НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій Стіренко

«___»_____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмування комп'ютерних систем та мереж»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Розподілений сервер карткової гри»

Виконав (-ла):
студент (-ка) IV курсу, групи ІВ-71
Федоряченко Ярослав Павлович _____

Керівник:
Асистент
Шевело Олексій Павлович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецезент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

(Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Федоряченка Ярослава Павловича

1. Тема проєкту *Розподілений сервер карткової гри,*
керівник проєкту Шевело Олексій Павлович,
затверджені наказом по університету від _____ 2021 року №_____
2. Термін здачі студентом закінченого проєкту _____ *2021 р.*
3. Вихідні дані до проєкту *див. технічне завдання*
4. Зміст пояснювальної записки (перелік питань, які розробляються) *Аналіз задачі та існуючих рішень, підбір технологій для реалізації задачі, реалізація сервера*
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, функціональна схема системи, принципова схема алгоритму взаємодії сервера з користувачем

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*17.06.2021*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

Головна ідея даної роботи – розробка розподіленого серверу карткової гри для надання прикладу написання простих розподілених серверів, а також потенційно для майбутньої розробки клієнта цієї гри. Головною метою для мене є створення сервера, щоб він був розподіленим, складався з незалежно розгортуваних модулів, та щоб його можна було легко розвивати й доповнювати як шляхом зміни існуючих модулів, так і шляхом написання нових модулів.

Щоб зробити каркас для надбудов максимально простим він не матиме деяких функцій звичних для користувачів сучасних багатокористувацьких ігор, проте він надаватиме більше свободи розробникам, що захочуть його розвивати, доповнювати, або написати свій сервер зі своєю бізнес логікою на основі модулів мого сервера.

Abstract

The main idea of this work is to develop a distributed card game server to provide an example of writing simple distributed servers, as well as potentially for the future development of the client for the game. My main goal is to create a server that is distributed, consists of independently deployed modules, and such that it can be easily extended and developed both by modifying existing modules and by writing new modules.

To make the framework for add-ons as simple as possible, it will not have some of the features familiar to users of modern multiplayer games, but it will give more freedom to developers who want to develop it, extend it, or write their own server with its business logic based on my server's modules.

№ з/п	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№	Примітки
1	А4	ІАЛЦ.467100.001 ВП	Розподілений сервер	1		
			карткової гри			
			Відомість проекту			
2	А4	ІАЛЦ.467100.002 ТЗ	Розподілений сервер	3		
			карткової гри			
			Технічне Завдання			
3	А4	ІАЛЦ.467100.003 ПЗ	Розподілений сервер	61		
			карткової гри			
			Пояснювальна записка			
4	А4	ІАЛЦ.467100.004 Д1	Розподілений сервер	1		
			карткової гри			
			Принципова схема			
5	А4	ІАЛЦ.467100.005 Д2	Розподілений сервер	1		
			карткової гри			
			Функціональна схема			
6	А4	ІАЛЦ.467100.006 Д3	Розподілений сервер	1		
			карткової гри			
			Структурна схема			

					ІАЛЦ.467100.001 ВП		
Змн.	Арк.	№ документа	Підпис	Дата			
Розроб.		Федоряченко Я.П.			Розподілений сервер карткової гри Відомість проекту	Літ.	Аркуш 1 / Аркушів 1
Перев.		Шевело О.П.					
Реценз.						КПІ імені Ігоря Сікорського, ФІОТ, Група ІВ - 71	
Н. Контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г.					

Технічне завдання
до дипломного проєкту
на тему: «Розподілений сервер карткової гри»

Київ – 2021 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Це технічне завдання поширюється на розробку розподіленого сервера карткової гри. Область застосування: робочій приклад для написання сучасних розподілених серверів, а також основа для написання клієнта гри.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту по освітньо-професійної програми “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка розподіленого сервера карткової гри.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література по комп’ютерних мережах, публікації в періодичних виданнях, довідники по платформах дистанційного навчання, публікації в Інтернеті з цих питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого серверу

- Розподіленість.
- Можливість незалежного розгортання модулів.
- Можливість доповнення функціоналу за рахунок нових модулів або доповнення існуючих.

5.2. Вимоги до програмного забезпечення

- Операційна система на основі GNU/Linux
- Компілятор мови C++ (наприклад g++)
- Система налаштування збору проектів CMake
- Пакетний менеджер бібліотек C++ vcpkg
- Бібліотеки RapidJSON, gRPC, Boost.Beast, Boost.Asio
- Середовище запуску контейнерів (наприклад Docker)
- Стек програмного забезпечення для використання Kubernetes

5.3. Вимоги до апаратної частини

- Комп'ютер на базі процесора Ryzen 3 та вище
- Оперативної пам’яті не менше 2 Гбайт
- Вільний простір диска не менше 2 Гбайт
- Підключення до Інтернету

6. ЕТАПИ РОЗРОБКИ

Дата

6.1 Вивчення літератури

6.2 Складання і узгодження технічного завдання

6.3 Аналіз структури розробленого курсу навчання

6.4 Створення модулів розробленого курсу

6.5 Тестування дистанційного доступу до курсу

6.6 Відлагодження і виправлення помилок

6.7 Оформлення документації дипломного проєкту

Пояснювальна записка
до дипломного проєкту
на тему: «Розподілений сервер карткової гри»

Київ – 2021 року

ЗМІСТ

ВСТУП

В сучасному світі набувають популярності комп’ютерні ігри. Все більше людей звертають увагу на прості цікаві короткі сесійні багатокористувацькі ігри. Такі типи ігор найбільш цікаві, адже люди генерують унікальний ігровий досвід один для одного коли грають в багатокористувацькі ігри. Зараз як ніколи потрібна гра для, що буде працювати всюди і завжди. Щоб забезпечити такі критерії важливо зробити сервер гри розподіленим. Розподілений сервер можна завжди розширити, якщо збільшується кількість клієнтів, а також відкрити нові точки в різних кутках світу для покращення затримки й відповідно покращення якості сервісу, що надає гра. Проте зараз, на жаль, нема готових рішень з відкритим кодом для розподіленого сервера, адже технології написання розподіленого сервера зазвичай корисні лише в рамках великих проектів, якими займаються великі компанії, і які складно написати власноруч. Тому люди зазвичай пишуть сервер без задуму на майбутнє (коли/якщо їм знадобиться надавати сервіс більшій кількості клієнтів).

Мета даної роботи написати простий розподілений сервер гри, на прикладі якого можна буде потім писати інші сервери.

Гру для демонстрації сервера необхідно обрати просту, цікаву, але й з глибиною ігрового процесу. Такою грою можна назвати карткову гру Hanamikoji. На прикладі правил цієї гри й буде написаний сервер.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Загальні положення

Сервер гри це умовне поняття.

Сервери відрізняються за багатьма факторами, а також саме слово сервер може означати декілька різних речей.

Для користувача сервер це абстрактна чорна скриня, що дозволяє грати з друзями.

Для розробників, замовників, адміністраторів сервер може значити

• конкретну машину (комп'ютер),

• абстрактну роль в архітектурі програмного комплексу (наприклад, коли кажуть що авторизація повинна працювати на боці сервера),

• програму, котра власне забезпечує комунікацію між гравцями та/або виконує інші функції.

Під розподіленим сервером розуміють програмний або програмно-апаратний комплекс, що розроблений із ціллю виконуватися на більш ніж одній машині.

Розподілення може буди на:

• окремі незалежні частини, що спілкуються між собою для виконання кожною своїх функцій (microservices),

• однакові модулі, які можуть виконувати весь спектр функцій, але обслуговують лише якусь частку клієнтів, і мають можливість синхронізації (content distributing networks).

Також раніше для серверів популярних ігор використовувався метод створення одного великого сервера, що після досягнення лімітів залишався незмінним, а для нових гравців відкривався новий такий самий сервер, що не має зв'язку з першим.

Але й підхід мікросервісів не є надзвичайно новим і також змінювався з часом

У першому поколінні мікросервісів (див. Рис 1.1 a) окремі послуги упаковувались за допомогою легких контейнерних технологій, таких як LXC. Кожна служба повинна була знати місця розташування інших служб, які використовувались за певними протоколами зв'язку. Будь-який механізм обробки несправностей був реалізований безпосередньо у вихідному коді служб. Оскільки кількість служб на додаток зростала, а потреба у розгортанні та передислокації служб у різних середовищах виконання все частішала, знаходження відповідних екземплярів служби для виклику стало величезною проблемою. Крім того, оскільки нові послуги впроваджувались з використанням різних мов програмування, повторне використання наявного коду виявлення та обробки несправностей ставало все складнішим.

Для вирішення деяких з цих питань друге покоління (див. Рис. 1.1 b) запровадило послуги виявлення та багаторазові бібліотеки зв'язку, що не допускають відмов. Служби використовували загальну службу виявлення, таку як Консул, для реєстрації своїх функціональних можливостей. Потім клієнтські служби могли динамічно виявляти та викликати ці функціональні можливості без явного посилання на місце розташування викликаних служб.

Нова стратегія спростила впровадження та тестування послуг, дозволила повторно використовувати шаблонний код зв'язку між службами, однак відтоді розробники часто були змушені впроваджувати нові служби, використовуючи лише мови, для яких вже були доступні бібліотеки.

Отже, розробники більше не отримували всі переваги мікросервісів, особливо щодо передбачуваної самостійності розробників вибирати будь-яку мову програмування або технологію розробки, яку вони вважали найбільш доцільною для задоволення конкретних потреб.

У відповідь на новий виклик у третьому поколінні (див. Рис. 1.1 c) з'являються стандартні проксі-сервіси або бічні візки (sidecars), такі як Envoy, як прозорі проміжні сервісні послуги.

Ідея полягала в подальшому поліпшенні багаторазового використання програмного забезпечення, оскільки бічні візки інкапсулюють всі функції виявлення послуг та зв'язку. Ця стратегія негайно принесла всі переваги існуючих відмовостійких бібліотек зв'язку до будь-якої нової мови програмування, тим самим також збільшивши автономність розробки. При використанні в якості посередників мережі бічні вагони стають природним місцем для моніторингу поведінки всі взаємодії служб у програмі мікросервісу Тому з’явилися інструменти, що розширюють поняття автономних бічних візків, щоб забезпечити більш інтегроване рішення для комунікації послуг. Оператори додатків можуть динамічно контролювати та керувати поведінкою декількох розподілених бічних візків за допомогою централізованої площини управління.

Четверте покоління (див. Рис. 1.1 d) змінило підхід до парадигми. Нова ідея полягає у використанні сучасних технологій FaaS для подальшого спрощення розробки та надання мікросервісів. Завдяки цій так званій “безсерверній” (serverless) архітектурі програми мікросервісів по суті перетворюються на колекції “віддалених” функцій, кожну з яких можна створювати, оновлювати, замінювати та видаляти так швидко і довільно, наскільки це необхідно.

Один з цікавих аспектів нового підходу полягає в тому, що технологій попередніх поколінь (як бічні візки та сервісні сітки) можуть в майбутньому стати не потрібні розробникам. Існуючі платформи FaaS ще не забезпечують усіх функцій зв'язку та управління трафіком, які надають ці дві технології. Однак, можна уявити сценарій, в якому функції, подібні до бічних колясок, створюються для проміжного проміжку всіх взаємодій між функціями в безсерверному додатку (див. Рис. 1.1 d). Потім функція площини управління вищого рівня могла б контролювати та керувати поведінкою цих функцій бічних колясок, утворюючи новий вид сітки послуг (або функцій).[13]

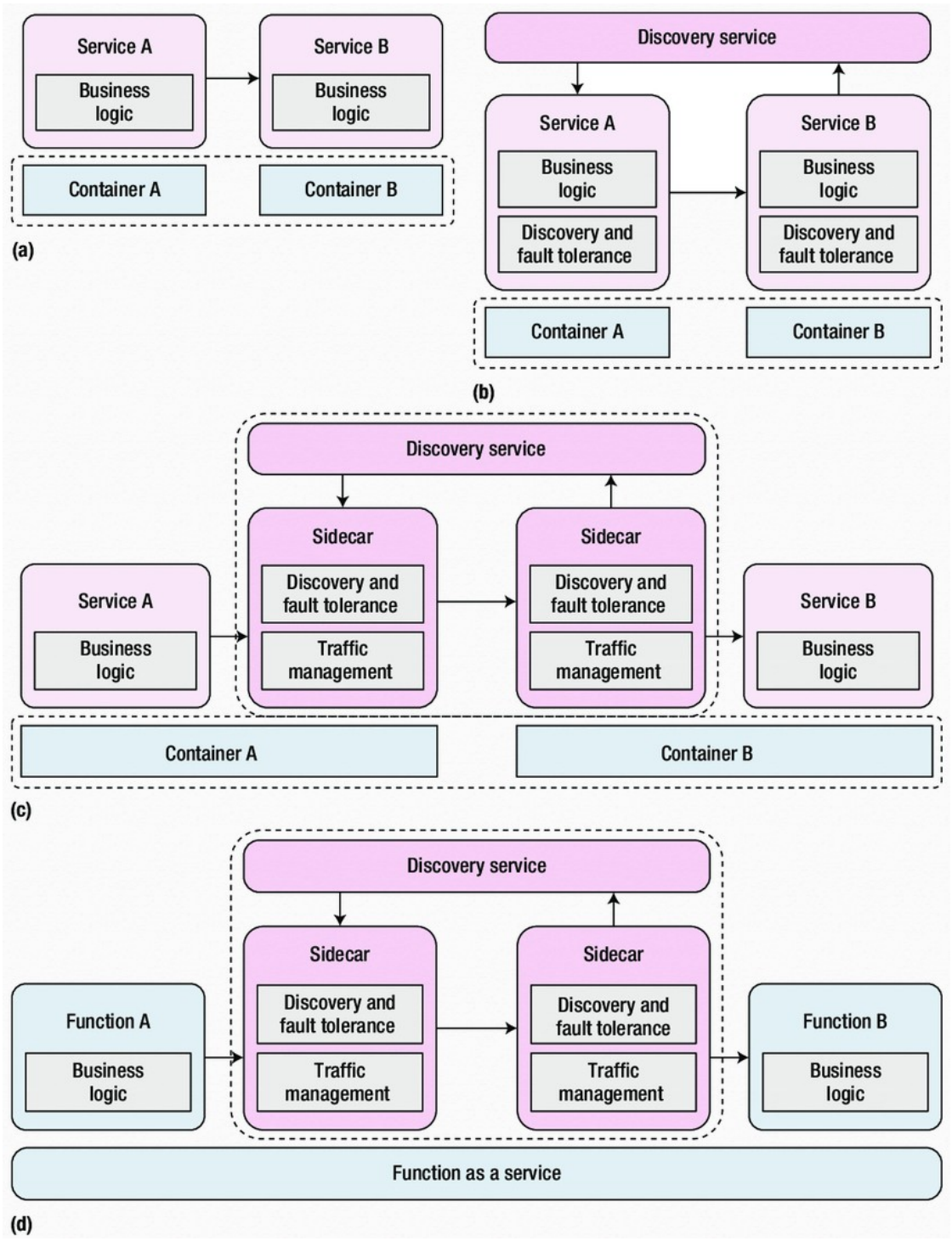


Рис 1.1 Чотири покоління архітектури мікросервісів

Окрім вищезазначених відмінностей сервери ігор також різняться процесами, за які відповідає сервер (server responsibilities).

