Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Северо-Восточный федеральный университет имени М.К. Аммосова»

Институт математики и информатики

Кафедра «Информационные технологии»

**ОТЧЕТ**

**Учебной (эксплуатационной) практики**

ФИО студента: Никифоров Арсен Германович

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии, профиль: Программирование и информационные технологии

Курс обучения: 2 (второй), группа БА-ФИИТ-21

Вид практики: Учебная (эксплуатационная)

Сроки практики: 22.06.2023 - 05.07.2023 (2 недели)

Место прохождения практики: кафедра «Информационные технологии» (дистанционно)

Руководитель практики: Николай Николаевич Качалин, ст. преподаватель

Ф.И.О. должность, ученое звание руководителя

Якутск, 2023

**Ссылка на репозиторий учебной практики:**<https://github.com/Arsenshmid/Training_Practice>

**Введение**

В данном отчете представлен обзор и анализ трех электронных курсов, пройденных в рамках учебной практики. Целью практики являлось повышение профессиональных навыков и компетенций в области информационных технологий, анализа данных и экономики.

Курсы, рассмотренные в отчете, включают следующие темы:

1. **Курс на платформе Stepik «Интерактивный тренажер по SQL»** (ДВФУ, 35 ч.) - изучение основ реляционной модели данных, запросов SQL к связанным таблицам, работы с базами данных и пользовательскими запросами. Курс состоит из 22 уроков, 10 тестов и 172 интерактивных задач. [Ссылка на курс](https://stepik.org/course/63054/promo)
2. **Курс на платформе НОУ «ИНТУИТ» «Табличный процессор Excel в экономических и финансовых расчетах»** (72 ч.) - обучение работы с различными функциями и инструментами Microsoft Excel для проведения экономических и финансовых анализов. Курс включает 17 лекций, 8 тестов и экзамен. [Ссылка на курс](https://intuit.ru/studies/courses/3659/901/info)
3. **Курс на платформе Stepik «Проектный практикум»** - ознакомление с основами представления данных, протоколами SOAP и RESTful, а также работа с серверной и клиентской частями приложения. Курс состоит из 14 уроков и 3 тестов. [Ссылка на курс](https://stepik.org/course/52624/promo)

В ходе прохождения курсов были получены теоретические знания и практические навыки, которые могут быть применены в дальнейшей профессиональной деятельности. В данном отчете приведены подробные разъяснения и анализ материалов каждого курса, а также обобщение полученных знаний и опыта.

**Основная часть**

В течение учебной практики я прошел три курса, которые охватывают различные аспекты работы с данными и их анализа. Следующие курсы были пройдены на разных платформах:

1. "Интерактивный тренажер по SQL" на платформе Stepik;
2. "Табличный процессор Excel в экономических и финансовых расчетах" на платформе НОУ «ИНТУИТ»;
3. "Проектный практикум" на платформе Stepik.

**Курс "Интерактивный тренажер по SQL"**

На курсе "Интерактивный тренажер по SQL" я изучил основы реляционной модели данных и теории баз данных. В частности, я освоил запросы SQL к связанным таблицам, углубился в работу с базами данных и закрепил навыки написания SQL запросов. В ходе обучения я успешно выполнил 22 урока, 10 тестов и 172 интерактивных задачи, что подтверждает мою компетентность в данной области.

**Курс "Табличный процессор Excel в экономических и финансовых расчетах"**

Курс "Табличный процессор Excel в экономических и финансовых расчетах" позволил мне освоить широкий спектр возможностей программы Excel. В рамках курса я изучил создание и редактирование таблиц и диаграмм, научился использовать встроенные функции для проведения статистического анализа, работу с базами данных, структурирование таблиц, создание сводных таблиц, консолидацию данных, использование финансовых функций и анализ данных, прогнозирование, анализ временных рядов и применение метода скользящей средней. В результате обучения я прошел 17 лекций, 8 тестов и успешно сдал экзамен.

**Курс "Проектный практикум"**

В ходе прохождения курса "Проектный практикум" я изучил различные способы представления данных, такие как SOAP, API, RESTful, правила именования ресурсов в RESTful архитектуре, а также серверную и клиентскую части приложения. Завершив 14 уроков и 3 теста, я смог успешно применить полученные знания на практике.

**Проектная работа**

После завершения всех курсов я разработал проект по курсу "Проектный практикум". В проекте я использовал знания, полученные на курсах, для решения практических задач, связанных с анализом данных и созданием информационных систем. Разработанный проект содержит код программной разработки, который я приложил к этому отчету. Этот проект продемонстрировал мою способность к синтезу и применению полученных знаний в реальных условиях.

