WireMock Project Documentation

1. Project Overview:

The objective of this project was to create a mock server using WireMock to simulate multiple endpoints across various services. The services involved are User, Product, Order, and Payment services. This setup allows for testing and development without the need for actual backend implementations, providing a controlled environment to test client applications.

2. Project Structure:

- **Directory Layout:**
  The project directory is organized to separate mappings and response files, following a standard structure recommended by WireMock. This organization helps in maintaining clarity and ease of access:

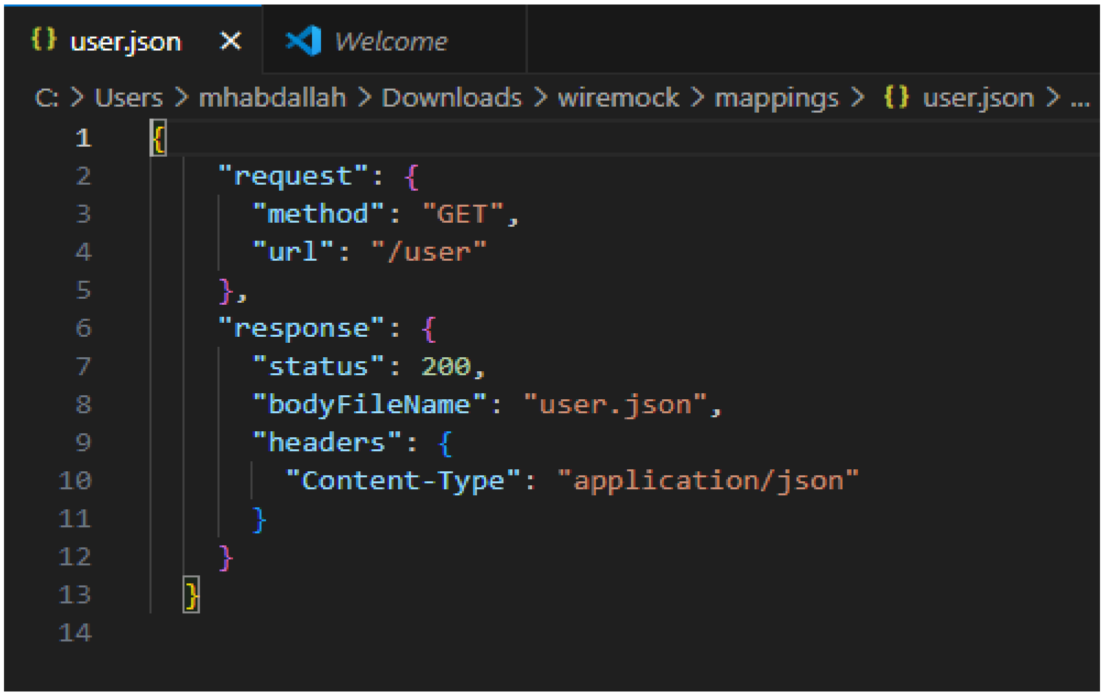
  Wiremock/: Root directory for WireMock setup.
  Mapping/: Contains all the mapping files defining the request-response stubs.
  __files/: Contains the JSON response files used by the mapping files.
  Wiremock-standalone-3.8.0.jar: The WireMock standalone JAR file used to run the server.

- **Services and Endpoints:**
  Each service (User, Product, Order, Payment) includes multiple endpoints to handle common CRUD operations. Here is a summary of the endpoints created for each service: GET, POST, PUT, DELETE.

3. Approach:

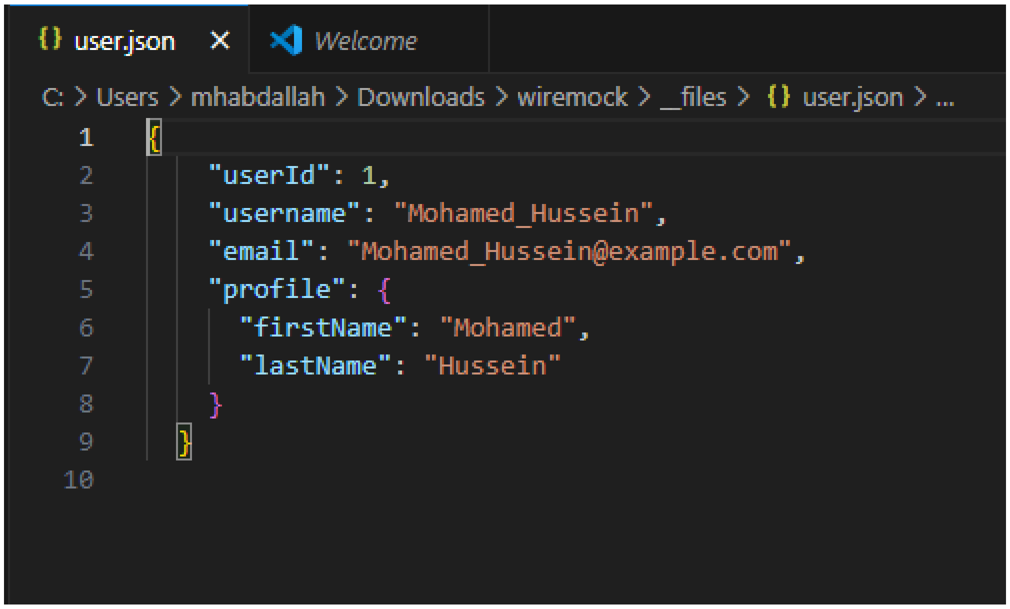
- Setup and Installation
  Download wiremock:
  Directory Setup: Created a directory structure with “mapping” and “__files” folders to store mappings and response files, respectively.

