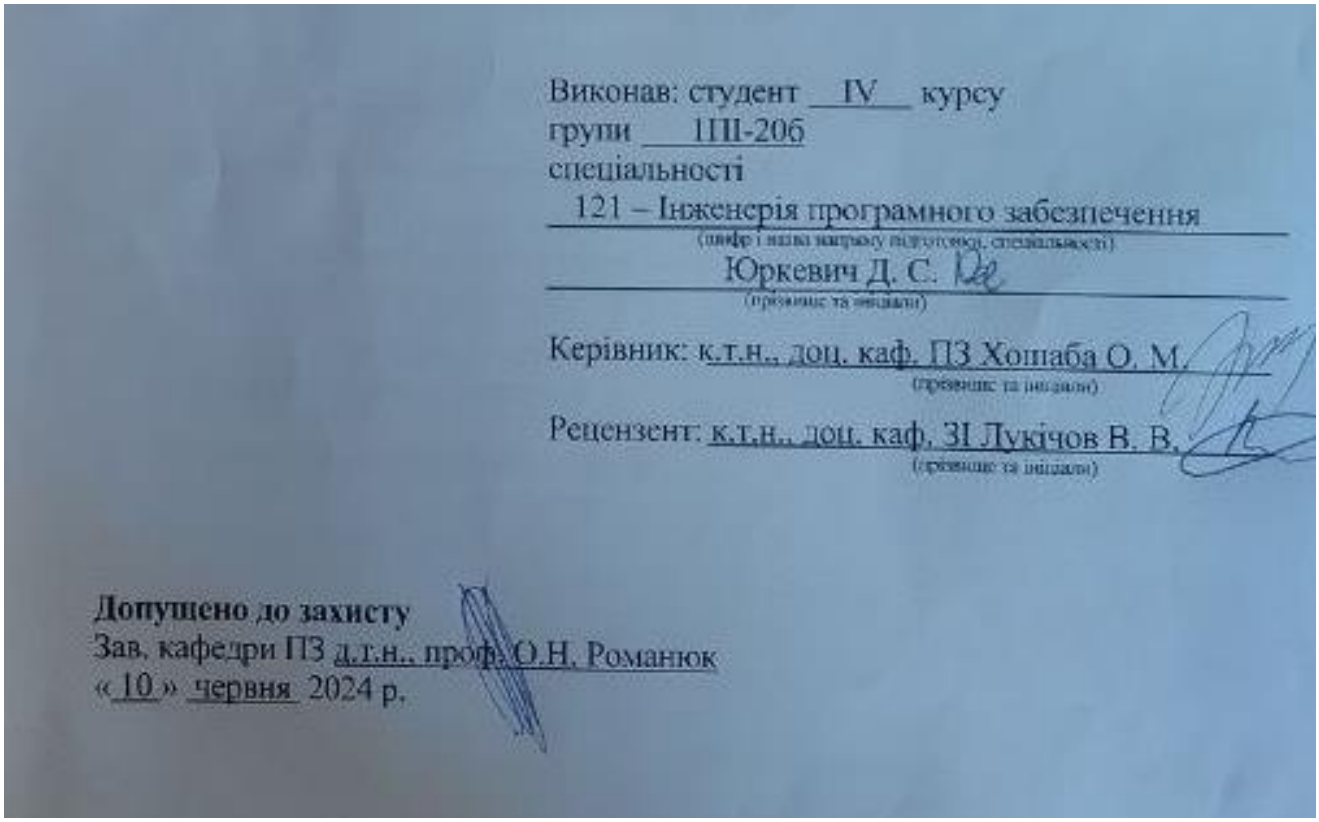
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань»



Виконав: студент IV курсу
групи ІПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Юркевич Д. С.
(прізвище та ініціали)

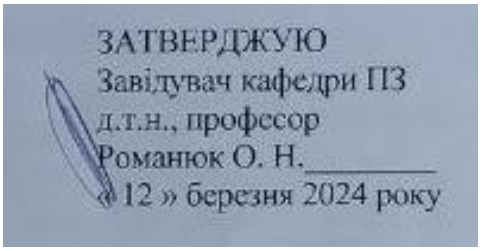
Керівник: к.т.н., доц. каф. ПЗ Хошаба О. М.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ЗІ Лукічов В. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ д.т.н., проф. О.Н. Романюк
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення



З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Юркевичу Дмитру Сергійовичу

1. Тема роботи – Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань.

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: мова програмування – Java, фреймворк Spring Boot, алгоритм пошуку шляху A*. Розмір вікна застосунку не менше 1024x768; кількість можливих точок маршруту – 8436; кількість постійних з'єднань між можливими точками маршруту – 13776; ефективна глибина пошуку – не менше 40 точок; критеріїв налаштування маршрутів – 5; просторова складність алгоритму не більше 1024 МБ; навантаження на систему – не більше 5% при рекомендованих параметрах.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка структури, методів та алгоритмів програмного продукту; розробка програмних

компонентів; тестування програми, висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: мета та актуальність дослідження; аналоги програмного застосунку; наукова новизна; діаграма прецедентів; діаграма діяльності; використані технології; шаблонні сторінки; результати роботи; кінцевий слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Хошаба О. М., к.т.н. доцент кафедри ПЗ	13.03.24	31.05.24

7. Дата видачі завдання 13 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи постановка задачі	14.03.24 – 24.03.24	Виконано
2	Розробка структури, методів та алгоритму роботи програмного продукту	25.03.24 – 15.04.24	Виконано
3	Розробка програмних компонентів	15.04.24 – 30.04.24	Виконано
4	Тестування програми	01.05.24 – 10.05.24	Виконано
5	Оформлення матеріалів до захисту БДР	11.05.24 – 30.05.24	Виконано

Студент _____ Д. С. Юркевич
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ О. М. Хошаба
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 00465

Юркевич Д. С. Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань : бакалаврська дипломна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 69 с.

На укр. мові. Бібліогр. : 20 назв ; рис. : 27; табл. 10.

У бакалаврській дипломній роботі проведено аналіз сучасного стану аналогів систем для пошуку шляхів між зірковими системами гри EVE Online.

Встановлено об’єкт, предмет, завдання та методи дослідження. Сформульовано мету дослідження – забезпечення можливості одночасно обмінюватись даними про тимчасові з’єднання та формувати за допомогою них шляхи. Щоб реалізувати мету було розроблено три ланкову архітектуру, яка складається з таких компонентів : клієнт, сервер і база даних.

Програмний продукт розроблено з використанням мови програмування Java та фреймворку Spring Boot, середовища розробки Intellij IDEA та веб-серверу Apache Tomcat. Також було використано систему керування базами даних PostgreSQL для управління базою даних.

Отримані в бакалаврській дипломній роботі результати можна використати обміну тимчасовими з’єднаннями та побудовою за допомогою них шляхів.

Ключові слова: вебсистема, програмне забезпечення, Java, Spring Boot, алгоритм обходу графу.

ABSTRACT

Yurkevich D. S. Development of a web application for building routes between star systems of the game EVE Online using temporary connections: bachelor's thesis on the specialty 121 – «software engineering», educational program - software engineering. Vinnytsa: VNTU, 2024. 69 p.

In Ukrainian language. Bibliographer: 20 titles ; fig. : 27; tabl. 12.

In the bachelor's thesis, an analysis of the current state of analog systems for finding paths between the star systems of the EVE Online game was carried out.

The object, subject, tasks and research methods are established. The goal of the research is formulated – to provide the possibility to simultaneously exchange data on temporary connections and to form paths with their help. To realize the goal, a three-link architecture was developed, which consists of the following components: client, server and database.

The software application is developed using the Java programming language and the Spring Boot framework, the Intellij IDEA development environment, and the Apache Tomcat web server. The PostgreSQL database management system was also used to manage the database.

The results obtained in the bachelor's thesis can be used for the exchange of temporary connections and the construction of paths using them.

Keywords: web system, software, Java, Spring Boot, graph traversal algorithm.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розробка програмних засобів для побудови маршрутів між зірковими системами гри EVE Online є завданням, що значно покращить досвід користувачів, що потребують часто будувати маршрути через віддалені системи [1]. Ця задача вимагає високого рівня комплексності та ефективності, оскільки система повинна надавати користувачам швидкий та оптимальний маршрут для подорожі між різними зірковими системами.

Розробка програмних модулів для такої вебсистеми включає кілька ключових етапів [2]. Спочатку, необхідно створити модуль, який забезпечить ефективне управління великою кількістю даних про зіркові системи та їхні взаємозв'язки. Це включає в себе розробку алгоритмів для обробки та аналізу даних кожної системи. Далі, модуль повинен забезпечувати можливість швидкого пошуку оптимального маршруту між будь-якою парою систем, враховуючи різноманітні фактори, такі як відстань, безпека шляху, наявність небезпек. Це вимагає розробки складних алгоритмів маршрутизації, які здатні оптимізувати подорожі у реальному часі.

Крім того, важливим аспектом є створення інтерфейсу, який би був зручним та інтуїтивно зрозумілим для користувачів. Це включає в себе розробку вебсистеми або застосунку, де система автоматично розрахує оптимальний маршрут та надасть необхідну інформацію про нього.

У зв'язку з постійним розвитком гри EVE Online та зростаючим попитом на зручні та ефективні засоби навігації між зірковими системами, розробка програмних модулів для побудови маршрутів стає все більш актуальною та важливою. Такі модулі допомагають гравцям ефективно використовувати ресурси своєї космічної підприємницької діяльності.

Наявні аналогічні web-системи не надають таких можливостей одночасного збору даних про тимчасові з’єднання від великої кількості користувачів та побудови маршрутів між ними.

Засобами для розробки стали мова програмування Java, фреймворк Spring Boot, сервер Apache Tomcat і система керування базою даних PostgreSQL.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Основною метою бакалаврської дипломної роботи є надання можливості користувачам обмінюватись даними про тимчасові з’єднання та будувати за допомогою них шляхи за допомогою одного вебресурсу.

Відповідно до поставленої мети необхідно вирішити такі завдання:

- провести аналіз аналогічних програмних продуктів для визначення актуальності покращень таких продуктів;
- розробити методи, принципи роботи та способи використання вебсистеми;
- розробити модифікацію алгоритму пошуку шляху для потреб вебсистеми;
- розробити функціональні та нефункціональні вимоги розроблюваної вебсистеми;
- розробити базу даних, змоделювати сутності відповідно до вимог серверної частини;
- розробити програмні модулі для забезпечення функціонування бізнес-логіки вебсистеми;
- розробити програмні модулі для обробки запитів користувачів;
- розробити модулі браузерного інтерфейсу, який забезпечуватиме користувачеві взаємодію із системою;
- провести тестування розробленої вебсистеми.

Об’єкт дослідження – процес побудови шляхів між зірковими системами гри з використанням тимчасових з’єднань.

Предмет дослідження – методи та алгоритми пошуку оптимальних маршрутів у динамічному графі.

Методи дослідження. У процесі досліджень використовувались: методи пошуку оптимального шляху між вузлами графа, методи побудови блок-схем алгоритмів, методи реалізації вебзастосунків за допомогою мови Java та фреймворку Spring Boot.

Наукова новизна отриманих результатів.

1. Запропоновано алгоритм пошуку шляху, який враховує можливість налаштування пошуку, що дозволяє знаходити коротші шляхи між точками графу за рахунок включення в пошук тимчасових з’єднань.

2. Вперше розроблено модель заохочення надання актуальних даних користувачами вебсистеми без можливості отримання принципово достовірних даних, яка полягає в колективній оцінці цих даних, що дає можливість ефективного обміну даних користувачами.

Практична цінність отриманих результатів. Ортимані в ході дослідження результати є практично цінними, оскільки вони послужили основою для розробки алгоритмів та програмних засобів для побудови шляхів між зірковими системами гри EVE Online відповідно до теоретичних принципів, викладених у бакалаврській дипломній роботі.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто.

Апробація та публікація результатів роботи. Результати роботи доповідалися на: Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи» (2024 р., м. Вінниця) та опубліковані в тезах доповіді.

Публікації. За тематикою дослідження опубліковано 1 наукову працю у збірниках матеріалів конференцій [3].

РОЗДІЛ 1
АНАЛІЗ СЕРЕДОВИЩА ГРИ ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Аналіз середовища гри та можливостей з’єднань зіркових систем

Із розвитком різних аспектів гри EVE Online та ускладненням економіки, все більш актуальним стає питання оптимізації зв’язку між зірковими системами.

В галактиці, що доступна гравцям є велика кількість ключових зіркових систем. Це і торгівельні вузли, граничні системи між регіонами та інші. Також окремі кораблі, спорядження цих кораблів та інші предмети потребують унікальних ресурсів для виготовлення. Ці ресурси можуть бути як і розповсюдженими всюди, так і локальними для певного регіону галактики. Можливість сполучати ці віддалені регіони є однією із ідей, що реалізує програмний продукт бакалаврської дипломної роботи.

Тимчасові з’єднання в якості кротових нор, що сполучають різні зіркові системи випадковим способом та мають різні властивості є найперспективнішим інструментом для оптимізації маршрутів між зірковими системами [4]. Вони можуть існувати до 24 годин, після чого згортаються та з’являються в іншій випадковій зірковій системі. На відміну від стабільних постійних з’єднань в якості зіркових воріт, інформація про які доступна всім гравцям на карті галактики, кротові нори необхідно сканувати за допомогою розвідувальних зондів та зберігати їх координати вручну. Передавати їх іншим гравцям можливо за допомогою тек координат або через ідентифікатор сигналу цієї кротової нори та зіркову систему, де вона знаходиться, після чого гравець може її сканувати.

Для обміну інформацією про тимчасові з’єднання гравці об’єднуються в групи для спільного користування папками координат тимчасових з’єднань. Проте цей шлях передачі має багато недоліків, таких як:

- Доступ лише для обмеженої кількості гравців;
- Обмежені можливості автоматичного видалення неактуальних координат;
- Необхідність вручну планувати маршрут;

- Обмежені можливості прикріплення інформації про стан з’єднання.

Для вирішення цих проблем створюються аналогічні програмні продукти, проте вони не надають всі бажані можливості та функціонал. Детальніше про аналогічні програмні продукти у наступному розділі.

Отже, розробка програмних модулів вебсистеми побудови маршрутів між зірковими системами гри EVE Online, є задачею, виконання якої значно покращить можливості обміну інформації про тимчасові з’єднання між зірковими системами та можливість побудови маршрутів із їх використанням.

1.2. Порівняльний аналіз аналогів

Розробка програмних модулів вебсистеми побудови маршрутів між зірковими системами гри EVE Online значно покращить можливості обміну інформації про тимчасові з’єднання між зірковими системами та можливість побудови маршрутів із їх використанням. Наявність аналогічних програмних продуктів ще раз доводить, що розробка модулів є актуальною задачею, і в подальшому може зайняти ринок таких продуктів. Серед аналогічних програмних продуктів можна виокремити такі:

- EVE Scout Wormhole Connections;
- Dotlan EVE maps;
- Pathfinder.

Одним із прикладів аналогічних програмних продуктів є EVE Scout Wormhole Connections [5]. Він надає актуальні та детальні дані про тимчасові з’єднання, проте лише з двома зірковими системами – Thera та Turnur (рис. 1.1). Дані про тимчасові з'єднання надаються закритою групою відповідальних гравців, тому дані достовірні майже завжди. Вебсайт надає можливості будувати маршрути лише із використанням одного тимчасового з'єднання, тому користувачі повинні робити це вручну, якщо необхідно використати більше з'єднань для оптимізації маршруту без можливості оптимізації цих дій для збільшення зручності.

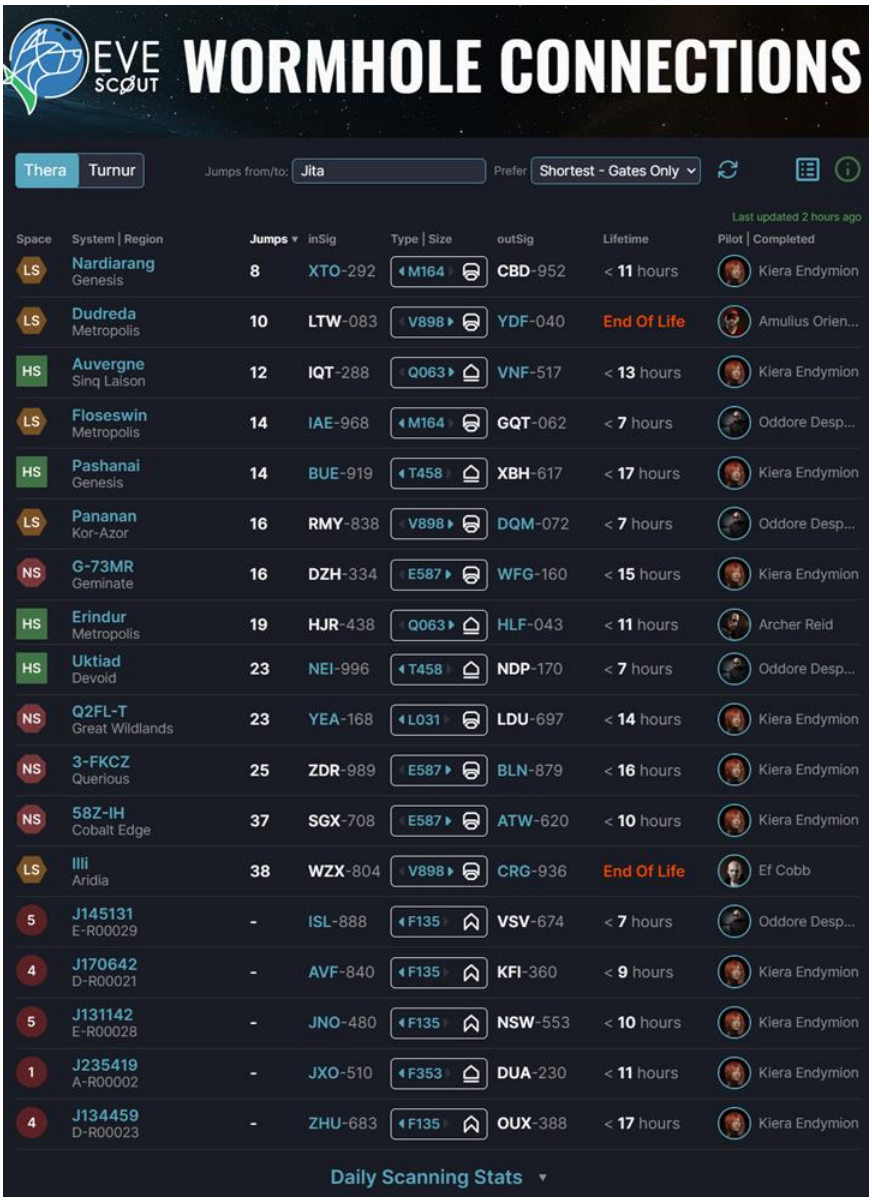


Рисунок 1.1 – Вебсайт EVE Scout Wormhole Connections.

Іншим прикладом аналогічного програмного продукту є Dotlan EVE maps [6]. Він надає можливість створювати маршрути, дозволяє додавати окремі зіркові системи в список для уникання, дозволяє обирати системи певного статусу безпеки та додавати тимчасові з’єднання для побудови маршруту. Головним недоліком є те, що всі тимчасові з’єднання користувач повинен вносити вручну, та немає можливості обміну ними між користувачами. Інтерфейс вебсайту Dotlan EVE maps (рис. 1.2) дозволяє ігнорувати окремі зіркові системи, проте не вистачає більш детальних налаштувань.

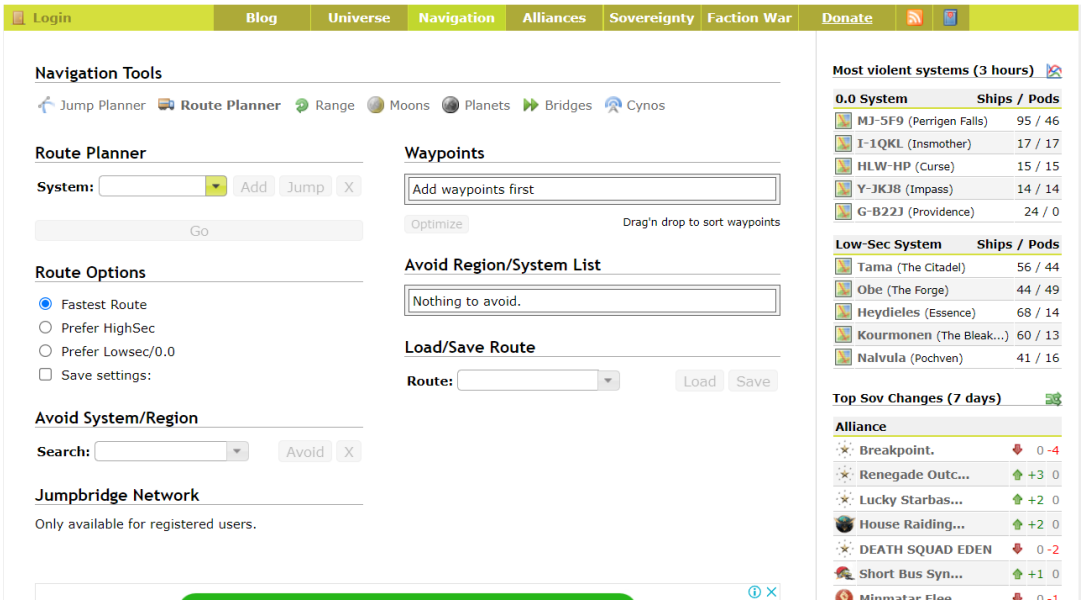


Рисунок 1.2 – Інтерфейс Dotlan EVE maps

Pathfinder в свою чергу надає можливість як будувати маршрути, так і обмінюватись тимчасовими з'єднаннями [7]. Проте, не має можливості дізнатись актуальність певного маршруту чи з'єднання. Тому цей вебсайт дозволяє обмін лише між окремими групами користувачів, унеможливлюючи обмін між всіма користувачами EVE Online (рис. 1.3).

