МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система обліку спортивних подій»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Коробов А.Г.**

Студента групи ІН – 62 — **Полушкіна В.М.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-62 спеціальності "Інформатика" денної форми навчання Полушкіна Віталія Миколайовича.

Тема: " Інформаційна система обліку спортивних подій "

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) аналітичний огляд існуючих інформаційних систем; 2) архітектурна модель системи; 3) дизайн додатку;

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Коробов А.Г.

Завдання приняв до виконання _____ Полушкін В.М.

Записка: 41 стор., 18 рис., 2 додатки, 8 джерел.

Мета роботи — розробка комплексної інформаційної системи обліку спортивних подій з можливістю дослідження та аналізу а також придбання товарів. При розробці важливим фактором є використанням сучасних технологій таких як, бібліотека ReactJS, база данних Firebase , а також Node.js та Stipe для довгострокового існування та підтримки продутку. А також створення адаптивного дизайну для коректного відображення на різних пристроях.

Результати — розроблено програмне забезпечення інформаційної системи огладу спортивних подій. Створено базу даних, розроблено авторизацію користувчів, реалізовано придбання товарів користувачами, а також розроблено відповідний веб-інтерфейс. Приведені приклади роботи системи. Клієнтська частина розроблена з використанням мови JavaScript, біблотеки React.js а також технології HTML5,CSS3. Для придбання товарів використано Node.js та система платежів Stripe; база даних – Firebase.

JAVASCRIPT, REACT.JS, JSX, NODE.JS, FIREBASE, STRIPE,
HTML5, CSS3.

ЗМІСТ

ВСТУП

Робота присвячена створенню інформаційної системи обліку спортивних подій з можливістю придбання товарів . Створення інформаційної системи – це процес визначення типу архітектури, інтерфейсів, компонентів та інших характеристик системи або її частин.

Добре та вчасно спроектована система несе у собі багато переваг. За допомогою проектування можна оцінити вартість та час, що необхідні для розробки програмного продукту. Налагоджена комунікація між замовником і виконавцем на етапі проектування допоможе запобігти розбіжностям між поглядами клієнта та виконавця на систему. Розбіжності можуть призвести до переробок системи, зайвих витрат часу і грошей, довгого узгодження подробиць реалізації.

Правильно спроектована система є легко змінюваною та розширюваною. Це зменшує грошові та часові витрати подальшого використання та вдосконалення, спрощує процес підтримки програмного продукту.

У зв’язку початком відновлення більшості спортивних подій, у наш час проблема отримання актуальних даних постає актуальною, як ніколи раніше.

Таким чиному у цій сфері існують соціально значущі проблеми такі , як: відсутня актуальна інформація про проведені спортивні заходи, відсутня актуальна інформація про існуючі спортивні команди, а також неможливо замовити необхідну продукцію не виходячи з дому;

Всі вищенаведені проблеми сильно гальмують надання актуальнихданих, а також поширення продукції галузі спортивної індустрії.

Отже, метою даної дипломної роботи є розробка інформаційної системи, необхідної для дослідження та аналізу даних, а також розробка користувацьких інтерфейсів для роботи з базою даних з адаптивним відображенням на більшості пристроїв.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Так як існує безліч різних технологій розробки програм що можуть допомогти користувачам отримувати необхідну інформацію, необхідно провести огляд, щоб зрозуміти, що краще використовувати. Основні аспекти, які необхідно проаналізувати і вибрати, є:

- патерни проектування;
- мова програмування та відповідне середовище для розроблення (IDE);
- бібліотеки, та фреймворки, які можуть бути використані для прискорення створення програми.

Таким чином, постає задача розробки інформаційної системи та виконати наступні підзадачі:

1) Вибір серверу – додатків для подальшого розміщення готового проекту
2) Розроблення веб-дизайну додатку.
3) Здійснити збір та формування вхідних даних.

1.1 Огляд патернів проектування

Патерн проектування – це загальне рішення ревної проблеми в дизайні архітектури. Патерн являє собою не конкретний код, а загальну концепцію або приклад рішення яке має бути налаштоване під потреби вашої програми.

Використання перевірених патернів при проектуванні архітектури інформаційної системи полегшить подальший розвиток системи. Завдяки можливості повторного використання коду, яку дають нам патерни проектування, майбутнє розширення системи або внесення змін до неї будуть відбуватися швидко та з мінімальними витратами.

Оглянемо патерни проектування, що можуть бути використані для створення нашої інформаційної системи.

У розробці веб-додатків існує велика кількість патернів. Їх можна згрупувати на породжувальні, структурні та поведінкові. Наприклад MVC, декоратор, одинак та Flux, які на даний момент є одними з найпопулярніших. доцільно порівняти.

Декоратор — відноситься до структурних патернів проектування, що динамічно допомогає додавати новий функціонал новостворенним об'єктам, загортаючи їх у корисні обгортки.

Іншою назвою декоратору є обгортка.Саме так можно найцвдаліше описати всю суть даного патерна: цільовий об'єкт розміщується у іншому об'єкті, який знаходиться вище за ієрархією, наче у новому об'єкті-обгортці , котрий реалізую головну поведінку, таку як отримання даних з серверу або з бази змінюючи виконання роботи і повернтає новий результат.

Ці два об'єкти мають суміжний інтерфейс керування, для нас, як користувача немає жодної різниці з чим працювати але при цьому зберігається архітектурний підхід, що допомагає змінювати або поліпшувати функціонал новими можливостями без без громіздкого рефакторингу коду. Це дає можливість об'єднати поведінку цих обгорток, використвоуючи декілька різних обгорток-декораторів одночасно.

У нашому житті ми часто використовуємо цей патерн не здогадуючись про це. Прикладом декоратора може бути будь-який одяг. Використовуючи Декоратор, початковий клас залишається без змін, при цьому не треба створювати безліч дочірніх класів. Змінюючи одяг ми не перестаємо бути собою, при цьому отримуємо нову здатність, здатність захисту від холоду.

У нашому прикладі програма декорує клас даних обгорткою, при цьому вона шифрую її та стискає, а при зчитуванні видає оригінальні дані, і навпаки, видає зашифровані та стислі при записі.

Як і у будь якого патрна декоратор має свої переваги та недоліки.

До переваг можна віднести:

– Набагато гнучкіший у порвінянні зі спадкуванням.

– Дозволяє створювати декілька обгорток , замість того щоб створювати один великий об’єкт для реалізації декількох функцій, що значно полегшує знаходження і вирішення помилок у випадку збою.

Проте декоратор має недоліки які не дозволяють використати його у розробці даної інформаційної системи. До них можна віднести:

– Важка конфігурація обгорток , які в свою чергу загорнуто в декілька обгорток одночасно.
– Зі збільшенням функціоналу, збільшується кількість цих самих декораторів.

Напротивагу декоратору є патерн одинак. Одинак — відноситься до групи породжувальних патернів проектування, він надає гарантії, що клас має лише один екземпляр, а такоє гарантує єдину глобальну точку доступу до нього.

Одинак одночасно вирішує декілька проблем :

– По-перше, він гарантує наявність одного екземпляра класу. Це буває корисно при доступі у єдиний спільний ресурс, сервер, клієнтська частина або база даних.
– По-друге, надає глобальну точку доступу. При цьому Це не тільки глобальна змінна, за допомоги якої можна отримати доступ до певного об’єкта.
–

Проте одинак має ряд недоліків:

– Через те що змінні ,що знаходяться у глобальному дорступі, вони не мають захисту від редагування чи видалення, тому протягом життєвого циклу їх значення можуть змінитися без відома користувача.
– Головним недоліком виступає порушення принципа єдиного обов’язку класу.

– Приховує поганий дизайн.
– Створює нову проблему багатопоточності.
– При проведенні тестів постійно вимагає створення об’єктів, що реалізують задані аспекти модельованого програмного оточення.

Володіючи спільним інтерфейсом декоратори, а також і сам клас даних, мають суміжний інтерфейс. Розробнику не важливо, працює він із загорнутим об’єктом чи зі звичайним об’єктом даних.

Залишаються ще два патрени: MVC та Flux.

MVC - це патерн проектування для клієнтських та серверних додатків, а Flux - це новий вид проектування додатків від Facebook, який обіцяє те саме, що і MVC, але з іншим підходом, який зосереджується на однонаправленому потоці даних. Яке застосування краще? Зробимо порівняння: Flux проти MVC.

Головний концепт кожного з них зводиться до одного:

Модель - підтримує поведінку і дані домену додатки.

Відображення - відображає модель в інтерфейсі.

Контроллер - приймає введення користувача, управляє моделлю і оновлює відображення.

До плюсів MVC можна віднести:

- Чудове розмежування коду, проста реалізація в JavaScript фреймворках, таких як Ember.js, а також до плюсів можна віднести відносно легка підтримка.
- Відокремлення презентації та моделі, що поліпшує тестування.
- Відокремлення відображення та контролера.

Хоча MVC і має низку переваг на серверній стороні і окремих фреймворках, але на стороні клієнта більшість фреймворків JS надають підтримку прив'язки даних, що дозволяє представленню безпосередньо синхронизуватися з моделлю. Розробники широко масштабованих проектів

зіткнулися з проблемою масштабування своїх додатків використовуючи патерн MVC, і внаслідок цього світ отримав новий потік архітектури. На конференції розробників FB вони показали, з якими трудностями вони зіткнулися(Рис1.1).

