МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ИТОГОВОМУ ПРОЕКТУ №3

по дисциплине

«Технология разработки программных приложений»

Тема: «WEB-разработка сайта социальная сеть для IT специалистов»

Выполнили студенты группы ИКБО-20-21

«29» 04 2023г. Фомичев Р.А.
Сидоров С.Д.
Опришко В.Д.

Преподаватель «29» 04 2023 г. Петренко А.А.

Москва 2023

СОДЕРЖАНИЕ

ЧАСТЬ 3

1.1 Упаковка приложения

1.1.1 Создание Dockerfile

Каждому образу Docker соответствует файл, который называется Dockerfile. При запуске команды docker build для создания нового образа подразумевается, что Dockerfile находится в текущей рабочей директории.

Dockerfile — это текстовый файл с инструкциями, необходимыми для создания образа контейнера. Эти инструкции включают идентификацию существующего образа, используемого в качестве основы, команды, выполняемые в процессе создания образа, и команду, которая будет выполняться при развертывании новых экземпляров этого образа контейнера. Инструкции при сборке образа обрабатываются сверху вниз.

Всего в проекте содержится два таких файла, каждый из которых отвечает за свою часть приложения. На рисунке 1 представлено содержание Dockerfile для клиентской части.

```
FROM node:current-slim

WORKDIR /client

COPY /src ./src
COPY index.html .
COPY package*.json .
COPY tsconfig*.json .
COPY vite.config.ts .
COPY /.storybook .

RUN npm i && npm cache clean --force \
    && npm i @esbuild/linux-x64 esbuild-linux-64 \
    && npm run build

EXPOSE 80

# development
# CMD ["npm", "run", "dev"]

# production
CMD ["npm", "run", "preview"]
```

Рисунок 1 – Dockerfile клиентской части проекта

На рисунке 2 представлено содержание Dockerfile для серверной части.

```
FROM openjdk:17-alpine
FROM maven:3.8.5-openjdk-17-slim

WORKDIR /server

COPY . .

RUN mvn clean package -DskipTests

EXPOSE 8080

# development
# CMD ["mvn", "spring-boot:run"]

# production
CMD ["java", "-jar", "/server/target/specIT-0.1.0.jar"]
```

Рисунок 2 – Dockerfile серверной части проекта

Здесь используются аналогичные команды. Сначала создается два базовых образа (openjdk и maven), затем создается рабочая директория, в которую копируются все файлы. Очищаются все ранее скомпилированные файлы, подключается порт 8080 и выполняется запуск проекта через собранный jar файл.

1.2.2 Файл docker-compose

Docker-compose.yml — конфигурационный файл в YAML-формате, описывающий логику запуска и взаимодействия контейнеров между собой и внешним миром. Он позволяет запускать множество контейнеров

одновременно и маршрутизировать потоки данных между ними. В сущности инструкции заложенные в docker-compose.yml по логике работы идентичны ключам команды docker run.

Docker-compose.yml мало чем похож на docker-файлы. В отличие от них, docker-compose.yml записан в древовидном YAML, это отлично видно на рисунке 3, который показывает файл, используемый в текущей работе.

Рисунок 3 – Содержание docker-compose.yml

1.1.3 Сборка приложения

После того, как в docker-compose.yml внесены все необходимые инструкции, проект нужно собрать. Этот шаг нашей работы напоминает использование команды docker build, но соответствующая команда docker-compose build имеет отношение к нескольким сервисам. Пример работы сборки проекта показан на рисунке 4.

