Instructors:
Mutasim Aldiab
Fahed Jubair

done by:

Hamza Hassan

Containerization Project

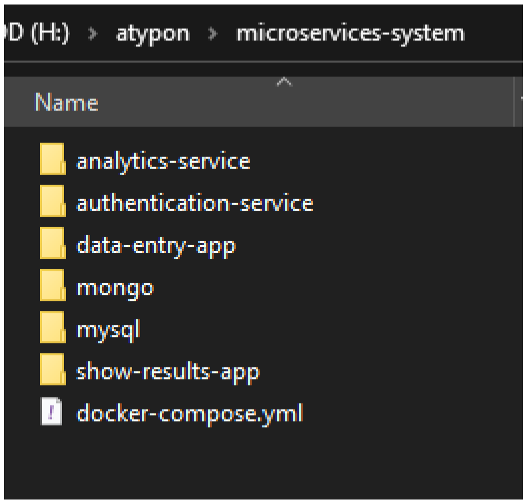
ABSTRACT

This report describes the development of a containerized microservices data collection and analytics system for student grades. The system consists of six services: enter data web app, authentication Service, MySQL Database, Analytics Service, MongoDB Database, and Show Results web app. Each service is implemented as a Docker container, which allows the services to be easily deployed and scaled, and it makes it easy to update the software without affecting the other services. The system was successfully developed and deployed to a production environment, and it is currently being used to collect and analyze student grades.

Introduction

The system is designed to handle the collection of student grade data, perform statistical analysis, and present the results in a user-friendly manner. This report delves into the problem statement, architecture, implementation, challenges faced, and the final outcomes of the project.

System Architecture



D (H:) › atypon › microservices-system

Name

- analytics-service
- authentication-service
- data-entry-app
- mongo
- mysql
- show-results-app
- docker-compose.yml

each one of this directories have own dockerfile

Service 1: Data Entry Web Application:
This microservice is responsible for collecting student grade data. Users interact with the web application to input student grades.

Service 2: Authentication Service:
The authentication service ensures the security of the system. It verifies user credentials before granting access to the data entry functionality. Only authenticated users are allowed to enter data.

Service 3: MySQL Database:
Student grade data entered through the data entry web application is stored in the MySQL database. This service ensures data persistence and provides a reliable storage solution.

Service 4: Analytics Service:
The analytics service computes statistics from the data stored in the MySQL database. It calculates metrics such as maximum, minimum, and average grades. The calculated analytics results are then written to the MongoDB database.

Service 5: MongoDB Database:
The MongoDB database stores the analytics results computed by the analytics service. MongoDB's flexibility and scalability make it a suitable choice for storing unstructured or semi-structured data such as analytics results.

Service 6: Results Presentation Web Application:
This microservice is responsible for presenting the analytics results to end-users. Users can access the results presentation web application to view the calculated statistics and gain insights from the data.

System Flow

The system's flow follows a structured sequence of steps to ensure data collection, processing, and presentation. The interactions between the microservices are as follows:

A user accesses the Data Entry Web Application (Service 1)

To access the data entry functionality, the user's credentials are verified by the Authentication Service (Service 2). If the credentials are valid, the user gains access to enter student grade data.
The entered student name and grade data is sent to the MySQL Database (Service 3) for storage. The database ensures that the data is persisted and can be retrieved when needed.

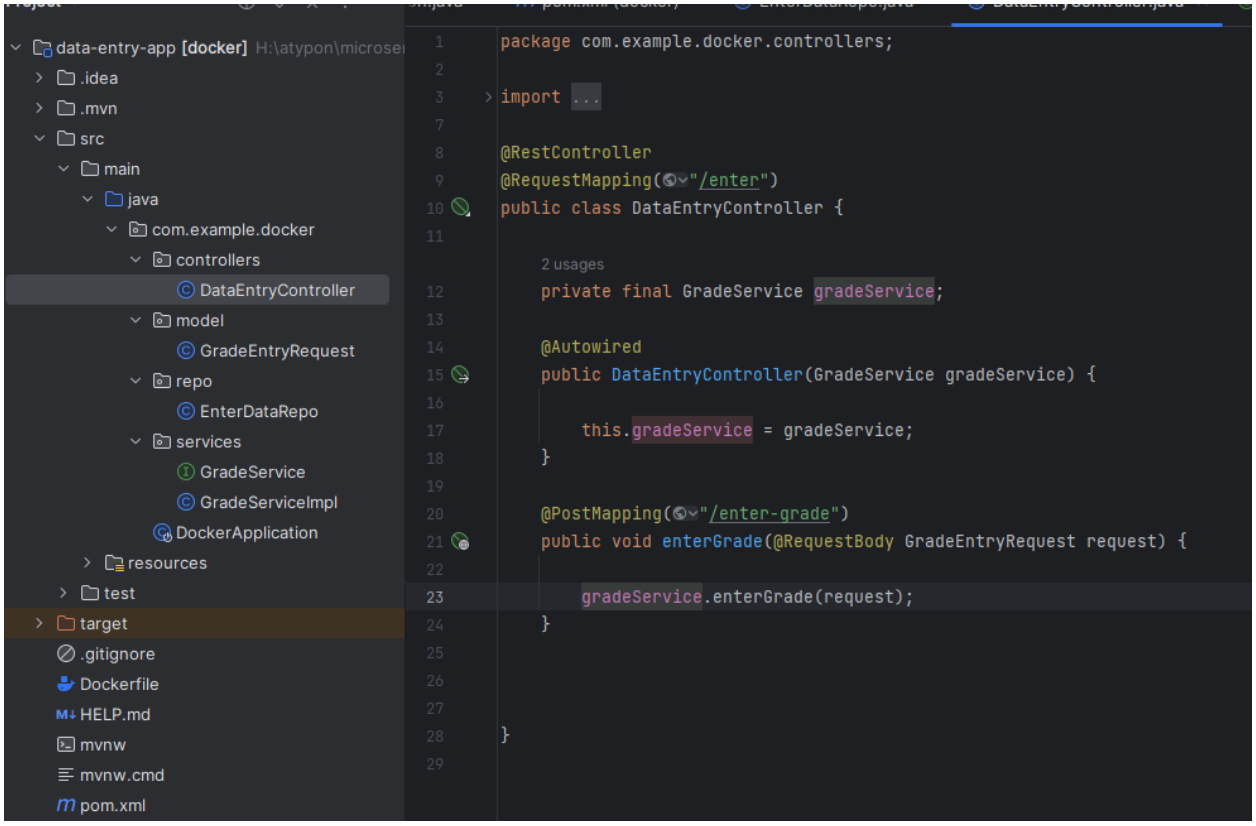
The Analytics Service (Service 4) periodically reads data from the MySQL database and computes analytics metrics such as maximum, minimum, and average grades.

