|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Задание №5**  **по теме «*SPWA via Node.js*»**  **по дисциплине «Web-программирование»** | |
|  | Выполнил:  Студент группы ТКИ-542  Шуриков Д. А.  Кожак И. А.  Проверил:  Доцент кафедры УиЗИ, к.т.н.  Сафронов А. И. |
| Москва 2024 | |

Оглавление

[1. Цель работы 3](#_Toc185632405)

[2. ФОРМУЛИРОВАКА ЗАДАЧИ 4](#_Toc185632406)

[3. Описание предметной области 5](#_Toc185632407)

[3.1. Формулировка задачи 5](#_Toc185632408)

[3.2. ER-диаграмма данных 6](#_Toc185632409)

[3.3. Таблица соответствия переменных 6](#_Toc185632410)

[3.4. Архитектура системы (связка: frontend, backend, database) 8](#_Toc185632411)

[4. Web-страница 9](#_Toc185632412)

[4.1. Инструкция по развёртке системы (подгрузка запрашиваемых (required) модулей, взаимная увязка технологий). 9](#_Toc185632413)

[4.2. Код web-приложения (server.js) 9](#_Toc185632414)

[4.3. Код Web-приложения (CustomersList.vue) 22](#_Toc185632415)

[4.4. Код Web-приложения (DevicesList.vue) 28](#_Toc185632416)

[4.5. Код Web-приложения (ServicesList.vue) 33](#_Toc185632417)

[4.6. Код Web-приложения (TariffsList.vue) 39](#_Toc185632418)

[4.7. Результат отображения в браузере 45](#_Toc185632419)

[4.8. Схема Сетей Петри 52](#_Toc185632420)

[5. вывод по работе 56](#_Toc185632421)

1. Цель работы

Создать масштабируемое одностраничное web-приложение на базе фреймворка Vue.js (по шаблону Vite.js, подгружаемому командой npm init vue@latest), отражающее требуемый функционал и тематику согласно варианту индивидуального задания.

1. ФОРМУЛИРОВАКА ЗАДАЧИ

Разработать фронтенд на Vue.js+Vite.js (порт 5173), бэкенд на базе фреймворка Express.js (порт 8080), базу данных PostgreSQL (порт 5432). Также требуется реализовать методологию CRUD, а именно:

* отображать сущности основного отношения выбранной тематики в функциональных списках (<select></select>);
* предоставлять пользователю возможность редактирования атрибутов выбранной / выбранных из списка сущностей;
* предоставлять пользователю возможность для добавления новых сущностей;
* предоставлять возможность удаления избыточных сущностей.

Дополнительно необходимо имплементировать дружественный к пользователю графический интерфейс, а именно:

* отдельные окна (шаблоны / templates) для решения отдельных задач по тематике – не решать все задачи в одном окне (шаблоне / template) – использовать компоненты Vue.js;
* разместить информацию от наиболее приоритетной к менее приоритетной;
* предусмотреть демонстрацию иллюстрации «ожидание» в момент ожидания пользователем откликов на запросы к базе данных;
* предусмотреть текстовую справку-инструкцию по работе с системой.

1. Описание предметной области
   1. Формулировка задачи

В данном проекте для проектирования реляционной базы данных была выбрана компания, занимающаяся поставкой услуг связи.

База данных создаётся для автоматизации обработки заказов. Конечный вариант базы данных содержит в себе информацию об покупателях, тарифах, устройств и их услугах.

Данная системы должна выполнять следующие функции:

1. Показывать список тарифов и цену за единицу услуги.
2. Показывать список абонентов по данному тарифу.
3. Показывать список самых распространенных тарифов.
4. Показывать процентное распределение тарифов среди абонентов.