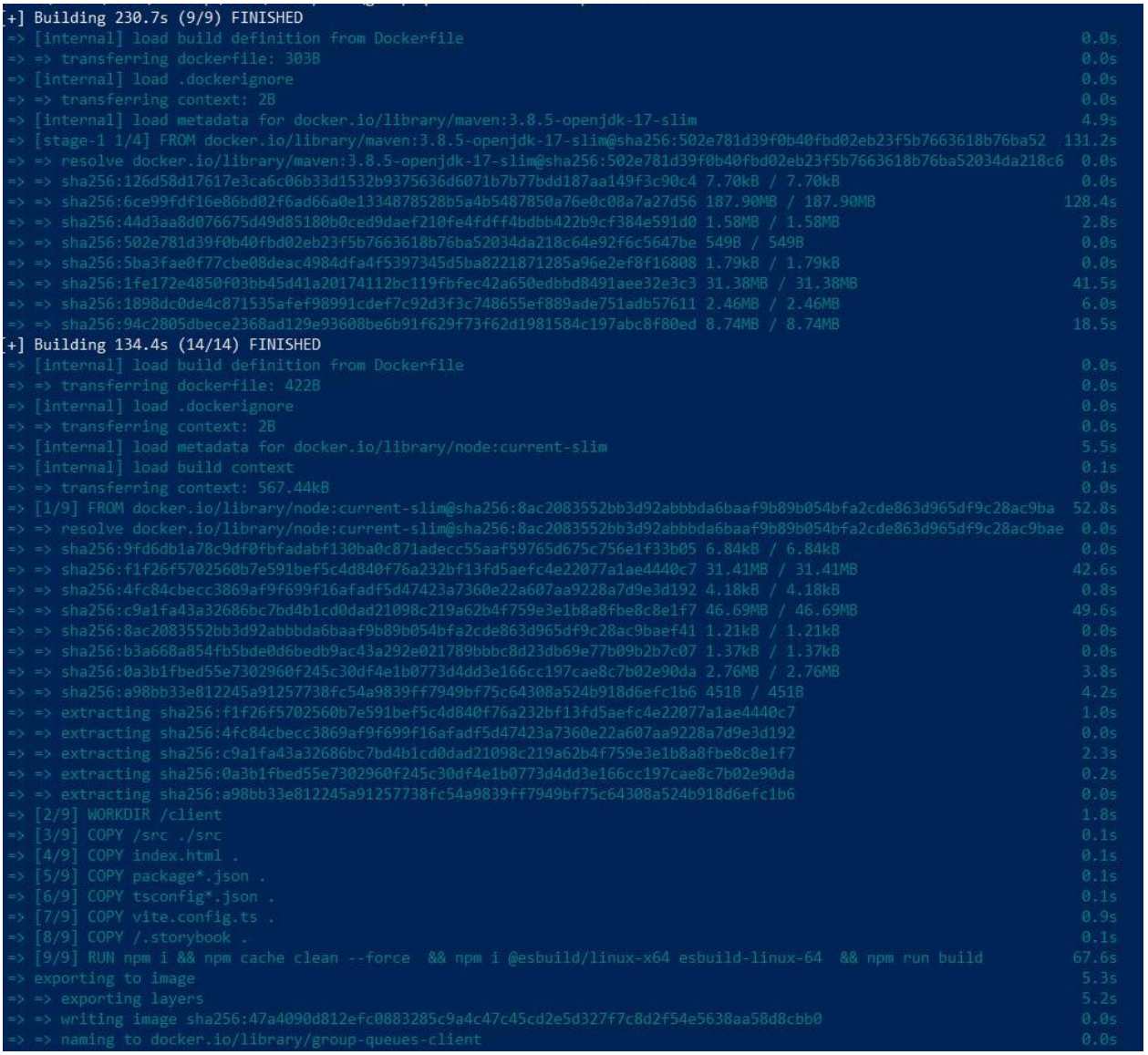

Рисунок 4 – Выполнение команды docker-compose build

Команда docker-compose build собирает проект. На этом этапе происходит скачивание указанных зависимостей, установка пакетов. В общем, выполнение инструкций, записанных в Docker-файлах.

После успешной сборки проект можно запустить используя команду

docker-compose up (рисунок 5). Эта операция соответствует шагу, на котором, при работе с отдельными контейнерами, выполняется команда docker run.

