

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

**ПОГОДЖЕНО
Декан факультету**

_____ **Ігор БОЛБОТ**
(підпис)

«___» _____ 2026 р.

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук**

_____ **Белла ГОЛУБ**
(підпис)

«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система обліку кіберспортивних турнірів»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор

_____ **Роман РУДЕНСЬКИЙ**
(підпис)

**Керівник бакалаврської
кваліфікаційної роботи**
к.т.н., доцент

_____ **Ірина БОРОДКІНА**
(підпис)

Виконав

_____ **Сергій ЧЕХ**
(підпис)

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

**ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук**

К.Т.Н., доцент _____

(підпис)

Белла ГОЛУБ

«__» _____ 2026 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Чеху Сергію Олександровичу

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Інформаційна система обліку кіберспортивних турнірів»

затверджена наказом від «06» січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проекту): вихідний код мобільного Flutter-застосунку, NestJS API, MongoDB-схеми, Docker-конфігурація, тестові дані CS2 HLTV Top 10, методичні вимоги кафедри комп'ютерних наук.

Перелік питань, що підлягають дослідженню:

Перелік питань, що підлягають дослідженню:

- проаналізувати предметну область кіберспортивних баз даних і турнірного менеджменту;
- сформулювати функціональні та нефункціональні вимоги до клієнт-серверної системи;
- розробити модель даних для користувачів, гравців, команд, турнірів, матчів, повідомлень і модераційних об'єктів;
- реалізувати REST API, realtime-оновлення, рольову модель доступу та мобільний клієнт;
- описати алгоритми пошуку, генерації турнірної сітки, оновлення матчів і повідомлень;
- провести перевірку працездатності та сформулювати рекомендації щодо впровадження.

Перелік графічних документів: структурна схема системи, модель даних, схема REST/realtime-взаємодії, діаграма сценаріїв ролей, схема розгортання, скріншоти основних екранів.

Дата видачі завдання « 06 » січня 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Ірина БОРОДКІНА

Завдання взяв до виконання

(підпис)

Сергій ЧЕХ

ЗМІСТ

ЗМІСТ.....	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	6
ВСТУП.....	7
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Постановка завдання.....	10
1.2 Аналіз предметної області.....	11
1.3 Огляд існуючих рішень.....	14
1.4 Функціональні та нефункціональні вимоги.....	17
1.5 Моделювання предметної області.....	18
1.6 Ролі користувачів та межі відповідальності.....	20
1.7 Користувацькі історії та критерії приймання.....	21
1.8 Межі проєкту та припущення.....	23
Висновки до розділу 1.....	24
2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ.....	25
2.1 Загальна модель даних.....	25
2.2 Обґрунтування вибору СУБД.....	26
2.3 Опис основних сутностей.....	27
2.4 Довідники, статуси та контроль допустимих значень.....	29
2.5 Індексація та пошук.....	31
2.6 Обмін даними між клієнтом і сервером.....	32
2.7 Життєвий цикл даних у системі.....	33

2.8 Правила валідації та узгодженості даних.....	35
2.9 Приклади структур даних.....	36
Висновки до розділу 2.....	37
3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.....	38
3.1 Архітектура програмної системи.....	38
3.2 Серверна частина NestJS API.....	39
3.3 Деталізація адміністративних функцій.....	45
3.4 Мобільний застосунок Flutter.....	46
3.5 Керування станом, локалізація та якість UI.....	59
3.6 Алгоритмізація програмних модулів.....	59
3.6.1 Генерація турнірної сітки.....	59
3.6.2 Оновлення результату матчу.....	62
3.6.3 Локальний кеш і fallback-дані.....	62
3.6.4 Повідомлення та push hooks.....	63
3.7 Реалізація графіків, діаграм і растрової графіки.....	63
3.8 Тестування програмної системи.....	66
3.9 Основні маршрути API.....	68
3.10 Маршрутна карта мобільного застосунку.....	69
3.11 Скріншоти, які потрібно вставити під час остаточного оформлення	70
3.12 Деталізація реалізації серверних модулів.....	71
3.13 Деталізація мобільних екранів і взаємодії з користувачем.....	75
3.14 Обробка помилок і нестандартних ситуацій.....	76
Висновки до розділу 3.....	78

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ.....	79
4.1 Вимоги до апаратного та програмного забезпечення.....	79
4.2 Порядок локального запуску.....	80
4.3 Склад інсталяційного пакета.....	81
4.4 Опис основних користувацьких сценаріїв.....	81
4.5 Обмеження та напрями подальшого розвитку.....	83
4.6 Експлуатаційні ризики та заходи зменшення.....	83
4.7 Детальний план тестування.....	85
4.8 Чеклист демонстрації програмного продукту.....	87
4.9 Контроль якості та критерії готовності.....	88
4.10 Файли та конфігурація, які потрібно зберегти.....	89
4.11 Матриця трасування вимог.....	90
4.12 Сценарій підготовки до захисту.....	92
4.13 План супроводу та подальшої підтримки.....	93
4.14 Практична цінність результатів.....	95
Висновки до розділу 4.....	96
ВИСНОВКИ.....	97
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	99
ДОДАТОК А.....	102

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Таблиця 1 - Перелік скорочень

Позначення	Пояснення
API	Application Programming Interface, програмний інтерфейс взаємодії компонентів.
REST	Архітектурний стиль побудови HTTP API з ресурсами та стандартними методами.
JWT	JSON Web Token, маркер автентифікації користувача.
DTO	Data Transfer Object, об'єкт передавання та валідації даних.
БД	База даних.
СУБД	Система управління базами даних.
CRUD	Create, Read, Update, Delete, базові операції з даними.
UI	User Interface, користувацький інтерфейс.
FCM	Firebase Cloud Messaging, сервіс push-повідомлень.
CS2	Counter-Strike 2, кіберспортивна дисципліна, що використовується в тестових даних.
HLTV	Публічна кіберспортивна статистична платформа, дані якої використано як приклад seed-наповнення.
BYE	Автоматичний прохід команди в наступний раунд за непарної кількості учасників або неповної сітки.

ВСТУП

Кіберспорт за останні роки перетворився з вузького сегмента ігрової культури на самостійну індустрію з командною інфраструктурою, професійними турнірами, медіа-трансляціями, рейтингами, статистикою і великою кількістю взаємопов'язаних даних. Для організаторів, команд і глядачів важливо швидко отримувати актуальну інформацію про склади, матчі, результати, статус турніру та зміни, що відбуваються під час live-етапів. Використання розрізнених таблиць, чатів і ручних списків призводить до дублювання інформації, помилок у результатах та ускладнює контроль доступу.

Актуальність теми полягає у потребі створення єдиного мобільного інструмента, який об'єднує базу гравців і команд, турнірний менеджмент, live-операції матчів, повідомлення, адміністративну модерацію та аналітику. Така система повинна бути зручною для повторного використання організаторами турнірів, підтримувати різні ролі користувачів і забезпечувати синхронізацію змін між сервером та мобільним клієнтом.

Об'єктом дослідження є процеси інформаційного супроводу кіберспортивних команд, гравців, турнірів і матчів у мобільному клієнт-серверному середовищі.

Предметом дослідження є методи, моделі та програмні засоби побудови мобільної інформаційної системи керування кіберспортивною базою даних з REST API, документно-орієнтованою базою даних, realtime-оновленнями, рольовим доступом і локальним кешуванням.

Метою роботи є підвищення ефективності ведення кіберспортивної бази даних і супроводу турнірів шляхом розроблення мобільної інформаційної системи VOID Esports Database, що забезпечує централізоване зберігання даних, керування життєвим циклом турнірів, live-оновлення матчів, адміністрування користувачів і аналітичні звіти.

Для досягнення мети необхідно вирішити такі задачі:

- проаналізувати предметну область кіберспортивних інформаційних систем і визначити проблеми, які виникають під час керування командами, гравцями, турнірами та матчами;
- виконати огляд існуючих теоретичних і практичних рішень, визначити їх переваги та обмеження;
- сформулювати функціональні та нефункціональні вимоги до системи, визначити ролі користувачів і сценарії роботи;
- розробити інформаційне забезпечення системи: модель даних, колекції MongoDB, зв'язки між сутностями, індекси та правила валідації;
- обґрунтувати вибір технологій Flutter, NestJS, MongoDB, Redis, Socket.IO, Docker і пов'язаних бібліотек;
- реалізувати серверні модулі API, мобільні екрани, локальне збереження сесій, fallback-дані та інтеграцію з realtime-каналом;
- описати алгоритми генерації турнірної сітки, текстового пошуку, оновлення результатів матчів, повідомлень і модераційних дій;
- провести перевірку працездатності, описати тестування, вимоги до запуску, склад інсталяційного пакета і перспективи розвитку системи.

Методи дослідження включають системний аналіз предметної області, об'єктно-орієнтоване моделювання, проєктування клієнт-серверної архітектури, проєктування документної бази даних, валідацію DTO, рольову модель доступу, алгоритмізацію турнірної сітки, модульне тестування та експериментальну перевірку локального запуску.

Практичне значення одержаних результатів полягає в тому, що розроблений застосунок може бути використаний як основа для кіберспортивної платформи локального або навчального рівня, для демонстрації принципів побудови сучасних мобільних інформаційних систем, а також для подальшого розширення функціоналу турнірного менеджменту.

Апробацію програмного продукту виконано у формі локального запуску серверної та мобільної частин, наповнення тестовими даними CS2 HLTV Top 10,

перевірки health endpoint, Swagger-документації, базових тестів і сценаріїв взаємодії з основними екранами застосунку.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків і списку використаних джерел. У першому розділі виконано системний аналіз предметної області. У другому розділі описано інформаційне забезпечення та модель даних. У третьому розділі розглянуто прикладне програмне забезпечення. У четвертому розділі наведено рекомендації щодо впровадження, експлуатації та тестування системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Розроблювана система належить до класу мобільних інформаційних систем з клієнт-серверною архітектурою. Її призначенням є автоматизація процесів збирання, зберігання, пошуку, оновлення та візуального подання інформації про кіберспортивних гравців, команди, турніри, матчі, повідомлення, модераційні звернення та адміністративні налаштування платформи.

Постановка завдання передбачає не лише створення окремого довідника гравців або команд, а комплексну реалізацію життєвого циклу кіберспортивної події. Система повинна дозволяти створювати турнір, формувати список учасників, генерувати сітку, керувати матчами, оновлювати результати, переглядати турнірну історію, отримувати повідомлення про важливі події та виконувати адміністративний контроль користувачів.

З погляду характеру використання інформації система є інформаційно-довідковою та інформаційно-управляючою. Довідкова частина надає глядачам і учасникам доступ до профілів гравців, команд, турнірів і матчів. Управляюча частина дає операторам, адміністраторам і менеджерам команд інструменти створення, редагування, перевірки, модерації та супроводу live-операцій.

Особливістю предметної області є поєднання статичних і динамічних даних. До статичних або відносно стабільних даних належать профілі користувачів, гравців, команд, налаштування платформи, склад ростеру. До динамічних належать статуси матчів, live-події, результати карт, veto-операції, повідомлення і зміни турнірної сітки. Тому система повинна підтримувати як стандартні REST-запити, так і realtime-канал оновлень.

Таблиця 2 - Основні об'єкти автоматизації

Об'єкт	Автоматизовані операції
Користувачі	Реєстрація, вхід, профіль, мова, ролі, сесії, зміна пароля.
Гравці	Профіль гравця, нікнейм, дисципліна, країна, команда, аватар, статистика, історія матчів.

Об'єкт	Автоматизовані операції
Команди	Створення чернетки, публікація, редагування, архівація, ростер, запрошення, ролі учасників.
Турніри	Чернетки, шаблони, реєстрація команд, lifecycle, check-in, seeding, roster lock, disputes, results.
Матчі	Створення, редагування, рахунок, карти, veto, події, медіа, паузи, технічні проблеми, undo.
Адміністрування	Користувачі, ролі, модерація, featured-контент, оголошення, налаштування платформи, аудит.
Аналітика	Dashboard, топ гравців і команд, статистичний звіт гравців за дисципліною та країною.
Повідомлення	Персональні повідомлення, системні повідомлення, налаштування, обрані гравці, push-token.

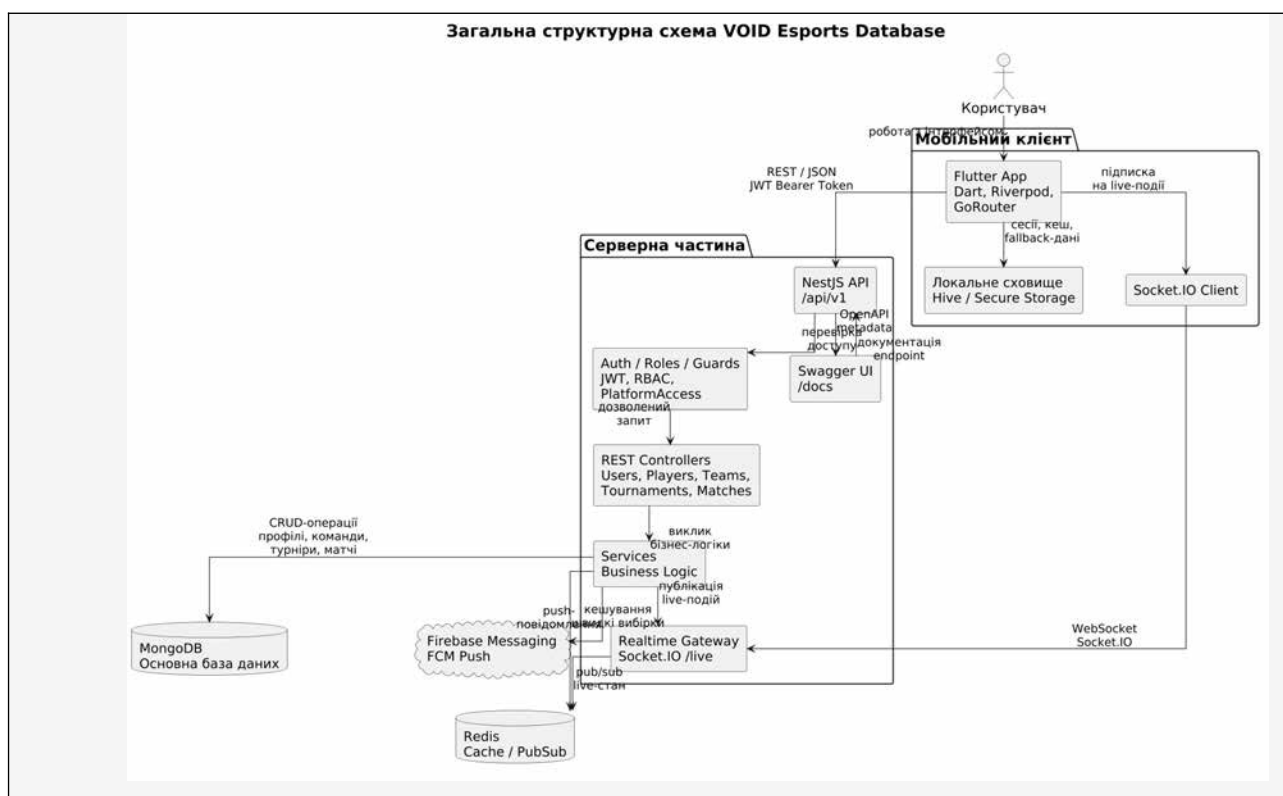


Рис. 1 - Загальна структурна схема VOID Esports Database: Flutter-клієнт, NestJS API, MongoDB, Redis, Socket.IO, Swagger, Firebase Messaging

1.2 Аналіз предметної області

Кіберспортивний турнір є сукупністю організаційних і технічних процесів: формування правил, реєстрація команд, перевірка складів, створення розкладу, проведення матчів, фіксація результатів, вирішення спірних ситуацій, публікація підсумків і підтримання історії подій. На відміну від звичайного довідника, така система потребує контролю статусів і переходів між етапами.

Гравець у межах системи є не лише записом у базі даних, а активною сутністю, пов'язаною з користувацьким обліковим записом, командою, дисципліною, країною, роллю в грі, доступністю, аватаром, статистикою та історією матчів. Це створює потребу в унікальності нікнейму, перевірці профілю і підтримці різних рівнів видимості.

Команда є агрегатом, що поєднує менеджера, учасників, ростер гравців, запрошення, турніри, статистику, медіа-ресурси та життєвий цикл. У системі команда може бути чернеткою, активною, рекрутинговою, неактивною або архівною. Такий підхід дозволяє не видаляти організаційно важливі дані передчасно.

Матч є найдинамічнішою сутністю системи. Він містить плановий і фактичний час, команди, формат серії, рахунок, статус, переможця, карти, veto-кроки, події, медіа, MVP, паузи, технічні проблеми та live-метрики. Зміна матчу повинна синхронізуватися з турнірною сіткою, повідомленнями та realtime-каналом.

Таблиця 3 - Ролі користувачів і зона відповідальності

Роль	Основні можливості
Admin	Повне адміністрування платформи, користувачів, ролей, модерації, налаштувань, featured-контенту та оголошень.
Tournament Operator / Organizer	Створення і ведення турнірів, генерація сітки, check-in, seeding, roster lock, результати, матчі.
Team Admin	Створення команди, керування ростером, запрошення гравців, редагування профілю команди.
Player	Створення власного профілю, прийняття або відхилення

	запрошень, перегляд турнірів і матчів.
Viewer	Перегляд публічних списків, профілів, турнірів, матчів і featured-контенту.

Таблиця 4 - Матриця основних сценаріїв використання

I D	Сценарій	Актор	Результат
UC-01	Реєстрація користувача	Новий користувач	Створений обліковий запис, виданий JWT, збережена сесія.
UC-02	Вхід у систему	Зареєстрований користувач	Активна сесія, доступ до ролей і персональних даних.
UC-03	Зміна пароля	Авторизований користувач	Оновлений hash пароля, за потреби завершені інші сесії.
UC-04	Створення профілю гравця	Player або Admin	Створений Player з унікальним nickname і прив'язкою до User.
UC-05	Пошук гравців	Viewer, Player, Admin	Відфільтрований список за nickname, teamName або realName.
UC-06	Створення команди	Team Admin або Admin	Чернетка команди з можливістю autosave і подальшої публікації.
UC-07	Запрошення гравця в команду	Team Admin	Створений invitation, гравець бачить запрошення у профілі.
UC-08	Прийняття запрошення	Player	Гравець додається до roster, оновлюється teamId.
UC-09	Створення турніру	Tournament Operator	Чернетка або опублікований турнір з параметрами format, region, capacity.
UC-10	Реєстрація команди на турнір	Operator або команда	Команда додана до participants/registerTeam списку.
UC-11	Генерація сітки	Operator	Створені rounds і matches з урахуванням seeding і BYE.
UC-12	Check-in команди	Operator	Зафіксований стан присутності команди перед стартом.
UC-13	Оновлення рахунку	Operator	Матч оновлено, bracket

I D	Сценарій	Актор	Результат
	матчу		синхронізовано, realtime-подія надіслана.
UC-14	Додавання match event	Operator	У timeline матчу додана подія з типом, описом і payload.
UC-15	Створення скарги	Користувач	Створений Report для подальшої модерації.
UC-16	Модерація скарги	Admin	Report отримує новий status/action, створюється moderation audit.
UC-17	Створення оголошення	Admin	Announcement з аудиторією, статусом і scheduled/send-now логікою.
UC-18	Перегляд аналітики	Admin	Звіт за гравцями, топи та dashboard з кешуванням.

1.3 Огляд існуючих рішень

Перед розробленням VOID Esports Database було розглянуто кілька реальних сервісів, які працюють у близькій предметній області. Їх можна умовно поділити на статистичні портали, wiki-довідники, матчмейкінгові платформи, турнірні SaaS-рішення та платформи для масових івентів. Кожен із цих напрямів вирішує важливу задачу, але не покриває повністю той набір функцій, який потрібний для навчальної мобільної клієнт-серверної системи.

HLTV.org є одним із найвідоміших ресурсів для Counter-Strike. На сторінці матчів сервіс показує live-матчі, майбутні матчі, формат серії, подію, команди та фільтрацію за подіями. Для VOID Esports Database корисною є ідея швидкого перегляду матчів, live-статусу, сторінок команд і гравців. Водночас HLTV орієнтований на професійну сцену та публічну статистику, а не на те, щоб локальний організатор сам створював турніри, керував ролями, реєстраціями, сіткою і результатами власної події.

Liquipedia Counter-Strike виконує роль великого wiki-довідника. Там зручно дивитися сторінки команд, гравців, турнірів, ростерів, історії виступів і довідкові матеріали. Для проєкту це корисний приклад структурованого подання кіберспортивних сутностей: гравець, команда, турнір, матч, організація, дисципліна. Обмеженням такого підходу є те, що wiki-формат не є операційною системою проведення турніру: він не замінює live-панель оператора, автоматичне просування переможців у bracket, рольовий доступ і push-повідомлення.

FACEIT є прикладом конкурентної ігрової платформи. У документації FACEIT описуються три основні способи гри: matchmaking, tournaments і clubs. Для CS2-турнірів платформа підтримує список турнірів, фільтри за регіоном, skill level, anti-cheat та інші параметри, а також варіанти участі як party, team або solo player. Для VOID Esports Database було враховано ідеї ролей, командної участі, check-in і матчевого досвіду. Проте FACEIT більше сфокусований на власній змагальній екосистемі, matchmaking, Elo, anti-cheat і лігах, а не на відкритій навчальній системі з власною базою даних, власним API і мобільною адміністративною частиною.

Toornament є прикладом організаторської платформи для турнірів. У довідці Toornament описано check-in, список турнірів і матчів для учасників, звітування результатів, а також велику кількість структур: single elimination, double elimination, round robin, bracket groups, league, Swiss system і custom bracket. Для розробленої системи це підтверджує, що турнірний модуль має підтримувати не лише назву турніру, а й формат, кількість учасників, seeding, check-in, реєстрацію, матчі та результати. Обмеженням Toornament для цієї роботи є залежність від зовнішньої SaaS-платформи і відсутність контролю над внутрішньою архітектурою системи.

Battlefy і start.gg також можна віднести до класу турнірних платформ. Battlefy у довідці для організаторів має розділи про tournament setup, brackets, participants, registration, check-in, score reporting, seeding і embed codes. start.gg позиціонується як сервіс для організації подій, реєстрації, bracket та event

management. Такі системи добре підходять для публічних івентів, але вони не розв'язують задачу створення власної мобільної інформаційної системи з окремими модулями backend, локальним кешем, Firebase hooks, Redis, Socket.IO і власною схемою MongoDB.

Отже, аналіз існуючих рішень показує, що готові сервіси добре закривають окремі частини предметної області: HLTV - статистику та live-матчі, Liquipedia - довідкову базу, FACEIT - matchmaking і змагальну участь, Toornament/Battlefy/start.gg - організацію турнірів. VOID Esports Database не копіює конкретний сервіс, а поєднує ключові ідеї цих класів у навчальному проєкті: мобільний клієнт, NestJS API, MongoDB-модель, ролі, турніри, матчі, повідомлення, аналітику, модерацию і локальне розгортання.

Таблиця 5 - Порівняння існуючих рішень

Приклад	Сильні сторони	Обмеження для теми роботи	Врахування у проєкті
HLTV.org	Live-матчі, розклад, результати, сторінки команд і гравців, статистика CS2	Не призначений для створення локальних турнірів і власного адміністрування	Використано ідею match center, live-статусів, профілів і історії матчів.
Liquipedia	Wiki-структура турнірів, команд, гравців, ростерів і історичних результатів	Wiki не замінює операційну панель турніру, realtime і рольову модель	Використано ідею структурованої довідкової бази сутностей.
FACEIT	Matchmaking, tournaments, clubs, Elo, anti-cheat, участь solo/team/party	Фокус на власній competitive-екосистемі, а не на відкритій локальній БД	Враховано ролі, командну участь, check-in і матчеві сценарії.
Toornament	Single/double elimination, round robin, Swiss, check-in, score reporting	SaaS-залежність і відсутність контролю над внутрішньою архітектурою	Реалізовано власні tournament lifecycle, bracket generation, seeding і results.
Battlefy	Tournament setup,	Підходить для івентів,	Враховано

Приклад	Сильні сторони	Обмеження для теми роботи	Врахування у проєкті
	registration, participants, brackets, check-ins, score reporting	але не дає власного mobile/backend стека	організаторські сценарії та підтвердження результатів.
start.gg	Реєстрація, bracket, event management, підтримка великих подій	Більше орієнтований на event-hosting, ніж на власну предметну БД	Враховано ідею подієвої структури та зв'язку учасників із bracket.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги визначають, які дії система повинна виконувати для різних ролей. Для кінцевого користувача важливими є реєстрація, вхід, перегляд dashboard, пошук гравців і команд, перегляд турнірів, матчів і повідомлень. Для адміністратора важливі керування користувачами, ролями, модерацією, аналітикою, оголошеннями та платформними параметрами.

