NodeJS – авторизация (на примере репозитория node)

1.Пакеты

"dependencies": {

    "bcryptjs": "^2.4.3", - шифрование паролей

    "cors": "^2.8.5", - работа с CORS

    "dotenv": "^16.3.1", - работа с фаилом .env

    "express": "^4.18.2",

    "express-validator": "^7.0.1",- валидация данных в body запроса

    "jsonwebtoken": "^9.0.2", - JWT токены

    "mongoose": "^8.0.4", - ОРМ (или ОДМ) для работы с БД Mongo

    "pg": "^8.11.3",

    "pg-hstore": "^2.3.4",

    "sequelize": "^6.35.2", - ОРМ

  },

  "devDependencies": {

    "nodemon": "^3.0.2" - пересборка проекта при сохранении изменений

  }

Так же

express-fileupload – пакет для возможности получать передавать фаилы/картинки (подключается через app.use())

uuid – для генерации рандомных имен

express-fileupload – работа с загрузкой фаилов

2. Этапы написания проекта

1. Установить POSTMAN и зарегистрироваться в mongo и создать там БД (предварительно разрешить доступ с любого IP)

2. В index.js поднимаем express сервер и коннектиктимся с mongo

3. Создаем папку models в ней будут храниться все схемы по структурированию коллекций в БД

3. Создаем папку с роутами и в ней добавляем фаилы с роутами (в частности authRouter.js) .

Каждый роут будет включать путь, мидлвару (если нужна) и контроллер – он будет выполнять все основные действия(получать обрабатывать данные и формировать ответ)

Далее эти роуты подключаем в index.js через

app.use("/", myRouter);

4. Создаем контроллеры authController.js (можно как на классах так на функциях).

В нем будут 3 метода – регистрация, логин, и получение пользователей(тестово).

При **регистрации** проверяем если ли такой пользователь в БД. Если есть – ошибка. Если нет – шифруем его пароль и сохраняем пользователя в БД.

При **логине** берем логин пароль пользователя – проверяем по логину – если такой пользователь. Если есть – проверяем соответсвие пароля тому который сохранен в базе. Если все ок – генерируем JWT токен и отправляем его на фронт

При получении пользователей (реализация функционала когда только **авторизованные** пользователи либо определенные **роли** могут иметь **доступ** к этому) – контроллер пишется как обычно, а весь функционал по доступу будет реализован в middleware которые прокинутся в роутер 2ым аргументом. Суть – мидлвара всегда принимает (запрос, ответ, следующая функция) и должна вызывать следующую функцию. Так же если нужны доп параметры – можно использовать функцию которая возвращает мидлвару (пример реализован в авторизации).

Доступ зарегестрированных пользователей – забираем из заголовков токен – декодируем получаем логин и сравниваем с БД пользователей – если есть – все ок, нет – ошибка.

Доступ по ролям - забираем из заголовков токен – декодируем получаем роль и сравниваем с БД пользователей – если есть – все ок, нет – ошибка.

Авторизация на клиенте.

При логине бэк присылает токен – он сохраняется на фронте в локалсторадж и переменная isAuth=true. На каждый новый запрос на бэк интерцептором добавляем токен в заголовок запроса – таким образом бэк понимает что пользователь авторизован. Если же бэк возвращает ошибку по проверке токена – на фронте все обнуляется (локалсторадж, данные пользователя в стектменеджере, там же переменная isAuth) и редиректится на страницу авторизации.

Работа со статическими фаилами (картинками и т д)

Картинка приходит с фронта – ее получаем как

Const {img} = req.files;

Генерируем ей имя через пакет uuid

Создаем папку static в корне

Сохраняем

img.mv(path.resolve(\_\_dirname, '..', 'static', newFileName))

Для того что бы с фронта можно было получать и отдавать статические фаилы, в индекс добавляем

app.use(fileupload({}));

app.use(express.static(path.resolve(\_\_dirname, ‘static’)))