Министерство цифрового развития, связи и массовых коммуникаций РФ

Уральский технический институт связи и информатики (филиал) ФГБОУ ВО "Сибирский государственный университет телекоммуникаций и информатики" в г. Екатеринбурге (УрТИСИСибГУТИ)

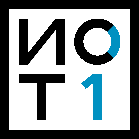
Изображение выглядит как текст

Автоматически созданное описание

Кафедра информационных

систем и

технологий



КАФЕДРА

Информационных систем и технологий

**ОТЧЕТ**

По дисциплине «Сетевое программирование»

Практическая работа № 2

Разработка API на node.js

|  |  |
| --- | --- |
| Выполнил: | студент гр. П11БС  Ахметханов Р.Р. |
| Проверил: | преподаватель  Бурумбаев Д.И. |

Екатеринбург, 2024

1. Цель работы:
   1. Научиться работать с API;
   2. Закрепить знания по теме «Знакомство с API».
2. Задание:

Разработать REST API и протестировать его при помощи программного обеспечения Postman.

1. Выполнение работы:

В начале была создана база данных в СУБД MySQL с наименованием «p-11bs\_akhmetkhanov», содержащая таблицу «users» для хранения пользователей (Рисунок 1).

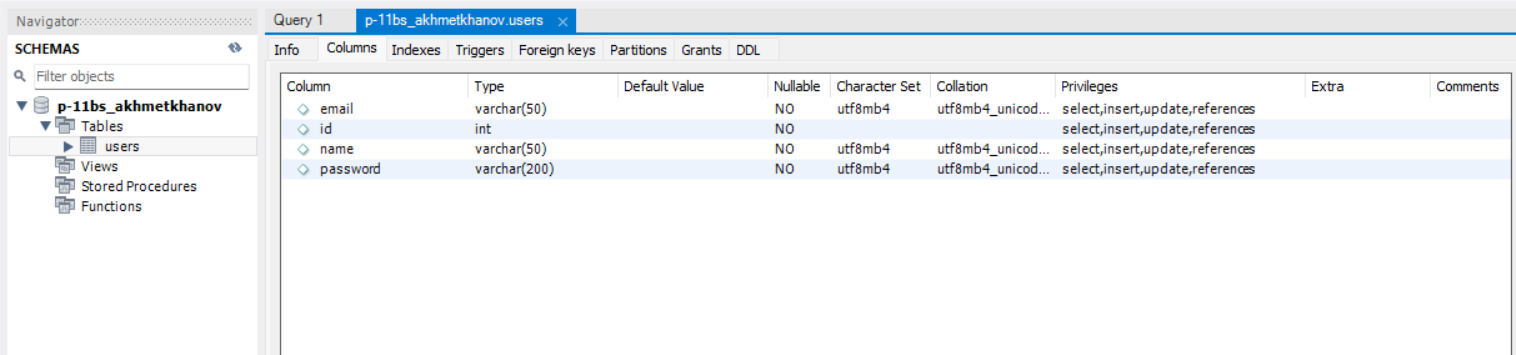


Рисунок 1 – База данных

Далее была создана структура разрабатываемого API под названием «akhmetkhanov-rest-api», хранящая скрипты (Рисунок 2).

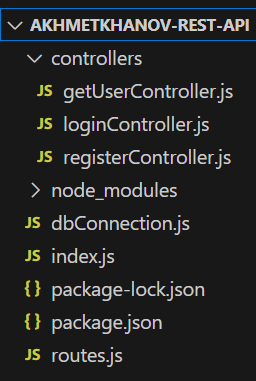


Рисунок 2 – Структура разрабатываемого API

После инициализации в среде разработки было необходимо установить пакеты командой «npm i express express-validator mysql2 jsonwebtoken bcryptjs». После установки вышеуказанных пакетов node файл package.json выглядит так, как показано в листинге 1.

Листинг 1 – Файл package.json

{

"dependencies": {

"bcryptjs": "^2.4.3",

"express": "^4.21.1",

"express-validator": "^7.2.0",

"jsonwebtoken": "^9.0.2",

"mysql2": "^3.11.5"

}

}

В файле «dbConnection.js» был написан код для связи с базой данных, представленный в листинге 2.