Нефункціональні вимоги визначають якість реалізації. Система повинна мати зрозумілу структуру API, валідацію вхідних даних, захист від несанкціонованого доступу, передбачувану обробку помилок, можливість локального запуску, зручність мобільного інтерфейсу, адаптацію до недоступності backend і можливість розширення новими дисциплінами та модулями.

Таблиця 6 - Нефункціональні вимоги

Вимога	Реалізація у проєкті
Безпека	JWT, bcrypt для пароля, role guards, platform access guard, DTO whitelist validation, session revocation.
Продуктивність	Індекси MongoDB, Redis-кеш dashboard/analytics, пагінація адміністративних списків.
Надійність	Fallback sample/cache у мобільному клієнті, health endpoint, retry-

	friendly startup через Docker.
Супроводжуваність	Поділ на модулі NestJS, DTO, схеми, сервіси, контролери, окремі Flutter screens/widgets.
Масштабованість	REST API, Socket.IO gateway, MongoDB collections, окремі модулі для нових функцій.
Зручність	Мобільні екрани, пошук, фільтри, chips, wizard-форми, quick actions, локалізація uk/en.

1.5 Моделювання предметної області

Предметну область доцільно подати як множину взаємопов'язаних сутностей. Нехай $D = \{U, P, C, T, M, N, R\}$, де U - множина користувачів, P - гравців, C - команд, T - турнірів, M - матчів, N - повідомлень, R - модераторських записів. Основними відношеннями є user-player, team-roster, tournament-team, tournament-match, user-notification і match-audit.

Користувач може мати нуль або один профіль гравця, одна команда має багато гравців у ростері, турнір має багато команд і матчів, матч належить до турніру та пов'язує дві команди. Така структура відповідає документній моделі MongoDB, де частина вкладених даних зберігається в основному документі, а ключові сутності мають окремі колекції та ObjectId-зв'язки.

Для процесного моделювання виділено кілька основних сценаріїв: автентифікація, створення команди, створення турніру, реєстрація команди на турнір, генерація сітки, оновлення матчу, отримання повідомлення, модерація звернення, перегляд аналітики. Усі сценарії відображають взаємодію мобільного клієнта, API, MongoDB, Redis і realtime gateway.

Viewer

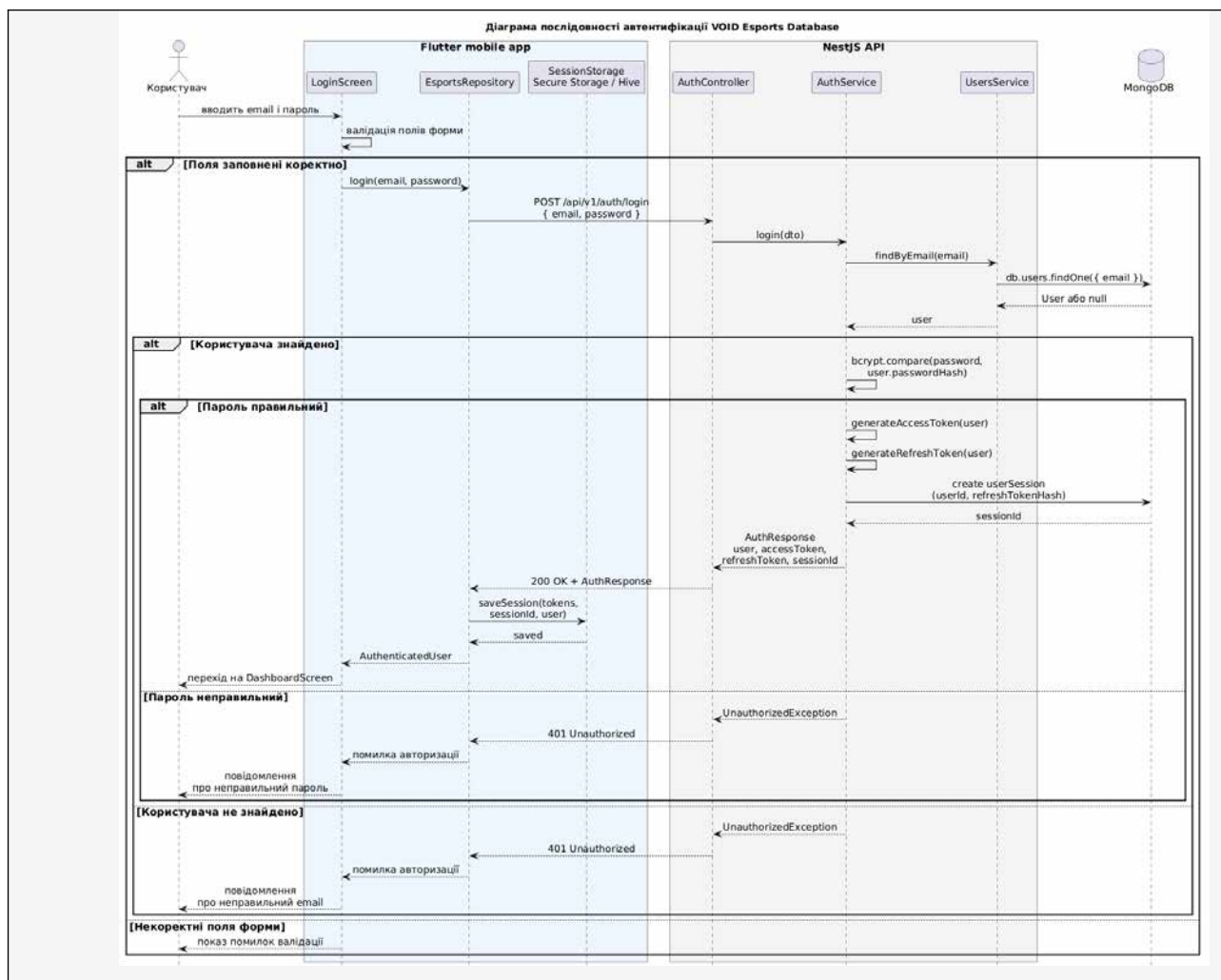


Рис. 3 - Діаграма послідовності автентифікації: Flutter LoginScreen, EsportsRepository, AuthController, AuthService, UserService, MongoDB, SessionStorage

1.6 Ролі користувачів та межі відповідальності

Під час аналізу предметної області важливо було не просто перелічити ролі, а зрозуміти, які дії справді потрібні кожній групі користувачів. У кіберспортивній системі одна і та сама сутність може бути видимою для всіх, але змінювати її мають право лише окремі ролі. Наприклад, профіль гравця переглядає глядач, редагує сам гравець або адміністратор, а підтвердження профілю виконує користувач з підвищеними правами.

Рольова модель у VOID Esports Database побудована навколо практичних сценаріїв. Admin відповідає за платформу загалом, Tournament Operator - за турнірний процес, Team Admin - за склад команди, Player - за власний профіль і участь у команді, Viewer - за перегляд відкритої інформації. Organizer

розглядається як роль, близька до оператора, але з акцентом на організаційні дії.

Такий розподіл потрібний для того, щоб система не перетворювалася на один великий адміністративний екран. Кожен користувач повинен бачити доступні йому команди, кнопки та форми, а сервер у будь-якому разі має повторно перевіряти право на дію. Це особливо важливо для матчів і турнірів, оскільки помилкове оновлення результату може вплинути на сітку та повідомлення інших користувачів.

Таблиця 7 - Ролі користувачів і межі їх відповідальності

Роль	Основні дії	Обмеження
Admin	Налаштування платформи, користувачі, ролі, модерація, оголошення, featured-контент, аналітика, аудит.	Потрібен повний доступ, але всі дії мають фіксуватися в audit log.
Tournament Operator	Створення турніру, реєстрація команд, seeding, check-in, генерація сітки, оновлення матчів.	Не повинен керувати глобальними налаштуваннями або чужими користувачами.
Organizer	Підготовка події, опис турніру, розклад, правила, контроль організаційних статусів.	Може працювати в межах конкретного турніру.
Team Admin	Створення команди, редагування ростеру, запрошення гравців, подання команди на турнір.	Не має права редагувати чужі команди або результати матчів без призначення.
Player	Власний профіль, аватар, статистика, прийняття запрошень, перегляд матчів і турнірів.	Не може самовільно змінювати турнірні результати.
Viewer	Перегляд відкритих профілів, команд, турнірів, матчів, новин і загальної статистики.	Працює переважно з публічними GET-запитами.

1.7 Користувацькі історії та критерії приймання

Для уточнення функціональних вимог було сформульовано користувацькі історії. Такий підхід зручний для дипломного проєкту, тому що він пов'язує технічну реалізацію з реальною поведінкою користувача. Якщо історія не має критерію приймання, її складно перевірити під час демонстрації.

У системі виділено історії для звичайного користувача, гравця, менеджера команди, турнірного оператора й адміністратора. Критерії приймання описують не весь код, а результат, який повинен побачити користувач: створений запис, оновлений статус, відображене повідомлення, помилка доступу або зміна турнірної сітки.

Таблиця 8 - Користувацькі історії та критерії приймання

Рольова історія	Потреба	Критерій приймання
Як новий користувач	я хочу зареєструватися та увійти в систему	після успішного входу збережено JWT, sessionId і відкрито головний екран.
Як гравець	я хочу створити профіль із nickname і дисципліною	профіль з'являється у списку гравців, nickname не дублюється.
Як гравець	я хочу додати аватар або URL зображення	профіль показує зображення або коректний fallback.
Як Team Admin	я хочу створити команду та зібрати ростер	команда має статус, slug, список гравців і може бути опублікована.
Як Team Admin	я хочу запросити гравця до команди	гравець отримує запрошення і може прийняти або відхилити його.
Як Tournament Operator	я хочу створити турнір із шаблону	форма заповнює базові параметри, турнір можна опублікувати.
Як оператор матчу	я хочу оновити рахунок і карти	матч змінює status/score, а переможець просувається у

		bracket.
Як глядач	я хочу переглядати live-матчі	екран показує актуальний статус, рахунок, карти та події.
Як Admin	я хочу модерувати скарги	звіт змінює статус, дія потрапляє до audit log.
Як Admin	я хочу опублікувати оголошення	оголошення має аудиторію, статус і може бути відправлене.
Як користувач	я хочу отримувати повідомлення про важливі події	повідомлення має пріоритет, категорію та actionRoute.
Як користувач без стабільного backend	я хочу бачити кешовані або демонстраційні дані	застосунок не падає і показує зрозумілий стан недоступності.

1.8 Межі проєкту та припущення

Під час виконання роботи межі проєкту були свідомо обмежені мобільним застосунком і backend API. Веб-адміністративна панель не розроблялася окремо, оскільки адміністративні функції вже представлені в мобільному клієнті. Це дозволило зосередитися на повному клієнт-серверному циклі: модель даних, API, мобільний UI, локальне зберігання, рольовий доступ і сценарії запуску.

Також у межах роботи не ставилася задача створити комерційний сервіс з гарантованою production-інфраструктурою. У проєкті підготовлено основу для production-розгортання: Docker, Nginx, змінні середовища, Firebase hooks, Redis і Swagger. Однак реальні push-повідомлення, OAuth-провайдери, HTTPS-сертифікати та резервне копіювання мають налаштовуватися окремо для конкретного середовища.

Предметна область у записці розглядається на прикладі CS2, але модель не прив'язана лише до однієї дисципліни. Поля game, discipline, format, region і metadata дають змогу надалі додати інші ігри, якщо для них будуть уточнені правила матчів, статистики та турнірних форматів.

Окреме припущення стосується джерела даних. Для демонстрації використано seed-наповнення, подібне до відкритих кіберспортивних рейтингів

і прикладів команд. У реальному впровадженні частина даних може імпортуватися з зовнішніх сервісів або вводитися операторами вручну.

Висновки до розділу 1

1. Проаналізовано предметну область кіберспортивного турнірного менеджменту та визначено, що система повинна поєднувати довідкові й управляючі функції.
2. Визначено ключові сутності: користувач, гравець, команда, турнір, матч, повідомлення, звіт, аудит, налаштування платформи.
3. Сформовано функціональні та нефункціональні вимоги, а також рольову модель доступу.
4. Показано, що розробка повинна підтримувати REST-взаємодію, realtime-оновлення, локальне кешування, адміністративний контроль і розгортання в Docker-середовищі.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Загальна модель даних

Інформаційне забезпечення системи VOID Esports Database побудовано на MongoDB з використанням Mongoose-схем у серверній частині NestJS. Такий підхід дає змогу зберігати як структуровані сутності з чіткими полями, так і гнучкі вкладені документи для статистики, live-метрик, технічної інформації, результатів турніру або payload подій.

Основними колекціями є users, userSessions, players, teams, tournaments, tournamentParticipants, tournamentStages, tournamentGroups, swissRounds, bracketNodes, rosterLockRecords, autoAdvanceRecords, tournamentDisputes, tournamentPlacements, matches, matchAuditLogs, notifications, reports, moderationAuditLogs, announcements, pushSubscriptions, featuredContent, platformSettings і platformSettingsAuditLogs.

У проєкті використано timestamps для автоматичного збереження createdAt та updatedAt, ObjectId-посилання між сутностями, enum-поля для контрольованих статусів, індекси для пошуку та сортування, а також DTO-рівень для перевірки даних до моменту запису в базу.

Таблиця 9 - Основні колекції бази даних

Колекція	Призначення
users	Облікові записи, ролі, мова, online-статус, notification preferences, favorite players.
players	Профілі гравців, discipline, country, teamId, avatarUrl, stats, matchHistory, visibility, availability.
teams	Команди, slug, game, manager, members, roster, invitations, lifecycle status, stats.
tournaments	Турніри, format, status, lifecycle, participants, bracket, operatorConfig, resultsConfig.
matches	Матчі, tournamentId, teams, stage, format, status, maps, veto, events, media, liveMetrics.
notifications	Персональні та системні повідомлення користувача, категорії,

	пріоритет, actionRoute.
reports	Модераційні звернення, причина, severity, status, targetType, audit trail.
announcements	Адміністративні оголошення, аудиторія, статус, scheduledAt, delivery stats.
featuredContent	Закріплення сутностей на головній сторінці, order, visibility, publishAt, expiresAt.
platformSettings	Глобальні налаштування доступу, реєстрацій, модерації та notification defaults.

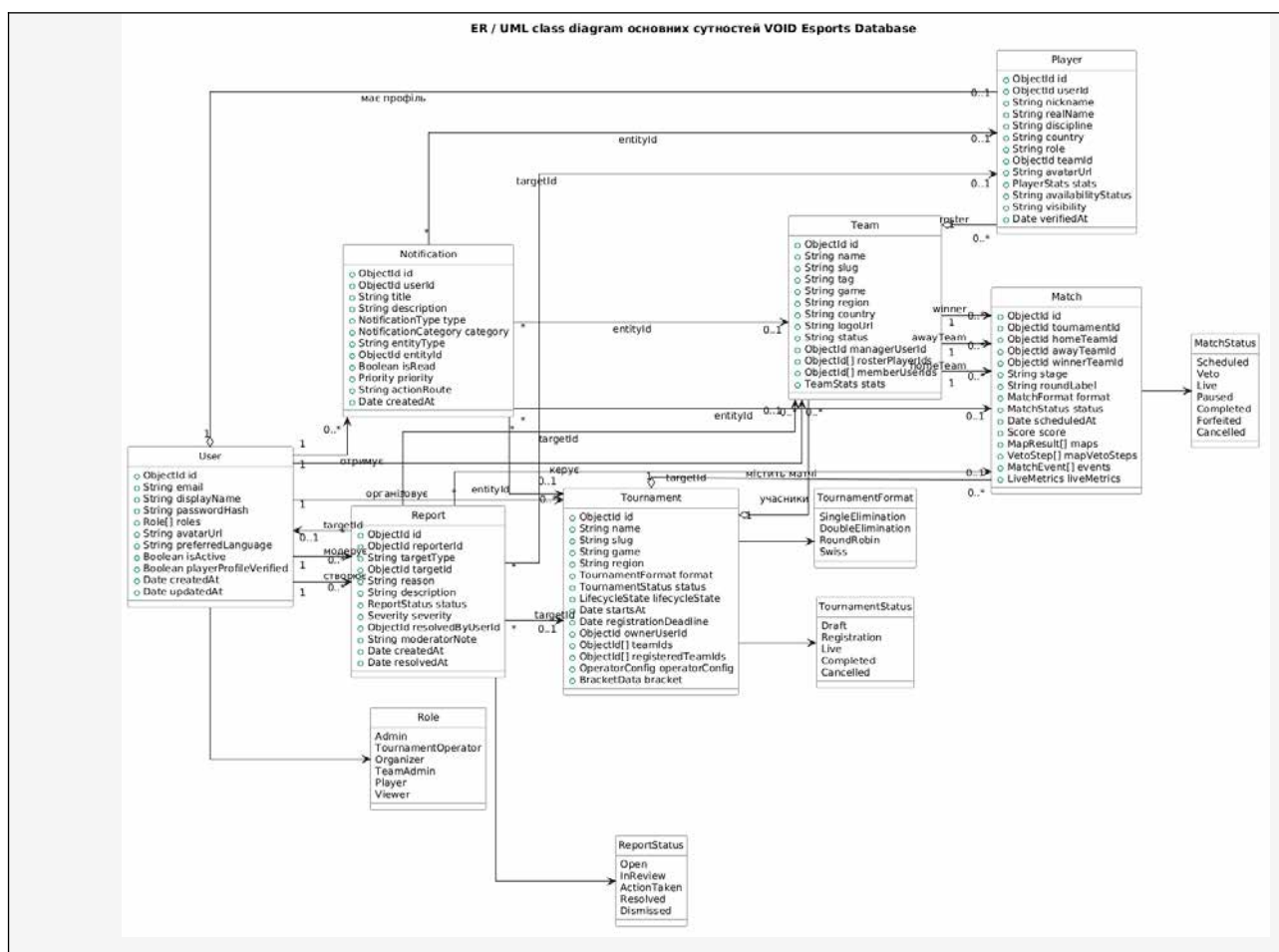


Рис. 4 - ER або UML class diagram основних сутностей: User, Player, Team, Tournament, Match, Notification, Report

2.2 Обґрунтування вибору СУБД

MongoDB обрано як основну СУБД з кількох причин. По-перше, предметна область містить багато вкладених структур, які не завжди зручно

нормалізувати в класичні таблиці: карти матчу, veto-кроки, live-події, MVP, медіа, bracket-дані, конфігурації турніру, notification metadata. Документна модель дозволяє зберігати ці структури природно.

По-друге, MongoDB добре підходить для швидкої розробки ітераційних проєктів, де модель може розширюватися новими полями. У системі вже є поля details, stats, metadata, operatorConfig, resultsConfig, liveMetrics, які дають змогу поступово додавати функції без складних міграцій на ранній стадії.

По-третє, Mongoose забезпечує програмну схему поверх документної бази: типи, enum-значення, required-поля, defaults, indexes, timestamps і методи роботи з ObjectId. Це зменшує ризик неконтрольованої неструктурованості даних і зберігає баланс між гнучкістю та дисципліною проєктування.

Redis використано як допоміжне сховище для кешування dashboard і аналітичних результатів, а також як потенційний pub/sub шар для realtime-подій. У локальному режимі Redis може бути вимкнений, що підвищує зручність запуску під час розробки.

Таблиця 10 - Переваги обраного підходу до зберігання даних

Компонент	Практичне значення
Документна структура	Зручне зберігання вкладених карт, подій, статистики, налаштувань і metadata.
ObjectId-зв'язки	Можливість пов'язувати користувачів, гравців, команди, турніри і матчі.
Індекси	Прискорення вибірок за email, slug, nickname, tournamentId, status, createdAt.
Mongoose	Схеми, enum, defaults, timestamps, populate, lean, validation на рівні моделі.
Redis	Короткоживучий кеш і pub/sub для живих оновлень.

2.3 Опис основних сутностей

User

Користувач зберігає email, displayName, passwordHash, roles, provider, avatarUrl, managedTeamId, online-статус, preferredLanguage, notificationPreferences, favoritePlayerIds і ознаку playerProfileVerified. Унікальний індекс users.email гарантує відсутність дублювання облікових записів.

Player

Гравець містить nickname, userId, realName, discipline, country, role, teamId, avatarUrl, socialLinks, stats, achievements, matchHistory, favoriteWeapons, availabilityStatus, visibility, verifiedByUserId і verifiedAt. Індеси nickname, discipline+teamId, availabilityStatus+createdAt, country і userId підтримують пошук та адміністрування.

Team

Команда має name, slug, tag, game, region, country, city, logoUrl, bannerUrl, description, website, socialLinks, visibility, type, status, managerUserId, memberUserIds, rosterPlayerIds, tournamentIds, invitations, stats, achievements, settings і lifecycle metadata. Запрошення зберігають invitedUserId, invitedByUserId, status, timestamps і повідомлення.

Tournament

Турнір є центральною сутністю організаційної частини. Він містить name, slug, game, region, format, startsAt, registrationDeadline, description, status, lifecycleState, teamIds, registeredTeamIds, operatorConfig, checkInSettings, seedingSettings, rosterLockSettings, resultsConfig, bracket, metadata, ownerUserId і audit-поля.

Match

Матч містить tournamentId, stage, stageType, source, roundLabel, groupId, swissRoundId, bracketNodeId, format, bestOf, status, scheduledAt, homeTeamId,

awayTeamId, score, seriesScore, winnerTeamId, resultType, maps, mapVetoSteps, events, vods, demos, pauseEvents і liveMetrics. Він підтримує як звичайний результат, так і forfeit, технічні проблеми та паузи.

Notification

Повідомлення пов'язане з userId і містить title, description, type, category, entityId, entityType, isRead, priority, actionRoute і metadata. Індокси за userId, isRead, category і createdAt підтримують швидке отримання стрічки повідомлень.

Report

Модераційний звіт фіксує reporterId, targetId, targetType, reason, description, attachments, status, severity, moderatorNote, resolvedByUserId і metadata. Це дозволяє адміністратору вести процес розгляду скарг і залишати аудит дій.

Таблиця 11 - Ключові поля та індокси сутностей

Сутність	Ключові поля	Індокси
User	email, displayName, roles, preferredLanguage, notificationPreferences, favoritePlayerIds	email unique
Player	nickname, userId, discipline, teamId, country, stats, availabilityStatus, visibility	nickname unique; discipline+teamId; userId sparse unique
Team	name, slug, game, status, managerUserId, rosterPlayerIds, invitations, stats	slug unique
Tournament	name, slug, game, format, status, lifecycleState, teamIds, bracket, operatorConfig	slug unique
Match	tournamentId, status, scheduledAt, teams, score, maps, veto, events, liveMetrics	tournamentId+status+scheduledAt; bracketMatchId
Notification	userId, title, type, category, isRead, priority, actionRoute	userId+createdAt; userId+isRead
Report	reporterId, targetType, targetId, reason,	status+severity+createdAt;

Сутність	Ключові поля	Індекси
	severity, status	targetType+targetId

2.4 Довідники, статуси та контроль допустимих значень

У системі активно використовуються enum-довідники. Вони виконують роль контрольованих словників і зменшують ризик появи некоректних статусів у базі даних. Наприклад, роль користувача, статус турніру, статус матчу, тип команди, пріоритет повідомлення або причина скарги не вводяться довільним текстом, а обираються з фіксованого набору значень.

Такий підхід важливий для надійності системи. Якщо status матчу має обмежений набір Scheduled, Live, Paused, Completed, Forfeited, Cancelled, то бізнес-логіка може явно перевіряти дозволені переходи. Аналогічно lifecycleState турніру дозволяє відокремити чернетку, реєстрацію, live-етап і завершення.

Контроль допустимих значень реалізовано на кількох рівнях: TypeScript enum у backend, class-validator у DTO, Mongoose enum у схемах і відповідні enum у Flutter доменних моделях. Це забезпечує узгодженість між клієнтом, API та базою даних.

Таблиця 12 - Статусні довідники системи

Довідник	Приклади значень	Призначення
Role	Admin, TournamentOperator, Organizer, TeamAdmin, Player, Viewer	Розмежування доступу до операцій.
TournamentStatus	Draft, Registration, Live, Completed, Cancelled	Публічний статус турніру.
TournamentFormat	SingleElimination, DoubleElimination, RoundRobin, Swiss	Вибір алгоритму проведення.
TournamentLifecycleState	Draft, Registration, CheckIn, Live, Completed	Деталізований життєвий цикл.

