Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультета (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: ***Інтелектуальна система менеджменту спортивних команд***

Виконав студент 6 курсу, групи КН-61м
спеціальності:
122 „Комп'ютерні науки”
(шифр і назва напрямку підготовки спеціальності)
Бокало Н.Д.
(прізвище та ініціали)

Керівник ***Павлюк У.В.***
(прізвище та ініціали)

Рецензент Дещук М.В.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ:
Завідувачка кафедри КН
_____ Борецька І.Б.
„10" грудня 2025 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бокалу Назару Дмитровичу
(прізвище, ім'я, по батькові)

1.Тема роботи: ***Інтелектуальна система менеджменту спортивних команд***
керівник роботи *Павлюк Уляна Володимирівна, к.е.н.*
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу від *"29" квітня 2025 року, №С-288.*
2.Термін подання студентом проєкту (роботи) *10 грудня 2025 р.*
3. Вихідні дані до проєкту (роботи) *Аналіз попередніх досліджень. Огляд технологій та програмного забезпечення для реалізації технічного завдання*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне забезпечення
Математичне забезпечення
Програмне забезпечення
Розроблення стартап-проєкту
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація із висвітленням ключових моментів розробки інтелектуальної системи в розрізі розділів пояснювальної записки (див. п.4).

6. Дата видачі завдання *1 травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів	01.05.2025-20.05.2025	Виконано
2.	Постановка задачі і її формалізація	21.05.2025-22.05.2025	Виконано
3.	Виконання вхідного етапу технології	06.06.2025-17.06.2025	Виконано
4.	Реалізація головних алгоритмів проєкту	22.06.2025-09.07.2025	Виконано
5.	Виконання етапу відлагодження проєкту	10.07.2025-22.07.2025	Виконано
6.	Тестування проєкту	20.08.2025-10.09.2025	Виконано
7.	Оформлення пояснювальної записки до дипломного проєкту	01.11.2025-30.11.2025	Виконано

Студент _____ Бокало Н.Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Павлюк У.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота містить 52 сторінки пояснювальної записки, 24 рисунки, 2 таблиці, 2 додатки, 5 формул, 20 джерел.

Магістерська робота присвячена розробці API для універсальної системи менеджменту спортивних команд. Додатковим сервісом виступає автоматизована формула для підрахунку цін спортсменів. Робота написана об’єктно орієнтованою мовою Java. Під час розробки API було використано декілька фреймворків, а саме Spring Boot як основа розробки, Hibernate для роботи з базою даних, PostgreSql, Liquibase для міграції баз даних.

Ключові слова: *Java, Spring Boot, Hibernate, Liquibase, PostgreSQL*.

ANNOTATION

The master's qualification thesis consists of 52 pages of explanatory notes, 24 figures, 2 tables, 2 appendices, 5 formulas, and 20 references.

The work is dedicated to the development of an API for a universal sport team management system. An additional service included is an automated formula for calculating athlete prices. The work is written in the object-oriented programming language Java. During API development, several frameworks were used, including Spring Boot as the core framework, Hibernate for database interaction, PostgreSQL as the database, and Liquibase for database migrations.

Keywords: *Java, Spring Boot, Hibernate, Liquibase, PostgreSQL*.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити API для управління спортивними командами та їхніми гравцями, який дозволяє керувати складом, фінансами, а також здійснювати трансфери гравців між командами з урахуванням бізнес-логіки, правил та фінансових обмежень. Створити базу даних з інформацією про команд, гравців.

Використати стек технологій: мови програмування Java, PostgreSQL для розробки бази даних та технологію Hibernate для комфортної взаємодії з базою даних.

Розробити серверну частини, що мають забезпечувати виконання наступних вимог:

1. Створення, читання, оновлення та видалення команд;
2. Зберігання назви команди, балансу рахунку, комісії трансферу;
3. Зв’язок гравця з командою та дані про гравця;
4. Перехід гравця з однієї команди до іншої;
5. Обчислення вартості трансферу за формулою;
6. Валідація та обробка помилок.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс програмування, що дозволяє взаємодіяти з іншими програмами або сервісами;

DTO – шаблон проєктування в програмуванні, для передачі даних;

KPI – Key Performance Indicators – ключові показники ефективності проєкту.

MVC – Model View Controller – архітектурний шаблон, що використовується в програмному забезпеченні для розділення компонентів додатка на три основні частини: Модель, Вид і Контролер.

SDK – набір засобів для розробки програмного забезпечення;

СУБД – система управління базами даних, що використовується для зберігання та обробки даних.

ВСТУП

Актуальність теми. У сучасному світі інформаційні технології відіграють ключову роль у розвитку спорту, зокрема в автоматизації процесів управління командами, гравцями та фінансовими операціями. Ефективний менеджмент спортивних команд потребує систем, здатних обробляти великі обсяги даних, контролювати рух гравців між командами, аналізувати фінансові показники та забезпечувати збереження й доступність інформації. Це особливо важливо для аматорських, молодіжних та напівпрофесійних ліг, де часто бракує систем керування. Сьогодні існуючі платформи для управління спортивними командами є переважно комерційними, дорогими або складними у впровадженні, що значно обмежує їх застосування на рівні місцевих клубів чи невеликих організацій. Тому потребою є створення відкритої, масштабованої та гнучкої системи, яка може автоматизувати ключові процеси: створення, оновлення, видалення даних про гравців і команди, проведення трансферів, розрахунок їхньої вартості та контроль фінансових потоків.

Об’єкт дослідження – інтелектуальна система керування спортивними командами.

Предметом дослідження виступають архітектурні рішення та серверні технології, необхідні для реалізації функціоналу управління гравцями, командами та трансферними операціями.

Метою роботи є розробка серверної частини для інтелектуального менеджменту спортивних команд із використанням Java, Spring Boot та пов’язаних технологій. Завданням є реалізація логіки трансферу гравців між командами з урахуванням бізнес-правил, створення бази даних з інформацією про спортсменів та команди, а також валідація операцій.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

— розробити алгоритм динамічного розрахунку вартості гравця на основі його ігрових показників та віку, а також проведення фінансових трансферів між командами;
— спроєктувати архітектуру програмної системи та схему бази даних, що забезпечують надійне зберігання інформації та цілісність даних при виконанні транзакцій;
— написати серверну частину (Back-end) з використанням мови Java та фреймворку Spring Boot, реалізувавши REST API для взаємодії з клієнтом;
— дослідження розробленої системи: перевірити коректність роботи алгоритму оцінювання вартості гравців та протестувати сценарії валідації фінансових операцій.

Наукова новизна. На відміну від існуючих комерційних платформ, які орієнтовані на великий бюджет, у даному дослідженні запропоновано відкрите, модульне рішення, побудоване на основі технологій Java, Spring Boot та Hibernate, яке забезпечує гнучкість, масштабованість та простоту адаптації під конкретні потреби. Особливістю розробленої системи є автоматизована модель розрахунку вартості трансферу гравців, яка враховує фактори, а також комісійний відсоток команди-продавця, що забезпечує прозорість фінансових операцій. Також наукова новизна полягає в інтеграції механізму Liquibase для управління версіями бази даних, що дозволяє ефективно керувати змінами структури бази даних протягом усього життєвого циклу проєкту.

Практична значення розробки полягає в тому, що створена система може послужити основою для розробки більш широких інструментів управління спортивною інфраструктурою, а також легко адаптуватись до інших галузей, таких як корпоративний, нерухомість, автобізнес тощо.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність проблемної області

У сучасному світі спортивна індустрія активно розвивається, що зумовлює зростаючу потребу у цифрових рішеннях для ефективного управління командами, гравцями та фінансовими процесами, зокрема трансферами. Особливо це стосується аматорських, молодіжних та напівпрофесійних ліг, де відсутні централізовані системи керування.

Існуючі платформи часто є закритими, дорогими або складними у впровадженні. Тому актуальною є розробка вебсистеми, яка дозволить автоматизувати процеси управління спортивними командами: збереження даних про команди та гравців, проведення трансферів, а також контроль фінансів, із можливістю масштабування і розширення в майбутньому.

Особливу роль відіграє серверна частина – бекенд, яка є ядром всієї системи. Вона забезпечує обробку запитів, виконання бізнес-логіки, роботу з базою даних та взаємодію з іншими сервісами (наприклад, frontend або мобільним застосунком).

1.2 Аналіз проблемної області

Аналіз предметної області, пов’язаної з розробкою бекенд-частини інформаційної системи для управління спортивними командами, дозволяє виявити основні вимоги до системи, а також окреслити технічні та логічні аспекти, що визначають її ефективність і функціональність.

У ході аналізу були розглянуті аналогічні рішення, які використовуються в професійному та аматорському спорті для обліку гравців, менеджменту команд, трансферів, аналітики та фінансових операцій. Значна частина подібних систем є комерційними або закритими, що ускладнює їх використання для локальних клубів або власних ініціатив. Це підкреслює потребу у створенні відкритої та гнучкої серверної платформи.

Ключові аспекти, які були враховані при аналізі:

- *функціональність* – система має забезпечувати повну підтримку CRUD-операцій для гравців, команд та трансферів. Також важливими є перевірка логіки трансферу (наявність коштів у клубу, комісії), можливість оновлення фінансових даних після кожної операції, та ведення історії змін;
- *масштабованість і продуктивність* – система має бути здатна обробляти велику кількість запитів при зростанні кількості користувачів або записів у базі, не втрачаючи стабільності та швидкодії;
- *інтеграція* – для забезпечення ефективної взаємодії з іншими частинами системи (наприклад, frontend чи мобільним додатком), необхідно створити чітко задокументований REST API;
- *можливість тестування* – важливою складовою сучасного бекенду є підтримка модульного та інтеграційного тестування, що дозволяє виявляти помилки на ранніх етапах розробки.

Результати аналізу дозволили сформулювати основні вимоги до реалізації серверної частини: зручність структури, підтримка розширення функціоналу та стабільна взаємодія з базою даних. Усі ці критерії було враховано при виборі архітектури системи та технологій, зокрема Spring Boot як основного фреймворку для реалізації бекенду.

1.3 Вибір архітектурної моделі

У процесі розробки бекенд-частини системи управління спортивними командами було проведено аналіз різних архітектурних підходів, зокрема монолітної, мікросервісної та серверлесс архітектури, з метою визначення оптимального варіанту для реалізації даного проєкту.

Монолітна архітектура передбачає створення цілісного застосунку, де всі компоненти тісно пов’язані між собою. Цей підхід є простішим у реалізації на початковому етапі, однак ускладнює масштабування та впровадження змін у майбутньому. До переваг монолітної архітектури належить легкість розгортання та невисока початкова складність, що сприяє швидкій реалізації

основного функціоналу й підтриманню цілісності системи на початкових етапах розробки [18, с.9].

Мікросервісна архітектура, навпаки, передбачає поділ системи на незалежних сервісів, кожен з яких виконує окрему бізнес-функцію (наприклад, керування гравцями, управління командами, обробка трансферів, фінансові розрахунки). Цей підхід забезпечує гнучкість у розгортанні, масштабуванні та обслуговуванні окремих компонентів системи. Крім того, мікросервіси полегшують тестування та пришвидшують впровадження нових функцій [18, с.45].

Серверлесс архітектура базується на використанні хмарних функцій, що виконуються у відповідь на події. Такий підхід може бути ефективним для вузькоспеціалізованих, подієвих задач — наприклад, автоматичної обробки завантажених файлів у хмарне сховище або надсилання повідомлень при зміні статусу користувача. Водночас серверлесс має обмеження щодо гнучкості, контролю над середовищем виконання та складності інтеграції з базами даних і сторонніми сервісами в рамках проєкту [18, с.63].

