Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Багатокористувацький веб-сервіс для гри в шахи
на основі протоколу WebSocket
(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-1
Артем БЄЛІКОВ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Роман ЯРОШЕВИЧ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ ________ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________________
(підпис)
“____”________________ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бєлікову Артему Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Багатокористувацький веб-сервіс для гри в шахи на основі протоколу WebSocket

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) правила гри «Шахи»;
2) документація мови програмування Go;
3) документація мови програмування TypeScript;
4) специфікація протоколу WebSocket;
5) документація системи керування базами даних PostgreSQL;
6) документація бібліотеки React;
7) документація бібліотеки Gorilla WebSocket.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області та постановка завдання;
2) вибір сучасних технологій для розробки багатокористувацького веб-сервісу;
3) розробка програмного забезпечення;
4) тестування та вимірювання продуктивності розроблених компонентів;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій

Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз існуючих рішень та постановка	27.05.25-29.05.25	
	завдання		
2	Вибір технологій розробки веб-сервісу	30.05.25-31.05.25	
3	Проєктування структури бази даних	01.06.25-02.06.25	
4	Розробка та відлагодження програмного коду	03.06.25-07.06.25	
5	Тестування та вимірювання продуктивності	08.06.25-09.06.25	
	роботи веб-сервісу		
6	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
7	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання “ 26 ” травня 20 25 р.

Здобувач ____________________
(підпис)

Керівник роботи ____________________ ст. викл. Роман ЯРОШЕВИЧ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 92 с., 9 рис., 2 дод., 15 джерел.

МОВА ПРОГРАМУВАННЯ GO, МОВА ПРОГРАМУВАННЯ TYPESCRIPT, ПРОТОКОЛ WEBSOCKET, БІБЛІОТЕКА REACT, ОДНОЧАСНІ ШАХИ.

Метою кваліфікаційної роботи є розробка багатокористувацького веб-сервісу для гри в шахи, що реалізує авторизацію та автентифікацію користувачів, надає можливість грати проти інших гравців або вбудованого шахового рушія із різними рівнями складності, переглядати завершені партії та спілкуватися у ігровому чаті.

У ході виконання кваліфікаційної роботи було розроблено серверний та клієнтський додатки з використанням мов програмування Go і TypeScript. Для написання та налагодження коду використовувалось середовище розробки Visual Studio Code. Серверний додаток виконує обробку вхідних HTTP-запитів та забезпечує двосторонній канал зв’язку. Клієнтський додаток надає графічний інтерфейс користувача у браузері.

Методи дослідження – вивчення літератури, проєктування архітектури та алгоритмів програми, написання та налагодження програмного коду.

Перевагою розробленого проєкту є незалежність від клієнтської платформи. Застосування сучасних технологій знизило вимогливість сервісу до апаратних ресурсів. Зручний графічний інтерфейс дозволяє створювати ігрові кімнати, встановлювати параметри гри та взаємодіяти з дошкою. Передбачена можливість перегляду завершених ігор із відновленням результатів, інформації про гравців, зроблених ходів та показань таймерів. Інформація про завершені ігри зберігається у реляційній базі даних.

ABSTRACT

Bachelor’s thesis: 92 pages, 9 figures, 2 appendices, 15 sources.

GO PROGRAMMING LANGUAGE, TYPESCRIPT PROGRAMMING LANGUAGE, WEBSOCKET PROTOCOL, REACT LIBRARY, CONCURRENT CHESS.

The major goal of this thesis is to develop a multi-user web service for playing chess which implements user authorization and authentication, provides the ability to play against other players or a built-in chess engine with various difficulty levels, view completed games, and communicate via in-game chat.

During thesis completion, the server and client applications were developed using the Go and TypeScript programming languages. The Visual Studio Code development environment was used for writing and debugging the code. The server application handles incoming HTTP requests and provides a bidirectional communication channel. The client application provides a graphical user interface in the browser.

Methods of research – literature studying, program architecture and algorithms designing, writing and debugging program code.

A key advantage of the developed project is its independence from the client platform. The use of modern technologies reduced the hardware requirements of the service. The user-friendly graphical interface allows users to create game rooms, set game parameters, and interact with the chessboard. The service provides the ability to view completed games, including restored results, player information, completed moves history, and timer readings. Data about completed games is stored in a relational database.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 8
ВСТУП 9
1 АНАЛІЗ СУЧАСНИХ ВЕБ-СЕРВІСІВ ДЛЯ ГРИ В ШАХИ ТА ПОСТАНОВКА ЗАВДАННЯ 11
1.1 Основні характеристики багатокористувацьких веб-сервісів 11
1.2 Формат JSON як стандарт обміну даними в клієнт-серверних додатках 12
1.3 Протокол WebSocket та його використання 12
1.4 Етапи проєктування веб-сервісу для гри в шахи 14
1.5 Постановка завдання 15
2 ХАРАКТЕРИСТИКИ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ 16
2.1 Мова програмування Go як інструмент розробки серверних додатків 16
2.2 Бібліотека Gorilla WebSocket 17
2.3 Мова програмування TypeScript як інструмент розробки клієнтських додатків 18
2.4 Бібліотека React для створення графічного інтерфейсу 19
2.5 Реляційна база даних PostgreSQL 20
3 ОПИС РОЗРОБЛЕНИХ ПРОГРАМНИХ КОМПОНЕНТІВ 22
3.1 Опис програмного коду серверного додатка 22
3.1.1 Архітектура проєкту 22
3.1.2 Пакет main 23
3.1.3 Пакет auth 25
3.1.4 Пакет chess 28
3.1.5 Пакет db 37
3.1.6 Пакет game 39
3.1.7 Пакет player 40

3.1.8 Пакет ws 41
3.2 Опис програмного коду клієнтського додатка 44
3.2.1 Компоненти графічного інтерфейсу 45
3.2.2 Реалізація клієнт-серверної взаємодії 49
4 ТЕСТУВАННЯ ТА ВИМІРЮВАННЯ ПРОДУКТИВНОСТІ РОБОТИ ВЕБ-СЕРВІСУ 50
4.1 Інструменти автоматизованого тестування мови Go 50
4.2 Результати модульного тестування 51
4.3 Результати вимірювання продуктивності розроблених функцій 53
ВИСНОВКИ 54
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 55
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 57
ДОДАТОК Б Програмний код 64
Б.1 Реалізація автентифікації та авторизації 64
Б.2 Генерація можливих ходів пішака 71
Б.3 Код пакету game 73
Б.4 Код пакету player 76
Б.5 Код пакету ws 79

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

DOM – об’єктна модель документа (англ., Document Object Model)

FIFO – перший зайшов перший вийшов (англ., First In First Out)

HTML – мова розмітки гіпертексту (англ., Hypertext Markup Language)

HTTP – протокол передачі гіпертексту (англ., Hypertext Transfer Protocol)

JSON – запис об’єктів JavaScript (англ., JavaScript Object Notation)

JWT – JSON веб-токен (англ., JSON Web Token)

SQL – структурована мова запитів (англ., Structured Query Language)

SSE – події, що посилаються сервером (англ., Server-Side Events)

TCP – протокол управління передачею (англ., Transmission Control Protocol)

TLS – захист на транспортному рівні (англ., Transport Layer Security)

UML – уніфікована мова моделювання (англ., Unified Modeling Language)

URL – уніфікований локатор ресурсів (англ., Uniform Resource Locator)

ВСТУП

У сучасному світі актуальність шахів підвищилась завдяки розвитку веб-сервісів та мобільних додатків, що надають доступ до гри мільйонам гравців. Переваги шахів роблять їх не просто розвагою, але й гарним інструментом для навчання та особистісного зростання, пропонуючи цінність людям різного віку та походження.

Багато років поспіль найпопулярнішими платформами для гри в шахи постають веб-сервіси Chess.com та Lichess. Вони пропонують широкий функціонал, такий як аналіз партій штучним інтелектом, гру проти шахового рушія та ведення статистики гравців. Платформа Chess.com має закритий програмний код, порівняльно низьку продуктівність та платний функціонал. Платформа Lichess є безкоштовною та має відкритий програмний код, розроблений на мові Scala з дотриманням парадигми функціонального програмування. Однак майже повна відсутність коментарів та документації робить аналіз кодової бази та архітектури додатку складним завданням. Основним недоліком сучасних платформ є складність їх використання у навчальних цілях та в якості основи для власних проєктів.

Веб-сервіси, через їх нелазежність від апаратних платформ та високу популярність стали стандартом для реалізації класичних ігор, що не потребують тривимірної графіки. Гра проти шахового рушія зазвичай реалізується з використанням додатку Stockfish через те, що він має відкритий вихідний код та ефективно виконується у веб-браузері. Шахові партії можуть складатися з великої кількості ходів, тому для їх зберігання використовуються компактні формати запису, зазвичай у вигляді беззнакових цілих чисел. Авторизація користувачів без використання сховища сесій спрощує масштабування додатку та реалізується за допомогою стандарту JWT. Популярною практикою є впровадження гостьового режиму для незареєстрованих користувачів, що надає доступ лише до деяких ресурсів. Це

дозволяє ознайомитись із основним функціоналом сервісу без необхідності введення облікових даних. Обробка зроблених ходів та повідомлень в режимі реального часу реалізується за допомогою протоколу WebSocket, який поширює повідомлення між підключеними клієнтами без перезавантажень сторінки. Адаптивний дизайн забезпечує можливість користування інтерфейсом на будь-якому типі пристрою.

Створення веб-сервісу, що ефективно реалізує алгоритми гри, надає поширений функціонал та має документований код задовольняє потреби як гравців, так і розробників, що бажають навчитися на прикладі додатку з відкритою архітектурою.

Метою кваліфікаційної роботи є опис розробки багатокористувацького веб-сервісу для гри в шахи. Користувач може як створювати власні ігри із обраними параметрами, так і під'єднуватися до вже створених ігор. Після початку гри, натискаючи на будь-яку фігуру на дошці, відображаються доступні для ходу позиції, що робить процес гри більш зручним. Передбачено можливість комунікації за допомогою вбудованого чату. Зроблені ходи записуються до таблиці у стандартній алгебраїчній нотації. Кінець гри відображається у вигляді спливаючого вікна, де вказується результат гри та переможець. Авторизовані користувачі можуть повторно переглянути завершені партії з відновленням зроблених ходів та показників таймерів на сторінці власного профілю.

Завдання кваліфікаційної роботи полягає у створенні функціонуючого веб-сервісу з клієнт-серверною архітектурою. Для одночасної обробки багатьох ігрових сеансів, зберігання активних підключень та реалізації правил гри було створено серверний додаток на мові програмування Go. Графічний інтерфейс користувача було розроблено на мові програмування TypeScript з використанням бібліотеки React. В процесі розробки було побудовано схему дерева елементів графічного інтерфейсу. Використане середовище: Microsoft Visual Studio Code. Роботу сервісу протестовано у сучасних браузерах Firefox 128, Google Chrome 131 та Samsung Internet 19.

1 АНАЛІЗ СУЧАСНИХ ВЕБ-СЕРВІСІВ ДЛЯ ГРИ В ШАХИ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Основні характеристики багатокористувацьких веб-сервісів

Веб-сервіс є програмною системою, яка визначається уніфікованим ідентифікатором ресурсів і функціонує на базі браузера як клієнтської платформи. Стандартом проєктування веб-сервісів є клієнт-серверна архітектура, що складається з довільного набору клієнтів, які ініціюють запити та фіксованої кількості серверів, що обробляють запити та надсилають відповіді. Зазвичай, протоколом комунікації між клієнтом і сервером виступає HTTP або похідні від нього. Для забезпечення масштабування і відмовостійкості серверні програми дублюються та розгортаються у вигляді кластерів. Через зростання вимог до програмного забезпечення набуває поширення мікросервісна архітектура, що поділяє сервер на набір незалежних, відокремлено функціонуючих модулів.

Веб-сервіси для гри в шахи – це великі додатки із складною структурою, що зумовлено підтримкою великої кількості функціональних можливостей. Популярний сервіс Chess.com не надає відкритого доступу до вихідного коду та відомостей про внутрішню імплементацію. З позиції користувача, наявність платного функціоналу та велика кількість реклами погіршує загальний досвід використання даної платформи. Безкоштовний сервіс Lichess є відкритим проєктом, побудованим з використанням наступних технологій: мова програмування Scala, фреймворк Play, бази даних Redis, MongoDB та пошуковий рушій ElasticSearch. Велика кількість залежностей ускладнює аналіз архітектури і робить недоцільним використання Lichess у навчальних цілях. Ключовими перевагами розробленого сервісу є відкритість та відповідність принципам повторного використання програмних компонентів [1]. Реалізовані компоненти, такі як

інтерактивна шахова дошка, можуть бути легко інтегровані до сторонніх проєктів, що потребують аналогічного функціоналу. Наявність коментарів у кодовій базі значно прискорює процес вивчення архітектури додатку. Розмір скомпільованого виконуваного файлу серверного додатку становить 14 мегабайт. Це дозволяє розмістити додаток на сервері з малими апаратними ресурсами або у контейнеризованому хмарному середовищі.

1.2 Формат JSON як стандарт обміну даними в клієнт-серверних додатках

Найбільш популярним форматом обміну даними між клієнтом і сервером є JSON. Даний формат поєднує зручність читання людиною та компактність подання.

Структура JSON базується на наступних типах даних:

- числа, які можуть бути як цілими, так і містити дробову частину;
- рядки, відокремлені подвійними лапками;
- логічні значення;
- масиви елементів з довільним типом;
- набори пар ключ-значення, закодовані у вигляді рядків;
- порожнє значення null.