Довідник	Приклади значень	Призначення
MatchStatus	Scheduled, Veto, Live, Paused, Completed, Forfeited, Cancelled	Стан матчу і live-логіка.
MatchFormat	BO1, BO3, BO5	Формат серії і кількість карт.
MatchVetoAction	Ban, Pick, Decider, SidePick	Операції veto-процесу.
TeamStatus	Draft, Active, Recruiting, Inactive, Archived	Життєвий цикл команди.
TeamRosterStatus	Active, Bench, Inactive, Trial	Стан гравця у ростері.
ReportStatus	Open, InReview, ActionTaken, Resolved, Dismissed	Модераційний процес.
NotificationCategory	Match, Tournament, Team, System, Moderation	Групування повідомлень.
NotificationPriority	Low, Normal, High, Critical	Важливість повідомлення.

2.5 Індексація та пошук

Пошук у мобільному застосунку реалізовано як клієнтське фільтрування вже отриманих списків. Користувач вводить рядок у `SearchBarField` або `AppSearchBar`, значення нормалізується через `trim().toLowerCase()`, після чого список фільтрується за `contains` у ключових полях. Такий підхід достатній для мобільного довідника з обмеженим обсягом даних і дає миттєву реакцію без повторного запиту до API.

Серверна індексація при цьому не втрачає значення: API має швидко повернути базові списки, відсортувати їх і відфільтрувати за дисципліною, командою, статусом або турніром. Саме тому у MongoDB створено індекси за `nickname`, `email`, `slug`, `tournamentId`, `status`, `scheduledAt`, `createdAt`, `country` та іншими полями.

Формально текстовий пошук можна подати предикатом $m(q, x)$, де q - нормалізований запит, x - сутність списку. $m(q, x) = 1$, якщо хоча б одне з полів $F(x)$ після нормалізації містить q як підрядок; інакше $m(q, x) = 0$. Для гравця $F(x)$

включає nickname, teamName і realName, для команди - name і country, для турніру - name, game label і format.

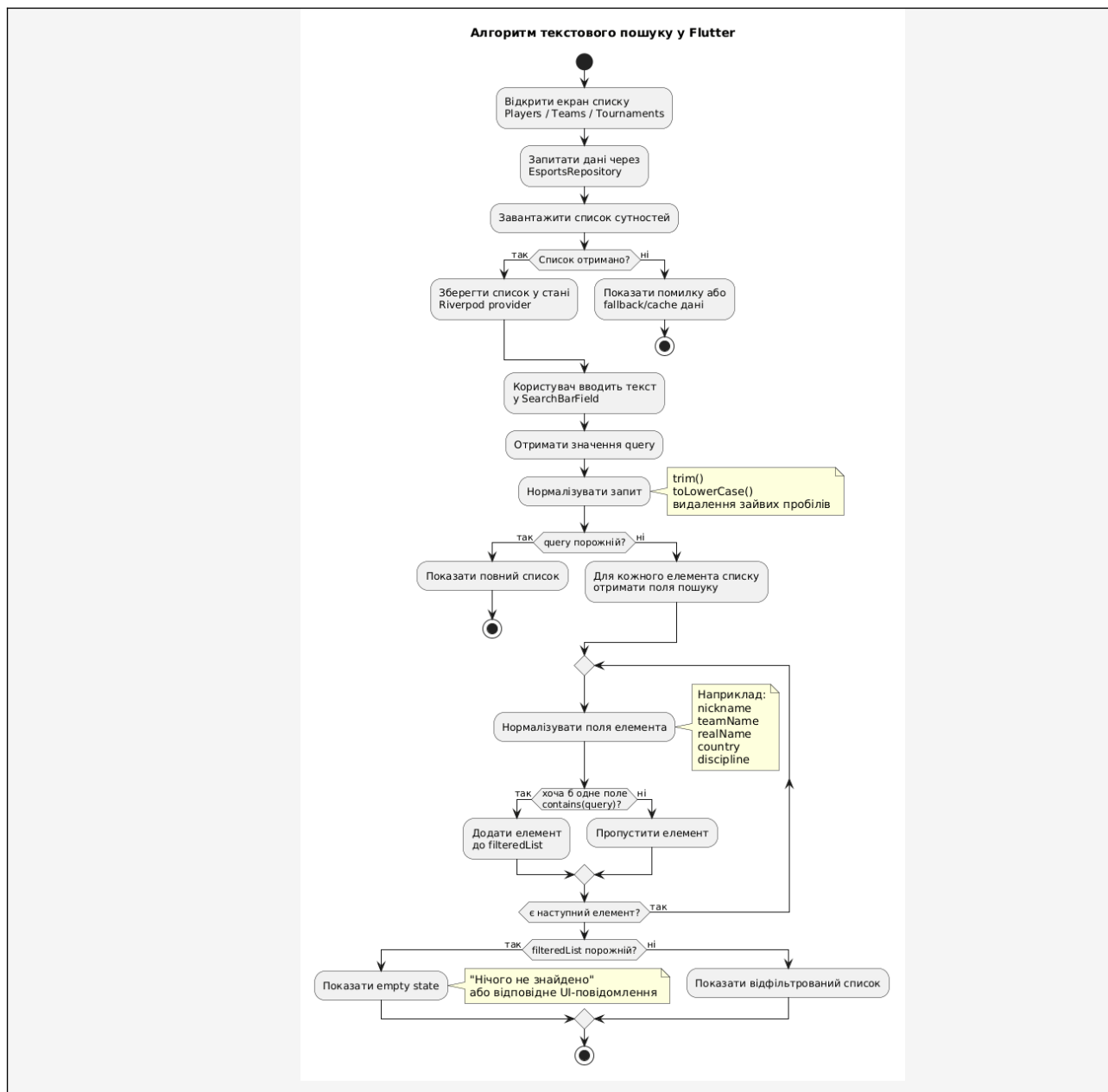


Рис. 5 - Алгоритм текстового пошуку у Flutter: завантаження списку, нормалізація запиту, фільтрація contains, показ empty state

2.6 Обмін даними між клієнтом і сервером

Мобільний застосунок взаємодіє з backend через клас EsportsRepository. Репозиторій інкапсулює HTTP-запити, обробку DioException, перетворення JSON у доменні моделі, збереження сесій, локальний кеш і fallback-дані. Такий шар відокремлює UI від деталей мережевої взаємодії.

API має глобальний префікс `/api/v1`. У `main.ts` підключено `helmet`, `compression`, `cookie-parser`, `CORS`, `ValidationPipe` з `whitelist`, `forbidNonWhitelisted` та `transform`. Swagger доступний за `/docs` і відображає групи endpoint для Auth, Users, Teams, Players, Tournaments, Matches, Analytics, Notifications, Reports та інших модулів.

Обмін даними відбувається у форматі JSON. Для захищених операцій клієнт додає заголовок `Authorization: Bearer JWT`. Сервер перевіряє токен через `JwtAuthGuard`, доступність платформи через `PlatformAccessGuard` і ролі через `RolesGuard`. На рівні DTO перевіряються типи, допустимі enum-значення, обов'язкові поля та обмеження рядків.

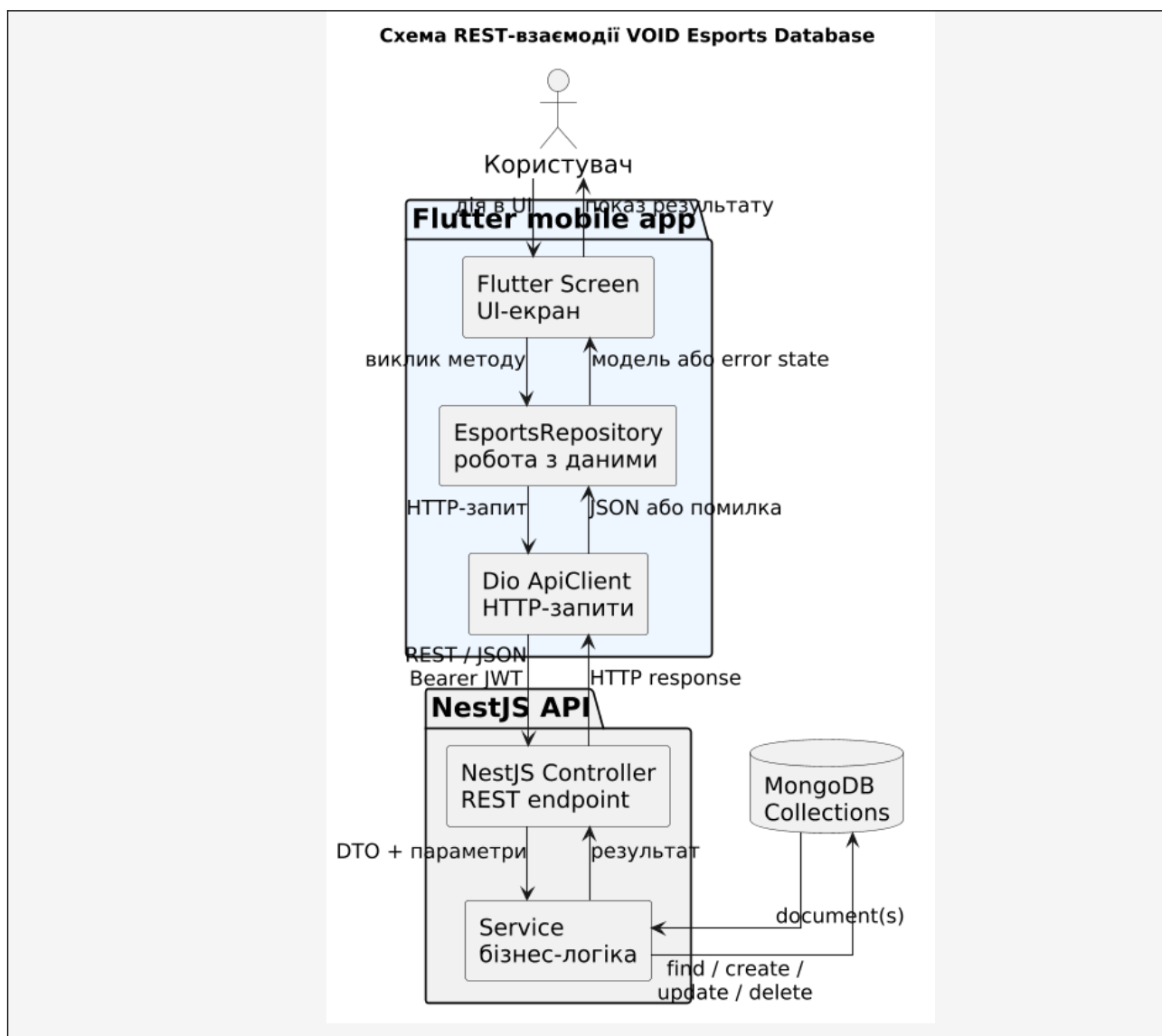


Рис. 6 - Схема REST-взаємодії: Flutter screen -> EsportsRepository -> Dio ApiClient -> NestJS Controller -> Service -> MongoDB

2.7 Життєвий цикл даних у системі

Дані у VOID Esports Database не є статичними. Більшість сутностей мають життєвий цикл: створення, чернетка, публікація, редагування, архівування або завершення. Для дипломної роботи це важливо, бо звичайний CRUD не описує повністю поведінку турнірів, команд і матчів.

Наприклад, турнір спочатку створюється як чернетка. Далі оператор заповнює параметри, відкриває реєстрацію, приймає команди, виконує check-in, генерує сітку, проводить матчі та завершує турнір. Кожен етап має власні дозволені дії. Якщо турнір уже live, випадкова зміна формату або capacity може порушити логіку сітки, тому такі зміни потрібно обмежувати.

Команда також проходить кілька станів. Вона може бути чернеткою, активною, recruiting, inactive або archived. Для користувача це виглядає просто, але на рівні БД важливо не видаляти команду одразу, тому що вона може бути пов'язана з попередніми матчами та статистикою.

Таблиця 13 - Життєвий цикл основних сутностей

Сутність	Типові стани	Операції переходу
User	registered, active, suspended, deleted	зміна ролей, мови, notification preferences, сесій.
Player	draft/profile, visible, verified, hidden	оновлення профілю, аватар, статистика, прив'язка до userId.
Team	draft, active, recruiting, inactive, archived	публікація, редагування ростеру, invitations, archive/restore.
Tournament	draft, registration, check-in, live, completed, cancelled	publish, register team, generate bracket, complete.
Match	scheduled, live, paused, completed, forfeited, cancelled	score update, maps, veto, pause, undo, realtime event.
Notification	created, delivered, read, cleared	створення, read/read-all, видалення прочитаних.
Report	open, inReview, resolved, rejected	призначення модератора, примітка, audit trail.
Announcement	draft, scheduled, sent, archived	планування, send-now, push-token

Сутність	Типові стани	Операції переходу
		delivery stats.

2.8 Правила валідації та узгодженості даних

Узгодженість даних забезпечується не одним механізмом, а кількома шарами. На вході API працюють DTO з class-validator. Потім сервіс перевіряє бізнес-умови, наприклад належність команди до турніру або право користувача редагувати матч. Після цього Mongoose-схема застосовує required-поля, enum-обмеження, defaults та індекси.

Такий багаторівневий підхід корисний для системи, де є багато ролей і пов'язаних сутностей. Якщо перевірити лише типи даних, можна пропустити логічну помилку: технічно коректний ObjectId може посилатися на команду, яка не бере участі в турнірі. Тому валідація не обмежується форматом JSON.

Окремо важливою є синхронізація денормалізованих полів. У мобільному UI зручніше швидко показувати назви команд, аватари або короткі статуси, але джерело істини повинно залишатися в основних колекціях. Для цього серверні сервіси мають оновлювати пов'язані поля разом із базовими змінами.

Таблиця 14 - Приклади правил валідації

Поле або об'єкт	Рівень перевірки	Суть правила
email	Auth DTO, User schema	формат email, унікальність, нормалізація регістру.
password	Auth DTO, AuthService	мінімальна довжина, bcrypt hash, ніколи не повертати passwordHash.
nickname	Players DTO, PlayersService, індекс	унікальність, довжина, перевірка перед створенням.
team slug	TeamsService, Team schema	унікальний slug, створення з назви, заборона дублювання.
tournament status	DTO, enum, service transitions	дозволені переходи між draft, registration, live, completed.

Поле або об'єкт	Рівень перевірки	Суть правила
match score	MatchesService	рахунок не може суперечити winnerTeamId і status.
map veto	MatchesService	послідовність ban/pick/sidePick не повинна ламати best-of формат.
role	Users DTO, RolesGuard	дозволені enum-ролі, перевірка на кожному захищеному endpoint.
notification route	NotificationsService	actionRoute має вести на існуючий екран або бути порожнім.
report target	ReportsService	targetType і targetId повинні відповідати реальній сутності.

2.9 Приклади структур даних

Для кращого розуміння інформаційного забезпечення нижче наведено спрощені приклади JSON-документів. Вони не дублюють повністю Mongoose-схеми, але показують, які групи даних зберігаються разом і чому документна модель є зручною для цього проєкту.

У документі матчу видно, що поряд із базовими полями зберігаються maps, mapVetoSteps, events і liveMetrics. Для реляційної моделі це потребувало б багатьох допоміжних таблиць, тоді як MongoDB дозволяє зберігати вкладені структури природніше.

```
{
  "tournamentId": "ObjectId",
  "status": "Live",
  "homeTeamId": "ObjectId",
  "awayTeamId": "ObjectId",
  "seriesScore": { "home": 1, "away": 0 },
  "maps": [
    { "name": "Mirage", "status": "Completed", "scoreHome": 13, "scoreAway":
10 }
  ],
  "mapVetoSteps": [
    { "teamId": "ObjectId", "action": "ban", "map": "Vertigo" }
  ],
  "events": [
```

```

    { "type": "scoreUpdate", "message": "Map 1 completed", "createdAt":
      "ISODate" }
  ]
}
```

Подібна структура є достатньо гнучкою для розвитку. Якщо надалі знадобляться детальніші статистичні події, їх можна додавати в `events` або `liveMetrics` без зміни всіх інших колекцій. Водночас ключові зв'язки, такі як `tournamentId`, `homeTeamId` і `awayTeamId`, залишаються явними.

Для повідомлень важливим є `actionRoute`. Саме це поле поєднує серверну подію з мобільною навігацією: користувач натискає повідомлення і переходить не просто на головний екран, а на конкретний матч, турнір, профіль або адміністративний розділ.

Висновки до розділу 2

1. Розроблено інформаційне забезпечення на базі MongoDB/Mongoose з окремими колекціями для користувачів, гравців, команд, турнірів, матчів, повідомлень, модерації, аудиту та налаштувань.
2. Обґрунтовано вибір документної СУБД, оскільки система містить багато вкладених структур і гнучких конфігурацій.
3. Описано ключові поля, зв'язки та індекси, що забезпечують унікальність, швидку вибірку та підтримку основних сценаріїв.
4. Визначено принципи обміну даними між мобільним клієнтом і backend: REST JSON, JWT, DTO validation, guards і realtime-канал для live-подій.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмної системи

Програмна система реалізована як монорепозиторій, що містить мобільний застосунок Flutter у каталозі apps/mobile, backend NestJS API у каталозі apps/api, Docker Compose конфігурацію, Nginx reverse проху та допоміжну документацію. Така структура дозволяє окремо розвивати клієнтську та серверну частини, але зберігати їх у спільному проєктному контексті.

Клієнтська частина відповідає за мобільний UI, навігацію, локальне збереження, звернення до API, відображення даних, формування запитів і користувацькі сценарії. Серверна частина відповідає за бізнес-логіку, перевірку ролей, роботу з MongoDB, кешування, повідомлення, генерацію сітки, модерацію, аналітику та Swagger-документацію.

Архітектура відповідає принципу розділення відповідальності. Flutter screens не працюють напряму з HTTP, а використовують провайдери та репозиторії. NestJS controllers не містять складної бізнес-логіки, а делегують її services. Схеми бази даних відокремлено від DTO, що дозволяє окремо контролювати формат зовнішнього API і внутрішню модель зберігання.

Таблиця 15 - Компоненти архітектури

Компонент	Призначення
Flutter mobile app	Dart, Flutter, Riverpod, GoRouter, Dio, Hive, Secure Storage, Socket.IO client, fl_chart.
NestJS API	TypeScript, controllers, services, DTO, guards, Swagger, Mongoose, JWT, bcrypt.
MongoDB	Основне сховище користувачів, гравців, команд, турнірів, матчів, повідомлень і аудиту.
Redis	Опційне кешування dashboard/analytics і підтримка pub/sub для realtime-сценаріїв.
Socket.IO /live	Підписка клієнтів на tournament/user події та broadcast оновлень матчів, турнірів, повідомлень.
Docker/Nginx	Локальне розгортання MongoDB, Redis, API і проксі для /api, /docs,

Компонент	Призначення
	/live.

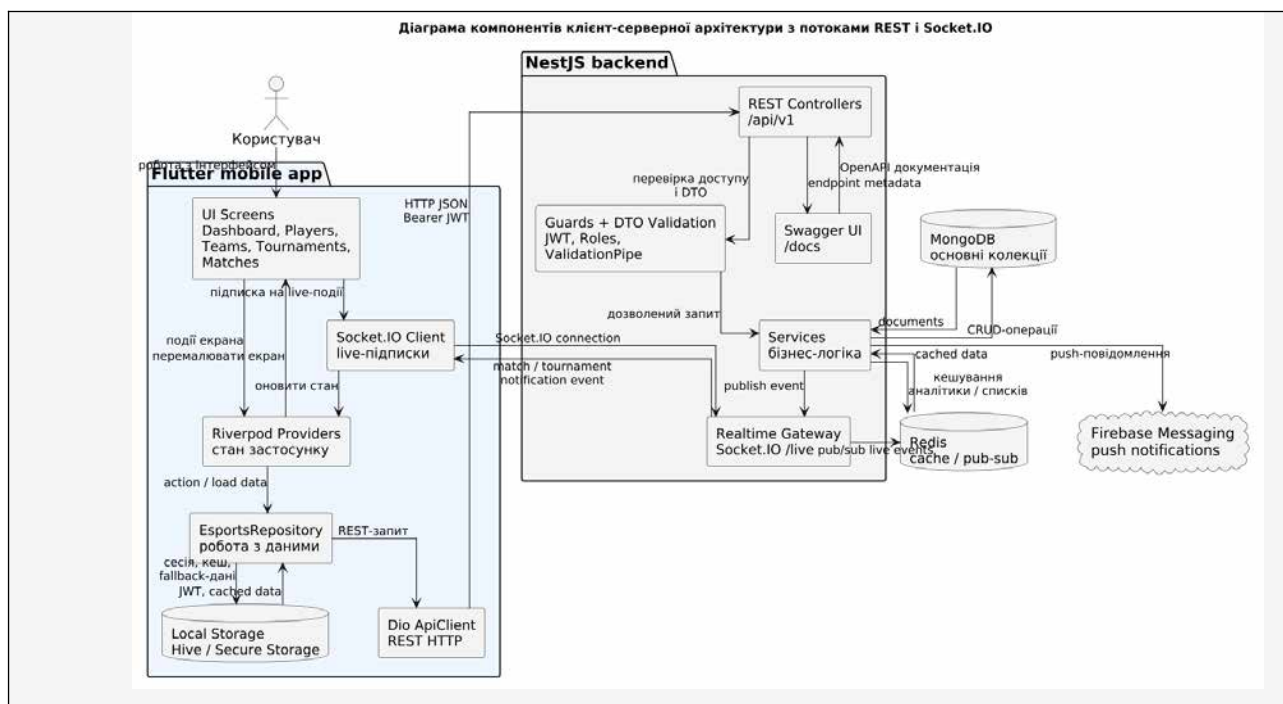


Рис. 7 - Діаграма компонентів клієнт-серверної архітектури з потоками REST і Socket.IO

3.2 Серверна частина NestJS API

Backend побудований на NestJS 11 і має модульну структуру. У AppModule підключено ConfigModule, ThrottlerModule, MongooseModule, RedisModule, HealthModule, AuditLogsModule, PlatformSettingsModule, AnnouncementsModule, FeaturedModule, UsersModule, AuthModule, SessionsModule, TeamsModule, PlayersModule, TournamentsModule, MatchesModule, NotificationsModule, RealtimeModule, ReportsModule та AnalyticsModule.

Глобальні guards застосовуються на рівні застосунку: ThrottlerGuard обмежує частоту запитів, JwtAuthGuard перевіряє автентифікацію, PlatformAccessGuard враховує платформні налаштування, RolesGuard перевіряє рольові вимоги endpoint. Публічні маршрути позначаються декоратором Public, а рольові - декоратором Roles.

Конфігурація завантажується з .env і містить PORT, CORS_ORIGIN, MONGODB_URI, MONGODB_DB_NAME, JWT_SECRET, JWT_EXPIRES_IN,

JWT_REFRESH_EXPIRES_IN, REDIS_ENABLED, REDIS_HOST, REDIS_PORT, REDIS_DB, FIREBASE_SERVICE_ACCOUNT_JSON і FIREBASE_SERVICE_ACCOUNT_PATH. Це дозволяє запускати систему як у Docker, так і локально.

Таблиця 16 - Функціональні модулі backend

Модуль	Функції
AuthModule	Реєстрація, логін, social login, зміна пароля, видача JWT, робота з сесіями.
UsersModule	Адміністративне керування користувачами, ролями, статусом, мовою, notification preferences.
PlayersModule	Публічні та приватні профілі гравців, валідація нікнейму, аватар, approve, CRUD.
TeamsModule	Команди, чернетки, autosave, publish/archive/restore/duplicate, ростер, інвайти.
TournamentsModule	Турніри, шаблони, lifecycle, check-in, seeding, bracket, swiss, disputes, results.
MatchesModule	Матчі, score, quick edit, maps, veto, events, undo, live sync, notifications.
NotificationsModule	Стрічка повідомлень, read/read-all, preferences, favorite players, maintenance notice.
ReportsModule	Користувацькі скарги, admin moderation, audit logs.
AnalyticsModule	Dashboard, top players, top teams, player statistics report.
AnnouncementsModule	Оголошення, scheduled/send-now, push-token, Firebase hooks.
FeaturedModule	Homepage featured content, upsert і reorder для сутностей.
PlatformSettingsModule	Maintenance mode, registrations, visibility, moderation sensitivity, notification defaults.

