Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Розробка веб-застосування для онлайн тоталізатора

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к.ф.-м.н., ст.викл. _____ *Гречко А.В.*
(прізвище та ініціали)

_____
(підпис)
“____” _____ 2020 р.

Виконав студент *ФІ-3*
Коношенко С.А.
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. Кафедри мережних технологій,

проф.,д.ф.-м.н.

_____ Г.І. Малашонок

(підпис)

“____” _____ 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студента Коношенка Сергія Артуровича факультету інформатики 3 курсу ТЕМА: Розробка веб-застосування для онлайн тоталізатора.

Вихідні дані:
Зміст ТЧ до курсової роботи:
Календарний план
Перелік умовних позначень
Вступ
Частина 1: Аналіз предметної області. Постановка завдання курсової роботи
Частина 2: Теоретичні відомості
Частина 3: Опис реалізації програмного продукту
Висновки
Список використаної літератури
Додатки

Дата видачі “____” _____ 2019 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання курсової роботи

Тема: Розробка веб-застосування для онлайн тоталізатора

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	07.10.2019	
2.	Ознайомлення з предметною областю	10.11.2019	
3.	Огляд і аналіз аналогів	25.11.2019	
4.	Пошук технологій для розробки	01.12.2019	
5.	Аналіз обраних технологій	25.01.2020	
7.	Початок створення практичної частини	01.03.2020	
8.	Початок написання теоретичної частини	05.04.2020	
9.	Подання проміжної версії теоретичної частини	03.05.2020	
10.	Подання проміжної версії практичної частини	06.05.2020	
11.	Корегування теоретичної частини	08.05.2020	
12.	Остаточне завершення практичної та теоретичної частини	10.05.2020	
13.	Створення презентації	11.05.2020	
14.	Захист курсової роботи	18.05.2020	

Студенту Коношенку С. А.

Керівник Гречко А. В.

“_____”_____

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – бази даних

Подія – матч, змагання

IDE – Integrated Development Environment

GIL - Global Interpreter Lock

ORM - Object-Relational Mapping

Гандикап – ставка на подію з урахуванням фори

Експрес – ставка на декілька подій

Коефіцієнт події – числовий множник, який виражає вірогідність тієї чи іншої події

Експрес – це ставка, яка складається з двох і більше виборів, в якій програш будь-якого вибору означає програш ставки і цілому.

Ординар – одинична ставка на певну подію

СКБД – система керування базами даних

AJAX - Asynchronous JavaScript and XML

ВСТУП

Онлайн тоталізатори з’явилися не так давно, тому досі здаються чимось новим. Ми легко можемо згадати часи, коли можливість робити ставки і грати за реальні гроші в інтернеті стала можлива. Насправді здається неймовірним те, що це відбулося більше 20 років тому, хоча азартні ігри тоді, безумовно, відрізнялися від сьогоднішніх.

З часів появи перших онлайн тоталізаторів в галузі відбулися численні зміни. В основному вони були направлені на відповідні закони і способи їх регулювання. Також відбулися численні покращення з боку якості сайтів - перші сайти практично невпізнанні в порівнянні з провідними гральними сайтами сьогодні.

Питання про актуальність таких застосунків взагалі не постає – вони переживають свій підйом. Однією з головних переваг онлайн тоталізатора над офлайн є можливість робити ставки, не виходячи з дому і сидячи перед комп’ютером чи телефоном. Вдома нічого не буде вас відволікати, а це дуже важливо для цієї діяльності.

За мету курсової роботи було поставлено дослідити основні можливості онлайн тоталізаторів та проаналізувати сучасні підходи для розробки веб-застосунків.

Завданням курсової роботи є створення веб-застосування для онлайн тоталізатора.

Об’єктом дослідження є створення веб-застосунку для автоматизованого створення, прийняття ставок на певні події.

Предметом дослідження є існуючі аналоги – веб-застосунки, основою яких є ставки на спортивні події в реальному часі.

Основною мовою програмування для розробки було обрано Python, а саме мікро-фреймворк Flask. Також в меншій мірі буде використано мову програмування JavaScript. В якості програмного забезпечення було обрано IDE PyCharm.

Для створення сторінок для браузера було використано шаблонізатор Jinja2.

Для зберігання та використання даних було обрано реляційну базу даних з відкритим кодом PostgreSQL.

Текстова частина курсової роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

В першому розділі буде описано стан розвитку обраної галузі. Буде проаналізовано існуючі аналоги, охарактеризовано основні переваги та недоліки обраних аналогів.

В другому розділі буде описано основні теоретичні відомості області онлайн тоталізаторів і засобів розробки веб-застосунків.

В третьому розділі буде присвячено опису реалізації програмного застосунку, файлів даних, інтерфейсу користувача, конкретизації постановки задачі, пояснення структури програми, тестування.

У висновку буде зазначено підсумки роботи, результати виконання кожного з поставлених завдань і роботи в загальному.

Після висновку буде надано список використаних джерел та додаток з вихідним кодом.

РОЗДІЛ 1

1.1. Аналіз сучасного стану питання та обґрунтування теми

За одним з визначень Академічного словника [1] тоталізатор – це бюро, що приймає грошові ставки на результати яких-небуть подій і сплачує виграші. Головне направлення роботи більшості онлайн тоталізаторів – прийом ставок на спортивні події. Існує багато видів спорту на які приймають ставки: від футболу, тенісу, баскетболу і хокею до крикету, дартсу і кіберспорту.

Більшість сучасних тоталізаторів є букмекером – організацією, в якій ви робите ставки. Завданням букмекера є визначення вірогідності того чи іншого результату чи події, в даному випадку під час спортивної події. Вірогідності вираховуються шляхом аналізу багатьох даних: що може відбутися під час гри з урахуванням минулих подій, фізичний та психологічний стан гравців і зацікавленість грою зі сторони глядачів. Після аналізу букмекер виставляє коефіцієнти на певні події в матчі.

1.2. Огляд існуючих аналогів розробки

Для визначення вимог до веб додатку, що є метою розробки курсової роботи варто проаналізувати аналогічні застосунки, що вже існують на ринку. Комплексний аналіз дозволить сформувати чіткий перелік необхідного функціоналу, виявити переваги та недоліки аналогічних застосунків.

1.2.1. Bet365

Букмекерська компанія Bet365[2] є однією з найстаріших в світі. На даний момент кількість унікальних користувачів в місяць сягає 3300000 [3].

Лінія Bet365 дуже глибока. Вона має більше сорока видів спорту, серед яких є волейбол, футбол, баскетбол, крикет, гольф, кінні перегони і багато інших. Основний функціонал включає:

- Передматчева і live лінії
- Конструктор ставок
- Ставки на статистичні показники

- Ставки на окремі часові проміжки гри
- Онлайн перегляд рахунку матчу

Після дослідження інтерфейсу користувача, можна зробити декілька важливих висновків. Застосунок підтримує 20 мов, але серед них немає української. Інтерфейс надто перевантажений і для нового користувача важко зрозуміти, як працювати в даному застосунку.

Головне меню застосунку зображено на рисунку 1.1

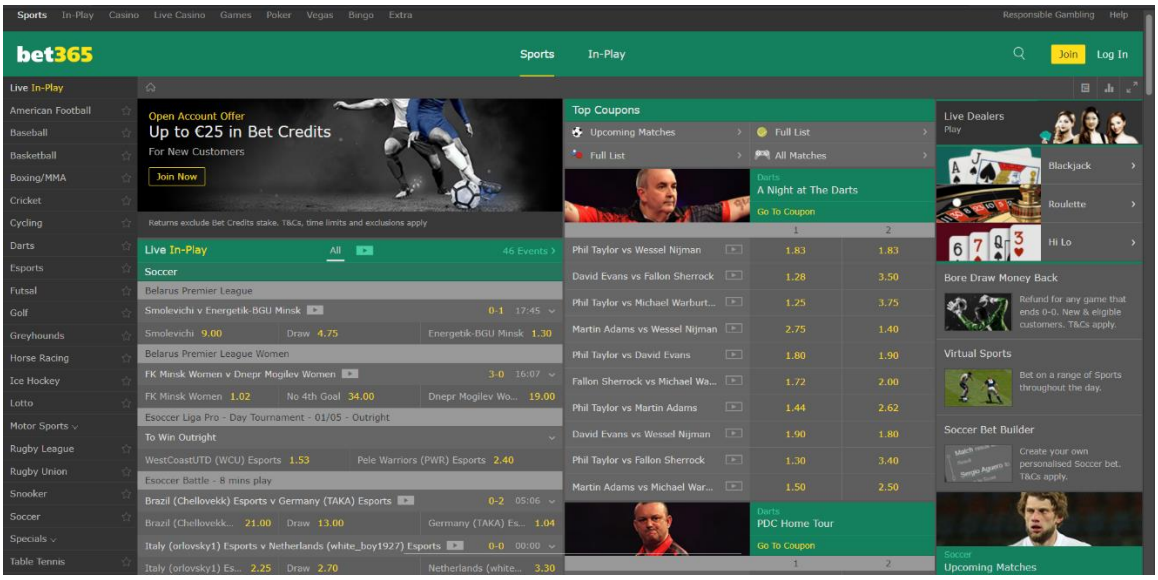

Рисунок 1.1 – Головне меню веб застосунку bet365

Сторінка матчу на bet365 зображена на рисунку 1.2

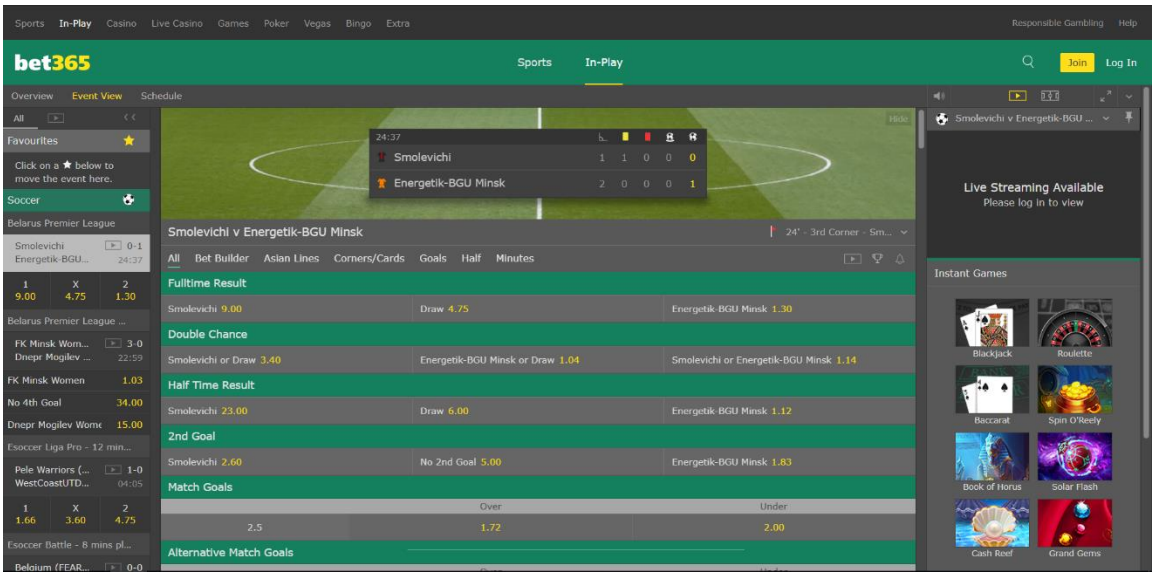

Рисунок 1.2 – Сторінка матчу веб застосунку bet365

З переваг застосунку можна виділити:

- широкий вибір спортивних подій

- великий набір ставок
- відсутній максимальний коефіцієнт по ставкам
- онлайн рахунок матчу

З недоліків можна виділити:

- перевантажений інтерфейс
- відсутність української мови
- всі матчі знаходяться на одній сторінці

1.2.2. Melbet

Melbet[4] є невеликою букмекерською компанією. Кількість унікальних користувачів в місяць не перевищує 10000 [5].

Melbet також містить більше сорока видів спорту, але кількість подій значно менша, в порівнянні з bet365.Основний функціонал подібний до попереднього застосунку, але крім цього ще є можливість перегляду виграшів в реальному часі.

Після дослідження інтерфейсу користувача, можна зробити декілька висновків. Застосунок підтримує 48 мов, серед них є українська. Інтерфейс більш зрозумілий, але відсутня адаптивність дизайну для мобільних пристроїв.

