontal Vs. Vertical Scaling: How Do They Compare?* [online] www.cloudzero.com. Available at: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling#:~:text=Horizontal%20scaling%20(aka%20scaling%20out [Accessed 16 Aug. 2022].

Dan Arias (2018). *Hashing in Action: Understanding bcrypt*. [online] Auth0 - Blog. Available at: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/.

Git-scm.com. (2019). *Git - About Version Control*. [online] Available at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

Hamilton, T. (2019). *What is Software Testing? Introduction, Definition, Basics & Types*. [online] Guru99.com. Available at: https://www.guru99.com/software-testing-introduction-importance.html.

microservices.io. (n.d.). *Microservices Pattern: Messaging*. [online] Available at: https://microservices.io/patterns/communication-style/messaging.html.

Rabbitmq.com. (2019). *Messaging that just works — RabbitMQ*. [online] Available at: https://www.rabbitmq.com/.

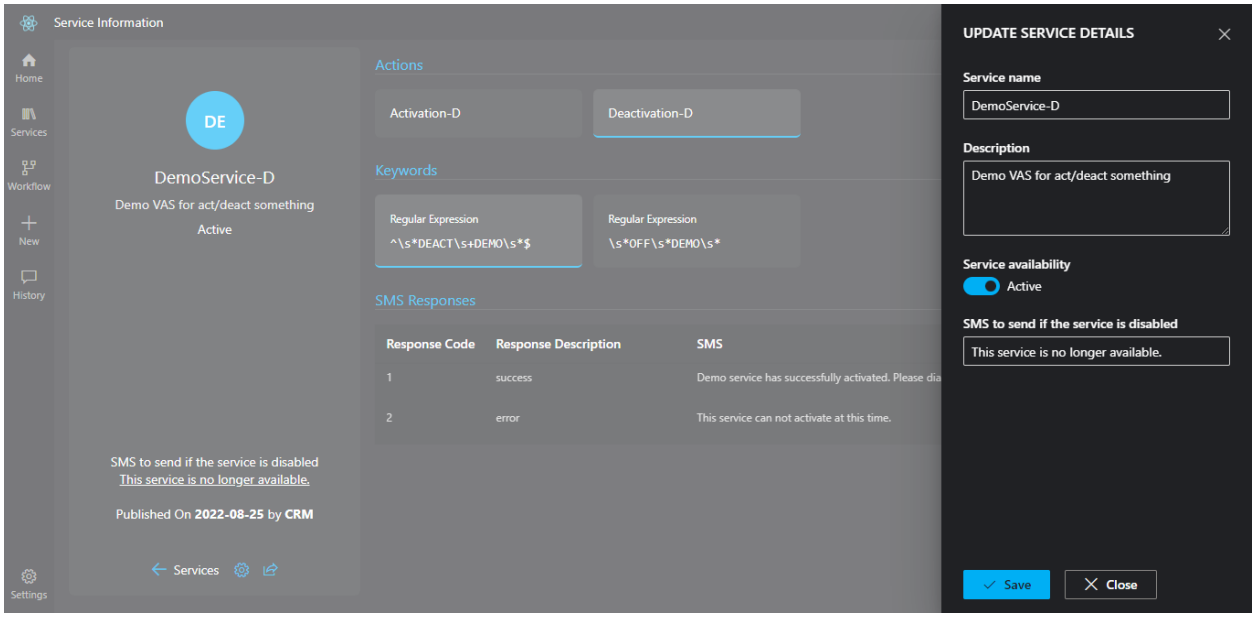
Spring.io. (2019). *spring.io*. [online] Available at: https://spring.io/.

The Economic Times. (n.d.). *What is Authentication? Definition of Authentication, Authentication Meaning*. [online] Available at: https://economictimes.indiatimes.com/definition/authentication.

