



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 2023 г.

Фомичев Р.А.
Сидоров С.Д.
Опришко В.Д.

Преподаватель

✓ «29» 04 2023 г.

Петренко А.А.

Москва 2023

СОДЕРЖАНИЕ

Часть 3	3
1.1 Упаковка приложения	3
1.1.1 Создание Dockerfile	4
1.1.2 Файл docker-compose	6
1.1.3 Сборка приложения	9
1.1.4 Тестирование приложения	13
Выводы	19

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.

```
version: '3'

services:
  mysql:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: group-queues-database
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - specIT

  server:
    build: specIT
    ports:
      - '8080:8080'
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/specIT
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: password
      SERVER_PORT: 8080
    volumes:
      - ./specIT/src:/server/src
      - ./specIT/target/classes:/server/target/classes
      - ./specIT/target/generated-sources:/server/target/generated-sources
      - ./specIT/target/generated-test-sources:/server/target/generated-test-sources
      - ./specIT/target/maven-status:/server/target/maven-status
      - ./specIT/target/test-classes:/server/target/test-classes
    depends_on:
      - mysql
    networks:
      - specIT

  client:
    build: specIT
    ports:
      - '80:80'
    volumes:
      - ./specIT/src:/client/src
      - ./specIT/vite.config.ts:/client/vite.config.ts
    depends_on:
      - server
    networks:
      - specIT

networks:
  group-queues-network:
```

