Лабораторная работа 8

Подготовка среды к реализации проекта.

1. Создание проекта в github
2. Инициализация образа Postgres в Docker
3. Инициализация образа сервера nodeJS в Docker

1. Создание проекта в github
1.1. В репозитории github создать новый проект. Название проекта должно отражать суть того варианта, который был выбран в лабораторной работе 6.
1.2. На страницу проекта README добавить информацию о названии проекта, общее описание проекта, автор проекта

2. Инициализация образа Postgres в Docker
2.1. Работа с Docker
Преимущества запуска PostgreSQL в Docker
Docker предоставляет элегантный способ запуска баз данных PostgreSQL. Он предлагает несколько очевидных преимуществ по сравнению с традиционными методами установки:

Упрощенная установка и настройка: с Docker вы можете запустить экземпляр PostgreSQL всего одной командой. Больше не нужно возиться с менеджерами пакетов, системными зависимостями или сложными скриптами установки. База данных поставляется с предварительно настроенными удобными параметрами по умолчанию и сразу готова к использованию.
Единообразная среда для всех команд: все участники вашей команды получают абсолютно одинаковую настройку PostgreSQL, независимо от операционной системы. Это устраняет проблему «всё работает на моём компьютере», распространённую в командах разработчиков.
Изоляция от других компонентов системы: контейнеры Docker работают изолированно, что означает, что ваш экземпляр PostgreSQL не будет мешать работе другого программного обеспечения на вашем компьютере. Вы можете запускать несколько версий PostgreSQL одновременно без конфликтов, а удаление PostgreSQL так же просто, как остановка и удаление контейнера. Контроль версий для вашей среды базы данных: Docker позволяет указывать точные версии PostgreSQL в файлах конфигурации. Это означает, что вы можете быть уверены в том, что среды разработки точно соответствуют производственным, и можете тестировать обновления изолированно перед их применением в производственных системах.

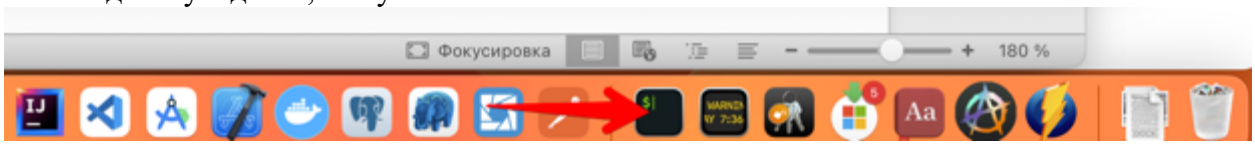
Эффективность использования ресурсов: контейнеры Docker легче виртуальных машин. Они потребляют меньше системных ресурсов и обеспечивают аналогичные преимущества изоляции. Это означает, что вы можете запускать PostgreSQL вместе с другими контейнерами без существенного влияния на производительность системы.

2.2. Установить Docker

Для установки Docker воспользуйтесь инструкцией, например,
Mac https://docs.docker.com/desktop/setup/install/mac-install/
Linux Ubunta https://docs.docker.com/desktop/setup/install/linux/ubuntu/

Windows https://docs.docker.com/desktop/setup/install/windows-install/

2.3 Установка PostgreSQL с помощью одной команды Docker
Далее инструкция дается для Mac
Необходимо убедится, что установлен Terminal или ITerm



Необходимо убедиться, что Docker установлен корректно

Выполните следующую команду в терминале
docker -v

```
okorolyov@MacBook-Pro-Oleg-2 ~ % docker -v
Docker version 20.10.14, build a224086
```

NOTE!! Перед запуском docker следует убедиться, что остановлен сервер PostgreSQL и на порте 5432 нет других запущенных приложений.

Откройте Terminal и введите:

```
docker run -d --name trade-app -p 5432:5432 -e POSTGRES_USER=trade-app
-e POSTGRES_PASSWORD=123 postgres:15-alpine
```

Эта команда выполняет следующие действия:

-d: Запускает контейнер в отсоединённом режиме.
--name: Присваивает имя контейнеру.
-p: Сопоставляет порт контейнера с хост-компьютером. При этом доступ к БД будет осуществляться через стандартный порт 5432.
-e: Устанавливает переменные среды, такие как пароль PostgreSQL.
Образ PostgreSQL 15 будет загружен и запущен в контейнере с именем «trade-app».