Головне меню зображено на рисунку 1.3

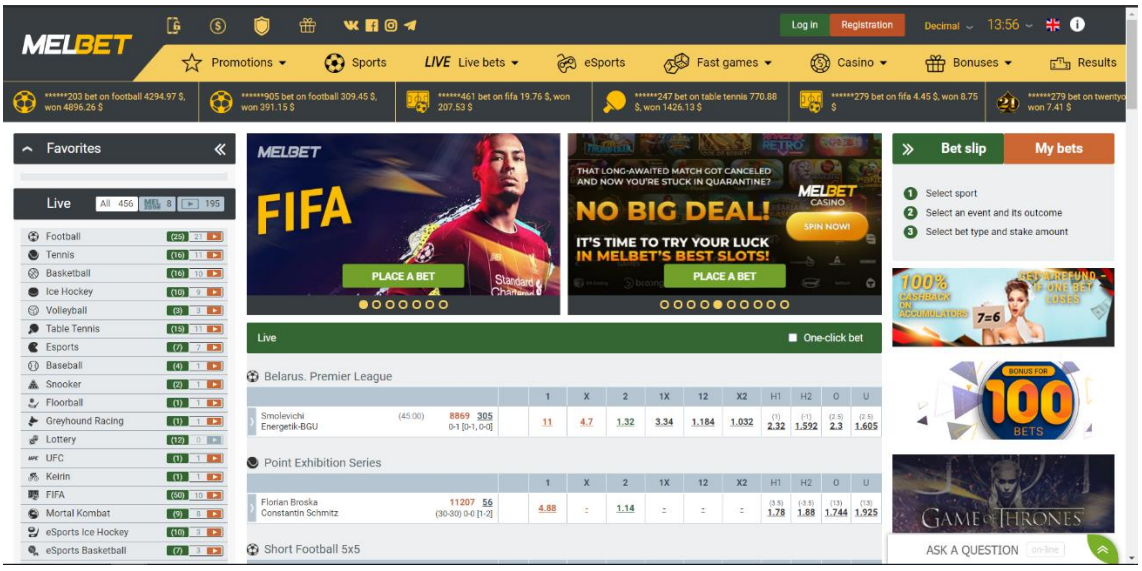

Рисунок 1.3 – Головне меню веб застосунку melbet

Сторінка матчу на melbet зображена на рисунку 1.4

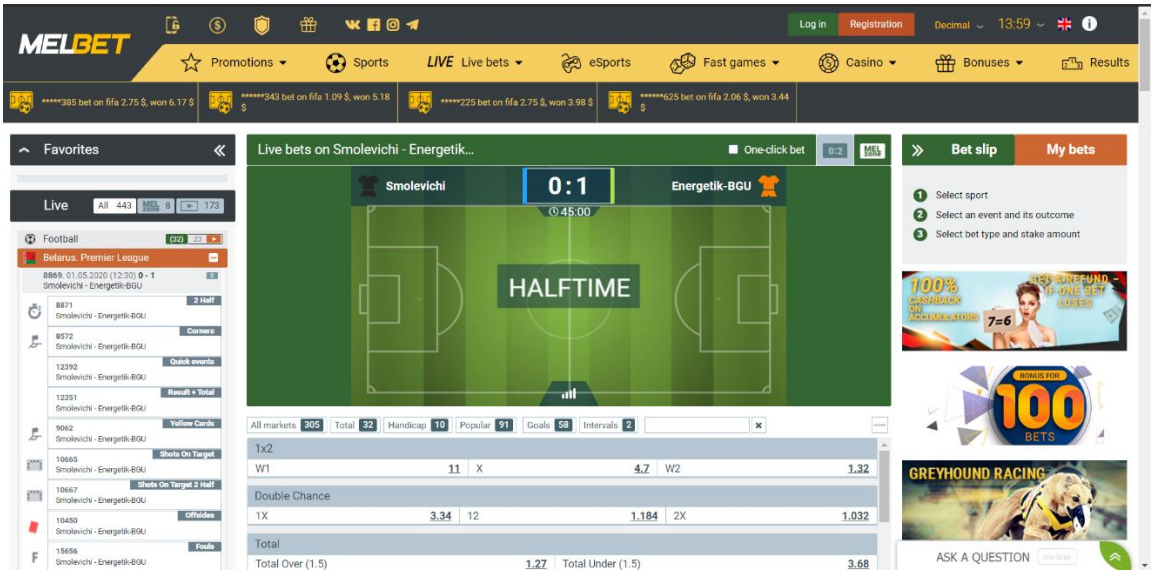

Рисунок 1.4 – Сторінка матчу веб застосунку melbet

З переваг застосунку можна виділити:

- зручний інтерфейс
- перегляд виграшів інших користувачів в реальному часі
- перегляд рахунку матчу в реальному часі
- великий вибір мов

З недоліків можна виділити:

- велика кількість банерів
- невелика кількість подій

1.3. Постановка завдання

Основною задачею є аналіз існуючих програмних продуктів, виділення їх недоліків та переваг і побудова веб-додатку для онлайн тоталізатора з урахуванням можливостей аналогічних продуктів. Створений застосунок має містити такі можливості:

- Реєстрація та авторизація
- Перегляд інформації про користувача
- Перегляд списку подій
- Створення ординару на подію

- Створення експресу
- Інтерфейс адміністратора
- Перегляд попередніх подій

Додатково потрібно:

- розробити інтуїтивно зрозумілий інтерфейс
- створений інтерфейс має бути адаптивним, тобто вигляд сайту має змінюватися в залежності від розміру екрану

РОЗДІЛ 2

2.1. Опис типів ставок

Перед початком розробки треба більш детально охарактеризувати відомості про онлайн тоталізатори.

Основні типи ставок[6]:

- Основний результат – W1, X, W2

  Основний тип ставок, в якому потрібно обрати один з трьох можливих результатів матчу: перемога першої команди W1, перемога другої команди W2 чи нічия X. В деяких видах спорту, наприклад в тенісі, більярді, дартсі, гольфі нічийний результат неможливий, тому передбачені тільки перемога однієї з команд.

  Наприклад: гравець ставить W1 на матч між командою А та Б. Ставка вважатиметься виграшною тільки при умові перемоги команди А з будь-яким рахунком. Якщо переможе команда Б чи гра завершиться нічиєю, то це програш.

- Подвійний шанс – 1X, 12, X2

  Такий тип ставок схожий на попередній, але він менш ризикований, бо два з трьох основних результатів матчу (W1, X, W2) є переможними. 1X – не перемога першої команди, 2X – не перемога другої команди, 12 – перемога однієї з команд.

  Наприклад: гравець ставить на 1X на матч між командою А та Б. Ставка вважатиметься виграшною, якщо команда А не програє. Тобто влаштовує і перемога команди А і нічия.

- Тотал більше або менше – ТБ(x) і ТМ(x), де x – будь-яке число

  Ставка на тотал – це сумарний результат матчу, який може бути відображеним в футболі - сумою забитих голів, в тенісі – сумою зіграних геймів, в баскетболі – закинутих очок. Менш популярні тотали на кутові, ейси і т. д. В ставці на тотал основне не те, хто програє чи виграє, а який результат матчу в сумі. Тотал виражається цілим чи дробовим числом.

Наприклад: гравець поставив на ТБ(4,5) в матчі між командами А та Б. Для виграшу необхідно, щоб сума отриманих очок, голів в матчі була більша, ніж 4,5.

Також існують цілочисельні тотали. Їх особливість полягає в тому, що окрім виграшу чи програшу ставки можливе повернення. Наприклад, ТМ(4,0) – тотал менше трьох очок(голів). Якщо результуюча сума більша 4 – то програш, якщо менша 4 – виграш, а якщо рівна 4 – повернення ставки з коефіцієнтом 1:1.

- Індивідуальний тотал

  Цей вид аналогічний попередньому, різниця полягає в тому, що треба відгадати не загальний результат матчу, кількість отриманих очок(забитих голів) однією командою.

- Азіатський гандикап

  Один з найпопулярніших видів ставок в онлайн тоталізаторах, який дозволяє ставити за будь-яку команду з встановленою форою, яка може бути з додатним або від'ємним знаком. Ставка вважається переможною, коли обрана команда перемагає з урахуванням фори.

  Наприклад: гравець ставить Ф1(-4) на гру між командами А та Б. В даному прикладі Ф1(-4) означає ставку на перемогу команди А з форою в мінус 4 очка. Якщо команда А перемагає з різницею в 5 і більше очка – ставка виграшна, якщо перемагає з різницею в 4 очка – повернення ставки, якщо команда А перемагає з різницею менше ніж 4 очка або програє – ставка програшна.

  Також існують дробові значення фори – Ф1(-2,5), Ф2(+3,5) і так далі.

- Тайм-матч

  Така ставка складається з двох частин – результат першого тайму(сету, періоду) і результат матчу в цілому. Тобто ставка є виграшною лише тоді, коли правильно спрогнозовано результат першого тайму(сету, періоду) і матчу.

Наприклад: гравець робить ставку X/W1 в грі між командами А та Б. Це означає, що буде нічия в першому таймі і перемога команди А в матчі.

Більшість онлайн тоталізаторів має таки види ставок:

- Ординар

  Ординар – ставка на те, що відбудеться певна подія. Подією в тоталізаторах вважається перемога чи поразка учасників гри, або нічия, або ще велика кількість варіантів, придуманих букмекерською конторою від рахунку матчу до передбачення більш конкретних подій в грі(наприклад, для футболу: на якій хвилині буде перших гол, скільки червоних карток буде, чи заб'є певний гравець). Розрахунок виграшу по ординару проводиться за такою формулою

  $P_{\text{орд}} = S * K$ де

  $P_{\text{орд}}$ – сума виплати ординару,

  S – сума ставки

  K – встановлений букмекером коефіцієнт на вірогідність вибраного результату.

№	Дата	Подія	W1(Перемога 1)	X(Нічия)	W2(Перемога2)
7067	27/03 20:00	Болонья Кальярі	2,50	3,40	2,60

Таблиця 2.1 – Приклад ординарної ставки

Припустимо, що ми хочемо зробити ставку в розмірі 1000 гривень на перемогу Болоньї. Коефіцієнт виграшу по ставкам на перемогу Болоньї – 2,50. Виплата при перемозі Болоньї становитиме

$$P_{\text{орд}} = 1000 * 2{,}50 = 2500$$

Власне виграш становить 2500 (виплата) – 1000 (ставка) = 1500

- Експрес

Експрес складається з кількох ординарів. Для того, щоб виграти експрес потрібно передбачити результати всіх подій в експресі. Якщо одна з подій не передбачена то експрес вважається програшним. Якщо ж всі події передбачені правильно, то проводиться виплата, для обрахування якої використовується формула

$P_{\text{екс}} = S * K_1 * K_2 * K_3 * \ldots * K_n$ де

$P_{екс}$ – сума виплати експресу

S – сума ставки

$K_{(n)}$ – коефіцієнт на результат n зі списку експрес ставок

№	Дата	Подія	W1(Перемога 1)	X(Нічия)	W2(Перемога2)
6383	03/04 16:00	Лаціо Кальярі	1,23	5,90	9,90
6389	04/04 13:30	Лечче Брешіа	2,01	3,30	3,60
6387	04/04 16:00	Парма Наполі	3,01	3,45	2,14
6388	05/04 16:00	СПАЛ Рома	4,50	3,85	1,67

Таблиця 2.2 Приклад експрес ставки

Припустимо, що ми хочемо зробити ставку на експрес в розмірі 100 гривень, який складається з чотирьох результатів: перемога Лаціо, нічия між Лечче і Брешіа, перемога Парми і перемога Роми. У разі перемоги виплата буде

$$P_{\text{екс}} = 100 * 1{,}23 * 3{,}30 * 3{,}01 * 1{,}67 = 2040{,}34$$

Власне виграш: 2040,34 (виплата) – 100 (ставка) = 1940,34

- Система

За допомогою ставок системою можна сформувати задане число комбінацій експрес ставок із обраних результатів. Основною характеристикою є однаковий розмір ставки на кожну подію і однакова

кількість результатів в кожному експресі(розмірність). Під час ставки на систему необхідно вказати загальну кількість можливих результатів і розмірність експресу. Виграш в системі дорівнює сумі виграшів експресів, які входять в систему.

Приклад ставки по системі:

1. Ліверпуль -1,5 (2,6) – Тоттенхем +1,5(1,9)
2. Іспанія -1,5 (1,7) – Люксембург +1 (2,3)
3. Манчестер Юнайтед -1 (1,4) – Арсенал Лондон +1 (2,5)
4. ПСЖ -1 (1,7) – Баварія +1 (1,9)

З чотирьох подій є можливість зробити ставка на систему з

А) шести результатів по два

Б) чотирьох результатів по три

В першому випадку це означає, що ми робимо ставку на шість запропонованих експресів, які містять по два варіанти результатів для кожного з обраних подій. Сума ставки розкладається на кожний з шести експресів:

1) Ліверпуль -1,5 (2,6) – Тоттенхем +1,5(1,9)
   Іспанія -1,5 (1,7) – Люксембург +1 (2,3)
2) Ліверпуль -1,5 (2,6) – Тоттенхем +1,5(1,9)
   Манчестер Юнайтед -1 (1,4) – Арсенал Лондон +1 (2,5)
3) ПСЖ -1 (1,7) – Баварія +1 (1,9)
   Ліверпуль -1,5 (2,6) – Тоттенхем +1,5(1,9)
4) Іспанія -1,5 (1,7) – Люксембург +1 (2,3)
   Манчестер Юнайтед -1 (1,4) – Арсенал Лондон +1 (2,5)
5) Іспанія -1,5 (1,7) – Люксембург +1 (2,3)
   ПСЖ -1 (1,7) – Баварія +1 (1,9)
6) Манчестер Юнайтед -1 (1,4) – Арсенал Лондон +1 (2,5)
   ПСЖ -1 (1,7) – Баварія +1 (1,9)

Сума виплати по системі розраховується як сума виграшних експресів з цих 6.

В другому випадку це означає, зо ми робимо ставку на чотири експреси по 3 результати з обраних подій в кожному з них. Сума ставки розбивається на чотири рівні частини для кожного з експресів. Сума виплати ставки на систему чотири по три буде дорівнювати сумі виграшних експресів по кожному з чотирьох подій.

2.2. Опис сайту

Після аналізу типових можливостей онлайн тоталізаторів необхідно вирішити, який тип сайту найкраще підійде для розробки такого застосунку: статичний або динамічний.

Статичний сайт – сайт, який містить незмінні сторінки. Для того щоб додати або змінити інформацію на статичній сторінці потрібно самостійно внести зміні в код сторінки. Серед плюсів статичних сайтів можна виділити те, що такі сайти створюють невелике навантаження на сервер, мають високу швидкість завантаження. Однак обрана галузь вимагає постійне оновлення інформації на сторінці, а фахівців веб-розробки та формування коефіцієнтів невелика кількість. Тому сайт, який розроблюється на даній роботі буде динамічним і містити інтерфейс адміністратора для додавання подій.

Для покращення процесу розробки необхідно визначити вид розроблюваного сайту. Основними видами є[7]:

- Інтернет-магазин

  Основним завданням такого сайту є продаж товару. Тому в користувача має бути можливість обрати товари і провести замовлення.
- Сайт-візитка

  Простий сайт з невеликою кількістю сторінок. Зазвичай це статичний сайт, який містить інформацію про певну компанію і контакти для зв’язку.
- Сайт компанії з каталогом продукції

  Ускладнений варіант сайту-візитки. Крім інформації про компанію також містить каталог послуг або товарів, які ця компанія надає.

