ГУАП

КАФЕДРА № 42

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С.Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1  Архитектура системы. MV – шаблон |
| по курсу: Технологии программирования |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |
| --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4321 |  | Г.В. Буренков |
|  |  | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2025

**СОДЕРЖАНИЕ**

[1 Цель работы 2](#_Toc191656533)

[2 Задание 3](#_Toc191656534)

[3 Ход работы 4](#_Toc191656535)

[4 Листинг с кодом программы 5](#_Toc191656536)

[5 Результат работы программы 14](#_Toc191656537)

[6 Вывод 16](#_Toc191656538)

**1 Цель работы**

Вспомнить и применить на практике принципы объектно-ориентированного программирования (ООП), используя основные элементы и понятия ООП: классы, объекты, интерфейсы, наследование, инкапсуляция, полиморфизм, абстракция. Внедрить в проект шаблон проектирования MVP (рекомендуется, но можно MVVM, MVI и др.). Изучить и применить на практике принципы SOLID.

**2 Задание**

Необходимо проект своей курсовой работы, написанной в третьем семестре по дисциплине «Основы программирования» переделать в соответствии со стандартами шаблона MVP [Model-View-Presenter] (рекомендуется, но можно MVVM, MVI и др.). Если же проект курсовой был неудачный, некачественный, не соответствует требованиям ниже или просто не нравится, то можно и нужно дополнить старый либо создать новый проект на любую тематику. Проект обязательно должен иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования.

**3 Ход работы**

В данной лабораторной работе был разработан проект веб-приложения для управления пользователями, который реализует паттерн Model-View-Presenter (MVP) и использует LocalStorage для хранения данных. Основная логика приложения сосредоточена в классе UserModel, который отвечает за управление пользователями. При инициализации класса происходит загрузка данных из LocalStorage, что позволяет сохранять состояние приложения между сессиями. Метод saveToStorage сериализует массив пользователей в JSON-формат и сохраняет его в LocalStorage, а метод loadFromStorage загружает данные, если они существуют. Это обеспечивает постоянный доступ к данным пользователей, даже после перезагрузки страницы.

Класс UserPresenter управляет взаимодействием между моделью и представлением. Он обрабатывает события, такие как добавление, редактирование и удаление пользователей, а также обновляет интерфейс в соответствии с изменениями в модели. При добавлении нового пользователя данные извлекаются из формы, и вызывается метод addUser модели, который добавляет пользователя в массив и сохраняет его в LocalStorage. При редактировании пользователя используется метод editUser, который обновляет данные конкретного пользователя по его идентификатору. Удаление пользователя осуществляется через метод deleteUser, который фильтрует массив пользователей, исключая удаляемого. Вся логика отображения пользователей в таблице реализована в методе renderUsers, который фильтрует и сортирует пользователей в зависимости от введенных критериев поиска и выбранного статуса. Таким образом, приложение обеспечивает удобный интерфейс для управления пользователями, сохраняя данные в LocalStorage для долговременного хранения.

**4 Листинг с кодом программы**

В данном разделе представлен исходный код программы:

//TYPESCRIPT

interface IUser {

    id: number;

    name: string;

    age: number;

    status: "active" | "blocked";

}

class UserModel {

    private users: IUser[] = [];

    constructor() {

        this.loadFromStorage();

    }

    private saveToStorage() {

        localStorage.setItem("users", JSON.stringify(this.users));

    }

    private loadFromStorage() {

        const storedUsers = localStorage.getItem("users");

        if (storedUsers) {

            this.users = JSON.parse(storedUsers);

        }

    }

    addUser(user: IUser): void {

        this.users.push(user);

        this.saveToStorage();

    }

    editUser(id: number, newUserData: Partial<IUser>): void {

        const userIndex = this.users.findIndex(user => user.id === id);

        if (userIndex !== -1) {

            this.users[userIndex] = { ...this.users[userIndex], ...newUserData };

            this.saveToStorage();

        }

    }

    deleteUser(id: number): void {

        this.users = this.users.filter(user => user.id !== id);

        this.saveToStorage();

    }

    getUsers(): IUser[] {

        return this.users;

    }

}

class UserPresenter {

    private model: UserModel;

    private tableBody: HTMLTableSectionElement;

    private form: HTMLFormElement;

    private nameInput: HTMLInputElement;

    private ageInput: HTMLInputElement;

    private statusInput: HTMLSelectElement;

    private searchInput: HTMLInputElement;

    private sortButton: HTMLButtonElement;

    private filterSelect: HTMLSelectElement;

    private isSorted: boolean = false;

    constructor (model: UserModel) {

        this.model = model;

        this.tableBody = document.getElementById("user-table-body") as HTMLTableSectionElement;

        this.form = document.getElementById("user-form") as HTMLFormElement;

        this.nameInput = document.getElementById("name") as HTMLInputElement;

        this.ageInput = document.getElementById("age") as HTMLInputElement;

        this.statusInput = document.getElementById("status") as HTMLSelectElement;

        this.searchInput = document.getElementById("search") as HTMLInputElement;

        this.sortButton = document.getElementById("sort-age") as HTMLButtonElement;

        this.filterSelect = document.getElementById("filter-status") as HTMLSelectElement

        this.init();

