Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Міллєра Микити Ігоровича*
(ПІБ)

академічної групи *121-19-2*
(шифр)

напряму підготовки *121 Інженерія програмного забезпечення*
(код і назва напряму підготовки)

на тему: *Розробка онлайн-сервісу бронювання квитків на культурні заходи за допомогою React.JS у середовищі програмування Visual Studio Code*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*проф. Мещеряков Л.І..*			
розділів:				
спеціальний				
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*ст. викл. Мартиненко А.А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 20 23 року

ЗАВДАННЯ
на дипломний проект

бакалавра
(назва освітньо-кваліфікаційного рівня)

студенту *121-19-2* *Міллєру Микиті Ігоровичу*
(група) (прізвище та ініціали)

Тема дипломного проекту *Розробка онлайн-сервісу бронювання квитків на культурні заходи за допомогою React.JS у середовищі програмування Visual Studio Code*

затверджена наказом ректора НТУ «ДП» від *16 . 05 .2023 р.* № *350 -с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *Проф. Мещеряков Л.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Міллєр М.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14 .01 .2023 р.*

Термін подання дипломного проекту до ДЕК ***21.06.2023 р.***

РЕФЕРАТ

Пояснювальна записка: 74 с., 26 рис., 3 дод., 19 джерел.

Об'єкт розробки: Веб-додаток пропонує розробку онлайн-сервісу бронювання квитків на культурні заходи. Дозволяє легко та зручно замовляти квитки на різноманітні культурні події. За допомогою сервісу, можна швидко ознайомитись з доступними заходами, вибрати потрібний час та місце, а також забронювати квитки без зайвих зусиль.

Мета роботи: розробка інтернет-магазину, за допомогою технології React.JS у середовищі програмування Visual Studio Code. Створення інтуїтивно зрозумілий інтерфейс для клієнтів сайту, щоб вони могли задовольнити свої потреби швидко і без зайвих труднощів.

Вступна частина кваліфікаційної роботи містить аналіз сучасного стану проблеми, що стосується бронювання квитків на культурні заходи. Визначення конкретної мети дослідження, обґрунтування актуальності обраної теми на основі цього наведено постановку завдання.

Перший розділ роботи присвячений детальному аналізу предметної галузі, актуальності задачі та цілей розробки. Сформульовано конкретні завдання і вимоги до програмної реалізації, описано використані технології та програмні засоби.

У другому розділі роботи проведено аналіз існуючих рішень, вибір платформи для розробки, проектування та реалізацію програми. Наведено детальний опис роботи програми, включаючи алгоритм та структуру її функціонування. Описано процес виклику та завантаження програми, визначено вхідні та вихідні дані, а також надано характеристику параметрів технічних засобів.

В економічному розділі роботи проведено оцінку трудомісткості розробленої інформаційної системи, розрахунок вартості роботи зі створення програми та оцінку необхідного часу для її реалізації.

Практичне значення даного проекту полягає в розробці веб-додатку з інтуїтивно-зрозумілим інтерфейсом, оскільки дозволяє користувачам безпечно

купувати квитки на культурні заходи.

Актуальність цього програмного продукту обумовлена великим попитом на дистанційні покупки через Інтернет, що спрощує процес отримання товару клієнтами та економить фінансові витрати на відкриття стаціонарного магазину.

Список ключових слів: КВИТКИ, БРОНЮВАННЯ, ОНЛАЙН-СЕРВІС, КУЛЬТУРНІ ЗАХОДИ, ВЕБ-ДОДАТОК, REACT.JS.

ABSTRACT

Explanatory note: 74 p., 26 fig., 3 appx., 19 sources.

Object of development: The web application offers the development of an online ticket reservation service for cultural events. Allows you to easily and conveniently order tickets for various cultural events. With the help of the service, you can quickly familiarize yourself with the available events, choose the right time and place, and book tickets without too much effort.

Purpose of work: development of an online store using React.JS technology in the Visual Studio Code programming environment. Creation of an intuitive interface for customers of the site so that they can meet their needs quickly and without unnecessary difficulties.

The introductory part of the qualification work contains an analysis of the current state of the problem related to booking tickets for cultural events. Determination of the specific purpose of the research, substantiation of the relevance of the chosen topic, based on this, the statement of the task is given.

The first section of the work is devoted to a detailed analysis of the subject area, the relevance of the task and development goals. Specific tasks and requirements for software implementation are formulated, the technologies and software tools used are described.

In the second part of the work, an analysis of existing solutions, selection of a platform for development, design and implementation of the program was carried out. A detailed description of the program's operation, including the algorithm and structure of its operation, is given. The process of calling and downloading the program is described, the input and output data are defined, and the characteristics of the parameters of the technical means are also provided.

In the economic section of the work, an assessment of the labor intensity of the developed information system, a calculation of the cost of work on creating the program and an assessment of the time required for its implementation were carried out.

The practical significance of this project is the development of a web application with an intuitive interface, as it allows users to securely purchase tickets for cultural events.

The relevance of this software product is due to the great demand for remote purchases via the Internet, which simplifies the process of receiving goods by customers and saves financial costs for opening a stationary store.

Keyword List: TICKETS, RESERVATION, ONLINE SERVICE, CULTURAL EVENTS, WEB APP, REACT.JS.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

СУБД - система управління базами даних.

HTML (HyperText Markup Language) - мова розмітки веб-сторінок.

JS (JavaScript) - високорівнева мова програмування, яка переважно використовується для веб-розробки.

React - відкрита бібліотека JavaScript для розробки інтерфейсів користувача.

JSX (JavaScript XML) - розширений синтаксис, який використовується в React для написання JavaScript-коду, що нагадує синтаксис HTML/XML.

CSS - мова стилів, яка використовується для визначення вигляду та форматування веб-сторінок.

ВСТУП

Онлайн-сервіс бронювання квитків на культурні заходи є популярним варіантом для швидкого та зручного придбання квитків без необхідності виходу з дому. Він надає можливість користувачам переглядати різноманітні культурні заходи, обирати підходящі квитки та забронювати їх онлайн. Цей сервіс є перспективним для підприємців, оскільки він дозволяє залучити широку аудиторію та ефективно просувати культурні заходи в Інтернеті.

Використання інтернету для продажу квитків на культурні події має свої переваги. Перш за все, воно дозволяє легко знайти та обрати бажані події, ознайомитися з детальною інформацією про них та перевірити наявність квитків. Крім того, цей сервіс забезпечує зручний спосіб оплати та можливість отримати квитки електронним шляхом, що робить процес бронювання та отримання квитків простим та швидким.

Для успішного функціонування онлайн-сервісу бронювання квитків на культурні заходи необхідно використовувати сучасні технології. Використання бібліотеки React.JS дозволяє створити потужний та інтуїтивно зрозумілий інтерфейс користувача, який забезпечує зручну навігацію та швидке бронювання квитків. Крім того, використання React.JS дозволяє легко масштабувати сервіс та додавати нові функції у майбутньому.

Розробка онлайн-сервісу бронювання квитків на культурні заходи з використанням React.JS є вигідною інвестицією для бізнесу в сучасному цифровому світі. Цей сервіс дозволяє підприємцям залучати нових клієнтів, ефективно просувати культурні заходи та забезпечувати зручний та надійний спосіб бронювання квитків. Застосування React.JS дозволяє створити сучасний та функціональний онлайн-сервіс, який забезпечить задоволення користувачів та сприятиме розвитку бізнесу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

Онлайн-сервіси бронювання квитків на культурні заходи - це електронні платформи, які надають користувачам можливість замовити квитки на різноманітні культурні події через Інтернет. Вони стали популярними завдяки зручності та широкому вибору подій, доступних для бронювання. Основною перевагою онлайн-сервісів бронювання квитків є можливість швидко та зручно придбати квитки без необхідності особистого візиту до каси або організатора події. Користувачі можуть легко переглянути доступні події, ознайомитися з деталями та цінами квитків, а потім забронювати їх за декілька простих кроків.

Онлайн-сервіси бронювання квитків на культурні заходи надають широкий вибір подій, включаючи концерти, театральні вистави, кінотеатри, фестивалі, спортивні змагання та багато іншого. Користувачі можуть швидко знайти та обрати подію, яка їх цікавить, а також вибрати місце та кількість квитків. Бронювання квитків онлайн дозволяє користувачам ефективно планувати свій час та уникнути черг на касах. Вони можуть зручно забронювати квитки з будь-якого місця, використовуючи комп'ютер, смартфон або планшет.

Такі сервіси є надійним та зручним рішенням для покупців та організаторів подій. Вони сприяють популяризації культурних заходів, полегшують процес бронювання квитків і допомагають людям насолоджуватися різноманітними культурними подіями.

1.2. Призначення розробки та область застосування

Розроблений веб-додаток націлений на надання клієнтам по всій Україні можливості забронювати квитки на різноманітні культурні заходи. Цей сервіс спрощує процес бронювання квитків, забезпечуючи зручний і швидкий доступ до важливих подій.

Основною областю застосування цього сервісу є культурна сфера, зокрема театральні вистави, концерти, кіносеанси, музичні фестивалі та інші заходи, що пропонуються в різних містах України. Користувачі можуть швидко переглянути доступні події, обрати бажану дату, час та місце, а потім забронювати квитки в зручний для них час.

1.3. Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи) є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- завдання на кваліфікаційну роботу на тему «Розробка онлайн-сервісу бронювання квитків на культурні заходи за допомогою React.JS у середовищі програмування Visual Studio Code».

1.4. Постановка завдання

Розробити веб-додаток, що надає можливість користувачам зручно та швидко бронювати квитки на різноманітні культурні заходи. Додаток буде заснований на технології React.JS та розроблено у середовищі програмування Visual Studio Code.

Основні завдання проекту:

- Розробка інтуїтивно зрозумілого та зручного інтерфейсу, який дозволяє користувачам швидко знайти потрібні квитки та здійснити їх бронювання.
- Розробка функціоналу для управління каталогом квитків, включаючи можливість додавання нових квитків, редагування цін та опису, видалення застарілих записів.
- Розробка функціоналу для управління особистим кабінетом користувача, включаючи можливість додавання необхідної інформації, яка буде використовуватись при обробці замовлення.

1.4.1. Функціональні особливості, актуальність сайту та можливості

Сервіс бронювання квитків на культурні заходи пропонує наступні функціональні особливості:

- Каталог подій: розміщення фото, опису та ціни квитків на різноманітні культурні заходи.
- Кошик бронювання: можливість додавати квитки до кошика, редагувати їх кількість та видаляти замовлення.
- Реєстрація: форма, що дозволяє користувачам зареєструватися на сайті, створити обліковий запис та здійснювати бронювання квитків.
- Адміністративна панель: інтерфейс для адміністраторів, який дозволяє керувати подіями, редагувати їх, видаляти та додавати нові до бази даних.
- Ці функціональні особливості є ключовими для забезпечення зручності та ефективності користування онлайн-сервісом бронювання квитків на культурні заходи.
- Створення такого сервісу має велику актуальність у сучасному світі. Зростання популярності культурних подій та попиту на їх відвідування вимагає зручного та швидкого способу бронювання квитків. Онлайн-сервіс надає можливість забезпечити користувачів доступом до різноманітних заходів та зручним способом придбання квитків.
- Можливості сайту включають:

- - Моніторинг подій на сайті, оновлення цін та описів.
- - Реєстрацію користувачів у базі даних.
- - Вхід до особистого облікового запису, що дозволяє здійснювати бронювання квитків, редагувати особисту інформацію та поповнювати баланс.
- - Додавання квитків до кошика та їх видалення.
- - Адміністраторський доступ: можливість додавання нових подій до бази даних безпосередньо через веб-інтерфейс, редагування та видалення подій.
- - Вкладки з додатковою інформацією про сайт.
- Такий сервіс бронювання квитків на культурні заходи має потенціал залучити широку аудиторію та забезпечити зручність та доступність придбання квитків онлайн. З урахуванням зростання популярності онлайн-покупок та підтримки культурних заходів, такий сайт може стати успішним рішенням для бізнесу.

1.4.2. Опис інтерфейсу користувача

Користувач починає свій досвід на головній сторінці, перейшовши за посиланням. Перш за все, його увагу привертають кнопки на сайті та їх назви. Інтуїтивно зрозуміло, яку функцію виконує кожна кнопка. У верхній частині сайту користувач знаходить такі кнопки:

- "Увійти" – натиснувши її, з'являється форма для авторизації в системі, якщо користувач вже зареєстрований;
- "Реєстрація" – при натисканні на цю кнопку з'являється форма для реєстрації користувача у базі клієнтів. Після успішної реєстрації користувач повертається на головну сторінку для авторизації;
- "Про нас" та "логотип сайту" – натиснувши на них, з'являється форма з основною інформацією про розробника;
- "Контакти" – після натискання відображається форма з контактними телефонами розробника та посиланнями на його соціальні мережі;

"Особистий кабінет" - при натисканні відображається одна з наступних форм:

1) якщо користувач не авторизувався – відображається форма з нагадуванням про необхідність авторизації;

2) якщо користувач авторизований як клієнт – відображається форма з додатковою інформацією про клієнта та можливістю редагування цієї інформації;

3) якщо користувач авторизований як адміністратор – відображається форма, яка надає можливість редагування, додавання та видалення товарів.

4) На головній сторінці користувач знайде інформацію про доступні товари, включаючи фото, назву та ціну. При натисканні на фото товару відображається додаткова форма з детальним описом товару.

5) Після успішної авторизації користувача очікують наступні об'єкти:

- текстове привітання зі зареєстрованим ім'ям;
- кнопка "Баланс" та поточний баланс клієнта. При натисканні на кнопку відкривається форма для поповнення балансу;
- кнопка "Вихід", яка дозволяє користувачеві вийти зі свого облікового запису.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Онлайн-сервіс бронювання квитків на культурні заходи повинен відповідати певним вимогам щодо своїх функціональних характеристик, щоб забезпечити максимальний комфорт та зручність для користувачів. Основні функціональні характеристики даного сервісу включають:

1. Зручний та інтуїтивно зрозумілий інтерфейс: Сайт повинен мати легкий у використанні інтерфейс, який дозволяє користувачам легко знаходити необхідні квитки на культурні заходи та виконувати інші операції в магазині.

2. Інформація про кожен квиток: Сайт повинен надавати детальну інформацію про кожен квиток, включаючи назву заходу, опис, фотографії та ціни. Це допомагає користувачам зробити правильний вибір і забезпечує прозорість у процесі бронювання.

3. Онлайн-бронювання: Користувачам повинна бути надана можливість зручного онлайн-бронювання квитків на культурні заходи.

Виконання цих функціональних характеристик є важливим для успішної роботи онлайн-сервісу бронювання квитків на культурні заходи. Вони забезпечують високу якість обслуговування та задоволеність клієнтів.

1.5.2. Вимоги до інформаційної безпеки

Онлайн-сервіс бронювання квитків на культурні заходи має певні вимоги до інформаційної безпеки.

Однією з вимог є реєстрація та авторизація користувачів на сайті. Для цього користувачі повинні створити обліковий запис та використовувати унікальний логін та пароль. Пароль повинен бути зашифрований за допомогою відповідної алгоритмічної функції для забезпечення безпеки. Ці дані зберігаються у базі даних з метою захисту від несанкціонованого доступу та шахрайства.

На сайті існують дві ролі: користувач та адміністратор. Кожна роль має свою відповідну панель управління. Користувачам надається доступ до функцій бронювання квитків та перегляду інформації про культурні заходи. Адміністратор, з свого боку, має розширені права, такі як додавання, редагування та видалення товарів.

Ці заходи забезпечують безпеку інформації та обмежують доступ до функцій тільки авторизованим користувачам. Таким чином, забезпечується конфіденційність та цілісність даних, а також унеможливлюється недобросовісна поведінка та зловживання з боку користувачів.

1.5.3. Вимоги до складу та параметрів технічних засобів

Онлайн-сервіс бронювання квитків на культурні заходи має певні вимоги до складу та параметрів технічних засобів. Для забезпечення зручного використання веб-додатка користувачем, необхідно, щоб його технічний засіб відповідав таким технічним вимогам:

- Операційна система: Windows 7-11, Linux або MacOS.
- Мінімум 2 ГБ оперативної пам'яті.
- Ці вимоги забезпечать оптимальну роботу веб-додатка і забезпечать зручність користування для користувачів.

