ТЕОРИЯ

JWT – JSON WEB TOKEN –закодированная строка, состоящая из 3 частей.

1 – заголовок, 2 – данные, 3 – сигнатура, разделены точкой.

Токены: access, refresh

Каждый jwt токен имеет срок своей жизни, через какое-то время он становится невалиден. Это делается для безопасности, если кто-то завладеет токеном, он сможет пользоваться сайтом только на время жизни токена

Access токен – токен доступа. Живет 15-30 минут. Используется для доступа к сервису. Храним в LocalStorage

Refresh токен – токен, который обновляет Access токен. Живет 15-60 дней. Храним в httpOnlyCookie

Как выглядит работа jwt

Мы пытаемся ойти, вводим данные, в теле запроса указываем эти данные и отправляем на сервер. После этого сервер генерирует пару токенов: access, refresh. Access токен мы сохраняем в localStorage, а refresh токен сервер устанавливает в Cookie.

Мы залогинились и совершаем какие-то действия, например чат. Отправляем запрос на сервер, в запросе мы должны установить заголовки, где должны указать authorization: ${accessToken}. Отправляем запрос.

Первое, что делает сервер – проверяет access токен.

Если токен валиден, то сервер присылает статус код 200 и присылает ответ на наш запрос.

Если токен не валиден – сервер возвращает статус код 401 и ошибку unauthorized.

Мы должны предусмотреть такой вариант исхода и на клиенте в обработчике ошибок мы должны отправить запрос на обновление access токена. В cookie у нас уже находится refreshToken, поэтому добавлять его никуда не надо. Отправляем запрос. Сервер его обрабатывает и возвращает новую пару токенов access и refresh.

ПРАКТИКА

1. Устанавливаем реакт приложение с названием client **npm create vite@latest**
2. Создаем папку server, переходим в нее и инициализируем проект командой **npm init –y**
3. Устанавливаем доп зависимости **npm i express cors cookie-parser** (express - фреймворк для Node.js, который помогает упростить создание серверных приложений, cors - это механизм безопасности, который контролирует, какие сайты могут делать запросы к серверу, cookie-parser - это middleware для Express, который упрощает работу с cookies (куки-файлами) в Node.js приложениях. Он позволяет парсить cookies, которые отправляются с запросами от клиента, и делает их доступными через req.cookies)
4. Также установим nodemon, чтобы сервер перезапускался при изменениях в коде **npm i nodemon --save-dev**
5. Делаем script для запуска сервера в package.json:

"scripts": {

  "dev": "nodemon index.js"

  },

1. Далее создаем базовый каркас экспресс-приложения

const express = require("express");

const cors = require("cors");

const cookieParser = require("cookie-parser");

const PORT = 5000;

const app = express();

const start = async () => {

  try {

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

  } catch (error) {

    console.log(error);

  }

};

start();

1. Устанавливаем dotenv - это популярный пакет для Node.js, который позволяет хранить конфиденциальные или специфичные для среды переменные в файле .env и загружать их в приложение (**npm i dotenv**)
2. Создаем файл .env в корне сервера и пишем туда переменную порта

PORT = 5000;

1. Переходим в index.js. Импортируем dotenv и заменяем порт на переменную из окружения.

require('dotenv').config();

const PORT = process.env.PORT;

1. Устанавливаем зависимости для БД MongoDB: **npm i mongodb mongoose**
2. Создаем проект на сайте mongodb.com, копируем там полученный url, прописываем в .env полученный url

DB\_URL = mongodb+srv://root:1234@cluster0.1oox1.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

1. Подключаемся к БД:

const mongoose = require("mongoose");

В блоке try функции старт прописываем подключение к БД

const start = async () => {

  try {

    await mongoose.connect(process.env.DB\_URL);

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

  } catch (error) {

    console.log(error);

  }

};

1. Подключаем необходимые мидлварки

app.use(express.json());

app.use(cookieParser());

app.use(cors());

**app.use(express.json());** — это middleware в Express, который парсит JSON-запросы.

Когда клиент отправляет запрос с типом контента application/json, эта строка кода позволяет Express автоматически распознавать и обрабатывать тело запроса в формате JSON, превращая его в объект JavaScript, доступный через req.body.

**app.use(cookieParser());** — это middleware в Express, который используется для парсинга cookies из HTTP-запросов.

Когда клиент отправляет запрос с cookies (например, сессии, предпочтений или других данных), cookieParser помогает Express автоматически извлечь эти cookies из заголовков запроса и сделать их доступными через объект req.cookies.

**app.use(cors());** — это middleware для обработки **CORS (Cross-Origin Resource Sharing)** в Express.

CORS — это механизм безопасности, который ограничивает, какие ресурсы с других доменов могут быть запрашиваемыми на твоем сервере. Например, если твой фронтенд (клиентская часть) работает на домене frontend.com, а сервер (API) находится на api.com, то браузер по умолчанию блокирует такие запросы, чтобы предотвратить возможные атаки.

Когда ты добавляешь app.use(cors());, ты разрешаешь запросы с других доменов (или всех, если не настроены ограничения).

1. Создаем папки в корне проекта: controllers, models, service, router
2. Опишем схему данных в моделях БД
3. Создадим роутинг.
4. Создаем каркасы контроллеров и сервисов.
5. Устанавливаем зависимости **npm i jsonwebtoken bcrypt uuid**