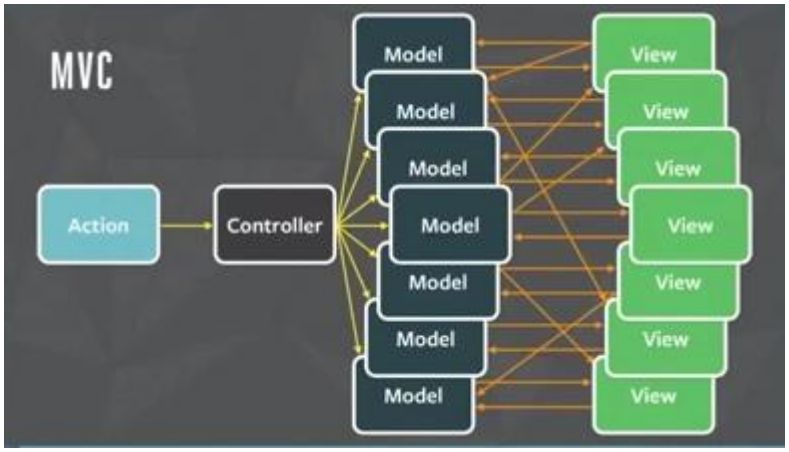


Рис1.1

Під час розробки інформаційної системи можна зіткнутися з такою проблемою, view1 маніпулює model1, а model1 оновлює погляд2, так як їх система має складну кругову залежність, саме тому розробники винайшли нове рішення - Flux.

Flux - це архітектура додатків, яку розробники використовують для створення веб-додатків на стороні клієнта. Він доповнює складові компоненти React для перегляду, використовуючи однонаправлений потік даних. Це скоріше шаблон, а не формальний фреймворк, і ви можете почати використовувати Flux негайно без додавання великої кількості нового коду.

Враховуючи, що розроблювана інформаційна система буде широкомасштабованою, а також матиме можливість модифікацій у майбутньому було обрано Flux.

1.2 Вибір бібліотеки

У сучасному світі починати будувати програму з нуля означає марно витратити купу грошей та часу. Саме тому кожного дня створюються нові бібліотеки та фреймворки. Фреймворки схожі на ярлики. Вони значно

прискорюють розробку та допомогають дотримуватися сучасних патернів розробки та дизайну програми.

Перш ніж перейти до фреймворку, треба знати, який фреймворк що пропонує. Справді, іноді можна легко зробити в 5 рядках коду, те що використовує декілька функцій фреймворку, і в цьому випадку ви можете легко написати власний код.

Використання фреймворку добре, коли є строгий термін, коли ви не повинні уникати багато кодування, коли ви хочете, щоб хтось інший дбав про ораганізацію вашого коду.

Заданними github в 2020 році найпопулярнішими бібліотеками веб розробці є:

1. React.js
2. Angular.js

Angular.js

Це повнофункціональний фреймворк, який використовують величезні корпорації. Розробники люблять за якісну документацію, а також за те, що не потрібно вивчати додаткові бібліотеки.

Переваги:

- повний набір інструментів;
- високоякісне генерування коду;
- елегантний стиль програмування.

Недоліки:

- використовує багато ресурсів;
- складна для вивчення;
- погана оптимізація вирішення проблем із низькою продуктивністю;

React.js

React - це бібліотека Javascript, розроблена та підтримувана Facebook. За останніми даними, цей продукт вважається найпопулярнішим. Так, у 2018 році він перевершив Angular майже за всіма показниками, як числовими, так і оціненими користувачами.

Переваги:

- гнучка бібліотека;
- невеликі розміри файлів;
- прості оновлення, які не порушують стабільність;
- прекрасно поєднується з іншими бібліотеками, якщо це необхідно.

Недоліки:

- використання додаткових інструментів, модулів та пакетів;
- крива навчання багато в чому залежить від того, яке фонове рішення обирає користувач;
- не оптимізована документація.

На даний момент React - це найбільш запитуваний набір рішень. Важливо, що скарг на бік цієї бібліотеки значно менше порівняно з Angular. Бібліотеку використовують такі корпорації, як Airbnb та Twitter, тому що Facebook підтримує свою функціональність та стабільність на високому рівні.

Саме тому було обрано цю бібліотеку для створення програмного продукту.

1.3 База даних

У створенні системи важливу роль відіграють дані. Простими словами, дані можуть бути фактами, пов'язаними з будь-яким предметом, що розглядається.Наприклад, ваше ім’я, вік, зріст, вага тощо - це деякі

дані, що стосуються вас.Зображення, зображення, файл, PDF тощо також можна вважати даними.

База даних - це систематичний збір даних. Бази даних підтримують зберігання та маніпулювання даними. Бази даних спрощують управління даними. Давайте обговоримо кілька прикладів.

Розглянемо для прикладу систему facebook. Вона повинна зберігати, маніпулювати та представляти дані, пов’язані з членами, їх друзями, активністю членів, повідомленнями, рекламою та багато іншого.

Можна навести велику кількість додатків, що не можуть існувати без бази даних.

Система управління базами даних (СУБД) - це сукупність програм, яка дає можливість її користувачам отримувати доступ до бази даних, маніпулювати даними, звітувати / представляти дані. Це також допомагає контролювати доступ до бази даних.

Системи управління базами даних не є новою концепцією, і як така вперше була впроваджена в 1960-х роках.

Кажуть, що інтегрований сховище Чарльза Бахмена є першою СУБД в історії.

З часом технології баз даних значно розвинулися, в той час як використання та очікувані функціональні можливості баз даних значно зросли.

Існує 4 основних типи СУБД. Розглянемо їх детально.

Ієрархічна - цей тип СУБД використовує відносини «батько-дитина» для зберігання даних. Цей тип СУБД сьогодні рідко використовується. Його структура схожа на дерево з вузлами, що представляють записи та гілки, що представляють поля. Реєстр Windows, що використовується в Windows XP, є прикладом ієрархічної бази даних.

Параметри конфігурації зберігаються у вигляді деревних структур з вузлами.

Мережеві СУБД - цей тип СУБД підтримує багатозахисні відносини. Зазвичай це призводить до складних структур бази даних. RDM Server - приклад системи управління базами даних, яка реалізує мережеву модель.

Реляційні СУБД - цей тип СУБД визначає відносини бази даних у вигляді таблиць, також відомих як відносини. На відміну від мережевих СУБД, RDBMS не підтримує багато стосунків. Реляційні СУБД зазвичай мають попередньо визначені типи даних, які вони можуть підтримувати. Це найпопулярніший тип СУБД на ринку. Приклади систем управління реляційними базами даних включають базу даних MySQL, Oracle та Microsoft SQL Server.

Об'єктно-орієнтована СУБД - цей тип підтримує зберігання нових типів даних. Дані, що підлягають збереженню, мають форму об'єктів. Об'єкти, що зберігаються в базі даних, мають атрибути (тобто стать, ager) та методи, що визначають, що робити з даними. PostgreSQL - приклад об'єктно-орієнтованої реляційної СУБД.

NoSQL - це майбутня категорія систем управління базами даних. Головною його характеристикою є невідповідність поняттям реляційних баз даних. NOSQL означає "Не тільки SQL".

Концепція баз даних NoSQL зросла з гігантськими інтернет-сайтами, такими як Google, Facebook, Amazon тощо, які займаються гігантськими обсягами даних.

Коли ви використовуєте реляційну базу даних для великих обсягів даних, система починає повільно ставитись до часу відгуку.

Щоб подолати це, ми могли б, звичайно, "розширити" наші системи, оновивши наявне обладнання. Альтернативою вищевказаній проблемі

було б розподілити навантаження на нашу базу даних по декількох хостах у міру збільшення навантаження.Це відомо як "масштабування".

База даних NOSQL - це нереляційні бази даних, які масштабуються краще, ніж реляційні бази даних, і розроблені з урахуванням веб-додатків.

Вони не використовують SQL для запиту даних і не дотримуються строгих схем, таких як реляційні моделі. З функціями NoSQL, ACID (Atomicity, Consistentcy, Isolation, Durability) не завжди гарантується.

Враховуючи усі вищеперерахвоаня перваги доцільним буде порівняти NoSQL бази данних.

Найпопулярнішими базами даних у 2020 році є MongoDB та Firebase.

MongoDB розробляється та управляється MongoDB Inc. Це база даних NoSQL з відкритим кодом. На відміну від Firebase, яка пропонує повну екосистему послуг, MongoDB - це лише (дуже потужна) база даних документів.

Масштабованість та гнучкість - два фактори, що розглядаються при розробці MongoDB. Він пропонує дуже потужний запит та індексацію. Хоча він пропонує лише цілеспрямовану послугу зі зберігання даних, MongoDB все ще широко застосовується завдяки потужним можливостям зберігання даних, які він пропонує. Завдяки MongoDB розробники отримують більше сил у розробці додатків. MongoDB відповідає їх потребам у розробці, тому додаток ефективно зберігає дані.

