Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка та організація порталу комп'ютерного клубу з можливістю проведення турнірів

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-20-1
_____ / М. С. Рубай /

Керівник
кваліфікаційної роботи _____ / Н. О. Карабут /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. Кафедри

_____ А. М. Стрюк

«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20-1 Рубаю Микиті Сергійовичу

1. Тема: Розробка та організація порталу комп'ютерного клубу з можливістю проведення турнірів затверджено наказом по КНУ від № 275с від 15 квітня 2024 р.
2. Термін подання студентом закінченої роботи: «01» червня 2024р.
3. Вихідні дані по роботі: розробити зручний портал комп’ютерного клубу який має у собі послуги управління обліковим записом, бронювання, переглядом інформації про послуги та проведення турнірів.
4. Зміст пояснювальної записки(перелік питань, що їх треба розробити): проаналізувати методи створення веб-порталів закладів комп’ютерних клубів та їх функціонал, створити дизайн, розробити інтерфейс та зробити програмну реалізацію, протестувати готовий проект.
5. Перелік ілюстрованого матеріалу: скріншоти сайтів аналогів, схематична таблиця структури та дизайну сайту, скріншоти реалізації сайту, реалізації бази даних та структури проекту.

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Аналіз існуючих аналогів та методів	17.03.2024 – 02.04.2024
2	Підготовка матеріалів першого розділу роботи	03.04.2024 – 10.04.2024
3	Вибір архітектури та сторонніх рішень для розробки	11.04.2024 – 20.04.2024
4	Підготовка матеріалів другого розділу роботи	21.04.2024 – 28.04.2024
5	Розробка дизайну та проектування програмного забезпечення	29.04.2024 – 13.05.2024
6	Підготовка матеріалів третього розділу роботи	14.05.2024 – 17.05.2024
7	Розробка програмного забезпечення	18.05.2024 – 24.05.2024
8	Доробка та виправлення дизайну під програмне забезпечення	25.05.2024 – 28.05.2024
9	Тестування та налагодження	29.05.2024 – 04.06.2024
10	Оформлення пояснювальної записки	05.06.2024 – 11.06.2024

Дата видачі завдання: «01» квітня 2024 р.

Студент _____ / М. С. Рубай /

Керівник роботи _____ / Н. О. Карабут /

ЗМІСТ

перелік умовних позначень

БД – база даних

ПЗ – програмне забезпечення

ООП - об’єктно-орієнтоване програмування

Скріншот – знімок екрану

Зона – зона обладнання та перефирії

Айді – ідентифікатор або унікальний код

РЕФЕРАТ

Пояснювальна записка: 46 с., 24 рис., 2 дод., 15 джерел.

Метою кваліфікаційної роботи є створення порталу комп’ютерного клубу для онлайн бронювання, надання інформації та проведення турнірів.

У роботі проведено аналіз наявних порталів комп’ютерних клубів, було досліджено функціональність, дизайн та досвід користувача. За допомогою аналізу було сформовано основні вимоги до розробки вебпортал.

Спроектований та створена структура сайту та його архітектура, включаючи функціональність, дизайн та досвід користувача. Однією з найважливіших частин роботи є функціональність.

Створена реалізація програмного забезпечення за допомогою сучасних технологій та інструментів розробки. Портал був протестований на усі функції, що були реалізовані.

REFERENCE

Explanatory note: 46 p., 24 pic., 2 attachments, 15 sources.

The purpose of the qualification work is to create a computer club portal for online booking, providing information and holding tournaments.

The paper analyzes existing computer club portals, examines the functionality, design and user experience. The analysis helped to formulate the main requirements for the development of the web portal.

The site structure and architecture, including functionality, design, and user experience, were designed and created. One of the most important parts of the work is functionality.

We created a software implementation using modern technologies and development tools. The portal was tested for all the functions that were implemented.

ВСТУП

Є багато місць використання комп’ютерів, за останній час йде тенденція розвитку та створення комп’ютерних клубів які потребуюсь у веб-сервісі для обслуговування. Створення зручного веб-сервісу для використання адміністратором та користувачем є актуальною темою на вимогу бізнесу у цьому секторі.

Зручність, ефективність та актуальність - ось три ключові принципи у контексті створення веб-сайту. Однак важливо проаналізувати існуючі аналоги веб-сервісів обслуговування клієнтів у цьому секторі, виявити проблеми та порівняти їх с аналогами, щоб створити якісне та актуальне рішення.

Метою цього дослідження є порівняння одного з порталів комп’ютерного клубу з метою виявлення та ретельного вивчення її проблемних аспектів. Отримані результати будуть порівняні з аналогічними рішеннями, доступними на ринку. В рамках даної роботи буде створено веб-портал, який стане незамінним інструментом для власників комп'ютерних клубів. Завдяки цьому порталу вони зможуть суттєво підвищити ефективність роботи, оптимізувати витрати на вирішення проблем, а також значно покращити якість та зручність користування для адміністраторів та клієнтів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз професійної області проблеми

Власники комп'ютерних клубів стикаються з низкою проблем, пов'язаних з управлінням та обслуговуванням своїх закладів через вебпортал. Однією із проблем є неефективне управління клубом через відсутність порталу для користувача та адміністратора, через який можливо уникнути зайвих речей.

Також існує проблема неефективного управління клубом, через яке може бути складно відстежити бронювання, оплату та зайнятість комп’ютерів.

Проблема низької якості та комфорту використання для адміністраторів та користувачів. Потрібно використовувати сучасні інтерфейси для уникнення незручностей та відсутність самообслуговування, може змусити користувачів звертатись до адміністратора з зайвими питаннями.

Також важливою проблемою є безпека та приватність, тому важливо забезпечити належний захист конфіденційних даних користувачів на сайті.

Аналізуючи ці проблеми, можна зробити висновок про те що вони вливають на відносини між адміністраторами та користувачами, що може понести прибуткові та репутаційні втрати.

1.2 Аналіз існуючих аналогів

У наш час все більш стають популярні комп'ютерні клуби, які потребують гарний та привабливий сучасний сайт та зрозумілий інтерфейс зі зручними функціями. Проте для початку розробки сайту потрібно проаналізувати цю галузь для виявлення сучасних тенденцій, переваг та недоліків. У цьому аналізі будуть розглянуті кілька популярних аналогів з міста:

- Веб-сайт комп'ютерний клубу LEGION (Рис. 1.1)

Рисунок 1.1 Скріншот веб-сайту комп'ютерний клубу LEGION

- Веб-сайт комп'ютерний клубу CUBE (Рис. 1.2)

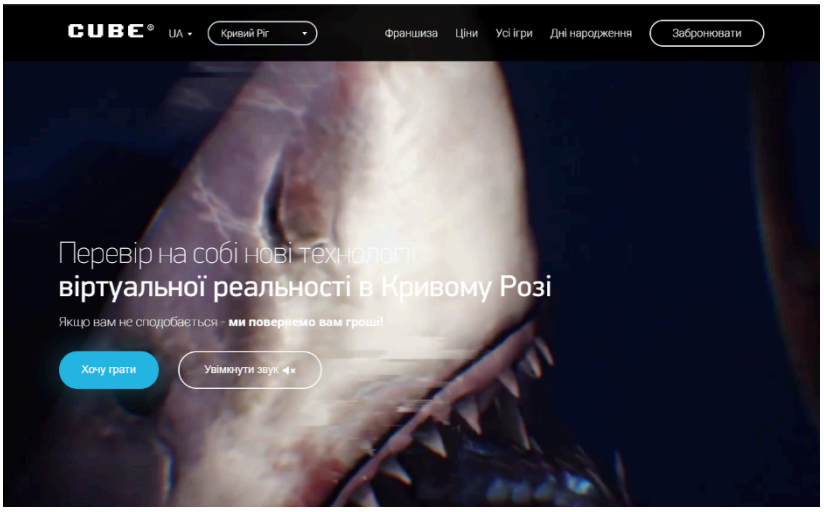

Рисунок 1.2 Скріншот веб-сайту комп'ютерний клубу CUBE

- Цільова аудиторія: Враховуючи, що цільовою аудиторією сайту є геймери, дизайн сайту має відігравати важливу роль у створенні позитивного досвіду користувача та залученні аудиторії.
- Послуги: На сайтах можна побачити як чітко та зрозуміло оформлюється інформація про ціни, послуги, обладнання.
- Привабливі функції: Однією з привабливих функцій сайту CUBE є можливість бронювання обладнання на конкретну дату.
- Дизайн: На сайтах використовується сучасний дизайн який використовую скролінг блоків з наданням потрібної інформації
- Контактна інформація: Важливим є правильне оформлення розділу контактної інформації яке в собі має телефон, адресу, соціальні мережі.