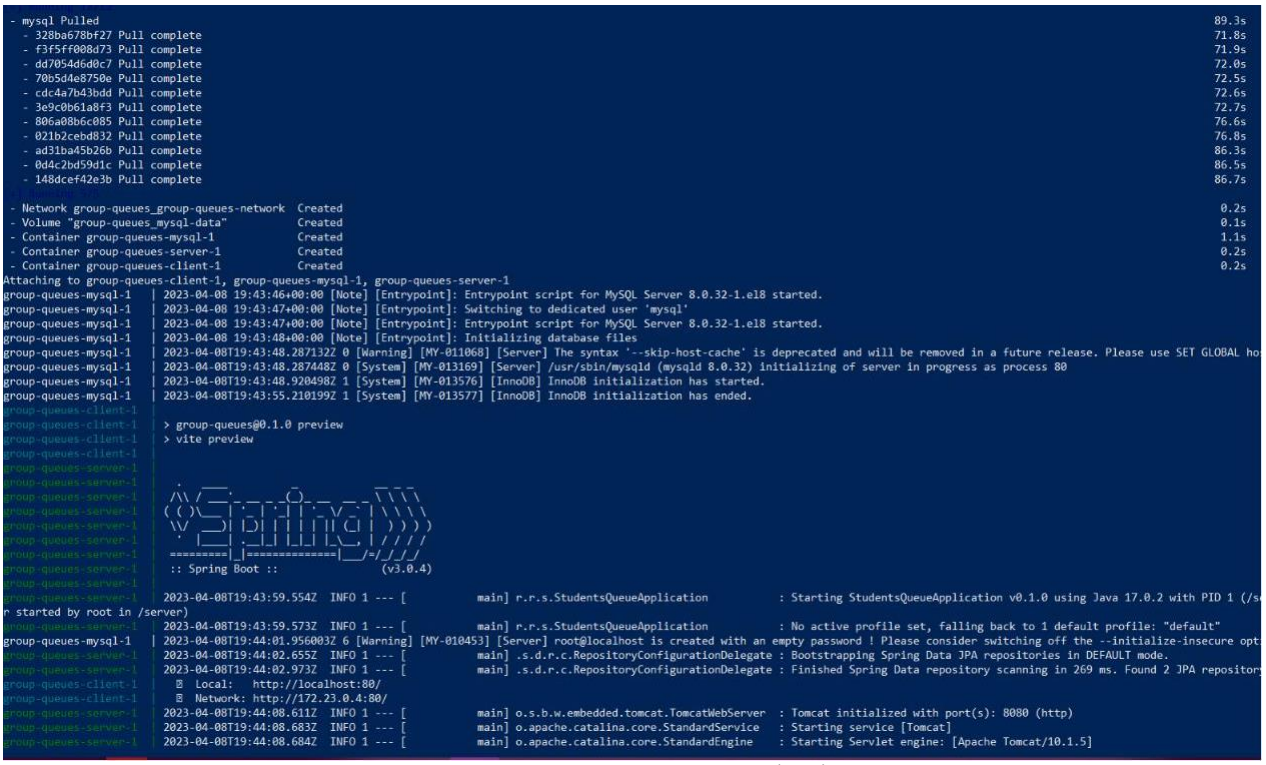

Рисунок 5 – Выполнение команды docker-compose up

Команда объединяет выходные данные каждого контейнера. Когда ее выполнение завершится, все контейнеры остановятся.

Ниже приведены основные команды, используемые для docker-compose:

- docker-compose build — собрать проект
- docker-compose up — запустить проект
- docker-compose down — остановить проект
- docker-compose logs -f [service name] — посмотреть логи сервиса
- docker-compose ps — вывести список контейнеров
- docker-compose exec [service name] [command] — выполнить команду в контейнере
- docker-compose images — список образов

Также стоит отметить, что управлять процессами контейнеров можно из Docker Desktop. На рисунке 6 представлен контейнер разрабатываемого приложения. Видим, что он содержит именно те службы, что мы указали в

docker-compose.yml.