1.5.4. Вимоги до інформаційної та програмної сумісності

Користувач може зареєструватись на сайті і забронювати будь-який квиток. Розробка сайту проводилась з використанням мови JavaScript і бібліотеки ReactJs, Firebase яка є однією з найпопулярніших в даний час та забезпечує інтерактивність та зручність використання.

Для безперебійної роботи сайту вимагається актуальна версія веб-браузера.

- Сайт сумісний з різними версіями популярних веб-переглядачів, що забезпечує комфортну роботу для користувачів.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Метою даного проекту є створення веб-сайту, спрямованого на бронювання квитків на культурні заходи, такі як ярмарки, виставки, спектаклі, фестивалі та концерти. Веб-сайт буде розроблений з використанням мови програмування JavaScript та фреймворку ReactJS[1]. Основний функціонал веб-сайту полягатиме в наданні користувачам можливості знаходити, вибирати та бронювати квитки з використанням системи фільтрації.

Веб-сайт буде мати два рівні доступу до функціоналу:

1) Клієнт: на цьому рівні користувачі зможуть реєструватись та увійти в систему, переглядати список доступних заходів та отримувати докладну інформацію про кожен захід.

2) Адміністратор: адміністратори матимуть можливість додавати або видаляти заходи зі списку, переглядати список заброньованих квитків та редагувати інформацію про заходи.

Крім основного функціоналу, важливими складовими веб-сайту будуть такі додаткові можливості:

- Можливість оновлювати та редагувати базу даних: ця функція дозволить адміністраторам додавати нові заходи, виправляти інформацію про існуючі заходи та вносити інші зміни до бази даних.

- Налаштування кошика: користувачам буде надана можливість додавати вибрані квитки до кошика або видаляти їх. Крім того, будуть реалізовані функції розрахунку загальної суми та оформлення покупки.

- Налаштування авторизації на сайті: для забезпечення безпеки та персоналізації веб-додатка буде реалізована система авторизації користувачів. Це дозволить клієнту створити обліковий запис, увійти у

свій профіль та керувати інформацією, необхідною для бронювання квитків.

Усі ці функції додаткового функціоналу допоможуть покращити роботу веб-сайту, забезпечать зручність користування та спростять процес бронювання квитків на культурні заходи.

2.2. Опис застосованих математичних методів

Під час проектування та розробки веб-додатка не використовувалися складні математичні алгоритми. Розрахунки та обчислення, які використовувалися, були простими арифметичними діями.

2.3. Опис використаної архітектури та шаблонів проектування

При проектуванні та розробці нашого веб-сайту для бронювання квитків на культурні заходи було використано архітектурний шаблон MVC (Model-View-Controller) та його варіацію, відому як MVVM (Model-View-ViewModel). Цей шаблон дозволяє ефективно організувати взаємодію між компонентами системи.

У нашому веб-сайті, компонент "App" виступає в ролі контролера, який керує взаємодією між моделлю (дані про події та квитки) і представленням (відображенням інтерфейсу користувача).

Архітектурний шаблон MVC (Model-View-Controller) допомагає нам чітко розділити компоненти системи на три основні ролі: Модель (Model), Представлення (View) та Контролер (Controller). Кожна з цих ролей має свої відповідності і виконує чітко визначені функції. Модель відповідає за управління даними, Представлення відповідає за відображення даних та інтерфейс користувача, а Контролер керує логікою взаємодії між ними та обробкою дій користувача.

Застосування архітектурного шаблону MVC дозволяє нам ефективно розділити логіку програми, відображення даних та управління взаємодією

користувача з нашим веб-сайтом для бронювання квитків на культурні заходи. (рис. 2.1)

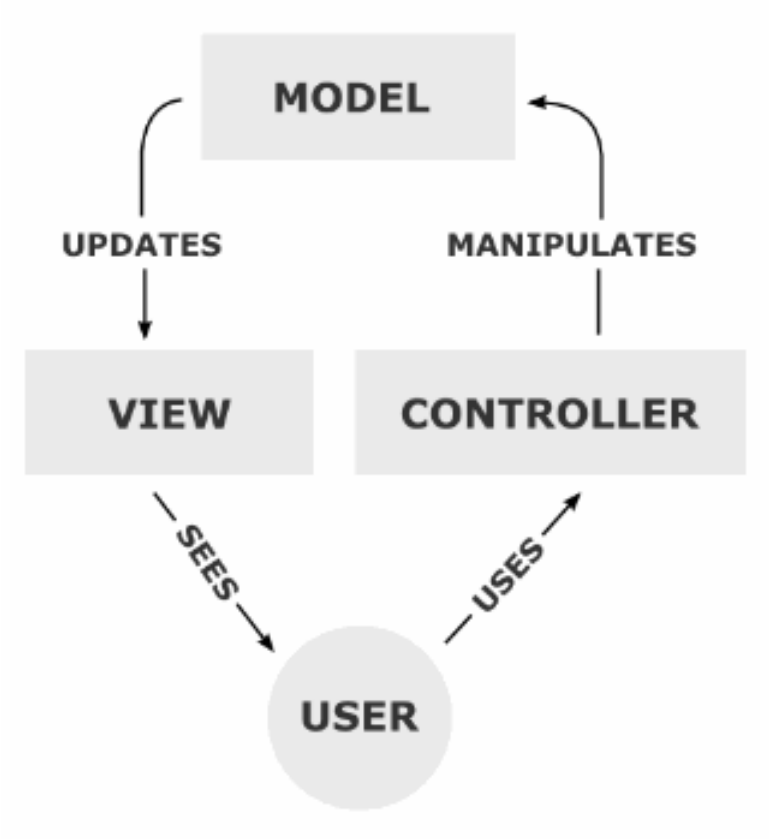

Рис. 2.1. Графічне представлення шаблону MVC

Наш сайт бронювання квитків на культурні заходи використовує архітектурний шаблон MVC (Model-View-Controller) та його варіацію, відому як MVVM (Model-View-ViewModel).

Основні компоненти MVC в контексті нашого сайту:

1) Модель (Model): цей компонент відповідає за збереження та обробку даних, а також за виконання операцій над ними. Він містить бізнес-логіку нашого додатку та надсилає сповіщення про будь-які зміни даних.

2) Представлення (View): представлення відповідає за візуалізацію даних та інтерфейс користувача. Воно отримує дані від моделі та відображає їх у відповідному форматі, щоб користувач міг взаємодіяти з нашим сайтом.

3) Контролер (Controller): контролер відповідає за обробку подій, взаємодію з користувачем та оновлення моделі та представлення. Він слухає події від представлення, виконує відповідні дії та змінює стан

моделі або представлення.

Крім того, у нашому проекті також використовується шаблон "Контейнер-представлення" (Container-Presentation). Контейнери виконують роль контролерів, керуючи станом додатку, обробкою подій та взаємодією з іншими компонентами. Представлення відповідають за візуалізацію даних та не мають власного стану.

Переваги використання MVC та шаблону "Контейнер-представлення" на нашому сайті:

1) Розділення відповідальностей: ці шаблони дозволяють розділити логіку, представлення та взаємодію з користувачем між різними компонентами. Це полегшує розробку та підтримку коду.

2) Покращена повторна використовуваність: можливість пере використання моделей, представлень та контейнерів у різних частинах сайту дозволяє зменшити дублювання коду та покращити ефективність розробки.

3) Тестованість: окреме тестування моделі, представлення та контейнерів дозволяє забезпечити надійність та якість нашого додатку.

Хоча MVC та шаблон "Контейнер-представлення" є потужними інструментами для розробки, важливо враховувати особливості нашого проекту та вибрати підхід, що найкраще підходить для наших потреб. Застосування цих шаблонів допоможе нам зробити наш сайт бронювання квитків на культурні заходи структурованим, легким у розробці та підтримці, а також надійним для користувачів.

Переваги шаблону "Контейнер-представлення" для сайту бронювання квитків на культурні заходи:

1) Розділення відповідальностей: Шаблон дозволяє чітко розділити логіку додатку від його візуалізації. Контейнери відповідають за управління станом та логікою, тоді як представлення відображає лише дані.

2) Покращена керованість: Контейнери дозволяють ефективно керувати станом додатку та взаємодіяти з іншими компонентами. Це спрощує розробку та

розуміння функціоналу додатку.

3) Повторне використання та тестуваність: Представлення можна повторно використовувати, оскільки воно не залежить від стану додатку. Крім того, компоненти легко піддаються тестуванню окремо.

Мінуси шаблону "Контейнер-представлення" для сайту бронювання квитків на культурні заходи:

1) Збільшена кількість компонентів: Використання даного шаблону може призвести до збільшення кількості компонентів у додатку, що може бути незручним для менш складних проектів.

2) Додатковий шар абстракції: Шаблон вводить додатковий рівень абстракції, що вимагає розуміння його концепцій та правильної організації компонентів.

Загалом, шаблон "Контейнер-представлення" є корисним інструментом для керування станом та розподілу відповідальностей у веб-додатках для бронювання квитків на культурні заходи. Використовуючи цей шаблон, розробники можуть створювати чистий, модульний та добре керований код.

Firebase — це Backend-as-a-Service (Baas), що належить Google і дозволяє розробникам створювати повний стек веб-додатків за кілька хвилин. Сервіси на кшталт Firebase дуже спрощують розробникам інтерфейсу створювати веб-програми з повним набором програм із незначними навичками програмування або зовсім без них.

Firebase надає різні методи автентифікації, базу даних NoSQL, базу даних у реальному часі, сховище файлів, хмарні функції, послуги хостингу та багато іншого.

Переваги Firebase Realtime Database: синхронізація в реальному часі для даних JSON, створення без серверних додатків, оптимізація для автономного використання, безпека користувацьких даних.

Вся інформація зберігається на хмарному сервісі FireBase. Для адміністратора (менеджера) додаються права на створення нових даних та редагування існуючих даних, які містяться в базі. Бази даних Firebase, на відміну

від mySQL, побудовані не таблицями, а деревом.

Подання бази даних у формі дерева спрощує роботу адміністратора при додаванні чи зміні даних про заходи. За допомогою адмін-панелі сервісу Firebase можна надати доступ до бази даних будь-якому користувачеві з правами власника, редактора чи глядача. Також можна додати права на окрему колекцію чи документ, що значно спрощує роботу адміністратора.

Усі центри обробки даних Firebase мають сертифікати SOC 2 Type 2 та ISO 27001. Firebase також використовує низку заходів безпеки для захисту ваших даних, зокрема:

1) шифрування даних. Усі дані Firebase шифруються під час передачі та передачі.
2) контроль доступу на основі ролей. Firebase використовує керування доступом на основі ролей (RBAC), щоб забезпечити детальний контроль над тим, хто може отримати доступ до даних програми.
3) журнал аудиту. Firebase реєструє всі доступи до даних, щоб компанії могли відстежувати, хто і коли отримував доступ до даних програми.

Приклад використання Firebase для виконання запиту і отримання даних приведено на рисунку 2.2:

```
65  export const getCategoriesAndDocuments = async () => {
66    const collectionRef = collection(db, 'categories');
67    const q = query(collectionRef);
68
69    const querySnapshot = await getDocs(q);
70    const categoryMap = querySnapshot.docs.map((docSnapshot) =>
71      docSnapshot.data()
72    );
73
74    return categoryMap;
75  };
```

Рис. 2.2. Приклад використання ‘Firebase

2.4. Опис використаних технологій та мов програмування

Технології та мови програмування, які були використані при створенні даного проекту:

- HTML (HyperText Markup Language);
- CSS (Cascading Style Sheets);
- JavaScript[7];
- React;
- Firebase[13];
- Stripe[18].

Технології та мови програмування, які були використані при створенні нашого веб-сайту для бронювання квитків на культурні заходи, включають:

- HTML (HyperText Markup Language) - мову розмітки, яка використовується для створення структури та вмісту веб-сторінок. HTML визначає елементи, атрибути та текстовий вміст, що впливає на вигляд і функціональність сторінки.

- CSS (Cascading Style Sheets) - мову стилів, яка визначає вигляд і форматування веб-сторінок. За допомогою CSS можна змінювати кольори, шрифти, розміри, розташування елементів і багато іншого.

- JavaScript - мову програмування, яка використовується для додавання інтерактивності до веб-сторінок. JavaScript дозволяє створювати динамічні ефекти, обробляти події користувача та взаємодіяти з сервером для отримання та оновлення даних.

- React - JavaScript-бібліотеку для створення інтерфейсів користувача. React дозволяє розбити інтерфейс на компоненти, які можуть бути повторно використовані та ефективно керовані.

- Firebase - платформу для розробки веб-додатків, яка надає різноманітні сервіси, такі як хостинг, бази даних, аутентифікація користувачів та інші, для спрощення процесу розробки та підтримки веб-додатків.

- Stripe - платіжну платформу, яка дозволяє здійснювати онлайн-платежі та обробляти платіжні транзакції безпечно і зручно.

HTML, CSS, JavaScript, React, Firebase та Stripe - це потужні інструменти, які допомогли нам створити функціональний та привабливий веб-сайт для бронювання квитків на культурні заходи. Вони дозволили нам створити зручний інтерфейс, забезпечити безпеку платежів та ефективно керувати даними.

JavaScript є важливою мовою програмування для нашого веб-сайту бронювання квитків на культурні заходи. Ця високорівнева, інтерпретована мова програмування широко використовується для розробки веб-додатків і надає нам можливість взаємодіяти з користувачем, маніпулювати веб-сторінками та створювати динамічні та інтерактивні елементи.

Основні риси та можливості JavaScript, які ми використовуємо на нашому веб-сайті, включають:

1) Веб-браузерний скриптінг: JavaScript дозволяє нам створювати веб-додатки, які взаємодіють з користувачем та змінюють вміст веб-сторінок. Ми можемо динамічно змінювати елементи сторінки, реагувати на події користувача та виконувати асинхронні запити на сервер.

2) Кросс-платформеність: JavaScript підтримується всіма сучасними веб-браузерами і може працювати на різних платформах, таких як Windows, macOS та Linux.

3) Об'єктно-орієнтований підхід: JavaScript базується на об'єктно-орієнтованому програмуванні, що дозволяє нам створювати об'єкти з властивостями і методами. Це дозволяє організовувати код у модулі та повторно використовувати його.

4) Функціональне програмування: JavaScript також підтримує функціональне програмування, де функції є об'єктами першого класу і можуть передаватись як аргументи, зберігатись у змінних та повертатись з функцій.

5) Асинхронність: JavaScript має підтримку асинхронного програмування, що дозволяє виконувати операції без блокування потоку виконання. Це особливо

корисно для виконання мережевих запитів та обробки відповідей з сервера без затримки інших частин програми.

6) Багатофункціональність: JavaScript може бути використаний для розробки різноманітних додатків, включаючи веб-додатки, мобільні додатки, настільні додатки, ігри та навіть серверну частину за допомогою платформи Node.js.

7) Велика екосистема: JavaScript має велику спільноту розробників і підтримується багатьма сторонніми бібліотеками і фреймворками, такими як React, Angular, Vue.js, які спрощують розробку веб-додатків і покращують продуктивність.

Ми використовуємо JavaScript для створення динамічних веб-сторінок, валідації форм, реалізації анімаційних ефектів, маніпулювання DOM, взаємодії з веб-сервером за допомогою Firebase та багато іншого. JavaScript є мовою, яку багато розробників вивчають і використовують для створення потужних та інтерактивних веб-додатків.

React є відкритою бібліотекою JavaScript, яка ідеально підходить для розробки нашого веб-сайту для бронювання квитків на культурні заходи. Вона створена компанією Facebook і має широке застосування у побудові ефективних та масштабованих веб-додатків[2-6].