Players

GET	/api/v1/players	List player profiles with optional filters	🔒
POST	/api/v1/players	Create a player profile	🔒
GET	/api/v1/players/validate-nickname	Validate nickname availability	🔒
GET	/api/v1/players/me/profile	Get current authenticated player profile	🔒
POST	/api/v1/players/me/profile	Create own player profile	🔒
PATCH	/api/v1/players/me/profile	Patch current authenticated player profile	🔒
GET	/api/v1/players/{id}	Get a player profile with analytics payloads	🔒
PATCH	/api/v1/players/{id}	Update a player profile	🔒
DELETE	/api/v1/players/{id}	Delete a player profile	🔒
POST	/api/v1/players/me/profile/avatar	Upload player avatar and return persisted URL	🔒
POST	/api/v1/players/{id}/approve	Approve a player profile	🔒

Teams

GET	/api/v1/teams	List esports teams	🔒
POST	/api/v1/teams	Create a team	🔒
GET	/api/v1/teams/admin/me	List teams managed by current team admin	🔒
POST	/api/v1/teams/drafts	Create team draft (wizard endpoint)	🔒
PATCH	/api/v1/teams/drafts/{id}/autosave	Autosave existing team draft	🔒
GET	/api/v1/teams/{id}	Get a team profile	🔒
PATCH	/api/v1/teams/{id}	Update a team	🔒
DELETE	/api/v1/teams/{id}	Archive a team (soft delete for team admins)	🔒
POST	/api/v1/teams/{id}/publish	Publish team profile (Draft -> Active)	🔒
POST	/api/v1/teams/{id}/archive	Archive team profile	🔒
POST	/api/v1/teams/{id}/restore	Restore archived team profile	🔒
POST	/api/v1/teams/{id}/duplicate	Duplicate team into a new draft profile	🔒
PATCH	/api/v1/teams/{id}/roster/quick	Quick edit roster, captain, coach and statuses	🔒
POST	/api/v1/teams/{id}/roster/undo	Undo last quick roster action	🔒
GET	/api/v1/teams/{id}/invitations	List invitations for a specific team	🔒
POST	/api/v1/teams/{id}/invitations	Invite a player account into this team	🔒
POST	/api/v1/teams/{id}/invitations/{invitationId}/resend	Resend existing invitation and extend expiry	🔒
PATCH	/api/v1/teams/{id}/invitations/{invitationId}/revoke	Revoke pending invitation	🔒
GET	/api/v1/teams/invitations/me	List invitations for current player account	🔒
PATCH	/api/v1/teams/invitations/{teamId}/{invitationId}/accept	Accept a team invitation	🔒
PATCH	/api/v1/teams/invitations/{teamId}/{invitationId}/decline	Decline a team invitation	🔒

Tournaments			
GET	/api/v1/tournaments	List tournaments	🔒
POST	/api/v1/tournaments	Create a tournament	🔒
GET	/api/v1/tournaments/operator/me	List tournaments managed by current operator	🔒
POST	/api/v1/tournaments/drafts	Create a tournament draft (wizard step flow)	🔒
PATCH	/api/v1/tournaments/drafts/{id}/autosave	Autosave tournament draft changes	🔒
GET	/api/v1/tournaments/{id}	Get tournament details, bracket, and stats	🔒
PATCH	/api/v1/tournaments/{id}	Update tournament details	🔒
DELETE	/api/v1/tournaments/{id}	Delete a tournament	🔒
POST	/api/v1/tournaments/{id}/publish	Publish tournament (Draft → Registration)	🔒
POST	/api/v1/tournaments/{id}/start	Start tournament (Registration → Live)	🔒
POST	/api/v1/tournaments/{id}/complete	Complete tournament (Live → Completed)	🔒
POST	/api/v1/tournaments/{id}/duplicate	Duplicate tournament into a new draft	🔒
POST	/api/v1/tournaments/{id}/templates/{templateKey}	Apply a template preset onto a draft tournament configuration	🔒
POST	/api/v1/tournaments/{id}/register-team	Register a team into a tournament	🔒
POST	/api/v1/tournaments/{id}/bracket/generate	Generate or regenerate tournament bracket with selected seeding rules	🔒
POST	/api/v1/tournaments/{id}/lifecycle/transition	Transition tournament lifecycle state	🔒
POST	/api/v1/tournaments/{id}/swiss/generate-next-round	Generate next Swiss round pairings	🔒
PATCH	/api/v1/tournaments/{id}/check-in/settings	Update check-in settings	🔒
POST	/api/v1/tournaments/{id}/check-in/open	Open tournament check-in window	🔒
POST	/api/v1/tournaments/{id}/check-in/close	Close tournament check-in window	🔒
POST	/api/v1/tournaments/{id}/check-in/team/{teamId}	Check in team for tournament	🔒
PATCH	/api/v1/tournaments/{id}/check-in/team/{teamId}	Manage team check-in status	🔒
POST	/api/v1/tournaments/{id}/seeding/preview	Preview tournament seeding order	🔒
POST	/api/v1/tournaments/{id}/seeding/finalize	Finalize and lock tournament seeding	🔒
PATCH	/api/v1/tournaments/{id}/roster-lock/settings	Update roster lock settings	🔒
POST	/api/v1/tournaments/{id}/roster-lock/activate	Activate roster lock for tournament teams	🔒
POST	/api/v1/tournaments/{id}/roster-lock/release	Release roster lock (admin override)	🔒
GET	/api/v1/tournaments/{id}/disputes	List tournament disputes	🔒
POST	/api/v1/tournaments/{id}/disputes	Submit tournament dispute	🔒
PATCH	/api/v1/tournaments/{id}/disputes/{disputeId}	Resolve/reject/escalate tournament dispute	🔒
POST	/api/v1/tournaments/{id}/auto-advance	Trigger auto-advance and persist upon	🔒
PATCH	/api/v1/tournaments/{id}/bracket/matches/{matchId}	Update bracket match score/state and propagate winner to next stage	🔒
GET	/api/v1/tournaments/{id}/results	Get tournament results and prize distribution	🔒
POST	/api/v1/tournaments/{id}/results/generate	Generate or regenerate tournament result slots	🔒
PATCH	/api/v1/tournaments/{id}/results/distribution	Update tournament prize distribution	🔒
PATCH	/api/v1/tournaments/{id}/results/override	Override locked placement or assigned team	🔒
DELETE	/api/v1/tournaments/{id}/teams/{teamId}	Remove a team from a tournament (allowed before live only)	🔒

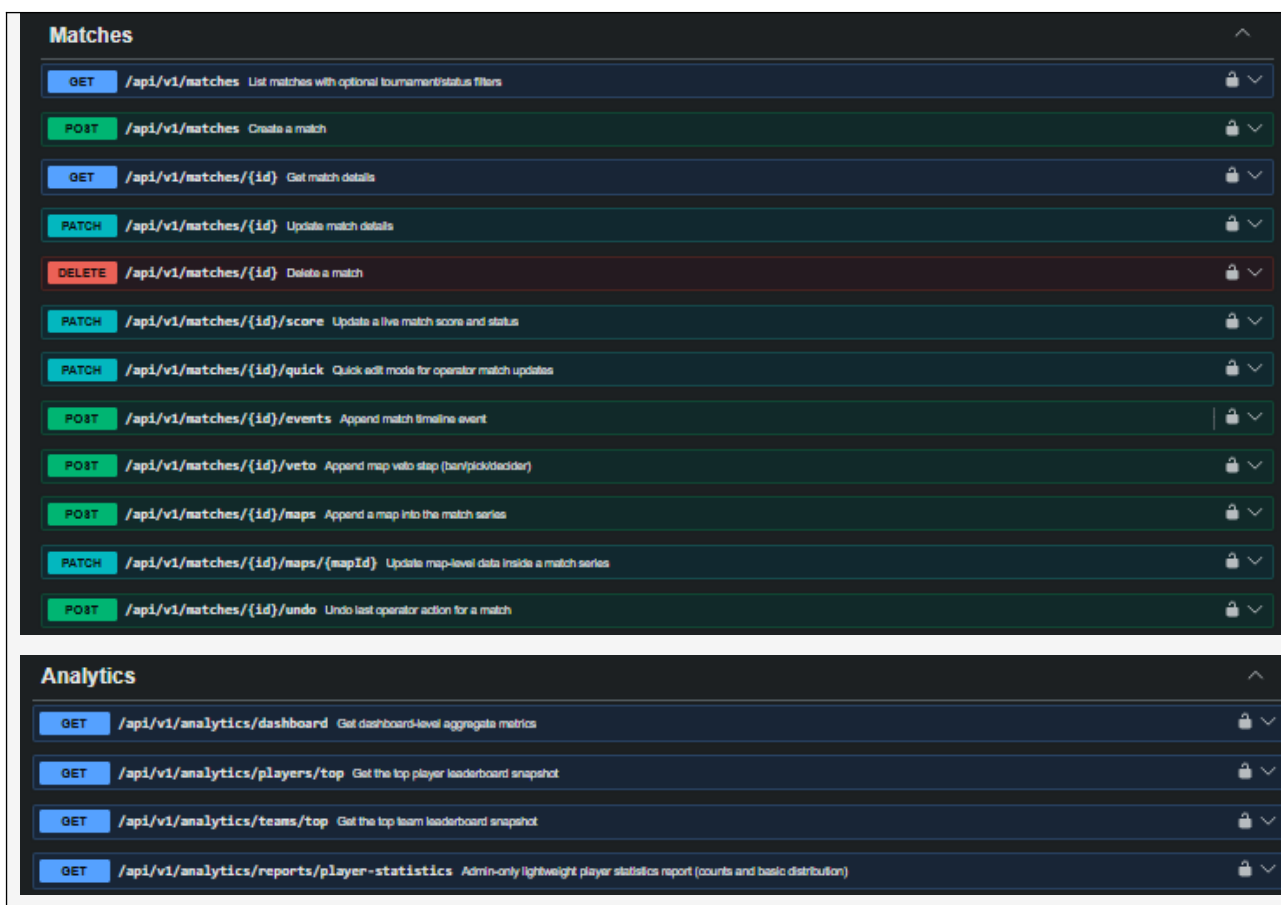


Рис. 8 - Скріншоти Swagger /docs з групами endpoint Players, Teams, Tournaments, Matches, Analytics

Автентифікація та сесії

AuthService реалізує register, login, socialLogin, changePassword і me. Пароль хешується за допомогою bcrypt, а після успішного входу користувач отримує access token, refresh token і sessionId. SessionsService зберігає активні сесії, дає змогу переглядати їх, завершувати конкретну сесію або всі інші сесії.

У мобільному застосунку сесія зберігається в захищеному сховищі та індексі акаунтів. Це дозволяє підтримувати активний акаунт, відновлювати вхід після перезапуску і коректно виходити з поточної сесії.

Користувачі та ролі

UsersController надає адміністративні endpoint для списку користувачів, детальної картки, зміни ролей, статусу, верифікації player profile, мови та видалення користувача. Це забезпечує повний цикл адміністрування облікових записів.

Ролі представлені enum Role і використовуються одночасно в backend guards та мобільній логіці відображення. Рольова модель важлива для розмежування можливостей адміністратора, турнірного оператора, менеджера команди, гравця і глядача.

Гравці

PlayersModule підтримує створення профілю, власний профіль користувача, редагування, валідацію нікнейму, завантаження або URL-аватар, approve і видалення. При завантаженні аватара файл перевіряється як image і обмежується розміром.

Профіль гравця є основним довідковим екраном для кіберспортивної бази. Він містить ідентифікаційні дані, команду, дисципліну, країну, статистику, історію матчів, availability і visibility.

Команди

TeamsModule реалізує створення команди, роботу з чернетками, autosave, публікацію, архівацію, відновлення, дублювання, швидке редагування ростеру, undo для ростеру та систему запрошень. Team Admin або Admin може керувати складом, а Player приймає чи відхиляє інвайти.

Командний модуль синхронізує rosterPlayerIds і teamId у профілях гравців. Це важливо, щоб список команди та профіль гравця не суперечили одне одному.

Турніри

TournamentsModule є найбільшим модулем системи. Він підтримує чернетки, autosave, створення, оновлення, publish/start/complete, duplicate, шаблони quick_cup, pro_series, round_robin_league, реєстрацію команд, генерацію bracket, lifecycle transition, swiss generate next round, check-in, seeding, roster lock, disputes, auto-advance, bracket match updates, results generation, distribution override і видалення.

Турнірний модуль показує комплексність дипломного проєкту: одна сутність має життєвий цикл, учасників, стадії, групи, швейцарські раунди, сітку, результати, блокування ростеру, спори та автоматичне просування команд.

Матчі

MatchesModule підтримує створення, список, деталі, редагування, оновлення score, quick edit, додавання events, veto, maps, оновлення конкретної карти, undo і видалення. Після мутації матч транслюється через RealtimeGateway і може синхронізувати прогрес турніру.

Матчі містять широкий набір live-полів: карти, map veto, події, pauseEvents, technicalIssueInfo, forfeitInfo, streamUrl, demoLink, VODs і demos. Завдяки цьому один модуль покриває як простий результат, так і повноцінний live-супровід.

Повідомлення, модерація та аналітика

NotificationsModule формує персональні повідомлення, відмітку про прочитання, очищення прочитаних, налаштування користувача, favorite players і системні maintenance notices. ReportsModule підтримує скарги, перегляд власних звернень, admin-список і модераційні дії.

AnalyticsModule агрегує dashboard, топи гравців і команд та звіт player-statistics. Для звіту використовується countDocuments і aggregate за discipline та country, а результат кешується в Redis на короткий період.

3.3 Деталізація адміністративних функцій

Адміністративна частина є важливою, оскільки вона перетворює застосунок із простого клієнтського довідника на керовану платформу. AdminPanelScreen об'єднує кілька підекранів: users, user detail, roles, teams, tournaments, approvals, featured, platform settings, announcements, analytics і audit log.

Підекран користувачів дає змогу шукати акаунти за displayName або email, фільтрувати за роллю чи статусом, відкривати деталі, змінювати ролі, блокувати

або деактивувати користувача, верифікувати профіль гравця. Ці дії відповідають endpoint `UsersModule` і захищені рольовим доступом.

Модераційна частина працює зі зверненнями користувачів. `Report` може бути відкритим, у розгляді, вирішеним, відхиленим або таким, що призвів до дії. Для прозорості дій створюється `moderation audit log`, де фіксується адміністратор, дія, попередній і новий стан.

`Featured`-контент дозволяє закріпити турнір, команду або гравця на головному екрані. Адміністратор може змінювати порядок, видимість, `publishAt` та `expiresAt`. Це демонструє розділення між основною сутністю і способом її промо-відображення на `dashboard`.

`Platform settings` керують глобальними правилами: `maintenance mode`, `registrationsOpen`, `publicVisibility`, `tournamentApprovalRequired`, `teamApprovalRequired`, `analyticsCollection`, `moderation sensitivity` і `notification defaults`. Завдяки цьому поведінку платформи можна змінювати без редагування коду.

Таблиця 17 - Підсистеми адміністративної панелі

Підсистема	Операції
Users	Пошук, пагінація, роль, статус, деталізація, <code>verification</code> .
Roles	Призначення або вилучення <code>Admin</code> , <code>TournamentOperator</code> , <code>TeamAdmin</code> , <code>Player</code> , <code>Viewer</code> .
Reports	Список скарг, <code>severity</code> , <code>status</code> , <code>moderate action</code> , <code>note</code> .
Teams	Адміністративний перегляд команд і контроль життєвого циклу.
Tournaments	Адміністративний контроль турнірів і статусів.
Featured	Закріплення сутностей на головній, <code>reorder</code> , <code>visibility</code> .
Platform settings	<code>Maintenance</code> , <code>registrations</code> , <code>public visibility</code> , <code>approval required</code> , <code>moderation</code> .
Announcements	<code>Draft</code> , <code>scheduled</code> , <code>send-now</code> , <code>audience</code> , <code>delivery statistics</code> .
Analytics	<code>Dashboard</code> , <code>player statistics report</code> , <code>cached aggregates</code> .
Audit log	Перегляд критичних змін і дій адміністраторів.

3.4 Мобільний застосунок Flutter

Мобільний клієнт реалізовано на Flutter з використанням Riverpod для стану, GoRouter для навігації, Dio для HTTP, Hive і Flutter Secure Storage для локального збереження, Socket.IO client для realtime, fl_chart для графіків, cached_network_image для мережеских зображень, image_picker для аватарів і flutter_localizations для локалізації.

Основний маршрутизатор містить публічні екрани splash і login, shell-навігацію з dashboard, tournaments, matches, players і profile, а також окремі маршрути для оператора турнірів, team admin, admin panel, notifications, details pages і wizard-форм. Це формує повноцінну мобільну структуру, а не одну демонстраційну сторінку.

AdaptiveShell забезпечує основну навігаційну оболонку, швидкий пошук і перехід між розділами. UI побудований у темній кіберспортивній стилістиці, але головну роль відіграє функціональність: списки, фільтри, вкладки, картки, форми, wizard-кроки, детальні екрани та адміністративні sheet/dialog сценарії.

Таблиця 18 - Основні екрани мобільного застосунку

Екран	Призначення
LoginScreen	Вхід, реєстрація, перемикач режимів, вибір ролі, обробка помилок backend.
DashboardScreen	Стартова панель, featured tournament, quick actions, live highlights, onboarding профілю.
PlayersScreen / TeamsScreen	Пошук, фільтри, перемикач Players/Teams, рядки гравців і команд.
PlayerProfileScreen	Профіль, overview, stats, rating trend LineChart, gear, match history.
MyPlayerProfileScreen	Wizard власного профілю, аватар, запрошення команди, валідація nickname.
TournamentsScreen	Список турнірів, статусні chips, featured tournament, перехід до деталей.
TournamentDetailScreen	Overview, bracket, groups, swiss, participants, disputes, results, operator actions.

Екран	Призначення
TournamentWizardScreen	Створення/редагування турніру, шаблони, регіон, дати, check-in, seeding, preview.
MatchesScreen / MatchDetailScreen	Список матчів, групування за турнірами, maps, veto, timeline, media, result editor.
TeamAdminScreen / TeamWizardScreen	Керування командами, створення, редагування, roster quick edit, invitations.
AdminPanelScreen	Користувачі, ролі, команди, турніри, approvals, featured, platform settings, announcements, analytics, audit log.
NotificationsScreen	Стрічка повідомлень, пріоритети, категорії, route-action, анімації.

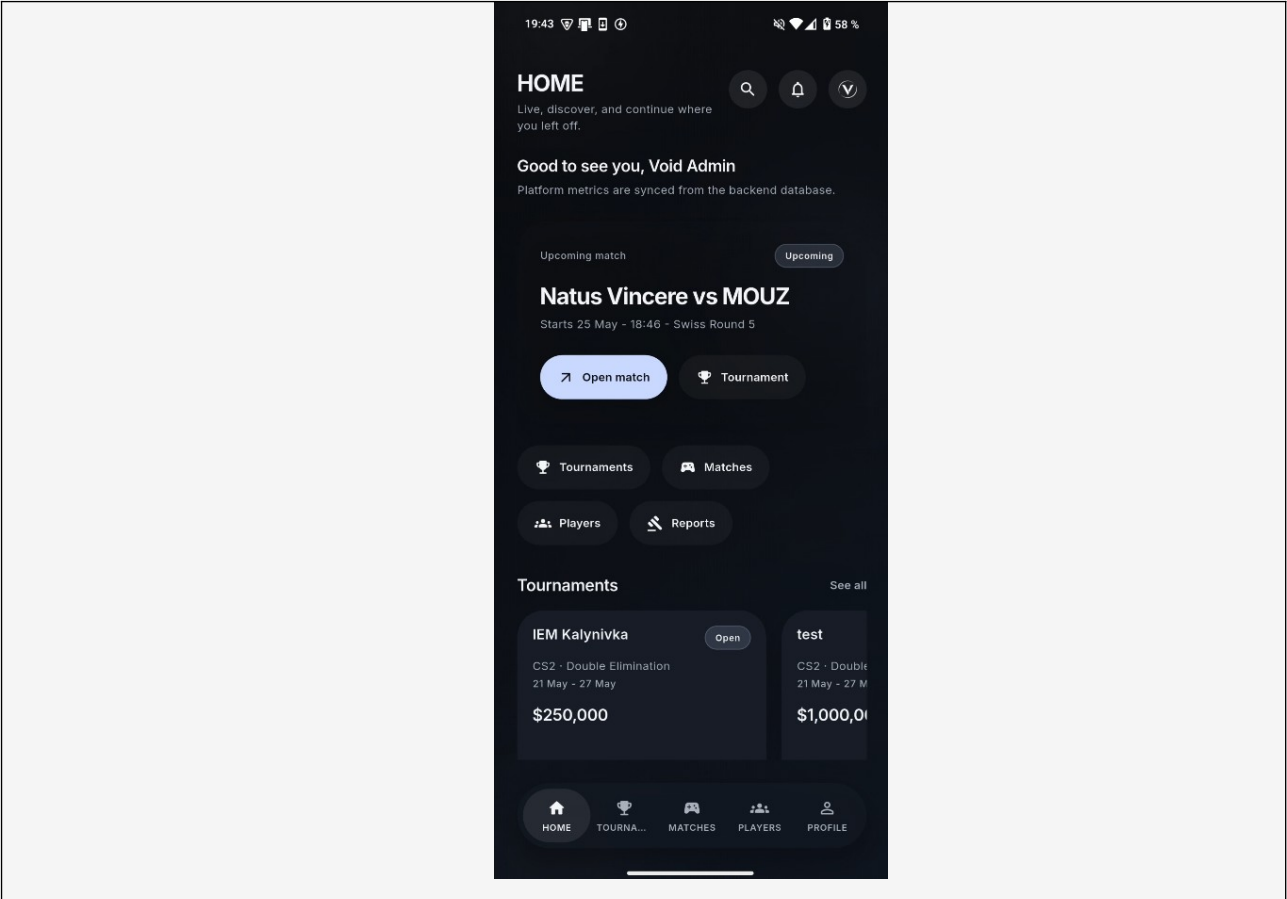


Рис. 9 - Головний екран DashboardScreen з featured tournament, quick actions і live highlights

