|  |  |
| --- | --- |
| **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  Волочинский И.О.  Ефремов Д.Ю.  Проверил:  Доцент кафедры УиЗИ, к.т.н., с.н.с  Сафронов А.И. |
| Москва 2024 | |

Оглавление

[1. Цель работы 3](#_gjdgxs)

[2. ФОРМУЛИРОВАКА ЗАДАЧИ 4](#_1fob9te)

[3. Описание предметной области 5](#_3znysh7)

[3.1. Формулировка задачи 5](#_2et92p0)

[3.2. ER-диаграмма данных 5](#_3dy6vkm)

[3.3. Таблица соответствия переменных 6](#_1t3h5sf)

[3.4. Архитектура системы (связка: frontend, backend, database) 7](#_4d34og8)

[4. Web-страница 8](#_2s8eyo1)

[4.1. Инструкция по развёртке системы (подгрузка запрашиваемых (required) модулей, взаимная увязка технологий). 8](#_17dp8vu)

[4.2. Код web-приложения (server.js) 8](#_3rdcrjn)

[4.3. Код Web-приложения (ClientList.vue) 13](#_26in1rg)

[4.4. Код Web-приложения (TransportList.vue) 17](#_lnxbz9)

[4.5. Результат отображения в браузере 21](#_35nkun2)

[4.6. Схема Сетей Петри 24](#_1ksv4uv)

[5. вывод по работе 24](#_44sinio)

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**

Разработать текстовую базу данных и веб-оболочку на тему «Транспортная компания». База данных создаётся для обслуживания работников и клиентов компании. Конечный вариант базы данных содержит в себе информацию о доступных видах транспорта для перевозки, цены на перевозки, а также данные о потенциальных покупателях. Готовые запросы:

* Выдавать список маршрутов, обслуживаемых компанией и цену на них;
* Выдавать список транспорта, занятого на данном маршруте;
* Выдавать список вариантов проезда по данному маршруту (морской и т. д.);
* Рассчитывать стоимость услуг, оказанных по перевозке данного груза. Постоянным клиентам предоставляется скидка;
* Показывать грузооборот по данному маршруту.
  1. **ER-диаграмма данных**

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

* 1. **Таблица соответствия переменных**

Таблица 1, 2, 3, 4, 5 – Сущности Базы Данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Работник | Идентификатор сотрудника | id | Целое число | Первичный ключ |
| Имя сотрудника | name | Строка (255) | Обязательное поле |
| Фамилия сотрудника | lastname | Строка (255) | Обязательное поле, |
| Отчество сотрудника | surname | Строка (255) | Необязательное поле |
| Тип сотрудника (должность) | type | Строка (255) | Обязательное поле, |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Дисциплина | Идентификатор | id | Целое число | Первичный ключ |
| Название | name | Строка (255) | Обязательное поле |
| Идентификатор  профессора | professor\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) работник |
| Идентификатор помощника | hellper\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) работник |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Лекция | Идентификатор | id | Целое число | Первичный ключ |
| Время начала | name | Время | Обязательное поле |
| Идентификатор  дисциплины | discipline\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) дисциплина |
| Время конца | hellper\_id | Время | Обязательное поле |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Научная исследовательская работа | Идентификатор | id | Целое число | Первичный ключ |
| Идентификатор  работника | implementer\_id | Целое число | Обязательное поле,  Внешний ключ к отношению (сущности) работник |
| Идентификатор  дисциплины | discipline\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) дисциплина |
| Тема работы | theme | Строка (255) | Обязательное поле |

* 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: 'jkoimni',

password:'qwerty',

host: 'localhost',

database: 'cathedra',

port: 5432

});

const app = express();

app.use(express.*json*());

app.use(cors());

class WorkerController {

// Получить список всех клиентов

async getWorker(*req*, *res*) {

try {

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

*res*.json(worker.rows);

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Добавить нового клиента

async addWorker(*req*, *res*) {

const { name, surname, lastname, type } = *req*.body;

if (!name || !surname || !lastname || !type) {

return *res*.status(400).json({ error: 'Name, address, and phone are required' });