У результаті аналізу було прийнято рішення реалізувати систему на основі монолітної архітектури із чітко виділеними модулями у межах одного застосунку. Такий підхід є доцільним з огляду на масштаби проєкту, ресурси команди та потребу у простій інтеграції між модулями. При цьому структура проєкту побудована з урахуванням можливої подальшої міграції до мікросервісної архітектури.

Таким чином, вибраний підхід дозволяє досягти балансу між простотою розробки та підтримкою чіткого поділу відповідальностей у кодовій базі, що є важливим для забезпечення надійності, зручності супроводу та безпроблемного масштабування бекенд-частини системи.

Візуальне порівняння моноліту та мікросервісів наведено на рисунку 1.1.

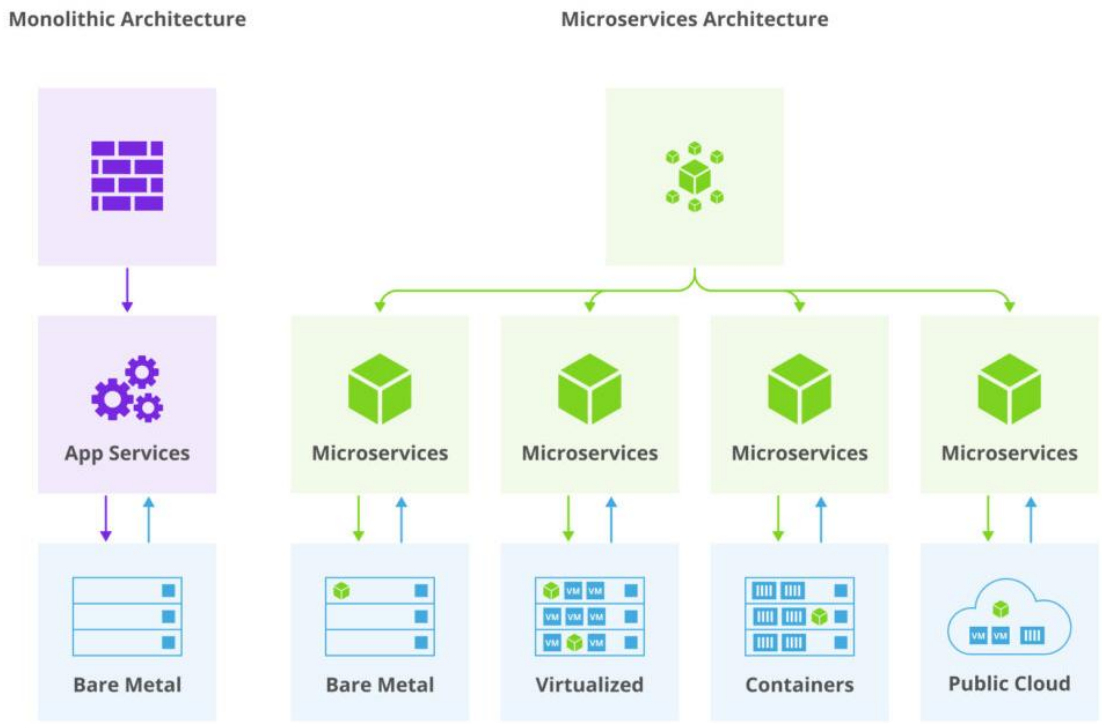

Рисунок 1.1 – Моноліт та мікросервіси

1.4 Постановка завдання

Завданням магістерської роботи є розробка API менеджемен-ту спортивних команд та гравцями, реалізованої з використанням технологій Java, Spring Boot, Hibernate. Система повинна забезпечити базові операції для управління командами та гравцями, зокрема можливість проведення трансферів гравців між командами з урахуванням вартості трансферу та комісії. Основною метою є створення методів, які дозволять ефективно виконувати операції щодо управління командами та гравцями в рамках футбольної ліги.

Головні завдання, які потрібно вирішити в рамках постановки завдання, включають наступне:

- *розробка базових CRUD операцій* (реалізація операцій створення, читання, оновлення та видалення для управління командами та гравцями через REST API; кожен запит до API має бути коректно оброблений, із відповідним статусом HTTP);
- *реалізація операції трансферу гравця*: операція трансферу гравця повинна здійснюватися за певною формулою:

1. Вартість трансферу = кількість місяців досвіду гравця * 100000 / вік гравця у роках.
2. Комісія зі сторони команди повинна бути врахована в межах від 0% до 10% від вартості трансферу.
3. Повна сума, яка включає вартість трансферу і комісію, повинна бути списана з рахунку покупця і переведена на рахунок продавця.

- *валідація операцій* (всі операції з гравцями та командами повинні проходити валідацію на сервері; помилки повинні оброблятися належним чином, із поверненням відповідного HTTP статусу та поясненням помилки);
- *наповнення бази даних* (реалізація початкового наповнення бази даних гравців та команд для забезпечення роботи системи, яке повинно включати створення декількох тестових записів, щоб система могла працювати в режимі тестування);
- *документація API* (створення Postman колекції для тестування реалізованого REST API, яка повинна включати запити для всіх основних операцій);
- *обробка помилок* (реалізація обробки помилок, включаючи перевірку наявності необхідних даних, коректність введення, а також обробку можливих помилок на сервері з відповідними повідомленнями та статусами HTTP);
- *тестування* (написання юніт-тестів для основних функціональностей з використанням тестових фреймворків для забезпечення високої якості коду.

Формулювання завдання є ключовим етапом у розробці серверної частини, адже воно визначає основні напрямки та цілі, яких необхідно досягти для успішної реалізації проєкту. Постановка завдання дозволяє ефективно планувати роботу, ресурси та визначати пріоритети, що суттєво впливає на кінцеву якість роботи.

Висновки до розділу

У результаті аналізу проблемної області та вибору архітектурної моделі для системи управління спортивними командами, було визначено основні вимоги та підходи до розробки серверної частини. Актуальність створення такої системи зумовлена зростаючою потребою у цифрових рішеннях для ефективного управління командами та гравцями, особливо в аматорських та напівпрофесійних лігах. Існуючі платформи часто є закритими та дорогими, тому створення відкритої та гнучкої системи для автоматизації цих процесів є важливою задачею.

Аналіз існуючих рішень показав необхідність забезпечення функціональності, масштабованості, продуктивності та інтеграції з іншими частинами системи. Завдяки вибору монолітної архітектури із чітко виділеними модулями, система забезпечить баланс між простотою розробки та можливістю подальшого масштабування. Це дозволить зберегти високу ефективність на етапі реалізації та забезпечити безперешкодне масштабування в майбутньому.

Розробка серверної частини на основі Spring Boot як основного фреймворку є доцільним вибором, який дозволяє забезпечити стабільну роботу системи з гнучкою архітектурою та можливістю інтеграції з іншими сервісами. Визначені вимоги до CRUD-операцій, валідації, фінансових розрахунків, тестування та документації API створюють чітку основу для успішної реалізації проєкту.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Java

Існує велика кількість мов програмування, і їх точну кількість визначити складно. Однак у сучасній IT-індустрії використовується лише близько десятої частини з них – саме ці мови застосовують для створення програмного забезпечення та різноманітних застосунків. Однією з найбільш популярних і затребуваних мов є Java. Уже понад десять років Java утримує позиції однієї з найвживаніших мов програмування на ринку.

Java – це мова програмування, випущена у 1995 році компанією «Sun Microsystems» як базова частина платформи Java. Після придбання «Sun Microsystems» у 2009 році подальший розвиток мови перейшов до компанії «Oracle». Синтаксис Java значною мірою наслідує C та C++. Зокрема, було взято за основу об’єктну модель C++, але її модернізовано та спрощено. Основна мета створення Java полягала у створенні мови, яка не залежить від типу платформи, тобто дає змогу писати програмне забезпечення, що може працювати на різних пристроях – від мобільних телефонів до побутової техніки [17, с.5].

Чому саме Java? Вона має низку переваг, які роблять її зручною та ефективною. Типи даних – Java є суворо типізованою мовою: кожна змінна має чітко визначений тип, відомий уже на етапі компіляції. Усі типи даних поділяються на примітивні (primitive) та посилальні (reference). До примітивних належать логічний тип boolean, числові типи byte, short, int, long, а також дійсні – float і double. Символьний тип представлений char. Посилальні типи охоплюють класи, інтерфейси та масиви, значенням яких є посилання на відповідний об’єкт або масив. Рядки в Java представлені класом String. Завдяки строгій типізації програміст може легше читати код і передбачати можливі помилки.

ООП. Java повністю дотримується принципів об’єктно-орієнтованого програмування, що дозволяє створювати модульні програми та багаторазово

використовувати код. Це значно спрощує підтримку та масштабування програмних систем великого рівня складності [19, с.12].

Кросплатформність. Однією з найбільш відомих переваг Java є її здатність працювати на будь-якій операційній системі без зміни вихідного коду. Програму достатньо один раз написати і скомпілювати у байт-код, який може бути виконаний на будь-якому пристрої, де встановлена Java Virtual Machine. Завдяки цьому немає потреби створювати окремі версії програм для Windows, Linux чи macOS.

Широке застосування. Java активно використовується у веброзробці, створенні мобільних застосунків, десктопних програм, серверних систем, корпоративних рішень, і навіть в ігровій індустрії.

Розробка вебдодатків на Java широко поширена й охоплює створення корпоративних платформ, онлайн-магазинів, соціальних сервісів та інших вебзастосунків. Основу технологій складають такі фреймворки, як Spring, а також різні бібліотеки і протоколи, що забезпечують роботу серверної логіки.

Мобільні додатки під Android також активно створюються за допомогою Java. Android SDK містить повний набір інструментів, необхідних для побудови функціональних мобільних програм. Багато популярних застосунків на Android, зокрема Instagram, Facebook, YouTube, спочатку розроблялися саме на Java.

Десктопні застосунки на Java дозволяють створювати кросплатформні програми, які працюють на різних операційних системах. Для побудови графічного інтерфейсу традиційно використовуються бібліотеки Swing і JavaFX. Вони забезпечують можливість створювати вікна, кнопки, текстові поля, таблиці та інші елементи інтерфейсу. До відомих застосунків, створених на Java, належать IntelliJ IDEA та Eclipse.

Отже, Java є однією з найпоширеніших мов програмування завдяки своїй надійності та універсальності. Сувора типізація, підтримка ООП підходу, кросплатформеність та широка екосистема інструментів роблять її ефективним вибором для розробки сучасних програмних рішень. Завдяки

активній спільноті та широкому спектру застосувань – від веброзробки до мобільних та десктопних систем – Java продовжує залишатися однією з ключових технологій у світовій IT-індустрії. Якщо врахувати її універсальність і стабільність, стає зрозуміло, чому ця мова зберігає свою популярність уже багато років.

2.2 Фреймворк Spring Boot

Spring Boot – це фреймворк для розробки Java додатків, який забезпечує швидке створення самостійних, готових до використання додатків на основі платформи Spring. Основна ідея Spring Boot полягає в спрощенні конфігурації та розгортання додатків, забезпечуючи заздалегідь налаштовані стартові умови, що дозволяють розробникам швидко створювати додатки без необхідності вручну налаштовувати велику кількість компонентів [18, с.20].

