ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

ФАКУЛТЕТ „ИНФОРМАТИКА“

НАПРАВЛЕНИЕ „ИНФОРМАТИКА И КОМПЮТЪРНИ НАУКИ“

Курсова работа

на тема

„Платформа за споделяне на снимки “DogTerest”“

по дисциплината Клиентско уеб програмиране

Изготвил:

Диана Десиславова Симеонова

фак. № 124125 спец. МУТ, гр. 37, 3 курс

Проверил:

Гл. ас. д-р Б. Банков

х.ас. Николай Цанков

ВАРНА 2025

Съдържание

1. Въведение и цел на проекта

Идеята на проекта “DogTerest” е да съчетае front-end и back-end технологии за изграждането на малка платформа за споделяне на снимки. В своята същност, тя е Single-Page Application (SPA) – динамично зареждаща се уеб страница. Реализацията се базира на комбиниране на JavaScript библиотеки за клиентската част на уеб приложението и Python framework за Application Programming Interface (API) за изпращане и приемане на информация към базата данни на проекта. Крайната версия на уеб приложението съдържа оптимизирани функционалности за зареждане на съдържание в браузъра, които могат да бъдат преизползвани с минимални промени според нуждите на бъдещи проекти.

2. Реализация на проекта

За целта на приложението е изграден API, осъществяващ обмен на данни между двете страни на уеб приложението на име “DogTerest”.

a. Front-end

Имплементацията на front-end частта включва използването на lit-html, което е лека JavaScript библиотека, позволяваща манипулиране на DOM дървото на уеб страниците, и page.js - библиотека, която изгражда проекта като SPA чрез рутиране между отделните html файлове за навигация и пренасочване от една страница в друга. Основно използвани функции от библиотеките в проекта са както следва:

1) lit-html
   - render(Template(params), main) – зареждане на lit-html код от променлива за шаблон Template, на която се подават необходими функции или променливи за запълване на шаблона, и main – основната HTML структура на проекта

- html – интерпретатор на template литерал, който превръща подадения код в HTML структура, която ще запълни съдържанието на уеб страницата;

2) page.js
   - page(url, pageView) – на подадения URL да се зареди подадения template
   - page.redirect(url) – пренасочване на текущата страница към друга страница от проекта
   - page.reload() – презареждане на текущата страница

b. Back-end (API)

За изграждане на API бе използван Python framework на име Flask. Предназначен за малки уеб приложения, той предоставя полезни инструменти и методи за създаване и управление на уеб съдържание. За изграждане на базата от данни е използван sqlite3 – чрез SQL заявки бяха изградени таблиците и обработката на данните във Flask. Основно използвани функции от библиотеките в проекта са както следва:

1) Flask
   - @app.route(url, methods=[]) – рутиране на различните крайни точки (endpoints) за изпращане и получаване на данни
   - Response – клас за генериране на отговори към HTTP заявки; определя статус, body, headers
   - Jsonify – сходно на JSON.stringify, превръща подадения му текст в JSON обект и Response за предаване към отговор на HTTP заявка
2) Sqlite3
   - Sqlite3.connect(filePath) – създава връзката с базата данни на подадения filePath
   - Cursor() – подаване на курсор, който осъществява връзката с базата данни и извършва всички заявки, които му подаваме

- Cursor.execute(query) – изпълнява подадената му заявка по базата данни
- Connection.commit() – ако сме правили промени по базата данни, то те се запаметяват

Проектът се стреми да изгради функциониращо уеб приложение в малък мащаб. Основната цел е да установи преизползваеми модули за бъдещи проекти, чиито функционалности са сходни с текущо изградените.

3. Структура на проекта

Проектът е разделен на основни директории, които разделят неговите части на front-end и back-end.

