ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «**Веб застосунок настільної гри «монополія» на основі JavaScript**»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ілля ОМЕЛЯНЧУК
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Ілля ОМЕЛЯНЧУК
група КНДМ-61

Керівник: Олександр ЗВЕНІГОРОДСЬКИЙ
науковий ступінь, вчене звання д.т.н., професор

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Омелянчук Іллі Володимирович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «**Веб застосунок настільної гри «монополія» на основі JavaScript**»

керівник кваліфікаційної роботи Олександр ЗВЕНІГОРОДСЬКИЙ д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література з питань, пов'язаних з розробкою веб-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1. Аналіз вимог та розробка концепту.
   4.2. Дослідження основних технологій для розробки.
   4.3. Розробка веб-застосунку гри «Монополія».
5. Перелік графічного матеріалу: *презентація*

5.1 Мета роботи, об`єкт та предмет дослідження.
5.2 Робота функцій веб-сайту.
5.3 Приклади програмного коду.
5.4 Результат роботи створеного веб-сайту
6. Дата видачі завдання «16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.24	виконано
2	Аналіз методів розвязування задачі	30.09-11.10.24	виконано
3	Застосування алгоритмів на практиці	14.10-25.10.24	виконано
4	Розробка серверної частини	28.10-08.11.24	виконано
5	Розробка клієнтської частини	11.11-22.12.24	виконано
6	Головні компоненти створеної системи	25.11-29.11.24	виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-06.12.24	виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.24	виконано

Здобувач вищої освіти _____ Ілля ОМЕЛЯНЧУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олександр ЗВЕНІГОРОДСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Магістерська робота містить 102 сторінки, включаючи 9 рисунків, 13 джерел.

Наукове завдання: Розробка веб-застосунку для гри "Монополія" з підтримкою реального часу та інтерактивними можливостями для користувачів на основі сучасних веб-технологій.

Мета роботи: Підвищити реалістичність ігрового процесу з використанням технологій JavaScript для обміну даними в реальному часі.

Об'єкт дослідження: Процес розробки клієнт-серверних веб-застосунків для інтерактивних ігор.

Предмет дослідження: Інструменти та технології для розробки клієнтської та серверної частини гри з функціями управління станом гри, обміну даними та обробки логіки.

Методи дослідження: Аналіз існуючих веб-технологій для реалізації інтерактивної взаємодії, архітектура серверної частини з Node.js і Express, а також клієнтської частини за допомогою Vue.js. Реалізовано протокол обміну даними через WebSocket для забезпечення миттєвих оновлень гри.

Короткий зміст роботи:

У першому розділі проведено дослідження предметної області та аналіз вимог до гри "Монополія". Визначено функції веб-застосунків і розглянуто існуючі рішення.

Другий розділ присвячено огляду технологій розробки: JavaScript, Express.js, Vue.js, а також інструментів для реального часу (WebSocket).

Третій розділ розробку гри: створення серверної частини для обробки логіки гри, а також клієнтської частини з інтерактивними компонентами, екранами та управлінням станом. Забезпечено обмін даними між клієнтом і сервером через API та WebSocket. Результат роботи демонструє повноцінний прототип гри, що дозволяє користувачам грати в реальному часі з можливістю взаємодії.

Ключові слова: веб-застосунок, JavaScript, Vue.js, Express.js, WebSocket, реальний час, клієнт-серверна архітектура, розробка ігор, багатокористувацька гра.

ABSTRACT

Master's Thesis contains 102 pages, including 9 figures and 13 references.

Scientific Task: Development of a web application for the game "Monopoly" with real-time support and interactive features for users based on modern web technologies.

Objective: To enhance the realism of the gameplay using JavaScript technologies for real- time data exchange.

Object of Study: The development process of client-server web applications for interactive games.

Subject of Study: Tools and technologies for developing the client and server parts of the game with game state management, data exchange, and logic processing features.

Research Methods: Analysis of existing web technologies for implementing interactive interaction, server-side architecture using Node.js and Express, and client-side development with Vue.js. A data exchange protocol using WebSocket was implemented to ensure instant game updates.

Summary of the Thesis:

The first chapter investigates the subject area and analyzes the requirements for the game "Monopoly." The functions of web applications are defined, and existing solutions are reviewed.

The second chapter focuses on the review of development technologies: JavaScript, Express.js, Vue.js, and real-time tools (WebSocket).

The third chapter covers the development of the game: creating the server side for game logic processing and the client side with interactive components, screens, and state management. Data exchange between client and server is ensured through API and WebSocket. The result demonstrates a full-fledged game prototype that allows users to play in real-time with interactive features.

Keywords: web application, JavaScript, Vue.js, Express.js, WebSocket, real-time, client-server architecture, game development, multiplayer game.

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій та веб-розробки, створення інтерактивних веб-застосунків стало важливою складовою розвитку цифрових продуктів. Веб-ігри, зокрема багатокористувацькі онлайн-ігри, набули значної популярності завдяки своїй доступності та можливості взаємодії між гравцями в реальному часі. Однією з таких ігор є «Монополія» – класична настільна гра, яка забезпечує інтерактивність і стратегічне мислення учасників. Перехід цієї гри в онлайн-формат відкриває нові можливості для вдосконалення досвіду користувачів, зокрема, завдяки використанню сучасних веб-технологій.

Метою цієї роботи є підвищення реалістичного ігрового процесу з використанням технологій JavaScript для обміну даними в реальному часі. Основними завданнями якого є створення серверної та клієнтської частини застосунку з використанням таких технологій, як Vue.js, Express.js, а також реалізація обміну даними через WebSocket для забезпечення реального часу.

Актуальність теми роботи обумовлена високим попитом на багатокористувацькі онлайн-ігри та необхідністю створення ефективних та зручних інструментів для їх розробки. Сучасні веб-технології дозволяють створювати інтерактивні системи, які підтримують реальний час.

У роботі розглянуто основні етапи розробки веб-застосунку для гри *Монополія*, зокрема вибір технологій, створення структури серверної та клієнтської частини, а також реалізація необхідних функцій для забезпечення ігрового процесу в реальному часі. В результаті дослідження є розроблений повноцінний прототип веб-застосунку, який дає змогу користувачам грати в гру «Монополія» онлайн, зберігаючи основні принципи та правила класичної настільної версії.

Вступна частина роботи підкреслює важливість дослідження і розвитку веб-технологій для створення інтерактивних ігор, що стають все більш популярними серед користувачів по всьому світу. У наступних розділах будуть розглянуті технології, що використовуються для розробки такого застосунку, а також продемонстровано процес реалізації цього проекту.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття та функції сайту

Веб-сайти — це набір сторінок, які розташовані на сервері та доступні через мережу Інтернет. Цей веб-сайт може містити текст, зображення, відео, аудіо та інші компоненти. Залежно від цілей і потреб користувачів веб-сайт може виконувати різні завдання.

Основні ідеї веб-сайту:

Мова розмітки гіпертексту, яка використовується для створення структур веб-сторінок, називається HTML.

CSS є мовою стилів, яка використовується для створення веб-сторінок, яка дозволяє змінювати елементи, такі як кольори, шрифти, розміри та розташування елементів.

JavaScript — це скриптова мова програмування, яка використовується для створення інтерактивних ефектів на веб-сторінках, таких як анімація та перехід на інші веб-сторінки.

CMS — це система управління вмістом, яка дозволяє створювати та оновлювати веб-сайти без програмування.

SEO — це процес оптимізації веб-сайту для підвищення його рейтингу в пошукових системах.

Основні функції веб-сайту: Одна з основних функцій веб-сайту — надання користувачам інформації про різні теми. Інформаційні веб-сайти можуть містити рейтинги, відгуки, статті, новини, довідники, каталоги товарів і послуг тощо.

Інформаційні веб-сайти можуть бути спеціалізованими, тобто займатися певною темою, наприклад, новинами або сайтами, присвяченими туризму, культурі, науці, спорту та іншим областям. Інформаційні веб-сайти також можуть бути

універсальними, що означає, що вони можуть містити інформацію на широкий спектр тем.

Також вони мають різний рівень деталізації та складності. На одних веб-сайтах можна знайти загальну інформацію з коротким описом теми, тоді як на інших веб-сайтах можна знайти більш детальну інформацію, таку як фото, відео та графіки.

Загалом інформаційна функція є важливою для бізнесу та інших організацій, оскільки вони можуть використовувати веб-сайти для просування своїх товарів і послуг, розміщення контактної інформації, розміщення новин і оголошень і зв'язатися з клієнтами. Крім того, веб-сайти, які мають інформаційні функції, можуть використовуватися для навчання та підвищення обізнаності населення про різноманітні теми.

Однією з основних функцій веб-сайту є комерційна функція, яка полягає в продажу товарів і послуг через Інтернет. Невеликі компанії та великі онлайн-магазини можуть мати комерційні веб-сайти.

Веб-сайти компаній можуть включати різні функції, щоб допомогти залучити та утримувати клієнтів. Наприклад, виробники товарів та послуг можуть збільшити продажі та залучити нових клієнтів за допомогою функцій, таких як корзина для зберігання товарів, пошук по сайту, система фільтрів для зручного вибору товарів, рекомендації для покупок, рейтинги товарів і відгуки покупців.

Що більше, вони можуть приймати оплату різними способами, такими як картки, готівка, електронні гаманці, онлайн-банкінг і інші платіжні системи, що дозволяє клієнтам вибирати зручний для них спосіб.

Теж можуть бути розроблені для продажу різних товарів і послуг, наприклад електроніки, одягу, харчових продуктів, туристичних послуг та багато іншого. Зараз вони також можуть бути спеціалізованими на певному товарі, наприклад, електроніці, косметикі та меблів.

Це також важливо для бізнесу, оскільки дозволяє продавати продукти та послуги в Інтернеті, що може збільшити продажі та прибуток.

Інтерактивна функція веб-сайту дозволяє відвідувачам взаємодіяти з ним, спілкуватися з ним і взаємодіяти з його елементами. Інтерактивність можна досягти різними способами, наприклад формами заповнення, елементами керування, інтерактивними графіками та анімацією, серед інших.

Наприклад, інтерактивна функція може містити форму, на яку можна заповнити замовлення, підписатися на новини або зв'язатися з підтримкою клієнтів. Форма заповнення може бути розроблена таким чином, щоб відвідувачі могли легко вводити свої дані та гарантувати точність даних.

Крім того, елементи керування, такі як кнопки, покажчики та випадаючі списки, роблять сторінку інтерактивною, дозволяючи користувачам взаємодіяти з нею та змінювати її вміст.

Інтерактивні графіки та анімація можуть допомогти клієнтам зрозуміти складну інформацію, надати відповіді на запитання та розповісти про товари та послуги. Наприклад, інтерактивні графіки можуть демонструвати процес виробництва або як працює новий продукт. Анімація може навчати клієнтів користуватися товаром або пояснювати складну інформацію про послугу.

Розважальна функція - це функція веб-сайту, яка призначена для того, щоб відвідувачі могли розважатися та розслабитися. Він може містити різні інтерактивні компоненти, такі як ігри, відео, музика, фотографії та анімація.

Відео-контент, який включає відеоблоги, рекламні ролики, відеоігри, уроки та інше, є одним із найпоширеніших видів розваг. Відео можна використовувати для навчання, розваг і інформування клієнтів про продукти та послуги.

Ще однією популярною функцією, яку пропонують веб-сайти, є інтерактивні ігри. Ігри можуть бути розроблені, щоб задовольнити різні вікові групи та інтереси. Ігровий контент може включати прості ігри на час, складні стратегічні ігри або ігри, у яких гравці взаємодіють один з одним.

Крім того, музика та аудіо контент можуть бути частиною розважальних функцій веб-сайтів. Відвідувачі мають можливість відтворювати музику в режимі онлайн або завантажувати її на свої пристрої. Аудіо-підкасти, радіо, аудіо-книги та інші види аудіоконтенту також можуть бути доступні.

Розважальні функції веб-сайтів можуть привернути увагу користувачів і збільшити відвідуваність. Вони також можуть залучати більше людей і створювати позитивне сприйняття бренду, що робить їх важливими інструментами для маркетингу та реклами.

Освітня функція — це функція веб-сайту, яка призначена для того, щоб надати відвідувачам можливість отримати освіту. Вона може включати навчальний контент, курси, інформаційні матеріали, тести та інші ресурси, які допомагають вивчати нові предмети або поглиблювати свої знання в певній галузі.

Текстові матеріали, відео уроки, інтерактивні підручники, віртуальні лабораторії, тести та інші навчальні ресурси можуть бути частиною навчального матеріалу. Курси можуть бути розроблені для різних рівнів складності, щоб задовольнити потреби різних типів відвідувачів, від новачків до досвідчених.

Освіта може бути використана для навчання багатьох предметів, наприклад математики, іноземних мов, історії та науки. Крім того, вона може допомогти вам розвинути професійні навички, такі як управління проектами, маркетинг, фінанси та інші.

Крім того, це може бути корисним для вчителів, студентів і фахівців, які шукають нові можливості для покращення своїх знань і навичок. Вона може допомогти відвідувачам знайти необхідні дані для вирішення питання, яке вони досліджують.

Ймовірно, це може бути корисним для компаній і компаній, які хочуть надати своїм співробітникам можливість навчатися та розвивати свої професійні навички.

Соціальна функція веб-сайту полягає в тому, щоб користувачі могли спілкуватися та взаємодіяти один з одним. Це можна зробити за допомогою багатьох функцій, наприклад коментарів, форумів, чатів, спільнот, можливостей ділитися контентом та інших.

Коментарі можуть бути корисними для взаємодії з користувачами, які хочуть обговорити конкретний контент, поділитися своїми думками та отримати відповіді на свої запитання від інших користувачів або авторів контенту.

Форуми дають людям можливість обговорювати різні теми та обмінюватися думками з іншими. Вони можуть бути спрямовані на різні сфери, такі як технології, мистецтво та наука, тощо.

Чат може бути корисним для спілкування в режимі реального часу між користувачами. Використовуйте його для спілкування з друзями, партнерами по бізнесу або навіть для вирішення проблем з підтримкою користувачів.

Користувачі, які мають спільні інтереси або хобі, можуть об’єднатися за допомогою спільнот. Вони чудово підходять для обговорення спільних ідей, планування заходів і зустрічей.

Можливість ділитися контентом може дозволити користувачам ділитися цікавими та корисними матеріалами, такими як новини, фотографії, відео та інформаційні статті, серед іншого.

Маркетингова функція веб-сайту полягає в тому, щоб надати можливість просувати товари та послуги на ринку, підвищувати обізнаність про бренд, залучати нових клієнтів і зміцнити лояльності існуючих клієнтів.

Це можна досягти за допомогою різноманітних маркетингових стратегій. Основними з них є реклама, контент-маркетинг, SEO (пошукова оптимізація), соціальні медіа-маркетинг і email-маркетинг.

Реклама може використовуватися для просування товарів і послуг на ринку. На веб-сайті можна розмістити банери та текстові оголошення, а також використовувати різні платформи для реклами, такі як Google AdWords або Facebook Ads.

Пошукова оптимізація, або SEO, може бути використана для підвищення видимості веб-сайту в пошуковиках. Це включає використання посилань на веб-сайти, оптимізацію метатегів і заголовків, включення ключових слів і фраз у контент сайту тощо.

Просування товарів і брендів у соціальних мережах (SMM) може бути використано. Це можна досягти шляхом створення брендових сторінок на Facebook, Twitter, Instagram та інших соціальних мережах.

Контент-маркетинг може бути використаний для підвищення зацікавленості клієнтів у товарах і послугах шляхом створення високоякісного та захоплюючого контенту, такого як статті, відео та блоги, серед іншого.

Адміністрація веб-сайту гарантує ефективне та безперебійне функціонування. Вона виконується адміністратором сайту або командою, яка відповідає за підтримку та технічне забезпечення сайту.

Основні обов'язки адміністративної функції веб-сайту включають наступне:

- Технічне забезпечення та підтримка: це включає гарантію безперебійної роботи веб-сайту, оперативне вирішення технічних проблем і оновлення програмного забезпечення.
- Керування контентом: це означає додавання та оновлення матеріалів на веб-сайти, таких як статті, відео, зображення та інші.
- Управління користувачами: це стосується реєстрації нових користувачів, контролю за правами доступу, відновлення паролів та інших елементів управління користувачами.
- Аналітика та статистика: це включає збір та аналіз статистичних даних про поведінку користувачів, джерело трафіку, відвідуваність веб-сайту та інші фактори.
- Безпека: це включає захист сайту від хакерських атак і вірусів, а також резервне копіювання даних.
- Планування та управління проектами: це включає в себе планування та управління різними проектами, пов'язаними з веб-сайтом, включаючи внесення змін до дизайну, впровадження нових функцій, інтеграцію з іншими системами тощо.

Для забезпечення успішної та ефективної роботи веб-сайту ці елементи адміністрування веб-сайту надзвичайно важливі. Регулярні оновлення та удосконалення необхідні для адміністративної функції, щоб забезпечити найкращі результати роботи веб-сайту та задовольнити потреби користувачів.

Залежно від бажань власника та користувачів веб-сайту він може мати одну або кілька з цих функцій. Створення веб-сайтів, які не тільки забезпечують потребу

користувачів, але й є простими у використанні, має вирішальне значення. Крім того, важливо пам’ятати про те, наскільки важливо оптимізувати веб-сайти для пошукових систем, щоб забезпечити високу відвідуваність і успіх в Інтернеті.

Веб-дизайн зараз відрізняється від попереднього, оскільки більше уваги приділяється дотриманню стандартів, покращенню функціональності та покращенню користувальницького досвіду. Веб-розробка загалом складається з трьох основних етапів: розробка серверної частини, клієнтської частини та зв’язку між ними. У цій статті розглядаються основні сучасні методи створення веб-інтерфейсів:

Responsive Web Design (RWD): цей підхід дозволяє створювати веб-сайти, які адаптуються до мобільних і десктопних пристроїв. RWD підвищує зручність використання веб-сайтів на різних пристроях за допомогою медіа-запитів і гнучкої сітки (flexbox або grid).