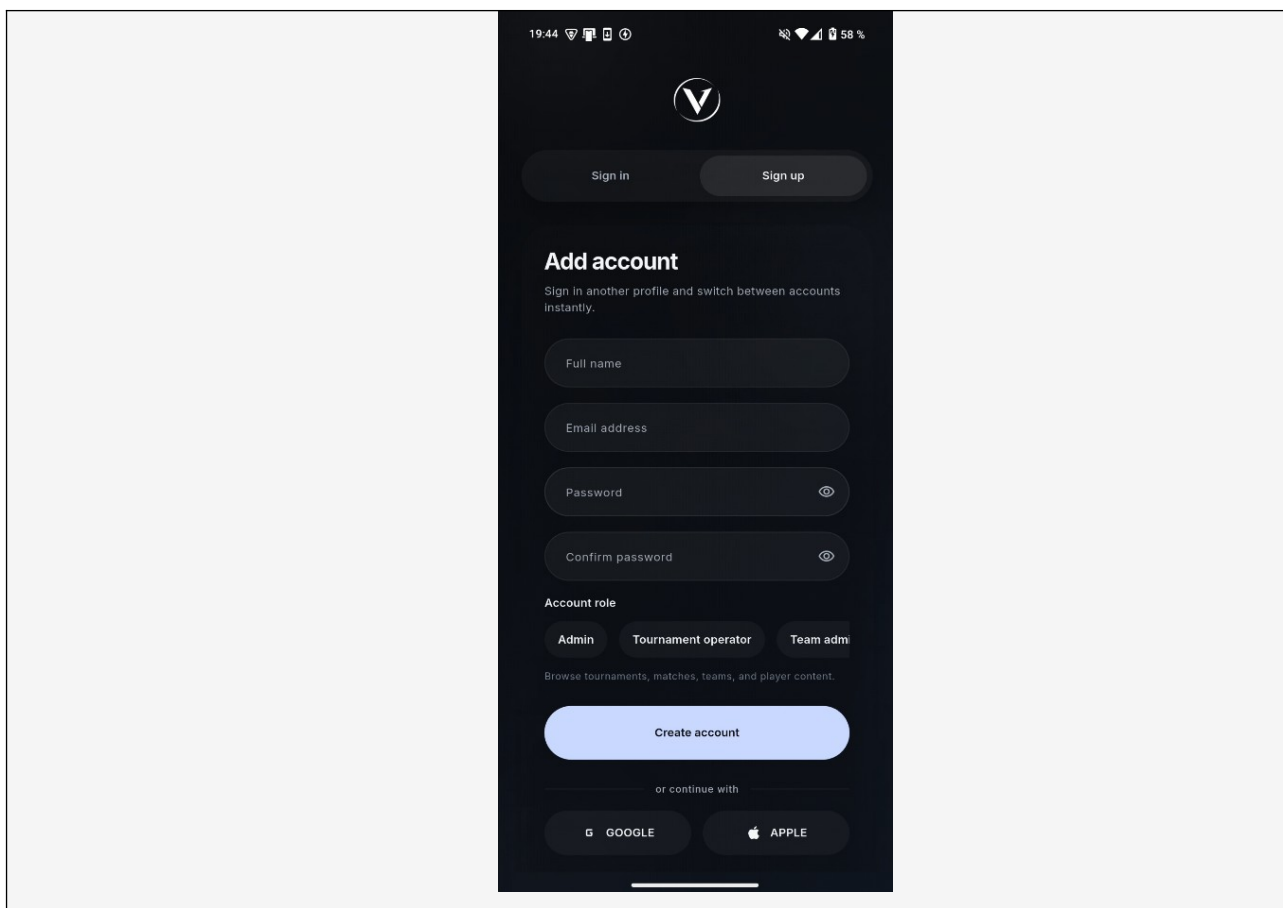


Рис. 10 - Екран LoginScreen у режимі реєстрації з вибором ролі користувача

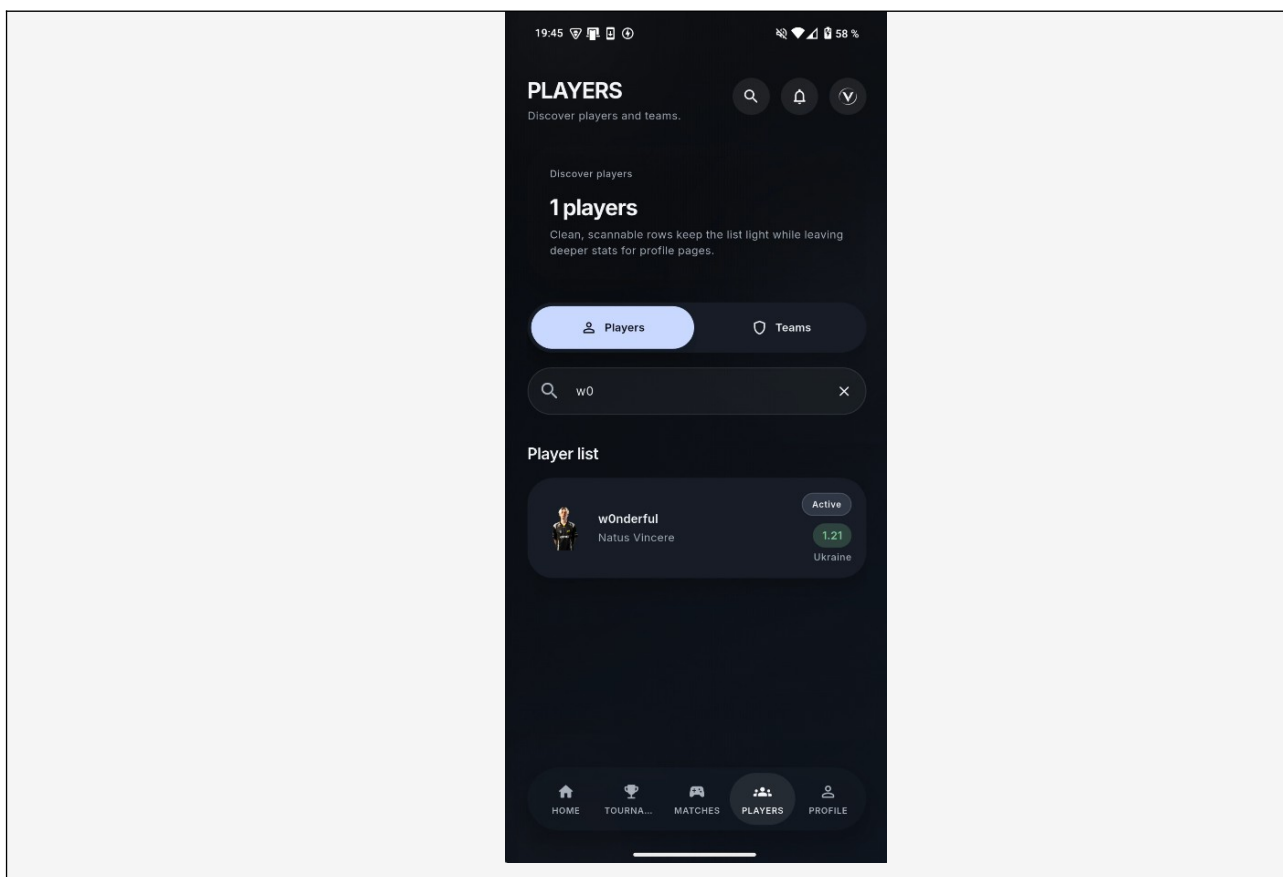


Рис. 11 - Екран PlayersScreen з введеним пошуковим запитом і відфільтрованими гравцями

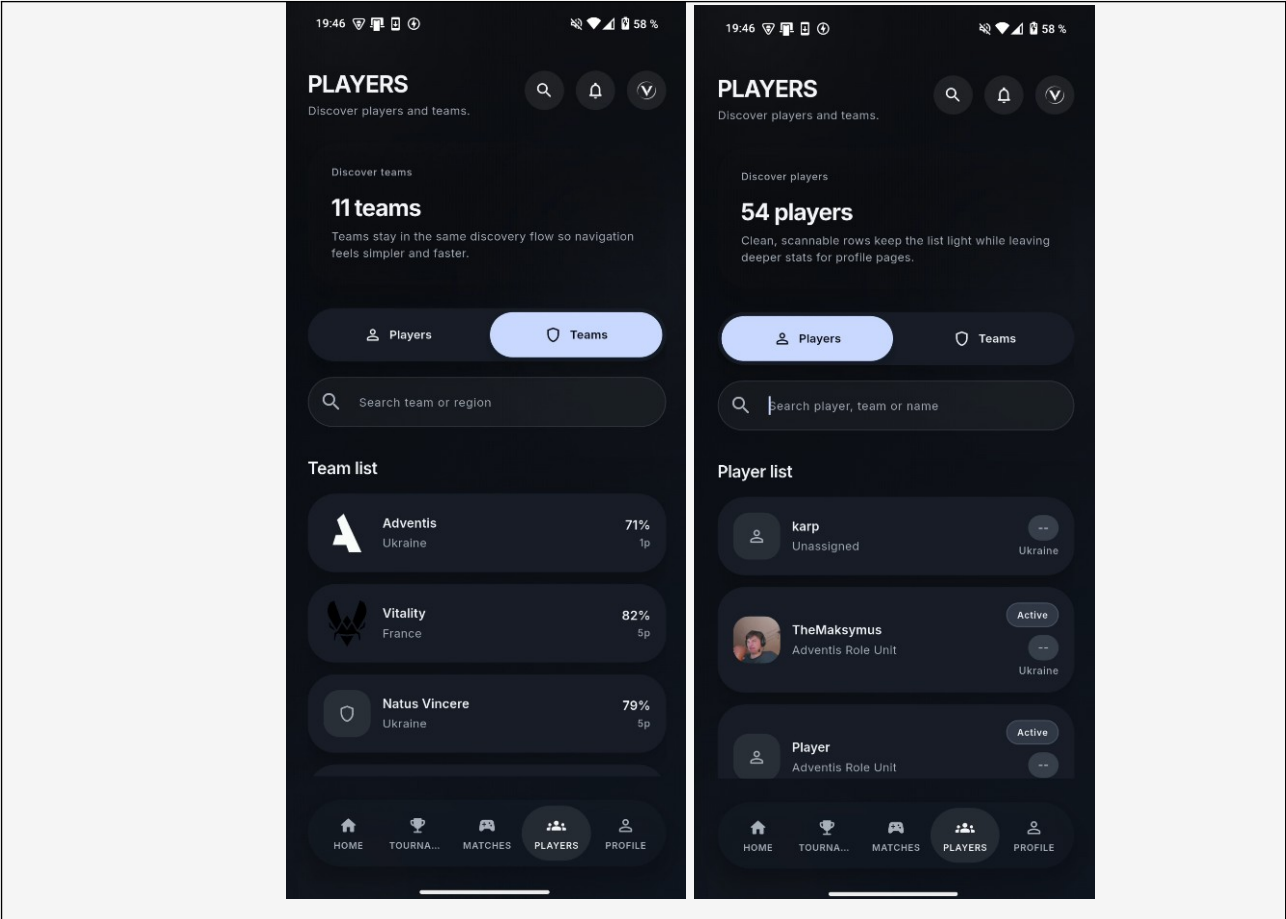


Рис. 12 - Екран Teams/Players у режимі списку команд із логотипами та країнами

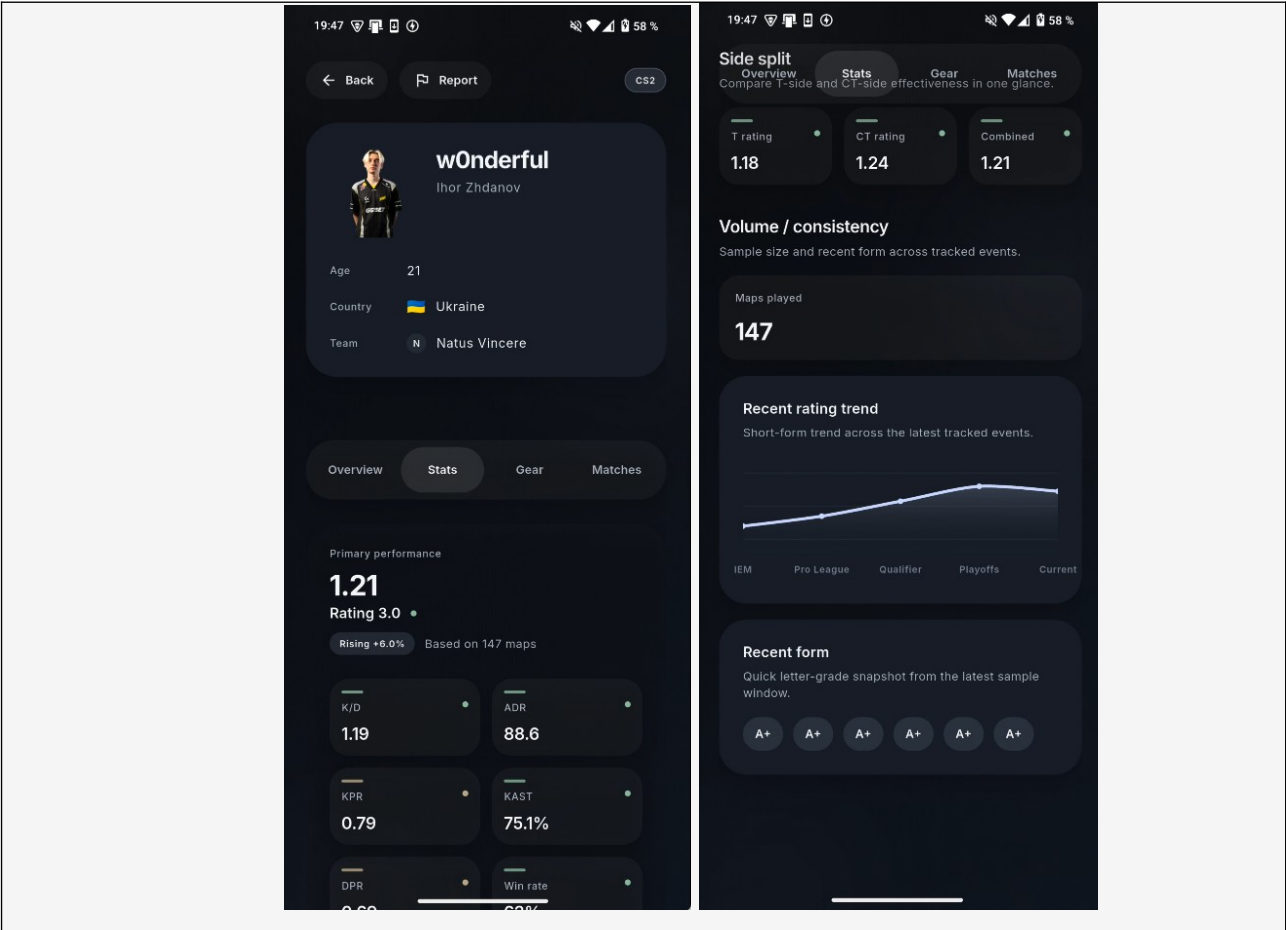


Рис. 13 - Екран PlayerProfileScreen з блоком rating trend LineChart

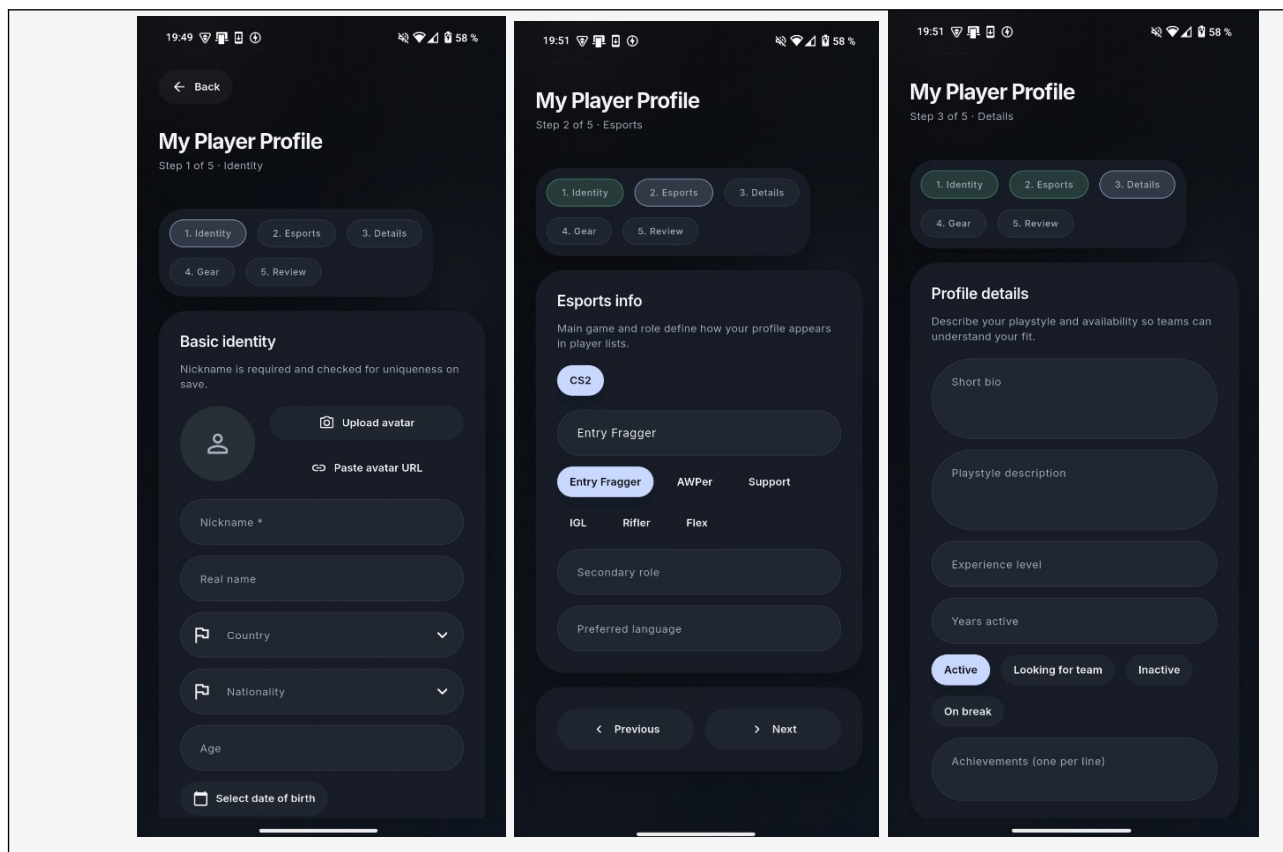
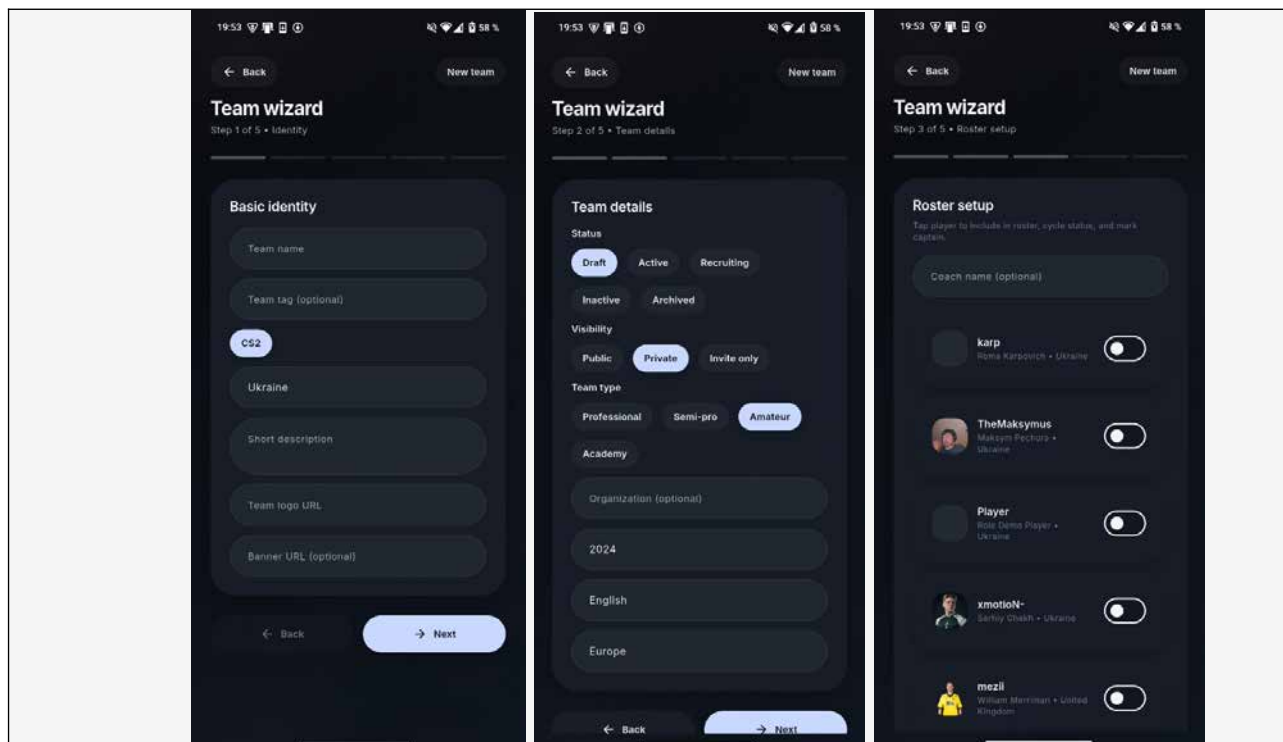


