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

(УрТИСИ СибГУТИ)

КАФЕДРА

ИСТ

**ОТЧЕТ**

По дисциплине «Сетевое программирование»

Практическое занятие № 3

«Работа с авторизацией и аутентификацией»

|  |  |
| --- | --- |
| Выполнил: | студент гр. ПЕ-12б  Камков Д.А. |
| Проверил: | Ст.преп.,   Бурумбаев Д.И. |
| Ассистент: |  |

 Екатеринбург, 2024

1 Цель работы:

1.1 Научиться работать с регистрацией, авторизацией и аутентификацией;

1.2 Закрепить знания по теме «Аутентификация и авторизация пользователей в клиент-серверных приложениях».

2 Перечень оборудования:

22.1 Персональный компьютер;

2.2 Postman;

2.3 node.js;

2.4 Visual Studio Code..

3.Ход работы:

3.1 Для реализации системы регистрации, авторизации и аутентификации необходимо выполнить ряд действий. Первым делом необходимо настроить базу данных. Для примера, можно использовать базу данных MySQL при помощи XAMPP, WAMP или другого ПО. Внутри необходимо создать базу данных familia\_login. Затем внутри этой базы данных нужно создать таблицу с именем users при помощи SQL-запроса, который представлен в листинге 1.

Листинг 1 – SQL-запрос для создания таблицы

|  |
| --- |
| CREATE TABLE users ( id int(10) unsigned NOT NULL AUTO\_INCREMENT, name varchar(50) COLLATE utf8mb4\_unicode\_ci NOT NULL, email varchar(50) COLLATE utf8mb4\_unicode\_ci NOT NULL, password varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL, PRIMARY KEY (id), UNIQUE KEY email (email) ) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci; |

Далее необходимо создать структуру разрабатываемого API, которая представлена на рисунке 4.

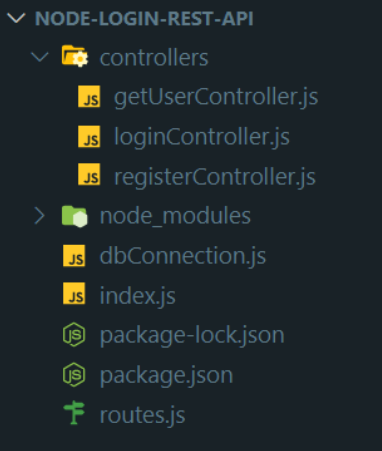


Рисунок 4 – Структура проекта

Файл dbConnection.js отвечает за связь с базой данных. Для того, чтобы произошла связка в нем необходимо прописать следующий код, представленный в листинге 2:

Листинг 2 – Файл dbConnectrion.js

|  |
| --- |
| const mysql = require("mysql2");  const db\_connection = mysql  .createConnection({  host: "localhost", // имя хоста  user: "root", // имя пользователя  database: "pe-12b\_kamkov", // имя базы данных  password: "root", // пароль от базы данных  })  .on("error", (err) => {  console.log("Failed to connect to Database - ", err);  });  module.exports = db\_connection; |

После установления соединения с БД теперь необходимо настроить маршруты, и для этого нужно в файле routes.js в корневом каталоге прописать код, представленный в листинге 3:

Листинг 3 – Код в файле routes.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 characterslength").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 characterslength").notEmpty().trim().isLength({ min: 4 }),  ],login);  router.get('/getuser',getUser);  module.exports = router; |

Внутри папки controllers необходимо создать три контроллера:

1) registerController.js – для вставки нового пользователя.

2) loginController.js – для пользователя, входящего в систему.

3) getUserController.js – для получения сведений о пользователе с использованием токена JWT.

Для первого файла листинг кода будет иметь следующий вид (листинг 4):

Листинг 4 – Код для registerController.js

|  |
| --- |
| const {validationResult} = require('express-validator');  const bcrypt = require('bcryptjs');  const conn = require('../dbConnection').promise();  const newLocal = 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`(`name`,`email`,`password`) VALUES(?,?,?)',[  req.body.name,  req.body.email,  hashPass  ]);  }  catch(err){  next(err);  }  } |

Для файла loginController.js листинг кода будет иметь следующий вид (листинг 5):

Листинг 5 – Код для 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-super-strong-secrect',{ expiresIn: '1h' });  return res.json({  token:theToken  });  }  catch(err) {  next(err);  }  } |

Для getUserController.js код представлен в листинге 6:

Листинг 6 – Код для 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-strong-secrect');  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 создать главный JS-файл с названием index.js, код для которого представлен в листинге 7:

Листинг 7 – Код для 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(4000,() => console.log('Server is running on port 4000')); |

В заключении, индивидуальным заданием каждого студента является тестирование разработанного API при помощи программного обеспечения Postman.