Основні *особливості Spring Boot*:

- конвенція над конфігурацією – Spring Boot надає заздалегідь налаштовані конфігураційні параметри, що дозволяють створювати додатки з меншим обсягом коду конфігурації;
- вбудований контейнер додатків – Spring Boot містить вбудовані сервери додатків (наприклад, Tomcat, Jetty або Undertow), що дозволяє створювати самостійні додатки, які можна запустити просто запустивши JAR-файл;
- управління залежностями – Spring Boot використовує систему автоматичного управління залежностями Maven або Gradle для автоматичного вирішення залежностей та включення необхідних бібліотек;
- автоматична конфігурація – Spring Boot надає автоматичну конфігурацію для багатьох бібліотек та компонентів Spring, що дозволяє автоматично налаштовувати додатки на основі класифікації класів та анотацій;

— підтримка вбудованих баз даних – Spring Boot дозволяє створювати додатки, які використовують вбудовані бази даних (наприклад H2), спрощуючи процес розгортання та тестування [18, с.34].

Переваги впровадження Spring Boot. В цілому, Spring Boot пропонує зручний і стандартизований спосіб створення Java додатків, зменшуючи кількість необхідного коду та спрощуючи процес розробки, конфігурації та розгортання додатків. Spring Boot виділяється серед інших фреймворків, оскільки він надає розробникам програмного забезпечення гнучке налаштування, надійну пакетну обробку, ефективний робочий процес та велику кількість інструментів, допомагаючи розробляти надійні та масштабовані програми на базі Spring [18, с.26].

Також слід зазначити, що Spring Boot включає в себе MVC (Model View Controller) – це архітектурний шаблон, що використовується в програмному забезпеченні для розділення компонентів додатка на три основні частини: Модель, Вид і Контролер. Цей шаблон допомагає створити добре організовану структуру програми, де кожна частина відповідає за свої власні обов'язки [19, с.549]. Загалом, шаблон MVC є добре встановленим підходом до розробки програмного забезпечення, який допомагає підтримувати структурований і ефективний код, зменшуючи залежність між різними компонентами додатка.

2.3 Hibernate

Hibernate – це один з найпопулярніших фреймворків для об'єктно-реляційного відображення (ORM — Object-Relational Mapping) у мові Java. Його головна мета — спростити роботу з базами даних, забезпечивши розробникам можливість взаємодіяти з реляційною базою даних, використовуючи об'єктно-орієнтовані підходи [17, с.5].

Основна ідея Hibernate – замість написання великої кількості SQL-запитів для взаємодії з базою даних, Hibernate дозволяє працювати з об'єктами Java, які автоматично перетворюються у відповідні записи таблиць бази даних. Це значно спрощує розробку та зменшує ймовірність помилок.

Особливості Hibernate:

— об'єктно-реляційне відображення (ORM) (автоматичне зіставлення між класами Java та таблицями в базі даних, наприклад, клас User може бути автоматично зв’язаний з таблицею user);

— автоматичне створення SQL-запитів (Hibernate самостійно генерує SQL-код на основі операцій з об’єктами Java);

— підтримка транзакцій (Hibernate має вбудовану підтримку управління транзакціями, що забезпечує цілісність даних);

— кешування (підтримка першого та другого рівня кешу для зменшення навантаження на базу даних);

— підтримка запитів HQL (Hibernate Query Language – мова запитів, схожа на SQL, але оперує не таблицями, а класами та їхніми властивостями).

Переваги використання Hibernate:

— зменшує обсяг шаблонного коду;

— спрощує роботу з транзакціями та підключенням до БД;

— підтримує наслідування, зв’язки один-до-одного, один-до-багатьох, багато-до-багатьох;

— незалежність від СУБД — можна легко змінити базу даних без зміни бізнес-логіки [1;17].

Hibernate часто використовується в корпоративних додатках, де потрібна гнучка та масштабована робота з базами даних. Він є основним ORM-рішенням у Spring-проєктах.

2.4 Liquibase

Liquibase – це інструмент для керування версіями бази даних, який дозволяє розробникам і командам впорядковувати, відслідковувати та застосовувати зміни до схеми бази даних в контрольований спосіб. Він підтримує різні бази даних, такі як PostgreSQL, MySQL, Oracle тощо.

Liquibase особливо зручний при використанні у великих проєктах, де важливо мати контроль над еволюцією бази даних упродовж всього життєвого

циклу додатку. Liquibase працює на основі changelog-файлів – файлів змін, які описують трансформації схеми бази даних у вигляді наборів змін. Ці файли можуть бути написані у форматах XML, YAML, JSON або SQL, що дозволяє вибрати найбільш зручний варіант відповідно до потреб проєкту [3].

Основні можливості Liquibase:

— *відстеження змін бази даних* (кожна зміна зберігається у changelog-файлі з унікальним ідентифікатором; Liquibase зберігає інформацію про застосовані зміни в спеціальній системній таблиці, що дозволяє уникнути повторного застосування тих самих змін);

— *інтеграція з CI/CD* (Liquibase легко інтегрується з CI/CD-пайплайнами, такими як Jenkins, GitHub Actions, GitLab CI тощо, що дозволяє автоматизувати оновлення схеми бази даних разом із деплоєм застосунку);

— *можливість rollback (відкату)* (для кожної зміни можна описати логіку відкату, що дозволяє безпечно скасовувати зміни при помилках);

— *гнучкий формат changelog* (Liquibase підтримує кілька форматів (XML, YAML, JSON, SQL), дозволяючи командам обирати найбільш зручний формат для роботи).

Найбільш часто використовується формат XML. Наприклад, changelog-файл може виглядати так (рис. 2.1):

```
<changeSet id="1" author="nvoxland">
    <createTable tableName="person">
        <column name="id" type="int" autoIncrement="true">
            <constraints primaryKey="true" nullable="false"/>
        </column>
        <column name="firstname" type="varchar(50)"/>
        <column name="lastname" type="varchar(50)">
            <constraints nullable="false"/>
        </column>
        <column name="state" type="char(2)"/>
    </createTable>
</changeSet>
```

Рисунок 2.1 – Changelog-файл

Переваги використання Liquibase:

— контроль версій бази даних (розробники можуть бачити історію змін, хто і коли їх вніс);
— автоматизація змін (changelog-файли можуть бути застосовані автоматично при запуску застосунку або через окрему команду);
— безпечні розгортання (Liquibase дозволяє перевіряти зміни перед їх застосуванням, а також створювати SQL-скрипти для ручної перевірки);
— сумісність із Spring Boot (Liquibase легко інтегрується в Spring Boot-додатки через залежність).

Liquibase – це потужний інструмент для управління змінами у базі даних, який ідеально підходить для командної розробки. Його інтеграція з Spring Boot дозволяє зручно керувати схемою бази даних протягом усього життєвого циклу проєкту. Завдяки підтримці rollback, мультиформатним changelog-файлам та автоматизації оновлень, Liquibase значно спрощує процес супроводу та оновлення реляційних баз даних у сучасних додатках.

2.5 Мова структурованих запитів SQL

SQL (Structured Query Language) – це стандартна мова програмування, яка використовується для керування реляційними базами даних і виконання різноманітних операцій над даними, зокрема: створення, оновлення, видалення та вибірка даних. SQL є основним засобом взаємодії з базами даних, забезпечуючи гнучкість, точність і контроль над інформацією, що зберігається у табличній формі.

Основна концепція SQL полягає у використанні простих декларативних запитів, які вказують, що саме необхідно зробити. Таким чином, користувач описує бажаний результат, а система управління базами даних самостійно визначає найефективніший спосіб його досягнення.

Основні можливості SQL [9]:

1. Операції з даними (DML – Data Manipulation Language):
    — SELECT – отримання даних з однієї або кількох таблиць;

— INSERT – додавання нових записів;

— UPDATE – оновлення існуючих записів;

— DELETE – видалення записів з таблиці.

2. Операції зі схемою бази даних (DDL – Data Definition Language):

— CREATE TABLE – створення нових таблиць;

— ALTER TABLE – зміна структури таблиць;

— DROP TABLE – видалення таблиць з бази даних.

3. Операції з доступом до даних (DCL – Data Control Language):

— GRANT, REVOKE – управління правами доступу до об'єктів бази даних.

```sql
-- Створення таблиці
CREATE TABLE users
(
    id       SERIAL PRIMARY KEY,
    username VARCHAR(255)        NOT NULL,
    email    VARCHAR(255) UNIQUE NOT NULL
);

-- Додавання нового користувача
INSERT INTO users (username, email)
VALUES ('ivan_k', 'ivan@example.com');

-- Отримання всіх користувачів
SELECT *
FROM users;

-- Оновлення даних
UPDATE users
SET email = 'new_email@example.com'
WHERE username = 'ivan_k';

-- Видалення користувача
DELETE
FROM users
WHERE id = 1;
```

Рисунок 2.2 – SQL-код

Цей набір SQL-запитів ілюструє типову послідовність дій для створення таблиці та керування даними в ній. Простота синтаксису SQL дозволяє легко інтегрувати її в будь-які додатки, які використовують бази даних.

Переваги використання SQL:

— стандартизованість – SQL є міжнародним стандартом, підтримуваним більшістю реляційних СУБД (PostgreSQL, MySQL, Oracle, SQL Server тощо);

— гнучкість – SQL дозволяє виконувати як прості запити, так і складні агрегації, об’єднання та підзапити;

— продуктивність – SQL дозволяє оптимізувати запити, використовувати індекси, кешування та інші механізми для прискорення доступу до даних.

Мова SQL є незамінним інструментом у сучасній розробці інформаційних систем, що працюють із реляційними базами даних. Вона забезпечує надійне, стандартизоване та ефективне управління структурованими даними, дозволяючи створювати, зберігати та обробляти інформацію з високим ступенем точності та контролю. Завдяки широкій підтримці, SQL залишається основною мовою взаємодії з базами даних у більшості корпоративних і вебзастосунків.

2.6 Система управління базами даних PostgreSQL

PostgreSQL – це система управління базами даних з відкритим вихідним кодом, яка підтримує розширюваність і відповідність стандартам SQL. Вона була створена в Каліфорнійському університеті в Берклі як частина проєкту POSTGRES і з часом перетворилась на одну з найпопулярніших СУБД у світі. PostgreSQL часто використовується у розробці корпоративних застосунків, вебсервісів, аналітичних систем, а також у фінансовому секторі завдяки своїй надійності, високій продуктивності та безпеці.

Основні особливості PostgreSQL: розширюваність, безпека, сильна підтримка транзакцій, типи даних. Користувачі можуть створювати власні функції, агрегати, типи даних, індекси. Це робить PostgreSQL надзвичайно гнучкою системою, яка легко адаптується до різноманітних бізнес-завдань. PostgreSQL дозволяє гнучко налаштовувати права доступу до бази даних за допомогою ролей, а також підтримує аутентифікацію. Поряд зі стандартними типами (INTEGER, VARCHAR, DATE тощо), PostgreSQL підтримує складні типи, зокрема масиви, JSON/JSONB, XML, UUID, ENUM [4].

Переваги PostgreSQL у порівнянні з іншими СУБД:

- безкоштовність – PostgreSQL є повністю відкритим продуктом з активною спільнотою та частими оновленнями без ліцензійних обмежень;
- масштабованість – PostgreSQL ефективно працює як на невеликих проєктах, так і в системах з великим навантаженням і обсягами даних;
- надійність – підтримка бекапів та вбудованих механізмів захисту забезпечує стабільність роботи в продакшн-середовищі;
- аналітичні можливості – завдяки підтримці CTE (Common Table Expressions), аналітичних функцій, віконних запитів і розширень PostgreSQL чудово підходить для складної аналітики.