Спільною відповідальністю для всіх (або майже всіх) серверів ігор на декілька гравців є налагоджування зв'язку між гравцями.

Усі інші відповідальності є, звісно, важливими чи навіть критично важливими, проте не обов’язковими. Вони включають у себе наприклад:

- забезпечення надійності зв’язку TODO: find references for as many points as you can
- забезпечення цілісності зв’язку
- авторизацію/аутентифікацію
- захист ігрового процесу від шахраїв (у тому числі проведенням усього процесу на сервері, виключаючи таким чином можливість втручання стороннього програмного забезпечення на боці клієнта) [11]
- пошук опонентів
- збереження поточного стану гри для продовження через деякий час
- забезпечення можливості повторного підключення гравців при втраті підключення до інтернету
- налагоджування CDN для зменшення затримки сигналу

В окремих випадках сервер гри також може забезпечувати:

- рендер деяких графічних частин [22]
- оновлення гри непомітно для гравців (без необхідності зупинки гри для гравця, чи припинення функціювання сервера на час оновлення), що досягається в тому числі за рахунок вище зазначеної CDN

1.2 Приклади існуючих рішень

Hearthstone — колекційна карткова онлайн-гра, розроблена компанією Blizzard Entertainment. Внутрішня структура серверів власне гри не розголошувалася компанією, проте відомо, що в інфраструктурі компанії існує чіткий поділ серверів (див. Рис. 1.2) на регіони не пов'язані між собою.

Розробники заявляли, що саме це є основною перепоною на шляху до сервера розподіленого по всьому світу, тому на даний момент нема планів об'єднання регіонів або введення повноцінного доступу до єдиної системи з будь-якого куточку світу.[8]

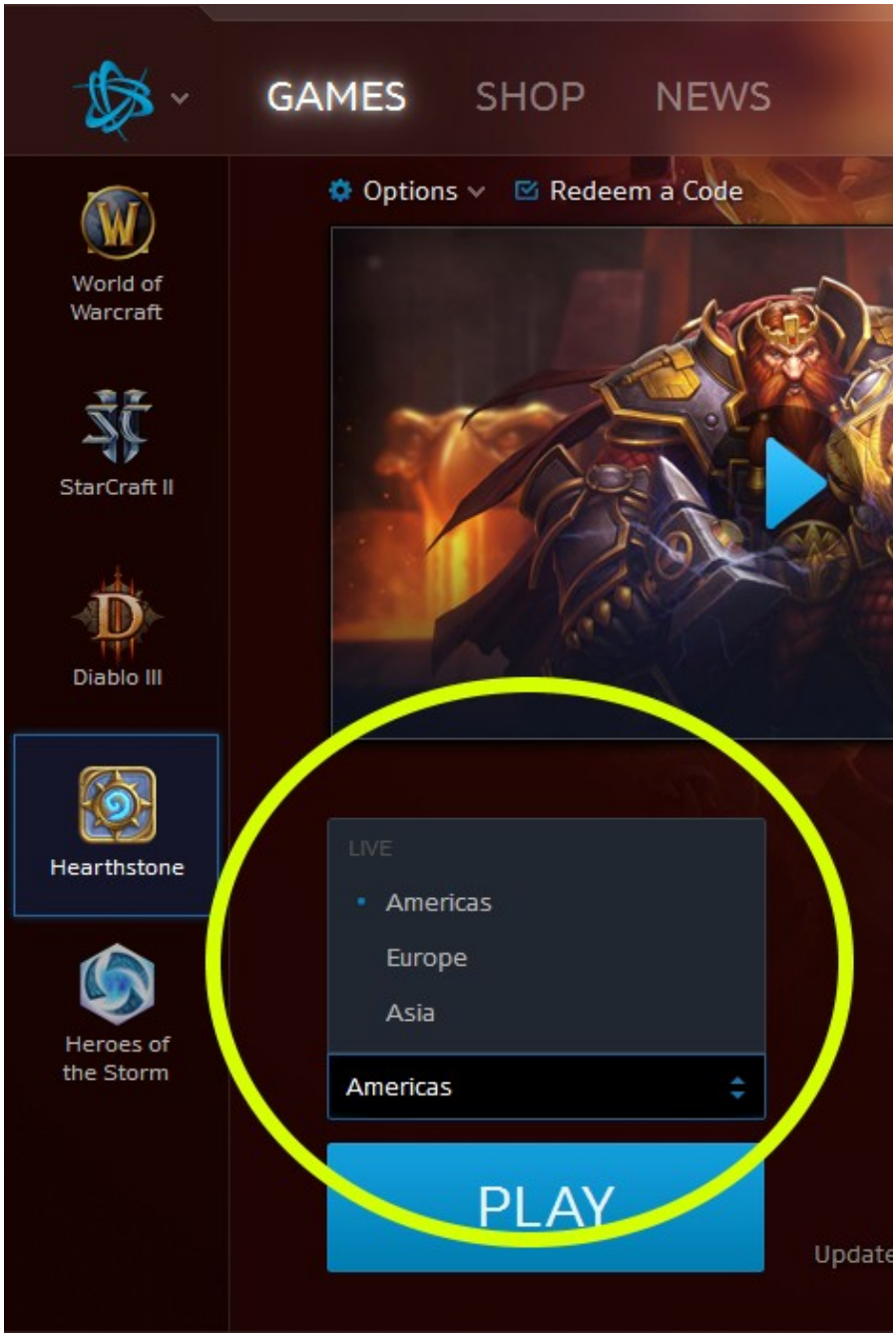


Рис. 1.2. Вхід в один з регіонів серверів Hearthstone

Із зазначених у пункті 1.1 відповідальностей сервера сервери Hearthstone відповідають за:

- забезпечення надійності зв’язку
- забезпечення цілісності зв’язку
- авторизацію/аутентифікацію
- захист ігрового процесу від шахраїв
- пошук опонентів
- забезпечення можливості повторного підключення гравців при втраті підключення до інтернету.

На відміну від Hearthstone, більшість сучасних ігор не обговорюють архітектуру своїх серверів. В першу чергу це пов'язано з тим, що вони вже підтримують усе що тільки міг придумати гравець.

В таких іграх як GWENT, Magic the gathering Arena не підіймають питання розподілу на регіони, бо уся серверна інфраструктура вже пов'язана між собою й не стає перепоною для гравців.

Усі популярні ігри мають реалізувати практично всі вищезазначені функції, щоб залишатися конкурентноспроможними на ринку, або ж вийти на цей ринок з новою грою.

Проте окрім великих проектів існують open source проекти, сервер яких можна роздивитися в деталях. Наприклад гра GameOfCards. [21]

На відміну від вже розглянутих рішень та концептів ця гра пропонує гравцям самим виступати в ролі сервера. Окрім цього власне гра не надає своїх правил, а дозволяє гравцям створювати свої правила для гри в карти. Таку концепцію сервера хоч і можна назвати розподіленою, проте це не є цілісною взаємопов'язаною системою. Ця гра пропонує одному з гравців виконувати роль сервера, проте сама не надає виділений сервер, що суперечить меті цієї роботи.

1.3 Порівняння рішень

У підсумку порівняємо згадані рішення між собою в таблиці за вищезгаданими критеріями (див. Таблицю 1.1).

Таблиця 1.1 - Порівняння серверних рішень згаданих ігор

	Hearthstone	GWENT	Magic the gathering Arena	GameOfCards	Моє рішення
налагоджування зв'язку між гравцями	+	+	+	+	+
забезпечення надійності зв'язку	-	-	-	-	-
забезпечення цілісності зв'язку	+	+	+	+	+
авторизація/ аутентифікація	+	+	+	-	+
захист ігрового процесу від шахраїв	+	+	+	+	+
пошук опонентів	+	+	+	-	-
збереження поточного стану гри для продовження через деякий час	-	-	-	+	-
забезпечення можливості повторного підключення гравців при втраті підключення до інтернету	+	+	+	+	-
налагоджування CDN для зменшення затримки сигналу	+	+	+	-	-
рендер деяких графічних частин	-	-	-	-	-
оновлення гри непомітно для гравців	-	-	-	-	-

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі було розглянуто різноманітні рішення серверів карткових ігор, що використовуються в даний момент. За відсутності будь-якого рішення, що хоч частково задовольняло б мету даної роботи, було вирішено розробляти повністю своє рішення розподіленого сервера. Розроблене рішення не матиме більшості функцій, які мають сучасні ігри, проте написання власне гри не є самоціллю роботи. Основною ціллю є розподілений сервер з відкритим кодом та зі слабко залежними модулями, що можна буде легко масштабувати й доповнювати згаданими функціями з часом.

РОЗДІЛ 2 ПІДБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ЗАДАЧІ

2.1 Загальні положення

В першу чергу необхідно визначитися з операційною системою під яку буде писатися сервер, або декілька операційних систем, якщо застосунок має бути платформонезалежним. В наш час для нових серверних застосунків все частіше використовують операційну систему Linux. Для розподілених серверних рішень ця операційна система є майже безальтернативним публічно доступним варіантом. Ми не будемо вигадувати велосипед й оберемо її для задачі.

Будь-якою Тьюрінг-повною мовою програмування можна написати сервер, проте в кожної мови є свої позитивні та негативні сторони, окрім цього необхідно зважати на наявність бібліотек та/або фреймворків для написання серверу, а також можливість написання на цих фреймворках саме розподіленого серверу.

Спершу оцінимо ці позитивні сторони з урахуванням специфіки задачі для таких популярних мов програмування (фреймворків):

- Java (Spring)
- Javascript (NodeJS)
- Python (Django)
- C# (ASP.NET Core)
- PHP
- Ruby (Ruby on Rails)
- Go
- C++

Потім оберемо бібліотеки та фреймворки.

Наостанок оберемо необхідні засоби для реалізації розподіленості сервера за допомогою технологій контейнеризації застосунків.

2.2 Мова програмування (фреймворк)

2.2.1 Java (Spring)

Власне Java переважно використовується в Enterprise рішеннях й підлаштована під це. Тому для нас це мінус.

Переваги фреймворка Spring:

Легка вага: Spring - це легкий каркас завдяки реалізації POJO. Це не змушує програміста успадкувати будь-який клас і реалізувати будь-який інтерфейс.

Потужна абстракція: вона забезпечує потужну абстракцію до специфікацій JEE, таких як JMS, JDBC, JPA та JTA.

Портативність.

Безпека: Він уважно стежить за залежностями сторонніх розробників. Випускається регулярне оновлення, яке робить наші дані та програми безпечними.

Недоліки:

Складність: робота з Spring є більш складною у порівнянні з більшістю фреймворків й вимагає досвіду. Крива навчання теж важка, тому, якщо у вас недостатньо досвіду розробки, навчитися важко.

Надає декілька варіантів розробникам. Ці параметри створюють плутанину для розробників щодо того, яку функцію використовувати, а яку ні, а неправильні рішення можуть призвести до значних затримок.

Ніяких специфічних стандартів коду: Це не стосується XSS або міжсайтових сценаріїв. Маючи це на увазі, нам потрібно з’ясувати способи, як зупинити хакерів від проникнення у вашу програму самостійно.

Крута крива навчання: Якщо у вас немає досвіду розвитку в цій галузі, навчитися було б досить важко. Це важко через нові методи програмування.

Багато XML: Розробка програми Spring вимагає багато XML. [14]

Приклади продуктів на Java:

- Google+
- Minecraft,
- amazon.com,
- ebay.com,
- PayPal,
- Офіційний сайт Пентагону.

2.2.2 Javascript (NodeJS)

Переваги NodeJS:

Одна мова для backend та frontend. Використовуючи Node.js для бекенду, ви автоматично отримуєте всі плюси розробки повного стека JavaScript, такі як: краща ефективність та загальна продуктивність розробників, обмін та повторне використання коду, легкий обмін знаннями.

Величезна екосистема безкоштовних інструментів розробки, що доступні в пакетному менеджері npm.

Швидка обробка та модель подій. Швидкість досягається за рахунок ефективного двигуна V8, написаного на C++, а також завдяки асинхронним не блокуючим операціям вводу/виводу та обробка запитів.

Масштабована технологія для мікросервісів. Node.js - це вибір номер один при створенні та впровадженні екосистемних рішень мікропослуг, згідно з Node.js User Survey Report 2017.

Краща вбудована підтримка JSON.

Недоліки:

Проблеми зі складними обчислювальними задачами, що закладені безпосередньо в дизайні NodeJS. Якщо необхідно використовувати багатопотокові обчислення з великим навантаженням на процесор, то NodeJS не задовольнить ваші потреби.

Незрілість більшості node модулів. Node модулів (бібліотек, фреймворків) дуже багато і через це дуже часто зустрічаються такі, що працюють значно повільніше ніж могли б через брак оптимізації. Окрім проблем продуктивності це призводить до величезних об’ємів коду (до 100ГБ), що може бути недопустимим наприклад для пристроїв інтернету речей. [23]

Приклади продуктів та компаній, що використовують NodeJS:

- Netflix
- Trello
- PayPal
- LinkedIn
- Walmart
- Uber
- NASA

2.2.3 Python (Django)

Переваги:

Python простий для вивчення.

Python як і NodeJS має велику екосистему бібліотек з відкритим кодом.

Django прискорює розробку великих проектів. Уся його функціональність спрямована на те, щоб розробники розгортали великі проекти швидше і з меншою кількістю помилок. Хоча Django монолітний, але він забезпечує просте розбиття задачі на модулі, що можут розроблятися незалежно.

Django зосереджується на безпеці. Django вийшов з уже вбудованими функціями безпеки.

Django забезпечує легке створення API.