Выделим базовые сущности предметной области:

1. Сущность «Тариф» (Общая, связывающая все нижеперчисленные) содержит в себе идентификатор тарифа, его название, описание, услуга, устройство, абонент, цена;
2. Сущность «Услуга» содержит в себе идентификатор услуги, название, а также описание услуги;
3. Сущность «Устройство» содержит в себе идентификатор устройства и его название;
4. Сущность «Абонент» содержит в себе идентификатор, имя и фамилию, электронную почту и идентификатор тарифа;
   1. ER-диаграмма данных

|  |
| --- |
|  |
| 1. – ER-диаграмма |

* 1. Таблица соответствия переменных

Таблица 1 – Сущность «Тариф»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Тариф | Идентификатор тарифа | id | Целое число | Первичный ключ |
| Название тарифа | name | Строка (255) | Обязательное поле |
| Описание | description | Строка (255) | Обязательное поле |
| Услуга | services\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) услуга |
| Устройство | device\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) устройство |
| Цена | price | Число |  |

Таблица 2 – Сущность «Устройство»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Устройство | Идентификатор устройства | id | Целое число | Первичный ключ |
| Название устройства | name | Строка (255) | Обязательное поле |

Таблица 3 – Сущность «Услуга»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Услуга | Идентификатор услуги | id | Целое число | Первичный ключ |
| Название | name | Строка (255) | Обязательное, уникальное поле |
| Описанние | description | Строка (255) |  |

Таблица 4 – Сущность «Абонент»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Абонент | Идентификатор абонента | id | Целое число | Первичный ключ |
| Имя | name | Строка (255) | Обязательное поле |
| Фамилия | surname | Строка (255) | Обязательное поле |
| Электронная почта | Email | Строка (255) | Обязательное поле |
| Идентификатор тарифа | tariff\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) тариф |

* 1. Архитектура системы (связка: frontend, backend, database)

|  |
| --- |
|  |
| 1. – Архитектура системы |

1. Web-страница
   1. Инструкция по развёртке системы (подгрузка запрашиваемых (required) модулей, взаимная увязка технологий).
2. Установка Node.js и npm:

* sudo apt update
* sudo apt install nodejs npm

1. Установка PostgreSQL:

* sudo apt install postgresql postgresql-contrib

1. Установка Axios:

* npm install axios

1. Установка зависимостей для сервиса:

* npm install express cors pg
  1. Код web-приложения (server.js)

import express from 'express';

import cors from 'cors';

import pg from 'pg';

const pool = new pg.Pool({

  user: 'postgres',

  host: 'localhost',

  database: 'communication\_services',

  port: 5432

});

const app = express();

app.use(express.json());

app.use(cors());

class ServicesController {

  async getServices(req, res) {

    try {

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

      res.json(services.rows);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error fetching services' });

    }

  }

  async addService(req, res) {

    const { name, description } = req.body;

    if (!name || !description) {

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

    }

    try {

      const newService = await pool.query(

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

          [name, description]

      );

      res.status(201).json(newService.rows[0]);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error adding service' });

    }

  }

  async updateService(req, res) {

    const { id } = req.params;

    const { name, description } = req.body;

    if (!name || !description) {

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

    }

    const serviceId = parseInt(id, 10);

    if (isNaN(serviceId)) {

      return res.status(400).json({ error: 'Invalid service ID' });

    }

    try {

      const updatedService = await pool.query(

          'UPDATE services SET name = $1, description = $2 WHERE id = $3 RETURNING \*',

          [name, description, serviceId]

      );

      if (updatedService.rows.length > 0) {

        res.json(updatedService.rows[0]);

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error updating service' });

    }

  }

  async deleteService(req, res) {

    const { id } = req.params;

    if (!id || isNaN(id)) {

      return res.status(400).json({ error: 'Invalid service ID' });

    }

    const serviceId = parseInt(id, 10);

    try {

      const result = await pool.query('DELETE FROM services WHERE id = $1', [serviceId]);

      if (result.rowCount > 0) {

        res.json({ message: 'Service deleted successfully' });

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error deleting service' });

    }

  }

}

class DevicesController {

  async getDevices(req, res) {

    try {

      const devices = await pool.query('SELECT \* FROM device');

      res.json(devices.rows);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error fetching devices' });

    }

  }

  async addDevice(req, res) {

    const { name } = req.body;

    if (!name) {

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

    }

    try {

      const newDevice = await pool.query(

          'INSERT INTO device (name) VALUES ($1) RETURNING \*',

          [name]

      );

      res.status(201).json(newDevice.rows[0]);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error adding device' });

    }

  }

  async updateDevice(req, res) {

    const { id } = req.params;

    const { name } = req.body;

    if (!name) {

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

    }

    const device\_id = parseInt(id, 10);

    if (isNaN(device\_id)) {

      return res.status(400).json({ error: 'Invalid device ID' });