- Новинні портали

  Сайт, основним завданням якого, є надання доступу до новин. Зазвичай в базі зберігається інформація з новин за увесь час, тому такі сайти вимагають багато часу для розробки і просування.

- Інформаційні портали

  На таких сайтах розміщуються статті на різні тематики. Також вони можуть об’єднувати інформацію, зібрану з інших ресурсів.

- Системи обліку, бухгалтерії

  Такі сайти призначені для внутрішнього використання фірми, тому зазвичай доступ до них закритий. Подібні сайти об’ємні і складні в розробці, але їх зручність окупає ціну.

- Форум

  Сайт, на якому відбувається обговорення тем. Складається зі стрічки, яка містить перелік створених користувачами тем. Користувач може перейти на певну тему і залишити повідомлення в ній

- Соціальні мережі

  Сайт, який створений для зручного спілкування між людьми. Зазвичай має простішу структуру і більш зручний інтерфейс, ніж сайти-форуми. Також соцмережі більш персоналізовані.

- Системи бронювання готелів, товарів

  Зростання обсягів туризму відображається на транспортній і комунікаційній сферах, які під впливом зростаючого попиту на подорожування стали одними із головних споживачів систем комп'ютерного бронювання, електронних систем інформації і комунікацій. Системи бронювання складають значну частину веб-застосунків, оскільки надають послуги проживання в готелях, оренду автомобілів, круїзні поїздки.

- Сайти-сховища

  До сайтів такого типу відносяться сайти, основне призначення яких це відображення відео або аудіо матеріалів в великій кількості.

- Сайт-блог
- Розважальні портали
- Фінансові портали

Окрім зазначених типів існують сайти, що містять в собі кілька типів. Онлайн тоталізатор є комбінацією типів сайтів, бо в ньому присутні фінансові операції, він є розважальним порталом, а також він може бути новинним порталом.

РОЗДІЛ 3

3.1. Аналіз технічного завдання

Основне призначення створюваного веб-застосунку – можливість ставити на певні події матчу. Для цього в застосунку повинне бути розділення по ролям: адміністратор, який може додавати нові події і клієнт, який може переглядати список матчів і ставити на них умовні гроші. Щоб переглядати деталі матчів клієнт має авторизованим.

Алгоритм роботи застосунку:

1) Адміністратор створює матч і додає певні події до нього (наприклад тотали, гандикапи). Після створення матч має статус pending.
2) Клієнт переглядає список матчів, переходить на один з них і робить на ставку ординаром або експресом, якщо вона має статус pending.
3) На початку гри статус матчу змінюється на waiting results і клієнти більше не мають можливості робити на нього ставки.
4) Після завершення гри, адміністратор виставляє результати. Клієнти, які перемогли отримують гроші, згідно з коефіцієнтами. Статус матчу змінюється на finished.

Клієнт має власний рахунок, який можна поповнити. Також він може переглянути свої минулі ставки.

3.2. Обґрунтування алгоритму й структури програми

Застосунок побудовано з використанням клієнт-серверної архітектури[8].

Додаток розділений на клієнтів і сервер. Клієнт не має прямого доступу до даних, які залишаються всередині серверу, тому мобільність коду клієнта збільшується. Сервер не пов’язаний з інтерфейсом користувача, тому вони простіше масштабуються. Сервер і клієнт можуть легко замінятися, якщо інтерфейс взаємодії між ними не змінюється.

Як зображено на рис 3.1) після авторизації на сайті користувач має роль клієнта або адміна. Клієнт має можливість переглядати список матчів,

робити на них ставки, переглядати свої попередні ставки. Адмін має можливість додати новий матч, додати події до матчу, встановити для подій коефіцієнт. Після завершення матчу адмін має виставити результати.

Розділення користувачів на різні ролі необхідне для обмеження доступу до певних функцій застосунку. Звичайний клієнт не повинен мати можливість впливати на встановлені дані матчу.

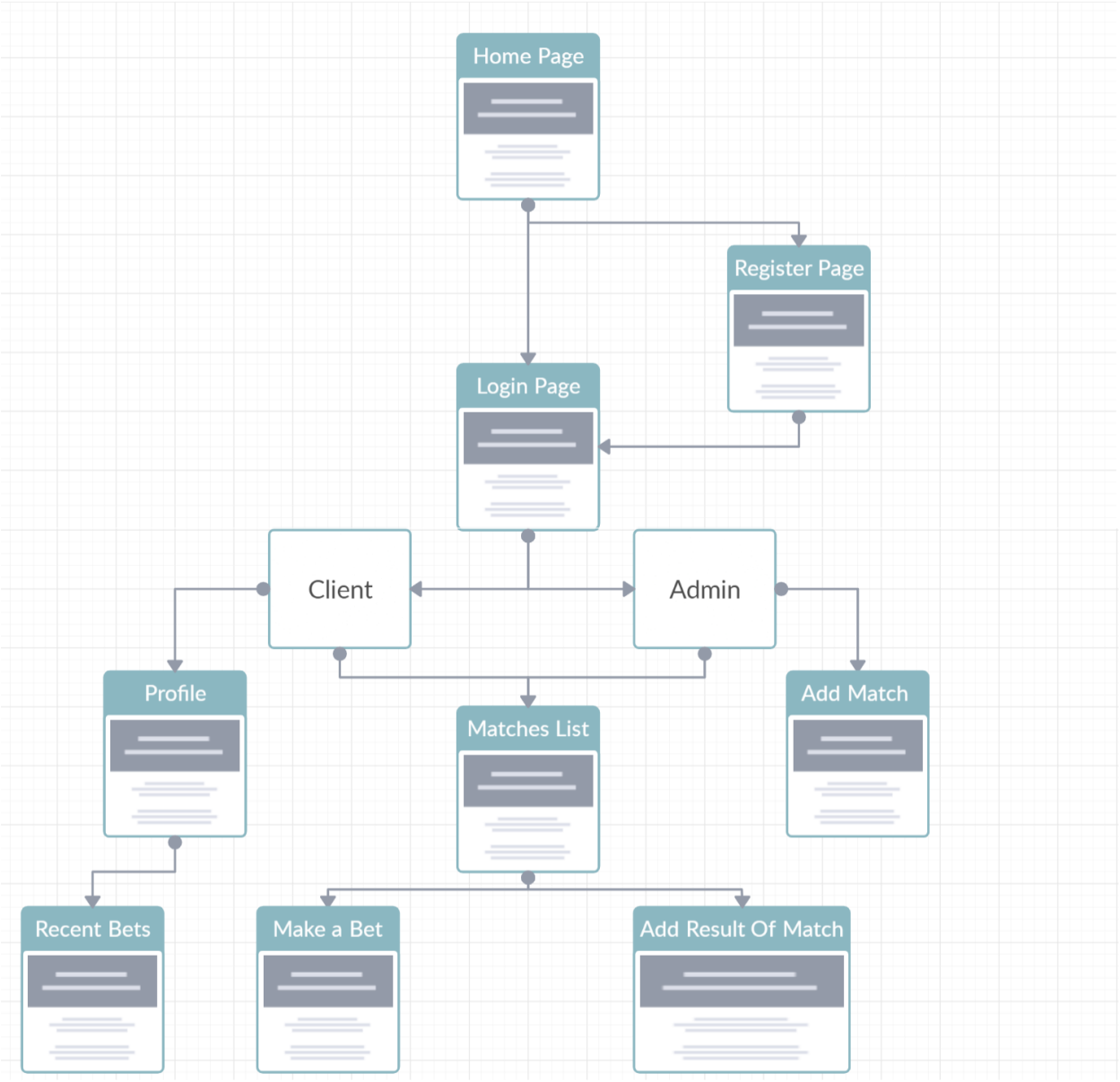

Рисунок 3.1 – Структура сайту

3.3. Обґрунтування вибору засобів розробки

Для розробки серверної частини було обрано мову програмування Python.

Python є високорівневою динамічно типізованою мовою програмування. В Python реалізовані об’єктно-орієнтована, функціональна, імперативна парадигми програмування. В Python автоматичне управління пам’яттю, механізм обробки виключень, підтримка багатопоточних обчислень.

В багатьох застосунках є необхідність в використанні асинхронних операцій вводу-виводу. Зазвичай для цього використовуються потоки. Але насправді програми на Python, які використовують багато потоків, не виконуються паралельно. В класичній реалізації мови програмування Python – CPython існує механізм GIL, який дозволяє тільки одному потоку виконувати байт-код в одиницю часу. Інтепретатор переключається між потоками і так імітує одночасне виконання, хоча це і призводить до втрат. Хоча такі затратні операції як операції вводу-виводу і робота з зображеннями виконуються поза межами GIL, створення потоків може значно погіршити швидкодію застосунку, тому треба обережно з ними працювати або використовувати асинхронні операції.

Серед переваг використання Python для веб розробки можна виділити:

- Продуманий дизайн

  Використати Python для написання архітектурно правильного коду простіше, ніж, наприклад, в PHP чи JavaScript.
- Python більш читабельний

  Сама філософія Python це читабельність коду
- Легкодоступні інструменти для відладки

  Python має велику екосистему для розробників і легкодоступні інструменти відладки. Python Debugger доволі потужний відладник, який простий в використанні і добре задокументований.
- Система управління пакетами

  Управління покетами є зв’язуючим елементом між різними проектами. З його допомогою є можливість створювати і обмінюватися пакетами в такому форматі, щоб інші розробники могли легко підключати їх до своїх застосунків.

Для середовища програмування була обрана PyCharm

PyCharm – інтегроване середовище розробки для мови програмування Python. Дане середовище програмування надає засоби для аналізу коду, графічний відладчик, інструмент для запуску юніт-тестів і підтримує розробку на Django. PyCharm розроблена компанією JetBrains на основі IntelliJ IDEA.

Середовище програмування PyCharm надає такі можливості:

- статичний аналіз коду і підсвічення синтаксису і помилок
- швидка навігація між файлами проекту завдяки деревоподібній структурі файлів
- покращений рефакторинг
- інструменти для веб-розробки
- внутрішній відладчик для мови програмування Python
- підтримка систем контролю версій, таких як Mercurial, Git, Perforce і так далі
- підтримка IronPython, Jython, Cython, PyPy, PyQt
- підтримка Flask фреймворка і шаблонізаторів Mako і Jinja2

Даний застосунок має бути розроблений для веб, тому він повинен мати можливість приймати запити сервера і відправляти відповідь у вигляді HTML сторінок. Для цього потрібен веб-сервер. Існує багато способів створити власний веб-сервер, які зможуть оброблювати HTTP запити користувачів і повертати їм в браузери результат. Основною мовою програмування є Python, тому бібліотеку має бути написана на Python.

Flask – це мікрофреймворк для Python, який надає всі основні функції необхідні для веб-застосунку, такі як: веб-сервер, система Blueprints, маршрутизація, рендеринг шаблонів, cookies, логування.

Для роботи з сесією користувача використано модуль Flask-User, який надає можливість простого управління операціями входу, виходу користувача.

Для структурованого збереження даних про користувачів, подій, ставок необхідно використати бази даних. Була обрана реляційна база даних PostgreSQL. Для зручної роботи з базами даних використано модуль Flask-

SQLAlchemy, яка побудована на технології ORM. ORM – це технологія програмування, яка встановлює об’єктно-реляційний зв’язок. Бази даних пов’язуються з концепціями об’єктно-орієнтованого програмування: замість реляцій використовуються класи. Серед переваг варто виділити можливість автоматично створювати SQL-запити, які перевірені і оптимізовані, можливість працювати з базою даних використовуючи елементи мов програмування (класи, атрибути, методи), ізоляція коду програми від деталей збереження даних.

Для зручного оновлення і переносу схеми бази даних використано модуль Flask-Migrate

В парі з Flask в більшості випадків використовується шаблонізатор Jinja2. Jinja2 надає можливість налаштування тегів, тестів, фільтрів, глобальних параметрів. З його допомогою можна викликати функції з об’єктами у якості аргументів.

3.4. Опис розробки програми

Застосунок побудований з використанням клієнт-серверної архітектури. Клієнт-серверні системи складаються з двох частин: серверу, який надає дані і клієнта, який робить запит до даних. Разом вони формують цілісну систему з чітким розподіленням ролей.

Клієнт, використовуючи браузер, здійснює запит на сервер і отримує у відповідь або дані або сторінку сайту, в залежності від запиту.

Сервер оброблює всі запити, які отримує, за необхідності звертається до бази даних засобами СКБД (Система керування базами даних), генерує HTML сторінку і відправляє її клієнту.

Сервер розділений на декілька модулів:

- controllers

  Містить код, який отримує запити, перевіряє їх правильність, робить запит на отримання даних з сервера, генерує HTML сторінку з даними зі серверу і відправляє її клієнту
- database

Містить моделі бази даних і запити до неї. Контролери звертається до коду, що знаходиться в цьому модулі щоб отримати дані з БД

- forms

  Містить форми, які можуть використатися при створення HTML сторінки. Форми відділено для полегшеного керування ними.

- templates

  Містить Jinja2 шаблони. Контролер при створенні HTML сторінки обирає потрібний шаблон, підставляє необхідні дані, отримані зі серверу і відправляє клієнту.

- static

  Містить код, який використовується в HTML сторінках, такий як клієнтські скрипти, AJAX запити, написані на мові програмування JavaScript з використанням бібліотеки JQuery. Також містить файли таблиць стилів CSS.

- migrations

  Містить міграції – різні версії схем бази даних. Міграції потрібні для безпечної зміни схеми бази даних.