Django спрощує процес розробки проектів пов’язаних з машинним навчанням завдяки вбудованим обчислювальним і математичним та статистичним можливостям.

Django забезпечує простуй й надійну роботу з базами даних

Недоліки:

Крута крива навчання. Незважаючи на простоту Python, розробники кажуть, що Django досить складно освоїти. Хоча Django було створено для веб-проектів Python, знання Python не є синонімом Django.

Django не підходить для невеликих проектів

Django монолітний.

Python це одна з більш повільних мов програмування, тому Django також не показує гарних результатів в цьому плані. [1]

Приклади продуктів на Django:

- YouTube
- Instagram
- Spotify
- Bitbucket
- DropBox
- Сайт служби підтримки Mozilla та сайт аддонів для Firefox
- Pinterest

2.2.4 C# (ASP.NET Core)

Як і Java, C# орієнтується на Enterprise ринок, таким чином фреймоворк .NET Core і його частина ASP.NET Core також орієнтуються на Enterprise.

Переваги:

Платформонезалежність. Можна запускати код на будь-якій операційній системі з підтримкою .NET Core

Швидка й проста інтеграція з продуктами Microsoft. Через те що фреймворк розробляють й використовують в Microsoft він отримує більше інтеграції з продуктами компанії.

Підтримка Web API. Web API є частиною фреймворку і може бути легко інтегрований із Swagger. Під час створення додатків API бажано задокументувати їх. Це спрощує роботу з вашим продуктом для інших розробників. Swagger можна легко інтегрувати, а документація автоматично генерується кодом.

Масштабування та докеризація. Додаток можна запустити в Docker. Це спрощує контейнеризацію, масштабування та створення інфраструктури для архітектури мікросервісу.

Недоліки:

Крута крива навчання.

Незважаючи на платформонезалежність розробка все ж залежить від продуктів Microsoft, адже компанія в першу чергу забезпечує підтримку засобів розробки власного виробництва.

Орієнтація в першу чергу на великі проекти. [4]

Приклади продуктів на ASP.NET:

- StackOverflow
- Сайти Microsoft
- Сайт компанії DELL
- Сайт TacoBell

2.2.5 PHP

Переваги:

Вбудований модуль для підключення до бази даних.

Велика кількість вже написаних безкоштовних бібліотек під більшість необхідних задач.

Стабільність. PHP існує близько 22 років. За ці роки було виявлено багато помилок, і команда розробників швидко їх виправила.

Легкість у використанні. На PHP легко розпочати програмувати навіть новачку в IT. Також для людей, які вже знають, як програмувати на мові програмування C, дуже просто почати користуватися цією мовою, виходячи зі своїх знань на мові C.

Недоліки:

Слабка типізація. Необізнані програмісти можуть бути здивовані неявним перетворенням типів в мові. Це може призвести до несподіваних помилок.

Не призначений для створення великих додатків. Оскільки мова програмування не є дуже модульною, величезні програми, створені з мови програмування, буде важко підтримувати.

Безпека. PHP сайти мають погану репутацію через те, що їх дуже часто зламували доволі нехитрими способами (як наприклад SQL ін’єкціями). Це напряму спричинено дизайном мови. Слабка типізація та простота отримання доступу до баз даних викликають помилкове почуття безпеки й довіру мові програмування в необачних програмістів, що вже спричинило шкоду як репутації програмістів PHP, так і самої мови програмування. [2]

Втрата популярності. Все менше людей хочуть стати PHP програмістами. Через це підтримка мови може постраждати в майбутньому.

Приклади продуктів, що працюють на PHP в 2021 році:

- Yahoo
- Facebook
- Wikipedia
- WordPress
- SourceForge
- Slack

2.2.6 Ruby (Ruby on Rails)

Переваги:

Простий у використанні.

Розвинута екосистема бібліотек зі своїм пакетним менеджером gem, що в середньому мають кращу якість коду за аналогічні пакети в інших мовах.

Безпека. Вбудовані механізми Ruby on Rails гарантують, що веб-сторінка або програма, розроблена з її допомогою, стійка до вразливостей та запобігає наступним загальним загрозам: Cross-Site Scripting, SQL Injection, Cross-Site Request Forgery, Insecure Direct Object Reference or Forceful Browsing.

Вбудована підтримка інтеграцій з популярними фронтенд фреймворками.

Недоліки:

Швидкість та продуктивність програм. Ruby on Rails не робить фокус на швидкості виконання, що негативно впливає на роботу великих застосунків.

Відсутність гнучкості. Завдяки жорсткій залежності між компонентами та модулями, Ruby on Rails ідеально підходить для стандартних веб-програм. Однак, коли йдеться про додаток з якоюсь унікальною функціональністю, налаштування може бути складним завданням.

Крута крива навчання.

Низька популярність. Протягом останніх кількох років спостерігається падіння популярності Ruby on Rails. Поки що спільнота ruby не планує зменшувати підтримку й кількість розробників росте, проте це може швидко змінитися. [20]

Приклади продуктів та компаній, що використовують Ruby:

- Github
- Netflix
- Hulu
- Airbnb
- Soundcloud

2.2.7 Go

Переваги:

Швидкість. Go - це дуже швидка мова. Оскільки Go компілюється до машинного коду, він, природно, перевершує мови, які інтерпретуються або мають віртуальні середовища виконання. Програми Go також компілюються надзвичайно швидко, а отриманий двійковий файл дуже малий.

Легкий у вивченні. Синтаксис Go невеликий порівняно з іншими мовами, і його легко вивчити. Ви можете помістити більшу частину в голову, а це означає, що вам не потрібно витрачати багато часу на пошук речей.

Статична типізація. Go - це сильно, статично типізована мова.

Зручна стандартна бібліотека. Вона забезпечує зручні вбудовані функції для роботи з примітивними типами. Існують пакети, які дозволяють легко підняти веб-сервер, обробляти введення-виведення, працювати з криптографією та маніпулювати необробленими байтами. Серіалізація та десеріалізація JSON, що надаються стандартною бібліотекою, є простими.

Простіша модель паралельності. Хоча паралельне програмування ніколи не буває простим, Go робить його простішим, ніж в інших мовах.

Недоліки:

Неявна реалізація інтерфейсів структурами. Це допомагає зменшити зв’язність модулів, проте заважає легкому читанню коду.

Проблемна спільнота. Спільнота Go може не сприймати пропозиції. Спільнота Go також має огиду до веб-фреймворків.

Роздроблена система управління пакетами. В інших мовах є єдина система управління пакетами (наприклад gem, npm, pip), а в Go довгий час не було стабільного офіційного менеджера пакетів. Після багатьох років благання спільноти, проект Go нещодавно випустив godep. [15]

Компанії, що використовують Go:

- Google
- Uber
- Twitch
- Dropbox
- SoundCloud

2.2.8 C++

Переваги:

Платформонезалежність.

Підтримка багатьох парадигм програмування. C++ - мова програмування з багатьма парадигмами. Термін "парадигма" стосується стилю програмування. Він включає логіку, структуру та процедуру програми. Узагальнена, імперативна та об’єктно-орієнтована - це три парадигми C++. Окрім цього C++ забезпечує декілька особливостей з функціональної парадигми, наприклад, лямбда (анонімні) функції.

Маніпуляція на низькому рівні. Оскільки C++ тісно пов'язаний із C, яка є процедурною мовою, тісно пов'язаною з машинною мовою, C++ дозволяє маніпулювати даними низького рівня.

Управління пам’яттю. Окремо з низькорівневих маніпуляцій можна виділити управління пам’яттю. C++ надає програмісту повний контроль над управлінням пам'яттю. Це можна розглядати і як актив, і як зобов'язання, оскільки це збільшує відповідальність користувача за управління пам'яттю, а не за тим, щоб нею керував збирач сміття.

Сумісність з C. C++ майже повністю сумісний з C. Практично кожна безпроблемна програма C є допустимою програмою C++. Залежно від використовуваного компілятора, кожна програма C++ може працювати на файлі з розширенням “.c”.

Масштабованість. Масштабованість означає здатність програми масштабуватись. Це означає, що програма C++ може працювати як в малому, так і в великому масштабі даних. Ми також можемо створювати додатки, що вимагають великих ресурсів. [3]

Недоліки:

Складний у дуже великій програмі високого рівня.

Зазвичай використовується для конкретних платформ.

Для певної операційної системи або платформи, набір бібліотек зазвичай визначає платформу.

Коли C++ використовується для веб-додатків, складний і важкий для налагодження.

C++ не є безпечним, оскільки він не підтримує збір сміття, має вказівники, friend функції та глобальні змінні.

2.2.9 Висновки щодо мови програмування

Критерії:

1. Вбудована багатопоточність
2. Можливість створення й використання потоків
3. Підтримка мережевого програмування
4. Нативна підтримка json
5. Інтеграція з контейнерними технологіями
6. Низькорівневий доступ до пам’яті
7. Орієнтація на швидкість програм
8. Відкритий код компілятора/інтерпретатора

9. Власний пакетний менеджер
10. Існує популярний пакетний менеджер
11. Активно підтримується офіційними розробниками впродовж останнього року
12. Безпечний
13. Є досвід використання
14. Використовується для написання ігор

Таблиця 2.1 Порівняння мов програмування

	C++	Go	Ruby on rails	PHP	C# asp.net	Python Django	NodeJS	Java Spring
1	-	+	+	-	+	-	-	+
2	+	+	+	+	+	-	-	+
3	+	+	+	+	+	+	+	+
4	-	+	+	+	+	+	+	-
5	+	+	+	-	+	+	+	-
6	+	-	-	-	+	-	-	-
	+	+	-	-	-	-	-	-
8	+	+	+	+	+	+	+	+-
9	-	+	+	-	+	+	+	-
10	+	+	+	-	+	+	+	-
11	+	+	+	+	+	+	+	+
12	-	+	+	-	+	+	+	+
13	+	-	-	-	-	-	-	-
14	+	-	-	-	-	-	-	-

Розглянувши усі варіанти (див. Таблицю 2.1) найкраще для даної задачі найкраще підійде мова програмування C++, в перчу чергу через досвід використання, а також через платформонезалежність й використання в ігровій індустрії. До того ж C++ має велику кількість бібліотек на вибір.

2.3 Вибір бібліотек

2.3.1 Мережева бібліотека

- Boost.Asio
- Boost.Beast
- ACE
- POCO
- grpc
- GameNetworkingSockets
- RCF
- facil.io
- ZeroMQ

Висунемо основні критерії вибору:

- Безкоштовне використання
- Швидкість вивчення
- Підтримка багатопоточності
- Підтримка контейнеризації
- Універсальний та простий стандарт для пов’язування з клієнтом гри

Boost.Asio

Boost.Asio покликана вирішувати такі цілі:

- Платформонезалежність. Бібліотека повинна підтримувати низку часто використовуваних операційних систем та забезпечувати послідовну поведінку в цих операційних системах.
- Масштабованість. Бібліотека повинна сприяти розробці мережевих додатків, які масштабуються до тисяч одночасних з'єднань. Реалізація бібліотеки для кожної операційної системи повинна використовувати механізм, який найкраще забезпечує цю масштабованість.

- Ефективність. Бібліотека повинна підтримувати такі методи, як розподілені операції вводу/виводу (scatter-gather), і дозволяти програмам мінімізувати копіювання даних.
- Модельні концепції з усталених API, таких як сокети BSD. API розетки BSD широко впроваджений і зрозумілий, і він висвітлений у великій кількості літератури. Інші мови програмування часто використовують подібний інтерфейс для мережевих API. Наскільки це можливо, Boost.Asio намагається використовувати існуючу практику.
- Простота використання. Бібліотека повинна забезпечити низький бар'єр для входу для нових користувачів, використовуючи підхід інструментарію, а не фреймворку. Тобто, йому слід намагатися мінімізувати попередні інвестиції вчасно, лише вивчивши кілька основних правил та рекомендацій. Після цього користувачеві бібліотеки слід лише зрозуміти конкретні функції, які використовуються.
- Основа для подальшої абстракції. Бібліотека повинна дозволяти розвиток інших бібліотек, які забезпечують вищий рівень абстракції. Наприклад, реалізації широко використовуваних протоколів, таких як HTTP. [5]

Boost.Beast

Beast - це бібліотека лише з заголовків на C ++, яка служить основою для написання взаємодіючих мережевих бібліотек, забезпечуючи низькорівневі типи і алгоритми словників протоколів HTTP / 1, WebSocket та мережевих протоколів, використовуючи послідовну асинхронну модель Boost.Asio.

Ця бібліотека призначена для:

✔ Симетрії: алгоритми є рольово-агностичними, тобто призначені для написання клієнтів, серверів або обох.

- ✔ Простота використання: користувачі Boost.Asio відразу зрозуміють Beast.
- ✔ Гнучкість: користувачі приймають такі важливі рішення, як управління буфером або потоком.
- ✔ Продуктивність: створюйте програми, що обробляють тисячі підключень або більше.
- ✔ Основа для подальшої абстракції. Компоненти добре підходять для нарощування. [6]

ACE

Переваги використання ACE:

- ✔ Підвищена портативність - компоненти ACE дозволяють легко писати паралельні мережеві програми на одній платформі ОС і швидко переносити їх на багато інших платформ ОС. Більше того, оскільки ACE - це безкоштовне програмне забезпечення з відкритим кодом, вам ніколи не доведеться турбуватися через блокування певної операційної системи або конфігурації компілятора.
- ✔ Підвищена якість програмного забезпечення - Компоненти ACE розроблені з використанням багатьох ключових моделей, що підвищують такі ключові якості, як гнучкість, розширюваність, багаторазовість та модульність комунікаційного програмного забезпечення.
- ✔ Підвищена ефективність і передбачуваність - ACE ретельно розроблена для підтримки широкого спектру вимог до якості обслуговування послуг (QoS), включаючи низьку затримку для програм, що чутливі до затримок, високу продуктивність для програм, що вимагають пропускної здатності, і передбачуваність для додатків у режимі реального часу.