    }

    try {

      const updatedDevice = await pool.query(

          'UPDATE device SET name = $1 WHERE id = $2 RETURNING \*',

          [name, device\_id]

      );

      if (updatedDevice.rows.length > 0) {

        res.json(updatedDevice.rows[0]);

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error updating device' });

    }

  }

  async deleteDevice(req, res) {

    const { id } = req.params;

    if (!id || isNaN(id)) {

      return res.status(400).json({ error: 'Invalid device ID' });

    }

    const device\_id = parseInt(id, 10);

    try {

      const result = await pool.query('DELETE FROM device WHERE id = $1', [device\_id]);

      if (result.rowCount > 0) {

        res.json({ message: 'Device deleted successfully' });

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error deleting device' });

    }

  }

}

class CustomersController {

  async getCustomers(req, res) {

    try {

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

      res.json(customer.rows);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error fetching customer' });

    }

  }

  async addCustomer(req, res) {

    const { name, surname, email, tariff\_id } = req.body;

    if (!name || !surname || !surname || !tariff\_id) {

      return res.status(400).json({ error: 'Name, surname, email and tariff\_id are required' });

    }

    try {

      const newCustomer = await pool.query(

        'INSERT INTO customer (name, surname, email, tariff\_id) VALUES ($1, $2, $3, $4) RETURNING \*',

        [name, surname, email, tariff\_id]

      );

      res.status(201).json(newCustomer.rows[0]);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error adding customer' });

    }

  }

  async updateCustomer(req, res) {

    const { id } = req.params;

    const { name, surname, email, tariff\_id } = req.body;

    if (!name || !surname || !email || !tariff\_id) {

      return res.status(400).json({ error: 'Name, surname, email and tariff\_id are required' });

    }

    const customerId = parseInt(id, 10);

    if (isNaN(customerId)) {

      return res.status(400).json({ error: 'Invalid customer ID' });

    }

    try {

      const updatedCustomer = await pool.query(

        'UPDATE customer SET name = $1, surname = $2, email = $3, tariff\_id = $4 WHERE id = $5 RETURNING \*',

        [name, surname, email, tariff\_id, customerId]

      );

      if (updatedCustomer.rows.length > 0) {

        res.json(updatedCustomer.rows[0]);

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error updating customer' });

    }

  }

  async deleteCustomer(req, res) {

    const { id } = req.params;

    if (!id || isNaN(id)) {

      return res.status(400).json({ error: 'Invalid customer ID' });

    }

    const customerId = parseInt(id, 10);

    try {

      const result = await pool.query('DELETE FROM customer WHERE id = $1', [customerId]);

      if (result.rowCount > 0) {

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

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error deleting customer' });

    }

  }

}

class TariffsController {

  // Получить список всего транспорта

  async getTariffs(req, res) {

    try {

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

      res.json(tariffs.rows);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error fetching tariffs' });

    }

  }

  // Добавить новый транспорт

  async addTariff(req, res) {

    const { name, description, services\_id, device\_id, price } = req.body;

    if (!name || !description || !services\_id || !device\_id || !price) {

      return res.status(400).json({ error: 'Name, description, services\_id, device\_id and price are required' });

    }

    try {

      const newTariff = await pool.query(

        'INSERT INTO tariffs (name, description, services\_id, device\_id, price) VALUES ($1, $2, $3, $4, $5) RETURNING \*',

        [name, description, services\_id, device\_id, price]

      );

      res.status(201).json(newTariff.rows[0]);

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error adding tariff' });

    }

  }

  // Обновить существующий транспорт

  async updateTariff(req, res) {

    const { id } = req.params;

    const { name, description, services\_id, device\_id, price } = req.body;

    if (!name || !description || !services\_id || !device\_id || !price) {

      return res.status(400).json({ error: 'Name, description, services\_id, device\_id and price are required' });

    }

    const tariff\_id = parseInt(id, 10);

    if (isNaN(tariff\_id)) {

      return res.status(400).json({ error: 'Invalid tariff\_id ID' });

    }

    try {

      const updatedTariff = await pool.query(

        'UPDATE tariffs SET name = $1, description = $2, services\_id = $3, device\_id = $4, price = $5 WHERE id = $6 RETURNING \*',