Майже всі сучасні мови програмування та бібліотеки підтримують вбудовані функції для перетворення формату JSON у типизовані змінні та зворотного перетворення.

1.3 Протокол WebSocket та його використання

Ефективна реалізація обміну даними між клієнтами є ключовим завданням при розробці багатокористувацьких веб-сервісів. Ігровий процес шахів може мати високу динамічність, тому забезпечення низької затримки у передаванні ходів і повідомлень між користувачами є необхідною вимогою.

Це обумовлює потребу у використанні двостороннього комунікаційного протоколу, який дозволяє серверу швидко поширювати інформацію. Крім того, важливо забезпечити стабільність та безпечність з'єднання, оскільки ці показники значно впливають на досвід використання сервісу.

Протокол HTTP підтримує відправку повідомлень лише від клієнта до сервера, що створює необхідність регулярно надсилати запити для отримання актуальної інформації про стан гри. Велика кількість запитів негативно впливає на масштабованість та продуктивність сервісу, викликаючи надмірне споживання ресурсів.

Технологія SSE надає можливість односпрямованої передачі даних від сервера до клієнта. Хоча це й дозволяє оперативно інформувати гравців про зміну ігрового стану, використання SSE ускладнює реалізацію ефективної відправки повідомлень з боку клієнта.

WebSocket є протоколом, що встановлює повнодуплексний канал зв'язку між клієнтським і серверним додатками. Завдяки використанню TCP-сокета забезпечується надійна передача даних [2]. Протокол дозволяє керувати параметрами вхідних та вихідних повідомлень, такими як максимальний розмір, крайній термін читання та допустима адреса відправника. Також можна встановити максимальну кількість активних з'єднань, яку буде обробляти сервер. Протокол підтримує шифрування з використанням TLS-сертифікатів.

Комунікація за протоколом WebSocket поділяється на чотири етапи. На першому етапі клієнт надсилає HTTP-запит із методом GET, заголовками Upgrade: websocket і Connection: Upgrade для визначення протоколу та набором заголовків Sec-WebSocket для встановлення параметрів шифрування повідомлень. На другому етапі сервер надсилає відповідь із заголовком Sec-WebSocket-Accept в якості підтвердження. На третьому етапі з'єднання вважається встановленим і обидва вузли можуть надсилати дані. На четвертому етапі одна зі сторін перериває з'єднання (наприклад, користувач закриває сторінку сайту) [3].

1.4 Етапи проєктування веб-сервісу для гри в шахи

У шахи грають на дошці фіксованого розміру двоє гравців, кожен з яких володіє однаковими фігурами: королем, королевою, двома турами, двома слонами, двома конями та восьмома пішаками. Кожен тип фігури має власну модель пересування. Первісна мета гри полягає в тому, щоб поставити мат королю опонента. Умовою мату є положення, де у гравця немає жодного ходу, що дозволяє уникнути взяття короля. Спеціальні правила включають рокірування для зміцнення безпеки короля, перетворення пішака після досягнення останнього рангу та «взяття на проході» для уникнення перетворення. Але мат не є єдиним шляхом завершення партії. Гра також може завершитися патовою позицією, здаванням однієї зі сторін, простроченням часу та погодженням нічиєї.

Початковим етапом проєктування є вибір алгоритмів та структур даних для реалізації правил гри. Шахова дошка складається з 64 квадратів, тому її можна закодувати у вигляді беззнакового 64-розрядного цілого числа. Кожен розряд позначає наявність або відсутність фігури на відповідному за номером квадраті. Такий метод має назву бітова дошка та через свою компактність та ефективність набув широкого використання [4]. Для представлення шахової дошки в цілому зручно об'єднати окремі бітові дошки для кожного типу та кольору фігур всередині масиву. Ключовою перевагою використання бітових дошок є можливість ефективно генерувати можливі ходи для фігур конкретного типу завдяки використанню порозрядних операцій.

На наступному етапі здійснюється проєктування модуля сервера, що підтримує комунікацію за протоколом WebSocket та обробляє вхідні повідомлення гравців. Зазвичай, багатокористувацькі веб-сервіси реалізують поділ ігрових сеансів на окремі кімнати, кожна з яких функціонує незалежно. Створення і видалення кімнат покладається на центральний керівний компонент, що має доступ до всіх активних кімнат. Важливо забезпечити можливість повторного підключення користувача у разі втрати з’єднання.

Визначення моделі авторизації та автентифікації є останнім етапом проєктування веб-сервісу. Сучасний стандарт JWT передбачає створення безпечних унікальних маркерів, які зберігають необхідні для ідентифікації користувачів дані.

1.5 Постановка завдання

Завданням даної кваліфікаційної роботи є розробка веб-сервісу, який дозволяє грати в шахи одночасно багатьом користувачам. Сервіс має:

- забезпечувати можливість створення ігор з обраними параметрами або підключення до вже створених ігор;
- реалізовувати правила шахів, відображати можливі ходи для обраної користувачем фігури;
- забезпечувати можливість комунікації гравців за допомогою ігрового чату;
- зберігати завершені партії та забезпечувати можливість їх переглядати;
- керувати доступом до ресурсів за допомогою безпечного стандарту JWT;
- підтримувати гру проти шахового рушія;
- забезпечувати шифрування трафіку з використанням протоколів HTTP Secure та WebSocket Secure.

Тестування розробленого сервісу є завершальним етапом розробки і дозволяє переконатися у його працездатності та стабільності функціонування. Тестові набори мають покривати різноманітні ситуації, включаючи обробку некоректних даних та дострокове переривання зв’язку між клієнтом і сервером. Важливим аспектом тестування є перевірка працездатності алгоритмів гри, таких як пошук можливих ходів гравця та перевірка правильності зробленого ходу.

2 ХАРАКТЕРИСТИКИ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Мова програмування Go як інструмент розробки серверних додатків

На етапі проєктування для реалізації серверного додатка було обрано мову програмування Go. Основними перевагами Go є статична типізація, що сприяє ранньому виявленню помилок під час компіляції, автоматичне управління пам’яттю, вбудована підтримка багатопотоковості, простота синтаксису, висока швидкість компіляції та виконання. Стандартна бібліотека мови надає реалізацію протоколу HTTP, що дозволяє ефективно створювати веб-сервіси [5].

Багатопотокове програмування в Go реалізовано через спеціальний тип даних goroutine (співпрограма). Goroutine являє собою функцію, яка виконується паралельно і незалежно від інших функцій. У контексті шахового додатку кожна ігрова сесія може відбуватись паралельно та забезпечувати швидку обробку ходів гравців. Процеси для виконання створюються за допомогою ключового слова go [6]. Слід зазначити, що планування виконання процесів (goroutine) здійснюється на рівні самої мови, а не операційної системи. Це дозволяє створювати велику кількість паралельних процесів без значного завантаження ком’ютерної системи.

Існує два класичні шляхи синхронізації процесів у Go: з використанням типу даних chan (канал) або пакету стандартної бібліотеки sync [7]. Sync надає реалізацію базових об’єктів синхронізації, таких як м’ютекси (Mutex) та групи очікування (WaitGroups).

Синхронізація за допомогою примітивів з пакету sync унеможливлює передачу даних між співпрограмами. Також розробнику необхідно пам’ятати, що неправильне блокування або розблокування співпрограми може призвести до непередбачуваної поведінки. Серед переваг використання пакету можна виділити швидкодію у порівнянні з типом chan.

Chan є вбудованим типом даних, який дозволяє організувати синхронізацію та обмін даними між співпрограмами. Канал працює як буфер, дозволяючи одному процесу записати дані, а іншому – отримати [8]. При записі даних у канал виконання процесу блокується, поки інший процес не отримає дані.

Використання каналів для синхронізації дозволяє уникнути одночасного доступу до пам'яті та є рекомендованим підходом при розробці веб-сервісів [9]. Пакет sync, через його швидкодію, частіше використовується при розробці низькорівневих бібліотек.

2.2 Бібліотека Gorilla WebSocket

Стандартна бібліотека мови Go не надає вбудованої підтримки протоколу WebSocket [10]. З цим пов'язано використання популярної бібліотеки Gorilla WebSocket, яка забезпечує просте та ефективне використання протоколу. Бібліотека легко інтегрується з серверами, розробленими на базі стандартного пакета net/http та підтримує всі специфіковані у стандарті RFC 6455 функції. Головним обмеженням є надання доступу до запису мережевих повідомлень лише одному процесу в момент часу.

Для створення з'єднання за протоколом WebSocket використовується метод Upgrade, який приймає значення HTTP-запиту та повертає покажчик на змінну Conn. Upgrader є структурою даних, що вказує параметри для створення та підтримування WebSocket з'єднання. Conn є структурою даних, що містить лише приховані поля, такі як низькорівнені читачі потоків. Для зчитування та запису повідомлень використовуються методи Conn.ReadMessage та Conn.WriteMessage.

Підтримка стабільності з'єднання реалізується через спеціальний механізм обміну службовими повідомленнями, відомий як ping-pong. Цей механізм потрібен для перевірки активності з'єднання та уникнення його

неочікуваного переривання. Принцип дії полягає у періодичному надсиланні сервером спеціальних повідомлень типу ping. Якщо клієнт відповідає повідомленням типу pong за встановлений час, з’єднання вважається активним. Для реалізації механізму в Gorilla WebSocket декларовані спеціальні типи повідомлень PingMessage та PongMessage, які потрібно передавати у методи WriteMessage та ReadMessage відповідно. Час очікування повідомлень pong має перевищувати інтервал надсилання повідомлень ping.

2.3 Мова програмування TypeScript як інструмент розробки клієнтських додатків

Браузери надають вбудовану підтримку лише мови програмування JavaScript. Ця мова є динамічно типізованою, що ускладнює розробку веб-сторінок із великою кількістю компонентів. Динамічна типізація не перевіряє правильність використання типів під час компіляції, що підвищує ризик виникнення помилок на етапі виконання. Крім того, відсутність чітко визначених типів у JavaScript унеможливлює роботу статичних аналізаторів, надання підказок середовищем розробки та збільшує потребу у інтегрованому тестуванні.

TypeScript – сучасна мова програмування, розроблена корпорацією Microsoft. Програма, написана на TypeScript, компілюється у стандартний JavaScript код, який можна виконувати у будь-якому сучасному браузері [11]. Основною метою створення мови стало додавання статичної типізації, використання якої значно зменшує кількість помилок на етапі написання коду.

Широкий спектр вбудованих типів TypeScript поділяється на примітивні, складені та спеціальні. Примітивні типи складаються з чисел (цілих і дробових), текстових рядків, логічних, порожніх або невизначених значень та унікальних ідентифікаторів. TypeScript дотримується об’єктно-

орієнтованої парадигми, тому до складених типів відносять об’єкти, класи, інтерфейси та масиви [12]. Існує багато спеціальних типів, деякі з них мають схожий принцип дії. В основному, спеціальні типи використовуються для визначення поведінки програмного коду та поліпшення його читабельності.

Більшість сучасних бібліотек для розробки динамічних веб-сторінок реалізовані мовою TypeScript. Код, написаний на JavaScript, є також коректним TypeScript кодом, що дозволяє поступово впроваджувати TypeScript у вже існуючі кодові бази.

2.4 Бібліотека React для створення графічного інтерфейсу

Створення динамічних веб-сторінок з використанням вбудованих механізмів TypeScript та прямих маніпуляцій з DOM зазвичай призводить до виникнення складнощів при розробці клієнтського додатка. Натомість у сучасних проєктах набуває поширення використання спеціалізованих бібліотек.

React є однією з найпопулярніших бібліотек для створення динамічних елементів інтерфейсу. Основна ідея полягає у розбитті вмісту сторінки на окремі компоненти, які є частинами інтерфейсу користувача та мають власну логіку. Перевагою такого підходу є можливість повторного використання компонентів, що дозволяє швидко розробляти складні графічні інтерфейси.

Впроваджуючи власну віртуальну модель DOM, React ефективно оновлює лише ті компоненти, внутрішній стан яких змінився. Програмування логіки компонентів можливе на мовах JavaScript та TypeScript [13].

Зовнішній вигляд компонентів описується за допомогою JSX (JavaScript XML) – розширення синтаксису JavaScript із вбудованою підтримкою TypeScript. Такий підхід дозволяє не розділяти логіку та опис зовнішнього вигляду компоненту, зберігаючи їх в одному файлі. JSX поєднує звичайний JavaScript-код із розміткою HTML, яка описує набір та структуру елементів.

Бібліотека React базується на декларативному підході. Це дозволяє зосередитись на описі графічного інтерфейсу для заданого стану, замість ручного оновлення вмісту сторінки. Під час зміни стану всі пов’язані з ним компоненти динамічно перемальовуються.

Одним із ключових недоліків бібліотеки є обмеження ефективності роботи пошукової оптимізації. Через використання віртуального DOM, первісна структура сторінки представлена єдиним HTML елементом, тоді як основний вміст додається лише після завантаження і виконання коду на боці клієнта. Зазвичай пошукові системи індексують веб-сторінки на основі надісланої сервером розмітки, що негативно впливає на відображення в результатах пошуку.

2.5 Реляційна база даних PostgreSQL

Основними типами даних, з якими взаємодіє шаховий веб-сервіс, є завершені ігрові партії та профілі гравців. Структура цих даних містить чітко визначені зв’язки, такі як асоціація кожної гри з двома гравцями. Виходячи з цього, було обрано реляційну модель даних, яка передбачає використання первісних та зовнішніх ключів для встановлення зв’язків між сутностями. Такий підхід підвищує ефективність виконання операцій вибірки та безпечність видалення записів.