Основні риси та можливості React:

1) компонентна структура: React дозволяє розбити веб-інтерфейс на незалежні компоненти, які можуть бути повторно використані і спрощують організацію коду. Кожен компонент може мати свій внутрішній стан (state) та властивості (props), що дозволяє створювати динамічний контент;

2) віртуальний DOM: React використовує внутрішню структуру даних, відому як віртуальний DOM. Він представляє легку копію реального DOM і дозволяє React ефективно виконувати оновлення та перерисування компонентів. React порівнює стару та нову версії віртуального DOM, знаходить різницю та застосовує тільки необхідні зміни до реального DOM;

3) односторінкові додатки: За допомогою React ми можемо створити односторінкові додатки, де весь контент завантажується один раз, а потім динамічно оновлюється без перезавантаження сторінки. Це забезпечує швидку та зручну взаємодію з нашим веб-сайтом;

4) JSX: React використовує JSX (JavaScript XML) для опису компонентів та їх структури. JSX поєднує JavaScript і синтаксис, схожий на HTML, що дозволяє легко описувати компоненти та їх взаємодію з даними;

5) універсальність: React може працювати як на стороні клієнта, так і на стороні сервера. На стороні клієнта React може бути використаний для розробки веб-додатків, а на стороні сервера він може генерувати HTML на основі компонентів для передачі клієнту;

6) розширюваність: React можна поєднувати з іншими бібліотеками та фреймворками, наприклад, Redux для керування станом додатків, або React Router для реалізації маршрутизації в односторінкових додатках;

7) розробка на основі компонентів: React підтримує підхід до розробки на основі компонентів, де кожен компонент відповідає за свою функціональність і може бути незалежно розроблений, тестований та підтримуваний.

React є дуже популярним веб-фреймворком і використовується для розробки великої кількості веб-додатків. Він надає зручні інструменти для побудови модульних, швидких та ефективних інтерфейсів користувача.

Firebase - це платформа розробки застосунків (backend-as-a-service), розроблена компанією Google. Вона надає набір інструментів та сервісів для створення веб-додатків, мобільних додатків та інших типів програм з хмарним бекендом [14-17].

Особливості та можливості Firebase:

1) аутентифікація користувачів: Firebase надає вбудовану підтримку для аутентифікації користувачів, дозволяючи реалізувати різні методи аутентифікації, такі як електронна пошта, соціальні медіа, номери телефонів і багато інших. Це дозволяє зручно керувати доступом та ідентифікацією користувачів у додатках;

2) база даних реального часу: Firebase Realtime Database - це розподілена база даних, яка забезпечує синхронізацію даних в режимі реального часу між різними клієнтами. Це дозволяє створювати додатки, які миттєво реагують на зміни даних і спільно використовують оновлені дані;

3) зберігання файлів: Firebase надає сервіс зберігання файлів, який дозволяє завантажувати та зберігати файли на хмарному сервері. Це можна використовувати, наприклад, для збереження зображень, відео, аудіофайлів та інших медіа-ресурсів у додатках;

4) хмарні функції: Firebase має можливість виконувати серверний код безпосередньо на хмарному сервері. Це дозволяє виконувати різноманітні обчислення та маніпулювати даними без необхідності власних серверів;

5) інтеграція з аналітикою та маркетингом: Firebase надає інструменти для збору та аналізу даних про користувачів, моніторингу використання додатків, відстеження конверсій та інших метрик. Це допомагає вдосконалювати додатки та здійснювати ефективний маркетинговий аналіз;

6) інструменти для розробки: Firebase надає набір інструментів для розробки, таких як Firebase Hosting для хостингу веб-сайтів, Firebase Cloud Functions для розгортання серверного коду, Firebase Test Lab для автоматизованого тестування додатків та багато інших;

7) швидке розгортання: Firebase надає простий спосіб розгортання додатків та доступу до хмарних сервісів. Розробникам не потрібно витрачати час на налаштування серверів або інфраструктури, оскільки Firebase бере на себе багато аспектів інфраструктури;

8) масштабованість: Firebase забезпечує масштабованість додатків, дозволяючи автоматично масштабувати інфраструктуру під зростання використання та завантаження.

Firebase є популярною платформою для розробки додатків, особливо мобільних додатків та додатків з реальним часом оновлення даних. Вона надає широкий спектр функціональності і спрощує процес розробки, дозволяючи розробникам швидко створювати потужні додатки з хмарним бекендом.

2.5. Опис структури програми та алгоритмів її функціонування

Наш сайт для бронювання квитків на культурні заходи має структуру односторінкового додатка (Single-Page Application, SPA).

SPA є архітектурним підходом, в якому весь контент та логіка розміщуються на одній сторінці, яка завантажується лише один раз під час початкового завантаження. Для динамічної взаємодії з користувачем використовується JavaScript, що дозволяє оновлювати вміст сторінки без перезавантаження.

Структура нашого проекту включає наступні компоненти:

1) index.html - це головний HTML-файл, який містить загальну структуру сторінки та основний контент;

2) App.js - це JavaScript-файл, відповідальний за керування логікою та взаємодією на сторінці. Він містить функції та обробники подій, які реагують на натискання кнопок та відкривають відповідні форми або елементи на сторінці;

3) index.css - це файл стилів, який містить CSS-стилі для оформлення сторінки;

4) форми - кожна кнопка на головній сторінці відкриває відповідну форму або елемент на сторінці. Форми можуть бути реалізовані як вбудовані елементи на сторінці або включати модальне вікно, яке з'являється поверх основного контенту.

Така структура дозволяє нам забезпечити взаємодію з користувачем без перезавантаження сторінки. Користувачі можуть заповнювати форми, відправляти дані та взаємодіяти з елементами за допомогою JavaScript.

Головний файл index.html розташований у папці "public" проекту і відповідає за завантаження на сторінку div з id='root' та встановлення деяких елементів, таких як назва сайту на вкладці браузера, підключення скриптів та інші. (рис. 2.3)

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Web site created using create-react-app" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
      href="https://fonts.googleapis.com/css2?family=Montserrat:wght@100;400;500;600;700;900&display=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.15.3/css/all.css"
      integrity="sha384-SZXxX4whJ79/gErwcOYf+zWLeJdY/qpuqC4cAa9rOGUstPomtqpuNWT9wdPEn2fk"
      crossorigin="anonymous"
    />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Miller tickets | shop</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Рис. 2.3. Вміст файлу index.html

Після завантаження, виконується файл index.js (з папки src) на нашому веб-сайті для бронювання квитків на культурні заходи. Цей файл включає основні бібліотеки React та ReactDOM, а також файли index.css та App.js.

Далі створюється компонент "root", який є головним контейнером для нашого веб-додатку. У цьому компоненті обробляється весь інтерфейс та функціонал нашого сайту для бронювання квитків на культурні заходи. (рис. 2.4)

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { Elements } from '@stripe/react-stripe-js';
import App from './App';
import { persistor, store } from './app/store';
import { stripePromise } from './utils/stripe/stripe.utils';
import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <BrowserRouter>
          <Elements stripe={stripePromise}>
            <App />
          </Elements>
        </BrowserRouter>
      </PersistGate>
    </Provider>
  </React.StrictMode>
);
```

Рис. 2.4. Вміст файлу index.js

ReactDOM - це пакет, який надає методи для взаємодії з DOM (Document Object Model) у контексті нашого веб-сайту для бронювання квитків на культурні заходи, заснованого на React. Цей пакет дозволяє нам рендерити React-компоненти у віртуальному DOM та вставляти їх у справжній DOM сторінки.

У нашому проекті ми використовуємо метод ReactDOM.render(element, container[, callback]), який дозволяє рендерити компоненти React. Цей метод приймає кореневий елемент (React-компонент або JSX-вираз) та контейнер (DOM-елемент), в який потрібно вставити рендеринг. Ми також можемо вказати опціональний зворотний виклик (callback), який виконається після успішного рендерингу.

Файл index.css використовується для налаштування стилів усіх елементів веб-сторінки. Це дозволяє нам зручно керувати зовнішнім виглядом нашого веб-додатку.

Усі елементи нашого веб-додатку завантажуються з компонента App.js, який є головним компонентом. Він також має свої дочірні компоненти, такі як Header, User, Items та інші. Директорія з різними компонентами має наступну структуру (рис. 2.5):

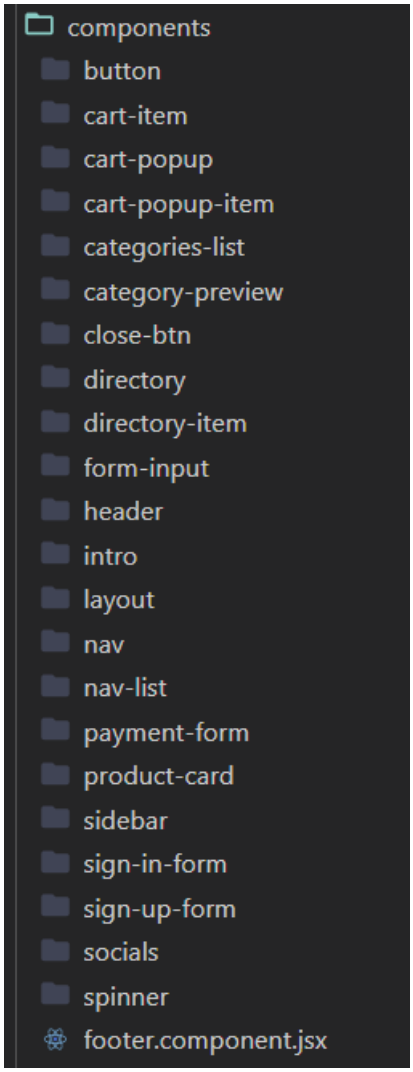

Рис. 2.5. Директорія компонентів веб-додатку

Ця директорія містить всі компоненти, що використовуються при завантаженні сторінки та виконанні різних дій на нашому веб-сайті для бронювання квитків на культурні заходи. Тут знаходяться файли і модулі, які забезпечують коректну роботу інтерфейсу, обробку даних, взаємодію з базою даних та інші функціональні можливості. Всі ці компоненти працюють разом, щоб забезпечити зручну та ефективну роботу веб-сайту та забезпечити користувачам максимальний комфорт при бронюванні квитків на культурні заходи.

За зв’язок з базою даних відповідає файл firebase.utils.js, який представлено нижче (рис. 2.6):

```
import { initializeApp } from 'firebase/app';
import {
  getAuth,
  signInWithRedirect,
  signInWithPopup,
  GoogleAuthProvider,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut,
  onAuthStateChanged
} from 'firebase/auth';
import {
  getFirestore,
  doc,
  getDoc,
  setDoc,
  collection,
  writeBatch,
  query,
  getDocs
} from 'firebase/firestore';

const firebaseConfig = {
  apiKey: "AIzaSyAqP7xN3d6bFv8nmqu4O4-5amiM84ZNy5k",
  authDomain: "tickets-sale-4150b.firebaseapp.com",
  projectId: "tickets-sale-4150b",
  storageBucket: "tickets-sale-4150b.appspot.com",
  messagingSenderId: "799325656375",
  appId: "1:799325656375:web:9c8be3e52d3463ac54f6b2"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