        [name, description, services\_id, device\_id, price, tariff\_id]

      );

      if (updatedTariff.rows.length > 0) {

        res.json(updatedTariff.rows[0]);

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error updating tariff' });

    }

  }

  // Удалить транспорт

  async deleteTariff(req, res) {

    const { id } = req.params;

    if (!id || isNaN(id)) {

      return res.status(400).json({ error: 'Invalid tariff ID' });

    }

    const tariff\_id = parseInt(id, 10);

    try {

      const result = await pool.query('DELETE FROM tariffs WHERE id = $1', [tariff\_id]);

      if (result.rowCount > 0) {

        res.json({ message: 'Tariff deleted successfully' });

      } else {

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

      }

    } catch (error) {

      console.error(error);

      res.status(500).json({ error: 'Error deleting Tariff' });

    }

  }

}

const servicesController = new ServicesController();

const devicesController = new DevicesController();

const customersController = new CustomersController();

const taiffsController = new TariffsController();

app.get('/services', (req, res) => servicesController.getServices(req, res));

app.post('/service', (req, res) => servicesController.addService(req, res));

app.put('/service/:id', (req, res) => servicesController.updateService(req, res));

app.delete('/service/:id', (req, res) => servicesController.deleteService(req, res));

app.get('/devices', (req, res) => devicesController.getDevices(req, res));

app.post('/device', (req, res) => devicesController.addDevice(req, res));

app.put('/device/:id', (req, res) => devicesController.updateDevice(req, res));

app.delete('/device/:id', (req, res) => devicesController.deleteDevice(req, res));

app.get('/customers', (req, res) => customersController.getCustomers(req, res));

app.post('/customer', (req, res) => customersController.addCustomer(req, res));

app.put('/customer/:id', (req, res) => customersController.updateCustomer(req, res));

app.delete('/customer/:id', (req, res) => customersController.deleteCustomer(req, res));

app.get('/tariffs', (req, res) => taiffsController.getTariffs(req, res));

app.post('/tariff', (req, res) => taiffsController.addTariff(req, res));

app.put('/tariff/:id', (req, res) => taiffsController.updateTariff(req, res));

app.delete('/tariff/:id', (req, res) => taiffsController.deleteTariff(req, res));

const PORT = process.env.PORT || 8080;

app.listen(PORT, () => {

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

});

* 1. Код Web-приложения (CustomersList.vue)