Для этого необходимо при помощи GET и POST запросов произвести тестирование на: регистрацию, авторизацию, получение данных о пользователе и получение данных о токене. Количество созданных пользователей в базе данных должно быть равно 3, первым из которых является студент, выполняющий практическую работу.

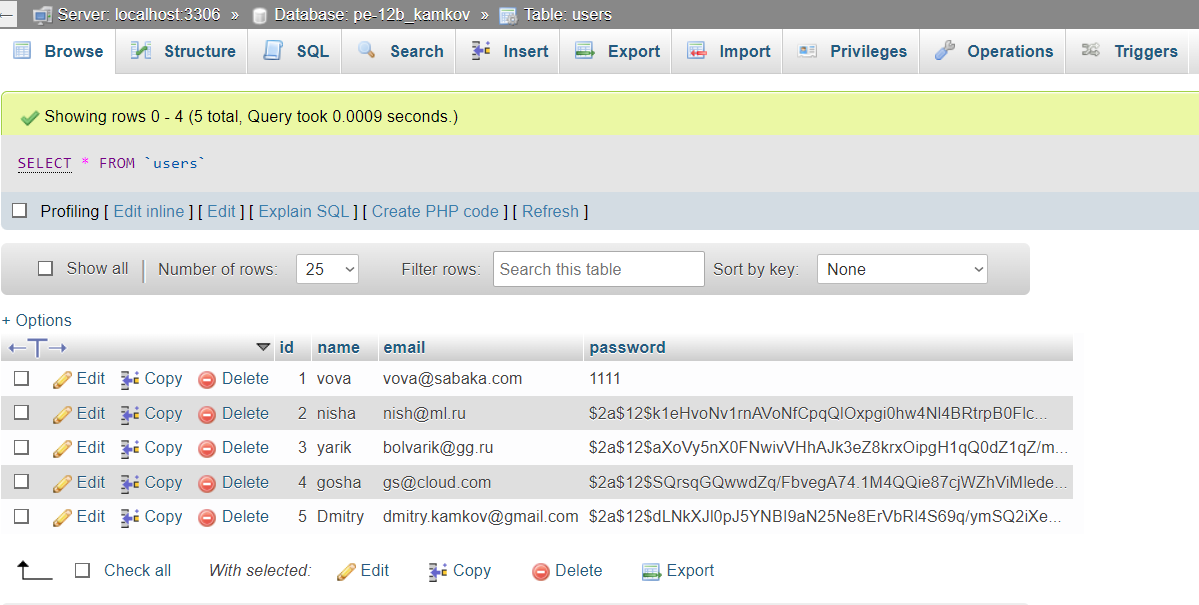


Рисунок 5 – Результат POST-запросов «register»