const googleProvider = new GoogleAuthProvider();
```

Рис. 2.6. Firebase файл зв’язку з даними

2.6. Обґрунтування та організація вхідних та вихідних даних програми

У проекті даної кваліфікаційної роботи вхідними даними є:

- логін, пароль;

Вихідними даними є дані про товар, які зберігаються у базі даних та виводяться у зручному для користувача графічному вигляді при відправленні певного запиту.

У проекті нашого веб-сайту для бронювання квитків на культурні заходи використовується платформа Firebase для обміну даними між клієнтською та серверною частинами.

Firebase надає зручні інструменти для зберігання та синхронізації даних в реальному часі. Дані можуть бути збережені у базі даних Firestore, яка працює з колекціями та документами, що містять структуровану інформацію у форматі JSON.

Обмін даними між клієнтом та сервером відбувається за допомогою API Firebase, яке дозволяє передавати дані у форматі JSON між клієнтською стороною (веб-браузером або мобільним додатком) та серверною частиною, яка хоститься на Firebase.

Цей підхід забезпечує швидкий та ефективний обмін даними між клієнтом та сервером, дозволяючи зберігати, оновлювати та отримувати дані у реальному часі, що є особливо важливим для функціоналу бронювання квитків на культурні заходи.

2.7. Опис роботи розробленого програмного продукту

Для роботи з сайтом на основі React, вам спочатку потрібно перейти за його посиланням. У випадку використання локального сервера Open Server Panel, ви можете виконати наступні кроки:

1. Запустіть локальний сервер Open Server Panel на вашому комп'ютері. Впевніться, що сервер працює та налаштований для роботи з React;

2. Перейдіть за посиланням на вашому веб-браузері, яке вказує на локальний сервер Open Server Panel. Наприклад, це може бути адреса `http://localhost:80` або `http://127.0.0.1:80`;

3. Після переходу за посиланням, ви побачите ваш веб-сайт на основі React, який запущений на локальному сервері;

Зверніть увагу, що Open Server Panel є інструментом для налаштування локального сервера та не є безпосередньо пов'язаним з React. При роботі з React ви також повинні мати налаштоване середовище розробки, таке як Node.js та пакетний менеджер npm або Yarn.

Важливо пам'ятати, що React можна використовувати як частину іншого веб-сайту або як самостійний проект. Після налагодження середовища розробки ви можете створювати та розгортати власні React-додатки на сервері.

Зовнішній вигляд сайту (рис. 2.8):

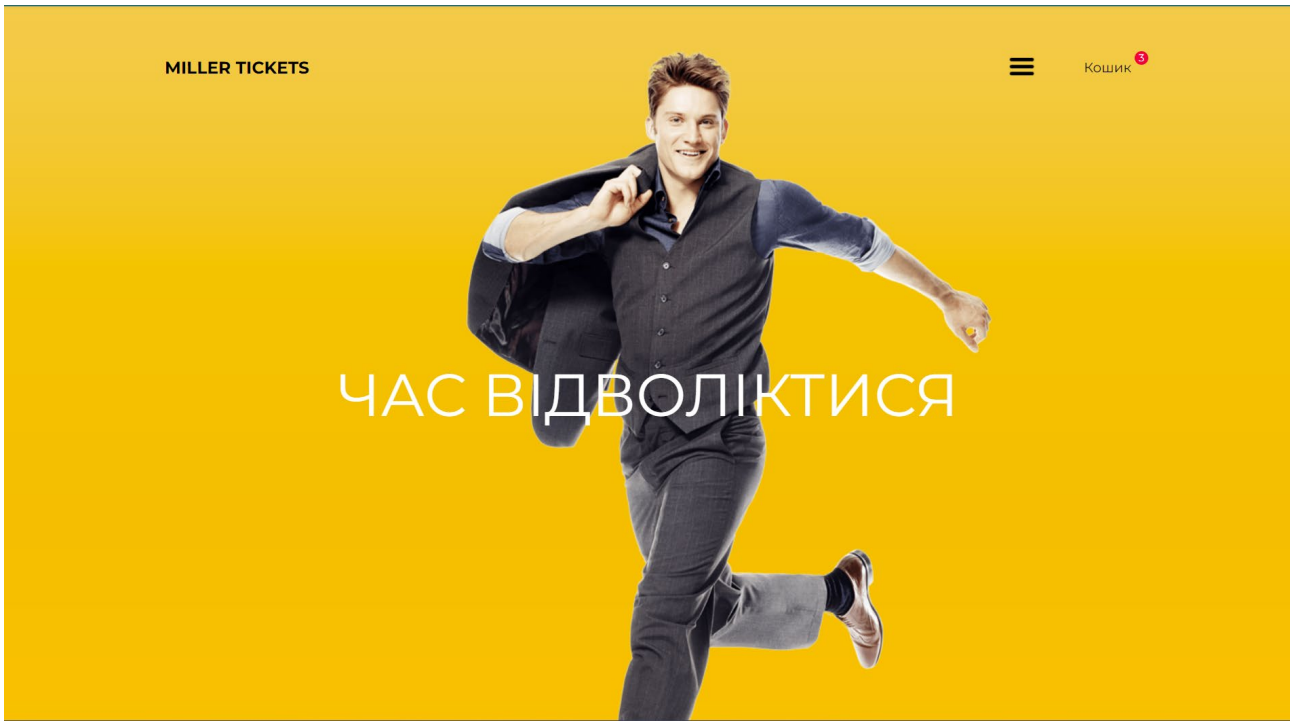

Рис. 2.8. Головна сторінка «Miller Tickets»

2.7.1. Використані технічні засоби

Для забезпечення зручної роботи з нашим React-додатком на сучасних браузерах, рекомендується мати наступні мінімальні параметри комп'ютера:

1) центральний процесор (ЦП): Рекомендується Intel Core i5 або еквівалент;

2) відеоадаптер: Рекомендується 3D адаптер від nVidia, Intel або AMD/ATI;

3) відеопам'ять: Мінімум 512 МБ відеопам'яті;

4) накопичувач: Рекомендується мати щонайменше 250 ГБ вільного простору на жорсткому диску;

5) оперативна пам'ять: Мінімум 4 ГБ оперативної пам'яті.

Ці рекомендації допоможуть забезпечити плавну роботу нашого React-додатку та ефективне відтворення веб-контенту на вашому комп'ютері. Звичайно, якщо плануєте використовувати важкі веб-додатки або обробку великих обсягів даних, може знадобитися більш потужний комп'ютер з більшим обсягом пам'яті та швидшим процесором.

Щодо моїх технічних засобів, я використовую наступне обладнання:

1) центральний процесор (ЦП): Intel Core i7;

2) відеоадаптер: 3D адаптер від nVidia;

3) відеопам'ять: 8 ГБ;

4) накопичувач: 500 ГБ SSD;

5) оперативна пам'ять: 32 ГБ;

6) клавіатура, миша та монітор.

Ці технічні характеристики дозволяють мені ефективно виконувати завдання та розробляти React-додатки з високою продуктивністю.

2.7.2. Використані програмні засоби

При розробці цього проекту для веб-сайту бронювання квитків на культурні заходи було використано наступні програмні засоби:

1. React: React є потужною бібліотекою JavaScript для розробки інтерфейсів користувача. Використання React дозволяє створювати ефективні та масштабовані компоненти для веб-додатків;

2. Firebase: Firebase є платформою, яка надає набір інструментів для розробки веб-додатків, включаючи хостинг, аутентифікацію, базу даних та зберігання файлів. Використання Firebase дозволяє зручно працювати з реальними часом, автентифікацією користувачів та збереженням даних;

3. Visual Studio Code: Visual Studio Code є безкоштовним редактором коду, який надає зручне середовище для розробки веб-додатків на основі React та Firebase. Він підтримує багато корисних функцій, таких як підсвічування синтаксису, автодоповнення, відладка та інші (Рис 2.9);

4. Firebase Hosting: Firebase Hosting надає можливість розгортання веб-сайту на хмарному хостингу Firebase. Завдяки цьому, веб-додаток може бути доступний онлайн з будь-якого пристрою;

5. Firebase Authentication: Firebase Authentication дозволяє забезпечити аутентифікацію користувачів у веб-додатку за допомогою різних методів, таких як електронна пошта та соціальні мережі. Це забезпечує безпеку та захист даних користувачів;

6. Firebase Realtime Database: Firebase Realtime Database - це реальний час, хмарна база даних, яка дозволяє зберігати та синхронізувати дані між різними пристроями в режимі реального часу. Це особливо корисно для створення функцій, які потребують оновлення даних миттєво.

За допомогою цих програмних засобів, ви можете розробити веб-сайт бронювання квитків на культурні заходи з використанням React для створення інтерфейсу користувача та Firebase для зберігання даних та забезпечення аутентифікації користувачів.

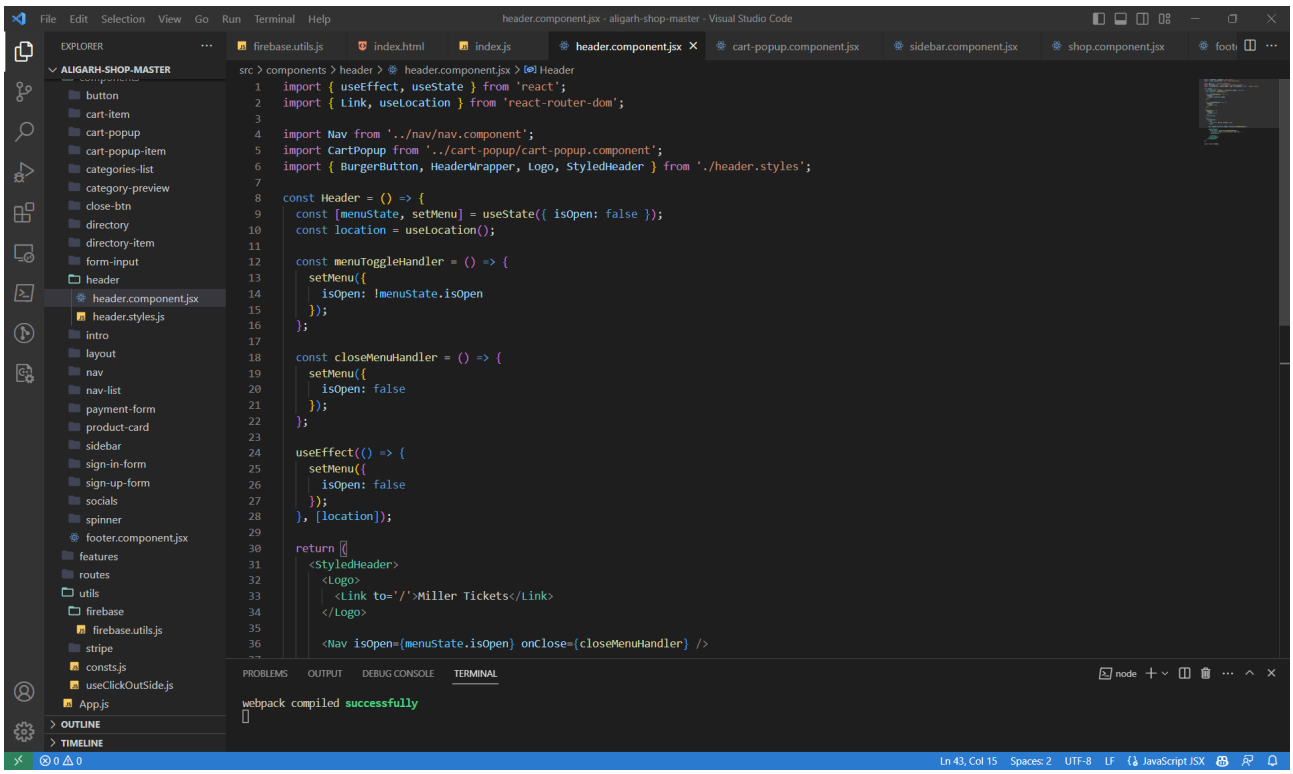

Рис. 2.9. Зовнішній вигляд VS Code

2.7.3. Виклик та завантаження програми

Для доступу до розробленого веб-застосунку, ви можете скористатись стандартизованою адресою (URL: http://localhost:3001), за допомогою сучасних браузерів, які підтримуються на різних операційних системах. Наприклад, на десктопних операційних системах, таких як Windows 10, Linux та macOS, ви можете використовувати популярні браузери, такі як Google Chrome, Firefox та Brave. На мобільних платформах підтримуються браузери, такі як Google Chrome, Dolphin, Opera Mobile, Mozilla Firefox та Safari.

Наш веб-застосунок гарантовано запускається на останніх двох версіях усіх підтримуваних браузерів, які визначені офіційними розробниками. Зазвичай, додаток також може працювати на старіших версіях браузерів, але ми не можемо гарантувати успішну роботу на них.