Листинг 2 – Файл dbConnectrion.js

const mysql = require("mysql2");

const db\_connection = mysql

.createConnection({

host: "127.0.0.1", // имя хоста

user: "root", // имя пользователя

database: "p-11bs\_akhmetkhanov", // имя базы данных

password: "1234", // пароль от базы данных

})

.on("error", (err) => {

console.log("Failed to connect to Database - ", err);

});

module.exports = db\_connection;

После установления соединения с БД необходимо было настроить маршруты, и для этого в файле «routes.js» был написан код, представленный в листинге 3

Листинг 3 – Файл rotes.js

const router = require('express').Router();

const {body} = require('express-validator');

const {register} = require('./controllers/registerController');

const {login} = require('./controllers/loginController');

const {getUser} = require('./controllers/getUserController');

router.post('/register', [

body('name',"The name must be of minimum 3 characters length")

.notEmpty()

.escape()

.trim()

.isLength({ min: 3 }),

body('email',"Invalid email address")

.notEmpty()

.escape()

.trim().isEmail(),

body('password',"The Password must be of minimum 4 characters length").notEmpty().trim().isLength({ min: 4 }),

], register);

router.post('/login',[

body('email',"Invalid email address")

.notEmpty()

.escape()

.trim().isEmail(),

body('password',"The Password must be of minimum 4 characters length").notEmpty().trim().isLength({ min: 4 }),

],login);

router.get('/getuser',getUser);

module.exports = router;

Для регистрации нового пользователя в файле «registerController.js» был написан код, представленный в листинге 4.

Листинг 4 – Файл registerController.js

const {validationResult} = require('express-validator');

const bcrypt = require('bcryptjs');

const conn = require('../dbConnection').promise();

exports.register = async(req,res,next) => {

const errors = validationResult(req);

if(!errors.isEmpty()){

return res.status(422).json({ errors: errors.array() });

}

try{

const [row] = await conn.execute(

"SELECT `email` FROM `users` WHERE `email`=?",

[req.body.email]

);

if (row.length > 0) {

return res.status(201).json({

message: "The E-mail already in use",

});

}

const hashPass = await bcrypt.hash(req.body.password, 12);

const [rows] = await conn.execute('INSERT INTO `users`(`id`,`name`,`email`,`password`) VALUES(?,?,?,?)',[

req.body.id,

req.body.name,

req.body.email,

hashPass

]);

if (rows.affectedRows === 1) {

return res.status(201).json({

message: "The user has been successfully inserted.",

});

}

}catch(err){

next(err);

}

}

Для входа пользователя в систему в файле «loginController.js» был написан код, представленный в листинге 5.

Листинг 5 – Файл loginController.js

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const {validationResult} = require('express-validator');

const conn = require('../dbConnection').promise();

exports.login = async (req,res,next) =>{

const errors = validationResult(req);

if(!errors.isEmpty()){

return res.status(422).json({ errors: errors.array() });

}

try{

const [row] = await conn.execute(

"SELECT \* FROM `users` WHERE `email`=?",

[req.body.email]

);

if (row.length === 0) {

return res.status(422).json({

message: "Invalid email address",

});

}

const passMatch = await bcrypt.compare(req.body.password, row[0].password);

if(!passMatch){

return res.status(422).json({

message: "Incorrect password",

});

}

const theToken = jwt.sign({id:row[0].id},'the-superstrong-secrect',{ expiresIn: '1h' });

return res.json({

token:theToken

});

} catch(err){

next(err);

}

}

Для получения сведений о пользователе с использованием токена JWT в файле «getUserController.js» был написан код, представленный в листинге 6.

Листинг 6 – Файл getUserController.js

const jwt = require('jsonwebtoken');

const conn = require('../dbConnection').promise();

exports.getUser = async (req,res,next) => {

try{

if(

!req.headers.authorization ||

!req.headers.authorization.startsWith('Bearer') ||

!req.headers.authorization.split(' ')[1]

){

return res.status(422).json({

message: "Please provide the token",

});

}

const theToken = req.headers.authorization.split('')[1];

const decoded = jwt.verify(theToken, 'the-super-strongsecrect');

const [row] = await conn.execute(

"SELECT `id`,`name`,`email` FROM `users` WHERE `id`=?",

[decoded.id]

);

if(row.length > 0){

return res.json({

user:row[0]

});

}

res.json({

message:"No user found"

});

} catch(err){

next(err);

}

}

В заключении необходимо было создать главный JS-файл с названием «index.js», код для которого представлен в листинге 7

Листинг 7 – Файл index.js

const express = require('express');

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

const app = express();

app.use(express.json());

app.use(routes);

// Handling Errors

app.use((err, req, res, next) => {

// console.log(err);

err.statusCode = err.statusCode || 500;

err.message = err.message || "Internal Server Error";

res.status(err.statusCode).json({

message: err.message,

});

});

app.listen(3000,() => console.log('Server is running on port 3000'));

После написания всех скриптов необходимо запустить разработанный API командой «node index.js». В терминале отобразится надпись, говорящая о том, что сервер запущен (Рисунок 3).