1.3 Формулювання актуальності і постановка задач кваліфікаційної роботи

Розробка вебпорталу для комп’ютерного клубу – одна з вузько спеціалізованих задач, але є актуальною в сучасному суспільстві як приклад зручної взаємодії між працівниками та користувачами. Швидкі темпи життя змушують людей не шукати собі зайвих черг чи незручностей у сфері обслуговування.

Хоча за кожним закладом сучасних комп’ютерних клубів вже стоїть вебпортал, але є потреба у задоволені нових закладів. Мета кваліфікаційної роботи полягає в розробці сучасного вебпорталу комп’ютерного клубу, який буде задовольняти користувача та адміністрацію. Це містить аналіз сучасних задач перед порталом обслуговування користувачів та впровадження актуальної задачі як проведення турніру між користувачами у комп’ютерному клубі.

Постановка задач:

1. Розробка дизайну за допомогою онлайн сервісу Figma
2. Створення таблиць в базі даних за допомогою MangoDB Compass, яка використовується для реєстрації, авторизації, бронювання та проведення турнірів
3. Створення форм реєстрації та входу
4. Створення можливості бронювання онлайн
5. Створення можливості проведення турніру онлайн
6. Реалізація дизайну сайту, зображень, анімацій для зручності за допомогою мови програмування React та мови сторінок Css.

2. ПРОЕКТУВАННЯ ПЗ ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1 Вибір сторонніх рішень для реалізації задач кваліфікаційної роботи

Figma - це потужний інструмент, який може використовуватися для створення UI та прототипів для вебсайтів, мобільних додатків та інших цифрових продуктів.

Visual Studio Code - це безкоштовний редактор коду з відкритим кодом, розроблений компанією Microsoft. VS Code є популярним вибором для розробників завдяки своїм численним функціям та розширенням,

Node.js - це середовище виконання JavaScript з відкритим кодом, побудоване на рушії V8 від Google. Це дозволяє використовувати JavaScript не лише для написання вебсторінок, але й для створення серверних застосунків, мережевих інструментів та інструментів командного рядка.

Npm (скорочення від Node Package Manager) - це менеджер пакетів для мови програмування JavaScript. Він використовується для встановлення, оновлення та видалення пакетів (бібліотек) в проектах Node.js. npm є частиною екосистеми Node.js і поставляється разом з встановленням Node.js.

MongoDB Compass - це безкоштовний, з відкритим кодом графічний інтерфейс користувача (GUI) для роботи з базами даних MongoDB. Він полегшує візуалізацію, навігацію, дослідження та керування даними MongoDB.

MongoDB - це гнучка, документо-орієнтована система керування базами даних (NoSQL), яка використовується для зберігання великих обсягів неструктурованих або напівструктурованих даних. MongoDB зберігає дані в JSON-подібних документах, що робить її більш гнучкою та масштабованою.

React - це JavaScript-бібліотека для створення інтерфейсів користувача (UI). Вона дозволяє розробникам створювати декларативні, динамічні та багаторазово використовувані UI, використовуючи компоненти.

2.2 Розробка структурної та функціональної схеми

Аналіз: Спочатку потрібно дослідити цільову аудиторію та їх очікування за сучасними стандартами. Це можна зробити за допомогою дослідження рішень конкурентів та відбиранню корисних, ефективних рішень. Наприклад вирішення які потрібні функції чи як структурувати інформацію на сайті.

За допомогою аналізу, на сайті потрібно реалізувати наступне:

- Меню навігації сайту
- Функціонал для взаємодії з аккаунтом
- Місце для комерційної інформації
- Кнопку для бронювання
- Місце для контактної інформації
- Місце та функціонал для взаємодії з турнірами

Проектування: За допомогою аналізу та продумування логіки треба розробити концепт інтерфейсу та взаємодії користувача з ним. Після додаткового аналізу конкурентів та креативної ідеї можна починати проектування візуального інтерфейсу. На великому макеті можна зробити структурні блоки з компонентами на сайті. За допомогою проектування можна розробити структурну та функціональну схеми.

Розробка структурної схеми(Рис. 2.1):

- Меню навігації сайту
- Ділянка взаємодії з обліковим записом
- Шапка сайту (Головне зображення та перша інформація що зустрічає користувача)
- Блок з зонами (Комплектуючі, периферія, монітори тощо)
- Блок з цінами.
- Блок зі списком ігор.
- Блок для взаємодії з турніром

Рисунок 2.1 Приклад структурної схеми сайту

Також блок цін повинен містити таблицю та кнопку для забронювання. (Рис. 2.2)

Назва зони	Кількість годин 1	Кількість годин 2	Кількість годин 3
Зона 1	Ціна 1.1	Ціна 1.2	Ціна 1.3
Зона 2	Ціна 2.1	Ціна 2.2	Ціна 2.3
Зона 3	Ціна 3.1	Ціна 3.2	Ціна 3.3
	Кнопка		

Рисунок 2.2 Приклад схеми блоку цін

3. РЕАЛІЗАЦІЯ ПЗ ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

3.1 Розробка та реалізація БД

Розробка БД є важливим та складним процесом для взаємодії з сайтом. БД у кваліфікаційній роботі використовується для входу / реєстрації, бронювання та створення турнірів. Процес створення БД відбувався в MangoDB Compass, саму БД я назвав “computerClub”, для кращого розуміння я поділю процес створення БД на прості етапи:

1) Перший етап – це створення бази даних у обраному середовищі.
2) Другим етапом є створення колекцій в базі даних. В моїй роботі всього їх буде три(Рисунок 3.1).
    1) Перша колекція має назву users та вона буде відповідати за облікові записи користувачів.
    2) Друга колекція має назву bookings та відповідає за бронювання зони для користувача
    3) Третя колекція має назву turnirs та слугує для проведення турнірів

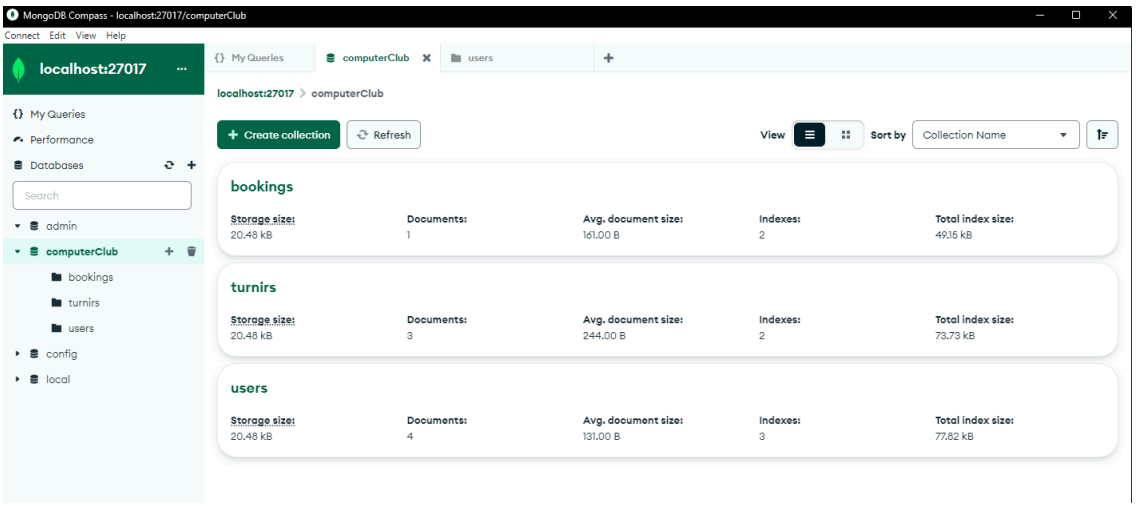

Рисунок 3.1 Скрішнот бази даних MongoDB Compass

3) Третім етапом є створення кількості стовпців у таблицях, позначення імен кожного стовпця та позначення типів даних у стовпцях, також створення унікального ObjectId. (Рисунки 3.2, 3.3, 3.4)

Рисунок 3.2 Скріншот колекції users

Рисунок 3.3 Скріншот колекції turnirs

Рисунок 3.4 Скріншот колекції bookings

4) Четвертим етапом э взаємодія користувачів з базою даних. Тому що користувачі будуть додавати нові дані до таблиць. Через сайт будуть автоматично поповнюватись усі колекції, якщо користувач буде взаємодіяти з функціоналом сайту.