Firebase - це набагато більше, ніж просто база даних. Це повне рішення, яке використовується для створення веб-та мобільних додатків. Google володіє цією послугою Backend as-a-Service в режимі реального часу. База даних у реальному часі Google Firebase ідеально підходить для додатків, які потребують обробки даних у режимі реального часу на кількох пристроях.

Служба баз даних Firebase називається Cloud Firestore. Вона працює в режимі реального часу, отримуючи зміни з вашої бази даних по мірі того, як вони відбуваються. Firestore є частиною послуг Cloud Firebase, а це означає, що вона прекрасно працює з усіма іншими продуктами Firebase такими як хостинг та сховище.

Переваги Firebase:

- Надійна бібліотека клієнтів
- Повна підтримка автономного режиму
- Комплексний набір правил безпеки
- Простий у використанні інструмент перегляду даних.

Беручи до уваги всі вище перераховані переваги та недоліки для створення продукту було обрано Firebase у якості бази даних а також розміщення на хостингу Firebase Hosting.

1.4 Постановка задачі

Для досягнення мети роботи, були визначені такі завдання: розробка адаптивного інтерфейсу сайту використовуючи бібліотеку React.js, розробка бази даних з використанням Firebase, наповнення даними з бази а також розміщення на хостингу.

Розроблений сайт повинен виконувати такі функції:

- отримання інформації про останні події;
- замовлення товарів;
- авторирзація користувачів;
- управління з використанням панелі адміністратора.
- адаптивне відображення на різних платформах.

А також створити інструкцію у додатку, щоб чітко розуміти , яким чином і з якою метою використовувати той чи інший функціонал системи.

2 АРХІТЕКТУРНА МОДЕЛЬ СИСТЕМИ

Для того щоб розроблювана система могла добре функціонувати довгий час, вона має забезпечувати поставленим вимогам, мати високу якість та бути доступною на легкою у використанні для звичайних користувачів. На всі ці параметри впливає архітектура програмного забезпечення.

Архітектура – це базова організація системи, втілена в її компонентах, зовнішніми та внутрішніми відношеннями у системі, а також принципи, що визначають проектування та розвиток системи.

Правильно та своєчасно спроектована архітектура полегшить подальшу розробку, впровадження, користування, модифікацію продукту.

З часом розвиваються технології та змінюються потреби користувачів, це змушую розробників змінювати архітектурні моделі. Сьогодні веб-розробник повинен використовувати інструменти та підхід до розробки, що
узгоджуються із сучасною веб-сценою. Наразі все більше розробників використовують клієнт-серверний підхід тому, що він дозволяє робити архітектуру додатку більш гнучкою. Завдяки цьому при необхідності змін – редагування або додавання, можуть бути зроблені набагато швидше.

Модифікація – це критерій легкості, з якою можна вносити зміни в архітектуру додатка. Навіть якщо можна було б створити програмну систему, що буде ідеально відповідати вимогам її користувачів, ці вимоги можуть змінюватися з часом подібно до того, як з часом змінюється суспільство. Оскільки компоненти додатку розподілені на декілька організаційних частин, система буде готова до часткових та поступових змін, коли існують декілька реалізацій, які не заважають одна одній, але дозволяють використовувати розширені можливості.

З подальшим розвитком буде набагато простіше вносити зміни у додаток, що створений з використання клієнт-серверної архітектури. Це дасть можливість клієнтській стороні обмінюватися повідомлення з сервером не вникаючи у подробиці його реалізації, а також зробить можливим внесення

змін тільки для однієї сторони, не змінюючи сам механізм обміну повідомленнями.

2.1 Flux

Так як, було обрано Flux доцільно розглянути дану модель детальніше, так як вона зможе забезпечити швидку комунікацію між користувачами.

У стандартній архітектурі Flux існують наступні компоненти:

Actions - помічники, які передають дані в Dispatcher

Dispatcher - отримує ці дії і передає необхідні дані зареєстрованим callback-ом.

Stores - діють як контейнери для стану програми та логіки. Реальна робота додатка відбувається в Stores. Stores, зареєстровані для прослуховування дій Dispatcher, будуть відповідно і оновлювати View.

Controller Views - React компоненти захоплюють стан з Stores, а потім передають дочірнім компонентів.

Контролери в MVC і Flux розрізняються. Тут контролери є Controller-View і знаходяться на самій вершині ієрархії. View - це React компоненти.

Весь функціонал, як правило, знаходиться в Store. В Store виконується вся робота і повідомляється Dispatcher, які події або дії він прослуховує.

Коли відбувається подія, Dispatcher відправляє "корисне навантаження" в Store, який зареєстрований для прослуховування конкретно цієї події. Тепер в Store необхідно оновити View, що в свою чергу викликає дію. Дія яку треба виконати, точно також як і ім'я, тип події та багато іншого відомі заздалегідь.

View поширює Action через центральний Dispatcher, і це буде відправлено в різні Stores. Ці Stores будуть відображати бізнес-логіку програми та інші дані. Store оновлює все View.

Найкраще працює спільно зі стилем програмування React, так як Store поновлює дані без необхідності детального опису, як змінювати відображення між станами, підсумовуючи усе вище сказане було обрано патерн Flux.

2.2 Бібліотека React.js

React - найпопулярніша фронтальна бібліотека JavaScript у галузі веб-розробки. Її використовують як великі, зареєстровані компанії, так і новоспечені стартапи (Netflix, Airbnb, Instagram та New York Times). React приносить багато переваг, що робить його кращим вибором, ніж інші фреймворки типу Angular.js.

Головною відмінністю React є «Virtual DOM» - інтерфейс програмування для документів HTML та XML (розширювана мова розмітки). Він визначає логічну структуру документів та спосіб доступу до документа та маніпулювання ним. Коли веб-сторінка завантажується, браузер створює DOM сторінки. Він представляє документи у вигляді вузлів або об'єктів.

Створення та оновлення DOM - це більш повільний процес. Щоб прискорити процес, facebook розробив ReactJS. ReactJS використовує концепцію віртуального DOM, що робить завантаження швидшим. Віртуальний DOM - це просто легкий об’єкт JavaScript, який є лише копією реального DOM. Віртуальний DOM працює в наступних 3 кроках:

- Кожного разу, коли відбувається будь-яка зміна опори або стану, весь інтерфейс відображається у віртуальному DOM.
- різницю між попереднім DOM (реальним DOM) і новим DOM (віртуальним DOM) оцінюється за допомогою алгоритму порівняння.
- Після того, як будуть зроблені остаточні розрахунки, реальний DOM буде оновлений лише тими елементами, які фактично змінилися.

Як працює алгоритм порівняння?

Порівннюючи будь-які 2 дерева, спочатку порівнюється корінь обох дерев. В залежності від кореня дерева алгоритм працює по-різному.

Кожного разу, коли кореневий елемент має інший тип , все дерево знищується, а нове дерево будується з нуля, тобто будується абсолютно новий DOM. Під час знищення старого дерева виконується функція

ComponentWillUnmount () , котра виконує всю необхідну заміну в рамках методу, наприклад, недійсні таймери, скасування мережевих запитів або очищення будь-яких підписок, створених під час ComponentDidMount (). Знову, будуючи нове дерево, ComponentWillMount ()функція виконується спочатку (це єдиний метод життєвого циклу, який викликається на стороні сервера під час використання візуалізації на стороні сервера), а потім ComponentDidMount () (який виконується один раз у життєвому циклі компонента та після першого відображення).

Коли кореневі елементи однотипні , тоді react порівнює атрибути елемента. Якщо він стикається зі зміною, він оновлює лише цю частину. Під час повторного проходження через батьківський вузол React одночасно проходить усі елементи списку та генерує нове відображення, коли виникає зміна.

Завдяки архітектурі Flux React використовує односторонню прив'язку даних. Односпрямований потік даних означає, що коли розробник розробляє додаток на React, то часто вкладають дочірні компоненти в батьківські компоненти. Таким чином, розробник знає, де і коли виникає помилка, надаючи їм кращий контроль над усім веб-додатком.

Найважливі концепції React:

- Компоненти (Components);
- Стан (State);
- Властивості (Props);

Компоненти це будівельні блоки будь-якої програми React, і один додаток зазвичай складається з декількох компонентів. Компоненти по суті є частиною інтерфейсу користувача. React розбиває інтерфейс користувача на незалежні багаторазові частини, які можна обробити окремо. Компонентами можуть бути кнопки, профілі користувачів, товари а також пости.

Компоненти поділяються на функціональні та классові

Функціональні компоненти не мають власного стану і містять лише метод візуалізації, тому їх також називають компонентами без стану . Вони можуть отримувати дані з інших компонентів як властивості (props).

Компоненти класу можуть утримувати та керувати своїм станом та мати окремий метод візуалізації для повернення JSX на екран. Їх ще називають «statefull», оскільки вони можуть мати стан

Стан - це вбудований об'єкт React, який використовується для зберігання даних або інформації про компонент. Стан компонента може змінюватися з часом; щоразу, коли він змінюється, компонент повторно відображається. Зміна стану може статися як відповідь на дії користувача або події, що генеруються системою, і ці зміни визначають поведінку компонента та спосіб його відображення.