Progressive Web Apps (PWA): цей метод поєднує можливості мобільних додатків і веб-сайтів, щоб надати кращий користувальницький досвід і збільшити швидкість роботи веб-додатка. PWA дозволяють працювати офлайн, дозволяють додавати веб-додатки на домашній екран і інформувати користувачів про нові функції.

Single Page Applications (SPA): цей метод дозволяє створювати веб-додаток, який не змінюється під час переходу між сторінками, а змінює лише відображаючий вміст. Це підвищує швидкість роботи веб-додатка та забезпечує більш інтерактивний досвід користувача. SPA роблять розробку та підтримку веб-додатків простішою за допомогою фреймворків, таких як Angular, React і Vue.js.

У цьому методі немає серверної архітектури, що означає, що вам не потрібно керувати власним сервером. Замість цього хмарні сервіси, такі як Amazon Web Services або Microsoft Azure, містять функції сервера. Це підвищує безпеку, масштабованість і інфраструктурні витрати.

Microservices: цей метод розділяє веб-додаток на окремі мікрослужби, кожна з яких виконує свою функцію. Це збільшує масштабованість, забезпечує більш гнучке керування веб-додатками та зменшує час відповіді.

Mobile-First Design: це метод створення веб-сайту, який орієнтований на мобільні пристрої. Це означає, що дизайн і функції веб-сайту спочатку розробляються для мобільних пристроїв, а потім адаптуються до перегляду на десктопі. Це покращує досвід використання веб-сайту на мобільних пристроях.

1.2 Огляд існуючих веб-сайтів

Необхідно знайти та проаналізувати вже існуючі веб-сайти, які схожі на тему цього магістерського проекту, перед початком розробки та дотриманням певних вимог.

Натхненний класичним настільною грою Монополія, перший розглянутий аналог richup (рис1.1) - це онлайн-гра. Вона дозволяє людям брати участь у віртуальній торгівлі нерухомістю з метою побудувати фінансову імперію, купуючи місця по всьому світу, такі як в Америці, Італії, Китаї та Франції.

Ключові особливості:

Ігрова механіка: Кожен гравець починає з $1500 і кидає кубики, щоб рухатися по дошці. Гравці можуть купувати землю, на якій вони висаджуються, а гравці, які висаджуються на власних ділянках, повинні платити орендну плату власнику. Мета полягає в тому, щоб придбати всі будинки та готелі в одній колірній групі, що дозволить їм модернізувати та підвищити ціни на оренду.

Багатокористувацька гра: Richup.io підтримує до чотирьох гравців, що робить гру ідеальною для гри з друзями або новими людьми.

Доступність: Гра доступна через веб-браузер без встановлення або реєстрації, що робить гру простою та швидкою.

Візуальні та тематичні варіації: Гра має оригінальний дизайн і теми (рис. 1.2), які відрізняють її від оригінальної, незважаючи на те, що вона відповідає базовому формату Монополії.

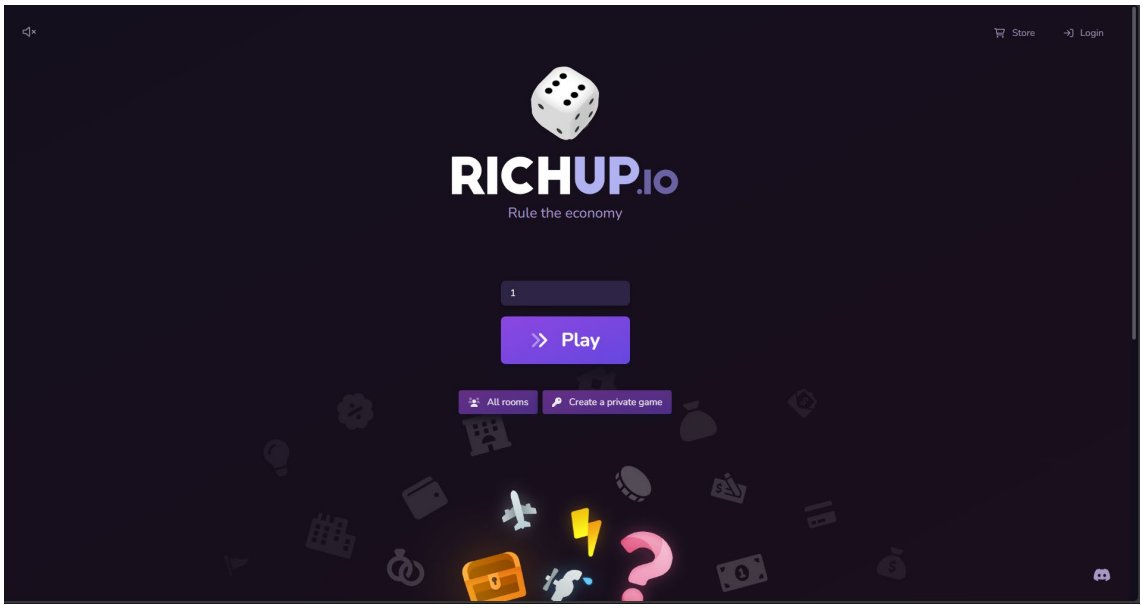

Рисунок 1.1 – Початковий вигляд веб-сайту richup

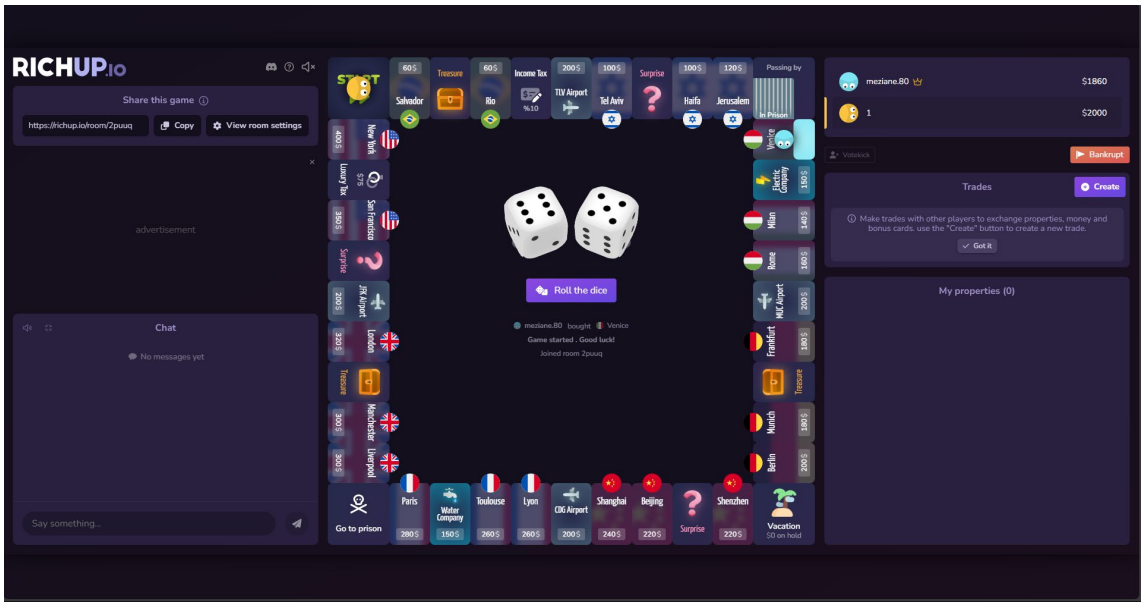

Рисунок 1.2 – Процес гри в гру «Монополія»

PlayMonopolyOnline — це другий аналог за темою. Це інтерактивний веб-сайт (рис. 1.3), на якому користувачі можуть грати в класичну настільну гру Монополія в цифровому вигляді. На цій платформі гравці можуть грати в матчі з двох або чотирьох гравців.

Кожен учасник може вибрати свого персонажа, кидати кубики і переміщатися по дошці, щоб купувати нерухомість, будувати будинки та планувати, як витрачати гроші, щоб збанкрутувати своїх опонентів. Ігровий процес схожий на традиційний

досвід «Монополії», де гравці прагнуть стати найбагатшими, використовуючи стратегічні угоди, сплачуючи орендну плату та купуючи нерухомість. Фізична версія гри зберігає основні функції, такі як скриня громади та карти шансів, але вона також містить правила, такі як іпотека нерухомості (рис. 1.4).

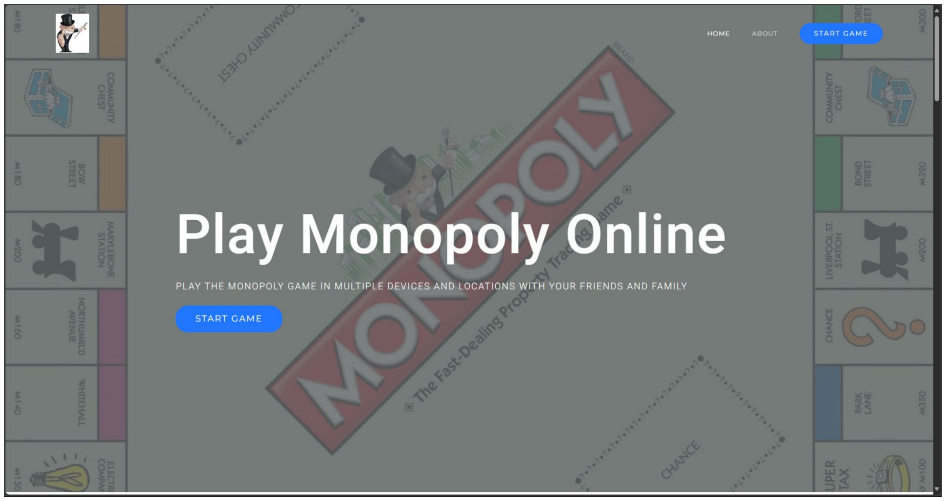

Рисунок 1.3 - Початкова сторінка веб-сайту playmonopolyonline

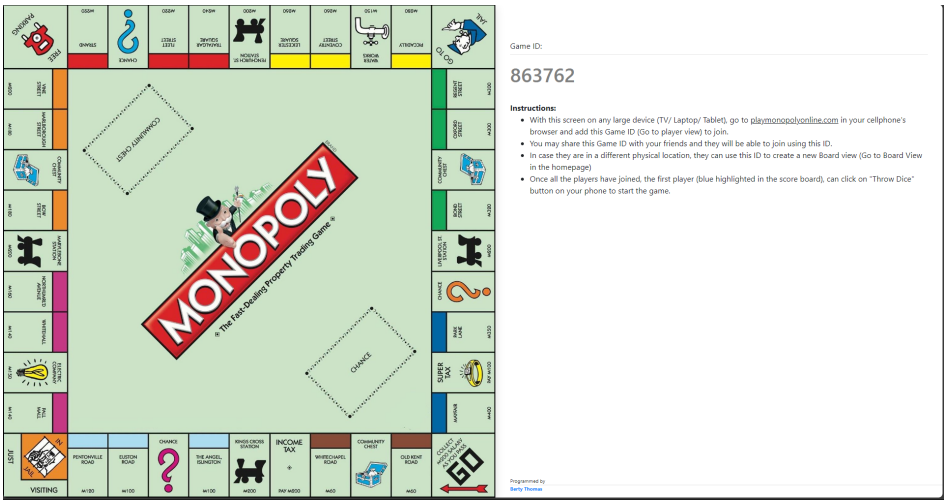

Рисунок 1.4 – Ігрова веб-сторінка гри «Монополія»

Онлайн-версія класичної настільні гри FreeWebArcade є ще однією альтернативою (рис. 1.5). У цій грі гравці змагаються за фінансове домінування, купуючи, продають і розвиваючи нерухомість. Вони пересуваються по полю (рис. 1.6), кидаючи кубики, роблячи угоди та сплачуючи ренту за інші гравці, які

опинилися на їхніх територіях. Щоб стати монополістом і уникнути банкрутства, гра дозволяє випробувати різні стратегії та приймати ризиковані рішення.

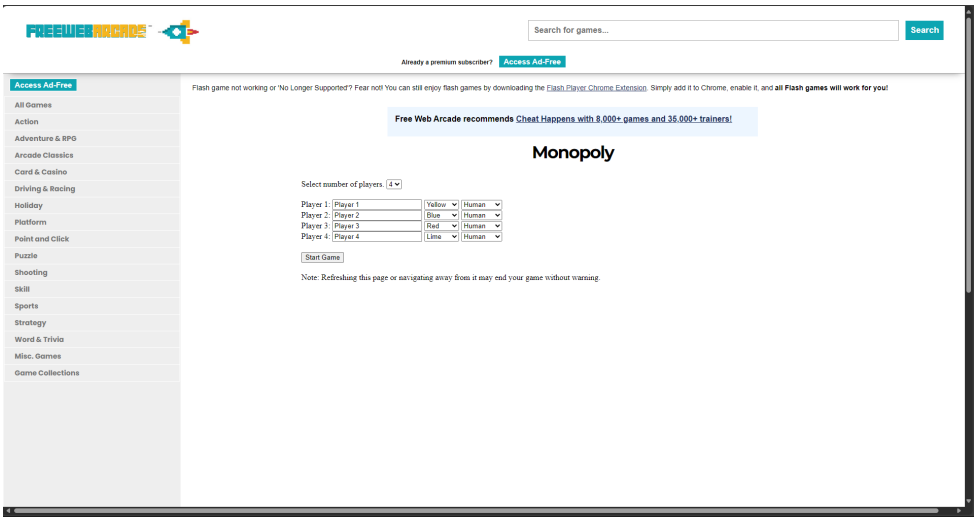

Рисунок 1.5 – Початкове вікно сайту з грою «Монополія»

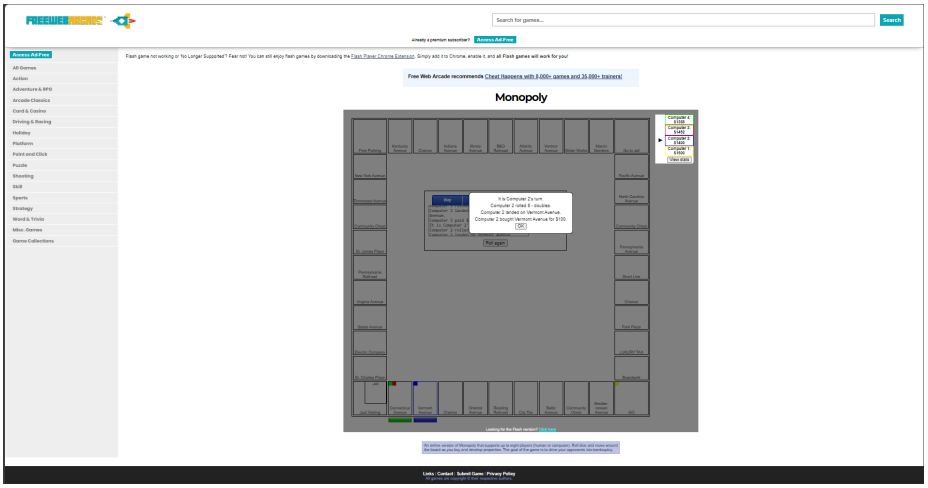

Рисунок 1.6 – Ігрова веб-сторінка гри

1.3 Огляд засобів для розробки веб-сайту

JavaScript — це мова програмування, яка використовується для створення програм і веб-сайтів, які працюють у браузері. Node.js — це відкритий JavaScript-середовище, яке дозволяє виконувати код на стороні сервера. Express — це фреймворк для веб-застосунків Node.js, який надає розробникам можливість

створювати високопродуктивні веб-застосунки з простим і елегантним інтерфейсом.

Основні характеристики Node.js та Express:

Node.js дозволяє виконувати код на стороні сервера, що дозволяє створювати застосунки з кількома користувачами та забезпечує більшу швидкість роботи, оскільки не потрібно відправляти запити на сервер.

За допомогою маршрутизації та створення контролерів Express спрощує процес розробки, дозволяючи розробникам створювати веб-застосунки.

Обидва середовища мають велику кількість модулів і бібліотек, що дозволяє використовувати сторонні бібліотеки для розширення функціональності застосування.

Node.js і Express є двома найпоширенішими інструментами для розробки веб-додатків. З іншого боку, Node.js є середовищем виконання JavaScript на сервері, а Express є фреймворком для веб-додатків, які базуються на Node.js.

Використання Node.js та Express може бути корисним для розробки веб-сайтів з кількох причин:

Швидкість: завдяки використанню асинхронної обробки запитів Node.js обробляє більше запитів одночасно, не блокуючи виконання коду.

Повторне використання коду: Завдяки модульній архітектурі та розширенню функціональності за допомогою middleware Express дозволяє легко перевикористовувати код.

Підтримка WebSocket: Node.js підтримує WebSocket, що дозволяє створювати веб-додатки, які є більш інтерактивними.

Розширюваність: Node.js та Express дозволяють легко збільшувати функціональність за допомогою модулів і пакетів з npm, що робить розробку швидшою та ефективнішою.

Підтримка REST API: Express полегшує створення REST API, що дозволяє створювати веб-додатки, які використовують API для взаємодії з базою даних та іншими додатками.

Активна спільнота: велика спільнота розробників Node.js та Express активно підтримує ці інструменти та сприяє їх розвитку.

Загалом, Node.js та Express є ефективними інструментами для розробки веб-додатків, які забезпечують швидке повторне використання коду та забезпечують швидке виконання.

Альтернативи Node.js та Express:

1. Ruby on Rails, який часто називають просто «Rails», — це популярна система розробки веб-застосунків, написаних мовою програмування Ruby. Він був випущений у 2004 році і з тих пір отримав широке поширення завдяки своїм потужним функціям, гнучкості та простоті використання.

Rails побудовано на основі архітектурного шаблону Model-View-Controller (MVC), який розділяє логіку програми на три окремі частини: модель (яка обробляє зберігання та пошук даних), представлення (яке обробляє презентацію та інтерфейс користувача) і контролер.

Автоматична генерація коду, проста міграція бази даних і вбудована структура тестування є одними з багатьох функцій Rails, які роблять веб-розробку простішою та ефективнішою. Крім того, оскільки він наголошує на конвенціях, а не на конфігурації, розробники можуть розпочати роботу швидко, не витрачаючи багато часу на налаштування чи конфігурацію.

