**[Создание REST API](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html" \l "%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-rest-api)**

[**Что такое REST API?**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A7%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-rest-api)

**REST (Representational State Transfer)** — архитектурный стиль для создания распределенных приложений, который основан на использовании стандартных HTTP-методов для взаимодействия с ресурсами, представленных в виде данных. RESTful API предоставляет механизм обмена данными между клиентом и сервером.

[**Основные принципы REST:**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B-rest)

* **Безсессионность (Stateless):** каждый запрос клиента к серверу должен содержать всю информацию, необходимую для его обработки (не сохраняется состояние между запросами).
* **Использование стандартных HTTP-методов:** для операций с ресурсами используются стандартные HTTP методы:
  + **GET** — для получения данных.
  + **POST** — для создания новых данных.
  + **PUT** — для обновления существующих данных.
  + **DELETE** — для удаления данных.
* **Единообразие интерфейса:** API должно иметь понятный и стандартизированный интерфейс, например, использовать правильные HTTP-коды для различных ситуаций (например, 200 — успех, 400 — ошибка запроса, 404 — не найдено).

Ознакомиться с полным списком актуальных HTTP кодов и их значения можно [по ссылке](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

[**Пример запроса и ответа:**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0-%D0%B8-%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0)

В большинстве случаев Rest API в качестве ответа предоставляют информацию в формате JSON.

**Запрос:**

GET /api/items HTTP/1.1

Host: example.com

**Ответ:**

[

{

"id": 1,

"name": "Item 1",

"description": "Description of Item 1"

},

{

"id": 2,

"name": "Item 2",

"description": "Description of Item 2"

}

]

[**Настройка сервера и подготовка к установке**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B0-%D0%B8-%D0%BF%D0%BE%D0%B4%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BA-%D1%83%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B5)

[**Установка WSL и настройка окружения**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-wsl-%D0%B8-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)

Для использования NGINX на Windows лучше всего установить WSL, который позволяет работать с Linux-утилитами прямо на Windows.

**Установка WSL:** Откройте PowerShell и выполните команду:

wsl --install

Перезагрузите систему и установите Ubuntu через Microsoft Store.

**Установка NGINX на WSL:**

sudo apt update

sudo apt install nginx

sudo service nginx start

[**Установка PostgreSQL**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-postgresql)

В терминале Ubuntu установите PostgreSQL:

sudo apt update

sudo apt install postgresql postgresql-contrib

sudo service postgresql start

**Создайте базу данных и пользователя:**

1. Войдите в PostgreSQL:
2. sudo -u postgres psql
3. Выполните команду для создания базы данных и пользователя:
4. CREATE DATABASE mydb;
5. CREATE USER myuser WITH PASSWORD 'mypassword';
6. GRANT ALL PRIVILEGES ON DATABASE mydb TO myuser;

**Создайте таблицу для хранения данных:**

CREATE TABLE items (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

description TEXT

);

[**Установка Node.js и npm**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-nodejs-%D0%B8-npm)

**Node.js** — это среда выполнения JavaScript вне браузера, которая позволяет создавать серверные приложения. **npm** (Node Package Manager) — это менеджер пакетов для Node.js, который используется для установки различных библиотек и инструментов. Чтобы установить Node.js и npm, выполните следующие шаги:

1. Откройте терминал Ubuntu в WSL и обновите список пакетов:
2. sudo apt update
3. Установите Node.js и npm:
4. sudo apt install nodejs npm
5. Проверьте, что установка прошла успешно:
6. node -v
7. npm -v

Эти команды должны вывести установленные версии Node.js и npm.

[**Создание REST API с использованием Node.js и Express**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-rest-api-%D1%81-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC-nodejs-%D0%B8-express)

[**Инициализация проекта**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%98%D0%BD%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0)

Создайте новый проект:

mkdir {название\_проекта}

cd {название\_проекта}

npm init -y

Установите зависимости:

npm install express pg

[**Структура проекта**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0)

Создайте структуру проекта с использованием принципов MVC (Model-View-Controller):

{название\_проекта}/

├── controllers/

│ └── itemsController.js

├── models/

│ └── itemModel.js

├── routes/

│ └── itemsRoutes.js

├── server.js

└── config/

└── db.js

[**Конфигурация подключения к базе данных**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D0%BE%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BA-%D0%B1%D0%B0%D0%B7%D0%B5-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