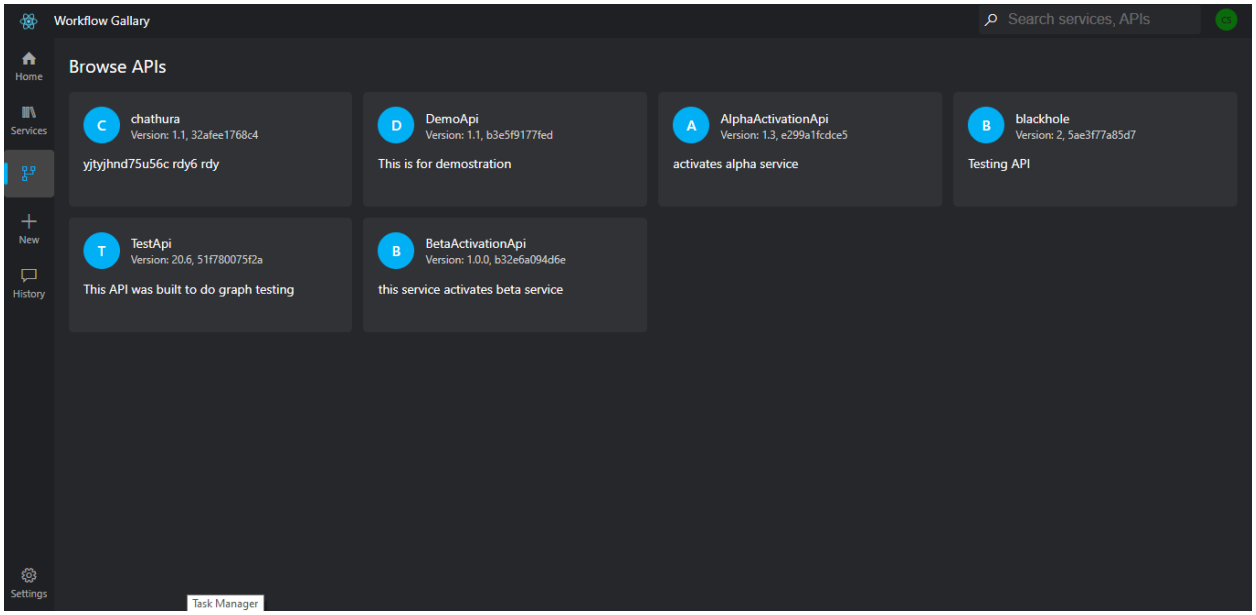
Wooldridge, B. (2022). *HikariCP It's Faster.Hi·ka·ri [hi·ka·'lē] (Origin: Japanese): light; ray.* [online] GitHub. Available at: https://github.com/brettwooldridge/HikariCP.

