**Государственное образовательное учреждение**

**высшего образования Московской области**

**«Государственный гуманитарно-технологический университет»**

**Ликино-Дулевский политехнический колледж – филиал ГГТУ**

Отчет по вариативной работе по дисциплине МДК.01.01

«Договорная деятельность организации»

Выполнил:

студент 3 курса

группы ИСП.23.1А

Боев Владислав

проверил:

преподаватель

Кузьмина Елена Евгеньевна

2025 г.

**Введение**

Предметная область: *Договорная деятельность организации.*

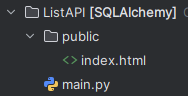
Минимальный список характеристик:

* шифр работы,
* название,
* трудоемкость,
* ФИО сотрудника,
* должность,
* дата выдачи поручения на работу,
* дата окончания работы.

Рассмотренного в прошлых темах материала достаточно для создания примитивного приложения. В этой теме попробуем реализовать простейшее приложение Web API в стиле REST. Архитектура REST предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером, где каждый тип запроса отвечает за определенное действие:

* **GET** (получение данных)
* **POST** (добавление данных)
* **PUT** (изменение данных)
* **DELETE** (удаление данных)

Для каждого из этих типов запросов класс **FastAPI** предоставляет соответствующие методы. Рассмотрим, как мы можем реализовать с помощью этих методов простейший API. Пусть у нас будет следующий проект:



### **Создание сервера**

В файле **main.py** определим следующий код:

import uuid  
from fastapi import FastAPI, Body, HTTPException, status  
from fastapi.responses import JSONResponse, FileResponse  
from fastapi.middleware.cors import CORSMiddleware  
from pydantic import BaseModel  
from typing import List, Optional  
from datetime import date  
  
app = FastAPI()  
  
# Добавьте CORS middleware  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=["\*"],  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
# Счетчик для генерации кодов  
contract\_counter = 1  
  
class ContractModel(BaseModel):  
 id: Optional[str] = None  
 code: str # Добавлено поле для шифра документа  
 name: str  
 labor: int  
 employeeName: str  
 post: str  
 issueDate: date  
 finalDate: date  
  
class Contract:  
 def \_\_init\_\_(self, name, labor, employeeName, post, issueDate, finalDate):  
 global contract\_counter  
 self.name = name  
 self.labor = labor  
 self.employeeName = employeeName  
 self.post = post  
 self.issueDate = issueDate  
 self.finalDate = finalDate  
 self.id = str(uuid.uuid4())  
 # Генерация шифра документа в формате К-001  
 self.code = f"К-{contract\_counter:03d}"  
 contract\_counter += 1  
  
# База данных  
acts = [  
 Contract("Подготовка договора", 8, "Иванов И.И.", "Юрист", date(2023, 1, 1), date(2023, 1, 3)),  
 Contract("Проведение переговоров", 16, "Петров А.А.", "Менеджер отдела", date(2023, 1, 5), date(2023, 1, 10)),  
 Contract("Анализ договорных рисков", 24, "Сидоров О.О.", "Руководитель", date(2023, 1, 12), date(2023, 1, 18))  
]  
  
def find\_contract(id):  
 for contract in acts:  
 if contract.id == id:  
 return contract  
 return None  
  
def find\_contract\_by\_code(code):  
 for contract in acts:  
 if contract.code == code:  
 return contract  
 return None  
  
@app.get("/")  
async def main():  
 return FileResponse("public/index.html")  
  
@app.get("/api/contracts", response\_model=List[ContractModel])  
def get\_contracts():  
 return acts  
  
@app.get("/api/contracts/{code}", response\_model=ContractModel)  
def get\_contract(code: str):  
 contract = find\_contract\_by\_code(code)  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 return contract  
  
@app.post("/api/contracts", response\_model=ContractModel)  
def create\_contract(data=Body()):  
 contract = Contract(data["name"], data["labor"], data["employeeName"],  
 data["post"], data["issueDate"], data["finalDate"])  
 acts.append(contract)  
 return contract  
  
@app.put("/api/contracts", response\_model=ContractModel)  
def update\_contract(data=Body()):  
 contract = find\_contract\_by\_code(data["code"])  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 contract.name = data["name"]  
 contract.labor = data["labor"]  
 contract.employeeName = data["employeeName"]  
 contract.post = data["post"]  
 contract.issueDate = data["issueDate"]  
 contract.finalDate = data["finalDate"]  
 return contract  
  