Загалом, Rails — це потужна та популярна платформа для створення веб-застосунків, яка використовується розробниками та компаніями по всьому світу для створення будь-якої продукції, починаючи від невеликих особистих веб-сайтів і закінчуючи великими корпоративними програмами.

2. Django — популярний фреймворк, який використовується для розробки високоякісних веб-застосунків, написаних мовою програмування Python. Завдяки своєму практичному дизайну, гнучкості та потужним функціям він отримав широке поширення з моменту виходу в 2005 році.

Django базується на архітектурі Model-View-Template (MVT), який схожий на шаблон Model-View-Controller (MVC) інших веб-фреймворків. Цей шаблон ділить логіку програми на три частини: модель (яка обробляє зберігання та пошук даних),

представлення (яка обробляє презентацію та інтерфейс користувача) і шаблон (який визначає, як дані відображаються в поданні).

Django пропонує багато функцій, які полегшують веб-розробку, наприклад простий у використанні ORM (Object-Relational Mapping), який полегшує роботу з базами даних, вбудований інтерфейс адміністратора, автоматичне маршрутизація URL-адрес і потужний механізм створення шаблонів.

Django добре підходить для створення великих веб-застосунків завдяки безпеці та масштабованості. У результаті його популярності утворилася велика спільнота розробників і велика екосистема сторонніх пакетів і плагінів.

Загалом Django є потужним і гнучким інструментом для створення веб-застосунків на Python, який використовується розробниками та компаніями в усьому світі для створення різноманітних речей, від невеликих персональних веб-сайтів до величезного корпоративного програмування.

3. Laravel — це безкоштовний фреймворк веб-застосунків PHP із відкритим вихідним кодом, який був випущений у 2011 році. Завдяки простому та елегантному синтаксису та широкому спектру вбудованих функцій він розроблений для спрощення та швидкості розробки веб-застосунків.

Laravel складається з двох основних компонентів, Blade і компоненти Symfony, які створюють надійну основу для розробки веб-застосунків. Blade також є потужним механізмом створення шаблонів, який полегшує створення складних веб-сторінок.

У Laravel є багато функцій, які покращують веб-розробку, наприклад вбудована ORM (Object-Relational Mapping), яка спрощує взаємодію з базами даних, проста система маршрутизації та проміжне програмне забезпечення, а також потужний інтерфейс командного рядка (CLI). що дозволяє розробникам автоматизувати велику кількість стандартних завдань.

Крім того, Laravel зосереджується на безпеці та має вбудовані функції, які допомагають захистити веб-застосунки від поширених загроз, таких як міжсайтовий сценарій (XSS) і SQL-атаки.

Загалом, Laravel — це популярна структура веб-застосунків PHP, яка надає розробникам широкий набір інструментів, щоб створювати міцні та масштабовані веб-застосунки. У результаті його популярності з'явилася велика й жвава спільнота розробників, а також велика екосистема пакетів і плагінів сторонніх розробників.

4. Flask — популярний мікровеб-фреймворк, який використовується на Python. Випущений у 2010 році, він отримав широке поширення завдяки своїй простоті, гнучкості та зручності у використанні.

Flask легкий і простий у використанні фреймворк, який надає розробникам основні інструменти для створення веб-застосунків, не створюючи надмірної структури чи шаблонного коду.

Однією з основних характеристик Flask є його простота. Flask має мінімальний дизайн, на відміну від більших архітектур, таких як Django. Це робить його популярним вибором для створення API або веб-застосунків для малих і середніх компаній.

Flask включає потужний механізм шаблонів Jinja2 і простий веб-сервер із підтримкою HTTP-запитів і відповідей. Крім того, Flask підтримує розширення, які дозволяють розробникам додати функціональні можливості до своїх програм, такі як кешування, автентифікація користувачів і підтримка підключень до бази даних.

Флак також дуже гнучкий. Flask не тільки надає розробникам можливість структурувати свої застосунки будь-яким способом, але й пропонує просту та гнучку систему маршрутизації, яка дозволяє легко зіставляти URL-адреси з функціями, які містяться в програмі.

Загалом, Flask — це потужний і гнучкий веб-фреймворк, який ідеально підходить для створення API або малих і середніх веб-застосунків. Його люблять розробники Python через його простоту, гнучкість і простоту використання.

5. ASP.NET — це поширена платформа веб-застосунків, розроблена Microsoft, яка призначена для створення динамічних веб-застосунків, API і веб-служб. З його першої випуску в 2002 році він постійно вдосконалювався та оновлювався.

ASP.NET базується на .NET framework, групі програмних продуктів, розроблених Microsoft. Цей фреймворк пропонує розробникам широкий спектр

функцій і інструментів для створення веб-застосунків, включаючи підтримку кількох мов програмування та велику бібліотеку попередньо зібраних компонентів.

Здатність працювати з кількома мовами програмування, такими як C#, VB.NET і F#, є однією з основних характеристик ASP.NET. Це полегшує роботу з будь-якою мовою та інтеграцію з іншими технологіями, які базуються на.NET.

Також ASP.NET пропонує багато вбудованих функцій, які роблять веб-розробку швидшою та простішою. До цих функцій належать проста архітектура Model-View-Controller (MVC), вбудована система ORM (Object-Relational Mapping) і потужні інструменти для створення динамічних і адаптивних інтерфейсів користувача.

Крім того, ASP.NET надає багато функцій, які допомагають захистити веб-застосунки від поширених загроз безпеці, таких як SQL-атаки та міжсайтовий сценарій (XSS).

Загалом ASP.NET — це потужна та поширена структура веб-застосунків, яка надає розробникам повний набір інструментів для створення динамічних і масштабованих веб-застосунків. У результаті його популярності утворилася велика і активна спільнота розробників, а також велика екосистема пакетів і плагінів сторонніх розробників.

6. Spring — популярний фреймворк на основі Java, який використовується для створення програмних систем і веб-застосунків для компаній. Вперше випущений у 2002 році, він став одним із найпоширеніших фреймворків Java для створення масштабованих, надійних програм.

Підтримка впровадження залежностей, аспектно-орієнтованого програмування та декларативне керування транзакціями є одними з широкого набору функцій і інструментів Spring, які доступні для розробки застосунків на основі Java.

Підтримка інверсії керування (IoC) є важливою характеристикою Spring, яка дозволяє розробникам відокремлювати конфігурацію програми від її реалізації. Це покращує підтримку коду та тестування, а також забезпечує більш гнучку архітектуру для створення складних систем.

Spring також надає потужну веб-платформу під назвою Spring MVC, яка використовується для створення веб-застосунків. Цей фреймворк надає низку функцій для створення адаптивних і динамічних інтерфейсів користувача, включаючи підтримку веб-сервісів AJAX і RESTful.

Крім того, Spring приділяє значну увагу безпеці та надає багато функцій, які допомагають захистити застосунки від поширених загроз безпеці, таких як міжсайтовий сценарій (XSS) і SQL-атаки.

Загалом Spring — це потужна і широко використовувана платформа Java для створення веб-застосунків і програмних систем корпоративного рівня. Його гнучкість, масштабованість і надійний набір функцій зробили його популярним вибором серед розробників протягом багатьох років, і його популярність не має ознак сповільнення.

1.4 Вимоги до проекту

1. Основні функціональні вимоги:

Реалізація грального поля: Створення інтерактивного ігрового поля з усіма необхідними елементами, такими як ігрові клітинки, карта власності та податки.

Переміщення гравців: Можливість для кожного гравця пересуватися ігровим полем відповідно до результатів випадіння кубиків.

Функції обміну власністю: Гравці мають змогу купувати, продавати, закладати майно, що відображатиметься на ігровому полі.

2. Нефункціональні вимоги:

Інтуїтивний інтерфейс: Веб-застосунок повинен мати зручний та зрозумілий інтерфейс, що дозволяє користувачам легко взаємодіяти з ігровими елементами.

Продуктивність: Сторінки повинні швидко завантажуватися та підтримувати плавний ігровий процес без затримок.

Масштабованість: Архітектура має бути готовою до можливого розширення функціоналу та збільшення кількості одночасних гравців у майбутньому.

1.5 Висновки до першого розділу

Ми проаналізували основні концепції та функції веб-сайту, а також досліджували подібні сайти щодо теми магістерської роботи, щоб визначити можливості, які веб-сайт може запропонувати своїм користувачам. Крім того, вони проаналізували переваги та недоліки різних технологій, які використовуються для створення веб-сайтів.

У результаті цього дослідження були визначені вимоги до проекту. Ці вимоги повинні бути спрямовані на задоволення потреб користувачів і включати основні вимоги до дизайну, функціональності та технічних характеристик веб-сайтів. Щоб створити ефективний веб-сайт, необхідно дотримуватися певних стандартів і використовувати відповідні технології, щоб досягти кращих результатів.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

2.1 Основні технології

2.1.1 Мова програмування JavaScript

JavaScript - це мова програмування, яка використовується для створення веб-застосунків, що динамічно змінюються, реагують на дії користувачів та взаємодіють з сервером. Вона є однією з найпоширеніших мов програмування в інтернеті, тому що її можна використовувати для створення різноманітних функцій на веб-сайті.

Історія JavaScript починається в 1995 році, коли Брендан Айк створив мову програмування на ім'я Mocha, яка була пізніше перейменована на LiveScript, а потім - на JavaScript. Вона була розроблена компанією Netscape і перші версії використовувались для створення ефектів на веб-сторінках.

Крім того він є мовою програмування з високим рівнем абстракції, що дозволяє програмістам створювати складні функції з декількох логічних блоків. Вона використовується для створення функцій, які можуть бути запущені на боковій стороні клієнта, що дозволяє виконувати операції без перезавантаження веб-сторінки.

Також може бути використана для створення декількох типів застосунків, таких як веб-сторінки, веб-застосунки, веб-сервери та мобільні застосунки. Вона є важливим інструментом для створення веб-сторінок з більш складними інтерфейсами та функціями.

Так само може взаємодіяти з HTML та CSS, що дозволяє створювати динамічні веб-сторінки з різноманітними ефектами, такими як анімація, зміна кольорів та зображень. Вона також може використовуватись для створення функцій, які взаємодіють з базами даних та сервером.

В свою чергу може використовуватися для створення клієнтських та серверних застосунків. На клієнтській стороні вона використовується для створення функцій, які взаємодіють з веб-сторінкою та забезпечують інтерактивність та динамічність сторінки. На серверній стороні вона використовується для розробки застосунків, які взаємодіють з базою даних, оброблюють запити від клієнтів та відправляють відповіді.

JavaScript є мовою з відкритим кодом, що дозволяє розробникам вносити свої внески в розвиток мови. Є багато бібліотек та фреймворків JavaScript, що допомагають розробникам зробити свою роботу більш ефективною та продуктивною. Найпопулярнішими бібліотеками та фреймворками є React, Angular, Vue, jQuery, Node.js та інші.

JavaScript має декілька особливостей, які відрізняють її від інших мов програмування. Одна з них - це замикання (closure), що дозволяє зберігати стан та контекст виклику функції. Інша особливість - це прототипне наслідування (prototypical inheritance), що дозволяє створювати нові об'єкти на основі існуючих об'єктів. JavaScript також є мовою з динамічною типізацією, що дозволяє змінювати тип змінної в процесі виконання програми.

Незважаючи на те, що JavaScript є потужною мовою програмування, вона має свої обмеження та пастки. Наприклад, JavaScript може викликати проблеми з безпекою, якщо не належним чином обробляти вхідні дані.

Наступною пасткою JavaScript є залежність від браузера. Різні браузери можуть інтерпретувати JavaScript по-різному, що може приводити до непередбачуваної поведінки. Це означає, що розробники повинні тестувати свої застосунки на різних браузерах та версіях, щоб переконатися, що вони працюють належним чином на всіх платформах.

Ще однією пасткою JavaScript є використання асинхронних запитів. Якщо програміст не зверне увагу на правильний порядок виконання асинхронних запитів, це може привести до непередбачуваної поведінки, що може призвести до помилок та збоїв.

Проте, наразі існує багато рішень та бібліотек, які допомагають уникнути цих пасток. Наприклад, для забезпечення безпеки веб-застосунків можна використовувати фреймворки, які автоматично відбирають та перевіряють вхідні дані. Для роботи з асинхронними запитами можна використовувати Promise або async/await, що забезпечують правильний порядок виконання запитів та дозволяють уникнути непередбачуваних помилок.

JavaScript - це потужна та популярна мова програмування, яка використовується для створення веб-застосунків та сервісів. Вона має свої переваги та пастки, але з правильним підходом може бути дуже ефективним інструментом для створення високоякісних застосунків та сервісів.

2.1.2 Редактор коду Visual Studio Code

Редактор коду - це програмний інструмент, що використовується для створення, редагування та форматування текстових файлів, які містять вихідний код програм. Для розробки веб-сайтів використовуються різноманітні редактори коду, які допомагають розробникам створювати високоякісний та ефективний код.

Один з найбільш популярних редакторів коду для розробки веб-сайтів - це Sublime Text. Він має безліч корисних функцій, таких як підсвічування синтаксису, автодоповнення, швидкий перехід до рядків коду, можливість використовувати макроси, а також безліч інших функцій, які допомагають розробникам під час написання коду.

Іншим популярним редактором коду є Visual Studio Code, який розробляється компанією Microsoft. Він має вбудовану підтримку для різних мов програмування, включаючи HTML, CSS та JavaScript, а також має безліч розширень та плагінів, що дозволяють розширити можливості редактора.

Існує кілька причин, чому Visual Studio Code може бути привабливим вибором для розробки веб-сайтів на JS порівняно з іншими редакторами коду. Деякі з цих причин включають:

- Безкоштовний та відкритий код: Visual Studio Code є безкоштовним для використання та відкритим для редагування, тому він може бути доступним для будь-якої людини, яка хоче займатися розробкою веб-сайтів.

- Підтримка для JavaScript: Visual Studio Code надає повноцінну підтримку для JavaScript, включаючи можливість відлагодження, автодоповнення, рефакторинг та інші корисні функції. Крім того, він підтримує різні фреймворки та бібліотеки, що значно полегшує роботу з JavaScript.

- Наявність розширень: Visual Studio Code має велику кількість розширень, які дозволяють розширити можливості редактора та налаштувати його під свої потреби. Наприклад, є розширення для роботи з Git, розширення для підтримки різних мов програмування, тем оформлення та багато інших.

- Легка настройка та використання: Visual Studio Code має простий та зрозумілий інтерфейс, що дозволяє легко налаштувати його під свої потреби. Крім того, він має швидкий запуск та малу вимогу до ресурсів комп'ютера, що дозволяє ефективно працювати з ним на різних пристроях.

- Активна спільнота: Visual Studio Code має активну спільноту, яка постійно вносить зміни та вдосконалює редактор. Це означає, що ви можете бути впевнені в тому, що ви отримуєте актуальну та надійну програм ну можливість отримати допомогу та підтримку від інших користувачів, а також знайти відповіді на свої запитання та проблеми у спільноті.

- Кроссплатформовість: Visual Studio Code підтримує операційні системи Windows, macOS та Linux, що дозволяє працювати з ним на різних пристроях та зручно перемикатися між ними.

- Інтеграція з іншими інструментами: Visual Studio Code підтримує інтеграцію з іншими інструментами для розробки веб-сайтів, такими як Node.js, npm, TypeScript та інші, що дозволяє зручно працювати з цими інструментами без виходу з редактора.

Інші популярні редактори коду, які можуть бути використані для розробки веб-сайтів, включають Atom, Brackets, Notepad++, IntelliJ IDEA, Eclipse та NetBeans.

Крім редакторів коду, існує також безліч інших інструментів для розробки веб-сайтів, таких як інтегровані середовища розробки (IDE), фреймворки та бібліотеки. Наприклад, для розробки веб-сайтів на основі технологій HTML, CSS та JavaScript можна використовувати фреймворки, такі як React, Angular та Vue.js, які дозволяють значно спростити процес розробки та забезпечити високу швидкість та ефективність веб-застосунків.

Загалом, Visual Studio Code може бути привабливим вибором для розробки веб-сайтів на JS через свою підтримку для JavaScript, наявність розширень та активну спільноту, а також легку настройку та кроссплатформовість. Однак вибір редактора коду є індивідуальним та залежить від особистих переваг та потреб користувача.

2.2 Технології серверної частини розробки веб-застосунку

2.2.1 Модуль dotenv

Модуль dotenv - це інструмент для зберігання конфігураційних даних в змінних середовища, який дозволяє використовувати змінні середовища з файлу .env. За допомогою цієї бібліотеки можна забезпечити безпеку конфігураційних даних, що не потрапляють в код, а зберігаються у файлі, що не публікується у репозиторії проекту.

Бібліотека Dotenv дозволяє завантажувати значення змінних середовища з файлу .env в об'єкт JavaScript, який можна використовувати для налаштування

середовища Node.js. Це дозволяє зберігати конфігураційні дані окремо від коду, що полегшує їх управління.

Бібліотека Dotenv дозволяє використовувати налаштування за допомогою змінних середовища, забезпечуючи безпеку конфігураційних даних, а також уніфікує процес розгортання проекту на різних середовищах. Вона є популярним інструментом для розробки в середовищі Node.js і забезпечує зручний та безпечний спосіб зберігання конфігураційних даних.

2.2.2 Фреймворк express

Бібліотека Express - це відкрита JavaScript бібліотека для розробки веб-застосунків та API з використанням Node.js. Вона була розроблена в 2010 році і з тих пір стала однією з найпопулярніших бібліотек для розробки серверної частини веб-застосунків.

Основна ідея Express полягає в тому, щоб дозволити розробникам з легкістю створювати веб-застосунки шляхом визначення шляхів маршрутизації, які відповідають на запити клієнтів та визначення обробника для кожного маршруту. Express дозволяє також робити взаємодію з базами даних, відправляти запити на інші веб-сервіси та створювати API.