3.5. Створення об’єктів і розробка головної програми

При запуску застосунку створюється екземпляр класу Flask, якому першим аргументом має передаватися назва модуля програми. Оскільки в застосунку існує один модуль, то я передаю його через __name__. Це потрібно для того, щоб Flask знав де шукати шаблони, статичні файли тощо.

```
app = Flask(__name__)
```

Далі об’єкту app вказується об’єкт конфігурації, який знаходиться в файлі config.py:

```
app.config.from_object(config_object)
```

Об’єкт config_object містить налаштування проекту такі як порт, хост, секретний ключ, CSRF токен, стрічка підключення до бази даних:

```
CSRF_ENABLED = True
PORT = 8000
HOST = "localhost"
SECRET_KEY = 'course-work-totalizator-secret'
```

```
DB_NAME = os.getenv('DB_NAME')
DB_PORT = os.getenv('DB_PORT')
DB_HOST = os.getenv('DB_HOST')
DB_PASS = os.getenv('DB_PASS')
DB_USER = os.getenv('DB_USER')

SQLALCHEMY_DATABASE_URI =
f"postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
SQLALCHEMY_TRACK_MODIFICATIONS = True
```

а також налаштування Flask такі як:

```
USER_ENABLE_EMAIL = False
USER_ENABLE_USERNAME = True

USER_ENABLE_CONFIRM_EMAIL = False

USER_LOGIN_TEMPLATE = 'flask_user/login.html'
USER_REGISTER_TEMPLATE = 'flask_user/register.html'
```

Далі створюється екземпляр класу SQLAlchemy і цей екземпляр підключається до об’єкту Flask:

```
db = SQLAlchemy()
db.init_app(app)
db.app = app
```

Також створюється об’єкт Migrate, який потрібен для зручної міграції схеми баз даних:

```
migrate = Migrate(app, db)
```

Всі дані для створення таблиць баз даних беруться з файлу models.py. Всі класи в цьому файлі наслідуються від класу Model, який необхідний для коректної роботи SQLAlchemy. Приклад класу з файлу models.py:

```
class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(60), nullable=False, unique=True)

    def __repr__(self):
        return f'<Role {self.name}>'
```

Далі створюється об’єкт класу UserManager, якому передається модель користувача User для того, щоб на його основу робити реєстрацію та авторизацію користувачів, а також перевірку повноважень для доступу до певних даних:

```
user_manager = UserManager(app, db, UserClass=User)
```

До написаних запитів додається об’єкт Flask, бази даних і UserManager:

```
db_queries.init_db(db, user_manager)
```

Приклад запиту з db_queries:

```
def create_user(self, username, password, email, balance=0,  is_admin=False):
    user = User(username=username, balance=balance, email=email)
    user.set_password(password)
    if is_admin:
        user.roles.append(self._get_or_create(Role, name='admin'))
    self.db.session.add(user)
    self.db.session.commit()
    return user
```

Далі додаються Blueprints – вони потрібні для зручного розділення коду. Наприклад, частина коду для перевірки існування користувача використовується в різних застосунках. Щоб перенести це в інше програму необхідно з декількох модулів взяти частини коду і додати їх в нові файли нового проекту. Це не зручно. Тому набагато краще, коли код, пов’язаний з аутентифікацією знаходиться в окремій частині програми. Для цього і потрібні Blueprints.

В застосунку реєструються такі blueprints: game_blueprint відповідає за створення ставок, admin_blueprint відповідає за створення і управління подій та матчів, authentication_blueprint відповідає за авторизацію та роботу з профілем користувача.

```
app.register_blueprint(authentication_blueprint)
app.register_blueprint(game_blueprint)
app.register_blueprint(admin_blueprint)
```

Кожний blueprint знаходиться в окремому файлі і містить створення самого blueprint:

```
authentication_blueprint = Blueprint('authentication', __name__)
```

та методи для обробки запитів.

Приклад коду, який знаходиться в authentication_blueprint і відповідає за реєстрацію користувача:

```
@authentication_blueprint.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect('/')
    form = UserRegistrationForm()
```

```
    if form.validate_on_submit():
        db_queries.create_user(username=form.username.data,
email=form.email.data, password=form.password.data)
        flash('Congratulation, you are registered')
        return redirect('/login')
    return render_template('authentication/register.html', title='Register',
form=form)
```

3.6. Опис файлів даних та інтерфейсу програми

3.6.1. Опис файлів даних

Для даного застосунку було використано реляційну базу даних PosgtgreSQL та СКБД для Python.

Використання реляційних баз даних та СКБД надає такі можливості:

- Контроль за надлишковістю та несуперечливістю даних
- Підтримка цілісності даних (цілісність бази даних означає коректність та несуперечливість даних, що в ній зберігаються).
- Незалежність прикладних програм від даних
- Спільне використання даних
- Забезпечення безпеки

Спільне використання даних та підвищений рівень безпеки забезпечується за допомогою таких засобів:

А) системи забезпечення захисту, яка контролює список користувачів, які мають доступ до бази даних

Б) системи відновлення, яка дозволяє відновити базу даних до попереднього несуперечливого стану

В) системи керування паралельною роботою прикладних програм

На рис. 3.2) зображено схему бази даних, яка була використана при розробці веб-застосунку.

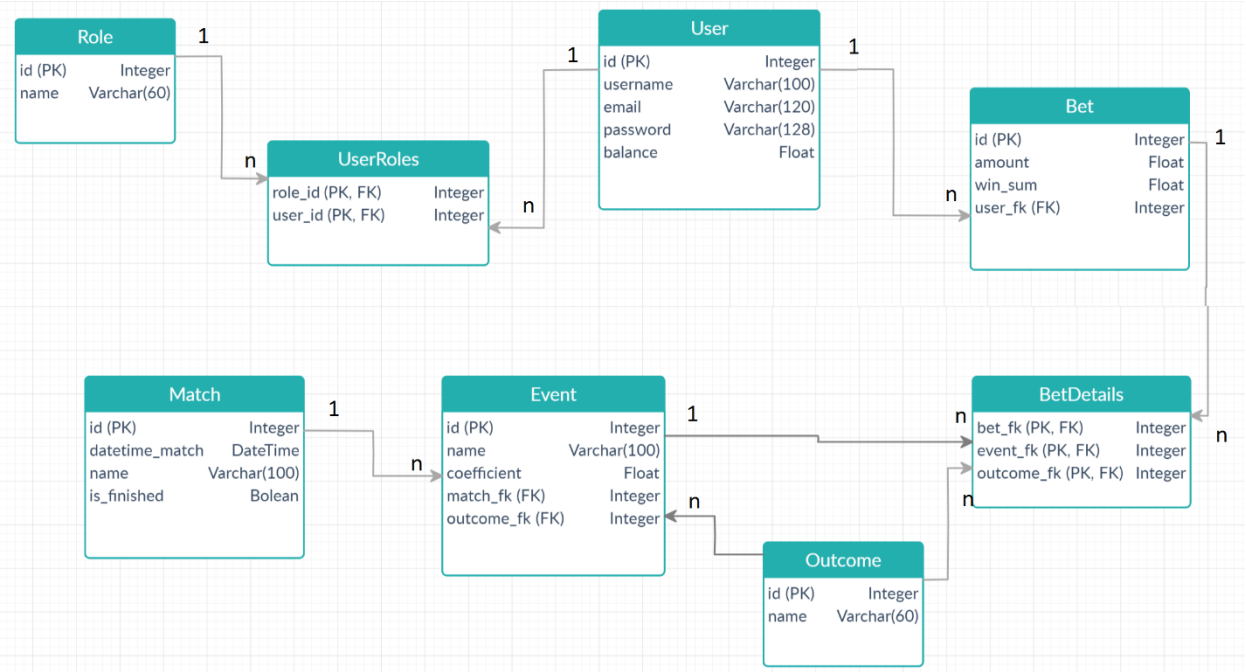

Рисунок 3.2 – Схема бази даних

3.6.2. Опис інтерфейсу програми

Коли неавторизованих користувач заходить на сайт, він потрапляє на сторінку зі всіма матчами, на які можна робити ставки.

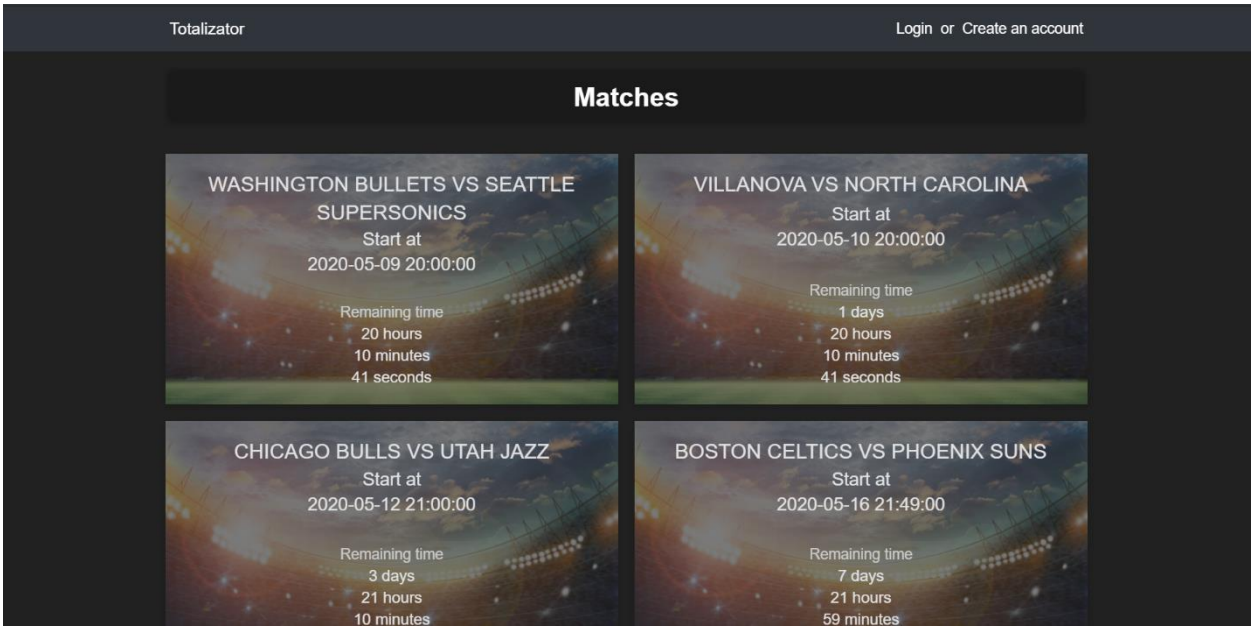

Рисунок 3.3 – Скріншот головної сторінки

Після авторизації користувач може переглядати деталі матчу. На рисунку 3.4) зображено сторінку оформлення ставки з такими позначеннями:

1 – Логотип, який при натисканні перенаправляє на сторінку з матчами

2 – Баланс користувача

3 – Випадаючий список з пунктами для перегляду попередніх ставок і виходу

4 – Список подій в матчі

5 – Поле для вводу суми ставки і кнопка для завершення оформлення ставки

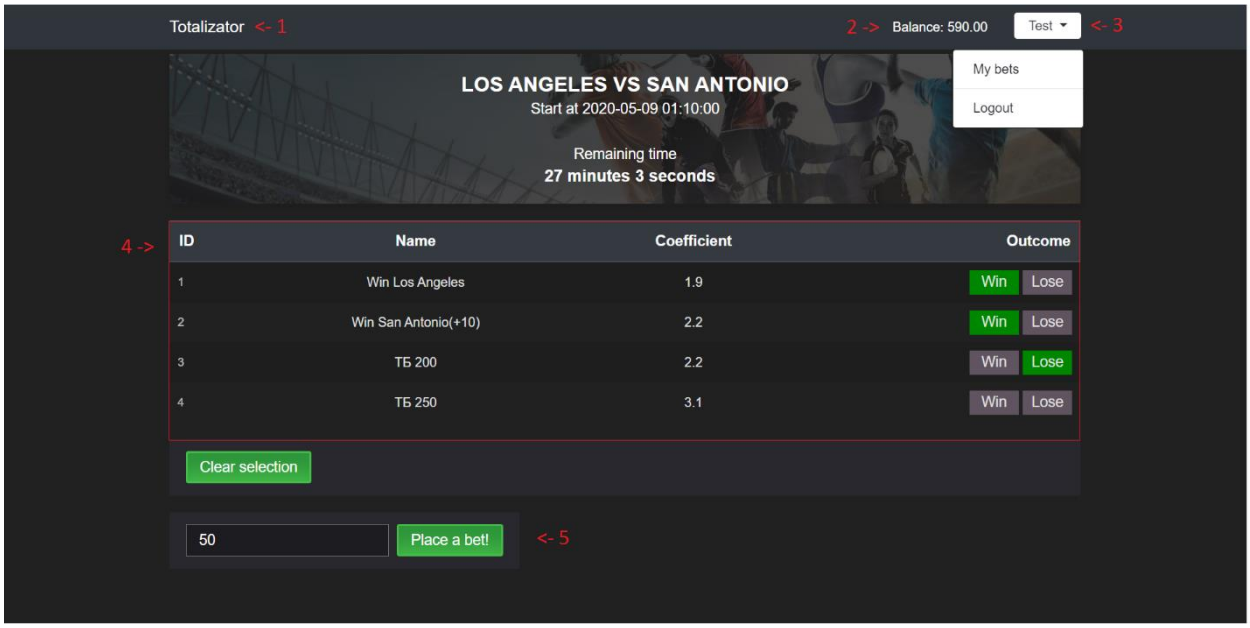

Рисунок 3.4 – Скріншот сторінки перегляду матчу

3.7. Тестування програми і результати її виконання

При спробі перейти на якийсь матч, користувача перейде на сторінку авторизації, де йому буде запропоновано або ввести свої данні, або перейти на сторінку реєстрації.