Буде 3 колекції:

Booking:

- _id – унікальний айді об’єкту у mangoDB (ObjectId);
- zone – Назва зони (string);
- hours – кількість годин (int);
- price – ціна (int);
- userId – унікальний айді який взятий з колекції users;
- userEmail – пошта користувача (string);
- date – дата на яку заброньовано (date);
- createdAt – дата створення запиту (date);

Turnirs:

- _id - унікальний айді об’єкту у mangoDB (ObjectId);
- pairs – масив який містить у собі назви команд та переможців(array);
- turnirName – назва турніру (string);
- uniqueCode – унікальний код турніру (string);
- createdAt – дата створення турніру (date);

users:

- _id - унікальний айді об’єкту у mangoDB (ObjectId);
- Username – Ім’я користувача (string);
- Email – пошта користувача (string);
- Password – пароль користувача (string);
- Role – роль користувача (string);
- createdAt – дата створення облікового запису (date);
- updatedAt – дата оновлення даних облікового запису (date);

Тепер після завершення створення БД, потрібно її під’єднати до сайту і зробити, щоб дані могли як вноситися, так і змінюватись, наприклад при реєстрації, зміні пароля, чи створюватись при бронюванні, тому потрібно реалізувати візуальний інтерфейс, для того, щоб користувачу було куди вводити дані та реалізувати функції взаємодії з інтерфейсом.

3.2 Реалізація візуального інтерфейсу

На основі проектування сайту, можна почати створення візуального інтерфейсу сайту за допомогою допоміжних інструментів. Сучасні інструменти та додатки набагато покращують досвід зі створення сайту та його креативного інтерфейсу.

На основі проектування було реалізовано наступні структурні блоки:

- Меню навігації (Було вирішено додати кнопку входу з правого боку).
- Шапка сайту. (Рис. 3.5)

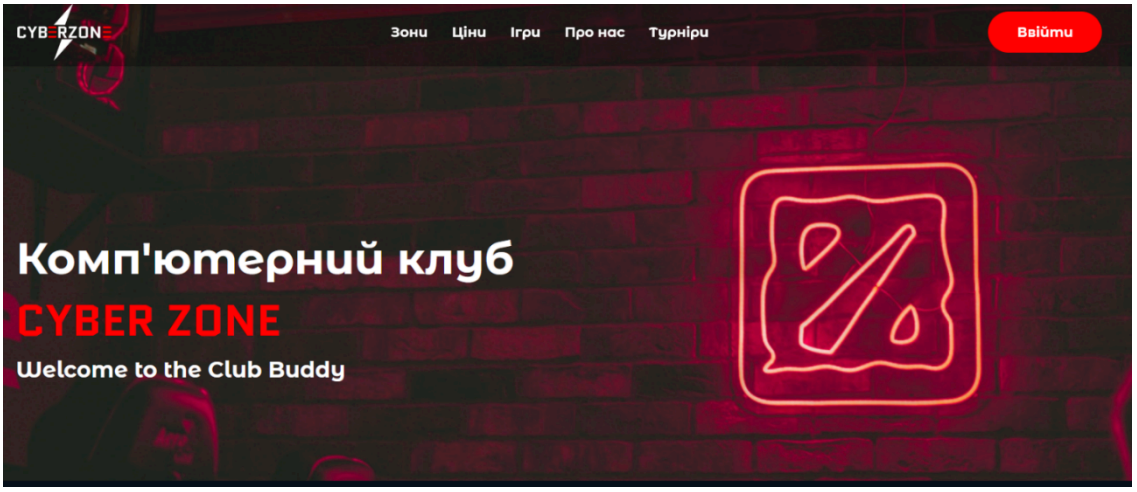

Рисунок 3.5 Скріншот шапки сайту та меню навігації

- Блок з зонами про інформацію комплектуючих було вирішено зробити за допомогою горизонтального списку. (Рис. 3.6)

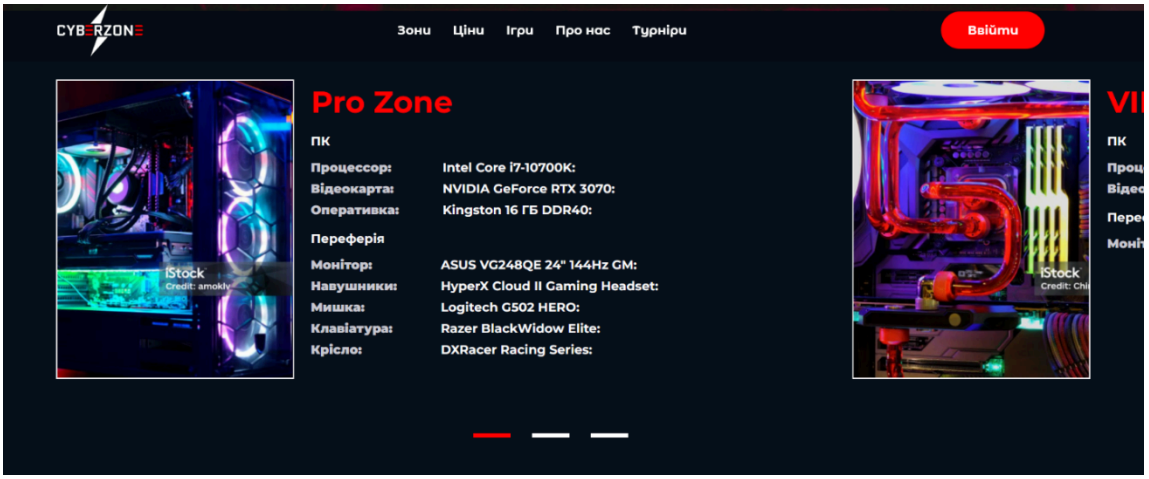

Рисунок 3.6 Скріншот блоку зон обладнання

- Блок з цінами було вирішено створити за допомогою таблиці та додати кнопку бронювання (Клітинки таблиці можна вибирати). (Рис. 3.7)

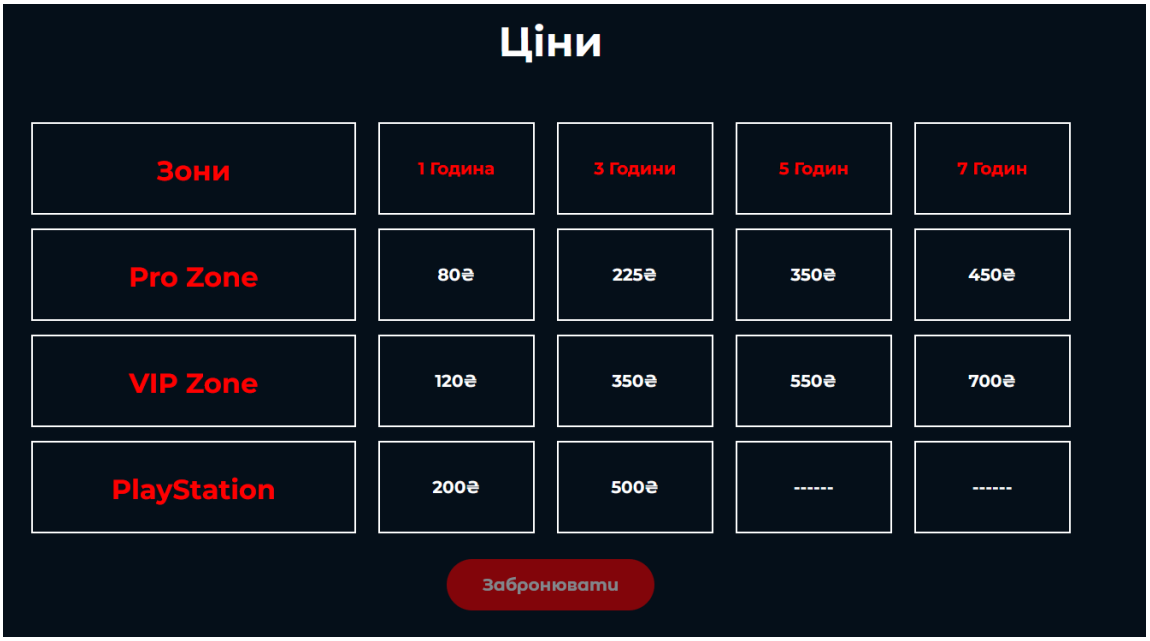

Рисунок 3.7 Скріншот блоку цін

- Блок з іграми. Черещ те, що блок з іграми може доповнюватись або скорочуватись, було вирішено зробити його вертикальним списком. (Рис. 3.8)