The calculated analytics results are written to the MongoDB Database (Service 5). MongoDB's schemaless nature allows the storage of varying types of data without the need for a predefined schema.

Users can access the Results Presentation Web Application (Service 6) to view the calculated analytics results. The presentation application retrieves the results from the MongoDB database and displays them to the users.

Coding

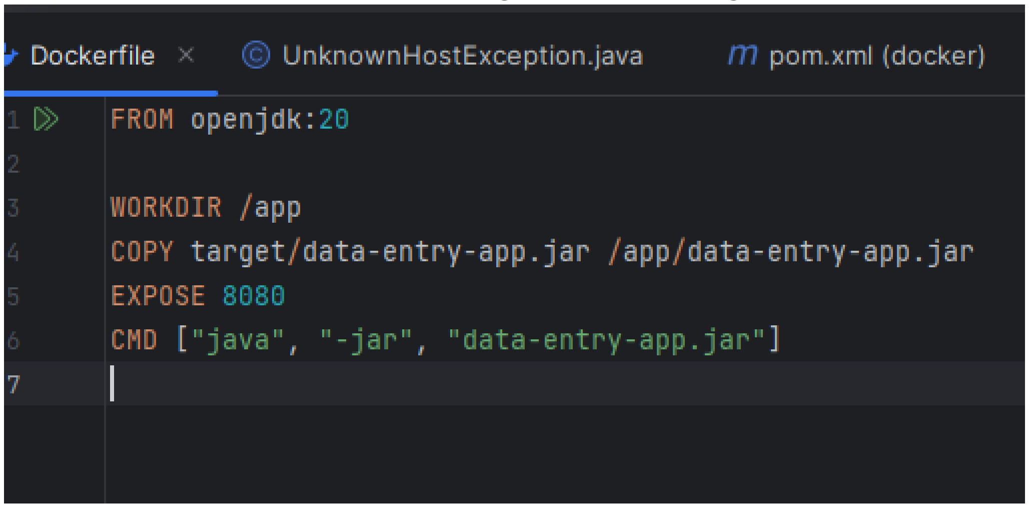
Data Entry Web Application :
to create this application I use Spring MVC and spring REST and mysql jdbc template to deal with database



```
data-entry-app [docker] H:\atypon\microse
  .idea
  .mvn
  src
    main
      java
        com.example.docker
          controllers
            DataEntryController
          model
            GradeEntryRequest
          repo
            EnterDataRepo
          services
            GradeService
            GradeServiceImpl
          DockerApplication
      resources
    test
  target
  .gitignore
  Dockerfile
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

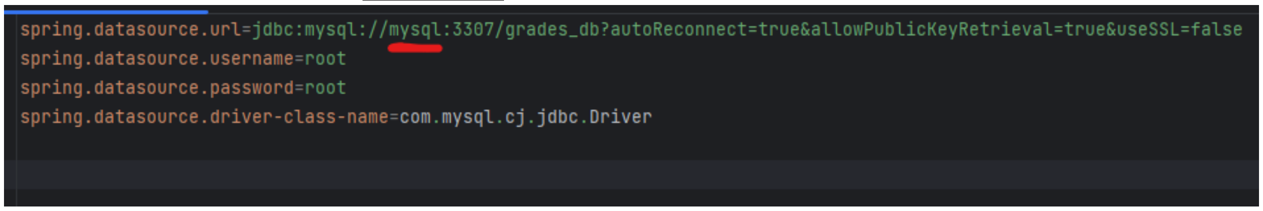
```java
package com.example.docker.controllers;

import ...

