**AGI MONITORING API**

**Contents**

* [Introduction 3](#_Toc173751664)
* [Project File Structure 4](#_Toc173751665)
* [Data Flow 6](#_Toc173751666)
  + [Data Flow Diagram 7](#_Toc173751667)
  + [Login flow 8](#_Toc173751668)
  + [Access Data 9](#_Toc173751669)
* [API List 9](#_Toc173751670)
  + [Authentication 9](#_Toc173751671)
  + [Applications 10](#_Toc173751672)
  + [App Units 10](#_Toc173751673)
  + [Users 11](#_Toc173751674)
  + [Companies 11](#_Toc173751675)
  + [App Unit 12](#_Toc173751676)
* [Deployment 13](#_Toc173751677)

# Introduction

This documentation provides a detailed overview of the AGI MONITORING API application, including its structure, data flow, API endpoints, and instructions for building and deploying the application using Docker. The application uses SQLite3 as the database.

# Project File Structure

**AGI\_Monitoring\_API**/

├──bin/

│ ├── main.py

├── Config/

│ ├── .env

│ ├── config.py

│ ├── security.py

│ ├── schema.sql.py

├── db/

│ ├── agent-assist.db

├── setup/

│ ├── requirements.txt

├── src/

│ ├── controller/

│ │ ├── base/

│ │ │ ├── controllerBase.py

│ │ │ ├── types.py

│ │ ├── cacheController

│ │ │ ├── appCacheController.py

│ │ │ ├── sessionController.py

│ │ ├── appController.py

│ │ ├── applicationController.py

│ │ ├── companyController.py

│ │ ├── loginController.py

│ │ ├── userController.py

│ ├── logs/

│ ├── models /

│ │ ├── base /

│ │ │ ├── modelBase.py

│ │ ├── app\_manager.py

│ │ ├── audit\_log.py

│ │ ├── authManager.py

│ │ ├── company\_manager.py

│ │ ├── db\_manager.py

│ │ ├── user\_manager.py

│ ├── routers /

│ │ ├── base /

│ │ │ ├── routeBase.py

│ │ ├── app.py

│ │ ├── application.py

│ │ ├── company.py