     }

     private init() {

        this.form.addEventListener("submit", (e) => this.handleAddUser(e));

        this.searchInput.addEventListener("input", () => this.renderUsers());

        this.sortButton.addEventListener("click", () => this.toggleSort());

        this.filterSelect.addEventListener("change", () => this.renderUsers());

        this.renderUsers();

     }

     private handleAddUser(event: Event) {

        event.preventDefault();

        const name = this.nameInput.value.trim();

        const age = Number(this.ageInput.value);

        const status = this.statusInput.value as "active" | "blocked";

        const editingId = this.form.dataset.editing;

        if (editingId) {

            this.model.editUser(Number(editingId), { name, age, status });

            delete this.form.dataset.editing;

        } else {

            this.model.addUser({ id: Date.now(), name, age, status });

        }

        this.renderUsers();

        this.form.reset();

    }

    private renderUsers() {

        const searchTerm = this.searchInput.value.toLowerCase();

        const filterStatus = this.filterSelect.value;

        let users = this.model.getUsers().filter(user =>

            user.name.toLowerCase().includes(searchTerm) &&

            (filterStatus === "all" || user.status === filterStatus)

        );

        if (this.isSorted) {

            users = users.slice().sort((a, b) => a.age - b.age);

        }

        this.tableBody.innerHTML = "";

        users.forEach(user => {

            const row = document.createElement("tr");

            row.innerHTML = `

                <td>${user.id}</td>

                <td>${user.name}</td>

                <td>${user.age}</td>

                <td>${user.status}</td>

                <td>

                    <button class="edit-btn" data-id="${user.id}">✏ Редактировать</button>

                    <button class="delete-btn" data-id="${user.id}">❌ Удалить</button>

                </td>

            `;

            this.tableBody.appendChild(row);

        });

        document.querySelectorAll(".delete-btn").forEach(button => {

            button.addEventListener("click", (e) => this.handleDeleteUser(e));

        });

        document.querySelectorAll(".edit-btn").forEach(button => {

            button.addEventListener("click", (e) => this.handleEditUser(e));

        });

    }

    private handleDeleteUser(event: Event) {

        const button = event.target as HTMLButtonElement;

        const userId = Number(button.dataset.id);

        this.model.deleteUser(userId);

        this.renderUsers();

    }

    private handleEditUser(event: Event) {

        const button = event.target as HTMLButtonElement;

        const userId = Number(button.dataset.id);

        const user = this.model.getUsers().find(user => user.id === userId);

        if (user) {

            this.nameInput.value = user.name;

            this.ageInput.value = user.age.toString();

            this.statusInput.value = user.status;

            this.form.dataset.editing = userId.toString();

        }

    }

    private toggleSort() {

        this.isSorted = !this.isSorted;

        this.renderUsers();

    }

}

const userModel = new UserModel();

const userPresenter = new UserPresenter(userModel)

//HTML

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="style.css">

    <title>Document</title>

</head>

<body>

    <h1>Управление пользователями</h1>

    <form id="user-form">

        <input type="text" id="name" placeholder="Имя" required>

        <input type="number" id="age" placeholder="Возраст" required>

        <select id="status">

            <option value="active">Активный</option>

            <option value="blocked">Заблокирован</option>

        </select>

        <button type="submit">Добавить</button>

    </form>

    <input type="text" id="search" placeholder="Поиск по имени...">

    <button id="sort-age">Сортировать по возрасту</button>

    <select id="filter-status">

        <option value="all">Все</option>

        <option value="active">Активные</option>

        <option value="blocked">Заблокированные</option>

    </select>

    <table border="1">

        <thead>

            <tr>

                <th>ID</th>

                <th>Имя</th>

                <th>Возраст</th>

                <th>Статус</th>

                <th>Действия</th>

            </tr>

        </thead>

        <tbody id="user-table-body">

*<!-- Здесь будет список пользователей -->*

        </tbody>

    </table>

    <script type="module" src="./dist/app.js"></script>

</body>

</html>

**5 Результат работы программы**

Результаты работы программы демонстрируют динамическую таблицу управления пользователями. Модуль можно импортировать в другие приложения. Программа умеет фильтровать по имени, возрасту, по статусу. На рисунке 1 изображено приложение управления пользователями.