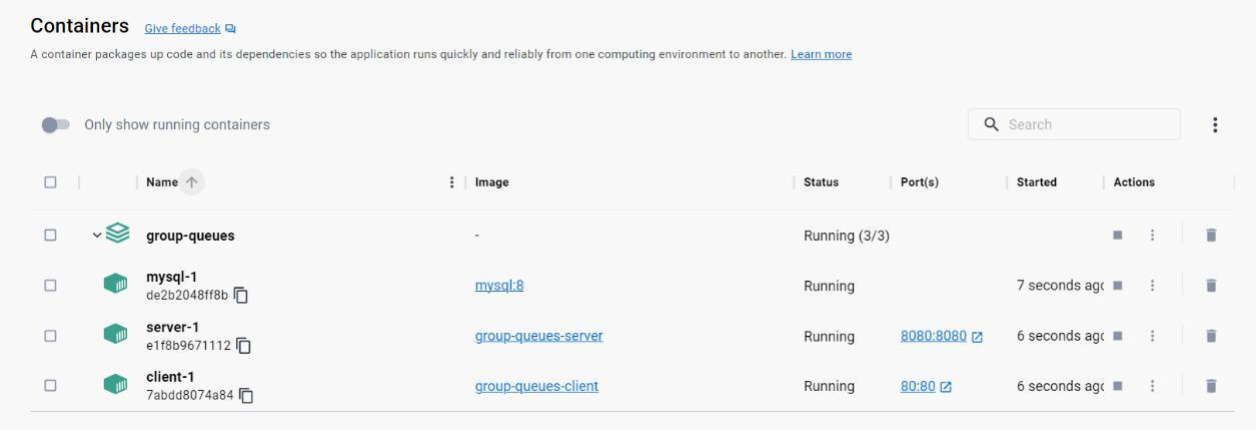

Рисунок 6 – Выполнение команды docker-compose build

Дополнительно можно посмотреть системную информацию о работе каждого контейнера, например, логи (рисунок 7) .

