# Практика 2

# 1 Создание проекта

Первым шагом является установка Node.js, если вы еще не сделали этого. Можете скачать установщик с официального сайта Node.js и следовать инструкциям по установке для вашей операционной системы.

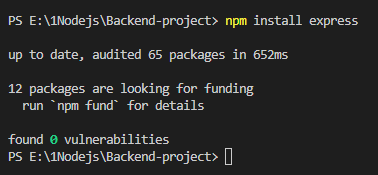
Создайте директорию проекта (папку) в которой можно сразу создать файл index.js.

В VScode выберите созданную папку.

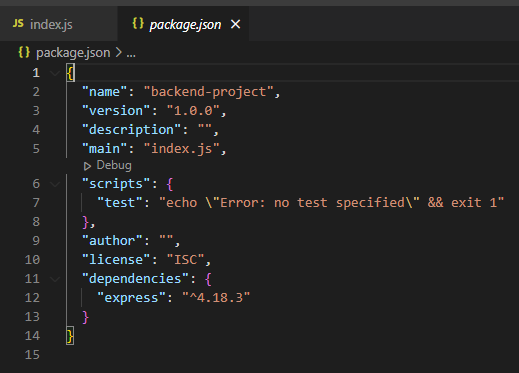
Внутри директории проекта выполните команду npm init, которая создаст файл package.json и попросит вас ввести основную информацию о вашем проекте, такую как название, версия, описание и т.д. Можно просто нажимать Enter, чтобы принять значения по умолчанию.

# 2 Установка необходимых инструментов

Далее нам необходимо установить пакет Express, с помощью которого легко и быстро можно поставить сервер. Делается это с помощью команды **npm install express** (вместо express может быть другой пакет, который нам необходим)



Если необходимо проверить, установился ли пакет, то можно зайти в файл package.json и посмотреть в поле dependencies



По аналогии установите пакеты **body-parser, express-session, cors** и **pg**npm i body-parser

npm i mysql2

npm i express-session

npm i cors

npm i express-mysql-session

# 3 Создание сервера

Следующий шаг – пишем такую конструкцию

const express = require('express');

const bodyParser = require('body-parser');

const session = require('express-session');

const MySQLStore = require('express-mysql-session')(session);

const pool = require('./db');

const cors = require('cors');

const app = express();

const PORT = 8000;

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(cors({

  origin: 'http://localhost:8080',

  methods: ['GET', 'POST', 'OPTIONS', 'PUT', 'PATCH', 'DELETE'],

  allowedHeaders: ['Content-Type', 'Authorization', 'username', 'password'],

  credentials: true

}));

const sessionStore = new MySQLStore({

    host: 'localhost',

    port: 3306,

    user: 'root',

    password: 'root',

    database: 'v4'

});

app.use(session({

    key: 'session\_cookie\_name',

    secret: 'session\_cookie\_secret',

    store: sessionStore,

    resave: false,

    saveUninitialized: false,

    cookie: {

        maxAge: 1000 \* 60 \* 60 \* 24 // 1 день

    }

}));

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`);

});

# 4 Создание файла с конфигурацией БД

Создадим модуль для настройки подключения к базе данных db.js (создайте необходимый файл).

// db.js

const mysql = require('mysql2/promise');

const pool = mysql.createPool({

    host: 'localhost',

    user: 'root',

    password: 'root',  // Замените на ваш пароль

    database: 'v4',

    waitForConnections: true,

    connectionLimit: 10,

    queueLimit: 0

});

module.exports = pool;

Необходимо заменить значение полей **user, database** и **password**.

# 5 Создание роутов (эндпоинтов)

Далее, нам необходимо написать наши ендпоинты, по которым клиент будет обращаться к нашему серверу.

**Обязательно! –** когда пишем коднеобходимо сохранять изменения, делается это комбинацией ctrl+s и перезапуском сервера.