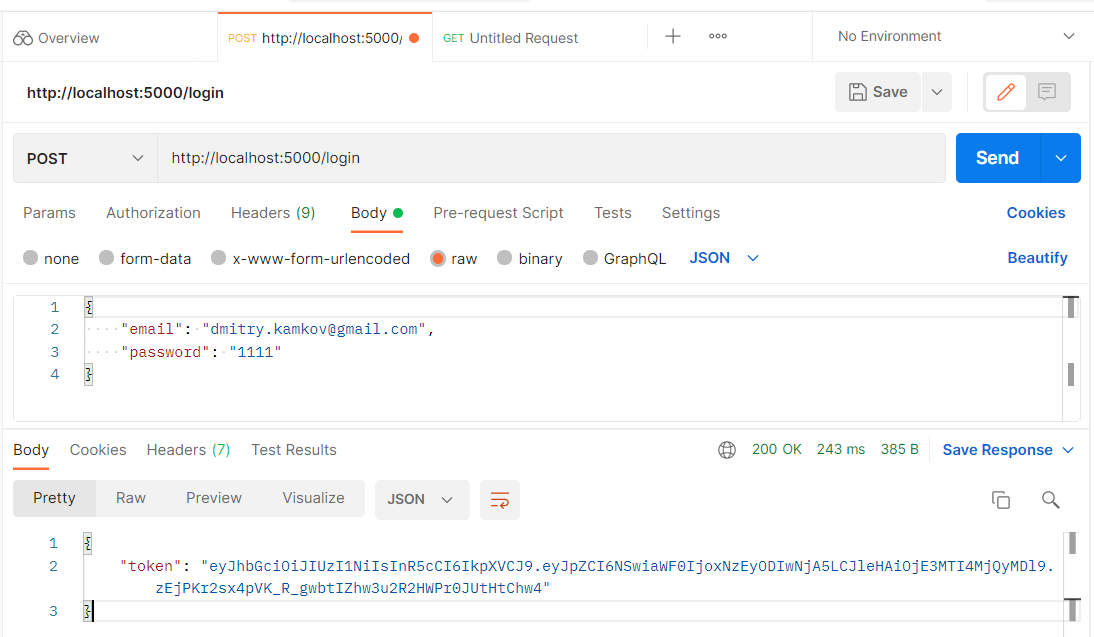


Рисунок 6 – Результат выполнения POST-запроса «login»

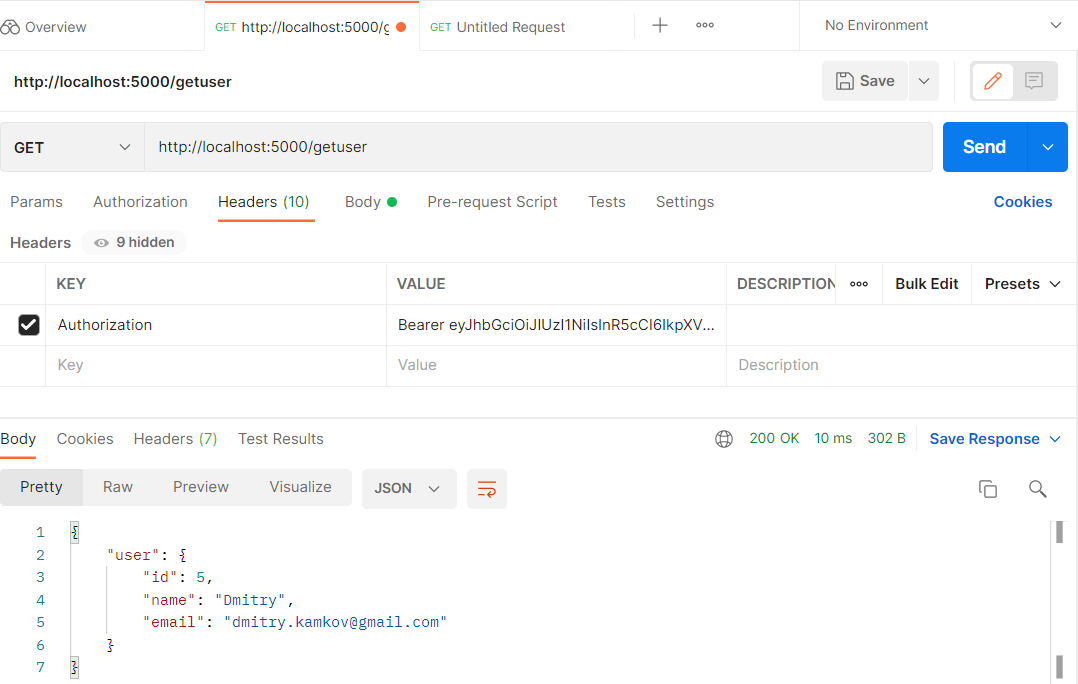
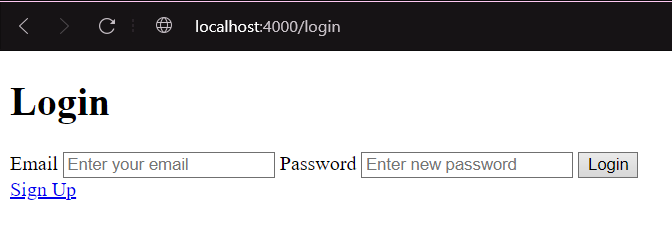
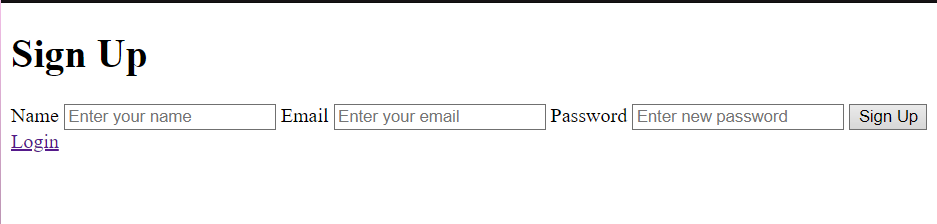
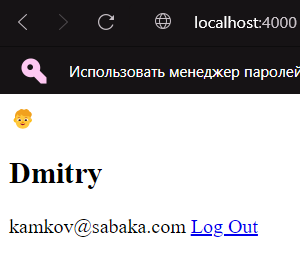