Express надає розробникам велику кількість різноманітних функцій, таких як обробка запитів HTTP, робота з кукісами, робота з параметрами запитів та маршрутизації, що значно полегшує процес розробки веб-застосунків.

2.2.3 Middleware morgan

Morgan — це middleware для Node.js, який зазвичай використовується з веб-фреймворком Express для логування HTTP-запитів. Це простий, але дуже потужний

інструмент, що допомагає розробникам відстежувати і реєструвати всі вхідні запити до сервера, що полегшує діагностику та моніторинг.

Особливості Morgan

- Гнучкі налаштування форматів логів: Morgan підтримує різні формати логів, наприклад, combined, common, dev, short, та tiny. Вони дозволяють визначати, які саме дані про запит будуть збережені (IP-клієнта, URL, статус відповіді, час відповіді тощо).
- Підтримка кастомних форматів: Morgan дозволяє створювати власні формати логів, завдяки чому можна адаптувати його під специфічні потреби проекту.
- Збереження логів в файл: Можна направляти логи в файл для довготривалого зберігання або подальшого аналізу.
- Інтеграція з іншими middleware: Morgan легко інтегрується з іншими middleware, такими як body-parser, cookie-parser, що робить його дуже гнучким у налаштуванні.

2.2.4 Бібліотека socket.io

Socket.IO — це популярна бібліотека JavaScript, яка спрощує створення додатків із функціоналом реального часу, зокрема для вебчатів, онлайн-ігор, систем нотифікацій тощо. Вона забезпечує двосторонню комунікацію між клієнтом та сервером через протокол WebSocket, що дозволяє надсилати й отримувати повідомлення в режимі реального часу без необхідності оновлювати сторінку.

Основні характеристики Socket.IO:

1. Двостороння комунікація: забезпечує миттєву передачу даних від клієнта до сервера і назад.
2. Автоматичний вибір транспорту: підтримує WebSocket і, у разі його недоступності, автоматично переходить на інші протоколи (наприклад, HTTP Long Polling).

3. Надійність з'єднання: автоматично встановлює та підтримує стабільне з'єднання, навіть якщо клієнт і сервер втратили зв'язок.
4. Підтримка подій: клієнт і сервер можуть надсилати та отримувати події, що робить API зручним для використання.
5. Просте масштабування: за допомогою адаптерів можна налаштувати розподілену архітектуру для великих додатків.

   Як працює Socket.IO?

   Socket.IO працює з двома основними компонентами:

- Клієнтська бібліотека для браузера, яка підключається до сервера.
- Серверна бібліотека, яка обробляє підключення клієнтів.

  Переваги та недоліки Socket.IO

  Переваги:

- Простий у використанні API для створення реального часу функціоналу.
- Підтримка подій та розширення можливостей.
- Автоматичне відновлення з'єднання.

  Недоліки:

- Підходить не для всіх сценаріїв (наприклад, коли потрібен високий рівень безпеки).
- Може вимагати додаткових налаштувань для масштабування.

Socket.IO є універсальним інструментом для створення додатків реального часу, який чудово підходить для багатьох сучасних веб додатків.

2.2.5 Інструмент nodemon

Інструмент nodemon - це пакет для Node.js, який дозволяє автоматично перезапускати сервер після змін у файловій структурі проекту. Він допомагає розробникам зекономити час і змінити код без необхідності кожного разу вручну

перезапускати сервер. Nodemon також може детектувати та повідомляти про помилки під час розробки, що полегшує процес налагодження.

Встановлення nodemon дуже просте. Після встановлення через менеджер пакетів npm, можна запустити сервер за допомогою команди nodemon app.js замість звичайної node app.js. При цьому, будь-які зміни в коді автоматично викличуть перезапуск сервера.

Бібліотека nodemon є важливим інструментом для зручного розроблення серверної частини застосунків на Node.js, зокрема тих, що використовуються веб-серверами Express або інші. Вона дозволяє зосередитися на розробці застосунку, а не на перезапуску сервера кожного разу, коли змінюється код.

2.3 Технології клієнтської частини розробки веб застосунку

2.3.1 Мова розмітки HTML

HTML (Hypertext Markup Language) є стандартною мовою розмітки документів для вебу. Вона використовується для створення структури веб-сторінок та визначення різних елементів веб-документа, таких як заголовки, параграфи, списки, таблиці, форми та інші. HTML кодує структуру та семантику вмісту веб-сторінки, а також забезпечує спілкування між веб-браузером та веб-сервером.

HTML є однією з найбільш важливих технологій для веб-розробників. Без HTML веб-сторінки були б простою колекцією текстових даних без якоїсь структури. Завдяки HTML веб-розробники можуть створювати документи з різноманітним вмістом, що зручно переглядати та змінювати.

Синтаксис HTML дуже простий та легко зрозуміти. Код в HTML складається з тегів та тексту між ними. Теги складаються з відкриваючої та закриваючої частин,

а текст розміщується всередині цих тегів. Наприклад, тег <p> використовується для визначення параграфу, і він виглядає так:

Лістинг 2.1 - Фрагмент тегу параграма

```
<p>Це приклад параграфу в HTML</p>
```

Також існують одиночні теги, які не мають закриваючої частини, такі як <img> для вставки зображення або
 для перенесення тексту на новий рядок.

HTML також підтримує атрибути, які використовуються для задання різних параметрів тегів. Наприклад, тег <img> може мати атрибут src, який вказує шлях до зображення:

Лістинг 2.2 - Фрагмент тегу зображення

```
<img src="image.jpg" alt="Це опис зображення">
```

Атрибут alt використовується для вказівки альтернативного тексту, який відображається, коли зображення не завантажується.

Є також тег , який дозволяє групувати елементи та розміщувати їх в окремих блоках. Це допомагає розділити веб-сторінку на різні частини та зручніше керувати її виглядом:

Лістинг 2.3 - Фрагмент тегу групування елементів

```
<div>
<h2>Це заголовок</h2>
<p>Це параграф</p>
</div>
```

HTML також дозволяє додавати веб-форми для збору даних від користувачів. Наприклад, можна використовувати тег <form> для створення форми та різні елементи форми, такі як <input> для введення даних, <select> для вибору зі списку та <textarea>.

HTML є важливою технологією для веб-розробки, і вона є стандартом для веб-сторінок в Інтернеті. Вона є доступною для всіх та дозволяє створювати веб-сторінки, які виглядають гарно та мають корисний вміст. Також HTML є базовою технологією для розробки веб-застосунків та інших веб-технологій, таких як CSS-каскадні таблиці стилів.

2.3.2 Мова стилю сторінок CSS

CSS є однією з ключових технологій веб-розробки, яка використовується для опису стилів та оформлення веб-сторінок. CSS дозволяє відокремити вміст веб-сторінки від її оформлення, що дозволяє створювати більш гнучкі та ефективні веб-сторінки.

Основним завданням CSS є надання стилів та оформлення веб-сторінок. CSS дозволяє задавати стилі для різних елементів веб-сторінки, таких як текст, фон, межі, розміри тощо. CSS використовується для створення привабливого та зручного дизайну веб-сторінок, що допомагає залучити більше користувачів та поліпшити їх взаємодію з веб-сторінкою.

Синтаксис CSS дуже простий. CSS складається з селекторів та декларацій. Селектор вказує, який елемент на веб-сторінці повинен бути оформлений, а декларація вказує, які стилі повинні бути застосовані до цього елемента.

CSS підтримує велику кількість властивостей стилю, які можуть бути застосовані до елементів веб-сторінки. Деякі з найбільш поширених властивостей включають:

font-size: задає розмір шрифту елемента;

color: задає колір тексту елемента;

background-color: задає колір фону елемента;

border: задає стиль, ширину та колір межі елемента;

padding: задає відступ від межі до вмісту елемента;

margin: задає відступ від межі до інших елементів на сторінці.

CSS також дозволяє використовувати класи та ідентифікатори для вибору елементів на сторінці. Класи та ідентифікатори дозволяють задати стилі для конкретних елементів або груп елементів, що дозволяє зменшити кількість декларацій CSS та зробити код більш читабельним.

Однією з найбільш важливих функцій CSS є розміщення стилів на окремій сторінці або в окремому файлі. Це дозволяє відокремити вміст веб-сторінки від її оформлення та зменшити кількість дублювання коду. Крім того, це дозволяє зберігати стилі в одному місці та легко змінювати їх для всієї сторінки або групи сторінок.

Останнім часом CSS розвивається швидко та стає все більш потужним та гнучким інструментом для розробки веб-сторінок. Наприклад, CSS Grid та CSS Flexbox дозволяють легко створювати складні макети та вирівнювати елементи на сторінці. CSS Animations дозволяє створювати анімаційні ефекти на сторінках, а CSS Variables дозволяє використовувати змінні для задання значень стилів.

Узагальнюючи, CSS є важливою технологією для веб-розробки, яка дозволяє відокремлювати вміст сторінки від її оформлення та забезпечує багато можливостей для візуального оформлення веб-сторінок. CSS дозволяє використовувати різні типи селекторів для вибору елементів на сторінці та задавати для них стилі, що включають кольори, шрифти, відступи, межі та інші властивості. CSS також дозволяє використовувати класи та ідентифікатори для задання стилів для конкретних елементів або груп елементів, що дозволяє зробити код більш читабельним та зменшити кількість декларацій CSS.

CSS дозволяє відокремлювати стилі на окремій сторінці або в окремому файлі, що забезпечує зручність та зберігає код більш чистим та читабельним. Крім того, CSS забезпечує розширення можливостей веб-розробки, завдяки новим функціям та властивостям, які додаються до стандарту з часом.

Однією з головних переваг CSS є можливість створювати складні макети та вирівнювати елементи на сторінці. CSS Grid та CSS Flexbox дозволяють створювати різні види макетів, включаючи змішані макети та макети зі змінними розмірами, та

контролювати вирівнювання елементів. Крім того, CSS дозволяє створювати анімаційні ефекти, включаючи переміщення, зміну кольорів та прозорості, а також розвивається в напрямку розширення можливостей для анімацій та інтерактивності.

CSS є необхідною складовою веб-розробки, оскільки вона дозволяє забезпечити кращу доступність, читабельність та керованість коду. CSS дозволяє зробити веб-сторінки більш дружніми до користувача та забезпечити більш візуально привабливий та ефективний дизайн. CSS також дозволяє збільшити ефективність розробки, зменшивши кількість дублювання коду та забезпечуючи зручність відлагодження та редагування.

Однак, важливо пам'ятати, що CSS не є універсальним інструментом, і він не може вирішити всі проблеми веб-розробки. Є деякі обмеження CSS, які можуть бути важко перетнути в деяких випадках, і можуть вимагати використання інших технологій, таких як JavaScript або фреймворки.

У загальному, CSS є потужним інструментом для веб-розробки, який дозволяє створювати більш дружні, ефективні та привабливі веб-сторінки. Його можливості постійно розширюються та покращуються, що забезпечує зручність та ефективність розробки веб-застосунків.

2.3.3 Фреймворк vue.js

Vue.js — це прогресивний фреймворк для створення користувацьких інтерфейсів (UI), що дозволяє розробникам будувати сучасні веб-застосунки з ефективним і гнучким підходом. Його основна мета — забезпечити зручний спосіб створення динамічних веб-сторінок, особливо для односторінкових додатків (SPA, Single Page Applications).

Основні особливості Vue.js

1. Реактивність: Vue.js використовує реактивний підхід до управління станом компонентів. Це означає, що якщо змінюється стан певних даних, Vue автоматично оновлює інтерфейс користувача для відображення цих змін.
2. Компонентний підхід: У Vue можна розділити застосунок на незалежні компоненти, кожен з яких відповідає за певну частину інтерфейсу. Це дозволяє легко повторно використовувати код і спрощує процес підтримки.
3. Гнучкість і інтеграція: Vue легко інтегрується з іншими бібліотеками або існуючими проєктами, тому його можна використовувати як у невеликих частинах додатку, так і для розробки повноцінних SPA.
4. Декларативний синтаксис: Vue пропонує зручний синтаксис для опису того, як виглядає інтерфейс, а не як його будувати. Це дозволяє легше читати і розуміти код.
5. Шаблонний синтаксис: Vue використовує шаблони HTML з директивами (наприклад, v-if, v-for), які дозволяють легко управляти відображенням компонентів.

Основні компоненти Vue.js

- Vue Instance: центральний об’єкт у Vue-застосунку, який управляє даними і поведінкою компонентів.
- Computed Properties: дозволяють обчислювати значення на основі інших даних і кешувати результати для підвищення продуктивності.
- Директиви (Directives): спеціальні атрибути для управління DOM (наприклад, v-bind, v-model), що спрощують інтерактивність.

Vue CLI та Vue Router

Vue CLI допомагає швидко налаштувати проєкт, додаючи готові конфігурації для Webpack, Babel, ESLint тощо. Vue Router відповідає за навігацію між сторінками і дозволяє створювати багатосторінкові SPA.

Vuex

Vuex — це бібліотека для управління глобальним станом, яка інтегрується з Vue і полегшує управління даними, які необхідні кільком компонентам.

Переваги Vue.js

- Простота у вивченні та використанні
- Висока продуктивність та мінімальна затримка
- Велика спільнота та активна підтримка

Vue.js підходить як для невеликих інтерактивних частин, так і для масштабних застосунків, що робить його універсальним вибором для фронтенд-розробників.

2.3.4 Бібліотека axios

Axios - це бібліотека JavaScript, що використовується для здійснення запитів на сервер з боку клієнта. Axios забезпечує зручний та простий інтерфейс для виконання запитів, включаючи підтримку звернення до API за допомогою методів GET, POST, PUT та DELETE.

Однією з основних переваг Axios є його можливість перехоплювати та обробляти помилки запиту на рівні інтерсепторів. Інтерсептори дозволяють виконувати певні дії перед відправкою запиту на сервер та після його завершення, наприклад, встановити заголовки запиту або обробити відповідь сервера.

Axios може бути використаний як у браузері, так і на сервері з Node.js. Завдяки його простоті та зручності використання, він став дуже популярним інструментом серед розробників веб-застосунків та мобільних застосунків.

2.4 Висновок до другого розділу

У цьому розділі було досліджено різноманітні технології, що використовуються в процесі розробки веб-додатків.

Visual Studio Code - це потужний редактор коду, який підтримує багато мов програмування та забезпечує розширення для поліпшення продуктивності розробника.

nodemon - це інструмент, який дозволяє автоматично перезавантажувати сервер при зміні коду, що допомагає зекономити час розробника.

dotenv - це модуль, який дозволяє зберігати конфігураційні дані в змінних середовища, що допомагає зберігати конфіденційну інформацію.

Express - це фреймворк для Node.js, який дозволяє створювати веб-додатки та API швидко та просто.

JavaScript - це мова програмування, яка використовується для створення динамічних веб-сайтів та веб-додатків.

CSS - це мова стилів, яка використовується для оформлення та дизайну веб-сторінок.

HTML - це мова розмітки, яка використовується для створення структури веб-сторінок.

morgan - це middleware для Express, який забезпечує логування HTTP запитів, що дозволяє зручніше відслідковувати взаємодії з сервером.

socket.io - це бібліотека, яка дозволяє реалізовувати двостороннє спілкування в реальному часі між клієнтом та сервером.

axios - це бібліотека для роботи з HTTP запитами, яка спрощує процес взаємодії з API та сервером.

vue.js - це популярний JavaScript-фреймворк для створення інтерактивних веб-інтерфейсів, який забезпечує гнучкість та зручність у розробці.

Усі ці технології є важливими складовими веб-розробки та дозволяють створювати зручні, безпечні та привабливі веб-додатки для користувачів. Комбінація цих технологій дозволяє забезпечити високу якість розробки та ефективну взаємодію між фронтенд та бекенд частинами додатку.

3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Розробка серверної частини веб-застосунку

3.1.1 Загальна структура серверної частини

Почнемо з інсталяції середовища розробки Visual Studio code, після створюємо пустий проект та ініціалізуємо його з попередньо встановленими значеннями і в якому встановлюємо наступні технології.

Команда для ініціалізації проекту – npm init -y.

Для встановлення залежностей визиваємо термінал через Menu – Terminal – New Terminal та заходимо в кореневу папку серверної частини коду та вводимо наступні команди для тієї чи іншої технології.

- Фреймворк express
  npm i express
- Middleware morgan
  npm i morgan
- Бібліотеку Socket.io
  npm i socket.io
- Інструмент Nodemon
  npm i nodemon
- Бібліотеку dotenv
  npm i dotenv

Тепер створимо структуру серверної частини проекту. Monopoly >>server>> game>>config
>> components

Monopoly коренева папка гри.

Server папка з серверною частиною веб-застосунку.

Папка game містить основні файли гри.

Config має початкові конфігураційні дані гри.

Components містить компоненти гри з якими взаємодіють гравці. Створимо файл cards.js в якому реалізується функціонал для карток, наприклад, ігрових об'єктів чи елементів.

Створимо файл field.js для роботи з ігровим полем, включаючи його генерацію, розмірність та властивості.

Створимо файл player.js для управління логікою гравця, збереження його стану та взаємодії в грі.

Створимо файл tile.js для опису логіки плиток, які використовуються на ігровому полі.

Створимо файл tracker.js для відстеження станів чи прогресу в грі.

Створимо папку config для зберігання конфігураційних файлів, які визначають налаштування гри.

Створимо файл console.js, який містить функціонал для роботи з ігровою консоллю або командним введенням.

Створимо файл controller.js, у якому реалізовано основну логіку управління ігровими подіями.

Створимо файл core.js, що містить основну логіку ядра гри, об'єднуючи всі компоненти в один функціонал.

Створимо файл executor.js для виконання основних ігрових сценаріїв чи дій.

Створимо файл utils.js із допоміжними функціями, які полегшують роботу інших компонентів гри.

Створимо файл validator.js, який перевіряє правильність введених даних або станів у грі.

Створимо файл apiRouter.js – файл, який відповідає за маршрутизацію API-запитів для взаємодії із сервером.

Створимо основний файл застосунку app.js, який ініціалізує сервер і пов'язує всі компоненти.

Створимо файл rooms.js для управління логікою кімнат (локацій або сесій) у багатокористувацькій грі.