Властивості - це вбудований об'єкт React, який зберігає значення атрибутів тегу і працює аналогічно атрибутам HTML. Він забезпечує спосіб передачі даних від одного компонента до інших компонентів так само, як передаються аргументи у функції.

2.2.1 Material UI

Бібліотека React дозволяє використовувати сторонню бібліотеку MaterialUI. Макети Material Design залишаються однаковими незалежно від використовуваних платформ, середовищ виконання і дозволів екрану завдяки використанню однакової системи елементів і інтервалів.

Адаптивні макети в Material Design підлаштовуються під всі можливі розміри екрану. Для забезпечення адаптивності користувальницького інтерфейсу ми надаємо наступні засоби:

- Grid: це адаптивна сітка макетів Material Design, що може підлаштоуватися під будь-який розмір а також орієнтацію екрану, забезпечуючи узгодженість макетів.

– Container: Контейнер центрує ваш контент по горизонталі. Це базовий елемент всіх макетів.
– Breakpoints: API, що дозволяє використовувати контрольні точки в широкому діапазоні контекстів.
– useMediaQuery: Це хук медіа-запиту для React. Він відстежує збіги з медіа-запитом CSS.
– Hidden: Швидко і оперативно управляє видимістю компонентів.

У ході огляду технології а також беручи до уваги, що бібліотека React гармонічно поєднується з Material UI, її було обрано для створення адаптивного дизайну.

2.3 База данних Firebase

Firebase - це платформа для розробки програмного забезпечення, запущена в 2011 році, і придбана Google в 2014 році. Розпочата як база даних у режимі реального часу, зараз вона має 18 сервісів (4 з них зараз у бета-версії) та спеціальні API. Вся платформа - це рішення Backend as-a-Service як для мобільних, так і веб-додатків, що включає послуги зі створення, тестування та управління додатками.

Рішення BaaS дозволяють усунути необхідність в управлінні базами даних для резервного копіювання та отриманні відповідного обладнання. Натомість ви можете підключити їх до свого додатку через спеціальні API для кожної окремої послуги. У випадку з Firebase є 7 з них, які охоплюють увесь спектр бек-енд-технологій для програми

Firebase Realtime Database була першим продуктом, який з’явився під прапором Firebase, тому це найвідоміший і стабільний сервіс на всій платформі. База даних в реальному часі - це по сутіхмарне сховище NoSQL, яке можна підключити до програми, щоб забезпечити доступ у реальному часі до даних на

різних платформах. Однією з переваг є те, що база даних може працювати в автономному режимі, кешуючи дані в пам'яті пристрою та після підключення до Інтернету, синхронізуючи їх.

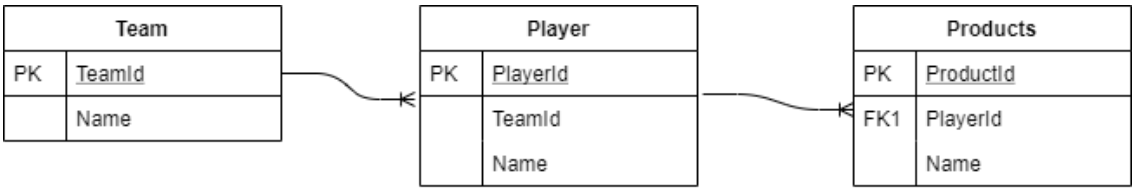
Дані зберігаються в JSON і користувачі можуть їх запитувати. З точки зору безпеки, база даних в реальному часі забезпечує доступ до даних на основі дозволів. Це можна зробити за допомогою аутентифікації Firebase та надання дозволів за особами користувача або правилами безпеки.

Аутентифікація Firebase - це функція аутентифікації Google, розроблена для програм, що використовують Firebase. Це дозволяє використовувати попередньо вбудований або створити користувальницький інтерфейс користувача для аутентифікації користувача та ввійти в систему користувачів через користувацькі облікові дані, електронні листи чи соціальні медіа.

Аутентифікація Firebase дозволяє авторизовувати користувачів без надлишкового навантаження на серверній частині.

Якщо ви будуєте веб-додаток, прогресивний веб-додаток або мобільну цільову сторінку, вам точно знадобиться хостинг. Firebase пропонує статичний веб-хостинг для додатків, створених за допомогою HTML, CSS та JavaScript . З точки зору безпеки, він використовує стандартні протоколи HTTPS і SSL для доставки файлів та інших типів даних.

Для відображення ролі бази даних в одній з найважливіших функцій розробленого сайту, було сформовано ER-діаграму фрагменту усієї бази (рис. 2.1), вона графічно відображає сутності, їх атрибути та взаємозв’язки між сутностями.



2.4 Структурно-функціональна модель

У розробці будь-якої програми важливим є створення різних допоміжних діаграм , які спрощують весь процес розробки. Для створення дизайну додатку використаємо загально прийняту мову моделювання UML.

Уніфікована мова моделювання (UML) – це сімейство графічних нотацій, в основі яких лежить єдина метамодель. Вона допомагає в описанні та проектуванні інформаційних систем, особливо систем побудованих з використанням об’єктно-орієнтованих властивостей.

Для зображення можливих дій клієнт або сервера в системі використаємо use case діаграму. Суть даної діаграми полягає в наступному: проектована система подається у вигляді множини сутностей або акторів, що взаємодіють з системою за допомогою за допомогою так званих варіантів використання(use case). При цьому актором називають будь-яку зовнішню сутність, що взаємодіє з системою. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на створювану систему так, як визначить сам розробник. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який чинять системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів з системою.

Даний вид діаграм використовується для зображення відношення між акторами та прецедентами(варіантами дій) в системі.

На Рисунку №2.1 зображено use case діаграму інформаційної системи.

Акторами системи являються:

- server – сервер інформаційної системи;
- web client – не авторизований користувач;
- client – авторизований користувач;

Сервер прослуховує запити клієнтів, контролює процес обміном даними, надаючи клієнтам необхідну інформацію. Клієнт, в залежності від свого статусу, може посилати відповідні запити на сервер та отримувати від нього відповіді.

Клієнти можуть виконувати такі дії, як :

- авторизуватись;
- переглянути інформацію стосовно подій ;
- створити нову подію;
- редагувати існуючі події;
- створити платіж обраного продукту;

Сервер може виконувати такі дії, як :

- авторизувати користувача;
- показати існуючі події;
- сотворити нову подію;
- редагувати подію;
- обробити платіж

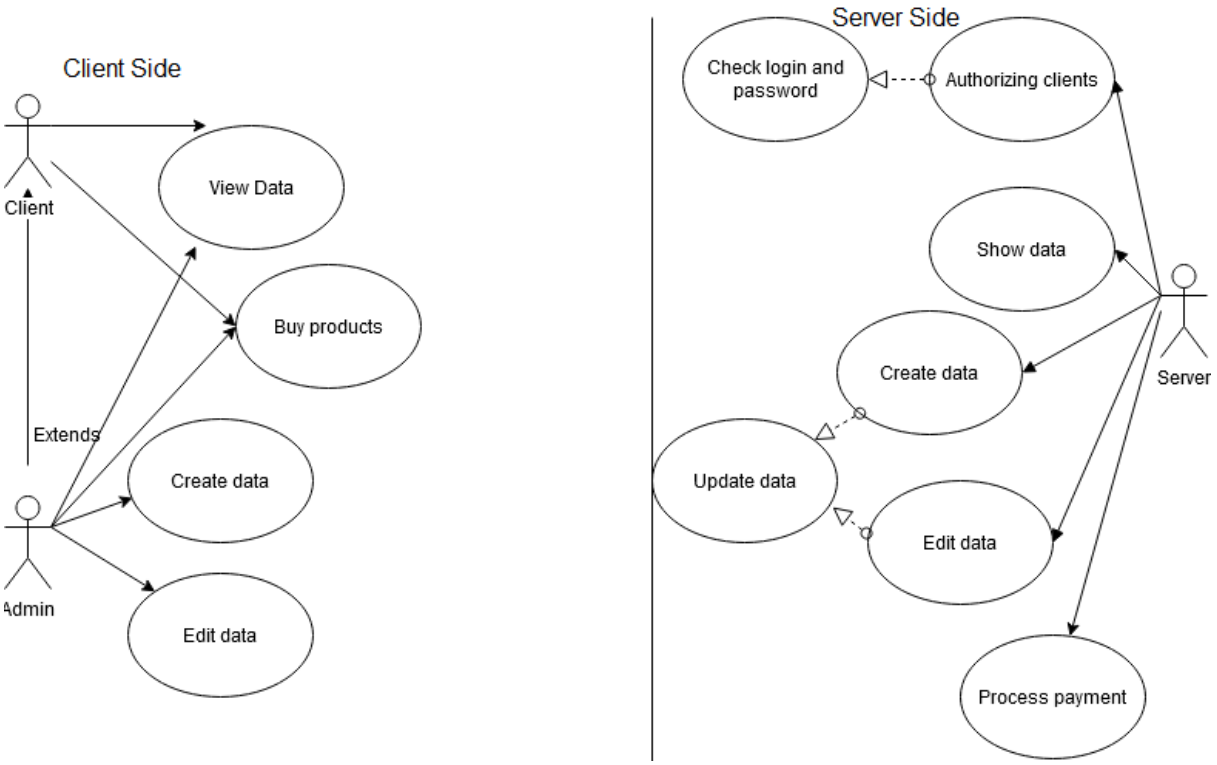


Рисунок 3.1 «Use Case Diagram»

3 РОЗРОБКА WEB ДОДАТКУ

3.1 Розробка компонентів веб додатку

Під час проведення аналізу веб додатка було вирішено використати бібліотеку React.js клієнтській та Node.js на серверній частині. Використання однієї мови спрощую розробку та подальшу підтримку коду. В якості бази данних використано Firebase. Роботу можна розпочати на локальному сервері а в подальшому розмістити на хостингу Firebase.