@RestController
@RequestMapping("/enter")
public class DataEntryController {

    2 usages
    private final GradeService gradeService;

    @Autowired
    public DataEntryController(GradeService gradeService) {

        this.gradeService = gradeService;
    }

    @PostMapping("/enter-grade")
    public void enterGrade(@RequestBody GradeEntryRequest request) {

        gradeService.enterGrade(request);
    }



}
```

then create dockerfile for building a Docker image



```
FROM openjdk:20

WORKDIR /app
COPY target/data-entry-app.jar /app/data-entry-app.jar
EXPOSE 8080
CMD ["java", "-jar", "data-entry-app.jar"]
```

and to connect with the **database**

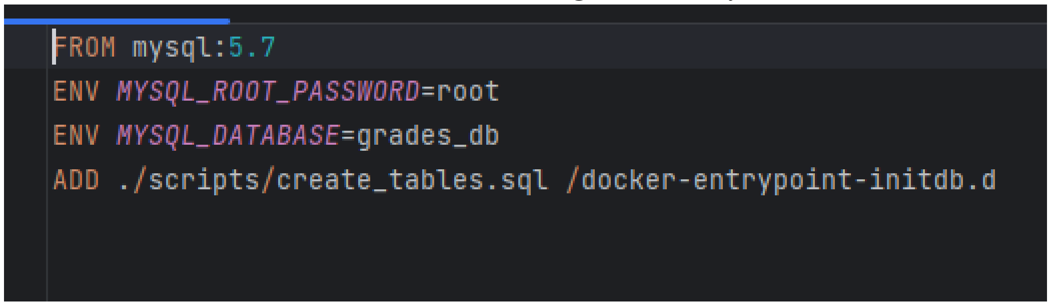


```
spring.datasource.url=jdbc:mysql://mysql:3307/grades_db?autoReconnect=true&allowPublicKeyRetrieval=true&useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

will connect to a MySQL database **container** named "mysql" on port 3307, and the database to be used is named "grades_db"

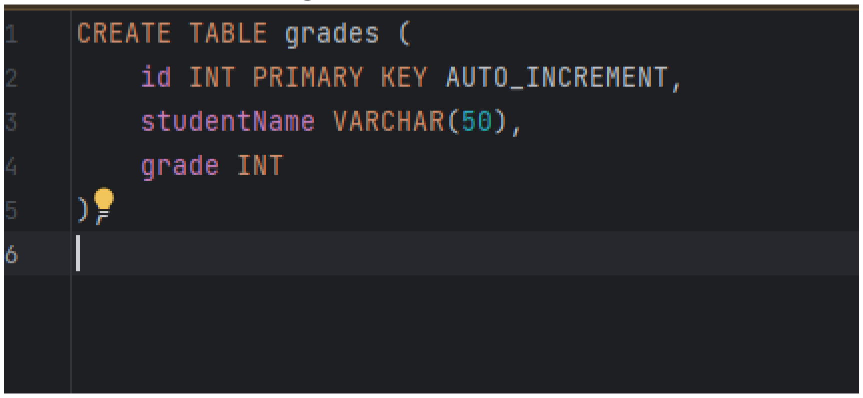
MySQL Database :

Use dockerfile to create a Docker image for a MySQL database container.



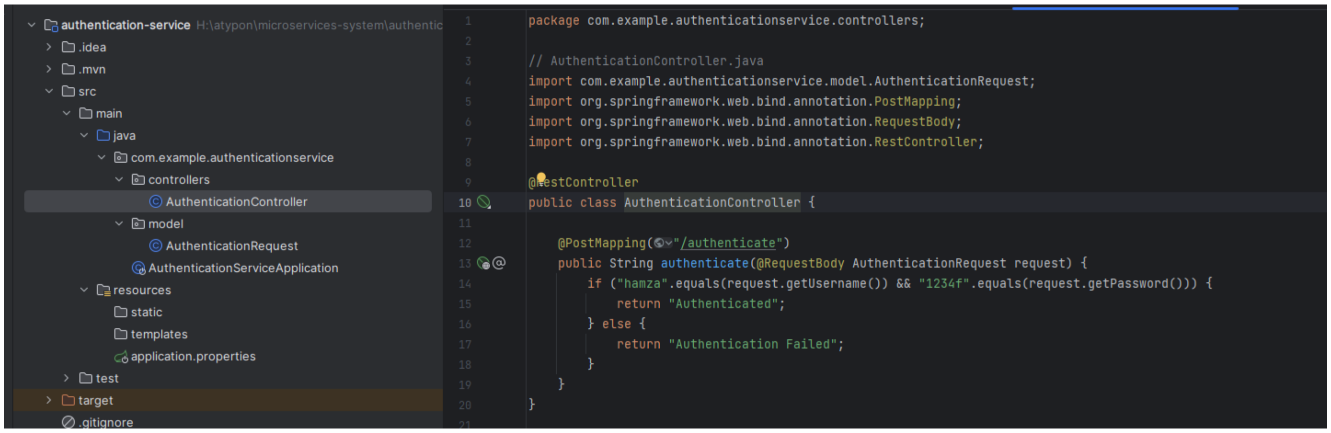
```
FROM mysql:5.7
ENV MYSQL_ROOT_PASSWORD=root
ENV MYSQL_DATABASE=grades_db
ADD ./scripts/create_tables.sql /docker-entrypoint-initdb.d
```