После запуска будет выведен примерно такой лог

```
Unable to find image 'postgres:15-alpine' locally
15-alpine: Pulling from library/postgres
2d35ebdb57d9: Pull complete
d7cf304fb91a: Pull complete
01a8e1e8a6d3: Pull complete
c8d6a201b2ea: Pull complete
665050181beb: Pull complete
e7d8b5a29e19: Pull complete
b7a5b6d84454: Pull complete
8e7af31e0abd: Pull complete
0a297d5cb757: Pull complete
78753f2bcd40: Pull complete
9229d0c336e2: Pull complete
Digest: sha256:64583b3cb4f2010277bdd9749456de78e5c36f8956466ba14b0b96922e510950
Status: Downloaded newer image for postgres:15-alpine
```

```
67a39ea83c82e66c2061375c6fbcd5cfbeedd2c2f7c789a974da06c1d8b544b6
```

Проверим, что докер-контейнер запущен. Выполним в терминале команду

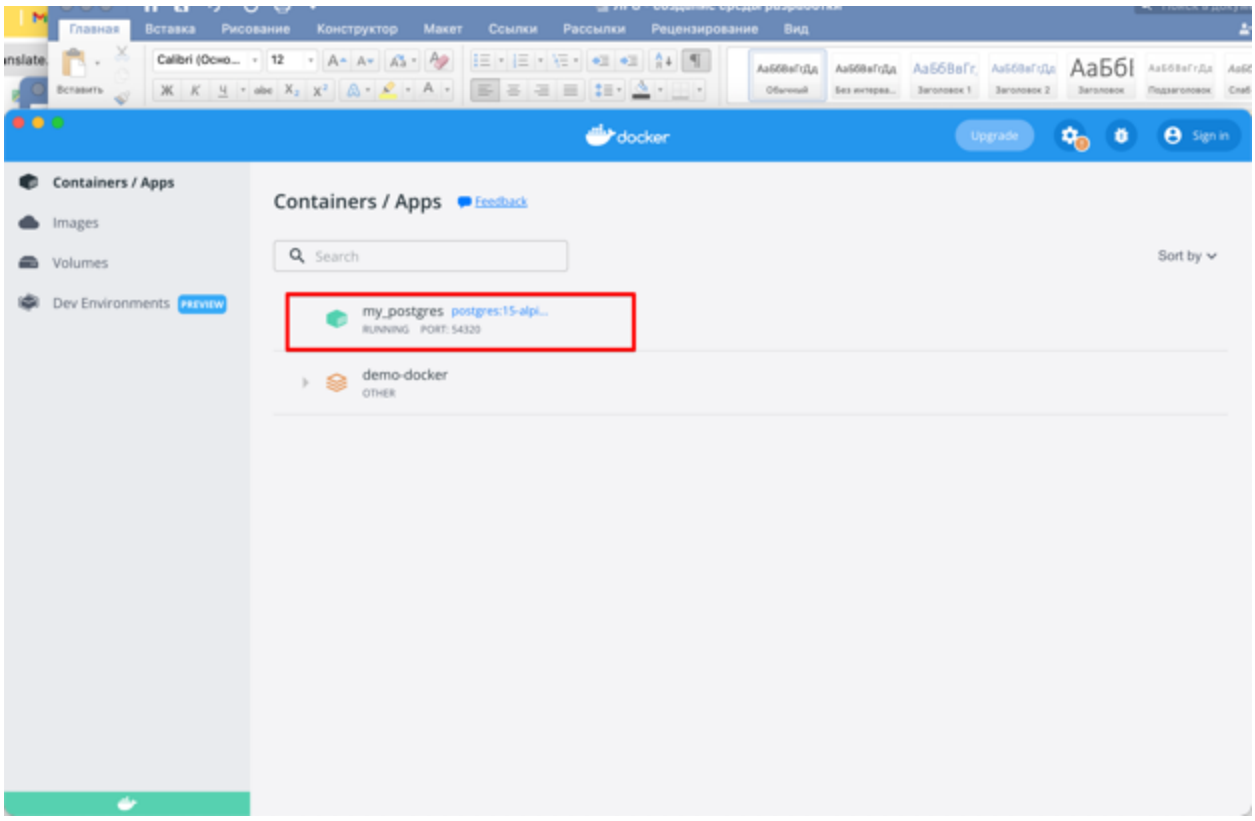
docker ps

Результат

```
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                    NAMES
55a16c4e4bd8   postgres:15-alpine   "docker-entrypoint.s…"   15 seconds ago   Up 14 seconds   0.0.0.0:5432->5432/tcp   trade_app
```

2.4 Доступ к контейнеру Docker PostgreSQL
Теперь Docker работает с PostgreSQL, настроим доступ к БД.