Кореневий каталог клієнтьскої частини (Рис.3.1) містить дві папки:

– Папка src, у якій знаходиться головний файл index.js (Рис.3.2) , з якого починається робота програми.

– Папка node_modules, у якій знаходяться усі необхідні бібліотеки та модулі.

А також файл package.json, котрій дозволяє запустити додаток на локальному сервері а також містить інформацію про залежності та додаткові бібліотеки, які використані в ході розробки програмного продукту

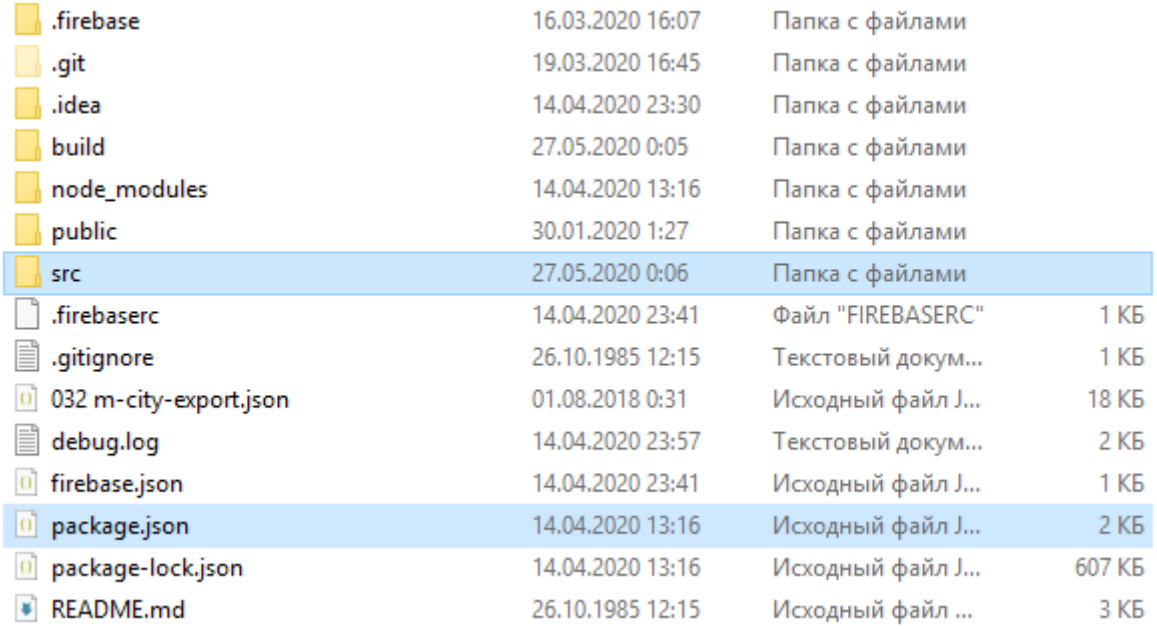
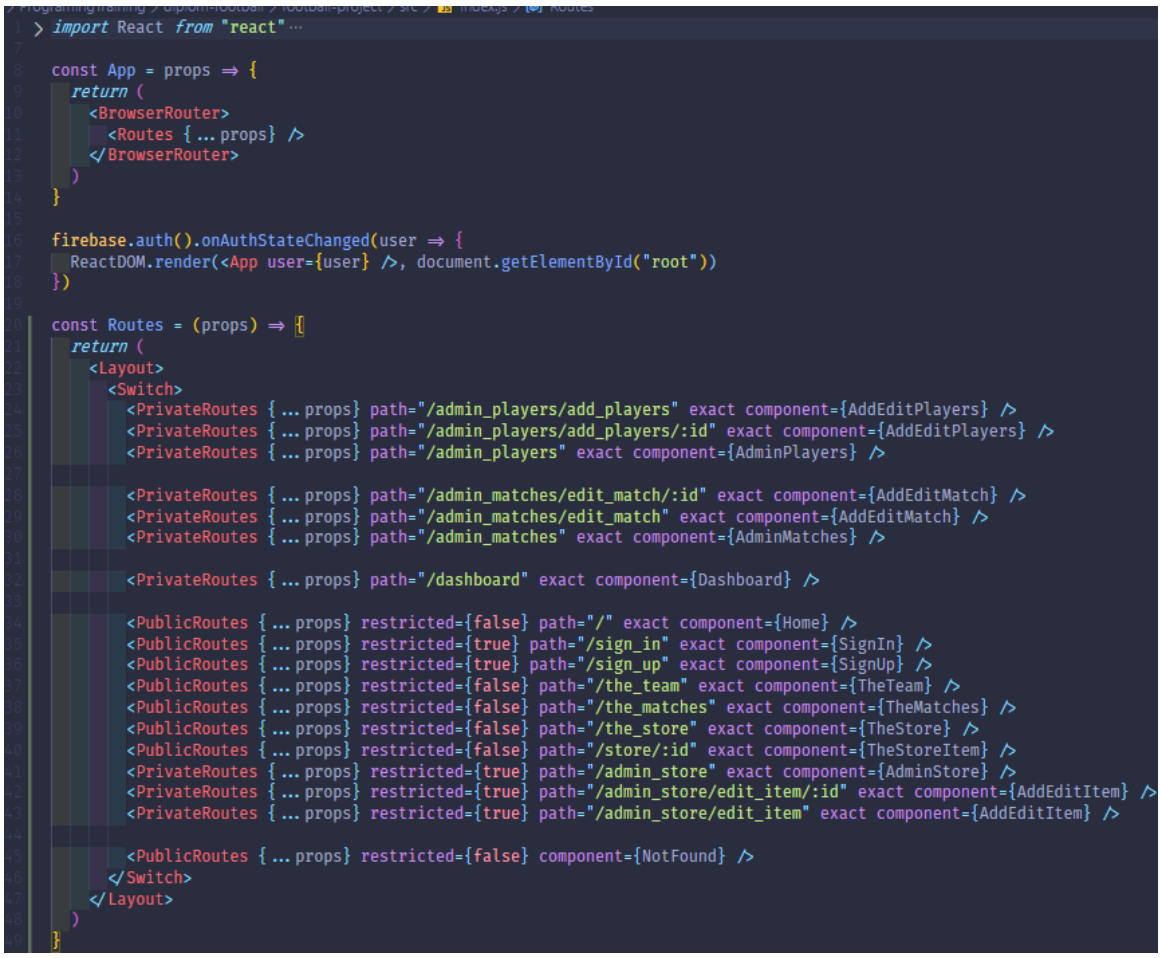


Рис 3.1 Кореневий каталог клієнтьскої частини



```
import React from "react"

const App = props => {
  return (
    <BrowserRouter>
      <Routes {...props} />
    </BrowserRouter>
  )
}

firebase.auth().onAuthStateChanged(user => {
  ReactDOM.render(<App user={user} />, document.getElementById("root"))
})

const Routes = (props) => {
  return (
    <Layout>
      <Switch>
        <PrivateRoutes {...props} path="/admin_players/add_players" exact component={AddEditPlayers} />
        <PrivateRoutes {...props} path="/admin_players/add_players/:id" exact component={AddEditPlayers} />
        <PrivateRoutes {...props} path="/admin_players" exact component={AdminPlayers} />

        <PrivateRoutes {...props} path="/admin_matches/edit_match/:id" exact component={AddEditMatch} />
        <PrivateRoutes {...props} path="/admin_matches/edit_match" exact component={AddEditMatch} />
        <PrivateRoutes {...props} path="/admin_matches" exact component={AdminMatches} />

        <PrivateRoutes {...props} path="/dashboard" exact component={Dashboard} />

        <PublicRoutes {...props} restricted={false} path="/" exact component={Home} />
        <PublicRoutes {...props} restricted={true} path="/sign_in" exact component={SignIn} />
        <PublicRoutes {...props} restricted={true} path="/sign_up" exact component={SignUp} />
        <PublicRoutes {...props} restricted={false} path="/the_team" exact component={TheTeam} />
        <PublicRoutes {...props} restricted={false} path="/the_matches" exact component={TheMatches} />
        <PublicRoutes {...props} restricted={false} path="/the_store" exact component={TheStore} />
        <PublicRoutes {...props} restricted={false} path="/store/:id" exact component={TheStoreItem} />
        <PrivateRoutes {...props} restricted={true} path="/admin_store" exact component={AdminStore} />
        <PrivateRoutes {...props} restricted={true} path="/admin_store/edit_item/:id" exact component={AddEditItem} />
        <PrivateRoutes {...props} restricted={true} path="/admin_store/edit_item" exact component={AddEditItem} />

        <PublicRoutes {...props} restricted={false} component={NotFound} />
      </Switch>
    </Layout>
  )
}
```