And add to this image table can store the s



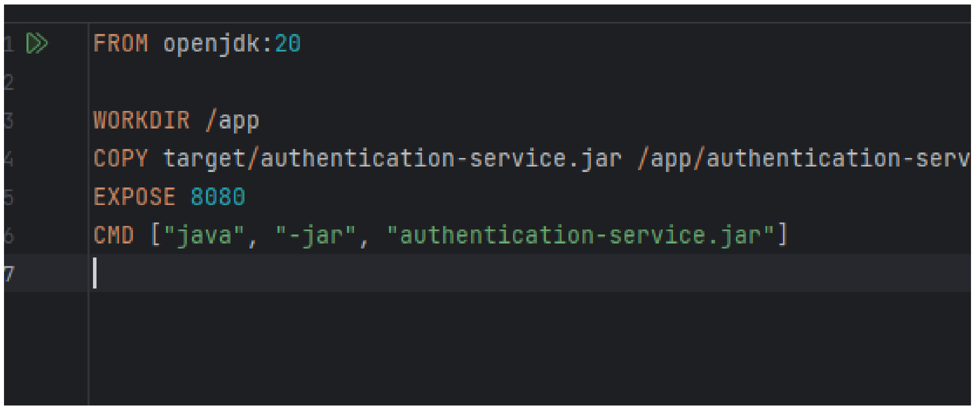
```sql
CREATE TABLE grades (
    id INT PRIMARY KEY AUTO_INCREMENT,
    studentName VARCHAR(50),
    grade INT
)
```

Authentication Service :
is a basic user authentication mechanism for a web service. When a POST request is made to the "/authenticate" endpoint with a JSON payload containing a username and password, the controller attempts to validate these credentials. If the provided credentials match the predefined values, the response will be "Authenticated". Otherwise, the response will be "Authentication Failed".



```java
package com.example.authenticationservice.controllers;

// AuthenticationController.java
import com.example.authenticationservice.model.AuthenticationRequest;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AuthenticationController {

    @PostMapping("/authenticate")
    public String authenticate(@RequestBody AuthenticationRequest request) {
        if ("hamza".equals(request.getUsername()) && "1234f".equals(request.getPassword())) {
            return "Authenticated";
        } else {
            return "Authentication Failed";
        }
    }
}
```

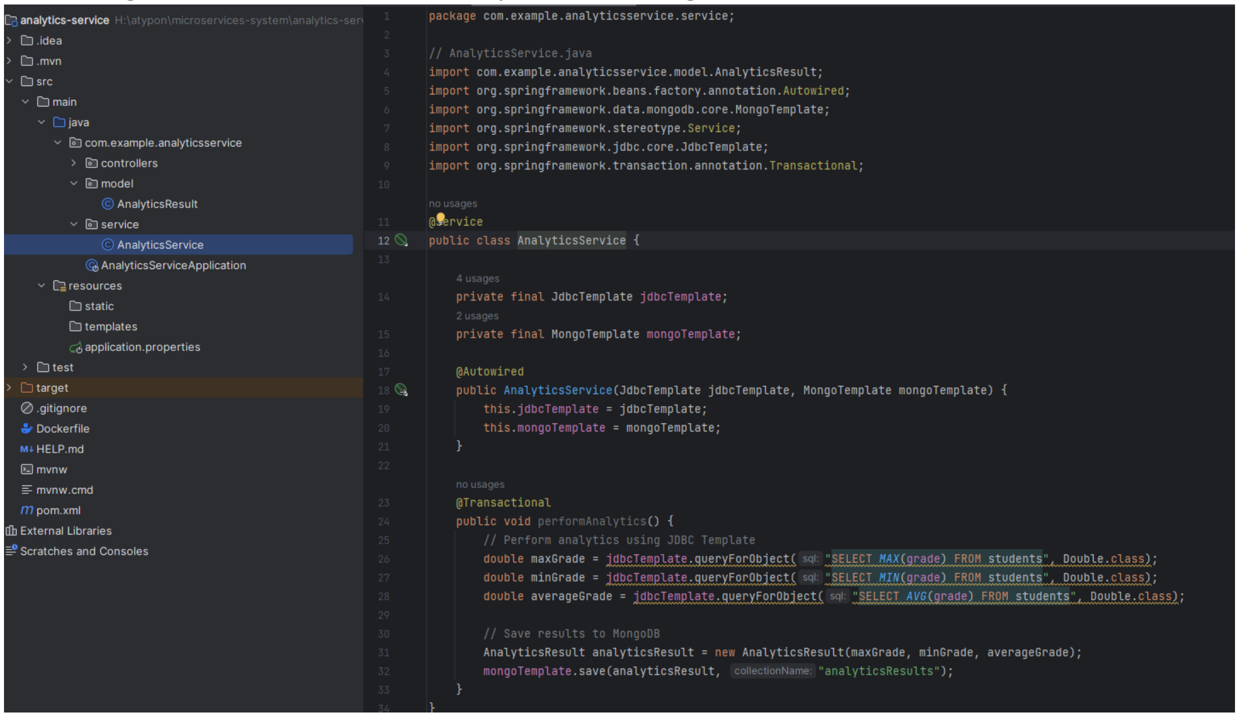
dockerfile



```dockerfile
FROM openjdk:20

WORKDIR /app
COPY target/authentication-service.jar /app/authentication-serv
EXPOSE 8080
CMD ["java", "-jar", "authentication-service.jar"]
```