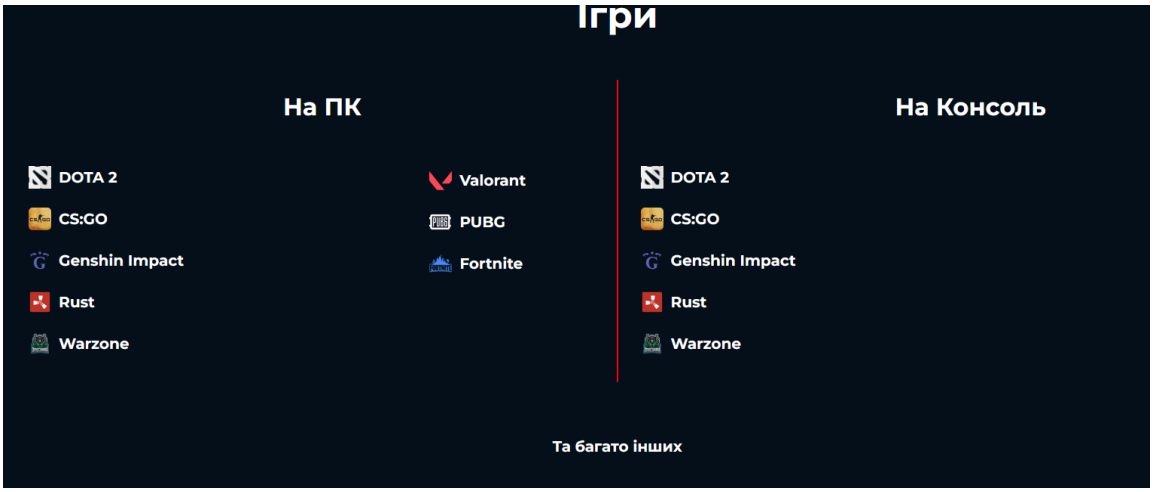

Рисунок 3.8 Скріншот блоку ігор

Блок з контактною інформацією. (Рис. 3.9)

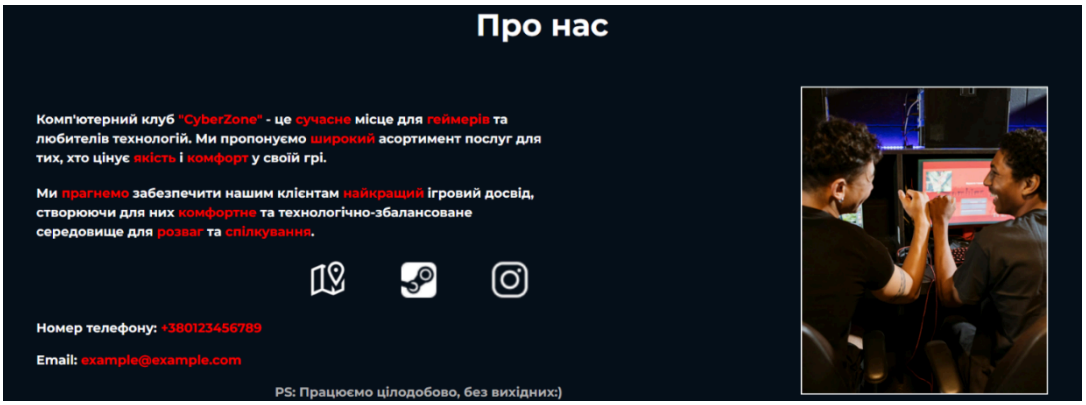

Рисунок 3.9 Скріншот блоку

Місце для проведення турнірів є динамічним, тому цей блок є різним в залежності від стану

Перший стан (Рис. 3.10) – користувач не авторизований

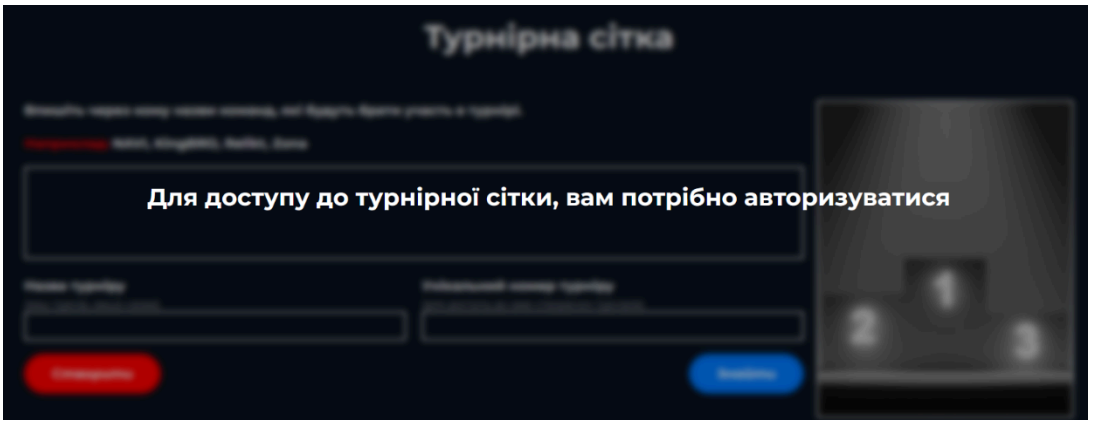

Рисунок 3.10 Скріншот частини сайту, відповідає за проведення турніру у стані коли користувач не авторизований

Другий стан (Рис. 3.11) – користувач авторизовний

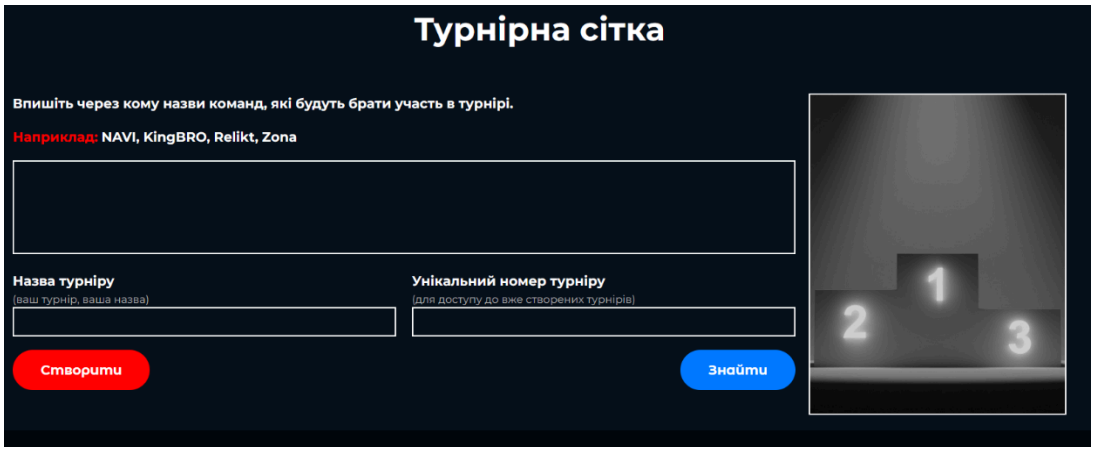

Рисунок 3.11 Скріншот частини сайту, відповідає за проведення турніру у стані коли користувач авторизований