1. Front-end
    a. Constants – два основни шаблона, които се преизползват многократно за запълване на съдържанието на проекта;
    b. Styles – отделни CSS файлове със стилове за различни части на уеб съдържанието;
    c. Views – всички файлове описват различните страници в уеб приложението – шаблоните и техните функционалности;
    d. Отделени файлове
        a. Api.js – файл, описващ основна структура на една HTTP заявка;
        b. App.js – основен файл, където се рутира приложението
        c. Auth.js – функции за автентикация в приложението
        d. Functions.js – множество функции за извикване на функциите от api.js за достъп до back-end API чрез endpoints
        e. Index.html – базовия скелет на уеб приложението
        f. Util.js – отделени функции за боравене с user data, submit handlers, toast notifications
2. Back-end

a. App.py – основен файл, където е описано рутирането на endpoints и съответните им функционалности
b. Storage.py – описанието на таблиците на базата данни и тяхното създаване
c. App.db – файлът, съдържащ базата от данни

3. Imgs – добавените снимки, които се визуализират в приложението

4. Преглед на функциите

4.1. Util.js

```
const itemName = "userData";

export function getUserData() {
    return JSON.parse(localStorage.getItem(itemName))
}

export function setUserData(data) {
    return localStorage.setItem(itemName, JSON.stringify(data))
}


export function clearUserData() {
    localStorage.removeItem(itemName)
}

export function createSubmitHandler(callback) {
    debugger
    return function (event) {
        event.preventDefault();
        const form = event.currentTarget;
        const formData = new FormData(form);
        const data = Object.fromEntries(formData.entries());

        callback(data, form);
    }
}
```

Figure 1. Функции за боравене с потребителски данни и данни от формуляр

```
const toast = document.querySelector('.toast');
export function showToast(message, type = "success") { // ако няма подаден тип, по default е "success"
    toast.style.display = 'block';
    toast.classList.add('show');

    toast.textContent = message;

    if (type === "error") toast.style.background = "#f44336";

    setTimeout(() => {
    toast.classList.remove('show');
    toast.style.display = 'none';
  }, 4000); // 4000ms to match the animation duration
}
```

Figure 2. Toast известия, изчезващи след 4 секунди

Функционалностите в util.js са най-често преизползваните в целия проект. При тях наблюдаваме бораване с данните на текущия потребител (тяхното

установяване, взимане и премахване от браузъра), централизирано обработване на данни от формуляр чрез FormData, както и функция за toast известия, които отразяваме при определени действия в уеб страницата.

4.2. App.js

Основната страница, където се базира рутирането на уеб сайта. Импортирани са всички функции, които описват шаблон за всяка една страница в сайта.

```
function decorateContext(ctx, next) {
    renderView();
    next();
}

function renderView() {
    const userData = getUserData();
    render(layoutTemplate(userData), root)
}

function logoutAction() {
    logout();
    page.redirect('/home');
}
```

Figure 3. Основно зареждане на уеб страницата и премахване на потребителски данни при logout

По-важни функционалности, които може да отбележим:

- decorateContext() - "декорира контекста", т.е. запълва базовото съдържанието на страницата чрез помощната функция renderView() и чрез next() извиква следващата по ред функция/елемент да се зареди;
- renderView() – установява наличните данни на потребителя (дали има log-нат потребител, неговите права) в променлива, която се подава на layoutTemplate() – основата на уеб страницата (навигационно поле), и заедно с това подава и root, което е header таг на страницата към функцията от page.js render, която зарежда основния шаблон

- logoutAction() – извиква изпълнението на функцията logout(), която премахва потребителските данни, и пренасочва уеб страницата към home page

4.3. Auth.js

```
import { setUserData, clearUserData } from './util.js';
import { post } from './api.js';

const endpoints = {
    login: '/signin',
    register: '/signup',
};

export async function login({ email, password }) {
    const userData = await post(endpoints.login, { email, password });

    setUserData(userData.data);
}

export async function register(user) {
    await post(endpoints.register, user);
}

export function logout() {
    clearUserData();
}
```

Figure 4. Основни функции за регистрация и вписване на потребител

Описани са основните функционалности за автентикация на потребител в сайта. Има възможност да създаде профил, да се впише в съответния профил, както и да излезе от него.