<template>

<div>

<h1>Список покупателей</h1>

<div v-if="!showAddCustomerForm">

<div>

<label for="showCustomer">Показать покупателя в БД:</label>

<select id="showCustomer" v-model="selectedCustomer">

<option v-for="customer in customers" :value="customer" :key="customer.id">

{{ customer.name }}

</option>

</select>

</div>

<div>

<button @click="toggleAddCustomerForm">Добавить покупателя</button>

</div>

<div>

<button @click="toggleUpdateCustomerForm">Обновить покупателя</button>

</div>

<div>

<button @click="toggleDeleteCustomerForm">Удалить покупателя</button>

</div>

</div>

<!-- Форма для добавления нового покупателя -->

<div v-if="showAddCustomerForm">

<h2>Добавить нового покупателя</h2>

<p>Имя: <input v-model="newCustomer.name"></p>

<p>Фамилия: <input v-model="newCustomer.surname"></p>

<p>Электронная почта: <input v-model="newCustomer.email"></p>

<p>Id тарифа: <input v-model="newCustomer.tariff\_id"></p>

<button @click="addCustomer">Сохранить покупателя</button>

</div>

<!-- Форма для удаления покупателя -->

<div v-if="showDeleteCustomerForm">

<h2>Удалить покупателя</h2>

<p>Вы уверены, что хотите удалить покупателя {{ selectedCustomer.name }}?</p>

<button @click="confirmDeleteCustomer">Да, удалить</button>

</div>

<!-- Информация о выбранном покупателя -->

<div v-if="selectedCustomer && !showAddCustomerForm">

<div v-if="showUpdateCustomerForm">

<h2>Обновить информацию о покупателя</h2>

<p>Имя: <input v-model="selectedCustomer.name"></p>

<p>Фамилия: <input v-model="selectedCustomer.surname"></p>

<p>Электронная почта: <input v-model="selectedCustomer.email"></p>

<p>Id тарифа: <input v-model="selectedCustomer.tariff\_id"></p>

<button @click="saveUpdatedCustomer">Сохранить изменения</button>

</div>

<div v-else>

<h2>Информация о покупателе</h2>

<p>Имя: <input v-model="selectedCustomer.name"></p>

<p>Фамилия: <input v-model="selectedCustomer.surname"></p>

<p>Электронная почта: <input v-model="selectedCustomer.email"></p>

<p>Id тарифа: <input v-model="selectedCustomer.tariff\_id"></p>

</div>

</div>

</div>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

customers: [],

selectedCustomer: null,

newCustomer: {

name: '',

surname: '',

email: '',

tariff\_id: ''

},

showAddCustomerForm: false,

showUpdateCustomerForm: false,

showDeleteCustomerForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/customers');

},

methods: {

getListByReference: async function(refer) {

try {

const res = await axios.get(this.host + refer);

const data = res.data;

this.customers = data;

} catch (e) {

console.log('Ошибка: ', e);

}

},

toggleAddCustomerForm: function() {

this.showAddCustomerForm = !this.showAddCustomerForm;

if (!this.showAddCustomerForm) {

this.newCustomer = {

name: '',

surname: '',

email: '',

tariff\_id: ''

};

}

},

toggleUpdateCustomerForm: function() {

this.showUpdateCustomerForm = !this.showUpdateCustomerForm;

},

toggleDeleteCustomerForm: function() {

this.showDeleteCustomerForm = !this.showDeleteCustomerForm;

},

addCustomer: async function() {

if (!this.newCustomer.name || !this.newCustomer.surname || !this.newCustomer.email || !this.newCustomer.tariff\_id) {

console.log('Введите имя, фамилию, электронную почту и id тарифа');

return;

}

try {

const res = await axios.post(this.host + '/customer', this.newCustomer);

this.customers.push(res.data);

this.toggleAddCustomerForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

saveUpdatedCustomer: async function() {

const updatedCustomer = {

id: this.selectedCustomer.id,

name: this.selectedCustomer.name,

surname: this.selectedCustomer.surname,

email: this.selectedCustomer.email,

tariff\_id: this.selectedCustomer.tariff\_id

};

try {

const res = await axios.put(this.host + '/customer/' + updatedCustomer.id, updatedCustomer);

const index = this.customers.findIndex(customer => customer.id === res.data.id);

this.customers.splice(index, 1, res.data);

this.toggleUpdateCustomerForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

confirmDeleteCustomer: async function() {

try {

const res = await axios.delete(this.host + '/customer/' + this.selectedCustomer.id);

const index = this.customers.findIndex(customer => customer.id === res.data.id);

this.customers.splice(index, 1);

this.toggleDeleteCustomerForm();

} catch (e) {

console.log('Ошибка: ', e);

}

}

}

};

</script>

* 1. Код Web-приложения (DevicesList.vue)

<template>

<div>

<h1>Список устройств</h1>

<div v-if="!showAddDeviceForm">

<div>

<label for="showDevice">Показать устройство в БД:</label>

<select id="showDevice" v-model="selectedDevice">

<option v-for="device in devices" :value="device" :key="device.id">

{{ device.name }}

</option>

</select>

</div>

<div>

<button @click="toggleAddDeviceForm">Добавить устройство</button>

</div>

<div>

<button @click="toggleUpdateDeviceForm">Обновить устройство</button>

</div>

<div>

<button @click="toggleDeleteDeviceForm">Удалить устройство</button>

</div>

</div>

<!-- Форма для добавления нового устройства -->

<div v-if="showAddDeviceForm">

<h2>Добавить новое устройство</h2>

<p>Название: <input v-model="newDevice.name"></p>

<button @click="addDevice">Сохранить устройство</button>

</div>

<!-- Форма для удаления устройства -->

<div v-if="showDeleteDeviceForm">

<h2>Удалить устройство</h2>

<p>Вы уверены, что хотите удалить устройство {{ selectedDevice.name }}?</p>

<button @click="confirmDeleteDevice">Да, удалить</button>

</div>

<!-- Информация о выбранном устройстве -->

<div v-if="selectedDevice && !showAddDeviceForm">

<div v-if="showUpdateDeviceForm">

<h2>Обновить информацию об устройстве</h2>

<p>Название: <input v-model="selectedDevice.name"></p>

<button @click="saveUpdatedDevice">Сохранить изменения</button>

</div>

<div v-else>

<h2>Информация об устройстве</h2>

<p>Название: <input v-model="selectedDevice.name"></p>

</div>

</div>

</div>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

devices: [],

selectedDevice: null,

newDevice: {

name: ''