**Лабораторная работа №1**

const express = require('express');

const app = express();

app.get('/:number', (req, res) => {

const number = req.params.number;

const numToWords = require('number-to-words');

const result = numToWords.toWords(number);

res.json({ result });

});

app.listen(3000);

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

const express = require('express');

const app = express();

const fs = require('fs');

// Обработчик GET-запроса для решения уравнения

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

const a = parseFloat(req.query.a);

const b = parseFloat(req.query.b);

const c = parseFloat(req.query.c);

// Решение уравнения

const discriminant = b \* b - 4 \* a \* c;

let result;

if (discriminant > 0) {

const x1 = (-b + Math.sqrt(discriminant)) / (2 \* a);

const x2 = (-b - Math.sqrt(discriminant)) / (2 \* a);

result = { x1, x2 };

} else if (discriminant === 0) {

const x = -b / (2 \* a);

result = { x };

} else {

result = 'No real roots';

}

// Отправляем результат в виде JSON

res.json(result);

// Сохраняем результат в файл

const data = JSON.stringify({ url: req.originalUrl, response: result, method: 'GET' }, null, 2);

fs.writeFile('lab\_1.json', data, (err) => {

if (err) {

console.error('Ошибка при сохранении файла:', err);

} else {

console.log('Файл сохранен');

}

});

});

// Запуск сервера на порту 3001

app.listen(3001, () => {

console.log('Сервер запущен на порту 3001');

});

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Возвращает день недели по заданной дате, которая передается как параметр запроса (url?date=12.12.2028);

const express = require('express');

const app = express();

// Обработчик GET-запроса для получения дня недели по заданной дате

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

const date = req.query.date;

const dayOfWeek = getDayOfWeek(date);

// Отправляем результат в виде JSON

res.json({ dayOfWeek });

// Сохраняем результат в файл

const data = JSON.stringify({ url: req.originalUrl, response: { dayOfWeek }, method: 'GET' }, null, 2);

fs.writeFile('lab\_1.json', data, (err) => {

if (err) {

console.error('Ошибка при сохранении файла:', err);

} else {

console.log('Файл сохранен');

}

});

});

// Функция для получения дня недели по заданной дате

function getDayOfWeek(dateStr) {

const date = new Date(dateStr);

const daysOfWeek = ['Вс', 'Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб'];

return daysOfWeek[date.getDay()];

}

// Запуск сервера на порту 3001

app.listen(3001, () => {

console.log('Сервер запущен на порту 3001');

});

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

* Возвращает число Фибоначчи, заданное своим индексом (*в виде параметра пути*);

const express = require('express');

const app = express();

// Обработчик GET-запроса для получения числа Фибоначчи по индексу

app.get('/:index', (req, res) => {

const index = parseInt(req.params.index);

const fibonacciNumber = getFibonacciNumber(index);

// Отправляем результат в виде JSON

res.json({ fibonacciNumber });

// Сохраняем результат в файл

const data = JSON.stringify({ url: req.originalUrl, response: { fibonacciNumber }, method: 'GET' }, null, 2);

fs.writeFile('lab\_2.json', data, (err) => {

if (err) {

console.error('Ошибка при сохранении файла:', err);

} else {

console.log('Файл сохранен');

}

});

});

// Функция для получения числа Фибоначчи по индексу

function getFibonacciNumber(index) {

if (index < 0) {

return NaN;

} else if (index === 0) {

return 0;

} else if (index === 1) {

return 1;

} else {

let a = 0;

let b = 1;

let c;

for (let i = 2; i <= index; i++) {

c = a + b;

a = b;

b = c;

}

return c;

}

}

// Запуск сервера на порту 3001

app.listen(3001, () => {

console.log('Сервер запущен на порту 3001');

});

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

* Возвращает название региона России по заданному номеру региона (*в виде параметра пути*). Например, при заданном *url/72*, должно возвращаться «Тюменская область».

const express = require('express');

const app = express();

// Словарь номеров и названий регионов

const regions = {

'01': 'Республика Адыгея',

'02': 'Республика Башкортостан',

'03': 'Республика Бурятия',

'04': 'Республика Алтай',

'05': 'Республика Дагестан',

'06': 'Республика Ингушетия',

'07': 'Кабардино-Балкарская Республика',

'08': 'Республика Калмыкия',

'09': 'Карачаево-Черкесская Республика',

'10': 'Республика Карелия',

'11': 'Республика Коми',

'12': 'Республика Марий Эл',

'13': 'Республика Мордовия',

'14': 'Республика Саха (Якутия)',

'15': 'Республика Северная Осетия - Алания',

'16': 'Республика Татарстан',

'17': 'Республика Тыва',

'18': 'Удмуртская Республика',

'19': 'Республика Хакасия',

'20': 'Чувашская Республика',

'21': 'Алтайский край',

'22': 'Краснодарский край',

'23': 'Красноярский край',

'24': 'Приморский край',

'25': 'Ставропольский край',

'26': 'Хабаровский край',

'27': 'Амурская область',

'28': 'Архангельская область',

'29': 'Астраханская область',

'30': 'Белгородская область',

'31': 'Брянская область',

'32': 'Владимирская область',

'72': 'Тюменская область'

};

// Обработчик GET-запроса для получения названия региона

app.get('/:regionNumber', (req, res) => {

const regionNumber = req.params.regionNumber;

const regionName = regions[regionNumber];

if (regionName) {

res.json({ region: regionName });

} else {

res.status(404).json({ error: 'Region not found' });

}

});

// Запуск сервера на порту 3000

app.listen(3000, () => {

console.log('Сервер запущен на порту 3000');

});

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Лабораторная работа №2**

Разработать веб-службы для работы с базой данных.