Appendices

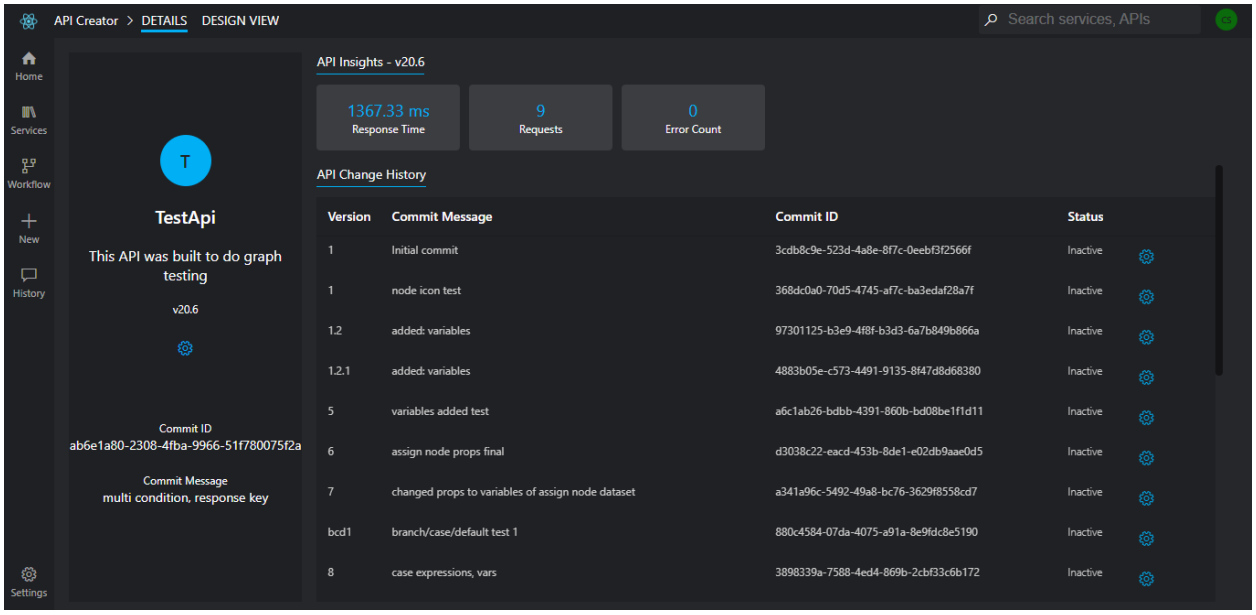
Appendix A - Web application user interfaces



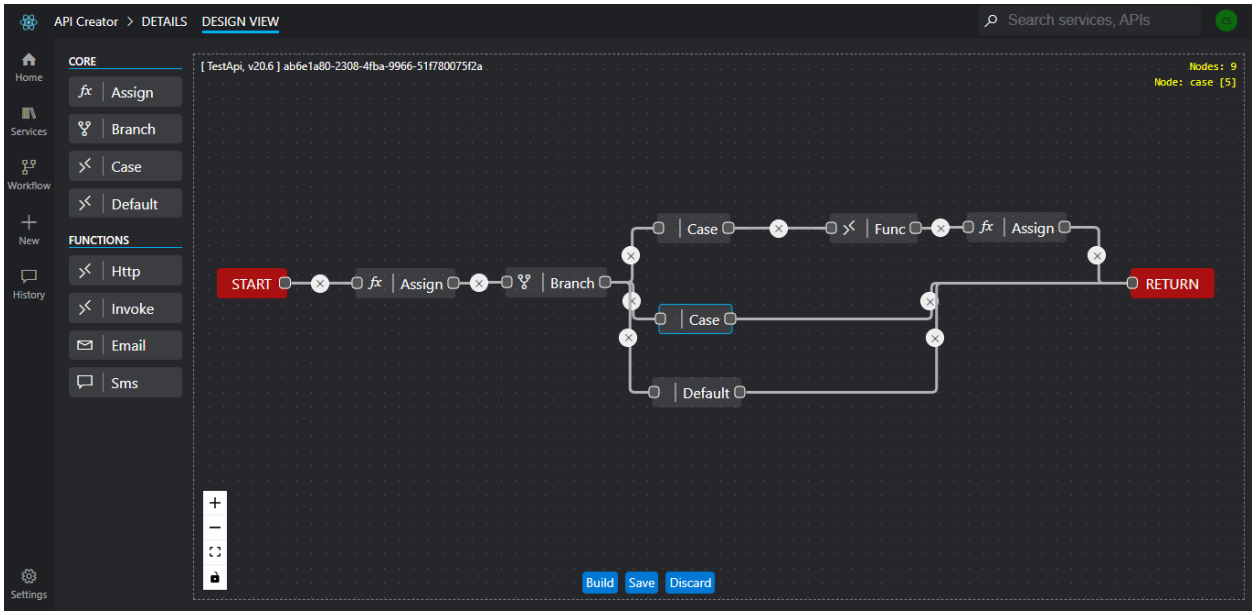
Appendix A - 1 Service information edit pane