}

try {

const newWorker = await pool.query(

'INSERT INTO worker (name, surname, lastname, type) VALUES (*$1*, *$2*, *$3*, *$4*) RETURNING \*',

[name, surname, lastname, type]

);

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

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Обновить существующего клиента

async updateWorker(*req*, *res*) {

const { id } = *req*.params;

const { name, surname, lastname, type } = *req*.body;

if (!name || !surname || !lastname || !type) {

return *res*.status(400).json({ error: 'Name, lastname, surname, and type are required' });

}

const workerId = parseInt(id, 10);

if (isNaN(workerId)) {

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

}

try {

const updatedWorker = await pool.query(

'UPDATE worker SET name = *$1*, surname = *$2*, lastname = *$3*, type = *$4* WHERE id = *$5* RETURNING \*',

[name, surname, lastname, type, workerId]

);

if (updatedWorker.rows.length > 0) {

*res*.json(updatedWorker.rows[0]);

} else {

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

}

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Удалить клиента

async deleteWorker(*req*, *res*) {

const { id } = *req*.params;

if (!id || isNaN(id)) {

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

}

const workerId = parseInt(id, 10);

*console*.log(`Deleting Worker with ID: ${workerId}`); // Отладочное сообщение

try {

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

if (result.rowCount > 0) {

*console*.log(`Worker with ID ${workerId} deleted successfully`); // Отладочное сообщение

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

} else {

*console*.log(`Worker with ID ${workerId} not found`); // Отладочное сообщение

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

}

} catch (*error*) {

*console*.error(*error*);

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

}

}

}

class DisciplineController {

// Получить список всего транспорта

async getDiscipline(*req*, *res*) {

try {

const discipline = await pool.query('SELECT d.id, d.name, w.lastname FROM discipline AS d INNER JOIN worker w ON w.id = d.professor\_id');

*res*.json(discipline.rows);

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Добавить новый транспорт

async addDiscipline(*req*, *res*) {

const { name, lastname } = *req*.body;

if (!name || !lastname) {

return *res*.status(400).json({ error: 'Type and cost are required' });

}

try {

let workerId = await pool.query(

'SELECT id FROM worker WHERE lastname = (*$1*) LIMIT 1',

[lastname]

);

*console*.log(workerId.rows[0].id)

const newDiscipline = await pool.query(

'INSERT INTO discipline (name, professor\_id) VALUES (*$1*, *$2*) RETURNING \*',

[name, workerId.rows[0].id]

);

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

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Обновить существующий транспорт

async updateDiscipline(*req*, *res*) {

const { id } = *req*.params;

const { name, lastname } = *req*.body;

if (!name || !lastname) {

return *res*.status(400).json({ error: 'Type and cost are required' });

}

const disciplineId = parseInt(id, 10);

if (isNaN(disciplineId)) {

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

}

try {

let workerId = await pool.query(

'SELECT id FROM worker WHERE lastname = (*$1*) LIMIT 1',

[lastname]

);

*console*.log(workerId.rows[0].id)

const updatedDiscipline = await pool.query(

'UPDATE discipline SET name = *$1*, professor\_id = *$3* WHERE id = *$2* RETURNING \*',

[name, disciplineId, workerId.rows[0].id]

);

if (updatedDiscipline.rows.length > 0) {

*res*.json(updatedDiscipline.rows[0]);

} else {

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

}

} catch (*error*) {

*console*.error(*error*);

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

}

}

// Удалить транспорт

async deleteDiscipline(*req*, *res*) {

const { id } = *req*.params;

if (!id || isNaN(id)) {

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

}

const disciplineId = parseInt(id, 10);

*console*.log(`Deleting Discipline with ID: ${disciplineId}`); // Отладочное сообщение

try {

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

if (result.rowCount > 0) {

*console*.log(`Discipline with ID ${disciplineId} deleted successfully`); // Отладочное сообщение