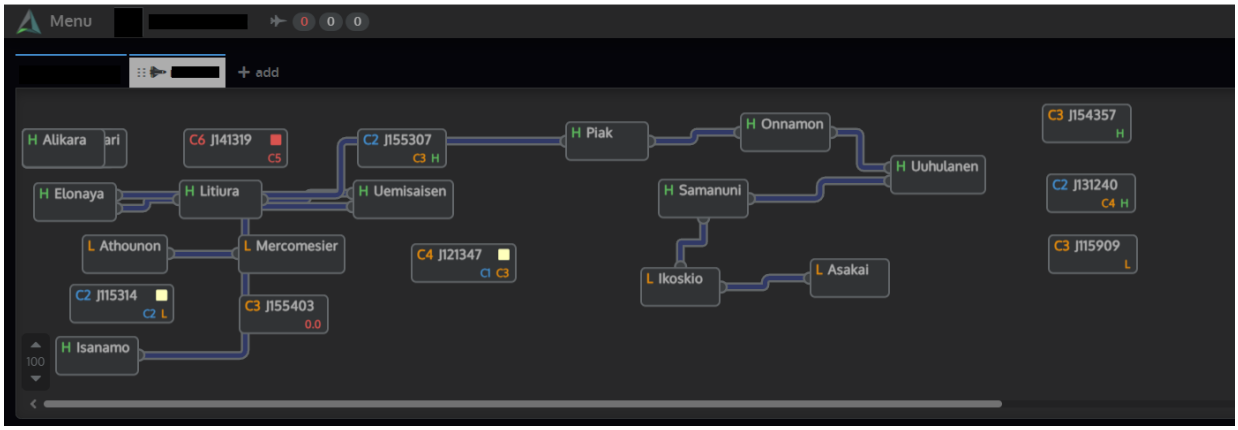


Рисунок 1.3 – Інтерфейс Pathfinder

Результати порівняння аналогів наведено в табл. 1.1. Критерії оцінюються балом від 0 до 5, де 0 – повна відсутність відповідності очікуванням, а 5 – повна відповідність очікуванням.

Таблиця 1.1 – Порівняння аналогічних програмних продуктів

Критерії порівняння	**Програмні продукти**			
	EVE Scout	**Dotlan**	**Pathfinder**	**Власний програмний продукт**
Побудова маршрутів	1	4	4	5
Налаштування побудови маршрутів	1	3	3	5
Актуальність даних про наявні тимчасові з’єднання	5	0	4	5
Зручність роботи з інтерфейсом	5	3	4	5
Обмін тимчасовими з’єднаннями	0	0	3	5
Сума	12	10	18	25

Відповідно до табл. 1.1 порівняльних характеристик розробка власної вебсистеми є доцільною, бо власний програмний продукт має найвищу сумарну оцінку – 25. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити новий досвід та можливості застосування.

Отже, розробка програмних модулів вебсистеми побудови маршрутів між зірковими системами гри EVE Online є доцільною, адже надає можливість всім гравцям обмінюватись тимчасовими з’єднаннями між зірковими системами, створювати маршрути із їх використанням та налаштовувати створення маршрутів без використання недосконалої вбудованої системи обміну

координатами [8]. Самостійний пошук тимчасових з’єднань може займати великий проміжок часу, а може взагалі не принести результату.

Програмний застосунок дозволить користувачам значно зменшити час пошуку необхідного оптимізованого маршруту. Система нагород за надання актуальних та важливих тимчасових з’єднань вмотивує користувачів спільними силами побудувати екосистему, що дозволить шукати необхідні маршрути вчасно без необхідності довготривалого самостійного пошуку.

1.3. Аналіз підходів до розробки застосунків

Розробка програмних модулів для реалізації веб-системи з використанням механіки побудови маршрутів між зірковими системами гри EVE Online вимагає ретельного розгляду найкращих підходів до вирішення проблеми. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних модулів.

Одним з варіантів вирішення проблеми розробки програмних модулів для реалізації веб-системи побудови маршрутів між зірковими системами гри EVE Online є створення окремої бібліотеки, яка дозволяє інтегрувати модуль в існуючі рішення. Однак такий підхід має певні недоліки. Хоча це швидкий спосіб розробки модуля, він вимагає додаткових зусиль від існуючих онлайн-платформ. Цей підхід також може призвести до значного обсягу роботи, покладеного на розробку онлайн-платформи, яка хоче інтегрувати це рішення. Тому такий підхід не варто застосовувати.

Інший підхід полягає в розробці окремих сервісів (мікросервісів) для інтеграції модуля в існуючі системи [9]. Такий підхід дозволяє розробникам зосередитися на розробці модуля без необхідності створювати повномасштабну онлайн-платформу. Користувачі такого сервісу можуть інтегрувати готове рішення у свої системи, що дозволить зменшити ресурси, необхідні для інтеграції, порівняно з попереднім рішенням. Цей підхід також включає розробку внутрішньої системи авторизації для захисту даних онлайн-платформ,

які хочуть інтегрувати цей модуль. Однак цей сервіс може потребувати значних апаратних ресурсів у разі інтеграції багатьма сервісами. Тому цей метод вважається ефективним, якщо його можна швидко і стабільно масштабувати.

Найпоширенішим рішенням є розробка власної онлайн-платформи з вбудованим модулем для реалізації веб-системи [10]. У цьому випадку немає необхідності адаптувати систему під різні потреби сторонніх платформ. Більше того, в результаті такого методу розробки виходить унікальна платформа, яка містить унікальний функціонал. Однак такий підхід має і свої недоліки, оскільки вимагає повноцінної розробки платформи.

Розглянувши переваги та недоліки кожного методу, було вирішено використати метод розробки власної онлайн-платформи з вбудованим модулем для реалізації веб-системи побудови маршрутів між зірковими системами гри EVE Online. Такий підхід дозволить отримати повноцінний програмний продукт без необхідності використання потужного апаратного забезпечення. Більше того, такий метод розробки дозволить створити унікальну платформу з унікальним функціоналом.

1.4. Постановка задач бакалаврської дипломної роботи

Проаналізувавши переваги та недоліки існуючих системи з використанням засобів доповненої реальності було визначено наступні завдання, які необхідно виконати для розробки мобільної системи:

- провести аналіз аналогічних програмних продуктів для визначення актуальності покращень таких продуктів;
- розробити базу даних, змоделювати сутності системи, з вимогами серверної частини;
- розробити модифікацію алгоритму пошуку шляху для потреб вебсистеми;
- розробити модулі серверної частини, що взаємодіють з базою даних;
- розробити модуль графічного інтерфейсу, який ефективно забезпечуватиме користувачеві взаємодію із системою;

- провести тестування розробленої вебсистеми.

Отже, в даному розділі проведено аналіз зв'язку економіки гри EVE Online з віддаленістю ключових зіркових систем, а також поставлено задачі для подальшого дослідження та розробки програмного продукту. Підкреслено актуальність оптимізації зв'язку між системами у зв'язку з ускладненням економіки гри.

Порівняно різноманітні аналоги систем для побудови маршрутів між зірковими системами, виявлено їх переваги та недоліки. Виокремлено необхідність розробки власної системи з урахуванням позитивних аспектів існуючих рішень. Важливою є можливість обміну тимчасовими з'єднаннями між гравцями та швидке побудова оптимальних маршрутів.

Аналіз методів розв'язання поставленої задачі підтвердив вибір розробки власної платформи з вбудованим модулем. Цей підхід дозволить створити унікальну систему з урахуванням потреб гравців EVE Online та забезпечить повноцінний функціонал без необхідності використання додаткового апаратного забезпечення.

Зазначено завдання бакалаврської дипломної роботи, які включають створення бази даних, серверного модуля та веб-інтерфейсу для зручного користування програмним продуктом. Реалізація цих завдань дозволить розробити ефективний інструмент для оптимізації маршрутів та співпраці користувачів у грі EVE Online.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМІВ, МЕТОДІВ, ПРИНЦИПІВ РОБОТИ ТА СПОСОБІВ ВИКОРИСТАННЯ ВЕБСИСТЕМИ

2.1. Аналіз та моделювання системи заохочення надання користувачами актуальних даних.

Дані про тимчасові з’єднання не можуть бути отримані автоматично із програмного інтерфейсу гри. Натомість, їх може сканувати лише користувач із спеціальним спорядженням [11]. У користувачів не завжди буває можливість переносити це спорядження, хоча лише вони можуть надавати такі дані. Також інформація про неактуальні чи вигадані тимчасові з’єднання може як і витратити час, так і навіть стати загрозою для космічного корабля.

Отже, для подібної системи насамперед необхідно розробити модель, при якій користувачі вмотивовані надавати актуальні дані.

Одним з ключових аспектів розробки системи заохочення є визначення механізмів перевірки достовірності наданих даних. Це може включати в себе аналіз історії користувача, перевірку збігу інформації з іншими джерелами або використання алгоритмів машинного навчання для виявлення неактуальних даних.

Паралельно з розробкою моделі заохочення важливо також врахувати механізми захисту користувачів. Оскільки надання актуальних даних може вимагати розкриття особистої інформації, система повинна гарантувати безпеку та конфіденційність цих даних.

Нарешті, успішна реалізація системи заохочення також залежить від виваженості між стимулюванням користувачів та запобіганням зловживанням. Важливо розробити механізми, які стимулюють надання даних, не збільшуючи ризик отримання неправдивої інформації. Це може бути досягнуто шляхом введення системи перевірки вірогідності наданих даних та регулярного

оновлення правил та стимулів з урахуванням динаміки гри та поведінки користувачів.

Крім того, важливим елементом системи заохочення є створення прозорої та справедливої системи розподілу винагород. Наприклад, можна розглянути варіанти, де винагороди розподіляються залежно від значності та актуальності наданих даних.

Першим аспектом такої системи стане те, що для користування системою користувачі повинні бути зареєстрованими.

Також за всі розміщені користувачем тимчасові з’єднання, якими користуються інші гравці, він буде отримувати спеціальні бали, за які він може отримувати доступ до найкращих тимчасових з’єднань.

Користувачі будуть мати можливість оновлювати дані по з’єднанню, що дозволить підтримувати актуальність цих з’єднань.

Через те, що тимчасові з’єднання не можуть існувати більше, ніж 24 години, дані про них автоматично видаляються із системи.

Ці принципи системи заохочення надання актуальних даних користувачами вебсистеми забезпечують актуальність даних (рис. 2.1).

Рисунок 2.1 – Принципи системи заохочення надання актуальних даних

Галактика гри наповнена різноманітними небезпеками, більшість з яких можуть бути уникнені правильним вибором маршруту та мінімальними заходами безпеки від користувачів [12].

Для моделювання системи налаштування пошуку маршруту, необхідно проаналізувати які фактори можуть бути негативними для мандрівників галактикою.

Першим та найпростішим фактором є статус безпеки системи. Статус безпеки системи коливається між значеннями 1.0 та -1.0 у різних зіркових систем [13]. Він залежить від того, наскільки захищеною є система галактичними силами поліції Конкорд. В залежності наскільки статус безпеки високий, в системі діють такі правила:

- якщо статус безпеки більше або дорівнює 0.5, поліція буде атакувати користувачів, що зашкоджують кораблі інших гравців, при чому чим число ближче до 1.0, тим швидше реакція поліції на такі дії;
- якщо статус безпеки менше 0.5, то поліція не буде атакувати агресорів, проте захисні системи станцій та зоряних воріт все ще будуть їх атакувати;
- якщо статус безпеки менше або дорівнює 0.0, в таких системах повністю відсутній захист та дозволено використання різноманітних потужних озброєнь кораблів.

Іншим, менш очевидним фактором є наближеність до відомих між гравцями зіркових систем чи найкоротших шляхів між ними [14]. Ігнорування найбільш наближених систем до космічних торгівельних вузлів чи вузьких коридорів між скупченнями зіркових систем допоможе значно зменшити ризик для користувача.

Одним із важливих факторів є те, що тимчасові з’єднання можуть пропускати лише кораблі лише до певного класу маси, отож необхідно будувати маршрут лише через з’єднання що може пройти користувач на своєму космічному кораблі.

Також важливо надати можливість користувачам ігнорувати окремі системи вручну, адже власний досвід гравців є найбільш достовірним.

Активні зіркові системи – такі, що мають високу активність гравців в них.

Реалізуватиметься система налаштування пошуку маршрутів таким чином, що при запиті на формування маршруту користувач буде передавати дані:

- про мінімальний статус безпеки, що дозволяється для пошуку оптимального маршруту;
- про клас маси космічного корабля гравця;
- про те, які конкретні системи ігнорувати.

Додатковим аспектом, який слід враховувати при моделюванні системи налаштування пошуку маршрутів, є збалансованість між швидкістю та безпекою. Хоча швидке переміщення через галактику може бути важливим для досягнення цілей гравця, необхідно також враховувати ризики, пов'язані з потенційно небезпечними зірковими системами. Тому система повинна забезпечувати можливість врахування цих факторів при визначенні оптимального маршруту для подорожі.

Після впровадження системи важливо провести аналіз її ефективності та здійснити необхідні вдосконалення. Це може включати в себе збір та аналіз статистичних даних про використання системи, зворотний зв'язок від користувачів щодо їхнього досвіду та пропозиції щодо поліпшень, а також оцінку відповідності системи поставленим завданням та цілям. На основі цього аналізу можна буде розробити стратегії подальшого розвитку та удосконалення системи налаштування пошуку маршрутів для більш ефективного та задоволеного користувача.

Отже, зіркова система може бути частиною знайденого маршруту, якщо має статус безпеки вище, або такий самий як і заданий, повинна пропускати корабель гравця по всім з’єднанням, не повинна бути в списку ігнорованих зіркових систем. Вже після цього обирається оптимально віддалений від активних систем шлях. Оптимальним шляхом в такому випадку можна вважати рівновіддалений від усіх активних систем.

2.2. Аналіз способів використання системи.

Додавання нового тимчасового з’єднання. Для додавання нового тимчасового з’єднання в систему користувач повинен заповнити форму з такими даними:

- Перша сполучена зіркова система;
- Друга сполучена зіркова система;
- Приблизний час існування тимчасового з’єднання до його знищення;
- Клас маси цього тимчасового з’єднання.

Після цього користувач натискає кнопку створити, і тимчасове з’єднання додається в систему.

Оновлення даних тимчасового з’єднання. Для оновлення даних користувач повинен обрати, яке тимчасове з’єднання він оновлює та надати актуальні дані про залишок маси, що може пропустити тимчасове з’єднання до розпаду да скільки часу його існування залишилось за приблизними оцінками.

Підтвердження актуальності тимчасового з’єднання. Для цього користувачу необхідно ідентифікувати яке тимчасове з’єднання він підтверджує. Після цього автор з’єднання та користувачі, що редагували дані по ньому отримують винагороди, що можуть бути використані задля доступу до актуальних з’єднань.

Пошук маршруту. Користувач використовує цю функцію для знаходження оптимального маршруту для своєї подорожі через галактику. При використанні цієї опції, користувач передає системі параметри, такі як мінімальний статус безпеки, клас маси космічного корабля, допустима відстань до активних зіркових систем, а також список ігнорованих систем. Після обробки цих даних, система відображає оптимальний маршрут для користувача, забезпечуючи збалансований підхід між швидкістю та безпекою. Користувач може обрати варіант, який відповідає його потребам найкраще, і вирушити у подорож. Після завершення подорожі користувачі можуть дати зворотній зв'язок щодо використаного маршруту. Цей зворотній зв'язок може містити відгуки щодо

безпеки та ефективності маршруту, а також пропозиції щодо можливих поліпшень. Адміністратори системи можуть аналізувати цю інформацію для подальшого вдосконалення алгоритмів планування маршрутів, що дозволить забезпечити ще кращий та безпечніший вибір маршрутів для користувачів. Все це повинно покращити досвід користування системою у користувачів.

Було створено діаграму прецедентів системи (рис. 2.2), що відображає способи використання системи візуально.

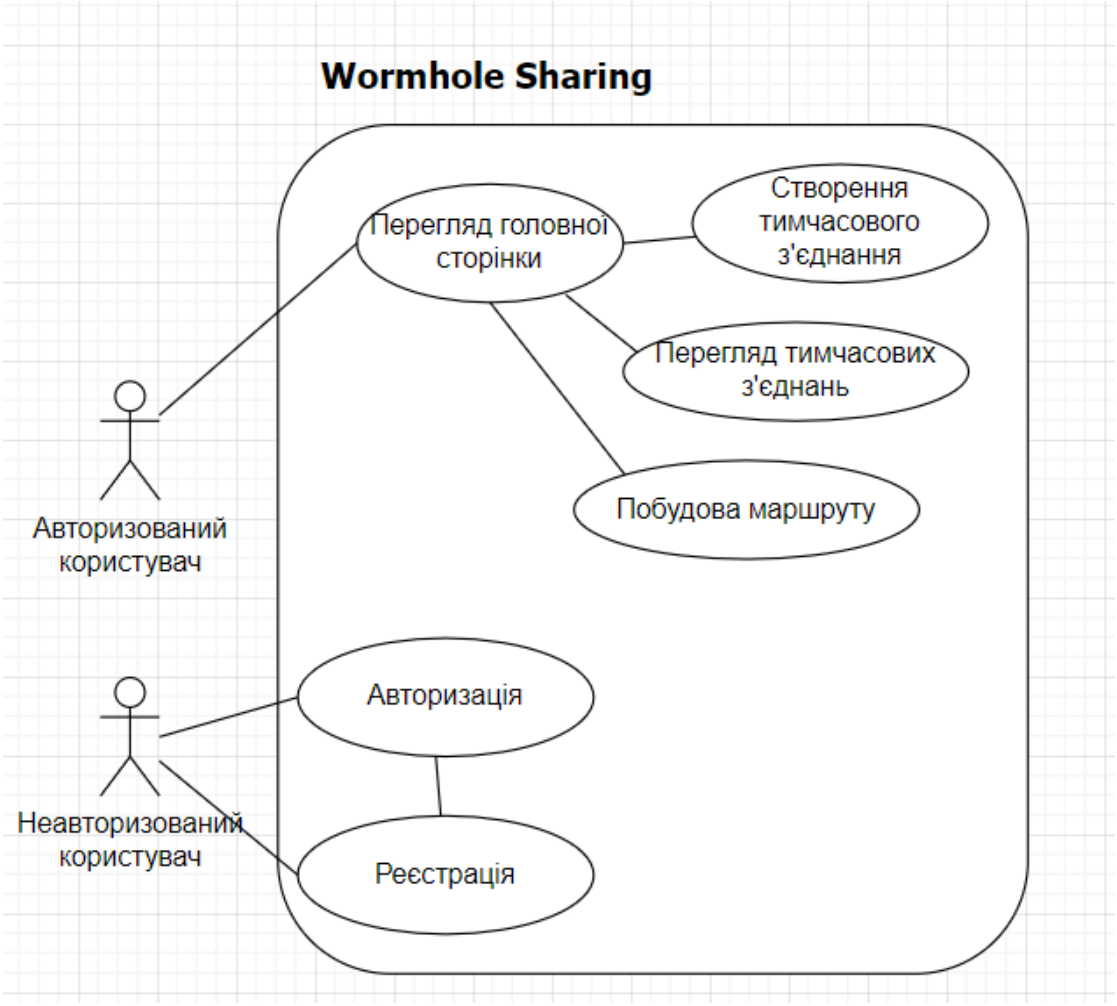


Рисунок 2.2 – Діаграма прецедентів системи

Діаграма діяльності системи може ілюструвати процеси та послідовність дій, які відбуваються під час взаємодії користувача з системою налаштування пошуку маршрутів у грі. Наприклад, діаграма може відобразити послідовність кроків в процесі пошуку та вибору оптимального маршруту, починаючи з

введення параметрів користувачем до отримання результатів і відображення їх на екрані. Кожен крок може бути представлений у вигляді дії або операції, яка виконується системою або користувачем, і з'єднаний стрілками для показу порядку виконання. Діаграма діяльності (рис. 2.3) допомагає зрозуміти, як система реагує на вхідні дані користувача та як вона обробляє їх для надання корисної інформації або функціоналу.