В ході проєктування структури бази даних було розроблено схему таблиць та зв’язків, представлену на рисунку 2.1. Таблиця player призначена для зберігання інформації про зареєстрованих гравців. Таблиця tokenregistration призначена для тимчасового зберігання облікових даних користувачів під час процедури підтвердження електронної пошти. Первинний ключ token містить рядок випадково згенерованих символів, який надсилається користувачу з метою підтвердження реєстрації. У разі ініціації запиту на скидання створюється запис у таблиці tokenreset, який містить хешоване значення нового пароля та унікальний токен. Після підтвердження

операції користувачем, дані таблиці player оновляються. Завершені шахові партії зберігаються у таблиці game, кожен запис якої ідентифікується унікальним ідентифікатором, що забезпечує можливість вибірки даних конкретної гри.

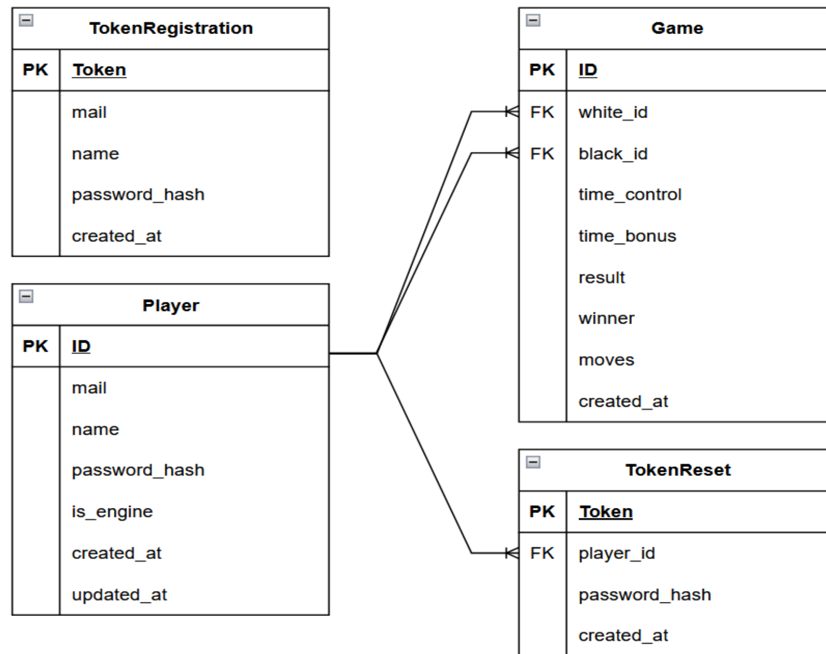

Рисунок 2.1 – Діаграма сутностей та зв’язків бази даних

Система керування базами даних PostgreSQL є безкоштовною та надає вбудовану підтримку транзакцій, складених запитів і механізмів створення користувацьких типів [14]. Транзакції використовуються для безпечного скасовування змін, коли на будь-якому етапі їх впровадження виникає помилка. Якщо під час реєстрації відправка електронного листа не відбулася, транзакції дозволяють безпечно видалити облікові дані, щоб користувач мав змогу повторити спробу. Складені запити огортають декілька SQL-виразів у єдиний запит, що спрощує отримання повної інформації про завершені партії разом з даними обох гравців. Користувацькі типи даних застосовуються для зберігання нестандартних видів інформації, таких як результат гри та колір переможця. Для створення користувацьких типів використовується вираз create domain, після чого вказується назва та можливі значення типу [15].

3 ОПИС РОЗРОБЛЕНИХ ПРОГРАМНИХ КОМПОНЕНТІВ

3.1 Опис програмного коду серверного додатка

Проєктування веб-сервісу дозволяє прискорити процеси написання та налагодження коду. Створення архітектури проєкту відображає, як компоненти системи пов’язані між собою. Правильно організована архітектура дозволяє легко орієнтуватися в кодовій базі та швидко вносити зміни. Кращими практиками при проєктуванні файлової структури є уникнення надмірної вкладеності каталогів, використання лаконічних назв файлів та відокремлення конфігураційних файлів від програмного коду.

3.1.1 Архітектура проєкту

Пакет є базовою одиницею організації коду в мові програмування Go. Кожен пакет складається з логічно пов’язаних програмних файлів, які зберігаються у єдиному каталозі. Такий підхід базується на сучасних принципах проєктування програмного забезпечення, зокрема модульності та єдиної відповідальності. Розподіл функціоналу на невеликі компоненти полегшує процес покриття тестовими наборами та сприяє багаторазовому використанню коду. Утиліта go test виконує файли тестування, що знаходяться у каталогах пакетів. Утиліта go doc генерує HTML-документацію розроблених пакетів на основі коментарів у вихідному коді. Команда go get дозволяє завантажувати та використовувати зовнішні пакети, які мають відкриту ліцензію.

На етапі проєктування було розроблено архітектуру, зображену на рисунку 3.1. Користувачі взаємодіють з клієнтським додатком за протоколом HTTP. У свою чергу клієнтський додаток надсилає запити до пакетів auth, player, game та ws.

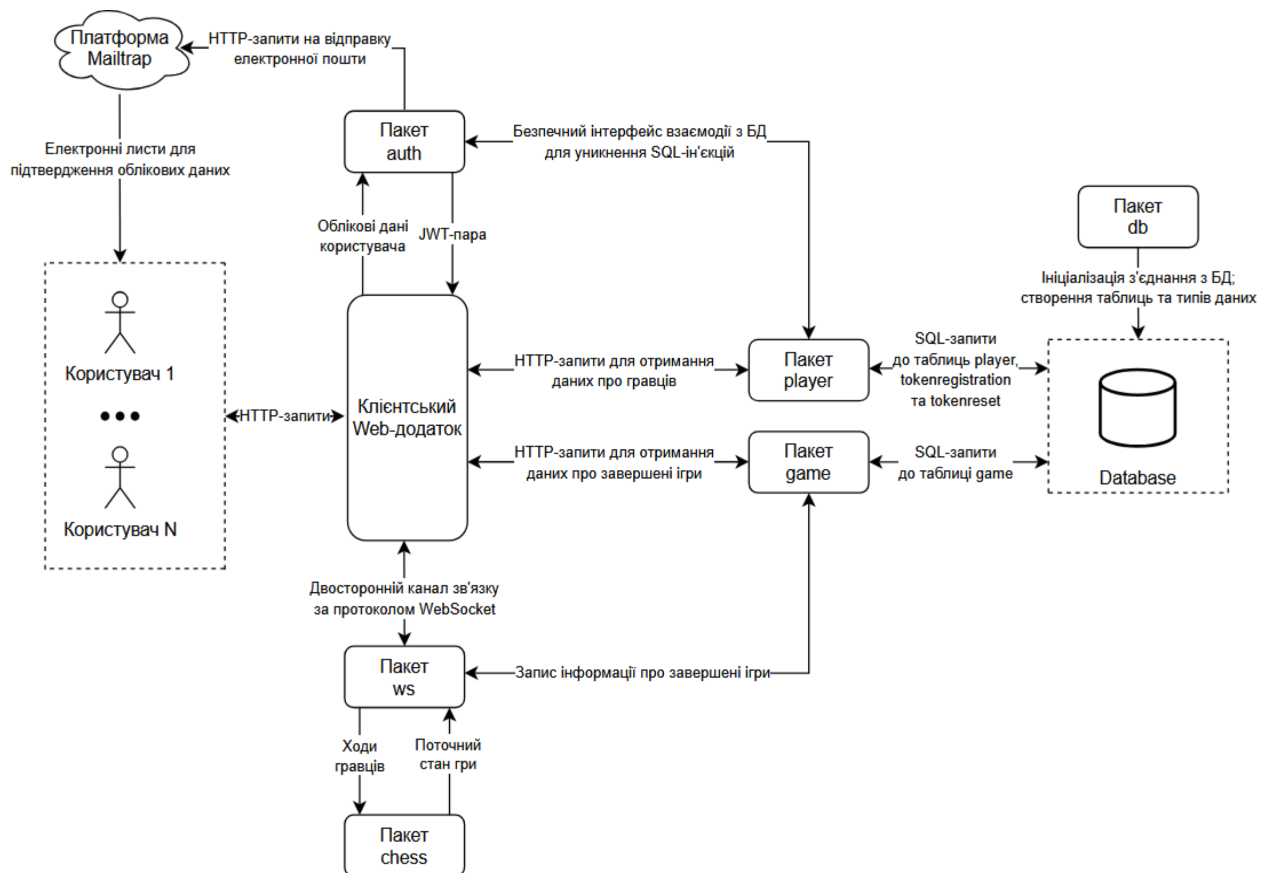

Рисунок 3.1 – Архітектура веб-сервісу

3.1.2 Пакет main

Призначенням пакету main є запуск веб-сервера, який виконує обробку вхідних запитів. Функція main (лістинг 3.1) є вхідною точкою виконання програми. Спочатку відбувається відкриття файлу log.txt для запису повідомлень про функціонування додатку. Після чого з операційної системи завантажується конфігураційний файл dev.env, параметри якого мають глобальний рівень доступу через механізм змінних середовища. Процедурою setupMux створюється мережевий мультиплексор, який приймає вхідні HTTP-запити та передає їх у відповідні функції обробки. За допомогою функції ListenAndServeTLS пакета net/http додаток прослуховує мережевий трафік на захищеному порту 443 та очікує на вхідні запити. Для шифрування трафіку використовуються ключ та сертифікат, які зберігаються у файлах key.pem та cert.pem.

Лістинг 3.1 – Функція main

```
func main() {
    logFile, err := os.OpenFile("./log.txt",
os.O_RDWR|os.O_APPEND, os.ModeAppend)
    if err != nil { log.Fatalf("cannot open 'log.txt' file for
writing: %v\n", err) }
    log.SetFlags(log.Ldate | log.Ltime | log.Lshortfile)
    log.SetOutput(logFile)
    loadEnv()
    log.Println("environment variables are loaded successfully")
    db.Open()
    defer db.Pool.Close()
    mux := setupMux()
    err = http.ListenAndServeTLS(":443", "cert.pem", "key.pem",
mux)
    if err != nil { log.Printf("%v\n", err) }}
```

Для реалізації контролю доступу до ресурсів сервера із зовнішніх доменів було застосовано механізм Cross-Origin Resource Sharing. Механізм базується на додаванні HTTP-заголовків із параметрами безпеки до кожного запиту. Заголовок Access-Control-Allow-Origin визначає домени, які можуть отримати доступ до ресурсів сервера. Автоматичне додавання заголовків реалізовано функцією allowCors, код якої наведено в лістингу 3.2.

Лістинг 3.2 – Функція allowCors

```
func allowCors(next http.Handler) http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        rw.Header().Add("Access-Control-Allow-Origin",
os.Getenv("DOMAIN"))
        rw.Header().Add("Access-Control-Allow-Credentials",
"true")
        rw.Header().Add("Access-Control-Allow-Headers",
"Authorization")
        rw.Header().Add("Access-Control-Allow-Methods", "GET,
POST, PUT, OPTIONS")
        if r.Method == "OPTIONS" {
            rw.WriteHeader(http.StatusOK)
            return }
        next.ServeHTTP(rw, r) }}
```

Функція isAuthorized (лістинг 3.3) забезпечує відхилення запитів, що надходять від неавторизованих користувачів. Відповідно до специфікації JWT, вхідні запити мають містити заголовок Authorization із токеном,

згенерованим сервером. У разі успішної авторизації за допомогою пакету context до запиту прикріплюються облікові дані користувача. Кожен запит проходить перевірку функцією isAuthorized.

Лістинг 3.3 – Функція isAuthorized

```
func isAuthorized(next http.Handler) http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        h := r.Header.Get("Authorization")
        if len(h) < 100 || h[:7] != "Bearer " {
            log.Printf("unauthorized request: %s\n",
r.RemoteAddr)
            rw.WriteHeader(http.StatusUnauthorized)
            return }
        cms, err := auth.DecodeToken(h[7:],
"ACCESS_TOKEN_SECRET")
        if err != nil { log.Printf("unauthorized request: %s\n",
r.RemoteAddr)
            rw.WriteHeader(http.StatusUnauthorized)
            return }
        ctx := context.WithValue(r.Context(), auth.Cms, cms)
        next.ServeHTTP(rw, r.WithContext(ctx)) }}
```

3.1.3 Пакет auth

Авторизація та автентифікація користувачів є ключовими компонентами забезпечення безпеки веб-сервісу. Пакет auth складається з двох файлів, які містять реалізацію функцій авторизації та автентифікації. Файл auth.go описує логіку обробки HTTP-запитів, тоді як файл jwt.go зосереджується на реалізації процедур генерації та декодування JWT-токенів.

Автентифікація поділяється на два види: первинна та повторна. Під час проходження первинної автентифікації користувач вводить облікові дані та має підтвердити електронну адресу за допомогою посилання із унікальним токеном. Механізм проходження процесу первинної автентифікації зображено на рисунку 3.2.

Більшість постачальників послуг електронної пошти позначать як спам листи, надіслані від нових доменів. Це зумовлює потребу у використанні зовнішніх платформ доставки електронної пошти. Такі платформи

дозволяють відправляти листи за допомогою HTTP-запитів і надають вбудовані механізми запобігання потраплянню листів у спам. Було обрано платформу Mailtrap, яка надає безкоштовний тарифний план з можливістю відправки 1000 листів на місяць та лімітом у 200 листів на добу.