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

} else {

*console*.log(`Discipline with ID ${disciplineId} not found`); // Отладочное сообщение

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

}

} catch (*error*) {

*console*.error(*error*);

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

}

}

}

const workerController = new WorkerController();

const disciplineController = new DisciplineController();

// Маршруты для методов клиентов

app.get('/worker', (*req*, *res*) => workerController.getWorker(*req*, *res*));

app.post('/worker', (*req*, *res*) => workerController.addWorker(*req*, *res*));

app.put('/worker/:id', (*req*, *res*) => workerController.updateWorker(*req*, *res*));

app.delete('/worker/:id', (*req*, *res*) => workerController.deleteWorker(*req*, *res*));

// Маршруты для методов транспорта

app.get('/discipline', (*req*, *res*) => disciplineController.getDiscipline(*req*, *res*));

app.post('/discipline', (*req*, *res*) => disciplineController.addDiscipline(*req*, *res*));

app.put('/discipline/:id', (*req*, *res*) => disciplineController.updateDiscipline(*req*, *res*));

app.delete('/discipline/:id', (*req*, *res*) => disciplineController.deleteDiscipline(*req*, *res*));

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

app.listen(PORT, () => {

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

});

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

<template>

<div>

<h1>**Список сотрудников**</h1>

<div>

<label for**="showWorker"**>**Показать сотрудников в БД:**</label>

<select id**="showWorker"** v-model**="**selectedWorker**"** @change**="**resetForms**"**>

<option v-for**="**worker in lstWorker**"** :value**="**worker**"** :key**="**worker.id**"**>

**{{ worker.name }}**

</option>

</select>

</div>

<div v-if**="**selectedWorker && !showAddWorkerForm && !showUpdateWorkerForm && !showDeleteWorkerForm**"**>

<h2>**Информация о сотруднике**</h2>

<p>**Имя: {{ selectedWorker.name }}**</p>

<p>**Фамилия: {{ selectedWorker.surname }}**</p>

<p>**Отчество: {{ selectedWorker.lastname }}**</p>

<p>**Должность: {{ selectedWorker.type }}**</p>

</div>

<div>

<button @click**="**toggleAddWorkerForm**"**>**Добавить сотрудника**</button>

</div>

<div>

<button @click**="**toggleUpdateWorkerForm**"**>**Обновить сотрудника**</button>

<select v-if**="**showUpdateWorkerForm**"** v-model**="**selectedWorkerForUpdate**"**>

<option v-for**="**worker in lstWorker**"** :value**="**worker**"** :key**="**worker.id**"**>

**{{ worker.name }}**

</option>

</select>

</div>

<div>

<button @click**="**toggleDeleteWorkerForm**"**>**Удалить сотрудника**</button>

<select v-if**="**showDeleteWorkerForm**"** v-model**="**selectedWorkerForDelete**"**>

<option v-for**="**worker in lstWorker**"** :value**="**worker**"** :key**="**worker.id**"**>

**{{ worker.name }}**

</option>

</select>

</div>

<div v-if**="**showAddWorkerForm**"**>

<h2>**Добавить нового сотрудника**</h2>

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

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

<p>**Отчество:** <input v-model**="**newWorker.lastname**"**></p>

<p>**Должность:** <input v-model**="**newWorker.type**"**></p>

<button @click**="**addWorker**"**>**Сохранить клиента**</button>

</div>

<div v-if**="**showUpdateWorkerForm && selectedWorkerForUpdate**"**>

<h2>**Обновить информацию о сотруднике**</h2>

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

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

<p>**Отчество:** <input v-model**="**selectedWorkerForUpdate.lastname**"**></p>

<p>**Должность:** <input v-model**="**selectedWorkerForUpdate.type**"**></p>

<button @click**="**saveUpdatedWorker**"**>**Сохранить изменения**</button>

</div>

<div v-if**="**showDeleteWorkerForm && selectedWorkerForDelete**"**>

<h2>**Удалить сотрудника**</h2>

<p>**Вы уверены, что хотите удалить клиента {{ selectedWorkerForDelete.name }}?**</p>