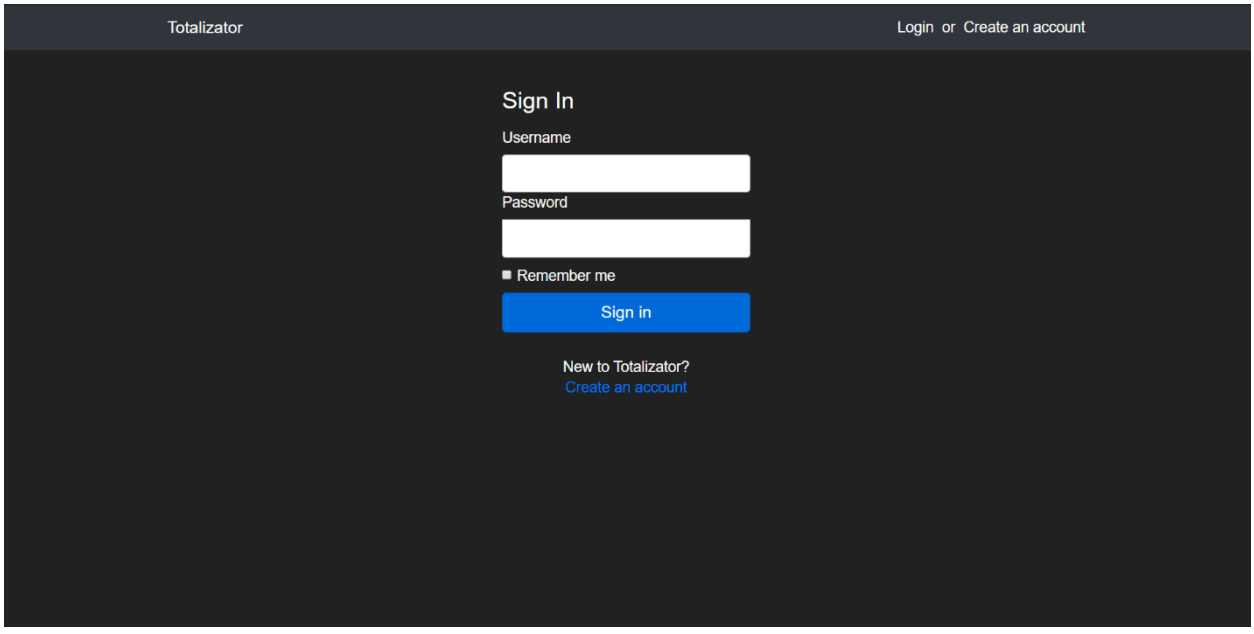

Рисунок 3.5 – Скріншот сторінки авторизації

При введенні невірних даних буде виведене повідомлення про помилку.

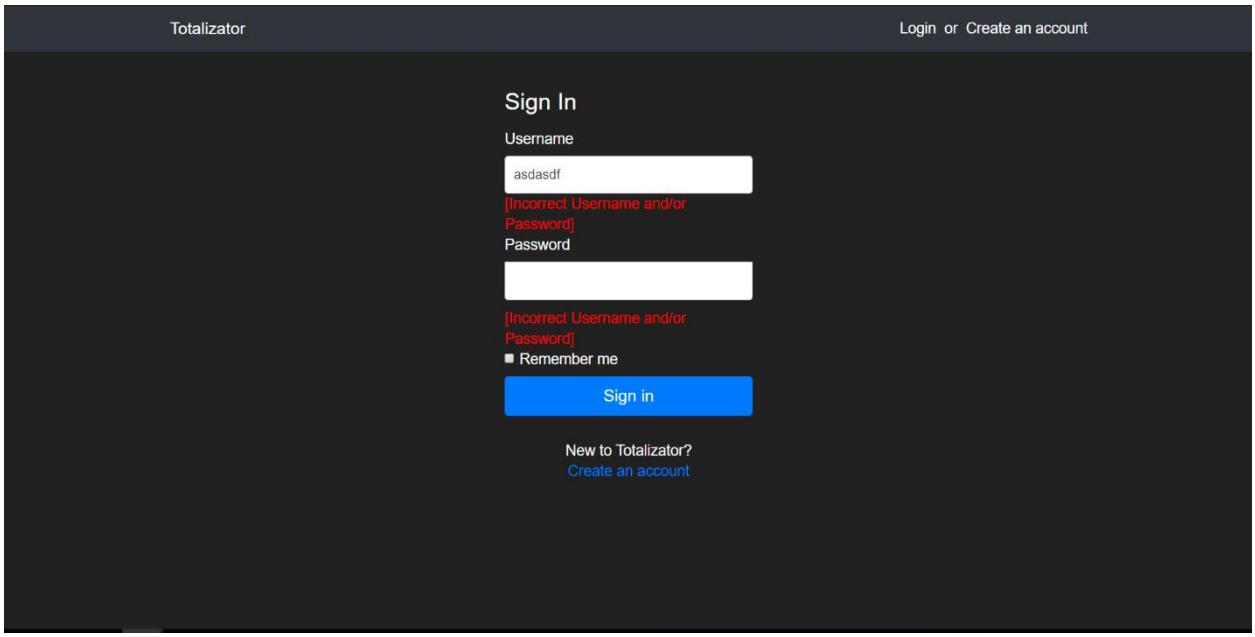

Рисунок 3.6 – Скріншот введення невірних даних

Якщо користувач раніше не був зареєстрований він може перейти на сторінку реєстрації та заповнити необхідні дані.

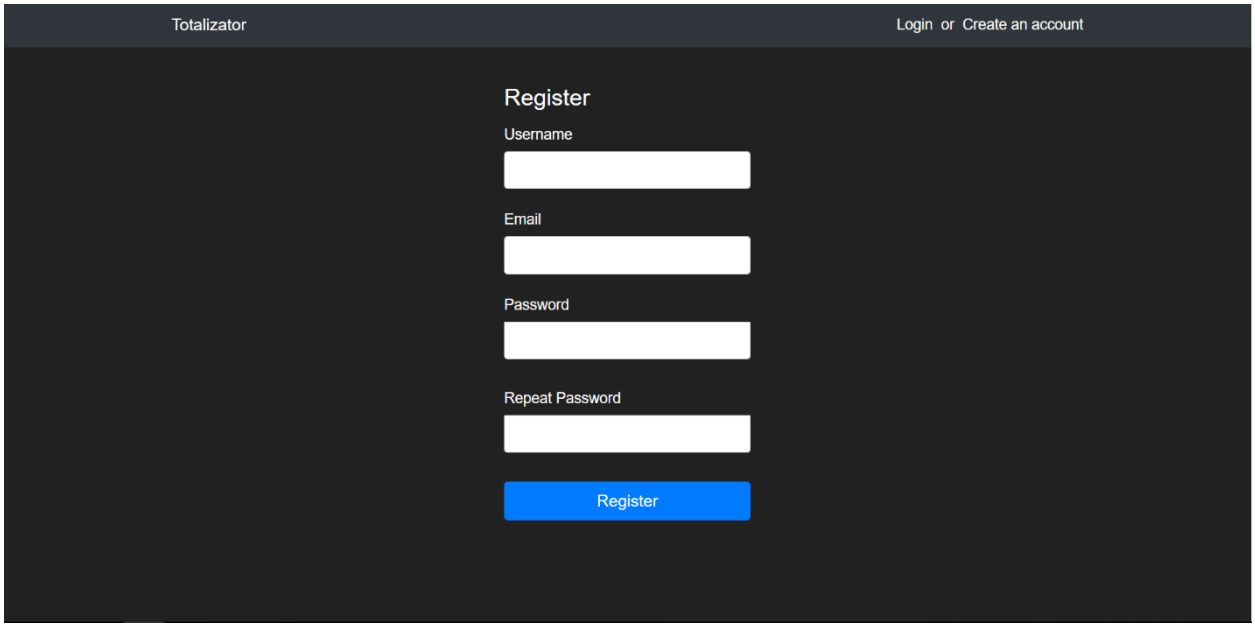

Рисунок 3.7 – Скріншот сторінки реєстрації

Основне завдання адміністратора – це створення матчів і коректне заповнення результатів подій. Після заповнення всіх результатів і натискання кнопки відправляється запит на сервер, який змінює статус матчу, додає результати і обраховує виграші.

Якщо при авторизації ввести дані адміна, то ви перейдете на сторінку зі всіма матчами, а також формою для створення нового матчу.

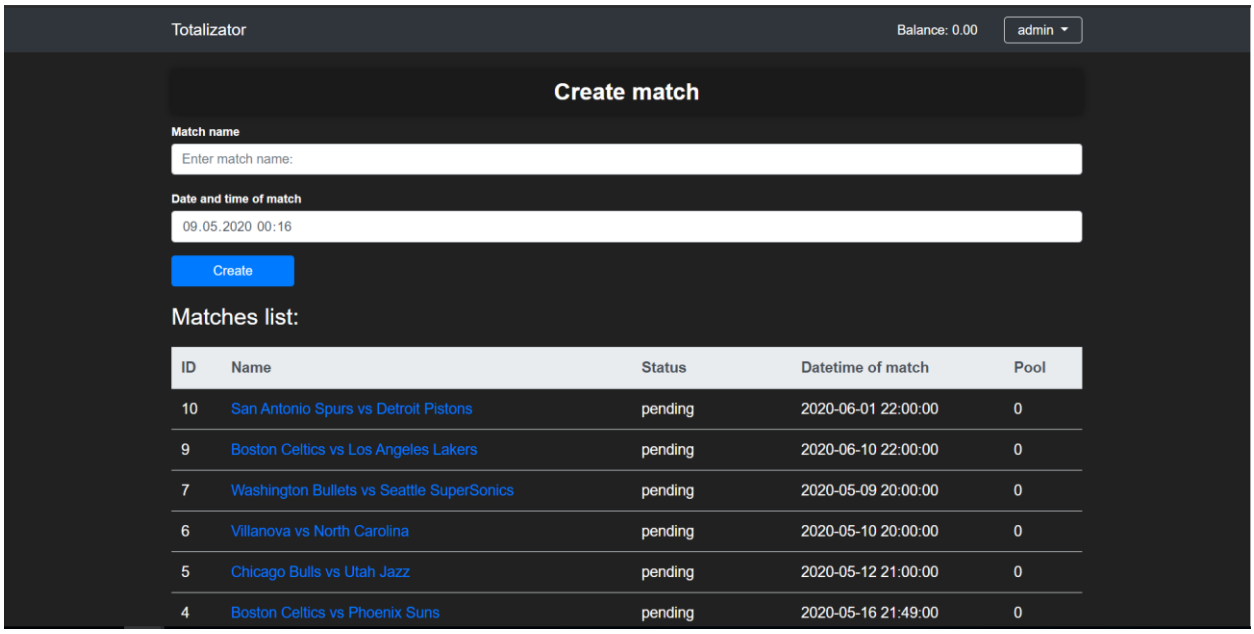

Рисунок 3.8 – Скріншот сторінки перегляду і додавання матчів адміністратором

Далі адміністратор може перейти на сторінку матчу і додати події з певним коефіцієнтом.

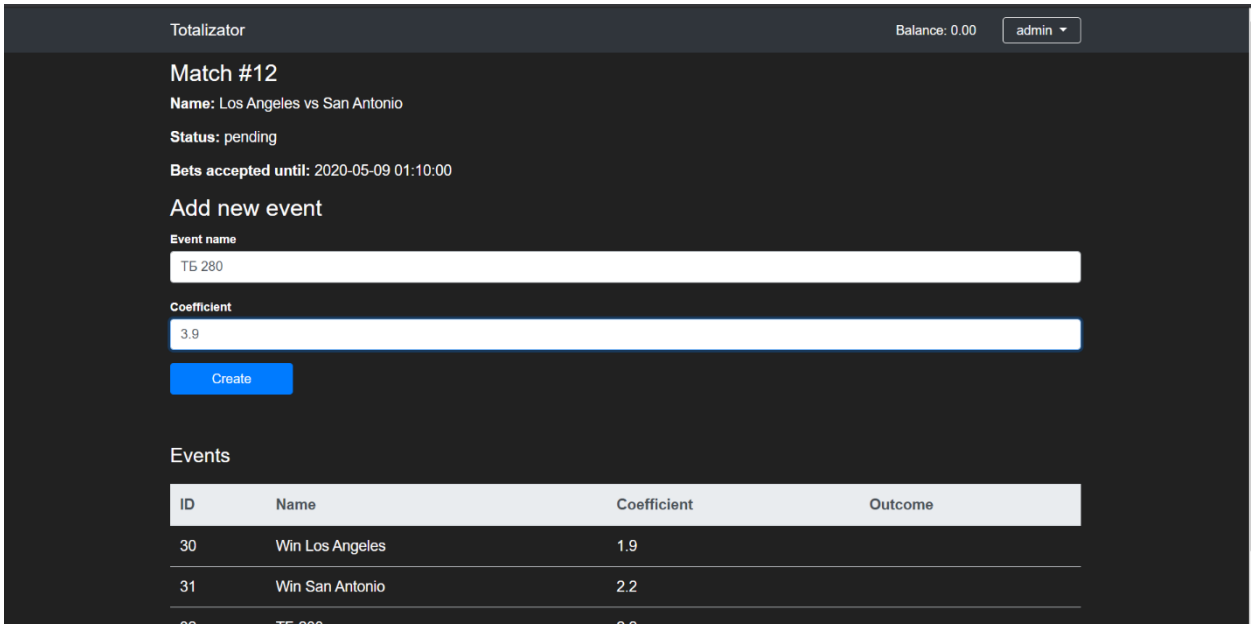

Рисунок 3.9 – Скріншот сторінки перегляду і додавання подій до матчу

Після завершення гри (статус матчу “waiting_results), адміністратор заповнює результати подій в матчі.