@app.delete("/api/contracts/{code}")  
def delete\_contract(code: str):  
 contract = find\_contract\_by\_code(code)  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 acts.remove(contract)  
 return {"message": "Контракт удален", "code": code}

Разберем в общих чертах этот код. Прежде всего для представления данных, с которыми мы будем работать, определяем класс **Contract**.

class Contract:  
 def \_\_init\_\_(self, name, labor, employeeName, post, issueDate, finalDate):  
 global contract\_counter  
 self.name = name  
 self.labor = labor  
 self.employeeName = employeeName  
 self.post = post  
 self.issueDate = issueDate  
 self.finalDate = finalDate  
 self.id = str(uuid.uuid4())  
 # Генерация шифра документа в формате К-001  
 self.code = f"К-{contract\_counter:03d}"  
 contract\_counter += 1

Этот класс содержит 7 атрибутов. Шесть атрибутов – name, labor, employeeName, post, issueDate, finalDate будут представлять название, ФИО сотрудника, должность, трудоемкость, дата начала и окончания договора и будут устанавливаться через конструктор. А седьмой атрибут - code будет служить для уникальной идентификации данного объекта и будет хранить значение guid. Для генерации guid применяется функция uuid.uuid4() из пакета uuid.

Для хранения данных в приложении определим список acts, который будет выполнять роль условной базы данных и будет хранить объекты Contract.

acts = [  
 Contract("Подготовка договора", 8, "Иванов И.И.", "Юрист", date(2023, 1, 1), date(2023, 1, 3)),  
 Contract("Проведение переговоров", 16, "Петров А.А.", "Менеджер отдела", date(2023, 1, 5), date(2023, 1, 10)),  
 Contract("Анализ договорных рисков", 24, "Сидоров О.О.", "Руководитель", date(2023, 1, 12), date(2023, 1, 18))  
]

Для поиска объекта Contract в этом списке определена вспомогательная функция find\_contract().

При обращении к корню веб-приложения, то есть по пути "/", оно будет отправлять в ответ файл index.html, то есть веб-страницу, посредством которой пользователь сможет взаимодействовать с сервером:

@app.get("/")  
def main():  
 return FileResponse("public/index.html")

Далее определяются функции, которые собственно и представляют API. Вначале определяется функция, которая обрабатывает запрос типа GET по пути "api/contracts":

@app.get("/api/contracts", response\_model=List[ContractModel])  
def get\_contracts():  
 return acts

Запрос GET предполагает получение объектов, и в данном случае отправляем выше определенный список объектов Person.

Когда клиент обращается к приложению для получения одного объекта по id в запрос типа GET по адресу "api/contracts/{code}", то срабатывает другая функция:

@app.get("/api/contracts/{code}", response\_model=ContractModel)  
def get\_contract(code: str):  
 contract = find\_contract\_by\_code(code)  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 return contract

Здесь через параметр id получаем из пути запроса идентификатор объекта Contract и по этому идентификатору ищем нужный объект в списке acts. Если объект по code не был найден, то возвращаем с помощью класса JSONResponse статусный код 404 с некоторым сообщением в формате JSON. Если объект найден, то отправляем найденный объект клиенту.

При получении запроса типа DELETE по маршруту “api/contracts/{code}" срабатывает другая функция:

@app.delete("/api/contracts/{code}")  
def delete\_contract(code: str):  
 contract = find\_contract\_by\_code(code)  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 acts.remove(contract)  
 return {"message": "Контракт удален", "code": code}

Здесь действует аналогичная логика - если объект по id не найден, отправляет статусный код 404. Если же объект найден, то удаляем его из списка и посылаем клиенту.

При получении запроса с методом POST по адресу "/api/contracts " срабатывает следующая функция:

@app.post("/api/contracts", response\_model=ContractModel)  
def create\_contract(data=Body()):  
 contract = Contract(data["name"], data["labor"], data["employeeName"],  
 data["post"], data["issueDate"], data["finalDate"])  
 acts.append(contract)  
 return contract

Запрос типа POST предполагает передачу приложению отправляемых данных. Причем мы ожидаем, что клиент отправит данные, которые содержат значения name, labor, employeeName, post, issueDate, finalDate. Для краткости мы пока опускаем валидацию входных данных. И для получения данных из тела запроса с помощью класса Body получаем данные в параметр data и затем используем данные из этого параметра для создания объекта Contract. Затем созданный объект добавляется в список acts и отправляется назад клиенту.