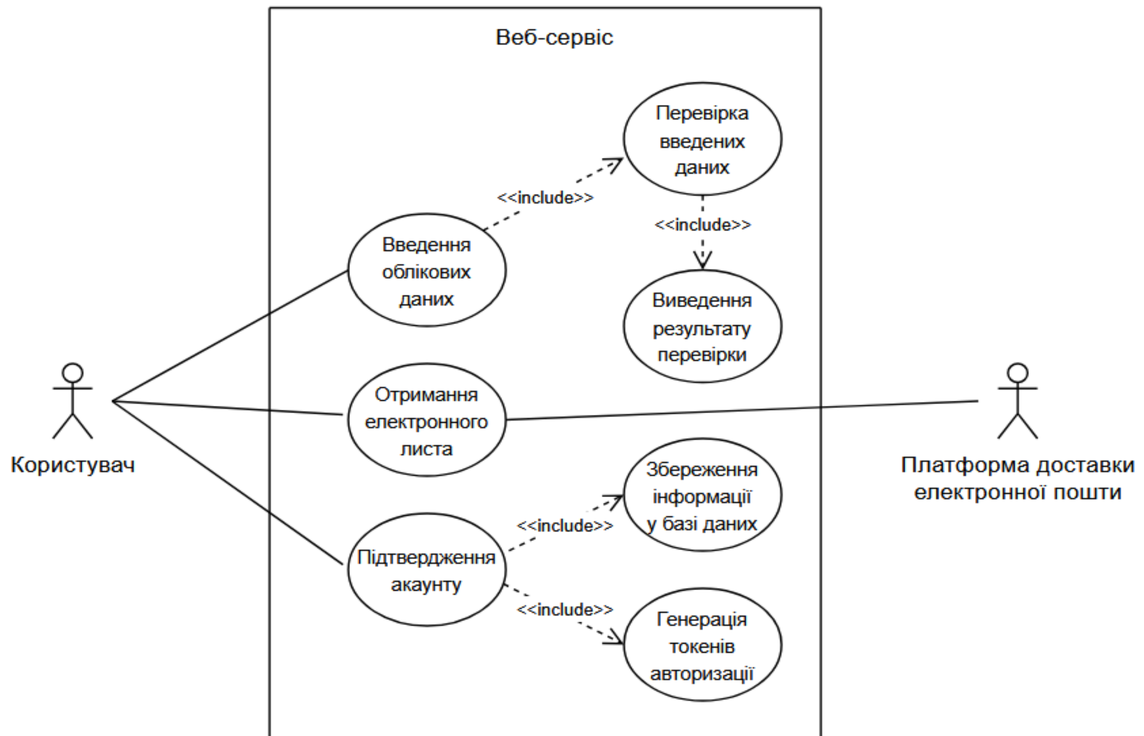

Рисунок 3.2 – Use Case діаграма процесу автентифікації

Код файлу auth.go наведено у Додатку Б, лістинг Б.1.1. Функція signUp приймає запити на реєстрацію та записує облікові дані до таблиці tokenregistration, виконуючи хешування паролю для безпечного зберігання. Функція signIn обробляє повторну автентифікацію шляхом порівняння наданих облікових даних зі збереженими. У разі коректного введення даних функція signIn виконує авторизацію користувача.

Авторизацію реалізовано з дотриманням стандарту JWT, який передбачає використання токенів доступу та оновлення. Кожен токен складається із заголовку, зашифрованих користувацьких даних та криптографічного підпису. За допомогою заголовку визначається алгоритм підпису, наприклад, симетричний алгоритм HS256. Користувацькі дані є

основною інформацією, яка необхідна для ідентифікації гравців та виконання авторизації. Сервер здійснює підпис перших двох частин токена за допомогою секретного ключа, який унеможливлює модифікацію або підробку токена. Процес авторизації поділяється на шість етапів, зображених на рисунку 3.3.

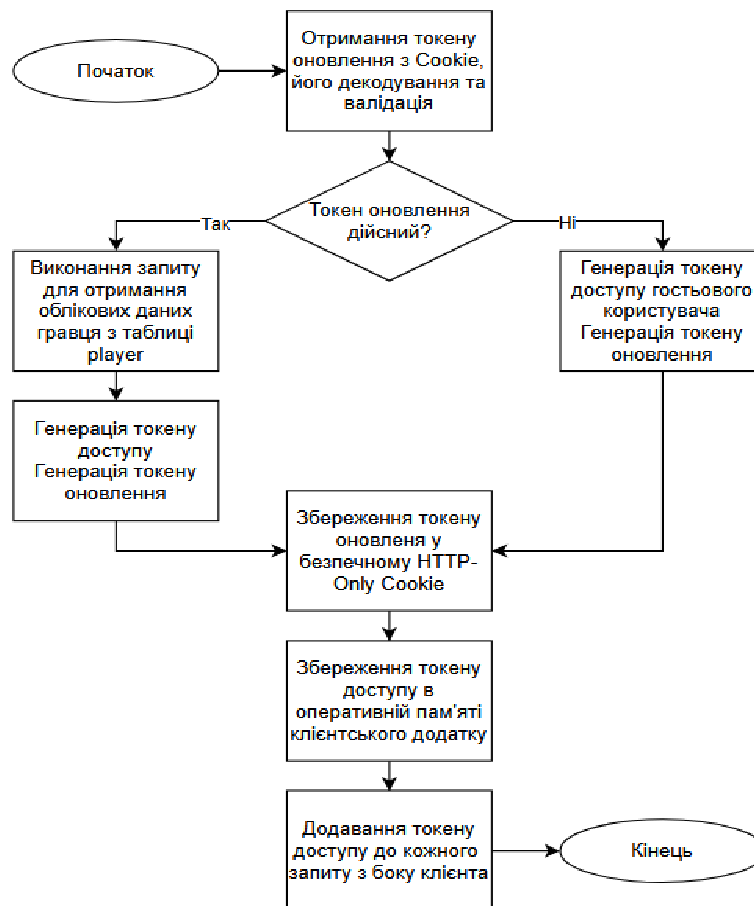

Рисунок 3.3 – Алгоритм проходження авторизації

Вхідні запити до захищених ресурсів мають містити заголовок Authorization із токеном доступу, який має обмежений час дії. Для створення нового токена доступу не вимагаючи повторного введення облікових даних, використовується токен оновлення. За допомогою механізму безпечних HTTP-Only Cookie забезпечується збереження токену оновлення у браузері клієнта протягом тривалого проміжку часу. Гостьова авторизація дозволяє взаємодіяти з веб-сервісом без необхідності створення облікового запису.

Спочатку генеруються унікальні ідентифікатор та ім’я користувача, після чого на основі цих даних створюються маркери доступу та оновлення. Токен оновлення додається до сховища Cookie за допомогою функції setRefreshTokenCookie. Токен доступу надсилається клієнтові у відповідь на запит. Для генерації JWT було розроблено функцію generateToken (Додаток Б, лістинг Б.1.2), з використанням функції NewWithClaims пакету golang-jwt.

3.1.4 Пакет chess

Розміщення основної логіки у відокремленому пакеті є кращою практикою під час розробки масштабних додатків. Пакет chess зосереджується на реалізації правил гри, описі структур даних та взаємодії між ними. Такий підхід спрощує підтримку коду, оскільки основна логіка залишається незалежною від інших програмних компонентів.

Стан гри описується структурою Game, наведеною у лістингу 3.5. Публічні поля структури зберігають підсумковий результат гри, колір фігур переможця, вказівник на структуру Bitboard, значення таймерів, ідентифікатори гравців та часові параметри гри. Поля, що залишилися, описують канали взаємодії, необхідні для керування шаховою грою у режимі реального часу.

Лістинг 3.5 – Структура Game

```
type Game struct { Result      enums.Result
    Winner      enums.Color
    Bitboard    *bitboard.Bitboard
    WhiteTime   int
    BlackTime   int
    WhiteId     uuid.UUID
    BlackId     uuid.UUID
    TimeControl int
    TimeBonus   int
    Moves       []CompletedMove
    clock       *time.Ticker
    Move chan MoveEvent
    Info chan GameInfoEvent
    End chan struct{} }
```

Метод RunRoutine структури Game (лістинг 3.6) виконує безперервну обробку подій гри. Зчитування вхідного потоку подій з каналів структури реалізується за допомогою вбудованої конструкції for select. Використання каналів забезпечує синхронізовану комунікацію між паралельними процесами гри та кімнати. Такій підхід утворює чергу вхідних повідомлень, яка обслуговується за дісципліною FIFO.

Лістинг 3.6 – Метод RunRoutine структури Game

```
func (g *Game) RunRoutine(timeout chan<- struct{}) {
    defer func() { g.clock.Stop() }()
    for {
        select {
        case me := <-g.Move:
            me.Response <- g.ProcessMove(me.Move)
            if g.Result != enums.Unknown { return }
        case <-g.clock.C:
            g.handleTimeTick()
            if g.WhiteTime == 0 || g.BlackTime == 0 {
                g.SetEndInfo(enums.Timeout,
g.Bitboard.ActiveColor)
                timeout <- struct{}{}
                return
            }
        case gie := <-g.Info:
            gie.Response <- GameInfo{
                WhiteTime: g.WhiteTime, BlackTime: g.BlackTime,
                Result: g.Result, Winner: g.Winner, Moves:
g.Moves[:], LegalMoves: g.Bitboard.LegalMoves[:] }
        case <-g.End: return
}}}
```

Для представлення ігрової позиції було створено тип даних Bitboard, який реалізує підхід бітової дошки. Код структури наведено в лістингу 3.7. Масив Pieces складається з дванадцяти бітових дошок: по одній для кожного типу фігури обох кольорів. Поле ActiveColor зберігає колір гравця, який може зробити хід. Масив прапорів CastlingRights визначає права гравців на рокіровку. Поле EPTarget зберігає номер квадрату, доступного для взяття на проході, або значення enums.NoSquare, якщо такий квадрат відсутній. Лічильник півходів HalfmoveCnt зберігає кількість ходів від моменту останнього взяття або переміщення пішака, що використовується при

реалізації правила п’ятидесяти ходів. Загальна кількість повних ходів зберігається у полі FullmoveCnt та інкрементується після кожного ходу чорної фігури. Масив LegalMoves зберігає згенеровані допустимі ходи.

Лістинг 3.7 – Структура Bitboard

```
type Bitboard struct {
    Pieces      [12]uint64
    ActiveColor enums.Color
    CastlingRights [4]bool
    EPTarget int
    HalfmoveCnt int
    FullmoveCnt int
    LegalMoves []Move }
```

Одним із найбільш компактних способів подання шахових ходів є 16-ти розрядні цілі беззнакові числа. Такий підхід широко застосовується під час проєктування баз даних, оскільки дозволяє ефективно зберігати велику кількість зроблених ходів. Перші шість розрядів відповідають за координати кінцевого квадрату, наступні шість – за координати початкового квадрату, тоді як останні чотири розряди позначають тип ходу. На основі вбудованого типу uint16 було описано користувацький тип Move, представлений у лістингу 3.8.

Лістинг 3.8 – Тип даних Move та його методи

```
type Move uint16
func NewMove(to, from int, mt enums.MoveType) Move {
    return Move(to | (from << 6) | int(mt<<12))
}
func (m Move) To() int { return int(m) & 0x3F }
func (m Move) From() int { return int(m>>6) & 0x3F }
func (m Move) Type() enums.MoveType {
    return enums.MoveType(m>>12) & 0xF }
```

Генерація допустимих ходів здійснюється у три етапи. Спочатку відбувається обчислення псевдодопустимих ходів для всіх типів фігур, за винятком короля. Далі виконується фільтрація отриманих ходів з метою виділення лише допустимих. Завершальним етапом є генерація допустимих

ходів короля. Оскільки обчислення псевдодопустимих ходів короля є затратною операцією, доцільно відразу генерувати допустимі ходи. Метод GenLegalMoves (лістинг 3.9) реалізує описану процедуру.

Лістинг 3.9 – Метод GenLegalMoves структури Bitboard

```
func (bb *Bitboard) GenLegalMoves() {
    pseudoLegal := make([]Move, 0)
    c, opC := bb.ActiveColor, bb.ActiveColor^1
    allies := bb.Pieces[0+c] | bb.Pieces[2+c] | bb.Pieces[4+c] |
bb.Pieces[6+c] | bb.Pieces[8+c] | bb.Pieces[10+c]
    enemies := bb.Pieces[0+opC] | bb.Pieces[2+opC] |
bb.Pieces[4+opC] | bb.Pieces[6+opC] | bb.Pieces[8+opC] |
bb.Pieces[10+opC]
    pseudoLegal = append(pseudoLegal,
genPawnsPseudoLegalMoves(bb.Pieces[0+c], allies, enemies, c,
        bb.EPTarget)...)
    for i := 2; i < len(bb.Pieces)-2; i++ {
        if i%2 != int(bb.ActiveColor) || bb.Pieces[i] == 0 {
            continue
        }
        pseudoLegal = append(pseudoLegal,
genPseudoLegalMoves(enums.PieceType(i),
            bb.Pieces[i], allies, enemies)...)
    }
    bb.LegalMoves = bb.filterIllegalMoves(pseudoLegal)
    king := bb.Pieces[10+c]
    bb.Pieces[10+c] ^= king
    attacked := GenAttackedSquares(bb.Pieces, opC)
    bb.Pieces[10+c] ^= king
    bb.LegalMoves = append(bb.LegalMoves,
genKingLegalMoves(bb.Pieces[10+c], allies,
        enemies, attacked, bb.CastlingRights[c],
bb.CastlingRights[c+2], c)...) }
```

Фільтрація псевдодопустимих ходів здійснюється методом filterIllegalMoves, код якого наведено в лістингу 3.10. На кожній ітерації циклу створюється копія бітової дошки, відбувається виконання ходу та здійснюється перевірка, чи не знаходиться союзний король під атакою. Якщо умова виконується, хід вважається допустимим та додається до масиву legal. Після завершення перевірки стан дошки відновлюється до попереднього. Такий підхід дозволяє зменшити кількість виконуваних операцій на етапі генерації допустимих ходів.