Таким чином, в залежності від вашої операційної системи та пристрою, ви можете отримати доступ до нашого веб-застосунку через обраний вами браузер, забезпечуючи широку сумісність та доступність на різних платформах.

2.7.4. Опис інтерфейсу користувача

Після запуску нашого веб-додатку, користувач автоматично переходить на головну сторінку нашого інтернет-магазину. Оскільки структура нашого сайту є односторінковою, додаткові форми, поля та кнопки з'являються на цій сторінці при натисканні на відповідні елементи (рис. 2.8).

Для початку використання нашого сайту необхідно пройти процес реєстрації у нашій базі користувачів. Це можна зробити, натиснувши кнопку "Зареєструватися", розташовану у правому верхньому куті екрану (рис. 2.12).

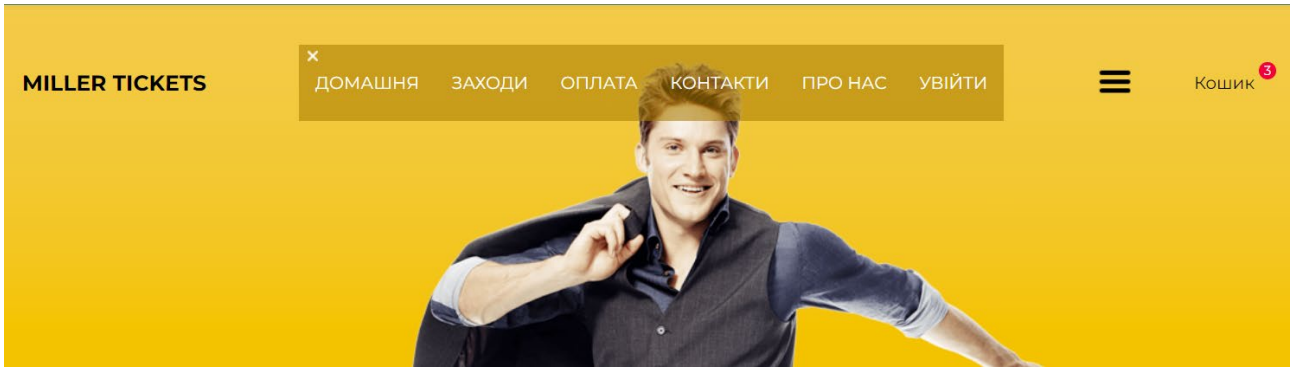

Рис. 2.12. Кнопка реєстрації

Після натискання цієї кнопки з’являється форма для реєстрації (рис. 2.13).

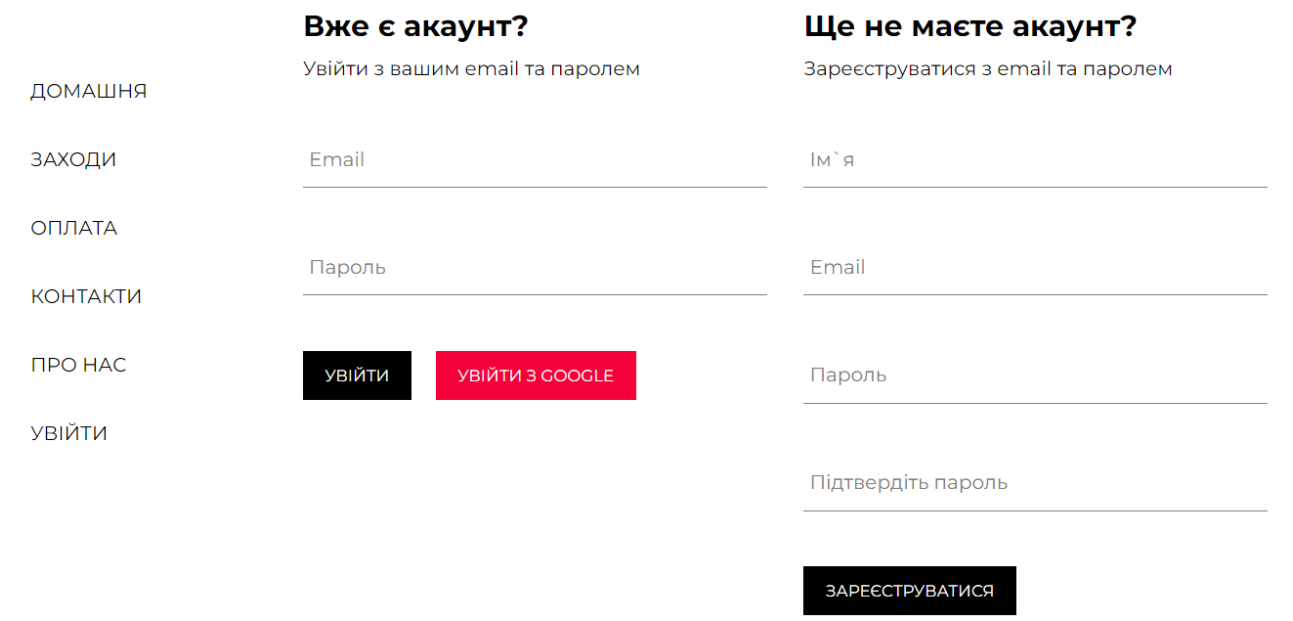

Рис. 2.13. Форма реєстрації

При редагуванні даних в консолі Firebase, ви маєте наступні можливості:

1) Додавання та редагування користувачів:

- Ви можете створити нового користувача, вказавши його логін, пароль та іншу необхідну інформацію.
- Також у вас є можливість редагувати існуючі дані користувачів, наприклад, змінити їх ім'я або додаткову інформацію.

2) Додавання та оновлення товарів:

- Ви можете створити новий товар, вказавши необхідну інформацію про нього, наприклад, назву, опис та ціну.
- Також у вас є можливість оновити дані існуючих товарів, наприклад, змінити їх ціну або оновити опис.

3) Керування ролями користувачів:

- Ви можете змінити ролі користувачів, встановивши їх як адміністраторів або звичайних користувачів.
- Це дозволить вам контролювати доступ та функціонал користувачів на основі їх ролі.

Використовуючи ці можливості, ви зможете налаштувати функціонал для різних ролей користувачів. Наприклад, для адміністратора сайту ви можете відкрити спеціальний інтерфейс (рис. 2.14), який надає доступ до вищезазначених можливостей редагування даних та керування користувачами

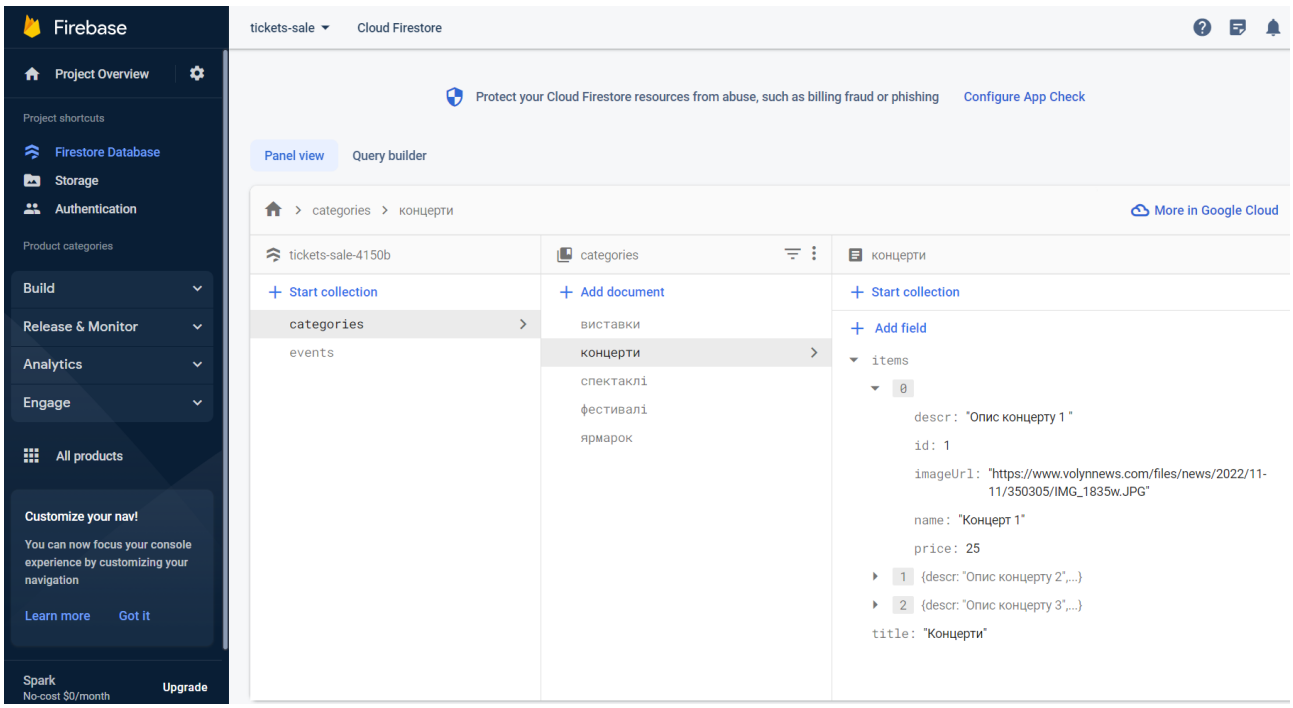

Рис. 2.14. Інтерфейс адміністратора

Цей функціонал доступний лише для адміністраторів. Проте, інші користувачі також мають доступ до інших можливостей, не обмежених роллю адміністратора.

Наприклад, якщо користувач є клієнтом, він має наступний функціонал: Для прикладу, розглянемо інтерфейс, який відкривається клієнтові (див. рис. 2.15 – 2.17).

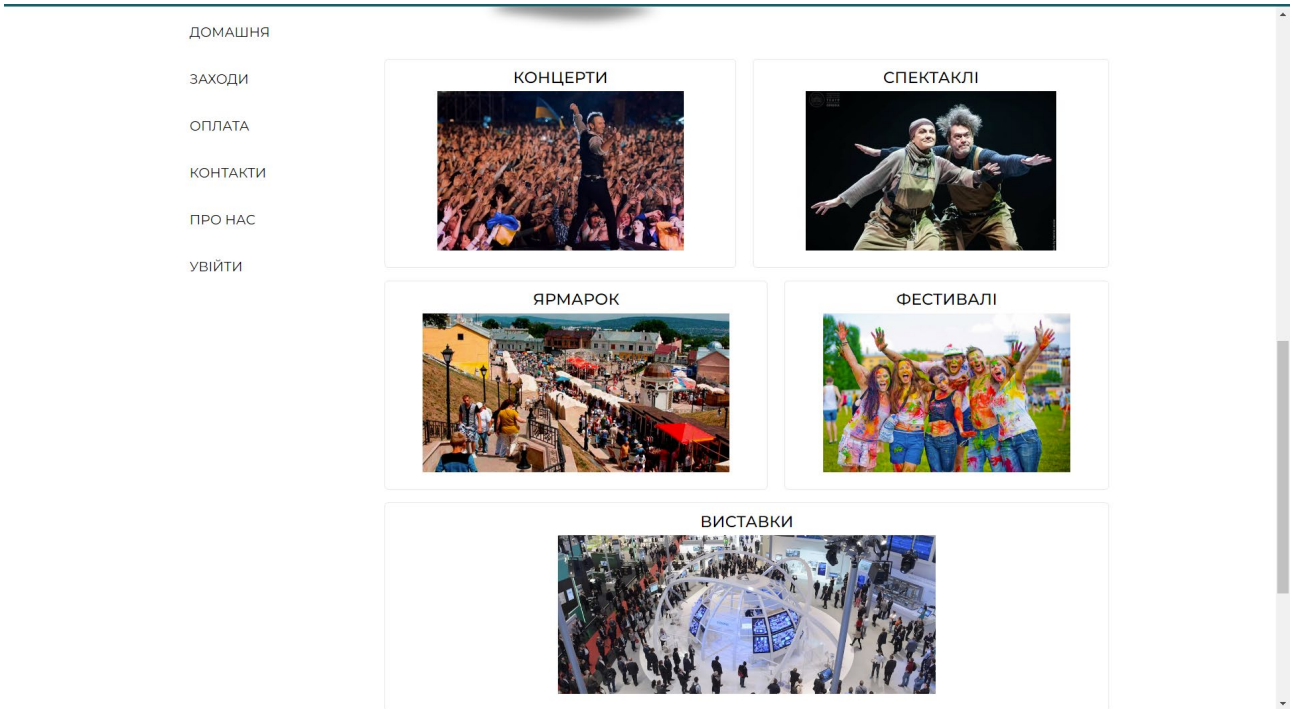

Рис. 2.15. Інтерфейс клієнта

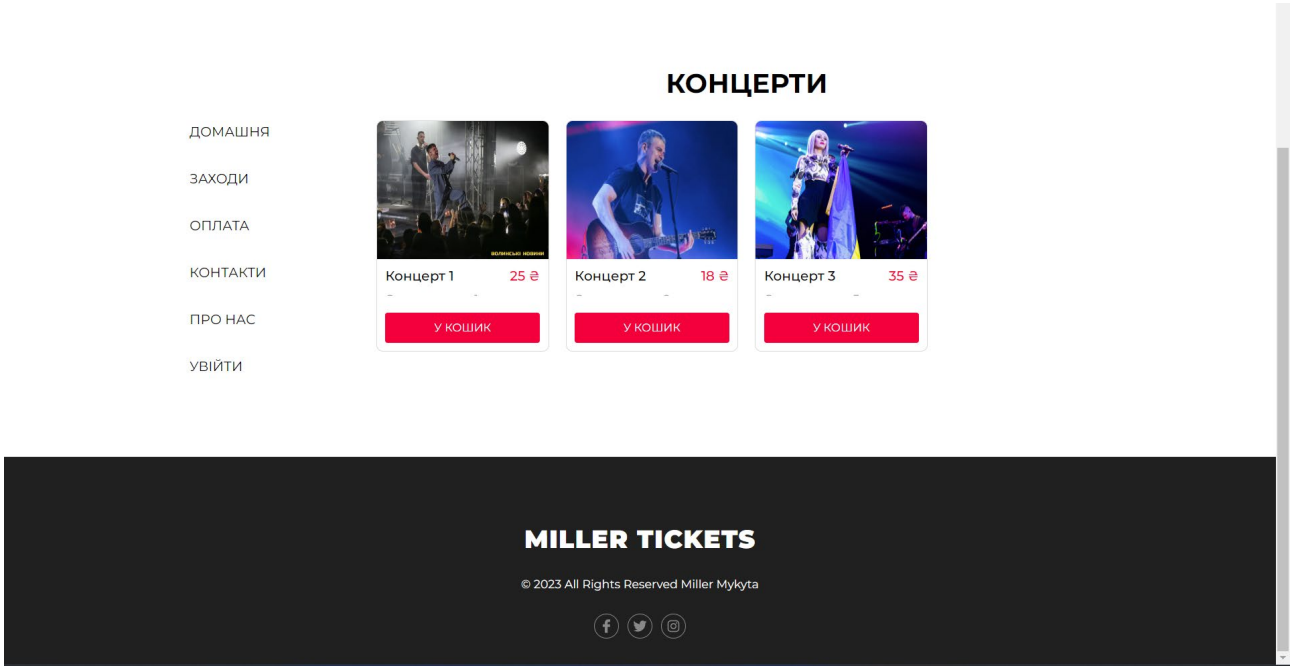

Рис. 2.16. Інтерфейс клієнта

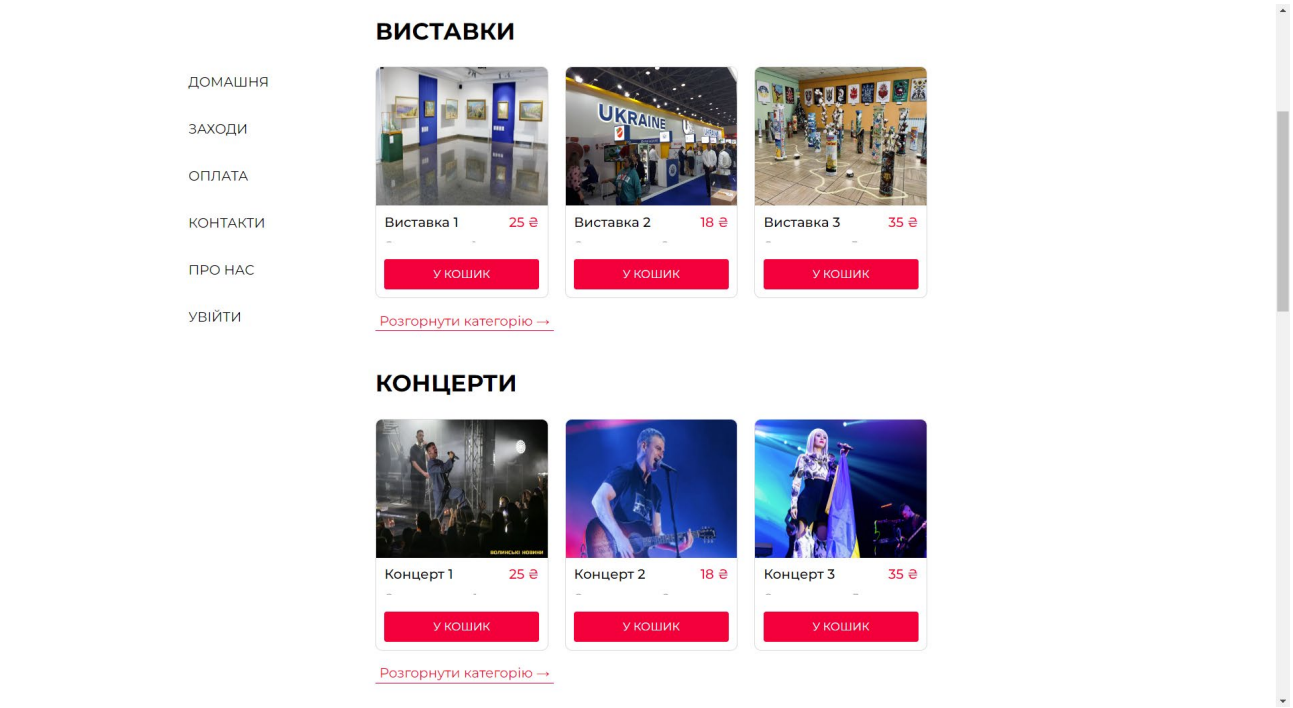

Рис. 2.17. Інтерфейс клієнта

Вгорі сторінки розташовані кнопки: «Меню», «Кошик (рис. 2.18).

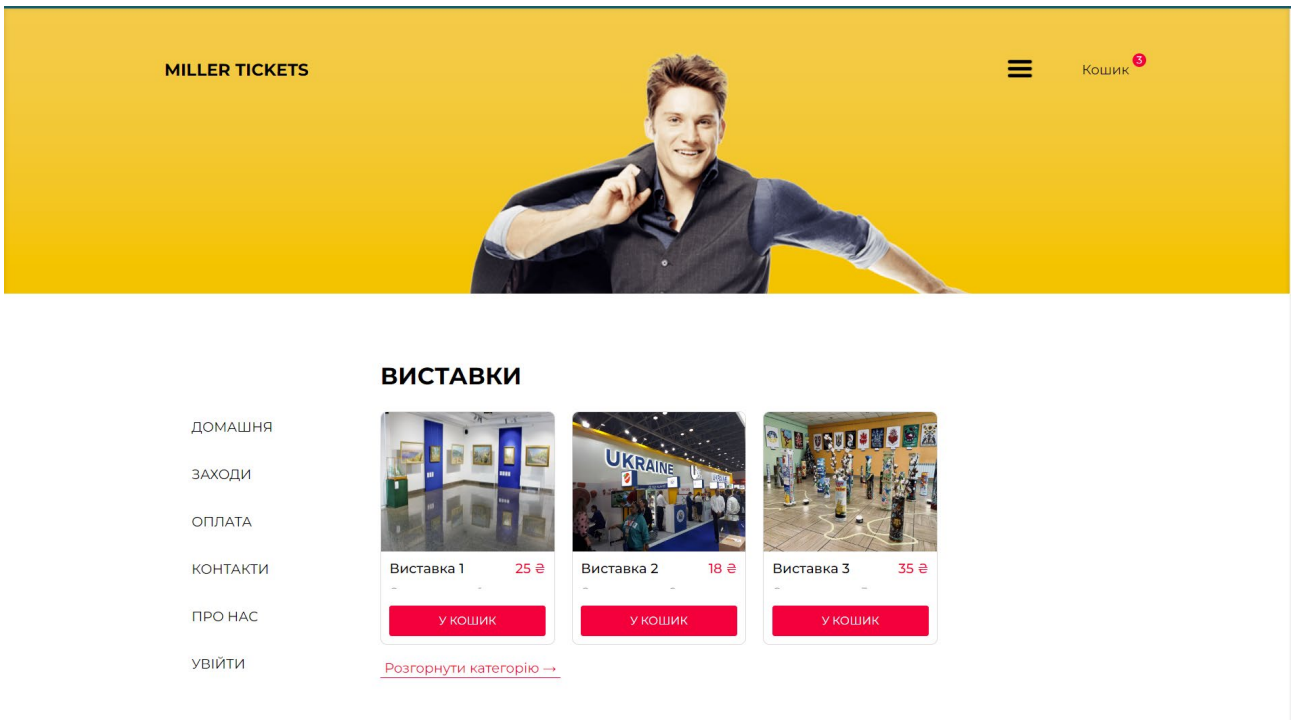

Рис. 2.18. Клієнтська частина

Кнопка «Про нас» відкриває сторінку з основною інформацією про розробника веб-додатку (рис. 2.19).

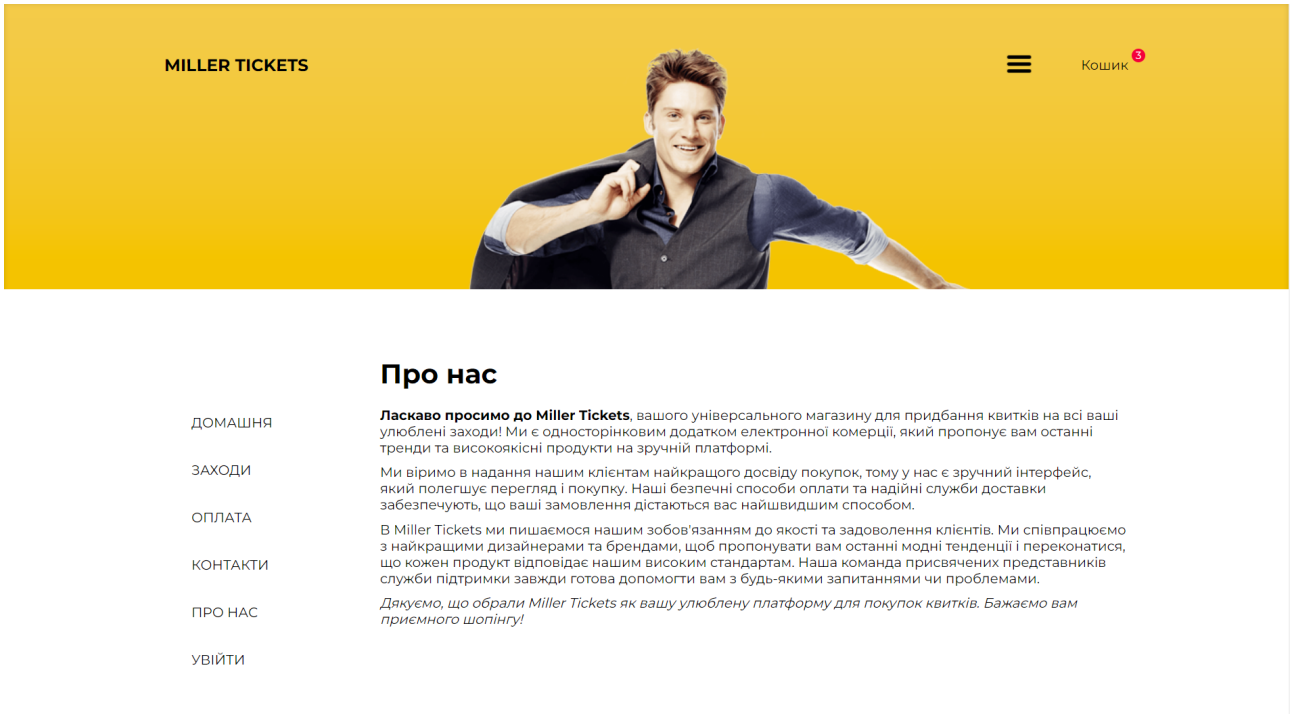

Рис. 2.19. Форма з основною інформацією

Посилання «Контакти» відкриває сторінку з контактами розробника магазину: мобільні телефони, соціальні мережі (Twitter, Instagram, Facebook) (рис. 2.20).

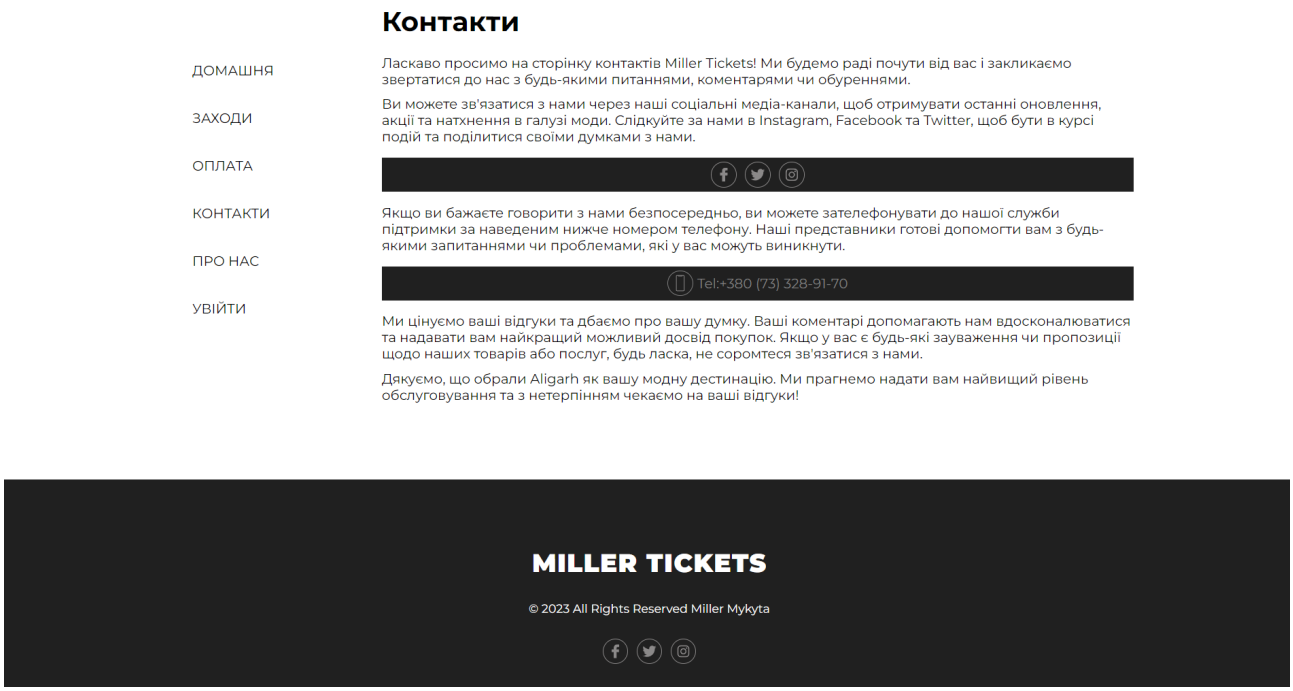

Рис. 2.20. Форма з контактами розробника

Натискання посилання «Кошик» виводить модальне вікно з кошиком, у якому відображаються білети, обрані користувачем для покупки. Додані у кошик білети можна легко видалити звідти, за допомогою кнопки «X» (рис. 2.21-22).

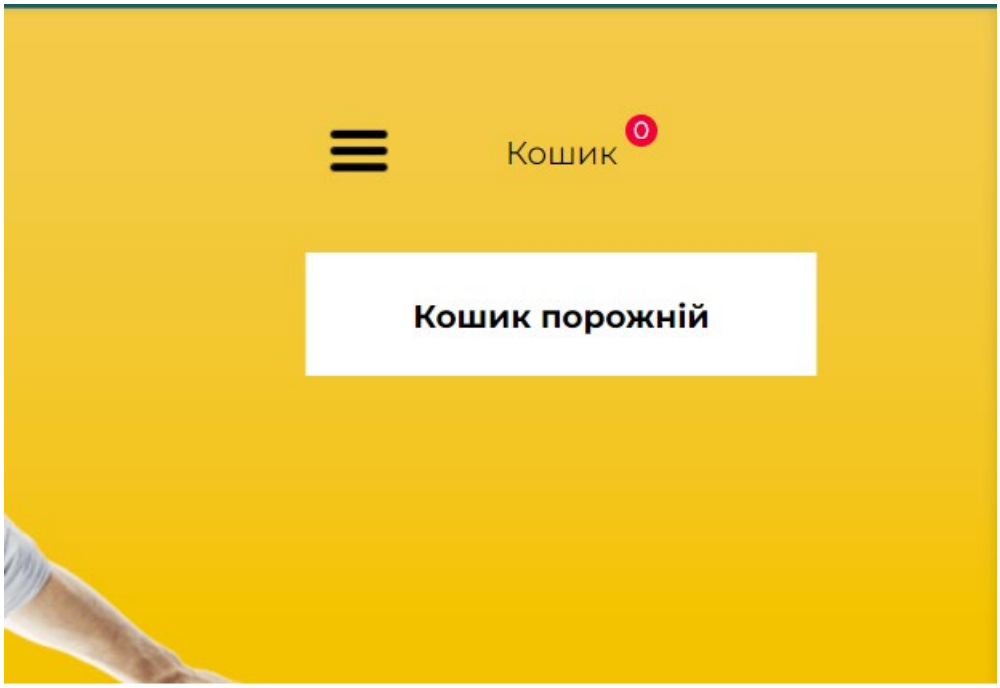

Рис. 2.21. Порожній кошик

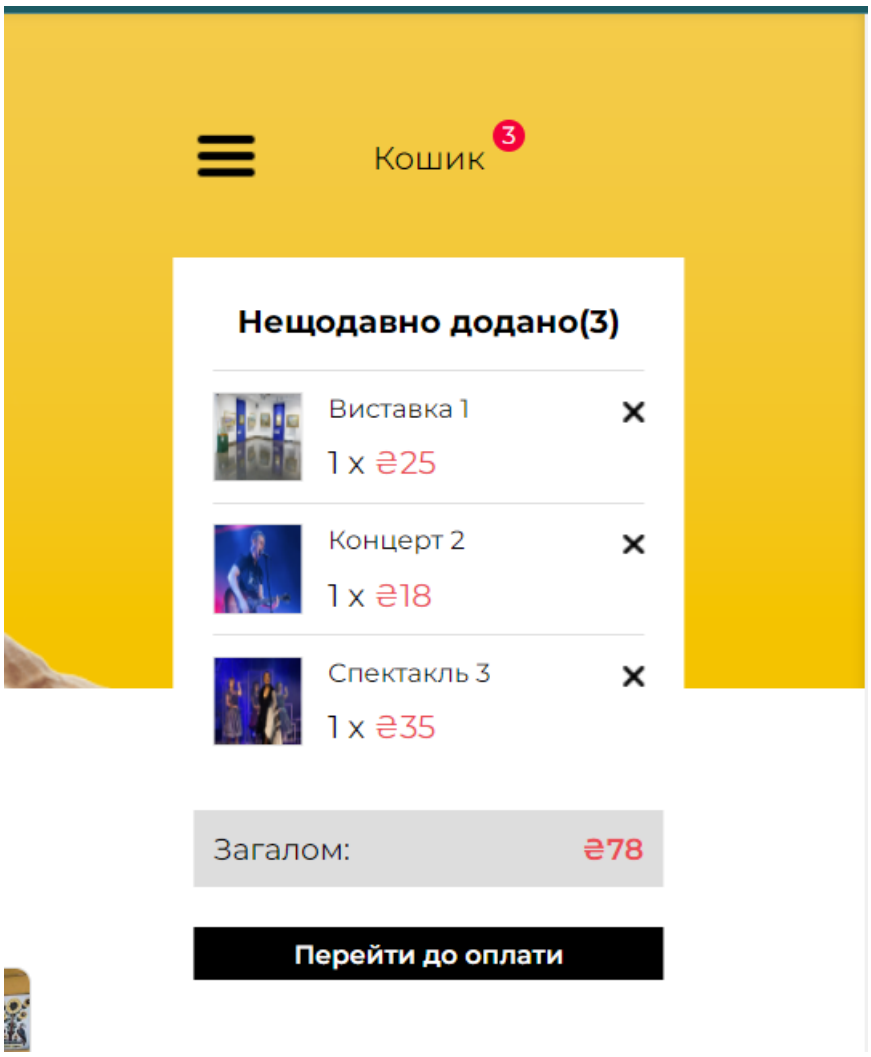

Рис. 2.22. Кошик, наповнений товарами

Кнопка «Вийти» виходить з облікового запису користувача. Після її натискання, для здійснення покупок необхідна нова авторизація.

Також, при наведенні натисканні на назву інтернет-магазину «Miller Tickets», можна перейти на головну сторінку.

Нижче, під фоновим зображенням, розташовані посилання з назвами категорій заходів, за якими можна перейти до цікавлячих (рис. 2.23).

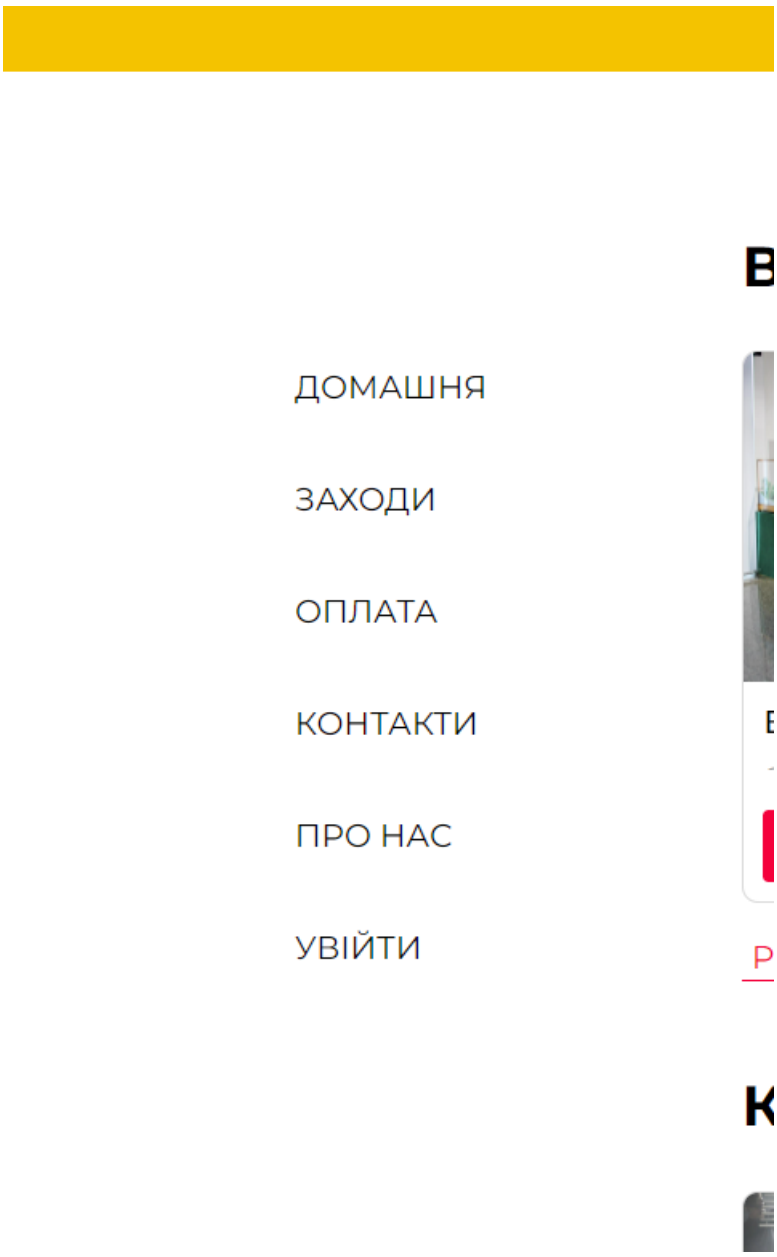

Рис. 2.23. Категорії заходів

Кнопка «Заходи» відображає список усіх, доступних для покупки білетів на заходи (рис. 2.24).

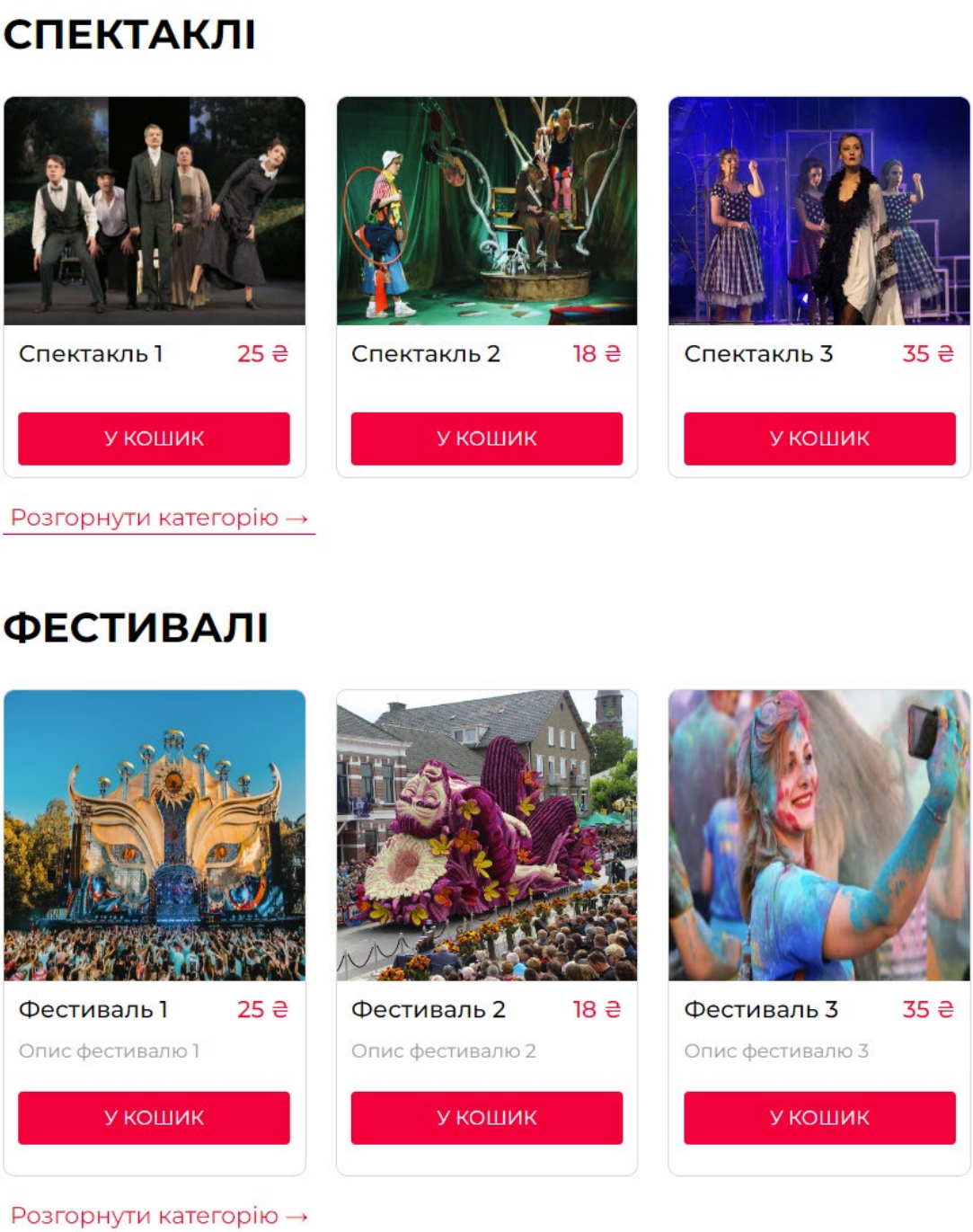

Рис. 2.24. Список усіх товарів

Кнопка «Розгорнути категорію» відображає список заходів, які відносяться до певної категорії (рис. 2.25).

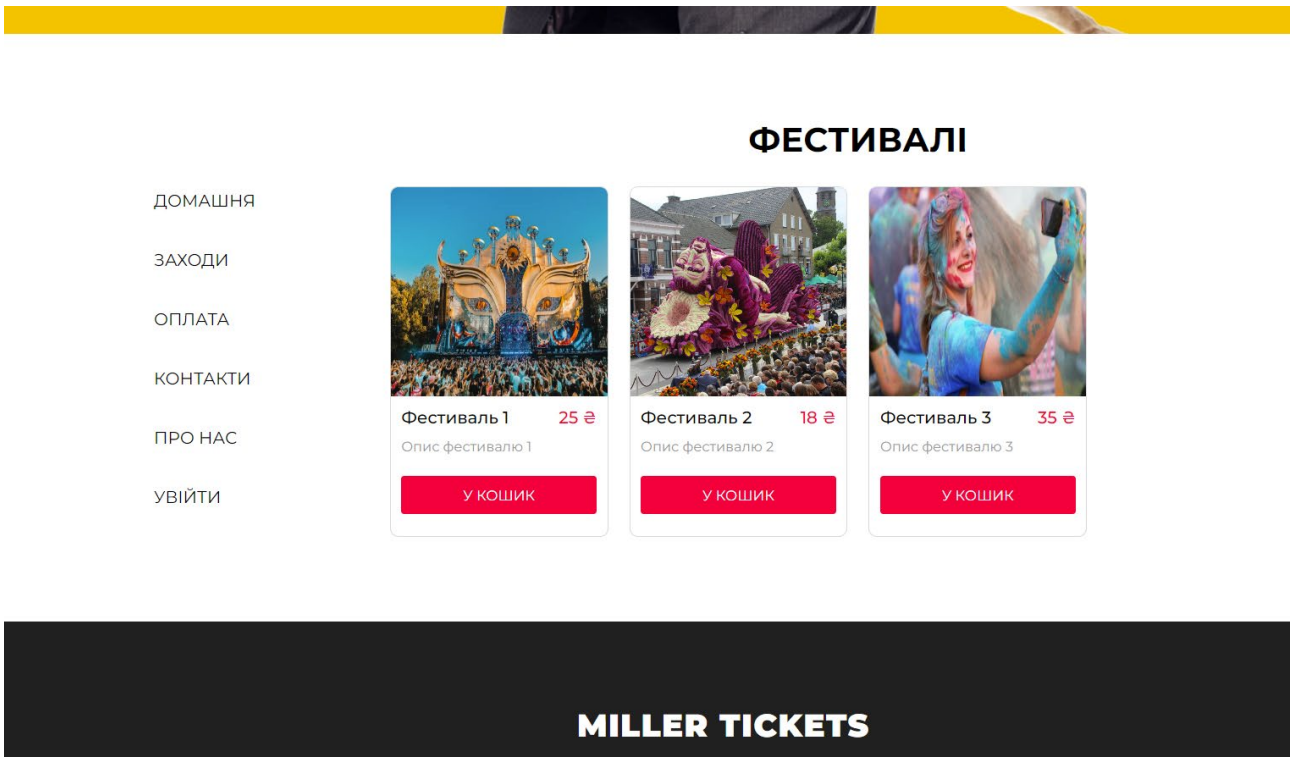

Рис. 2.25. Категорія «Фестивалі»

У футері сайту знаходиться попередження про авторські права створеного веб-додатку (рис. 2.26).

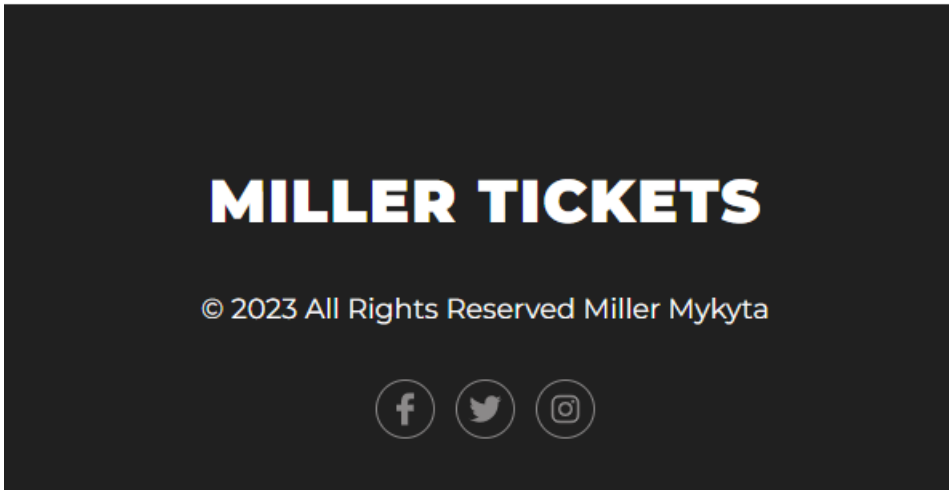

Рис. 2.26. Авторські права

Загалом, створений інтерфейс користувача є легким у освоєнні, високий рівень досвіду користувача, інтуїтивно зрозумілим, забезпечує високу якість обслуговування та задоволеність клієнтів.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

У процесі розробки програмного продукту для сайту бронювання квитків на культурні заходи проведено розрахунок трудомісткості та вартості з урахуванням наступних вихідних даних:

1. Передбачуване число операторів програми: 1632;

2. Коефіцієнт складності програми: 1.25;

3. Коефіцієнт корекції програми в ході розробки: 0.2;

4. Годинна заробітна плата розробника: 115 грн/год (згідно зі статистикою з сайту WorkUA для регіону Дніпро);

5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1.2;

6. Коефіцієнт кваліфікації програміста, залежний від стажу роботи: 1.1;

7. Вартість машино-години ЕОМ: 18 грн/год (включаючи оплату електроенергії, програмного забезпечення та амортизацію обладнання).

Трудомісткість розробки програмного продукту можна розрахувати за допомогою формули, яка враховує зазначені параметри та використовує систему моделей для оцінки трудомісткості розробки ПЗ:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,}$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_{д}$ – витрати праці на підготовку документації.

Умовне число операторів (підпрограм) використовується для визначення складових витрат праці при розробці програмного забезпечення. Це число відображає кількість операторів або підпрограм, які потрібно реалізувати в програмі.

Умовне число операторів враховує різні типи операцій, такі як арифметичні операції, умовні вирази, цикли, звернення до функцій та інші дії, які виконуються в програмі. Це число може використовуватись для оцінки трудомісткості проекту, розрахунку часу, затраченого на розробку, а також для планування ресурсів і контролю витрат праці.

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів (1632);

C – коефіцієнт складності програми (1,25);

p – коефіцієнт корекції програми в ході її розробки (0,2).