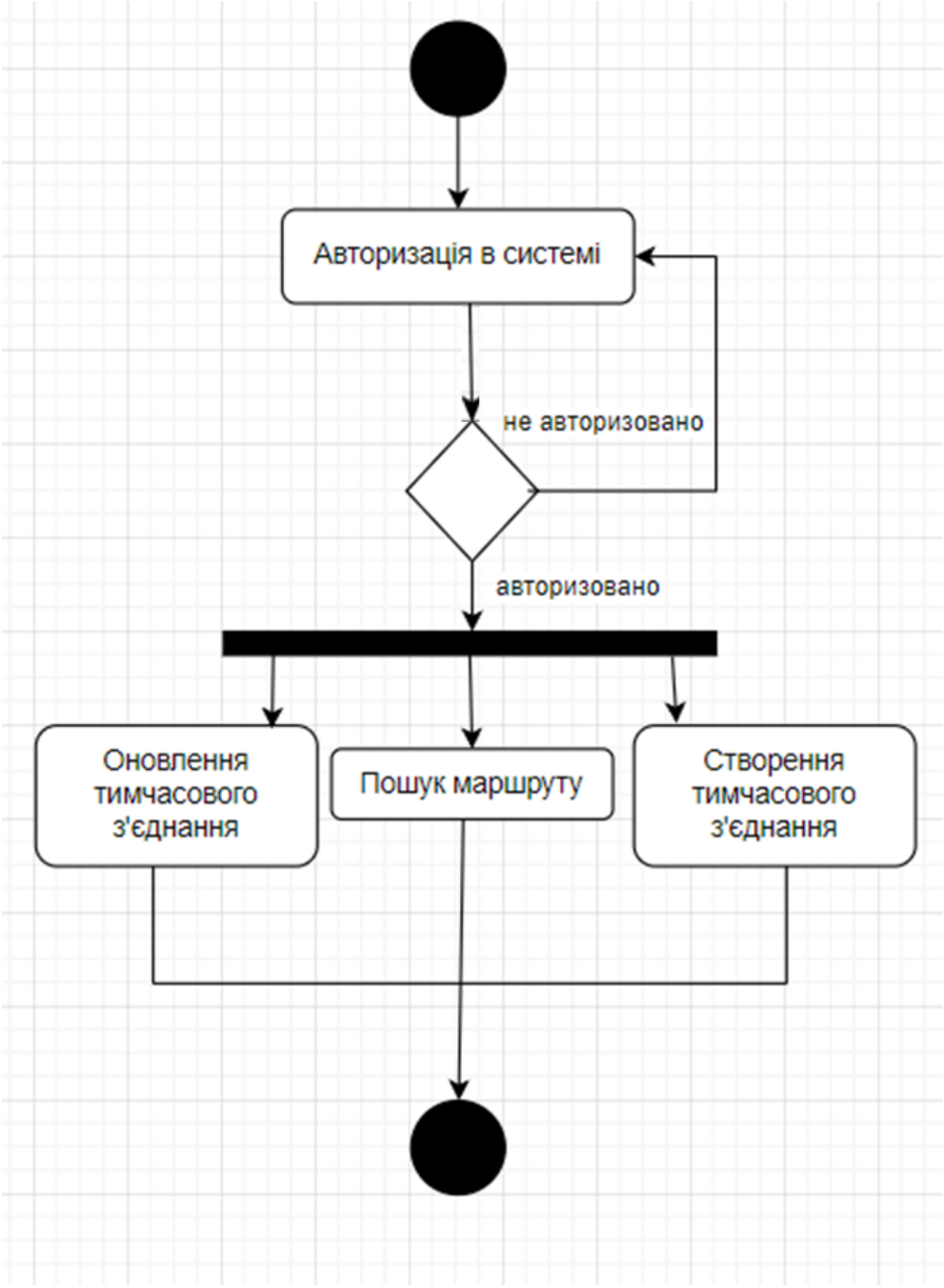


Рисунок 2.3 – Діаграма діяльності системи

2.3. Аналіз та розробка алгоритму пошуку шляхів

Зіркові системи та сполучення між ними формують граф. Для пошуку необхідного маршруту потрібно використати алоритм обходу графу. Граф – фундаментальна структура даних, яка складається з множини вершин і множини ребер, що з'єднують пари вершин (рис. 2.4). Графи можуть бути орієнтованими або неорієнтованими залежно від того, чи мають їхні ребра напрямок. В орієнтованих графах ребра вказують на односторонній зв'язок між вершинами, тоді як в неорієнтованих графах зв'язки є двосторонніми. Графи широко використовуються для моделювання та аналізу реальних систем, таких як соціальні мережі, транспортні мережі, комп'ютерні мережі, а також для вирішення різноманітних задач, включаючи пошук найкоротших шляхів, планування маршрутів, виявлення спільнот і оптимізацію мереж.

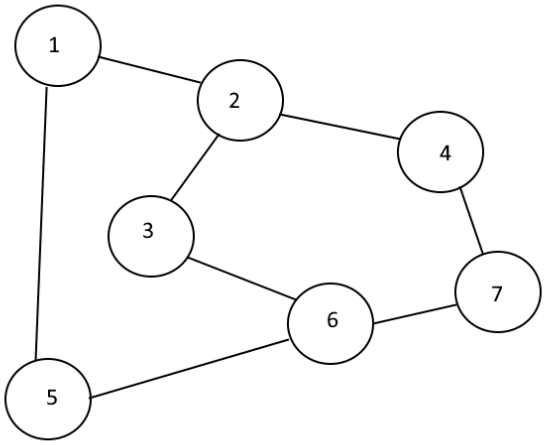


Рисунок 2.4 – Приклад неорієнтованого графу

Доцільно розглянути класифікацію алгоритмів обходу графів в глибину та в ширину.

Пошук у глибину (англ. Depth-First Search) є одним з фундаментальних алгоритмів для обходу графів, який широко використовується для вирішення різних задач, таких як пошук шляхів, виявлення компонентів зв'язності, перевірка наявності циклів і топологічне сортування. Алгоритм працює шляхом

відвідування вершини, потім рекурсивного (або з використанням явного стеку) переходу до одного з її сусідів, повторюючи цей процес для кожного відвіданого вузла, поки не будуть досліджені всі шляхи. Якщо алгоритм досягає вершини без невідвіданих сусідів, він повертається назад до останньої відвіданої вершини з невідвіданими сусідами і продовжує пошук. Такий підхід дозволяє алгоритму глибоко занурюватися в граф перед поверненням і дослідженням інших шляхів.

DFS ефективно працює на графах, де необхідно вивчити всі можливі шляхи або виявити складні структури, такі як цикли. Він також застосовується для задач, де важливо дослідити всі вершини. Завдяки своїй рекурсивній природі, DFS є простим для реалізації, однак може вимагати значного обсягу пам'яті для підтримки стека рекурсивних викликів, особливо в дуже глибоких або щільних графах. Важливо зазначити, що при неправильному використанні або на графах з великими глибинами, рекурсивна версія DFS може призвести до переповнення стека.

Обхід графу в ширину (англ. Breadth-First Search) є методом для дослідження графів, що забезпечує систематичний підхід до відвідування всіх вершин у графі. Алгоритм починається з початкової вершини і досліджує всі її сусіди на одному рівні перед переходом до вершин наступного рівня. Це означає, що BFS обробляє вершини в порядку їхньої віддаленості від початкової точки, що робить його ідеальним для задач пошуку найкоротших шляхів у незважених графах. Використання черги дозволяє алгоритму ефективно організувати процес відвідування вершин, додаючи нові вершини до черги тільки після обробки поточних. Таким чином, BFS гарантує, що кожна вершина буде відвідана лише один раз, уникаючи дублювання роботи і заощаджуючи час.

BFS знаходить широке застосування в різних областях. Наприклад, у комп'ютерних мережах його використовують для знаходження найкоротшого шляху між двома пристроями. У транспортних системах цей алгоритм дозволяє знайти найшвидші маршрути без урахування ваги (довжини) доріг. Крім того, BFS використовується в алгоритмах штучного інтелекту для планування дій та пошуку рішень у лабіринтах. Завдяки своїй простоті і надійності, BFS

залишається одним з найбільш використовуваних алгоритмів у комп'ютерних науках та багатьох інших дисциплінах, де необхідно ефективно досліджувати або аналізувати графові структури.

Проте, наведені вище алгоритми можуть бути повільними при великих графах, адже є неоптимізованими та обходять граф повністю в будь-якому випадку. Саме тому, за основу розробки власної модифікації алгоритму було обрано алгоритм A*.

Алгоритм A* (англ. A-star) є одним з найефективніших і найпопулярніших евристичних алгоритмів пошуку шляхів у графах, особливо корисний для знаходження найкоротших шляхів у складних середовищах. Він комбінує переваги двох інших алгоритмів: жадібного пошуку за найменшою вартістю і алгоритму Дейкстри. A* використовує дві функції для оцінки вартості шляху: g(n), яка представляє собою фактичну вартість шляху від початкової вершини до поточної вершини n, і h(n), яка є евристичною оцінкою вартості найкоротшого шляху від поточної вершини n до цільової вершини. Загальна вартість вершини n (f(n)) обчислюється як сума g(n) і h(n), що дозволяє алгоритму ефективно знаходити шлях, який є найкоротшим.

A* широко застосовується в багатьох областях, включаючи робототехніку, ігровий розвиток, картографію та системи навігації. Його здатність швидко знаходити оптимальні шляхи робить його незамінним у динамічних середовищах, де необхідно часто оновлювати маршрути, наприклад, у навігаційних системах для автономних транспортних засобів або в іграх для планування руху персонажів. Однією з головних переваг A* є його гнучкість: різні евристичні функції h(n) можуть бути розроблені для конкретних задач, що дозволяє алгоритму адаптуватися до різних типів проблем і середовищ. Однак, ефективність A* сильно залежить від вибору евристичної функції; неправильний вибір може призвести до збільшення часу виконання або до знаходження неоптимальних шляхів. Попри це, A* залишається одним з найпотужніших і найпоширеніших інструментів для пошуку шляхів у багатьох прикладних задачах.

Особливість власної модифікації цього алгоритму стало те, що для пошуку використовуються як і постійні, так і тимчасові з’єднання. При цьому вартість вершин обчислюється з урахуванням заданих користувачем налаштувань, що передаються у вигляді об’єкту класу RouteSettings.

2.4. Аналіз алгоритмів реєстрації та авторизації

Алгоритми реєстрації та авторизації є критичними компонентами систем безпеки сучасних веб-додатків. Процес реєстрації зазвичай включає створення нового облікового запису користувача шляхом надання необхідної інформації, такої як електронна адреса та пароль.

Для забезпечення безпеки даних користувачів насамперед необхідно реалізувати хешування пароля. Для хешування було обрано алгоритм sha 256. Цей вибір є актуальним, бо одночасно запезпечує високий рівень безпеки та швидкість хешування.

Для забезпечення можливості користувачам користуватись вебсистемою необхідно надати їм можливість реєстрації. Було розроблено блок-схему алгоритму авторизації (рис. 2.5). Блок-схема алгоритму авторизації (див. рис. 2.5) детально описує процес перевірки облікових даних користувача під час входу в систему. Після введення користувачем електронної адреси та пароля, система порівнює хешований пароль з тим, що зберігається в базі даних. Якщо дані збігаються, користувач отримує доступ до своїх ресурсів відповідно до своєї ролі та прав доступу. У разі невідповідності даних, користувач отримує повідомлення про помилку, і йому пропонується повторити спробу входу. Важливо також реалізувати механізми блокування облікового запису після певної кількості невдалих спроб входу, що допомагає запобігти атакам типу "грубої сили". Такі заходи, у поєднанні з регулярними оновленнями та перевірками безпеки, забезпечують надійний захист користувачів та їх даних у вебсистемі.

Рисунок 2.5 – Блок-схема алгоритму авторизації

Для нових користувачів необхідно забезпечити простий і зручний процес створення особистого облікового запису. Це включає надання користувачам можливості зареєструватися, заповнивши форму з базовою інформацією, такою як електронна адреса та пароль. Важливо, щоб цей процес був інтуїтивно зрозумілим та не вимагав надмірної кількості даних на початковому етапі, щоб не відлякувати потенційних користувачів. Після реєстрації користувачам може бути надіслано підтвердження на вказану електронну адресу для верифікації їх

облікового запису, що допоможе підтвердити їхню особу та запобігти створенню фальшивих профілів. Такий підхід не лише покращує користувацький досвід, але й сприяє підвищенню загальної безпеки системи.

Було розроблено блок-схему алгоритму реєстрації користувачів (рис. 2.6).

Рисунок 2.6 – Алгоритм реєстрації користувачів

2.5. Аналіз допоміжних алгоритмів

У зв’язку з тим, що світ гри постійно розвивається, відкриваються нові зіркові системи, стає актуальною аналіз та розробка алгоритмів для швидкого оновлення даних про зіркові системи та постійні з’єднання між ними.

Розробка алгоритму для автоматизованого збору та оновлення даних про зіркові системи і їх з’єднання має велике значення для підтримки роботи вебсистеми на високому рівні. Такий алгоритм забезпечує швидкий доступ до актуальної інформації, що дозволяє користувачам краще орієнтуватися в галактиці, що постійно розвивається. Використання відкритого програмного інтерфейсу полегшує цей процес, надаючи можливість отримувати дані про зіркові системи та їхні зв'язки у зручному форматі, що сприяє створенню актуальних дани про зіркові системи.

Крім того, алгоритм автоматичного оновлення дозволяє швидко адаптувати систему до нових умов та впроваджувати оновлення, які роблять досвід користувачів кращим. Постійне оновлення даних про зіркові системи та їх з’єднання допомагає уникнути застарілої інформації та зменшує ймовірність виникнення помилок. Це забезпечує більш стабільний і безперебійний процес роботи вебсистеми, покращуючи взаємодію з системою користувачів. Такий підхід також відкриває нові можливості для розвитку, дозволяючи додавати тимчасові з’єднання, що базуються на актуальних даних.

Існує відкритий програмний інтерфейс від розробників, що дозволяє отримувати такі дані:

- список ідентифікаторів всіх зіркових систем гри;
- список ідентифікаторів всіх зоряних воріт конкрентої зіркової системи;
- отримання назви системи та іншої важливої інформації по ідентифікатору;
- отримання назв сусідніх систем по ідентифікаторам зіркових воріт;
- отримання числа загальної кількості зіркових систем в галактиці.

Завдяки відкритому програмному інтерфейсу розробка такого алгоритму стає можливою та актуальною, адже без цього було б необхідно збирати такі дані вручну.

Було розроблено алгоритм збору інформації про всі зіркові системи галактики та постійні з’єднання між ними (рис. 2.7).

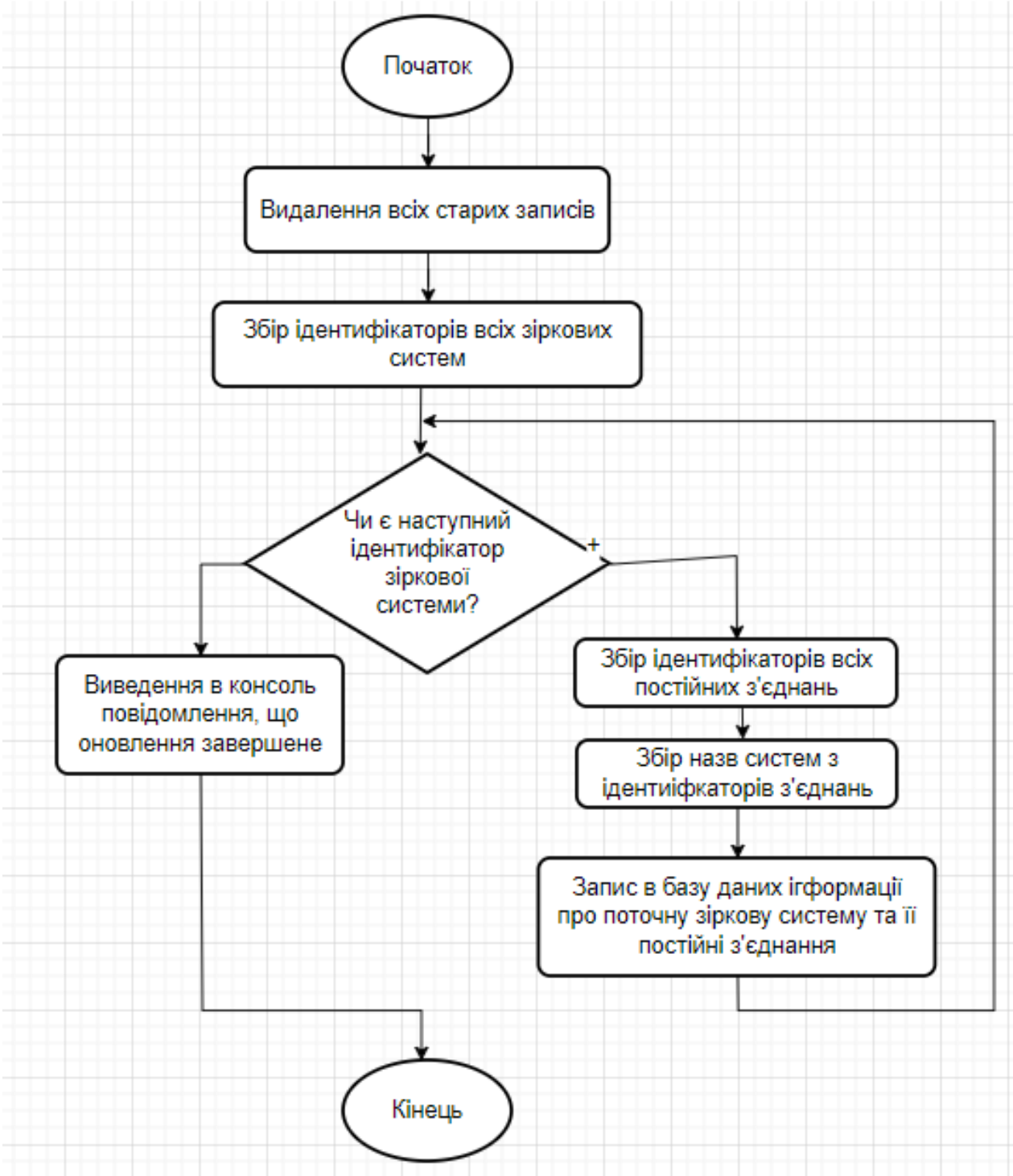


Рисунок 2.7 – Алгоритм збору інформації про зіркові системи

Отже, було проведено глибокий аналіз системи заохочення користувачів надавати актуальні дані тимчасових з’єднань. Зазначено, що надання таких даних є важливим елементом гри, проте вимагає спеціального спорядження та ініціативи самого гравця. Викладено ключові аспекти розробки такої системи, зокрема моделі заохочення, механізмів перевірки достовірності даних та захисту особистої інформації. Детально розглянуто можливі негативні фактори окремих точок маршрутів та викладено модель системи налаштування пошуку маршрутів з урахуванням цих факторів. Подальше дослідження охопило аналіз способів використання системи, включаючи додавання та оновлення тимчасових з’єднань, підтвердження їх актуальності та пошук оптимального маршруту. Відзначено важливість діаграми діяльності системи для ілюстрації процесів взаємодії користувача з системою, а також використання діаграми використання для візуалізації функцій системи та послідовності дій користувача. Цей аналіз відображає значущість та комплексність розробки та моделювання системи, що сприятиме покращенню взаємодії користувачів між собою. За основу розробки алгоритму пошуку шляху в неорієнтованому графі було обрано алгоритм A*. Вартість вершин обчислюється з урахуванням заданих користувачем налаштувань. Для пошуку шляху будуть використані як постійні, так і тимчасові з’єднання.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ВЕБСИСТЕМИ

3.1. Вибір мови програмування та стеку технологій

При розробці вебсистем для побудови маршрутів між зірковими системами з використанням тимчасових з'єднань, вибір правильного набору технологій визначає успішність проєкту. Різні технології пропонують різні переваги та недоліки, і їх вибір повинен бути обґрунтованим і зорієнтованим на конкретні потреби проєкту.

Варіант 1. Python і фреймворк Django.

Переваги:

1. Швидка розробка. Django має велику кількість вбудованих функцій і стандартних бібліотек, що сприяє швидкому старту проєкту.

2. Велика спільнота. Django має активну спільноту розробників і багато готових рішень, що полегшує пошук допомоги та розв'язання проблем.

3. Безпека. Django має вбудовані заходи безпеки, такі як захист від SQL-ін'єкцій та захист від злому міжсайтового сценарію (CSRF).

Недоліки:

1. Менша масштабованість. Django може мати проблеми з масштабуванням для дуже великих проєктів порівняно з іншими фреймворками.

2. Специфічний стиль програмування. Django нав'язує певний стиль програмування, що може бути не найкращим вибором для всіх розробників.

3. Через специфіку роботи python-програм, а саме через те, що такі програми інтерпретуються та виконуються через єдиний потік, програми на python є значно менш продуктивними за програми на інших мовах.

Варіант 2. Javascript і фреймворк Node.js.

Переваги:

1. Висока продуктивність. Node.js використовує асинхронний необхідний введення/виведення, що робить його дуже продуктивним для високонавантажених застосунків.

2. Оптимальність для систем реального часу. Node.js добре підходить для застосунків реального часу, таких як чати або онлайн-ігри.

3. Широкий вибір бібліотек для будь-яких потреб, що спрощує розробку.

Недоліки:

1. Менша стабільність. Node.js може бути менш стабільним в порівнянні з іншими технологіями, такими як Java або Python.

2. Менше вбудованих функцій. Node.js має менше вбудованих функцій, ніж інші фреймворки, що може вимагати більше сторонніх бібліотек.

Варіант 3. Java і фреймворк Spring Boot [15].