Если приложению приходит PUT-запрос по адресу "/api/contracts", то аналогичным образом получаем отправленные клиентом данные в виде объекта Contract и пытаемся найти подобный объект в списке acts. Если объект не найден, отправляем статусный код 404. Если объект найден, то изменяем его данные и отправляем обратно клиенту:

@app.put("/api/contracts", response\_model=ContractModel)  
def update\_contract(data=Body()):  
 contract = find\_contract\_by\_code(data["code"])  
 if not contract:  
 raise HTTPException(status\_code=404, detail="Контракт не найден")  
 contract.name = data["name"]  
 contract.labor = data["labor"]  
 contract.employeeName = data["employeeName"]  
 contract.post = data["post"]  
 contract.issueDate = data["issueDate"]  
 contract.finalDate = data["finalDate"]  
 return contract

Таким образом, мы определили простейший API. Теперь добавим код клиента.

### Определение клиента

Теперь в проекте определим папку public, в которую добавим новый файл index.html

Определим в файле index.html следующим код для взаимодействия с сервером FastAPI:

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8" />  
 <title>METANIT.COM</title>  
 <style>  
 td {padding:5px;}  
 button {margin: 5px;}  
 </style>  
</head>  
<body>  
 <h2>Список договоров</h2>  
 <div>  
 <input type="hidden" id="actCode" />  
 <p>  
 Название:<br/>  
 <input id="actName" />  
 </p>  
 <p>  
 Трудоемкость:<br />  
 <input id="actLabor" type="number" />  
 </p>  
 <p>  
 ФИО сотрудника:<br/>  
 <input id="actEmployeeName" />  
 </p>  
 <p>  
 Должность:<br />  
 <input id="actPost"/>  
 </p>  
 <p>  
 Дата выдачи поручения на работу:<br/>  
 <input id="actIssueDate" type="date"/>  
 </p>  
 <p>  
 Дата окончания работы:<br />  
 <input id="actFinalDate" type="date"/>  
 </p>  
 <p>  
 <button id="saveBtn">Сохранить</button>  
 <button id="resetBtn">Сбросить</button>  
 </p>  
 </div>  
 <table>  
 <thead>  
 <tr>  
 <th>Шифр документа</th>  
 <th>Имя</th>  
 <th>Трудоемкость</th>  
 <th>ФИО сотрудника</th>  
 <th>Должность</th>  
 <th>Дата выдачи поручения на работу</th>  
 <th>Дата окончания работы</th>  
 <th></th>  
 </tr>  
 </thead>  
 <tbody>  
 </tbody>  
 </table>  
  
 <script>  
 // Получение всех контрактов  
 async function getUsers() {  
 const response = await fetch("/api/contracts", {  
 method: "GET",  
 headers: { "Accept": "application/json" }  
 });  
  
 if (response.ok === true) {  
 const users = await response.json();  
 const rows = document.querySelector("tbody");  
 rows.innerHTML = "";  
 users.forEach(user => rows.append(row(user)));  
 }  
 }  
  
 // Получение одного контракта по шифру  
 async function getUser(code) {  
 const response = await fetch(`/api/contracts/${code}`, {  
 method: "GET",  
 headers: { "Accept": "application/json" }  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 document.getElementById("actCode").value = user.code;  
 document.getElementById("actName").value = user.name;  
 document.getElementById("actLabor").value = user.labor;  
 document.getElementById("actEmployeeName").value = user.employeeName;  
 document.getElementById("actPost").value = user.post;  
 document.getElementById("actIssueDate").value = user.issueDate;  
 document.getElementById("actFinalDate").value = user.finalDate;  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
 }  
  