Необходимо создать базу данных. В базе данных должны присутствовать следующие сущности: «**заявка**», «**клиент**» и «**услуга**». Заявка оформляется на одного клиента. Заявка может содержать несколько услуг.

-- Таблица "клиент"

CREATE TABLE clients (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL

);

-- Таблица "услуга"

CREATE TABLE services (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

price DECIMAL(10, 2) NOT NULL

);

-- Таблица "заявка"

CREATE TABLE requests (

id SERIAL PRIMARY KEY,

client\_id INT NOT NULL,

service\_id INT NOT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT NOW(),

FOREIGN KEY (client\_id) REFERENCES clients (id),

FOREIGN KEY (service\_id) REFERENCES services (id)

);

// Добавить клиента

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

// Вставка данных в таблицу клиентов

});

// Получить список клиентов

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

// Получение списка клиентов

});

// Получить клиента по ID

app.get('/api/clients/:id', async (req, res) => {

// Получение клиента по ID

});

// Обновить клиента по ID

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

// Обновление данных клиента по ID

});

// Удалить клиента по ID

app.delete('/api/clients/:id', async (req, res) => {

// Удаление клиента по ID

});

Server.js для создания сервера

const express = require('express');

const cors = require('cors');

const knex = require('knex');

const db = knex({

client: 'pg',

connection: {

host: '127.0.0.1',

user: 'postgres',

password: '1',

database: 'lab2\_db'

}

});

const app = express();

app.use(cors());

app.use(express.json());

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

try {

const clients = await db.select('\*').from('clients');

res.json(clients);

} catch (err) {

res.status(500).json({ error: 'Ошибка на сервере' });

}

});

// Добавить нового клиента

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

try {

const { name } = req.body;

const [id] = await db('clients').insert({ name }).returning('id');

res.json({ id, name });

} catch (err) {

res.status(500).json({ error: 'Ошибка на сервере' });

}

});

// Обновить клиента

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

try {

const { id } = req.params;

const { name } = req.body;

await db('clients').where({ id }).update({ name });

res.json({ id, name });

} catch (err) {

res.status(500).json({ error: 'Ошибка на сервере' });

}

});

// Удалить клиента

app.delete('/clients/:id', async (req, res) => {

try {

const { id } = req.params;

await db('clients').where({ id }).del();

res.json({id });

} catch (err) {

res.status(500).json({ error: 'Ошибка на сервере' });

}

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

Добавил код для создания API:

Index.js

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

const express = require('express');

const app = express();

const pool = new Pool({

user: 'your\_db\_user',

host: 'localhost',

database: 'your\_db\_name',

password: 'your\_db\_password',

port: 5432,

});

// Добавить клиента

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

try {

const { name, email, phone } = req.body;

const result = await pool.query('INSERT INTO clients (name, email, phone) VALUES ($1, $2, $3) RETURNING id', [name, email, phone]);

res.status(201).json({ id: result.rows[0].id });

} catch (err) {

console.error(err);

res.status(500).json({ error: 'An error occurred while adding a client' });

}

});

// Получить список клиентов

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

try {

const result = await pool.query('SELECT \* FROM clients');

res.status(200).json(result.rows);

} catch (err) {

console.error(err);

res.status(500).json({ error: 'An error occurred while fetching clients' });

}

});

// Получить клиента по ID

app.get('/api/clients/:id', async (req, res) => {

try {

const { id } = req.params;

const result = await pool.query('SELECT \* FROM clients WHERE id = $1', [id]);

if (result.rowCount === 0) {

res.status(404).json({ error: 'Client not found' });

return;

}

res.status(200).json(result.rows[0]);

} catch (err) {

console.error(err);

res.status(500).json({ error: 'An error occurred while fetching the client' });

}

});

// Обновить клиента по ID

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

try {

const { id } = req.params;

const { name, email, phone } = req.body;

await pool.query('UPDATE clients SET name = $1, email = $2, phone = $3 WHERE id = $4', [name, email, phone, id]);

res.status(200).json({ message: 'Client updated successfully' });

} catch (err) {

console.error(err);

res.status(500).json({ error: 'An error occurred while updating the client' });

}

});

// Удалить клиента по ID

app.delete('/api/clients/:id', async (req, res) => {

try {

const { id } = req.params;

await pool.query('DELETE FROM clientsWHERE id = $1', [id]);

res.status(200).json({ message: 'Client deleted successfully' });

} catch (err) {

console.error(err);

res.status(500).json({ error: 'An error occurred while deleting the client' });

}

});

В итоге server и App выглядят следующим образом:

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function App() {

const [clients, setClients] = useState([]);

const [services, setServices] = useState([]);

const [requests, setRequests] = useState([]);

useEffect(() => {

getClients();

getServices();

getRequests();

}, []);

const getClients = async () => {

try {

const res = await axios.get('http://localhost:3000/clients');

setClients(res.data);

} catch (err) {

console.error(err);

}

};

const getServices = async () => {

try {

const res = await axios.get('http://localhost:3000/services');

setServices(res.data);

} catch (err) {

console.error(err);

