**Построим REST API с использованием фреймворка Express и СУБД postgres.**

В первую очередь вам понадобится установить node JS на компьютере, если вы этого ещё не сделали, как СУБД установить Postgresql, и, чтобы это дело всё протестировать нам понадобится Postman, чтобы отправлять HTTP запросы к REST API.

Итак, приступим.

Я сделал пустой проект в шторме и сразу же его проинициализировал. Для этого откроем терминал и напишем команду

>npm init -y

Чтобы все настройки стали по умолчанию укажу флаг –y. Должен был создаться файл package.json. И теперь мы можем устанавливать зависимости. Нам понадобится само собой Express и модуль PG. Это модуль для работы с postgres. Их мы установим как основные зависимости.



В качестве зависимости для разработки установим ещё один модуль, для этого укажем флаг –D. Модуль называется nodemon



чуть дальше вы поймёте, для чего он нужен. И, пока модуль устанавливается, создадим index.js файл. Здесь будет запускаться наш сервер, и теперь сделаем базовую конфигурацию, чтобы сервер мог принимать запросы. Для этого импортируем экспресс

const express = require('express');

И зададим порт. Порт будем в первую очередь будем пытаться получить из системных переменных и, если ли такая переменная не задана, то по умолчанию сделаем порт 8090.

const express = require('express');

const PORT = process.env.PORT || 8090;

Далее из экспресса создадим сам сервер

const app = express();

и заставим этот сервер слушать порт. Для этого вызовем команду listen и первым параметром туда передаём порт, а вторым функцию callback которая будет просто в логе выводить новую информацию о том что сервер был запущен на таком-то порту

app.listen(PORT, () => console.log(`server started on port ${PORT}`));

Теперь откроем package.json и сделаем скрипт запуска сервера. Для этого находим поле scripts, заменяем на start и тут пишем команду

  "scripts": {

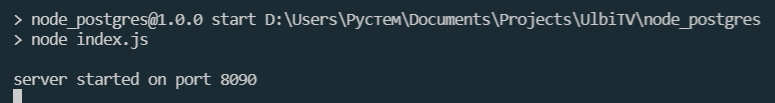
    "start": "node index.js",

  },

В терминале пишем



и если вы увидели надпись консоли, что сервер запустился, значит всё хорошо.



Давайте теперь сделаем первый маршрут, по которому мы сможем отправлять запросы. Вызываем функцию, которая первым параметром принимает как раз путь, а вторым функцию callback, которая принимает также параметрами запрос и ответ. И у запроса вызываем функцию send чтобы отправить на клиент какие-то данные. В нашем случае это просто текст.

const express = require('express');

const PORT = process.env.PORT || 8090;

const app = express();

app.get('/', (req, res) => {

    res.send('HELLO POSTGRES + NODEJS')

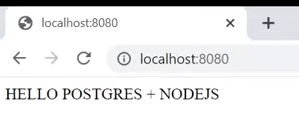
})

app.listen(PORT, () => console.log(`server started on port ${PORT}`));

Теперь попробуем перейти по url  в браузере и увидим что сервер не может обработать запрос, поскольку мы функцию написали а сервер не перезапустили



Обновим страницу и сервер обрабатывает запрос



Но есть некая сложность. Если мы будем изменять как-то код, каждый раз придётся перезапускать сервер. Именно для этого мы установили nodemon. Он необходим только как dev зависимость на этапе разработки. Поэтому создадим ещё один скрипт, который будет запускать index.js-сервер, но уже через nodemon.

  "scripts": {

    "start": "node index.js",

    "dev": "nodemon index.js"

  },



При внесении изменений в код и сохранении сервер автоматически перезапускается.

Теперь необходимо сконфигурировать базу данных.

Создаём соответствующий файл db.js, и здесь первый же строчкой получаем Pool из пакетов, которые устанавливали.

const Pool = require('pg').Pool;

Поскольку pool – это класс, нам необходимо создать объект этого класса. Именно с помощью него мы будем делать запросы к базе данных, и, в конце концов, мы его экспортируем, чтобы использовать в других файлах.

const pool = new Pool;

module.exports = pool;

Далее в конструктор передадим объект, в котором будем писать некоторые настройки.

const pool = new Pool({

    user: 'postgres',

    password: 'root',

    host: 'localhost',

    port: 5434,

    database: 'node\_postgres'

});

Первое поле – это пользователь, от которого мы будем подключаться к базе данных. Здесь пишем postgres.

Теперь пароль. Его вы скорее всего указывали при установке СУБД, в моём случае это root.