4.4. Api.js

```
import { clearUserData } from "./util.js";

const host = 'http://localhost:5001'

async function request(method, url, data) {
    const options ={
        method,
        headers: {}
    };

    if (data instanceof FormData) {
        options.body = data;
    }
    else if (data !== undefined) {
        options.headers['Content-Type'] = 'application/json';
        if (method !== 'GET' && method !== 'HEAD') {
            options.body = JSON.stringify(data);
        }
    }
    try {
        const response = await fetch(host+url, options);

        let result;
        if(response.status != 204) {
            result = await response.json();
        };
        if (response.ok == false) {
            if(response.status == 403) {
                clearUserData();
            }
            const error = result;
            throw error;
        }

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

Figure 5. Структура на функция за HTTP заявки на front-end

Универсален скрипт, който обработва HTTP заявки към back-end**.** Задава body и headers според типа заявка, която изпращаме**.** Обработва отговора според получения статус от back-end**.** В края на ск**ри**пта има 5 различни експорта, които са bind-нати стойности към заявката, по един за различните типове HTTP заявки. По този начин, според типа, ще се запълнят данните в скрипта коректно и ще се подготвят за изпращане.

4.5. Functions.js

Файл, където са описани множество функции, които служат за комуникация със специфични функционалности от back-end. Приложени са основни функционалности за боравене с потребител/и – взимане и обновяване на информацията им, изтриване на конкретен потребител; боравене със снимките в сайта – харесване, добавяне в любими, и др, както и тяхното достъпване по ID от базата данни.

```
export async function updateUser(user) {
    const result = await patch(endpoints.updateUser, user);
    return result;
}

export async function deleteUser(id) {
    return await del(endpoints.deleteUser, id);
}
```

Figure 6. Обновяване и изтриване на потребител, функции от API

```
export async function uploadPicture(data) {
    return await post(endpoints.uploadPicture, data)
}

export async function getPictureById(id) {
    return await get(endpoints.getPicturebyId + id);
}
```

Figure 7. Качване на снимка и взимане на данни за снимка по нейно ID

4.6. Constants/imageTemplate.js и likesTemplate.js

Двата посочени файла описват два шаблона, които се преизползват многократно в реализацията на уеб страницата. Целта е да не се преповтаря код, а да се изнесе в отделна функция, която взима обект с нужната информация, с която ще запълни шаблона и съдържанието на страницата.

```
import { html } from '../../node_modules/lit-html/lit-html.js'

export const imageTemplate = (image) => html`
<div class="item">
            <div class="content">
                <img src="/${image.uploaded_image_url}" alt="Post Image" />
                <div class="overlay">${image.description}
                    <a href="/details/${image.image_id}">See more...</a>
                </div>
            </div>
        </div>`
```

Figure 8. Шаблон за пост на снимка

```
export const likeTemplate = (like) => {
    const date = like.created_at;
    const dateObj = new Date(date.replace(" ", "T"));
    const formattedDate = `${dateObj.getDate()}.${dateObj.getMonth() + 1}`;

    return `
      <div class="like-entry">
        <p>
          <a href="/profile/${like.user_id}">${like.user_first_name} ${like.user_last_name}</a> liked this
        </p>
        <span class="like-date">${formattedDate}</span>
      </div>
    `
};
```

Figure 9. Шаблон за прозорец на харесвания на снимка

4.7. Views/layout.js

Основният layout на уеб страницата, където се определя какви елементи ще се визуализират на навигационната лента**.** Според наличието на userData (информация за текущия потребител) и нейните специфики може да имаме

- Основни бутони за Home и Dashboard, Login и Register
- Бутони за Profile
- Бутони за Admin Panel/Admin Dashboard

```
import { html } from '../../node_modules/lit-html/lit-html.js'
export const layoutTemplate = (userData) => html`
<nav>
<div class="logo">
    ${userData ? html `<a href="/dashboard">
        <img src="/imgs/yorkshire-terrier-dog-head-hand-drawn-c0475e.webp" alt="Logo" />
    </a>` : html`<a href="/home">
        <img src="/imgs/yorkshire-terrier-dog-head-hand-drawn-c0475e.webp" alt="Logo" />`
    }
</div>
<div>
<a href="/home">Home</a>
<a href="/dashboard">Dashboard</a>
${userData
        ? html`
    <a href="/current_user_profile">Profile</a>
</div>
<div class="user">
    <a href="/logout">Logout</a>
    ${userData.is_admin === 1 ?
        html`<div class="admin"> <a href="/admin">Admin Panel</a>
        </div>
        <div class="admin"> <a href="/admin_dashboard">Admin Dashboard</a>
        </div>` : null}

        : html`
        <a href="/login">Login</a>
        <a href="/signup">Register</a>

    }
    </div>
</nav>`
```

Figure 10. Шаблон за въвеждаща страница

4.8. Login/Signup.js