Лістинг 3.10 – Метод filterIllegalMoves структури Bitboard

```
func (bb *Bitboard) filterIllegalMoves(pseudoLegal []Move)
(legal []Move) {
    boardCopy := bb.Pieces
    for _, move := range pseudoLegal {
        bb.MakeMove(move)
        if GenAttackedSquares(bb.Pieces, bb.ActiveColor^1)&
            bb.Pieces[10+bb.ActiveColor] == 0 { legal =
append(legal, move) }
        bb.Pieces = boardCopy
    }
    return }
```

Виконання ходів реалізується за допомогою методу MakeMove (лістинг 3.11), який переміщує фігури шляхом застосування порозрядних операцій до внутрішнього стану дошки. При звичайному ході виконується операція виключної диз’юнкції між поточною бітовою дошкою та маскою, що відповідає цільовому квадрату. У випадку атаки додатково скидається біт, порядковий номер якого відповідає позиції захопленої фігури. При виконанні рокірування змінюються позиції як короля, так і тури. При короткому рокіруванні тура переміщується на квадрат праворуч від короля, при довгому – на квадрат ліворуч від короля.

Лістинг 3.11 – Метод MakeMove структури Bitboard

```
func (bb *Bitboard) MakeMove(m Move) {
    var from, to uint64 = 1 << m.From(), 1 << m.To()
    fromTo := from ^ to
    movedPT := GetPieceOnSquare(from, bb.Pieces)
    switch m.Type() {
    case enums.Capture:
        bb.Pieces[GetPieceOnSquare(to, bb.Pieces)] ^= to
    case enums.EPCapture:
        if movedPT == enums.WhitePawn {
            bb.Pieces[enums.BlackPawn] ^= to >> 8
        }
        else { bb.Pieces[enums.WhitePawn] ^= to << 8 }
    case enums.KingCastle:
        rookFrom, rookTo := to<<1, to>>1
        bb.Pieces[movedPT-4] ^= rookFrom ^ rookTo
    case enums.QueenCastle:
        rookFrom, rookTo := to>>2, to<<1
        bb.Pieces[movedPT-4] ^= rookFrom ^ rookTo
    case enums.KnightPromo, enums.BishopPromo,
```

```
        enums.RookPromo, enums.QueenPromo:
        bb.Pieces[movedPT] ^= to
        bb.Pieces[movedPT+(promoPieces[m.Type()-6])] ^= to
    case enums.KnightPromoCapture, enums.BishopPromoCapture,
        enums.RookPromoCapture, enums.QueenPromoCapture:
        bb.Pieces[GetPieceOnSquare(to, bb.Pieces)] ^= to
        bb.Pieces[movedPT] ^= to
        bb.Pieces[movedPT+(promoPieces[m.Type()-10])] ^= to
    }
    bb.Pieces[movedPT] ^= fromTo }
```

Для коректного обчислення можливих ходів було визначено константні бітові маски notA, notH, notAB та notGH (лістинг 3.12). Бітова кон'юнкція із такими масками виключає обчислення ходів для фігур, які знаходяться на крайніх файлах. Це дозволяє уникнути вихід за межі дошки під час виконання арифметичних зсувів та спрощує алгоритми обчислень за рахунок зменшення кількості логічних перевірок у коді.

Лістинг 3.12 – Блок константних бітових масок

```
const (
    notA  uint64 = 0xFEFEFEFEFEFEFEFE
    notH  uint64 = 0x7F7F7F7F7F7F7F7F
    notAB uint64 = 0xFCFCFCFCFCFCFCFC
    notGH uint64 = 0x3F3F3F3F3F3F3F3F
)
```

Пішак може пересуватися тільки на один квадрат у напрямку до протилежної сторони дошки та на два квадрати з початкової позиції. Пішак атакує сусідні квадрати по діагоналі, а також за спеціальним ходом «взяття на проході», якщо ворожий пішак просунувся на два квадрати у попередньому ході. Для генерації ходів пішака було розроблено функцію genPawnsPseudoLegalMoves, код якої наведено у Додатку Б, лістинг Б.2.1. Алгоритм складається з циклу, який проходить по всіх встановлених розрядах бітової дошкі, обчислює напрямок атаки в залежності від кольору пішака та перевіряє можливість виконання спеціального ходу. Також виконується перевірка на досягнення останнього рангу з метою здійснення перетворення пішака на іншу фігуру.

Шаблон пересування коня охоплює вісім квадратів, розташованих у формі літери «Г». Функція genKnightsAttackDests (лістинг 3.13) виконує обчислення псевдодопустимих ходів одночасно для всіх конів на бітовій дошці. На відміну від інших алгоритмів, обчислення ходів не потребує циклу, через те що кінь не має спеціальних ходів та охоплює однакову кількість квадратів незалежно від позиції.

Лістинг 3.13 – Функція genKnightsAttackDests

```
func genKnightsAttackDests(knights uint64) (moves uint64) {
    moves |= (knights & notA) >> 17
    moves |= (knights & notH) >> 15
    moves |= (knights & notAB) >> 10
    moves |= (knights & notGH) >> 6
    moves |= (knights & notAB) << 6
    moves |= (knights & notGH) << 10
    moves |= (knights & notA) << 15
    moves |= (knights & notH) << 17
    return moves }
```

Слон є фігурою, яка пересувається діагоналями у чотирьох напрямках. Алгоритм генерації псевдодопустимих ходів слона базується на послідовному проходженні кожного з напрямків, причому у разі виявлення союзної або ворожої фігури подальше просування в цьому напрямку припиняється. Оскільки положення фігур на різних діагоналях може відрізнятися, генерація ходів для всіх слонів на дошці вимагає виконання даної функції для кожного встановленого розряду. Реалізацію алгоритму наведено у лістингу 3.14.

Лістинг 3.14 – Функція genBishopAttackDests

```
func genBishopAttackDests(bishop, occupied uint64)
(moves uint64) {
    for i := (bishop & notA) >> 9; i != 0; i = (i & notA) >> 9 {
        moves |= i
        if occupied&i != 0 { break } }
    for i := (bishop & notH) >> 7; i != 0; i = (i & notH) >> 7 {
        moves |= i
        if occupied&i != 0 { break } }
    for i := (bishop & notA) << 7; i != 0; i = (i & notA) << 7 {
```

```
        moves |= i
        if occupied&i != 0 { break } }
    for i := (bishop & notH) << 9; i != 0; i = (i & notH) << 9 {
        moves |= i
        if occupied&i != 0 { break } }
    return moves }
```

Тура пересувається вздовж вертикальних та горизонтальних ліній. Псевдодопустимі ходи визначаються шляхом покрокового проходження кожного напрямку до появи перешкоди у вигляді іншої фігури. При виконанні ходу турою права на рокірування короля із відповідною турою скидаються. Функція genRookAttackDests (лістинг 3.15) виконує генерацію псевдодопустимих ходів тури.

Лістинг 3.15 – Функція genRookAttackDests

```
func genRookAttackDests(rook, occupied uint64) (moves uint64) {
    for i := rook << 8; i != 0; i <<= 8 {
        moves |= i
        if occupied&i != 0 { break } }
    for i := (rook & notA) >> 1; i != 0; i = (i & notA) >> 1 {
        moves |= i
        if occupied&i != 0 { break } }
    for i := (rook & notH) << 1; i != 0; i = (i & notH) << 1 {
        moves |= i
        if occupied&i != 0 { break } }
    for i := rook >> 8; i != 0; i >>= 8 {
        moves |= i
        if occupied&i != 0 { break } }
    return moves }
```

Шаблон пересування королеви є простим поєднанням рухів тури та слона, тобто королева може пересуватися по горизонталі, вертикалі та по чотирьох діагоналях. Для уникнення повторень коду, генерацію псевдодопустимих ходів королеви було побудовано на основі вже створених функцій (лістинг 3.16).

Лістинг 3.16 – Функція genQueenAttackDests

```
func genQueenAttackDests(queen, occupied uint64) uint64 {
    return genBishopAttackDests(queen, occupied) |
        genRookAttackDests(queen, occupied) }
```

Король є найважливішою шаховою фігурою, тому генерація його ходів потребує особливої уваги. За правилами, король може переміщуватись на сусідній квадрат у будь-якому напрямку, якщо цей квадрат не зайнятий союзною фігурою та не знаходиться під атакою. Королі протилежних кольорів ніколи не можуть розміщуватися на сусідніх квадратах. Для генерації допустимих ходів короля необхідно попередньо згенерувати бітову дошку квадратів, які знаходяться під атакою ворожих фігур.

Існує спеціальний ход короля – рокірування, при якому король рухається на два квадрати в сторону тури, а тура переміщується на останній квадрат, який король пройшов. Рокірування дозволяється за такими умовами:

- король та тура раніше не робили ходів;
- всі квадрати між королем і турою вільні та не знаходяться під атакою;
- король не знаходиться під атакою.

Код розробленої функції genKingLegalMoves наведено у лістингу 3.17.

Лістинг 3.17 – Функція genKingLegalMoves

```
func genKingLegalMoves(king, allies, enemies, attacked uint64,
    canOO, canOOO bool, c enums.Color) (moves []Move) {
    kingPos := GetLSB(king)
    movesBB := genKingAttackDests(king) & ^allies & ^attacked
    for ; movesBB > 0; movesBB &= movesBB - 1 {
        to := GetLSB(movesBB)
        if (1<<to)&enemies != 0 {
            moves = append(moves, NewMove(to, kingPos,
enums.Capture))
        } else { moves = append(moves, NewMove(to, kingPos,
enums.Quiet)) } }
    if c == enums.White {
        if canOO && (0x60&allies == 0) && (0x70&attacked == 0) {
            moves = append(moves, NewMove(enums.G1, kingPos,
enums.KingCastle))
        }
        if canOOO && (0xE&allies == 0) && (0x1E&attacked == 0) {
            moves = append(moves, NewMove(enums.C1, kingPos,
enums.QueenCastle))
        } } else {
        if canOO && (0x6000000000000000&allies == 0) &&
(0x7000000000000000&attacked == 0) {
            moves = append(moves, NewMove(enums.G8, kingPos,
```

```
enums.KingCastle)) }
        if canOOO && (0xE000000000000000&allies == 0) &&
(0x1E00000000000000&attacked == 0) {
            moves = append(moves, NewMove(enums.C8, kingPos,
enums.QueenCastle))
        } }
    return }
```

3.1.5 Пакет db

Вбудована бібліотека мови програмування Go містить пакет database/sql, який реалізує високорівнений інтерфейс взаємодії з реляційними базами даних. Функція sql.Open використовує спеціалізований рядок підключення для визначення параметрів з’єднання та повертає покажчик на структуру sql.DB. Структура DB є дескриптором бази даних із низькорівневими атрибутами, модифікація яких неможлива. Одним із полів структури є м’ютекс, що дозволяє декільком потокам одночасно виконувати запити. Метод Ping дозволяє перевірити активність з’єднання, повторно встановлюючи його у разі виявлення помилки. Метод Exec виконує SQL-запити, що не повертають результати. Деякі типи запитів, такі як SELECT, мають повернути результати, тому їх потрібно виконувати за допомогою методу Query. Метод Close перериває з’єднання з базою даних та звільняє ресурси дескриптора. Через автоматичне керування пам’яттю мовою Go використання методу Close не є обов’язковим.

Для спрощення процесу розгортання було розроблено файл, що містить SQL-запити створення необхідних таблиць. За допомогою функції os.ReadFile відбувається зчитування файлу у символьний рядок. Після цього запити виконуються методом Exec.

Таблиця player (лістинг 3.18) містить первинний ключ, який генерується за допомогою функції uuid_generate_v4 під час створення запису. До облікових даних гравців відносяться адреса електронної пошти, ім’я та хешований пароль. Прапорець is_engine визначає, чи є відповідний гравець шаховим рушієм. Окремий запис для шахового рушія є необхідним для

коректного створення записів про зіграні партії, оскільки кожна гра повинна посилатися на двох гравців. Поля created_at та updated_at зберігають часові мітки створення та останньої модифікації запису.

Лістинг 3.18 – SQL-запит створення таблиці player

```
CREATE TABLE IF NOT EXISTS player (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    mail TEXT NOT NULL UNIQUE,
    name TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    is_engine BOOLEAN NOT NULL DEFAULT false,
    created_at TIMESTAMP NOT NULL DEFAULT now(),
    updated_at TIMESTAMP NOT NULL DEFAULT now()
);
```

Поле token таблиці tokenregistration зберігає рядок випадково згенерованих символів, який надсилається користувачу з метою підтвердження реєстрації. Токен є одноразовим та має термін дії дванадцять хвилин. У разі успішної верифікації облікові дані переносяться до основної таблиці player. Повторне використання облікових даних обмежується ідентифікатором UNIQUE. SQL-запит для створення таблиці tokenregistration наведено в лістингу 3.19.