Третій стан (Рис. 3.12)– користувача приєднано до турніру

Рисунок 3.12 Скріншот частини сайту, відповідає за проведення турніру у стані приєднання до турніру

3.3 Реалізація ПЗ

Реалізація реєстрації

1) При натисканні у вікні входу кнопки реєстрація, відкривається вікно реєстрації що містить 3 поля: нікнейм(login), пошта(email), пароль(password). Для початку, при введених даних у поля, при завершені їх обробки (наприклад клік на інше поле, що зробить його не активним), для кожного поля є своя валідація.
2) По-перше, поле не може бути пустим, при відсутності символів у полях login, email чи паролю, над кожним полем буде виведено повідомлення “Пошта / Нікнейм / Пароль не може бути пустим”.
3) Поле нікнейму перевіряється на довжину від 3 до 15 символів за допомогою .length, при неправильній довжині буде виведено повідомлення "Нікнейм має містити від 3 до 16 символів". Також поле перевіряється на некоректні символи за допомогою .test, якщо поле містить їх буде виведено повідомлення "Нікнейм містить некоректні символи".
4) Поле пошти має наступну валідацію, для початку створюємо новий параметр reEmeil, який переводиться до нижнього реєстру та перевіряється на коректність за допомогою .match (наприклад відсутність символів до @ та відсутність цифр після @), за некоректних даних виводиться повідомлення "Некоректна пошта"
5) Поле паролю має валідацію яка потребує довжини пароля більшу за 8 символів(.length) та мати що найменше 1 літеру(.search(/[a-z]/i)) та 1 цифру(.search(/[0-9]/)), за цих помилок буде виведено притаманне повідомлення "Ваш пароль повинен містити щонайменше 8 символів"/"Ваш пароль повинен містити принаймні одну літеру"/"Ваш пароль повинен містити хоча б одну цифру".
6) При натисканні кнопки реєстрація далі відбувається POST запит за допомогою .post, який передає дані до backend, де відбувається підключення до бази даних за допомогою .connect, створюється тіло

для колекції, та за допомогою .findOne перевіряється чи існує такий нікнейм чи пошта у базі даних, якщо буде знайдений збіг, буде повернуто повідомлення "User already exists", якщо збігу не було дані зберігаються до бази даних за допомогою .save, та виводиться повідомлення "User registered successfully". Якщо буде якась помилка при створенні користувача є запобіжник який відловлює помилки через catch (err), який виведе повідомлення "Failed to register user". Тепер ми маємо готове вікно з формою для реєстрації. (Рис. 3.13)

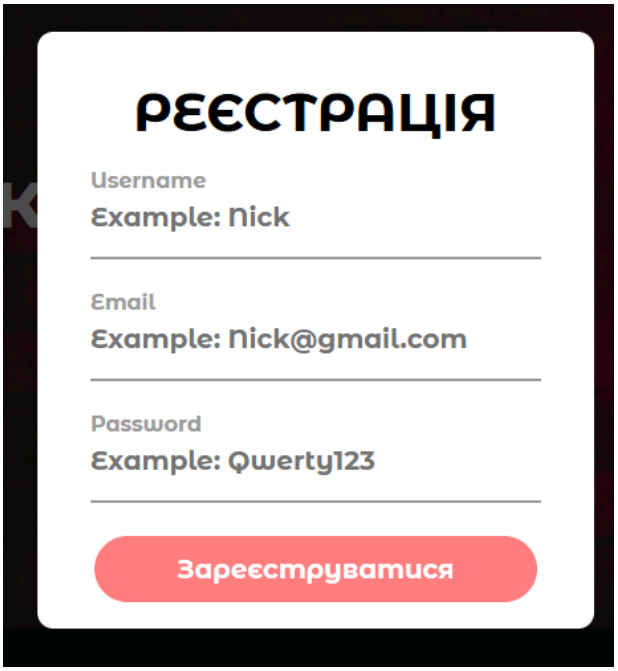

Рисунок 3.13 Скріншот вікна реєстрації

Реалізація входу

1) Після натискання кнопки “Ввійти” у меню сайту, відкривається вікно, що має містить у собі 2 поля. Перше поле призначене для введення нікнейму, чи пошти, друге поле для паролю. Також під ними є кнопка “Ввійти”.
2) При натисканні кнопки “Ввійти” виконується POST запит за допомогою .post, після чого перевіряється збіг по одному з полів колекції users через .findOne, а саме чи існує запис с таким нікнеймом

чи поштою, якщо дані не знайшлись у колекції користувачу виводиться повідомлення "Неправильний логін або пошта".

3) Коли дані збіглись по одному зі стовпців (пошта або нікнейм), йде перевірка пароля і якщо пароль не збігається виводиться повідомлення "Неправильний пароль".
4) Якщо на якомусь етапі виникла помилка чи сервер перестав відповідати, за допомогою catch відловлюється помилки та виводиться повідомлення "Помилка сервера".
5) Якщо поля збіглись, то на сайті змінюються параметри setLoginModalActive(false) – зникає кнопка “ввійти”. setIsAuthenticated(true) – з’являється інформація про користувача та додається кнопка “вийти”.

Тепер ми маємо готове вікно входу до сайту. (Рис. 3.14)

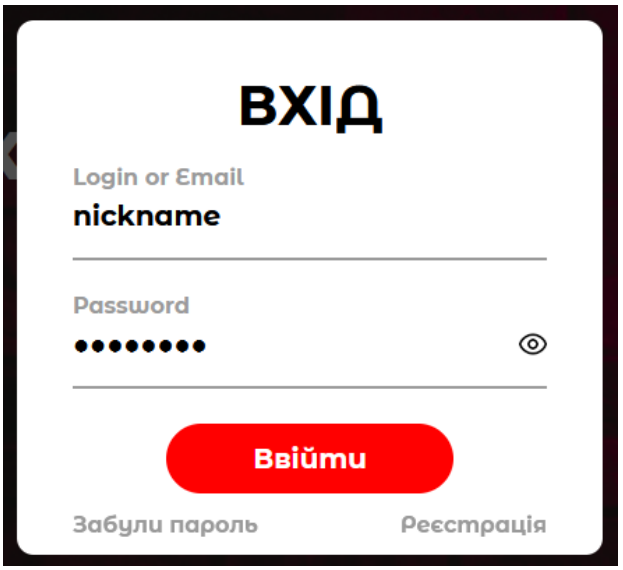

Рисунок 4.14 Скріншот вікна входу

Реалізація зміни пароля

У вікні інформації про користувача є кнопка “змінити пароль”, яка відкриває вікно для зміни пароля, вікно має у собі 3 поля: Поточний пароль, новий пароль та підтвердження нового пароля. Також вікно має кнопку для підтвердження зміни пароля

1) При натисканні кнопки спочатку йде валідація даних з полів, спочатку перевіряється чи містить новий пароль довжину більшу за 8 символів(.length) чи має він що найменше 1 літеру(.search(/[a-z]/i)) та 1 цифру(.search(/[0-9]/)), за цих помилок буде виведено притаманне повідомлення "Ваш пароль повинен містити щонайменше 8 символів"/"Ваш пароль повинен містити принаймні одну літеру"/"Ваш пароль повинен містити хоча б одну цифру".
2) Далі перевіряється чи збігається поле нового паролю та підтвердження нового паролю, якщо паролі не збігаються, виводиться повідомлення "Новий пароль та підтвердження пароля не співпадають".
3) Створюється POST запит, який бере поточний обліковий запис та порівнює поточний пароль та поле поточного пароля і якщо вони не збігаються то виводиться "Неправильний поточний пароль", при успішній зміні пароля, користувач отримає повідомлення "Пароль успішно змінено".
4) Якщо на якомусь етапі виникла помилка чи сервер перестав відповідати, за допомогою catch відловлюється помилки і виводиться повідомлення "Помилка сервера".