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.

```
+ ] Building 230.7s (9/9) FINISHED
-> [internal] load build definition from Dockerfile                                0.0s
-> => transferring dockerfile: 303B                                              0.0s
-> [internal] load .dockerignore                                                0.0s
-> => transferring context: 2B                                                  0.0s
-> [internal] load metadata for docker.io/library/maven:3.8.5-openjdk-17-slim  4.9s
-> [stage-1 1/4] FROM docker.io/library/maven:3.8.5-openjdk-17-slim@sha256:502e781d39f0b40fbd02eb23f5b7663618b76ba52 131.2s
-> => resolve docker.io/library/maven:3.8.5-openjdk-17-slim@sha256:502e781d39f0b40fbd02eb23f5b7663618b76ba52034da218c6 0.0s
-> => sha256:126d58d17617e3ca6c06b33d1532b9375636d6071b7b77bdd187aa149f3c90c4 7.70kB / 7.70kB 0.0s
-> => sha256:6ce99fd8f16e86bd02f6ad66a0e1334878528b5a4b5487850a76e0c08a7a27d56 187.90MB / 187.90MB 128.4s
-> => sha256:44d3aa8d076675d49d85180b0ced9daef210fe4fdff4b4bb422b9cf384e591d0 1.58MB / 1.58MB 2.8s
-> => sha256:502e781d39f0b40fbd02eb23f5b7663618b76ba52034da218c64e92f6c5647be 549B / 549B 0.0s
-> => sha256:5ba3fae0f77cbe08deac4984dfa4f5397345d5ba8221871285a96e2ef8f16808 1.79kB / 1.79kB 0.0s
-> => sha256:1fe172e4850f03bb45d41a20174112bc119fbfec42a650edbbd8491aee32e3c3 31.38MB / 31.38MB 41.5s
-> => sha256:1898dc0de4c871535afef98991cdef7c92d3f3c748655ef889ade751adb57611 2.46MB / 2.46MB 6.0s
-> => sha256:94c2805dbce2368ad129e93608be6b91f629f73f62d1981584c197abc8f80ed 8.74MB / 8.74MB 18.5s
+ ] Building 134.4s (14/14) FINISHED
-> [internal] load build definition from Dockerfile                                0.0s
-> => transferring dockerfile: 422B                                              0.0s
-> [internal] load .dockerignore                                                0.0s
-> => transferring context: 2B                                                  0.0s
-> [internal] load metadata for docker.io/library/node:current-slim            5.5s
-> [internal] load build context                                                0.1s
-> => transferring context: 567.44kB                                             0.0s
-> [1/9] FROM docker.io/library/node:current-slim@sha256:8ac2083552bb3d92abbbda6baaf9b89b054bfa2cde863d965df9c28ac9bae 52.8s
-> => resolve docker.io/library/node:current-slim@sha256:8ac2083552bb3d92abbbda6baaf9b89b054bfa2cde863d965df9c28ac9bae 0.0s
-> => sha256:9fd6db1a78c9df0fbfadabf130ba0c871adecc55aaf59765d675c756e1f33b05 6.84kB / 6.84kB 0.0s
-> => sha256:f1f26f5702560b7e591bef5c4d840f76a232bf13fd5aefc4e22077a1ae4440c7 31.41MB / 31.41MB 42.6s
-> => sha256:4fc84cbecc3869af9f699f16afadf5d47423a7360e22a607aa9228a7d9e3d192 4.18kB / 4.18kB 0.8s
-> => sha256:c9a1fa43a32686bc7bd4b1cd0dad21098c219a62b4f759e3e1b8a8f8e8c8e1f7 46.69MB / 46.69MB 49.6s
-> => sha256:8ac2083552bb3d92abbbda6baaf9b89b054bfa2cde863d965df9c28ac9baef41 1.21kB / 1.21kB 0.0s
-> => sha256:b3a668a854fb5bde0d6bedb9ac43a292e021789bbbc8d23db69e77b09b2b7c07 1.37kB / 1.37kB 0.0s
-> => sha256:0a3b1fbcd55e7302960f245c30df4e1b0773d4dd3e166cc197cae8c7b02e90da 2.76MB / 2.76MB 3.8s
-> => sha256:a98bb33e812245a91257738fc54a9839ff7949bf75c64308a524b918d6efc1b6 451B / 451B 4.2s
-> => extracting sha256:f1f26f5702560b7e591bef5c4d840f76a232bf13fd5aefc4e22077a1ae4440c7 1.0s
-> => extracting sha256:4fc84cbecc3869af9f699f16afadf5d47423a7360e22a607aa9228a7d9e3d192 0.0s
-> => extracting sha256:c9a1fa43a32686bc7bd4b1cd0dad21098c219a62b4f759e3e1b8a8f8e8c8e1f7 2.3s
-> => extracting sha256:0a3b1fbcd55e7302960f245c30df4e1b0773d4dd3e166cc197cae8c7b02e90da 0.2s
-> => extracting sha256:a98bb33e812245a91257738fc54a9839ff7949bf75c64308a524b918d6efc1b6 0.0s
-> [2/9] WORKDIR /client                                                        1.8s
-> [3/9] COPY /src ./src                                                        0.1s
-> [4/9] COPY index.html .                                                      0.1s
-> [5/9] COPY package*.json .                                                  0.1s
-> [6/9] COPY tsconfig*.json .                                                 0.1s
-> [7/9] COPY vite.config.ts .                                                 0.9s
-> [8/9] COPY /.storybook .                                                    0.1s
-> [9/9] RUN npm i && npm cache clean --force && npm i @esbuild/linux-x64 esbuild-linux-64 && npm run build 67.6s
-> exporting to image                                                            5.3s
-> => exporting layers                                                            5.2s
-> => writing image sha256:47a4090d812efc0883285c9a4c47c45cd2e5d327f7c8d2f54e5638aa58d8cbb0 0.0s
-> => naming to docker.io/library/group-queues-client                          0.0s
```