- ✔ Більш легкий перехід на стандартне проміжне програмне забезпечення вищого рівня - ACE забезпечує багаторазові компоненти та шаблони, що використовуються в ACE ORB (TAO), що є реалізацією CORBA, що відповідає стандартам з відкритим кодом. [9]

POCO

- ✔ Крос-платформа. Потужні абстракції платформ дозволяють створювати крос-платформенний код, який працює на всіх загальноприйнятих настільних, серверних, мобільних та вбудованих платформах.
- ✔ Продуктивність. Написаний на ефективному сучасному C ++, POCO не витрачає дорогоцінні цикли процесора та пам’ять.
- ✔ Простий у використанні. Комплексні та послідовні API в поєднанні з легкодоступною базою коду роблять розробників C ++ більш продуктивними.
- ✔ Модульний та масштабований. Не платіть за те, чим не користуєтесь. Використовуйте на вбудованих пристроях Linux із 8-16 МБ оперативної пам'яті або на багатоядерних багатогігабайтних серверах.
- ✔ Багатопотоковість. Удосконалені багатопотокові абстракції спрощують розробку багатопотокових програм.
- ✔ Лісозаготівля. Універсальний, низький накладні витрати та розширювана система реєстрації для всіх ваших потреб ведення журналу.
- ✔ Доступ до бази даних. Доступ до баз даних SQL, таких як SQLite, MySQL / MariaDB, PostgreSQL та SQL Server (через ODBC). Або бази даних NoSQL, такі як MongoDB та Redis.
- ✔ JSON і XML. Кілька API (потокове та орієнтоване на документи) для аналізу та створення JSON та XML.

- ✔ Мережа та Інтернет Починаючи від базових сокетів, закінчуючи HTTP / HTTPS клієнтом та сервером, POCO покриває всі ваші потреби в мережевому програмуванні.
- ✔ Шифрування та безпека. Прості у використанні обгортки для OpenSSL спрощують інтеграцію шифрування та SSL / TLS у вашу програму. [19]

grpc

Переваги:

- ✔ Просте визначення послуги. Визначте свою послугу, використовуючи буфери протоколів, потужний інструмент двійкової серіалізації та мову
- ✔ Почніть швидко і масштабуйте. Встановлюйте середовища виконання та розробники одним рядком, а також масштабуйте до мільйонів RPC в секунду за допомогою фреймворку
- ✔ Працює на різних мовах та платформах. Автоматично генеруйте ідіоматичні заглушки клієнта та сервера для вашої служби різними мовами для різних платформам
- ✔ Двонаправлене потокове передавання та інтегрована аутентифікація. Двонаправлене потокове передавання та повністю інтегрована роз'ємна аутентифікація за допомогою транспорту на основі HTTP / 2
- ✔ Задуманий для використання в розподілених системах з архітектурою мікросервісів.
- ✔ Підтримка компанії Google гарантує, що бібліотека буде розвиватися й оновлюватися. [10]

GameNetworkingSockets

Особливості:

- ✔ API, орієнтований на підключення (наприклад, TCP)
- ✔ ... але орієнтований на повідомлення (як UDP), а не на потік.

- ✔ Підтримує як надійні, так і ненадійні типи повідомлень
- ✔ Повідомлення можуть бути більшими, ніж базові MTU. Протокол виконує фрагментацію, повторну збірку та повторну передачу для отримання надійних повідомлень.
- ✔ Рівень надійності значно витонченіший, ніж базове розсувне вікно в стилі TCP. Він заснований на моделі "векторного контролю" від DCCP (RFC 4340, розділ 11.4) та Google QUIC,. Основна ідея полягає в тому, щоб одержувач ефективно повідомляв відправнику статус кожного номера пакета (незалежно від того, отриманий пакет з цим номером чи ні). Запам'ятавши, які сегменти були надіслані в кожному пакеті, відправник може визначити, які сегменти потрібно повторно передавати.
- ✔ Шифрування. AES-GCM-256 за пакет, Curve25519 для обміну ключами та підписів сертифіката.
- ✔ Інструменти для моделювання затримки / втрати пакетів та детального вимірювання статистики
- ✔ Підтримка IPv6
- ✔ Однорангові мережі:
  - Обхід NAT через реалізацію ICE google WebRTC.
  - Підключіть власну службу сигналізації.
  - Унікальний режим "симетричне підключення".
  - ISteamNetworkingMessages - це інтерфейс, розроблений для спрощення портування коду на основі UDP до випадків використання P2P. [24]

RCF

Особливості:

- ✔ Односторонні та двосторонні повідомлення.

- ✔ Пакетний односторонній обмін повідомленнями.
- ✔ Публікація / передплата повідомлень у стилі.
- ✔ Багатоадресні та широкомовні повідомлення через UDP.
- ✔ Асинхронні віддалені дзвінки.
- ✔ Двонаправлені з'єднання для обміну повідомленнями між сервером і клієнтом.
- ✔ Кілька реалізацій транспорту (TCP, UDP, канали з іменами Windows та сокети локального домену UNIX).
- ✔ Тунелювання через HTTP та HTTPS.
- ✔ Стиснення транспортного рівня (zlib) та шифрування (Kerberos, NTLM, Schannel та OpenSSL).
- ✔ Підтримка IPv6.
- ✔ Вбудована система серіалізації.
- ✔ Вбудовані можливості передачі файлів.
- ✔ Надійна підтримка версій.
- ✔ Підтримка Protocol Buffers.
- ✔ Портативний у широкому діапазоні компіляторів, платформ та операційних систем.
- ✔ Масштабований доступ до широкого спектру програм, від простого parent-child IPC, до великих розподілених систем.
- ✔ Ефективна, з zero-copy та zero-heap алокаціях на критичних шляхах як на сервері, так і на клієнті.
- ✔ Ніяких залежностей. zlib, OpenSSL та Protocol Buffers необов’язкові.[7]

facil.io

facil.io - це мережна бібліотека, написана на мові C.

Особливості:

- ✔ Надає високопродуктивні мережеві послуги TCP / IP, використовуючи рівномірний дизайн, який був протестований для простого вирішення проблеми C10K.
- ✔ Включає міні-фреймворк для веб-додатків, із швидким сервером HTTP / WebSocket, інтегрованим Pub / Sub, додатковим підключенням Redis, простою обробкою JSON, візуалізацією шаблону Mustache та більш витонченими лакомічними шматочками.
- ✔ Легко використовувати в коді й він націлений на мінімізацію кривої навчання розробника.
- ✔ На додаток до детальної документації та прикладів, API уніфікований за стилем, а ті самі типи та API, що використовуються для HTTP-запитів, використовуються для рендерінгу JSON та Mustache - так що вчитися тут менше.
- ✔ Працює на Linux / BSD / macOS (і, можливо, CYGWIN), і постійно тестується як на Linux, так і на macOS.
- ✔ Підтримує як однопотокові, так і багатопотокові режими роботи, а також гібридний режим (багатопроцесорний з однопотоковими або багатопотоковими робітниками).
- ✔ Бібліотека має відкритий код, що полегшує включення в будь-який проект. API був розроблений для простоти та розширюваності, а це означає, що писати нові розширення та власні мережеві протоколи легко. [17]

ZeroMQ

Особливості:

- ✔ Універсальна. Підключіть свій код будь-якою мовою на будь-якій платформі.
- ✔ Реалізує шаблони pub-sub, push-pull, fan-out, task distribution.

- ✔ Висока швидкість. Асинхронні механізми вводу-виводу в крихітній бібліотеці.
- ✔ Мульти-транспорт. Передає повідомлення через inproc, IPC, TCP, UDP, TIPC, multicast та WebSocket
- ✔ Спільнота. За підтримки великої та активної спільноти з відкритим кодом.
- ✔ Документація. Пояснює, як використовувати ØMQ із 60+ діаграмами та 750 прикладами на 28 мовах [26]
- ✔ Використовується такими брендами як
    - Microsoft
    - Samsung
    - AT&T
    - Spotify
    - Facebook
    - Digital Ocean
    - Auth0
    - Bitcoin
    - Jupyter
    - Mongrel2
    - Jina

У підсумку маємо аж 4 непоганих кандидата для використання як основної мережевої бібліотеки:

- Boost.Beast пропонує WebSocket API на основі популярної Boost.Asio, знання якої стануть у нагоді в майбутньому
- gRPC прогресивна технологія від Google, що стрімко набирає популярності як для IPC так і для мережевої комунікації в першу чергу в розподілених мережевих системах

- ZeroMQ одна з найбільш розповсюджених event-based бібліотек, якою користуються відомі бренди для реалізації розподілених систем
- GameNetworkingSockets мережева бібліотека, що із самого початку розроблялася із задумом використовуватися для написання серверів ігор, й окрім цього має широкий спектр можливостей для комунікації.

Зважаючи на вбудовану підтримку багатопоточності в Boost.Beast за рахунок потужностей Boost.Asio оберемо її як основну бібліотеку. Для обміну даними між модулями програми будемо використовувати gRPC.

2.3.2 Бібліотека для роботи з JSON

Щоб обмінюватися даними з клієнтом необхідно подати їх в зручному форматі. Найкраще для цього підходить JSON.

Щоб передати дані в форматі JSON їх спочатку треба сереалізувати, окрім цього для отримання даних від клієнта в форматі JSON треба отриманий JSON десералізувати [12]. Для цього підберемо бібліотеку для роботи з JSON.

Оберемо найбільш продуктивну бібліотеку. Для цього подивимося на тести продуктивності різних бібліотек та фреймворків при сереалізації та десереалізації.

Нижче наведені результати дослідженя Native JSON Benchmark (див. Рис. 2.1, Рис. 2.2, Рис. 2.3, Рис. 2.4) [25]

З дослідження видно, що найбільш швидкою бібліотекою і для сереалізації, і для десереалізації є RapidJSON.

За показниками витрати пам’яті RapidJSON також лідує (через використання inline функцій майже для всього функціоналу бібліотеки).

Окрім швидкості RapidJSON також є header-only бібліотекою, що значно спрощує її інтеграцію в проект.

Зважаючи на це для роботи з JSON оберемо RapidJSON.

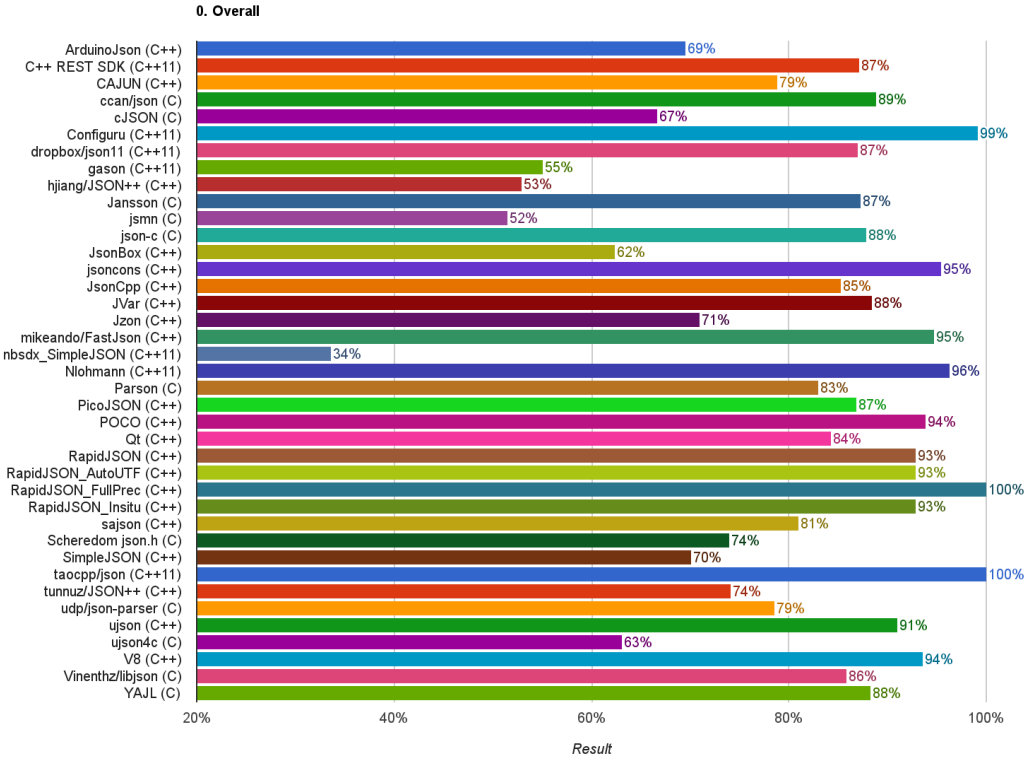


Рис. 2.1 Покриття стандартів (у тому числі RFC7159[12])

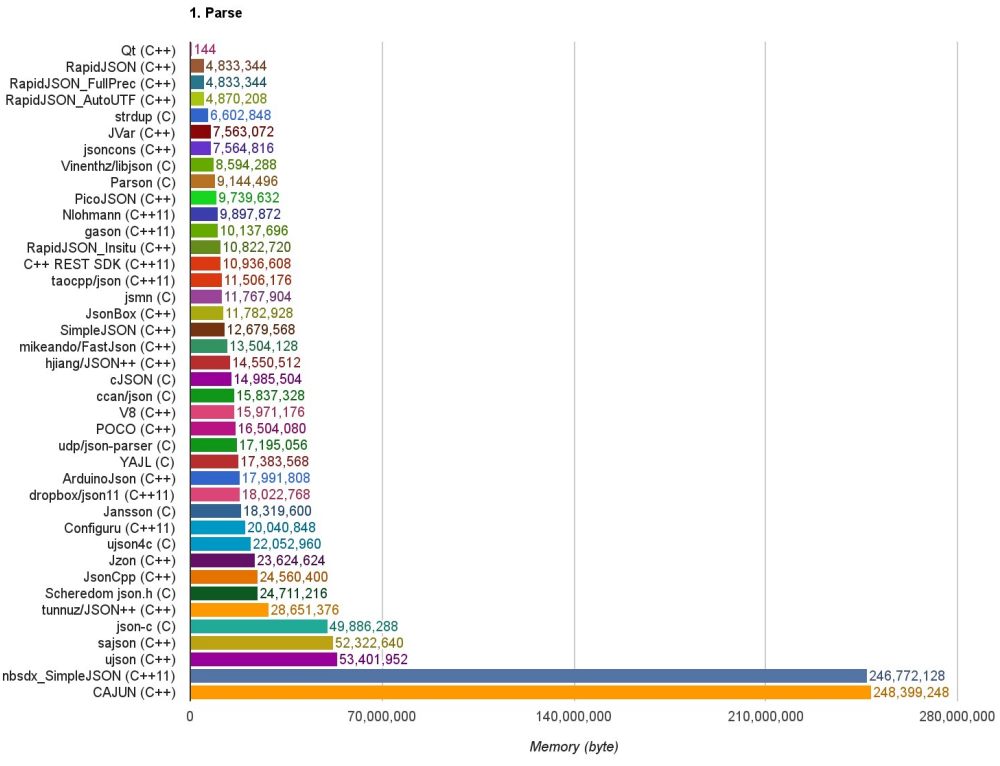


Рис. 2.2 Витрати на пам'ять під час парсингу (десереалізації)