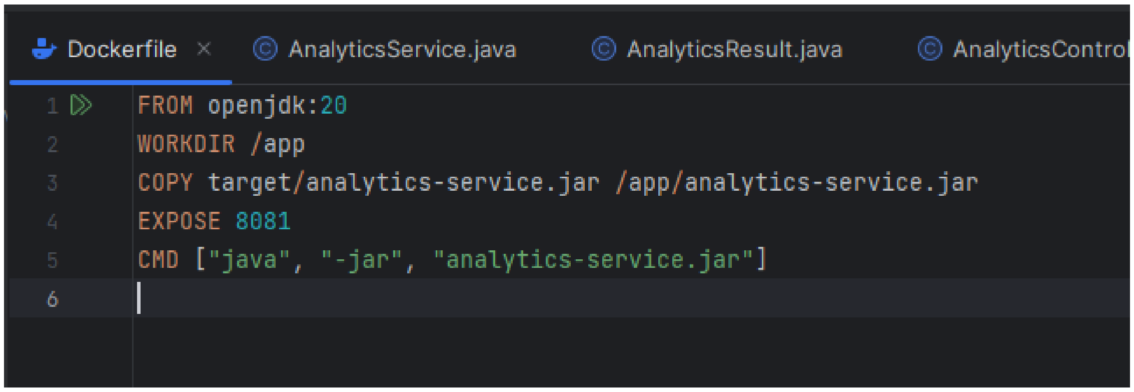
Analytics Service:

Used to get the data form mysql database container then find the max, min, avg for the student grad then stored this analytics in mongo database container



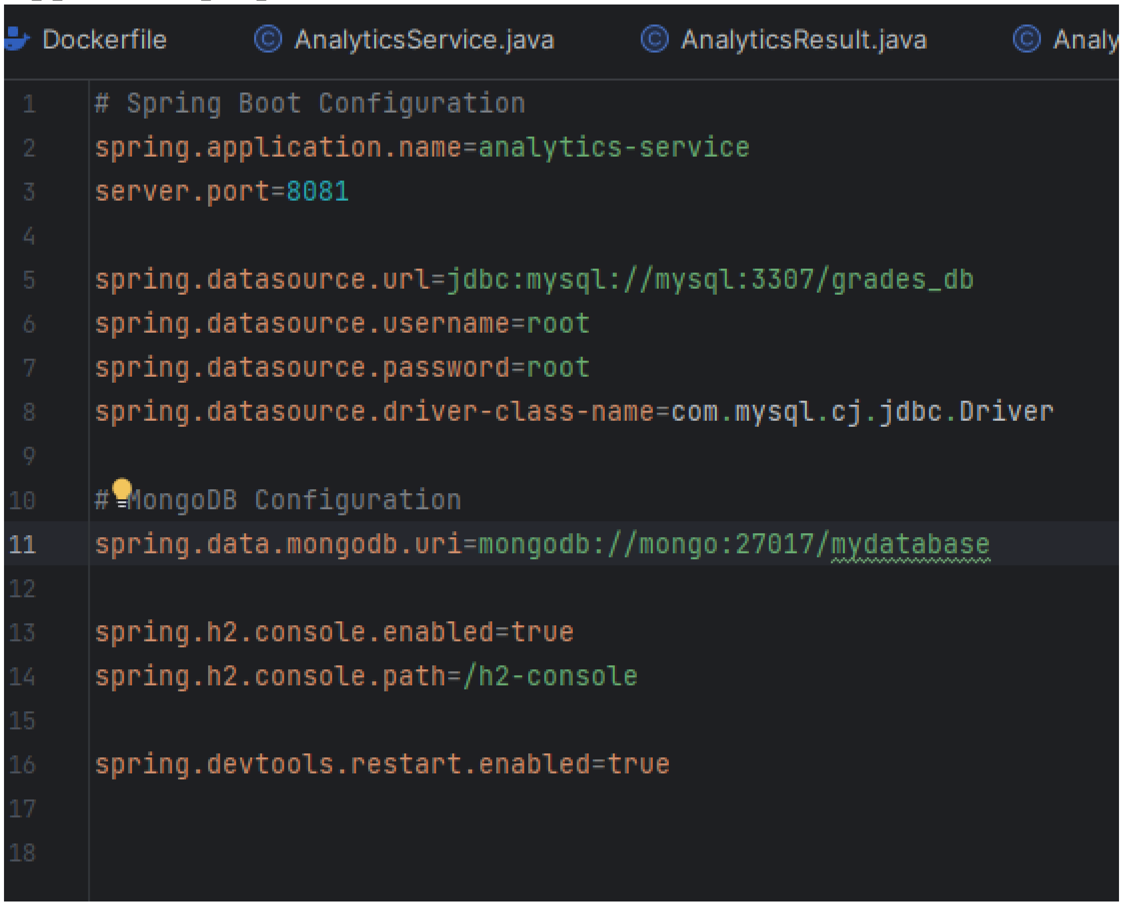
```java
package com.example.analyticsservice.service;

// AnalyticsService.java
import com.example.analyticsservice.model.AnalyticsResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Service;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.transaction.annotation.Transactional;

@Service
public class AnalyticsService {

    private final JdbcTemplate jdbcTemplate;
    private final MongoTemplate mongoTemplate;

    @Autowired
    public AnalyticsService(JdbcTemplate jdbcTemplate, MongoTemplate mongoTemplate) {
        this.jdbcTemplate = jdbcTemplate;
        this.mongoTemplate = mongoTemplate;
    }

    @Transactional
    public void performAnalytics() {
        // Perform analytics using JDBC Template
        double maxGrade = jdbcTemplate.queryForObject( sql: "SELECT MAX(grade) FROM students", Double.class);
        double minGrade = jdbcTemplate.queryForObject( sql: "SELECT MIN(grade) FROM students", Double.class);
        double averageGrade = jdbcTemplate.queryForObject( sql: "SELECT AVG(grade) FROM students", Double.class);

        // Save results to MongoDB
        AnalyticsResult analyticsResult = new AnalyticsResult(maxGrade, minGrade, averageGrade);
        mongoTemplate.save(analyticsResult,  collectionName: "analyticsResults");
    }
}
```