Следующим этапом надо указать хост, в нашем случае это локальная машина, поэтому localhost.

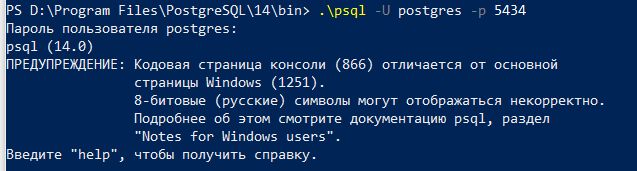
И порт. По умолчанию при установке он указывается 5432. Если вы меняли, то соответственно указывает тот, на который вы меняли. И последним свойством указываем название базы данных.

Она у нас конечно же ещё не создана. Поэтому необходимо открыть Postgres и создать её. По умолчанию он устанавливается на диск C\Program Files\PostgreSQL\, внутри лежит папка 14 и папка bin. В этой папке открываем Powershell и командой



убеждаемся в том, что он установлен.

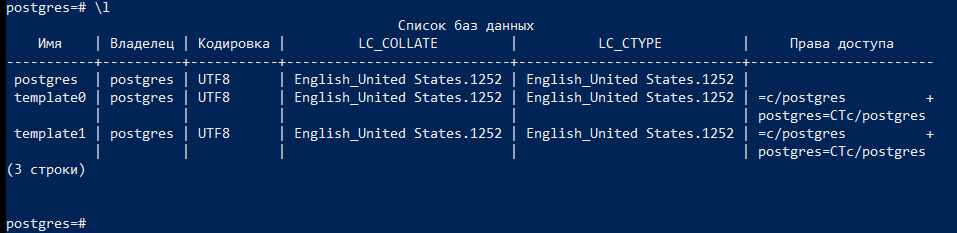
Далее заходим в СУБД под пользователем postgres, пишем порт и пароль, который вы указывали при установке.



Для решения проблемы с кодировкой введем



Команда \l покажет список существующих баз данных

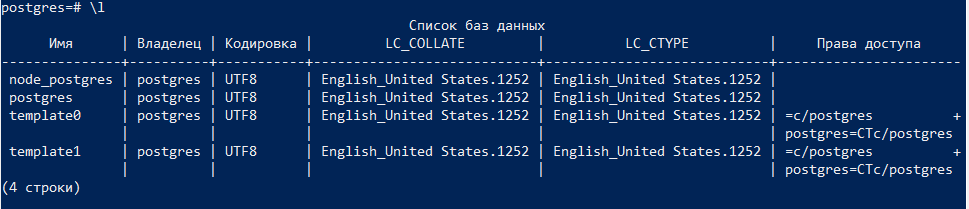


Cоздадим новую. Назовем ее node\_postgres. Для этого напишем команду



Не забываем в конце поставить точку с запятой.

Командой \l убеждаемся в том, что она была создана.



Подключимся к созданной базе данных. Для этого пишем \connect и соответственно название самой БД.

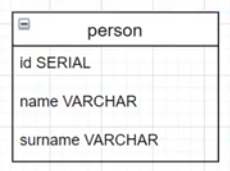


Команда \dt покажет существующие в базе данных таблицы. На данный момент они отсутствуют.

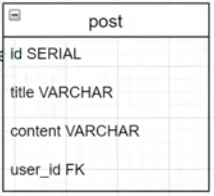


Следующим этапом определимся, какие в принципе сущности будут в нашей базе данных.

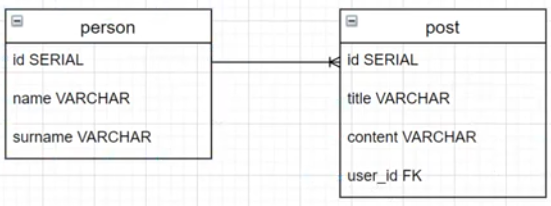
Сэмулируем сайт ведения блога. Будет какой-то человек и этот человек может создать какое-то количество постов. То есть у нас есть его сущность person, вторая сущность это post. Теперь давайте разбираться, какие здесь будут поля. Обязательно уникальный идентификатор id SERIAL, то есть это поле будет числовым значением, но при создании каждого нового объекта оно будет инкрементироваться. Соответственно пользователю добавим имя и фамилию. Это будут уже строковые значения.



У поста также будет id и 2 строковых значения, это название и какая-то контентная часть. Это будет только текст. И, так как один пользователь может иметь несколько постов, необходимо определить внешний ключ. То есть таблица post будет ссылаться на таблицу person. Для этого создадим ещё одно поле, назовём его user\_id.