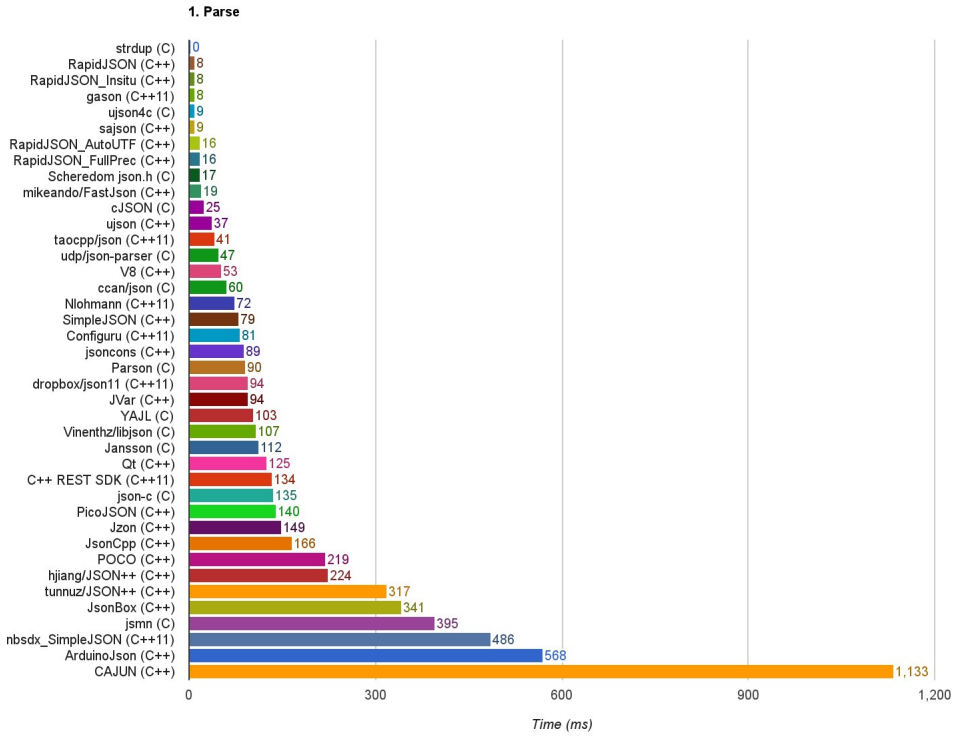


Рис. 2.3 Час парсингу (десереалізації)

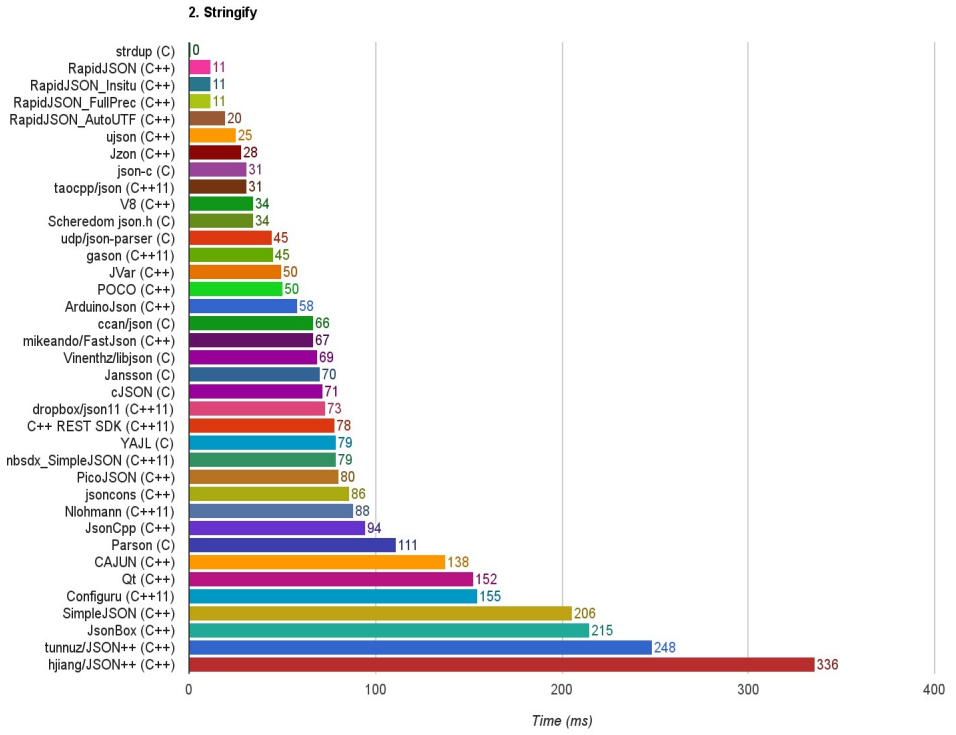


Рис. 2.4 Час сереалізації DOM to JSON

2.3.3 Контейнеризація та оркестрування

Для створення розподіленого сервера в сучасних системах використовують технології контейнеризації та оркестрування.

Для контейнеризації існує безліч засобів, але вони усі задовольняються єдині стандарт контейнерів, який визначає організація Open Container Initiative [18].

Окрім власне контейнеризації є ще декілька компонентів системи, наприклад середовище запуску контейнерів (container runtime environment), що знадобиться нам для оркестрації (див. Рис. 2.5).

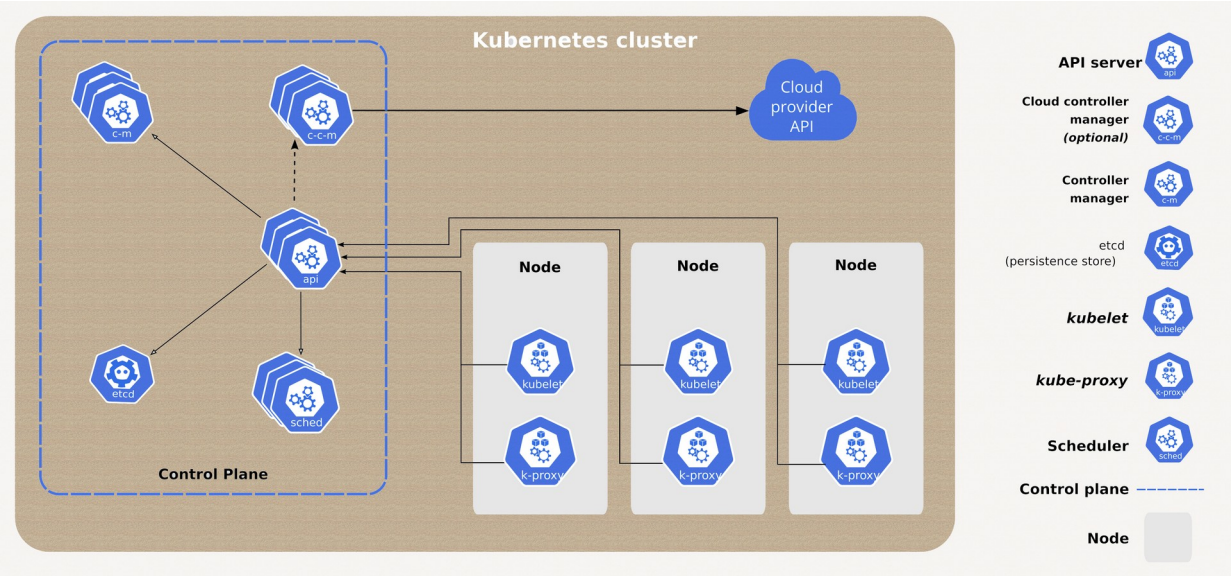


Рис. 2.5. Компоненти кластера оркестратора Kubernetes

Компоненти площини управління

Компоненти площини управління приймають глобальні рішення щодо кластера (наприклад, планування), а також виявляють і реагують на події кластера (наприклад, запускають новий блок, коли поле реплік розгортання не задовольняється).

Компоненти площини управління можна запускати на будь-якій машині кластера. Однак для простоти налаштовані сценарії зазвичай запускають усі компоненти площини управління на одній машині і не запускають на цій машині контейнери користувачів.

- kube-apiserver

  Компонент площини управління Kubernetes, який відкриває API Kubernetes. Сервер API - це інтерфейс для площини управління Kubernetes.
- kube-scheduler

  Компонент площини управління, який спостерігає за нещодавно створеними стручками без призначеного вузла та вибирає вузол для запуску.
- etcd

  Послідовне та доступне сховище значень ключів, що використовується як резервне сховище Kubernetes для всіх даних кластера.

  Якщо ваш кластер Kubernetes використовує etcd як резервне сховище, переконайтеся, що у вас є план резервного копіювання цих даних.
- kube-controller-manager

  Компонент Control Plane, який запускає процеси контролера.

  Логічно, що кожен контролер - це окремий процес, але для зменшення складності всі вони компілюються в один двійковий файл і запускаються в одному процесі.

  Деякі типи цих контролерів:
  - Контролер вузлів (Node controller): відповідає за контроль за роботою вузлів та оброку ситуацій, коли вузли “падають”.
  - Контролер завдань (Job controller): переглядає об’єкти завдань, що представляють одноразові завдання, а потім створює стручки для запуску цих завдань до завершення.
  - Контролер кінцевих точок (Endpoints controller): заповнює об’єкт Кінцеві точки (тобто приєднується до Служб і підрозділів).
  - Контролери службових облікових записів і маркерів (Service Account & Token controllers).

- cloud-controller-manager

  Компонент площини управління Kubernetes, який вбудовує специфічну для хмари логіку управління.

Компоненти вузла

Компоненти вузла працюють на кожному вузлі, підтримуючи запущені стручки та забезпечуючи середовище виконання Kubernetes.

- kubelet

  Агент, який працює на кожному вузлі кластера. Це гарантує, що контейнери працюють в Pod.

- kube-proxy

  kube-proxy - це мережевий проксі, який працює на кожному вузлі у вашому кластері, реалізуючи частину концепції служби Kubernetes.

- Container runtime

  Container runtime - це програмне забезпечення, яке відповідає за запуск контейнерів. [16]

Ми використовуватимемо Docker (або containerd, якщо Docker не буде задовольняти наші потреби) разом з Kubernetes для нашої системи.

ВИСНОВКИ ДО РОЗДІЛУ 2

У даному розділі були розглянуті варіанти технологій для реалізації проекту даної роботи. З поміж іншого були підібрані підходящі для вирішення задачі:

1. Мова програмування C++
2. Мережева бібліотека для асинхронного зв'язку з клієнтами по WebSocket Boost.Beast разом з потужним Boost.Asio
3. Швидка бібліотека для роботи з JSON RapidJSON
4. Бібліотека для зв'язків між модулями gRPC
5. Засіб для запуску контейнерів та контейнеризації Docker (або альтернатива conteinerd).
6. Оркестратор контейнерів для пов'язування контейнеризованих модулів в єдине середовище та контролю над модулями Kubernetes

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СЕРВЕРА

3.1 Загальна структура проекту

Проект складається з двох окремих модулів, що відповідають кожен за свій функціонал і пов’язані лише спільним grpc API (див. Рис. 3.1).

```
1  syntax = "proto3";
2
3  package hanaauth;
4
5  // GameSession creation service definition
6  service GameSession {
7      rpc CreateSession (NewSessionRequest) returns (NewSessionResponse) {}
8  }
9
10 message NewSessionRequest {
11     uint64 session_tag = 1;
12 }
13
14 message NewSessionResponse {
15     int32 port_p1 = 2;
16     int32 port_p2 = 3;
17     string auth_token_p1 = 4;
18     string auth_token_p2 = 5;
19 }
20
NORMAL  gamesession.proto  master
```

Рис. 3.1 - файл, що описує API зв’язку двох модулів проекту

Перший модуль представляє собою сервер аутентифікації. Він відповідає за:

- все що пов'язано з первинною аутентифікацією користувачів
- пошуком опонентів
- перенаправлення гравців на сервер з ігровою сесією
- виклик якогось із екземплярів модуля ігрової сесії (модуль аутентифікації виступаючи в якості grpc клієнта "викликає" віддалену

процедуру CreateSession (на Рисунку 3.1), для того щоб була створена ігрова сесія й дані для аутентифікації гравців на сервері ігрової сесії.

Другий модуль представляє собою сервер ігрової сесії. Він відповідає за:

- створення ігрових сесій
- підготовку до підключення гравців, що включає в себе
  - завчасне попарне відкриття серверних TCP сокетів для миттєвого підключення обох гравців
  - створення токенів аутентифікації для того, щоб не було неавторизованих підключень
- передачу даних для аутентифікації гравців ігрової сесії на сервер аутентифікації
- контроль за перебігом ігрової сесії

3.2 Файлова структура модуля аутентифікації

Файлова структура модуля показана на Рисунку 3.2.

У директорії Debug містяться файли, що генеруються при збиранні проекту. Вона отримала свою назву через те, що в ній проект збирається разом з символами для відлагоджування. Власне виконуваний файл модуля це AuthServer.

Директорія proto містить в собі файли, що описують grpc API. Наразі в ній лише файл для зв'язку з модулем ігрової сесії, проте архітектура сервера розроблялась із задумом, що можна буде додавати інші модулі, наприклад, модуль для роботи з базою даних, або модуль, що буде відстежувати стан системи й помічати підозрілих користувачів. Директорія отримала свою назву від мови опису API proto.

У директорії res знаходяться наразі лише JSON схеми для зв'язків з користувачем. Проте там також можна в майбутньому зберігати картинки, відео та інші матеріали, що можуть допомогти стороннім розробникам.

Рис. 3.2 Файлова структура модуля аутентифікації

У директорії src знаходяться усі вхідні файли модуля. Кожен файл за задумом містить один клас, що відповідає за свою частину функціоналу в модулі та, можливо, один чи декілька допоміжних класів та/або структур даних.

Файл CMakeLists.txt містить налаштування та інструкції генератора сценаріїв збору проектів CMake.

