МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра технологий программирования

Кот Максим Артурович
Ларионова Анастасия Александровна
Щербаченя Михаил Евгеньевич

Технологии программирования

Отчет по лабораторной работе №8

«LocalStorage(или indexedDB), JSON. События. Формы, элементы управления.»

студентов 3 курса 8 группы

Преподаватель
Терех Владимир Сергеевич

Минск, 2024

Описание предметной области

АИС «ЖД вокзал покупка билетов на поезд» — это система, которая предназначена для автоматизации процесса выбора билетов на пригородные, междугородние и международные рейсы поездов, упрощения процесса их бронирования, оплаты и возврата. Приложение автоматизирует процесс покупки билетов как в онлайн-режиме, так и через кассу. Кроме того, система предоставляет пользователям возможность зарегистрироваться и просмотреть доступные рейсы. Приложение включает механизм взаимодействия с банковской системой для обработки оплаты и возврата средств.

Мы сделали проект динамическим. Теперь он может реагировать на действия пользователя, сохранять результат действий пользователя в localStorage. Теперь изменения не должны пропадать после перезагрузки страницы в браузере.

Реализация задания:

Использование LocalStorage

В работе реализованы методы save и restore в модели ViewModel, отвечающие за сохранение и восстановление данных из LocalStorage:

Метод save: Сохраняет текущие данные объектов (tickets, towns, ticketTypes) и статус авторизации пользователя (isAutorized) в LocalStorage.

```
save() {
        const tickets = this.#ticketManager.getObjs().map(ticket => ({
            start_place: ticket.Obj.start_place,
            end_place: ticket.Obj.end_place,
            start_date: ticket.Obj.start_date.toISOString(),
            finish_date: ticket.Obj.finish_date.toISOString(),
            cost: ticket.Obj.cost,
            type: ticket.Obj.type,
```

```
            purchased: ticket.Obj.purchased,
        }));
        localStorage.setItem('tickets', JSON.stringify(tickets));

        const towns = this.#townManager.getObjs().map(town => ({
            value: town.Obj.value,
            text: town.Obj.text,
        }));
        localStorage.setItem('towns', JSON.stringify(towns));

        const ticketTypes = this.#ticketTypeManager.getObjs().map(ticketType => ({
            value: ticketType.Obj.value,
            text: ticketType.Obj.text,
        }));
        localStorage.setItem('ticketTypes', JSON.stringify(ticketTypes));

        localStorage.setItem('isAutorized', this.#isAutorized);
    }
```

Метод restore: Загружает данные из LocalStorage, восстанавливая объекты, такие как билеты, города и типы билетов. Также проверяет статус авторизации.

```
restore() {
        const storedTicketTypes = JSON.parse(localStorage.getItem('ticketTypes'));
```

```
        storedTicketTypes.forEach(({ value, text }) =>
this.#ticketTypeManager.addObj(new TicketType(value, text)));

        this.#ticketTypeManager.getObjs().forEach(item => {

            const option = document.createElement('option');

            option.value = item.Obj.value;

            option.textContent = item.Obj.text;

            let selectBox = document.getElementById('type');

            selectBox.appendChild(option);

        });

        const storedTowns = JSON.parse(localStorage.getItem('towns'));

        storedTowns.forEach(({ value, text }) =>
this.#townManager.addObj(new Town(value, text)));

        this.#townManager.getObjs().forEach(item => {

            const option1 = document.createElement('option');

            option1.value = item.Obj.value;

            option1.textContent = item.Obj.text;

            const option2 = document.createElement('option');

            option2.value = item.Obj.value;

            option2.textContent = item.Obj.text;

            let selectBox = document.getElementById('start-point');

            selectBox.appendChild(option1);

            selectBox = document.getElementById('finish-point');

            selectBox.appendChild(option2);

        });

        const storedTickets = JSON.parse(localStorage.getItem('tickets'));
```

```
        storedTickets.forEach(({
                start_place,
                end_place,
                start_date,
                finish_date,
                cost,
                type,
                purchased
        }) => this.#ticketManager.addObj(
                                                                new Ticket(
start_place,
end_place,
new Date(start_date),
new Date(finish_date),
cost,
type,
purchased,
                                                                        )));
        this.#isAutorized = JSON.parse(localStorage.getItem('isAutorized'));
        const header = document.getElementById('header-view-home');
```

```
        if (this.#isAutorized) {
            header.innerHTML = this.autorizedTemplate;
        } else {
            header.innerHTML = this.unautorizedTemplate;
        }
        this.#restoreMenuButton();
    }
```

Реализация MVVM-подхода

- **Модель (Model):**
  Отвечает за хранение и управление данными, включая бизнес-логику приложения. Например, используются менеджеры объектов ObjInfManager, которые обрабатывают операции с данными (создание, удаление, изменение) и обеспечивают взаимодействие с хранилищем данных (localStorage).
- **Представление (View):**
  Управляет отображением элементов интерфейса, таких как списки данных, формы авторизации и фильтры. Представление полностью изолировано от модели и взаимодействует с данными только через ViewModel. Динамически обновляет элементы интерфейса на основе изменений данных в модели.
- **Модель представления (ViewModel):**
  Реализует связь между моделью и представлением. ViewModel обрабатывает события, такие как навигация, авторизация, работа с формами, и предоставляет данные для отображения. Все изменения в модели автоматически отражаются в представлении благодаря использованию привязки данных.

Например, теперь вместо onclick="navigateTo('home-page')"

используется onclick="viewModel.navigateTo('home-page')"

```
class ViewModel {
    #currentPage;
    #filteredTickets;
    #ticketManager;
```

```
#townManager;
#ticketTypeManager;
#isAutorized;

constructor() {
        const home = document.getElementById('home-page');
        home.classList.add('active');
        this.#ticketTypeManager = new ObjInfManager('TicketType');
        this.#townManager = new ObjInfManager('Town');
        this.#ticketManager = new ObjInfManager('Ticket');
}

save() {
        const tickets = this.#ticketManager.getObjs().map(ticket => ({
                start_place: ticket.Obj.start_place,
                end_place: ticket.Obj.end_place,
                start_date: ticket.Obj.start_date.toISOString(),
                finish_date: ticket.Obj.finish_date.toISOString(),
                cost: ticket.Obj.cost,
                type: ticket.Obj.type,
                purchased: ticket.Obj.purchased,
        }));
        localStorage.setItem('tickets', JSON.stringify(tickets));
```

```
            const towns = this.#townManager.getObjs().map(town => ({
                value: town.Obj.value,
                text: town.Obj.text,
            }));
            localStorage.setItem('towns', JSON.stringify(towns));

            const ticketTypes = this.#ticketTypeManager.getObjs().map(ticketType => ({
                value: ticketType.Obj.value,
                text: ticketType.Obj.text,
            }));
            localStorage.setItem('ticketTypes', JSON.stringify(ticketTypes));

            localStorage.setItem('isAutorized', this.#isAutorized);
        }

        restore() {
            const storedTicketTypes = JSON.parse(localStorage.getItem('ticketTypes'));
            storedTicketTypes.forEach(({ value, text }) => this.#ticketTypeManager.addObj(new TicketType(value, text)));
            this.#ticketTypeManager.getObjs().forEach(item => {
                const option = document.createElement('option');
                option.value = item.Obj.value;
                option.textContent = item.Obj.text;
                let selectBox = document.getElementById('type');
```

```
            selectBox.appendChild(option);
        });
        const storedTowns = JSON.parse(localStorage.getItem('towns'));
        storedTowns.forEach(({ value, text }) =>
this.#townManager.addObj(new Town(value, text)));
        this.#townManager.getObjs().forEach(item => {
            const option1 = document.createElement('option');
            option1.value = item.Obj.value;
            option1.textContent = item.Obj.text;
            const option2 = document.createElement('option');
            option2.value = item.Obj.value;
            option2.textContent = item.Obj.text;
            let selectBox = document.getElementById('start-point');
            selectBox.appendChild(option1);
            selectBox = document.getElementById('finish-point');
            selectBox.appendChild(option2);
        });
        const storedTickets = JSON.parse(localStorage.getItem('tickets'));
        storedTickets.forEach(({
            start_place,
            end_place,
            start_date,
            finish_date,
            cost,
            type,
```

```
                purchased
        }) => this.#ticketManager.addObj(
                                                                new Ticket(

start_place,

end_place,

new Date(start_date),

new Date(finish_date),

cost,

type,

purchased,
                                                                        )));
        this.#isAutorized = JSON.parse(localStorage.getItem('isAutorized'));
        const header = document.getElementById('header-view-home');
        if (this.#isAutorized) {
                header.innerHTML = this.autorizedTemplate;
        } else {
                header.innerHTML = this.unautorizedTemplate;
        }
        this.#restoreMenuButton();
}
```

```
        navigateTo(formId) {
                document.querySelectorAll('.form').forEach(form => {
                        form.classList.remove('active');
                });
                const home = document.getElementById('home-page');
                if (formId === 'home-page') {
                        const header = document.getElementById('header-view-
browsing');
                        header.innerHTML = '';
                        const header_home = document.getElementById('header-view-
home');
                        if (this.#isAutorized) {
                                header_home.innerHTML = this.autorizedTemplate;
                        } else {
                                header_home.innerHTML = this.unautorizedTemplate;
                        }
                        this.#restoreMenuButton();
                        home.classList.add('active');
                        return;
                }
                if (formId === 'browsing') {
                        const header = document.getElementById('header-view-home');
                        header.innerHTML = '';
                        const header_browsing = document.getElementById('header-
view-browsing');
```

```
                if (this.#isAutorized) {
                        header_browsing.innerHTML = this.autorizedTemplate;
                } else {
                        header_browsing.innerHTML =
this.unautorizedTemplate;
                }
                this.#restoreMenuButton();
                const targetForm = document.getElementById(formId);
                targetForm.classList.add('active');
                this.#currentPage = 0;
                this.#findTickets();
                return;
        }
        const targetForm = document.getElementById(formId);
        if (targetForm) {
                home.classList.add('active');
                targetForm.classList.add('active');
        } else {
                console.error(`Form with id "${formId}" not found.`);
        }
    }


    ticketTemplate = `
  <div class="ticket_background">
    <div class="info_vb">
```

```
          <div class="info_container">
@@ -366,93 +405,258 @@ const ticketTemplate = `
          </div>
        </div>
    </div>`;


        #findTickets() {
                const ticket_properties = {};
                const startPlace = document.getElementById('start-point').value;
                if (startPlace !== 'none') {
                        ticket_properties["start_place"] = startPlace;
                }
                const endPlace = document.getElementById('finish-point').value;
                if (endPlace !== 'none') {
                        ticket_properties["end_place"] = endPlace;
                }
                const type = document.getElementById('type').value;
                if (type !== 'none') {
                        ticket_properties["type"] = type;
                }
                const startDate = document.getElementById('input_date_start').value;
                if (startDate !== '') {
                        ticket_properties["start_date"] = new Date(startDate);
                }
```

```
            const endDate = document.getElementById('input_date_finish').value;
            if (endDate !== '') {
                ticket_properties["finish_date"] = new Date(endDate);
            }
            const minCost = document.getElementById('input_price_min').value;
            if (minCost !== '') {
                ticket_properties["min_cost"] = minCost;
            }
            const maxCost = document.getElementById('input_price_max').value;
            if (maxCost !== '') {
                ticket_properties["max_cost"] = maxCost;
            }
            ticket_properties['purchased'] = false;
            this.#filteredTickets = this.#ticketManager.getObjs(0, 0,
ticket_properties);
            this.#renderTickets(this.#filteredTickets.slice(0, Math.min(3,
this.#filteredTickets.length)));
        }
        #renderTickets(tickets) {
            const ticketsContainer =
document.getElementById('ticket_container');
            ticketsContainer.innerHTML = '';
            const ticketsToRender = tickets.slice(0, 3);
            ticketsToRender.forEach(ticket => {
                const ticketElement = document.createElement('div');
                ticketElement.innerHTML = this.ticketTemplate;
```

```
                ticketElement.querySelector('.start-place').textContent =
this.#townManager.getObjByValue(ticket.Obj.start_place).Obj.text;

                ticketElement.querySelector('.end-place').textContent =
this.#townManager.getObjByValue(ticket.Obj.end_place).Obj.text;

                ticketElement.querySelector('.type-field').textContent =
this.#ticketTypeManager.getObjByValue(ticket.Obj.type).Obj.text;

                ticketElement.querySelector('.start-date').textContent =
ticket.Obj.start_date.toLocaleDateString(

                        'ru-RU', {

                                        day: '2-digit',

                                        month: '2-digit',

                                        year: 'numeric',

                                        hour: '2-digit',

                                        minute: '2-digit',

                        });

                ticketElement.querySelector('.end-date').textContent =
ticket.Obj.finish_date.toLocaleDateString(

                        'ru-RU', {

                                        day: '2-digit',

                                        month: '2-digit',

                                        year: 'numeric',

                                        hour: '2-digit',

                                        minute: '2-digit',

                        });

                ticketElement.querySelector('.cost').textContent =
ticket.Obj.cost + ' ₽';

                ticketsContainer.appendChild(ticketElement);
```

```
            });
        }
        prevPage() {
            this.#currentPage = this.#currentPage - 1;
            let new_start = this.#currentPage * 3;
            if (new_start < 0) {
                this.#currentPage = Math.floor((this.#filteredTickets.length - 1)
/ 3);
                new_start = this.#currentPage * 3;
                this.#renderTickets(this.#filteredTickets.slice(new_start,
Math.min(new_start + 3, this.#filteredTickets.length)));
            } else {
                this.#renderTickets(this.#filteredTickets.slice(new_start,
Math.min(new_start + 3, this.#filteredTickets.length)));
            }
        }
        nextPage() {
            this.#currentPage = this.#currentPage + 1;
            let new_start = this.#currentPage * 3;
            if (new_start >= this.#filteredTickets.length) {
                this.#renderTickets(this.#filteredTickets.slice(0, Math.min(3,
this.#filteredTickets.length)));
                this.#currentPage = 0;
            } else {
                this.#renderTickets(this.#filteredTickets.slice(new_start,
Math.min(new_start + 3, this.#filteredTickets.length)));
```

```
        }
    }

    #restoreMenuButton() {
        const button = document.getElementById('menu-button');
        const menu = document.getElementById('menu');
        let isHovering = false;
        button.addEventListener('mouseenter', () => {
            button.classList.add('hover');
            menu.classList.add('active');
            isHovering = true;
        });
        button.addEventListener('mouseleave', () => {
            isHovering = false;
            setTimeout(() => {
                if (!isHovering) {
                    button.classList.remove('hover');
                    menu.classList.remove('active');
                }
            }, 50);
        });
        menu.addEventListener('mouseenter', () => {
            button.classList.add('hover');
            menu.classList.add('active');
```

```
            isHovering = true;
        });
        menu.addEventListener('mouseleave', () => {
            isHovering = false;
            setTimeout(() => {
                if (!isHovering) {
                    button.classList.remove('hover');
                    menu.classList.remove('active');
                }
            }, 50);
        });
    }

    exit() {
        this.#isAutorized = false;
        localStorage.setItem('isAutorized', this.#isAutorized);
        const header = document.getElementById('header-view-home');
        header.innerHTML = this.unautorizedTemplate;
        this.#restoreMenuButton();
        document.querySelectorAll('input').forEach(input => {
            input.value = '';
        });
        document.querySelectorAll('select').forEach(select => {
            select.selectedIndex = 0;
```

```
        });
        this.navigateTo('home-page');
    }

    logIn() {
        const email = document.getElementById('login');
        let isMistake = false;
        if (email.value === '' || !Validator.validateEmail(email.value)) {
            email.style.border = '2px solid red';
            isMistake = true;
        } else {
            email.style.border = '2px solid green';
        }
        const password = document.getElementById('login-password');
        if (password.value === '' ||
!Validator.validatePassword(password.value)) {
            password.style.border = '2px solid red';
            isMistake = true;
        }
        if (isMistake) {
            return;
        }
        if (localStorage.getItem(email.value) != password.value) {
            password.style.border = '2px solid red';
            return;
```

```
            }
            email.style.border = 'none';
            email.value = '';
            password.style.border = 'none';
            password.value = '';
            this.#isAutorized = true;
            localStorage.setItem('isAutorized', this.#isAutorized);
            const header = document.getElementById('header-view-home');
            header.innerHTML = this.autorizedTemplate;
            this.#restoreMenuButton();
            this.navigateTo('home-page');
        }

        signIn() {
            const FIO = document.getElementById('FIO');
            let isMistake = false;
            if (FIO.value === '' || !Validator.validateFIO(FIO.value)) {
                FIO.style.border = '2px solid red';
                isMistake = true;
            } else {
                FIO.style.border = '2px solid green';
            }
            const passport = document.getElementById('passport');
            if (passport.value === '' || !Validator.validatePassport(passport.value))
{
```

```
                passport.style.border = '2px solid red';
                isMistake = true;
            } else {
                passport.style.border = '2px solid green';
            }
            const email = document.getElementById('e-mail');
            if (email.value === '' || !Validator.validateEmail(email.value)) {
                email.style.border = '2px solid red';
                isMistake = true;
            } else {
                email.style.border = '2px solid green';
            }
            const password = document.getElementById('password');
            if (password.value === '' ||
!Validator.validatePassword(password.value)) {
                password.style.border = '2px solid red';
                isMistake = true;
            } else {
                password.style.border = '2px solid green';
            }
            const password_repeat = document.getElementById('password-
repeat');
            if (password_repeat.value === '' ||
!Validator.validatePasswordRepeat(password.value, password_repeat.value)) {
                password_repeat.style.border = '2px solid red';
                isMistake = true;
```

```
} else {
        password_repeat.style.border = '2px solid green';
}
if (isMistake) {
        return;
}
FIO.style.border = 'none';
FIO.value = '';
passport.style.border = 'none';
passport.value = '';
email.style.border = 'none';
email.value = '';
password.style.border = 'none';
password.value = '';
password_repeat.style.border = 'none';
password_repeat.value = '';
if (localStorage.getItem(email.value) != null) {
        document.getElementById('login').value = email.value;
        this.navigateTo('log-in');
        return;
}
localStorage.setItem(email.value, password.value);
this.#isAutorized = true;
localStorage.setItem('isAutorized', this.#isAutorized);
```

```
            const header = document.getElementById('header-view-home');

            header.innerHTML = this.autorizedTemplate;

            this.#restoreMenuButton();

            this.navigateTo('home-page');

        }


        unautorizedTemplate = `

    <button onclick="viewModel.navigateTo('log-in')"
class="log_in_button">Вход</button>

        <button onclick="viewModel.navigateTo('sign-in')"
class="sign_in_button">Регистрация</button>

        <button id="menu-button" class="menu_button">Меню</button>

        <div class="header_underline"></div>

        <img onclick="viewModel.navigateTo('home-page')" class="logo_icon"
alt="Company logo" src="resources/Logo.png">

        <img class="log_in_icon" alt="Log in icon" src="resources/Log in.png">

        <img class="sign_in_icon" alt="Sign in icon" src="resources/Sign
in.png">`;


        autorizedTemplate = `

    <button onclick="viewModel.exit()" class="log_out_button">Выход</button>

        <button id="menu-button" class="menu_button">Меню</button>

        <div class="header_underline"></div>

        <img onclick="viewModel.navigateTo('home-page')" class="logo_icon"
alt="Company logo" src="resources/Logo.png">

        <img class="log_out_icon" alt="Log in icon" src="resources/Log
out.png">`;
```

```
}
```

Таким образом была добавлена возможность регистрироваться в систему и в дальнейшем заходить с соответствующими логин/пароль.

В каждом поле проводится соответствующая валидация введённых значений:

```
class Validator {
    static validateFIO(FIO) {
        const FIORegex = /^[A-Za-zA-яа-я]+$/;
        const parts = FIO.split(' ');
        let isValid = true;
        parts.forEach(part => {
            if (isValid) {
            isValid = FIORegex.test(part)
            }
        })
        return isValid && parts.length === 3;
    }
    static validatePassport(passport) {
        const passportRegex = /^[A-Za-z]{2}\d{7}$/;
        return passportRegex.test(passport.trim());
    }
    static validateEmail(email) {
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        return emailRegex.test(email.trim());
    }
    static validatePassword(password) {
        return password.trim().length >= 8;
    }
    static validatePasswordRepeat(password, password_repeat) {
```

```
            return password_repeat === password;
        }
}
```

Также был произведён рефакторинг кода для приведения его в “чистый вид”.

Результаты работы:

- Реализована обработка действий пользователя через контроллер.
- Добавлены методы сохранения и восстановления данных из LocalStorage.
- Приложение сохраняет изменения после перезагрузки страницы.
- Код структурирован в соответствии с MVVM.
- Использованы формы для работы с авторизацией, добавлением и редактированием данных.
- Обеспечено делегирование событий для динамического контента.

Вывод:

В ходе лабораторной работы была достигнута основная цель — создание динамического веб-приложения с использованием LocalStorage, событий и форм. Приложение соответствует MVVM-подходу и обеспечивает сохранение пользовательских данных. Работа выполнена в полном объеме в соответствии с заданием.

Все задачи выполнены в соответствии с требованиями задания.