│ │ ├── login.py

│ │ ├── user.py

│ ├── templates /

│ │ ├── config\_template.py

│ ├── utilities/

│ │ ├── settings.py

│ │ ├── utilities.py

├── .gitignore

├── Dockerfile

├── docker-compose.yml

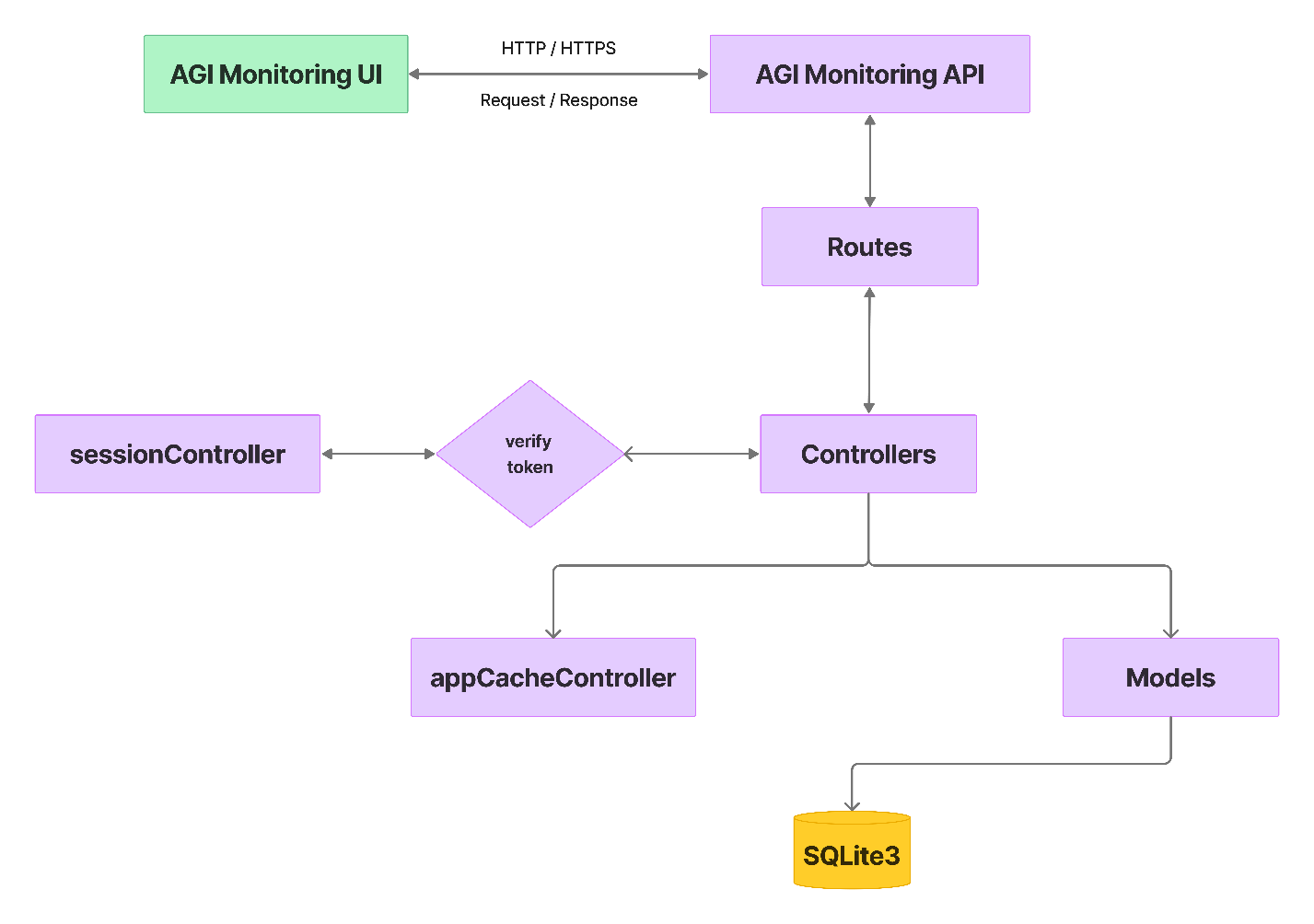
└── README.md

# Data Flow

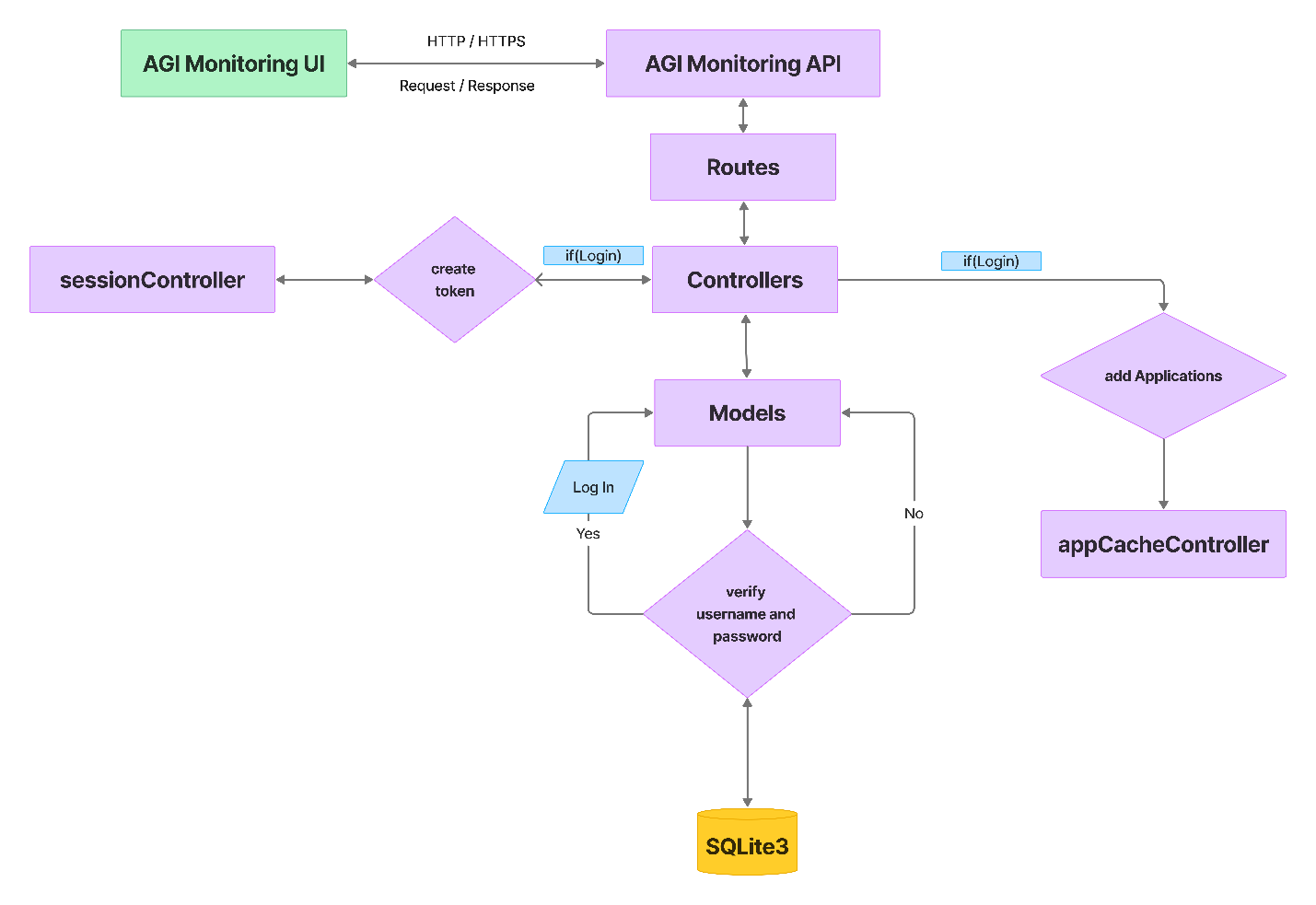
The data flow in the AGI monitoring Api ( FastAPI ) application follows below pattern:

* Request: A client sends a request to the API.
* Routing: The request is routed to the appropriate endpoint based on the URL and HTTP method.
* Controller: The route handler (controller) processes the request, interacts with the model if necessary, and prepares a response.
* Model: The model represents the data structure and handles database interactions.
* Response: The controller returns a response, which is sent back to the client side ui.