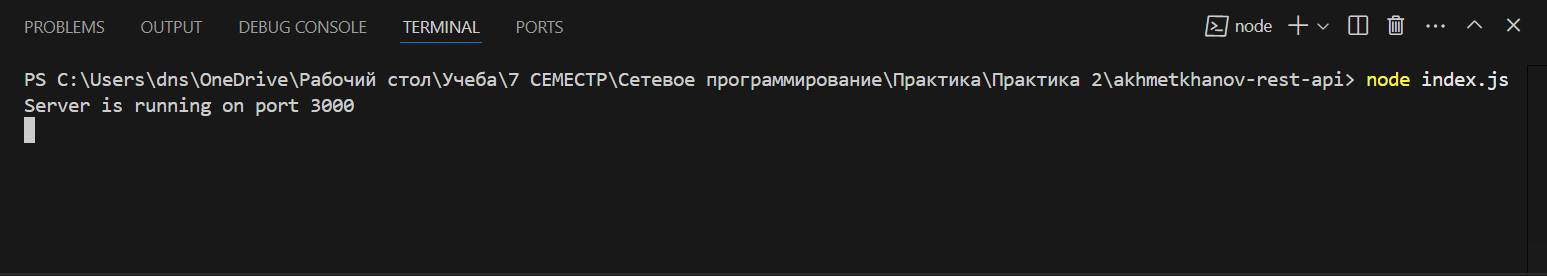


Рисунок 3 – Запуск разработанного API

Если в браузере прописать адрес «http://localhost:3000», то на экране появится следующее (Рисунок 4):

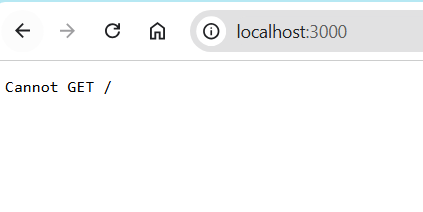


Рисунок 4 – Открытие сервера в браузере

Далее было произведено тестирование разработанного API при помощи запросов.

Первый запрос POST на регистрацию пользователя. Для этого в программе Postman необходимо создать новый POST запрос и прописать адрес «http://localhost:3000/register». Ниже нужно выбрать «Body» и «raw», тип «JSON». Ниже прописать переменные и значения, которые нужно добавить в базу данных. После этого нажать «Send» и ниже отобразится сообщение об успешном добавлении пользователя в базу данных (Рисунок 5).