Сначала откройте приложение Docker, и вы увидите запущенный контейнер. Если по какой-либо причине он не запущен, просто нажмите кнопку «Запустить».



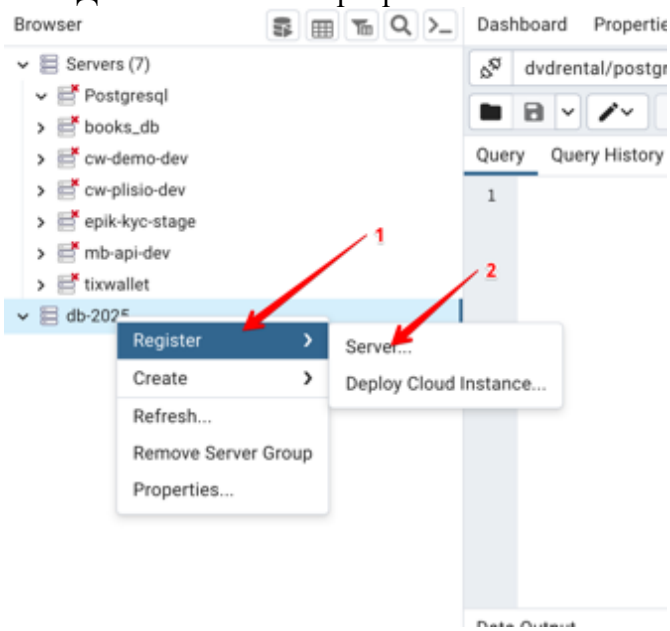
Перейти в детали контейнера

Для взаимодействия с контейнером используйте команду docker exec, а для доступа к PostgreSQL — команду psql. Откройте новое окно терминала и выполните:

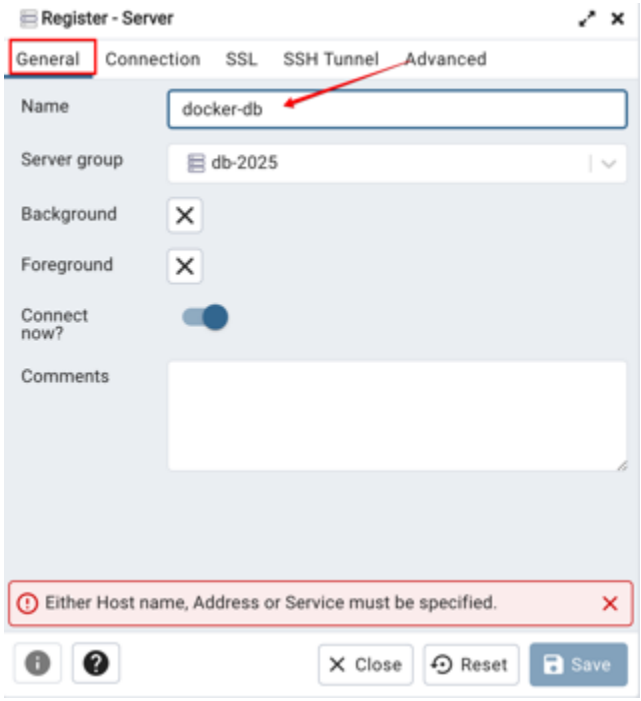
docker exec -it trade_app psql -U postgres

Откроется CLI для работы с БД напрямую из командной строки.