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.

```
- mysql Pulled 89.3s
- 328ba678bf27 Pull complete 71.8s
- f3f5ff00bd73 Pull complete 71.9s
- dd7054d6dc0c7 Pull complete 72.0s
- 70e504e8750e Pull complete 72.5s
- cd4a7043dd Pull complete 72.6s
- 3c9db061af89 Pull complete 72.7s
- 806ca086fc085 Pull complete 76.6s
- 071b2cebdf832 Pull complete 76.8s
- ad31ba4582eb Pull complete 86.3s
- 0d4c2bd59d1c Pull complete 86.5s
- 148dcef42e3b Pull complete 86.7s

- Network group-queues_group-queues-network Created 0.2s
- Volume "group-queues_mysql-data" Created 0.1s
- Container group-queues-mysql-1 Created 1.1s
- Container group-queues-server-1 Created 0.2s
- Container group-queues-client-1 Created 0.2s
Attaching to group-queues-client-1, group-queues-mysql-1, group-queues-server-1
group-queues-mysql-1 | 2023-04-08 19:43:46+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.32-1.el8 started.
group-queues-mysql-1 | 2023-04-08 19:43:47+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
group-queues-mysql-1 | 2023-04-08 19:43:47+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.32-1.el8 started.
group-queues-mysql-1 | 2023-04-08 19:43:48+00:00 [Note] [Entrypoint]: Initializing database files
group-queues-mysql-1 | 2023-04-08T19:43:48.287132Z 0 [Warning] [MY-011068] [Server] The syntax '--skip-host-cache' is deprecated and will be removed in a future release. Please use SET GLOBAL host_cache_size=0 instead.
group-queues-mysql-1 | 2023-04-08T19:43:48.287448Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.32) initializing of server in progress as process 80
group-queues-mysql-1 | 2023-04-08T19:43:48.920498Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
group-queues-mysql-1 | 2023-04-08T19:43:55.210199Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.

> group-queues@0.1.0 preview
> vite preview

      vvvvvv
    (\_/_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_)
   (/_____/_____\\_____\\___/)
  (//_____|_____|_____|_____|_____)
(//_____|_____|_____|_____|_____)
(//_____|_____|_____|_____|_____)
(//_____|_____|_____|_____|_____)
(//_____|_____|_____|_____|_____)
(//_____|_____|_____|_____|_____)
=====|=====\\=====\\\\
:: Spring Boot ::                (v3.0.4)

2023-04-08T19:43:59.554Z INFO 1 --- [main] r.r.s.StudentsQueueApplication : Starting StudentsQueueApplication v0.1.0 using Java 17.0.2 with PID 1 (/server)
2023-04-08T19:43:59.573Z INFO 1 --- [main] r.r.s.StudentsQueueApplication : No active profile set, falling back to 1 default profile: "default"
2023-04-08T19:44:01.956003Z 6 [Warning] [MY-018453] [Server] root@localhost is created with an empty password! Please consider switching off the --initialize-insecure option.
2023-04-08T19:44:02.655Z INFO 1 --- [main] s.d.c.RetryRepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-04-08T19:44:02.973Z INFO 1 --- [main] s.d.c.RetryRepositoryConfigurationDelegate : Finished Spring Data repository scanning in 269 ms. Found 2 JPA repositories
[Local: http://localhost:80/]
[Network: http://172.23.0.4:80/]
2023-04-08T19:44:08.611Z INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-04-08T19:44:08.683Z INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-04-08T19:44:08.684Z INFO 1 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.5]
```

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.