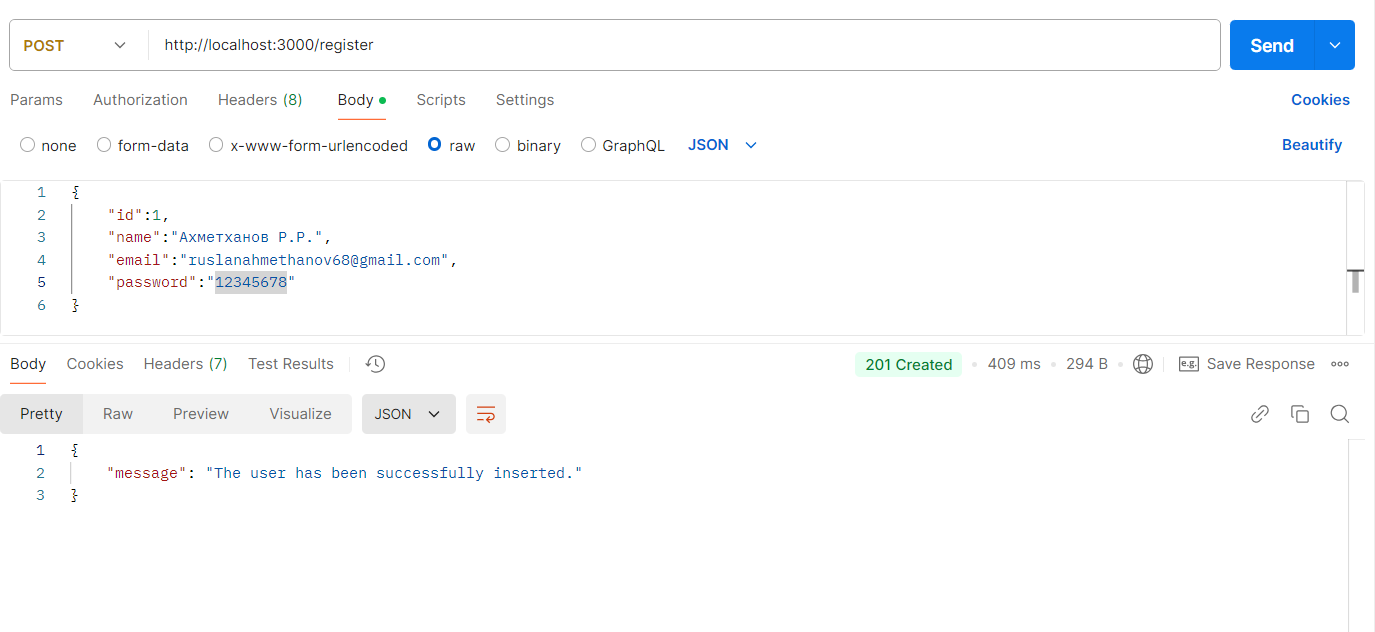


Рисунок 5 – POST запрос на регистрацию пользователя

Было добавлено с помощью данного запроса 3 пользователя (Рисунок 6).

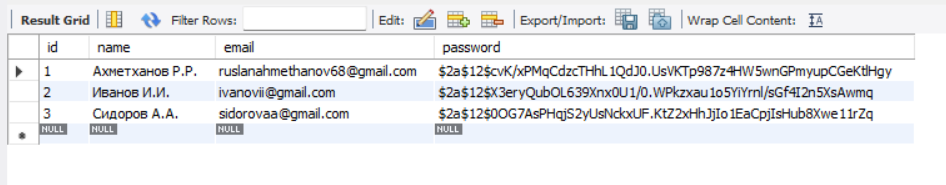


Рисунок 6 – База данных после добавления пользователей

Второй запрос POST на авторизацию пользователя в системе. Для этого в программе Postman необходимо создать новый POST запрос и прописать адрес «http://localhost:3000/login». Ниже нужно выбрать «Body» и «raw», тип «JSON». Ниже прописать данные для входа, которые были введены при регистрации. После этого нажать «Send» и ниже отобразится JWT токен авторизованного пользователя (Рисунок 7).