```
export function loginPage() {
    render(loginTemplate(createSubmitHandler(onLogin)), main);

    async function onLogin() {
        const email = document.getElementById('email').value;
        const password = document.getElementById('password').value;

        if (email === "" || password === "") {
            return alert("All fields are required");
        }
        try {
            await login({ email, password });
            showToast("Login successful");
            page.redirect("/dashboard");
        } catch (error) {
            alert(error.message);
            showToast(error.message, "error");
```

Figure 11. Функционалност на страница за вписване

Функционалностите за login и signup са сходни. Чрез създаване на шаблон (loginTemplate, signupTemplate) визуализираме формуляр за вписване на потребителски данни, който чрез submit event изпраща данните към асинхронните функции onLogin и onRegister. Те събират информация, подават я на съответните функции от auth.js, където ще се изпратят backend.

4.9. Views/dashboard.js

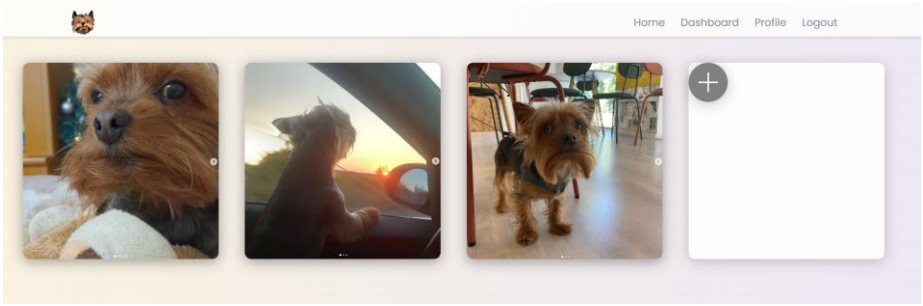

Figure 12. Основно табло

При успешно вписване, уеб сайтът ни препраща към Dashboard, където се визуализират всички снимки, които са запазени в базата данни. За тяхното представяне е използван шаблона imageTemplate.js. Последния блок ни дава възможност да добавим собствена снимка, която веднага ще се появи на екрана;

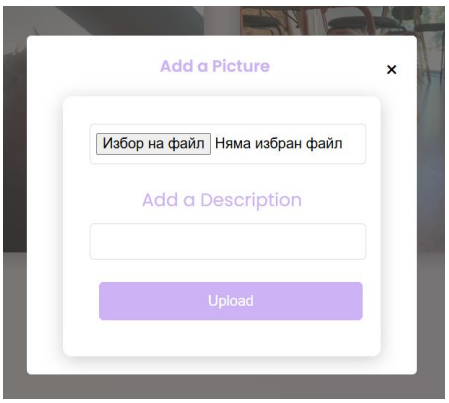

Figure 13. Формуляр за качване на нова снимка

- Чрез input поле от тип file, което е програмирано да приема единствено снимки, можем да качим снимка от собственото си устройство;
- Добавяме и описание за снимката и след натискане на бутон Upload тя се представя на таблото и излиза съобщение за успешно добавена снимка

4.10. Views/Details.js

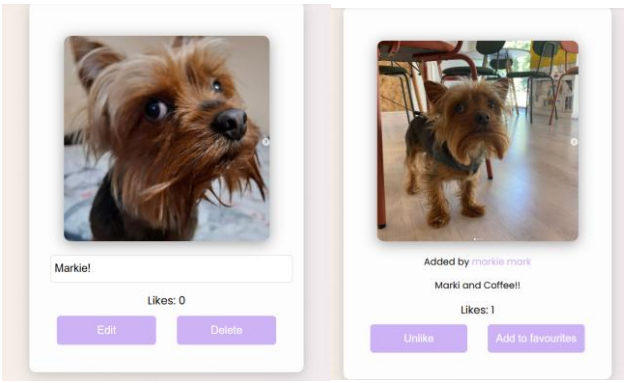

Figure 14. Изглед на снимка (потребител-създател/несъздател)

При влизане на Details за всяка една снимка се визуализира конкретното изображение, поле с описанието му, колко харесвания има и два бутона, които могат да са различни според правата на текущия потребител:

- Edit/Delete – ако потребителя е собственик на снимката, той ще може да променя нейното описание и да я изтрие

- Like/Unlike и Add to favourites/Remove from favourites – ако потребителят не е собственик на снимката, той може само да я хареса или да я добави в своите любими снимки; при него също така излиза поле над описанието с линк към профила на потребителя, който е собственик на снимката