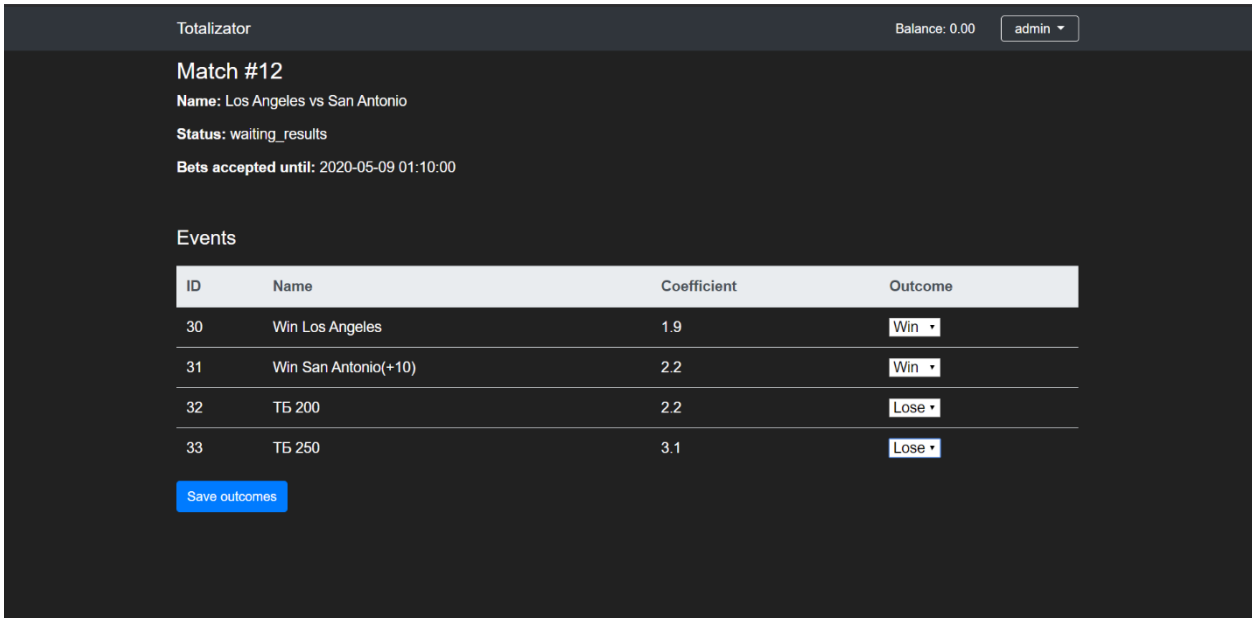

Рисунок 3.10 – Скріншот додавання результатів подій в матчі адміністратором

Основні можливості звичайного користувача – це перегляд стану рахунку, його поповнення, перегляд попередніх ставок користувача і оформлення ставки на події певного матчу.

Якщо зайти на сторінку матчу як звичайний користувач, то можна переглянути всі запропоновані події матчу і зробити на них ставку. Для того щоб зробити ставку треба обрати результат однієї або більше подій, ввести суму ставки і натиснути “Place a bet!”.

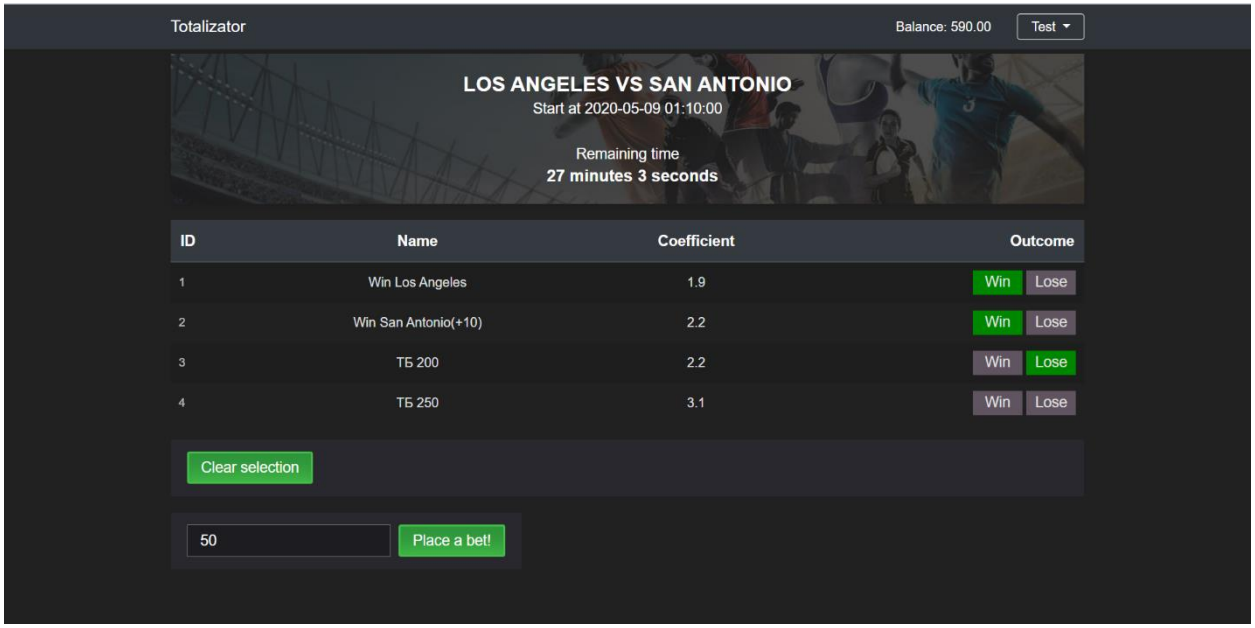

Рисунок 3.11 – Скріншот сторінки оформлення ставки

Також користувач може навести курсор на свій нікнейм і переглянути свої минулі ставки.

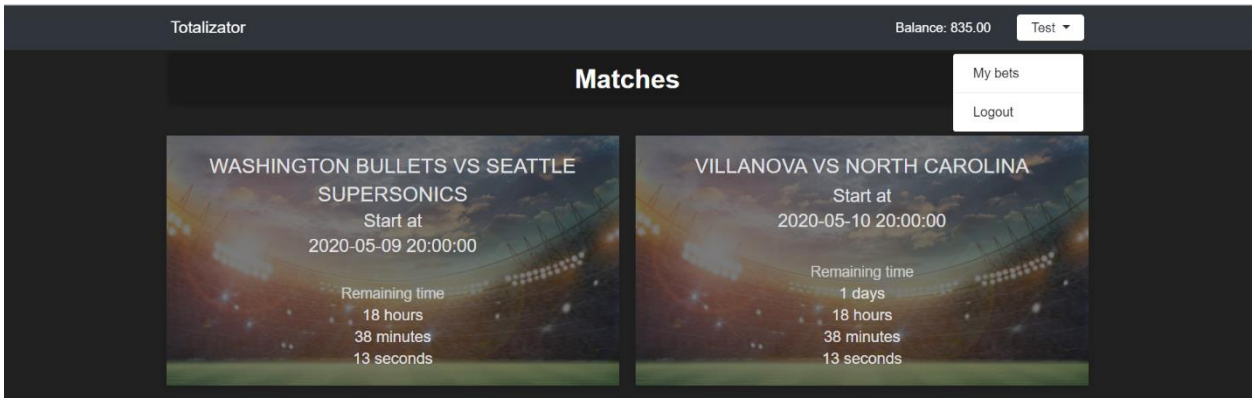

Рисунок 3.12 – Скріншот випадаючого списку

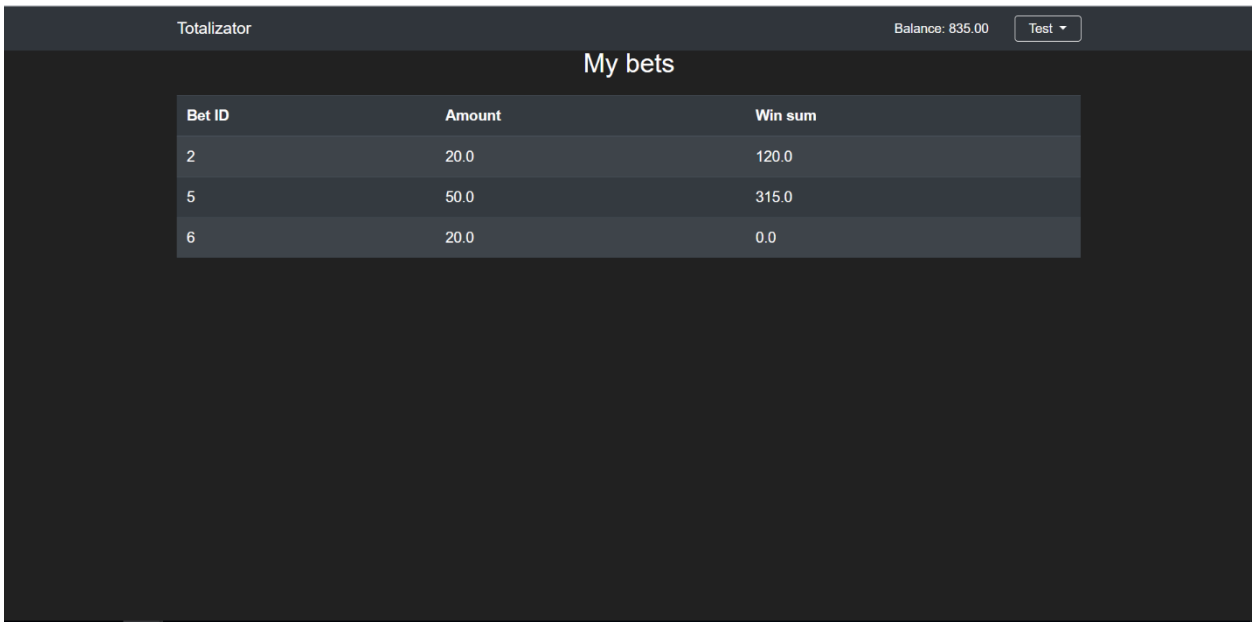

Рисунок 3.13 – Скріншот сторінки перегляду попередніх ставок користувача

Користувач має можливість додати до свого рахунку віртуальні кошти натиснувши на кнопку “Balance”.

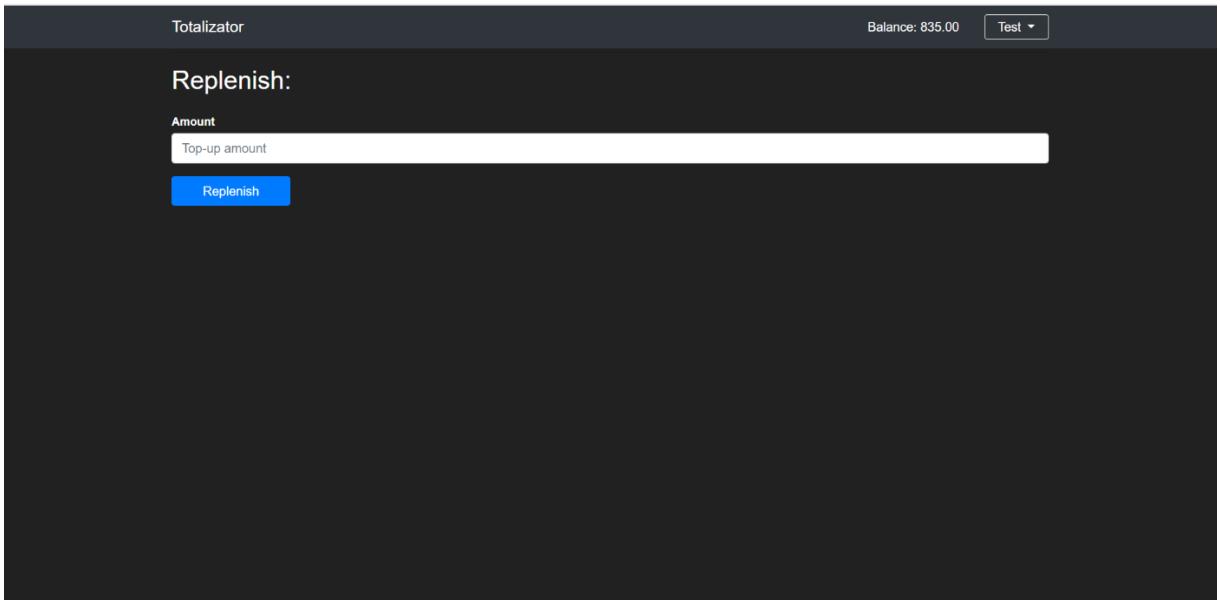

Рисунок 3.14 – Скріншот сторінки додавання коштів користувача

Однією з вимог було створення адаптивного дизайну. На рис 3.15) зображено інтерфейс програми на мобільних пристроях

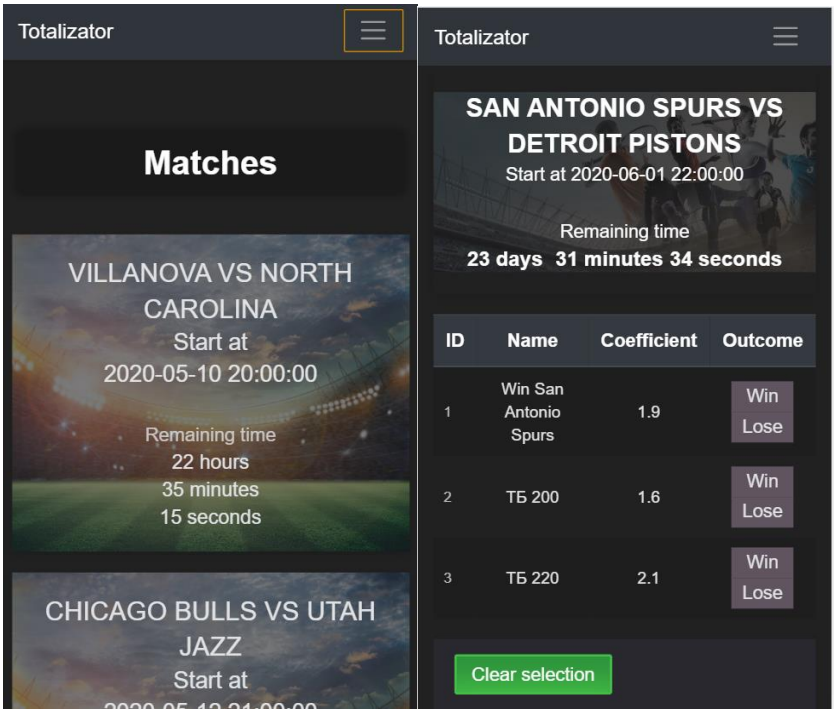

Рисунок 3.15 – Скріншот застосунку на мобільному пристрої

ВИСНОВКИ

В ході виконання курсової роботи було оглянуто існуючі аналоги застосунків на ринку, виокремлено їх недоліки та переваги; проаналізовано принципи створення веб-застосувань, переваги та недоліки різних їх типів; досліджено такі технології для розробки веб-застосунку як: клієнт-серверна технологія, реляційні бази даних, міграції баз даних, об’єктно-реляційне відображення; розроблено продукт, який відповідає всім поставленим вимогам і описаним в роботі технологіям.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. [Електронний ресурс] Академічний тлумачний словник української мови - http://sum.in.ua/s/totalizator
2. [Електронний ресурс] bet365 Online Sport Betting - https://www.bet365.com/
3. [Електронний ресурс] bet365.com Competitive Analysis, Marketing Mix, and Website Traffic - https://www.alexa.com/siteinfo/bet365.com
4. [Електронний ресурс] MELbet Betting Company - https://melbet.com/
5. [Електронний ресурс] melbet.com Competitive Analysis, Marketing Mix and Website Traffic - https://www.alexa.com/siteinfo/melbet.com
6. [Електронний ресурс] Sports Betting Guide - https://www.gamblingsites.com/sports-betting/introduction/bets-wagers/
7. [Електронний ресурс] Інфографіка — Internetdevels офіційний блог - https://internetdevels.ua/blog/most-common-types-of-websites
8. [Електронний ресурс] Client-server architecture | computer science | Britannica - https://www.britannica.com/technology/client-server-architecture
9. [Електронний ресурс] Glossary – Python 3.8.3rc1 documentation - https://docs.python.org/3/glossary.html
10. [Електронний ресурс] Flask documentation - https://flask.palletsprojects.com/en/1.1.x/
11. [Електронний ресурс] PyCharm: the Python IDE - https://www.jetbrains.com/pycharm/
12. [Електронний ресурс] Jinja Documentation - https://jinja.palletsprojects.com/en/2.11.x/
13. [Електронний ресурс] Flask WTF 0.14 - https://flask-wtf.readthedocs.io/en/stable/
14. [Електронний ресурс] PostgreSQL Documentation - https://www.postgresql.org/docs/