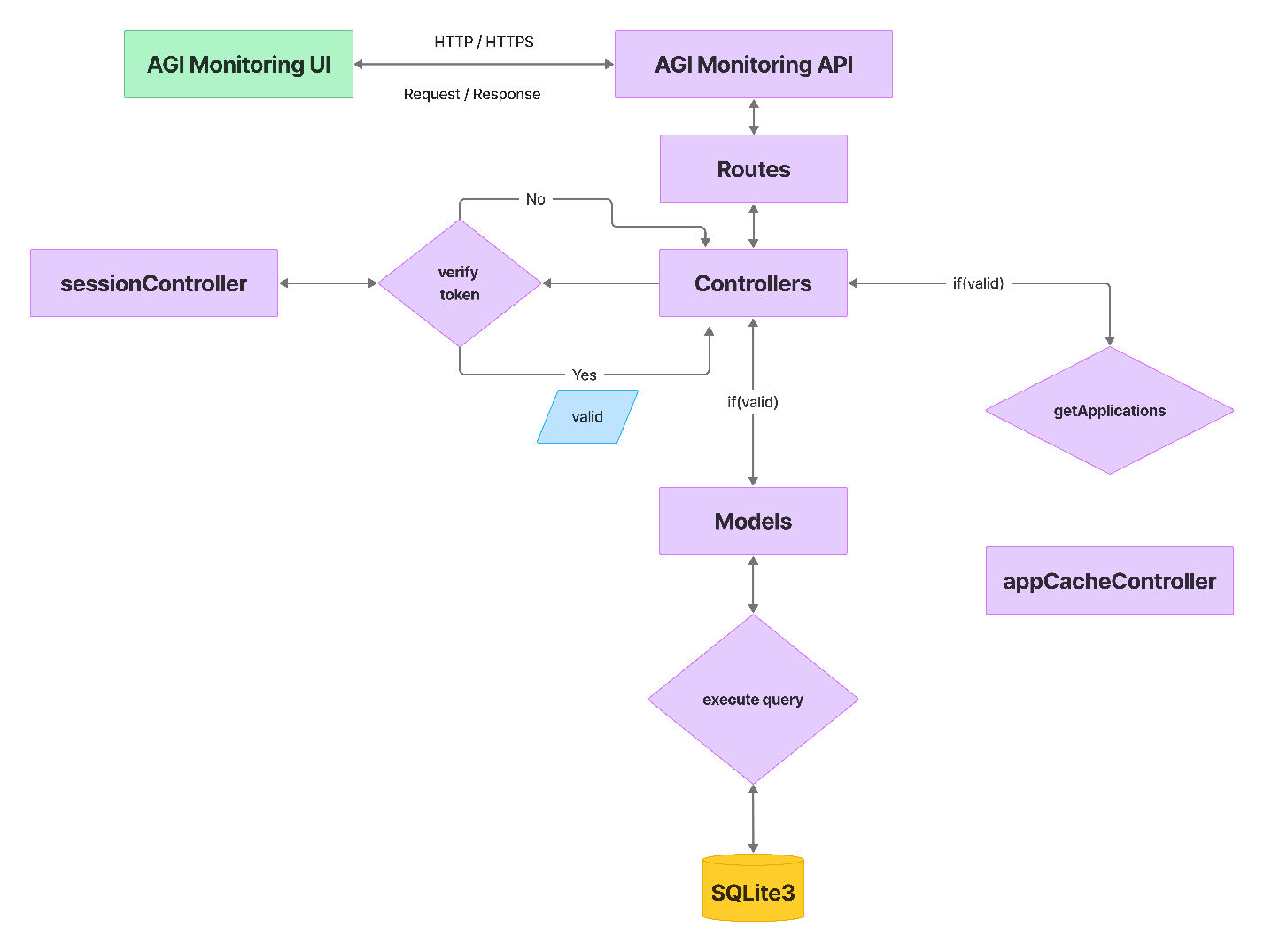
## Data Flow Diagram

****

## Login flow

****

## Access Data

****

# API List

## Authentication

1. POST /auth/login
   * Description: Authenticate a user and return a token
   * Request body: { username, password }
   * Response: { token, user\_info }
2. GET /api/auth/logout
   * Description: Invalidate the current user's token
   * Headers: Authorization: Bearer {token}

## Applications

1. GET / applications
   * Description: Retrieve all applications
   * Headers: Authorization: Bearer {token}
2. POST / applications
   * Description: Create a new application
   * Headers: Authorization: Bearer {token}
   * Request body: { name , ip , rest\_port , ws\_port , zid , key , desc , enable , cid , version}
3. PUT / applications/{aid}
   * Description: Update an existing application
   * Headers: Authorization: Bearer {token}
   * Request body: { name , ip , rest\_port , ws\_port , zid , key , desc , enable , cid , version}
4. DELETE / applications/{aid}
   * Description: Delete an application
   * Headers: Authorization: Bearer {token}

## App Units

1. GET / application / appUnits / { zid }
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
2. POST / application / appUnits / { zid }
   * Description: Create a new app
   * Headers: Authorization: Bearer {token}
   * Request body: { json\_data, file (**binary**) }
3. POST / application / appUnits / { zid } / { id } - update full app unit
   * Description: Update an existing app
   * Headers: Authorization: Bearer {token}
   * Request body: { json\_data, file (**binary**) }
4. PUT / application / appUnits / { zid } / { id } - update app unit without binary
   * Description: Update an existing app
   * Headers: Authorization: Bearer {token}
   * Request body: { ifname , path , enable , name , pool\_size , uname}
5. DELETE / application / appUnits / { zid } / { id }
   * Description: Delete an app
   * Headers: Authorization: Bearer {token}