PostgreSQL — це сучасна, потужна та безпечна СУБД, яка ідеально підходить для використання у поєднанні зі Spring Boot. Вона забезпечує гнучкість, продуктивність і відповідність стандартам, роблячи її надійною основою для побудови високонавантажених, критично важливих додатків. Завдяки своїм розширеним можливостям, PostgreSQL є хорошим вибором для багатьох професійних розробників і великих компаній у всьому світі.

Висновки до розділу

У цьому розділі було розглянуто ключові технології, які забезпечують реалізацію програмного забезпечення: мову програмування Java, фреймворк Spring Boot, Hibernate та систему керування базами даних PostgreSQL.

Java, як одна з найпопулярніших мов програмування, забезпечує кросплатформеність, об'єктно-орієнтований підхід і велику спільноту підтримки. Spring Boot значно спрощує створення та розгортання Java-додатків завдяки автоматичній конфігурації та вбудованим серверам. Hibernate, у свою чергу, усуває потребу у великій кількості SQL-коду, дозволяючи працювати з базами даних у вигляді об'єктів. PostgreSQL – це надійна, потужна та відкрита СКБД, яка ідеально підходить для розробки складних та масштабованих застосунків.

Разом ці інструменти формують ефективне, сучасне середовище для створення продуктивного програмного забезпечення.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритми

Алгоритм – це інструкція, яка задає спосіб розв'язання певної задачі або досягнення поставленої мети. Це логічна схема дій, що приводить від вхідних даних до очікуваного результату.

Пошукові алгоритми – для знаходження певних даних (наприклад, бінарний пошук, лінійний).

Сортувальні алгоритми – впорядковують елементи за певним критерієм (наприклад, сортування бульбашкою, швидке сортування та інші) [20, с.7].

У таблиці 3.1 наведено приклади алгоритмів, що використовуються в Java (на прикладі ArrayList).

Таблиця 3.1 – Приклади алгоритмів, що використовуються в Java

Операція	Алгоритм	Пояснення
Додавання елементу в кінець	Просте додавання + динамічне розширення	Якщо є місце - елемент додається. Якщо масив заповнений створюється новий масив із більшим розміром (зазвичай у 1,5 рази більший)
Пошук елементу	Лінійний пошук	При використанні contains() проходить по елементах послідовно.
Доступ за індексом	Прямий доступ	надає миттєвий доступ до будь-якого елемента за індексом.
Сортування	Сортування злиттям	list.sort(), використовується алгоритм MergeSort.

У рамках розробленої програми алгоритм розглядається як інструкція обчислювальних кроків, що перетворюють вхідні дані (запити від клієнта, дані про гравців) у вихідні результати (розрахована вартість, оновлений баланс команди, відсортовані списки). Оскільки програма реалізована мовою Java з використанням фреймворку Spring Boot, у ній застосовуються як базові алгоритми роботи зі структурами даних, так і спеціалізовані алгоритми бізнес-логіки.

Алгоритм розрахунку вартості трансферу є ключовим компонентом інтелектуальної системи оцінки вартості гравця. На відміну від фіксованих значень у баз даних, система динамічно обчислює ціну на основі поточних параметрів спортсмена. Цей алгоритм реалізовано в сервісному шарі (PlayerService). Даний алгоритм отримує вхідні дані (вік гравця та його досвід у місяцях), розрахунок комісії визначається частка команди-продавця (від 0% до 10%), валідація даних (наприклад, вік не може бути нульовим, щоб уникнути ділення на нуль). Цей алгоритм забезпечує фінансові операції і є однозначними при однакових вхідних даних.

Для зберігання та обробки списків команд і гравців у оперативній пам'яті (до моменту збереження в БД) використано структуру даних. У Java основною структурою для цього є ArrayList реалізації інтерфейсу List. Вибір цієї структури базується на необхідності частого доступу до елементів за індексом(номером елемента) та передачі даних у форматі JSON [10]. Таким чином, використання ArrayList є хорошим рішенням для даної системи, оскільки воно поєднує високу швидкість при читанні даних із простотою їх перетворення для обміну інформацією через REST API.

3.2 Математична модель трансферу

У процесі реалізації функціоналу трансферу гравців була розроблена математична модель, яка дозволяє обчислити вартість трансферу спортсмена з урахуванням кількості місяців ігрового досвіду, віку гравця та комісійного відсотка команди-продавця. Ця формула лягла в основу бізнес-логіки,

реалізованої на сервері, та забезпечує коректність фінансових розрахунків між клубами.

Вартість трансферу гравця знаходимо за формулою:

$$T = \frac{E \cdot 100000}{A} \quad (3.1)$$

Формула для комісії:

$$K = \frac{C}{100} \cdot T \quad (3.2)$$

Повна сума трансферу за формулою:

$$S = T + K \quad (3.3)$$

Оновлення балансів команд після трансферу:

— Баланс команди-покупця зменшується:

$$B1 = B1 - S \quad (3.4)$$

— Баланс команди-продавця збільшується:

$$B2 = B2 + S \quad (3.5)$$

Умови валідації:

— $B2 \geq S$ (у команди-покупця має бути достатньо коштів для трансферу)

— E – досвід гравця в місяцях;

— A – вік гравця в роках;

— C – комісійний відсоток команди-продавця (від 0 до 10%);

— B1 – баланс команди-продавця;

— B2 – баланс команди-покупця.

Висновки до розділу

У даному розділі було розглянуто математичне забезпечення програмного продукту, яке охоплює як алгоритмічну, так і обчислювальну частину. Зокрема, розглянуто основні види алгоритмів, такі як пошукові та сортувальні, що є фундаментом для ефективної обробки даних у програмуванні.

Крім того, представлено математичну модель для обчислення вартості трансферу гравця, що дозволяє точно моделювати фінансові операції між спортивними клубами. Формули враховують досвід та вік гравця, а також комісію команди-продавця, що забезпечує прозорість розрахунків. Валідаційні умови гарантують, що операції проводяться лише при наявності достатніх коштів. Таким чином, алгоритмічні методи та математичні моделі відіграють ключову роль у реалізації функціональності програми, забезпечуючи її ефективність, надійність та відповідність бізнес-логіці.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Середовище розробки

У процесі розробки проєкту було використано інтегроване середовище розробки IntelliJ IDEA від компанії JetBrains. Це потужне середовище спеціалізується на розробці застосунків на Java та підтримує широкий спектр інших мов програмування і фреймворків.

Рисунок 4.1 – Логотип IntelliJ

Однією з ключових переваг IntelliJ є зручність створення нових проєктів з необхідними бібліотеками та залежностями. Завдяки вбудованій підтримці таких інструментів, як Maven та Gradle, розробник може за кілька кліків налаштувати проєкт, обрати потрібні бібліотеки (наприклад, Spring Boot, Lombok тощо) та автоматично інтегрувати їх у структуру проєкту. Це значно прискорює старт розробки та зменшує кількість помилок, пов’язаних з ручним налаштуванням конфігурацій.

Вікно створення нового проєкту у середовищі IntelliJ IDEA можна побачити на рисунку 4.2.

Після надання назви, вибору папки, версії Java нам необхідно додати необхідні бібліотеки (рис. 4.3).

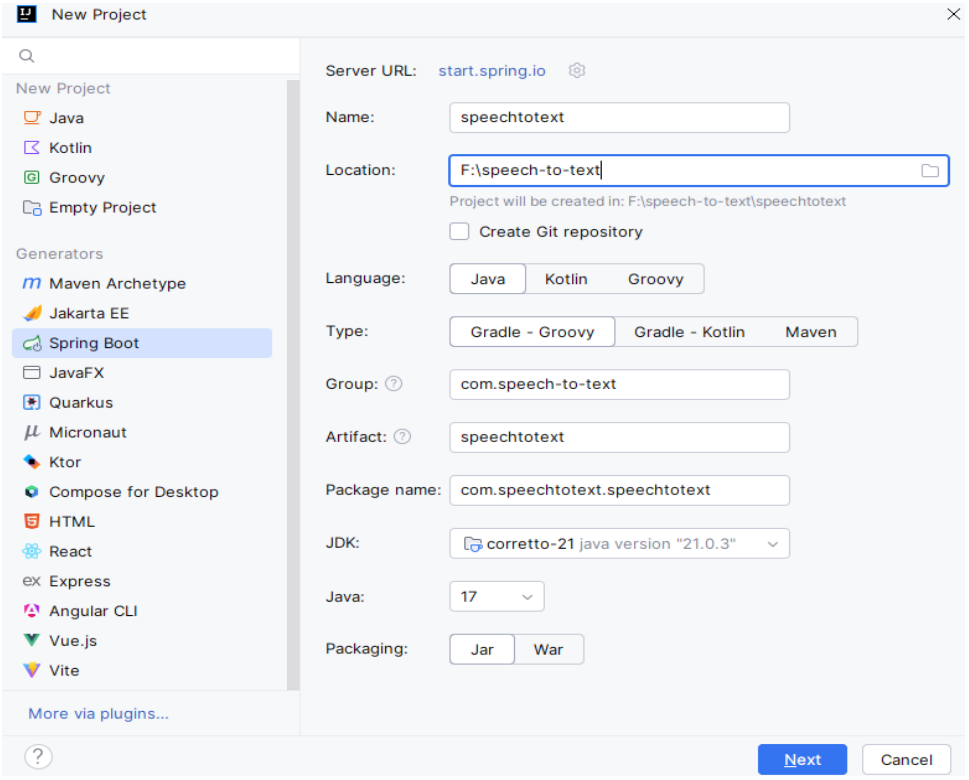

Рисунок 4.2 – Вікно створення нового проєкту

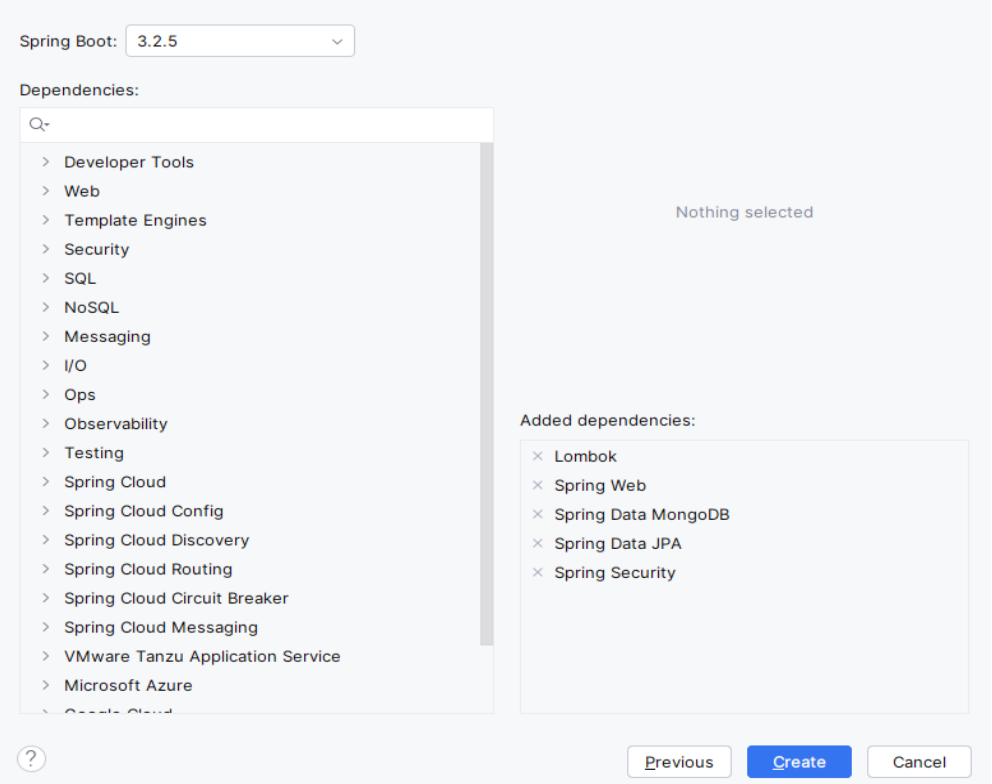

Рисунок 4.3 – Вікно проєкту з необхідними бібліотеками

Після вибору версії технології, середовище розробки створить структуру проєкту в якому можна починати роботу.