Рис. 14 - Екран MyPlayerProfileScreen з аватаром і кроками wizard-профілю



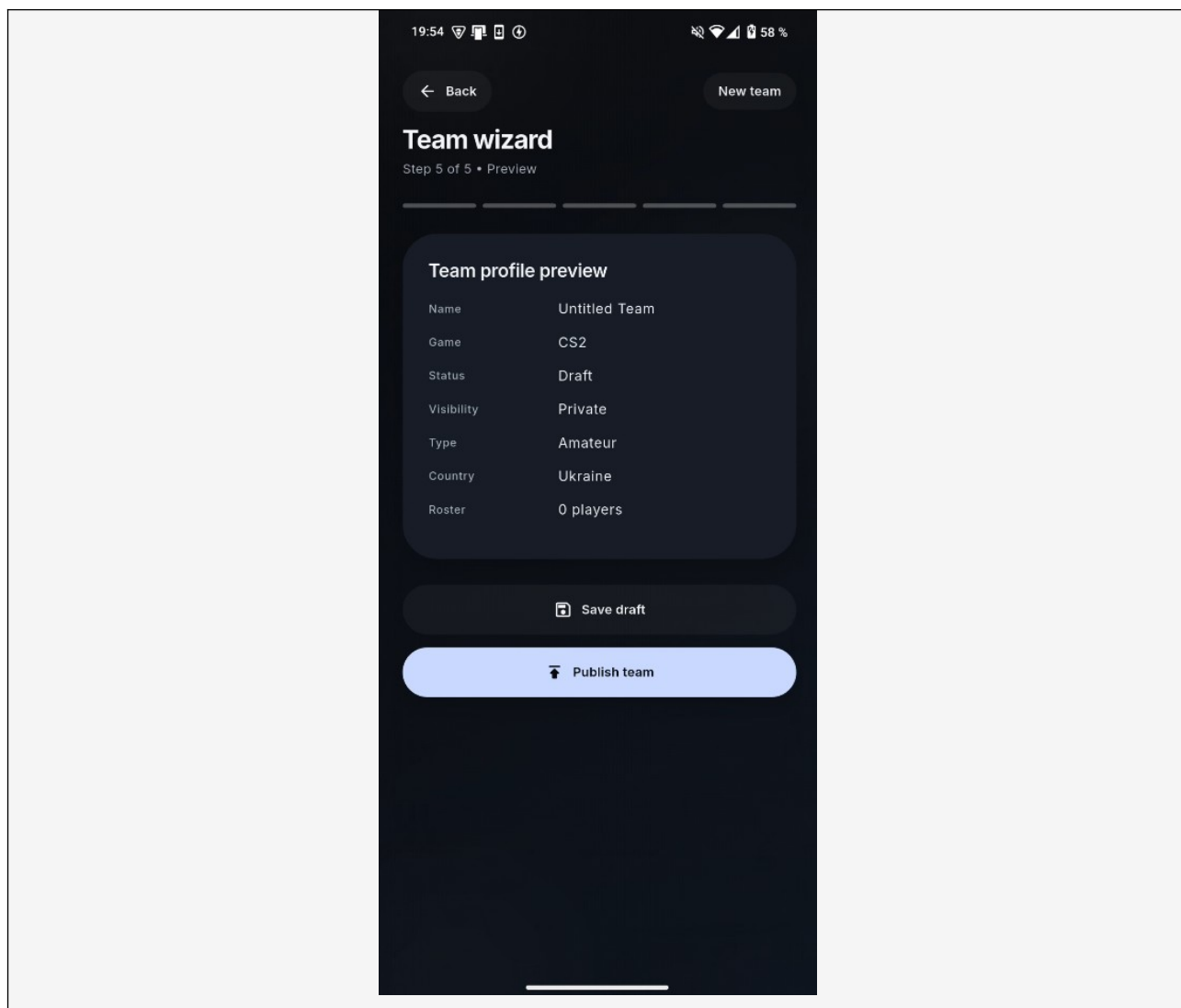


Рис. 15 - Экран TeamAdminScreen із quick roster sheet

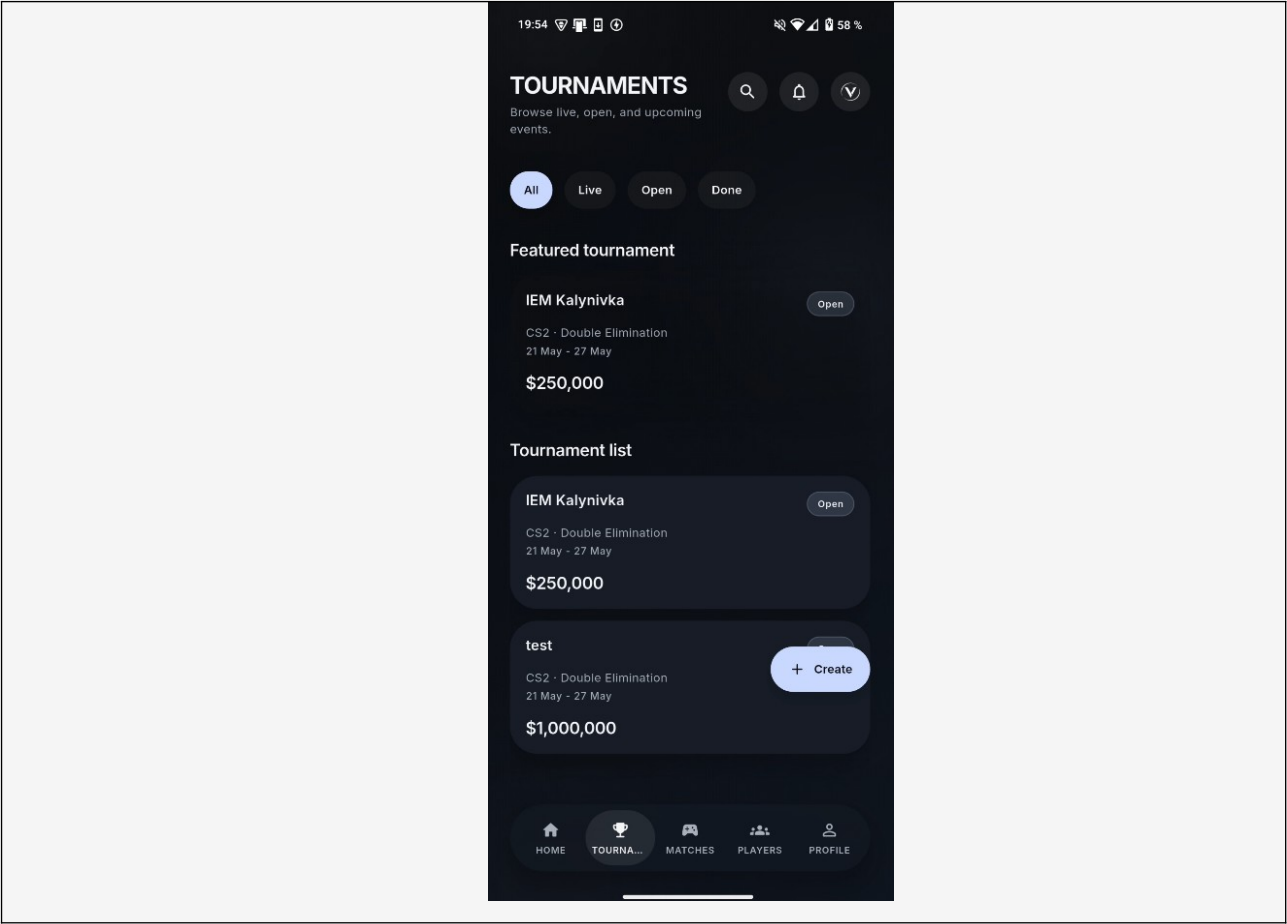
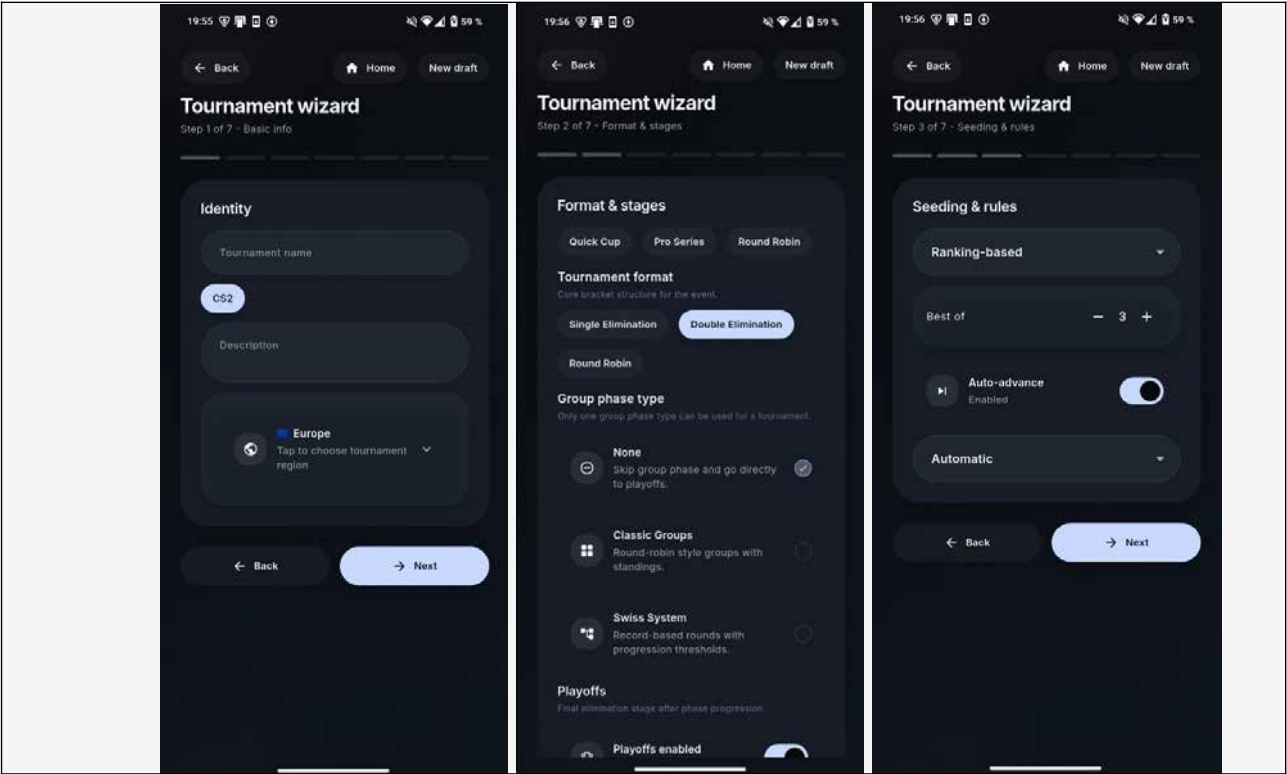


Рис. 16 - Экран TournamentsScreen зі статусними chips All/Live/Open/Done



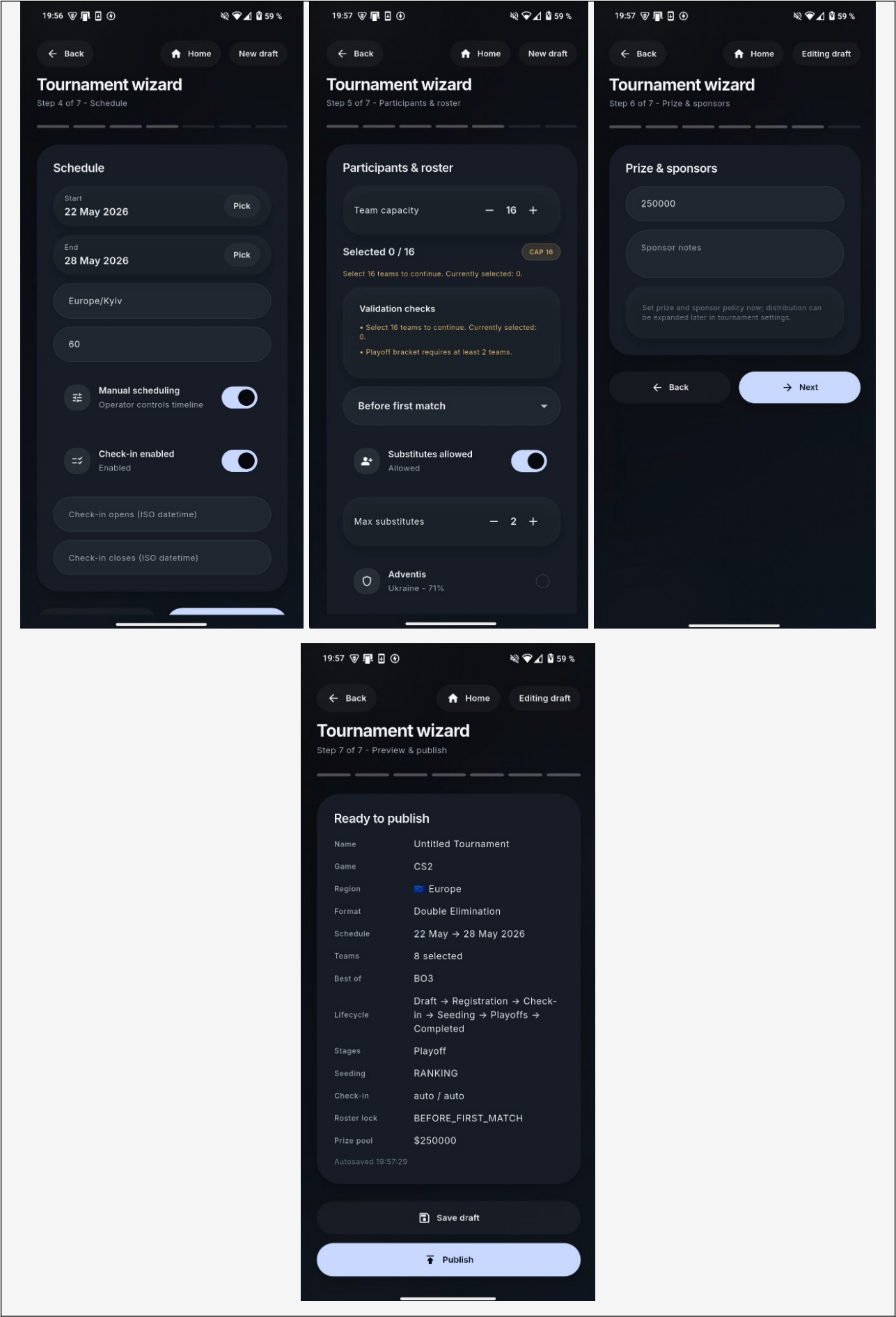


Рис. 17 - Екран TournamentWizardScreen зі створенням турніру та template presets

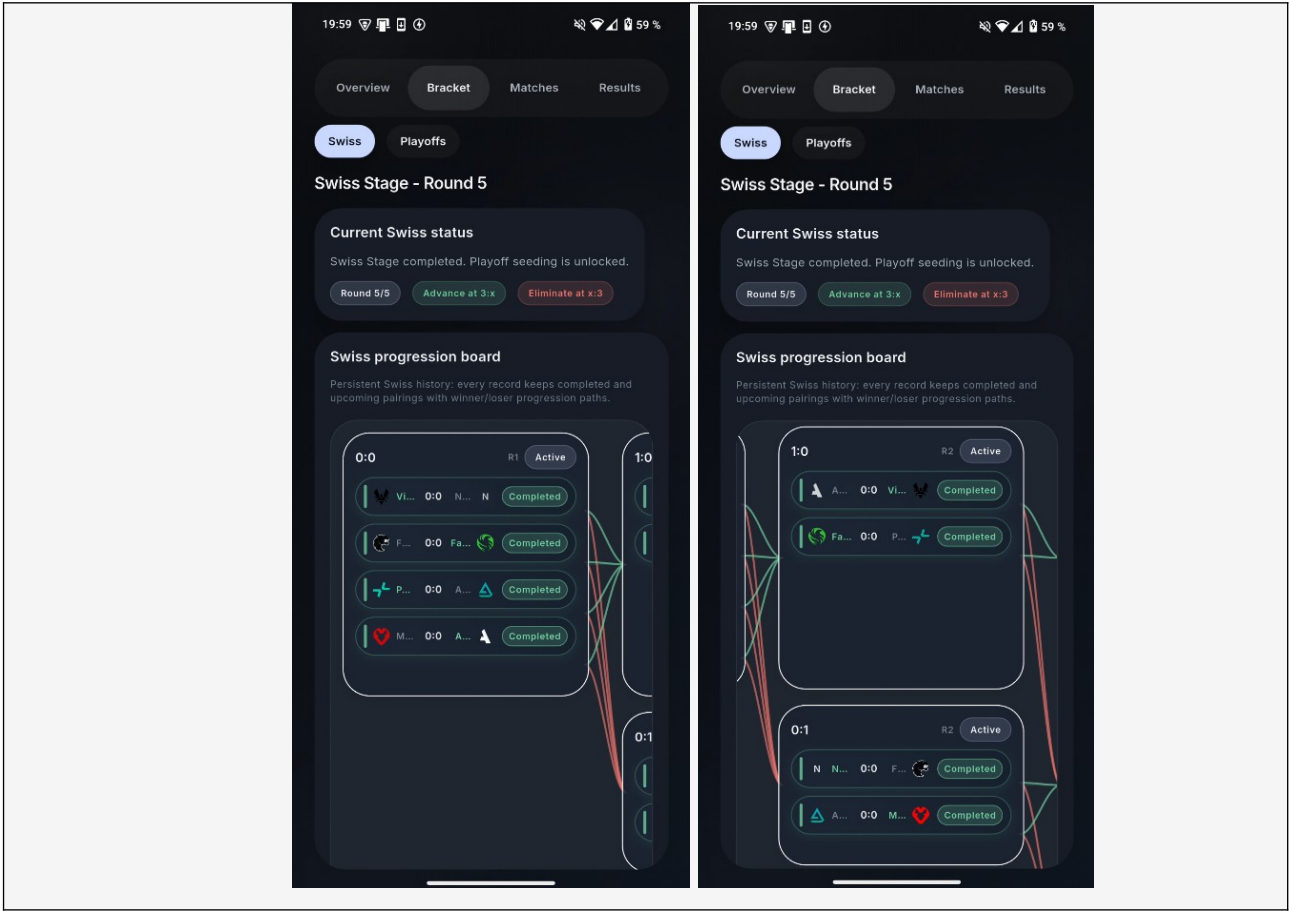


Рис. 18 - Екран TournamentDetailScreen на вкладці bracket із турнірною сіткою

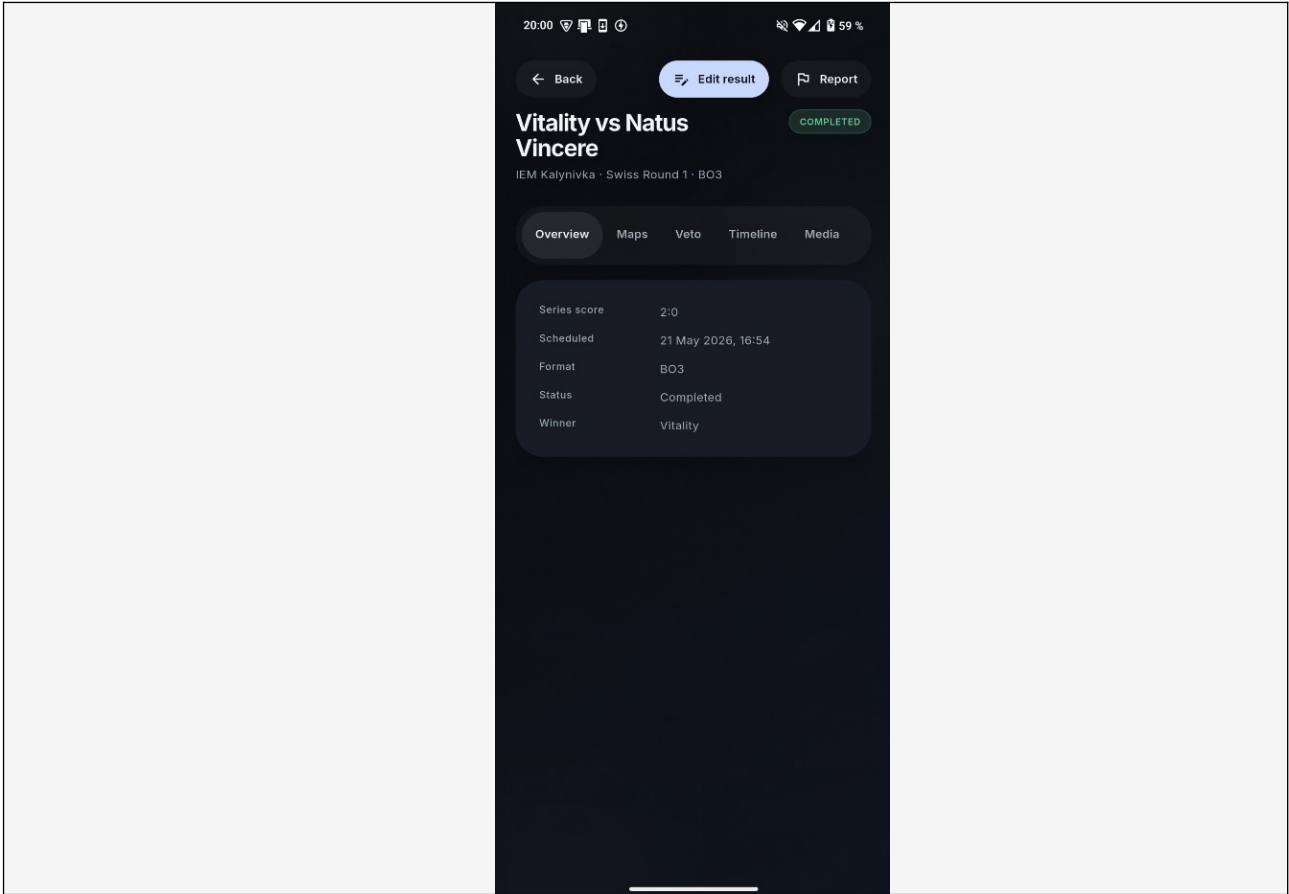
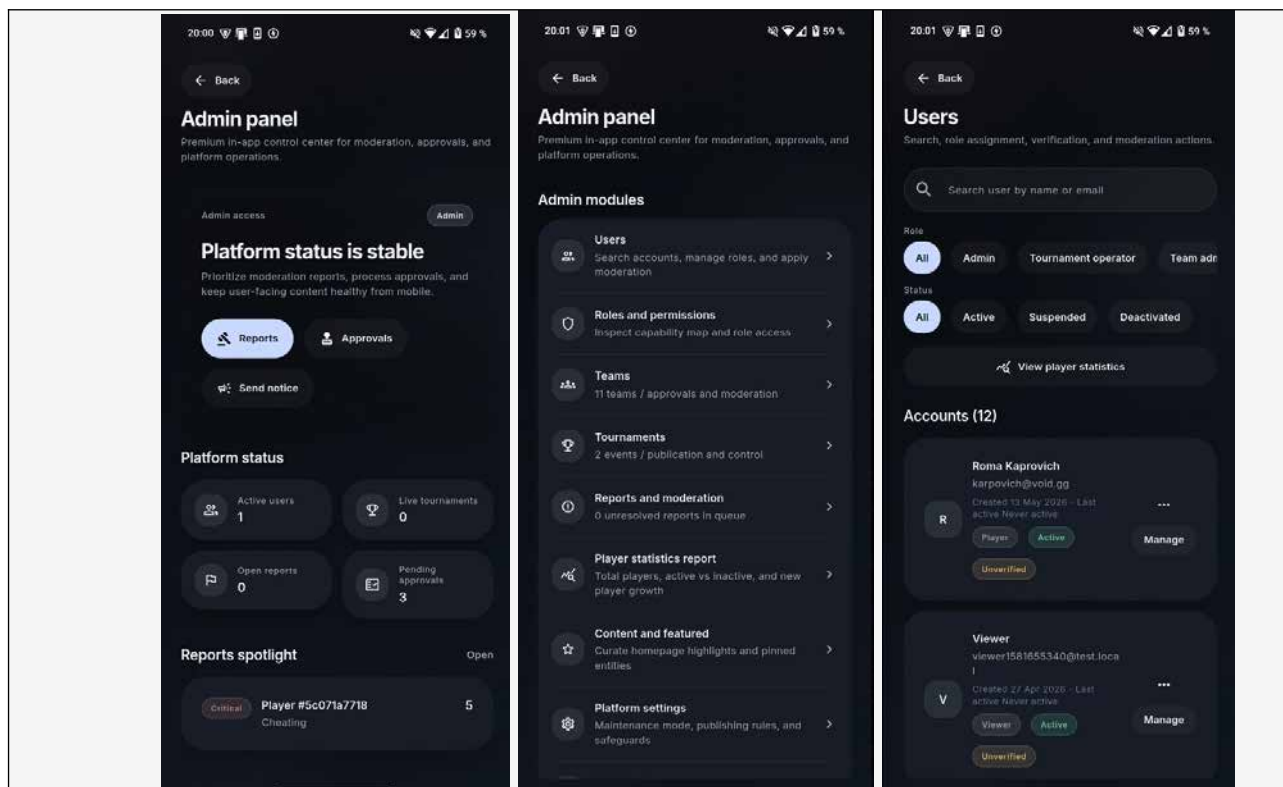


Рис. 19 - Экран MatchDetailScreen на вкладках maps/veto/timeline/media



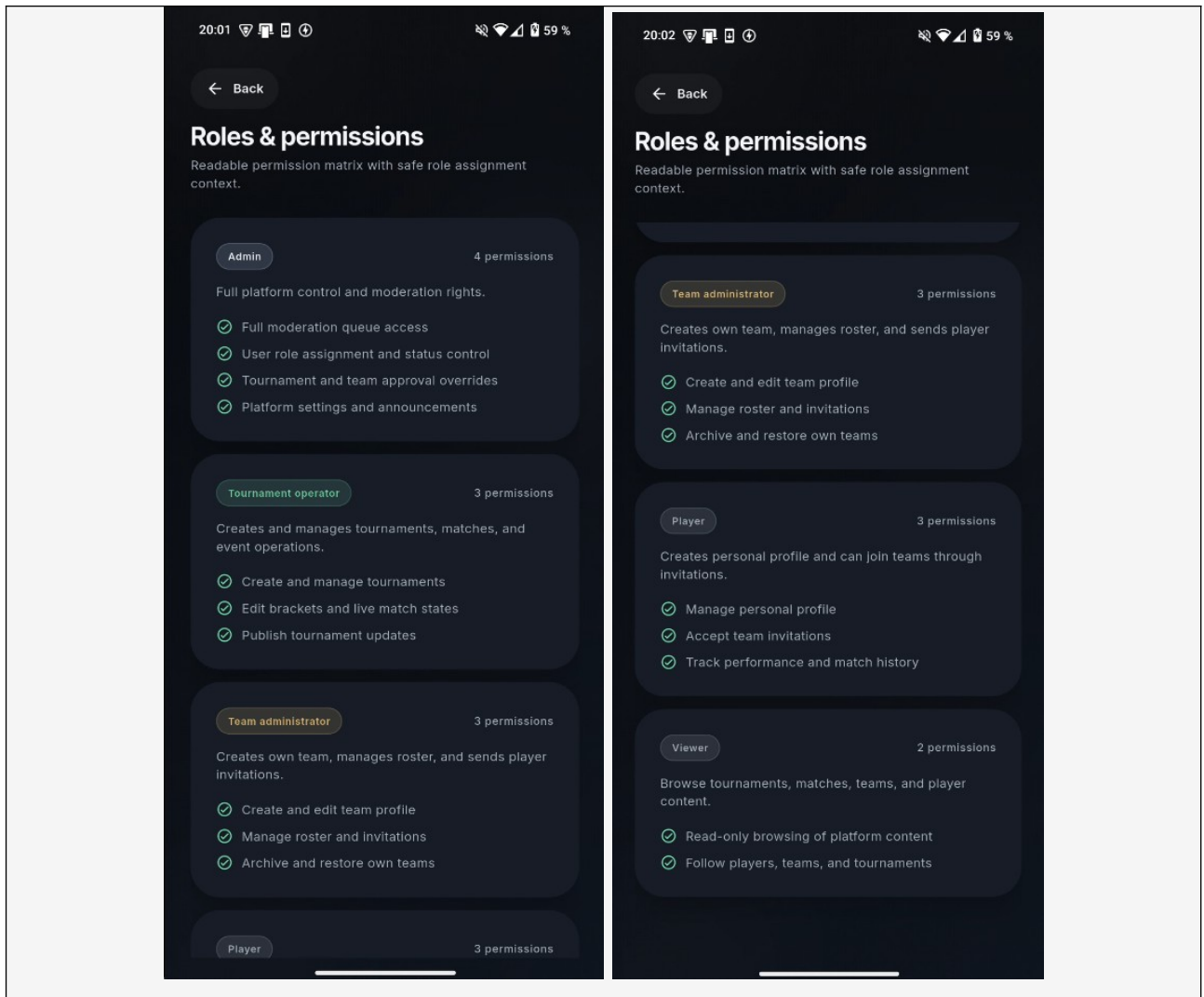


Рис. 20 - Екран AdminPanelScreen зі списком користувачів або ролей

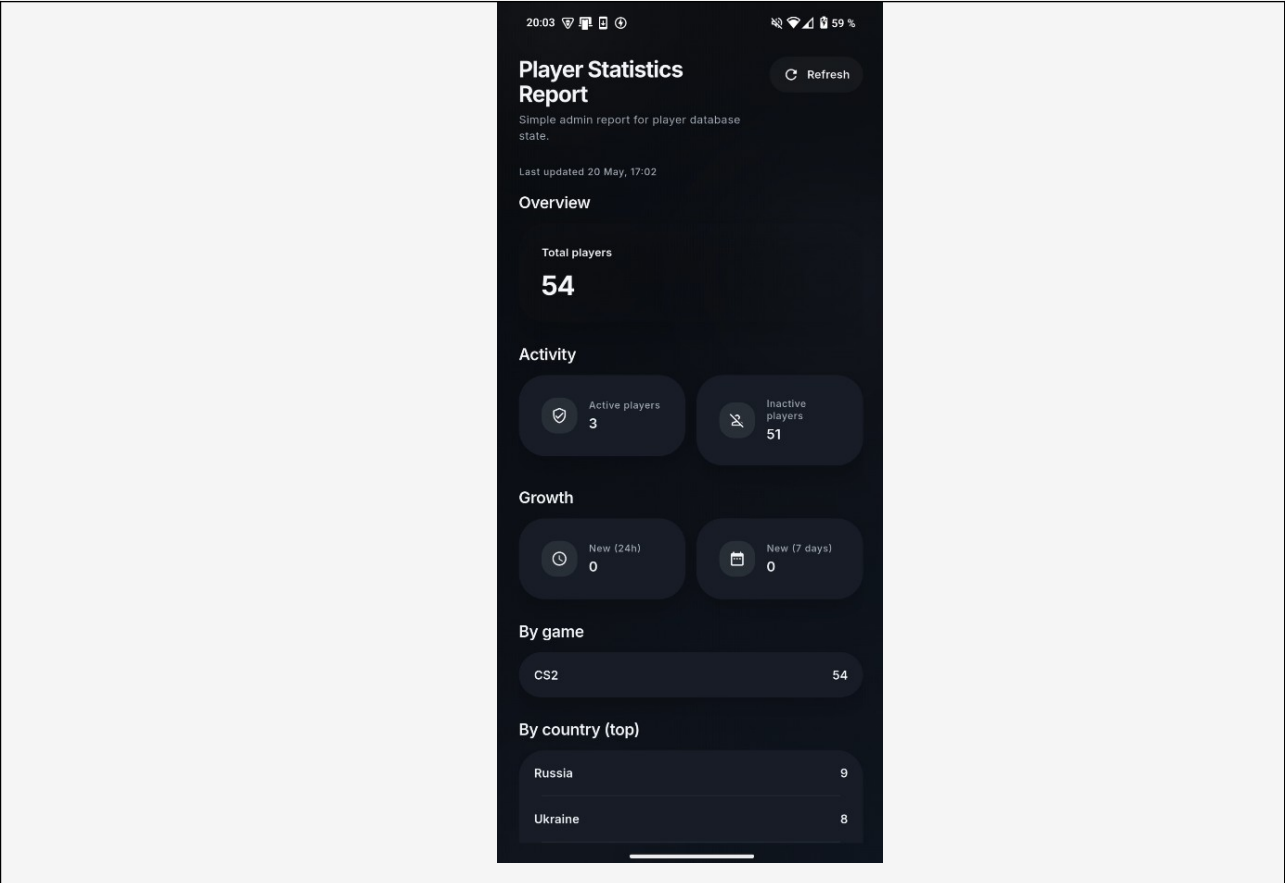


Рис. 21 - Екран AdminPlayerStatisticsReportScreen з total/active/new players і розподілами