Dockerfile



```dockerfile
FROM openjdk:20
WORKDIR /app
COPY target/analytics-service.jar /app/analytics-service.jar
EXPOSE 8081
CMD ["java", "-jar", "analytics-service.jar"]
```

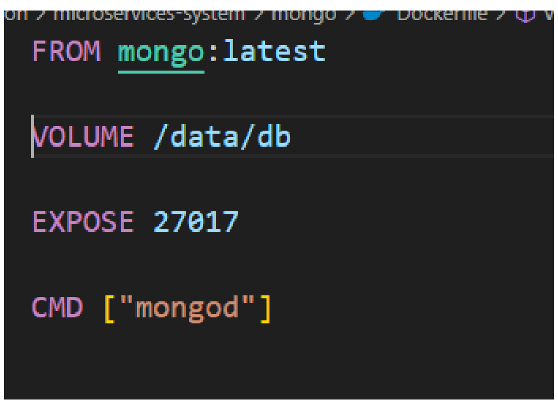
Application properties



```
# Spring Boot Configuration
spring.application.name=analytics-service
server.port=8081

spring.datasource.url=jdbc:mysql://mysql:3307/grades_db
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#MongoDB Configuration
spring.data.mongodb.uri=mongodb://mongo:27017/mydatabase

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.devtools.restart.enabled=true
```

connecting to a MySQL database container named "grades_db" on port 3307.
connecting to a MongoDB container named "mongo" on port 27017

MongoDB:
Dockerfile



```
FROM mongo:latest

VOLUME /data/db

EXPOSE 27017

CMD ["mongod"]
```

DOCKER COMPOSER :

The provided Docker Compose configuration defines a multi-container application consisting of several services. Each service is isolated within a Docker container and is part of a custom bridge network called my-network. The application components include a MySQL database, a data entry application, an authentication service, an analytics service, a MongoDB instance, and a show results application.

Services:

1. mysql:
    - Docker Image: MySQL 5.7
    - Environment Variables:
        - MYSQL_DATABASE: Specifies the name of the MySQL database (grades_db in this case).
        - MYSQL_ROOT_PASSWORD: Sets the root password for the MySQL instance.
    - Ports:
        - Maps host port 3307 to container port 3306 to access the MySQL service.
    - Volumes:
        - Mounts a local SQL file setup.sql into the container's initialization script directory (/docker-entrypoint-initdb.d) to initialize the database schema during container setup.
2. data-entry-app:
    - Custom Docker Image: Built from the ./data-entry-app context.
    - Ports:
        - Maps host port 8081 to container port 8080 to access the data entry application.
    - Environment Variables:
        - Configures the Spring Data Source for connecting to the MySQL database.
    - Dependencies:
        - Depends on the mysql service.
    - Network:
        - Connects to the my-network bridge network.
3. authentication-service:
    - Custom Docker Image: Built from the ./authentication-service context.

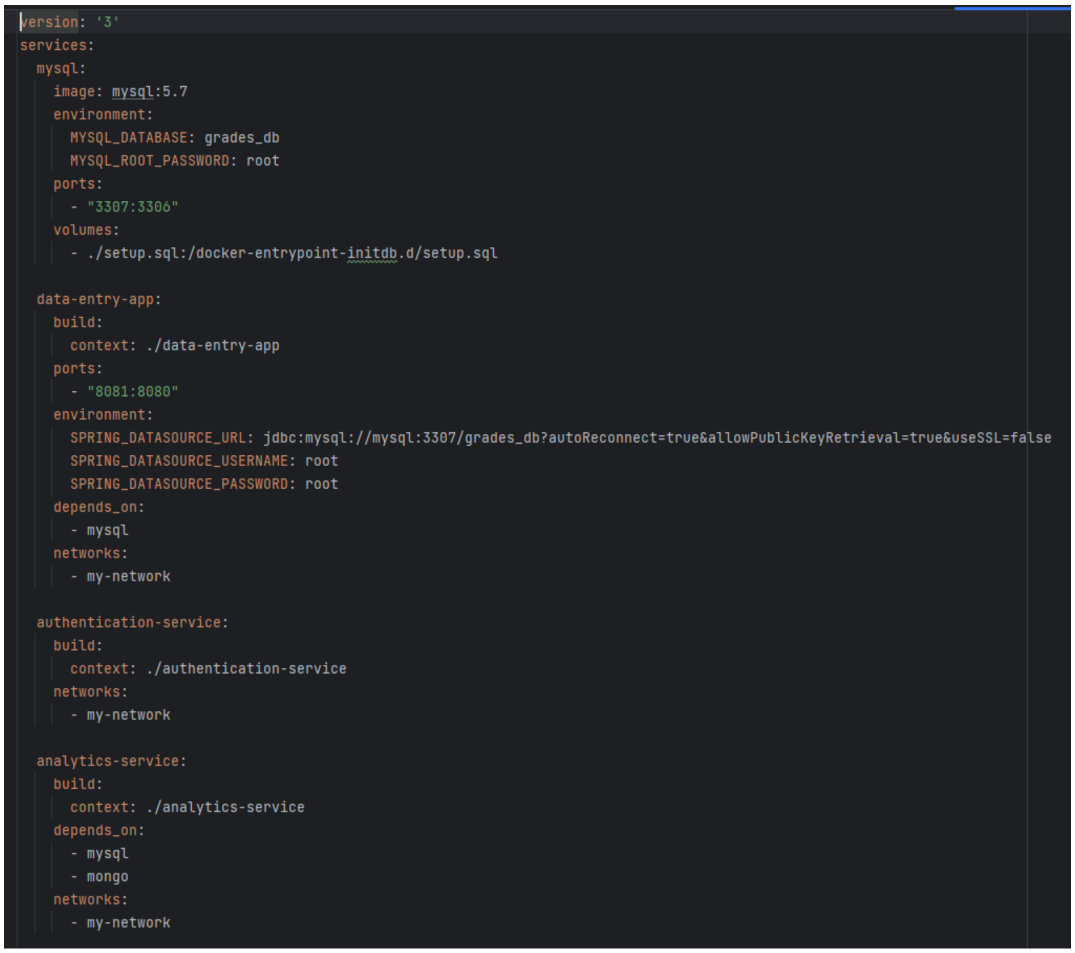
  - Network:
    - Connects to the my-network bridge network.
4. analytics-service:
  - Custom Docker Image: Built from the ./analytics-service context.
  - Dependencies:
    - Depends on the mysql and mongo services.
  - Network:
    - Connects to the my-network bridge network.
5. mongo:
  - Custom Docker Image: Built from the ./mongo context.
  - Volumes:
    - Mounts a volume named mongo-data to persist MongoDB data.
  - Network:
    - Connects to the my-network bridge network.
6. show-results-app:
  - Custom Docker Image: Built from the ./show-results-app context.
  - Ports:
    - Maps host port 8082 to container port 8080 to access the show results application.
  - Dependencies:
    - Depends on the mongo service.
  - Network:
    - Connects to the my-network bridge network.

