Практическая работа №2. Создание клиентской WEB страницы для приложения «Моя ГИБДД»

Single Page Application (SPA) - это одностраничное приложение (англ. «сингл пейдж аппликейшен, SPA) — это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX.

Одностраничники похожи на «native» (англ) или иным словом «родные»(рус) приложения для операционной системы, с той лишь разницей, что исполняются в рамках браузера, а не в собственном процессе операционной системы.

Для создания Single Page Application (SPA) с разделами "Список владельцев", "Список авто" и "Регистрация авто", создадим отдельную папку для frontend-части. В этой папке будет HTML, CSS и JavaScript файлы.

Файловая структура проекта:

```
mygibdd/
├── frontend/
│   ├── index.html
│   ├── styles.css
│   ├── script.js
├── backend/ (ваша предыдущая backend-часть)
```

1. HTML (файл frontend/index.html)

Шаг 1.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>МояГИБДД</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>МояГИБДД</h1>
    <div class="container">···
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Шаг 2.

```
<div class="container">
    <!-- Список владельцев -->
    <section>
        <h2>Список владельцев</h2>
        <table id="owners-table">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Фамилия</th>
                    <th>Имя</th>
                    <th>Телефон</th>
                    <th>Почта</th>
                    <th>Город</th>
                </tr>
            </thead>
            <tbody>
                <!-- Данные будут загружены через JavaScript -->
            </tbody>
        </table>
    </section>
```

Шаг 3.

```
    <!-- Список авто -->
    <section>
        <h2>Список авто</h2>
        <table id="autos-table">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Марка</th>
                    <th>Модель</th>
                    <th>Цвет</th>
                    <th>Госномер</th>
                </tr>
            </thead>
            <tbody>
                <!-- Данные будут загружены через JavaScript -->
            </tbody>
        </table>
    </section>
```

Шаг 4.

```
    <!-- Регистрация авто -->
    <section>
        <h2>Регистрация авто</h2>
        <form id="register-auto-form">
            <label for="owner-select">Владелец:</label>
            <select id="owner-select">
                <!-- Опции будут загружены через JavaScript -->
            </select>

            <label for="auto-select">Автомобиль:</label>
            <select id="auto-select">
                <!-- Опции будут загружены через JavaScript -->
            </select>

            <button type="submit">Зарегистрировать</button>
        </form>

        <h3>Принадлежащие авто:</h3>
        <ul id="owned-autos-list">
            <!-- Данные будут загружены через JavaScript -->
        </ul>
    </section>
```

2. CSS (файл frontend/styles.css)

Впишите комментарии для повторения css атрибутов на против строк, щначение которых вы не знали или забыли.

Часть 1

```css
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

h1, h2 {
    color: #333;
}

.container {
    max-width: 800px;
    margin: 0 auto;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 20px;
}

table, th, td {
    border: 1px solid #ddd;
}

th, td {
    padding: 8px;
    text-align: left;
}

th {
    background-color: #f4f4f4;
}

form {
    margin-bottom: 20px;
}
```

Часть 2.

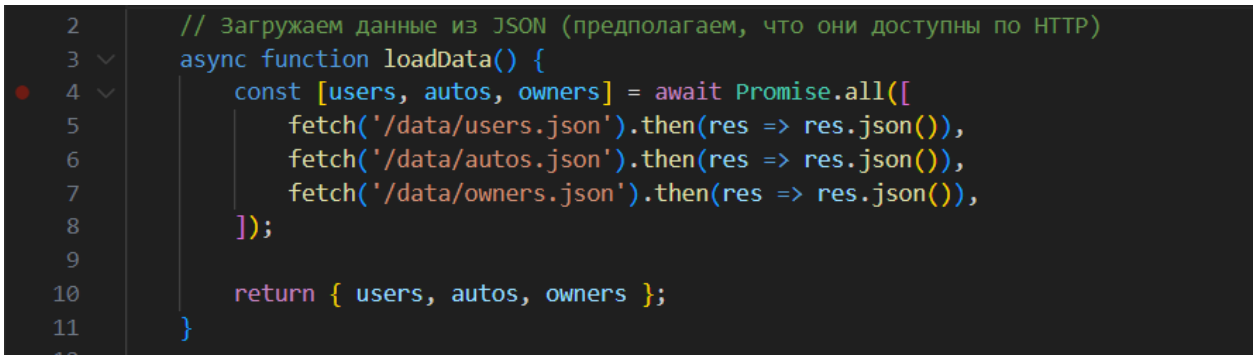
```css
label {
    display: block;
    margin-bottom: 5px;
}

select, button {
    padding: 8px;
    margin-bottom: 10px;
    width: 100%;
    max-width: 300px;
}

button {
    background-color: #28a745;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #218838;
}

ul {
    list-style-type: none;
    padding: 0;
}

ul li {
    background-color: #f8f9fa;
    padding: 8px;
    margin-bottom: 5px;
    border: 1px solid #ddd;
}
```