15. [Електронний ресурс] Flask-SQLAlchemy Documentation - https://flask-sqlalchemy.palletsprojects.com/en/2.x/
16. [Електронний ресурс] Flask-Migrate Documentation - https://flask-migrate.readthedocs.io/en/latest/
17. [Електронний ресурс] PostgreSQL database adapter for Python - https://www.psycopg.org/docs/
18. [Електронний ресурс] Flask User documentation - https://flask-user.readthedocs.io/en/latest/
19. [Електронний ресурс] Guide to Horse Racing - https://www.racingpost.com/guide-to-racing/types-of-bets/
20. [Електронний ресурс] Best Betting Sites 2020 - https://bookiesbonuses.com/
21. [Електронний ресурс] How Online Gambling Works - https://entertainment.howstuffworks.com/online-gambling.htm

Додаток А. Код серверної частини

config.py – файл конфігурації застосунку

```
import os
from dotenv import load_dotenv

load_dotenv()


class Config(object):
    DEBUG = False
    CSRF_ENABLED = True
    PORT = 8000
    HOST = "localhost"
    SECRET_KEY = 'course-work-totalizator-secret'

    DB_NAME = os.getenv('DB_NAME')
    DB_PORT = os.getenv('DB_PORT')
    DB_HOST = os.getenv('DB_HOST')
    DB_PASS = os.getenv('DB_PASS')
    DB_USER = os.getenv('DB_USER')

    SQLALCHEMY_DATABASE_URI =
f"postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
    SQLALCHEMY_TRACK_MODIFICATIONS = True

    USER_ENABLE_EMAIL = False
    USER_ENABLE_USERNAME = True

    USER_PASSLIB_CRYPTCONTEXT_SCHEMES = ["sha256_crypt"]

    USER_ENABLE_CONFIRM_EMAIL = False

    USER_LOGIN_TEMPLATE = 'authentication/login.html'
    USER_REGISTER_TEMPLATE = 'authentication/register.html'


class ProductionConfig(Config):
    PORT = 80
    HOST = "0.0.0.0"
    DEBUG = False


class DevelopmentConfig(Config):
    DEVELOPMENT = True
    DEBUG = True


config_object = DevelopmentConfig()
```

app.py – файл запуску застосунку

```
from app import app

if __name__ == '__main__':
    print(app)
    app.run(host=app.config['HOST'], port=app.config['PORT'])
```

__init__.py – файл ініціалізації застосунку

```
from flask import Flask
from config import config_object
from flask_migrate import Migrate
from flask_user import UserManager
from app.database.models import User
from app.controllers import index_blueprint, authentication_blueprint,
game_blueprint, admin_blueprint

# init flask app
app = Flask(__name__)
app.config.from_object(config_object)

# init database and migrations
from app.database import db
db.init_app(app)
db.app = app
migrate = Migrate(app, db)
user_manager = UserManager(app, db, UserClass=User)

from app.database.db_queries import db_queries
db_queries.init_db(db, user_manager)

# register blueprints
app.register_blueprint(index_blueprint)
app.register_blueprint(authentication_blueprint)
app.register_blueprint(game_blueprint)
app.register_blueprint(admin_blueprint)
from app.database import models
```

game_controller.py – файл, які містить роутінги, пов’язані з діями звичайного користувача

```
from flask import Blueprint, render_template, request, jsonify, redirect, abort
from flask_user import login_required, current_user

from app.exceptions.game_exceptions import PlaceBetException
from app.database.db_queries import db_queries
from datetime import datetime

game_blueprint = Blueprint('game', __name__)


@game_blueprint.route('/play')
def play():
    try:
        if current_user.has_roles('admin'):
            return redirect('/admin')
    except AttributeError:
        pass
    matches = db_queries.get_pending_matches()
    return render_template('pages/games/games.html', matches=matches)


@game_blueprint.route('/play/<match_id>')
@login_required
def match_play(match_id):
```

```
    match = db_queries.get_match_by_id(match_id)
    datetime_now = datetime.now()
    if match.datetime_match < datetime_now:
        return abort(404)
    possible_outcomes = db_queries.get_all_possible_outcomes()
    return render_template('pages/games/game_placebet.html', match=match,
possible_outcomes=possible_outcomes)


@game_blueprint.route('/play/placebet', methods=['POST'])
@login_required
def place_bet():
    json = request.get_json()
    amount = json['info']['amount']
    if amount == '':
        raise PlaceBetException(message="Provide bet amount data")

    events_data = json['events']
    db_queries.place_bet(amount=amount, events_data=events_data, user=current_user)
    response = jsonify({"message": "Bet placed successfully"})
    return response


@game_blueprint.errorhandler(PlaceBetException)
def handle_invalid_usage(error):
    response = jsonify(error.to_dict())
    response.status_code = error.status_code
    return response
```

authentication_controller.py – файл, які містить роутінги, пов’язані з процесом авторизації і перегляду інформації користувача

```
from flask import Blueprint, render_template, flash, redirect, url_for
from flask_user import current_user, login_required
from flask_login import login_user, logout_user
from app.forms.UserLoginForm import UserLoginForm, UserRegistrationForm,
UserBalanceReplenish

from app.database.db_queries import db_queries

authentication_blueprint = Blueprint('authentication', __name__)


@authentication_blueprint.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect('/')
    form = UserLoginForm()
    if form.validate_on_submit():
        user = db_queries.get_user_by_username(form.username.data)
        if user is None or not user.check_password(form.password.data):
            flash('Invalid username or password')
            return redirect('/login')
        login_user(user, remember=form.remember_me.data)
        if len(user.roles) != 0 and user.roles[0].name == "admin":
            return redirect('/admin')
        return redirect('/')

    return render_template('authentication/login.html', title='Sign In', form=form)
```

```
        @authentication_blueprint.route('/logout')
def logout():
    logout_user()
    return redirect('/')

@authentication_blueprint.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect('/')
    form = UserRegistrationForm()
    if form.validate_on_submit():
        db_queries.create_user(username=form.username.data, email=form.email.data,
password=form.password.data)
        flash('Congratulation, you are registered')
        return redirect('/login')
    return render_template('authentication/register.html', title='Register',
form=form)

@authentication_blueprint.route('/user/bets')
@login_required
def bets():
    bets = current_user.bets
    return render_template('user/bets.html', bets=bets)


@authentication_blueprint.route('/user/balance', methods=['GET', 'POST'])
@login_required
def balance():
    form = UserBalanceReplenish()

    if form.validate_on_submit():
        db_queries.update_user_balance(current_user, form.amount.data)
        flash(f'Your balance replenished (+{form.amount.data})', 'success')
    return render_template('user/balance.html', form=form)
```

admin_controller.py – файл, які містить роутінги пов’язані з діями адміністратора

```
from flask_user import roles_required
from flask import Blueprint, render_template, flash, request, jsonify, redirect

from app.forms.AdminForm import *
from app.database.db_queries import db_queries
from app.exceptions.game_exceptions import PlaceBetException

admin_blueprint = Blueprint('admin', __name__)


@admin_blueprint.route('/admin')
@roles_required('admin')
def main_page():
    return redirect("/admin/matches")


@admin_blueprint.route('/admin/matches', methods=['GET', 'POST'])
@roles_required('admin')
def matches():
```

```
    form = AdminCreateMatchForm()
    if form.validate_on_submit():
        try:
            match = db_queries.create_match(match_name=form.name.data,
datetime_match=form.date.data)
            flash(f"Match #{match.id} created successfully", "success")
        except PlaceBetException:
            flash(f"This match is already exists", "error")
    matches_list = db_queries.get_all_matches()
    return render_template('pages/adminka/admin_bets.html', form=form,
matches=matches_list)


@admin_blueprint.route('/admin/matches/<match_id>', methods=['GET', 'POST'])
@roles_required('admin')
def match_edit(match_id):
    form = AdminCreateEventForm()
    match = db_queries.get_match_by_id(match_id)
    possible_outcomes = db_queries.get_all_possible_outcomes()
    if form.validate_on_submit():
        event = db_queries.create_event(event_name=form.name.data,
                                        coefficient=form.coefficient.data,
match=match)
        flash(f"Event #{event.id} created successfully", "success")
    return render_template('pages/adminka/admin_bet_edit.html', form=form,
match=match,
                           possible_outcomes=possible_outcomes)


@admin_blueprint.route('/admin/matches/<match_id>/update_outcome', methods=['POST'])
@roles_required('admin')
def update_outcome(match_id):
    print('request = '+str(request.get_json()))
    print('match_id = '+str(match_id))
    outcomes = request.get_json()
    for outcome in outcomes:
        db_queries.update_event_outcome(outcome['event_id'], outcome['outcome_id'])
    match = db_queries.get_event_by_id(outcomes[0]['event_id']).match
    db_queries.calculate_results(match)
    return jsonify({"message": f"Outcome successfully updated for match
#{match_id}"})
```

models.py – файл з всіма моделями бази даних

```
from app.database import db
from datetime import datetime
from functools import reduce
from passlib.hash import sha256_crypt

from sqlalchemy.ext.hybrid import hybrid_property
from flask_user import UserMixin

class Event(db.Model):
    __tablename__ = 'events'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(100), nullable=False)
    coefficient = db.Column(db.Float, nullable=False)
```

```
    match_fk = db.Column(db.Integer, db.ForeignKey('matches.id', ondelete='CASCADE',
onupdate='CASCADE'), nullable=False)
    outcome_fk = db.Column(db.Integer, db.ForeignKey('possible_outcomes.id',
ondelete='CASCADE', onupdate='CASCADE'))

    outcome = db.relationship('Outcome')
    match = db.relationship('Match', back_populates="events")
    bets = db.relationship('Bet', secondary='bet_details')


class Match(db.Model):
    __tablename__ = 'matches'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(150), nullable=False)
    datetime_match = db.Column(db.DateTime, nullable=True)
    is_finished = db.Column(db.Boolean, nullable=False, default=False)
    events = db.relationship('Event', back_populates='match', order_by='Event.id')

    @hybrid_property
    def match_status(self):
        if self.is_finished:
            return "finished"

        if self.datetime_match < datetime.now():
                return "waiting_results"
        else:
            return "pending"

    @hybrid_property
    def all_bets(self):
        bets = set()
        if len(self.events) > 0:
            for event in self.events:
                for b in event.bets:
                    bets.add(b)
        return bets


class Outcome(db.Model):
    __tablename__ = 'possible_outcomes'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(50), nullable=False, unique=True)

    def __repr__(self):
        return f'<Outcome {self.name}'

class Bet(db.Model):
    __tablename__ = 'bets'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    amount = db.Column(db.Float, nullable=False)
    win_sum = db.Column(db.Float, nullable=True)
    user_fk = db.Column(db.Integer, db.ForeignKey('users.id', ondelete='CASCADE',
onupdate='CASCADE'), nullable=False)

    user = db.relationship('User', back_populates="bets")
    bet_details = db.relationship('BetDetails')

    @hybrid_property
```

```
    def all_events(self):
        events = set()
        for bd in self.bet_details:
            events.add(bd.event)
        return events


class BetDetails(db.Model):
    __tablename__ = 'bet_details'

    bet_fk = db.Column(db.Integer, db.ForeignKey('bets.id', ondelete='CASCADE',
onupdate='CASCADE'),
                       nullable=False, primary_key=True)
    event_fk = db.Column(db.Integer, db.ForeignKey('events.id', ondelete='CASCADE',
onupdate='CASCADE'),
                         nullable=False, primary_key=True)
    outcome_fk = db.Column(db.Integer, db.ForeignKey('possible_outcomes.id',
ondelete='CASCADE', onupdate='CASCADE'),
                           nullable=False, primary_key=True)

    outcome = db.relationship('Outcome')
    event = db.relationship('Event')
```