Лістинг 3.19 – SQL-запит створення таблиці tokenregistration

```
CREATE TABLE IF NOT EXISTS tokenregistration (
    token TEXT PRIMARY KEY,
    mail TEXT NOT NULL UNIQUE,
    name TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT now() );
```

Зберігання електронних адрес користувачів дозволяє безпечно реалізувати механізм скидання пароля. Для зберігання значення нового пароля було створено таблицю tokenreset (лістинг 3.20). Поле player_id є зовнішнім ключем, який посилається на таблицю player. Поле created_at використовується для перевірки терміну дії токена. Записи, що зберігаються більше 20 хвилин, вважаються недійсними.

Лістинг 3.20 – SQL-запит створення таблиці tokenreset

```
CREATE TABLE IF NOT EXISTS tokenreset (
    token TEXT PRIMARY KEY,
    player_id UUID NOT NULL REFERENCES player(id),
    password_hash TEXT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT now() );
```

Завершені шахові партії зберігаються у таблиці game, структуру якої наведено у лістингу 3.21. Кожен запис ідентифікується первинним ключем id, який забезпечує можливість вибірки даних конкретної гри. Таблиця містить зовнішні ключі, які встановлюють зв’язки з двома гравцями. Кожен хід кодується у форматі 32-х розрядного цілого числа, перші шістнадцять бітів якого зберігають інформацію про тип та характер переміщення фігури, а решта бітів зберігає кількість секунд, які залишились на годиннику після виконання ходу.

Лістинг 3.21 – SQL-запит створення таблиці game

```
CREATE TABLE IF NOT EXISTS game (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    white_id UUID NOT NULL REFERENCES player(id),
    black_id UUID NOT NULL REFERENCES player(id),
    time_control SMALLINT NOT NULL,
    time_bonus SMALLINT NOT NULL,
    result GAME_RESULT NOT NULL,
    winner COLOR NOT NULL,
    moves INTEGER[] NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT now() );
```

3.1.6 Пакет game

Пакет game реалізує функції обробки HTTP-запитів, що пов’язані з отриманням даних про завершені ігри. Безпечна взаємодія з таблицею game забезпечується за рахунок використання параметризованих SQL-запитів, які унеможливлюють здійснення атак типу SQL-ін’єкція. При реалізації функцій обробки HTTP-запитів було використано стандартний пакет net/http, який надає структури ResponseWriter для формування та надсилання відповіді та Request для отримання параметрів запиту.

Файл game.go (Додаток Б, лістинг Б.3.1) інкапсулює логіку взаємодії з користувачем на рівні протоколу HTTP. Функція Mux виконує реєстрацію обробників запитів за двома маршрутами. Функція getById обробляє запити за маршрутом «/game/id/{id}», де параметр id визначає унікальний ідентифікатор гри. У разі успішного звертання до бази даних інформація про гру кодується у форматі JSON та надсилається у відповіді. Функція getByPlayerId прослуховує маршрут «/game/player/{id}» та повертає стислу інформацію про завершені ігри, у яких брав участь гравець із зазначеним у адресі запиту ідентифікатором.

Логіку взаємодії з таблицею game було розміщено у файлі gamerepository.go (Додаток Б, лістинг Б.3.2). Функція selectById виконує пошук повної інформації про гру на основі переданого ідентифікатора. Якщо запис із таким ідентифікатором існує, selectById повертає дані про гравців, масив зроблених ходів, результат та час проведення гри. Вибірка інформації про гравців здійснюється SQL-виразом JOIN, який дозволяє поєднувати записи кількох таблиць на основі спільних даних. Функція selectByPlayerId виконує вибірку даних про ігри, в яких приймав участь гравець. Вибірка базується на SQL-конструкції WHERE, яка дозволяє створювати умови фільтрації записів таблиці. Призначенням функції Insert є додавання нових рядків до таблиці game. Після завершення гри її інформація автоматично передається у функцію Insert для подальшого збереження у базі даних. Допоміжна функція Array пакету pq перетворює масив закодованих ходів у формат, сумісний із системою PostgreSQL.

3.1.7 Пакет player

Призначенням пакета є забезпечення доступу до таблиць player, tokenregistration та tokenreset. Під час взаємодії з базою даних використовується структура Tx пакета database/sql, яка описує SQL-транзакцію та дозволяє об’єднувати декілька запитів у єдину операцію.

Кожна транзакція має завершуватися викликом методу Commit або Rollback. Метод Commit застосовує всі зміни в межах транзакції, тоді як Rollback скасовує їх у разі виникнення помилки.

Файл player.go (Додаток Б, лістинг Б.4.1) містить реалізацію обробника HTTP-запитів getById, який повертає дані про зареєстрованого гравця за унікальним ідентифікатором. Чутливі дані, такі як хеш пароля та електронна адреса, не включаються до відповіді. Використання ідентифікатора в якості атрибута вибірки гарантує цілісність зв’язків між записами у пов’язаних таблицях.

Файл playerrepository.go (Додаток Б, лістинг Б.4.2) описує логіку вибірки, створення та оновлення записів. Функція SelectPlayerById забезпечує отримання облікових даних гравця за його унікальним ідентифікатором. Під час проходження автентифікації пакет auth використовує функцію SelectPlayerByLogin, яка виконує вибірку гравця за допомогою імені або електронної адреси. Додавання нових записів до відповідних таблиць реалізується процедурами InsertTokenRegistration, InsertPlayer та InsertTokenReset із застосуванням оператору INSERT. Оновлення пароля здійснюється функцією UpdatePasswordHash за допомогою оператора UPDATE.

3.1.8 Пакет ws

Пакет ws описує логіку взаємодії з клієнтським додатком за протоколом WebSocket. З технічної точки зору, пакет базується на використанні структури Conn, яка надається бібліотекою Gorilla WebSocket. Ця структура представляє сокетне з’єднання та забезпечує методи ReadMessage та WriteMessage для обміну WebSocket-повідомленнями через мережу. Відповідно до специфікації RFC 6455, повідомлення можуть бути закодовані в текстовому або двійковому форматі. Пакет ws також відповідає за керування паралельними процесами створених ігрових кімнат.

Файл client.go описує інтерфейс взаємодії між клієнтським та серверним додатками на рівні протоколу WebSocket. Код структури client та її методів наведено у Додатку Б, лістинг Б.5.1. Поля id та name використовуються для ідентифікації клієнта. Покажчик на тип Room визначає поточну ігрову кімнату, до якої підключений клієнт. У разі відсутності підключення до жодної кімнати, значення покажчика дорівнює nil. Такий підхід забезпечує можливість передачі ходів та надсилання текстових повідомлень за рахунок звернення до каналів ігрової кімнати. Крім того, структура client містить покажчик на тип Hub, який керує створенням та видаленням ігрових кімнат. Методи readRoutine та writeRoutine виконують зчитування та запис мережевих повідомлень.

Візуальне уявлення поведінки програмної системи сприяє виявленню потенційних проблем на етапі проєктування. Множина станів та умови переходів між ними визначають реакцію об’єкта на зовнішні події. За стандартом мови графічного моделювання UML, для візуалізації поведінки програмних компонентів використовуються діаграми типу State Machine. Графічні оператори таких діаграм визначають початковий, кінцевий та функціональні стани системи.

Початковий стан може містити тільки один перехід до першого функціонального стану. Функціональний стан описує поведінку об’єкта під час виконання певного етапу програми. Кінцевий стан позначає завершення життєвого циклу об’єкта. Перехід між станами зображується у вигляді стрілки з текстовим описом події, яка його спричиняє.

Для об’єкта ігрової кімнати було створено діаграму, зображену на рисунку 3.4. Стан Open відповідає етапу очікування гравців, під час якого будь-який користувач може під’єднатися до кімнати. Після підключення двох гравців відбувається перехід у стан InProgress. Від’єднання гравця до завершення гри спричиняє потрапляння у стани WhiteDisconnected або BlackDisconnected, залежно від кольору його фігур. Після завершення гри кімната переходить у стан Over.

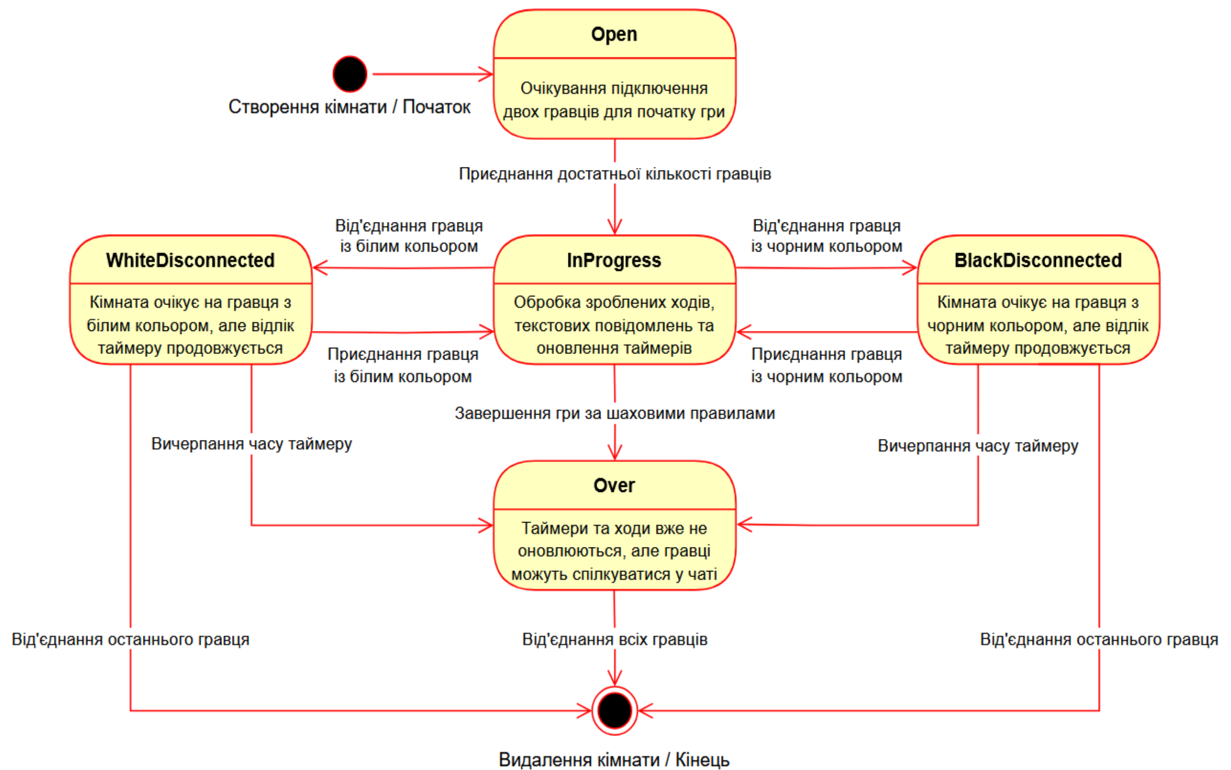

Рисунок 3.4 – State Machine діаграма життєвого циклу ігрової кімнати

Структура Room, код якої міститься у Додатку Б, лістинг Б.5.2, реалізує поведінку ігрової кімнати. Поле Id зберігає унікальний ідентифікатор, за яким гравці можуть підключатися. Поле CreatorName визначає ім’я гравця, що створив кімнату. Поточний стан об’єкта зберігається у полі Status. Покажчик на структуру Hub використовується для реалізації автоматичного видалення кімнати після завершення гри. Канал clientEvents утворює чергу подій, пов’язаних з додаванням та видаленням гравців. Підключені клієнти зберігаються в асоціативному масиві clients. Канал timeout використовується співпрограмою RunRoutine структури Game для надсилання повідомлень про вичерпання часу одним з гравців. Канали move та chat утворюють черги для безпечної обробки зроблених ходів та текстових повідомлень відповідно.

Тип даних Hub (Додаток Б, лістинг Б.5.3) виконує роль глобального сховища створених ігрових кімнат та підключених користувачів. У контексті багатокористувацької взаємодії виникає значна кількість паралельних процесів, які можуть одночасно звертатися до спільних ресурсів структури

Hub. Така ситуація створює ризик виникнення невизначеного стану виконання програми. Для захисту від одночасного доступу до полів структури було використано об’єкт синхронізації м’ютекс. Метод HandleNewConnection відповідає за обробку запитів, пов’язаних з підключенням нових клієнтів. Методи add та remove відповідають за додавання та видалення ігрових кімнат. Описані методи починаються з операції блокування м’ютекса. Після безпечного звертання до полів структури виконується розблокування м’ютекса для забезпечення доступу до полів структури іншим потокам.

3.2 Опис програмного коду клієнтського додатка

Бібліотека React групує компоненти графічного інтерфейсу у вигляді деревоподібної структури даних. Такий підхід дозволяє контролювати процес перемальовування компонентів, уникаючи зайвих апаратних витрат. Кожен графічний компонент описується за допомогою функції, яка повертає розмітку JSX. Розмітка формує зовнішній вигляд компонента, тоді як логіка перемальовування описується усередині самої функції. Для ініціалізації внутрішнього стану компоненту використовується вбудована функція useState, яка повертає поточне значення стану та функцію для оновлення цього значення. Під час оновлення внутрішнього стану виконується перемальовування батьківського та дочірніх компонентів, що забезпечує динамічність вебсторінки.