(Рис.3.2) Головний файл index.js

Підключення до бази данних відбувається у файлі firebase.js (Рис. 3.3) у якому зберігається ключ доменне ім’я на якому знаходиться додаток а також ідентифікатор проекту у базі. Також саме тут відбувається посилання на відповідні колекції.

```
import firebase from "firebase/app"
import "firebase/app"
import "firebase/database"
import "firebase/auth"
import "firebase/storage"

const firebaseConfig = {
  apiKey: "AIzaSyD9-G4MmzbI37LjuJT1HabcmXoPp-l7t9c",
  authDomain: "m-city-caf04.firebaseapp.com",
  databaseURL: "https://m-city-caf04.firebaseio.com",
  projectId: "m-city-caf04",
  storageBucket: "m-city-caf04.appspot.com",
  messagingSenderId: "87972102603",
  appId: "1:87972102603:web:719c99b47e97cf7f40364e",
  measurementId: "G-DFYFFDTV2X"
}

firebase.initializeApp(firebaseConfig)
/* firebase.analytics(); */

const firebaseDB = firebase.database()

//requests
const firebaseMatches = firebaseDB.ref("matches")
const firebasePromotions = firebaseDB.ref("promotions")
const firebaseTeams = firebaseDB.ref("teams")
const firebasePlayers = firebaseDB.ref("players")
const firebaseStore = firebaseDB.ref("items")

//export
export {
  firebase,
  firebaseDB,
  firebaseMatches,
  firebasePromotions,
  firebaseTeams,
  firebasePlayers,
  firebaseStore
}
```

Рис. 3.3 Файл підключення бази даних firebase.js

Для подальшого управління даними використовується Firebase Console. На Рис.3.4 продемонстровано створену базу даних.

Рис 3.4 База даних у Firebase Console

Для поліпшення візуального оформлення було використано файл style.css

Кінцевий варіант файлу style.css має у собі 800 рядків коду , на Рис 3.5 відображено лише частину коду.

```
{
  font-size: 2.5em;
  padding: 6px 10px;
}
.home_matches_wrapper .home_matches {
  display: block;
  margin-top: 60px;
}
.home_matches .item .wrapper {
  padding: 30px;
  width: 90vw;
}
.featured_number {
  font-size: 20vw;
  color: #0e1731;
  transform: translate(460px, 570px);
  position: absolute;
  margin-left: -33vw;
  margin-top: 15vh;
  background: #ffffff52;
  border-radius: 10%;
}
.featured_first {
  position: absolute;
  border-radius: 10%;
  background: #ffffff52;
  color: #0e1731;
  font-size: 5.5rem;
  text-transform: uppercase;
  padding: 0px 20px;
  margin-left: -35vw;
}
.featured_second {
  position: absolute;
  border-radius: 10%;
  background: #ffffff52;
  color: #0e1731;
  font-size: 3.5rem;
  text-transform: uppercase;
  padding: 0px 20px;
  margin-left: -35vw;
}
```

Рис 3.5 Файл style.css

Для управління додатком створено панель адміністратора (Рис.3.6), доступ до якої має лише авторизований користувач.

Рис.3.6 Панель адміністратора

3.2 Результат розробки веб додатку

Для наглядного використання розробленої систему, приведено інструкцію користувача(Додаток Б) . Після введення в пошуковому рядку адресу додатку, а саме https://m-city-caf04.firebaseapp.com, відображається головна сторінка(Рис.3.7)

(Рис.3.7) Головна сторінка

А також адаптивне відображення головної сторінки :

(Рис.3.8) Адаптивне відображення

У головному меню можна обрати одну із сторінок :

– «The Strore» (Рис.3.8) – дає змогу користувачеві переглянути товари, відсортувати їх, а також придбати необхідний продукт;
– «Matches» (Рис.3.9) – відображає інформацію про останні події з можливістю фільтрування даних;
– «Sign In» (Рис.3.10) – дозволяє авторизувати користувача та перейти до панелі адміністратора;

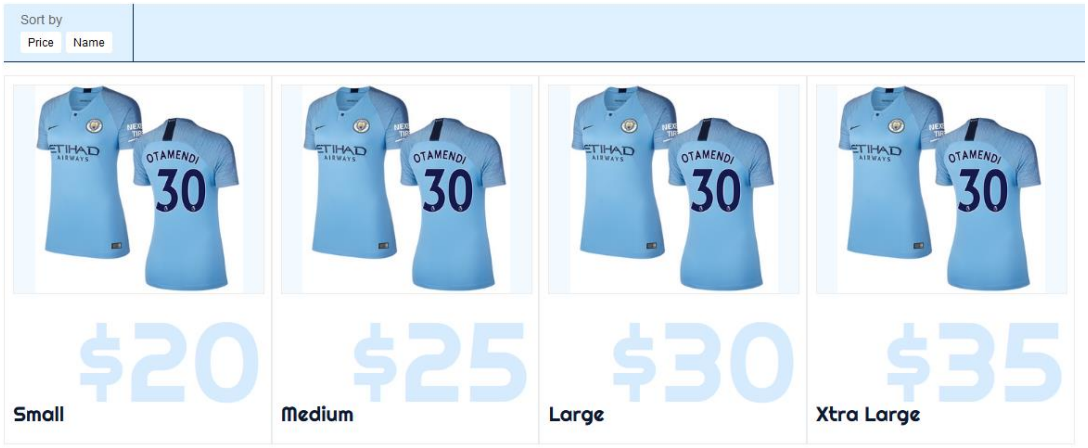


Рис.3.8 Сторінка «The Strore»

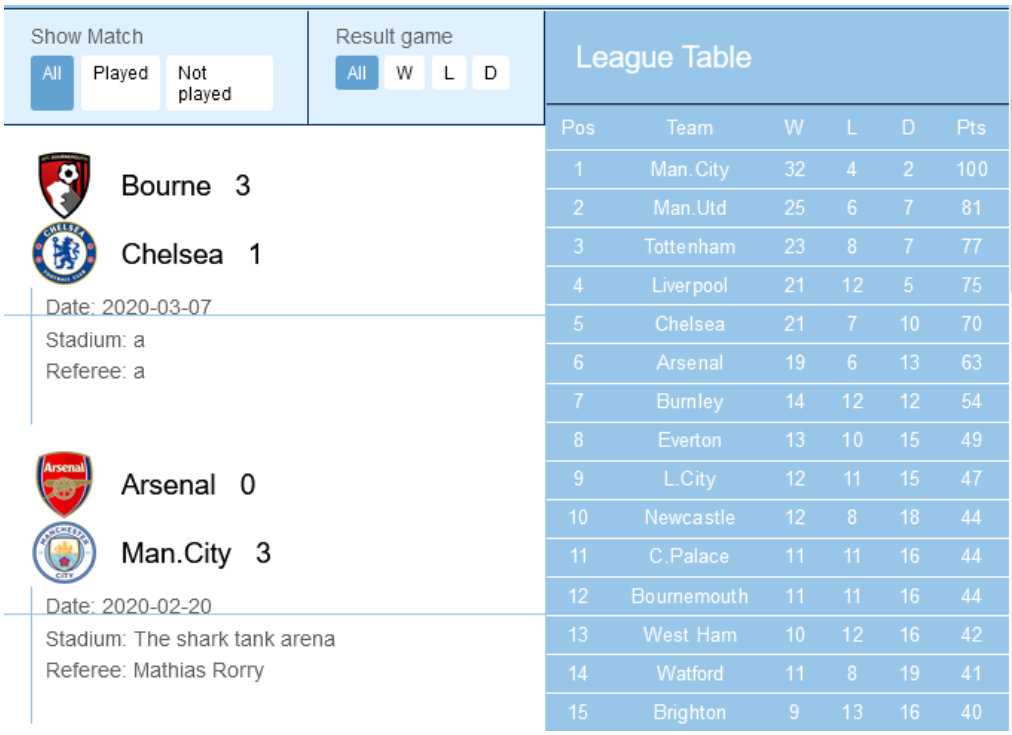


Рис.3.9 Сторінка «Matches»

