const express = require('express'); // Импортируем Express — основной фреймворк для создания сервера

const cookieParser = require('cookie-parser'); // Импортируем middleware для работы с куками

const bcrypt = require('bcryptjs'); // Импортируем bcryptjs для хэширования паролей

const jwt = require('jsonwebtoken'); // Импортируем jsonwebtoken для создания и верификации JWT

const app = express(); // Создаём экземпляр приложения Express

app.use(express.json()); // Подключаем middleware для парсинга JSON из тела запросов

app.use(cookieParser()); // Активируем парсер кук

const JWT\_SECRET = 'your\_secret\_key'; // Секретный ключ для подписи JWT (обязательно замените на уникальный)

const users = []; // Примитивное хранилище пользователей (вместо настоящей БД)

// Генерация JWT токена для пользователя

function generateToken(user) {

return jwt.sign(

{ username: user.username }, // Данные, которые запишем в токен (payload)

JWT\_SECRET, // Секрет для подписи токена

{ expiresIn: '1h' } // Время жизни токена — 1 час

);

}

// Middleware для проверки аутентификации пользователя

function authMiddleware(req, res, next) {

const token = req.cookies.token; // Получаем токен из куки

if (!token) return res.status(401).json({ message: 'Нет токена' }); // Если нет токена, "не авторизован"

try {

const decoded = jwt.verify(token, JWT\_SECRET); // Проверяем валидность токена и извлекаем payload

req.user = decoded; // Сохраняем данные пользователя в req.user для доступа в обработчиках

next(); // Передаём управление следующей функции в цепочке

} catch (err) {

return res.status(401).json({ message: 'Неверный токен' }); // Ошибка — токен невалиден/протух

}

}

// Эндпоинт регистрации пользователя

app.post('/register', async (req, res) => {

const { username, password } = req.body; // Получаем username и password из тела запроса

if (users.find(u => u.username === username)) { // Проверяем — есть ли уже такой пользователь

return res.status(400).json({ message: 'Пользователь уже существует' }); // Возвращаем ошибку, если есть

}

const hashedPassword = await bcrypt.hash(password, 10); // Хэшируем пароль (10 — это "соль", степень хэширования)

users.push({ username, password: hashedPassword }); // Сохраняем нового пользователя с хэшем пароля

res.json({ message: 'Регистрация успешна' }); // Отправляем ответ об успешной регистрации

});

// Эндпоинт авторизации (логина)

app.post('/login', async (req, res) => {

const { username, password } = req.body; // Получаем username и password из запроса

const user = users.find(u => u.username === username); // Ищем пользователя по имени

if (!user) return res.status(400).json({ message: 'Неверные данные' }); // Не найден — ошибка

const isPasswordValid = await bcrypt.compare(password, user.password); // Сравниваем введённый пароль с хэшем

if (!isPasswordValid) return res.status(400).json({ message: 'Неверные данные' }); // Не совпало — ошибка

const token = generateToken(user); // Генерируем jwt токен

res.cookie('token', token, { httpOnly: true }); // Отправляем токен клиенту в httpOnly cookie

res.json({ message: 'Успешная авторизация' }); // Оповещаем об успехе

});

// Проверка “авторизован ли пользователь” (endpoint только для авторизованных)

app.get('/me', authMiddleware, (req, res) => {

res.json({ message: 'Авторизован', user: req.user }); // Отправляем, что пользователь авторизован, вместе с его данными

});

// Эндпоинт выхода пользователя (logout)

app.post('/logout', (req, res) => {

res.clearCookie('token'); // Очищаем куку с токеном (выход)

res.json({ message: 'Выход выполнен' }); // Сообщаем клиенту, что выход совершен

});

// Запускаем сервер на порту 3000

app.listen(3000, () => console.log('Server started'));

\*\*1. Для чего используется `SECRET` и какое значение задавать\*\*

- \*\*Назначение:\*\*

`SECRET` (точнее, `JWT\_SECRET` в коде) используется для \*подписи и проверки подлинности\* JWT-токенов. Это секретный ключ, благодаря которому можно удостовериться, что токен действительно был сгенерирован вашим сервером, а не кем-то извне. Без него нельзя создать или подделать корректный токен.

- \*\*Какое значение задать:\*\*

- Должно быть \*\*уникальное\*\* и \*\*сложное значение\*\*, содержащее буквы, цифры и специальные символы.

- Не должно быть публичным или простым способом вычисляемым (например, не "123", "secret", "password").

- В продакшене его обычно хранят в переменных окружения:

export JWT\_SECRET="a\_really\_long\_and\_secure\_random\_string"

- Пример:

const JWT\_SECRET = 'W91G8$%t4hLk@123qweZXCASD!jklm,./';

\*\*2. Для чего используются `app.use(express.json())` и `app.use(cookieParser())`\*\*

- \*\*`app.use(express.json())`:\*\*

- Позволяет Express автоматически \*парсить (разбирать) входящие JSON-данные\* из тела HTTP-запроса.

- Например, когда клиент отправляет через POST на `/register`

{ "username": "pete", "password": "12345" }

- Express без этого middleware не увидит данные в `req.body`.

- С этим middleware данные автоматически доступны через `req.body`.

- \*\*`app.use(cookieParser())`:\*\*

- Этот middleware позволяет \*легко работать с куками, которые пришли в запросе пользователя\*.

- После подключения все куки из заголовка запроса доступны в объекте `req.cookies`.

- Это упрощает доступ к JWT-токену, который мы кладём в cookie.

\*\*3. Как генерируется токен и как используется\*\*

- \*\*Генерация токена:\*\*

- JWT-токен создаётся через функцию

jwt.sign({ username: user.username }, JWT\_SECRET, { expiresIn: '1h' })

- \*Первый параметр\* – это \*\*payload\*\* (какие данные о пользователе вы кладёте в токен), здесь это `username`.

- \*Второй параметр\* – ваш секретный ключ (`JWT\_SECRET`).

- \*Третий параметр\* – опции, например, срок действия токена.

- \*\*Использование токена:\*\*

- \*\*После создания:\*\*

Токен помещается в cookie:

res.cookie('token', token, { httpOnly: true });

- После этого браузер будет автоматически отправлять куку с каждым запросом к серверу.

- \*\*При проверке авторизации:\*\*

- Миддлварка `authMiddleware` достаёт токен из cookie:

const token = req.cookies.token;

- Затем валидирует его:

const decoded = jwt.verify(token, JWT\_SECRET);

- Если токен корректный – пользователь считается авторизованным, его данные доступны в `req.user`.

\*\*Кратко\*\*

- `SECRET` — ваш супер-важный ключ для подписи токенов, держите его в секрете.

- `express.json()` — чтобы работать с JSON-запросами, `cookieParser()` — чтобы работать с куками.

- Токен — это "пропуск", который после логина кладём в куку; каждый раз сервер проверяет этот пропуск, чтобы понять, кто вы и авторизованы ли вы.

\*\*Итог:\*\*

Эти части отвечают за безопасность (SECRET), работу с данными (JSON, cookies) и авторизацию (токены). Это фундамент для построения безопасности в Express-приложениях.