Після цього ми маємо готове вікно для зміни пароля. (Рис. 3.15)

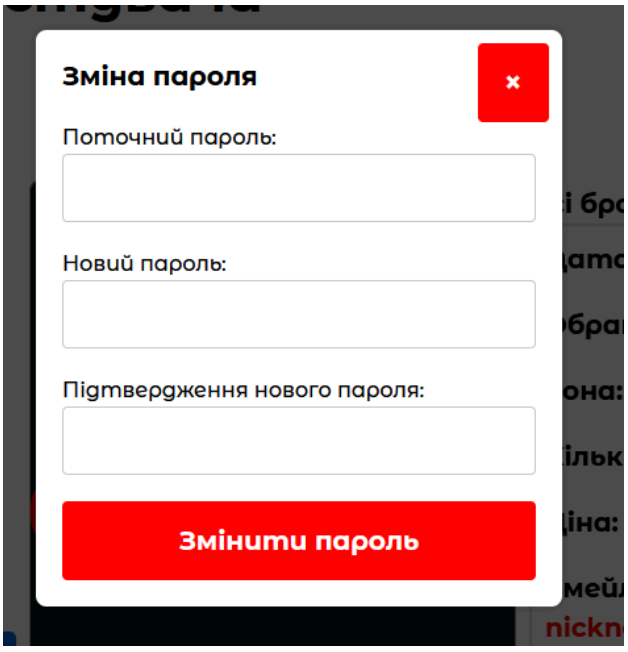

Рисунок 3.15 Скріншот вікна зміни паролю

Реалізація бронювання

Бронювання має такі функції створення бронювання, перегляд бронювання та видалення бронювання. Перегляд та видалення бронювання будуть розглянуті у реалізації перегляду профілю користувача.
Роздивимось як створюється бронювання:

1) За активного користувача стає активна кнопка кнопка “забронювати”, та з’являється можливість вибрати клітинку з таблиці.
2) Після натискання кнопки “забронювати”, відкривається вікно де виводиться інформація про обраний товар (Зона, кількість годин, ціна) та з’являється можливість вибрати дату за допомогою getDate з HTML, по замовчуванню стоїть поточна дата, у таблиці можна обрати дату лише на 3 дні вперед за допомогою maxDate = today.getDate() + 3, якщо користувач вказав дату більше ніж на 3 дні, виводиться повідомлення "Ви не можете забронювати на минуле число або більше ніж на 3 дні вперед."
3) Після за допомогою запиту POST перевіряється чи існують інші бронювання та якщо їх загальна кількість перевищує 10 годин на день, користувачу виведеться помилка "Ви не можете забронювати більше ніж 10 годин на день.", якщо сервер перестав відповідати, користувачу буде виведено помилку “помилка серверу”.
4) Якщо все пройшло успішно вікно закриється та у профілі користувача з’явиться помітка на даті яку він обрав.

Тепер у нас є готове вікно бронювання. (Рисунок 3.16)

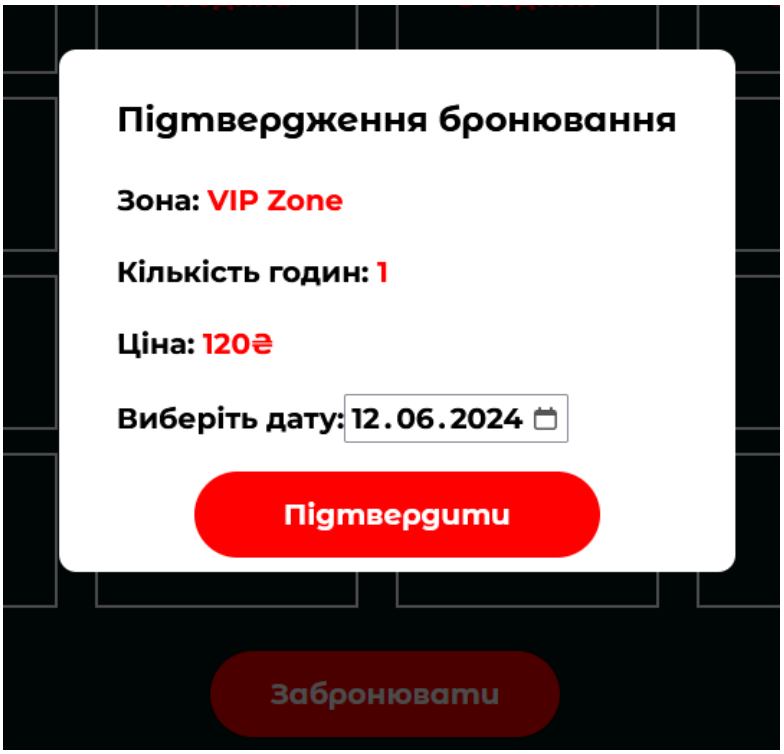

Рисунок 5.16 Скріншот вікна бронювання

Реалізація профілю користувача:

При натисканні на профіль :

1) При натисканні на профіль користувача створюється POST запит, який бере інформацію про поточний обліковий запис з колекції users – ім’я користувача, пошту та його роль, передивляється колекцію bookings та створює календар з поміченими датами бронювання(Через find, беруться усі бронювання користувача та беруться параметри з поля date у колекції booking, далі календар помічає усі унікальні дати). Також у профілі користувача є кнопка зміни пароля та кнопка перегляду усіх забронювань.
2) При перегляді усіх забронювань, створюється GET запит, який за допомогою find, переглядає усі бронювання користувача* за всі дати та виводить усю інформацію про них у вигляді списку.

3) При натисканні на окрему дату у календарі, створюється GET запит, який за допомогою find, переглядає усі бронювання користувача за окрему дату та виводить усю інформацію про них у вигляді списку.

*Якщо через базу даних, профіль отримав роль admin – він отримає усі бронювання на кожну дату незалежно від своїх бронювань.

*Якщо користувач має роль admin, він може виконувати видалення бронювань, при виведенні бронювання, буде додана іконка корзини, при натисненні кнопки йде DELETE запит, який за вибраного бронювання, знаходить його id та шукає збіг, якщо збіг був знайдений, бронювання видаляється та виводиться повідомлення "Booking deleted successfully".

*При помилках, існують catch метод який відловлює усі помилки, наприклад "Error deleting booking:", "Server error", "Error fetching bookings:".

Тепер ми маємо готовий профіль користувача (Рис. 3.17)

Рисунок 3.17 Скріншот вікна профілю користувача

Реалізація проведення турнірів:

За умови активного поточного облікового запису, буде наданий доступ до частини сайту з турнірною таблицею, де будуть доступні три поля : Перше

поле назв команд; Друге поле назви турніру; Трете поле вводу унікального номера турніру (Для можливості приєднання окремих користувачів), також знизу є дві кнопки, одна для створення турніру, друга для приєднання.

1) При натисканні кнопки створення турніру, йде валідація даних з поля назв команд, та розподіл їх за символом “,”, якщо поле було пустим, користувачу виводиться помилка "Будь ласка, заповніть всі поля".
2) Якщо дані підходять, створюється POST запит для створення турнірної сітки, дані додаються до колекції turnirs у базі даних та користувач отримує повідомлення "Турнір створено успішно! Унікальний код турніру: ${uniqueCode}" з унікальним кодом, а знизу додається слайд сторінки з турніром.

1) При натисканні кнопки приєднання, створюється POST запит для знайдення турніру, який перевіряє збіг даних за унікальними номерами турніру через функцію .findOne, якщо турнір не був знайдений користувач отримає повідомлення "Турнір не знайдено"
2) Якщо запит знайшов збіг даних, користувач отримає повідомлення "turnir found" та знизу буде додано слайд сторінки з турніром.
3) На сторінці турніру є кнопка збереження даних, яка при натисканні робить POST запит, який оновлює результати турніру за вибраними списками переможців.