3.3 Файлова структура модуля ігрових сесій

Файлова структура модуля показана на Рисунку 3.3.

```
hanaserv/
| Debug/
| include/
| | DataBase.hpp
| | Deck.hpp
| | GameSession.hpp
| | Geisha.hpp
| | GrpcServer.hpp
| | Player.hpp
| | Queue.hpp
| | Server.hpp
| | WebSocket.hpp
| proto/
| | gamesession.proto
| res/
| | schema/
| | | choose.schema.json
| | | turn.schema.json
| src/
| | DataBase.cpp
| | Deck.cpp
| | GameSession.cpp
| | Geisha.cpp
| | GrpcServer.cpp
| | Player.cpp
| | Queue.cpp
| | Server.cpp
| | WebSocket.cpp
| | main.cpp
| CMakeLists.txt
```

Рис. 3.3 Файлова структура модуля ігрових сесій

У директорії Debug як і у випадку з модулем ауиентифікації містяться файли, що генеруються при збиранні проекту.

Директорія proto містить в собі файли, що описують grpc API. Наразі в ній лише файл для зв'язку з модулем аутентифікації, проте архітектура сервера розроблялась із задумом, що можна буде додавати інші модулі, тому тут можна в майбутньому додати API зв'язку з якимось іншим модулем. Задля підвищення незалежності модулів між собою було прийнято рішення не зберігати навіть спільний API в одному місці. Таким чином кожен модуль буде максимально незалежним від інших й матиме змогу розгортатися зі змінами без впливу на інші модулі. Окремо варто зазначити, що через зберігання копій в обох модулях змінювати щось в API, що вже використовується, можна лише шляхом додавання.

У директорії res знаходяться JSON схеми для зв'язків з користувачем. Проте там також можна в майбутньому зберігати інші матеріали, що можуть допомогти стороннім розробникам використовувати API модуля.

У директорії src знаходяться вхідні файли модуля з кодом програми. Кожен файл за задумом містить один клас, що відповідає за свою частину функціоналу в модулі та, можливо, один чи декілька допоміжних класів та/або структур даних.

У директорії include знаходяться заголовки файлів для взаємного використання одних класів іншими.

Файл CMakeLists.txt як і в першому модулі містить налаштування та інструкції генератора сценаріїв збору проектів CMake.

3.4 Архітектура модуля аутентифікації

Модуль аутентифікації складається з наступних компонентів:

- Основний клас, що пов'язує весь модуль - AuthServer (див. Рисунок 3.4). Його екземпляр створюється одноразово при запуску програми й містить в собі інформацію про усі активні з'єднання та сесії
- Клас що відповідає за роботу з TCP сокетами - listener. Його екземпляр

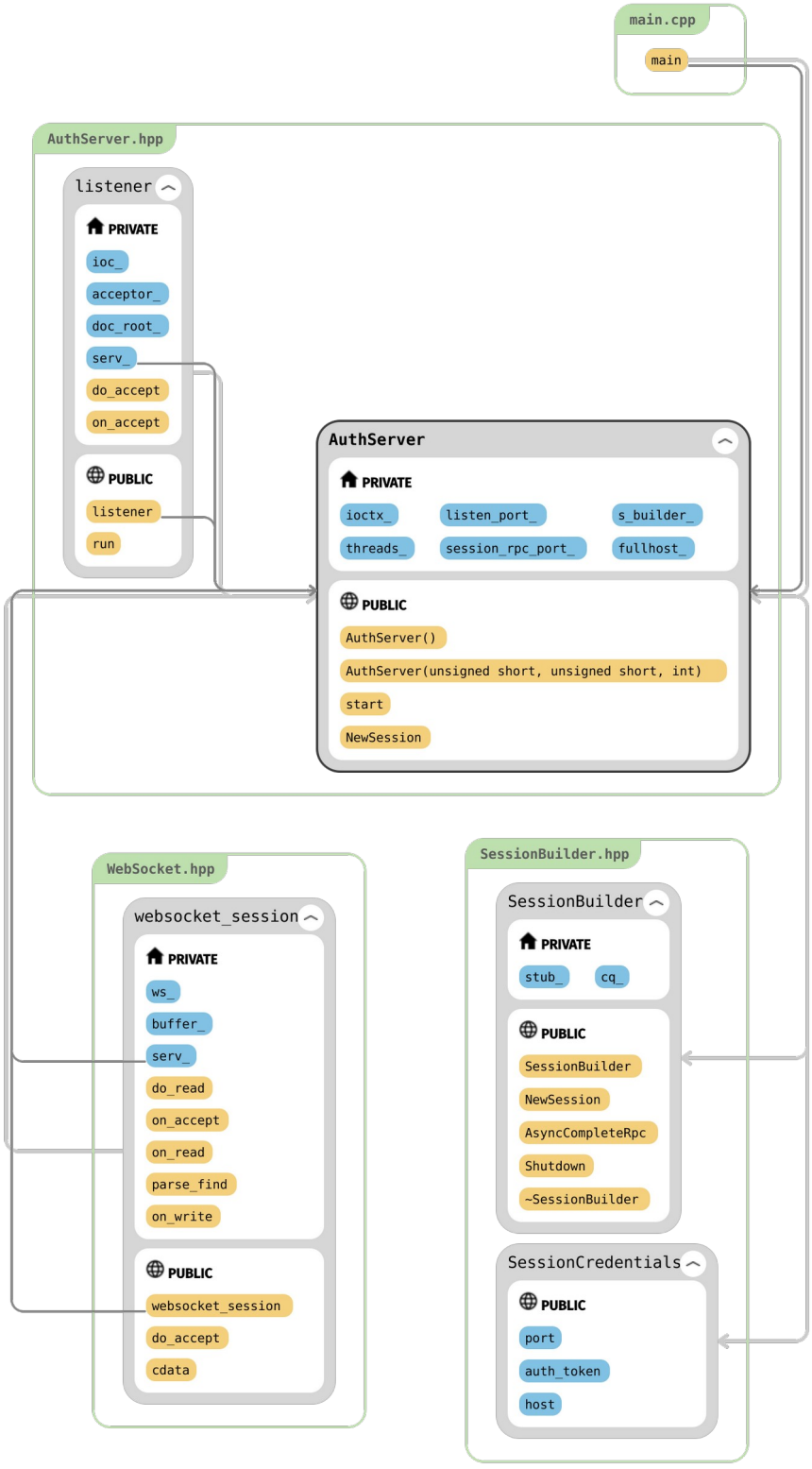


Рис. 3.4 Граф зв'язків основного класу AuthServer

створюється в режимі "на вимогу", тобто спочатку програма запустить

лише один екземпляр, але при надходженні з'єднань від клієнтів на порт будуть створені нові екземпляри. Так як сервер не оперує сирими TCP сокетами listener відразу намагається конвертувати з’єднання в WebSocket.

- Клас що відповідає за роботу з WebSocket - websocket_session. Екземпляри цього класу створюються окремо для кожного з’єднання в класі listener. Цей клас містить ті callback методи для зв’язку з користувачем, які не залежать від інших модулів та/або інших екземплярів класу. Якщо необхідна взаємодія, то викликаються методи інших класів.
- Клас що відповідає за роботу з технологією grpc - SessionBuilder. Цей клас як і AuthServer створюється одноразово. Його основні функції - викликати віддалену процедуру CreateSession з модуля ігрових сесій та відстежувати відповідь модуля (адже зв’язок через grpc для зменшення затримок та покращення паралельних властивостей працює асинхронно).

В модулі використовуються безліч програмних засобів для роботи з паралелізмом, а також для забезпечення асинхронності процесів.

- Boost.Beast (на основі Boost.Asio) надає простий механізм асинхронних callback функцій, що усі працюють в рамках єдиного пулу потоків, який можна легко налаштовувати під наявну систему (див. Рисунок 3.5 та Рисунок 3.6).
- Звичайне виконання функції в потоці для обробки відповідей на асинхронні grpc запити з допомогою механізму CompletionQueue, що дозволяє однаково обробляти відповіді на запити в одній функції в циклі, але при цьому робити це в багатьох потоках (тобто декілька потоків можуть запустити одну й ту саму функцію обробки відповідей).

```cpp
void websocket_session::on_read(beast::error_code ec,
                                std::size_t bytes_transferred)
{
    static net::const_buffer s_waiting_str(R"({"status":"WAITING OPONENT"})", 29);
    boost::ignore_unused(bytes_transferred);

    // This indicates that the websocket_session was closed
    if (ec == websocket::error::closed)
        return;

    if (ec)
        fail(ec, "read");

    ws_.text(ws_.got_text());
    const char *data = static_cast<char *>(buffer_.data().data());
    p_dbg("on_read", data);
    if (!check_auth_token()) {
        buffer_.consume(buffer_.size());
        ws_.async_write(
            s_waiting_str,
            beast::bind_front_handler(&HanaServer::on_game_waiting, &serv_,
                                      shared_from_this(), session_token_));
    } else {
        ws_.close("Wrong token");
    }
}
```

Рис. 3.5 Приклад callback функції без додаткових параметрів

```cpp
// Start the asynchronous accept operation
void websocket_session::do_accept()
{
    // Set suggested timeout settings for the websocket
    ws_.set_option(websocket::stream_base::timeout::suggested(
        beast::role_type::server));

    // Set a decorator to change the Server of the handshake
    ws_.set_option(websocket::stream_base::decorator(
        [](websocket::response_type &res) {
            res.set(http::field::server, "hanaserv");
        }));

    // Accept the websocket handshake
    ws_.async_accept(beast::bind_front_handler(
        &websocket_session::on_accept, shared_from_this()));
}

void websocket_session::do_read()
{
    buffer_.consume(buffer_.size());
    ws_.async_read(buffer_,
                   beast::bind_front_handler(&websocket_session::on_read,
                                             shared_from_this()));
}
```

Рис. 3.6 Приклади асинхронних викликів

- Механізм future-promise для одночасного асинхронного сповіщення двох користувачів про готовність ігрової сесії в яку вони зайшли, а також для передачі їм даних для входу в цю сесію.

3.5 Архітектура модуля ігрових сесій

Модуль аутентифікації складається з наступних компонентів:

- Клас, що пов'язує складові модуля непов'язані з ігровим процесом - HanaServer. Його екземпляр створюється у функції main одноразово при запуску й містить інформацію про активні з'єднання (користувачі, що вже підключилися та чекають на підключення свого опонента) та неактивні сесії, що створюються для якомога швидшого підключення двох гравців, адже при підключенні гравцям не потрібно очікувати на створення сесії, бо вона вже створена заздалегідь (див. Рис. 3.7).

```
SessionCredentials HanaServer::SpawnSession()
{
    auto const address = net::ip::make_address("0.0.0.0");
    auto const doc_root = std::make_shared<std::string>(".");
    auto session_token = new int;
    auto socket1 = std::make_shared<listener>(ioctx_, tcp::endpoint{address, 0},
                                              doc_root, *this,
                                              reinterpret_cast<uintptr_t>(session_token));
    auto socket2 = std::make_shared<listener>(ioctx_, tcp::endpoint{address, 0},
                                              doc_root, *this,
                                              reinterpret_cast<uintptr_t>(session_token));
    socket1->run();
    socket2->run();

    return SessionCredentials{socket1->get_port(),socket2->get_port(),"TOKEN1","TOKEN2"};
}
```

Рис. 3.7 Метод, що створює сесію

- Клас, що відповідає за TCP сокети - listener. Працює схожим чином з аналогічним класом з модуля аутентифікації, проте нові екземпляри створюються не поодинці при кожному підключенні, а парами після кожної сесії, що була виділена користувачам, й кожного разу на різні випадкові порти.
- Клас що відповідає за роботу з WebSocket - websocket_session. Екземпляри цього класу створюються так само як і в аналогічному

класі модуля аутентифікації. В даному випадку цей клас містить не лише callback методи, а й синхронні методи для читання й запису, адже під час нашої карткової гри нема необхідності в асинхронності через послідовну природу обраної карткової гри.

- Клас, що відповідає за роботу з grpc - HanaGrpcServer. Цей клас так само як і grpc клас з модуля аутентифікації працює асинхронно. На додачу до асинхронності клас реалізує свій функціонал у формі кінцевого автомата з декількома послідовними станами. Саме цей клас запускає створення нової сесії, коли виділяє й передає вже готову сесію користувачу через модуль аутентифікації.
- Клас, що власне реалізує логіку гри - GameSession. Цей клас містить методи, що безпосередньо керують перебігом гри й координує/використовує інші класи, що також пов'язані з ігровим процесом (див. Рис. 3.8). А саме класи:
  - Player - містить інформацію, про ігровий стан гравця (наприклад вміст його руки, сховану карту, тощо), а також надає функції обгортки для передачі й читання даних по WebSocket з'єднанню цього гравця.
  - Deck - містить інформацію про колоду карт й надає методи для простого використання контейнера колоди в рамках правил гри.
  - Geisha - містить інформацію про стан одного з основних ігрових об'єктів цієї картокової гри, а також надає зручні методи для використання й зміни цієї інформації.

Також варто згадати окремий клас обгортку Queue для зберігання WebSocket з'єднань, що очікують на підключення своїх опонентів.