<button @click**="**confirmDeleteWorker**"**>**Да, удалить**</button>

</div>

</div>

</template>

<script>

import *axios* from 'axios';

export default {

data() {

return {

lstWorker: [],

selectedWorker: null,

selectedWorkerForUpdate: null,

selectedWorkerForDelete: null,

newWorker: {

name: '',

surname: '',

lastname: '',

type: ''

},

showAddWorkerForm: false,

showUpdateWorkerForm: false,

showDeleteWorkerForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/worker');

},

methods: {

getListByReference: async function(*refer*) {

try {

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

this.lstWorker = res.data;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

toggleAddWorkerForm: function() {

this.resetForms();

this.showAddWorkerForm = true;

},

toggleUpdateWorkerForm: function() {

this.resetForms();

this.showUpdateWorkerForm = true;

},

toggleDeleteWorkerForm: function() {

this.resetForms();

this.showDeleteWorkerForm = true;

},

resetForms: function() {

this.showAddWorkerForm = false;

this.showUpdateWorkerForm = false;

this.showDeleteWorkerForm = false;

},

addWorker: async function() {

if (!this.newWorker.name || !this.newWorker.surname || !this.newWorker.lastname) {

*console*.log('Введите имя, адрес и телефон');

return;

}

try {

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

this.lstWorker.push(res.data);

this.showAddWorkerForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

saveUpdatedWorker: async function() {

if (!this.selectedWorkerForUpdate) {

*console*.log('Выберите клиента для обновления');

return;

}

try {

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

const index = this.lstWorker.findIndex(*worker* => *worker*.id === res.data.id);

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

this.showUpdateWorkerForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

confirmDeleteWorker: async function() {

if (!this.selectedWorkerForDelete) {

*console*.log('Выберите клиента для удаления');

return;

}

try {

await axios.delete(this.host + '/worker/' + this.selectedWorkerForDelete.id);

this.lstWorker = this.lstWorker.filter(*worker* => *worker*.id !== this.selectedWorkerForDelete.id);

this.showDeleteWorkerForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

}

}

};

</script>

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

<template>

<div>

<h1>**Список дисциплин**</h1>

<div>

<label for**="showDiscipline"**>**Показать дисциплины в БД:**</label>

<select id**="showDiscipline"** v-model**="**selectedDiscipline**"** @change**="**resetForms**"**>

<option v-for**="**discipline in lstDiscipline**"** :value**="**discipline**"** :key**="**discipline.id**"**>

**{{ discipline.name }}**

</option>

</select>

</div>

<div v-if**="**selectedDiscipline && !showAddDisciplineForm && !showUpdateDisciplineForm && !showDeleteDisciplineForm**"**>

<h2>**Информация о дисциплине**</h2>

<p>**Имя: {{ selectedDiscipline.name }}**</p>

<p>**Фамилия преподавателя: {{ selectedDiscipline.lastname }}**</p>

</div>

<div>

<button @click**="**toggleAddDisciplineForm**"**>**Добавить дисциплину**</button>

</div>

<div>

<button @click**="**toggleUpdateDisciplineForm**"**>**Обновить дисциплину**</button>

<select v-if**="**showUpdateDisciplineForm**"** v-model**="**selectedDisciplineForUpdate**"**>

<option v-for**="**discipline in lstDiscipline**"** :value**="**discipline**"** :key**="**discipline.id**"**>

**{{ discipline.name }}**

</option>

</select>

</div>

<div>

<button @click**="**toggleDeleteWorkerForm**"**>**Удалить дисциплину**</button>

<select v-if**="**showDeleteDisciplineForm**"** v-model**="**selectedDisciplineForDelete**"**>

<option v-for**="**discipline in lstDiscipline**"** :value**="**discipline**"** :key**="**discipline.id**"**>