$$Q = 1632 \cdot 1{,}25 \cdot (1 + 0{,}2) = 2448;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot K}, \text{ людино-годин}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,1);

$$t_u = \frac{2448 \cdot 1{,}2}{85 \cdot 1{,}1} = 31{,}42 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K};$$

$$t_a = \frac{2448}{20 \cdot 1,1} = 111,27 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K};$$

$$t_n = \frac{2448}{25 \cdot 1,1} = 89 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K};$$

$$t_{отл} = \frac{2448}{5 \cdot 1,1} = 445 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,2 \cdot t_{отл};$$

$$t_{отл}^{к} = 1,2 \cdot 445 = 534 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до};$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису;

$$t_{\partial} = \frac{Q}{(15...20)\cdot K};$$

$$t_{\partial p} = \frac{2448}{20\cdot 1,1} = 111,27 \text{ людино-годин.}$$

де $t_{\partial o}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{\partial o} = 0,75 \cdot t_{\partial p};$$

$$t_{\partial o} = 0,75 \cdot 111,27 = 83,45 \text{ людино-годин.}$$

$$t_{\partial} = 111,27 + 83,45 = 194,72 \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 31,42 + 111,27 + 89 + 445 + 194,72 = 921,41 \text{ людино-годин.}$$

Підсумково, розрахунки показали, що загальна трудомісткість розробки даного програмного забезпечення складає 921,41 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $Кпо$ включають у себе витрати на фінансове винагородження для виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного для налагодження даної програми на ЕОМ.

$$Кпо = З_{ЗП} + З_{МВ}, \text{ грн,}$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}\text{, грн,}$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 115 грн/год, то отримаємо:

$$З_{ЗП} = 921{,}41 \cdot 115 = 105\,962{,}15 \text{ грн}$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot С_{М}\text{, грн,}$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$С_{М}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 445 \cdot 18 = 8010 \text{ грн}$$

Звідси витрати на створення програмного продукту:

$$Кпо\ 105\,962{,}15 + 8010 = 113\,972{,}15 \text{ грн}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}\text{, мес.}$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 176 годин).

Витрати на створення програмного продукту:

$$T = \frac{921{,}41}{1 \cdot 176} = 5{,}23 \text{ міс.}$$

Вартість розробки нашого інтернет-магазину становить 113 972,15 грн. Орієнтовний час, потрібний для розробки, складає приблизно 5,23 місяців при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Цей термін включає час, необхідний для проведення досліджень, розробки алгоритму, дизайну та документування. Загальна кількість людино-годин, яку буде витрачено на розробку, складає 921,41.

ВИСНОВКИ

Метою даного проекту, що присвячений веб-сайту для бронювання квитків на культурні заходи, є розробка веб-орієнтованого додатку з використанням React.JS та Firebase в середовищі програмування Visual Studio Code.

Актуальність даної задачі пояснюється потребою у створенні інтернет-платформи для продажу квитків на різноманітні культурні заходи. Це ефективний спосіб розширити аудиторію клієнтів та уникнути витрат на відкриття фізичного магазину.

Розроблений додаток є інтернет-платформою, яка надає користувачам інтуїтивно зрозумілий інтерфейс для придбання квитків, реєстрації у системі клієнтів та керування особистими даними.

Сайт забезпечує наступні можливості:

1) Моніторинг наявних квитків, їх цін та опису.

2) Реєстрацію користувачів у системі.

3) Вхід у особистий обліковий запис, що дозволяє здійснювати покупки, редагувати особисту інформацію та поповнювати баланс.

4) Додавання квитків до кошика та їх видалення.

5) Для адміністраторів: можливість додавати нові квитки у систему безпосередньо через веб-інтерфейс, видаляти їх та редагувати.

6) Вкладки з інформацією про сайт, в яких надається загальна інформація про розробника.

Для збереження та обміну даними між клієнтською та серверною частинами використовується Firebase. Firebase є хмарною платформою, яка надає інструменти для збереження, синхронізації та обміну даними в реальному часі, а також для аутентифікації користувачів та управління даними.

Структура програми передбачає розробку веб-додатку з використанням мови програмування JavaScript. Код програми розробляється у середовищі Visual Studio Code та взаємодіє з базою даних Firebase.

У розділі, присвяченому економічному аспекту, було розраховано трудомісткість розробки програмного забезпечення, оцінено витрати на його створення та передбачений очікуваний час розробки. Визначено трудомісткість розробки програмного забезпечення 921,41 люд-год, та підраховані витрати щодо створення сайту-додатка - 113 972,15 грн і гаданий період розробки 5,23 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ReactJS. URL: https://reactjs.org/
2. Alex Banks and Eve Porcello. Learning React: Modern Patterns for Developing React Apps. — O'Reilly Media, 2017. — 350 с.
3. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. — O'Reilly Media, 2014. — 254 с.
4. Cay Horstmann. React for the Real World: Large-scale, Fast, and Scalable Web Applications. — Packt Publishing, 2020. — 432 с.
5. Robin Wieruch. The Road to React: Your journey to master plain yet pragmatic React.js. — Independently published, 2018. — 394 с.
6. Adam Freeman. Pro React 16. — Apress, 2019. — 1079 с.
7. JavaScript. URL: https://uk.javascript.info
8. Douglas Crockford. JavaScript: The Good Parts. — O'Reilly Media, 2008. — 176 с.
9. Eric Freeman, Elisabeth Robson. Head First JavaScript Programming: A Brain-Friendly Guide. — O'Reilly Media, 2014. — 704 с.
10. Marijn Haverbeke. Eloquent JavaScript: A Modern Introduction to Programming. — No Starch Press, 2018. — 472 с.
11. David Flanagan. JavaScript: The Definitive Guide. — O'Reilly Media, 2020. — 706 с.
12. Kyle Simpson. You Don't Know JS: Up & Going. — O'Reilly Media, 2015. — 88 с.
13. Firebase. URL: https://firebase.google.com/docs/web/setup?hl=en
14. Cloud Firestore. URL: https://firebase.google.com/docs/firestore?authuser=0&hl=en
15. Eric A. Meyer, Estelle Weyl. CSS: The Definitive Guide. — O'Reilly Media, 2017. — 1090 с.

16. Jon Duckett. HTML & CSS: Design and Build Websites. — Wiley, 2011. — 512 с.
17. David McFarland. CSS: The Missing Manual. — O'Reilly Media, 2015. — 718 с.
18. Rachel Andrew. The New CSS Layout. — Smashing Magazine, 2017. — 218 с.
19. Dan Cederholm. CSS3 For Web Designers. — A Book Apart, 2010. — 112 с.

ДОДАТОК А
КОД ПРОГРАМИ