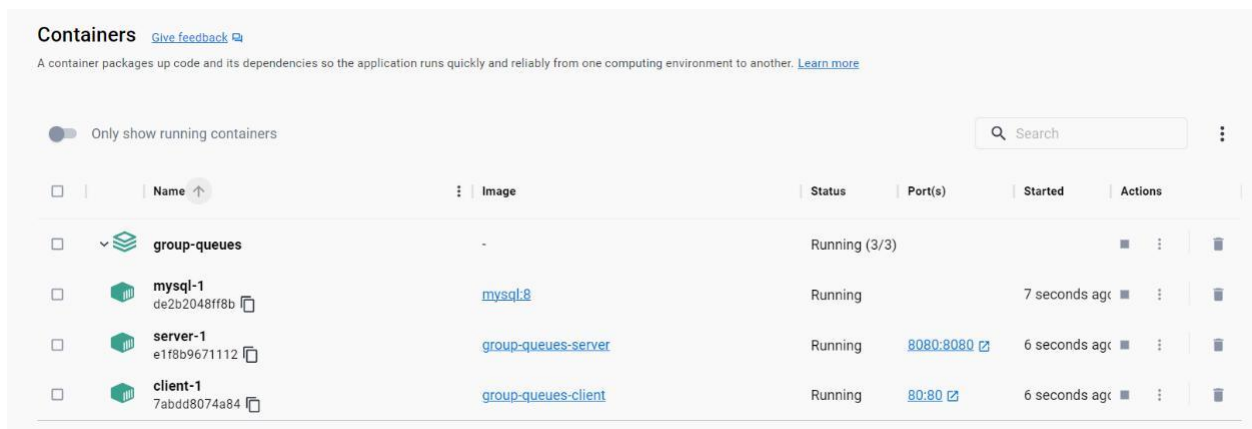


Рисунок 6 – Выполнение команды docker-compose build

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

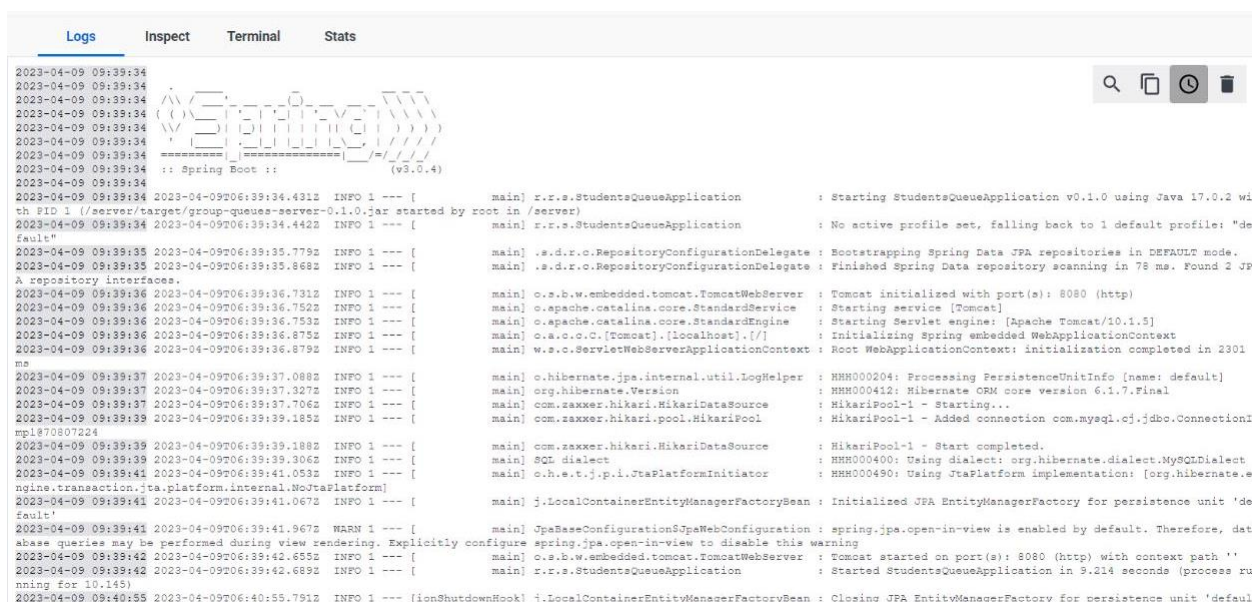


Рисунок 7 – Просмотр logs контейнера group-queues-server-1

1.1.4 Тестирование приложения

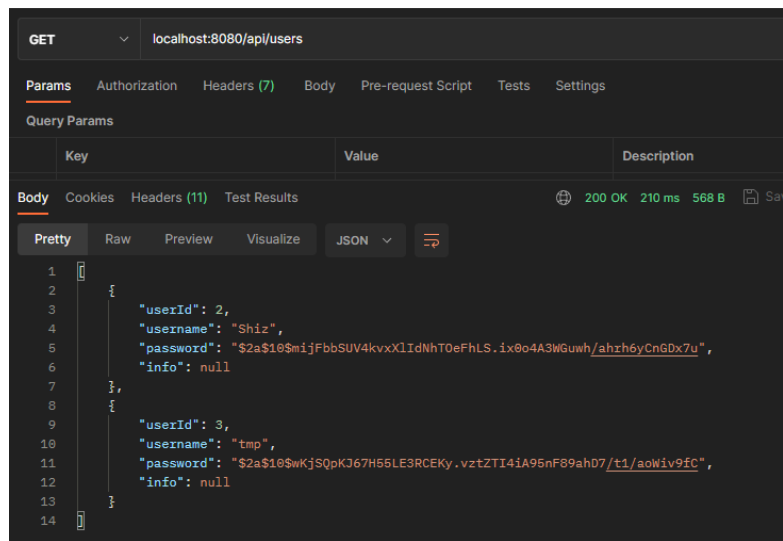