- Creating Mappings and Response Files:
  For each service, specific endpoints were defined. Each endpoint has an associated mapping file that defines the request criteria and the response to be returned. Corresponding JSON response files were created in the “__files” directory.
  Ex: Get/user request


{} user.json ×
Welcome
C: > Users > mhabdallah > Downloads > wiremock > mappings > {} user.json > ...

```
{
  "request": {
    "method": "GET",
    "url": "/user"
  },
  "response": {
    "status": 200,
    "bodyFileName": "user.json",
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```

Ex: Get/user response



{} user.json ×
Welcome
C: > Users > mhabdallah > Downloads > wiremock > __files > {} user.json > ...

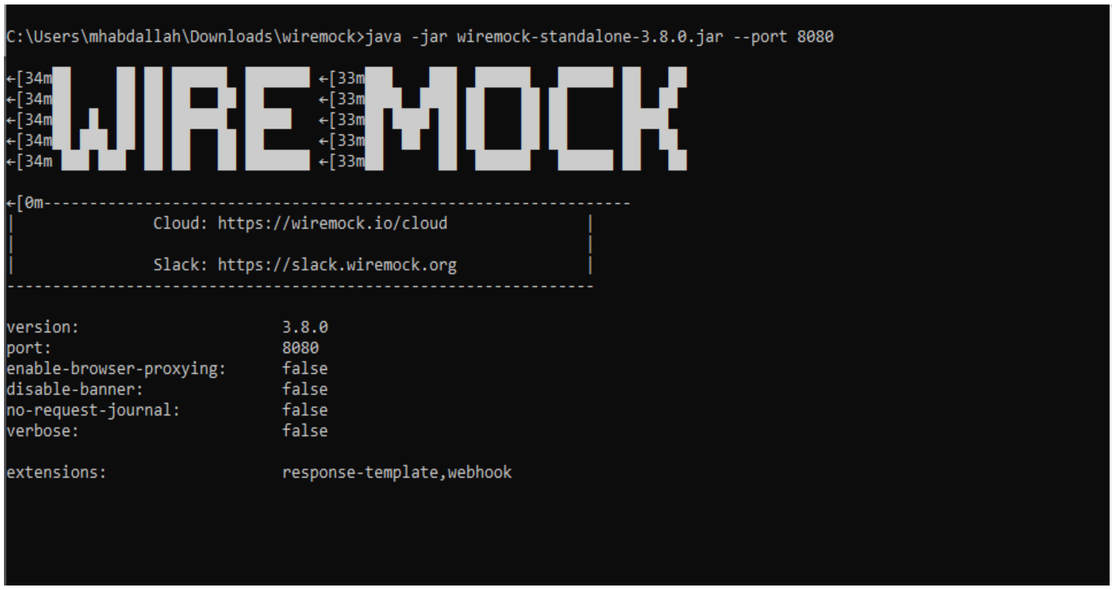
```json
{
    "userId": 1,
    "username": "Mohamed_Hussein",
    "email": "Mohamed_Hussein@example.com",
    "profile": {
      "firstName": "Mohamed",
      "lastName": "Hussein"
    }
}
```

- Running WireMock:

  WireMock was run using the standalone JAR file with the appropriate root directory specified. The command used is:

  “Java -jar wiremock-standalone-3.8.0.jar –port 8080”

  This command starts the wiremock server on port 8080, using the specified directory for mappings and response files.



```
C:\Users\mhabdallah\Downloads\wiremock>java -jar wiremock-standalone-3.8.0.jar --port 8080

←[34m WIRE ←[33m MOCK
←[34m      ←[33m
←[34m      ←[33m
←[34m      ←[33m
←[34m      ←[33m

←[0m-----------------------------------------------------------------
|               Cloud: https://wiremock.io/cloud                |
|                                                               |
|               Slack: https://slack.wiremock.org               |
-----------------------------------------------------------------

version:                      3.8.0
port:                         8080
enable-browser-proxying:      false
disable-banner:               false
no-request-journal:           false
verbose:                      false

extensions:                   response-template,webhook
```

- Testing Endpoints with Postman

  Postman was used to test the endpoints. For each endpoint, a request was created in postman, and assertions were added to verify the response.

4. Findings:

- Effectiveness: WireMock proved to be an effective tool for simulating Restful services. It allowed for rapid testing and development without the need for actual backend services.
- Flexibility: The flexibility of WireMock in handling different HTTP methods and body patterns made it possible to create comprehensive test scenarios.
- Ease of Use: The directory structure and the clear separation of mappings and response files made the setup easy to understand and manage.

5. Conclusion:

This project successfully demonstrated how WireMock can be used to create a mock server for simulating multiple endpoints across various services. The setup provides a robust environment for testing client applications, ensuring that development can proceed without dependency on actual backend implementations.