Переваги:

1. Велика стабільність та надійність. Java відома своєю стабільністю та надійністю, що робить її ідеальним вибором для великих проєктів.

2. Широкий вибір бібліотек. Java має велику екосистему бібліотек, серед яких Spring Framework, який спрощує розробку вебсистем.

3. Масштабованість. Java добре масштабується і підходить для великих проєктів з високим обсягом трафіку.

4. Підтримка засобів шаблонізації створення вебсторінок. Вони дозволяють створювати шаблони вебсторінок для відображення інформації з серверної частини застосунку, що спрощує розробку.

Недоліки:

1. Більше коду: Java може вимагати більше коду порівняно з іншими мовами, такими як Python або Ruby.

2. Помірна швидкість розробки: Хоча Spring Boot спрощує розробку, вона все ж може бути менш швидкою порівняно з іншими фреймворками.

Отже, з урахуванням вимог до проєкту, включаючи стабільність, масштабованість та швидкість розробки, рекомендованим варіантом для

розробки вебсистеми побудови маршрутів між зірковими системами є Java з використанням Spring Boot. Ця комбінація надає необхідний рівень надійності та масштабованості, разом з широким вибором бібліотек і фреймворків для полегшення розробки. Саме ці характеристики найкраще підходять для поставленої задачі по розробці програмних модулів.

3.2. Вибір необхідних для розробки залежностей та створення проекту

Програмні модулі, що будуть розроблятись вимагають підключення різноманітних бібліотек, тому необхідно обрати правильний інструмент для початку роботи.

Вибір правильного інструменту для початку роботи є ключовим кроком у розробці будь-якого проєкту. Spring Initializr є ідеальним вибором для початку роботи з проєктами на основі Spring Boot з наступних причин:

Простота використання інструменту Spring Initializr виражається в тому, що він пропонує зручний і простий інтерфейс для швидкого створення нового проєкту [16]. Завдяки його інтуїтивно зрозумілому інтерфейсу користувач може легко налаштувати основні параметри проєкту, такі як мову програмування, версію Spring Boot та список залежностей.

Гнучкість конфігурації дозволяє налаштовувати проєкт згідно з конкретними вимогами. Користувач може вибрати необхідні залежності, які включатимуться в проєкт, що дозволяє створювати проєкти згідно з потребами конкретної задачі.

Також Spring Initializr дозволяє розробникам швидко створювати нові проєкти без необхідності вручну налаштовувати інфраструктуру. Це дозволяє зосередитися на основній логіці застосунку, мінімізуючи витрати часу на початкову конфігурацію.

Широкий вибір початкових шаблонів у Spring Initializr пропонує різноманітні початкові шаблони, які можна використовувати для різних типів

проєктів. Це дозволяє вибрати найбільш підходящий шаблон для конкретної задачі та швидко розпочати роботу.

Spring Initializr це проєкт із відкритим вихідним кодом, що підвищує його надійність та швидкість оновлень. Вся спільнота користувачів може приймати участь у розвитку цього інструменту.

І наостанок, Spring Initializr постійно оновлюється і підтримується спільнотою, що гарантує актуальність інструменту і наявність новітніх функцій і можливостей для розробників.

Враховуючи ці переваги, Spring Initializr стає очевидним вибором для початку роботи з проєктами на основі Spring Boot, допомагаючи розробникам ефективно створювати якісний програмний продукт.

Було запущено інструмент для початку роботи з Spring Boot – Spring Initilaizr (рис. 3.1).

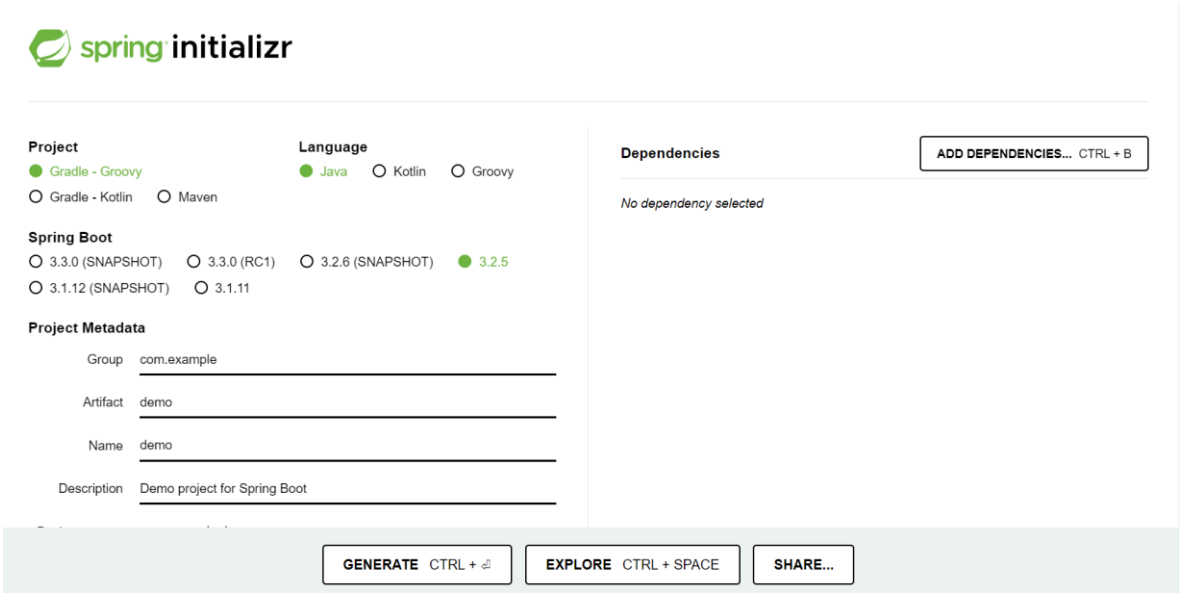


Рисунок 3.1 – Інтерфейс Spring Initializr

Для генерування необхідного шаблону проєкту, необхідно обрати залежності (бібліотеки) для підключення.

Насамперед, важливо не забувати про чистоту коду, простоту та його читабельність. Це дозволить пришвидшити розробку, мінімізувати наявність

помилок та спростити їх пошук та виправлення. Також це дозволить швидше та простіше розширювати програмний модуль. З цим допоможе чудова бібліотека з відкритим вихідним кодом Lombok [17]. Вона дозволяє створювати різноманітні шаблонні методи та поля у класах через прості анотації – позначки, що можуть бути додані до програмних компонентів у мові програмування Java.

Першою функціональною залежністю, яку ми додамо до цього проєкту за допомогою Spring Initializr, буде Spring Web. Ця бібліотека надає набір інструментів і функцій для створення вебзастосунків з використанням Spring Framework. Завдяки Spring Web розробники можуть легко створювати веб-сервіси і вебзастосунки, використовуючи архітектурні принципи REST або інші стандарти комунікації між клієнтом і сервером. Ця залежність додає необхідні класи та анотації для створення контролерів, які обробляють HTTP-запити та повертають відповіді клієнтам. Використання Spring Web спрощує розробку вебзастосунків та робить їх більш масштабованими та ефективними.

Наступною залежністю, яку ми додамо до цього проєкту, буде Spring Data JPA. Ця бібліотека є важливим компонентом для роботи з базами даних в контексті програмних модулів Spring. Вона надає спрощений спосіб взаємодії з базами даних через Java Persistence API (JPA). Spring Data JPA дозволяє розробникам працювати з об'єктно-реляційним відображенням даних, не занурюючись в деталі SQL-запитів або маппінгу об'єктів до таблиць бази даних. Ця залежність спирається на Hibernate, який виступає реалізацією Java Persistence API. Hibernate використовується для управління об'єктами в базі даних, забезпечуючи вищий рівень абстракції над реляційними даними. Використання Spring Data JPA разом з Hibernate спрощує роботу з базами даних в застосунках Spring, роблячи розробку більш ефективною та забезпечуючи швидкий доступ до даних.

Для того щоб Spring Data JPA могла ефективно зберігати дані, необхідно також підключити базу даних. В цьому випадку ми обираємо PostgreSQL як систему керування базами даних. Щоб забезпечити взаємодію між застосунком і базою даних PostgreSQL, ми додамо драйвер підключення до бази даних

PostgreSQL (PostgreSQL Driver) як ще одну залежність у цьому проєкті. Це дозволить Spring Data JPA використовувати PostgreSQL для зберігання даних та здійснювати з ними взаємодію через Hibernate. Підключення PostgreSQL Driver дозволить цьому програмному модулю встановлювати з'єднання з базою даних PostgreSQL і виконувати SQL-операції, необхідні для зберігання та отримання даних. Цей крок є важливим для створення повноцінної веб-системи, яка може ефективно взаємодіяти з базою даних та забезпечувати постійний доступ до інформації для користувачів.

Spring Security є ще однією важливою залежністю для нашого проєкту, яка забезпечить надійність та безпеку веб-додатку. Ця бібліотека надає рішення для аутентифікації, авторизації та захисту від атак нашого додатку. Використання Spring Security дозволить нам забезпечити захист доступу до ресурсів системи, контролювати права користувачів і здійснювати перевірку безпеки на всіх рівнях додатку. За допомогою конфігурацій та анотацій Spring Security ми зможемо легко налаштувати доступ до різних сторінок нашого додатку, встановити права доступу до програмного інтерфейсу та інших ресурсів, а також здійснювати інтеграцію з іншими методами аутентифікації, такими як OAuth або JWT. Загалом, використання Spring Security допоможе нам забезпечити високий рівень безпеки в нашому проєкті та захистити його від потенційних загроз.

Останньою доданою залежністю у нашому проєкті є Thymeleaf - потужний інструмент для роботи з шаблонами веб-сторінок у Spring-застосунках. Thymeleaf надає можливість створювати динамічний та добре структурований HTML-контент, який може відображати дані з сервера [18]. Це забезпечує зручну і гнучку можливість розробки користувацького інтерфейсу, дозволяючи вбудовувати вирази, умовні конструкції та цикли прямо у HTML-коді. Thymeleaf інтегрується з Spring Framework, що робить його відмінним вибором для створення веб-інтерфейсу у розроблюваному проєкті. Його потужність полягає в простоті використання та великій функціональності. Завдяки Thymeleaf, ми можемо швидко та ефективно розробляти динамічні сторінки, що відображають дані з нашого додатку, і додаємо інтерактивність до користувацького досвіду. Це

дозволить нам створювати привабливий та функціональний інтерфейс, який відповідає вимогам проєкту.

Залежності, які ми додали до нашого проєкту, є ключовими компонентами для розробки високоякісної та функціональної веб-системи. Використання Spring Web дозволить нам створити маршрутизацію та контролери для обробки HTTP-запитів. Spring Data JPA та PostgreSQL Driver дозволять нам зберігати та взаємодіяти з даними у базі даних PostgreSQL. Spring Security надасть захист від несанкціонованого доступу та забезпечить безпеку застосунку. Нарешті, Thymeleaf дозволить нам створювати динамічні та привабливі веб-сторінки для відображення даних користувачам. Використання цих залежностей допоможе нам створити надійну, функціональну та зручну для користувача веб-систему, яка відповідає всім нашим потребам та вимогам.

Було обрано всі необхідні залежності, обрано тип проєкту, мову програмування, версію Spring Boot, тип збірки кінцевого файлу та версію Java. Отже, було встановлено початкові налаштування проекту розробки в інструменті Spring Initializr (рис. 3.2).

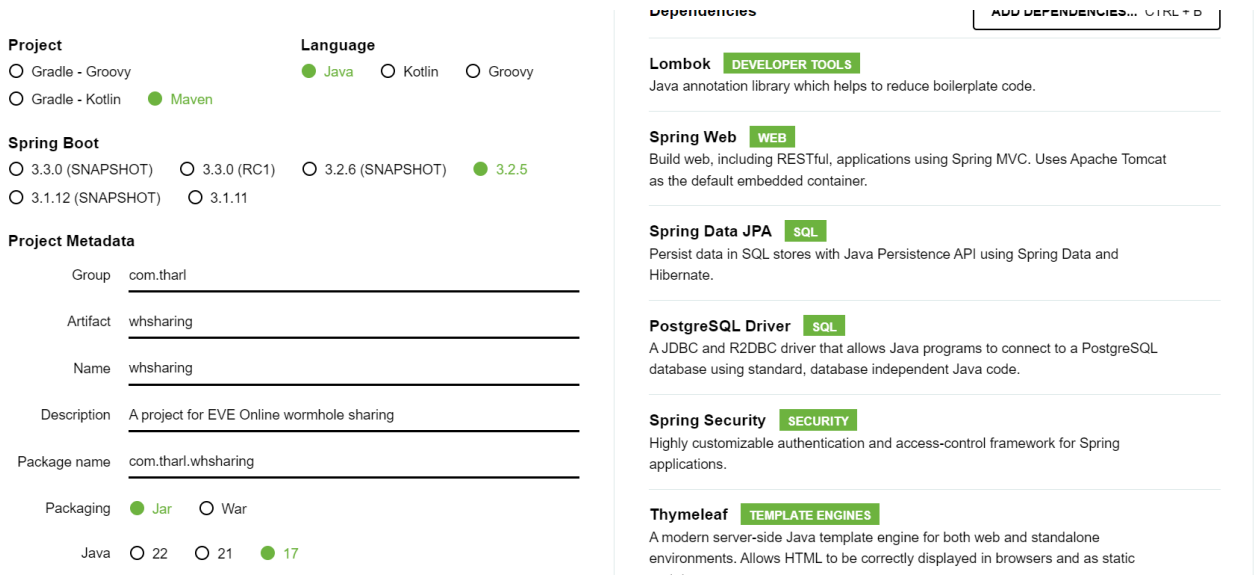


Рисунок 3.2 – Кінцевий вигляд налаштувань проєкту у Spring Initializr

3.3. Розробка структури, моделей даних і налаштувань безпеки вебсистеми

У цьому підрозділі буде розглядатись процес створення програмних модулів для нашої веб-системи. Цей етап включатиме розробку різних функціональних компонентів, які складатимуть основу нашого додатку. Ми розглянемо створення модулів для обробки HTTP-запитів, взаємодії з базою даних, забезпечення безпеки та автентифікації користувачів, а також створення користувацького інтерфейсу за допомогою HTML-шаблонів та CSS-стилізації. Крім того, було проведене дослідження можливості розширення функціональності нашого додатку за допомогою створення власних модулів та інтеграції зі сторонніми сервісами. В результаті цього етапу очікується отримати готові програмні компоненти, які будуть готові для інтеграції в загальну веб-систему та надання користувачам необхідного функціоналу.

Початковим етапом розробки проєкту вебсистеми є створення логічної структури пакетів, що є фундаментальним кроком у визначенні організації програмного коду. Правильно структурований проєкт спрощує його подальше розвиток, розуміння та управління. Зазвичай, логічна структура пакетів відображає архітектурні принципи застосунку та відображає його функціональні компоненти. Кожен пакет (англ. package) відповідає за свою певну область функціональності та допомагає забезпечити чітке розділення обов'язків у проєкті, що сприяє його модульності та легкості супроводу. Зазвичай застосунок Spring Boot має чітку структуру, розділену 3 рівнями: рівень доступу до даних, рівень бізнес логіки та рівень відображення, чи просто рівень взаємодії із системою. Завдяки такій структурі легко вносити зміни в програмний код чи шукати та виправляти помилки в ньому. Ретельно розроблена структура пакетів є основою для подальшого ефективного розвитку та масштабування вебсистеми [19].

Було розроблено структуру пакетів вебсистеми (рис. 3.3).

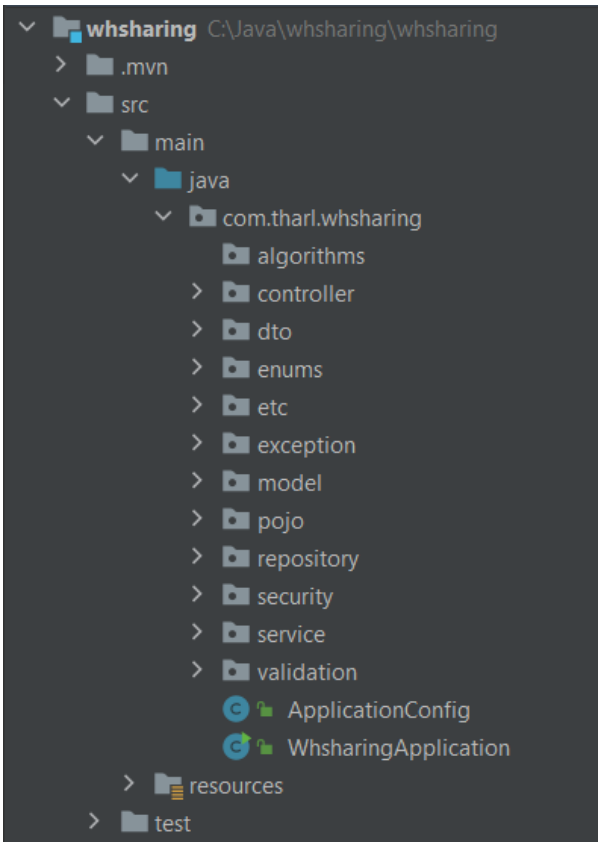


Рисунок 3.3 – Структура проєкту

Пакет algorithms буде використаний для зберігання коду, необхідного для роботи алгоритмів пошуку маршрутів.

Пакет controller буде використаний для зберігання контролерів – програмних компонентів Spring, що відповідають за настройку прийому http-запитів та надсилання відповідей по різнім кінцевим точкам програмного інтерфейсу.

Пакет dto буде містити класи для створення об’єктів передачі даних (англ. Data transfer object).

Пакет enums містить класи, що відображають пронумеровані списки значень.

Пакет exception містить класи-виключення, що можуть виникати в ситуаціях, коли подальша рбоота програми не має сенсу та потрібно обробити таку ситуацію.

Пакет model зберігатиме моделі даних, що зберігатимуться в базі даних.

Пакет pojo містить класи, що не управляються фреймворком Spring.

Пакет repository зберігатиме інтерфейси взаємодії з базою даних.

Пакет service буде зберігати класи, що відповідають за бізнес-логіку системи.

Пакет validation відповідає за зберігання класів перевірки правильності введених користувачем даних, тобто валідації.

Пакет security буде зберігати налаштування безпеки системи.

Було розроблено класи-нумерації для відображення можливих значень деяких полів в сутностях, їх опис наведено нижче.

TimeState – відображає мінімально гарантовану інформацію про те, скільки з’єднання буде існувати. Цей нумерований клас має 3 значення:

1. VERY_FRESH – мінімум 24 години;

2. FRESH – мінімум 4 години;

3. EOL – на межі зникнення (англ. end of life).

Size – відображає максимальну пропускну можливість з’єднання. Має 4 значення:

1. FRIGATE;

2. CRUISER;

3. BATTLESHIP;

4. CAPITAL.

SecurityStatus – відображає значення статусу безпеки системи. Має 3 значення:

1. HIGHSEC – від 0.5 до 1.0 коефіцієнту безпеки;

2. LOWSEC – від 0.1 до 0.4 коефіцієнту безпеки;

3. NULLSEC – від -1.0 до 0.0 коефіцієнту безпеки.

MassState – відображає стан маси з’єднання, тобто наскільки воно пошкоджене кораблями, що проходять через нього. Має 3 значення:

1. STABLE – менше, ніж 50% від можливої маси;

2. HALFCRIT – більше, ніж 50% від можливої маси;

3. CRIT – більше, ніж 90% від можливої маси.

RouteSettings – має 6 полів:

1. SecurityStatus minimalSecStatus;

2. Set<String> ignoredSystemNames;

3. boolean newlyConfirmedByReport;

4. private Size size;

5. private System start;

6. private System end.

Наступним етапом розробки буде створення моделей даних, що будуть зберігатись в базі даних та інтерфейсів взаємодією з базою даних, що буде оперувати цими моделями даних.

Кожна сутність повинна мати первинний ключ, що дозволяє ідентифікувати конкретний запис. Первинний ключ позначається анотацією @Id. Якщо ключ генерується автоматично, необхідно додати анотацію @GeneratedValue.

Для різних типів відношень між сутностями існують спеціальні анотації, наприклад, відношення «один-до-багатьох» відображає анотація @OneToMany. Якщо за забезпечення відношення відповідатиме колонка таблиці, необхідно використати анотацію @JoinColumn, а для забезпечення відношення в окремій таблиці треба використати @JoinTable. Також можливо записати всі рядки в одне поле тиблиці за допомогою @ElementCollection.

Для того, щоб налаштувати зберігання нумерованих типів, треба використати анотацію @Enumerated. В аргументі можна обрати, чи записувати номер типу чи його рядкове значення.

За записи тимчасових з’єднань буде відповідати клас Wormhole. Він є стандартним класом-сутністю, що має випадковий ідентифікатор. Цей клас має поле reports, що відображає список записів про оновлення даних по з’єднанню.

Створені поля класу та анотації для реляційної розмітки сутності Wormhole наведено в табл. 3.1.