## Users

1. GET / users
   * Description: Retrieve all users
   * Headers: Authorization: Bearer {token}
2. POST /users
   * Description: Create a new user
   * Headers: Authorization: Bearer {token}
   * Request body: { name, email, password, enable, cid, utid }
3. PUT / users/{id}
   * Description: Update an existing user
   * Headers: Authorization: Bearer {token}
   * Request body: { name, email, password, enable, cid, utid }
4. DELETE / users/{id}
   * Description: Delete a user
   * Headers: Authorization: Bearer {token}

## Companies

1. GET /company
   * Description: Retrieve all companies
   * Headers: Authorization: Bearer {token}
2. POST /company
   * Description: Create a new company
   * Headers: Authorization: Bearer {token}
   * Request body: { name, enable}
3. PUT /company/{id}
   * Description: Update an existing company
   * Headers: Authorization: Bearer {token}
   * Request body: { name, enable}
4. DELETE /company/{id}
   * Description: Delete a company
   * Headers: Authorization: Bearer {token}

## App Unit

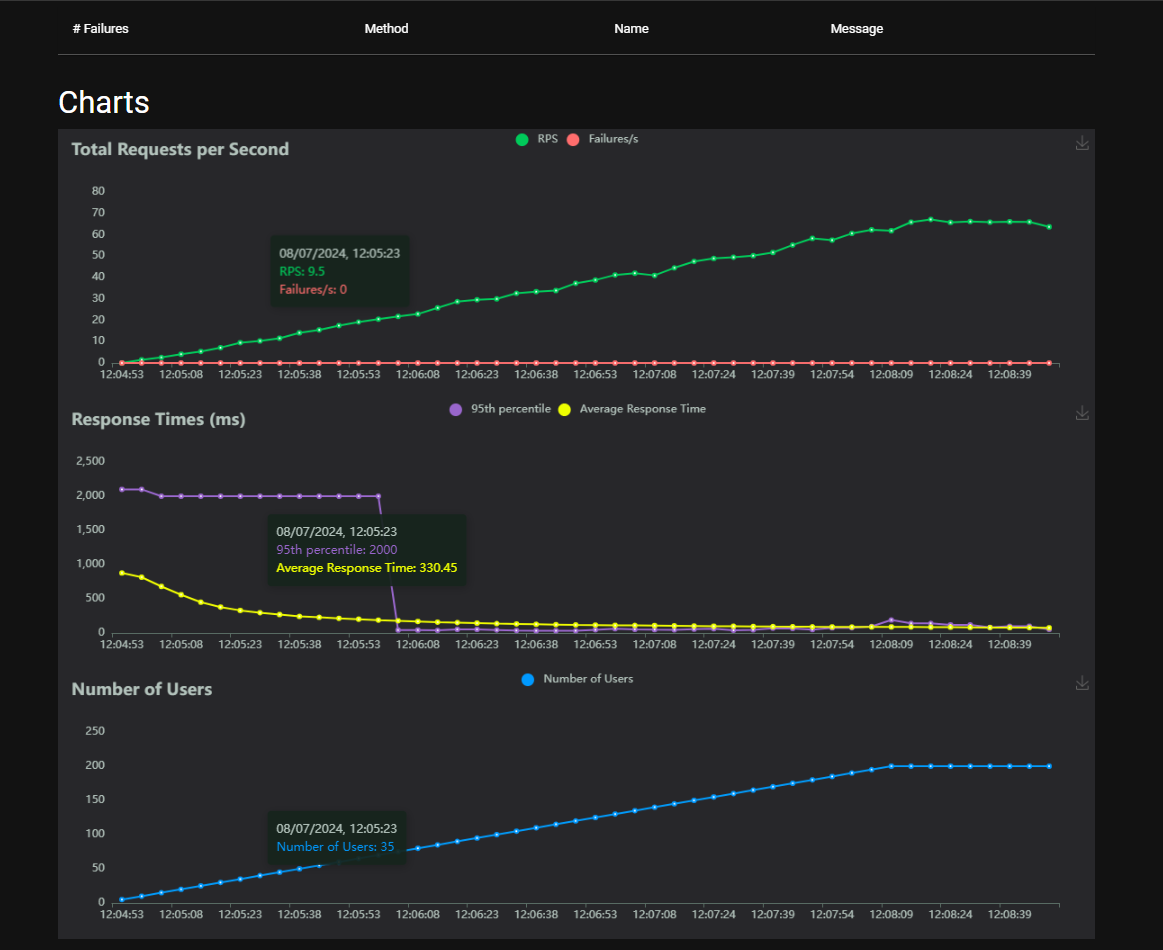
1. POST / app { aid } / info
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
2. POST / app { aid } / live
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
3. POST / app { aid } / status
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
4. POST / app { aid } / logs
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
5. POST / app { aid } / WSMonitor-start
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
6. POST / app { aid } / WSMonitor-stop
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
7. POST / app { aid } / config-reload
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }
8. POST / app { aid } / config-save
   * Description: Retrieve all app
   * Headers: Authorization: Bearer {token}
   * Request body: { ip, port }