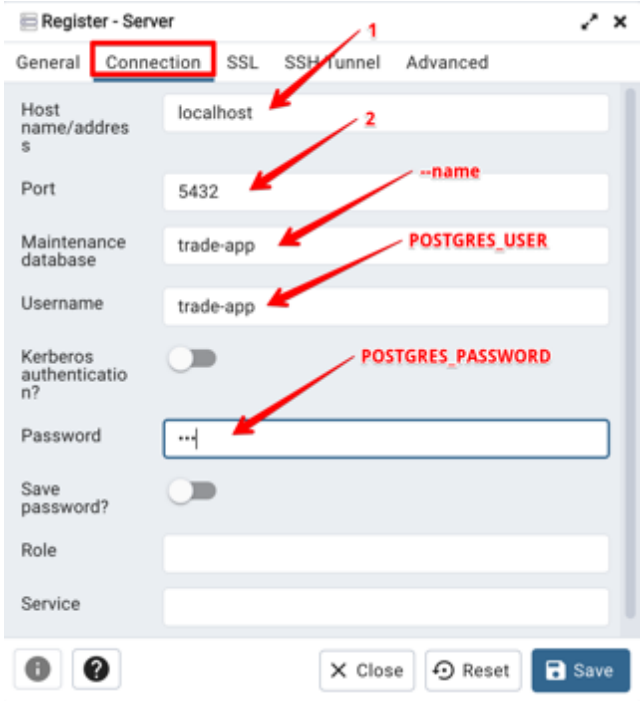
Подключение БД к pgAdmin.
2.4.1 Добавить новый сервер



2.4.2 Заполнить поля подключения
General

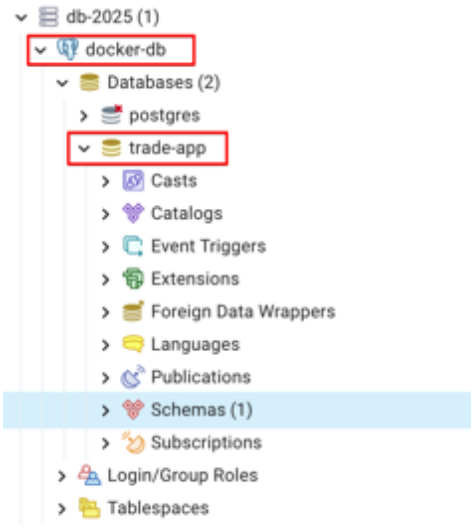


Connection



Вы можете получить доступ к базе данных PostgreSQL прямо из вашего любимого клиента баз данных, например, PgAdmin или DBeaver: используйте порт, имя_базы_данных, имя пользователя и пароль, указанные при создании контейнера Docker.
Port 5432
Database name
User name postgres
User password postgres

Результат



2.5 Альтернативный вариант запуска docker. Использование docker-compose.yml

В рабочем каталоге создайте файл docker-compose.yml

```
version: "3"
services:
  db:
    image: "postgres:15-alpine"
    container_name: "trade-app"
    environment:
      POSTGRES_USER: "trade-app"
      POSTGRES_PASSWORD: "123"
    ports:
      - "5432:5432"
    volumes:
      - my_dbdata:/var/lib/postgresql/data
```

Для запуска контейнера используйте команду

`docker-compose up -d`

3. Инициализация сервера NodeJS в docker

3.1 Необходимо проверить, что NodeJS установлен.

https://nodejs.org/en/download

Чтобы протестировать, наличие nodejs, в терминале выполните команду

node -v

Должна быть выведена версия установленного сервера NodeJS

3.2 Создать структуру проекта

Создать корневую папку trade-project. В ней разместить папку приложения trade-app.
В терминале

```
mkdir trade-project
cd trade-project
mkdir trade-app
cd trade-app
```

Инициализировать проект NodeJs

```
npm init
```

Добавить к проекту одули: библиотеки и фреймворки

```
npm install express sequelize pg pg-hstore cors --save
```

В папке приложения trade-app создать файл server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to trade-app application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

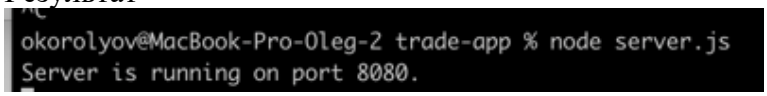
Сохранить файл server.js.

Выполнить команду

```
node server.js
```

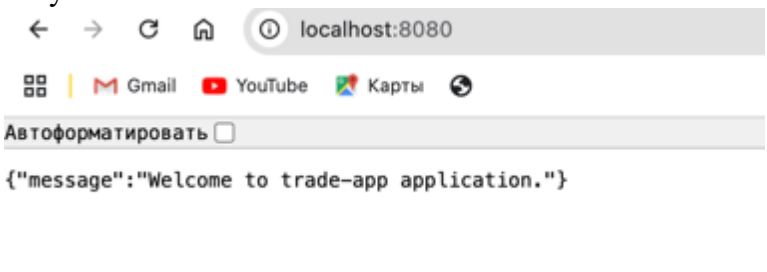
из папки приложения trade-app.