Створемо файл socket.js, який реалізує логіку роботи з WebSocket для обробки реального часу.

Створимо файл .env.example конфігурації змінних середовища для налаштувань сервера.

Створимо файл index.js для запуску основного сервера, що об'єднує всі інші модулі.

3.1.2 Створимо файл app.js

Цей код є частиною Express.js програми, яка налаштовує сервер для обробки HTTP-запитів. Першим кроком підключається бібліотека express, що дає можливість створювати сервери для веб-додатків. Потім підключається бібліотека morgan, яка використовується для логування HTTP-запитів, що надходять до сервера. Використовується middleware morgan("tiny"), що забезпечує короткий формат логування.

Далі створюється об'єкт app за допомогою express(), який є основним об'єктом для налаштування та запуску сервера. За допомогою app.use(express.json()) налаштовується обробка JSON-формату тіла запитів, що дозволяє серверу приймати дані у вигляді JSON.

Наступний middleware app.use("/api", apiRouter) використовує API-маршрутизатор, підключений через файл apiRouter, що дозволяє організувати обробку запитів за маршрутом /api.

В кінці весь налаштований додаток експортується за допомогою module.exports = app, що дозволяє використовувати цей сервер у інших частинах програми, наприклад, для запуску на сервері.

3.1.3 Створимо файл rooms.js

Цей код реалізує механізм управління ігровими кімнатами для гри. Він складається з двох основних класів: **Room** та RoomManager, а також деяких допоміжних функцій.

Клас Room

Цей клас представляє одну кімнату для гри, в якій можуть перебувати кілька гравців.

- Конструктор:
    - this.title — назва кімнати.
    - this.players — об'єкт, що зберігає інформацію про гравців у кімнаті.
    - this.game — об'єкт гри (початково null).
- Методи:
    - _setHost(): Призначає першого гравця хостом (якщо хоча б один гравець є в кімнаті).
    - getCountPlayers(): Повертає кількість гравців у кімнаті.
    - addPlayer(username, idSocket): Додає нового гравця до кімнати. Якщо кімната переповнена або ім'я користувача вже існує, повертається відповідна помилка.

- removePlayerByName(username): Видаляє гравця за його ім'ям.
- removePlayerById(idSocket): Видаляє гравця за його idSocket.
- startGame(optionsGame): Створює і запускає нову гру для цієї кімнати з заданими параметрами.

Клас RoomManager

Цей клас управляє набором всіх кімнат.

- Конструктор:
  - this.rooms — об'єкт, що зберігає всі кімнати, де ключем є назва кімнати, а значенням — об'єкт класу Room.
- Методи:
  - getTitles(): Повертає список назв всіх кімнат.
  - getDataRooms(): Повертає список об'єктів, що містять назви кімнат, кількість гравців і статус (чи йде зараз гра в кімнаті).
  - findPlayer(id): Шукає гравця по його idSocket. Якщо він знайдений, повертає кімнату та ім'я гравця.
  - createRoom(title): Створює нову кімнату, якщо така ще не існує.
  - deleteRoom(title): Видаляє кімнату за її назвою.

Експортування

- Room та RoomManager експортуються для використання в інших частинах програми.
- Створено один екземпляр RoomManager (roomManager), який також експортується, щоб зручно працювати з ним в інших частинах коду.
- LIMIT_ROOM — максимальна кількість гравців в одній кімнаті, експортується для використання в інших частинах програми.

Цей код використаний в серверній частині гри для організації ігрових сесій, де кожна кімната має свою гру, список гравців і можливість запуску гри при заповненні кімнати.

3.1.4 Створимо файл socket.js

Цей код описує клас WrapSocket, який обробляє різні події та взаємодії з веб-сокетами (через бібліотеку socket.io) для управління кімнатами гри та їх гравцями. Ось що відбувається в кожному розділі:

Імпорт:

- Socket та Server з бібліотеки socket.io — це класи для роботи з веб-сокетами.
- roomManager та LIMIT_ROOM з файлу rooms — об'єкти для управління кімнатами та лімітом гравців у кімнаті.
- Controller з файлу гри — клас для управління логікою гри.

Клас WrapSocket

Цей клас обгортає сокет і додає логіку для обробки подій для кожного користувача.

Конструктор:

- Приймає об'єкти originalSocket та server, які передаються ззовні.
- Ініціалізує об'єкт data, що зберігає інформацію про кімнату та користувача.
- Прив'язує кілька подій (наприклад, "createRoom", "pingRooms", "entryLobby" тощо) до відповідних методів класу.

Методи:

1. Системні методи:
    - _useGame(command, options): Використовує клас Controller для виконання команд гри (наприклад, рухи, повідомлення) і оновлює гру після кожної команди.
2. Методи отримання (рецептори подій):
    - receiveCreateRoom(title): Створює нову кімнату з заданою назвою та відправляє відповідь клієнту (успіх або помилка).
    - receivePingRooms(): Додає користувача до спеціальної групи "findingRooms" для пошуку доступних кімнат.
    - receivePongRooms(): Видаляє користувача з групи "findingRooms".
    - receivePingGame(): Використовується для відновлення з'єднання з грою, якщо воно було втрачено. Повідомляє гру про повторне підключення.
    - receiveEntryLobby(titleRoom, username): Додає користувача до конкретної кімнати за її назвою і ім'ям користувача.
    - receiveLeaveLobby(options={}): Виходить з поточної кімнати. Якщо гра вже почалася, гравець може бути вимкнений через втрату з'єднання або примусове відключення.
    - receiveStartGame(options={}): Починає гру в кімнаті з переданими параметрами.
3. Методи для відправки даних (сигналів):
    - sendListRooms(): Відправляє список кімнат, що відповідають певним критеріям (кількість гравців менша за ліміт і кімната у стані "lobby").
    - sendUpdateGame(): Оновлює гру в кімнаті та надсилає актуальний стан гри всім підключеним клієнтам цієї кімнати.

Спеціальні групи:

- FINDING_ROOMS: Це спеціальна група сокетів, в якій перебувають клієнти, що шукають доступні кімнати для гри. Всі клієнти, що знаходяться в цій групі, отримують оновлення про доступні кімнати через подію listRooms.

Експортування:

- Експортується клас WrapSocket, щоб його можна було використовувати в інших частинах програми для обробки взаємодії з клієнтами через сокети.

Як це працює:

1. Створення кімнат: Коли користувач надсилає запит на створення кімнати, система перевіряє, чи не існує вже кімнати з такою назвою. Якщо кімната успішно створена, вона додається до списку доступних.
2. Пошук кімнат: Користувачі, що знаходяться в групі "findingRooms", отримують список кімнат, до яких можна приєднатися.
3. Приєднання до кімнат: Користувачі можуть приєднуватися до кімнат за допомогою події entryLobby, і після приєднання вони можуть почати гру, якщо всі умови виконані.
4. Гра та відключення: Під час гри, якщо з'єднання з гравцем втрачається, він може бути відновлений через подію pingGame. Якщо гравець не підключається протягом певного часу, він автоматично відключається.

Цей код реалізує основні механізми для управління кімнатами та взаємодією між клієнтами в реальному часі, використовуючи веб-сокети.

3.1.5 Створимо файл apiRouter.js

Цей код описує маршрутний обробник для серверу Express.js, який взаємодіє з менеджером кімнат, використовуючи API для створення, отримання списку та видалення кімнат.

Імпорт:

- express — бібліотека для роботи з веб-серверами в Node.js.
- roomManager та LIMIT_ROOM — імпортуються з файлу rooms, де міститься логіка для управління кімнатами (створенням, видаленням, отриманням даних тощо).

Основна логіка маршрутизатора:

1. Маршрут для створення кімнати (POST /create):
    - Запит: Очікує, що в тілі запиту буде передано поле title, яке містить назву нової кімнати.
    - Логіка: Перевіряє, чи можна створити кімнату за вказаною назвою. Якщо кімната з такою назвою вже існує, повертається помилка. Якщо кімната успішно створена, повертається повідомлення про успіх.
    - Відповідь: Відправляє JSON-об'єкт, що містить поле ok (статус операції) та поле desc (опис результату).
2. Маршрут для отримання списку кімнат (GET /list):
    - Запит: Не потребує параметрів у тілі запиту, оскільки просто отримує список всіх доступних кімнат.
    - Логіка: Отримує всі кімнати через метод getDataRooms() з roomManager. Потім фільтрує кімнати, щоб залишити лише ті, які не

переповнені (кількість гравців менша за LIMIT_ROOM) і знаходяться в стані "lobby".

- Відповідь: Відправляє список доступних кімнат у форматі JSON.

3. Маршрут для видалення кімнати (POST /delete):

    - Запит: Очікує, що в тілі запиту буде передано поле title, яке містить назву кімнати, яку потрібно видалити.
    - Логіка: Перевіряє, чи існує кімната з такою назвою. Якщо вона є, видаляє її. Якщо кімнати не існує, повертається помилка.
    - Відповідь: Відправляє JSON-об'єкт, що містить поле ok та поле desc, яке вказує на успіх чи помилку операції.

Загальний процес:

- Клієнт може створювати, отримувати та видаляти кімнати через API.
- Створення кімнати: відправляється запит на /create з назвою кімнати, і сервер відповідає, чи була кімната створена.
- Отримання списку кімнат: запит на /list повертає список всіх доступних кімнат, які можуть бути використані для приєднання нових гравців.
- Видалення кімнати: запит на /delete дозволяє видаляти кімнати за їх назвами.

Експортування:

- Маршрути експортуються через module.exports = router, що дозволяє підключити їх до основного додатку Express.

Приклад використання:

1. Створення кімнати:

    - Запит: POST /create
    - Тіло запиту: { "title": "Room1" }

- Відповідь: { "ok": true, "desc": "Room created" }

2. Отримання списку кімнат:
    - Запит: GET /list
    - Відповідь: [{ "title": "Room1", "count": 2, "status": "lobby" }, { "title": "Room2", "count": 1, "status": "lobby" }]
3. Видалення кімнати:
    - Запит: POST /delete
    - Тіло запиту: { "title": "Room1" }
    - Відповідь: { "ok": true, "desc": "Room deleted" }

3.1.6 Створимо файл index.js

Цей код налаштовує сервер на основі Node.js та Express.js з інтеграцією Socket.io для реального часу, а також забезпечує обслуговування статичних файлів (наприклад, для клієнтської частини додатка). Ось детальний опис кожного з елементів:

Імпорти:

1. dotenv: Бібліотека для завантаження змінних середовища з файлу .env. Вона дозволяє зберігати конфігурацію, таку як порт або інші налаштування, без необхідності захардкоджувати їх у коді.
2. http: Модуль для створення HTTP-сервера. Це стандартний модуль у Node.js, необхідний для запуску сервера.

3. express: Бібліотека для створення серверів в Node.js, яка спрощує роботу з маршрутизацією та middleware.
4. path: Модуль для роботи з файловими шляхами в Node.js.
5. Server з socket.io: Модуль для роботи з веб-сокетами, що дозволяє організувати двостороннє спілкування між сервером і клієнтами.

Логіка:

1. Налаштування статики:
    - staticFolderPath: Створюється шлях до папки, що містить клієнтські статичні файли (наприклад, HTML, CSS, JS, які зібрані для продакшн середовища).
    - app.use(express.static(staticFolderPath)): Вказує Express обробляти файли з цієї папки як статичні ресурси, щоб їх можна було завантажувати за запитами від клієнтів.
    - app.get("*", (req, res) => {...}): Усі інші запити, що не відповідають конкретним статичним файлам, направляються на index.html. Це важливо для клієнтських додатків на основі односторінкових застосунків (SPA), де клієнтська частина сама управляє маршрутами.
2. Завантаження змінних середовища:
    - dotenv.config(): Завантажує налаштування з файлу .env, де можуть бути збережені такі дані, як порт сервера або ключі для API.
3. Створення HTTP-сервера та Socket.io:
    - const server = http.createServer(app): Створює HTTP-сервер, який обробляє запити за допомогою Express.
    - const io = new Server(server): Ініціалізує сервер Socket.io для обробки веб-сокетів, прив'язуючи його до HTTP-сервера.

4. Обробка підключень Socket.io:

    - io.on("connection", (socket) => new WrapSocket(socket, io)): Кожен раз, коли клієнт підключається через веб-сокет, створюється новий об'єкт WrapSocket, який обробляє взаємодію з конкретним клієнтом через сокет.

5. Запуск сервера:

    - const PORT = process.env.PORT ?? 5500: Якщо порт заданий у змінній середовища, використовується його значення, інакше — використовується порт 5500 за замовчуванням.

    - server.listen(PORT, () => {...}): Запускає сервер і виводить повідомлення в консоль про його старт.

Експортування:

- module.exports = server: Експортується сам сервер, щоб він міг бути використаний в інших частинах програми або в тестах.

Цей код налаштовує повноцінний сервер, який обробляє як статичні файли, так і з'єднання через веб-сокети для реального часу. Завдяки цьому можна створювати інтерактивні багатокористувацькі додатки, де клієнти можуть взаємодіяти в реальному часі, а також отримувати статичний контент.

3.1.7 Створимо файл console.js

Цей код представляє клас Console, який керує виконанням команд у грі через консоль. Він дозволяє користувачам виконувати різні операції, такі як покупка, будівництво, продаж, переказ грошей та інші функції, що взаємодіють з внутрішньою логікою гри.

Пояснення кожної частини коду:

Властивості:

1. validator: Об'єкт, що перевіряє коректність введених даних (наприклад, чи існує користувач, чи правильний формат ідентифікатора тайлу).
2. core: Основна логіка гри, що містить поле та гравців, на основі яких виконуються дії.

Методи:

1. _parseBool(valueString): Перетворює рядок на булевий тип (true або false). Якщо значення не відповідає одному з цих варіантів, викидає помилку.
2. _parseUsername(username): Обробляє ім'я користувача, замінюючи підкреслення (_) на пробіли, а потім перевіряє його валідність.
3. _createExecutor(username): Створює екземпляр об'єкта Executor для виконання команд від імені користувача.
4. _createExecutorByTile(idTile): Створює екземпляр Executor, базуючись на інформації про тайл (клітинку на полі).
5. _getTile(idTile): Повертає об'єкт тайлу за його ідентифікатором.
6. command(commandString): Розбирає рядок команди, виконує перевірку на валідність команди, а потім викликає відповідний метод для її виконання.

Команди:

1. echo(args): Виводить повідомлення від конкретного користувача.
2. buy(args): Купує тайл для користувача, де free вказує, чи безкоштовно це (за умовами гри).
3. buyAll(args): Купує всі доступні тайли для конкретного користувача, а також будує на них (якщо є монополія).
4. pledge(args): Виконує операцію застави (наприклад, закладає майно).

5. build(args): Будує на конкретному тайлі.
6. sell(args): Продає тайл, звільняючи його.
7. money(args): Змінює кількість грошей у користувача, перевіряючи, чи правильна величина введена та чи не призведе операція до боргу.
8. move(args): Переміщує користувача по полю на певний тайл. Також передає тайл іншому гравцю (якщо вказано dispath).
9. arrest(args): Виконує арешт конкретного гравця. Якщо арештований гравець — це поточний, то виконується додаткове правило для переходу ходу.
10. add(args): Додає об'єкти (наприклад, картки, як "відкриття в'язниці" або інші ефекти гри).
11. monopoly(args): Якщо гравець має монополію на певний колір (набір тайлів), то автоматично купує всі тайли цього набору і може побудувати на них будинки.

Клас Console забезпечує інтерфейс для виконання різних команд у грі. Кожна команда розбивається на аргументи, перевіряється на валідність і потім передається для виконання в відповідні методи гри. Важливою частиною є також обробка помилок та перевірка стану перед виконанням кожної операції (наприклад, перевірка чи є у користувача достатньо грошей або чи можна виконати дію).

3.1.8 Створимо файл controller.js

Цей код представляє клас Controller, який є основною одиницею керування для виконання дій в грі. Він взаємодіє з іншими класами (наприклад, Executor, Validator, Console) для виконання команд користувача та обробки помилок. Детальне пояснення:

Властивості:

1. core: Являється основним об'єктом гри, з яким працює контролер. Це об'єкт гри, який містить всю логіку (наприклад, стан гри, гравців, поля тощо).
2. validator: Об'єкт, що відповідає за перевірку коректності дій. Наприклад, перевірка чи правильна команда або чи можливо виконати певну операцію в грі.
3. console: Об'єкт, який дозволяє взаємодіяти з консольним інтерфейсом гри (наприклад, через команди користувача).
4. catchErrors: Логічне значення (за замовчуванням true), яке визначає, чи мають бути оброблені помилки в процесі виконання команд. Якщо значення false, помилки будуть кидатися далі.

Методи:

1. constructor(game): Конструктор класу, який ініціалізує основні компоненти: гру, валідатор та консоль.
2. execute(username, action, options): Основний метод для виконання дій. Приймає:
    - username: Ім'я користувача, який виконує дію.
    - action: Тип дії (наприклад, "command", "buy", "move" тощо).
    - options: Додаткові параметри для дії (наприклад, аргументи для команд або опції для гри).
    - Якщо дія є командою (наприклад, консольна команда), вона передається в метод command класу Console.
    - Якщо це інша дія (наприклад, покупка або переміщення), викликається відповідний метод екземпляра Executor.

Якщо під час виконання виникає помилка, вона обробляється відповідно до налаштувань catchErrors. Якщо помилка є специфічною для гри (наприклад, помилка від ErrorGame), то вона записується в лог гри.

Як це працює:

1. execute ініціює перевірку команди або дії.
    - Якщо це команда, вона передається в Console для виконання.
    - Якщо це інша дія, то контролер перевіряє, чи існує така дія в Executor, і викликає її, передаючи необхідні опції.
2. У разі помилки:
    - Якщо catchErrors встановлено в true, помилка записується в лог гри (в залежності від типу помилки).
    - Якщо catchErrors встановлено в false, помилка просто кидається далі, щоб її можна було обробити в іншому місці.