Создайте файл config/db.js:

const { Client } = require('pg');

const client = new Client({

user: 'myuser',

host: 'localhost',

database: 'mydb',

password: 'mypassword',

port: 5432,

});

client.connect();

module.exports = client;

[**Создание модели**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8)

В файле models/itemModel.js:

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

const getAllItems = async () => {

const result = await db.query('SELECT \* FROM items');

return result.rows;

};

const addItem = async (name, description) => {

const result = await db.query(

'INSERT INTO items (name, description) VALUES ($1, $2) RETURNING \*',

[name, description]

);

return result.rows[0];

};

module.exports = { getAllItems, addItem };

[**Создание контроллеров**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80%D0%BE%D0%B2)

В файле controllers/itemsController.js:

const itemModel = require('../models/itemModel');

const getItems = async (req, res) => {

try {

const items = await itemModel.getAllItems();

res.status(200).json(items);

} catch (error) {

res.status(500).json({ message: 'Server Error' });

}

};

const createItem = async (req, res) => {

const { name, description } = req.body;

try {

const newItem = await itemModel.addItem(name, description);

res.status(201).json(newItem);

} catch (error) {

res.status(500).json({ message: 'Server Error' });

}

};

module.exports = { getItems, createItem };

[**Создание маршрутов**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%BE%D0%B2)

В файле routes/itemsRoutes.js:

const express = require('express');

const router = express.Router();

const itemsController = require('../controllers/itemsController');

router.get('/items', itemsController.getItems);

router.post('/items', itemsController.createItem);

module.exports = router;

[**Основной сервер**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D0%BE%D0%B9-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80)

В файле server.js:

const express = require('express');

const app = express();

const itemsRoutes = require('./routes/itemsRoutes');

app.use(express.json()); // для обработки JSON-тел запросов

app.use('/api', itemsRoutes);

const port = 3000;

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}`);

});

[**Расширение API**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D0%B5-api)

Добавьте новые маршруты для обновления и удаления данных. Ориентируетесь на пример с добавлением новых записей, который был представлен раньше.

[**Настройка NGINX**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-nginx)

**NGINX** — это высокопроизводительный веб-сервер, который также может работать в качестве прокси-сервера. Прокси-сервер — это посредник между клиентом и сервером. Обратный прокси (reverse proxy) принимает запросы от клиентов и пересылает их на сервер приложений. Он может распределять нагрузку, управлять кэшированием, а также обеспечивать безопасность и масштабируемость приложения.

В нашем случае NGINX будет выступать в качестве прокси-сервера между клиентом и сервером Node.js, принимая запросы на порту 80 и перенаправляя их на наш API, запущенный на порту 3000.

**CORS (Cross-Origin Resource Sharing)** — это механизм, который позволяет управлять доступом к ресурсам API с других доменов. В конфигурации NGINX мы разрешим CORS, чтобы веб-клиенты могли взаимодействовать с нашим API.

[**Создание конфигурационного файла проекта**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE-%D1%84%D0%B0%D0%B9%D0%BB%D0%B0-%D0%BF%D1%80%D0%B5%D0%BE%D0%BA%D1%82%D0%B0)

Для нашего API создадим новый конфигурационный файл в /etc/nginx/sites-available/ и настроим прокси-сервер с поддержкой CORS.

1. Перейдите в директорию конфигурации NGINX и создайте файл {название\_проекта}.conf:
2. sudo nano /etc/nginx/sites-available/{название\_проекта}.conf
3. Вставьте следующую конфигурацию в файл и сохраните его.

# Блок server, обрабатывающий запросы, поступающие на сервер

server {

# Порт для прослушивания. Поскольку NGINX работает на HTTP-порту по умолчанию, указываем порт 80.

listen 80;

# Имя сервера, которое будет использоваться для обработки запросов. Здесь можно указать IP или домен.

server\_name localhost;

# Настройка расположения для обработки запросов к API

location / {

# Прокси-передача запросов на наш API, работающий на Node.js

proxy\_pass http://localhost:3000;

# Установка версии HTTP-протокола

proxy\_http\_version 1.1;

# Настройка заголовков, необходимых для корректной работы прокси-сервера

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

# Разрешение CORS, чтобы API был доступен для кросс-доменных запросов

add\_header 'Access-Control-Allow-Origin' '\*';

add\_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS';

add\_header 'Access-Control-Allow-Headers' 'Content-Type, Authorization';

}

}

1. Сохраните файл и закройте редактор.