Можно сразу создать необходимые заготовки к будущим роутам:

// Регистрация пользователя

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

// Авторизация пользователя

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

// Получение заявок пользователя

app.get('/bookings', async (req, res) => {};

// Создание новой заявки

app.post('/bookings', async (req, res) => {};

// Маршрут для входа администратора

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

// Ендпоинт для получения списка всех заявок

app.get('/admin/bookings', async (req, res) => {};

// ендпоинт для получения статусов заявления

app.get('/statuses', async (req, res) => {}

// запрос на изменение статуса заявки

app.put('/admin/bookings/:id', async (req, res) => {};

Реализуем каждый роут:

**Регистрация**

// Регистрация пользователя

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

    const { full\_name, phone, login, password } = req.body;

    console.log(req.body);

    try {

      const [result] = await pool.query(

        'INSERT INTO user (full\_name, phone, login, password, id\_role) VALUES (?, ?, ?, ?, ?)',

        [full\_name, phone, login, password, 1]

      );

      res.status(201).json({ id: result.insertId });

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

**Авторизация**

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

    const { login, password } = req.body;

    console.log(req.body);

    try {

      const [users] = await pool.query('SELECT \* FROM user WHERE login = ?', [login]);

      const user = users[0];

      if (user && password == user.password) {

        req.session.userId = user.id;

        res.json({ message: 'Login successful' });

      } else {

        res.status(401).json({ error: 'Invalid credentials' });

      }

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

**Получение заявок пользователя**

// Получение заявок пользователя

app.get('/bookings', async (req, res) => {

    if (!req.session.userId) {

      return res.status(401).json({ error: 'Unauthorized' });

    }

    try {

      const [requests] = await pool.query(`

      SELECT r.id, u.full\_name AS user\_name, m.name AS master\_name, s.name AS status\_name, r.booking\_datetime

      FROM request r

      INNER JOIN user u ON r.id\_user = u.id

      INNER JOIN master m ON r.id\_master = m.id

      INNER JOIN status s ON r.id\_status = s.id

      WHERE r.id\_user = ?

      `, [req.session.userId]);

      res.json(requests);

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

**Создание новой заявки**

// Создание новой заявки

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

    if (!req.session.userId) {

      return res.status(401).json({ error: 'Unauthorized' });

    }

    const { id\_master, booking\_datetime } = req.body;

    try {

      const [result] = await pool.query(

        'INSERT INTO request (id\_user, id\_master, id\_status, booking\_datetime) VALUES (?, ?, ?, ?)',

        [req.session.userId, id\_master, 1, booking\_datetime]

      );

      res.status(201).json({ id: result.insertId });

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

**Вспомогательный запрос для определения мастеров**

// Ендпоинт для получения списка мастеров

app.get('/masters', async (req, res) => {

  try {

    const [masters] = await pool.query('SELECT \* FROM master');

    res.json(masters);

  } catch (error) {

    res.status(500).json({ error: 'Ошибка сервера при получении списка мастеров' });

  }

});

**Авторизация администратора**

// Маршрут для входа администратора

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

    const { username, password } = req.body;

    if (username === 'admin' && password === 'password') {

      req.session.userId = 'admin';

      res.status(200).json({ message: 'Успешная аутентификация' });

    } else {

      res.status(403).json({ error: 'Неверный логин или пароль' });

    }

});

**Получение списка всех заявок (только для администратора)**

// Ендпоинт для получения списка всех заявок

app.get('/admin/bookings', async (req, res) => {

    if (req.session.userId !== 'admin') {

      return res.status(401).json({ error: 'Unauthorized' });

    }

    try {

      const query = `

        SELECT request.id, user.full\_name AS user\_full\_name, user.phone AS user\_phone,

               master.name AS master\_name, request.booking\_datetime, status.name AS status\_name

        FROM request

        INNER JOIN user ON request.id\_user = user.id

        INNER JOIN master ON request.id\_master = master.id

        INNER JOIN status ON request.id\_status = status.id

      `;

      const [requests] = await pool.query(query);

      res.json(requests);

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

**Вспомогательный запрос для получение списка статусов заявления**

// ендпоинт для получения статусов заявления

app.get('/statuses', async (req, res) => {

    try {

        const [statuses] = await pool.query('SELECT \* FROM status');

        res.json(statuses);

    } catch (error) {

        res.status(500).json({ error: 'Ошибка сервера при получении списка статусов' });

    }

});

**Запроса на изменение статуса заявки**

// запрос на изменение статуса заявки

app.put('/admin/bookings/:id', async (req, res) => {

    if (req.session.userId !== 'admin') {

      return res.status(401).json({ error: 'Unauthorized' });

    }

    const { id } = req.params;

    const { status\_id } = req.body;

    try {

      const [result] = await pool.query('UPDATE request SET id\_status = ? WHERE id = ?', [status\_id, id]);

      res.json({ message: 'Status updated' });

    } catch (err) {

      res.status(400).json({ error: err.message });

    }

});

После реализации роутов, необходимо запустить веб-сервер с помощью команды **node index.js**

Теперь ваш сервер будет запущен и будет отвечать на запросы на <http://localhost:8000/>