},

showAddDeviceForm: false,

showUpdateDeviceForm: false,

showDeleteDeviceForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/devices');

},

methods: {

getListByReference: async function(refer) {

try {

const res = await axios.get(this.host + refer);

const data = res.data;

this.devices = data;

} catch (e) {

console.log('Ошибка: ', e);

}

},

toggleAddDeviceForm: function() {

this.showAddDeviceForm = !this.showAddDeviceForm;

if (!this.showAddDeviceForm) {

this.newDevice = {

name: ''

};

}

},

toggleUpdateDeviceForm: function() {

this.showUpdateDeviceForm = !this.showUpdateDeviceForm;

},

toggleDeleteDeviceForm: function() {

this.showDeleteDeviceForm = !this.showDeleteDeviceForm;

},

addDevice: async function() {

if (!this.newDevice.name) {

console.log('Введите название и описание');

return;

}

try {

const res = await axios.post(this.host + '/device', this.newDevice);

this.devices.push(res.data);

this.toggleAddDeviceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

saveUpdatedDevice: async function() {

const updatedDevice = {

id: this.selectedDevice.id,

name: this.selectedDevice.name

};

try {

const res = await axios.put(this.host + '/device/' + updatedDevice.id, updatedDevice);

const index = this.devices.findIndex(device => device.id === res.data.id);

this.devices.splice(index, 1, res.data);

this.toggleUpdateDeviceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

confirmDeleteDevice: async function() {

try {

const res = await axios.delete(this.host + '/device/' + this.selectedDevice.id);

const index = this.devices.findIndex(device => device.id === res.data.id);

this.devices.splice(index, 1);

this.toggleDeleteDeviceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

}

}

};

</script>

* 1. Код Web-приложения (ServicesList.vue)

<template>

<div>

<h1>Список сервисов</h1>

<div v-if="!showAddServiceForm">

<div>

<label for="showService">Показать сервис в БД:</label>

<select id="showService" v-model="selectedService">

<option v-for="service in services" :value="service" :key="service.id">

{{ service.name }}

</option>

</select>

</div>

<div>

<button @click="toggleAddServiceForm">Добавить сервис</button>

</div>

<div>

<button @click="toggleUpdateServiceForm">Обновить сервис</button>

</div>

<div>

<button @click="toggleDeleteServiceForm">Удалить сервис</button>

</div>

</div>

<!-- Форма для добавления нового сервиса -->

<div v-if="showAddServiceForm">

<h2>Добавить новый сервис</h2>

<p>Название: <input v-model="newService.name"></p>

<p>Описание: <input v-model="newService.description"></p>

<button @click="addService">Сохранить сервис</button>

</div>

<!-- Форма для удаления сервиса -->

<div v-if="showDeleteServiceForm">

<h2>Удалить сервис</h2>

<p>Вы уверены, что хотите удалить сервис {{ selectedService.name }}?</p>

<button @click="confirmDeleteService">Да, удалить</button>

</div>

<!-- Информация о выбранном сервисе -->

<div v-if="selectedService && !showAddServiceForm">

<div v-if="showUpdateServiceForm">

<h2>Обновить информация о сервисе</h2>

<p>Название: <input v-model="selectedService.name"></p>

<p>Описание: <input v-model="selectedService.description"></p>

<button @click="saveUpdatedService">Сохранить изменения</button>

</div>

<div v-else>

<h2>Информация о сервисе</h2>

<p>Название: <input v-model="selectedService.name"></p>

<p>Описание: <input v-model="selectedService.description"></p>

</div>

</div>

</div>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

services: [],

selectedService: null,

newService: {

name: '',

description: ''

},

showAddServiceForm: false,

showUpdateServiceForm: false,

showDeleteServiceForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/services');

},

methods: {

getListByReference: async function(refer) {

try {

const res = await axios.get(this.host + refer);

const data = res.data;

this.services = data;