```
!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Web site created using create-react-app" />

    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link

href="https://fonts.googleapis.com/css2?family=Montserrat:wght@100;400;500;600;700;900&
display=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.15.3/css/all.css"
      integrity="sha384-
SZXxX4whJ79/gErwcOYf+zWLeJdY/qpuqC4cAa9rOGUstPomtqpuNWT9wdPEn2fk"
      crossorigin="anonymous"
    />

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>Miller tickets | shop</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>

mport React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { Elements } from '@stripe/react-stripe-js';
import App from './App';
import { persistor, store } from './app/store';
import { stripePromise } from './utils/stripe/stripe.utils';
import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <BrowserRouter>
          <Elements stripe={stripePromise}>
            <App />
          </Elements>
```

```
        </BrowserRouter>
      </PersistGate>
    </Provider>
  </React.StrictMode>
);

import React, { useEffect } from 'react';
import { Routes, Route } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import { checkUserSession } from './features/user/userSlice';
import MainPage from './routes/main-page.component';
import Home from './routes/home/home.component';
import Shop from './routes/shop/shop.component';
import Checkout from './routes/checkout/checkout.component';
import About from './routes/about/about.component';
import Authentication from './routes/authentication/authentication.component';
import Contacts from './routes/contacts/contacts.component';

function App() {
  const dispatch = useDispatch();

  useEffect(() => {
    dispatch(checkUserSession());
  }, [dispatch]);

  return (
    <div className='App'>
      <Routes>
        <Route path='/' element={<MainPage />}>
          <Route index element={<Home />} />
          <Route path='events/*' element={<Shop />} />
          <Route path='checkout' element={<Checkout />} />
          <Route path='contacts' element={<Contacts />} />
          <Route path='about' element={<About />} />
          <Route path='sign-in' element={<Authentication />} />
        </Route>
      </Routes>
    </div>
  );
}

export default App;

import { useEffect } from 'react';
import DirectoryItem from '../directory-item/directory-item.component';
import { DirectoryContainer } from './directory.styles';
import { addCollectionAndDocuments } from '../../utils/firebase/firebase.utils';
import SHOP_DATA from '../../shop-data';

const categories = [
  {
    id: 0,
    title: 'Концерти',
    imageUrl: 'images/directory/pic1.jpg',
    route: 'events/концерти'
  },
  {
    id: 1,
    title: 'Спектаклі',
    imageUrl: 'images/directory/cap.jpg',
    route: 'events/спектаклі'
  },
```

```
  {
    id: 2,
    title: 'Ярмарок',
    imageUrl: 'images/directory/pic2.jpg',
    route: 'events/ярмарок'
  },

  {
    id: 3,
    title: 'Фестивалі',
    imageUrl: 'images/directory/pic9.jpg',
    route: 'events/фестивалі'
  },