Отже, система модуля ігрових сесій в багатьох моментах умисно збігається з модулем аутентифікації для кращого розуміння та простоти

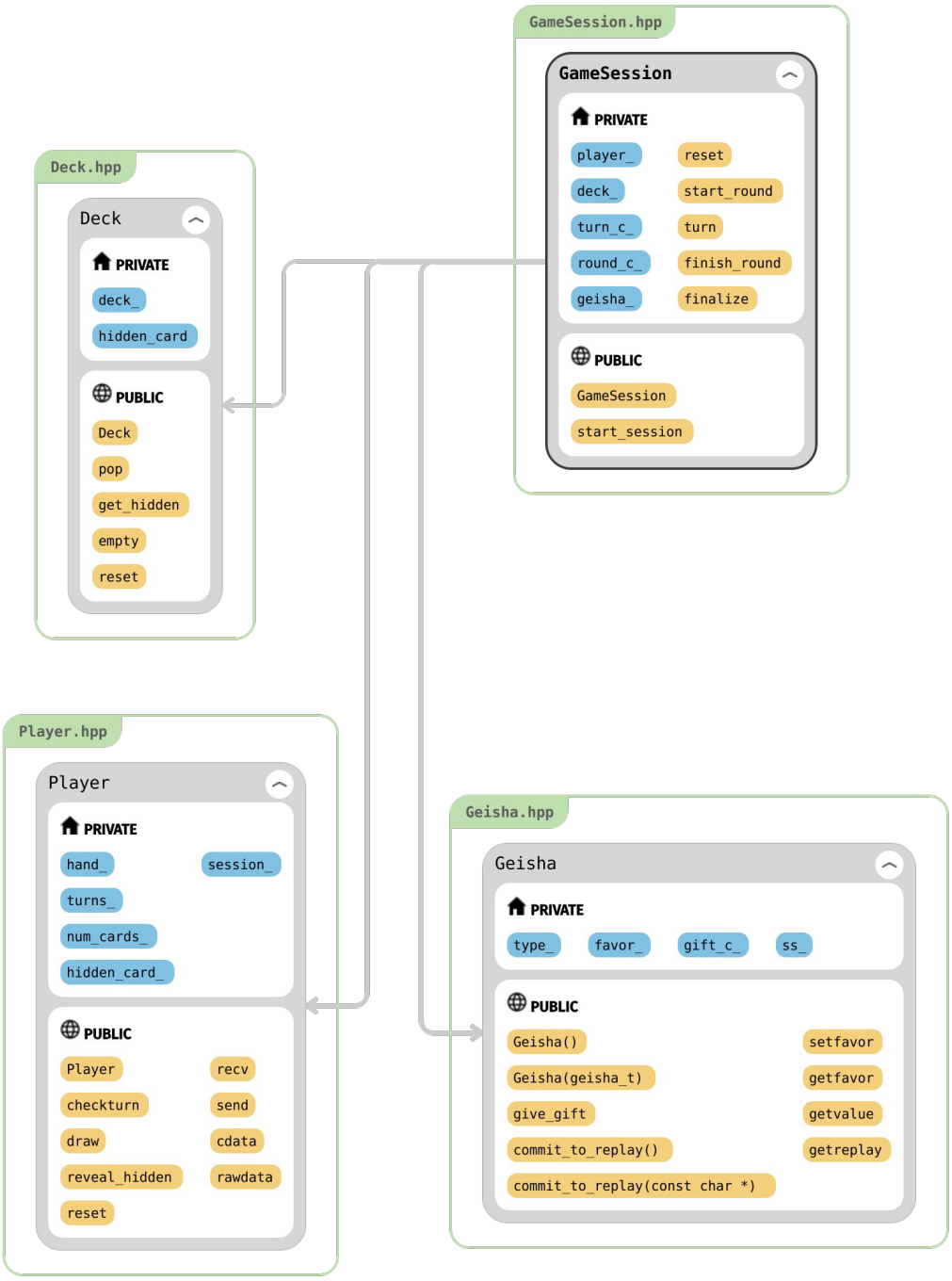


Рис. 3.8 Система класів, відповідальних за логіку ігрового процесу

імплементації обох, проте це не є обов'язковою умовою. Використання grpc дозволяє за необхідності робити й підключати в систему модулі з допомогою будь-якої мови, що має grpc бібліотеку. Так само й зі зв'язком з користумачем через JSON - можна робити майже на будь-якій мові.

3.6 JSON API модулів

Для стандартизації та валідації повідомлень від користувача використовуються файли JSON схем (див. Рис. 3.9). Повідомлення від сервера до користувача не мають JSON схем, адже серверу не потрібно їх валідувати.

Для формування JSON сервером для користувача використовуються функції побудови JSON на основі DOM API. Цей метод дозволяє швидко (в плані виконання коду) серіалізувати JSON.

```
{
    title: Turn,
    description: Turn,
    type: object,
    properties: {
        turn: {
            description: Turn type,
            type: integer,
            minimum: 0,
            maximum: 3
        },
        cards: {
            description: Sequence numbers of cards in hand to use,
            type: array,
            minItems: 1,
            maxItems: 4,
            uniqueItems: true,
            items: {
                type: integer,
                minimum: 0,
                maximum: 6
            }
        }
    },
    required: [
        turn,
        cards
    ]
}
```

Рис. 3.9 JSON схема ходу гравця

3.7 Клієнт

Ціль цієї роботи - це створити сервер. Проте для тестування й

відлагоджування сервера необхідно зробити хоча б мінімальний клієнт або його імітацію, що будуть слідувати основному сценарію використання сервера користувацькою програмою.

Для простоти було використано ту саму мову й WebSocket бібліотеку.

Це допомогло швидше зробити перший робочій екземпляр, а потім додати в нього другий сценарій, щоб можна було симулювати гру двох гравців за допомогою одного виконуваного файлу.

На рисунку 3.10 показані обидва сценарії взаємодії під час ігрового процесу ("r" означає прочитати).

```
static const char *p1_turn[] {
    "r", /*round start*/
    "r", "r", R"({"turn":3,"cards":[0,1,2,3]})", "r", /*turn 1*/
    "r", "r", R"({"choice":3})", /*turn 2*/
    "r", "r", R"({"turn":2,"cards":[0,1,2]})", "r", /*turn 3*/
    "r", "r", R"({"choice":2})", /*turn 4*/
    "r", "r", R"({"turn":1,"cards":[0,1]})", /*turn 5*/
    "r", "r", /*turn 6*/
    "r", "r", R"({"turn":0,"cards":[0]})", /*turn 7*/
    "r", "r", /*turn 8*/
    "r" /*reveal hidden*/
};
static const char *p2_turn[] {
    "r", /*round start*/
    "r", "r", R"({"choice":2})", /*turn 1*/
    "r", "r", R"({"turn":2,"cards":[1,0,2]})", "r", /*turn 2*/
    "r", "r", R"({"choice":3})", /*turn 3*/
    "r", "r", R"({"turn":3,"cards":[2,0,3,1]})", "r", /*turn 4*/
    "r", "r", /*turn 5*/
    "r", "r", R"({"turn":0,"cards":[1]})", /*turn 6*/
    "r", "r", /*turn 7*/
    "r", "r", R"({"turn":1,"cards":[0,1]})", /*turn 8*/
    "r" /*reveal hidden*/
};
```

Рис. 3.10 Приклад сценаріїв однієї роздачі

3.8 Основний сценарій роботи сервера

1. Перший користувач приєднується до модуля аутентифікації й просить створити собі сесію для гри з другом.

2. Модуль аутентифікації передає першому користувачу id його сесії, щоб той передав це id своєму другу, зберігає з'єднання з першим користувачем й очікує на запит від другого користувача, що має містити вищезгаданий id.
3. Модуль аутентифікації приймає нове підключення, що передає id сесії, яке вже записане на сервері. Відразу модуль викликає віддалену процедуру створення сесії в модулі ігрових сесії.
4. Модуль ігрових сесій вже має готову сесію, тому він асинхронно передає дані для входу в ігрову сесію для обох гравців й відкриває два нових сокети для можливих майбутніх сесій.
5. Модуль аутентифікації передає аутентифікаційні токени та порти для підключення обом гравцям й розриває з'єднання.
6. Один з гравців підключається до "нового" сервера (модуля ігрових сесій) за наданим портом й токеном.
7. Модуль ігрових сесій звіряє отриманий токен з тим, що він генерував для порту на якому відбулося підключення, й зберігає сесію з токеном, що має ідентифікувати порт підключення другого гравця, окремо в очікуванні на другого гравця.
8. Модуль ігрових сесій отримує друге підключення, звіряє токен асоційований з портом та аутентифікаційний токен, якщо обидва співпадають, то обом гравцям надсилається повідомлення про початок гри.
9. Проходить ігровий процес (див. пункт 3.9).
10. Після завершення гри з'єднання розривається й на цьому закінчується взаємодія з користувачем.

3.9 Правила гри та їх реалізація в модулі ігрових сесій

Гра проводиться за участі двох гравці.

Основним завданням гри є завойовування 7 об'єктів на полі.

Це робиться за допомогою відповідних кожному об'єкту карт.

Об’єкти можна ідентифікувати за кількістю відповідних їм карт. Так є 3 об’єкта “двійки” (з двома картами), 2 об’єкта “трійки”, одна “четвірка” й одна “п’ятірка”.

В сумі маємо 2+2+2+3+3+4+5 = 21 карту в колоді.

До початку одна карта випадкова карта в закриту відкладається.

Гравці на початку раунду отримують по 6 карт й кожен свій хід додатково витягують з колоди по 1 карті, не показуючи.

Існує 4 варіанти ходу гравця, кожен з яких можна використовувати лише раз за раунд:

1. Сховати *одну* карту не показуючи, щоб застосувати її на свою користь в кінці раунду
2. Сховати *дві* карти у відбій не показуючи
3. Запропонувати опоненту *три* карти, з яких він вибере *одну* й застосує на свою користь, а *дві* інших застосуєте ви
4. Запропонувати опоненту *чотири* карти в парах, з яких він вибере *одну* пару (із запропонованих *пар*, а не будь-які дві карти з 4) й застосує на свою користь, а іншу пару застосуєте ви

Таким чином кожен гравець зробить по 4 дії за раунд, що складе в сумі 20 карт з колоди й 1 карта вже схована.

В кінці раунду гравці відкривають усі карти, застосовують сховані карти й підраховують очки:

- якщо в одного з гравців перевага по кількості карт на об'єкті, то він займає цей об'єкт
- якщо у гравців однакова кількість карт на об'єкті, то нічого не змінюється (якщо об'єкт був в одного з гравців за результатами попередніх раундів, то він і залишається в тому ж стані)
- кожен об'єкт дає стільки очок скільки є відповідних йому карт

- якщо один із гравців набрав 11 очок, то він перемагає
- інакше, якщо один із гравців зайняв 4 об'єкта, то він перемагає

Кінцем гри вважається перемога одного з гравців під кінець раунду або кінець 3 раунду.

В кінці третього раунду оголошують нічию за відсутності переможця.

На рисунку 3.11 кольоровим текстом зображено згаданий в пункті 3.7 сценарій гри.

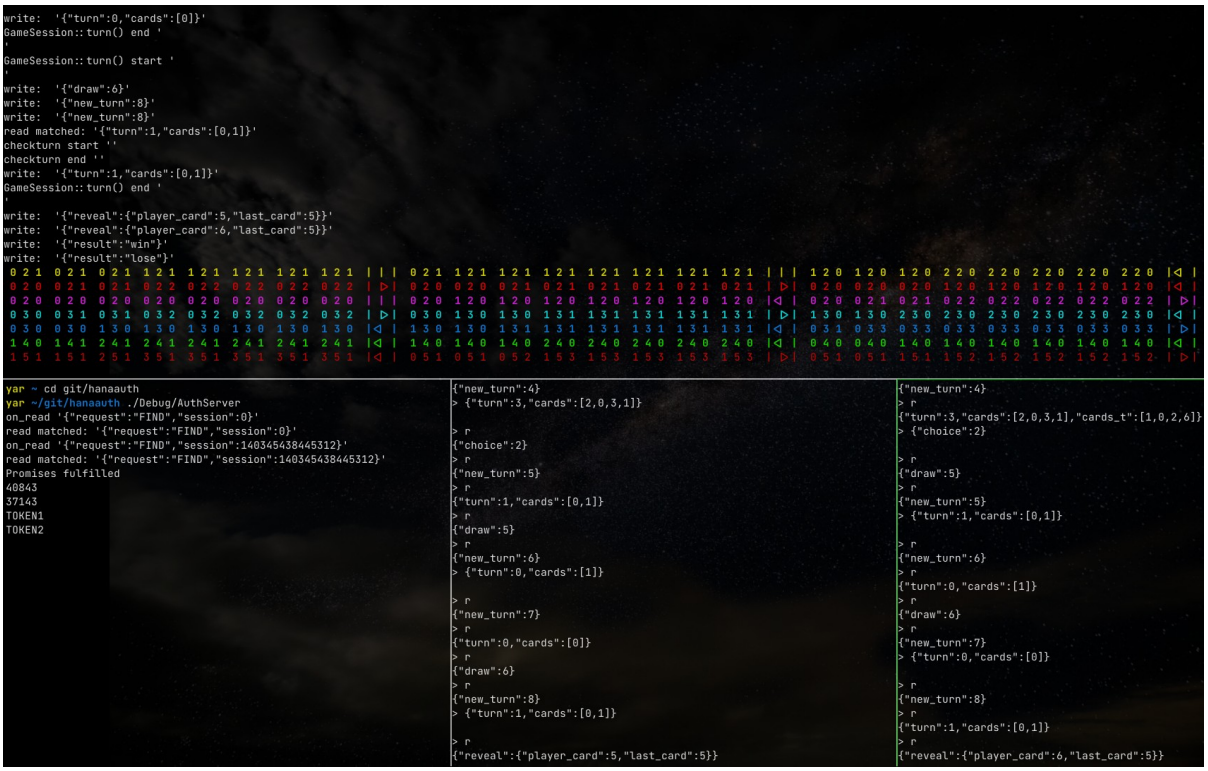


Рис. 3.11 Приклад виконання сценарію обома модулями та двома клієнтами

3.10 Kubernetes кластер

Для розгортання сервера був налаштований Kubernetes кластер з однією фізичною вершиною на якій розгортався модуль аутентифікації та з двома вершинами на яких розгорталися екземпляри модуля ігрових сесій.

Для підключення й тестування використовувався комп'ютер, що не входить до кластеру.

Схема підключення наведена в Додатку В.

3.11 Можливості розвитку й покращення

1. Найперше варто було б використовувати шифрування при передачі даних. Для цього під час підключення треба додатково зробити SSL handshake.
2. Можна додати обробку неправильно сформованих JSON від користувачів, як шахрайство та/або погане інтернет з'єднання.
3. Можна додати шифрування токенів аутентифікації за допомогою публічних ключів, що модуль аутентифікації не мав доступу до них.
4. Варто розробити повноцінну систему обробки помилок, адже наразі кожна помилка просто завершує сесію для обох користувачів без пояснення причин.
5. Можна написати повноцінний клієнт гри, щоб можна було простіше тестувати логіку ігрового процесу.

ВИСНОВОК ДО 3 РОЗДІЛУ

У даному розділі було описано структуру та архітектуру проекту розподіленого сервера, структуру кластера, на якому проводилося розгортання проекту. Під час розробки був зроблений акцент на загальності, фундаментальності, простоті підтримки, використання та розвитку сервера. Архітектура розроблена на основі паттерну мікросервісів. Було описано сценарії роботи й використання сервера найпростішим клієнтським застосунком, а також правила гри, для якої написано сервер. Також описані декілька ключових сценаріїв використання з поясненням яка поведінка додатку та яка логіка знаходиться всередині.

ВИСНОВКИ

Результатом даної бакалаврської дипломної роботи є реалізація розподіленого сервера карткової гри з двома модулями, що можуть незалежно один від одного розгортатися на Kubernetes кластері в будь-якій кількості екземплярів, взаємодіють між собою за допомогою технології grpc та обмінюються даними з користувачами у форматі JSON.

В ході виконання даної роботи були розглянуті різноманітні рішення серверів карткових ігор, які вже використовуються в різних компаніях, а також можливі підходи до написання розподілених серверів, що використовуються в сучасних проектах.