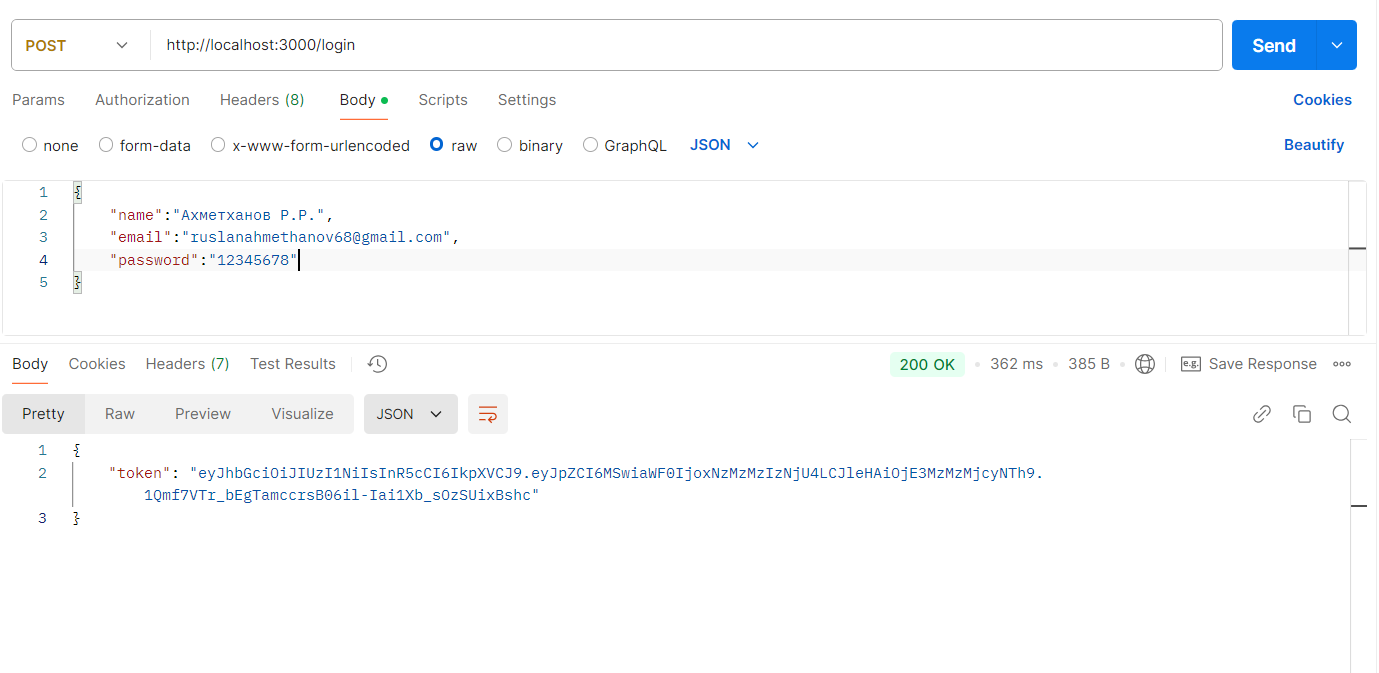


Рисунок 7 – POST запрос на авторизацию пользователя

При вводе неправильного пароля или почты программа выдает соответствующую ошибку (Рисунок 8).

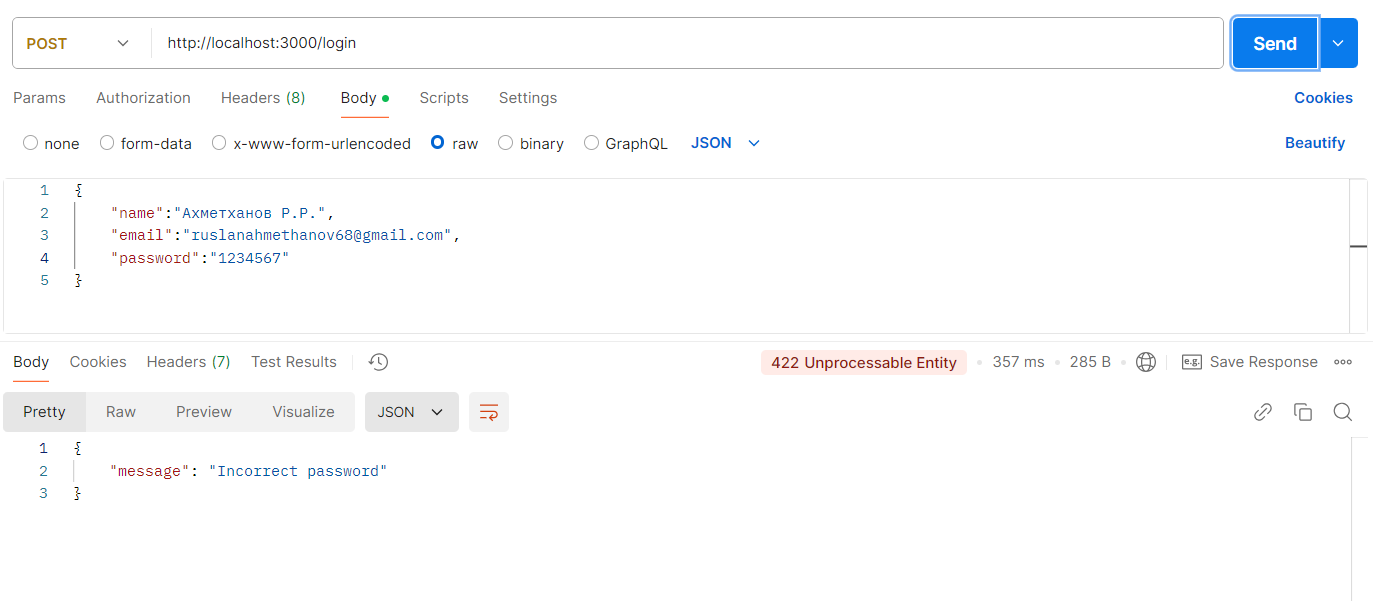


Рисунок 8 – POST запрос на авторизацию пользователя

1. Ответы на контрольные вопросы:

4.1 Что такое API и какую роль оно играет в разработке программного обеспечения?