}

};

const getRequests = async () => {

try {

const res = await axios.get('http://localhost:3000/requests');

setRequests(res.data);

} catch (err) {

console.error(err);

}

};

return (

<div>

<h1>Клиенты</h1>

<table>

<thead>

<tr>

<th>ID</th>

<th>Имя</th>

</tr>

</thead>

<tbody>

{clients.map((client) => (

<tr key={client.id}>

<td>{client.id}</td>

<td>{client.name}</td>

</tr>

))}

</tbody>

</table>

<h1>Услуги</h1>

<table>

<thead>

<tr>

<th>ID</th>

<th>Название</th>

<th>Цена</th>

</tr>

</thead>

<tbody>

{services.map((service) => (

<tr key={service.id}>

<td>{service.id}</td>

<td>{service.name}</td>

<td>{service.price} руб.</td>

</tr>

))}

</tbody>

</table>

<h1>Заявки</h1>

<table>

<thead>

<tr>

<th>ID</th>

<th>ID клиента</th>

<th>ID услуги</th>

</tr>

</thead>

<tbody>

{requests.map((request) => (

<tr key={request.id}>

<td>{request.id}</td>

<td>{request.client\_id}</td>

<td>{request.service\_id}</td>

</tr>

))}

</tbody>

</table>

</div>

);

}

export default App;  
  
Результат работы веб-службы для работы с базой данных, при запуске localhost отображает следующее:  
Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

**Лабораторная работа №3**

CREATE TABLE clients (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

CREATE TABLE services (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

price NUMERIC NOT NULL

);

CREATE TABLE requests (

id SERIAL PRIMARY KEY,

client\_id INTEGER REFERENCES clients(id),

service\_id INTEGER REFERENCES services(id)

);

Client.java

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Client {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// конструктор

public Client() {}

public Client(String name) {

this.name = name;

}

// геттеры и сеттеры

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

}

Service.java

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Service {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private double price;

// Constructors

public Service() {}

public Service(String name, double price) {

this.name = name;

this.price = price;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

Request.java

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class Request {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

private Client client;

@ManyToOne

private Service service;

// Constructors

public Request() {}

public Request(Client client, Service service) {

this.client = client;

this.service = service;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Client getClient() {

return client;

}

public void setClient(Client client) {

this.client = client;

}

public Service getService() {

return service;

}

public void setService(Service service) {

this.service = service;

}

}

Контроллер

package com.example.demo.controller;

import com.example.demo.model.Client;

import com.example.demo.model.Request;

import com.example.demo.model.Service;

import com.example.demo.repository.ClientRepository;

import com.example.demo.repository.RequestRepository;

import com.example.demo.repository.ServiceRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

public class WebController {

private final ClientRepository clientRepository;

private final ServiceRepository serviceRepository;

private final RequestRepository requestRepository;

@Autowired

public WebController(ClientRepository clientRepository, ServiceRepository serviceRepository, RequestRepository requestRepository) {

this.clientRepository = clientRepository;

this.serviceRepository = serviceRepository;

this.requestRepository = requestRepository;

}

// Метод для инициализации тестовыми данными

@GetMapping("/initTestData")

public String initTestData() {

// Создание клиентов

Client client1 = new Client("John Doe");

Client client2 = new Client("Jane Smith");

// Создание услуг

Service service1 = new Service("Web Development", 2500);

Service service2 = new Service("Graphic Design", 1200);

// Создание запросов

Request request1 = new Request(client1, service1);

Request request2 = new Request(client2, service2);

// Сохранение данных в базу данных

clientRepository.save(client1);

clientRepository.save(client2);

serviceRepository.save(service1);

serviceRepository.save(service2);

requestRepository.save(request1);

requestRepository.save(request2);

return "Test data initialized successfully!";

}

// Примеры других методов для операций чтения/добавления/изменения/удаления записей:

// Получение всех клиентов

@GetMapping("/clients")

public List<Client> getAllClients() {

return clientRepository.findAll();

}

// Добавление нового клиента

@PostMapping("/clients")

public Client addNewClient(@RequestBody Client client) {

return clientRepository.save(client);

}

// Получение всех услуг

@GetMapping("/services")

public List<Service> getAllServices() {

return serviceRepository.findAll();

}

// Изменение информации об услуге по ID

@PutMapping("/services/{id}")

public Service updateService(@PathVariable Long id, @RequestBody Service updatedService) {

Service service = serviceRepository.findById(id).orElse(null);

if (service != null) {

service.setName(updatedService.getName());

service.setPrice(updatedService.getPrice());

return serviceRepository.save(service);

}

return null;

}

// Удаление услуги по ID

@DeleteMapping("/services/{id}")

public String deleteService(@PathVariable Long id) {

Service service = serviceRepository.findById(id).orElse(null);

if (service != null) {

serviceRepository.delete(service);

return "Service with ID " + id + " deleted successfully!";

}

return "Service with ID " + id + " not found!";

}

}

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Создание SOAP веб-службы**

**package** com.example.demo.soap;

**import** com.example.demo.model.Service;

**import** com.example.demo.repository.ServiceRepository;

**import** com.example.services.GetServiceRequest;

**import** com.example.services.GetServiceResponse;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.ws.server.endpoint.annotation.Endpoint;

**import** org.springframework.ws.server.endpoint.annotation.PayloadRoot;

**import** org.springframework.ws.server.endpoint.annotation.RequestPayload;

**import** org.springframework.ws.server.endpoint.annotation.ResponsePayload;

@Endpoint

**public** **class** ServiceEndpoint {

**private** **static** **final** String NAMESPACE\_URI = "http://www.example.com/services";

**private** **final** ServiceRepository serviceRepository;

@Autowired

**public** ServiceEndpoint(ServiceRepository serviceRepository) {

**this**.serviceRepository = serviceRepository;

}

@PayloadRoot(namespace = NAMESPACE\_URI, localPart = "GetServiceRequest")

@ResponsePayload

**public** GetServiceResponse getService(@RequestPayload GetServiceRequest request) {

GetServiceResponse response = **new** GetServiceResponse();

Service service = serviceRepository.findById(request.getId()).orElse(**null**);

**if** (service != **null**) {

com.example.services.Service soapService = **new** com.example.services.Service();

soapService.setId(service.getId());

soapService.setName(service.getName());

soapService.setPrice(service.getPrice());

response.setService(soapService);

}

**return** response;

}

}

Теперь, SOAP веб-служба готова к использованию. Тестировал её с помощью SOAP UI. Пример SOAP запроса для проверки работы веб-службы:

**<soapenv:Envelope** xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://www.example.com/services"**>**

**<soapenv:Header/>**

**<soapenv:Body>**

**<ser:GetServiceRequest>**

**<ser:id>**1**</ser:id>**

**</ser:GetServiceRequest>**

**</soapenv:Body>**

**</soapenv:Envelope>**

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, линия

Автоматически созданное описание

Пример 2: Запрос на добавление новой услуги

SOAP запрос:

xml

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://www.example.com/services">

<soapenv:Header/>

<soapenv:Body>

<ser:AddNewServiceRequest>

<ser:name>Web Design</ser:name>

<ser:price>1800</ser:price>

</ser:AddNewServiceRequest>

</soapenv:Body>

</soapenv:Envelope>

SOAP ответ:

xml

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://www.example.com/services">

<soapenv:Header/>

<soapenv:Body>

<ser:AddNewServiceResponse>

<ser:serviceAdded>true</ser:serviceAdded>

</ser:AddNewServiceResponse>

</soapenv:Body>

</soapenv:Envelope>

Изображение выглядит как текст, снимок экрана, линия

Автоматически созданное описание

Пример 3: Запрос на изменение информации об услуге

SOAP запрос:

xml

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://www.example.com/services">

<soapenv:Header/>

<soapenv:Body>

<ser:UpdateServiceRequest>

<ser:id>3</ser:id>

<ser:name>Mobile App Development</ser:name>

<ser:price>3500</ser:price>

</ser:UpdateServiceRequest>

</soapenv:Body>

</soapenv:Envelope>

SOAP ответ:

xml

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://www.example.com/services">

<soapenv:Header/>

<soapenv:Body>

<ser:UpdateServiceResponse>

<ser:serviceUpdated>true</ser:serviceUpdated>

</ser:UpdateServiceResponse>

</soapenv:Body>

</soapenv:Envelope>

Изображение выглядит как текст, линия, снимок экрана

Автоматически созданное описание

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function App() {

const [data, setData] = useState({});

useEffect(() => {

// Вызываем серверную часть для получения данных и замены пробелов

axios

.get('http://localhost:5000/api/data')

.then((response) => setData(response.data))

.catch((error) => console.error('Error fetching data:', error));

}, []);

return (

<div>

<h1>Modified Data from External API</h1>

<pre>{JSON.stringify(data, null, 2)}</pre>

</div>

);

}

export default App;

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

**Лабораторная работа №5**

-- Создание таблицы "заявка"

CREATE TABLE заявка (

id INT PRIMARY KEY AUTO\_INCREMENT,

клиент\_id INT NOT NULL,

название VARCHAR(255) NOT NULL,

описание TEXT,

FOREIGN KEY (клиент\_id) REFERENCES клиент(id)

);

-- Создание таблицы "клиент"

CREATE TABLE клиент (

id INT PRIMARY KEY AUTO\_INCREMENT,

имя VARCHAR(100) NOT NULL,

телефон VARCHAR(20),

электронная\_почта VARCHAR(100)

);

-- Создание таблицы "услуга"

CREATE TABLE услуга (

id INT PRIMARY KEY AUTO\_INCREMENT,

название VARCHAR(255) NOT NULL,

описание TEXT

);

App.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import ClientList from './components/ClientList';

function App() {

const [newClientName, setNewClientName] = useState('');

const [clients, setClients] = useState([]);

const handleAddClient = () => {

if (newClientName.trim() === '') return;

axios.post('/api/clients', { name: newClientName })

.then((response) => {

setClients([...clients, response.data]);

setNewClientName('');

})

.catch((error) => console.error('Error adding client', error));

};

const handleGetClients = () => {

axios.get('/api/clients')

.then((response) => setClients(response.data))

.catch((error) => console.error('Error fetching clients', error));

};

useEffect(() => {

handleGetClients();

}, []); // Добавили пустой массив зависимостей, чтобы выполнить это только при первом рендеринге

return (

<div className="App">

<h1>Client List</h1>

<ClientList clients={clients} />

<div>

<input

type="text"

value={newClientName}

onChange={(e) => setNewClientName(e.target.value)}

/>

<button onClick={handleAddClient}>Add Client</button>

<button onClick={handleGetClients}>Get Clients</button> {/\* Добавили кнопку \*/}

</div>

</div>

);

}

export default App;

server.js

const express = require('express');

const knex = require('knex');

const app = express();

const port = 3001;

const db = knex({

client: 'pg',

connection: {

host: '127.0.0.1',

user: 'postgres',

password: '1',

database: 'lab2\_db'

}

});

app.use(express.json());

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

try {

const clients = await db.select().from('clients');

res.json(clients);

} catch (err) {

console.error('Error executing query', err);

res.status(500).json({ error: 'Internal server error' });

}

});

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

const { name } = req.body;

if (!name) {

return res.status(400).json({ error: 'Name is required' });

}

try {

const newClient = await db('clients').insert({ name }).returning('\*');

res.status(201).json(newClient[0]);

} catch (err) {

console.error('Error executing query', err);

res.status(500).json({ error: 'Internal server error' });

}

});

app.listen(port, () => {

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

});

ClientList.js

import React from 'react';

function ClientList({ clients }) {

return (

<div>

<h2>Client List</h2>

<ul>

{clients.map((client) => (

<li key={client.id}>{client.name}</li>

))}

</ul>

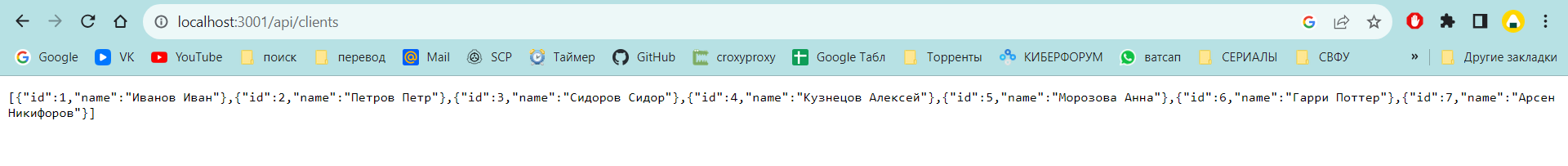
</div>

);

}

export default ClientList;

Изображение выглядит как текст, снимок экрана, Шрифт, вода

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import ClientList from './components/ClientList';

function App() {

const [newClientName, setNewClientName] = useState('');

const [clients, setClients] = useState([]);

const [mockData, setMockData] = useState(null);

const [modifiedMockData, setModifiedMockData] = useState(null);

const handleAddClient = () => {

if (newClientName.trim() === '') return;

axios.post('/api/clients', { name: newClientName })

.then((response) => {

setClients([...clients, response.data]);

setNewClientName('');

})

.catch((error) => console.error('Error adding client', error));

};

const handleGetClients = () => {

axios.get('/api/clients')

.then((response) => setClients(response.data))

.catch((error) => console.error('Error fetching clients', error));

};

const handleGetMockData = () => {

axios.get('http://www.mocky.io/v2/5c7db5e13100005a00375fda')

.then((response) => {

setMockData(response.data);

const modifiedData = response.data.replace(/ /g, '\_');

setModifiedMockData(modifiedData);

})

.catch((error) => console.error('Error fetching mock data', error));

};

useEffect(() => {

handleGetClients();

}, []);

return (

<div className="App">

<h1>Client List</h1>

<ClientList clients={clients} />

{mockData && (

<div>

<h2>Mock Data</h2>

<p>{JSON.stringify(mockData)}</p>

<h2>Modified Mock Data</h2>

<p>{JSON.stringify(modifiedMockData)}</p>

</div>

)}

<div>

<input

type="text"

value={newClientName}

onChange={(e) => setNewClientName(e.target.value)}

/>

<button onClick={handleAddClient}>Add Client</button>

<button onClick={handleGetClients}>Get Clients</button>

<button onClick={handleGetMockData}>Get Mock Data</button>

</div>

</div>

);

}

export default App;

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Итоговая лабораторная работа.**

Сервер

const express = require('express');

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

const knex = require('knex');

const db = knex({

client: 'pg',

connection: {

host: '127.0.0.1',

user: 'postgres',

password: '1',

database: 'arsenshmid'

}

});

const app = express();

app.use(bodyParser.json());

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

const { title, questions } = req.body;

try {

const [testId] = await db('tests').insert({ title }).returning('id');

const formattedQuestions = questions.map(question => ({

test\_id: testId,

text: question.text

}));

const [questionIds] = await db('questions').insert(formattedQuestions).returning('id');

const formattedAnswers = [];

questions.forEach((question, index) => {

const answers = question.answers.map(answer => ({

question\_id: questionIds[index],

text: answer.text,

is\_correct: answer.isCorrect

}));

formattedAnswers.push(...answers);

});

await db('answers').insert(formattedAnswers);

return res.status(201).json({ message: 'Тест успешно создан.' });

} catch (error) {

console.error('Ошибка при создании теста:', error);

return res.status(500).json({ error: 'Ошибка при создании теста.' });

}

});

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

try {

const tests = await db('tests').select('id', 'title');

return res.status(200).json(tests);

} catch (error) {

console.error('Ошибка при получении тестов:', error);

return res.status(500).json({ error: 'Ошибка при получении тестов.' });

}

});

app.get('/api/tests/:testId', async (req, res) => {

const { testId } = req.params;

try {

const test = await db('tests').where('id', testId).first();

if (!test) {

return res.status(404).json({ error: 'Тест не найден.' });

}

const questions = await db('questions').where('test\_id', testId);

const formattedQuestions = await Promise.all(questions.map(async question => {

const answers = await db('answers').where('question\_id', question.id);

return { ...question, answers };

}));

return res.status(200).json({ test, questions: formattedQuestions });

} catch (error) {

console.error('Ошибка при получении теста:', error);

return res.status(500).json({ error: 'Ошибка при получении теста.' });

}

});

app.post('/api/tests/:testId/submit', async (req, res) => {

const { testId } = req.params;

const userAnswers = req.body;

try {

const questions = await db('questions').where('test\_id', testId);

let score = 0;

questions.forEach((question, index) => {

const correctAnswer = question.answers.find(answer => answer.is\_correct);

if (correctAnswer && correctAnswer.id === userAnswers[index].answerId) {

score++;

}

});

return res.status(200).json({ score, totalQuestions: questions.length });

} catch (error) {

console.error('Ошибка при отправке результатов теста:', error);

return res.status(500).json({ error: 'Ошибка при отправке результатов теста.' });

}

});

const PORT = 5000;

app.listen(PORT, () => {

console.log(`Сервер запущен на порту ${PORT}.`);

});

клиент

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function App() {

const [title, setTitle] = useState('');

const [questions, setQuestions] = useState([{ text: '', answers: [{ text: '', isCorrect: false }] }]);

const [isExitModalOpen, setIsExitModalOpen] = useState(false);

const [userAnswers, setUserAnswers] = useState([]);

const [availableTests, setAvailableTests] = useState([]);

const [selectedTest, setSelectedTest] = useState(null);

useEffect(() => {

axios.get('/api/tests')

.then(response => {

setAvailableTests(response.data);

})

.catch(error => {

console.error('Error fetching available tests:', error);

});

}, []);

const handleAddQuestion = () => {

setQuestions([...questions, { text: '', answers: [{ text: '', isCorrect: false }] }]);

};

const handleRemoveQuestion = (index) => {

const updatedQuestions = [...questions];

updatedQuestions.splice(index, 1);

setQuestions(updatedQuestions);

};

const handleAddAnswer = (questionIndex) => {

const updatedQuestions = [...questions];

updatedQuestions[questionIndex].answers.push({ text: '', isCorrect: false });

setQuestions(updatedQuestions);

};

const handleRemoveAnswer = (questionIndex, answerIndex) => {

const updatedQuestions = [...questions];

updatedQuestions[questionIndex].answers.splice(answerIndex, 1);

setQuestions(updatedQuestions);

};

const handleCheckboxChange = (questionIndex, answerIndex) => {

const updatedQuestions = [...questions];

updatedQuestions[questionIndex].answers = updatedQuestions[questionIndex].answers.map((answer, index) => ({

...answer,

isCorrect: index === answerIndex,

}));

setQuestions(updatedQuestions);

};

const handleSaveTest = () => {

if (!title || !questions.some(q => q.text)) {

alert('Заполните все поля перед сохранением теста.');

return;

}

const data = { title, questions };

axios.post('/api/tests', data)

.then(response => {

alert(response.data.message);

setTitle('');

setQuestions([{ text: '', answers: [{ text: '', isCorrect: false }] }]);

})

.catch(error => {

alert(error.response.data.error);

});

};

const handleSubmitTest = (testId) => {

const formattedAnswers = userAnswers.map(answer => ({

answerId: answer.answerId,

isCorrect: answer.isCorrect

}));

axios.post(`/api/tests/${testId}/submit`, formattedAnswers)

.then(response => {

alert(`Ваш результат: ${response.data.score}/${response.data.totalQuestions}`);

setSelectedTest(null);

setUserAnswers([]);

})

.catch(error => {

console.error('Ошибка при отправке теста:', error);

});

};

const handleStartTest = (test) => {

axios.get(`/api/tests/${test.id}`)

.then(response => {

setSelectedTest(response.data.test);

setUserAnswers(Array(response.data.questions.length).fill({ answerId: null, isCorrect: false }));

})

.catch(error => {

console.error('Ошибка при получении теста:', error);

});

};

const handleCancelTest = () => {

setSelectedTest(null);

setUserAnswers([]);

};

const handleUserAnswer = (questionIndex, answerId, isCorrect) => {

const updatedUserAnswers = [...userAnswers];

updatedUserAnswers[questionIndex] = { answerId, isCorrect };

setUserAnswers(updatedUserAnswers);