Помилки:

Якщо виникає помилка під час виконання дії:

- Якщо помилка є гри (наприклад, неприпустима операція чи помилка в бізнес-логіці гри), вона записується в лог з повідомленням.
- Якщо помилка не є ігровою, просто записується повідомлення про помилку.

3.1.9 Створимо файл core.js

Цей код описує клас Game, який є основним компонентом для реалізації логіки гри (ймовірно, це гра, подібна до Monopoly). Він містить всю необхідну інформацію про стан гри, гравців, поле, картки і т. д. Детальне пояснення класу:

Властивості:

1. stage: Етап гри. Початково встановлений на "start", потім змінюється на "main" або "end" в залежності від прогресу гри.
2. winner: Гравець, який виграв гру. Якщо гра ще не завершена, значення winner буде null.
3. players: Об'єкт, що містить інформацію про всіх гравців, де ключем є ім'я гравця, а значенням — екземпляр класу Player.
4. tracker: Об'єкт для відстеження поточного гравця, черги ходів гравців тощо.
5. field: Об'єкт, який представляє поле гри. Це ймовірно містить інформацію про квадрати на полі (наприклад, властивості кожної клітинки, яка може бути куплена, тип клітинки тощо).
6. dices: Массив значень, що відповідає за результат кидка кубиків (ймовірно, двох кубиків).
7. logs: Лог ігрових подій, що записує всі дії у грі, наприклад, повідомлення про події, що відбулися.
8. chests і chance: Об'єкти, що представляють картки "community chest" та "chance", ймовірно, вони використовуються для випадкових подій або завдань в грі.

Методи:

1. constructor(usernames, lang="en"): Конструктор класу, який ініціалізує:

- Імена гравців (масив usernames).
- Етап гри (stage).
- Логіку трекера (Tracker).
- Поле гри (Field).
- Гравців (масив players).
- Значення кубиків (массив dices).
- Картки (відповідно до мовної конфігурації).

2. getRent(idTile): Визначає орендну плату для гравця за певний клітинок поля. У залежності від типу клітинки (наприклад, стандартна, станція або комунальна клітинка) оренда обчислюється по-різному.
3. getRepairBuilding(player, card): Визначає вартість ремонту будівель для певного гравця, якщо це передбачено карткою з колоди.
4. getCapital(username): Обчислює капітал гравця, враховуючи грошові кошти, власні клітинки та будівлі на цих клітинках.
5. dispathTile(tile, username): Обробляє події на клітинці, на яку став гравець. В залежності від типу клітинки (наприклад, може бути податок, оренда або картка з колоди), гравець може отримати або заплатити кошти.
6. disablePlayer(username): Відключає гравця від гри, коли він втрачає всі свої гроші або більше не може продовжувати гру.
7. checkWin(): Перевіряє, чи є тільки один активний гравець, що дозволяє визначити переможця гри.
8. arrest(username, next=true): Кидає гравця в тюрму і встановлює статус арешту для гравця.

9. next(): Переходить до наступного гравця в черзі, якщо умови для цього виконуються (наприклад, гравець не арештований, не відключений і не має кубиків однакових значень).
10. pushLog(type, mes, sender="system", bold=null): Додає повідомлення до журналу логів гри. Це може бути важливе повідомлення про події гри або дії гравців.

Логіка гри:

- Етапи гри:
    - start: Початковий етап гри, де створюються всі гравці і налаштовується поле.
    - main: Основний етап гри, де гравці виконують свої ходи.
    - end: Етап завершення гри, коли визначено переможця.
- Механізм переміщення гравців:
    - Гравці переміщуються по полю в залежності від результатів кидка кубиків, при цьому кожна клітинка може мати різні ефекти (наприклад, податки, оренда, челенджі).
- Перевірка перемоги: Гра завершується, коли залишається лише один активний гравець.
- Взаємодія з картками: Під час гри можуть використовуватись картки "chance" і "community chest", що дають бонуси або штрафи для гравців (наприклад, ремонт будівель чи штрафи).

3.1.10 Створимо файл executor.js

Цей код визначає клас Executor, який відповідає за виконання різних дій у грі. Клас надає методи, що взаємодіють із станом гри, перевіряють умови та записують дії для гравця.

Ось розбір основних компонентів:

1. Конструктор

- Приймає параметри: username (ім'я гравця), core (об'єкт логіки гри) та validator (перевірка умов).
- Налаштовує методи та допоміжні функції для взаємодії з грою.

2. Допоміжні методи

- _log: Записує повідомлення про дії, такі як кидок кубиків, покупка майна тощо.
- _getPlayer: Повертає об'єкт гравця, що відповідає вказаному імені (або поточного гравця).
- _getTile: Повертає майно за його ID.
- _passNewLap: Нагороджує гроші за проходження нового кола.
- _dispathTile: Визначає дії залежно від того, на яке поле потрапив гравець (наприклад, сплата оренди, взяття картки або потрапляння в тюрму).
- _next: Переходить до ходу наступного гравця.

3. Методи для ігрових дій

- ping: Надсилає пінг до системи гри.
- message: Надсилає повідомлення до журналу гри.
- roll: Кидає кубики та обробляє рух і можливі дії на полі.

- buy: Обробляє дію покупки майна.
- pledge: Керує заставою майна (іпотека).
- build: Дозволяє гравцеві будувати чи зносити будівлі на майні.
- rent: Керує оплатою оренди при потраплянні на майно іншого гравця.
- sell: Керує продажем майна та поверненням грошей гравцеві.
- tax: Обробляє сплату податків.
- deal: Ініціює торгову угоду (обмін) між гравцями.
- trade: Обробляє виконання угоди між двома гравцями.
- card: Керує діями карток, такими як рух по полю або отримання грошей.
- surrender: Дозволяє гравцеві здатися і вийти з гри.
- jailbreak: Дозволяє гравцеві вибратися з тюрми, використовуючи картку звільнення.
- disconnect: Відключає гравця та припиняє його участь у грі.

4. Дії карток (через метод card)

Метод card відображає конкретні дії, що відповідають типам карток, наприклад:

- goTo: Переміщає гравця на певне поле.
- release: Звільняє гравця з тюрми.
- money: Змінює гроші гравця.
- goToBack: Переміщає гравця назад на кілька клітинок.
- repairBuilding: Вимагає від гравця оплатити ремонт будівель.
- arrest: Затримує гравця (поміщає його в тюрму).

- happyBirthday: Розподіляє гроші між гравцями на день народження одного з них.

3.1.11 Створимо файл utils.js

Цей код реалізує систему роботи з конфігурацією і мовними налаштуваннями для програми на Node.js.

1. Імпортуємо модулі:
    - fs — стандартний модуль для роботи з файловою системою в Node.js.
    - path — стандартний модуль для роботи з шляхами до файлів.
2. Константи для шляхів до папок конфігурацій:
    - CONFIG_FOLDER — назва папки, де зберігаються конфігураційні файли (у цьому випадку, "config").
    - LANG_FOLDER — папка, що містить файли мовних локалізацій (папка "locale").
3. Функція getConfig(nameConfig)
   Ця функція приймає ім'я конфігураційного файлу, знаходить його у папці config, зчитує вміст файлу за допомогою fs.readFileSync(), і перетворює його у JavaScript-об'єкт за допомогою JSON.parse().
4. Зчитування налаштувань за замовчуванням:
   Константа settings зберігає конфігурацію, отриману з файлу settings.json, який містить основні налаштування програми.
5. Функція getConfigLang(lang)
   Ця функція призначена для завантаження файлу мовної локалізації. Вона

приймає параметр lang, що вказує на бажану мову (наприклад, "en" для англійської).

- Якщо файл для цієї мови існує в папці locale, то його вміст зчитується і повертається.
- Якщо файл для такої мови відсутній, функція завантажує локалізацію за замовчуванням (яка визначена в налаштуваннях у settings["defaultLang"]).

6. Експорт об'єктів:

Наприкінці, код експортує:

- settings — об'єкт з основними налаштуваннями програми.
- getConfig — функцію для зчитування загальних конфігураційних файлів.
- getConfigLang — функцію для зчитування мовних файлів.

Цей код дозволяє зручно працювати з конфігурацією та локалізаціями в програмі, підтримуючи можливість динамічно завантажувати налаштування та локалізації на основі запитів користувача.

3.1.12 Створимо файл validator.js

Цей код описує клас Validator, який використовується для перевірки різних умов у грі, а також для обробки помилок, що можуть виникати в процесі гри.

Опис основних частин коду:

1. Клас ErrorGame:

- Наслідує від стандартного класу Error, щоб створювати специфічні помилки для гри.
- Конструктор: приймає два параметри:
  - message — повідомлення про помилку.
  - detail (необов'язковий) — додаткова інформація, що надається разом із помилкою.
- В конструкторі створюється текст помилки, який складається з основного повідомлення і додаткової інформації, якщо вона є.
- Встановлюється ім'я помилки як "ErrorGame", а також об'єкт objLog, який містить повідомлення і деталі помилки.

2. Клас Validator:
   - Призначений для виконання перевірок умов, що стосуються гри.
   - Використовує інстанс класу Core, який представляє основну логіку гри (в даному випадку передається порожній масив при ініціалізації, але зазвичай в ньому зберігаються дані гри).
3. Методи класу Validator:
   - check(condition, message, detail=null): перевіряє певну умову. Якщо умова не виконана (тобто condition — це false), викидається помилка типу ErrorGame з відповідним повідомленням і деталями.
   - checkHasOwner(tile): перевіряє, чи є власник у даного тайлу (необхідно, щоб у тайлу був вказаний власник).
   - checkIdTile(idTile): перевіряє, чи існує тайл з заданим idTile у полі гри.
   - checkMoney(cost, money): перевіряє, чи вистачає грошей у гравця для здійснення витрат (порівнює cost і money).

- checkUsername(username): перевіряє, чи існує гравець з таким username.
- checkObjDeal(obj): перевіряє правильність структури угоди (зокрема, чи існують ініціатор та ціль угоди, чи правильно вказані гроші).
- throwError(message, detail=null): цей метод генерує помилку з повідомленням та деталями.

4. Експорт класу Validator:
   Клас Validator експортується для використання в інших модулях. Він дозволяє перевіряти різні умови і в разі невиконання умов викидати помилки, що допомагає ефективно обробляти помилки в процесі гри.

   Як працює цей код:

- Перевірка умови: У кожному методі класу Validator перевіряється певна умова (наприклад, чи є власник у тайлу або чи є у гравця достатньо грошей).
- Викидання помилок: Якщо умова не виконується, викидається спеціальна помилка (ErrorGame), яка містить повідомлення про помилку та її деталі.
- Гнучкість: Цей підхід дозволяє централізовано обробляти помилки, забезпечуючи підтримку різних перевірок і забезпечуючи зручність при налагодженні гри.

Цей код є важливою частиною ігрової логіки, оскільки гарантує, що перед будь-якою важливою дією в грі (наприклад, покупка, обмін або будівництво) перевіряються усі необхідні умови для того, щоб уникнути помилок і неправильних операцій.

3.1.13 Створимо файл cards.js

Цей код описує клас Cards, який використовується для створення колоди карт у грі, а також для їхньої шифрування та отримання нових карт.

Опис основних частин коду:

1. Імпорти:
    - settings та getConfigLang імпортуються з ../utils для доступу до загальних налаштувань гри та мовних конфігурацій.
2. Константи:
    - COUNT_CARDS — кількість карт, яка повинна бути у колоді, береться з налаштувань гри (settings["countCards"]).
3. Клас Cards:
    - Конструктор constructor(arrCardsSource, type, lang):
        - Приймає три параметри:
            - arrCardsSource — вихідний масив карт для створення колоди.
            - type — тип колоди (наприклад, “спеціальна” або “стандартна”).
            - lang — мова, яка використовується для текстових даних на картках.
        - Викликає getConfigLang(lang) для отримання локалізованих текстів карток.
        - Встановлює COUNT_CARDS карт у колоду:

- Для кожної карти в масиві arrCardsSource створюється новий об'єкт картки (objCard), який включає тип колоди (type) і локалізовані дані (textCards[objCard.id]).
- Цикл ітерації через COUNT_CARDS повторюється для заповнення колоди.
- Якщо кількість карт вичерпується (count == len), цикл повертається до початку (count = 0).

- Метод shuffle():
  - Розмішує колоду карток, перемішуючи їх випадковим чином, використовуючи алгоритм Фішера-Йетса.
  - Використовує цикл while для повторного перемішування, доки не буде оброблена остання картка.
- Метод get():
  - Повертає наступну карту з колоди (this.list[this.count++]).
  - Якщо всі карти використані (this.count == COUNT_CARDS), метод повертає колоду назад і перетасовує її (this.shuffle()), готуючи новий порядок карток.
  - Повертає об'єкт карти (card) і інкрементує this.count для відстеження поточного індексу в колоді.

4. Експорт класу Cards:
   - Клас Cards експортується для використання в інших модулях. Він надає функціонал для створення, шифрування та витягування карток з колоди, що є важливими функціями для ігрового процесу.

Як працює цей код:

- Створення колоди: В конструкторі класу Cards створюється колода карток, використовуючи вихідний масив і налаштування гри.
- Шифрування колоди: Метод shuffle() дозволяє випадковим чином перемішувати картки в колоді, забезпечуючи різні порядки карток під час кожного циклу гри.
- Витягування картки: Метод get() повертає наступну карту з колоди, а при потребі перетасовує колоду, якщо вона використана повністю. Це дозволяє забезпечити повторюваність і змінність ігрового процесу.

Цей клас є корисним інструментом для реалізації механіки гри, забезпечуючи гравцям різноманітний ігровий досвід за рахунок випадкового порядку карток і повторюваних циклів гри.

3.1.14 Створимо файл field.js

Цей клас Field, який представляє ігрове поле з певною кількістю клітин для гри, де кожна клітина може містити гравців. Клас відповідає за управління клітинами, переміщення гравців по полю, перевірку наявності гравців та інші операції, що пов'язані з полем гри.

Опис основних частин коду:

1. Імпорти:
    - getConfig імпортується з ../utils для доступу до налаштувань гри (в даному випадку для отримання інформації про тайли з файлу tiles.json).
    - Tile імпортується з модуля tile для створення об'єктів клітин (тайлів), які є частинами поля.

2. Константа COUNT_TILES:
    - Визначає кількість клітин на полі, яке має бути рівним 40, що є необхідною умовою для правильної роботи гри.
3. Клас Field:
    - Конструктор constructor(listPlayers, lang):
        - Приймає два параметри:
            - listPlayers — список гравців, які будуть додані до поля.
            - lang — мова для локалізації текстів клітин.
        - Зчитує конфігурацію тайлів з файлу tiles.json за допомогою getConfig("tiles.json").
        - Перевіряє, що кількість тайлів в конфігурації дорівнює 40.
        - Для кожної клітини генерується унікальний ID (якщо він не заданий) через функцію genId(). Якщо в клітині не вказаний ID, створюється ID на основі її параметрів (наприклад, кольору або типу).
        - Створює масив this.tiles, який містить об'єкти типу Tile, де кожен тайл ініціалізується з конфігурації і додатково локалізується для певної мови.
        - Для клітини з ID "start" додаються всі гравці до цієї клітини як стартова точка.
    - Методи класу Field:
        - getById(idTile):
            - Повертає клітину по її ID.
            - Якщо клітина з таким ID не знайдена, кидається помилка.

- findPlayer(username):
  - Повертає клітину, де знаходиться гравець з певним іменем.
  - Якщо гравець не знайдений, кидається помилка.
- getIndexTile(tile):
  - Повертає індекс клітини в масиві this.tiles.
- replacePlayer(username, newIndex):
  - Переміщує гравця з одного тайла на інший (за новим індексом).
  - Якщо новий індекс -1, гравець видаляється з поля.
  - Якщо індекс некоректний (менше за 0 або більший за кількість тайлів), кидається помилка.
- move(username, countSteps):
  - Переміщає гравця за певну кількість кроків.
  - Функція перевіряє, чи потрібно переходити на новий коло (якщо поточний індекс більший за новий).
  - Якщо кроки переносять гравця за межі поля, вони автоматично повертаються на початок.
- moveById(username, idTile):
  - Переміщає гравця на тайл за його ID.
  - Якщо гравець пересувається на тайл з меншим індексом (повертається на початок), повертається значення true, в іншому випадку — false.

4. Експорт класу Field:

- Клас експортується для використання в інших частинах програми. Це дозволяє створювати об'єкти поля, взаємодіяти з клітинами та гравцями, а також керувати ігровим процесом.

Як працює цей код:

- Створення поля: Конструктор класу Field генерує поле з 40 клітин (тайлів), де кожен тайл отримує унікальний ID та відповідну локалізацію на основі конфігурацій.
- Переміщення гравців: Методи move і moveById дозволяють переміщати гравців по полю в залежності від їхніх кроків або ID тайлів, з перевіркою на коректність переміщення та наявність гравців.
- Робота з клітинами: Клас дозволяє знаходити тайли за їх ID, а також управляти гравцями на конкретних клітинах, додаючи або видаляючи їх з тайлів.

Цей клас є важливою частиною гри, оскільки він управляє механізмом ігрового поля, на якому відбувається переміщення гравців, а також перевіряє коректність всіх операцій з клітинами та гравцями.

3.1.15 Створимо файл player.js

Цей код описує клас Player, який представляє гравця в грі. Клас має властивості та методи для управління гравцем, його активами, грошей, монополіями (власність на клітини поля), послугами та іншими аспектами, пов'язаними з грою. Ось розбір основних частин цього коду:

Опис основних частин коду:

1. Імпорт:

- getConfig імпортується з ../utils, щоб отримати конфігурацію гри (в даному випадку інформацію про тайли поля з файлу tiles.json).

2. Конструктор constructor(username, color, money):
    - Приймає три параметри:
        - username — ім'я гравця.
        - color — колір гравця.
        - money — початковий баланс гравця.
    - Ініціалізує властивості:
        - disable — позначає, чи гравець неактивний (встановлено в false за замовчуванням).
        - username — ім'я гравця.
        - color — колір гравця.
        - money — сума грошей, якими володіє гравець.
        - own — масив, що містить ID тайлів, які належать гравцеві.
        - releasePrison — кількість спроб вийти з в'язниці.
        - arrested — кількість разів, коли гравець був заарештований.
        - service — об'єкт, що зберігає поточні послуги, надані гравцю (наприклад, пропозиції, оренда, угоди).
        - monopoly — об'єкт, що зберігає кількість володінь кожного типу клітин (наприклад, для монополії).
3. Методи класу Player:
    - _canBuy(tile):
        - Перевіряє, чи може гравець купити даний тайл.