3. JavaScript (файл frontend/script.js)

Шаг 1. Создаем фугкцию, которая выполняется ПОСЛЕ загрузки HTML DOM-объекта (DOM – это содержимое HTML для открываемой страницы)

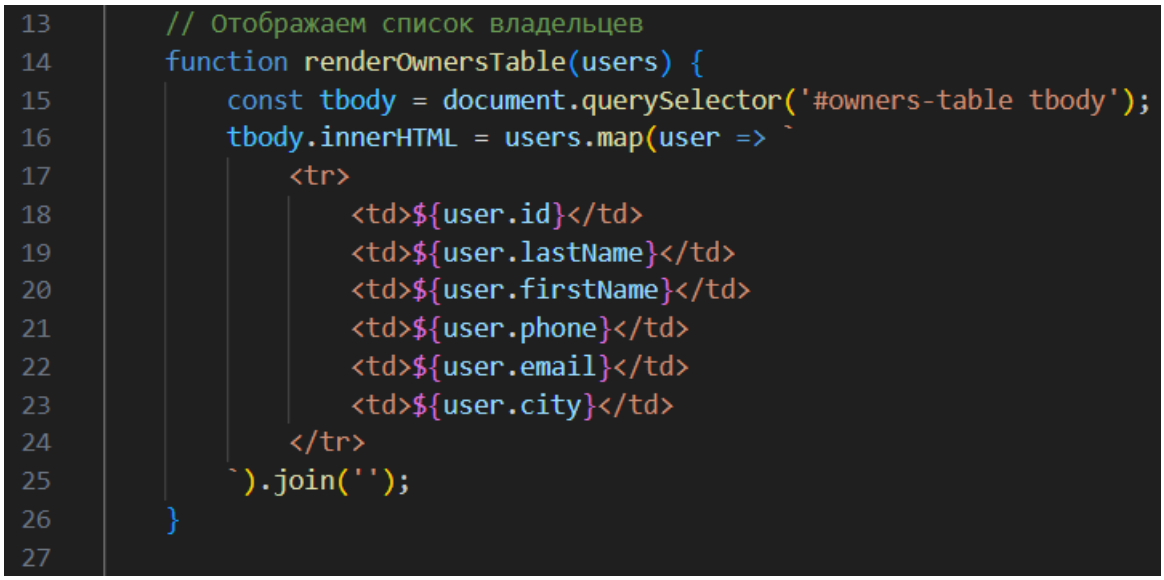
Шаг 2. Создаем асинхронную = работает параллельно с загрузкой страницы HTML DOM – это основной поток. Асинхронность нужна для долгих или рискованных операций, которые задерживают загрузку DOM или работу других функций. Пользователю нельзя показывать медленную страницу иначе user просто закроет страницу, а нам надо, чтобы наш сайт был удобен и быстр, как пуля.

После выполнения асинхронная функция отправляет сигнал основному потоку, что она закончила работу. Основной поток = отображение страницы может показать то, что подготовила функция loadData()



```
// Загружаем данные из JSON (предполагаем, что они доступны по HTTP)
async function loadData() {
    const [users, autos, owners] = await Promise.all([
        fetch('/data/users.json').then(res => res.json()),
        fetch('/data/autos.json').then(res => res.json()),
        fetch('/data/owners.json').then(res => res.json()),
    ]);

    return { users, autos, owners };
}
```

Шаг 3. Создаем функцию рендеринга (размещения данных объекта в таблице, другими словами: функция рендеринга создает строки таблицы и размещает в них данные)



```
// Отображаем список владельцев
function renderOwnersTable(users) {
    const tbody = document.querySelector('#owners-table tbody');
    tbody.innerHTML = users.map(user => `
        <tr>
            <td>${user.id}</td>
            <td>${user.lastName}</td>
            <td>${user.firstName}</td>
            <td>${user.phone}</td>
            <td>${user.email}</td>
            <td>${user.city}</td>
        </tr>
    `).join('');
}