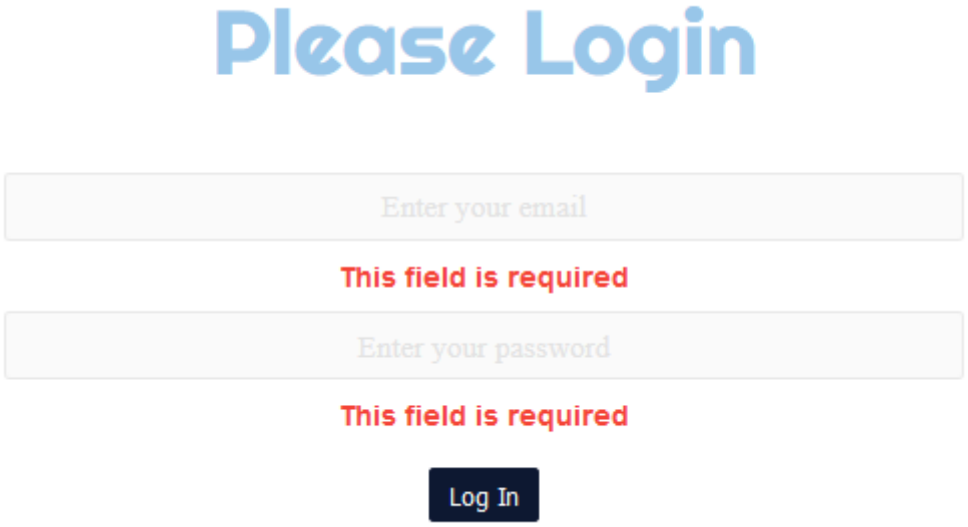
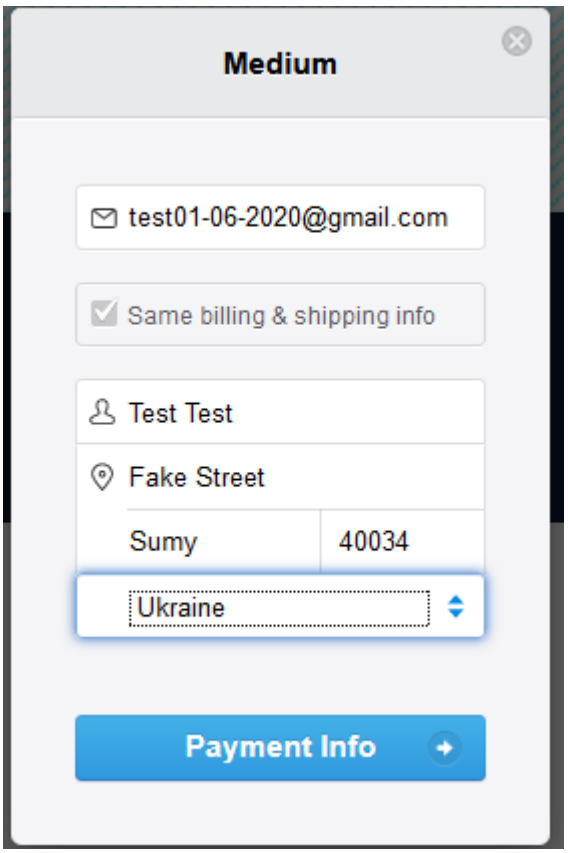
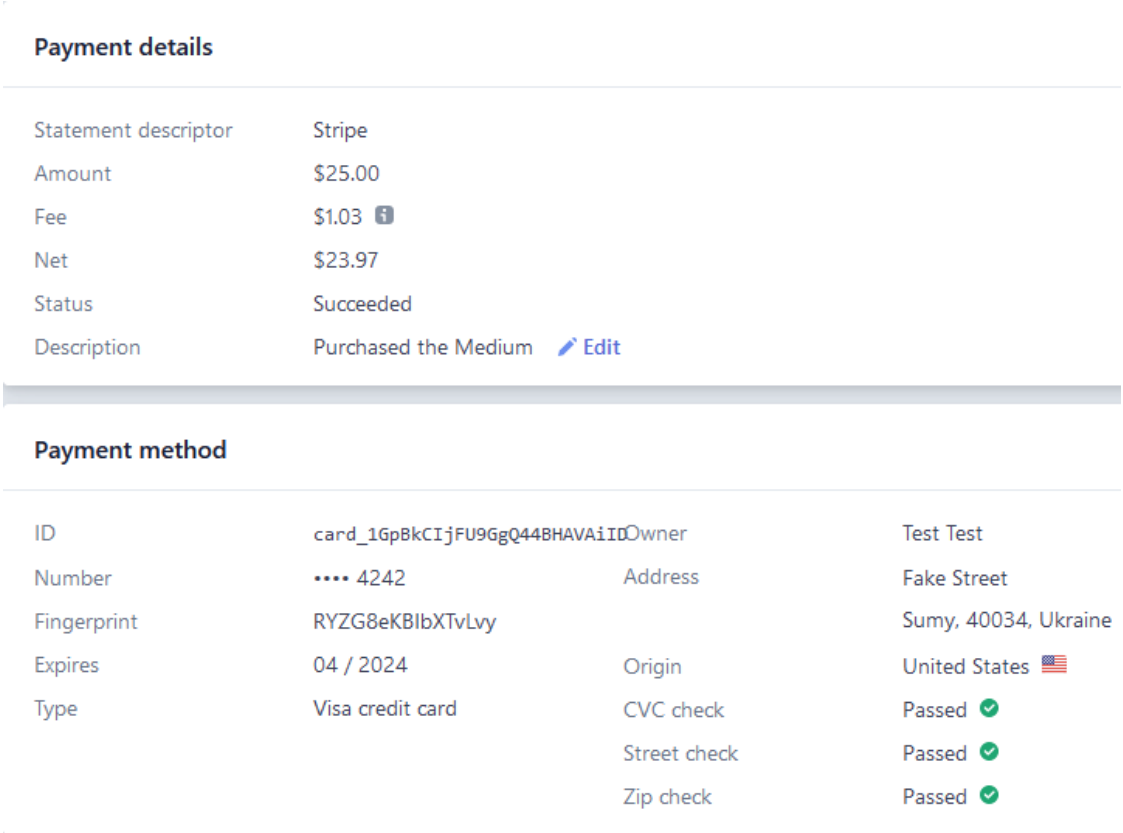


Рис.3.10 Сторінка «Sign In»

Приклад оплати обраного товару з використанням платіжної системи Stripe(Рис.3.11)



(Рис.3.11) Приклад оплати обраного товару



(Рис.3.12) Отримані дані

ВИСНОВКИ

В результаті огляду схожих інформаційних систем дослідження та аналізу подій було вирішено створити власну інформаційну систему, оскільки системи мають як переваги так і недоліки. Серед надоліків: цінова політика, а також надлишкова універсальність деяких систем.

В ході розробки було оглянуто та обрано патерн проектування Flux а також бібліотеку React.js, що дозволило розділити відображення даних від їх обробки та зробило компоненти незалежними один від одного. Для маніпулювання даними було використано базу данних Firebase та створено панель адміністратора доступ до якої мають лише авторизовані користувачі. Також було реалізовано функціонал продажу товарів за допомогою платіжно] системи Stripe.

Дизайну додатку створено з використанням загальноприйнятої мови моделювання UML. На основі якої було розроблено use case діаграму для відображення усіх можливих варіантів використання системи.

Створена автоматизована інформаційна система для онлайн обліку спортивних подій, що підвищить ефективність роботи дослідження та аналізу подій у світі спорту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Design Patterns and Best Practices: Build easy to scale modular applications using the most powerful components and design patterns / 2017, – 320 с.
2. Firebase Cookbook: Over 70 Recipes to Help You Create Real-time Web and Mobile applications / 2017. – 277 с.
3. Ульман Дж. Основы систем баз данных / Пер. с англ. М. Р. Когаловского//, 1983. -334 с.
4. React и Redux: функциональная веб-разработка / 2018, – 336 с.
5. Javascript для чайников. Крис Минник и Ева Холланд / 2017, – 321 с.
6. Веб-разработка с применением Node и Express. Полноценное использование стека javascript / Итан Браун , 2019, – 320 с
7. Выразительный Javascript, 3 издание / Антон Кармазин. 2019, – 756 с.
8. Изучаем JavaScript: руководство по созданию современных веб-сайтов / Флэнаган Дэвид. 2017. – 1080 с
9. Створення ER-діаграм [Електронний ресурс] – Режим доступу до ресурсу: http://inf-teh-lotos.ru/sozdanie-er-diagramm
10. База даних Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Firebase

ДОДАТОК А.

Головний файл index.js в якому знаходиться точка входу програми.

```
const App = props => {
  return (
    <BrowserRouter>
      <Routes {...props} />
    </BrowserRouter>
  )
}

firebase.auth().onAuthStateChanged(user => {
  ReactDOM.render(<App user={user} />, document.getElementById("root"))
})

const Routes = (props) => {
  return (
    <Layout>
      <Switch>
        <PrivateRoutes {...props} path="/admin_players/add_players" exact component={AddEditPlayers} />
        <PrivateRoutes {...props} path="/admin_players/add_players/:id" exact component={AddEditPlayers} />
        <PrivateRoutes {...props} path="/admin_players" exact component={AdminPlayers} />

        <PrivateRoutes {...props} path="/admin_matches/edit_match/:id" exact component={AddEditMatch} />
        <PrivateRoutes {...props} path="/admin_matches/edit_match" exact component={AddEditMatch} />
        <PrivateRoutes {...props} path="/admin_matches" exact component={AdminMatches} />

        <PrivateRoutes {...props} path="/dashboard" exact component={Dashboard} />

        <PublicRoutes {...props} restricted={false} path="/" exact component={Home} />
        <PublicRoutes {...props} restricted={true} path="/sign_in" exact component={SignIn} />
        <PublicRoutes {...props} restricted={true} path="/sign_up" exact component={SignUp} />
```

```
            <PublicRoutes {...props} restricted={false} path="/the_team" exact
component={TheTeam} />
            <PublicRoutes  {...props}  restricted={false}  path="/the_matches"
exact component={TheMatches} />
            <PublicRoutes   {...props}   restricted={false}   path="/the_store"
exact component={TheStore} />
            <PublicRoutes   {...props}   restricted={false}   path="/store/:id"
exact component={TheStoreItem} />
            <PrivateRoutes  {...props}  restricted={true}  path="/admin_store"
exact component={AdminStore} />
            <PrivateRoutes            {...props}            restricted={true}
path="/admin_store/edit_item/:id" exact component={AddEditItem} />
            <PrivateRoutes            {...props}            restricted={true}
path="/admin_store/edit_item" exact component={AddEditItem} />

            <PublicRoutes  {...props}  restricted={false}  component={NotFound}
/>
          </Switch>
        </Layout>
      )
    }
```