Таблиця 3.1 – Поля сутності Wormhole

Java-клас	**Ім’я поля**	**Анотації**
Long	id	@Id
		@GeneratedValue
System	fromSystem	@ManyToOne
		@JoinColumn(name = "from_system_name", nullable = false)
System	toSystem	@ManyToOne
		@JoinColumn(name = "to_system_name", nullable = false)
LocalDateTime	discoveryTime	@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
Size	size	@Enumerated(value = EnumType.STRING)
List<StateReport>	reports	@OneToMany(fetch = FetchType.LAZY)
		@JoinTable(name = "wormhole_state_reports", joinColumns = @JoinColumn(name = "wormhole_id", referencedColumnName = "id"), inverseJoinColumns = @JoinColumn(name = "state_report_id", referencedColumnName = "id"))

За записи оновлень інформації по тимчасовим з’єднанням відповідає клас StateReport. Він містить інформацію про те, який користувач створив цей запис.

Розроблені поля класу сутності StateReport та анотації для реляційної розмітки Spring JPA наведено в табл. 3.2.

Таблиця 3.2 – Поля сутності StateReport

Java-клас	**Ім’я поля**	**Анотації**
Long	id	@Id
		@GeneratedValue
MassState	massState	@Enumerated(value = EnumType.STRING)
TimeState	timeState	@Enumerated(value = EnumType.STRING)
LocalDateTime	time	@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")

Інформація про стабільні з’єднання буде зберігатись у класах SolarSystem у вигляді списку систем, до яких є стабільне з’єднання з поточної. Розроблені поля класу сутності System та анотації для реляційної розмітки Spring JPA наведено на табл. 3.3.

Таблиця 3.3 – Поля сутності System

Java-клас	**Ім’я поля**	**Анотації**
String	id	@Id
Double	security	-
Set<System>	jumpgates	@JsonIgnore
		@ManyToMany(fetch = FetchType.LAZY)
		@JoinTable

Продовження таблиці 3.3

List<Wormhole>	fromWormholes	@OneToMany(mappedBy = "fromSystem", cascade = CascadeType.ALL, orphanRemoval = true)
List<Wormhole>	toWormholes	@OneToMany(mappedBy = "toSystem", cascade = CascadeType.ALL, orphanRemoval = true)

Інформацію про користувачів системи відображає клас User. Він містить ім’я, хеш пароля та рахунок користувача. Розроблені поля класу сутності System та анотації для реляційної розмітки Spring JPA наведено на табл. 3.4.

Таблиця 3.4 – Поля сутності User

Java-клас	**Ім’я поля**	**Анотації**
Long	id	@Id
		@GeneratedValue(strategy=GenerationType.AUTO)
String	username	@NotBlank(message = "username cannot be blank")
		@NonNull
		@Column(nullable = false, unique = true)
String	password	@NotBlank(message = "password cannot be blank")
		@NonNull
		@Column(nullable = false)

Для запезпечення доступу до вебсистеми лише довіреним користувачам, необхідно розробити модуль реєстрації та авторизації. В Spring Boot за такий функціонал відповідає модуль Spring Security. Розглянемо принцип роботи Spring Security.

Кожен запит в вебсистему проходить через фільтр запитів – SecurityFilterChain. Першим кроком в реалізації необхідного функціоналу стане реалізація цього класу в об’єкт, що буде створюватись та використовуватись Spring Boot автоматично. Такі об’єкти називаються бінами (англ. Bean). Вони створюються за допомогою анотації @Bean над методом, що повертає необхідний об’єкт.

При проходженні через цей фільтр, всі запити на авторизацію та реєстрацію будуть виконані безумовно. Всі інші запити будуть виконані лише при успішній авторизації.

Наступним етапом буде налаштування джерела з даними про корисувачів, а саме їх логін та хеш пароля. В Spring Security за це відповідає бін UserDetailsService. Було вирішено зв’язати спосіб отримання даних про користувачів із методом getUser сервісу UserService.

Отже, було розроблено модуль для реєстрації та авторизації. Дані про корисувачів будуть зберігатись в базі даних. Також було розроблено та описано способом розмітки об’єктів Java на сутності реляційної бази даних JPA сутності, необхідні для роботи системи. Відповідні до сутностей інтерфейси доступу до бази даних розширюються від інтерфейсів JpaRepository, що дозволяє Spring автоматично реалізовувати всі базові операції із сутностями.

3.4. Розробка класів з бізнес-логікою вебсистеми

Розробка класів з бізнес-логікою для вебсистеми є критично важливим етапом у створенні вебсистеми, оскільки ці класи відповідають за реалізацію основних функціональних вимог і правил, які визначають поведінку системи. Вони містять методи і атрибути, що обробляють дані, виконують операції з ними і забезпечують виконання завдань відповідно до дій користувача. Правильна

організація класів з бізнес-логікою дозволяє забезпечити модульність, повторне використання коду і спрощує підтримку та розширення системи в майбутньому. Такий підхід допомагає підтримувати чітку структуру програмного коду і забезпечує логічний поділ між різними рівнями системи, такими як інтерфейс користувача, бізнес-логіка і доступ до даних.

У Spring Boot класи, що відповідають за бізнес-логіку прийнято називати сервісами та додавати до них анотацію @Service. Розглянемо розроблені класи-сервіси:

1. SystemService – має такі методи:

- updateSolarSystemsDB, за допомогою відкритого програмного інтерфейсу отримує дані про зіркові системи та постійні з’єдання між ними для запису в базу даних.

- countSystems, повертає кількість систем в базі даних, необхідний для адміністрування вебсистеми.

- findAllSystems, повертає список зіркових систем, необхідний для адміністрування вебсистеми.

- findSolarSystemByName, здійснює пошук в базі даних по імені зіркової системи, повертає контейнер, в якому система є, якщо знайдена, та пустий контейнер, якщо зіркова система не знайдена.

- findSystemById, здійснює пошук в базі даних по ідентифікатору зіркової системи, повертає знайдену зіркову систему.

- saveSolarSystem приймає об’єкт зіркової системи як аргумент та зберігає в базу даних.

При первинному оновленні даних про зіркові системи та постійні з’єднання між ними, було отримано такі дані:

- зіркових систем: 8436.

- постійних з’єднань: 13776.

2. RouteService – має метод findRoute, відповідає за пошук шляху. За аргумент приймає об’єкт класу RouteSettings. Виконує пошук із заданими

налаштуваннями по тимчасовим та постійним з’єднанням. Результатом роботи методу буде список назв систем – маршрут.

3. UserService – містить методи, необхідні для реєстрації та авторизації користувачів, а саме методи для пошуку користувача та метод для зберігання нових користувачів в базу даних.

4. WormholeService – містить такі методи:

- findAll, повертає список всіх доступних тимчасових з’єднань.

- findAllWormholeConnections, приймає 2 агрументи: зіркову систему та клас маси корабля, знаходить всі тимчасові з’єднання в заданій зірковій системі, що дозволяють подорожі кораблям заданої маси.

- addReport, приймає 2 агрументи: ідентифікатор тимчасового з’єднання та об’єкт StateReport, записує оновлені дані про тимчасове з’єднання в базу даних.

3.5. Розробка браузерного інтерфейсу для взаємодії із вебсистемою

В Spring Boot клсси, що відповідають за обробку запитів користувачів називаються контролерами та анотуються як @Controller. Для web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань доцільно розробити кінцеві точки для реєстрації, авторизації, створення тимчасових з’єднань, отримання інформації по тимчасовим з’єднанням, розробити форми реєстрації та авторизації, форми створення тимчасивих з’єднань, кінцева точка для запитів про постійні з’єднання, форму для налаштування та запит на формування маршруту.

Для того щоб задатизагальний шлях для запитів для конкретного контролеру необхідно використати анотацію @RequestMapping("/*"), де * – шлях. Для кожного типу запиту є окремі анотацій, в кожну з яких треба ввести рядок із шляхом. Найпопулярнішими є:

1. @GetMapping();
2. @PostMapping();
3. @PatchMapping();
4. @DeleteMapping().

Для даного вебзастосунку вирішено для всіх сторінок, що будуть відображатись користувачам відповідати через @GetMapping(), всі запити для збереження даних чи створення даних через @PostMapping(). Через @PatchMapping буде викликатись метод оновлення даних про зіркові системи.

В табл. 3.4 наведені кінцеві точки для реєстрації та авторизації.

Таблиця 3.4 – Кінцеві точки для реєстрації та авторизації

Тип запиту	**Шлях**	**Опис**
GET	/register	Відображає форму реєстрації
POST	/register-submit	Реєструє користувача, зберігає його в базу даних
GET	/login	Створює сесію та авторизує користувача
GET	/logout	Закінчує сесію користувача

В табл. 3.5 наведені кінцеві точки для роботи з тимчасовими з’єднаннями. Дані кінцеві точки нададуть змогу відображати форму створенно нового тимчасового з’єднання, зберігати тимчасове з’єднання з форми та переглядати список вже збережених з’єднань.

Таблиця 3.5 – Кінцеві точки для роботи з тимчасовими з’єднаннями

Тип запиту	**Шлях**	**Опис**
GET	/wormhole/create	Відображає форму нового тимчасового з’єднання
POST	/wormhole/submit	Зберігає введене тимчасове з’єднання в базу даних
GET	/wormhole/list	Відображає всі доступні тимчасові з’єднання

В табл. 3.6 наведені кінцеві точки для роботи із зірковими системами та постійними з’єднаннями. Ці кінцеві точки дозволять взаємодіяти із системою для оновлення бази даних постійних з’єднань з програмного інтерфейсу гри та можливості перегляду повного списку цих систем. Вони не розраховані для використання звичайними користувачами та мають бути доступні лише адміністраторам системи.

Таблиця 3.6 – Кінцеві точки для роботи з зірковими системами

Тип запиту	**Шлях**	**Опис**
PATCH	/system/update	Оновлює з відкритого програмного інтерфейсу гри дані про постійні з’єднання між зірковими системами
GET	/system/list	Відображає всі постійні з’єднання

У табл. 3.7 представлено кінцеві точки для формування маршрутів, враховуючи як постійні, так і тимчасові з'єднання. Зокрема, кінцеві точки включають два основні типи запитів: GET-запити. Перший, GET-запит за шляхом /route/form, відображає форму для налаштування маршруту перед його формуванням, дозволяючи користувачам задавати необхідні параметри. Другий, GET-запит за шляхом /route/result, відображає вже сформований маршрут

Таблиця 3.7 – Кінцеві точки для формування маршрутів

Тип запиту	**Шлях**	**Опис**
GET	/route/form	Відображає форму для налаштування маршруту для формування
GET	/route/result	Відображає сформований маршрут

Таким чином було створено кінцеві точки для взаємодії із вебсистемою. Наступним кроком буде розробка форм та сторінок для взаємодії користувача із вебсистемою.

Насамперед, неообхідно розробити форми реєстрації та авторизації, для того, щоб користувачі могли отримати доступ до системи. При цьому важливо, щоб користувач міг отримати доступ до цих форм незалежно від поточної авторизації. Всі ресурси повинні бути недоступні без авторизації та перенаправляти на форму авторизації.

Форма авторизації повинна мати посилання на форму реєстрації, щоб користувач без облікового запису все ще міг його створити. Також необхідно показати посилання на інші сторінки вебсистеми, для того щоб користувач мав уявлення на що спроможна вебсистема.

Було розроблено форму авторизації користувача (рис. 3.4).

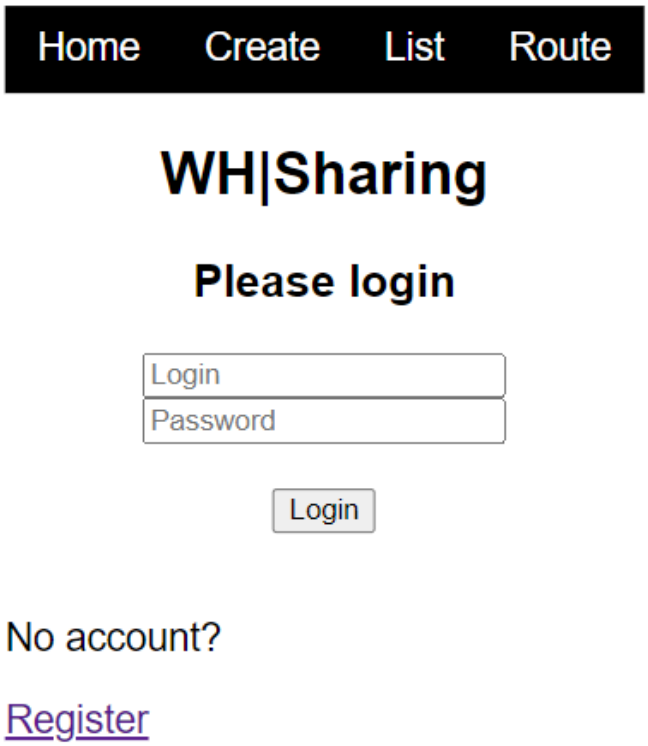


Рисунок 3.4 – Форма авторизації

Також необхідно створити форму реєстрації, що буде доступна за гіперпосиланням з форми авторизації (рис. 3.5).

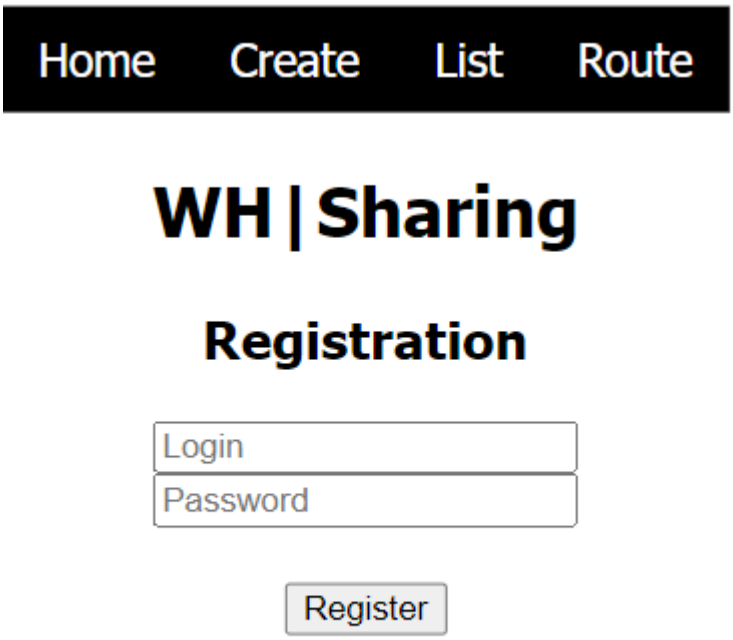


Рисунок 3.5 – Форма реєстрації

Після успішної авторизації користувачу будуть доступні всі необхідні для роботи можливості вебсистеми.

Було розроблено форму створення нового тимчасового з’єднання (рис 3.6).

Рисунок 3.6 – Форма створення тимчасового з’єднання

Було створено сторінку для відображення створених користувачами тимчасових з’єднань (рис. 3.7).

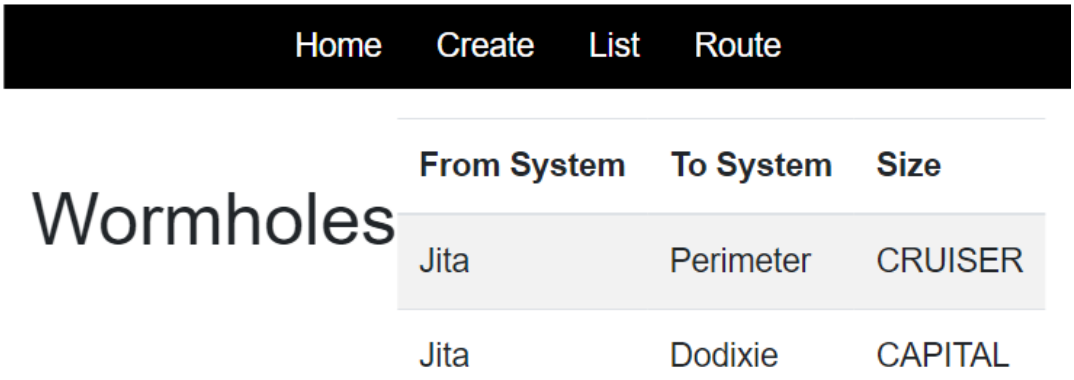


Рисунок 3.7 – Сторінка із списком всіх збережених тимчасових з’єднань

Отже, було проаналізовано різні тезнології розробки вебсистем та обрано мову Java та фреймворк Spring Boot. Було виконано підготовку до розробки програмних модулів вебсистеми, створено початковий проєкт за допомогою інструмента Spring Initializr. Було розроблено структуру програмних модулів та моделей даних для взаємодією з базою даних. Було проаналізовано алгоритми пошуку шляху та обрано алгоритм A* як основу для розробки власного алгоритму пошуку шляху. Було розроблено алгоритм пошуку шляху з урахуванням налаштувань. Були розроблені класи з бізнес-логікою вебсистеми. Було реалізовано механізми реєстрації та авторизації користувачів. Було розроблено контролери – класи для обробки запитів користувачів. Було створено форми та вебсторінки для взаємодії з вебсистемою в браузерному інтерфейсі.

РОЗДІЛ 4
ТЕСТУВАННЯ ВЕБСИСТЕМИ

4.1. Тестування продуктивності системи

Тестування продуктивності вебсистеми є важливою стадією його тестування, що допомагає виявити можливі недоліки роботи [20]. Також одним з ключових етапів тестування є перевірка сумісності системи з різними браузерами та версіями операційних систем.

Початковим кроком є встановлення тестової версії системи на персональний комп'ютер з операційною системою Windows 10. Далі проводяться функціональні тести, щоб перевірити коректність роботи системи, включаючи тести на введення та виведення даних, реакцію на негативні вхідні дані та відповідність бізнес-логіці.

Після цього виконуються тести на безпеку системи, включаючи перевірку на проникнення, збереження конфіденційної інформації та виявлення вразливостей. Для цього можуть використовуватись спеціальні програмні засоби для моделювання атак та виявлення потенційних загроз.

Після успішного проходження тестів проводиться оцінка продуктивності системи, включаючи тести на швидкість роботи, використання ресурсів та життєвий цикл. Було протестовано роботу вебсистеми в операційній системі Windows 10 (рис. 4.1).

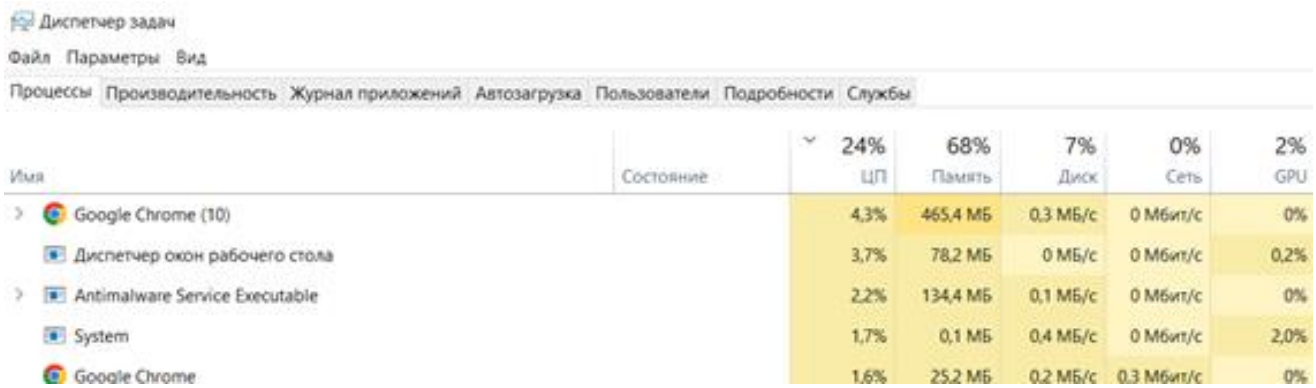


Рисунок 4.1 – Тестування продуктивності системи

Як можна побачити, запущена вебсистема дає помірне навантаження (див. рис. 4.1), що свідчить про те, що вона оптимізована.

Наступним етапом буде тестування продуктивності вебсистеми за допомогою профайлера – аналізатора роботи програм, що дозволить виявити можливі проблематичні сторони чи властивості роботи вебсистеми.

JProfiler – потужний інструмент для профілювання Java-застосунків, який надає розробникам детальну інформацію про роботу досліджуваної програми. Завдяки можливостям JProfiler, можна аналізувати використання пам’яті, продуктивність процесора, поведінку потоків, а також знаходити вузькі місця роботи програми і витік пам’яті. Інструмент забезпечує інтуїтивно зрозумілий інтерфейс, що дозволяє швидко знаходити і вирішувати проблеми в програмному коді, роблячи процес оптимізації зручним та ефективним. Однією з основних переваг JProfiler є його здатність надавати динамічний аналіз в реальному часі, що дозволяє миттєво реагувати на виявлені проблеми.
Також, JProfiler підтримує інтеграцію з популярними середовищами розробки (IDE), зокрема IntelliJ IDEA, що і використовався для розробки вебсистеми. Це робить його зручним для використання в поставленій задачі профілювання вебсистеми. Цей застосунок також пропонує можливості для віддаленого профілювання, що є корисним для аналізу програм, які працюють на серверах або в хмарних середовищах. Це актуально, бо вебсистеми, подібні рзробленій часто розгортають саме в хмарних середовищах. Завдяки широкому спектру функцій, JProfiler допомагає покращувати продуктивність Java-застосунків і забезпечує простоту оптимізації програмного забезпечення, зменшуючи час, необхідний для виявлення та усунення проблем.

Для аналізу продуктивності роботи розробленої вебсистеми було обрано застосунок JProfiler.