 // Добавление контракта  
 async function createUser(actName, actLabor, actEmployeeName, actPost, actIssueDate, actFinalDate) {  
 const response = await fetch("/api/contracts", {  
 method: "POST",  
 headers: {  
 "Accept": "application/json",  
 "Content-Type": "application/json"  
 },  
 body: JSON.stringify({  
 name: actName,  
 labor: parseInt(actLabor, 10),  
 employeeName: actEmployeeName,  
 post: actPost,  
 issueDate: actIssueDate,  
 finalDate: actFinalDate  
 })  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 reset();  
 getUsers(); // Обновляем список  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
 }  
  
 // Изменение контракта  
 async function editUser(actCode, actName, actLabor, actEmployeeName, actPost, actIssueDate, actFinalDate) {  
 const response = await fetch("/api/contracts", {  
 method: "PUT",  
 headers: {  
 "Accept": "application/json",  
 "Content-Type": "application/json"  
 },  
 body: JSON.stringify({  
 code: actCode,  
 name: actName,  
 labor: parseInt(actLabor, 10),  
 employeeName: actEmployeeName,  
 post: actPost,  
 issueDate: actIssueDate,  
 finalDate: actFinalDate  
 })  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 reset();  
 getUsers(); // Обновляем список  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
 }  
  
 // Удаление контракта по шифру  
 async function deleteUser(code) {  
 if (!confirm("Вы уверены, что хотите удалить эту запись?")) return;  
  
 const response = await fetch(`/api/contracts/${code}`, {  
 method: "DELETE",  
 headers: { "Accept": "application/json" }  
 });  
  
 if (response.ok === true) {  
 // Удаляем строку из таблицы  
 const row = document.querySelector(`tr[data-rowcode="${code}"]`);  
 if (row) row.remove();  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
 }  
  
 // Сброс данных формы  
 function reset() {  
 document.getElementById("actCode").value = "";  
 document.getElementById("actName").value = "";  
 document.getElementById("actLabor").value = "";  
 document.getElementById("actEmployeeName").value = "";  
 document.getElementById("actPost").value = "";  
 document.getElementById("actIssueDate").value = "";  
 document.getElementById("actFinalDate").value = "";  
 }  
  
 // Создание строки для таблицы  
 function row(user) {  
 const tr = document.createElement("tr");  
 tr.setAttribute("data-rowcode", user.code);  
  
 // Ячейка с шифром документа  
 const codeTd = document.createElement("td");  
 codeTd.append(user.code);  
 tr.append(codeTd);  
  
 const nameTd = document.createElement("td");  
 nameTd.append(user.name);  
 tr.append(nameTd);  
  
 const laborTd = document.createElement("td");  
 laborTd.append(user.labor);  
 tr.append(laborTd);  
  
 const employeeNameTd = document.createElement("td");  
 employeeNameTd.append(user.employeeName);  
 tr.append(employeeNameTd);  
  
 const postTd = document.createElement("td");  
 postTd.append(user.post);  
 tr.append(postTd);  
  
 const issueDateTd = document.createElement("td");  
 issueDateTd.append(user.issueDate);  
 tr.append(issueDateTd);  
  
 const finalDateTd = document.createElement("td");  
 finalDateTd.append(user.finalDate);  
 tr.append(finalDateTd);  
  
 const linksTd = document.createElement("td");  
  
 const editLink = document.createElement("button");  
 editLink.append("Изменить");  
 editLink.addEventListener("click", async () => await getUser(user.code));  
 linksTd.append(editLink);  
  
 const removeLink = document.createElement("button");  
 removeLink.append("Удалить");  
 removeLink.addEventListener("click", async () => await deleteUser(user.code));  
 linksTd.append(removeLink);  
  
 tr.appendChild(linksTd);  
  
 return tr;  
 }  
  
 // Сброс значений формы  
 document.getElementById("resetBtn").addEventListener("click", () => reset());  
  
 // Отправка формы  
 document.getElementById("saveBtn").addEventListener("click", async () => {  
 const code = document.getElementById("actCode").value;  
 const name = document.getElementById("actName").value;  
 const labor = document.getElementById("actLabor").value;  
 const employeeName = document.getElementById("actEmployeeName").value;  
 const post = document.getElementById("actPost").value;  
 const issueDate = document.getElementById("actIssueDate").value;  
 const finalDate = document.getElementById("actFinalDate").value;  
  
 if (code === "") {  
 await createUser(name, labor, employeeName, post, issueDate, finalDate);  
 } else {  
 await editUser(code, name, labor, employeeName, post, issueDate, finalDate);  
 }  
 });  
  