Результат



В браузере в новом окне запросить localhost:8080

Результат



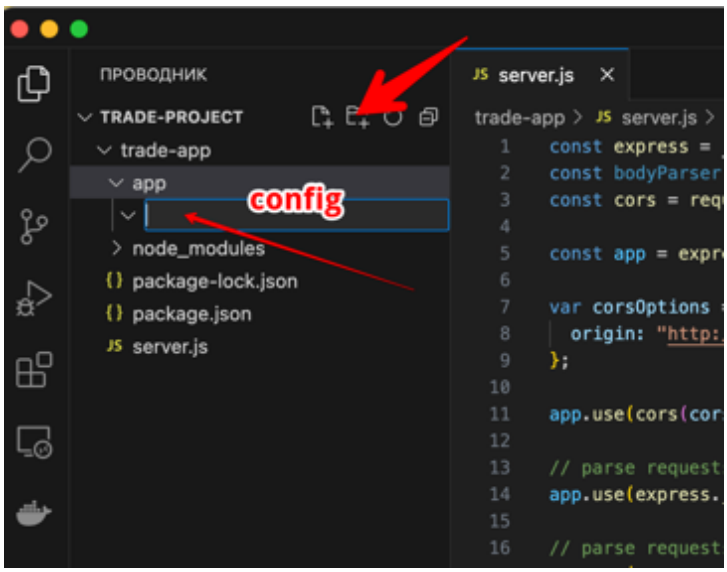
Сервер работает.

3.3 Конфигурация подключения к БД

Открываем проект в любой IDE, например, Visual Studio Code (VS Code). В дальнейшем работа будет описана в VSCode.

Создаем в папке приложения папку app.
В ней папку config и в нем файл db.config.js

```
module.exports = {
  HOST: "localhost",
  USER: "postgres",
  PASSWORD: "123",
  DB: "testdb",
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```



3.4 Инициализация модуля **Sequelize**

Создайте файл app/models/index.js

```
const dbConfig = require("../config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD
, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  operatorsAliases: false,

  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle
  }
});

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.tutorials = require("./tutorial.model.js")(sequelize, Sequelize);

module.exports = db;
```

В файл server.js добавить блок кода

```
...
const app = express();
app.use(...);

const db = require("./app/models");
db.sequelize.sync()
  .then(() => {
    console.log("Synced db.");
```

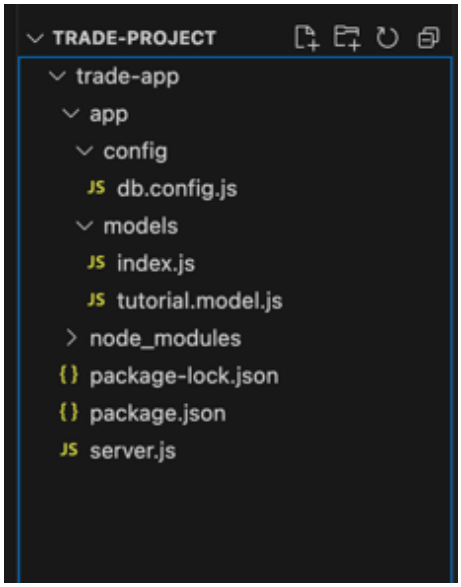
```
  })
  .catch((err) => {
    console.log("Failed to sync db: " + err.message);
  });