4.2 Опис розробки

Розробку нашого проєкту почнемо з моделей. Створимо два класи – Player та Team – і визначимо для них відповідні поля. У класі Player це, зокрема, firstName, lastName, id, birthDate тощо (див. рис. 4.4). Також встановимо зв’язок між класами за допомогою анотації @ManyToOne.

Рисунок 4.4 – Клас Player

У класі Team визначимо поля name, balance, id тощо (див. рис. 4.5). Зв’язок між класами встановимо @OneToMany.

Рисунок 4.5 – Клас Team

Після створення моделей перейдемо до реалізації рівня repository, щоб відокремити логіку взаємодії з базою даних. Для цього створимо два класи — PlayerRepository та TeamRepository.

У класі PlayerRepository реалізуємо методи для збереження гравця (savePlayer), оновлення даних (updatePlayer), видалення гравця з бази (deletePlayer) та отримання списку всіх гравців (getAllPlayers) (див. рисунок 4.6). У класі TeamRepository реалізуємо аналогічну логіку (див. рисунок 4.7).

Рисунок 4.6 – Логіка класу PlayerRepository

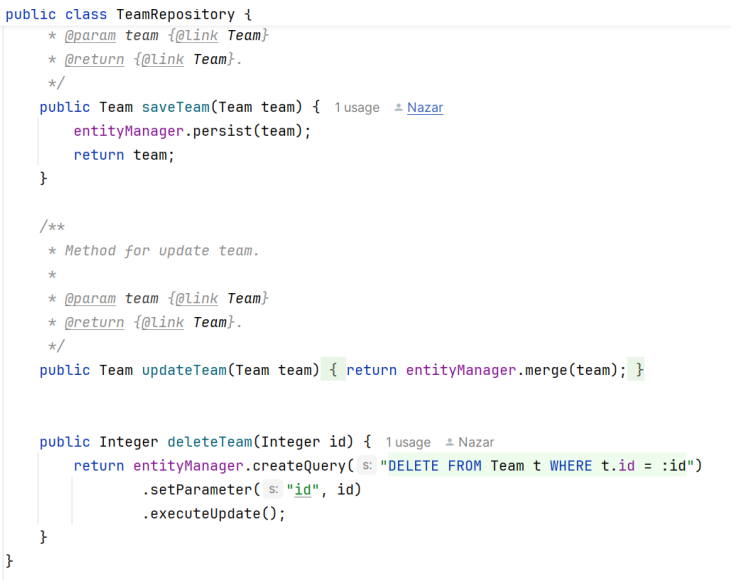

Рисунок 4.7 – Логіка класу TeamRepository

Після реалізації рівня repository перейдемо до створення сервісного рівня, який відповідатиме за бізнес-логіку застосунку. Для цього створимо два класи — PlayerService та TeamService. У класі PlayerService реалізуємо методи для: додавання нового гравця (addPlayer), оновлення інформації про гравця (updatePlayer), видалення гравця з системи (removePlayer), отримання списку всіх гравців (listAllPlayers) та методи для трансферу гравців (calculateTotalTransferCost, transferPlayer) в яких логіка підрахувань, обчислень для трансферу гравця враховуючи його досвід, вік, комісійні.

```
19    public class PlayerService {
124       public PlayerTeamDtoResponse transferPlayer(Integer playerId, Integer teamId) {  1 usage  Naza
127
128           Double totalTransferCost = calculateTotalTransferCost(player, toTeam);
129
130           if (toTeam.getBalance() < totalTransferCost) {
131               throw new InsufficientBalanceException(INSUFFICIENT_BALANCE_MESSAGE);
132           }
133
134           Team fromTeam = player.getTeam();
135           fromTeam.setBalance(fromTeam.getBalance() + totalTransferCost);
136           toTeam.setBalance(toTeam.getBalance() - totalTransferCost);
137
138           player.setTeam(toTeam);
139           playerRepository.savePlayer(player);
140           return mapToDto(player);
141       }
142
143 @     private Double calculateTotalTransferCost(Player player, Team team) {  1 usage  Nazar
144           double playerPrice = (player.getExperienceMonths() * 100000.0) / player.getAge();
145           double commission = playerPrice * (team.getCommissionPercentage() / 100);
146           double totalCost = playerPrice + commission;
147           return Math.round(totalCost * 100.0) / 100.0;
148       }
```

Рисунок 4.8 – Логіка класу PlayerService

У класі TeamService реалізуємо методи для: створення нової команди (createTeam), оновлення інформації про команду (updateTeam), видалення команди з системи (deleteTeam), отримання списку всіх команд (getAllTeams).

```
19    public class TeamService {
48        * @param id {@link Integer}
49        * @return {@link TeamPlayerDtoResponse}.
50        */
51       public TeamPlayerDtoResponse getTeam(Integer id) {  1 usage   Nazar
52           Team team = getTeamById(id);
53           return mapToTeamPlayerDto(team);
54       }
55
56       /**
57        * Method for create new team.
58        *
59        * @param teamDtoRequest {@link TeamDtoRequest}
60        * @return {@link TeamDtoResponse}.
61        */
62   @   public TeamDtoResponse createTeam(TeamDtoRequest teamDtoRequest) {  1 usage   Nazar +1
63           Team team = Team.builder()
64                   .name(teamDtoRequest.getName())
65                   .balance(teamDtoRequest.getBalance())
66                   .commissionPercentage(teamDtoRequest.getCommissionPercentage())
67                   .build();
68
69           Team createdTeam = teamRepository.saveTeam(team);
70           return mapToTeamDto(createdTeam);
71       }
```

Рисунок 4.9 – Логіка класу TeamService

Останній шар в архітектурі нашого додатку – це контролер. Контролери відповідатимуть за обробку HTTP-запитів і слугуватимуть зв’язковою ланкою між клієнтом і бізнес-логікою застосунку. Для цього створимо два класи – PlayerController та TeamController. Кожен із методів контролера делегуватиме обробку відповідного запиту сервісному.

Усі запити до контролерів та відповіді від них обмінюються даними у форматі JSON, що є зручним і широко використовуваним форматом для передачі інформації між клієнтом і сервером. При створенні або оновленні гравця чи команди клієнт надсилає JSON-об’єкт, який містить відповідні дані (наприклад, ім’я, позицію, назву команди тощо). Контролер отримує ці дані, перетворює їх у Java-об'єкти за допомогою анотацій @RequestBody, а далі передає їх на обробку сервісному рівню.

```java
public class PlayerController {
    */
    @PostMapping("/{playerId}/transfer/{teamId}")  Nazar
    public ResponseEntity<PlayerTeamDtoResponse> transferPlayer(
            @PathVariable Integer playerId,
            @PathVariable Integer teamId) {
        return ResponseEntity.ok(playerService.transferPlayer(playerId, teamId));
    }

    /**
     * The controller which returns all players.
     *
     * @return list of {@link PlayerTeamDtoResponse}.
     */
    @GetMapping  Nazar +1
    public ResponseEntity<List<PlayerTeamDtoResponse>> getAllPlayers() {
        return ResponseEntity.ok(playerService.getAllPlayers());
    }

    @GetMapping("/{id}")  Nazar
    public ResponseEntity<PlayerTeamDtoResponse> getPlayer(@PathVariable Integer id) {
        return ResponseEntity.ok(playerService.getPlayer(id));
    }
```

Рисунок 4.10 – PlayerController

```java
public class TeamController {
    */
    @PostMapping  Nazar +1
    public ResponseEntity<TeamDtoResponse> createTeam(@Valid @RequestBody TeamDtoReques
        return ResponseEntity.ok(teamService.createTeam(teamDtoRequest));
    }

    @PatchMapping("/{id}")  Nazar
    public ResponseEntity<TeamDtoResponse> updateTeam(
            @PathVariable Integer id,
            @Valid @RequestBody TeamDtoRequest teamDtoRequest) {
        return ResponseEntity.ok(teamService.updateTeam(id, teamDtoRequest));
    }

    /**
     * The controller which delete team.
     *
     * @param id {@link Integer}
     * @return {@link String}.
     */
    @DeleteMapping("/{id}")  Nazar
    public ResponseEntity<String> deleteTeam(@PathVariable Integer id) {
        return ResponseEntity.ok(teamService.deleteTeam(id));
    }
}
```

Рисунок 4.11 – TeamController

У проєкті також використовується підхід із DTO (Data Transfer Object) – спеціальними класами, які призначені для передачі даних між шарами застосунку, зокрема між контролером і сервісом [17, с.114]. Це є хорошою практикою. Наприклад, замість того, щоб надсилати клієнту повну сутність Player, ми можемо створити окремий клас PlayerDto, який містить лише потрібні для відображення дані, такі як ім'я гравця, позиція та назва команди. Такий підхід робить проєкт чистішим, безпечнішим і гнучкішим до змін у майбутньому.

```java
package com.football_manager.dto.response;

import lombok.Builder;
import lombok.Data;

@Data  15 usages  Nazar +1
@Builder
public class TeamDtoResponse {

    private Integer id;

    private String name;

    private Double balance;

    private Double commissionPercentage;
}
```

Рисунок 4.12 – Приклад використання DTO у проєкті

4.3 Liquibase в проєкті

Так як ми вже згадували, що liquibase це інструмент для керування версіями бази даних та за допомогою нього в майбутньому проєкт можна буде масштабувати та легко вносити зміни у структуру БД без втрати даних. Для того, щоб додати liquibase в проєкт потрібно додати залежність в файл pom.xml (рис. 4.13).

```xml
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
</dependency>
```

Рисунок 4.13 – Залежність в файлі pom.xml

Після додавання залежності, можна керувати нашою базою даних. Створимо таблиці, валідацію та заповнемо її значенням. Приклад з створенням таблиці на рисунку 4.14. Приклад заповнення таблиці значеннями на рисунку 4.15.

```sql
--liquibase formatted sql

-- changeset nazar:create-teams-table
CREATE TABLE teams
(
    id                    SERIAL PRIMARY KEY,
    name                  VARCHAR(255),
    balance               NUMERIC(10, 2),
    commission_percentage NUMERIC(5, 2)
);

-- changeset nazar:create-player-table
CREATE TABLE players
(
    id                SERIAL PRIMARY KEY,
    first_name        VARCHAR(255),
    last_name         VARCHAR(255),
    birth_date        DATE,
    experience_months INTEGER,
    team_id           INTEGER,
    CONSTRAINT fk_team FOREIGN KEY (team_id) REFERENCES teams (id) ON DELETE SET NULL
);
```