Networks:

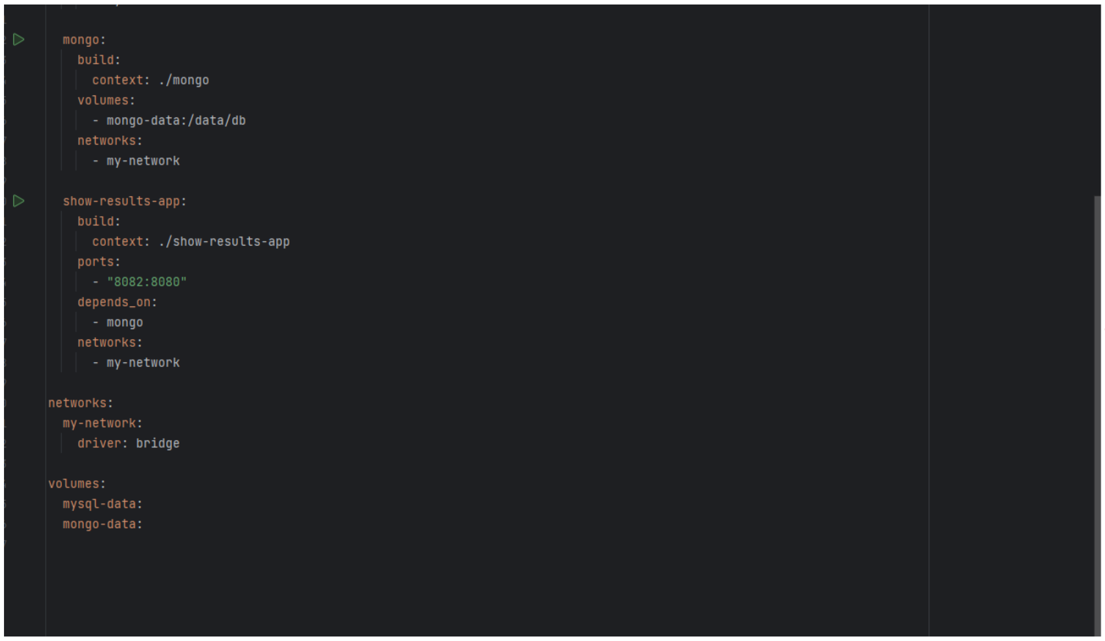
- my-network: A custom bridge network used for communication between containers. All services connect to this network, allowing them to communicate with each other using their service names as hostnames.

Volumes:

- mysql-data: Volume for persisting MySQL data.
- mongo-data: Volume for persisting MongoDB data.