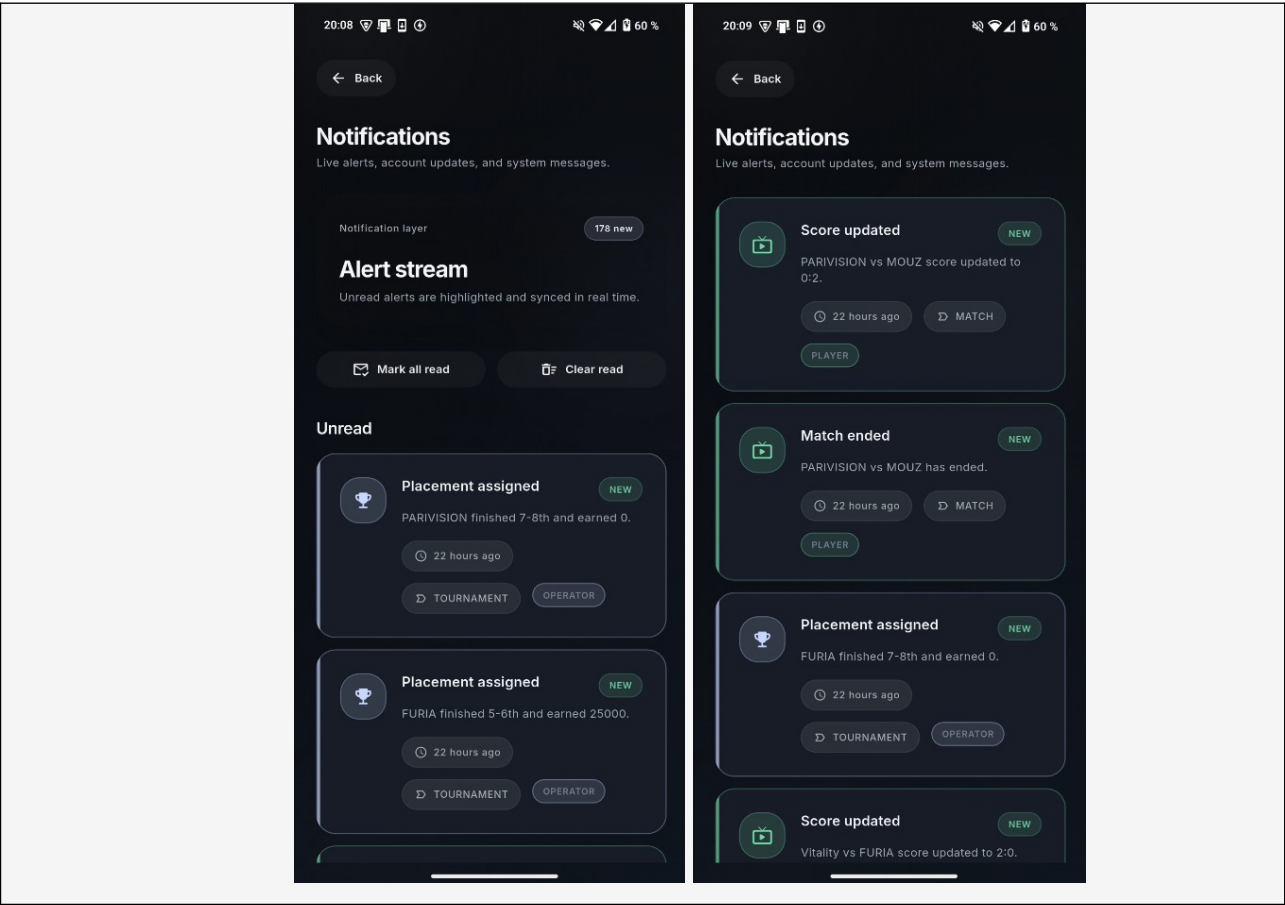


Рис. 22 - Екран NotificationsScreen зі стрічкою персональних повідомлень

3.5 Керування станом, локалізація та якість UI

Стан мобільного застосунку організовано через Riverpod-провайдери. Це дозволяє екранам підписуватися на дані репозиторію, отримувати оновлення і не дублювати логіку завантаження в кожному віджеті. Контролери в `app_controllers.dart` поєднують репозиторій, session state, dashboard state, списки гравців, команд, турнірів і повідомлень.

GoRouter описує маршрути декларативно. ShellRoute або StatefulShellRoute забезпечує головну навігацію, а окремі GoRoute відкривають детальні екрани, wizard-форми, admin-підекрани та notifications. Такий підхід полегшує deep-link логіку і дає змогу використовувати actionRoute з повідомлень.

Локалізація зберігається в `assets/l10n/app_uk.arb` і `app_en.arb`. У доменній області кіберспорту важливо мати український інтерфейс для навчального та захисного контексту, але англійські технічні терміни залишаються зрозумілими для назв дисциплін, форматів, ролей і API.

Якість UI підтримується повторно використовуваними компонентами `premium_components.dart`, `glass_widgets.dart`, `adaptive_shell.dart` і `app_theme.dart`. Це зменшує дублювання стилів і допомагає зберегти єдину візуальну систему на dashboard, списках, профілях, деталях турніру, admin-панелі та wizard-формах.

3.6 Алгоритмізація програмних модулів

3.6.1 Генерація турнірної сітки

BracketGeneratorService підтримує Single Elimination, Round Robin і Double Elimination. Для single-elimination команди спочатку впорядковуються методом seeding: random, ranked або manual. Далі обчислюється розмір сітки $B = 2^{\lceil \log_2 n \rceil}$, де n - кількість команд. Якщо команд менше, ніж слотів, додаються BYE-слоти.

Порядок seed будується рекурсивно, щоб сильні команди не зустрічалися на ранніх етапах за ranked/manual seeding. Для кожного раунду створюється набір BracketMatchNode із slotA, slotB, scoreA, scoreB, winnerTeamId, посиланням на наступний матч і вхідними sourceMatchAId/sourceMatchBId. Після кожної зміни результату виконується reconcileMatch і rebuild progression.

Для double-elimination у проєкті реалізовано спеціальний граф для чотирьох команд: upper bracket semifinals, upper final, lower bracket, lower final і grand final. Для більшої кількості команд використовується fallback на winner bracket, що залишає простір для подальшого розширення повної lower-bracket генерації.

Блок-схема алгоритму генерації single-elimination bracket із BYE-слотами

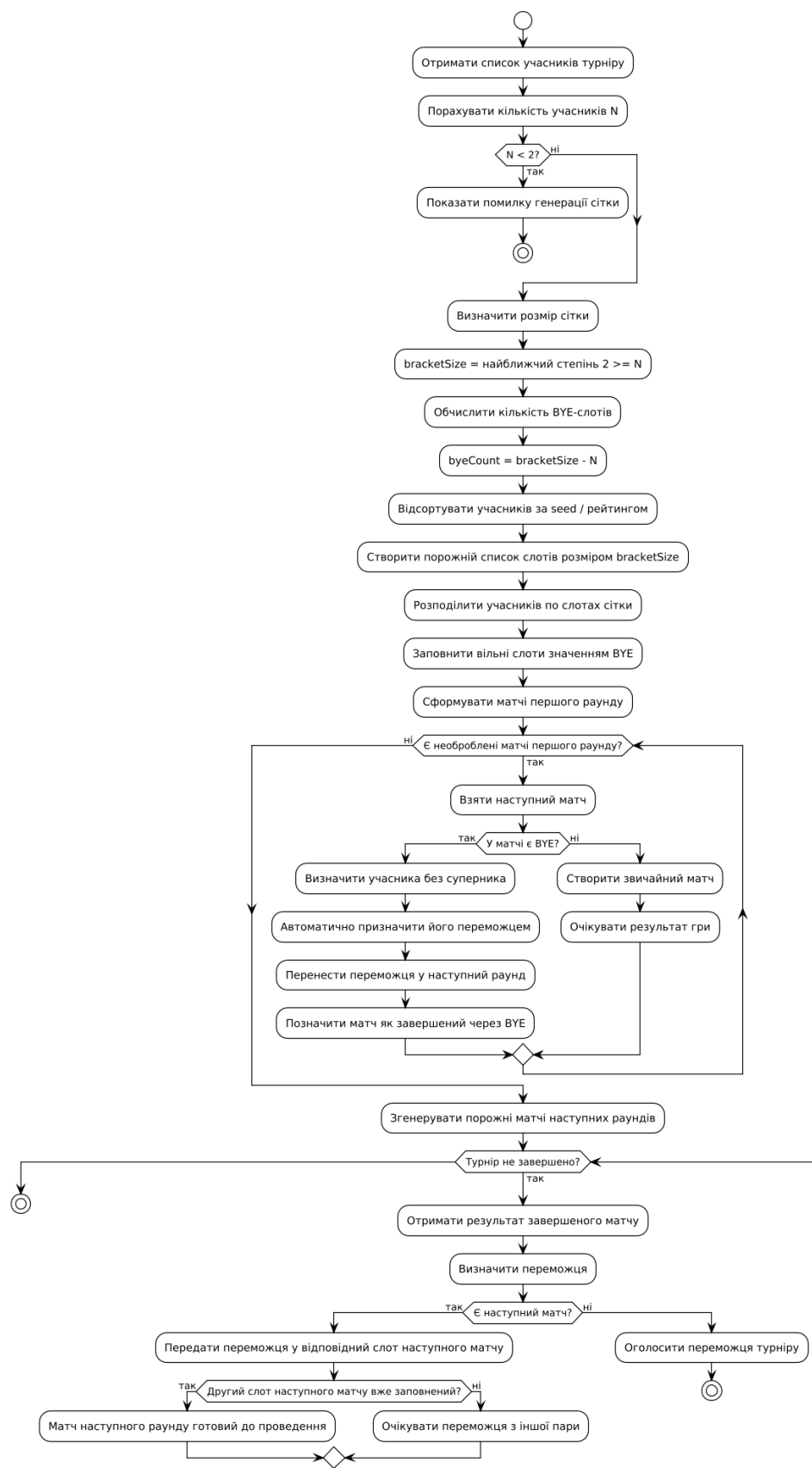


Рис. 23 - Блок-схема алгоритму генерації single-elimination bracket із BYE-слотами та просування переможця

3.6.2 Оновлення результату матчу

Оновлення результату матчу виконується через PATCH /matches/:id/score або PATCH /tournaments/:id/bracket/matches/:matchId. Спочатку сервер перевіряє, чи має користувач право керувати турніром. Потім зберігається undo snapshot, нормалізуються score, status, winnerTeamId, resultType і пов'язані поля. Після збереження матчу надсилається realtime-подія та, за потреби, повідомлення користувачам.

Якщо матч прив'язаний до bracket, зміна результату синхронізується з турнірною сіткою: переможець переходить у наступний слот, для double-elimination програвий може перейти до lower bracket. Це запобігає ситуації, коли score показує одного переможця, а bracket - іншого.

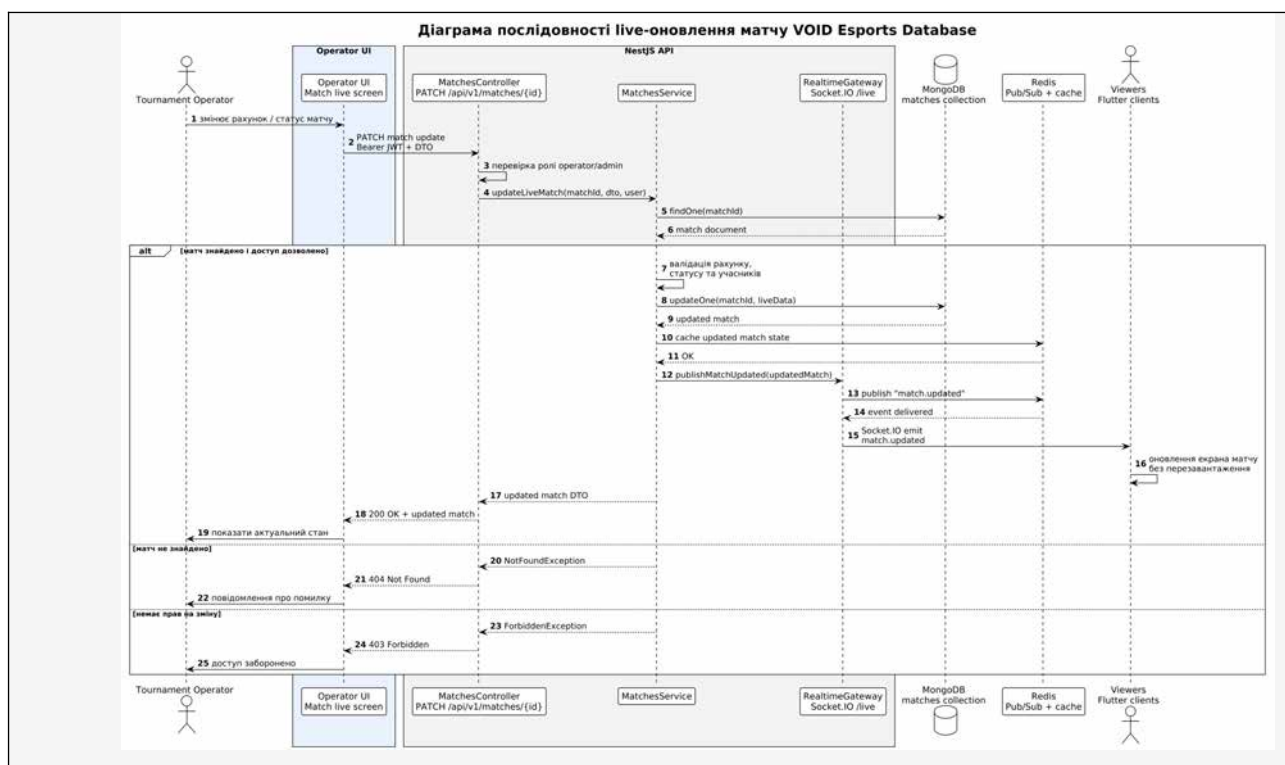


Рис. 24 - Діаграма послідовності live-оновлення матчу: operator UI, MatchesController, MatchesService, MongoDB, RealtimeGateway, Redis, viewers

3.6.3 Локальний кеш і fallback-дані

Мобільний репозиторій перед виконанням критичних операцій перевіряє доступність backend через health-запит. Якщо backend недоступний, частина сценаріїв повертає локальні sample data або кешовані дані. Це не замінює сервер,

але дозволяє демонструвати UI і базову навігацію без повністю піднятого середовища.

Локально зберігаються індекс акаунтів, активний акаунт, managed teams, managed tournaments, кеш dashboard, кеш повідомлень, черга операцій повідомлень, notification preferences і favorite players. Завдяки цьому застосунок краще переживає перезапуск і тимчасову відсутність мережі.

3.6.4 Повідомлення та push hooks

NotificationsService створює повідомлення з category, priority, actionRoute і metadata. Модуль підтримує події початку та завершення матчу, старт live stream, перенесення розкладу, технічні проблеми, улюблених гравців і системні maintenance notices. Користувач може позначати повідомлення прочитаними, очищати прочитані та змінювати preferences.

AnnouncementsModule і AnnouncementPushService містять hooks для Firebase Messaging. У локальному режимі production push-повідомлення потребують реальних Firebase service account файлів, але архітектура вже передбачає реєстрацію push-token і відправлення оголошень визначеній аудиторії.

3.7 Реалізація графіків, діаграм і растрової графіки

Вимога використання графіків і растрової графіки реалізована кількома способами. У мобільному застосунку підключено fl_chart, а на екрані профілю гравця використовується LineChart для показу тренду рейтингу. Графік містить точки, згладжену лінію, підписи та заповнену область під лінією.

Растрова графіка представлена локальними assets: void_logo.png, adventis_logo_white.png, xmotion_profile.jpg, а також мережевими або data URI зображеннями через avatarUrl, logoUrl і bannerUrl. Компоненти AdaptiveImage та CachedNetworkImage відображають локальні assets, мережеві зображення або fallback-іконки.

Турнірна сітка є структурною діаграмою в UI: Flutter відображає раунди, матчі та зв'язки між ними. Backend генерує bracket як набір вузлів, а клієнт подає його у вкладці деталей турніру. Для дипломної записки цей фрагмент потрібно підтвердити скріншотом bracket-вкладки.

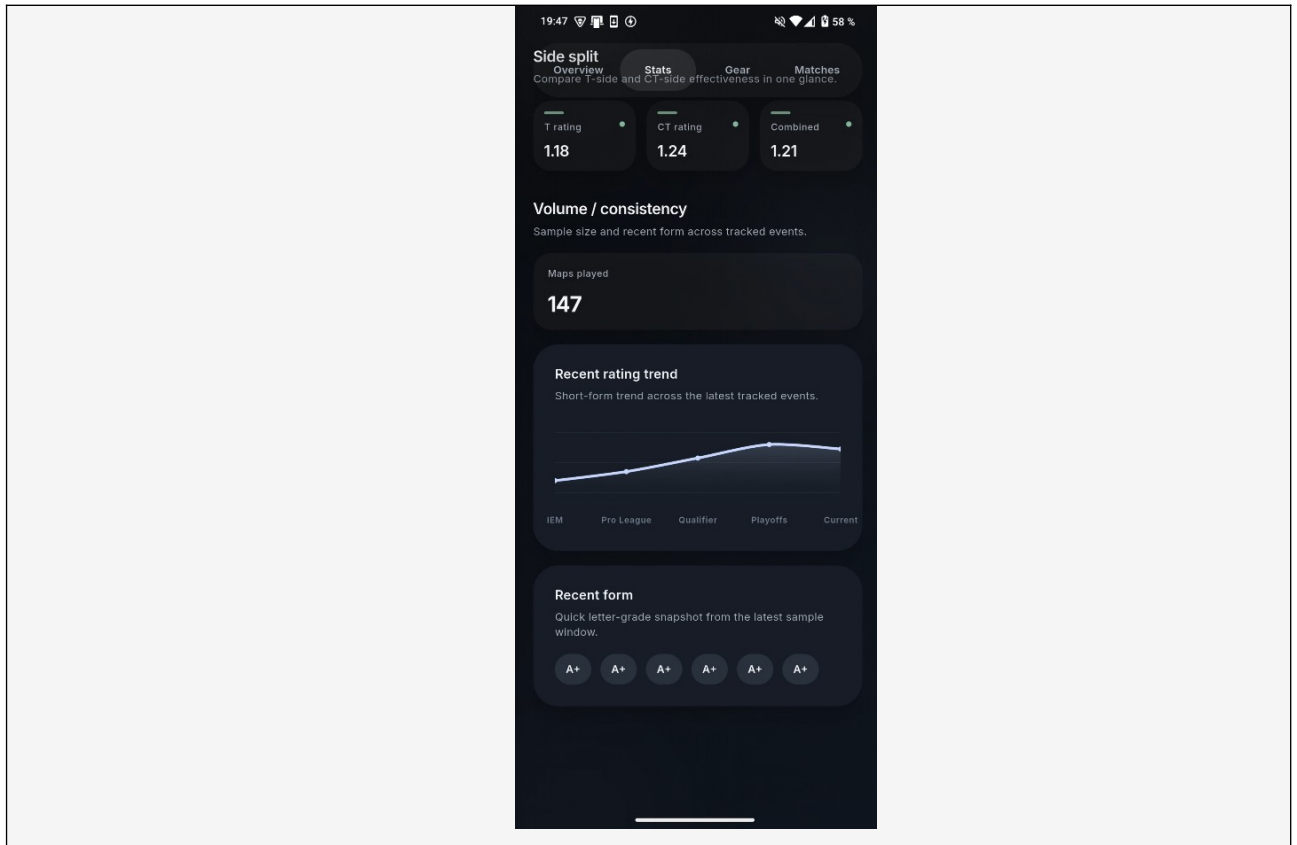
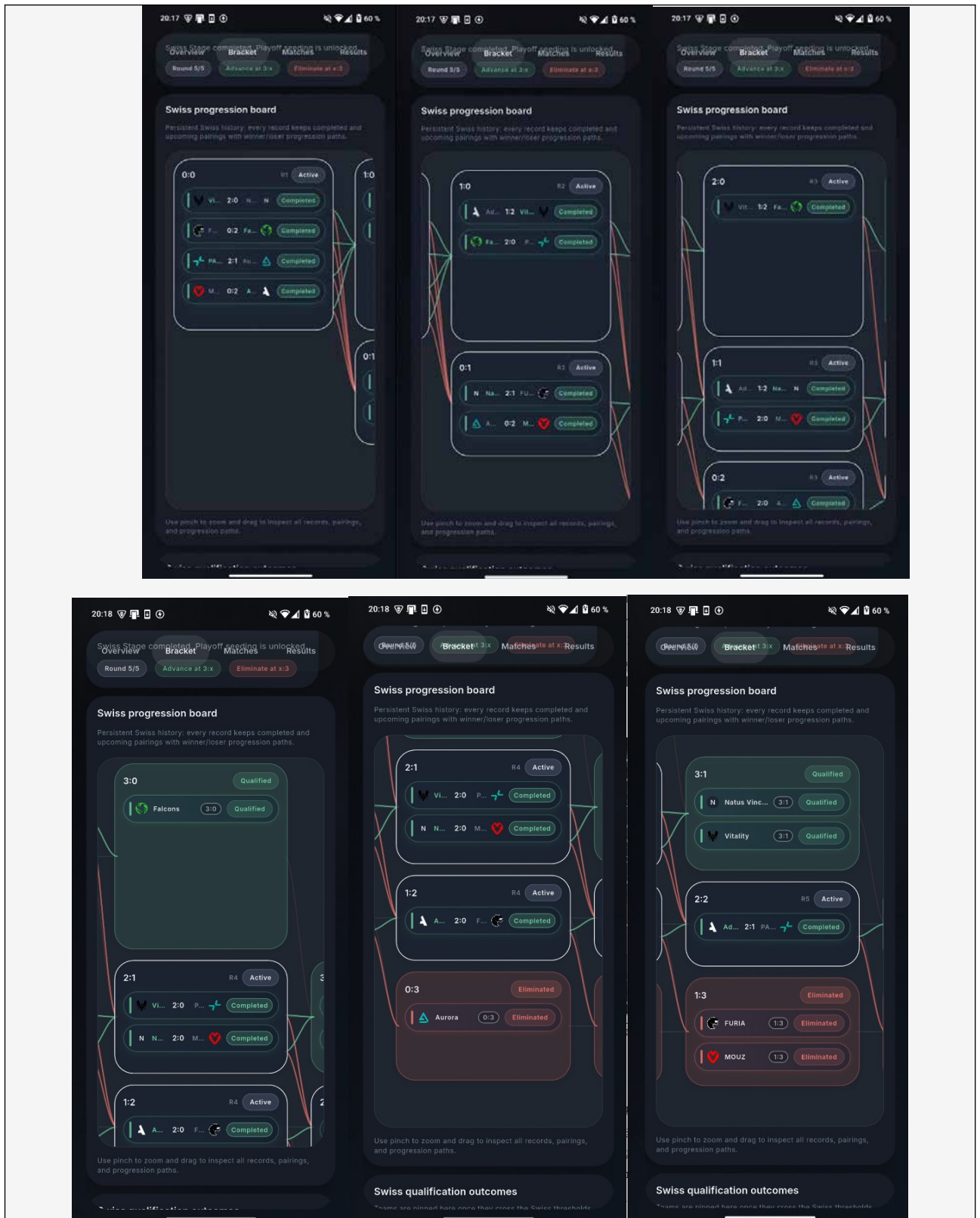