Рисунок 4.14 – Створення таблиці з валідацією

```sql
INSERT INTO teams (name, balance, commission_percentage)
VALUES
    ( name 'Liverpool', balance 1000000, commission_percentage 9),
    ( name 'Arsenal', balance 1200000, commission_percentage 9),
    ( name 'Nottingham Forest', balance 400000, commission_percentage 6),
    ( name 'Chelsea', balance 2000000, commission_percentage 9),
    ( name 'Newcastle', balance 600000, commission_percentage 7),
    ( name 'Manchester City', balance 2202000, commission_percentage 10),
    ( name 'Bournemouth', balance 402600, commission_percentage 6),
    ( name 'Aston Villa', balance 806300, commission_percentage 7),
    ( name 'Fulham', balance 553000, commission_percentage 5),
    ( name 'Tottenham', balance 1156200, commission_percentage 8);

-- changeset nazar:insert-players-data
INSERT INTO players (first_name, last_name, birth_date, experience_months, team_id)
VALUES
    ( first_name 'Mohamed', last_name 'Salah', birth_date '1992-07-15', experience_months 30, team_id 1),
    ( first_name 'William', last_name 'Saliba', birth_date '1995-02-20', experience_months 15, team_id 2),
    ( first_name 'Matz', last_name 'Sels', birth_date '1989-08-25', experience_months 26, team_id 3),
    ( first_name 'Cole', last_name 'Palmer', birth_date '1996-06-30', experience_months 24, team_id 4),
    ( first_name 'Alexander', last_name 'Isak', birth_date '1996-11-10', experience_months 12, team_id 5),
    ( first_name 'Erling', last_name 'Haaland', birth_date '1998-10-17', experience_months 13, team_id 6),
    ( first_name 'Illia', last_name 'Zabarnyi', birth_date '1999-12-20', experience_months 15, team_id 7),
```

Рисунок 4.15 – Заповнення таблиці даними

4.4 Тестування проєкту

Для перевірки коректності роботи нашого REST API було використано Postman – інструмент для відправки HTTP-запитів та аналізу відповідей. Postman дозволяє автоматизувати тестування, документувати API та співпрацювати з командою. На рисунку 4.16 зображено запит для отримання списку гравців. На рисунку 4.17 зображено успішний трансфер між командами.

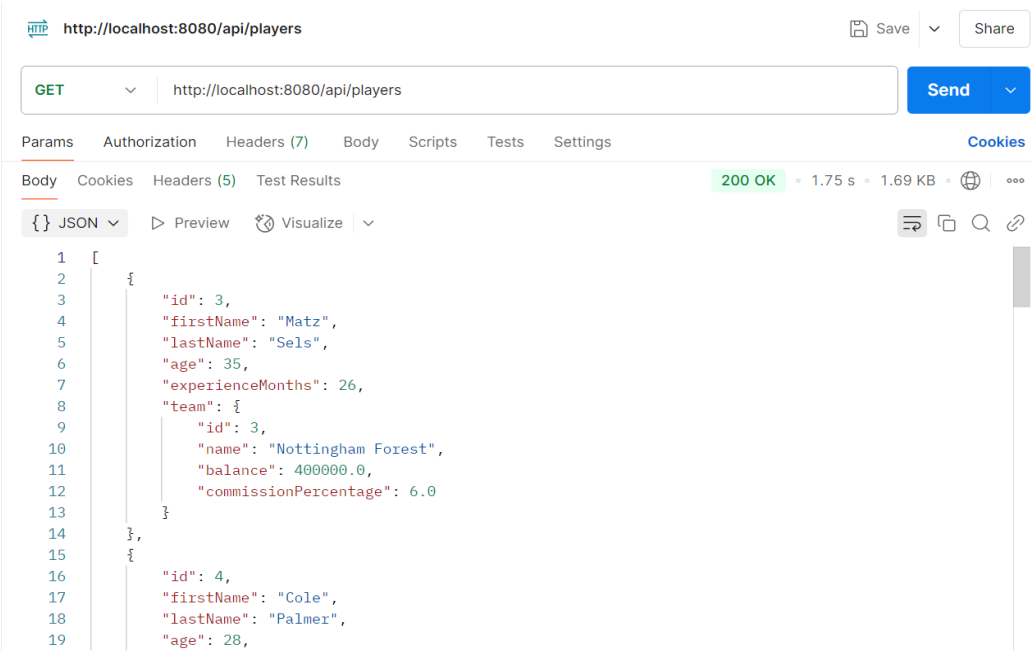

Рисунок 4.16 – Запит для отримання списку гравців

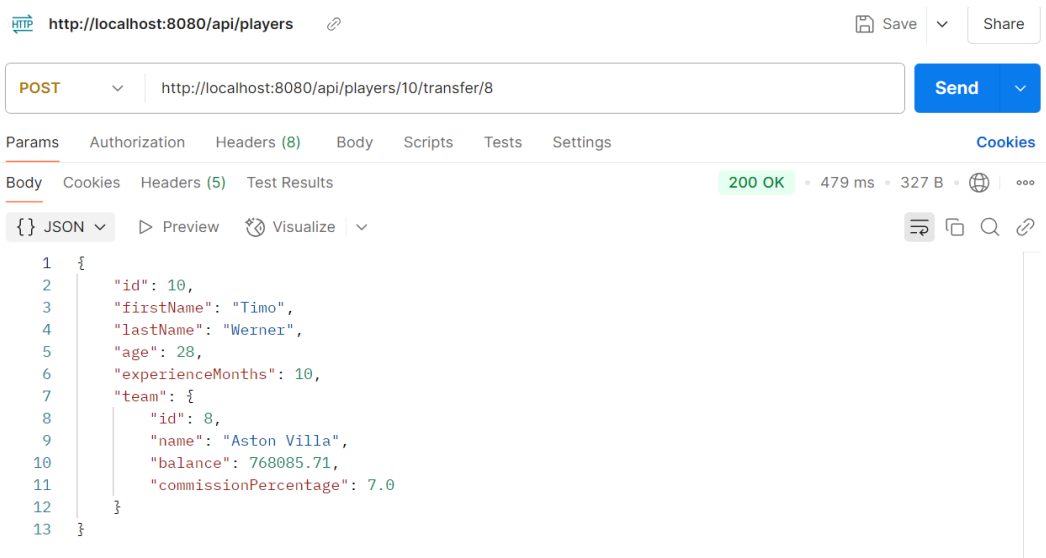

Рисунок 4.17 – Успішний трансфер гравця

Для перевірки ціни гравця за досвідом створили 3-ох гравців з різним досвідом та провели трансфер гравців рис. 4.18, рис. 4.19.

	id	first_…	last_n…	birth_…	experi…
1	1	Mohamed	Salah	1992-07-15	30
2	4	Cole	Palmer	1996-06-30	24
3	6	Erling	Haaland	1998-10-17	13

Рисунок 4.18 – Збережені гравці

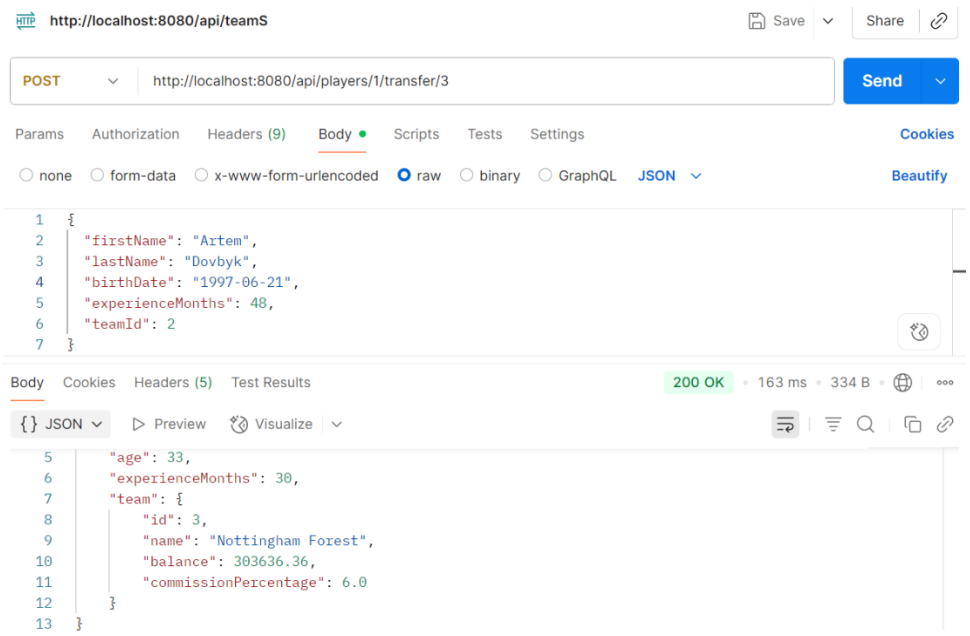

Рисунок 4.19 – Проведення трансферу

На основі цього бачимо, що гравці з більшим досвідом оцінюються дорожче, як і планувалося за початковим задумом. Графік порівнянь гравців за досвідом та ціною рис. 4.20 та 4.21

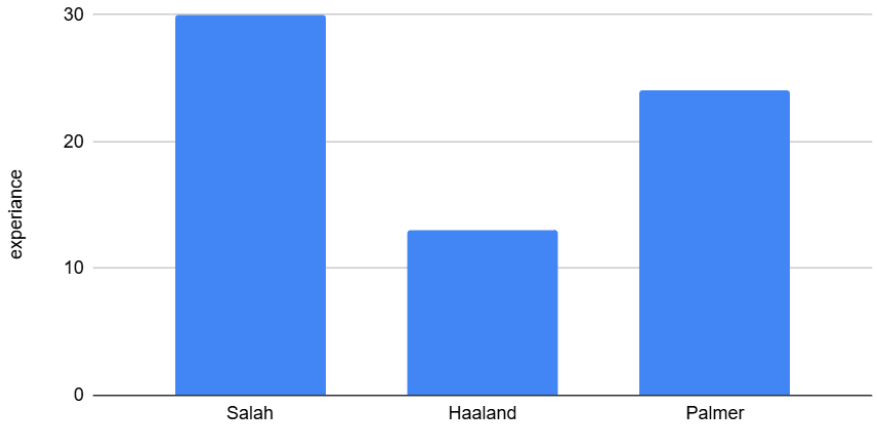

Рисунок 4.20 – Графік досвіду гравців у місяцях

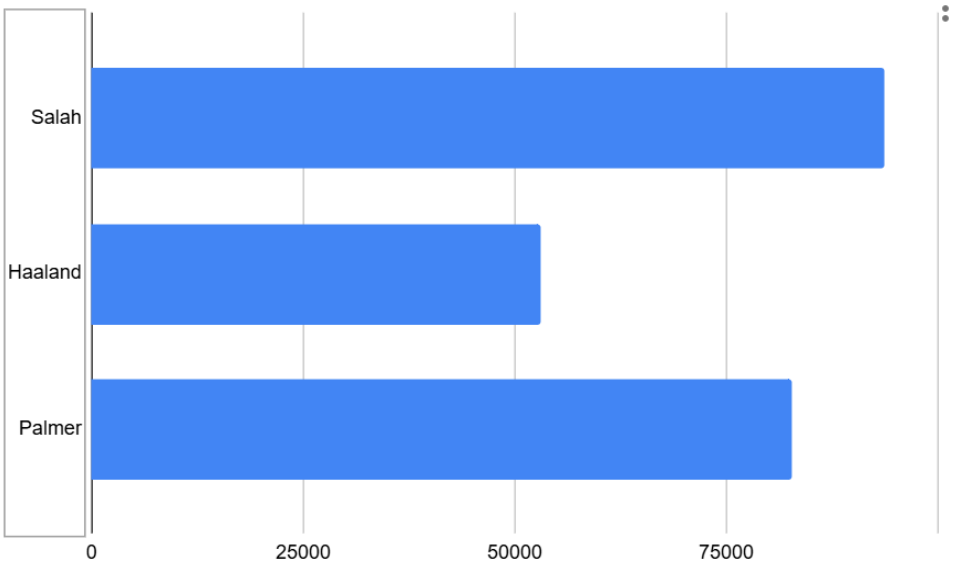

Рисунок 4.21 – Графік вартості гравців

Висновки до розділу

У даному розділі було детально розглянуто процес розробки програмного забезпечення для нашого проєкту, починаючи із налаштування середовища розробки та закінчуючи тестуванням готового API.