Комунікація між компонентами відбувається за допомогою механізмів props (властивість) та Context (контекст). Механізм props дозволяє батьківському компоненту передавати у функцію дочірнього необхідні параметри. При передачі даних до дочірніх компонентів через декілька проміжних шарів виникає проблема надмірної вкладеності (props-drilling), яка негативно впливає на продуктивність інтерфейсу через зайві перемальовування. Для уникнення цього явища використовується механізм

контексту. Дочірні компоненти, огорнуті у контекст, можуть отримати доступ до необхідних даних без використання props. Ініціалізація контексту виконується функцією createContext. Для отримання даних у дочірніх компонентах використовується функція useContext.

3.2.1 Компоненти графічного інтерфейсу

Під час розробки клієнтського додатка було спроєктовано дерево компонентів графічного інтерфейсу (рисунок 3.5). Кореневим елементом дерева є компонент Router, який реалізує динамічну маршрутизацію сторінок на основі поточної URL-адреси. До кореневого елементу вкладено три провайдери контекстів. Контекст Theme визначає колір елементів графічного інтерфейсу – темний або світлий. Контекст Authorization автоматично надсилає запити для генерації JWT-токенів доступу до ресурсів серверного додатка та зберігає облікові дані користувача. Контекст Engine configuration встановлює параметри шахового рушія, зокрема рівень складності гри та обмеження на використання апаратних ресурсів.

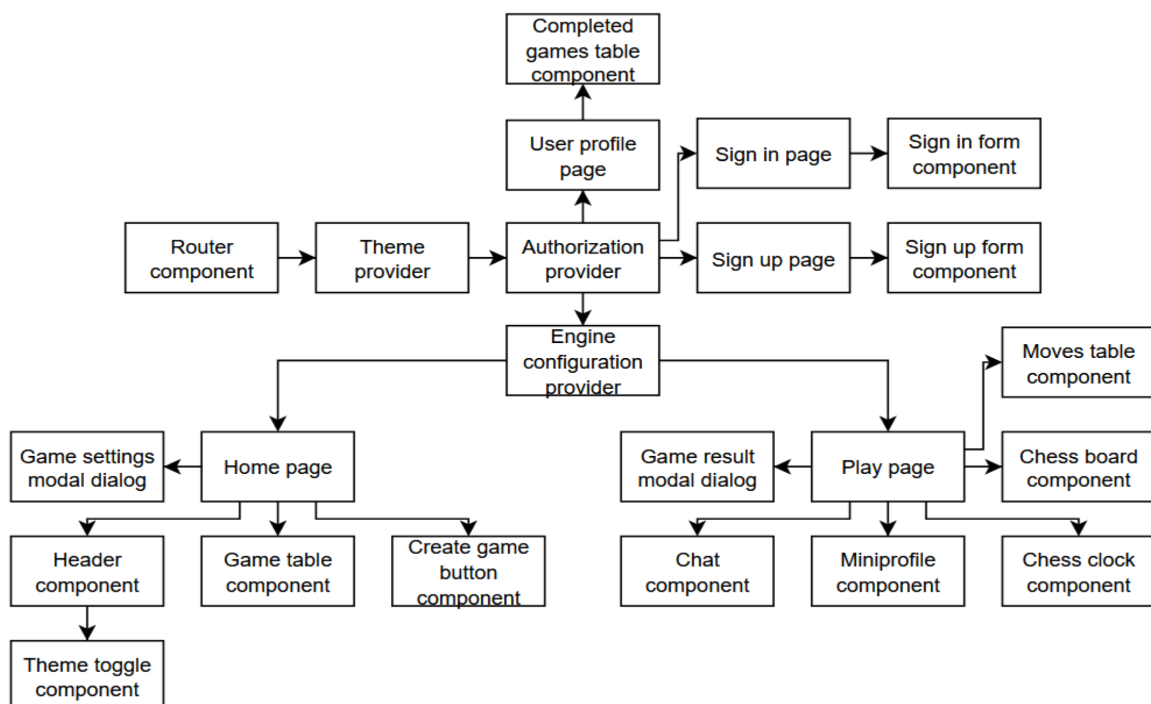

Рисунок 3.5 – Ієрархія компонентів графічного інтерфейсу

Для реалізації динамічної маршрутизації вебсторінок було використано механізм регулярних виразів. Функція route компонента Router викликає процедуру відображення вебсторінки у разі збігу поточної URL-адреси із розробленими регулярними виразами. Якщо збігів не знайдено, повертається текстове повідомлення про помилкову адресу. Зміна адресного рядка ініціює подію popstate, яка обробляється функцією onPathChange. Реєстрація цієї функції як обробника події відбувається на етапі ініціалізації компонента за допомогою методу addEventListener інтерфейсу window. Програмний код компоненту Router наведено у лістингу 3.22.

Лістинг 3.22 – Компонент Router

```
export default function Router() {
    const [currentPath, setCurrentPath] =
useState<string>(window.location.pathname +
window.location.search)
    useEffect(() => {
        const onPathChange = function () {
            setCurrentPath(window.location.pathname +
window.location.search) }
        window.addEventListener("popstate", onPathChange)
        return () => {
            window.removeEventListener("popstate", onPathChange)
        }}, [])
    function route() {
        if (currentPath.match(/^\/$/) ||
            (currentPath.match(/^\/[0-9a-f]{8}-[0-9a-f]{4}-[0-
9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$/))) {
            return <EngineConfProvider>
                {currentPath.length == 1
                    ? <Home />
                    : <Play />}
            </EngineConfProvider>
        } else if (currentPath.match(/^\/player\/[a-zA-Z0-9\-
]{2,}$/)) { return <Profile /> } else if
(currentPath.match(/^\/verify\?action\=((reset)|(registration))\
&token\=[A-Z0-9]+$/)) {
            return <MailVerify />
        } else if (currentPath.match(/^\/signup$/)) {
            return <Signup />
        } else if (currentPath.match(/^\/signin$/)) {
            return <Signin /> }
        return <NotFound /> }
    return <ThemeProvider>
        <AuthProvider> {route()} </AuthProvider>
    </ThemeProvider> }
```

Забезпечення можливості вибору кольорової схеми елементів графічного інтерфейсу сприяє покращенню користувацького досвіду. Для зберігання обраної користувачем схеми використовується локальне сховище браузера (localStorage). Таке сховище дає змогу відновлювати збережені дані під час наступних сеансів користування веб-сервісом. Компоненти дерева графічного інтерфейсу можуть отримати значення кольорової схеми за допомогою провайдера контексту Theme (лістинг 3.23).

Лістинг 3.23 – Провайдер контексту Theme

```
export default function ThemeProvider({ children }: any) {
    const [theme, setTheme] = useState<string>("dark")
    useEffect(() => {
        const themeFromLocal = localStorage.getItem("theme")
        if (!themeFromLocal) localStorage.setItem("theme",
"dark")
        setTheme(localStorage.getItem("theme") === "light" ?
"light" : "dark") }, [])
    return <ThemeContext.Provider value={{ theme, setTheme }}>
        {children}
    </ThemeContext.Provider> }
```

Провайдер контексту авторизації Auth автоматично надсилає HTTP-запити для генерації токенів доступу та оновлення під час завантаження веб-сервісу. Застосування контексту дозволяє уникнути дублювання логіки авторизації у різних частинах кодової бази. Функція refresh надсилає HTTP-запит для отримання JWT-пари. У разі успішної відповіді облікові дані гравця та токен доступу зберігаються в оперативній пам’яті клієнтського додатку. Якщо токен оновлення відсутній або втратив чинність, ініціюється запит на створення гостьового акаунту. У відповідь на запит сервер генерує токен із гостьовим режимом доступу. Код провайдера контексту Auth наведено у лістингу 3.24.

Лістинг 3.24 – Провайдер контексту Auth

```
export default function AuthProvider({ children }: any) {
    const [isReady, setIsReady] = useState<boolean>(false)
    const [accessToken, setAccessToken] = useState<string>("")
```

```
    const [player, setUser] = useState<Player>({
        id: "", username: "", createdAt: "", isEngine: false,
        role: Role.Guest })
    useEffect(() => {
        const getUser = async function () {
            const res = await refresh()
            if (res) {
                setUser({ ...res.player, role: res.role })
                setAccessToken(res.accessToken)
                setIsReady(true)
            } else {
                const player = await guest()
                if (player) { setUser({ ...player.player, role:
player.role })
                    setAccessToken(player.accessToken)
                    setIsReady(true)
            } } }
        getUser() }, [])
    return <AuthContext.Provider value={{ player, setUser,
accessToken, setAccessToken }}> {isReady && children}
    </AuthContext.Provider> }
```

Каскадні таблиці стилів є мовою опису параметрів відображення елементів вебсторінок. Бібліотека React надає вбудовану підтримку файлів із таблицями стилів для покращення зовнішнього вигляду компонентів. Зовнішній вигляд розробленого графічного інтерфейсу зображено на рисунку 3.6.

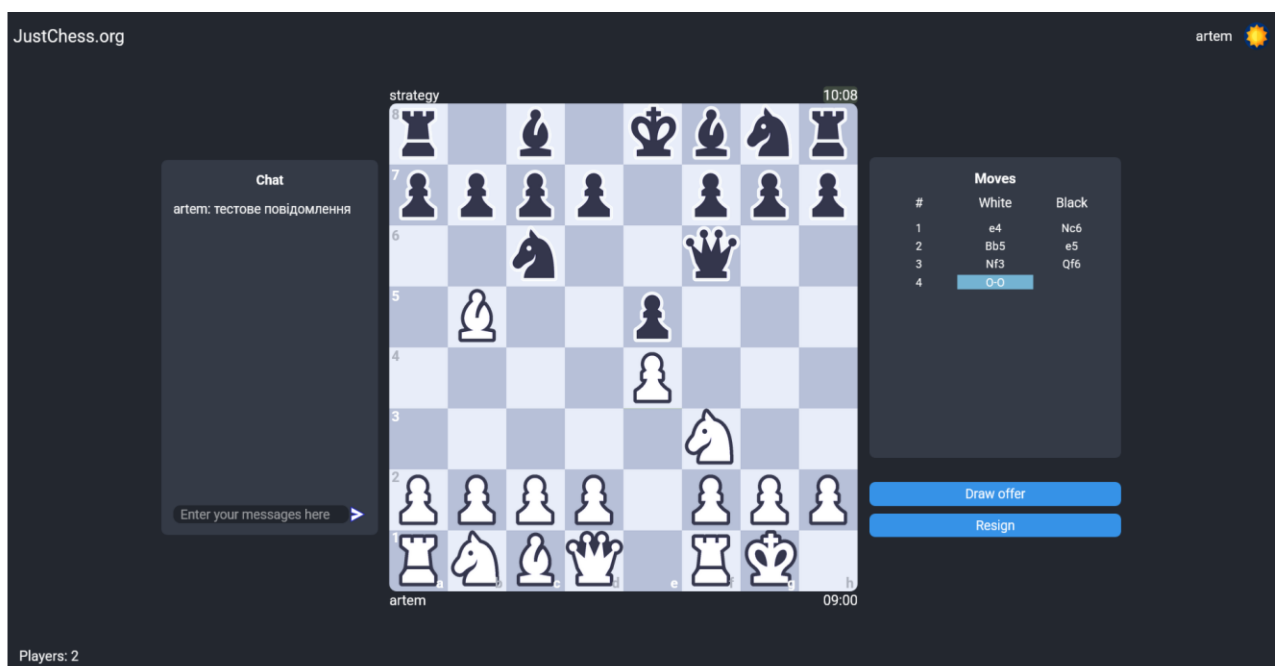

Рисунок 3.6 – Розроблений графічний інтерфейс

3.2.2 Реалізація клієнт-серверної взаємодії

Створення та підтримка з’єднань за протоколом WebSocket реалізується за допомогою вбудованого у програмний інтерфейс браузера класу WebSocket. Механізм обробки вхідних повідомлень базується на парадигмі подійно-орієнтованого програмування. Подія open виникає після завершення ініціалізації двостороннього з’єднання з серверним додатком. Подія error викликається у разі неочікуваного переривання з’єднання. Вхідні повідомлення породжують події типу message. Подія close формується при завершенні з’єднання з боку сервера.

З метою уникнення повторень коду було розроблено клас-обгортку «_WebSocket», який ініціалізує внутрішній об’єкт з’єднання у конструкторі. Методи sendCreateRoom, sendMakeMove та sendChat використовуються для надсилання повідомлень різних типів. У лістингу 3.25 наведено програмний код створення об’єкта з’єднання під час завантаження сторінки за допомогою функції useEffect бібліотеки React.

Лістинг 3.25 – Створення WebSocket-з’єднання

```
useEffect(() => {
        const s = new _WebSocket(url, accessToken)
        s.socket.onmessage = (raw) => {
            const msg = JSON.parse(raw.data)
            handleMessage(msg as Message) }
        s.socket.onopen = () => dispatch({ type:
Action.SET_SOCKET, payload: s })
        s.socket.onclose = () => dispatch({ type:
Action.SET_SOCKET, payload: null })
        return () => s.close()
}, [])
```

Надсилання HTTP-запитів будується на асинхронній моделі мови TypeScript. Ключове слово async дозволяє виконувати трудомістку функцію без блокування подальшого виконання програми. Глобальна функція fetch створює асинхронні запити до серверного додатку.