Було встановлено застосунок JProfiler та підключено до середовища розробки Intellij IDEA (рис. 4.2). Інтерфейс дозволяє у реальному часі спостерігати, які програмні компоненти мають на систему найбільше навантаження.

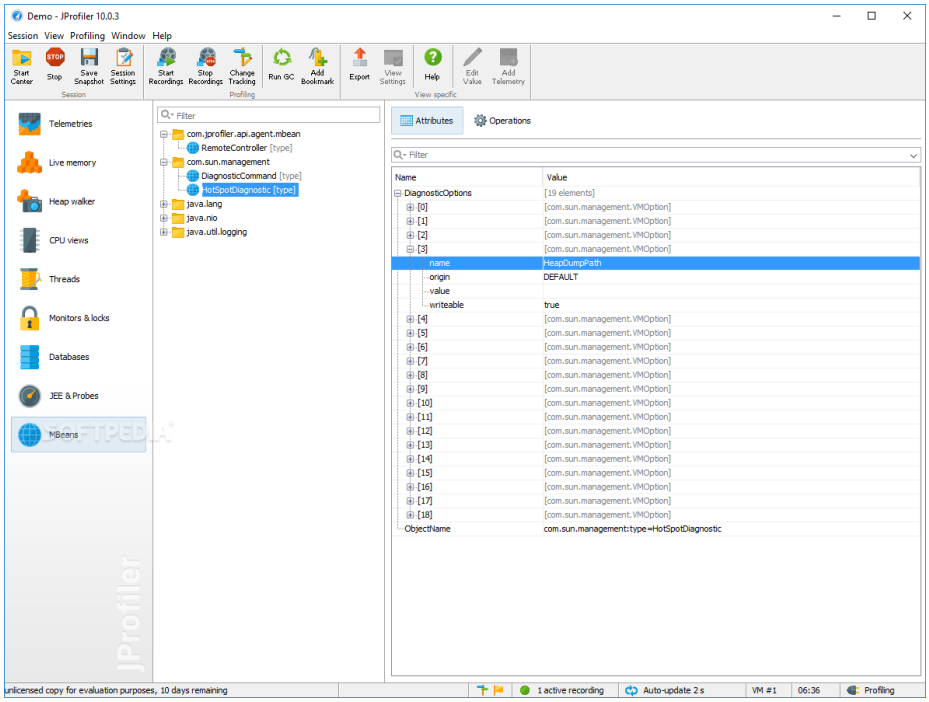


Рисунок 4.2 – Інтерфейс застосунку JProfiler

Було проведене тестування за допомогою застосунку JProfiler, результатами якого не було виявлено занадто екстремальних навантажень від окремих компонентів під час активної роботи вебсистеми (рис. 4.3).

Objects comparison

Objects: All objects

Aggregation: Classes

Name	Instance count	Size
java.util.HashMap$Node	+238 (+3 %)	+5,712 bytes
java.awt.geom.AffineTransform	+154 (+4 %)	+9,856 bytes
java.awt.geom.Path2D$Float$TxIterator	+62 (+4 %)	+1,984 bytes
java.awt.geom.Point2D$Double	+62 (+4 %)	+1,488 bytes
java.util.IdentityHashMap$KeyIterator	+62 (+4 %)	+2,480 bytes
java.util.ArrayList	+60 (+3 %)	+1,440 bytes
java.util.HashMap	+60 (+3 %)	+2,400 bytes
java.awt.geom.GeneralPath	+31 (+4 %)	+992 bytes
java.awt.geom.Point2D$Float	+31 (+4 %)	+496 bytes
java.util.concurrent.atomic.AtomicInteger	+31 (+3 %)	+496 bytes
java.awt.geom.Rectangle2D$Float	+30 (+4 %)	+720 bytes
java.util.HashMap$KeyIterator	+30 (+4 %)	+960 bytes
java.util.HashMap$KeySet	+30 (+4 %)	+480 bytes
java.util.HashMap$Node[]	+30 (+3 %)	+1,440 bytes
java.util.HashSet	+30 (+4 %)	+480 bytes
java.util.ArrayList$Itr	+1 (+2 %)	+24 bytes
java.util.concurrent.ConcurrentHashMap$V...	+1 (+3 %)	+16 bytes
java.util.concurrent.locks.AbstractQueuedS...	-126 (-12 %)	-4,032 bytes
Total:	+817 (+3 %)	+27,432 bytes

Рисунок 4.3 – Тестування вебсистеми застосунком JProfiler

4.2. Тестування можливостей системи

У сучасному світі, де інформатизація охоплює всі сфери життя, перед розробниками постають нові завдання, які потребують ефективного вирішення. Одним з таких завдань є тестування програмного забезпечення. На сьогоднішній день тестування програмного забезпечення є одним з найдорожчих етапів розробки, його вартість складає до 25% від загальних витрат на проєкт. Існує два основних підходи до реалізації процесу тестування: інтеграційне та юніт тестування. Тому важливо розглянути особливості кожного підходу, їх типи, а також визначити головні відмінності та специфіку застосування.

Тестування програмного забезпечення — це процес перевірки та оцінки програмного продукту з метою виявлення дефектів, помилок і недоліків, а також перевірки відповідності вимогам та очікуванням користувачів. Воно включає різні активності, спрямовані на перевірку функціональності, якості, безпеки та інших характеристик програмного продукту. Розглянемо методи інтеграційного та юніт тестування більш детально.

Методи інтеграційного та юніт тестування є двома основними підходами до тестування програмного забезпечення. Вони використовуються для перевірки різних аспектів програмного продукту та мають різні цілі й методики.

Інтеграційне тестування.

Мета: перевірка взаємодії між різними модулями або компонентами системи.

Методика: тестування інтеграції здійснюється на основі сценаріїв, які перевіряють взаємодію між модулями. Включає такі методи, як інкрементальне тестування, тестування на основі інтерфейсів та тестування з використанням симуляторів. Інтеграційне тестування дозволяє виявити проблеми, які виникають при взаємодії окремих частин системи.

Юніт тестування.

Мета: перевірка окремих модулів або компонентів системи на рівні вихідного коду.

Методика: юніт тестування здійснюється на рівні окремих функцій або методів програмного коду. Включає такі методи, як тестування граничних значень, тестування виключень, тестування поведінки та тестування станів. Юніт тестування дозволяє виявити дефекти на ранніх етапах розробки та забезпечити високу якість програмного коду.

Ці два підходи до тестування доповнюють один одного і застосовуються на різних етапах процесу тестування програмного забезпечення залежно від потреб і цілей тестування.

З іншого боку, для тестування подібних вебсистем більш доречно застосовувати практичний підхід – розробку та проведення тест-кейсів. Обумовлений такий вибір тим, що у розробленій вебсистемі є такі компоненти, як реєстрація та авторизація, що обумовлює користування вебсистемою лише авторизованих осіб.

Тест-кейси (тестові випадки) є важливим інструментом в арсеналі тестування програмного забезпечення. Вони визначають конкретні сценарії, які перевіряють правильність роботи окремих функцій або компонентів системи. Кожен тест-кейс містить набір передумов, кроків виконання, вхідних даних, очікуваних результатів та фактичних результатів. Основною метою тест кейсів є забезпечення систематичності та повноти тестування, що допомагає виявляти дефекти на ранніх стадіях розробки, забезпечуючи тим самим високу якість програмного забезпечення. Виконання тест-кейсів є одним із різновидів мануального (ручного) тестування.

Класифікація тест кейсів дозволяє організувати процес тестування більш ефективно. Вони можуть бути розподілені за рівнем тестування (юніт тестування, інтеграційне тестування, системне тестування та приймальне тестування) та за типом тестування (функціональне, нефункціональне, регресійне та інше). Така класифікація допомагає сконцентруватися на різних аспектах роботи системи, забезпечуючи комплексний підхід до виявлення та усунення потенційних проблем.

Загалом, тестування вебсистеми дасть змогу бути впевненими, що вебсистема спроможна виконувати всі поставлені вимоги. Цей підхід дозволить швидко та якісно провести тестування вебсистеми. Під час тестування системи для побудови маршрутів між зірковими системами гри EVE Online з використанням тимчасових з'єднань було проведено такі тести:є

1. Відкриття вебсистеми. Насамперед, важливо перевірити, наскільки швидко завантажується вебсистема, та чи працює вона взагалі.

Було відкрито вебсистему (рис. 4.4).

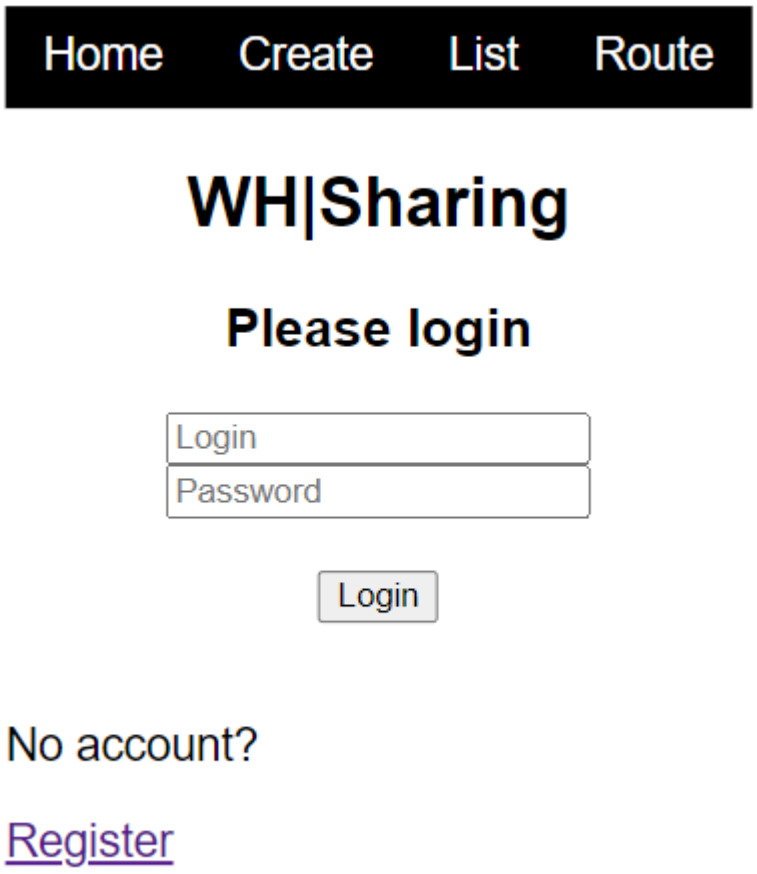


Рисунок 4.4 – Головна сторінка вебсистеми

Вебсистема була запущена менше ніж за 1 секунду, що є задовільним результатом.

2. Авторизація користувача. Для доступу користувача в вебсистему йому необхідно авторизуватись.

Було відкрито вебсистему (див. рис. 4.4).

Було введено дані авторизації (рис 4.5).

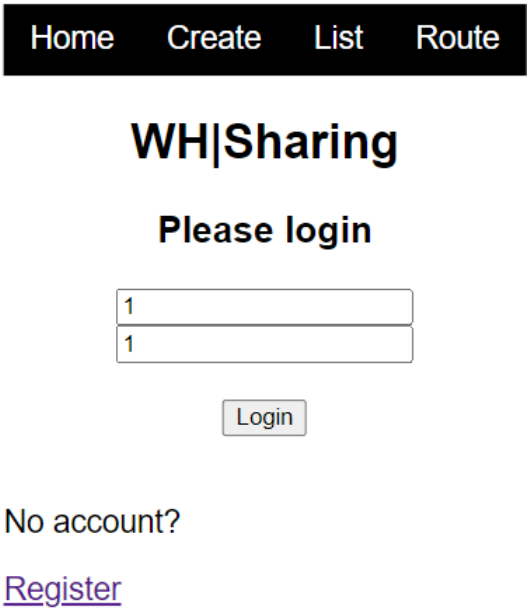


Рисунок 4.5 – Введення даних авторизації

Авторизація успішна, було надано доступ в вебсистему (рис 4.6). Тепер можливо переходити по іншим сторінкам та користуватись можливостями вебсистеми.

Рисунок 4.6 – Авторизація успішна

3. Реєстрація користувача. Якщо користувач ще не має облікового запису, у нього повинна бути можливість його створити.

Була відкрита вебсистема.

Була відкрита форма реєстрації (рис. 4.7).

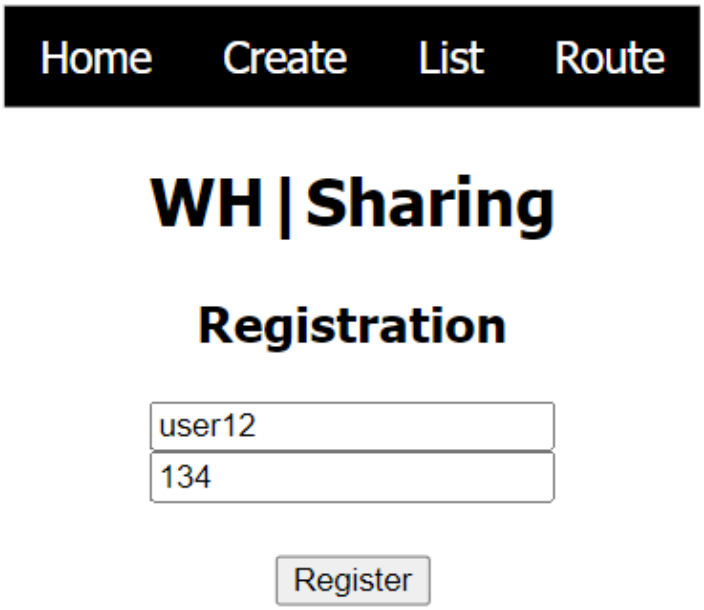


Рисунок 4.7 – Форма реєстрації

Реєстрація успішна (рис. 4.8).

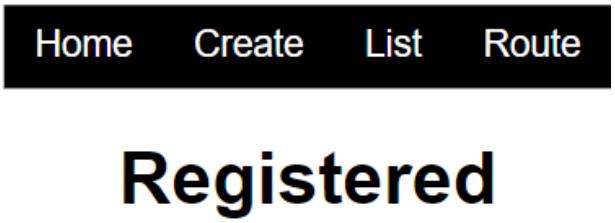


Рисунок 4.8 – Реєстрація успішна

4. Додавання нового тимчасового з’єднання. Для ефективного обміну користувачів даними про тимчасові з’єднання необхідна коректна робота додавання нового тимчасового з’єднання.

Було відкрита вебсистема.

Було авторизовано користувача.

Було відкрито сторінку створення тимчасового з'єднання (рис. 4.9).

Рисунок 4.9 – Сторінка створення тимчасового з’єднання

Було відкрито сторінку, що відображає всі тимчасові з’єднання. В списку з’явилась нещодавно додане тимчасове з’єднання.

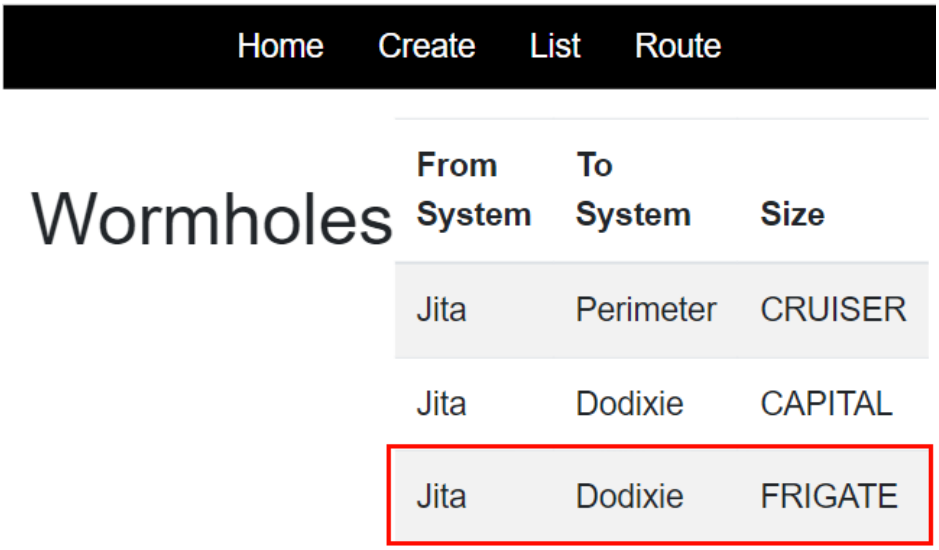


Рисунок 4.10 – Список доданих тимчасових з’єднань

Ці тест-кейси покривають більшість можливих способів користування вебсистемою та дозволять впевнитись в її роботоспроможності.

Короткий опис тест-кейсів наведено у таблиці 4.1.

Таблиця 4.1 – Тестові випадки

Іденти-фікатор	Назва	Кроки виконання	Очікуваний результат	Результат
ТК-1	Перевірка, чи відкривається програмне забезпечення	1. Відкрити вебсистему	1. Відкрита сторінка авторизації	Виконано
ТК-2	Перевірка переходу між сторінками та чи відчиняється відповідна сторінка	1. Відкрити вебсистему 2. Авторизувати користувача 3. Перевірити, що відповідна сторінка відкривається	1. Відкрита сторінка авторизації 2. Відбувся перехід на головну сторінку після авторизації 3. Перехід між сторінками працює як слід	Виконано
ТК-3	Перевірити, що користувач зможе виконати реєстрацію	1. Відкрити вебсистему 3. Ввести дані реєстрації	1. Відкрита сторінка авторизації 3. Реєстрація успішна	Виконано
ТК-4	Перевірити, що користувач зможе авторизуватись	1. Відкрити вебсистему 2. Ввести дані авторизації	1. Відкрита сторінка авторизації 2. Авторизація успішна	Виконано
ТК-5	Перевірити, що авторизований користувач може додавати нові тимчасові з’єднання	1. Відкрити вебсистему 3. Перейти на сторінку додання нового тимчасового з’єднання 4. Додати нове тимчасовез’єднання	1. Відкрита сторінка авторизації 3. Відкрита форма додавання нового тимчасового з'єднання 4. Додано нове з'єднання	Виконано

Отже, в процесі тестування вебсистеми було перевірено її функціональність, безпеку та коректність роботи. Проведені мануальні тести показали правильність реалізації введення та виведення даних та відповідність бізнес-логіці. Також були виконані тести на безпеку системи, які виявили й усунули потенційні вразливості. Оцінка продуктивності системи підтвердила її ефективність та оптимальне використання ресурсів. Було проаналізовано роботу вебсистеми профайлером та підтверджено оптимізованість використання ресурсів системи.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмні модулі вебсистеми пошуку маршрутів з допомогою тимчасових з’єднань.

Для розробки було використано середовище програмної розробки IntelliJ IDEA. Роботу виконано згідно методичних вказівок.

Було розглянуто основні аналоги програмного продукту. Було визначено їхні недоліки та порівняно з власним програмним продуктом. Базуючись на порівнянні було встановлено основні задачі бакалаврської дипломної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Java та фреймворку Spring Boot.

У бакалаврській дипломній роботі було проведено аналіз зв'язку економіки гри з віддаленістю ключових зіркових систем та сформульовано постановку задачі дослідження. Проведений аналіз середовища гри та можливостей з'єднань між зірковими системами, порівняльний аналіз аналогів та аналіз методів розв'язання поставленої задачі. Було проведено аналіз та моделювання системи заохочення надання користувачами актуальних даних тимчасових з'єднань, а також аналіз можливих негативних факторів окремих точок маршрутів та моделювання системи налаштування пошуку маршрутів. Було розглянуто способи використання системи та створено діаграми діяльності системи.

Був проведений аналіз різних наборів технологій розробки вебсистем, підготовка до розробки програмних модулів вебсистеми та сам процес розробки програмних модулів.

Було розроблено вебсистему пошуку маршрутів між зірковими системами гри EVE Online із використанням тимчасових з’єднань.

В останньому розділі було проведено тестування системи. Було проведено тестування продуктивності та тестування можливостей вебсистеми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Масова багатокористувацька онлайн гра. *EVE Online :* веб-сайт. URL: https://uk.wikipedia.org/wiki/EVE_Online (дата звернення: 14.03.2024).

2. Які є етапи в розробці вебсайтів? *Розробка вебсайтів* : веб-сайт. URL: https://webtune.com.ua/statti/web-rozrobka/etapy-stvorennya-veb-sajtiv (дата зверненя: 17.03.2024).

3. Юркевич Д.С. Хошаба О.М. Аналіз засобів розробки вебсистем фреймворку Spring Boot. *Молодь в науці: дослідження, проблеми, перспективи*. 30 травня 2024, Вінниця: ВНТУ, 2024. 1 с.

4. Як шукати тимчасові з'єднання самостійно? *EVE Online. Wormholes* : веб-сайт. URL: https://wiki.eveuniversity.org/Wormholes (дата звернення: 23.03.2024).

5. Де знайти точні дані по тимчасовим з’єднанням найбільших зіркових систем? *EVE Scout. Wormhole Connections* : веб-сайт. URL: https://www.eve-scout.com/# (дата звернення: 26.03.2024).

6. Де знайти карту галактики гри? *Dotlan. EVE maps* : веб-сайт. URL: https://evemaps.dotlan.net (дата звернення: 29.03.2024).

7. Де обмінюватись тимчасовими з’єднаннями? *Pathfinder* : веб-сайт. URL: https://pathfinder.eve-linknet.com (дата звернення: 01.04.2024).