    - Гравець може купити лише клітини типу "standard", "communal" або "station".

- _getKey(tile):
    - Повертає унікальний ключ для кожної клітини, що дозволяє відслідковувати монополії гравця.
    - Ключ формується на основі кольору клітини або її типу (якщо колір відсутній).
- addOwn(tile):
    - Додає клітину до власності гравця.
    - Оновлюється властивість owner клітини, додається її ID до масиву own, і збільшується лічильник монополії для цього типу клітини.
- removeOwn(tile):
    - Видаляє клітину з власності гравця.
    - Оновлюється властивість owner клітини, видаляється її ID з масиву own, і зменшується лічильник монополії для цього типу клітини.
- transfer(tile, receivingPlayer):
    - Переміщує клітину від поточного гравця до іншого гравця.
    - Для цього клітина спочатку видаляється з власності поточного гравця, а потім додається до власності отримувача.
- setService(setting, val):
    - Встановлює значення для певної послуги гравця.

- Наприклад, може встановлювати значення для пропозицій, оренди, угод тощо.

- clearService(setting):
  - Очищає значення для певної послуги гравця.
- resetServices():
  - Очищає всі послуги для гравця, скидаючи їх значення на null.

4. Експорт класу Player:
   - Клас експортується для використання в інших частинах програми. Це дозволяє створювати об'єкти гравців, керувати їх власністю, грошима та послугами.

Як працює цей код:

- Створення гравця: Конструктор класу ініціалізує гравця з базовими параметрами, такими як ім'я, колір і кількість грошей. Також гравець отримує можливість володіти певними клітинами та отримувати послуги.
- Управління власністю: Гравець може додавати і видаляти клітини зі своєї власності, а також передавати їх іншим гравцям. Це дозволяє йому управляти своїми активами в грі.
- Послуги: Гравець має набір послуг, що можуть бути налаштовані, очищені або скинуті. Це може включати оренду, пропозиції, угоди тощо.
- Монополія: Гравець може мати монополії на певні типи клітин, і це відображається в його об'єкті monopoly, що дозволяє вести облік його власності на різних типах клітин.

Цей клас є важливою частиною механіки гри, оскільки він забезпечує управління гравцями, їхніми активами та фінансами.

3.1.16 Створимо файл tile.js

Цей код описує клас Tile, який представляє клітину на ігровому полі (ймовірно, в грі типу "Монополія"). Клас містить властивості та методи для управління клітинами, їхнім власником, будівлями, а також гравцями, що перебувають на цих клітинах.

Опис основних частин коду:

1. Імпорт:
    - getConfig і getConfigLang імпортуються з ../utils. Вони використовуються для отримання налаштувань гри та локалізованих даних для клітин з файлів конфігурації (наприклад, tiles.json).
2. Конструктор constructor(tileConfig, lang):
    - Приймає два параметри:
        - tileConfig — конфігурація клітини, що містить її основні властивості (тип, колір, ціна тощо).
        - lang — мова, яка використовується для локалізації тексту клітини.
    - У конструкторі ініціалізуються наступні властивості:
        - Object.assign(this, tileConfig) — копіює всі властивості з tileConfig в екземпляр класу Tile.
        - type — тип клітини (наприклад, "standard", "communal", "community_chest" тощо).
        - В залежності від типу клітини, додаються специфічні параметри:

- Для клітин типу "standard" (стандартні клітини) додатково визначаються ціна на будівлю, кількість клітин в зоні і змінні для будівництва та готелю.
- Для "communal" (громадські клітини) додається опис з локалізованих даних.

- Для "community_chest" та "chance" клітин встановлюються назви з файлу локалізації.
- players — масив гравців, що знаходяться на цій клітині.
- owner — власник клітини (якщо є).
- pledge — вказує, чи було заставлено майно.
- canBuy — визначає, чи можна придбати клітину (тільки для типів "standard", "communal" і "station").

3. Методи класу Tile:

- addPlayer(username):
  - Додає гравця до клітини, якщо він ще не знаходиться на цій клітині.
  - Повертає true, якщо гравець успішно доданий, і false в іншому випадку.
- removePlayer(username):
  - Видаляє гравця з клітини, якщо він там є.
  - Повертає true, якщо гравець успішно видалений, і false, якщо такого гравця немає на клітині.
- playerIn(username):

- Перевіряє, чи є гравець на клітині. Повертає true, якщо гравець знаходиться на клітині, і false в іншому випадку.

- addBuilding():
  - Додає будівлю на клітину (можна додавати будівлі, поки їх кількість не досягне 5, після чого встановлюється готель).
  - Повертає true, якщо будівлю вдалося додати, і false, якщо на клітині вже є готель.

- removeBuilding():
  - Видаляє будівлю з клітини (не можна видаляти будівлю, якщо їх немає).
  - Повертає true, якщо будівлю вдалося видалити, і false, якщо будівлі немає.

- resetBuilding():
  - Скидає всі будівельні зміни на клітині (сбрасывает кількість будівель і готель).

Важливі аспекти:

- Типи клітин: Клас дозволяє працювати з різними типами клітин, такими як "standard", "communal", "community_chest" і "chance". Для кожного типу можуть бути різні правила і властивості.
- Власність і будівництво: Клас підтримує додавання/видалення гравців на клітини, а також управління будівництвом. Клітини можуть мати будівлі, і після досягнення певного ліміту (5 будівель) клітина перетворюється на готель.

- Механізм локалізації: Для підтримки різних мов (через параметр lang в конструкторі) клас використовує локалізовані дані для назв і описів клітин. Це дозволяє змінювати текст на різні мови залежно від налаштувань гри.

Цей клас забезпечує всі необхідні методи для управління клітиною на ігровому полі, враховуючи локалізацію, типи клітин, власність, будівництво та взаємодію з гравцями.

3.1.17 Створимо файл tracker.js

Це клас Tracker, який використовується для відслідковування порядку ходів гравців у грі, що включає в себе облік результатів кубиків (ймовірно, для гри типу "Монополія" або подібної). Клас допомагає керувати поточним гравцем і визначати, хто повинен ходити далі, враховуючи результати кидка кубиків.

Опис основних частин коду:

1. Конструктор constructor(countPlayers):
    - Приймає параметр countPlayers, що вказує на кількість гравців у грі.
    - this.current — поточний гравець, який має ходити.
    - this.order — масив гравців, відсортованих за результатами кидка кубиків.
    - this.countPlayers — кількість гравців.
    - this.valuesDices — об'єкт, що зберігає результат кидка кубиків для кожного гравця. Ключами є імена гравців, а значеннями — масиви з результатами кубиків.
2. Метод _setOrder():

- Приватний метод, що встановлює порядок ходів гравців.
- Він перевіряє, чи всі гравці подали свої результати кидка кубиків (якщо кількість гравців дорівнює countPlayers).
- Для кожного гравця обчислюється сума результатів його кидка кубиків. Гравці сортуються за цією сумою, а гравець з найвищим результатом стає першим у черзі.
- Після сортування порядок гравців зберігається в this.order, а поточний гравець встановлюється на першого в порядку.

3. Метод removePlayer(username):
    - Видаляє гравця з гри, зменшуючи кількість гравців (countPlayers).
    - Видаляє результат кидка кубиків для цього гравця з об'єкта valuesDices.
    - Перевизначає порядок гравців, викликаючи метод _setOrder().
4. Метод setDiceValue(username, dices):
    - Встановлює результат кидка кубиків для конкретного гравця (username).
    - Результат передається як масив значень кубиків (dices).
    - Після цього викликається метод _setOrder(), щоб оновити порядок гравців.
5. Метод next():
    - Визначає наступного гравця, хто має ходити.
    - Він знаходить поточного гравця в масиві this.order і обирає наступного.
    - Якщо поточний гравець був останнім в черзі, наступним стає перший.
    - Повертає ім'я наступного гравця.

Важливі аспекти:

- Оновлення порядку гравців: Порядок ходів гравців змінюється після кожного кидка кубиків, щоб врахувати їхні результати.
- Динамічне оновлення: Якщо гравець покидає гру (використовується метод removePlayer()), порядок гравців оновлюється автоматично.
- Чітка черговість: Метод next() дозволяє відслідковувати, хто повинен ходити далі, що важливо для управління чергою в іграх з кількома учасниками.

Клас Tracker дозволяє зручно відслідковувати результати кидків кубиків для гравців, сортувати їх за результатами та автоматично управляти чергою гравців у грі. Це забезпечує зручний механізм для гри, де важливе правильне визначення порядку ходів на основі випадкових результатів.

3.2 Розробка клієнтської частини веб-застосунку

3.2.1 Загальна структура клієнтської частини

Для початку заходимо в кореневу папку нашого проекту та прописуємо npm install -g @vue/cli. Ця команда завантажить та встановить vue у систему та дозволить створювати нові проекти на vue.js. Далі введемо vue create “nameFolder” ця команда створить папку де буде знаходитися клієнтська частина проекту.

Для встановлення додаткових залежностей визиваємо термінал через Menu – Terminal – New Terminal та заходимо в кореневу папку клієнтської частини коду та вводимо наступні команди для тієї чи іншої технології.

- Бібліотека axios

npm i axios

- Бібліотека socket.io

npm install socket.io-client

Тепер створимо структуру клієнтської частини проекту.

Monopoly >>client>> public

>> src>>api >>sevices

>>components>>common

>>router >>game>>cards

>>store >>Field

>>views >>Panel

>>PlayerTable

>>WorkSpace

3.2.2 Компоненти гри «Монополія»

Папка components є важливою частиною клієнтської частини веб-застосунку. Вона містить компоненти Vue.js, які відповідають за різні частини користувацького інтерфейсу. Структура цієї папки організована таким чином, щоб забезпечити легкість у підтримці та масштабуванні проєкту у майбутньому. Усі компоненти згруповані за їх функціональністю та використанням.

1. Підкаталоги та їх призначення

- common: Цей підкаталог містить загальні компоненти, які можуть використовуватися в різних частинах додатку. Наприклад, ButtonMain.vue - це базовий компонент кнопки, який може бути використаний повторно на різних сторінках.

- game: Найважливіший підкаталог, що містить компоненти, специфічні для гри.
    - cards: У цьому підкаталозі знаходяться компоненти, пов'язані з картами гри. Наприклад, CardTile.vue відповідає за відображення кожної ігрової карти.
    - Field: Компоненти, що відображають ігрове поле. Наприклад, FieldMain.vue представляє основне поле гри.
    - Panel: Цей підкаталог містить панелі з інформацією або опціями для гравців. Наприклад, PanelMain.vue відображає основну панель з інформацією.
    - PlayerTable: Компоненти, пов'язані з таблицями гравців. Наприклад, PlayerTableMain.vue відображає таблицю з інформацією про всіх гравців.
    - Workspace: Робоче середовище гри, де відбуваються основні дії. Наприклад, WorkspaceMain.vue є основним компонентом робочого середовища.

2. Опис ключових компонентів

- ButtonMain.vue: Базовий компонент кнопки, який використовується на різних сторінках додатку. Він забезпечує стандартний вигляд і поведінку кнопок.
- CardTile.vue: Компонент, що відображає ігрові карти. Він використовується для представлення різних типів карт, таких як майно або випадкові події.
- FieldMain.vue: Основний компонент поля, який відображає ігрове поле і розміщення гравців на ньому.
- PanelMain.vue: Панель, яка містить важливу інформацію для гравців, таку як баланси, кількість карт та інші ресурси.

- PlayerTableMain.vue: Таблиця, що показує інформацію про гравців, їх статки та майно.
- WorkspaceMain.vue: Основний компонент робочого середовища, де відбуваються основні ігрові дії, такі як кидання кубика та пересування фішок.

3.2.3 Ігрові екрани та їх функціональність

1 Опис основних екранів гри

У процесі розробки клієнтської частини гри Monopoly було створено декілька ключових екранів, які забезпечують основний функціонал гри. Кожен з цих екранів відповідає за певний етап взаємодії користувача з додатком.

2 Створення кімнати (компонент CreateRoom)

Екран створення кімнати дозволяє користувачам створювати нові ігрові кімнати, де інші гравці можуть приєднуватися для гри. Компонент CreateRoom містить форму для введення необхідної інформації для створення нової кімнати, як-от назва кімнати, максимальна кількість гравців та інші параметри.

3 Список кімнат (компонент ListRooms)

Екран списку кімнат відображає всі доступні кімнати, до яких можуть приєднатися гравці. Компонент ListRooms містить таблицю або список з інформацією про кожну кімнату, таку як назва, кількість гравців та кнопка для приєднання.

4 Панель управління грою (компоненти поля та панелі гравця в Workspace)

Панель управління грою є основним екраном, де відбувається сама гра. Вона складається з різних компонентів, таких як поле гри та панелі гравця, які

розташовані у робочому середовищі (Workspace). Ці компоненти забезпечують інтерактивний досвід гри, дозволяючи гравцям кидати кубики, переміщувати фішки та здійснювати інші дії.

Для кожного екрану представлено приклади коду, які демонструють основні елементи та функціональність. Це дає зрозуміти, як компоненти взаємодіють між собою і з користувачем.

3.2.4 Управління станом гри за допомогою Vue

1 Опис стану гри

Управління станом гри є важливим аспектом розробки додатку Monopoly. Для цього використовується бібліотека Vuex, яка дозволяє централізовано керувати станом додатку. У store зберігаються такі дані:

- Стан поля: інформація про розташування кожної клітинки, власників майна, розташування фішок гравців тощо.
- Інформація про гравців: дані про кожного гравця, включаючи ім'я, баланс, власність та інші ресурси.
- Хід гри: стан, який відображає поточний хід, чергу гравців, стан кубиків та інші важливі параметри гри.

2 Структура модулів Vuex

Для зручності та структурованості, стан гри в додатку Monopoly організовано за допомогою модулів Vuex.

3 Модуль deal.js

Модуль deal.js відповідає за управління угодами між гравцями. Він містить інформацію про активні пропозиції, прийняті угоди та логіку для проведення угод.

4 Модуль field.js

Модуль field.js зберігає стан клітинок ігрового поля. Він містить інформацію про власників клітинок, розташування фішок та інші параметри поля.

3.2.5 Маршрутизація клієнтської частини

1. Реалізація маршрутизації через Vue Router

Для управління маршрутизацією в клієнтській частині додатку Monopoly використовуємо бібліотеку Vue Router. Вона дозволяє створювати динамічні, багатосторінкові веб-додатки з легкістю. Vue Router інтегрується з Vue.js і забезпечує плавну навігацію між сторінками додатку без перезавантаження браузера.

2. Структура маршрутів (src/router/index.js)

У проєкті структура маршрутів організована в файлі src/router/index.js. Визначаємо різні маршрути, які відповідають різним частинам нашого додатку.

Приклад коду наведений в додатку А.

3.2.5 Реалізація обміну даними в реальному часі через WebSocket

1. Використання WebSocket для синхронізації гри між клієнтами

Для забезпечення обміну даними в реальному часі між клієнтами гри Monopoly використовуються WebSocket. WebSocket дозволяє встановлювати постійне з'єднання між клієнтами та сервером, що значно зменшує затримки при

передачі даних. Це є ключовим для синхронізації стану гри, де кожна дія одного гравця повинна миттєво відображатися у всіх інших гравців.

2. Реалізація логіки у файлі socket.js

У файлі socket.js реалізовано логіку для підключення до WebSocket сервера, обробки подій та передачі даних між клієнтами.

1. Подія "responseCreateRoom": Ця подія активується після спроби створення кімнати. Якщо кімната успішно створена, користувач перенаправляється на сторінку гри.
2. Подія "listRooms": Ця подія активується для отримання списку доступних кімнат.
3. Подія "responseEntryLobby": Ця подія активується після спроби входу в лобі. Якщо вхід успішний, зберігається ім'я користувача.
4. Подія "dataRoom": Ця подія активується для отримання даних про гравців у кімнаті.
5. Подія "updateGame": Ця подія активується для оновлення стану гри на клієнті.
6. Подія "reconnect": Ця подія активується при повторному підключенні до WebSocket сервера.
7. Подія "connect": Ця подія активується при підключенні до WebSocket сервера.
8. Подія "disconnect": Ця подія активується при відключенні від WebSocket сервера.

6.4 Відправка подій на сервер

1. gameApi: Функція для відправки команд гри на сервер.
2. leave: Функція для виходу з лобі.

5. Переваги використання WebSocket

Використання WebSocket для обміну даними в реальному часі має кілька переваг:

- Низька затримка: завдяки постійному з'єднанню між клієнтами і сервером.
- Спілкування: дані можуть відправлятися як з клієнта на сервер, так і з серверу на клієнт без необхідності повторних запитів.
- Ефективність: скорочення кількості HTTP-запитів, що знижує навантаження на сервер і мережу.

Приклад коду наведений в додатку А.

3.2.6 Сервісні компоненти та допоміжні вікна

Опис компонентів

У процесі розробки гри «Monopoly» створено декілька сервісних компонентів та допоміжних вікон, які забезпечують додаткову функціональність та покращують користувацький досвід. Ці компоненти використовуються для відображення результатів гри, повідомлень про помилки та реєстрації гравців.

1 EndWindow

Компонент EndWindow використовується для відображення результатів гри. Він показує підсумкові результати, такі як переможці та інші важливі статистики гри. Цей компонент зазвичай викликається після завершення гри.

2 OopsWindow

Компонент OopsWindow використовується для відображення повідомлень про помилки. Він показує користувачеві, що сталася помилка, і надає інформацію

про те, що робити далі. Цей компонент може викликатися у випадку технічних неполадок або неправильних дій з боку користувача.