Використано Spring Boot для спрощення конфігурації та інтеграції з базою даних. Інструмент Liquibase дозволив керувати міграціями БД, забезпечуючи контроль версій та легке оновлення схеми без втрати даних. Налаштовано валідацію полів та початкове наповнення таблиць даними. Перевірено коректність відповідей та обробку помилок.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Проєкт "Інтелектуальна сиестема менеджменту спортивних команд" – це програмне рішення для керування спортивними командами та гравцями, яке автоматизує ключові процеси такі як: створення, редагування та видалення даних про команди та гравців, проведення трансфертних операцій з автоматичним розрахунком вартості, валідацію операцій для запобігання помилкам (наприклад, недостатньо коштів у команди). Система реалізована на базі Spring Boot з використанням Hibernate для роботи з базою даних, що забезпечує масштабованість та легку інтеграцію з іншими сервісами.

Хоча система розроблена для спортивних команд (на прикладі футбольних команд), її архітектура та логіка дозволяють адаптувати її для інших сфер:

- інші види спорту (баскетбол, хокей, волейбол);
- корпоративні рішення (управління персоналом у компаніях з аналогічними процесами "трансферів", наприклад, переведення співробітників між відділами з розрахунком компенсацій);
- бізнес-сфера (продаж автомобілів між салонами, операції з квартирами між агентствами).

Розроблена система має ряд суттєвих переваг, які роблять її гнучким та масштабованим рішенням. Основною силою проєкту є його модульна архітектура, яка побудована на чіткому розділенні логіки. Такий підхід дозволяє безболісно розширювати функціонал – наприклад, додавати нові модулі, не вносячи змін у вже існуючий код. Особливу увагу варто приділити механізму розрахунку трансферів, який реалізований з можливістю легкої модифікації базової формули - це відкриває широкі перспективи для адаптації системи під різні бізнес-вимоги, якщо знадобиться враховувати додаткові фактори. Важливою перевагою є відкритий REST API інтерфейс, що забезпечує зручну інтеграцію з будь-якими сучасними фронтенд-рішеннями,

будь то вебдодатки, мобільні застосунки або інші зовнішні сервіси. Ця особливість значно розширює потенційні сфери застосування системи.

5.2 Розроблення ринкової стратегії

Розробка ефективної ринкової стратегії для програмного продукту є ключовим етапом його успішного впровадження на ринок. Дана система, має ширший потенціал застосування завдяки своїй гнучкій модульній архітектурі. Вона може бути успішно адаптована для різних бізнес-секторів, включаючи управління персоналом, нерухомістю, автосалонами та інші сфери, де необхідно керування ресурсами.

Ринкова стратегія для нашої розробки спрямована на поетапне виведення інтелектуальної системи на ринок цифрових сервісів для спортивного менеджменту з подальшим масштабуванням на суміжні галузі. Основна мета нашої стратегії – формування стабільної клієнтської бази, досягнення впізнаваності бренду (який додатково необхідно розробити) та забезпечення фінансової стійкості продукту в середньостроковій перспективі.

Для ефективного просування продукту планується використовувати комплекс маркетингових інструментів. У цифровому маркетингу основний акцент буде зроблений на рекламі в соціальних мережах (LinkedIn, Facebook та Instagram), контент-маркетингу (корисні статті).

Важливим напрямком розвитку продукту є постійне вдосконалення його функціоналу. У перспективі планується розширення системи за рахунок додавання нових модулів аналітики, звітності, а також інтеграції інструментів штучного інтелекту. Для забезпечення високої якості продукту та його відповідності потребам клієнтів передбачається регулярне проведення опитувань та збір зворотного зв'язку.

Реалізація даної стратегії, поєднана з постійним моніторингом ринкових потреб та вдосконаленням функціоналу, створює міцну основу для успішного позиціонування продукту та забезпечення його довгострокової конкурентоспроможності на ринку програмних рішень для бізнесу.

Аналіз ринку. Цифрові рішення для спортивного менеджменту належать до sports-tech сегменту, якому притаманне стабільне зростання внаслідок низького поточного рівня цифровізації операційної діяльності в спорті.

Тенденції ринку, такі як інтеграція аналітики та ШІ, зростання сектору кіберспорту, відкритість спортивної галузі до цифровізації, а також поточний стан вітчизняного ринку цифрових систем для спорту створює сприятливі умови для входження нового продукту у практично низькоконкурентне середовище.

Цільова аудиторія стартап-проєкту. На початковому етапі впровадження основними споживачами розробленої системи можуть бути користувачі із обмеженими фінансовими ресурсами, які перебувають у пошуку доступного цифрового рішення:

— аматорські футбольні клуби;

— спортивні школи та академії;

— приватні тренери та напівпрофесійні ліги;

— кіберспорту;

— спортивні менеджери та адміністратори;

— агенти спортивних талантів.

Такий контингент клієнтів є зручним для подальшого тестування та вдосконалення розробленої системи, адже опираючись на їх досвід мікроменеджменту, запити та побажання, в розробку у подальшому можна масштабувати на великі спортивні корпорації.

Позиціонування продукту на ринку. Проєкт "Інтелектуальна система менеджменту спортивних команд" позиціонується як доступна платформа для управління командами, інтелектуальна система фінансового контролю, універсальне рішення для різних видів спорту та продукт із відкритою серверною архітектурою. Тому, основною конкурентною перевагою є автоматизована фінансова модель трансферів, відсутня у більшості конкурентів.

Конкурентний аналіз. Основні міжнародні конкуренти стартапу – TeamSnap, SportEasy та PlayerTek, які характеризуються високою вартістю, орієнтацією на професійні клуби, відсутністю гнучкого API та відсутністю фінансової моделі трансферів.

У порівнянні із ними розроблена інтелектуальна система має низьку вартість, підтримку трансферних операцій, REST API, можливість локалізації та гнучку модульність.

Стратегія виходу на ринок передбачатиме 3 орієнтовні етапи:

1. Етап запуску (до 6 місяців)
   — запуск MVP-версії та безкоштовне підключення перших 50 клієнтів;
   — збір зворотного зв'язку та усунення недоліків;
   — формування кейсів використання.
2. Етап зростання (до 1,5 року)
   — запуск тарифних планів та рекламної кампанії;
   — залучення партнерів;
   — вихід на ринок кіберспорту.
3. Етап масштабування (після 1,5 року)
   — підтримка кількох мов;
   — інтеграція ШІ;
   — створення мобільного застосунку тощо.

Цінова стратегія. Для нашого стартапу, зважаючи на особливості та платоспроможність вітчизняного ринку, обрана стратегія доступної підписки для швидкого залучення клієнтів. А саме, пропонуємо наступні тарифні плани:

— *Free* (0€/міс, до 10 гравців) – рішення для перших 100 клієнтів (фізичні особи для індивідуального користування);
— *Pro* (10€/міс, до 50 гравців, трансфери)– рішення для аматорських спортивних клубів;
— *Business* (25€/міс, необмежено, аналітика) – для професійних спортивних асоціацій та компаній;

— *Enterprise* (ціна договірна, пакет+, кастомні модулі) – для великих спортивних компаній та клубів;

— *додаткові джерела прибутку:* платні API-інтеграції, кастомні модулі під замовлення, реклама спортивних брендів / колаборації, white-label рішення для федерацій спорту тощо.

Орієнтовний фінансовий план на 1й рік впровадження стартапу. Зважаючи на весь комплекс обставин, що заставляють представників вітчизняного бізнесу використовувати модифіковані технічні рішення для своєї цифрової інфраструктури, ми оцінили стартові фінансові витрати та потоки впродовж першого року запуску стартапу.

Таблиця 5.1 – Фінансовий план стартапу (1й рік)

Стаття витрат	Сума, €
Серверна інфраструктура	1200
Маркетинг	2000
Домен + SSL	200
Підтримка	800
Разом витрати	*4200*

Очікуваний дохід:

100 клієнтів х 10 €/міс х 10 міс = 12000 €

Окупність стартових інвестицій: *6-8 місяців.*

Оцінка ризиків ринкової стратегії. Дослідивши особливості ринку, передбачаємо низку ризиків на шляху впровадження стартапу. Зокрема, за умов низького попиту вважаємо за доцільне адаптувати тарифи, а також запропонувати гнучкі програми лояльності у випадку відтоку клієнтів. Також, у випадку змін на ринку (поява конкурентів чи продуктів-аналогів), вважаємо за потребу дотримуватися стратегії постійного оновлення та розширення функціоналу інтелектуальної системи.

5.3. Розроблення маркетингової програми

Метою розроблення маркетингової програми нашого стартапу є забезпечення стійкого зростання кількості користувачів платформи та досягнення позитивного фінансового результату вже у перший рік функціонування заради досягнення окупності заявленої у попередньому підрозділі.

Працюючи над програмою ми переслідували кілька основних завдань, серед яких якісне інформування потенційних клієнтів щодо нашої розробки, створення бренду та формування його позитивного іміджу, стимулювання переходу Free клієнтів на розширені платні тарифи та, безумовно, утримання клієнтів.

Маркетингова програма для проєкту будується на стратегії поступового залучення аудиторії. Тому на ***першому (підготовчому) етапі*** (1-2 місяці) варто створити лендинг-сайт проєкту та запустити комунікацію через соціальні мережі (Facebook, Instagram, Telegram), де будуть публікуватися оновлення, анонси функцій та інтерактивні опитування. Це дозволить сформувати ком'юніті, залучити бета-тестувальників і отримати зворотний зв’язок для доопрацювання продукту. Очевидною є необхідність підготовки презентаційних матеріалів, відеороликів та реклами, запуск аналітики (наприклад Google Analytics) для відслідковування прогресу маркетингової кампанії. Орієнтовний бюджет заходів – 200 €.

На ***етапі запуску*** (3-6 місяці) ключовим завданням стане масова рекламна кампанія через:

— таргетовані оголошення у соцмережах (для тренерів, менеджерів клубів, спортивних ентузіастів);

— YouTube-огляди для тренерів і спортивних менеджерів;

— Telegram-боти для спортивних клубів;

— публікації в спортивних спільнотах (форуми, чати);

— участь в спортивних заходах та партнерство із спортивними школами/гуртками, спонсорство;

— збір зворотного зв’язку.

Для підвищення лояльності варто розробити програму рефералів (наприклад, бонуси за запрошення друзів) та обмежені пропозиції для перших користувачів. Цей етап традиційно найважчий, адже вимагає попри все фінансових витрат. Орієнтовний бюджет заходів – 900 €.