8. Поширення координат вбудованими засобами. *Теки координат* : веб-сайт. URL: https://forums.eveonline.com/t/shared-bookmarks-use-case-discussion/171216 (дата звернення: 04.04.2024).

9. Sam Newman. Building Microservices: Designing Fine-Grained Systems : навч. посібник. O'Reilly, 2021. 3 с.

10. Розробка вебсистем. *Розробка вебсистем на мові Java* : веб-сайт. URL: https://avada-media.ua/ua/services/razrabotka-veb-sistem-dlya-logistiki-na-java (дата звернення: 09.04.2024).

11. Як сканувати тимчасові з'єднання? *Сканування тимчасових з’єднань* : веб-сайт. URL: https://wiki.eveuniversity.org/Wormhole_scouting (дата звернення: 12.04.2024).

12. Як захистити корабель на стадії планування маршруту подрожі? *Заходи безпеки при плануванні маршруту* : веб-сайт. URL: https://gamerok.com/en/How-to-stay-safe-in-Eve-Online (дата звернення: 15.04.2024).

13. Що таке статус безпеки зіркової системи? *Статус безпеки зіркової системи* : веб-сайт. URL: https://wiki.eveuniversity.org/System_security (дата звернення: 18.04.2024).

14. Що таке торгівельний вузол? *Торгівельні вузли всесвіту гри* : веб-сайт. URL: https://wiki.eveuniversity.org/Trade_hubs (дата звернення: 21.04.2024).

15. Craig Walls. Spring Boot in action : навч. посібник. O`Reilly, 2018. 20 с.

16. Інструмент для початку роботи в Spring Boot. *Spring Initializr* : веб-сайт. URL: https://start.spring.io (дата звернення: 27.04.2024).

17. Як покращити читабельність Java-коду? *Project Lombok* : веб-сайт. URL: https://projectlombok.org (дата звернення 30.04.2024).

18. Як використовувати шаблонізатор вебсторінок? *Thymeleaf tutorial* : веб-сайт. URL: https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf (дата звернення: 03.05.2024).

19. Як структурувати проєкти Spring Boot? *Структура проєктів Spring Boot* : веб-сайт. URL: https://foxminded.ua/ru/spring-boot (дата звернення: 06.05.2024).

20. Як тестувати роботу веб-сайтів? *Тестування веб-сайту* : веб-сайт. URL: https://brainlab.com.ua/uk/blog-uk/yak-testuvati-veb-sayt-osnovn-etapi-poradi (дата звернення: 10.05.2024).

ДОДАТКИ

Додаток А
Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань»
121 Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., проф. Хошаба О.М.
«12» березня 2024 р.

Виконав:
_____ студент гр. 1ПІ-20б Д. С. Юркевич
«12» березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань».

Галузь застосування аналітика даних і оптимізація алгоритмів пошуку.

2. Підстава для розробки.

Підставою для розробки даної бакалаврської дипломної роботи є рішення засідання кафедри програмного забезпечення.

3. Мета та призначення розробки.

Основною метою бакалаврської дипломної роботи є надання можливості користувачам обмінюватись даними про тимчасові з’єднання та будувати за допомогою них шляхи за допомогою одного вебресурсу. Відповідно до поставленої мети необхідно вирішити такі завдання:

- провести аналіз аналогічних програмних продуктів для визначення актуальності покращень таких продуктів;
- розробити базу даних, змоделювати сутності системи, з вимогами серверної частини;
- розробити модифікацію алгоритму пошуку шляху для потреб вебсистеми;
- розробити модулі серверної частини, що взаємодіють з базою даних;
- розробити модуль графічного інтерфейсу, який ефективно забезпечуватиме користувачеві взаємодію із системою;
- провести тестування розробленої вебсистеми.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Масова багатокористувацька онлайн гра. *EVE Online :* веб-сайт. URL: https://uk.wikipedia.org/wiki/EVE_Online (дата звернення: 14.03.2024).

2. Як сканувати тимчасові з'єднання? *Сканування тимчасових з'єднань* : веб-сайт. URL: https://wiki.eveuniversity.org/Wormhole_scouting (дата звернення: 12.04.2024).

3. Що таке торгівельний вузол? *Торгівельні вузли всесвіту гри* : веб-сайт. URL: https://wiki.eveuniversity.org/Trade_hubs (дата звернення: 21.04.2024).

4. Як структурувати проєкти Spring Boot? *Структура проєктів Spring Boot* : веб-сайт. URL: https://foxminded.ua/ru/spring-boot (дата звернення: 06.05.2024).

5. Технічні вимоги

- процесор 2-ядерний з частотою не менше 2.0 ГГц;
- об'єм оперативної пам'яті 4 ГБ;
- місце на жорсткому диску 4 ГБ;
- операційна система Windown, Linux або MacOS.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи постановка задачі	14.03.24 – 24.03.24
2	Розробка структури, методів та алгоритму роботи програмного продукту	25.03.24 – 15.04.24
3	Розробка програмних компонентів	15.04.24 – 30.04.24
4	Тестування програми	01.05.24 – 10.05.24
5	Оформлення матеріалів до захисту БДР	11.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Перевірка на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка web-застосунку для побудови маршрутів між модулями гри EVE Online з використанням тимчасових з'єднань»

Тип роботи: БДР

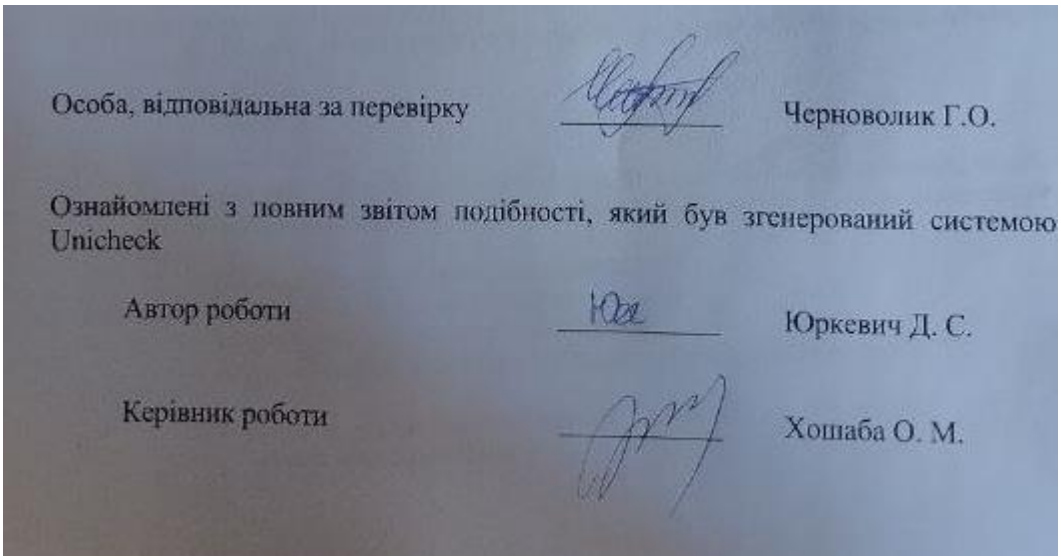
Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. Хошаба О. М.

Оригінальність	**97.6%**
Схожість	**2.4%**

Аналіз звіту подібності

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
 Виявлені у роботі запозичення не мають ознак плагіату, їх велика надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора.
 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.


Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Юркевич Д. С.

Керівник роботи Хошаба О. М.

Додаток В. Лістинг програмних модулів

```
@Component
@RequiredArgsConstructor
public class PathfindingAlgorithm {

   private final WormholeService wormholeService;
   private final SystemService systemService;

   public List<System> findOptimalRoute(RouteSettings routeSettings) {
      System start = systemService.findSolarSystemByNameOrElseThrow(routeSettings.getStart());
      System goal = systemService.findSolarSystemByNameOrElseThrow(routeSettings.getEnd());

      PriorityQueue<RouteNode> openSet = new
PriorityQueue<>(Comparator.comparingDouble(RouteNode::getFScore));
      Map<System, RouteNode> allNodes = new HashMap<>();
      Set<System> closedSet = new HashSet<>();

      RouteNode startNode = new RouteNode(start, 0, heuristic(start, goal));
      allNodes.put(start, startNode);
      openSet.add(startNode);

      while (!openSet.isEmpty()) {
         RouteNode currentNode = openSet.poll();
         System currentSystem = currentNode.getSystem();

         if (currentSystem.equals(goal)) {
            return reconstructPath(currentNode);
         }

         closedSet.add(currentSystem);

         for (System neighbor : wormholeService.findAllWormholeConnections(currentSystem,
Size.valueOf(routeSettings.getSize()))) {
            if (closedSet.contains(neighbor) || routeSettings.getIgnoredSystems().contains(neighbor))
{
               continue;
            }

            double tentativeGScore = currentNode.getGScore() + distance(currentSystem, neighbor);
            RouteNode neighborNode = allNodes.getOrDefault(neighbor, new RouteNode(neighbor));

            if (tentativeGScore < neighborNode.getGScore()) {
               neighborNode.setPreviousNode(currentNode);
               neighborNode.setGScore(tentativeGScore);
               neighborNode.setFScore(tentativeGScore + heuristic(neighbor, goal));

               if (!openSet.contains(neighborNode)) {
                  openSet.add(neighborNode);
               }
            }
```

```
                allNodes.put(neighbor, neighborNode);
            }
        }

        throw new RuntimeException("No route found");
    }

    private double heuristic(System a, System b) {
        // Implement a heuristic function, e.g., Euclidean distance or another heuristic appropriate for your domain
        return 0; // Placeholder
    }

    private double distance(System a, System b) {
        // Implement the actual distance calculation between systems
        return 1; // Placeholder
    }

    private List<System> reconstructPath(RouteNode node) {
        List<System> path = new ArrayList<>();
        while (node != null) {
            path.add(node.getSystem());
            node = node.getPreviousNode();
        }
        Collections.reverse(path);
        return path;
    }

    private static class RouteNode {
        private final System system;
        private RouteNode previousNode;
        private double gScore;
        private double fScore;

        public RouteNode(System system) {
            this(system, Double.POSITIVE_INFINITY, Double.POSITIVE_INFINITY);
        }

        public RouteNode(System system, double gScore, double fScore) {
            this.system = system;
            this.gScore = gScore;
            this.fScore = fScore;
        }

        public System getSystem() {
            return system;
        }

        public RouteNode getPreviousNode() {
            return previousNode;
        }

        public void setPreviousNode(RouteNode previousNode) {
```

```
            this.previousNode = previousNode;
        }

        public double getGScore() {
            return gScore;
        }

        public void setGScore(double gScore) {
            this.gScore = gScore;
        }

        public double getFScore() {
            return fScore;
        }

        public void setFScore(double fScore) {
            this.fScore = fScore;
        }
    }
}

@RequiredArgsConstructor
@Controller
@RequestMapping("/route")
public class RouteController {

    private final RouteService routeService;

    private final SystemService systemService;


    @GetMapping("/form")
    public String createForm(Model model) {
        model.addAttribute("settings", new RouteSettings());
        return "/route/create";
    }

    @GetMapping("/result")
    public String makeRoute(Model model, RouteSettings routeSettings) throws ApiException {
        String routeString = String.join(" -> ", routeService.findRoute(routeSettings));
        model.addAttribute("route", routeString);
        return "/route/result";
    }

}

@Controller
@RequiredArgsConstructor
public class SecurityController {

    private final UserService userService;
```

```
    @GetMapping("/register")
    public String registerForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }

    @PostMapping("/register-submit")
    public String registerSubmit(@Valid User user) {
        userService.saveUser(user);
        return "redirect:/login";
    }

    @GetMapping("/login")
    public String loginForm(Model model) {
        model.addAttribute("user", new User());
        return "login";
    }

    @GetMapping("/logout")
    public String logout(HttpServletRequest request) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if (authentication != null) {
            request.getSession().invalidate();
        }
        return "redirect:/login";
    }

}

@RestController
@RequiredArgsConstructor
@RequestMapping("/system")
public class SystemController {

    private final SystemService systemService;

    @PatchMapping("/update")
    public String updateSystemsDBTable() {
        systemService.updateSolarSystemsDB();
        return "updating";
    }

    @GetMapping("/count")
    public String countSystems() {
        return String.valueOf(systemService.countSystems());
    }

}

@Controller
@RequiredArgsConstructor
@RequestMapping("/wormhole")
public class WormholeController {
```

```
    private final WormholeService wormholeService;
    private final SystemService systemService;

    @GetMapping("/create")
    public String wormholeCreatingPage(Model model) {
        model.addAttribute("wormhole", new WormholeDTO());
        return "wormhole/create";
    }

    @PostMapping("/submit")
    public String createWormhole(WormholeDTO wormholeDTO) {
        Wormhole wormhole = Wormhole.builder()

.fromSystem(systemService.findSolarSystemByNameOrElseThrow(wormholeDTO.getFrom()))

.toSystem(systemService.findSolarSystemByNameOrElseThrow(wormholeDTO.getTo()))
                .size(wormholeDTO.getSize())
                .build();
        wormholeService.saveWormhole(wormhole);
        return "redirect:/wormhole/create";
    }

    @GetMapping("/list")
    public String listAllWormholes(Model model) {
        model.addAttribute("wormholes", wormholeService.findAll());
        return "wormhole/list";
    }


}

@Getter
@ToString
public class SystemDTO {

    private final String name;

    private final Double security;

    private final Set<String> jumpgates;


    public SystemDTO(System system) {
        name = system.getName();
        security = system.getSecurity();
        jumpgates = system.getJumpgates().stream()
                .map(System::getName)
                .collect(Collectors.toSet());
    }

}
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class WormholeDTO {

   private String from;

   private String to;

   private Size size;

}

public enum MassState {
   STABLE,
   HALFCRIT,
   CRIT
}

public enum SecurityStatus {

   HIGHSEC,
   LOWSEC,
   NULLSEC;

   public static SecurityStatus of(double sec) {
      return sec >= 0.5 ? HIGHSEC
            : sec > 0.0 ? LOWSEC
            : NULLSEC;
   }

   public boolean test(double sec) {
      SecurityStatus actual = of(sec);
      return actual.ordinal() < this.ordinal();
   }

}

public enum Size {
   FRIGATE,
   CRUISER,
   BATTLESHIP,
   CAPITAL
}

public enum TimeState {
   VERY_FRESH,
   FRESH,
   EOL
}

//@Component
```

```
@RequiredArgsConstructor
public class ServerInitializer implements ApplicationRunner {

    private final SystemService systemService;

    @Override
    public void run(ApplicationArguments args) {
//        systemService.updateSolarSystemsDB();
    }
}

public class EntityNotFoundException extends RuntimeException {

    public EntityNotFoundException(Long id, Class<?> entity) {
        super("The " + entity.getSimpleName().toLowerCase() + " with id '" + id + "' does not exist
in our records");
    }

}

public class SystemNotFoundException extends RuntimeException {

    public SystemNotFoundException(String system) {
        super(system);
    }

}


public class WormholeNotFoundException extends RuntimeException {

    public WormholeNotFoundException(Long wormholeId) {
        super("Wormhole not found: " + wormholeId);
    }

}

@Entity
@Data
public class StateReport {

    @Id
    @GeneratedValue
    private Long id;

    @Enumerated(value = EnumType.STRING)
    private MassState massState;

    @Enumerated(value = EnumType.STRING)
    private TimeState timeState;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime time = LocalDateTime.now();
```

```
}

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class System {

   @Id
   private String name;

   private Integer id;

   private Double security;

   @JsonIgnore
   @ManyToMany(fetch = FetchType.LAZY)
   @JoinTable(
         name = "jumpgate_connection",
         joinColumns = @JoinColumn(name = "left_system_name", referencedColumnName =
"name"),
         inverseJoinColumns = @JoinColumn(name = "right_system_name",
referencedColumnName = "name")
   )
   private Set<System> jumpgates;

   @OneToMany(mappedBy = "fromSystem", cascade = CascadeType.ALL, orphanRemoval =
true)
   private List<Wormhole> fromWormholes = new ArrayList<>();

   @OneToMany(mappedBy = "toSystem", cascade = CascadeType.ALL, orphanRemoval = true)
   private List<Wormhole> toWormholes = new ArrayList<>();

   public List<Wormhole> getAllWormholes() {
      List<Wormhole> allWormholes = new ArrayList<>();
      allWormholes.addAll(fromWormholes);
      allWormholes.addAll(toWormholes);
      return allWormholes;
   }

}

@Table(name = "users")
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class User implements UserDetails {
```

```
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @NotBlank(message =  "username cannot be blank")
    @NonNull
    @Column(nullable = false, unique = true)
    private String username;

    @NotBlank(message =  "password cannot be blank")
    @NonNull
    @Column(nullable = false)
    private String password;



    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(() -> "user");
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

}

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
public class Wormhole {

    @Id
    @GeneratedValue
    private Long id;
```

```
    @ManyToOne
    @JoinColumn(name = "from_system_name", nullable = false)
    private System fromSystem;

    @ManyToOne
    @JoinColumn(name = "to_system_name", nullable = false)
    private System toSystem;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime discoveryTime = LocalDateTime.now();

    @Enumerated(value = EnumType.STRING)
    private Size size;

    @OneToMany(fetch = FetchType.LAZY)
    @JoinTable(
            name = "wormhole_state_reports",
            joinColumns = @JoinColumn(name = "wormhole_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "state_report_id", referencedColumnName =
"id")
    )
    private List<StateReport> reports;

}

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class RouteSettings {

    private String start;

    private String end;

    private String size;

    private String sec;

    private Set<String> ignoredSystems = Set.of();

}

@Repository
public interface StateReportRepository extends JpaRepository<StateReport, Long> {
}

@Repository
public interface SystemRepository extends JpaRepository<System, String> {
    Optional<System> findSystemById(Integer id);
}
```

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
   Optional<User> findByUsername(String username);
   boolean existsByUsername(String username);
}

@Repository
public interface WormholeRepository extends JpaRepository<Wormhole, Long> {
}

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {

   private final UserService userService;

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
      httpSecurity
            .authorizeHttpRequests(auth -> auth
                  .requestMatchers("/css/**").permitAll()
                  .requestMatchers("/js/**").permitAll())
            .authorizeHttpRequests(auth -> auth
                  .requestMatchers(LOGIN_ENDPOINT).permitAll()
                  .requestMatchers(REGISTER_ENDPOINT).permitAll()
                  .requestMatchers(HttpMethod.POST, "/register-submit").permitAll())
            .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
            .formLogin(login -> login
                  .loginPage("/login"))
            .logout(logout -> logout
                  .logoutUrl("/logout")
                  .logoutSuccessUrl("/login"));
      return httpSecurity.build();
   }

   @Bean
   public UserDetailsService userDetailsService() {
      return userService::getUser;
   }

}

public class SecurityConstants {

   public static final String LOGIN_ENDPOINT = "/login";

   public static final String REGISTER_ENDPOINT = "/register";

}

@Service
@RequiredArgsConstructor
```

```
public class RouteService {

    private final RoutesApi routesApi;

    private final SystemService systemService;

    private final WormholeService wormholeService;


    public List<String> findRoute(RouteSettings routeSettings) throws ApiException {
        return routesApi.getRouteOriginDestination(
                systemService.findSolarSystemByNameOrElseThrow(routeSettings.getStart()).getId(),
                systemService.findSolarSystemByNameOrElseThrow(routeSettings.getEnd()).getId(),
                routeSettings.getIgnoredSystems().stream()
                        .map(name -> systemService.findSolarSystemByNameOrElseThrow(name).getId())
                        .collect(Collectors.toList()),
                new ArrayList<>(),
                "tranquility",
                "shortest",
                ""
        ).stream()
                .map(i -> systemService.findSystemById(i).getName())
                .toList();
    }

}

@Service
@RequiredArgsConstructor
public class SystemService {

    private final SystemRepository systemRepository;
    private final UniverseApi universeApi;

    @SneakyThrows
    public void updateSolarSystemsDB() {

        String datasource = "tranquility";
        systemRepository.deleteAll();

        List<Integer> systemIds = universeApi.getUniverseSystems(datasource, "");

        List<List<Integer>> slicedSystemIds = new ArrayList<>();
        for (int i = 0; i < systemIds.size(); i+= 1000) {
            slicedSystemIds.add(systemIds.subList(i, Math.min(1000 + i, systemIds.size())));
        }

        List<PostUniverseNames200Ok> universeNames = new ArrayList<>();
        for (List<Integer> ids: slicedSystemIds) {
            universeNames.addAll(universeApi.postUniverseNames(ids, datasource));
        }

        for (PostUniverseNames200Ok systemName: universeNames) {
```

```
        if (systemName.getCategory() == SOLAR_SYSTEM) {
            GetUniverseSystemsSystemIdOk systemInfo =
universeApi.getUniverseSystemsSystemId(systemName.getId(), "en", datasource, "false", "en");

            List<Integer> stargatesIds = systemInfo.getStargates();
            List<String> connectedSystemNames = new ArrayList<>();
            for (Integer stargateId: stargatesIds) {
                GetUniverseStargatesStargateIdOk stargateInfo =
universeApi.getUniverseStargatesStargateId(stargateId, datasource, "false");
                String neighbourSystemName = stargateInfo.getName().substring(10,
stargateInfo.getName().length() - 1);
                connectedSystemNames.add(neighbourSystemName);
            }