API – это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов. Например, система ПО метеослужбы содержит ежедневные данные о погоде.

Вот какие возможности даёт API:

* Предоставляет доступ к готовым инструментам. Например, к функциям библиотеки для машинного обучения TensorFlow — они помогают быстро создать нейросеть, не тратя время на разработку инструментов с нуля.
* Повышает безопасность. API позволяет вынести в отдельное приложение функциональность, которая должна быть защищена. Так снижается вероятность некорректного использования этих функций другими программами.
* Связывает разные системы. Если вам нужно подключить к сайту платёжную систему или авторизацию через соцсети, без API не обойтись.
* Снижает стоимость разработки. Часто бывает, что дешевле воспользоваться платным API, чем создавать функциональность с нуля.

4.2 Какие принципы проектирования API следует учитывать при его разработке?

Всего в REST есть шесть требований к проектированию API. Пять из них обязательные, одно — опциональное:

* Клиент-серверная модель (client-server model).
* Отсутствие состояния (statelessness).
* Кэширование (cacheability).
* Единообразие интерфейса (uniform interface).
* Многоуровневая система (layered system).
* Код по требованию (code on demand) — необязательно.

4.3 Что такое RESTful API и какие основные принципы он соблюдает?

RESTful API — это интерфейс,используемые двумя компьютерными системами для безопасного обмена информацией через Интернет. Большинство бизнес-приложений должны взаимодействовать с другими внутренними и сторонними приложениями для выполнения различных задач. Например, чтобы генерировать ежемесячные платежные ведомости, ваша внутренняя бухгалтерская система должна обмениваться данными с банковской системой вашего клиента, чтобы автоматизировать выставление счетов и взаимодействовать с внутренним приложением по учету рабочего времени. RESTful API поддерживают такой обмен информацией, поскольку они следуют безопасным, надежным и эффективным стандартам программного взаимодействия.

4.4 Какую роль играет формат данных (например, JSON или XML) при разработке API?

Payload body играет ключевую роль в API-запросах, поскольку именно здесь содержатся данные, которые вы хотите передать. Как форматируется payload важно для обеспечения совместимости между клиентом и сервером. Обычно payload форматируется в JSON или XML, что делает данные легко читаемыми как для человека, так и для машины.

4.5 Какие механизмы аутентификации и авторизации могут использоваться в API?

Два из наиболее распространённых методов аутентификации в современных веб-приложениях и API – это Bearer-токены и API-ключи. Эти методы обеспечивают безопасность и контроль доступа, позволяя серверам удостоверяться в том, что запросы исходят от доверенных клиентов.

4.6 Как можно предоставить документацию для API, чтобы облегчить работу разработчикам?

В отличие от большинства пользовательских справок, ориентированных на конечного пользователя продукта, API-документация обычно создается для внутреннего использования с фокусом на потребностях разработчиков, потому что API чаще всего применяется для взаимодействия между разными компонентами или сервисами внутри одного программного комплекса. В качестве примера можно рассмотреть микросервисную архитектуру, когда в одном приложении задействуется ряд микросервисов – для авторизации пользователей, для доступа к базе данных, для интеграции с другими сервисами и так далее. Взаимодействие происходит через API, каждое из которых может быть разработано отдельной командой. И чтобы разработчики приложения могли использовать эти API корректно, требуется внутренняя API-документация.

Тем не менее, иногда бизнес предоставляет доступ к API внешним разработчикам. Например, CRM-системы с возможностью интеграции в сервисы клиентов, платежные сервисы с интеграцией в сайт покупки товаров, система контроля доступа в здание, интегрированная с системой распознавания лиц на входе. И здесь тоже не обойтись без качественной документации возможностей и особенностей конкретного API.

Один из самых распространенных современных подходов к написанию API-документации – автоматическая генерация документа на основе формального описания в специализированных форматах вроде OpenAPI, Swagger, SOAP.

4.7 Какие инструменты и технологии используются для разработки и тестирования API?

Популярные инструменты для автоматизированного тестирования включают Postman, SoapUI и JMeter. Автоматизированное тестирование помогает сократить время на выполнение тестов и повысить их точность. Оно также позволяет легко повторять тесты и интегрировать их в процессы CI/CD (непрерывной интеграции и доставки).