Ну и теперь наглядно покажем, что связи между таблицами один ко многим. То есть на одного человека приходятся несколько постов.



Теперь давайте создадим уже в БД соответствующие таблицы. Для этого вернемся в шторм и создадим здесь файлик с расширением .sql. Здесь мы напишем sql-запросы.

Начнём с первой таблицы - таблицы пользователя.

Для этого напишем команду CREATE TABLE название таблицы. ID имеет тип SERIAL и это первичный ключ. Имя, фамилия это строковые значения. Таблица полностью соответствуют диаграмме, которую мы создавали

create TABLE person(

    id SERIAL PRIMARY KEY,

    name VARCHAR(255),

    surname VARCHAR(255)

);

Ну, и сразу же напишем скрипт для создания таблицы с постами. Здесь у нас в принципе всё тоже самое, заменим только название и добавим ещё одно поле с внешним ключом, который будет ссылаться на таблицу person, назовём его user\_id и укажем, что тип у него числовой, и также необходимо пометить, что это внешний ключ: на следующей строчке напишем FOREIGN KEY, в скобках укажем название поля, и далее надо указать, на какую таблицу будет идти ссылка - ссылка идёт на таблицу person на поле id.

create TABLE post(

    id SERIAL PRIMARY KEY,

    title VARCHAR(255),

    content VARCHAR(255),

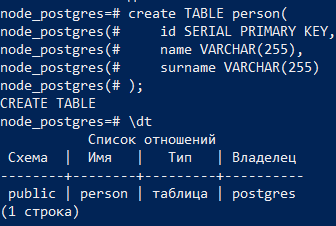
    user\_id INTEGER,

    FOREIGN KEY (user\_id) REFERENCES person (id)

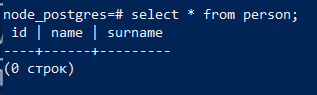
);

Для многих зрителей, я уверен, это информация покажется слишком простой, но я должен это показать так же и для начинающих зрителей.

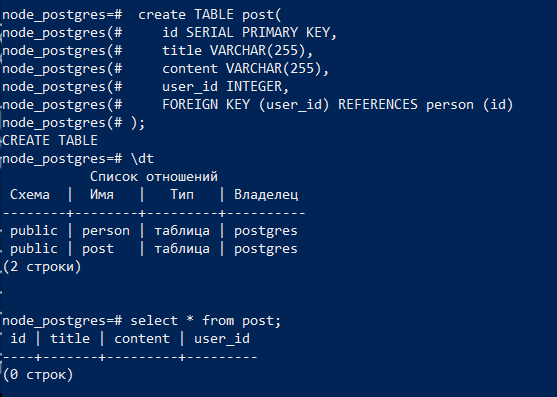
Итак, скрипт мы написали. Теперь перейдём к созданию таблицы. Возвращаемся в консоль и вставляем сюда первый скрипт. Командой \dt убедимся, что таблица создана.



Для того. чтобы получить какие-то данные из таблицы, используется команда Select. \* говорит о том, что мы хотим получить все поля.



Создадим вторую таблицу: также копируем скрипт возвращаемся в powershell и вставляем его. Убеждаемся в том, что таблица создалась, и так же сделаем select запрос, чтобы посмотреть, какие поля там у нас есть.



Итак, таблицы созданы. Теперь мы можем с этим работать.

Создадим папку controller – в ней мы будем работать как раз непосредственно с запросами. И папку roots - там будут уже соответственно маршруты, по которым надо будет отправлять запросы. И в каждой папке создаем одноименные js-файлы.

Начнём с контроллера. Создадим класс, внутри которого будем определять методы.

class UserController {

};

Эти методы будут показывать то, что будет уметь наше приложение. И первый метод будет «Создать пользователя». Параметрами функции будут принимать запрос request и ответ response. Вторая функция будет возвращать клиенту всех пользователей, третья функция будет возвращать конкретного какого-то пользователя по его уникальному идентификатору, четвертая функция будет обновлять данные о пользователе и пятая – удалять. Полноценный CRUD цикл.

class UserController {

    async createUser(req, res) {

    };

    async getUser(req, res) {

    };

    async getOneUser(req, res) {

    };

    async updateUser(req, res) {

    };

    async deleteUser(req, res) {

    };

}

И чтобы использовать этот контроллер в роутах, нам необходимо экспортировать объект этого контроллера.

module.exports = new UserController();

9 мин 40 сек

Далее возвращаемся к роутам и получаем из Express’а роутер. И так как это класс, то мне необходимо создать объект этого класса.