            Set<System> neighbours = connectedSystemNames.stream()
                    .map(name -> {
                        Optional<System> ssOptional = findSolarSystemByName(name);
                        if (ssOptional.isPresent()) {
                            return ssOptional.get();
                        } else {
                            System ss = System.builder().name(name).build();
                            saveSolarSystem(ss);
                            return ss;
                        }
                    })
                    .collect(Collectors.toSet());

            System system = findSolarSystemByName(systemName.getName())
                    .orElseGet(() -> System.builder()
                            .name(systemName.getName())
                            .build());

            system.setJumpgates(neighbours);
            system.setSecurity(Double.valueOf(systemInfo.getSecurityStatus()));
            system.setId(systemInfo.getSystemId());
            saveSolarSystem(system);
        }
    }
}

public Long countSystems() {
    return systemRepository.count();
}

public List<SystemDTO> findAllSystems() {
    return systemRepository.findAll().stream()
            .map(SystemDTO::new)
            .toList();
}

public Optional<System> findSolarSystemByName(String name) {
    return systemRepository.findById(name);
}
```

```
    public System findSolarSystemByNameOrElseThrow(String name) {
        return systemRepository.findById(name).orElseThrow(() -> new
SystemNotFoundException(name));
    }

    public System findSystemById(Integer id) {
        return systemRepository.findSystemById(id).orElseThrow(() -> new
SystemNotFoundException("Unknown"));
    }

    public void ensureSystemExistsByName(String name) {
        findSolarSystemByName(name).orElseThrow(() -> new SystemNotFoundException(name));
    }

    public void saveSolarSystem(System system) {
        systemRepository.save(system);
    }

}

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    private final BCryptPasswordEncoder bCryptPasswordEncoder;

    public User getUser(Long id) {
        Optional<User> user = userRepository.findById(id);
        return unwrapUser(user, id);
    }

    public User getUser(String username) {
        Optional<User> user = userRepository.findByUsername(username);
        return unwrapUser(user, 404L);
    }

    public User saveUser(User user) {
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        return userRepository.save(user);
    }

    static User unwrapUser(Optional<User> entity, Long id) {
        if (entity.isPresent()) return entity.get();
        else throw new EntityNotFoundException(id, User.class);
    }
}

@Service
@RequiredArgsConstructor
public class WormholeService {
```

```
    private final StateReportRepository stateReportRepository;

    private final WormholeRepository wormholeRepository;

    public List<Wormhole> findAll() {
        return wormholeRepository.findAll();
    }

    public List<System> findAllWormholeConnections(System system, Size maxSize) {
        List<System> result = new ArrayList<>();
        wormholeRepository.findAll().forEach(wh -> {
            if (! (wh.getSize().ordinal() >= maxSize.ordinal())) return;
            if (wh.getFromSystem().getName().equals(system.getName())) {
                result.add(wh.getToSystem());
            } else if (wh.getToSystem().getName().equals(system.getName())) {
                result.add(wh.getFromSystem());
            }
        });
        return result;
    }

    public Wormhole saveWormhole(Wormhole wormhole) {
        return wormholeRepository.save(wormhole);
    }

    public Wormhole addReport(Long wormholeId, StateReport stateReport) {
        Wormhole wormhole = wormholeRepository.findById(wormholeId)
                .orElseThrow(() -> new WormholeNotFoundException(wormholeId));
        stateReportRepository.save(stateReport);
        wormhole.getReports().add(stateReport);
        wormholeRepository.save(wormhole);
        return wormhole;
    }

}

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = SolarSystemExistsValidator.class)
public @interface SolarSystemExists {

    String message() default "Wrong solar system name";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

}

@RequiredArgsConstructor
@Component
public class SolarSystemExistsValidator implements ConstraintValidator<SolarSystemExists,
System> {
```

```
    private final SystemService systemService;

    @Override
    public boolean isValid(System ss, ConstraintValidatorContext constraintValidatorContext) {
        return systemService.findSolarSystemByName(ss.getName()).isPresent();
    }

}

@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UniverseApi getUniverseApi() {
        return new UniverseApi();
    }

    @Bean
    public RoutesApi getRoutesApi() {
        return new RoutesApi();
    }

}

@SpringBootApplication
public class WhsharingApplication {

        public static void main(String[] args) {
                SpringApplication.run(WhsharingApplication.class, args);
        }

}
```

Додаток Г. Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ ПОБУДОВИ МАРШРУТІВ МІЖ МОДУЛЯМИ ГРИ EVE ONLINE З ВИКОРИСТАННЯМ ТИМЧАСОВИХ З’ЄДНАНЬ

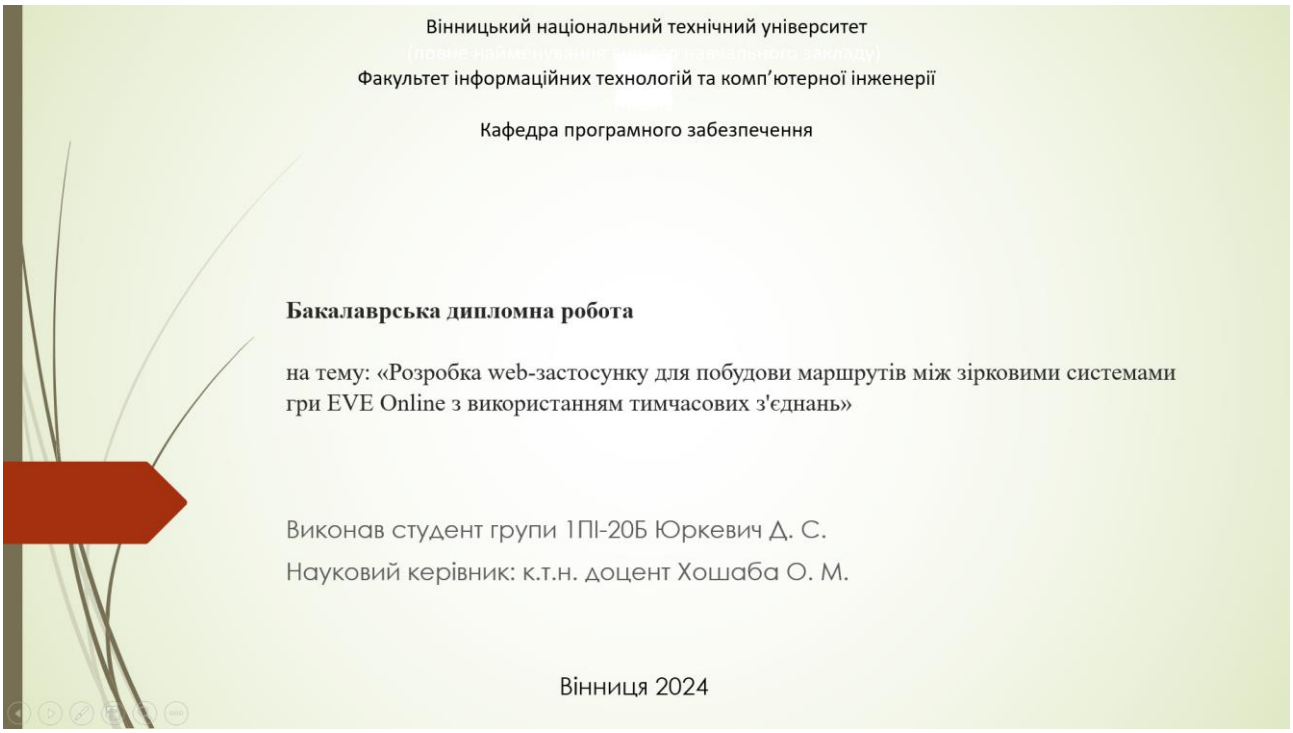


Рисунок Г.1 – Титульний слайд

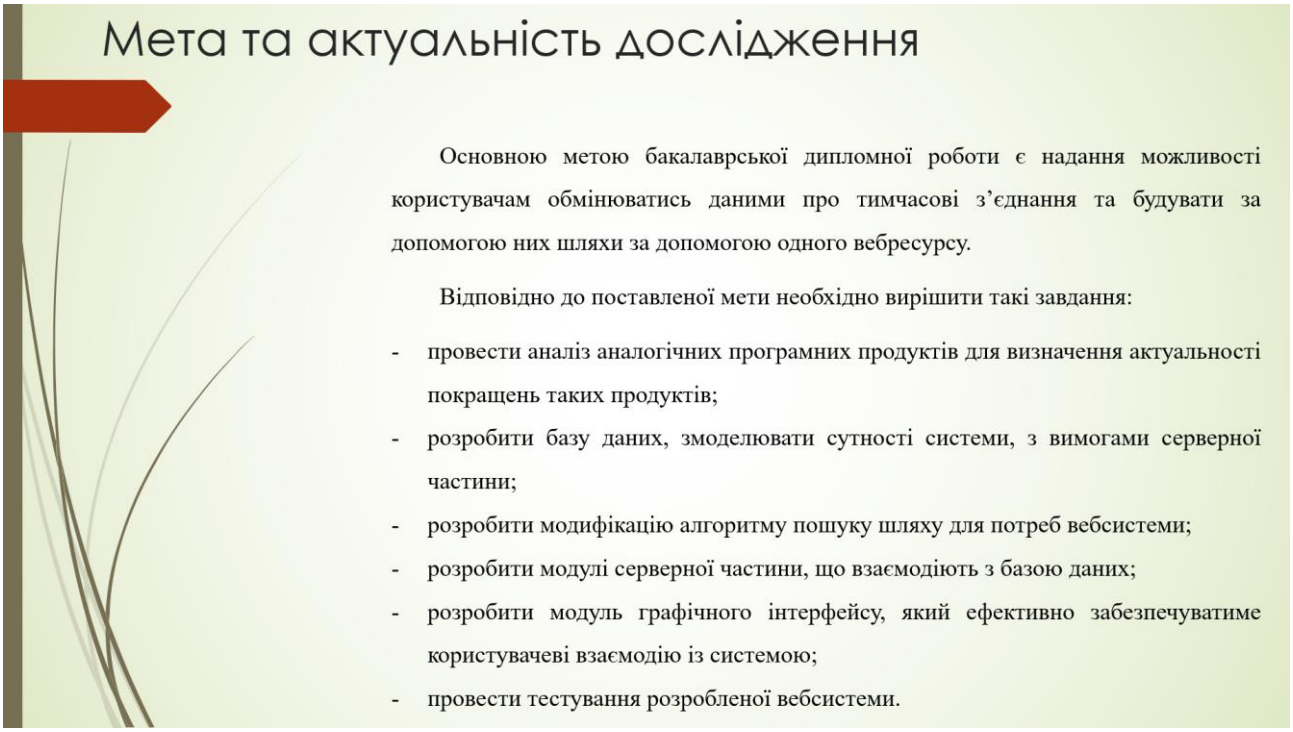


Рисунок Г.2 – Мета та актуальність дослідження

Аналіз аналогів

Критерії порівняння	Програмні продукти			
	EVE Scout	Dotlan	Pathfinder	Власний програмний продукт
Побудова маршрутів	1	4	4	5
Налаштування побудови маршрутів	1	3	3	5
Актуальність даних про наявні тимчасові з'єднання	5	0	4	5
Зручність роботи з інтерфейсом	5	3	4	5
Обмін тимчасовими з'єднаннями	0	0	3	5

Рисунок Г.3 – Аналіз аналогів

Наукова новизна

1. Модифіковано алгоритм пошуку шляху A*, що полягає у можливості налаштування пошуку та дозвоняє його використання при побудові шляхів в динамічному графі.
2. Розроблено модель заохочення надання актуальних даних користувачами вебсистеми без можливості отримання принципово достовірних даних, що полягає в колективної оцінці даних та дає можливість ефективного обміну даних користувачами.

Рисунок Г.4 – Наукова новизна

Рисунок Г.4 – Діаграма прецедентів

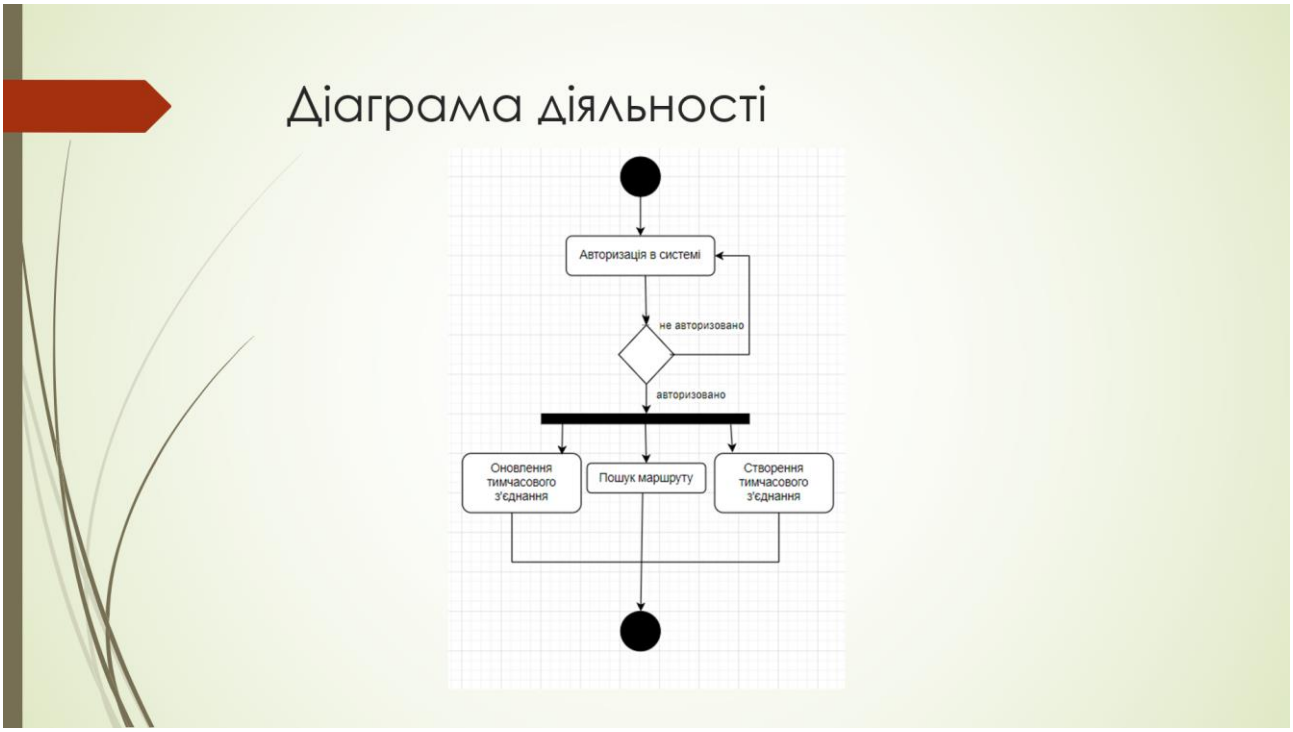


Рисунок Г.5 – Діаграма діяльності

Рисунок Г.6 – Використані технології

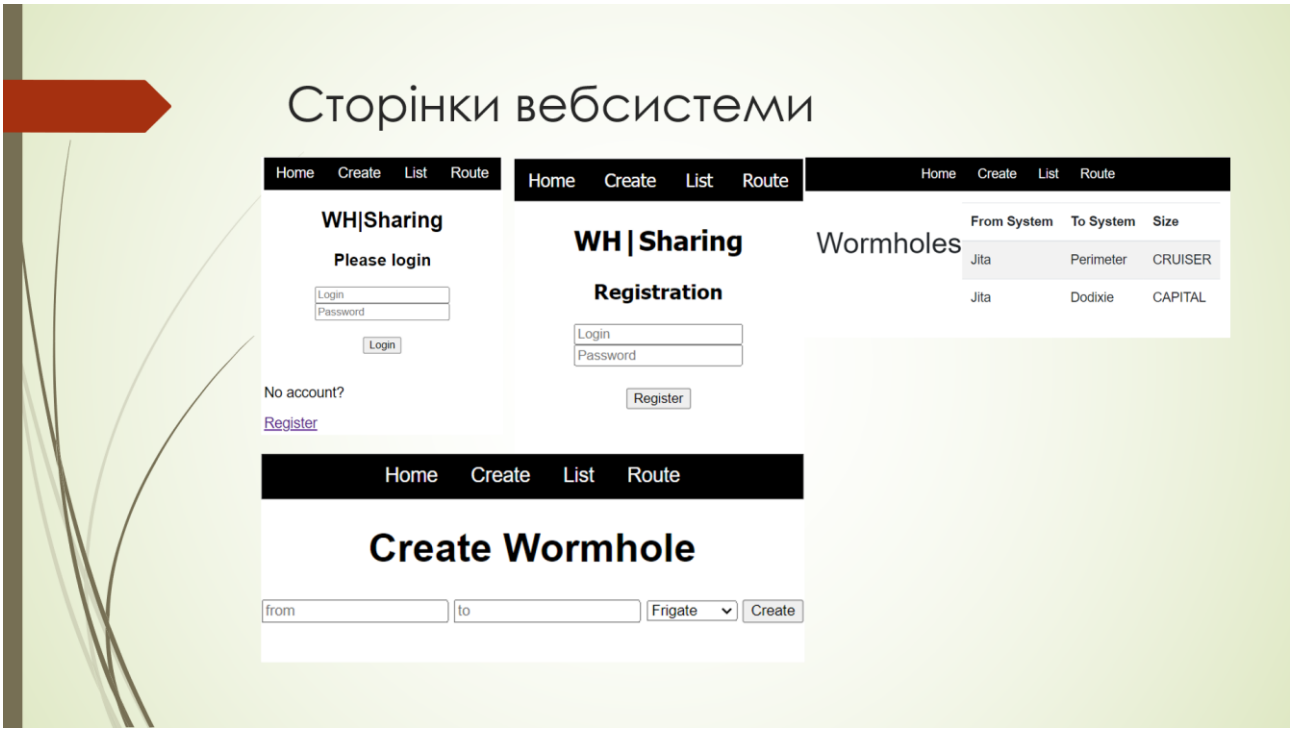


Рисунок Г.7 – Шаблонні сторінки вебсистеми

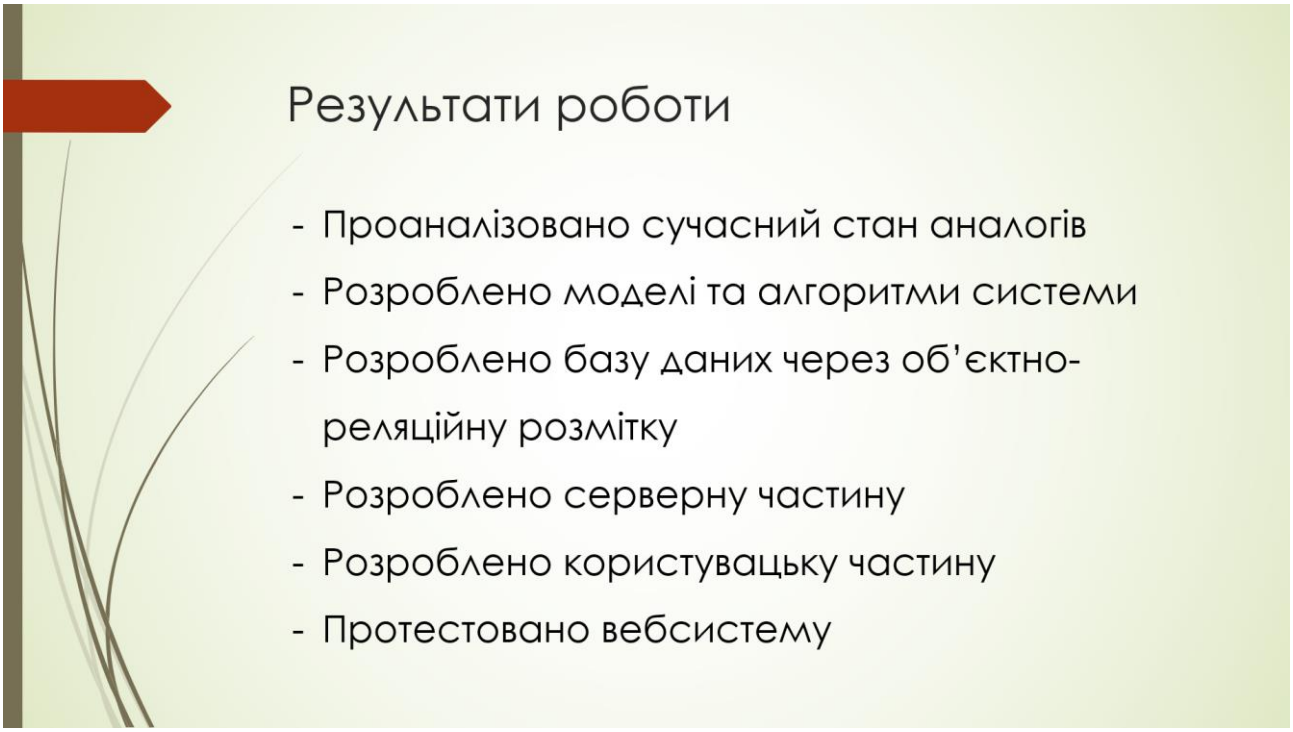


Рисунок Г.8 – Результати роботи

Рисунок Г.9 – Кінцевий слайд