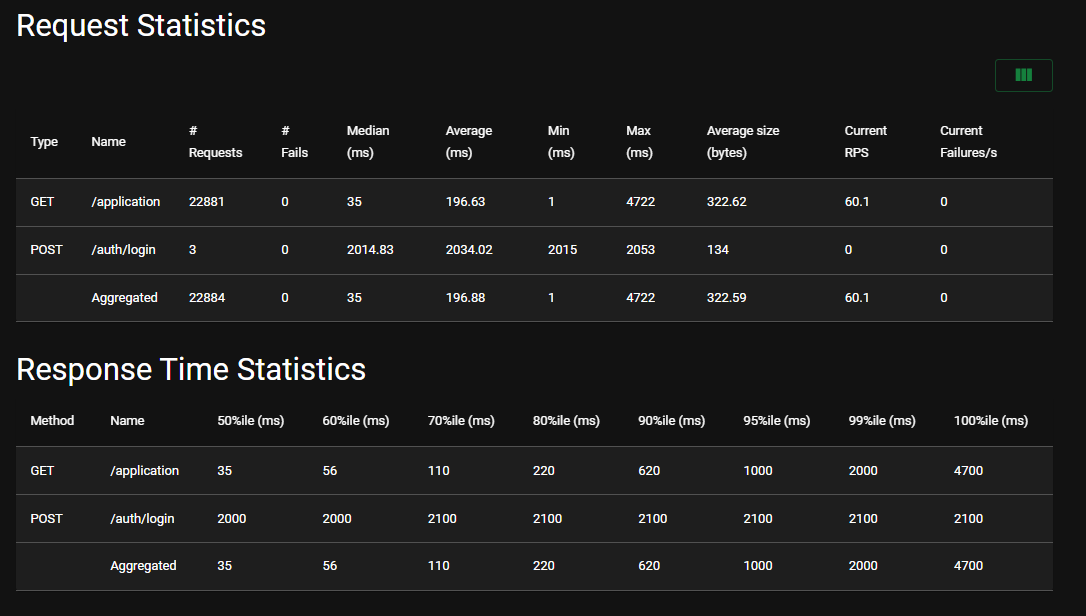
**Notes:**

* All endpoints except login should require authentication via a Bearer token in the Authorization header.
* Return appropriate HTTP status codes (200 for success, 201 for creation, 400 for bad requests, 401 for unauthorized, 403 for forbidden, 404 for not found, etc.).
* added more specific endpoints based on your application's requirements, such as endpoints for user roles, permissions, or specific actions related to applications, users, or company.

# Load Testing







# Deployment

Copy Project folder to the server. Then execute below commands.

docker build -t zaion-monitoringap .

docker run -d -p 8000:8000 --name zaion-monitoringapi monitoringapi