  {
    id: 4,
    title: 'Виставки',
    imageUrl: 'images/directory/s6.jpg',
    route: 'events/виставки'
  }
];

const Directory = () => {
  useEffect(() => {
    addCollectionAndDocuments('categories', SHOP_DATA);
  });

  return (
    <DirectoryContainer>
      {categories.map(({ id, ...otherProps }) => (
        <DirectoryItem key={id} {...otherProps} />
      ))}
    </DirectoryContainer>
  );
};

export default Directory

import { Link, useNavigate } from 'react-router-dom';
import { DirectoryItemContainer } from './directory-item.styles';

const DirectoryItem = ({ title, imageUrl, route }) => {
  const navigate = useNavigate();
  const onNavigateHandler = () => navigate(route);

  return (
    <DirectoryItemContainer onClick={onNavigateHandler}>
      <h4>{title}</h4>
      <div>
        <Link to={route}>
          <img src={imageUrl} alt={title} />
        </Link>
      </div>
    </DirectoryItemContainer>
  );
};

export default DirectoryItem

import React from 'react';

function Balance({ userBalance }) {
```

```
  return (
    <span>{userBalance}</span>
  );
}

export default Balance;

import { useState, useEffect } from 'react';
import { useLocation } from 'react-router-dom';

import { IntroTitle, StyledIntro } from './intro.styles.js';
import Header from '../header/header.component.jsx';

const Intro = () => {
  const location = useLocation();

  const [isMain, setIsMain] = useState(false);

  useEffect(() => {
    if (location.pathname === '/') {
      setIsMain(true);
    } else {
      setIsMain(false);
    }
  }, [location]);

  return (
    <StyledIntro isMain={isMain}>
      <div className='container'>
        <Header />

        {isMain ? <IntroTitle>ЧАС ВІДВОЛІКТИСЯ</IntroTitle> : null}
      </div>
    </StyledIntro>
  );
};

export default Intro

import { useDispatch, useSelector } from 'react-redux';
import { signOutStart } from '../../features/user/userSlice';
import { selectCurrentUser } from '../../features/user/user.selector';

import { List, NavLink, RowList } from './nav-list.styles';

import { links } from '../../utils/consts';

export const LIST_DIRECTIONS = {
  row: 'row',
  column: 'column'
};

const getList = (direction = LIST_DIRECTIONS.column) =>
  ({
    [LIST_DIRECTIONS.column]: List,
    [LIST_DIRECTIONS.row]: RowList
  }[direction]);

const NavList = ({ direction }) => {
  const dispatch = useDispatch();
  const currentUser = useSelector(selectCurrentUser);
  const CustomList = getList(direction);
```

```
  const signOutHandler = () => {
    dispatch(signOutStart());
  };

  return (
    <CustomList>
      {links.map((link, index) => (
        <li key={index}>
          <NavLink to={link.to}>{link.label}</NavLink>
        </li>
      ))}

      <li>
        {currentUser ? (
          <NavLink onClick={signOutHandler} as={'span'}>
            Вийти
          </NavLink>
        ) : (
          <NavLink to='/sign-in'>Увійти</NavLink>
        )}
      </li>
    </CustomList>
  );
};

export default NavList

import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { setIsCartOpen } from '../../features/cart/cartSlice';
import CartPopupItem from '../cart-popup-item/cart-popup-item.component';

import {
  selectCartCount,
  selectCartItems,
  selectCartTotal,
  selectIsCartOpen
} from '../../features/cart/cart.selector';

import { useClickOutside } from '../../utils/useClickOutSide';

import {
  CartPopupContainer,
  CartPopupDiv,
  CartPopupList,
  CartPopupTotal
} from './cart-popup.styles';
import { useRef } from 'react';

const CartPopup = () => {
  const cartItems = useSelector(selectCartItems);
  const isCartOpen = useSelector(selectIsCartOpen);
  const cartTotal = useSelector(selectCartTotal);
  const cartCount = useSelector(selectCartCount);
  const dispatch = useDispatch();
  const cartRef = useRef();

  const onClickHandler = () => {
    dispatch(setIsCartOpen(!isCartOpen));
  };
```

```
  useClickOutside(cartRef, () => dispatch(setIsCartOpen(false)));

  return (
    <CartPopupContainer ref={cartRef}>
      <span onClick={onClickHandler}>Кошик</span>
      <div className='rate'>{cartCount}</div>

      <CartPopupDiv cartIsOpen={isCartOpen}>
        {cartItems.length ? (
          <>
            <h3 className='shopping-cart__title'>
              Нещодавно додано({cartCount})
            </h3>

            <CartPopupList>
              {cartItems.map((cartItem) => (
                <CartPopupItem key={cartItem.id} cartItem={cartItem} />
              ))}
            </CartPopupList>

            <CartPopupTotal>
              <div className='total__left'>Загалом:</div>
              <div className='total__right'>₴{cartTotal}</div>
            </CartPopupTotal>

            <Link className='check-out' to='/checkout'>
              Перейти до оплати
            </Link>
          </>
        ) : (
          <h3 className='shopping-cart__title'>Кошик порожній</h3>
        )}
      </CartPopupDiv>
    </CartPopupContainer>
  );
};

export default CartPopup;
```

```
import React from 'react';
import Socials from './socials/socials.component';

const year = new Date().getFullYear();

export default function Footer() {
 return (
  <footer className='footer'>
   <h5>Miller Tickets</h5>
   <p className='copy'>© {year} All Rights Reserved Miller Mykyta</p>
   <Socials marginTop='27px' />
  </footer>
 );
}
```

```
import { useSelector, useDispatch } from 'react-redux';
import { clearItemFromCart } from '../../features/cart/cartSlice';

import CloseBtn from '../close-btn/close-btn.component';
import {
  PopupItemContent,
  PopupItemImage,
  PopupListItem
```

```
} from './cart-popup-item.styles';
import { selectCartItems } from '../../features/cart/cart.selector';

const CartPopupItem = ({ cartItem }) => {
  const { name, price, imageUrl, quantity } = cartItem;
  const cartItems = useSelector(selectCartItems);
  const dispatch = useDispatch();

  const onClearItemHanlder = () => {
    dispatch(clearItemFromCart(cartItems, cartItem));
  };

  return (
    <PopupListItem>
      <PopupItemImage>
        <img src={imageUrl} alt={name} />
      </PopupItemImage>
      <PopupItemContent>
        <h4>{name}</h4>
        <article>
          {quantity} x <span>₴{price} </span>
        </article>
        <CloseBtn
          imgUrl={'/images/icons/close.png'}
          onClickHandler={onClearItemHanlder}
        />
      </PopupItemContent>
    </PopupListItem>
  );
};

export default CartPopupItem;

import { useSelector, useDispatch } from 'react-redux';
import { selectCartItems } from '../../features/cart/cart.selector';
import {
  addItemToCart,
  removeItemFromCart,
  clearItemFromCart
} from '../../features/cart/cartSlice';

import { CartItemContainer, CartItemImage, CartItemOptions } from './cart-item.styles';
import Button, { BUTTON_TYPES } from '../button/button.component';

const CartItem = ({ cartItem }) => {
  const { imageUrl, name, price, descr, quantity } = cartItem;

  const dispatch = useDispatch();
  const cartItems = useSelector(selectCartItems);

  const addItemHandler = () => dispatch(addItemToCart(cartItems, cartItem));
  const removeItemHandler = () => dispatch(removeItemFromCart(cartItems, cartItem));
  const clearItemHandler = () => dispatch(clearItemFromCart(cartItems, cartItem));

  return (
    <CartItemContainer>
      <CartItemImage>
        <img src={imageUrl} alt={name} />
      </CartItemImage>

      <CartItemOptions>
        <h1 className='checkout__title'>{name}</h1>
```

```
        <p className='checkout__descr'>{descr}</p>
        <div className='price checkout__price'>
          <span className='price_actual'>&#36;{price}</span>
        </div>

        <div className='single__options-wrapper'>
          <article className='product-qty'>
            <label htmlFor='quantity'> кількість: </label>
            <span>
              <button onClick={removeItemHandler} className='btn_dec-inc'>
                -
              </button>
              <input
                name='quantity'
                id='quantity'
                autoComplete='off'
                maxLength='3'
                value={quantity}
                readOnly
              />
              <button onClick={addItemHandler} className='btn_dec-inc'>
                +
              </button>
            </span>
          </article>
        </div>
        <Button buttonType={BUTTON_TYPES.rounded} onClick={clearItemHandler}>
          видалити
        </Button>
      </CartItemOptions>
    </CartItemContainer>
  );
};

export default CartItem;

import { Link } from 'react-router-dom';
import { categories } from '../../utils/consts';

import { StyledCategoriesList } from './categories-list.styles';

const CategoriesList = () => {
  return (
    <StyledCategoriesList>
      {categories.map(({ category, link }, idx) => (
        <li key={category + idx}>
          <Link to={link}>{category}</Link>
        </li>
      ))}
    </StyledCategoriesList>
  );
};

export default CategoriesList;

import ProductCard from '../product-card/product-card.component';

import {
  CategoryPreviewContainer,
  Title,
  Preview,
  StyledLink
```

```
} from './category-preview.styles';

const CategoryPreview = ({ title, products }) => {
  return (
    <CategoryPreviewContainer>
      <h2>
        <Title to={title}>{title.toUpperCase()}</Title>
      </h2>
      <Preview>
        {products
          .filter((_, idx) => idx < 4)
          .map((product) => (
            <ProductCard key={product.id} product={product} />
          ))}
      </Preview>
      <StyledLink to={title}>Розгорнути категорію &#8594; </StyledLink>
    </CategoryPreviewContainer>
  );
};

export default CategoryPreview;

import { FormInputLabel, Input, Group } from './form-input.styles';

const FormInput = ({ label, ...otherProps }) => {
  return (
    <Group>
      <Input {...otherProps} />
      {label && (
        <FormInputLabel
          shrink={Boolean(
            otherProps.value &&
              typeof otherProps.value === 'string' &&
              otherProps.value.length
          )}
        >
          {label}
        </FormInputLabel>
      )}
    </Group>
  );
};

export default FormInput;

import { useEffect, useState } from 'react';
import { Link, useLocation } from 'react-router-dom';

import Nav from '../nav/nav.component';
import CartPopup from '../cart-popup/cart-popup.component';
import { BurgerButton, HeaderWrapper, Logo, StyledHeader } from './header.styles';

const Header = () => {
  const [menuState, setMenu] = useState({ isOpen: false });
  const location = useLocation();

  const menuToggleHandler = () => {
    setMenu({
      isOpen: !menuState.isOpen
    });
  };
```

```
  const closeMenuHandler = () => {
    setMenu({
      isOpen: false
    });
  };

  useEffect(() => {
    setMenu({
      isOpen: false
    });
  }, [location]);

  return (
    <StyledHeader>
      <Logo>
        <Link to='/'>Miller Tickets</Link>
      </Logo>

      <Nav isOpen={menuState.isOpen} onClose={closeMenuHandler} />

      <HeaderWrapper>
        <BurgerButton onClick={menuToggleHandler}>
          <img src='/images/icons/menu.png' alt='menu' />
        </BurgerButton>

        <CartPopup />
      </HeaderWrapper>
    </StyledHeader>
  );
};

export default Header;

import React from 'react';
import NavList from '../nav-list/nav-list.component';

import { StyledNav } from './nav.styles';

import { LIST_DIRECTIONS } from '../nav-list/nav-list.component';
import CloseBtn, { CLOSE_BTN_TYPES } from '../close-btn/close-btn.component';

const Nav = ({ isOpen, onClose }) => {
  return (
    <StyledNav isOpen={isOpen}>
      <NavList direction={LIST_DIRECTIONS.row} />
      <CloseBtn
        onClickHandler={onClose}
        imgUrl={'/images/icons/close-white.png'}
        btnType={CLOSE_BTN_TYPES.white}
      />
    </StyledNav>
  );
};

export default Nav;

import { useDispatch, useSelector } from 'react-redux';
import { signOutStart } from '../../features/user/userSlice';
import { selectCurrentUser } from '../../features/user/user.selector';

import { List, NavLink, RowList } from './nav-list.styles';
```

```
import { links } from '../../utils/consts';

export const LIST_DIRECTIONS = {
  row: 'row',
  column: 'column'
};

const getList = (direction = LIST_DIRECTIONS.column) =>
  ({
    [LIST_DIRECTIONS.column]: List,
    [LIST_DIRECTIONS.row]: RowList
  }[direction]);

const NavList = ({ direction }) => {
  const dispatch = useDispatch();
  const currentUser = useSelector(selectCurrentUser);
  const CustomList = getList(direction);

  const signOutHandler = () => {
    dispatch(signOutStart());
  };

  return (
    <CustomList>
      {links.map((link, index) => (
        <li key={index}>
          <NavLink to={link.to}>{link.label}</NavLink>
        </li>
      ))}

      <li>
        {currentUser ? (
          <NavLink onClick={signOutHandler} as={'span'}>
            Вийти
          </NavLink>
        ) : (
          <NavLink to='/sign-in'>Увійти</NavLink>
        )}
      </li>
    </CustomList>
  );
};

export default NavList;

import { ProductCardBody, ProductCardContainer } from './product-card.styles';

import Button, { BUTTON_TYPES } from '../button/button.component';
import { useDispatch, useSelector } from 'react-redux';
import { selectCartItems } from '../../features/cart/cart.selector';
import { addItemToCart } from '../../features/cart/cartSlice';

const ProductCard = ({ product }) => {
  const { name, imageUrl, price, descr } = product;

  const dispatch = useDispatch();
  const cartItems = useSelector(selectCartItems);

  const addItemHandler = () => {
    dispatch(addItemToCart(cartItems, product));
  };
```

```
  return (
    <ProductCardContainer>
      <img src={imageUrl} alt={name} />

      <ProductCardBody>
        <h5>
          <span title={name}>
            {name.length > 12 ? `${name.slice(0, 12)}...` : name}
          </span>
          <span
            style={{
              color: '#ff0d40'
            }}
          >
            {price} ₴
          </span>
        </h5>
        <p title={descr}>{descr}</p>
        <Button buttonType={BUTTON_TYPES.rounded} onClick={addItemHandler}>
          У кошик
        </Button>
      </ProductCardBody>
    </ProductCardContainer>
  );
};

export default ProductCard;

import { useState } from 'react';
import { useDispatch } from 'react-redux';

import FormInput from '../form-input/form-input.component';
import Button, { BUTTON_TYPES } from '../button/button.component';

import { SignInContainer, ButtonsContainer } from './sign-in-form.styles';
import {
  googleSignInStart,
  emailSignInStart
} from '../../features/user/userSlice';

const defaultFormFields = {
  email: '',
  password: ''
};

const SignInForm = () => {
  const [formFields, setFormFields] = useState(defaultFormFields);
  const { email, password } = formFields;
  const dispatch = useDispatch();

  const resetFormFields = () => {
    setFormFields(defaultFormFields);
  };

  const signInWithGoogle = () => {
    dispatch(googleSignInStart());
```

```
};

const handleSubmit = async (event) => {
  event.preventDefault();

  try {
    dispatch(emailSignInStart({ email, password }));
    resetFormFields();
  } catch (error) {
    console.log('user sign in failed', error);
  }
};

const handleChange = (event) => {
  const { name, value } = event.target;

  setFormFields({ ...formFields, [name]: value });
};

return (
  <SignInContainer>
    <h2>Вже є акаунт?</h2>
    <span>Увійти з вашим email та паролем</span>
    <form onSubmit={handleSubmit}>
      <FormInput
        label='Email'
        type='email'
        required
        onChange={handleChange}
        name='email'
        value={email}
      />

      <FormInput
        label='Пароль'
        type='password'
        required
        onChange={handleChange}
        name='password'
        value={password}
      />
      <ButtonsContainer>
        <Button buttonType={BUTTON_TYPES.black} type='submit'>
          Увійти
        </Button>
        <Button type='button' onClick={signInWithGoogle}>
          Увійти з Google
        </Button>
      </ButtonsContainer>
    </form>
  </SignInContainer>
```

```
  );
};

export default SignInForm;
```

ДОДАТОК Б
ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В
ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ Міллєр.docx	Пояснювальна записка до дипломного проекту. Документ Word.
Диплом_ Міллєр.pdf	Пояснювальна записка до дипломного проекту в форматі PDF
Програма	
Program.zip	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_Міллєр.ppt	Презентація дипломного проекту