```
async function onEdit() {
    const toEdit = document.getElementById("description").value;
    const data = { toEdit };
    try {
        await editPicture(id, data);
        console.log('Updated picture');
        showToast("Picture edited successfully", "success");
        page.redirect('/dashboard');
    } catch (e) {
        alert(e.message);
        console.log('Error updating item:', e);
    }
}

async function onDelete() {
    const choice = confirm('Are you sure you want to delete?');
    if (choice) {
        await deletePicture(id);
        showToast("Picture deleted successfully", "success");
        page.redirect('/dashboard');
    }
}
```

Figure 15. Функционалност за редактиране и изтриване на снимка

Когато натиснем текста Likes, се отваря прозорец, който показва кой потребител кога е харесал снимката. За тази цел се използва шаблона likesTemplate.js**.** Можем да достъпим профила на всеки един потребител, който е харесал снимката.

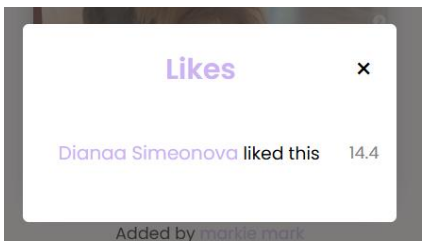

Figure 16. Прозорец, визуализиращ харесвания на снимка

4.11. Views/currentUserProfile.js и profile.js

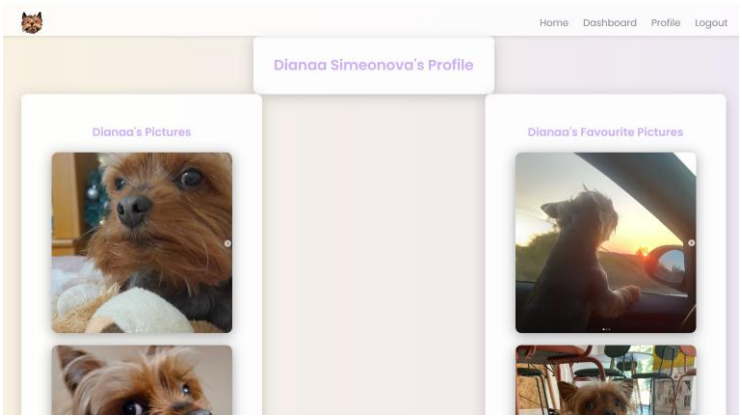

Figure 17. Страница на текущия потребител

При натискане на бутона Profile от навигационното поле, уеб страницата се изпълва с информация за текущия потребител: неговото име, всички добавени от него снимки, както и любимите му снимки**.** Също така има възможност потребителя да редактира своя профил.

Ако достъпим нечии профил от детайлите на някое от изображението, ще получим съдържание, което е сходно на гореописаното**.** Разликата тук е, че единствено може да разгледаме собствените снимки на съответния потребител, както и любимите му снимки; не можем да редактираме част от неговия профил;

4.12. Views/admin.js и adminDashboard.js

При вписан потребител с права на админ имаме допълнителни възможности. В навигационното поле имаме два нови бутона за страници, където се разгръщат разширените права на потребителя.

- Admin Panel – представя формуляр, където са заредени всички потребители, които съществуват в платформат**а**