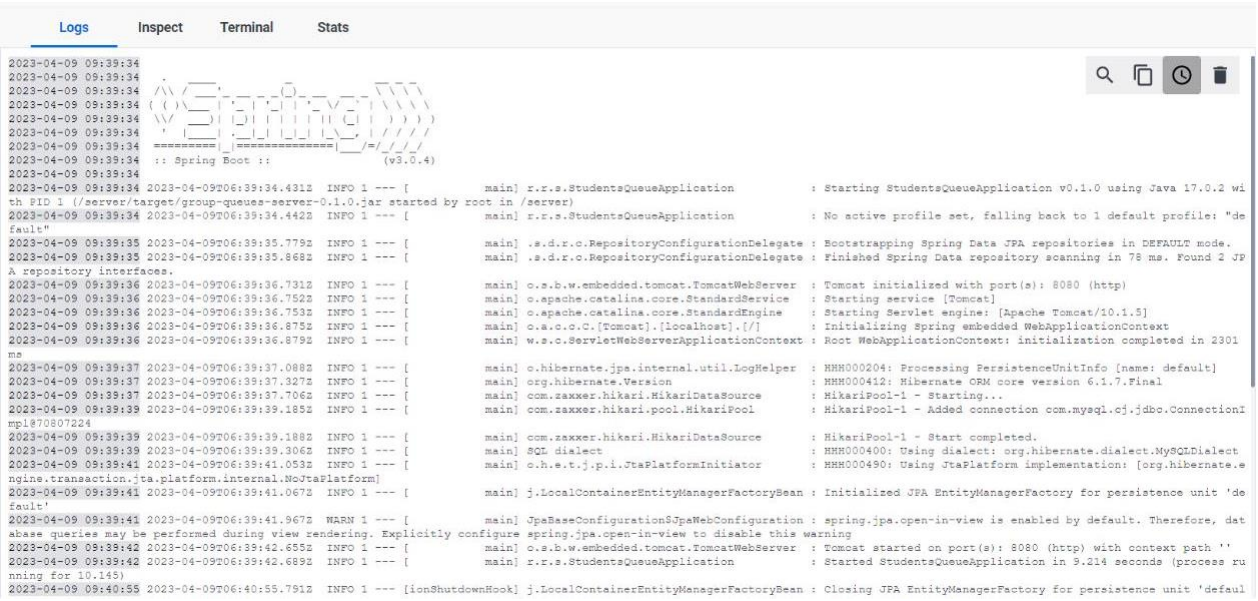

Рисунок 7 – Просмотр logs контейнера group-queues-server-1

1.1.4 Тестирование приложения

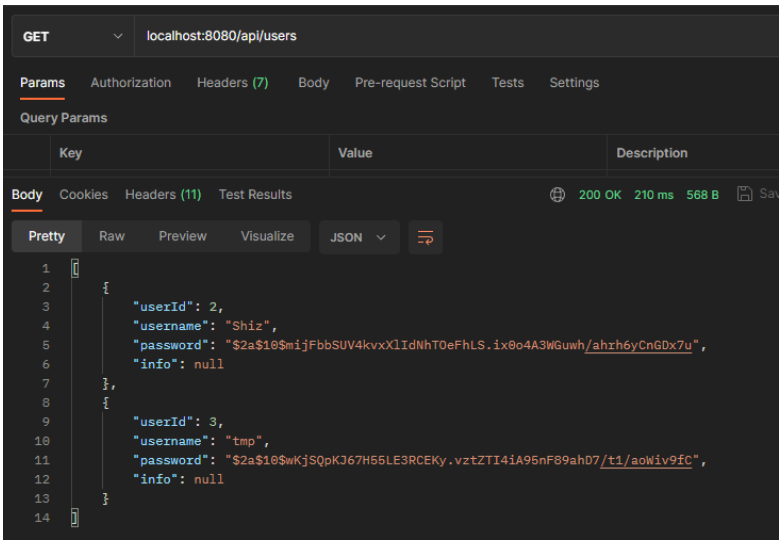

Рисунок 8 - работа серверной части веб-приложения

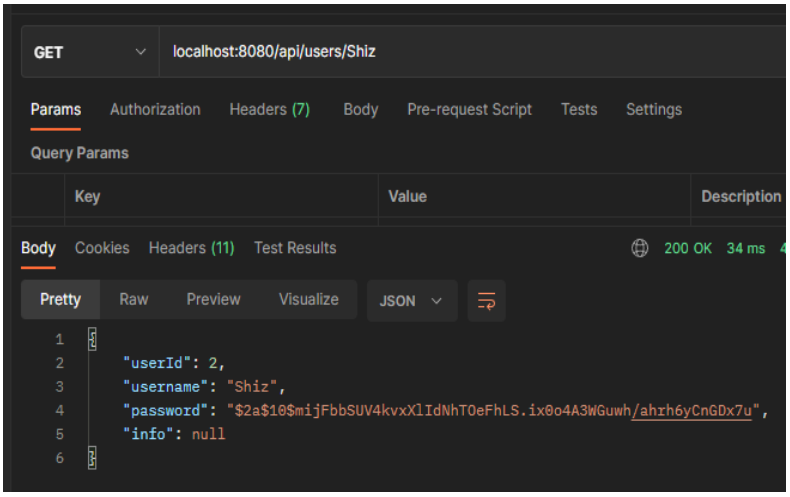

Рисунок 9 - работа серверной части веб-приложения

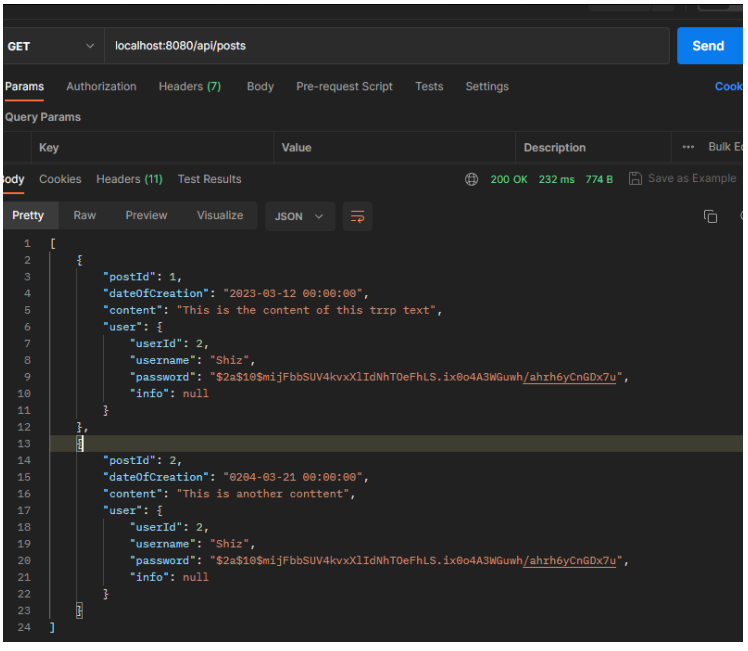

Рисунок 10 - работа серверной части веб-приложения

ВЫВОДЫ

В результате проделанной работы удалось создайте Dockerfile, в который было запаковано приложение. Также был сделан docker-compose.yml. Работа веб-приложения была протестирована внтури контейнера, запуск прошел успешно, приложение работает стабильно.