3.4 Інструкція користувача

У цьому розділі я покажу як користуватися сайтом та базою даних

Для початку потрібно запустити Visual Studio Code та MongoDB Compass, у Visual Studio Code створюємо два термінали, перший термінал відповідає за backend, другий термінал за frontend (Назва папки computer_club), у проекті містяться дві папки відповідно. (Рис. 3.18)

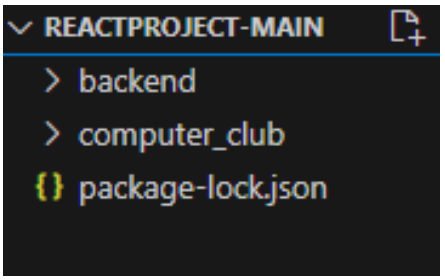

Рисунок 3.18 Скріншот структури проекту

У першому терміналі ми відкриваємо папку backend за допомогою команди:

```
cd backend
```

Та запускаємо app.js файл через команду:

```
node app.js
```

У другому терміналі ми відкриваємо папку computer_club за допомогою команди:

```
cd computer_club
```

Та запускаємо бібліотеку npm через команду:

```
npm start
```

Все, сайт запущено та відчинено, для підключення до бази даних потрібно зайти до MongoDB Compass та створити підключення до mongodb://localhost:27017. (Рис. 3.19)

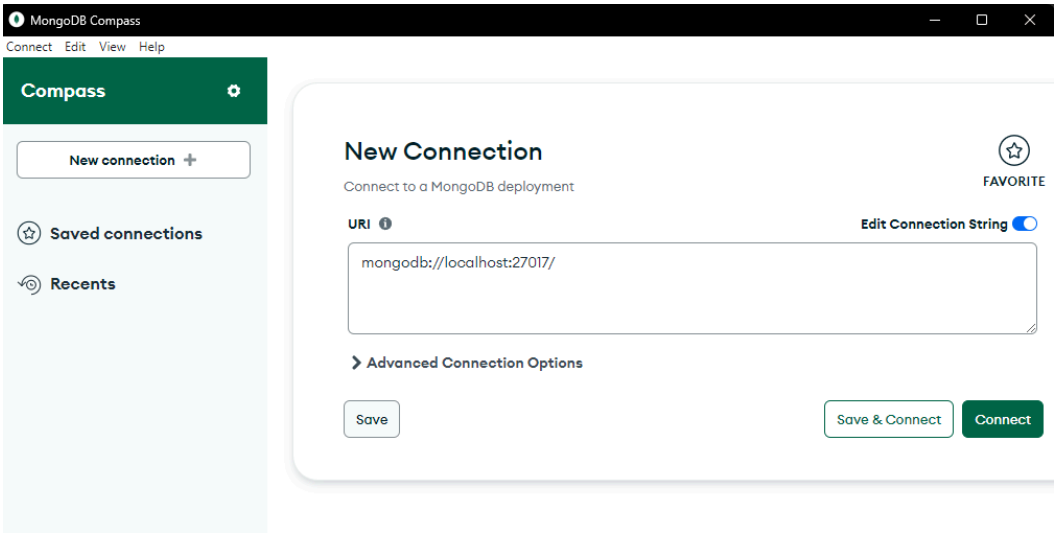

Рисунок 3.19 Скріншот підключення до бази даних

Після чого потрібно натиснути кнопку connect, тепер ми маємо доступ до бази даних. (Рис. 3.20)

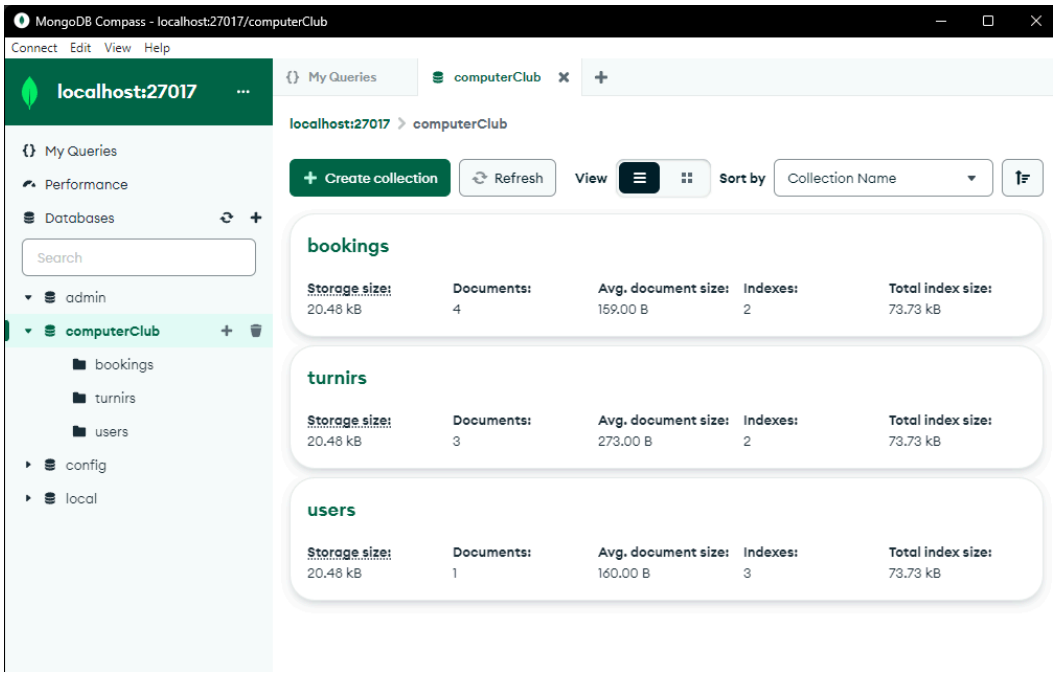

Рисунок 3.20 Скріншот підключеної бази даних

ВИСНОВКИ

Під час розробки програмного продукту було створено сайт комп’ютерного клубу з підключенням до бази даних, який має систему авторизації, бронювання та проведення турнірів.

У ході виконання роботи була використана мова JavaScript з використанням React та Node.js, та додатковою бібліотекою npm, також була використана MongoDB для створення бази даних.

Даний програмний продукт може бути використаний у комерційних цілях для зручної та автоматизованої організації у комп’ютерних клубах.

При розробці програмного продукту я покращив свої навички при роботі зі скриптами, створенням логіки сайту та баз даних. Було здобуто знання та навички при роботі з React та Node.js та засвоєно на практиці використання бази даних MongoDB, також були використані знання ООП.

Завдяки сайту значно підвищиться ефективність роботи комерційного закладу, як і збільшиться набіг людей які змогли знайти сайт через інтернет.

Для демонстраційної роботи сайт був запущений на локальній мережі, але може бути запущений на сервері та стабільно і якісно відпрацьовувати.

ПЕРЕЛІК ПОСИЛАНЬ

Figma. Онлайн інструмент для створення прототипів інструментів. URL:https://www.figma.com/

Український веб-довідник з мови HTML та CSS. URL:https://html-css.co.ua/

Сучасний онлайн підручник з вивчення JavaScript. URL:https://uk.javascript.info/

Веб-ресурс з безкоштовними іконками та фотографіями для веб-дизайну. URL:https://www.freepik.com/

Веб-ресурс з безкоштовними веб-шрифтами для сайту. URL:https://fonts.google.com/

Комп'ютерний клуб LEGION у Кривому Розі. URL:https://www.legion-club.com.ua/krivoj-rog.html

Комп'ютерний клуб CUBE у Кривому Розі. URL:https://cubevr.com.ua/ua/krivoy_rog.html

Технічна документація для реєстру npm, веб-сайту та інтерфейсу командного рядка. URL:https://docs.npmjs.com/

Завантаження та технічна документація для MongoDB. URL:https://www.mongodb.com/

Завантаження та технічна документація для Node.js. URL:https://nodejs.org/en

Технічна документація бібліотеки React. URL:https://react.dev/

Фреймворк та технічна документація Expressjs. URL:https://expressjs.com/

Форум з використання кастомних хуків бібліотеки React. URL:https://www.freecodecamp.org/news/react-hooks-useeffect-usestate-and-usecontext/

Технічна документація з використання бібліотеки Axios. URL:https://axios-http.com/

Технічна документація з використання бібліотеки Swiper. URL:https://swiperjs.com/

Додатки

Додаток А

Лістинг коду

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const cors = require("cors");
const shortid = require("shortid");
const WebSocket = require("ws");

const app = express();
const port = 3001;

app.use(cors());
app.use(bodyParser.json());

// Підключення до бази даних MongoDB
mongoose.connect("mongodb://localhost:27017/computerClub", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const db = mongoose.connection;

// Перевірка підключення до бази даних
db.on("error", console.error.bind(console, "Connection
error:"));
db.once("open", () => {
  console.log("Connected to the database");
});