**{{ discipline.name }}**

</option>

</select>

</div>

<div v-if**="**showAddDisciplineForm**"**>

<h2>**Добавить новую дисциплину**</h2>

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

<p>**Фамилия преподавателя:** <input v-model**="**newDiscipline.lastname**"**></p>

<button @click**="**addDiscipline**"**>**Сохранить дисциплину**</button>

</div>

<div v-if**="**showUpdateDisciplineForm && selectedDisciplineForUpdate**"**>

<h2>**Обновить информацию о дисциплине**</h2>

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

<p>**Фамилия преподавателя:** <input v-model**="**selectedDisciplineForUpdate.lastname**"**></p>

<button @click**="**saveUpdatedDiscipline**"**>**Сохранить изменения**</button>

</div>

<div v-if**="**showDeleteDisciplineForm && selectedDisciplineForDelete**"**>

<h2>**Удалить дисциплину**</h2>

<p>**Вы уверены, что хотите удалить дисциплину {{ selectedDisciplineForDelete.name }}?**</p>

<button @click**="**confirmDeleteDiscipline**"**>**Да, удалить**</button>

</div>

</div>

</template>

<script>

import *axios* from 'axios';

export default {

data() {

return {

lstDiscipline: [],

selectedDiscipline: null,

selectedDisciplineForUpdate: null,

selectedDisciplineForDelete: null,

newDiscipline: {

name: '',

lastname: ''

},

showAddDisciplineForm: false,

showUpdateDisciplineForm: false,

showDeleteDisciplineForm: false,

host: 'http://localhost:8080'

};

},

mounted() {

this.getListByReference('/discipline');

},

methods: {

getListByReference: async function(*refer*) {

try {

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

this.lstDiscipline = res.data;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

toggleAddDisciplineForm: function() {

this.resetForms();

this.showAddDisciplineForm = true;

},

toggleUpdateDisciplineForm: function() {

this.resetForms();

this.showUpdateDisciplineForm = true;

},

toggleDeleteWorkerForm: function() {

this.resetForms();

this.showDeleteDisciplineForm = true;

},

resetForms: function() {

this.showAddDisciplineForm = false;

this.showUpdateDisciplineForm = false;

this.showDeleteDisciplineForm = false;

},

addDiscipline: async function() {

if (!this.newDiscipline.name || !this.newDiscipline.lastname) {

*console*.log('Введите имя, адрес и телефон');

return;

}

try {

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

this.lstDiscipline.push(res.data);

this.showAddWorkerForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

saveUpdatedDiscipline: async function() {

if (!this.selectedDisciplineForUpdate) {

*console*.log('Выберите клиента для обновления');

return;

}

try {

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

const index = this.lstDiscipline.findIndex(*discipline* => *discipline*.id === res.data.id);

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

this.showUpdateDisciplineForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

},

confirmDeleteDiscipline: async function() {

if (!this.selectedDisciplineForDelete) {

*console*.log('Выберите клиента для удаления');

return;

}

try {

await axios.delete(this.host + '/discipline/' + this.selectedDisciplineForDelete.id);

this.lstWorker = this.lstDiscipline.filter(*discipline* => *discipline*.id !== this.selectedDisciplineForDelete.id);

this.showDeleteWorkerForm = false;

} catch (*e*) {

*console*.log('Ошибка: ', *e*);

}

}

}

};

</script>

* 1. **Результат отображения в браузере**

|  |
| --- |
|  |
| 1. – Запуск backend/frontend |
|  |
| 1. – Отображение начальной БД |
|  |
| 1. – Начальная страница |
|  |
|  |
| 1. – Результат добавления сотрудника в БД |
|  |
| 1. – Результат удаления сотрудника из БД |
|  |
| 1. – Результат добавления дисциплины в БД |
|  |
| 1. – Результат удаления дисциплины из БД |

* 1. **Схема Сетей Петри**

|  |
| --- |
|  |
| 1. – Сети Петри для «Сотрудников» |
|  |
| 1. – Сети Петри для «Дисциплин» |

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.