Були проаналізовані доступні та актуальні засоби для розробки системи, в тому числі мови програмування, фреймворки, мережеві бібліотеки, засоби контейнеризації, оркестрації та інше. Було вирішено писати сервер на мові програмування C++ з допомогою бібліотек Boost.Beast, Boost.Asio, grpc та RapidJSON. При виборі бібліотек основним критерієм була швидкість роботи кінцевого продукту.

В ході виконання дипломної роботи була досягнута поставлена мета та вирішена поставлена задача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Advantages & disadvantages of django framework. *CodeCondo*. URL: https://codecondo.com/pros-cons-of-django-framework/ (дата звернення: 31.05.2021).
2. Advantages and disadvantages of c++. *DataFlair*. URL: https://data-flair.training/blogs/advantages-and-disadvantages-of-cpp/ (дата звернення: 31.05.2021).
3. Advantages and disadvantages of PHP. BigCheapHosting. URL: https://bigcheaphosting.com/advantages-and-disadvantages-of-php/ (дата звернення: 31.05.2021).
4. ASP.NET core pros and cons. *UKAD R&D center*. URL: https://www.ukad-group.com/blog/aspnet-core-8-pros-and-3-cons/ (дата звернення: 31.05.2021).
5. Boost.Beast introduction. Boost C++ Libraries. URL: https://www.boost.org/doc/libs/1_73_0/libs/beast/doc/html/beast/introduction.html (дата звернення: 31.05.2021).
6. Delta V Software. Remote call framework (RCF). Delta V Software. URL: https://www.deltavsoft.com/doc/#rcf_user_guide.Intro.WhatIsRcf (дата звернення: 31.05.2021).
7. Developer interview with eric dodds and ben brode. Hearthpwn. URL: https://www.hearthpwn.com/news/296-developer-interview-with-eric-dodds-and-ben-brode (дата звернення: 30.05.2021).
8. Distributed Object Computing (DOC) Group. ACE overview. DRE Systems. URL: http://www.dre.vanderbilt.edu/~schmidt/ACE-overview.html (дата звернення: 31.05.2021).
9. Google. gRPC overview. gRPC. URL: https://www.grpc.io/ (дата звернення: 31.05.2021).

10. Hexolus' server-sided anticheat. DevForum | Roblox. URL: https://devforum.roblox.com/t/hexolus-server-sided-anticheat/983706 (дата звернення: 30.05.2021).
11. ISSN: 2070-1721. The JavaScript Object Notation (JSON) Data Interchange Format. Чинний від 2014-03-16. Вид. офіц. IETF, 2014. 16 с. URL: https://www.ietf.org/rfc/rfc7159.txt (дата звернення: 31.05.2021).
12. Jamshidi, Pooyan & Pahl, Claus & Mendonça, Nabor & Lewis, James & Tilkov, Stefan. (2018). Microservices: The Journey So Far and Challenges Ahead. IEEE Software. 35. 24-35. 10.1109/MS.2018.2141039.
13. Java spring pros and cons - javatpoint. www.javatpoint.com. URL: https://www.javatpoint.com/java-spring-pros-and-cons (дата звернення: 31.05.2021).
14. Jones S. The pros and cons of programming in go. WillowTree. URL: https://willowtreeapps.com/ideas/the-pros-and-cons-of-programming-in-go (дата звернення: 31.05.2021).
15. Kubernetes Components. Kubernetes. URL: https://kubernetes.io/docs/concepts/overview/components/ (дата звернення: 31.05.2021).
16. Myst B. facil.io - the c web application framework. facil.io. URL: http://facil.io/ (дата звернення: 31.05.2021).
17. Open Container Initiative Runtime Specification. GitHub. URL: https://github.com/opencontainers/runtime-spec/blob/master/spec.md (дата звернення: 31.05.2021).
18. POCO C++ Library overview. POCO C++ Libraries. URL: https://pocoproject.org/#usecases (дата звернення: 31.05.2021).
19. Pros & cons of using ruby on rails for web development. Codica. URL: https://www.codica.com/blog/pros-and-cons-of-ruby-on-rails-for-web-development/ (дата звернення: 31.05.2021).

20. Ramkumar R. A simple client – server game. Medium. URL: https://medium.com/@rohitramkumar308/a-simple-client-server-game-26ceb7be673c#.a2pvvnlan (дата звернення: 30.05.2021).
21. Rationale - 1.75.0. Boost.Asio rationale. URL: https://www.boost.org/doc/libs/1_75_0/doc/html/boost_asio/overview/rationale.html (дата звернення: 31.05.2021).
22. Royston Q. Server-Side rendering. Medium. URL: https://medium.com/@quinn.royston/server-side-rendering-213f66b27229 (дата звернення: 30.05.2021).
23. The good and the bad of node.js web app development. AltexSoft. URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/ (дата звернення: 31.05.2021).
24. ValveSoftware. GameNetworkingSockets advantages and disadvantages. GitHub. URL: https://github.com/ValveSoftware/GameNetworkingSockets (дата звернення: 31.05.2021).
25. Yip M. Native json benchmark. GitHub. URL: https://github.com/miloyip/nativejson-benchmark (дата звернення: 31.05.2021).
26. ZeroMQ overview. ZeroMQ. URL: https://zeromq.org/ (дата звернення: 31.05.2021).

Додаток А

ПРИНЦИПОВА СХЕМА

до дипломного проєкту

на тему: «Розподілений сервер карткової гри»

Київ – 2021 року

					ІАЛЦ.467100.004 Д1	
Змн.	Арк.	№ документа	Підпис	Дата		
Розроб.		Федоряченко Я.П.			Розподілений сервер карткової гри Принципова схема	Літ. / Аркуш 1 / Аркушів 1
Перев.		Шевело О.П.				
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, Група ІВ - 71
Н. Контр.		Сімоненко В.П.				
Затверд.		Стіренко С.Г.				

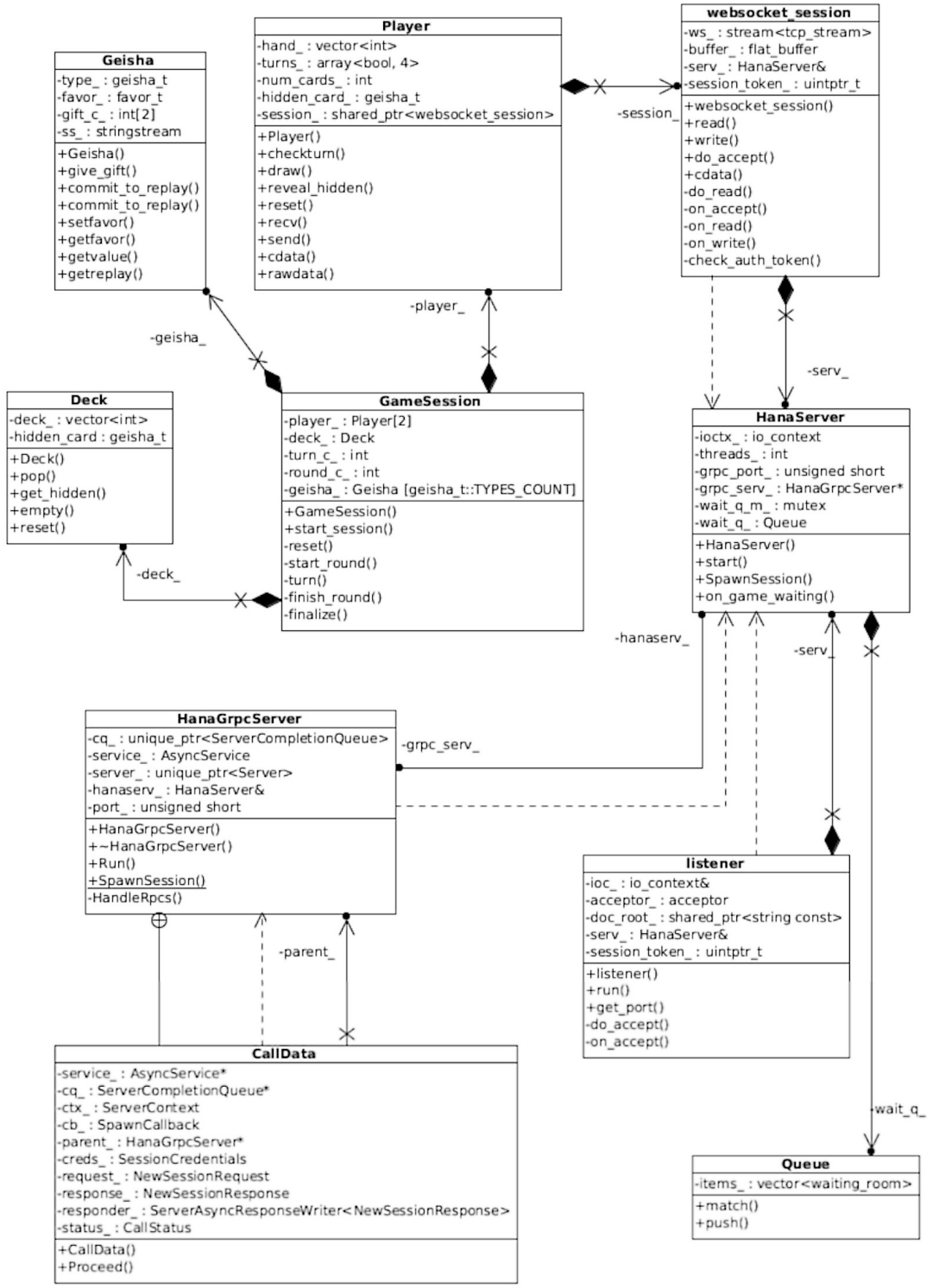
Додаток Б

ФУНКЦІОНАЛЬНА СХЕМА

до дипломного проєкту

на тему: «Розподілений сервер карткової гри»

Київ – 2021 року



					ІАЛЦ.467100.005 Д2			
Змн.	Арк.	№ документа	Підпис	Дата				
Розроб.		Федоряченко Я.П.			Розподілений сервер карткової гри Функціональна схема	Літ.	Аркуш	Аркушів
Перев.		Шевело О.П.					1	1
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, Група ІВ - 71		
Н. Контр.		Сімоненко В.П.						
Затверд.		Стіренко С.Г.						

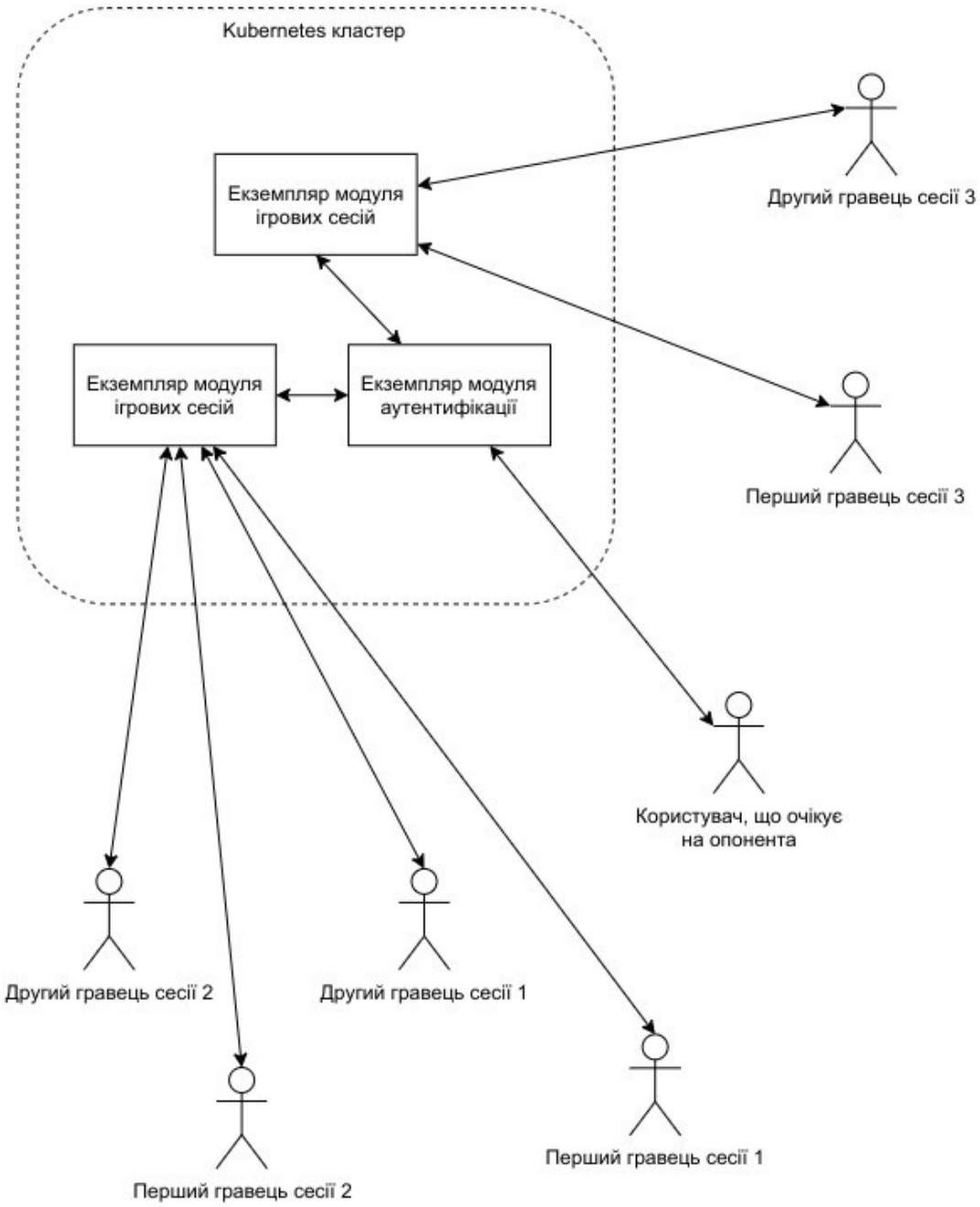
Додаток В

СТРУКТУРНА СХЕМА

до дипломного проєкту

на тему: «Розподілений сервер карткової гри»

Київ – 2021 року



					ІАЛЦ.467100.006 Д3
Змн.	Арк.	№ документа	Підпис	Дата	
Розроб.		Федоряченко Я.П.			Розподілений сервер карткової гри Структурна схема
Перев.		Шевело О.П.			
Реценз.					
Н. Контр.		Сімоненко В.П.			
Затверд.		Стіренко С.Г.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, Група ІВ - 71