Рисунок 8 - работа серверной части веб-приложения

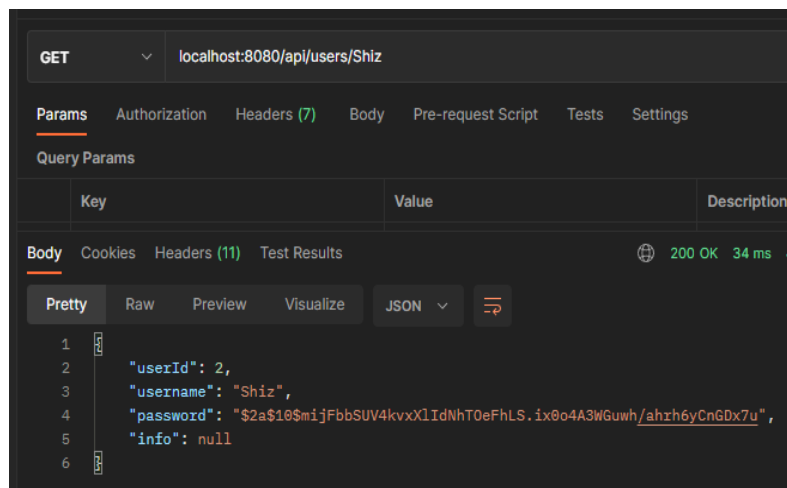


Рисунок 9 - работа серверной части веб-приложения

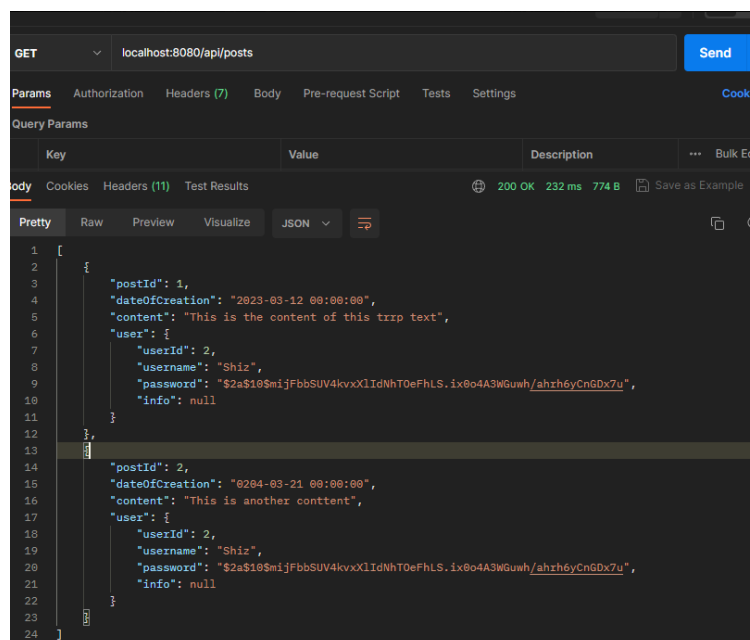


Рисунок 10 - работа серверной части веб-приложения

ВЫВОДЫ

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