// Створення схеми користувача
const userSchema = new mongoose.Schema(
  {
```

```
    username: {
      type: String,
      required: true,
      unique: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      enum: ["user", "admin"],
      default: "user",
    },
  },
  { timestamps: true }
);

// Створення моделі користувача на основі схеми
const User = mongoose.model("User", userSchema);
// Створення схеми турніру
const turnirSchema = new mongoose.Schema({
  pairs: [
    {
      team1: String,
      team2: String,
      winner: String,
    },
  ],
```

```
  turnirName: String,
  uniqueCode: String,
  createdAt: { type: Date, default: Date.now, index: { expires: "7d" } },
});

const Turnir = mongoose.model("Turnir", turnirSchema);

// Створення схеми бронювання
const BookingSchema = new mongoose.Schema({
  zone: String,
  hours: Number,
  price: Number,
  userId: mongoose.Schema.Types.ObjectId,
  userEmail: String,
  date: Date,
  createdAt: { type: Date, default: Date.now, index: { expires: "3d" } },
});

const Booking = mongoose.model("Booking", BookingSchema);

// WebSocket сервер
const wss = new WebSocket.Server({ port: 8000 });

wss.on("connection", (ws) => {
  console.log("New client connected");
  ws.on("message", (message) => {
    console.log(`Received message => ${message}`);
  });
  ws.on("close", () => {
    console.log("Client disconnected");
  });
});
```

```
const notifyClients = (data) => {
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(JSON.stringify(data));
    }
  });
};

const onNewBooking = (booking) => {
  notifyClients({ type: "NEW_BOOKING", booking });
};

// Обробник POST-запиту для створення нового користувача
app.post("/register", async (req, res) => {
  const { username, email, password, role } = req.body;
  try {
    const existingUser = await User.findOne({ $or: [{
username }, { email }] });
    if (existingUser) {
      return res.status(400).send("User already exists");
    }

    const newUser = new User({ username, email, password,
role });
    await newUser.save();
    console.log("User registered successfully:", newUser);
    res.status(200).send("User registered successfully");
  } catch (err) {
    console.error(err);
    res.status(500).send("Failed to register user");
  }
});
```

```
// Обробник POST-запиту для входу користувача
app.post("/login", async (req, res) => {
  const { login, password } = req.body;

  try {
    const user = await User.findOne({
      $or: [{ username: login }, { email: login }],
    });

    if (!user) {
      return res.status(401).json({ message: "Неправильний логін або пошта" });
    }

    if (user.password === password) {
      res.status(200).json(user);
    } else {
      res.status(401).json({ message: "Неправильний пароль" });
    }
  } catch (error) {
    console.error("Помилка при авторизації:", error);
    res.status(500).json({ message: "Помилка сервера" });
  }
});

// Обробник POST-запиту для створення нового турніру
app.post("/createTurnir", async (req, res) => {
  const { pairs, turnirName } = req.body;
  try {
    const uniqueCode = shortid.generate();
    const newTurnir = new Turnir({ pairs, turnirName, uniqueCode });
    await newTurnir.save();
    res.status(200).json({ uniqueCode });
```

```
    } catch (error) {
      console.error("Помилка при створенні турніру:", error);
      res.status(500).json({ message: "Помилка сервера" });
    }
  });

  // Обробник POST-запиту для пошуку турніру за унікальним
кодом
  app.post("/findTurnir", async (req, res) => {
    const { uniqueCode } = req.body;

    try {
      const turnirData = await Turnir.findOne({ uniqueCode });

      if (turnirData) {
        const { pairs, turnirName, uniqueCode } = turnirData;
        res.status(200).json({ turnirData: { pairs,
turnirName, uniqueCode } });
        console.log("turnir found");
        console.log(turnirData);
      } else {
        res.status(404).json({ message: "Турнір не знайдено"
});
      }
    } catch (error) {
      console.error("Помилка при пошуку турніру:", error);
      res.status(500).json({ message: "Помилка сервера" });
    }
  });

  // Обробник POST-запиту для оновлення турніру
  app.post("/updateTurnir", async (req, res) => {
    const { turnir } = req.body;

    try {
```

```
        const updatedTurnir = await Turnir.findOneAndUpdate(
          { uniqueCode: turnir.uniqueCode },
          { pairs: turnir.pairs },
          { new: true }
        );

        if (updatedTurnir) {
          res.status(200).json({ message: "Турнір успішно
оновлено" });
        } else {
          res.status(404).json({ message: "Турнір не знайдено"
});
        }
      } catch (error) {
        console.error("Помилка при оновленні турніру:", error);
        res.status(500).json({ message: "Помилка сервера" });
      }
    });

    app.post("/bookings", async (req, res) => {
      const { zone, hours, price, userId, date } = req.body;

      try {
        const user = await User.findById(userId);
        if (!user) {
          return res.status(404).json({ message: "User not
found" });
        }
        const newBooking = new Booking({
          zone,
          hours,
          price,
          userId,
          userEmail: user.email,
          date,
```

```
      });
      await newBooking.save();
      onNewBooking(newBooking);
      res
        .status(200)
        .json({ message: "Booking created successfully",
booking: newBooking });
    } catch (error) {
      console.error("Error creating booking:", error);
      res.status(500).json({ message: "Server error" });
    }
  });

  // Обробник GET-запиту для отримання всіх бронювань
  app.get("/bookings", async (req, res) => {
    try {
      const bookings = await Booking.find();
      res.status(200).json(bookings);
    } catch (error) {
      console.error("Error fetching bookings:", error);
      res.status(500).json({ message: "Server error" });
    }
  });

  // Обробник GET-запиту для отримання бронювань користувача
  app.get("/bookings/user/:userId", async (req, res) => {
    const { userId } = req.params;

    try {
      const bookings = await Booking.find({ userId });
      res.status(200).json(bookings);
    } catch (error) {
      console.error("Error fetching bookings:", error);
      res.status(500).json({ message: "Server error" });
```

```
  }
});

// Обробник DELETE-запиту для видалення бронювання
app.delete("/bookings/:id", async (req, res) => {
  const { id } = req.params;
  console.log("Received delete request for booking ID:",
id);

  try {
    const deletedBooking = await
Booking.findByIdAndDelete(id);

    if (!deletedBooking) {
      return res.status(404).json({ message: "Booking not
found" });
    }

    notifyClients({ type: "DELETE_BOOKING", booking:
deletedBooking });
    res
      .status(200)
      .json({
        message: "Booking deleted successfully",
        booking: deletedBooking,
      });
  } catch (error) {
    console.error("Error deleting booking:", error);
    res.status(500).json({ message: "Server error" });
  }
});

// Обробник POST-запиту для зміни пароля
app.post("/change-password", async (req, res) => {
  const { userId, currentPassword, newPassword } = req.body;
```

```
    try {
      const user = await User.findById(userId);

      if (user && user.password === currentPassword) {
        user.password = newPassword;
        await user.save();
        res
          .status(200)
          .json({ success: true, message: "Пароль успішно змінено" });
      } else {
        res
          .status(400)
          .json({ success: false, message: "Неправильний поточний пароль" });
      }
    } catch (error) {
      console.error("Помилка при зміні пароля:", error);
      res.status(500).json({ success: false, message: "Помилка сервера" });
    }
  });

  // Обробник POST-запиту для перевірки унікальності користувацького імені
  app.post("/check-username", async (req, res) => {
    const { username } = req.body;

    try {
      const existingUser = await User.findOne({ username });
      if (existingUser) {
        return res.status(200).json({ exists: true });
      } else {
        return res.status(200).json({ exists: false });
```

```
      }
    } catch (err) {
      console.error(err);
      res.status(500).send("Failed to check username");
    }
  });

  // Обробник POST-запиту для перевірки унікальності емейлу
  app.post("/check-email", async (req, res) => {
    const { email } = req.body;

    try {
      const existingUser = await User.findOne({ email });
      if (existingUser) {
        return res.status(200).json({ exists: true });
      } else {
        return res.status(200).json({ exists: false });
      }
    } catch (err) {
      console.error(err);
      res.status(500).send("Failed to check email");
    }
  });

  // Додайте обробник помилок для всіх інших запитів
  app.use((req, res) => {
    res.status(404).send("Not Found");
  });

  app.listen(port, () => {
    console.log(`Server running at
http://localhost:${port}/`);
  });
```