 // Загрузка контрактов  
 getUsers();  
 </script>  
</body>  
</html>

Основная логика здесь заключена в коде javascript. При загрузке страницы в браузере получаем все объекты из БД с помощью функции getUsers():

async function getUsers() {  
 const response = await fetch("/api/contracts", {  
 method: "GET",  
 headers: { "Accept": "application/json" }  
 });  
  
 if (response.ok === true) {  
 const users = await response.json();  
 const rows = document.querySelector("tbody");  
 rows.innerHTML = "";  
 users.forEach(user => rows.append(row(user)));  
 }  
}

Для добавления строк в таблицу используется функция row(), которая возвращает строку. В этой строке будут определены ссылки для изменения и удаления пользователя.

Ссылка для изменения пользователя с помощью функции getUser() получает с сервера выделенного пользователя:

async function getUser(code) {  
 const response = await fetch(`/api/contracts/${code}`, {  
 method: "GET",  
 headers: { "Accept": "application/json" }  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 document.getElementById("actCode").value = user.code;  
 document.getElementById("actName").value = user.name;  
 document.getElementById("actLabor").value = user.labor;  
 document.getElementById("actEmployeeName").value = user.employeeName;  
 document.getElementById("actPost").value = user.post;  
 document.getElementById("actIssueDate").value = user.issueDate;  
 document.getElementById("actFinalDate").value = user.finalDate;  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
}

И выделенный пользователь добавляется в форму над таблицей. Эта же форма применяется и для добавления объекта. С помощью скрытого поля, которое хранит id пользователя, мы можем узнать, какое действие выполняется - добавление или редактирование. Если id не установлен (равен пустой строке), то выполняется функция createUser, которая отправляет данные в POST-запросе:

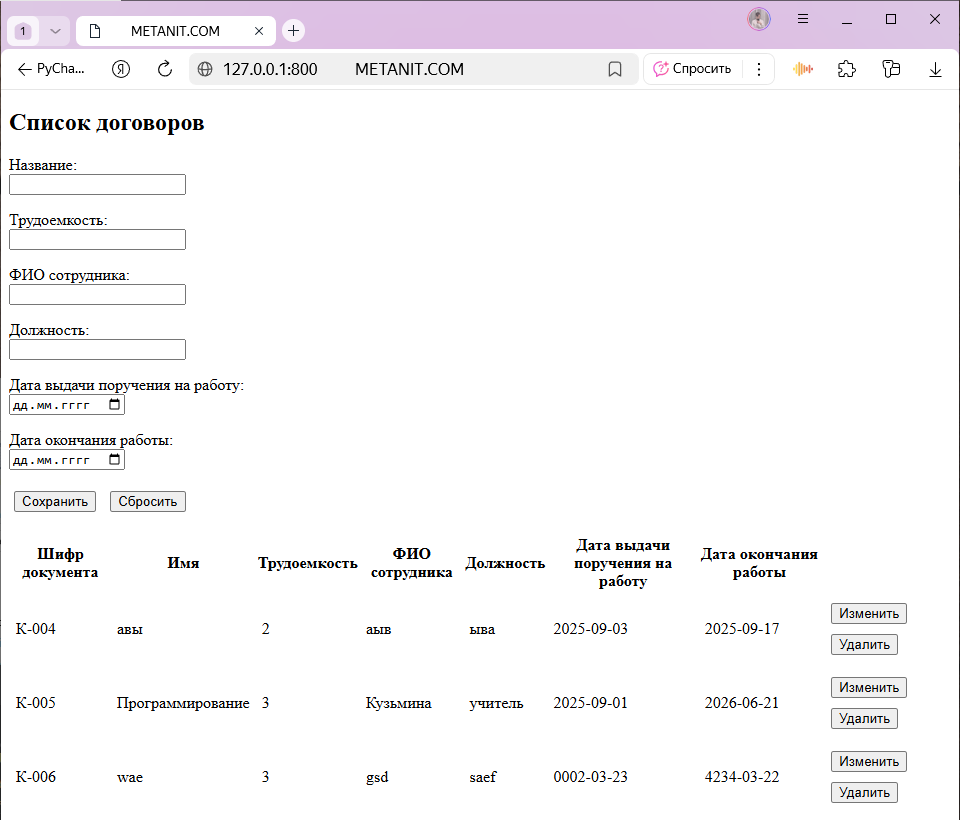
async function createUser(actName, actLabor, actEmployeeName, actPost, actIssueDate, actFinalDate) {  
 const response = await fetch("/api/contracts", {  
 method: "POST",  
 headers: {  
 "Accept": "application/json",  
 "Content-Type": "application/json"  
 },  
 body: JSON.stringify({  
 name: actName,  
 labor: parseInt(actLabor, 10),  
 employeeName: actEmployeeName,  
 post: actPost,  
 issueDate: actIssueDate,  
 finalDate: actFinalDate  
 })  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 reset();  
 getUsers(); // Обновляем список  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
}