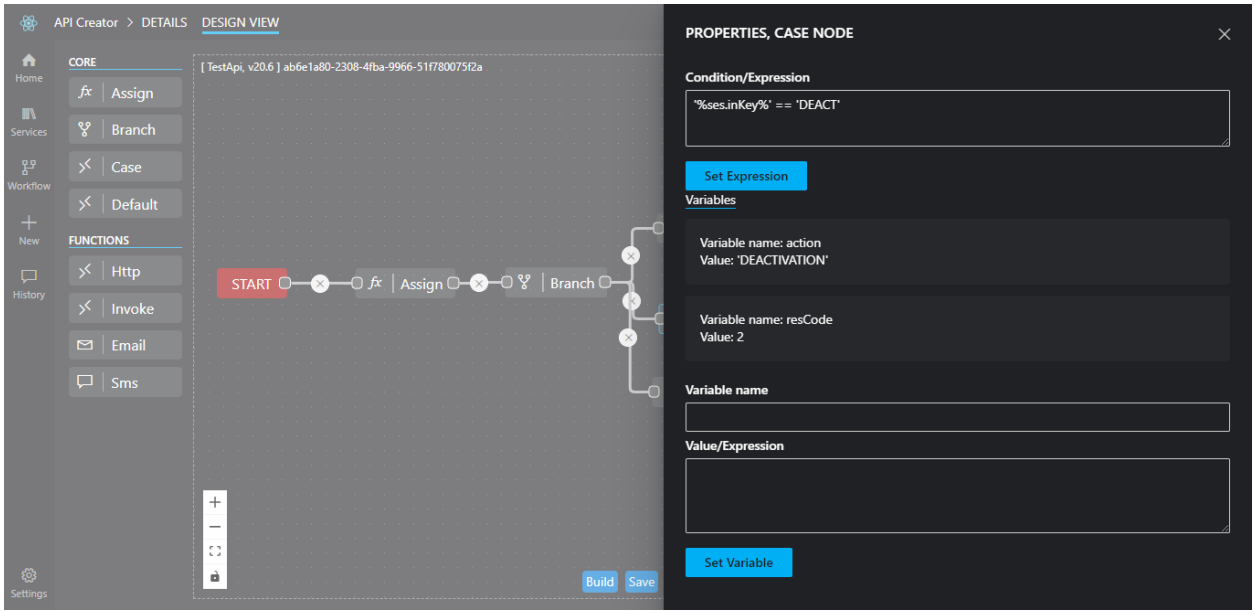
Appendix A - 2 API workflow list



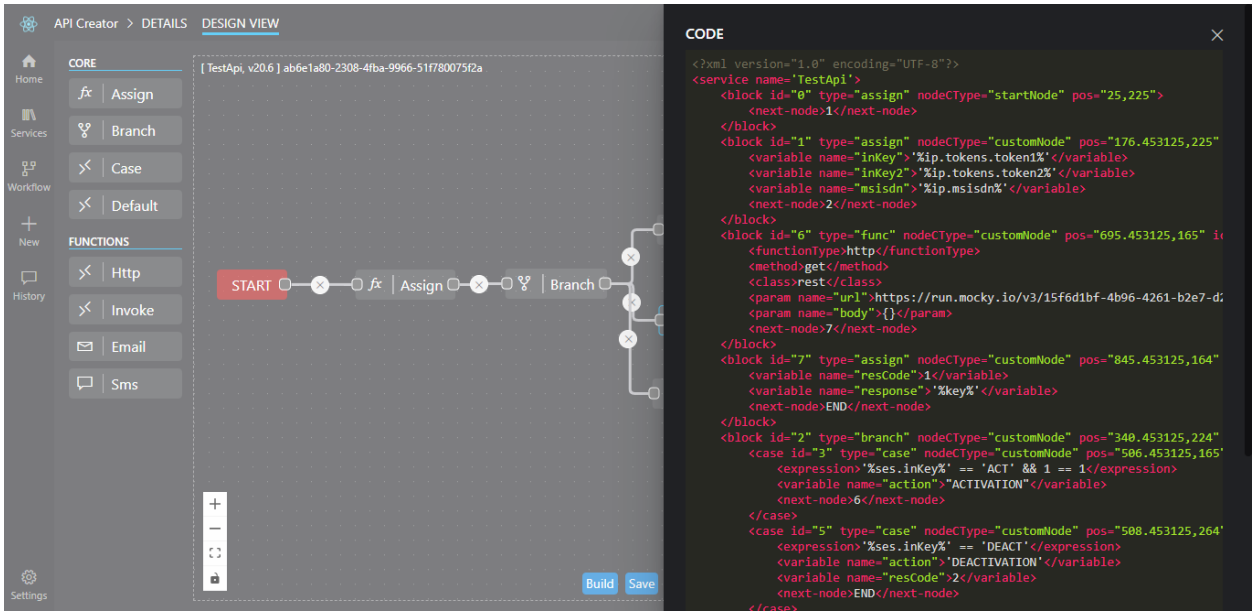