4 ТЕСТУВАННЯ ТА ВИМІРЮВАННЯ ПРОДУКТИВНОСТІ РОБОТИ ВЕБ-СЕРВІСУ

4.1 Інструменти автоматизованого тестування мови Go

Автоматизоване тестування забезпечує надійність та ефективність роботи веб-сервісу, мінімізуючи ризик виникнення помилок та збоїв. Процес тестування полягає у створенні функцій для перевірки працездатності основної логіки додатку. Зазвичай, програмні засоби тестуються з використанням методів модульного, інтеграційного, системного або приймального тестування. Модульне тестування є процесом перевірки правильності роботи окремих частин кодової бази. Організація серверного додатку за допомогою пакетів дозволила прискорити написання та налагодження модульних тестів.

Стандартна бібліотека мови Go містить пакет testing, який надає необхідні інструменти для написання функцій тестування та вимірювання продуктивності коду. Тестові набори для конкретного пакету зберігаються у файлі з назвою «name_test», де name – це ім’я пакета, що тестується. Тип даних testing.T керує станом виконання функції тестування. Методи Fatal та Fatalf переривають виконання тесту та виводять повідомлення про помилку.

Бенчмарк є процедурою тестування продуктивності роботи окремих функцій. Зазвичай, бенчмарки зберігаються в одному файлі із функціями тестування. Результатом виконання бенчмарку є відображення часу роботи функції у наносекундах, обсяг використаної оперативної пам’яті у байтах та кількість операцій динамічного виділення пам’яті. Ці показники дозволяють виявити місця обмеження загальної продуктивності веб-сервісу. Для запуску тестів та бенчмарків використовується утиліта командного рядка go test, яка автоматично виконує усі тестові файли проєкту та відображає відомості про їх виконання.

4.2 Результати модульного тестування

На етапі проєктування веб-сервісу не було можливим передбачити всі тонкощі, які виявилися лише на етапах написання та налагодження коду. При виникненні труднощів, логіка та структура проєкту зазнавали модифікацій. Для покриття створених програмних компонентів тестовими наборами було реалізовано дванадцять функцій, результати виконання яких зображено на рисунку 4.1. Успішне проходження тестів демонструє стабільність функціонування веб-сервісу. Загальний час виконання тестових наборів становить 6,2 секунди, що дозволяє швидко перевіряти правильність роботи додатку під час внесення змін до кодової бази.

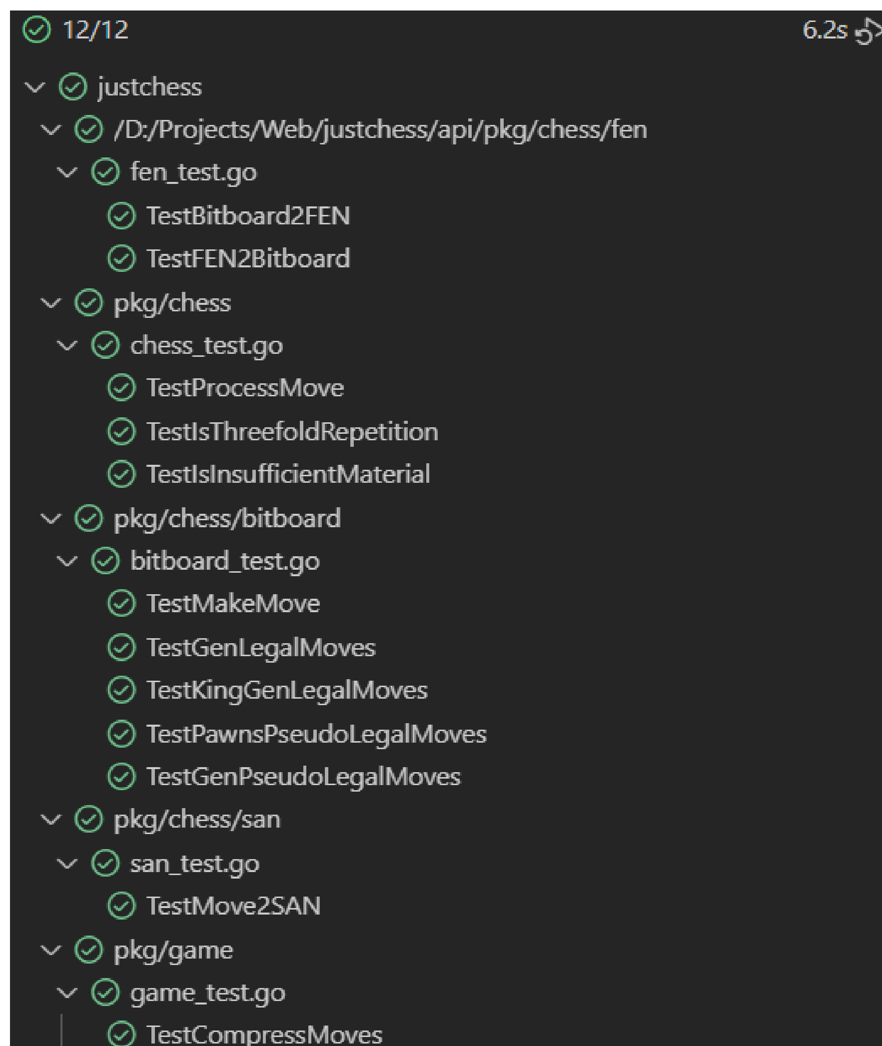

Рисунок 4.1 – Результати виконання функцій тестування

Функція TestGenLegalMoves виконує шість тестових наборів, які охоплюють процес генерації допустимих ходів для всіх типів фігур. Функція TestMakeMove ініціалізує об’єкт шахової дошки та послідовно виконує вісім ходів з метою перевірки правильності переміщення фігур. Генерація псевдодопустимих ходів є важливим етапом виконання програми, тому для тестування її реалізації було розроблено функцію TestGenPseudoLegalMoves. Функція TestGenKingLegalMoves виконує тестування генерації допустимих ходів короля, зокрема перевіряє неможливість переміщення короля на квадрати, що перебувають під атакою. Функція TestGenPawnsPseudoLegalMoves тестує процедуру генерації спеціального ходу перетворення пішака. Функція TestProcessMove забезпечує тестове покриття процесу обробки зроблених ходів, включаючи оновлення прав рокірування та цільового квадрату для взяття на проході. Функція TestIsThreefoldRepetition перевіряє логіку виявлення потрійного повторення позиції з метою оголошення нічиєї. Функція TestIsInsufficientMaterial тестує процедуру визначення достатньої для продовження гри кількості фігур на дошці. Тестування процесів кодування та декодування нотації Форсайта-Едвардса для компактного уявлення шахової позиції відбувається за допомогою функцій TestBitboard2FEN та TestFEN2Bitboard. Функція TestMove2SAN перевіряє процедуру кодування зроблених ходів у формат стандартної алгебраїчної нотації. Функція TestCompressMoves тестує алгоритм стиснення масиву зроблених ходів для забезпечення надійного зберігання у базі даних.

Процес комунікації за протоколом WebSocket складно охопити автоматизованими тестовими наборами через необхідність паралельної обробки декількох клієнтів. Використані інструменти тестування надають обмежену підтримку протоколу WebSocket. Це зумовило застосування методу ручного тестування, яке проводилось у сучасних браузерах: Firefox 128, Google Chrome 131, Microsoft Edge 136 та Samsung Internet 19. Під час роботи веб-сервісу не було знайдено помилок або збоїв.

4.3 Результати вимірювання продуктивності розроблених функцій

Для вимірювання продуктивності виконання основної логіки додатку було розроблено дев’ять тестів продуктивності. Тестування проводилося з використанням утиліти go test з параметрами «-bench=.» та «-benchmem» на процесорі Intel i7-10750H із 64-х розрядною архітектурою та базовою частотою 2,6 ГГц. Застосування ефективних алгоритмів та структур даних при реалізації шахової логіки забезпечило показники продуктивності, представлені на рисунку 4.2. Найбільш оптимізованими виявилися функції MakeMove та GenPseudoLegalMoves. Це пов’язано з відсутністю динамічного виділення пам’яті під час їх виконання. Функція GenLegalMoves винокується за більший проміжок часу, оскільки для зберігання довільної кількості можливих ходів необхідно створювати динамічні масиви.

```
$ go test ./... -bench=. -benchmem
goos: windows
goarch: amd64
pkg: justchess/pkg/chess
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkProcessMove-12          4220802          276.3 ns/op        256 B/op        2 allocs/op
PASS
ok      justchess/pkg/chess     1.755s
goos: windows
goarch: amd64
pkg: justchess/pkg/chess/bitboard
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkMakeMove-12                    70518960       16.90 ns/op          0 B/op        0 allocs/op
BenchmarkGenLegalMoves-12                 473390        2299 ns/op        416 B/op       17 allocs/op
BenchmarkGenKingLegalMoves-12           31914892       34.53 ns/op          8 B/op        1 allocs/op
BenchmarkGenPawnsPseudoLegalMoves-12     8881086       128.2 ns/op         56 B/op        3 allocs/op
BenchmarkGenPseudoLegalMoves-12          1580797       767.3 ns/op          0 B/op        0 allocs/op
PASS
ok      justchess/pkg/chess/bitboard    7.010s
goos: windows
goarch: amd64
pkg: justchess/pkg/chess/fen
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkBitboard2FEN-12         2404602          497.8 ns/op        328 B/op        8 allocs/op
BenchmarkFEN2Bitboard-12         4225862          265.7 ns/op        256 B/op        2 allocs/op
PASS
ok      justchess/pkg/chess/fen 3.269s
PASS
ok      justchess/pkg/chess/san 0.168s
goos: windows
goarch: amd64
pkg: justchess/pkg/game
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkCompressMoves-12       83004771           13.56 ns/op          8 B/op        1 allocs/op
PASS
ok      justchess/pkg/game      1.334s
```

Рисунок 4.2 – Результати виконання тестів продуктивності

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено, протестовано та розгорнуто багатокористувацький веб-сервіс для гри в шахи. Програмні компоненти реалізовано на сучасних мовах програмування Go та TypeScript у середовищі розробки Visual Studio Code. Динамічні вебсторінки клієнтського додатку надають зручний інтерфейс користувача, який відображає елементи гри. Серверний додаток обробляє запити гравців, підтримує двосторонній обмін даними за протоколом WebSocket та ефективно реалізує правила гри.

Розроблений веб-сервіс забезпечує швидку генерацію та обробку допустимих ходів. Під час вимірювання продуктивності були отримані наступні показники: генерація допустимих ходів відбувається за 2299 наносекунд та займає 416 байтів оперативної пам'яті, обробка зробленого ходу відбувається за 16,84 наносекунди та не потребує виділення пам'яті, генерація псевдодопустимих ходів займає 753,5 наносекунд та не потребує виділення пам'яті. Розроблений додаток можна використовувати для вивчення правил гри та організації турнірів з шахів серед здобувачів університету. Середовище виконання мови Go підтримує виконання на всіх сучасних операційних системах.

Для порівняння, сучасний шаховий веб-сервіс Lichess орієнтований на розгортання лише у Linux-системах та споживає значну кількість апаратних ресурсів під час виконання. Велика кількість використаних технологій збільшує фінальний розмір серверного додатку Lichess близьким до 120 мегабайтів, а розроблений в даній роботі додаток займає 14 мегабайтів.

Проєкт є повністю робочим та виконує всі поставлені завдання. Перспективними напрямами подальшого удосконалення проєкту є впровадження штучного інтелекту для аналізу партій та розширення можливостей комунікації гравців шляхом реалізації форумів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бєліков А. О., Ярошевич Р. О. Розробка веб-сервісу для гри в шахи. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : зб. тез доп. п’ятнадцятої міжнар. наук.-техн. конф., м. Баку – Харків – Жиліна, 24 – 25 квітня. 2025 р. 144 с. С. 14.
2. Wang V., Salim F., Moskovits P. The Definitive Guide to HTML5 WebSocket. New York City: Apress, 2013. 208 p.
3. Lombardi A. WebSocket: Lightweight Client-Server Communications. Sebastopol: O`Reilly Media, 2015. 144 p.
4. Ресурс для вивчення алгоритмів та структур даних, що використовуються при розробці шахових додатків. URL: https://www.chessprogramming.org.
5. Go (мова програмування). URL: https://uk.wikipedia.org/wiki/Go_(мова_програмування).
6. Plotka B. Efficient Go: Data-Driven Performance Optimization. O`Reilly Media, 2022. 499 p.
7. Harsanyi T. 100 Go Mistakes and How to Avoid Them. Manning, 2022. 384 p.
8. Комунікуючі послідовні процеси. URL: https://uk.wikipedia.org/wiki/Комунікуючі_послідовні_процеси.
9. Donavan A., Kernighan B. The Go Programming Language. Boston: Addison-Wesley Professional, 2015. 400 p.
10. Cox-Buday K. Concurrency in Go. Sebastopol: O`Reilly Media, 2017. 238 p.
11. Baumgartner S. TypeScript Cookbook: Real World Type-Level Programming 1st Edition. Sebastopol: O`Reilly Media, 2023. 416 p.
12. Freeman A. Essential TypeScript 5, Third Edition. Manning, 2023. 568 p.

13. Sakhniuk M., Boduch A. React and React Native: Build cross-platform JavaScript and TypeScript apps for the web, desktop, and mobile. Packt Publishing, 2024. 508 p.

14. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. O`Reilly Media, 2017. 312 p.

15. Booz R., Fritchey G. Introduction to PostgreSQL for the data professional. Redgate Books, 2025. 452 p.