А також файл підключення до бази данних firebaseConfig.js

```
import firebase from "firebase/app"
import "firebase/app"
import "firebase/database"
import "firebase/auth"
import "firebase/storage"

const firebaseConf = {
  apiKey: "AIzaSyD9-G4MmzbI37LjuJT1HabcmXoPp-l7t9c",
  authDomain: "m-city-caf04.firebaseapp.com",
  databaseURL: "https://m-city-caf04.firebaseio.com",
  projectId: "m-city-caf04",
  storageBucket: "m-city-caf04.appspot.com",
  messagingSenderId: "87972102603",
  appId: "1:87972102603:web:719c99b47e97cf7f40364e",
  measurementId: "G-DFYFFDTV2X"
}

firebase.initializeApp(firebaseConf)
```

```
const firebaseDB = firebase.database()

//requests
const firebaseMatches = firebaseDB.ref("matches")
const firebasePromotions = firebaseDB.ref("promotions")
const firebaseTeams = firebaseDB.ref("teams")
const firebasePlayers = firebaseDB.ref("players")
const firebaseStore = firebaseDB.ref("items")

//export
export {
  firebase,
  firebaseDB,
  firebaseMatches,
  firebasePromotions,
  firebaseTeams,
  firebasePlayers,
  firebaseStore
}
```

ДОДАТОК Б.
ІНСТРУКЦІЯ КОРИСТУВАЧА

Управління додатком здійстюється за допомогою панелі адміністратора.

Після переходу до сторінки «login» необхідно ввести дані почтової адреси та пароль .Якщо адреса та пароль співпадають з даними у базі, користувач буде перенаправлений до панелі адміністратора (рис. Б.1).

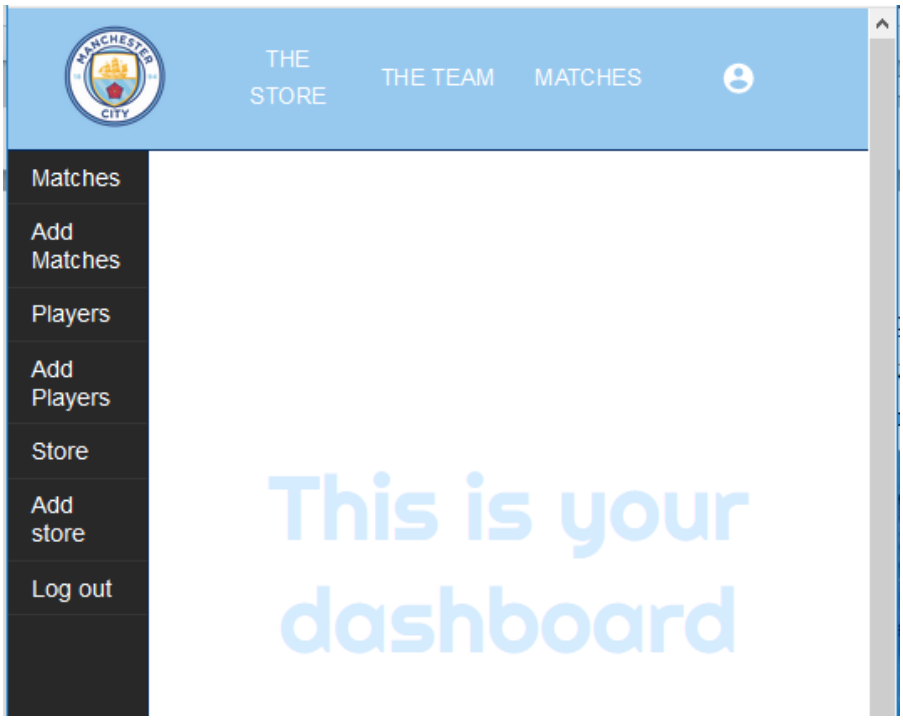


Рисунок Б.1 –Панель адміністратора

Щоб редагувати дані треба обрати відповідну вкладку Для того, щоб відредагувати меню, адміністратор має в лівому меню обрати Вигляд>Меню (рис. Б.2).

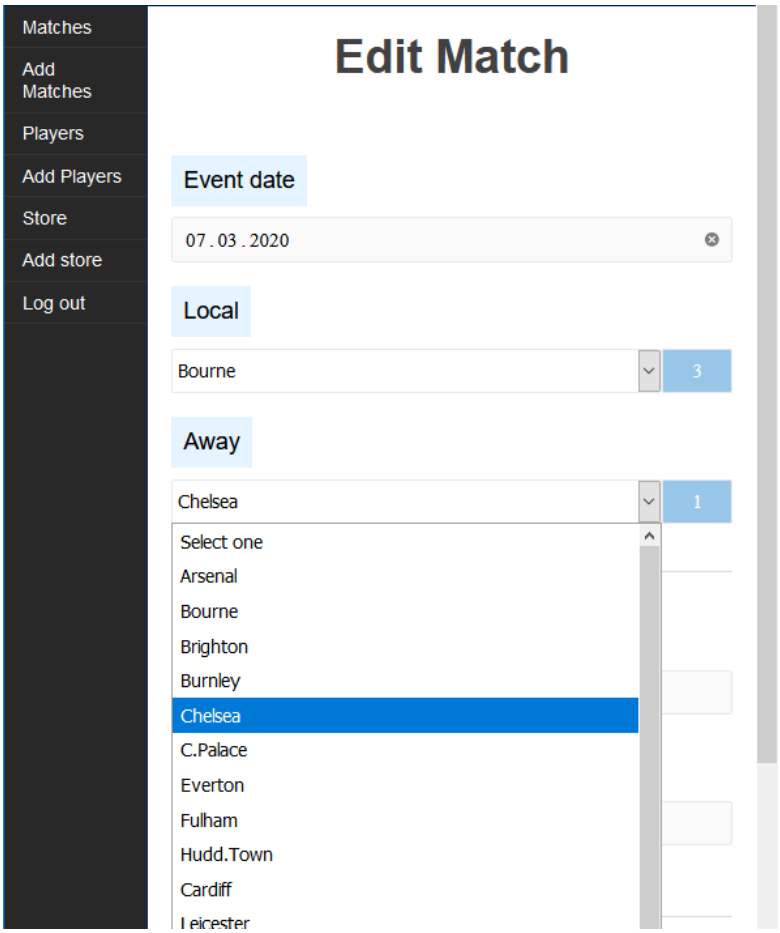
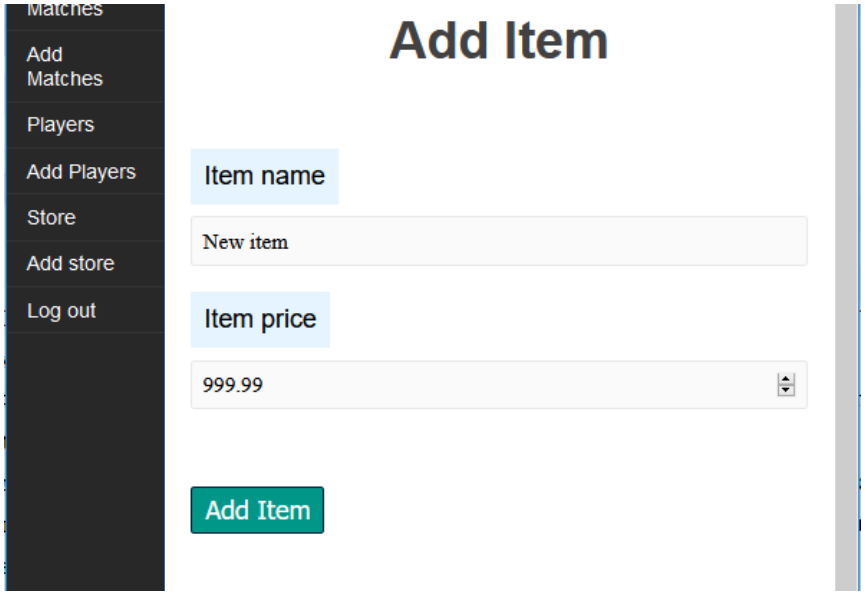


Рисунок Б.2 – Редагування події

Для додавання нового продукту необхідно обрати вкладку store та вкзати дані.



Вкладка Add Players відповідає за дордавання або зміну гравців(Рис. Б.3). При створенні новога запису про гравця, важливо не забути додати зображення для корректного відображення.