Appendix A - 3 API details



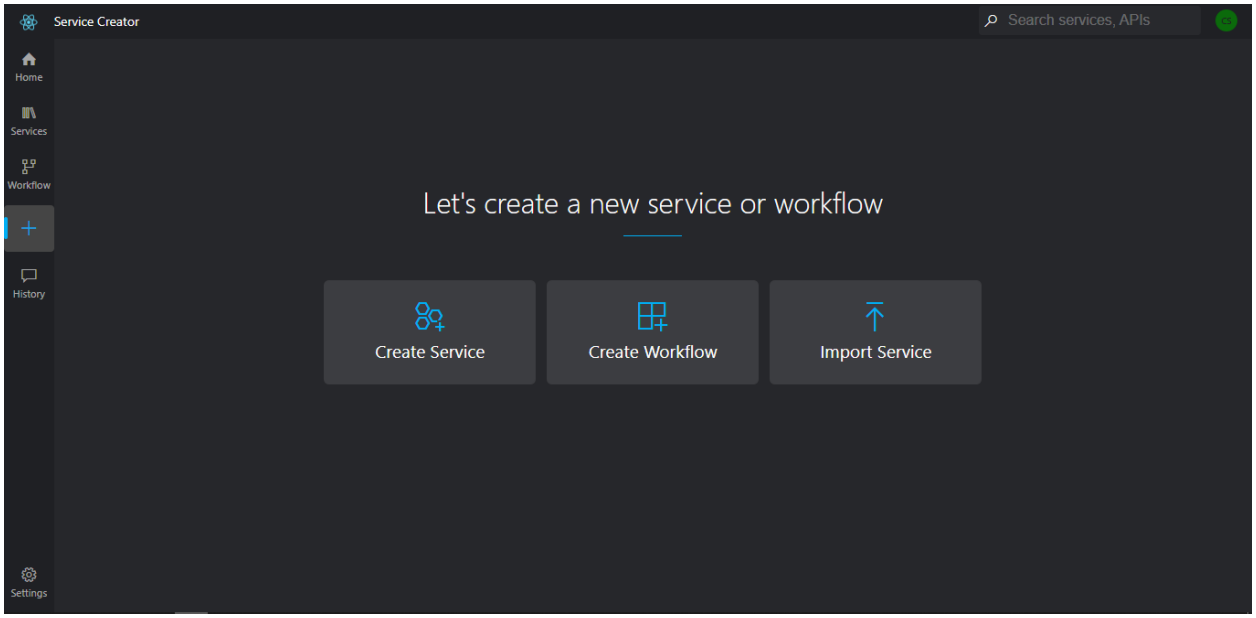
Appendix A - 4 API designer



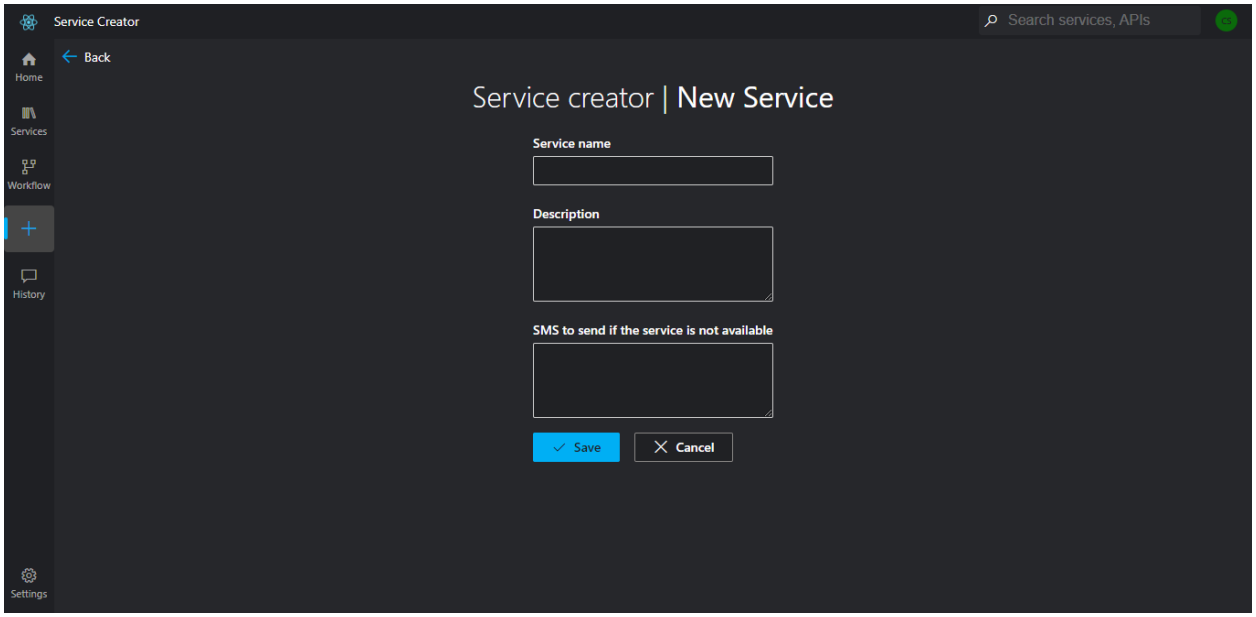