  - Ако селектираме някой от дадените профили, автоматично ще се зареди неговата информация в полетата – ID, име, фамилия, имейл, права;
  - Администраторът може да промени което и да е от съответните полета и при натискане на Edit ще се обнови информацията в базата данни; при натискане на Delete ще се изтрие съответния профил;

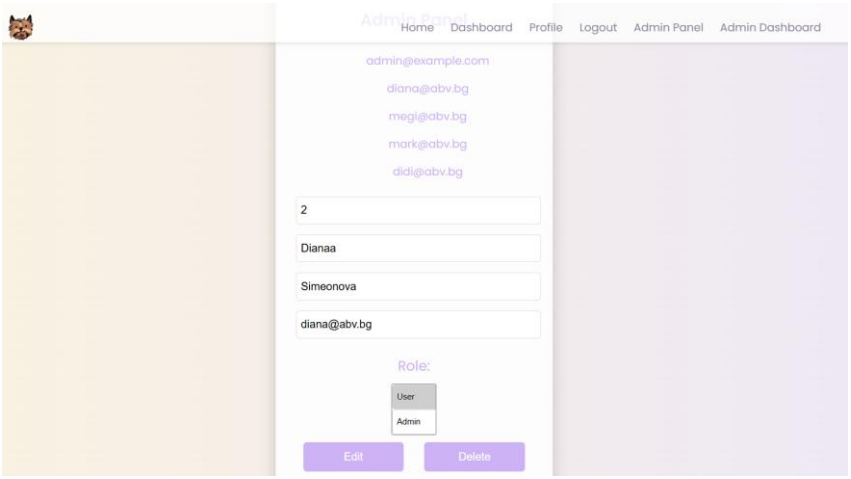

Figure 18. Admin Panel

- Admin Dashboard – страница, която показва основна информация за платформата

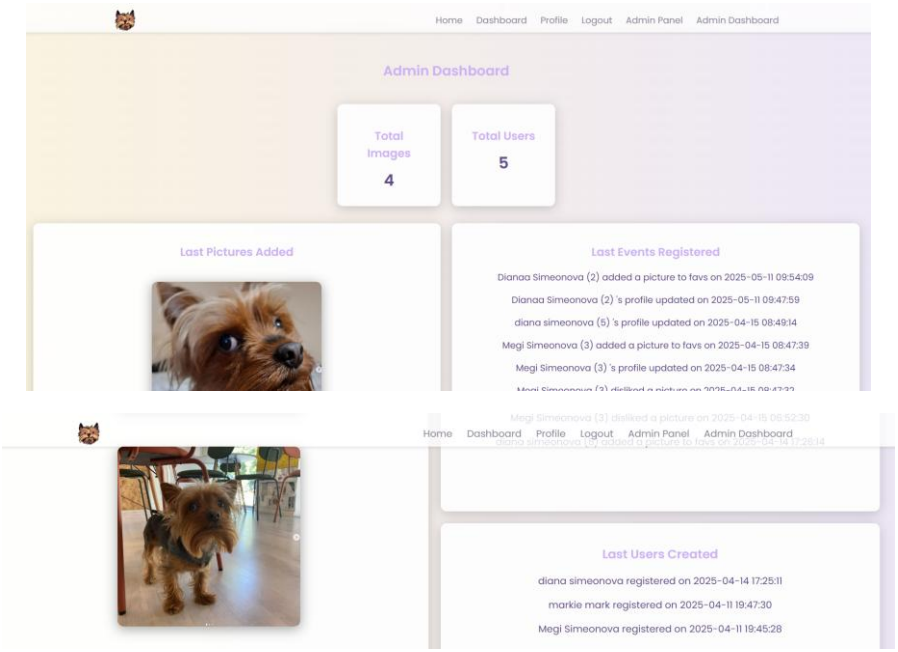

Figure 19. Admin dashboard

- Total Images – брой на всички снимки в платформата
- Total Users – брой на всички потребители в платформата
- Last Pictures Added – последните 3 снимки, добавени в платформата – достъпни са по сходен начин, както в Dashboard страницата

- Last Events Registered – информация за последните събития, които са отчетени в платформата с дата и информация на съответния потребител с неговото ID
- Last Users Created – показва последните трима потребителя, които са създали профил в платформата

4.13. Toast notifications

Отчетените събития в Admin Dashboard са отразени пред потребителя чрез своеобразни toast известия, които стоят на уеб страницата в продължение на 4 секунди. Някои примери за тяхното предизвикване:

Figure 20. Toast notifications

5. Заключение

Реализираното уеб приложение „DogTerest“ изгражда шаблонен модел на платформа за качване на снимки. Основните му функционалности за работа с потребители, визуализиране на съхранени в база данни снимки, както и допълнителна информация за потребителите и събитията в уеб страницата, работят в контекста на проекта и имат възможност да бъдат преизползвани в други бъдещи уеб страници. Използваните технологии са леки и разбираеми за начинаещи, които изучават основите на клиентското уеб програмиране. С развитието на знанията, платформата може да се разшири с множество нови възможности, които също да надградят знанията по програмния език JavaScript и да я направят по-интерактивна.