Рисунок 7 – Результат выполнения GET-запроса «getUser»



4. Контрольные вопросы:

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

API (интерфейс прикладного программирования) — это набор правил и протоколов, которые позволяют различным программным приложениям взаимодействовать друг с другом. Он определяет, как должны взаимодействовать компоненты программного обеспечения, позволяя разработчикам получать доступ к определенным функциям или данным из службы или приложения без необходимости понимать его внутреннюю работу. API играют решающую роль в разработке программного обеспечения, упрощая интеграцию, способствуя повторному использованию кода и позволяя создавать более сложные и универсальные приложения.

4.2 Какие принципы проектирования API следует учитывать при его разработке?

При разработке API учитывайте такие принципы, как:

- Последовательность. Сохраняйте единообразие в соглашениях об именах, конечных точках и ответах.

- Простота. Легкость понимания и использования.

- Гибкость: допускайте различные варианты использования без чрезмерной сложности.

- Масштабируемость. Убедитесь, что API способен справляться с растущими нагрузками без серьезной переработки.

- Надежность. Стремитесь к высокой доступности и минимальному времени простоя.

- Безопасность. Внедрите надежные меры аутентификации, авторизации и защиты данных.

- Производительность. Оптимизируйте скорость и эффективность.

- Документация. Предоставляйте разработчикам четкую и полную документацию.

- Управление версиями. Планируйте будущие изменения и обновления с учетом обратной совместимости.

- Обратная связь. Собирайте отзывы пользователей для итеративного улучшения API.

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

RESTful API — это архитектурный стиль разработки сетевых приложений. Он следует следующим основным принципам:

- Клиент-серверная архитектура

- Безгражданство

- Кэшируемость

- Многоуровневая система

- Единый интерфейс

- Код по запросу (необязательно)

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

Формат данных, такой как JSON или XML, определяет, как данные структурируются и обмениваются данными между клиентом и сервером при разработке API, что влияет на простоту анализа данных, их читаемость и совместимость. Это влияет на то, как информация передается, обрабатывается и понимается различными системами.

4.5 Какие механизмы аутентификации и авторизации могут использоваться в API?

Механизмы аутентификации: ключи API, OAuth (OAuth 2.0), JWT (веб-токены JSON), базовая аутентификация, аутентификация на основе токенов.

Механизмы авторизации: контроль доступа на основе ролей (RBAC), контроль доступа на основе атрибутов (ABAC), области OAuth, веб-токены JSON с утверждениями ролей.

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

Вы можете предоставить документацию по API, включив четкие описания конечных точек, параметров, заголовков, примеров запросов и ответов, кодов ошибок, методов аутентификации и сценариев использования. Кроме того, использование таких инструментов, как спецификации Swagger или OpenAPI, может помочь создать интерактивную документацию, позволяющую разработчикам легко понять и использовать API.

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

Инструменты и технологии, используемые для разработки и тестирования API, включают:

1. Развитие:

- Языки программирования (например, Python, Java, Node.js)

- Инструменты проектирования API (например, Swagger, Postman, Apiary)

- Интегрированные среды разработки (IDE), такие как Visual Studio Code, IntelliJ IDEA.

- Фреймворки (например, Express.js, Flask, Spring Boot)

- Системы контроля версий (например, Git)

2. Тестирование:

- Инструменты тестирования API (например, Postman, SoapUI, JMeter)

- Инструменты для создания макетов (например, WireMock, MockServer)

- Платформы автоматизированного тестирования (например, Selenium, JUnit)

- Инструменты нагрузочного тестирования (например, Apache JMeter, Loader.io)

- Инструменты мониторинга (например, Prometheus, New Relic)