3 PlayerRegister

Компонент PlayerRegister використовується для реєстрації гравців. Він надає форму для введення імені гравця та інших необхідних даних. Цей компонент зазвичай викликається на початку гри, коли користувачі приєднуються до ігрової кімнати.

Призначення компонентів.

- EndWindow: Відображення результатів гри після її завершення. Показує підсумкові результати та статистики.
- OopsWindow: Відображення повідомлень про помилки. Інформує користувача про те, що пішло не так, і що робити далі.
- PlayerRegister: Реєстрація гравців перед початком гри. Надає форму для введення імені та інших необхідних даних.

Приклад коду наведений в додатку А.

3.2.7 Взаємодія з сервером через Api запити

Реалізація запитів до бекенду в api/menu.js або інших модулях

Для взаємодії з сервером та отримання даних у реальному часі в додатку Monopoly використовується бібліотека axios. Вона забезпечує простий і зручний спосіб виконання HTTP-запитів до бекенду. Приклади коду наведені в додатку А.

3.3 Демонстрація результату розробленого додатку

Після розробки ми можемо його протестувати на помилки. Перейдемо на домен 127.0.0.1:5500 та перевіримо функціональність веб-застосунку де ми можемо побачити початкове вікно гри (рис.3.1).

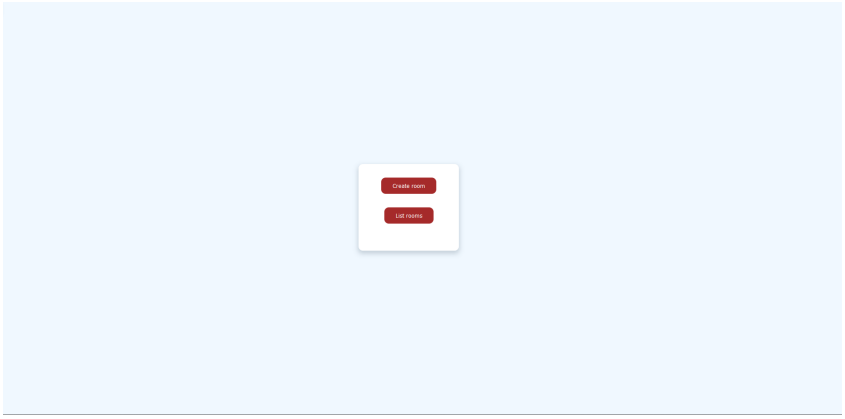

Рисунок 3.1 - Початкове вікно веб-застосунку

Як тільки створено кімнату та запрошено декілька гравців можна запустити сесію та побачити ігрове вікно гри (рис 3.2). Процес та розробка гри базується на цих правилах (рис 3.3).

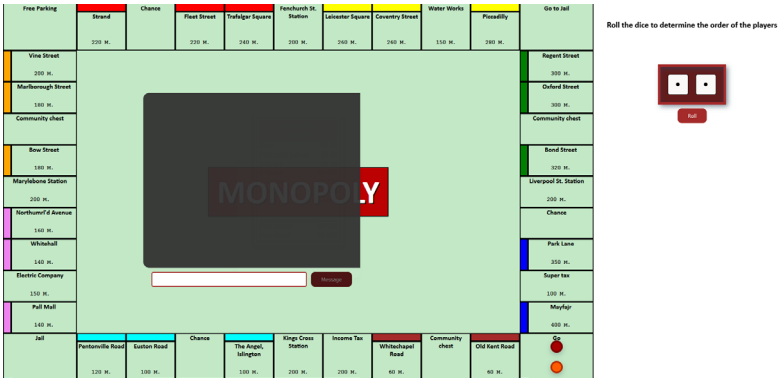

Рисунок 3.2 - Ігрове вікно веб-застосунку

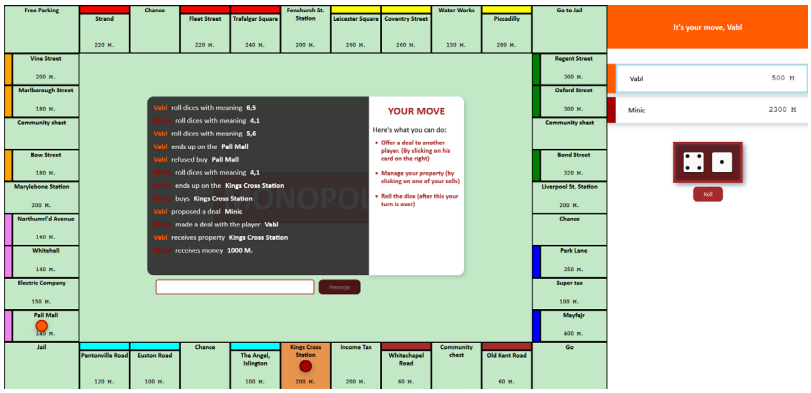

Рисунок – 3.3 Процес гри

1. Ціль гри: Стати найзаможнішим гравцем, купуючи, орендуючи та торгуючи власністю, а також довівши до банкрутства суперників.
2. Початок гри:
    - Кожен гравець отримує стартовий капітал.
    - Порядок ходів визначається киданням двох кубиків (гравець із найвищим результатом починає).
3. Ігровий процес:
    - У свій хід гравець кидає два кубики та пересуває свою фішку вперед на кількість клітин, що дорівнює сумі кубиків.
    - Якщо випадає дубль (однакові значення на обох кубиках), гравець отримує додатковий хід. Якщо дубль випадає тричі поспіль, гравець потрапляє у в'язницю.
4. Купівля власності:
    - Якщо гравець зупиняється на незайнятій ділянці, він може придбати її за вартість, вказану на картці.
5. Оренда:

- Якщо гравець зупиняється на ділянці, що належить іншому гравцеві, він має сплатити оренду. Розмір оренди залежить від вартості ділянки та наявності будівель на ній.

6. Будівництво:
    - Гравці можуть будувати будинки та готелі на своїх ділянках, якщо вони володіють усіма ділянками одного кольору.
7. Карти "Шанс" та "Громадська скарбниця":
    - Гравці, які зупиняються на цих клітинках, витягують відповідну картку, що може принести нагороду чи штраф.
8. В'язниця:
    - Гравець потрапляє у в'язницю, якщо:
        - Зупиняється на клітинці "В'язниця";
        - Тричі поспіль викидає дубль;
        - Витягує відповідну карту.
    - В'язницю можна покинути:
        - Сплативши штраф;
        - Використавши картку "Вийти з в'язниці безкоштовно";
9. Банкрутство:
    - Гравець оголошується банкрутом, якщо не може сплатити борги. Усі його активи передаються кредитору чи банку.

Гра триває, доки не залишиться один гравець, або доки гравці не домовляться завершити гру (у цьому випадку перемагає найбагатший).

3.4 Висновки до третього розділу

У третьому розділі було реалізовано клієнтську частину веб-застосунку гри "Монополія". Детальний аналіз і розробка включали:

1. Загальну структуру клієнтської частини — спроектовано основні компоненти клієнтської частини з акцентом на їхню модульність і інтеграцію із серверною частиною.
2. Компоненти гри "Монополія" — створено ключові елементи інтерфейсу гри, які забезпечують інтуїтивно зрозумілу взаємодію для користувачів.
3. Ігрові екрани та їх функціональність — розроблено логіку відображення ігрового поля, меню та інших візуальних компонентів.
4. Управління станом гри — впроваджено механізми управління даними в реальному часі з використанням сучасних бібліотек для роботи зі станом.
5. Сервісні компоненти — реалізовано функції допоміжних вікон, таких як чат і історія дій гравців.
6. Інтеграцію з сервером через API — забезпечено стабільний обмін даними між клієнтом і сервером для синхронізації ігрового процесу.

У результаті реалізації клієнтської частини створено функціональний ігровий веб-застосунок, який відповідає сучасним вимогам щодо інтерфейсу та продуктивності. Робота проведена на основі інтеграції з серверною частиною, що забезпечує стабільний і динамічний ігровий процес для користувачів.

ВИСНОВКИ

У процесі виконання дослідження були досягнуті наступні результати:

1. Аналіз предметної області
   Було вивчено поняття веб-сайтів, їхні функції та особливості використання. Проаналізовано існуючі приклади веб-сайтів, що дозволило сформулювати основні вимоги до створення проєкту.

2. Дослідження засобів для розробки веб-застосунків
   Розглянуто сучасні технології розробки клієнтської та серверної частин. Оцінено їх переваги та недоліки, що стало основою для вибору найбільш підходящих засобів для реалізації поставлених завдань.

3. Розробка веб-застосунку
   Реалізовано проєкт відповідно до визначених вимог. Створено клієнтську та серверну частини застосунку, що забезпечують інтерактивність та функціональність. Під час демонстрації результату було підтверджено відповідність реалізації поставленим цілям.

Результати дослідження та реалізації доводять доцільність обраного підходу та обраних інструментів розробки. Створений веб-застосунок демонструє ефективність сучасних технологій у розв'язанні поставлених завдань.

ПЕРЕЛІК ПОСИЛАНЬ

1. Haverbeke, M. (2018). Eloquent JavaScript: A Modern Introduction to Programming (3rd ed.). No Starch Press.
2. Crockford, D. (2008). JavaScript: The Good Parts. O'Reilly Media.
3. Simpson, K. (2020). You Don’t Know JS (Yet): Get Started. O'Reilly Media.
4. Copes, F. (2018). The Vue.js Handbook. Self-published.
5. Djirdeh, H., Murray, N., & Lerner, A. (2018). Fullstack Vue: The Complete Guide to Vue.js. Fullstack.io.
6. Wieruch, R. (2020). The Road to React: Your Journey to Mastery. Self-published.
7. Gordon, Z. (2019). React Explained. Self-published.
8. Duckett, J. (2011). HTML and CSS: Design and Build Websites. Wiley.
9. Duckett, J. (2014). JavaScript and JQuery: Interactive Front-End Web Development. Wiley.
10. Robbins, J. (2018). Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (5th ed.). O'Reilly Media.
11. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
12. Hunt, A., & Thomas, D. (2019). The Pragmatic Programmer: Your Journey to Mastery (2nd ed.). Addison-Wesley.
13. Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley.

Додаток А КОД-ВЕБ САЙТУ

Основний код веб сайту

Menu.js

```
import axios from "axios"

const baseUrl = "/api"

export async function createRoom(title) {

    if (!title) return {ok: false, desc: "Did not enter the name of the room"}

    const url = baseUrl + "/create"

    const response = await axios.post(url, {title})

    return response.data}

export async function getListRooms() {

    const url = baseUrl + "/list"

    const response = await axios.get(url)

    return response.data}
```

EndWindow.js

```
<template>

    <WindowComponent :fullscreen="true"><h1>

           {{ $t("game.end.label") }}

           <span class="winner" :style="{color: winner.color.primary}">

                {{ winner.username }}

            </span> </h1><p>{{ $t("game.end.desc") }}</p>

<template v-slot:btns>

            <ButtonMain @click="clickLeave">{{ $t("buttons.mainMenu")
}}</ButtonMain></template>

    </WindowComponent></template><script>

import WindowComponent from '@/components/common/WindowComponent.vue'

import ButtonMain from '@/components/common/ButtonMain.vue'

import { mapState } from "vuex"

import { leave } from '@/socket';
```

```
export default {
    name: "EndWindow",
    components: {
        WindowComponent,
        ButtonMain},
    computed: mapState({
        winner: (state) => state.game.winner  }),methods: {
        clickLeave() {
            leave({noUpdate: true})
            this.$router.push("/")}}}</script>
```

Socket.js

```
class WrapSocket {
    constructor(originalSocket=new Socket(), server=new Server()) {
        this.socket = originalSocket
        this.server = server
        this.data = {
            room: null,
            username: ""}
     const linkEvent = (event, func) => {
            this.socket.on(event, func.bind(this))}
        linkEvent("createRoom", this.receiveCreateRoom)
        linkEvent("pingRooms", this.receivePingRooms)
        linkEvent("pongRooms", this.receivePongRooms)
        linkEvent("pingGame", this.receivePingGame)
        linkEvent("entryLobby", this.receiveEntryLobby)
        linkEvent("leaveLobby", this.receiveLeaveLobby)
        linkEvent("startGame", this.receiveStartGame)
        linkEvent("game", this._useGame)
        linkEvent("disconnecting", this.receiveLeaveLobby)}
```

```
_useGame(command, options) {
    const {username, room} = this.data
    const controller = new Controller(room.game)
    controller.execute(username, command, options)
    this.sendUpdateGame()}
receiveCreateRoom(title) {
    const result = roomManager.createRoom(title)
    const res = {
        title,
        status: result,
        desc: result ? "ok" : "existRoom"}
    this.socket.emit("responseCreateRoom", res)
    this.sendListRooms()}
receivePingRooms() {
    this.socket.join(FINDING_ROOMS)
    this.sendListRooms()}
receivePongRooms() {
    this.socket.leave(FINDING_ROOMS) }
receivePingGame() {
    const { room, username } = this.data
    if (room && room.game) {
        let objPlayer = Object.values(room.players).find(player => !player.connect)
        if (!objPlayer) return
        clearTimeout(objPlayer.timerReconnect)
        objPlayer.timerReconnect = null
        objPlayer.idSocket = this.socket.id
        this.socket.join(room.title)
        this.socket.emit("reconnect", username)
        this._useGame("message", {message: "reconnect"})}    }
```

```
receiveEntryLobby(titleRoom, username) {
    const room = roomManager.rooms[titleRoom]
    let status = false
    let desc = ""
    if (room) [status, desc] = room.addPlayer(username, this.socket.id)
    else desc = "nonExist"
    this.socket.emit("responseEntryLobby", {username, status, desc})
    if (status) {
        this.socket.leave(FINDING_ROOMS)
        this.sendListRooms()
        this.data = { room, username }
        this.socket.join(titleRoom)
        this.server.to(titleRoom).emit("dataRoom", room.players) }}
receiveLeaveLobby(options={}) {
    if (!this.data.room) return
    const leave = () => {
        room.removePlayerById(this.socket.id)
        this.socket.to(room.title).emit("dataRoom", room.players)
        this.socket.leave(room.title)
        if (0 == room.getCountPlayers()) roomManager.deleteRoom(room.title)
        this.sendListRooms()}
    const { room, username } = this.data
    if (room.game) {
        const mesGame = (message) => {
            if (!options.noUpdate) this._useGame("message", {message})}
        if (options.force) {
            mesGame("leave game!")
            if (!options.noUpdate) this._useGame("disconnect")
            leave()  } else {
            mesGame("connection lost. Trying to reconnect")
```

```
                const player = room.players[username]
                player.connect = false
                player.timerReconnect = setTimeout(() => {
                    if (!options.noUpdate) this._useGame("disconnect")
                    leave()
                }, (process.env.NODE_ENV == "test") ? 0 : 8000) }
} else leave()}
    receiveStartGame(options={}) {
        this.data.room.startGame(options)
        this.sendUpdateGame()}
    sendListRooms() {
        const list = roomManager.getDataRooms()
        const filteredList =  list.filter(room => room.count < LIMIT_ROOM && room.status ==
"lobby")
        this.server.to(FINDING_ROOMS).emit("listRooms", filteredList) }
    sendUpdateGame() {
        const { title, game } = this.data.room
        this.server.to(title).emit("updateGame", game) }}
```

index.js

```
const staticFolderPath = path.join(__dirname, "client", "dist")
const app = require("./server/app")
const WrapSocket = require("./server/socket")
app.use(express.static(staticFolderPath))
app.get("*", (req, res) => {
    res.sendFile(path.join(staticFolderPath, "index.html"))})
dotenv.config()
const server = http.createServer(app)
const io = new Server(server)
io.on("connection", (socket) => new WrapSocket(socket, io))
const PORT = process.env.PORT ?? 5500
server.listen(PORT, () => console.log("Server start http://127.0.0.1:" + PORT))
```

Додаток Б ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

1

ДИПЛОМНА РОБОТА
на ступінь вищої освіти магістр
із спеціальності 122 Комп'ютерні науки

Веб застосунок настільної гри «монополія» на основі JavaScript

Виконав: студент 6 курсу, групи КНДМ-61
Омелянчук Ілля Володимирович
Керівник: завідувач кафедри Комп'ютерних наук
д.т.н., професор Звенігородський О.С.

Київ - 2024

2

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Тема	Веб застосунок настільної гри «монополія» на основі «JavaScript»
Мета дослідження	Підвищити реалістичність ігрового процесу з використанням технологій JavaScript для обміну даними в реальному часі.
Наукове завдання	Розробка веб-застосунку для гри "Монополія" з підтримкою реального часу та інтерактивними можливостями для користувачів на основі сучасних веб-технологій.
Об'єкт дослідження	Процес розробки клієнт-серверних веб-застосунків для інтерактивних ігор
Предмет дослідження	Інструменти та технології для розробки клієнтської та серверної частини гри з функціями управління станом гри, обміну даними та обробки логіки

3

Актуальність:

- Високий попит на багатокористувацькі онлайн-ігри.

- Необхідність створення ефективних та зручних інструментів для їх розробки.

- Використання сучасних веб-технологій для інтерактивних систем.

Аналіз предметної області:

- Поняття та функції веб-сайту.

- Огляд існуючих веб-сайтів.

- Вимоги до проекту.

Технології розробки:

- Серверна частина: Node.js, Express, Socket.io.

- Клієнтська частина: Vue.js, Axios.

- Інструменти: Visual Studio Code, nodemon, dotenv.

Архітектура системи:

- Серверна частина: обробка запитів, управління кімнатами, логіка гри.

- Клієнтська частина: інтерфейс користувача, взаємодія з сервером, відображенням гри

Реалізація серверної частини:

- Створення серверу на основі Express.

- Використання Socket.io для реального часу.

- Управління кімнатами та гравцями.

8

Реалізація клієнтської частини:

- Використання Vue.js для створення інтерфейсу.

- Axios для взаємодії з сервером.

- Компоненти гри: поле, карти, панелі гравців.

9

Тестування та результати:

- Тестування функціональності гри.

- Виправлення помилок.

- Демонстрація роботи веб-застосунку.

10

ВИСНОВКИ

- Досягнуті результати.

- Перспективи розвитку.