Если же ранее пользователь был загружен на форму, и в скрытом поле сохранился его id, то выполняется функция editUser, которая отправляет PUT-запрос:

async function editUser(actCode, actName, actLabor, actEmployeeName, actPost, actIssueDate, actFinalDate) {  
 const response = await fetch("/api/contracts", {  
 method: "PUT",  
 headers: {  
 "Accept": "application/json",  
 "Content-Type": "application/json"  
 },  
 body: JSON.stringify({  
 code: actCode,  
 name: actName,  
 labor: parseInt(actLabor, 10),  
 employeeName: actEmployeeName,  
 post: actPost,  
 issueDate: actIssueDate,  
 finalDate: actFinalDate  
 })  
 });  
  
 if (response.ok === true) {  
 const user = await response.json();  
 reset();  
 getUsers(); // Обновляем список  
 } else {  
 const error = await response.json();  
 console.log(error.message);  
 }  
}

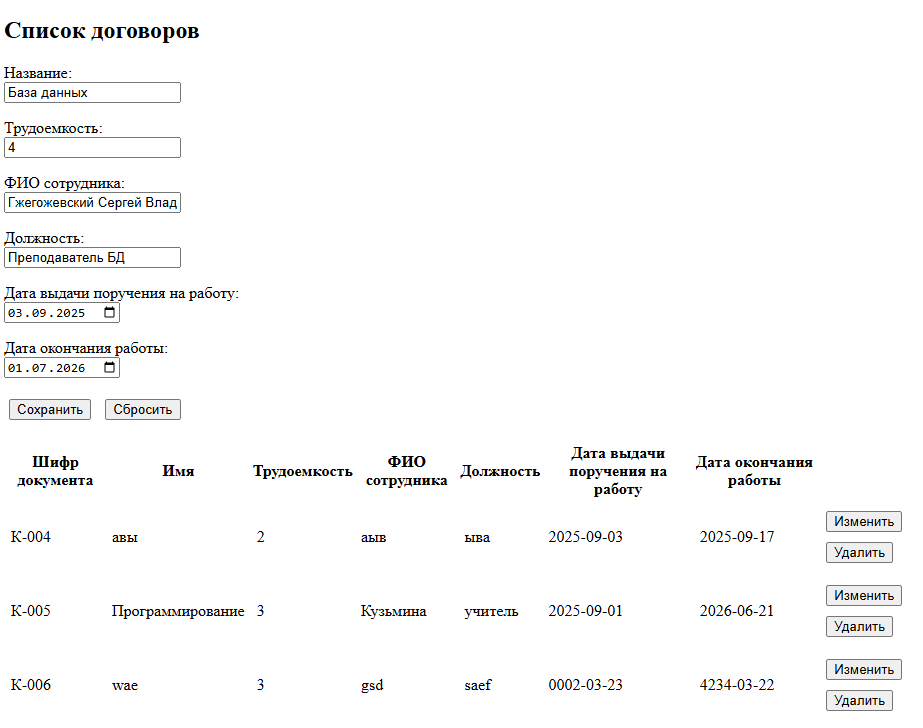
И функция deleteUser() посылает на сервер запрос типа DELETE на удаление пользователя, и при успешном удалении на сервере удаляет объект по id из списка объектов Contract.

Теперь запустим проект, и по умолчанию приложение отправит браузеру веб-страницу index.html, которая загрузит список объектов:



После этого мы сможем выполнять все базовые операции с договорами - получение, добавление, изменение, удаление. Например, добавим новый договор:

До:



После:

