ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

ФАКУЛТЕТ „ИНФОРМАТИКА“

Курсова работа

на тема

„Система за CRUD операции за уеб приложение за споделяне на пътувания“

по дисциплината Клиентско уеб програмиране

Изготвил:

Ния Димитрова

Фак. № 124157

Спец. МУТ, гр. 37

Проверил:

Гл. ас. д-р Б. Банков

Х. ас. Н. Цанков

ВАРНА 2025

Съдържание

1. Въведение

Целта на проекта е да се създаде система за CRUD операции за уеб приложението “You’ll come tool”. Уеб приложението представлява платформа за споделяне на информация и идеи за пътувания между приятели и познати. Функционалностите на уеб страницата включват създаване на потребителски профили, админ панел, през който могат да се изтриват потребители и да се манипулират вече създадени пътувания, качване на пълна информация за пътувания – дати, данни за полети, място на преспиване, транспорт и т.н., редактиране и изтриване на пътувания, списък с идващи, страница с всички пътувания, страница с пътувания, създадени от регистрираля се потребител, опция за търсене на конкретно пътуване по име. Уеб приложението използва SQLite база данни и фреймуорка Flask, за да обработва клиентските заявки и да съхранява данни.

Уеб приложението е SPA – Single Page Application, съответно зарежда различно съдържание в една и съща страница, която в конкретния случай е index.html. В клиентската част се използват библиотеката Page.js за по-лесно рутиране между страниците и Lit.html за по-лесно и бързо зареждане на съдържанието.

Системата за CRUD операции е слоят между потребителския интерфейс и сървърната част. Тя се състои от отделни JavaScript файлове, които са отговорни за зареждането на конкретна страница, зареждането на динамични данни от базата данни и изпращане на нови данни към базата. Тя е нужна, за да е възможна работата на приложението.

2. Структура на проекта

Структурата е както следва:

A. Папка .vscode – JSON конфигурационни файлове за стартиране на приложението
B. Папка back-end
    a. app.db – базата данни
    b. app.py – тук се съдържат всички функции, които обработват клиентските заявки, проверяват валидността на данните, модифицират базата данни
    c. storage.py – чрез този файл се създава базата данни и нейната структура, чрез изпълнение на отделни SQL команди
    d. Папка __pycache__ - кеширани данни
C. Папка front-end
    a. Папка images - изображения
    b. Папка node_modules – изтеглени модули
    c. Папка src
        i. Папка views – тук се съдържат всички файлове, които променят съдържанието на страницата и изпращат HTTP заявки
            1. addTrip.js – код на страницата за добавяне на ново пътуване
            2. admin.js – код на админ панела
            3. congrats.js – код на страницата, след присъединяване към пътуване
            4. dashboard.js – код на страницата с всички пътувания
            5. details.js – код на страница с детайли за конкретно пътуване

6. edit.js – код на страница за редактиране на конкретно пътуване
7. home.js – код на началната страница
8. login.js – код на страницата за вписване на потребител
9. myTrips.js – код на страницата на пътуванията на текущо вписания потребител
10. register.js – код на страницата за регистриране на нов потребител
11. search.js – код за извеждане на всички пътувания, чиито имена отговарят на заявката за търсене

ii. app.js – тук се рутират всички изгледи и се управляват бутона за излизане от профила и полето за търсене

iii. lib.js – тук се импортират библиотеки и се експортират с по-кратки имена нужните функции

iv. util.js – функции за управление на потребителски данни и управление на лентата за навигация

d. Папка styles – тук се съдържат всички CSS файлове за отделните страници

e. package-lock.json

f. package.json

D. index.html – главната страница, където се зарежда съдържанието

3. Преглед на използваните функции за извършване на CRUD операции

3.1 Метод submitTrip(e)

```
const userId = JSON.parse(localStorage.getItem('user')).userId;

let bdc = new Date(beginDate);
let edc = new Date(endDate);

if(edc < bdc){
    return alert('End date must be after the begin date!')
}

let tripObj = {
    destination,
    beginDate,
    endDate,
    goingCity,
    goingDeparture,
    goingArriveCity,
    goingArrival,
    goingPrice,
    returnCity,
    returnDeparture,
    returnArrivalCity,
    returnArrival,
    returnPrice,
    transportName,
    transportPrice,
    accomodationPlace,
    accomodationPrice,
    accomodationLink,
    userId
};

const formData = new FormData();

for (let key in tripObj) {
    if(!tripObj[key]){
        return alert('Fill all fields!');
    }

    formData.append(key, tripObj[key]);
}

if (tripImage && tripImage.files.length > 0) {
    formData.append('tripImage', tripImage.files[0]);
}
```

Фиг. 1 – Записване на данните от формулярите в обект и проверка на полетата

```
    const options = {
        method: 'POST',
        body: formData
    }

    const url = 'http://127.0.0.1:5001/add';

    try {
        const response = await fetch(url,options);

        const result = await response.json();
        page.redirect('/');


    } catch (error) {
        alert(error);
    }
}
```

Фиг. 2 – Изпращане на данните към сървъра

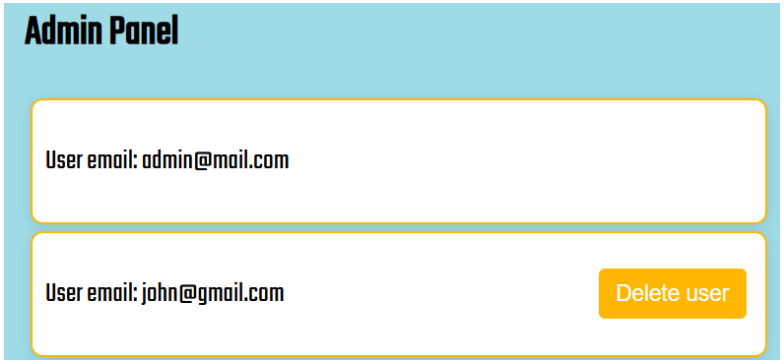
Методът submitTrip се извиква при натискане на submit бутона на формуляра на страницата за добавяне на ново пътуване. Функцията приема един аргумент – събитието на натискането. Първо функцията взима и запазва в константи данните от четирите формуляра – basicsForm, flightForm, transportForm, accomodationForm. Данните се взимат от FormData обекти, които са създадени чрез формулярите. След тяхното запазване и проверка на валидността на датите, данните се запазват в нов обект – tripObj. Прави се проверка дали всички полета са попълнени. Създава се нов FormData обект, в който се прехвърлят данните от полетата и каченото изображение за корица на пътуването. Данните се изпращат към сървърната част чрез POST заявка и потребителят бива прехвърлен на началната страница.

3.2 Метод deleteUser(e)

```
async function deleteUser(e){
    e.preventDefault();

    let userId = e.target.dataset.userid;

    try{
        const response = await fetch(`http://127.0.0.1:5001/deleteUser/${userId}`, {
            method: 'DELETE'
        });
        const data = await response.json();

        if(response.ok){
            showAdminView();
        }
    }catch(e){
        return alert(e)
    }

}
```

Фиг. 3 – Методът deleteUser

Методът deleteUser служи за изтриване на потребител чрез админ панела. Първо взима userId от HTML-а на страницата. При зареждане на админ панела се изреждат всички потребители един под друг с отделни бутони за изтриване. В dataset атрибутите на бутоните се съдържа ID на конкретния потребител, освен ако не е админ. ID-то на избрания потребител се добавя към линк и се изпраща DELETE заявка към сървъра. В случай, че изтриването е успешно, страницата се презарежда.



Фиг. 4 – Админ панела

3.3 Метод showDashboardView

```
export async function showDashboardView(){
    try{
        const response = await fetch('http://127.0.0.1:5001/getAllTrips');
        const data = await response.json();
        render(dashboardTemplate(data.data));
    }catch(error){
        return alert(error);
    }

}
```

Фиг. 5 – Методът showDashboardView

Методът showDashboardView изпраща GET заявка към сървъра. След това подава получените данни на функцията, която създава dashboard шаблона, чрез библиотеката lit-html. След това тази функция се подава на функцията render, която закача съдържанието в главната страница. В случай на грешка се извежда съобщението на грешката.

3.4 Метод showDetailsView(ctx)

Методът showDetailsView(ctx) приема един аргумент – контекст, чрез който взима tripId от адреса на страницата. Изпраща GET заявка към сървъра, за да получи данни за избраното пътуване. В случай, че в local storage има запазени данни за потребител, те биват запазени в променливи. Извършва смятане на пълната цена на пътуването с част от получените данни. След това изпраща втора GET заявка, за да получи данни за списъка от отиващи на пътуването потребители. Проверява дали текущо влезлият потребител е в списъка. Рендира

секцията в главната страница чрез шаблон, на който се подава списък с нужните аргументи. Накрая се добавят event listeners на бутоните за изтриване на пътуване, за присъединяване към списъка на отиващите и бутонът за копиране на линк.

```
export async function showDetailsView(ctx) {
    const tripId = ctx.params.tripId;
    tripIdg = tripId;
    const url = 'http://127.0.0.1:5001/details/' + tripId;
    try {
        const response = await fetch(url);
        const data = await response.json();
        let userId = 0;
        let isAdmin;

        if (localStorage.getItem('user')) {
            const user = JSON.parse(localStorage.getItem('user'));
            userId = user.userId;
            isAdmin = user.isAdmin;
            userIdg = userId;
        } else {
            userId = null;
        }

        let fullPrice =
            (parseFloat(String(data.data.goingFlight.price)) || 0) +
            (parseFloat(String(data.data.returnFlight.price)) || 0) +
            (parseFloat(String(data.data.extraTransportDetails.price)) || 0);

        aPrice = (parseFloat(String(data.data.accomodation.price)) || 0);
        fPrice = fullPrice;

        const response2 = await fetch(`http://127.0.0.1:5001/getGoingTrip/${tripId}`);
        const goingList = await response2.json();
        let userIdHasJoined = goingList.data.some(x => x.userId === userId);

        render(detailsTemplate(data, tripId, userId, fullPrice, calc, isAdmin, goingList, userIdHasJoined));

        if(document.getElementById('deleteBtn')){
            document.getElementById('deleteBtn').addEventListener('click', deleteTrip);
        }

        if( document.getElementById('addGoingBtn')){
            document.getElementById('addGoingBtn').addEventListener('click', toggleGoing);
        }

    } catch (error) {
        return alert(error);
    }
```

Фиг. 6 – Метод showDetailsView

3.5 Метод deleteTrip(e)

Методът deleteTrip използва глобалната променлива tripIdg (tripIdGlobal) и я добавя към URL-а, на който изпраща DELETE заявка. В случай, че е успешна, потребителят се пренасочва на актуализираната страница с всички пътувания.

3.6 Метод toggleGoing(e)

Методът toggleGoing приема един аргумент – събитието, чрез което е извикан. Първо проверява дали бутонът за присъединяване към списъка на идващите има клас „going“. В зависимост дали има, изпраща DELETE или POST заявка към съответния URL. Ако потребителят напуска списъка на отиващите, бива пренасочен към страницата с всички пътувания. В случай, че се присъединява, бива пренасочен към поздравителна страница. И в двата случая бутонът бива променен. В session storage се запазва tripId, за да може след пренасочване към поздравителната страница да се разбере към кое пътуване се е присъединил потребителят.

```
async function toggleGoing(e) {
    e.preventDefault();
    const isGoing = document.getElementById('addGoingBtn').classList.contains('going');

    try {
        const response = await fetch(
            isGoing
                ? `http://127.0.0.1:5001/removeGoingTrip/${tripIdg}/${userIdg}`
                : `http://127.0.0.1:5001/addGoingTrip/${tripIdg}/${userIdg}`,
            {
                method: isGoing ? 'DELETE' : 'POST',
                headers: {
                    "Content-Type": "application/json",
                },
            }
        );

        if (response.ok) {
            const goingList = response.json();

            const button = document.getElementById('addGoingBtn');
            if (isGoing) {
                button.textContent = "Will you join them?";
                button.classList.remove('going');
                return page.redirect('/alltrips');
            } else {
                button.textContent = "Leave the trip";
                button.classList.add('going');
                page.redirect('/congrats');
                sessionStorage.setItem('cTripId', tripIdg);
            }

        } else {
            console.error("Failed to toggle going status");
        }

    } catch (error) {
        console.error("Error in toggleGoing:", error);
    }
}
```

Фиг. 7 – Метод toggleGoing

3.7 Метод updateTrip(e)

Методът updateTrip извършва същите операции като метода submitTrip, с разликата, че данните във формулярите изначално се зареждат чрез GET заявка. След като потребителят редактира пътуването, всички данни се изпращат към сървъра с PUT заявка, за да може само променените данни да се актуализират.

```
    for (let key in tripObj) {
        if(!tripObj[key]){
            return alert('Fill all fields!');
        }

        formData.append(key, tripObj[key]);
    }

    if (tripImage && tripImage.files.length > 0) {
        formData.append('tripImage', tripImage.files[0]);
    }

    const options = {
        method: 'PUT',
        body: formData
    }

    const url = 'http://127.0.0.1:5001/edit/' + tripIdGl;

    try {
        const response = await fetch(url,options);

        const result = await response.json();
        page.redirect('/');


    } catch (error) {
        alert(error);
    }
}
```

Фиг. 8 – Кодът за проверка на данните и за изпращане на PUT заявка

3.8 Метод signup(e)

Методът signup приема като аргумент само събитието, чрез което е извикан. В случай, че в local storage няма данни за user, се създава обект user с полета firstName, lastName, email и password, като техните стойности са undefined. Ако има потребителски данни в local storage, то тогава те се запазват в променливата user. Взимат се стойностите на полетата за парола и за повторно изписване на парола. Проверява се дали полето за паролата не е оставено празно и дали двете пароли съвпадат. Проверява се дали са приети условията за ползване и

поверителност. В обекта се записват стойностите на полетата firstname, lastname, email и password. Изпраща се POST заявка с обекта към сървъра. В случай, че няма грешка, нужните данни за новия потребител се записват в local storage, лентата за навигация се обновява и потребителят се препраща на началната страница.

```
    const url = "http://127.0.0.1:5001/signup"
    const options = {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify(user)
    };

    try {
        let response = await fetch(url, options);

        let result = await response.json()

        if (!response.ok) {
            throw new Error(result.error)
        } else {
            localStorage.setItem('user', JSON.stringify({ 'email': result.data.email, 'userId': result.data.userId }))
            updateNav();
            page.redirect('/');
        }

    } catch (error) {
        alert(error)
    }
}
```

Фиг. 9 – Изпращане на POST заявката и записване на получените данни в local storage

3.8 Метод signin(e)

Методът signin приема един аргумент – събитието, чрез което е извикан. Първо записва в променлива референция към полето за грешка и взима имейла и паролата от input полетата. Проверява дали не са празни. В случай, че не са, ги записва в обект user и изпраща обекта към сървъра с POST заявка. При успешно вписване, сървърът връща имейл, userId и булева променлива isAdmin. Проверява се дали текущо вписаният потребител е админ. Ако е, се прехвърля към админ панела и в local storage се запазват имейлът, userId и променливата isAdmin със стойност true. Ако потребителят не е админ, в local storage се запазват само имейл и userId, лентата за навигация се обновява и потребителят се прехвърля на началната страница. В случай на грешка, съобщението за грешка се показва и цветът му се сменя на червен.

3.9 Метод showMyDashboardView(ctx)

Методът showMyDashboardView взима потребителските данни, записани в local storage и записва userId в променлива. След това изпраща GET заявка към сървъра, като включва и userId в URL-а. Накрая подава получените данни на функцията, която зарежда шаблона на страницата и след това тази функция се подава на функцията render, която зарежда съдържанието в главната страница.

3.10 Метод showSearchView(ctx)

Методът showSearchView приема един аргумент – контекст, чрез който взима заявката, по която трябва да търси, от URL-а. Изпраща GET заявка към сървъра. Презарежда dashboard страницата само с пътуванията, които отговарят на заявката.

Фиг. 10 – Данни, които връща сървъра при успешно изпращане на заявката

4. Заключение

Уеб приложението има пълен набор от функции, за да извършва нормалната си дейност. Могат да се изпращат всички видове заявки към сървъра. От сървъра може и да се получават данни, които биват правилно обработвани и визуализирани. Ефектът на заявките е дълготраен, понеже данните се пазят в локална база данни.