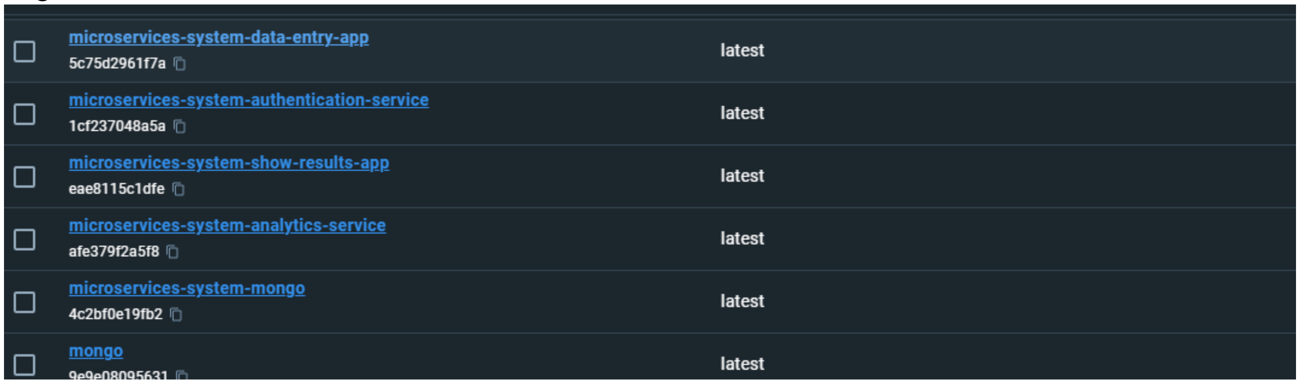


```yaml
version: '3'
services:
  mysql:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: grades_db
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3307:3306"
    volumes:
      - ./setup.sql:/docker-entrypoint-initdb.d/setup.sql

  data-entry-app:
    build:
      context: ./data-entry-app
    ports:
      - "8081:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3307/grades_db?autoReconnect=true&allowPublicKeyRetrieval=true&useSSL=false
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    depends_on:
      - mysql
    networks:
      - my-network

  authentication-service:
    build:
      context: ./authentication-service
    networks:
      - my-network

  analytics-service:
    build:
      context: ./analytics-service
    depends_on:
      - mysql
      - mongo
    networks:
      - my-network
```



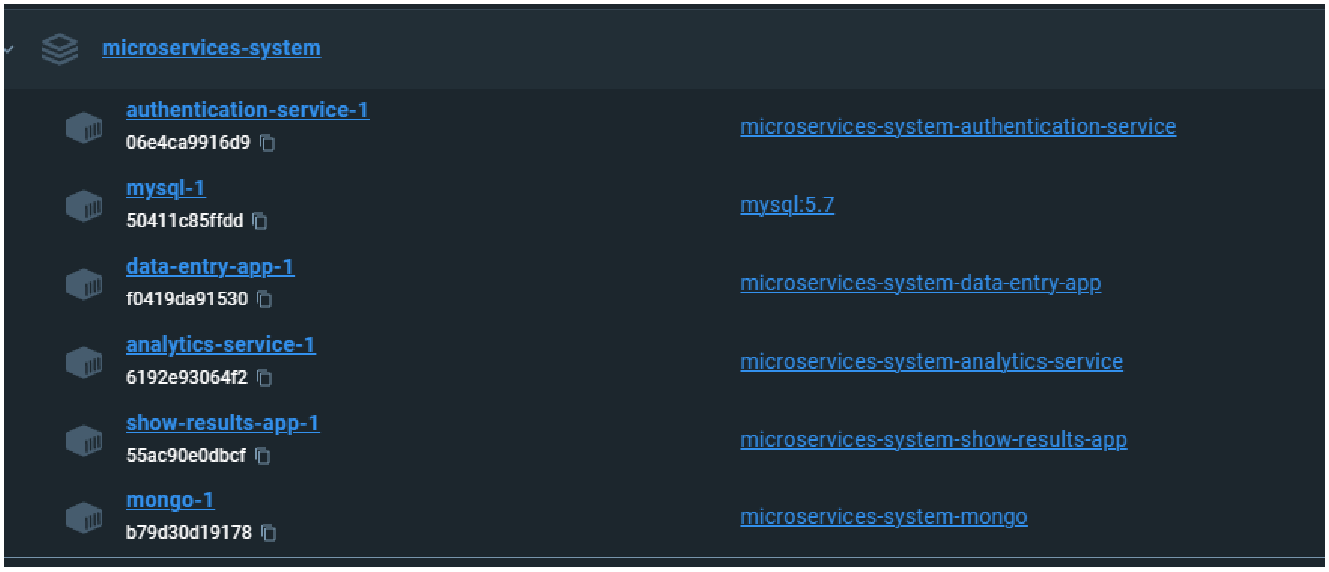
```yaml
  mongo:
    build:
      context: ./mongo
    volumes:
      - mongo-data:/data/db
    networks:
      - my-network

  show-results-app:
    build:
      context: ./show-results-app
    ports:
      - "8082:8080"
    depends_on:
      - mongo
    networks:
      - my-network

networks:
  my-network:
    driver: bridge

volumes:
  mysql-data:
  mongo-data:
```

When you run the command **docker-compose up**, you are instructing Docker Compose to start and manage a collection of containers defined in a **docker-compose.yml**
In our case that will create 6 image and 6 container
imgaes;



microservices-system-data-entry-app 5c75d2961f7a	latest
microservices-system-authentication-service 1cf237048a5a	latest
microservices-system-show-results-app eae8115c1dfe	latest
microservices-system-analytics-service afe379f2a5f8	latest
microservices-system-mongo 4c2bf0e19fb2	latest
mongo 9e9e08095631	latest

Containers:



microservices-system

authentication-service-1 06e4ca9916d9	microservices-system-authentication-service
mysql-1 50411c85ffdd	mysql:5.7
data-entry-app-1 f0419da91530	microservices-system-data-entry-app
analytics-service-1 6192e93064f2	microservices-system-analytics-service
show-results-app-1 55ac90e0dbcf	microservices-system-show-results-app
mongo-1 b79d30d19178	microservices-system-mongo

The end