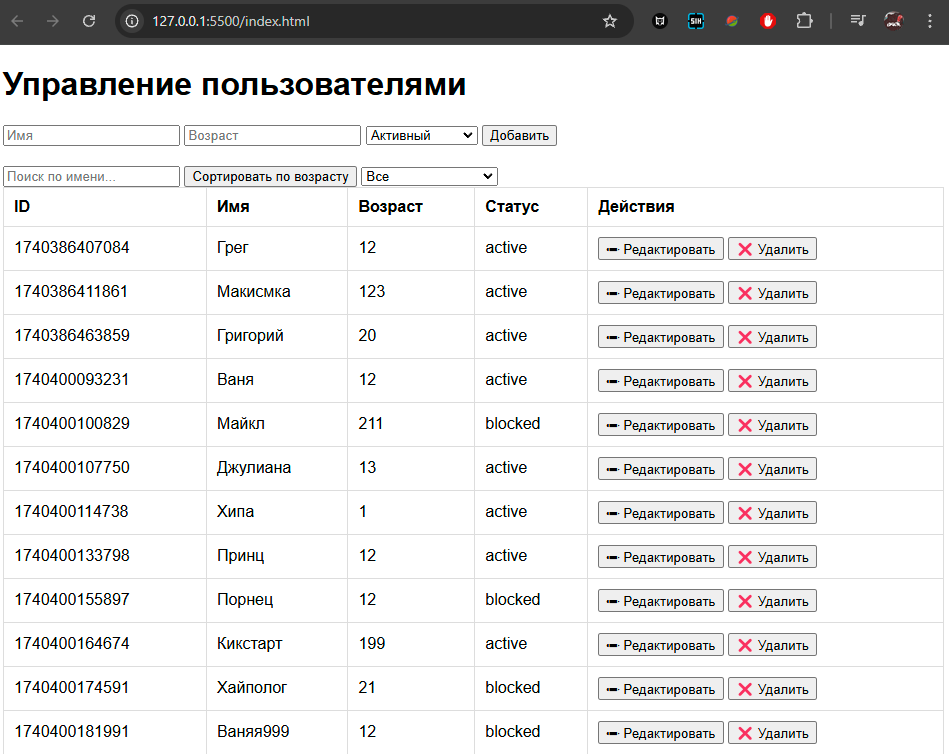


Рисунок 1 – Результат работы приложения

На рисунке 2 представлен результат работы приложения после удаления части юзеров, а также фильтрации по возрасту.

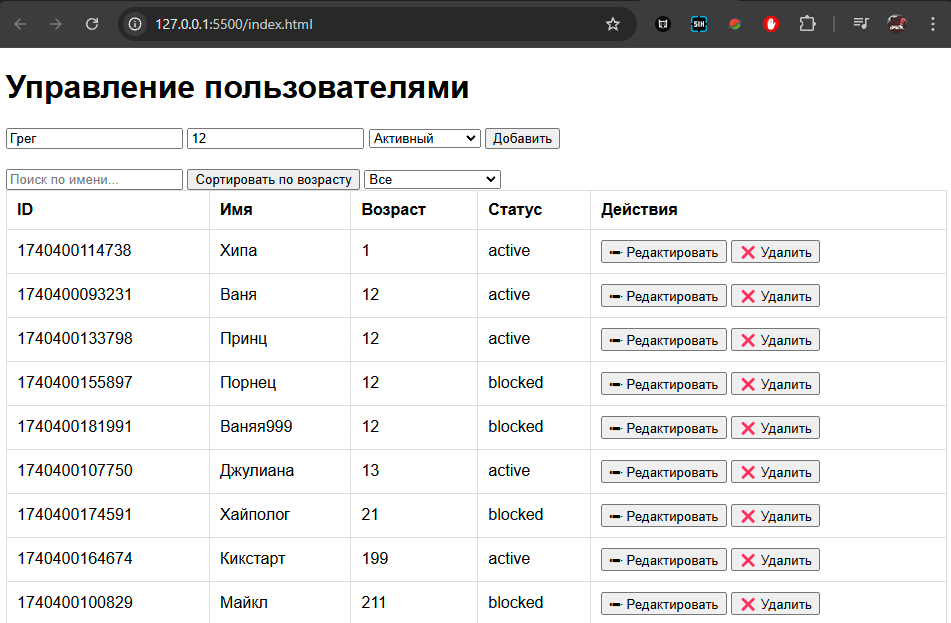


Рисунок 2 – Результат работы после изменений

**6 Вывод**

Результаты работы программы демонстрируют функциональное веб-приложение для управления пользователями, которое позволяет добавлять, редактировать, удалять и фильтровать пользователей по имени и статусу. Интерфейс приложения включает форму для ввода данных пользователя, таблицу для отображения списка пользователей и элементы управления для поиска и сортировки. Пользователи могут вводить имя, возраст и статус (активный или заблокированный), после чего данные сохраняются в локальном хранилище браузера, что обеспечивает их доступность даже после перезагрузки страницы.

При добавлении нового пользователя данные отображаются в таблице, где каждый пользователь представлен с уникальным идентификатором, именем, возрастом и статусом. Приложение также поддерживает функции редактирования и удаления пользователей, что позволяет легко управлять списком. Функция поиска позволяет пользователям находить нужные записи по имени, а фильтрация по статусу помогает быстро находить активных или заблокированных пользователей. На рисунке 1 представлен результат работы приложения, демонстрирующий интерфейс и функциональность управления пользователями, реализованную с использованием паттерна Model-View-Presenter (MVP), принципов SOLID, технологий HTML, CSS и TypeScript.