```

Шаг 4. аналогично

```
28      // Отображаем список авто
29      function renderAutosTable(autos) {
30          const tbody = document.querySelector('#autos-table tbody');
31          tbody.innerHTML = autos.map(auto => `
32              <tr>
33                  <td>${auto.id}</td>
34                  <td>${auto.brand}</td>
35                  <td>${auto.model}</td>
36                  <td>${auto.color}</td>
37                  <td>${auto.licensePlate}</td>
38              </tr>
39          `).join('');
40      }
```

Шаг 5. Повторите обращение к элементу DOM из js, опишите в комментариях ниже блока кода – как работает наполнение списка для выбора владельца.

```
42      // Заполняем выпадающий список владельцев
43      function renderOwnerSelect(users) {
44          const select = document.getElementById('owner-select');
45          select.innerHTML = users.map(user => `
46              <option value="${user.id}">${user.lastName} ${user.firstName}</option>
47          `).join('');
48      }
```

Шаг 6. Аналогично напишите кода выбора авто

```
50      // Заполняем выпадающий список авто
51      function renderAutoSelect(autos) {
52          const select = document.getElementById('auto-select');
53          select.innerHTML = autos.map(auto => `
54              <option value="${auto.id}">${auto.brand} ${auto.model} (${auto.licensePlate})</option>
55          `).join('');
56      }
```

Шаг 7.

```
58      // Отображаем принадлежащие авто
59      function renderOwnedAutos(owners, autos, userId) {
60          const ownedAutos = owners
61              .filter(owner => owner.userId === userId && !owner.deregistrationDate)
62              .map(owner => autos.find(auto => auto.id === owner.autoId));
63
64          const list = document.getElementById('owned-autos-list');
65          list.innerHTML = ownedAutos.map(auto => `
66              <li>${auto.brand} ${auto.model} (${auto.licensePlate})</li>
67          `).join('');
68      }
```

Шаг 8. Напишем функцию инициализации данных страницы

```
70      // Инициализация страницы
71      async function init() {···
107     }
```

Шаг 8.1. Вызываем функцию loadData() с модификатором await – который ждет выполнения функции

```
72          const { users, autos, owners } = await loadData();
```

Шаг 8.2. формируем данные

```
74          renderOwnersTable(users);
75          renderAutosTable(autos);
76          renderOwnerSelect(users);
77          renderAutoSelect(autos);
```

Шаг 8.3.

```
79          // Обработчик выбора владельца
80          const ownerSelect = document.getElementById('owner-select');
81          ownerSelect.addEventListener('change', (e) => {
82              const userId = parseInt(e.target.value);
83              renderOwnedAutos(owners, autos, userId);
84          });
```

Шаг 8.4.

```
86          // Обработчик формы регистрации авто
87          const form = document.getElementById('register-auto-form');
88          form.addEventListener('submit', (e) => {
89              e.preventDefault();
90              const userId = parseInt(ownerSelect.value);
91              const autoId = parseInt(document.getElementById('auto-select').value);
```

Шаг 8.5.

```
93          // Добавляем запись в owners (в реальном приложении это будет POST-запрос на сервер)
94          owners.push({
95              userId,
96              autoId,
97              registrationDate: new Date().toISOString().split('T')[0],
98              deregistrationDate: null,
99          });
```

Шаг 8.6.

```
101             // Обновляем список принадлежащих авто
102             renderOwnedAutos(owners, autos, userId);
103         });
104
```

Шаг 8.7.

```
105         // Инициализация начального состояния
106         renderOwnedAutos(owners, autos, parseInt(ownerSelect.value));
107     }
108
```

Шаг 8.8

```
109     init();
110 });
```

Как это работает:

1. **HTML**:
    - Три секции: "Список владельцев", "Список авто" и "Регистрация авто".
    - Таблицы и формы для взаимодействия с данными.
2. **CSS**:
    - Простые стили для оформления таблиц, форм и списков.
3. **JavaScript**:
    - Загружает данные из JSON-файлов (предполагается, что они доступны по HTTP).
    - Отображает данные в таблицах и выпадающих списках.
    - Позволяет регистрировать авто за владельцами и отображает принадлежащие авто.

Как использовать:

1. Запустите backend-часть (Node.js), чтобы данные были доступны по HTTP.
2. Откройте `index.html` в браузере.
3. Взаимодействуйте с интерфейсом: просматривайте списки, регистрируйте авто за владельцами.

Теперь у вас есть полноценное SPA для управления данными владельцев и автомобилей!