} catch (e) {

console.log('Ошибка: ', e);

}

},

toggleAddServiceForm: function() {

this.showAddServiceForm = !this.showAddServiceForm;

if (!this.showAddServiceForm) {

this.newService = {

name: ''

};

}

},

toggleUpdateServiceForm: function() {

this.showUpdateServiceForm = !this.showUpdateServiceForm;

},

toggleDeleteServiceForm: function() {

this.showDeleteServiceForm = !this.showDeleteServiceForm;

},

addService: async function() {

if (!this.newService.name) {

console.log('Введите название и описание');

return;

}

try {

const res = await axios.post(this.host + '/service', this.newService);

this.services.push(res.data);

this.toggleAddServiceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

saveUpdatedService: async function() {

const updatedService = {

id: this.selectedService.id,

name: this.selectedService.name,

description: this.selectedService.description

};

try {

const res = await axios.put(this.host + '/service/' + updatedService.id, updatedService);

const index = this.services.findIndex(service => service.id === res.data.id);

this.services.splice(index, 1, res.data);

this.toggleUpdateServiceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

confirmDeleteService: async function() {

try {

const res = await axios.delete(this.host + '/service/' + this.selectedService.id);

const index = this.services.findIndex(service => service.id === res.data.id);

this.services.splice(index, 1);

this.toggleDeleteServiceForm();

} catch (e) {

console.log('Ошибка: ', e);

}

}

}

};

</script>

* 1. Код Web-приложения (TariffsList.vue)

<template>

<div>

<h1>Список тарифов</h1>

<div v-if="!showAddTariffForm">

<div>

<label for="showTariff">Показать тариф в БД:</label>

<select id="showTariff" v-model="selectedTariff">

<option v-for="tariff in tariffs" :value="tariff" :key="tariff.id">

{{ tariff.name }}

</option>

</select>

</div>

<div>

<button @click="toggleAddTariffForm">Добавить тариф</button>

</div>

<div>

<button @click="toggleUpdateTariffForm">Обновить тариф</button>

</div>

<div>

<button @click="toggleDeleteTariffForm">Удалить тариф</button>

</div>

</div>

<!-- Форма для добавления нового тарифа -->

<div v-if="showAddTariffForm">

<h2>Добавить новый тариф</h2>

<p>Название: <input v-model="newTariff.name"></p>

<p>Описание: <input v-model="newTariff.description"></p>

<p>Id сервиса: <input v-model="newTariff.services\_id"></p>

<p>Id устройства: <input v-model="newTariff.device\_id"></p>

<p>Цена: <input v-model="newTariff.price"></p>

<button @click="addTariff">Сохранить тариф</button>

</div>

<!-- Форма для удаления тарифа -->

<div v-if="showDeleteTariffForm">

<h2>Удалить тариф</h2>

<p>Вы уверены, что хотите удалить тариф {{ selectedTariff.name }}?</p>

<button @click="confirmDeleteTariff">Да, удалить</button>

</div>

<!-- Информация о выбранном тарифе -->

<div v-if="selectedTariff && !showAddTariffForm">

<div v-if="showUpdateTariffForm">

<h2>Обновить информацию о тарифе</h2>

<p>Название: <input v-model="selectedTariff.name"></p>

<p>Описание: <input v-model="selectedTariff.description"></p>

<p>Id сервиса: <input v-model="selectedTariff.services\_id"></p>

<p>Id устройства: <input v-model="selectedTariff.device\_id"></p>

<p>Цена: <input v-model="selectedTariff.price"></p>

<button @click="saveUpdatedTariff">Сохранить изменения</button>

</div>

<div v-else>

<h2>Информация о тарифе</h2>

<p>Название: <input v-model="selectedTariff.name"></p>

<p>Описание: <input v-model="selectedTariff.description"></p>

<p>Id сервиса: <input v-model="selectedTariff.services\_id"></p>

<p>Id устройства: <input v-model="selectedTariff.device\_id"></p>

<p>Цена: <input v-model="selectedTariff.price"></p>

</div>

</div>

</div>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

tariffs: [],

selectedTariff: null,

newTariff: {

name: '',

description: '',

services\_id: '',

device\_id: '',

price: ''

},

showAddTariffForm: false,

showUpdateTariffForm: false,

showDeleteTariffForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/tariffs');

},

methods: {

getListByReference: async function(refer) {

try {

const res = await axios.get(this.host + refer);

const data = res.data;

this.tariffs = data;

} catch (e) {

console.log('Ошибка: ', e);

}

},

toggleAddTariffForm: function() {

this.showAddTariffForm = !this.showAddTariffForm;

if (!this.showAddTariffForm) {

this.newTariff = {

name: '',

description: '',

services\_id: '',

device\_id: '',

price: ''

};

}

},

toggleUpdateTariffForm: function() {

this.showUpdateTariffForm = !this.showUpdateTariffForm;

},

toggleDeleteTariffForm: function() {

this.showDeleteTariffForm = !this.showDeleteTariffForm;

},

addTariff: async function() {

if (!this.newTariff.name || !this.newTariff.description || !this.newTariff.services\_id || !this.newTariff.services\_id || !this.newTariff.price) {

console.log('Введите нзвание, описания, id сервсиа, id тарифа и цену');

return;

}

try {

const res = await axios.post(this.host + '/tariff', this.newTariff);

this.tariffs.push(res.data);

this.toggleAddTariffForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

saveUpdatedTariff: async function() {

const updatedTariff = {

id: this.selectedTariff.id,

name: this.selectedTariff.name,

description: this.selectedTariff.description,

services\_id: this.selectedTariff.services\_id,

device\_id: this.selectedTariff.device\_id,

price: this.selectedTariff.price

};

try {

const res = await axios.put(this.host + '/tariff/' + updatedTariff.id, updatedTariff);

const index = this.tariffs.findIndex(tariff => tariff.id === res.data.id);

this.tariffs.splice(index, 1, res.data);

this.toggleUpdateTariffForm();

} catch (e) {

console.log('Ошибка: ', e);

}

},

confirmDeleteTariff: async function() {

try {

const res = await axios.delete(this.host + '/tariff/' + this.selectedTariff.id);

const index = this.tariffs.findIndex(tariff => tariff.id === res.data.id);

this.tariffs.splice(index, 1);

this.toggleDeleteTariffForm();

} catch (e) {

console.log('Ошибка: ', e);

}

}

}

};

</script>

* 1. Результат отображения в браузере

|  |
| --- |
|  |
| 1. – Запуск backend/frontend |
|  |
| 1. – Отображение начальной (пустой) БД |
| Изображение выглядит как текст, снимок экрана, Шрифт, логотип  Автоматически созданное описание |
| 1. – Начальная страница |
|  |
| Изображение выглядит как текст, визитная карточка, снимок экрана, Шрифт  Автоматически созданное описание |
| 1. – Выбор пункта |
| Изображение выглядит как текст, снимок экрана, Шрифт, логотип  Автоматически созданное описание |
| 1. – Пример добавления нового покупателя |
| Изображение выглядит как текст, снимок экрана, Шрифт, логотип  Автоматически созданное описание |
| 1. – Вывод информации о покупателе |
| Изображение выглядит как текст, снимок экрана, Шрифт, логотип  Автоматически созданное описание |
| 1. – Обновление информации о покупателе   Изображение выглядит как текст, снимок экрана, Шрифт, дизайн  Автоматически созданное описание   1. – Удаление информации о покупателе |

* 1. Схема Сетей Петри

|  |
| --- |
| Изображение выглядит как круг, луна, снимок экрана, черно-белый  Автоматически созданное описание |
| 1. – Сети Петри для «Покупатели» |
| Изображение выглядит как круг, луна, снимок экрана, черно-белый  Автоматически созданное описание |
| 1. – Сети Петри для «Тарифы»   Изображение выглядит как круг, луна, снимок экрана, черно-белый  Автоматически созданное описание   1. – Сети Петри для «Сервисы»   Изображение выглядит как круг, луна, снимок экрана, черно-белый  Автоматически созданное описание   1. – Сети Петри для «Устройства» |

1. вывод по работе

В результате проведенной работы было создано масштабируемое одностраничное web-приложение на базе фреймворка Vue.js (по шаблону Vite.js, подгружаемому командой npm init vue@latest), отражающее требуемый функционал и тематику согласно варианту индивидуального задания, а именно была разработана текстовая база данных и веб-оболочка на тему «Библиотека» с соответствующими запросами.