Appendix A - 5 Node editor



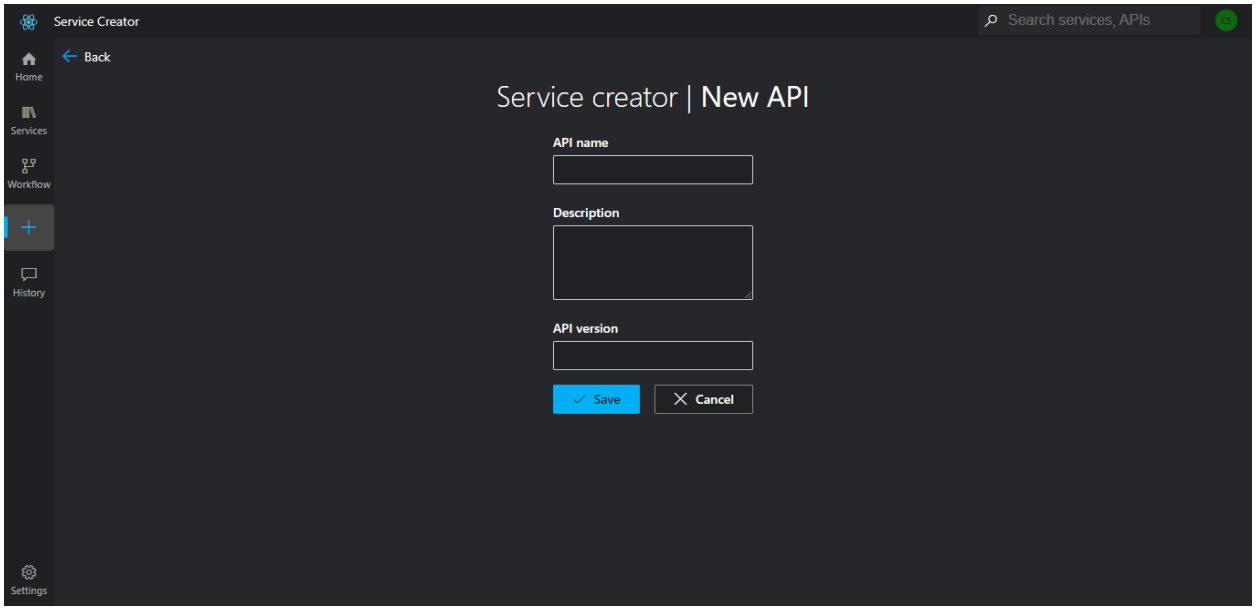
Appendix A - 6 API XML



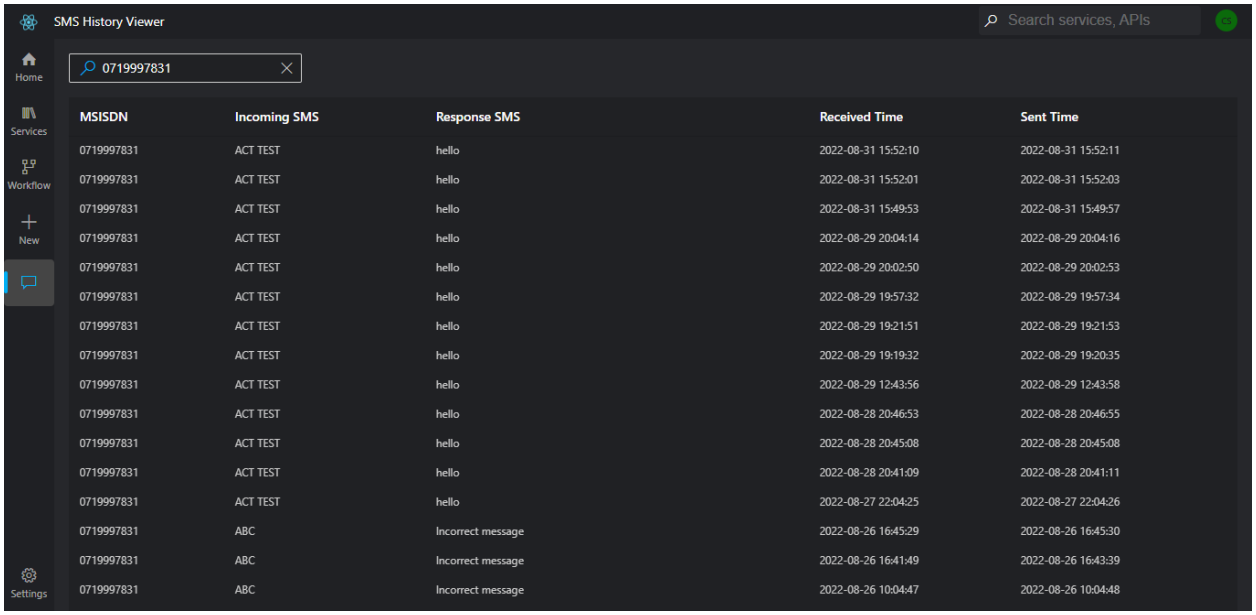
Appendix A - 7 Service creator



Appendix A - 8 new service



Appendix A - 9 new API



Appendix A - 10 SMS history