...
```

В папке models создать файл-прототип с описание модели данных tutorial.model.js

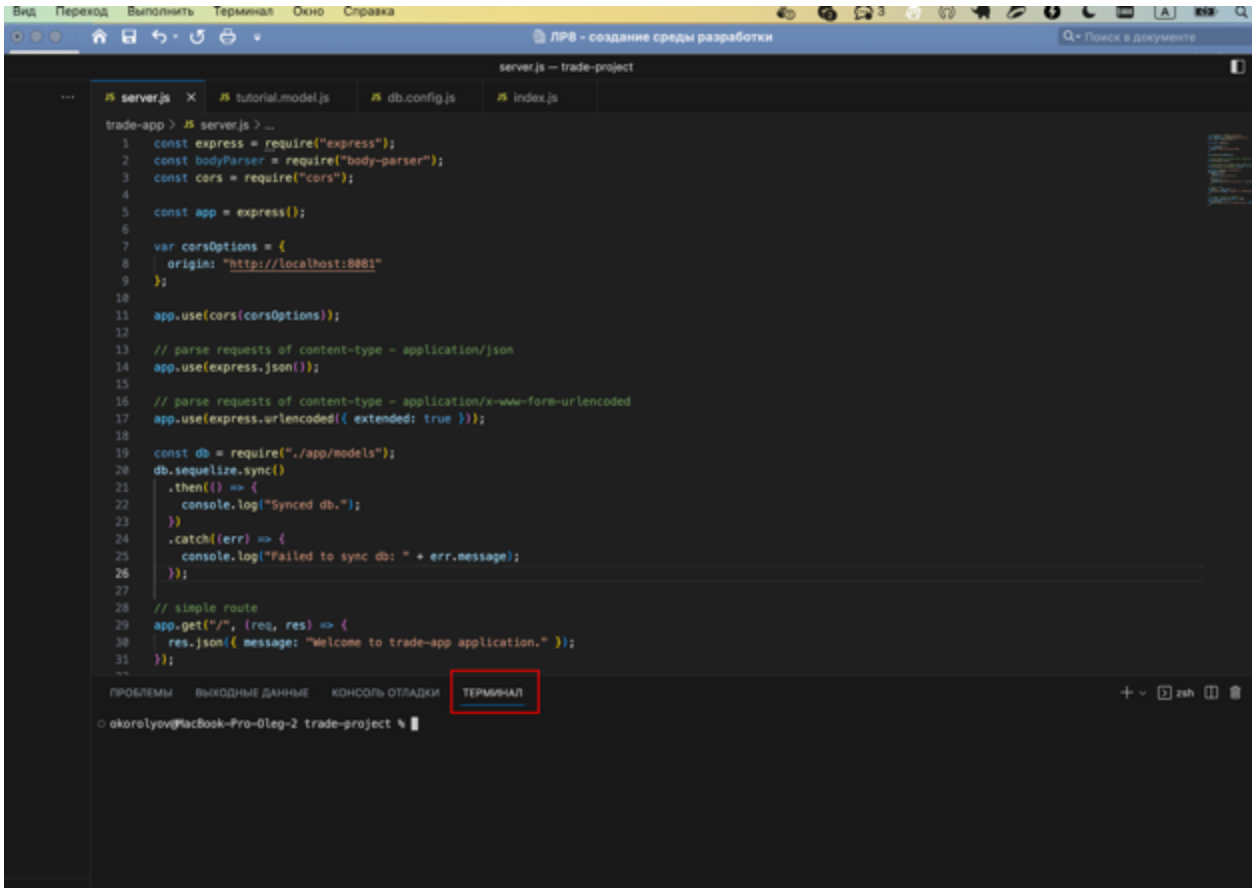
```
module.exports = (sequelize, Sequelize) => {
  const Tutorial = sequelize.define("tutorial", {
    title: {
      type: Sequelize.STRING
    },
    description: {
      type: Sequelize.STRING
    },
    published: {
      type: Sequelize.BOOLEAN
    }
  });

  return Tutorial;
};
```

3.5 Фиксация результата

Структура проекта должна получится такой



В терминале в папке приложения trade-app выполнить команду
NOTE!! Включите окно терминала в VSCode. Вид -> Терминал



node server.js

Результат

```
okorolyov@MacBook-Pro-Oleg-2 trade-app % node server.js
(node:35190) [SEQUELIZE0004] DeprecationWarning: A boolean value was passed to options.operatorsAliases.
This is a no-op with v5 and should be removed.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is running on port 8080.
Failed to sync db: password authentication failed for user "postgres"
```

Подключение к БД не выполнено, потому что конфигурация подключения к серверу не закончена.

3.6 Интеграция контейнеров docker в приложение

3.6.1 Добавим новый модуль в приложение для управления переменными окружения: `dotenv` модуль в *package.json*

```
{
  ...
  "dependencies": {
    "dotenv": "^10.0.0",
    ...
  }
}
```

3.6.2 Импортируем `dotenv` в server.js и используем process.env для установки порта

```
require("dotenv").config();
..
// set port, listen for requests
const PORT = process.env.NODE_DOCKER_PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

3.6.3 Изменяем конфигурацию подключения к базе данных

`app/config/db.config.js`

```
module.exports = {
  HOST: process.env.DB_HOST,
  USER: process.env.DB_USER,
  PASSWORD: process.env.DB_PASSWORD,
  DB: process.env.DB_NAME,
  port: process.env.DB_PORT,
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```

`app/models/index.js`

```
const dbConfig = require("../config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD
, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  port: dbConfig.port,
  operatorsAliases: false,

  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle
  }
});

...
```

Создать новый файл `trade-app/.env.sample`

```
DB_HOST=localhost
DB_USER=trade-app
DB_PASSWORD=123456
DB_NAME=trade-app-db
DB_PORT=5432
```

```
NODE_DOCKER_PORT=8080
```

3.6.4 Создадим Dockerfile

Dockerfile определяет список команд, которые Docker использует для настройки среды приложения Node.js. Помещаем этот файл в папку `trade-app`.

`trade-app/Dockerfile`

```
FROM node:18.18-alpine

WORKDIR /trade-app
COPY package.json .
RUN npm install
COPY . .
CMD npm start
```

Пояснения

FROM: установить образ версии Node.js.
WORKDIR: путь к рабочему каталогу.
COPY: скопировать файл package.json в контейнер (`COPY package.json .`), затем второй вызов (`COPY . .`) скопирует все файлы из каталога проекта.
RUN: выполнить командную строку внутри контейнера: `npm install` для установки зависимостей в `package.json`.
CMD: запустить скрипт `npm start` после сборки образа.

6.3.5 Описание `docker-compose.yml` файла

Подключим Node.js к PostgreSQL с помощью Docker.

В корневом каталоге проекта создадим файл `docker-compose.yml.` Следуем синтаксису версии 3, определённому Docker:

```
version: '3.8'

services:
    postgresdb:
    app:

volumes:
```

Пояснения
`version`: Будет использоваться версия формата файла Docker Compose.
`services`: Отдельные сервисы в изолированных контейнерах. Наше приложение состоит из двух сервисов: app (Nodejs) и postgresdb (база данных Postgres).
`volumes`: Именованные тома, которые сохраняют наши данные после перезапуска.

Определим конфигурацию

`docker-compose.yml`

```
version: '3.8'

services:
  postgresdb:
    image: postgres
    restart: unless-stopped
    env_file: ./.env
    environment:
      - POSTGRES_USER=$POSTGRESDB_USER
      - POSTGRES_PASSWORD=$POSTGRESDB_ROOT_PASSWORD
      - POSTGRES_DB=$POSTGRESDB_DATABASE
    ports:
      - $POSTGRESDB_LOCAL_PORT:$POSTGRESDB_DOCKER_PORT
    volumes:
      - db:/var/lib/postgres
  app:
    depends_on:
      - postgresdb
    build: ./trade-app
    restart: unless-stopped
    env_file: ./.env
    ports:
      - $NODE_LOCAL_PORT:$NODE_DOCKER_PORT
    environment:
      - DB_HOST=postgresdb
      - DB_USER=$POSTGRESDB_USER
      - DB_PASSWORD=$POSTGRESDB_ROOT_PASSWORD
      - DB_NAME=$POSTGRESDB_DATABASE
      - DB_PORT=$POSTGRESDB_DOCKER_PORT
    stdin_open: true
    tty: true

volumes:
  db:
```

Пояснения
– `postgresdb`:
`image`: официальный образ Docker
`restart`: настроить политику перезапуска
env_file: указать путь к файлу `.env`, который мы создадим позже
`environment`: задать настройки с помощью переменных окружения
`ports`: указать порты, которые будут использоваться
`volumes`: сопоставить папки томов

– `app`:
`depends_on`: порядок зависимостей, `postgresdb` запускается перед `app`
`build`: параметры конфигурации, применяемые во время сборки, которые мы определили в Dockerfile с относительным путем
`environment`: переменные окружения, используемые приложением Node
`stdin_open` и `tty`: оставить терминал открытым после сборки контейнера

Следует обратить внимание, что порт хоста (`LOCAL_PORT`) и порт контейнера (`DOCKER_PORT`) различаются. Сетевое взаимодействие между сервисами использует порт контейнера, а внешние устройства — порт хоста.

6.3.6 Определение переменных окружения Docker compose

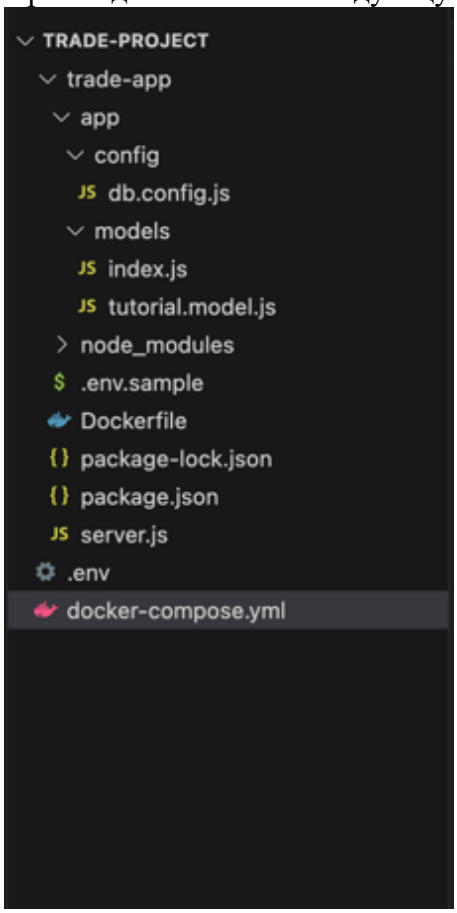
В конфигурации сервиса мы использовали переменные окружения, определённые в файле .env. Теперь приступим к его написанию.

.env

```
POSTGRESDB_USER=postgres
POSTGRESDB_ROOT_PASSWORD=123456
POSTGRESDB_DATABASE=bezkoder_db
POSTGRESDB_LOCAL_PORT=5433
POSTGRESDB_DOCKER_PORT=5432

NODE_LOCAL_PORT=6868
NODE_DOCKER_PORT=8080
```

6.3.7 Проверяем готовность проекта

Проект должен иметь следующую структуру



6.3.8 Запуск приложения

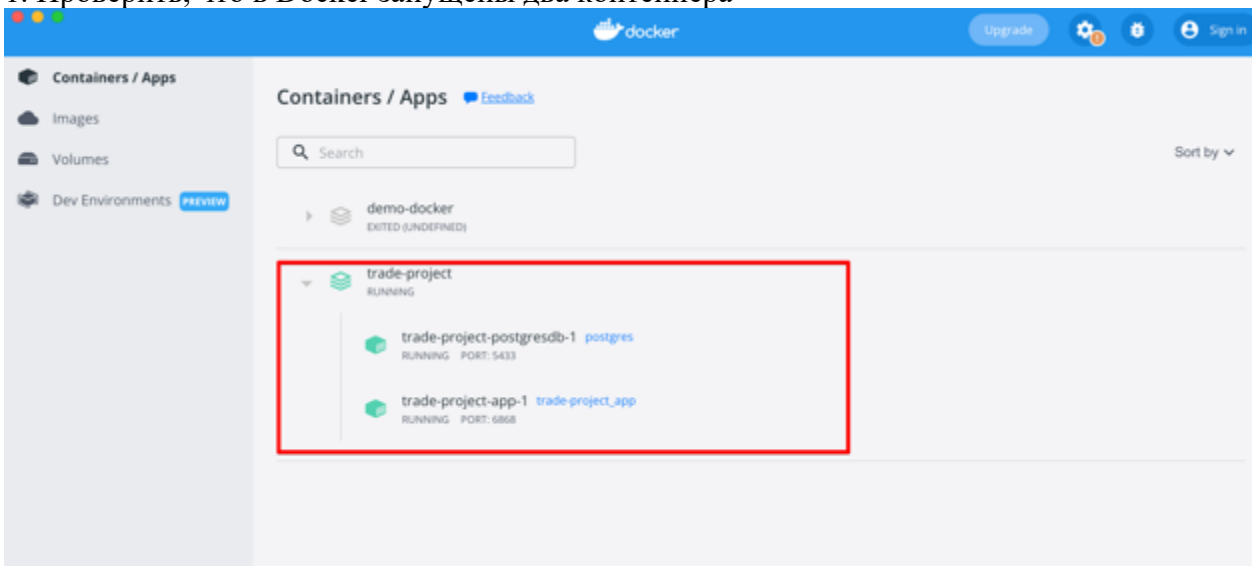
Перед запуском проекта убедитесь, что все изменения сохранены. Используйте команду «Сохранить все изменения».

В терминале из папки проекта trade-project выполнить команду

```
docker compose up –d
```

Результат

1. Проверить, что в Docker запущены два контейнера



2. В браузере открыть ссылку localhost:6868