**Software Document**

SUE Data Science platform – SA-RB-01

*By: Mohammed Bouali, Calvin Kwan, Andrei Pieleanu, Björn Wildeman*

# Project Overview

**Introduction**

This document provides an overview of the application designed for creating and managing notebooks with dedicated persistent storage. The application is targeted at data scientists who require a scalable, cloud-based platform for interactive data analysis, model development, and experimentation. By simplifying the creation and management of notebooks, this service empowers data scientists to focus on their research without needing to handle underlying infrastructure.

**Purpose of the Application**

The application provides an API that allows users to create notebooks with persistent volume claims (PVCs) that store their data and configurations. It leverages a microservices architecture with event-driven communication to ensure scalability and reliability, particularly in high-demand environments.

**Target Audience**

The primary users of this application are **data scientists** who need access to a powerful, cloud-based environment for developing and running experiments. These users benefit from being able to create notebooks on demand, save their work securely, and access their data and code across sessions. Additionally, **IT and DevOps teams** may interact with the application to manage resources and monitor the infrastructure.

**Key Features**

1. **Notebook Creation and Management**: Users can easily create notebooks through the API. Each notebook is provisioned with the necessary storage and configuration to meet data scientists' requirements.
2. **Persistent Storage**: Notebooks are equipped with Persistent Volume Claims (PVCs) to store code, data, and results. This allows users to save their work and pick up from where they left off seamlessly.
3. **Event-Driven Architecture**: The application uses an event-driven design to communicate between services, ensuring reliability and responsiveness under load. Events are used to manage notebook creation, updates, and deletion in real time.

**Usage Guide**

**Getting Started**

1. **Authentication**: Before accessing the API, users need to authenticate. This process ensures that only authorized data scientists and administrators have access to the services.
2. **Creating a Notebook**: Once authenticated, users can create a new notebook by making an API call to the endpoint for notebook creation. They can specify resources (such as CPU and memory) and choose from available notebook types (e.g., Jupyter, RStudio).
3. **Accessing Notebooks**: After creation, the user receives the URL or endpoint to access their notebook. Persistent storage allows users to continue their work from previous sessions, as data is retained.
4. **Deleting a Notebook**: Users can delete notebooks they no longer need, freeing up resources. This operation removes the associated storage after a confirmation process, ensuring no accidental data loss.

**What is left for SUE to modify**

We have implemented JSON Web Token (JWT) support. Users can obtain a JWT by authenticating with their username and password through the API. This token is then used to authorize subsequent API requests, ensuring secure access to the application’s services.

However, the JWT implementation provided is designed to be adaptable to the company’s specific security requirements. While the current setup enables token generation and basic authentication workflows, the company may need to modify the implementation to align with their existing security protocols and infrastructure. For instance, details about token validation, expiration policies, and integration with other identity providers have been left flexible, allowing the company to adjust these aspects as needed. This approach ensures that the application remains compatible with the company’s security architecture while still providing a functional authentication layer.

Company-Specific Adjustments Required The following aspects of the JWT implementation require further customization by the company:

1. **Token Validation:** The company should define and implement specific rules for validating the JWT, including audience claims, issuer claims, and any other relevant fields.
2. **Expiration and Refresh Policies:** The current implementation provides basic token expiration functionality. The company must determine the appropriate expiration times and implement token refresh mechanisms if needed.
3. **Integration with Existing Security Infrastructure:** The application is designed to integrate with external identity providers or additional security layers. The company should ensure compatibility with their existing systems, such as Single Sign-On (SSO) or directory services.
4. **Access Control Policies:** Fine-grained access control, such as role-based access, has not been fully implemented. The company should extend the JWT implementation to include these policies if required.

**How to use the API**

prerequisites:

1. Go 1.23.0
2. Docker

Below is the guideline on how to run the application:

1. **Make sure docker is running.**
2. **Open the Command Prompt or PowerShell terminal in your project directory.**
3. **Run “.\Setup.bat”:**  This command performs the following:
   1. Setup the API gateway and RabbitMQ server
   2. Setup the databases and services
   3. Generate the gRPC code for the gRPC server from the existing proto file
   4. Retrieve the Kubeflow’s Kustomization deployment
4. **After “Kong” container is running successfully, run “.\Kong.bat”:** This command imports the configurations for the Kong Gateway
5. **Run “go mod tidy”:** This command adds missing dependencies to your go.mod file and removes any that are no longer used in your project.

**Kong docker containers explanation**

In the docker environment, there will be 3 docker containers for Kong gateway to be able to work properly. Below is the explanation for each of them:

* 1. Kong-db: Database to store Kong configurations and registered users
  2. Kong-migrations: Exports the database schema required by Kong gateway to Kong-db
  3. Kong: The Kong gateway that is used to route the received gRPC requests to the appropriate service with authentication.

**How to make API calls**

We used Postman to make the requests, feel free to use any other tool but this explanation will be done using Postman.

1. First import the proto files, this can be done by clicking on Service definition on postman and clicking on “import a .proto file”, here is a screenshot:  
   A screenshot of a computer

   Description automatically generated

Import the proto file of each service, token service, pvc service and notebook service.

1. Let’s use PVC service as an example, after clicking on the “import a .proto file” button, navigate to where you have saved the API in your local machine and then navigate to “pvc-service/api” , there you will find a pvc.proto file, import that. Below is a screenshot of how it should look like.

A screenshot of a computer

Description automatically generated

Do the same for the other services.

1. Once that is done you will have access to the API calls, in the screenshot below there is an example of how it should look like once you have successfully imported:  
     
   A screenshot of a computer

   Description automatically generated

Those are the main steps for setting everything up, from this moment on you can make API calls using “localhost:8000”, as that is the URL for the Kong API gateway. In the zip file you will also find a video tutorial of the app working, there you can see how the requests were made etc...

As it shows step by step, first by using the Token service to retrieve a JWT and then using that to make gRPC calls.

**Note:** You can use the “Use Example Message” on postman in order to have a default JSON message, then you can modify that. But keep in mind that for the gRPC call “CreateNotebook” you can remove everything and keep only the property “name”, it will still work, as the other properties will have a default setting, as agreed on with the stakeholder (Nathan Keyaerts).

Below is a screenshot of how the example default message would look like:  
A screenshot of a computer program

Description automatically generated

But you can remove everything except for “name” and it will still work just fine.