Фінальним ***етапом (зростання)*** (6-12 місяців) може стати масштабування реклами, започаткування партнерських програм з реальними клубами, що перетворить проєкт застосунку на повноцінний інструмент для спортивного менеджменту, а також старий-добрий email-маркетинг. Гнучкість та постійний діалог з аудиторією стануть ключем до тривалого успіху. Орієнтовний бюджет заходів – 900 €.

При оптимістичному сценарії розвитку подій, орієнтовний бюджет на маркетингову програму становитиме 2000 €.

За таких витрат варто дотримуватися чіткого плану дій та відслідковувати основні KPI проєкту, серед яких пропоновано орієнтуватися на наступні значення:

>= 500 реєстрацій впродовж року;

>= 100 платних кореистувачів;

>= 20% конверсій між тарифними планами;

>= 10 €/міс середній дохід із клієнта;

<= 10% річний відтік клієнтів.

Щоб витримати вказані KPI стартапу варто окремо дотримуватися програми ***заходів стимулювання продажів***, пропонуючи клієнтам безкоштовний пробний період, знижки для спортивних шкіл та гуртків, бонуси за рекомендації та знижки за умови річної оплати тарифу.

Проте найбільш ефективним, на наш погляд, буде впровадження ***стратегії утримання клієнтів,*** що послугує додатковим промоційним

заходом: дотримуємося ідеї регулярного оновлення функціоналу інтелектуальної системи, запровадження служби технічної підтримки та практики персональних консультацій та автоматизованих підказок в системі, а також проведення навчальних вебінарів та оффлайн презентацій продукту.

Запропонована маркетингова система забезпечує поетапне та кероване виведення стартапу на ринок із використанням сучасних цифрових каналів просування. Чітке планування бюджету, контроль показників ефективності та комплексний підхід до залучення й утримання клієнтів створять передумови для досягнення запланованих економічних результатів уже в перший рік реалізації інтелектуальної системи.

Висновок до розділу

У даному розділі представлено комплексний підхід до створення та впровадження інноваційного продукту – "Інтелектуальної системи менеджменту спортивних команд". Проєкт відзначається унікальною гнучкістю архітектури, що дозволяє його адаптацію не лише у спортивній сфері (футбол, баскетбол, хокей), але й у таких бізнес-напрямках, як управління персоналом, автоторгівля та нерухомість. Ключовою перевагою системи є її модульність, що забезпечує легке масштабування та інтеграцію з іншими сервісами через REST API. Ринкова стратегія проєкту будується на гнучкому поєднанні цифрових маркетингових інструментів (таргетована реклама, контент-маркетинг) з прямими комунікаціями з цільовою аудиторією. Реалізація запропонованого підходу створює міцну основу для успішного позиціювання розробленої системи на ринку.

ВИСНОВКИ

У даній магістерській роботі було розроблено інтелектуальну систему менеджменту спортивних команд, яка спрощує процеси управління командами, гравцями та фінансовими операціями, зокрема трансферами.

Система реалізована на базі сучасних технологій, таких як Spring Boot, Hibernate, PostgreSQL та Liquibase, що забезпечило її масштабованість, гнучкість та легкість інтеграції з іншими сервісами. Використання Spring Boot спростило створення програми для користувачів, забезпечуючи зручний доступ до функціоналу додатку.

У процесі розробки було реалізовано наступні ключові функціональні можливості: механізм трансферів з автоматичним розрахунком вартості на основі віку, досвіду гравця та комісії команди, валідацію операцій, що запобігає помилкам, пов’язаним із недостатнім балансом або некоректними даними, перегляд усіх команд та гравців, видалення, оновлення, збереження даних для команд та гравців. Також варто зазначити, гнучкість системи оскільки вона має універсальну архітектуру, яка дозволяє адаптувати її не лише для спортивних клубів, але й для інших сфер, таких як управління персоналом, нерухомістю чи автосалонами з невеликою зміною додатка.

Практичне значення роботи полягає у можливості безпосереднього використання розробленої системи в діяльності спортивної галузі. Запропонований стартап-проєкт підтверджує економічну доцільність та ринкову перспективність програмного продукту.

Розроблена система є сучасним, гнучким та масштабованим рішенням для управління спортивними командами, яке може бути успішно адаптоване для різних галузей бізнесу. Подальший розвиток проєкту сприятиме його вдосконаленню та розширенню сфер застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hibernate [Електронний ресурс] – Режим доступу: https://hibernate.org/orm/ (дата звернення: 18.05.2025).
2. JavaScript Object Notation (JSON) [Електронний ресурс] – Режим доступу: https://jsonapi.org/ (дата звернення: 15.05.2025).
3. Liquibase [Електронний ресурс] – Режим доступу: https://docs.liquibase.com/concepts/changelogs/home.html (дата звернення: 20.05.2025).
4. PostgreSQL [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/current/ (дата звернення: 19.05.2025).
5. Postman навчальний центр [Електронний ресурс] – Режим доступу: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 18.05.2025).
6. Spring Boot [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-boot#learn (дата звернення: 14.05.2025).
7. Spring Boot Build Project [Електронний ресурс] – Режим доступу: https://start.spring.io/ (дата звернення: 18.05.2025).
8. Spring Documentation [Електронний ресурс] – Режим доступу: https://devdocs.io/spring_boot/ (дата звернення: 12.05.2025).
9. SQL [Електронний ресурс] – Режим доступу: https://www.w3schools.com/sql/ (дата звернення: 13.05.2025).
10. Udemy: Spring 5 with Spring Boot 2 [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/spring-5-with-spring-boot-2/ (дата звернення: 13.05.2025).
11. Ґетц Б. Java Concurrency на практиці / Б. Ґетц. – 2022. – 432 с.
12. Майерс Ґ. Мистецтво тестування програм / Ґ. Майерс. – 2020. – 272 с.
13. Мартін Р. Чистий код. — Київ: Діалектика, 2019. — 464 с.
14. Річардсон Л. Робота з веб-інтерфейсами / Л. Річардсон. – 2013. – 406 с.
15. Седжвік Р. Алгоритми / Р. Седжвік. – 4-те вид. – 2021. – 557 с.

16. Тудосе К. JUnit в дії / К. Тудосе. – 3-тє вид. – 2023. – 560 с.
17. Тудосе К., Бауер К. Java з даними Spring та Hibernate. — Харків: Видавництво Фабула, 2023. — 512 с.
18. Уоллс К. Спрінг в дії. — Київ: Наш Формат, 2022. — 560 с.
19. Фрімен Е., Патерни проектування. — Київ: Діалектика, 2020. — 694 с.
20. Хайнеман Дж. Алгоритми навчання: написання кращого коду. — Львів: Видавництво Старого Лева, 2021. — 384 с.

ДОДАТКИ

Додаток А. Лістинги коду розроблених контролерів

```
@RestController
@RequestMapping("/api/teams")
public class TeamController {
   private final TeamService teamService;
   @Autowired
   public TeamController(TeamService teamService) {
       this.teamService = teamService;
   }
   @GetMapping
   public ResponseEntity<List<TeamPlayerDtoResponse>> getAllTeams() {
       return ResponseEntity.ok(teamService.getAllTeams());
   }
   @GetMapping("/{id}")
   public ResponseEntity<TeamPlayerDtoResponse> getTeam(@PathVariable Integer
id) {
       return ResponseEntity.ok(teamService.getTeam(id));
   }
   @PostMapping
   public ResponseEntity<TeamDtoResponse> createTeam(@Valid @RequestBody
TeamDtoRequest teamDtoRequest) {
       return ResponseEntity.ok(teamService.createTeam(teamDtoRequest));
   }
   @PatchMapping("/{id}")
   public ResponseEntity<TeamDtoResponse> updateTeam(
           @PathVariable Integer id,
           @Valid @RequestBody TeamDtoRequest teamDtoRequest) {
       return ResponseEntity.ok(teamService.updateTeam(id, teamDtoRequest));
   }
   @DeleteMapping("/{id}")
   public ResponseEntity<String> deleteTeam(@PathVariable Integer id) {
       return ResponseEntity.ok(teamService.deleteTeam(id));
   }
```

Додаток Б. Лістинги коду Service сторінки

```
@Service
@Transactional
public class TeamService {
   private final String TEAM_NOT_FOUND_MESSAGE = "The team does not exist by
this id: ";
   private final String TEAM_DELETED_MESSAGE = "Team deleted successfully";
   private final TeamRepository teamRepository;
   @Autowired
   public TeamService(TeamRepository teamRepository) {
       this.teamRepository = teamRepository;
   }
   public List<TeamPlayerDtoResponse> getAllTeams() {
       List<Team> teams = teamRepository.getAllTeams();
       return teams.stream()
               .map(this::mapToTeamPlayerDto)
               .collect(Collectors.toList());
   }
   public TeamPlayerDtoResponse getTeam(Integer id) {
       Team team = getTeamById(id);
       return mapToTeamPlayerDto(team);
   }
   public TeamDtoResponse createTeam(TeamDtoRequest teamDtoRequest) {
       Team team = Team.builder()
               .name(teamDtoRequest.getName())
               .balance(teamDtoRequest.getBalance())
               .commissionPercentage(teamDtoRequest.getCommissionPercentage())
               .build();

       Team createdTeam = teamRepository.saveTeam(team);
       return mapToTeamDto(createdTeam);
   }
   public TeamDtoResponse updateTeam(Integer id, TeamDtoRequest
teamDtoRequest) {
       Team team = getTeamById(id);

       Team teamToBeUpdated = team.toBuilder()
               .name(teamDtoRequest.getName())
```

```
                .balance(teamDtoRequest.getBalance())
                .commissionPercentage(teamDtoRequest.getCommissionPercentage())
                .build();

        Team updatedTeam = teamRepository.updateTeam(teamToBeUpdated);
        return mapToTeamDto(updatedTeam);
    }

    public String deleteTeam(Integer id) {
        Integer deletedTeam = teamRepository.deleteTeam(id);
        if (deletedTeam == 0) {
            throw new IdNotFoundException(TEAM_NOT_FOUND_MESSAGE + id);
        } else {
            return TEAM_DELETED_MESSAGE;
        }
    }

    public Team getTeamById(Integer id) {
        return teamRepository.getTeamById(id)
                .orElseThrow(() -> new
IdNotFoundException(TEAM_NOT_FOUND_MESSAGE + id));
    }
    private TeamDtoResponse mapToTeamDto(Team team) {
        return TeamDtoResponse.builder()
                .id(team.getId())
                .name(team.getName())
                .balance(team.getBalance())
                .commissionPercentage(team.getCommissionPercentage())
                .build();
    }
    private TeamPlayerDtoResponse mapToTeamPlayerDto(Team team) {
        return TeamPlayerDtoResponse.builder()
                .id(team.getId())
                .name(team.getName())
                .balance(team.getBalance())
                .commissionPercentage(team.getCommissionPercentage())
                .players(mapPlayersToDto(team.getPlayers()))
                .build();
```

```
    }
    private List<PlayerResponse> mapPlayersToDto(List<Player> players) {
        return players.stream()
                .map(player -> PlayerResponse.builder()
                        .id(player.getId())
                        .firstName(player.getFirstName())
                        .lastName(player.getLastName())
                        .age(player.getAge())
                        .experienceMonths(player.getExperienceMonths())
                        .build())
                .collect(Collectors.toList());
    }
}
```