После создания конфигурации, необходимо активировать ее, создав символическую ссылку в папке sites-enabled и удалив стандартный файл default.

# Создайте символическую ссылку на новый файл

sudo ln -s /etc/nginx/sites-available/{название\_проекта}.conf /etc/nginx/sites-enabled/

# Удалите ссылку на стандартную конфигурацию

sudo rm /etc/nginx/sites-enabled/default

# Перезапустите NGINX для применения изменений

sudo systemctl restart nginx

[**Создание простого веб-клиента с поддержкой CRUD операций**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B3%D0%BE-%D0%B2%D0%B5%D0%B1-%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0-%D1%81-%D0%BF%D0%BE%D0%B4%D0%B4%D0%B5%D1%80%D0%B6%D0%BA%D0%BE%D0%B9-crud-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B9)

Теперь создадим простой веб-клиент с использованием HTML и JavaScript для взаимодействия с нашим API. Все элементы загружаются из API и отображаются в виде списка. Пользователи могут добавлять новые элементы, обновлять существующие, вводя новую информацию через диалоговые окна, и удалять элементы с подтверждением.

[**Пример кода веб-клиента**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80-%D0%BA%D0%BE%D0%B4%D0%B0-%D0%B2%D0%B5%D0%B1-%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0)

Создайте файл index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Simple Web Client</title>

<style>

body { font-family: Arial, sans-serif; }

.item { border: 1px solid #ddd; padding: 10px; margin-bottom: 10px; }

.actions { margin-top: 5px; }

</style>

</head>

<body>

<h1>Items</h1>

<div id="items"></div>

<h2>Add New Item</h2>

<form id="itemForm">

<input type="text" id="name" placeholder="Name" required>

<input type="text" id="description" placeholder="Description" required>

<button type="submit">Add Item</button>

</form>

<script>

// Fetch all items from the API

async function fetchItems() {

const response = await fetch('/api/items');

const items = await response.json();

const itemsContainer = document.getElementById('items');

itemsContainer.innerHTML = '';

items.forEach(item => {

const itemElement = document.createElement('div');

itemElement.className = 'item';

itemElement.innerHTML = `

<strong>Name:</strong> \${item.name}<br>

<strong>Description:</strong> \${item.description}

<div class="actions">

<button onclick="updateItem(\${item.id})">Update</button>

<button onclick="deleteItem(\${item.id})">Delete</button>

</div>

`;

itemsContainer.appendChild(itemElement);

});

}

// Add a new item to the API

document.getElementById('itemForm').addEventListener('submit', async (e) => {

e.preventDefault();

const name = document.getElementById('name').value;

const description = document.getElementById('description').value;

await fetch('/api/items', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ name, description })

});

document.getElementById('itemForm').reset();

fetchItems();

});

// Update an item

async function updateItem(id) {

const newName = prompt('Enter new name:');

const newDescription = prompt('Enter new description:');

if (newName && newDescription) {

await fetch(`/api/items/\${id}`, {

method: 'PUT',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ name: newName, description: newDescription })

});

fetchItems();

}

}

// Delete an item

async function deleteItem(id) {

if (confirm('Are you sure you want to delete this item?')) {

await fetch(`/api/items/\${id}`, { method: 'DELETE' });

fetchItems();

}

}

// Load items on page load

fetchItems();

</script>

</body>

</html>

[**Задание на разработку API**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%97%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BD%D0%B0-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D1%83-api)

Ваша задача — создать REST API с базой данных и веб- или десктоп-клиентом на основе одного из предложенных вариантов.

**Требования к отчету:**

1. ***Номер варианта***.
2. ***Описание структуры разработанной БД***.

* Краткое описание таблиц базы данных и связей.
* Включите ER-диаграмму и SQL-дамп для развертывания базы данных.