db_queries.py – файл, який містить запити до бази даних

```
from flask_user import UserManager
from flask_sqlalchemy import SQLAlchemy

from app.database.models import *
from app.exceptions.game_exceptions import PlaceBetException


class DatabaseQueries:
    def __init__(self):
        self.db: SQLAlchemy = None
        self.user_manager: UserManager = None

    def init_db(self, db_sqlalchemy: SQLAlchemy, user_manager: UserManager):
        self.db = db_sqlalchemy
        self.user_manager = user_manager

    def create_user(self, username, password, email, balance=0,  is_admin=False):
        user = User(username=username, balance=balance, email=email)
        user.set_password(password)
        if is_admin:
            user.roles.append(self._get_or_create(Role, name='admin'))
        self.db.session.add(user)
        self.db.session.commit()
        return user

    def _get_or_create(self, model, **kwargs):
        instance = self.db.session.query(model).filter_by(**kwargs).first()
        if instance:
            return instance
        else:
            instance = model(**kwargs)
            self.db.session.add(instance)
```

```
            self.db.session.commit()
            return instance

    def update_user_balance(self, user, diff):
        user.balance = user.balance + diff
        self.db.session.commit()

    def create_match(self, match_name, datetime_match):

        query = db.session.query(Match).filter(Match.datetime_match == datetime_match
and Match.name == match_name)
        count_q = query.statement.with_only_columns([query.count()]).order_by(None)
        count = query.session.execute(count_q).scalar()

        if count is None or count == 0:
            match = Match(name=match_name, datetime_match=datetime_match)
            self.db.session.add(match)
            self.db.session.commit()
            return match
        else:
            raise PlaceBetException(message="This match is already exists")

    def get_all_matches(self):
        return Match.query.order_by(Match.id.desc()).all()

    def create_event(self, event_name, coefficient, match, outcome=None):
        event = Event(name=event_name, coefficient=coefficient, match=match,
outcome=outcome)
        self.db.session.add(event)
        self.db.session.commit()
        return event

    def place_bet(self, amount: int, events_data, user: User):

        if user.balance < float(amount):
            raise PlaceBetException(message="Not enough funds")

        bet = Bet(amount=amount, user=user)
        self.db.session.add(bet)
        self.db.session.commit()

        for bet_info in events_data:
            self.db.session.add(
                BetDetails(bet_fk=bet.id, outcome_fk=bet_info['value'],
event_fk=bet_info['name']))

        user.balance -= float(amount)
        self.db.session.commit()

    def update_event_outcome(self, event_id, outcome_id):
        event = self.get_event_by_id(event_id)
        event.outcome_fk = outcome_id
        self.db.session.commit()

    def _check_match_status(self, match):
        if match.match_status != "waiting_results":
            return False
        for event in match.events:
            if event.outcome is None:
                return False
        return True
```

```
    def calculate_results(self, match):

        if self._check_match_status(match):
            bets_of_match = match.all_bets
            for bet in bets_of_match:
                win_sum = bet.amount*self._get_coefficient_all_event_of_bet(bet)
                bet.win_sum = win_sum
                bet.user.balance += win_sum
            match.is_finished = True
            self.db.session.commit()

    def _get_coefficient_all_event_of_bet(self, bet:Bet):
        total_coeff = 0

        for bet_details in bet.bet_details:
            event = self.get_event_by_id(bet_details.event_fk)
            if int(event.outcome_fk) == bet_details.outcome_fk:
                total_coeff += event.coefficient
            else:
                total_coeff = 0
                break
        return total_coeff

    def get_match_by_id(self, match_id):
        return Match.query.get(match_id)

    def get_event_by_id(self, event_id) -> Event:
        return Event.query.get(event_id)

    def get_all_possible_outcomes(self):
        return Outcome.query.order_by(Outcome.id).all()

    def get_user_by_username(self, username):
        return User.query.filter_by(username=username).first()

    def get_pending_matches(self):
        datetime_now = datetime.now()
        return Match.query.filter(Match.datetime_match >
datetime_now).order_by(Match.datetime_match)

db_queries = DatabaseQueries()
```

739dd77f14e1_project_database_ver1_1.py – файл міграції бази даних

```
"""Project database ver1.1

Revision ID: 739dd77f14e1
Revises:
Create Date: 2020-05-09 21:39:38.491414

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = '739dd77f14e1'
down_revision = None
```

```
branch_labels = None
depends_on = None


def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('matches',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('name', sa.String(length=100), nullable=False),
    sa.Column('datetime_match', sa.DateTime(), nullable=True),
    sa.Column('is_finished', sa.Boolean(), nullable=False),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_table('possible_outcomes',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('name', sa.String(length=30), nullable=False),
    sa.PrimaryKeyConstraint('id'),
    sa.UniqueConstraint('name')
    )
    op.create_table('roles',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('name', sa.String(length=60), nullable=False),
    sa.PrimaryKeyConstraint('id'),
    sa.UniqueConstraint('name')
    )
    op.create_table('users',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('username', sa.String(length=64), nullable=True),
    sa.Column('email', sa.String(length=120), nullable=True),
    sa.Column('password', sa.String(length=128), server_default='', nullable=True),
    sa.Column('balance', sa.Float(), nullable=False),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_users_email'), 'users', ['email'], unique=True)
    op.create_index(op.f('ix_users_username'), 'users', ['username'], unique=True)
    op.create_table('bets',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('amount', sa.Float(), nullable=False),
    sa.Column('win_sum', sa.Float(), nullable=True),
    sa.Column('user_fk', sa.Integer(), nullable=False),
    sa.ForeignKeyConstraint(['user_fk'], ['users.id'], onupdate='CASCADE',
ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_table('events',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('name', sa.String(length=150), nullable=False),
    sa.Column('coefficient', sa.Float(), nullable=False),
    sa.Column('match_fk', sa.Integer(), nullable=False),
    sa.Column('outcome_fk', sa.Integer(), nullable=True),
    sa.ForeignKeyConstraint(['match_fk'], ['matches.id'], onupdate='CASCADE',
ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['outcome_fk'], ['possible_outcomes.id'],
onupdate='CASCADE', ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_table('user_roles',
    sa.Column('user_id', sa.Integer(), nullable=False),
    sa.Column('role_id', sa.Integer(), nullable=False),
    sa.ForeignKeyConstraint(['role_id'], ['roles.id'], onupdate='CASCADE',
ondelete='CASCADE'),
```

```
    sa.ForeignKeyConstraint(['user_id'], ['users.id'], onupdate='CASCADE',
ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('user_id', 'role_id')
    )
    op.create_table('bet_details',
    sa.Column('bet_fk', sa.Integer(), nullable=False),
    sa.Column('event_fk', sa.Integer(), nullable=False),
    sa.Column('outcome_fk', sa.Integer(), nullable=False),
    sa.ForeignKeyConstraint(['bet_fk'], ['bets.id'], onupdate='CASCADE',
ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['event_fk'], ['events.id'], onupdate='CASCADE',
ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['outcome_fk'], ['possible_outcomes.id'],
onupdate='CASCADE', ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('bet_fk', 'event_fk', 'outcome_fk')
    )
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('bet_details')
    op.drop_table('user_roles')
    op.drop_table('events')
    op.drop_table('bets')
    op.drop_index(op.f('ix_users_username'), table_name='users')
    op.drop_index(op.f('ix_users_email'), table_name='users')
    op.drop_table('users')
    op.drop_table('roles')
    op.drop_table('possible_outcomes')
    op.drop_table('matches')
    # ### end Alembic commands ###
```

Додаток Б. Код клієнтської частини

login.html – шаблон сторінки авторизації

```
<!-- extend base layout -->
{% extends 'common/layout.html' %}
{% block styles %}
    {{ super() }}
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='css/login.css') }}">
{% endblock %}
{% block content %}
    <hr>
    <div class="container" style="padding-top: 50px">


        <form action="" method="post" class="form-signin" novalidate>
            {{ form.hidden_tag() }}
        <h1 class="h3 mb-3 font-weight-normal">Sign In</h1>
            <div id="loginUsername">
                {{ form.username.label }}<br>
                {{ form.username(size=32, class="form-control") }}
                {% for error in form.username.errors %}
                    <span style="color: red;">[{{ error }}]</span>
                {% endfor %}
            </div>
            <div id="loginPassword">
                {{ form.password.label }}
                {{ form.password(size=32, class="form-control") }}
                {% for error in form.password.errors %}
                    <span style="color: red;">[{{ error }}]</span>
                {% endfor %}
            </div>
            <div>{{ form.remember_me() }} {{ form.remember_me.label }}</div>
            <div>{{ form.submit(class_="btn btn-lg btn-primary btn-block login-
button") }}</div>
        </form>
    <p class="form-signin form-new-user">New to Totalizator?<br> <a
href="/register">Create an account</a></p>
    </div>
{% endblock %}
```

game_placebet.html – шаблон сторінки оформлення ставки

```
{% extends "common/layout.html" %}

{% block styles %}
    {{ super() }}
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='css/game-placebet.css') }}">
{% endblock %}

{% block content %}
    <section class="placebet">
        <div class="container">
            <div class="match-info-container match-info-wrapper">
                <div class="match-info">
                    <div class="match-name">
                        <div class="match-name-main">{{ match.name }}</div>
                    </div>
```

```
                    <div class="match-date">
                        <div>Start at {{ match.datetime_match }}</div>
                    </div>
                    {% include "/pages/games/components/countdown.html" %}
                </div>
            </div>
            <div>
                <form id="form-events">
                    <table class="table table-events">
                        <thead class="thead-dark">
                        <tr>
                            <th class="text-left">
                                ID
                            </th>
                            <th>
                                Name
                            </th>
                            <th>
                                Coefficient
                            </th>
                            <th class="text-right">
                                Outcome
                            </th>
                        </tr>
                        </thead>
                        {% for event in match.events %}
                            <tr class="event-wrapper">
                                <td class="event-index text-left">
                                    {{ loop.index }}
                                </td>
                                <td class="event-name text-center">
                                    {{ event.name }}
                                </td>
                                <td class="event-name text-center">
                                    {{ event.coefficient }}
                                </td>
                                <td class="event-outcomes text-right">
                                    <div class="text-center" style="display: inline-
block">
                                        {% for outcome in possible_outcomes %}
                                            <label class="outcome-label">
                                                <input type="radio" name="{{ event.id
}}"
                                                       value="{{ outcome.id }}">
                                                <span class="outcome-stylized">{{
outcome.name }}</span>
                                            </label>
                                        {% endfor %}
                                    </div>
                                </td>
                            </tr>
                        {% endfor %}
                    </table>
                </form>
                <div class="match-actions-wrapper">
                    <button class="btn-custom js-parlay-clear">Clear
selection</button>
                </div>
            </div>
            <div class="bet-info">
                <form class="bet-wrapper" id="form-placebet" action="/play/{{
```

```
match.id }}/placebet">
                    <input type="number" min="10" required class="input-placebet"
placeholder="Amount"
                                                name="amount">
                    <button type="submit" class="btn-custom green">Place a
bet!</button>
                </form>
            </div>
        </div>
    </section>
{% endblock %}

{% block scripts %}
    {{ super() }}
    <script src="{{ url_for('static', filename='js/placebet.js') }}"></script>
{% endblock %}
```

bets.html – шаблон сторінки перегляду попередніх ставок

```
{% extends 'common/layout.html' %}

{% block content %}
    <div class="container" style="padding-top: 50px">
        <h2 class="text-center" style="margin-bottom: 20px">My bets</h2>

        <table class="table table-dark table-striped">
            <thead class="thead-dark">
            <tr>
                <th>
                    Bet ID
                </th>
                <th>
                    Amount
                </th>
                <th>
                    Win sum
                </th>
            </tr>
            </thead>
            {% for bet in bets %}
                <tr>
                    <td>{{ bet.id }}</td>
                    <td>
                        {{ bet.amount }}
                    </td>
                    <td>
                        {{ bet.win_sum }}
                    </td>
                </tr>
            {% endfor %}
        </table>
    </div>
{% endblock %}
```

games.css – файл таблиць стилів для сторінки перегляду всіх матчів

```
.match-column {
```

```
  padding: 10px;
}
.matches-row {
  margin-top: 30px;
}

.match-countdown {
  margin-top: 30px;
}
.match-countdown-wrapper {

  font-size: 19px;
  font-weight: 300;
}
.match-countdown-text {
  opacity: 0.8;
}

.match-wrapper {
  position: relative;
  padding: 20px;
  cursor: pointer;
  height: 100%;
  min-height: 300px;
  box-shadow: 0 3px 10px rgba(0, 0, 0, 0.15);
  display: flex;
  justify-content: space-between;
  flex-direction: column;
  text-align: center;
  color: #e2e3e5;
  text-decoration: none;
}
.match-wrapper:before {
  position: absolute;
  content: "";
  background: url("/static/images/bg2.jpg") no-repeat center;
  background-size: 105%;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  opacity: 0.4;
  z-index: -1;
  transition-duration: 0.3s;
}
.match-wrapper:hover {
  box-shadow: none;
  text-decoration: none;
  color: #ffffff;
}
.match-wrapper:hover:before {
  background-size: 115%;
  opacity: 0.6;
}
.match-name {
  font-size: 25px;
  text-transform: uppercase;
  font-weight: 500;
}
.match-date {
```

```
  font-size: 22px;
}
```

admin.js – скрипт, який на сторінці додавання результатів подій матчу адміном. Він відповідає за те, щоб зібрати обрані адміном результати подій і відправити AJAX запит на сервер з цими даними.

```
$(function () {
    $('#save-outcomes-btn').on('click', function (event) {
        event.preventDefault();
        let json_out = $('.event-outcome-select').serializeArray().map(elem => {
            return {'event_id': elem['name'], 'outcome_id': elem['value']}
        });
        console.log();
        $.ajax({
            type: "POST",
            url: window.location.href.split("/").pop()+"/update_outcome",
            data: JSON.stringify(json_out),
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function (data) {
                showMessage(data['message'], "success");
            },
            error: function (errMsg) {
                showMessage(errMsg['responseJSON']['message'], "danger");
            }

        })
    });
});
```

alerts.js – скрипт, який знизу екрану додає повідомлення про певну операцію. Це повідомлення зникає через 10 секунд.

```
function showMessage(message, category = "info") {
    const $alert = $(
        `<div class="alert fade in alert-dismissible show alert-${category}">
            <button type="button" class="close" data-dismiss="alert" aria-
label="Close">
                <span aria-hidden="true" style="font-size:20px">×</span>
             </button>
                ${message}
          </div>`);
    $(".flashes-container").append($alert);

    setTimeout(function () {
        $alert.fadeOut();
        $alert.remove();
    }, 10000);
}
```