**Задачи курса «Web-программирование»**

|  |  |
| --- | --- |
| **Задание 8** | Создать масштабируемое одностраничное веб-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 ([*https://elibrary.ru/item.asp?id=46268632*](https://elibrary.ru/item.asp?id=46268632) стр. 81-93).  **Требование по функционалу**:  **Хранение данных:**  - база данных *PostgreSQL*.  **Backend:**  Вариант 1: на базе фреймворка *Express.js* (см. Приложение 1).  Вариант 2: предварительно согласованная технология.  **Frontend:**  - на базе фреймворка *Vue.js* (при подключении шаблона *Vite.js*). Пример шаблона для встраивания в компонент *App.vue* в Приложении 2. Пример настройки экземпляра по аналогии с *LSPWA* в Приложении 3. Приложение 3 также содержит пример асинхронных методов получения данных от *backend*. Приложение 4 содержит пример аналогичного получения данных в обход асинхронных методов. Приложение 5 содержит пример добавления новой сущности в отношение базы данных.  **Реализация методологии *CRUD*:**  - 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.  **Дружественный к пользователю графический интерфейс:**  - отдельные окна (шаблоны / *templates*) для решения отдельных задач по тематике – не решать все задачи в одном окне (шаблоне / *template*);  - 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.  **Рекомендация:** обучающимся, прошедшим и / или проходящим курсы «Информационное обеспечение систем управления», «Системы управления базами данных», «Основы построения защищённых баз данных», взять в качестве тематики базы данных выданную в рамках этих курсов.  **Подсказка:** в методическом пособии рассмотрен пример для «Отдела кадров» в простейшем проявлении – реализовать аналог под выбранную тематику.  *Базовая структура отчёта по работе:*  1. Цель работы.  2. Формулировка задачи.  3. Описание предметной области и сформулированных к ней задач.  4. *ER*-диаграмма данных.  5. Таблица соответствия переменных.  6. Архитектура системы (связка: *frontend*, *backend*, *database*).  7. Инструкция по развёртке системы (подгрузка запрашиваемых (*required*) модулей, взаимная увязка технологий).  8. Содержательная часть (код *web*-приложения + отображение в браузере + Сеть Петри).  9. Вывод. |

**Приложение 1**

|  |
| --- |
| import express from 'express'  import cors from 'cors'  import pg from 'pg'  let pool = new pg.Pool({  user: 'postgres',  host: 'localhost',  database: 'epDatabase',  password: '[пароль к базам данных]',  port: 5432});  let router = new express.Router();  class UserController  {  async createWorker(req, res) {  let { id, name, duty } = req.body  let newWorker = await pool.query(`INSERT INTO workers (id,  name, duty) values ($1, $2, $3)  RETURNING \*`, [ id, name, duty ])  res.json(newWorker.rows[0])  }  async getWorkers(req, res){  let workers = await pool.query('SELECT \* FROM workers')  res.json(workers.rows)  }  async getDuties(req, res){  let duties = await pool.query('SELECT \* FROM duties')  res.json(duties.rows)  }  }  let userController = new UserController()  router.post('/workers', userController.createWorker)  router.get('/workers', userController.getWorkers)  router.get('/duties', userController.getDuties)  router.put('/workers', userController.createWorker)  let PORT = process.env.PORT || 8080;  let app = express();  app.use(express.json())  app.use(cors())  //В таком написании: http://localhost:8080/workers  app.use('', router)  //В таком написании: http://localhost:8080/res/workers  //app.use('/res', router)  app.listen(PORT, () => console.log(`Сервер запущен на порте  ${PORT}`)) |

**Приложение 2**

|  |
| --- |
| <template>  <table>  <tr cols="2">  <td>  <p align="center"><b>Отдел кадров</b></p>  <p>Список сотрудников:</p>  <select style="width: 500px"  @blur="rst"  size="5"  v-model="selected">  <option v-for="w in lstWorkers"  :value="w"  :key="w.id">  {{ w.name }}  </option>  </select><br>  <br>  Ф.И.О.<br>  <input v-if="selected != {}" v-model="selected.name">  <input v-else v-model="name">  <br>  Должность  <br>  <select class="duties" v-if="selected != {}"  v-model="lstDuties[selected.duty - 1]">  <option>Выберите...</option>  <option v-for="d in lstDuties" :value="d"  :key="d.id">  {{ d.name }}  </option>  </select>  <select class="duties" v-else v-model="duty">  <option disabled  value="Выберите...">Выберите...</option>  <option v-for="d in lstDuties"  :value="d"  :key="d.id">  {{ d.name }}  </option>  </select>  <br/>  <button :disabled="selected != {}"  @click="addWorker">Ввести</button>  <br/>  </td>  </tr>  </table>  </template> |

**Приложение 3**

|  |
| --- |
| <script>  export default{  data(){  return {  lstWorkers: [],  lstDuties: [],  lstTmp: [],  selected: {},  duty: 'Выберите...',  name: '',  host: 'http://localhost:8080'  }  },  mounted: function() {  this.getListByReference('/duties');  this.getListByReference('/workers');  },  methods: {  getListByReference: async function(refer){  var fetchData = {  headers: {  'Content-Type': 'application/json',  }  }  try {  const res = await fetch(this.host + refer,  fetchData);  const data = await res.json();  for (let i = 0; i < data.length; i++){  switch (refer){  case '/workers':  this.lstWorkers.  push(Object.assign({},data[i]));  break;  case '/duties':  this.lstDuties.  push(Object.assign({},data[i]));  break;  }  }  }  catch (e){  console.log('Ошибка: ', e);  }  }  }  }  </script> |

**Приложение 4**

|  |
| --- |
| <script>  export default {  data() {  return {  lstWorkers: [],  lstDuties: [],  selectedWorker: null,  selectedDuty: 'Выберите...',  newName: '',  newDuty: '',  host: 'http://localhost:8080',  flgOnce: true  }  },  mounted: function() {  var fetchParams = {  headers: {  'Content-Type': 'application/json',  }  }  fetch(this.host + '/workers', fetchParams)  .then((response) => response.json())  .then(data => this.lstWorkers = data)  .catch(error => console.log("Ошибка: ", error)),  fetch(this.host + '/duties', fetchParams)  .then((response) => response.json())  .then(data => this.lstDuties = data)  .catch(error => console.log("Ошибка: ", error))  },  updated: function() {  if (this.flg &&  this.lstWorkers.length > 0 &&  this.lstDuties.length > 0){  this.ToLinkDuties();  this.flg = false;  }  },    methods: {  ToLinkDuties: function() {  for (let i = 0; i < this.lstWorkers.length; i++)  for (let j = 0; j < this.lstDuties.length; j++)  if (this.lstWorkers[i].duty ===  this.lstDuties[j].id)  this.lstWorkers[i].duty =  Object.assign({}, this.lstDuties[j]);  }  }  }  </script> |

**Приложение 5**

|  |
| --- |
| <script>  export default {  … … … … …  methods: {  ToReset: function() { this.selectedWorker = null; },    //Добавление нового сотрудника  addWorker: function() {  var fetchParams = {  method: 'POST',  headers: {  'Content-Type': 'application/json'  },  body: jsonData  }  var wrkr = new Object();  const newId = this.lstWorkers.length + 1  wrkr.id = newId;  wrkr.name = this.name;  const dutiesElement =  document.querySelectorAll('.duties option')  for (let i = 1; i < dutiesElement.length; i++)  if (this.duty.name ===  this.lstDuties[i - 1].name)  wrkr.duty = this.lstDuties[i - 1].id;  this.newName = '';  this.selectedDuty = 'Выберите...';  this.ToReset();  const data = wrkr;  let count = 0;  let jsonData = JSON.stringify(data);    fetch(this.host + '/workers', fetchParams)  .catch(error => {  console.log("Ошибка: ", error)  return count++  });  if (count === 0)  this.lstWorkers.push(wrkr);  }  }  }  </script> |