1. ***Документирование API***. Список всех маршрутов API с методами, описанием, параметрами и кодами ответа. Пример:

| **Маршрут** | **Метод** | **Описание** | **Параметры в body** | **Заголовки** | **Коды ответа** |
| --- | --- | --- | --- | --- | --- |
| /api/items | GET | Получение всех элементов | - | Content-Type: application/json | 200: Успех, 500: Ошибка |

1. ***Полный исходный код сервера***. Весь исходный код проекта, включая конфигурации, модели, контроллеры и маршруты.
2. ***Полный исходный код клиента***. Исходный код веб/десктоп-клиента с реализацией CRUD операций.

Код должен находиться в нормальном состоянии и быть структурированным, а не разбросанным как после пьянки.

1. ***Инструкция по развертыванию системы***.

С небольшим примером того как это все в теории может выглядеть, можете ознакомиться [тут](https://github.com/EgorTuryshev/Aermolenko)

[**Варианты заданий:**](https://student-labs.kernel-dev.ru/InformationSystems/lab4.html#%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D1%8B-%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B9)

К обязательным требованиям относится только создание CRUD. За дополнительное расширение функциональности, например генерация отчетов в 1 варианте, можно получить дополнительные баллы. Для создания интерфейса не обязательно использовать голую связку html + css + js, можно использовать фреймворки, например сочетание React (фреймворк) + Vite (сборщик) + TS (ЯП, по сути JS с типами данных).

1. **Вариант 1**: Платформа для обмена растениями
   * **База данных:** Таблицы для пользователей, растений, предложений обмена и истории обменов.
   * **Примерный список маршрутов API:**
     + CRUD операции для управления растениями, создания предложений обмена и отслеживания истории.
     + Поиск предложений обмена, совместимых с растениями пользователя.
     + Создание запросов на обмен и подтверждение/отклонение обменов.
     + Генерация отчетов о наиболее активных пользователях и самых популярных растениях для обмена.
   * **Клиент:** Веб-приложение с фильтрацией растений по типу и региону, поддержкой обменов и ведением истории.
2. **Вариант 2**: Система учета волонтерской активности
   * **База данных:** Таблицы для волонтеров, мероприятий, организаторов и отчетов об активности.
   * **Примерный список маршрутов API:**
     + CRUD операции для управления мероприятиями и участниками.
     + Регистрация и отмена участия в мероприятиях с учетом максимального количества участников.
     + Автоматическое уведомление волонтеров за день до мероприятия письмом на почту.
     + Генерация статистики об активности волонтеров и рейтинги за выполненные задачи.
   * **Клиент:** Веб-приложение с календарем, уведомлениями и разделом для просмотра личной статистики волонтера.
3. **Вариант 3**: Платформа для рекомендаций маршрутов путешествий
   * **База данных:** Таблицы для пользователей, маршрутов, точек интереса и отзывов.
   * **Примерный список маршрутов API:**
     + CRUD операции для добавления маршрутов и отзывов.
     + Генерация персонализированных маршрутов на основе предпочтений пользователя и рейтингов.
     + Создание коллекций маршрутов и возможность делиться ими с друзьями.
   * **Клиент:** Веб-приложение с картами, сохранением маршрутов в избранное и функцией совместного использования.
4. **Вариант 4**: Экологическая платформа для учета отходов
   * **База данных:** Таблицы для типов отходов, пунктов приема, пользователей и отчетов об утилизации.
   * **Примерный список маршрутов API:**
     + CRUD операции для управления пунктами приема и типами отходов.
     + Расчет экологического вклада пользователя на основе сданных отходов.
     + Система наград и достижений для активных пользователей.
   * **Клиент:** Веб-приложение с поиском пунктов приема, статистикой и системой достижений.
5. **Вариант 5**: Платформа для кулинарных мастер-классов
   * **База данных:** Таблицы для рецептов, кулинарных мастеров, пользователей и расписания мастер-классов.
   * **Примерный список маршрутов API:**
     + CRUD операции для управления рецептами и расписанием.
     + Генерация списка покупок на основе выбранного рецепта.
     + Рекомендации мастер-классов на основе предпочтений пользователя и истории посещений.
   * **Клиент:** Веб-приложение с видеоуроками, подписками на мастеров и разделом рекомендаций.