};

const handleOpenExitModal = () => {

setIsExitModalOpen(true);

};

const handleCloseExitModal = () => {

setIsExitModalOpen(false);

};

return (

<div>

<h1>Система тестирования</h1>

<div>

<label>Название теста:</label>

<input

type="text"

value={title}

onChange={(e) => setTitle(e.target.value)}

/>

</div>

<div>

{questions.map((question, index) => (

<div key={index}>

<label>Вопрос {index + 1}:</label>

<input

type="text"

value={question.text}

onChange={(e) => {

const updatedQuestions = [...questions];

updatedQuestions[index].text = e.target.value;

setQuestions(updatedQuestions);

}}

/>

{question.answers.map((answer, ansIndex) => (

<div key={ansIndex}>

<input

type="text"

value={answer.text}

onChange={(e) => {

const updatedQuestions = [...questions];

updatedQuestions[index].answers[ansIndex].text = e.target.value;

setQuestions(updatedQuestions);

}}

/>

<input

type="checkbox"

checked={answer.isCorrect}

onChange={() => handleCheckboxChange(index, ansIndex)}

/>

{ansIndex === question.answers.length - 1 && (

<button onClick={() => handleAddAnswer(index)}>Добавить ответ</button>

)}

{ansIndex > 1 && (

<button onClick={() => handleRemoveAnswer(index, ansIndex)}>Удалить ответ</button>

)}

</div>

))}

<button onClick={handleAddQuestion}>Добавить вопрос</button>

{index > 0 && (

<button onClick={() => handleRemoveQuestion(index)}>Удалить вопрос</button>

)}

</div>

))}

</div>

<button onClick={handleOpenExitModal}>Выйти</button>

{isExitModalOpen && (

<div>

<p>Вы хотите сохранить тест перед выходом?</p>

<button onClick={handleSaveTest}>Сохранить</button>

<button onClick={handleCloseExitModal}>Не сохранять</button>

</div>

)}

<button onClick={handleSaveTest}>Сохранить</button>

{availableTests.length > 0 && (

<div>

<h2>Доступные тесты</h2>

<ul>

{availableTests.map(test => (

<li key={test.id}>

{test.title}{' '}

<button onClick={() => handleStartTest(test)}>Начать</button>

</li>

))}

</ul>

</div>

)}

{selectedTest && (

<div>

<h2>Прохождение теста - {selectedTest.title}</h2>

{selectedTest.questions.map((question, questionIndex) => (

<div key={questionIndex}>

<p>{question.text}</p>

<ul>

{question.answers.map((answer, ansIndex) => (

<li key={ansIndex}>

<input

type="radio"

name={`question-${questionIndex}`}

value={answer.text}

checked={userAnswers[questionIndex].answerId === answer.id}

onChange={() => handleUserAnswer(questionIndex, answer.id, answer.is\_correct)}

/>

{answer.text}

</li>

))}

</ul>

</div>

))}

<button onClick={() => handleSubmitTest(selectedTest.id)}>Завершить тест</button>

<button onClick={handleCancelTest}>Отмена</button>

</div>

)}

</div>

);

}

export default App;

Получаем отображение тестов и возможность их пройти  
Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

**Заключение**

В результате учебной практики я успешно прошел три курса, расширил свои знания в области SQL, работы с табличным процессором Excel и разработки проектов. Благодаря этому опыту я могу применять полученные знания на практике и эффективно решать задачи, связанные с работой с данными, анализом данных и разработкой проектов.

Я предоставил подробный отчет о своей учебной практике, который включает краткое содержание выполненных работ, дневник практики, заполненный на каждый день практики, описание разработки проекта по курсу "Проектный практикум", скриншоты и полученные сертификаты. Все задания были выполнены в соответствии с календарным планом.

**Сертификаты**

БД <https://stepik.org/cert/2111716>

НОУ ИНТУИТ <https://intuit.ru/verifydiplomas/101611542>

**Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание**

Изображение выглядит как текст, Шрифт, круг, бумага

Автоматически созданное описание

ДНЕВНИК

Учебной технологической (проектно-технологической) практики

Календарный план выполненной работы за время прохождения практики:

|  |  |  |
| --- | --- | --- |
| Неделя | Наименование модуля и описание | Дата выполнения |
| 1. | Курс “Интерактивный тренажер по SQL”.  Выполнил 1 и 2 модули. | с 12.06.23  по 18.06.23 |
| 2. | Выполнил 3-й модуль “Интерактивный тренажер по SQL”. | с 18.06.23 по 25.06.23 |
| 3. | Изучил все лекции и сделал все тесты курса "Табличный процессор Excel в экономических и финансовых расчетах". Все сдал на 5 | с 25.06.23  по 1.07.23 |
| 4. | Прошел и сдал курс "Проектный практикум" на платформе Stepik. | c 1.07.23  по 8.07.23 |
| 5. | Завершение проекта и подготовка отчета. 5. Сформировал окончательный вид сайта и начал оформление отчета. | с 08.07.23  по 12.07.23 |