Рис. 25 - Фрагмент PlayerProfileScreen з лінійним графіком рейтингу, побудованим через `fl_chart LineChart`



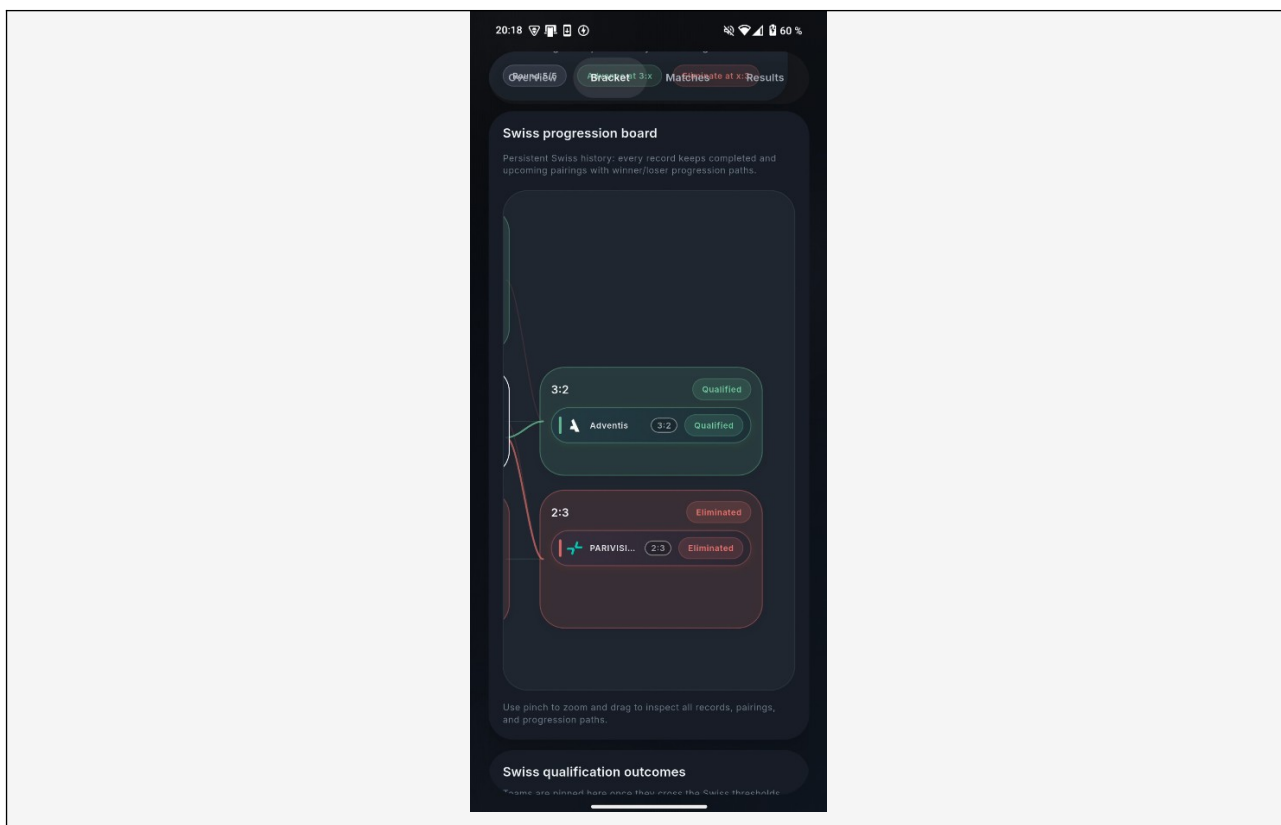


Рис. 26 - Фрагмент TournamentDetailScreen з bracket-візуалізацією та зв'язками між раундами

3.8 Тестування програмної системи

У репозиторії наявні базові smoke-тести. Для backend реалізовано тест HealthService, який перевіряє, що сервіс повертає status = ok, service = void-esports-api і timestamp. Для Flutter застосунку наявний placeholder smoke test, який підтверджує працездатність тестового середовища.

Окрім автоматизованих тестів, для такого проєкту важливе сценарне тестування. Необхідно перевірити реєстрацію та вхід, створення профілю гравця, створення команди, запрошення гравця, створення турніру, реєстрацію команди, генерацію сітки, оновлення матчу, появу повідомлення, роботу адмін-панелі та fallback при вимкненому backend.

Перевірка API може виконуватися через Swagger /docs або HTTP-клієнт. Для мобільного застосунку перевіряються Android emulator URL <http://10.0.2.2:3000/api/v1> і LAN URL для фізичного пристрою. Окремо важливо перевірити, що phone і backend знаходяться в одній мережі, а Windows firewall не блокує port 3000.

Таблиця 19 - Сценарії перевірки функціональності

Сценарій	Очікуваний результат
Health endpoint	GET /api/v1/health повертає ok, service і timestamp.
Auth	Реєстрація, вхід, зміна пароля, сесії, завершення інших сесій.
Players	Створення власного профілю, унікальність nickname, аватар, пошук.
Teams	Чернетка, autosave, publish, roster quick edit, invitation accept/decline.
Tournaments	Wizard, template, registration, bracket generation, lifecycle, check-in.
Matches	Score update, veto, maps, timeline events, undo, realtime broadcast.
Admin	Users, roles, reports, featured, announcements, analytics, platform settings.
Mobile fallback	Запуск без backend, sample data, кеш, повідомлення про недоступність.

Таблиця 20 - Edge cases, які необхідно перевірити

Ситуація	Очікувана поведінка
Backend недоступний	Flutter показує повідомлення про недоступність або sample/cache дані.
Невірний JWT	JwtAuthGuard блокує запит, клієнт просить повторний вхід.
Недостатня роль	RolesGuard повертає помилку доступу, UI не має показувати заборонену дію.
Дублікат nickname	PlayersService/індекс nickname відхиляє створення профілю.
Дублікат slug команди/турніру	Mongoose unique index або service validation блокує дублювання.
Команда не учасник турніру	MatchesService не дозволяє створити матч із чужими командами.
Неповна сітка	BracketGeneratorService додає BYE і автоматично просуває команду.
Технічна пауза	Match отримує isPaused, pauseReason і pauseEvents.
Forfeit	Матч завершується зі спеціальним resultType і winnerTeamId.
Redis вимкнений локально	Система продовжує працювати без кешу, якщо REDIS_ENABLED=false.

3.9 Основні маршрути API

Під час аналізу реалізації було окремо виділено маршрути API, які найчастіше використовуються мобільним клієнтом. Повний перелік можна переглянути у Swagger, але для пояснювальної записки доцільно показати саме функціональні групи маршрутів. Це дає змогу побачити, що backend покриває не лише довідники, а й адміністративні, турнірні, realtime-орієнтовані та модераційні сценарії.

Усі маршрути мають глобальний префікс `/api/v1`. Захищені endpoint викликаються з JWT-токеном, а публічні маршрути використовуються для перегляду відкритих даних, реєстрації, входу та health-перевірки.

Таблиця 21 - Основні маршрути API

Метод	Маршрут	Призначення
GET	/health	Перевірка доступності backend.
POST	/auth/register	Реєстрація користувача.
POST	/auth/login	Вхід і видача JWT/refresh token.
GET	/auth/me	Отримання активного профілю користувача.
GET	/users	Адміністративний список користувачів.
PATCH	/users/:id/roles	Зміна ролей користувача.
GET	/players	Список гравців із фільтрами.
POST	/players	Створення профілю гравця.
GET	/players/:id	Деталі профілю гравця.
POST	/players/validate-nickname	Перевірка унікальності nickname.
GET	/teams	Список команд.
POST	/teams	Створення команди або чернетки.
PATCH	/teams/:id/roster	Редагування ростеру.
POST	/teams/:id/invitations	Запрошення гравця.
GET	/tournaments	Список турнірів.
POST	/tournaments	Створення турніру.
POST	/tournaments/:id/register-team	Реєстрація команди на турнір.
POST	/tournaments/:id/generate-bracket	Генерація сітки.
PATCH	/tournaments/:id/lifecycle	Перехід стану турніру.

Метод	Маршрут	Призначення
GET	/matches	Список матчів.
GET	/matches/:id	Деталі матчу.
PATCH	/matches/:id/score	Оновлення рахунку.
POST	/matches/:id/veto	Додавання кроку veto.
GET	/notifications	Стрічка повідомлень.
PATCH	/notifications/:id/read	Позначення прочитаним.
GET	/analytics/dashboard	Дані головної аналітики.
GET	/reports	Модераційні звернення.
POST	/announcements	Створення оголошення.
PATCH	/platform-settings	Глобальні налаштування платформи.

3.10 Маршрутна карта мобільного застосунку

Маршрутна карта Flutter-застосунку показує, як користувач переходить між екранами. Для цього використано GoRouter. Частина маршрутів є публічною, частина доступна після входу, а адміністративні екрани відкриваються лише для користувачів із відповідною роллю.

Наявність `actionRoute` у повідомленнях робить маршрутну карту не лише елементом навігації, а й частиною інтеграції backend та UI. Сервер створює повідомлення з маршрутом, а клієнт відкриває потрібний екран після натискання.

Таблиця 22 - Маршрути Flutter-застосунку

Маршрут	Екран	Призначення
/splash	SplashScreen	Ініціалізація застосунку.
/login	LoginScreen	Вхід і реєстрація.
/home	DashboardScreen	Головна сторінка.
/players	PlayersScreen	Список гравців.
/players/:id	PlayerProfileScreen	Профіль гравця.
/profile/player	MyPlayerProfileScreen	Власний профіль гравця.

Маршрут	Екран	Призначення
/teams	TeamsScreen	Список команд.
/teams/:id	TeamDetailScreen	Деталі команди.
/team-admin	TeamAdminScreen	Керування командою.
/team-admin/wizard	TeamWizardScreen	Створення або редагування команди.
/tournaments	TournamentsScreen	Список турнірів.
/tournaments/:id	TournamentDetailScreen	Деталі турніру.
/operator/tournaments/new	TournamentWizardScreen	Створення турніру.
/matches	MatchesScreen	Список матчів.
/matches/:id	MatchDetailScreen	Деталі матчу.
/notifications	NotificationsScreen	Повідомлення.
/admin	AdminPanelScreen	Панель адміністратора.
/admin/users	AdminUsersScreen	Користувачі та ролі.
/admin/reports	AdminReportsScreen	Модераційні звернення.
/admin/featured	AdminFeaturedScreen	Featured content.
/admin/platform-settings	AdminPlatformSettingsScreen	Налаштування платформи.
/admin/announcements	AdminAnnouncementsScreen	Оголошення.
/admin/announcements/ new	AdminAnnouncementEditorScreen	Редактор оголошення.
/admin/analytics	AdminAnalyticsScreen	Адміністративна аналітика.
/admin/audit-log	AdminAuditLogScreen	Журнал аудиту.

3.11 Скріншоти, які потрібно вставити під час остаточного оформлення

Оскільки в цій версії записки не вставляються реальні зображення з екрана, у тексті залишено позначення місць, де потрібно додати скріншоти або схеми. Такий підхід зручний перед остаточним друком: спочатку формується структура й текст, а потім у вказані місця додаються фактичні знімки з працюючого застосунку.

Нижче наведено короткий перелік найважливіших скріншотів. Вони мають підтвердити, що описаний функціонал справді присутній у програмі, а не лише згаданий у тексті.

Таблиця 23 - Перелік рекомендованих скріншотів

№	Екран або схема	Що має бути видно
1	DashboardScreen	Головний екран із quick actions, featured tournament і live highlights.
2	LoginScreen	Форма входу або реєстрації з вибором ролі.
3	PlayersScreen	Список гравців із пошуком і фільтрами.
4	PlayerProfileScreen	Профіль гравця з графіком рейтингу.
5	MyPlayerProfileScreen	Редагування власного профілю та аватара.
6	TeamsScreen	Список команд із логотипами і країнами.
7	TeamAdminScreen	Керування ростером і запрошеннями.
8	TournamentsScreen	Список турнірів зі статусними chips.
9	TournamentWizardScreen	Створення турніру з template presets.
10	TournamentDetailScreen	Вкладка bracket із турнірною сіткою.
11	MatchDetailScreen	Карти, veto, timeline і медіа матчу.
12	NotificationsScreen	Стрічка персональних повідомлень.
13	AdminPanelScreen	Адміністративна панель.
14	Swagger /docs	Документація API з групами endpoint.
15	Docker/terminal	Локальний запуск або health endpoint.

3.12 Деталізація реалізації серверних модулів

Оскільки серверна частина є найбільш насиченою логікою частиною проєкту, її доцільно описати не тільки списком модулів, а й через їхню практичну роль у системі. У NestJS кожен модуль має приблизно однакову внутрішню структуру: controller приймає HTTP-запит, DTO описує формат вхідних даних, service виконує бізнес-логіку, schema визначає зберігання у MongoDB, а guards/interceptors обробляють доступ і загальні правила.

Такий стиль реалізації зробив код зрозумілішим під час розробки. Коли з'являлася нова функція, наприклад модерація скарг або оголошення, її можна було додати окремим модулем, не переписуючи вже наявну логіку матчів чи турнірів. Для дипломного проєкту це важливо, бо функціональність поступово розширювалася від простого довідника до більш повної інформаційної системи.

AuthModule

AuthModule відповідає за перший контакт користувача із системою. У ньому реалізовано реєстрацію, вхід, social login у демонстраційному форматі, зміну пароля та отримання поточного профілю. Після успішного входу користувач отримує access token, refresh token і sessionId, які мобільний застосунок зберігає локально.

Важливим рішенням було не зберігати пароль у відкритому вигляді. AuthService хешує пароль через bcrypt, а JwtStrategy перевіряє токен на захищених маршрутах. Це базовий, але необхідний рівень безпеки для будь-якої системи з ролями та персональними профілями.

UsersModule

UsersModule використовується переважно адміністративною частиною. Він дає змогу переглядати користувачів, змінювати ролі, статуси, мову, ознаку верифікації профілю гравця та інші службові параметри. Саме через цей модуль реалізується зв'язок між обліковим записом і правами доступу.

Під час розробки було важливо відокремити користувача як login-ідентичність від гравця як кіберспортивної сутності. Один користувач може мати профіль гравця, але не кожен користувач зобов'язаний бути гравцем. Це спрощує адміністрування глядачів, операторів і менеджерів команд.

PlayersModule

PlayersModule формує довідникову частину системи. Він підтримує створення профілю, редагування, пошук, перегляд деталей, завантаження аватара, перевірку nickname і адміністративне підтвердження профілю. Для

користувача це один із найпомітніших модулів, бо профілі гравців є основою кіберспортивної бази.

У профілі зберігаються не лише ім'я чи країна, а й дисципліна, роль у грі, команда, статистика, соціальні посилання, історія матчів і видимість. Завдяки цьому екран профілю можна використовувати як коротке портфоліо гравця.

TeamsModule

TeamsModule описує організаційну сторону кіберспорту. Команда має назву, тег, логотип, регіон, менеджера, учасників, rosterPlayerIds, запрошення, статус і статистику. У системі реалізовано чернетки, публікацію, архівування, відновлення, дублювання і швидке редагування ростеру.

Окремо реалізована логіка invitations. Це потрібно, щоб гравець не додавався до команди без підтвердження. Team Admin створює запрошення, а Player приймає або відхиляє його. Такий процес ближчий до реальної поведінки команд, ніж просте ручне додавання запису адміністратором.

TournamentsModule

TournamentsModule є центральним для всієї системи. Він поєднує створення турніру, шаблони, публікацію, реєстрацію команд, check-in, seeding, roster lock, disputes, auto-advance, bracket generation, swiss rounds і завершення результатів. Через велику кількість станів цей модуль потребує найбільшої уваги до бізнес-логіки.

Під час реалізації важливо було не змішувати турнір як опис події і матчі як конкретні ігрові зустрічі. Турнір зберігає загальні правила, формат і список учасників, а матчі зберігаються окремо й синхронізуються з bracket. Це дає змогу відкривати список матчів незалежно від екрана турніру.

MatchesModule

MatchesModule відповідає за live-частину системи. Він підтримує редагування рахунку, статусу, карт, veto, подій timeline, медіа, технічних пауз,

forfeit і undo snapshot. Саме тут найпомітніше поєднання звичайного REST-запиту і realtime-поведінки.

Коли оператор оновлює матч, сервер не просто записує score. Він визначає переможця, узгоджує seriesScore, може оновити bracket, створити подію аудиту та відправити повідомлення. Такий підхід зменшує ризик, що різні частини системи показуватимуть різний результат.

NotificationsModule

NotificationsModule створює персональні та системні повідомлення. Повідомлення мають priority, category, actionRoute, entityId, entityType і metadata. Це дозволяє не тільки показувати текст, а й вести користувача до потрібного екрану.

У мобільному клієнті повідомлення пов'язані з локальним кешем і чергою операцій. Якщо користувач позначив повідомлення прочитаним, але мережа тимчасово недоступна, застосунок може обробити це більш м'яко, не зупиняючи весь інтерфейс.

ReportsModule

ReportsModule потрібний для модерації. Користувач може створити звернення щодо гравця, команди, матчу або іншої сутності. Адміністратор переглядає звіт, змінює статус, додає примітку та закриває його.

Наявність цього модуля робить систему більш реалістичною. У відкритій платформі завжди можуть виникати некоректні профілі, помилкові дані або спірні ситуації, тому модерація є не другорядною, а практично необхідною функцією.

AnalyticsModule

AnalyticsModule формує дані для dashboard, топів гравців, топів команд і звіту player statistics. У дипломній роботі цей модуль показує, що зібрані дані можна не тільки зберігати, а й агрегувати для прийняття рішень.

Частина аналітики може кешуватися в Redis. Це корисно для dashboard, де одні й ті самі підсумкові показники можуть переглядатися часто, а змінюватися не після кожного запиту.

PlatformSettingsModule

PlatformSettingsModule зберігає глобальні параметри: режим технічного обслуговування, правила реєстрації, видимість, чутливість модерації та налаштування повідомлень за замовчуванням. Для адміністратора це спосіб керувати поведінкою платформи без зміни коду.

У системі також передбачено audit logs для змін налаштувань. Це важливо, тому що глобальні параметри можуть впливати на всіх користувачів, а тому потрібно знати, хто і коли їх змінив.

3.13 Деталізація мобільних екранів і взаємодії з користувачем

Мобільний застосунок розроблявся як основний інтерфейс системи, тому він має не лише відображати дані, а й вести користувача через сценарії. Для цього використано поєднання GoRouter, Riverpod, спільних UI-компонентів, локалізації та repository-шару. Така структура дозволила не писати HTTP-логіку прямо в екранах.

З погляду користувача застосунок поділяється на кілька зон: загальна навігація, довідники гравців і команд, турнірний блок, матчевий блок, профіль, повідомлення та адміністративні інструменти. Нижче описано, яку роль мають ключові екрани у цьому сценарії.

Таблиця 24 - Роль основних мобільних екранів

Екран	Роль у застосунку
DashboardScreen	Перший робочий екран після входу. Показує швидкі дії, live highlights, featured content і загальну картину системи.
LoginScreen	Поєднує вхід і реєстрацію. Для навчального проєкту також дає змогу швидко перевірити різні ролі.
PlayersScreen	Відображає список гравців, фільтрацію, пошук і перехід до профілю. Це основний довідник гравців.

Екран	Роль у застосунку
PlayerProfileScreen	Показує дані конкретного гравця, команду, статистику, історію та графік рейтингу.
MyPlayerProfileScreen	Дає користувачу змогу створити або редагувати власний профіль гравця.
TeamsScreen	Показує список команд, країни, логотипи, короткі характеристики та перехід до деталей.
TeamAdminScreen	Надає менеджеру команди інструменти для ростеру, invitations і швидких змін.
TournamentsScreen	Дає огляд турнірів за статусами: усі, live, відкриті для реєстрації, завершені.
TournamentWizardScreen	Проводить оператора через створення турніру, template presets і параметри формату.
TournamentDetailScreen	Об'єднує опис турніру, учасників, матчі, bracket, правила та результати.
MatchesScreen	Відображає список матчів і допомагає швидко знайти live або заплановані зустрічі.
MatchDetailScreen	Найбільш динамічний екран: score, maps, veto, timeline, media, status і live-події.
NotificationsScreen	Показує персональні повідомлення, пріоритети, прочитані/непрочитані стани та переходи.
AdminPanelScreen	Збирає адміністративні функції: користувачі, полі, reports, featured, settings, analytics і audit.

3.14 Обробка помилок і нестандартних ситуацій

Під час розроблення мобільної клієнт-серверної системи особливу увагу потрібно приділяти не тільки успішним сценаріям, а й помилкам. У реальному використанні backend може бути недоступним, токен може застаріти, користувач може втратити роль, а мобільний пристрій може мати нестабільне з'єднання. Якщо такі ситуації не обробляти, застосунок виглядатиме ненадійним навіть за наявності багатого функціоналу.

У Flutter-клієнті більшість мережових помилок проходить через repository-шар. Це зручно, тому що окремі екрани не повинні знати всі деталі DioException, HTTP-кодів або fallback-логіки. Екран отримує або дані, або зрозумілий стан помилки, який можна показати користувачу.

На сервері помилки обробляються через стандартні механізми NestJS: exceptions, guards, validation pipe і HTTP status codes. Наприклад, неправильний JWT повинен повертати 401, недостатня роль - 403, невалідні дані DTO - 400, а відсутня сутність - 404. Такий поділ допомагає клієнту правильно реагувати.

Окремим випадком є помилки бізнес-логіки. Формально запит може бути правильним, але дія все одно не дозволена. Наприклад, не можна зареєструвати команду після завершення реєстрації, не можна оновити матч чужого турніру, не можна створити профіль із зайнятим nickname. Такі помилки мають пояснювати причину, а не повертати загальне повідомлення.

У мобільному інтерфейсі помилка не повинна завжди блокувати весь екран. Для списків доречно показувати cached/sample дані, для форм - підсвічувати конкретне поле, для адміністративних дій - показувати коротке повідомлення про відмову доступу. Це робить застосунок більш передбачуваним.

Під час демонстрації помилки також корисно показати. Наприклад, можна спробувати виконати адміністративну дію без відповідної ролі або вимкнути backend і відкрити головний екран. Це підтверджує, що система розроблялася не тільки під ідеальний сценарій.

Таблиця 25 - Обробка помилок у клієнт-серверній взаємодії

Ситуація	Причина	Реакція системи
400 Bad Request	Невалідні DTO-дані	Показати помилку форми або повідомлення про неправильні дані.
401 Unauthorized	Відсутній або застарілий JWT	Запропонувати повторний вхід.
403 Forbidden	Недостатня роль	Приховати дію в UI і показати відмову доступу.

Ситуація	Причина	Реакція системи
404 Not Found	Сутність не знайдена	Повернутися до списку або показати empty state.
409 Conflict	Дублікат nickname, slug або конфлікт стану	Пояснити, яке поле потрібно змінити.
500 Server Error	Неочікувана помилка backend	Показати загальну помилку і не втрачати локальний стан.
Network timeout	Немає відповіді від API	Використати кеш або sample data, запропонувати повторити.
Socket disconnected	Втрачено realtime-з'єднання	Продовжити REST-режим і перепідключитися пізніше.

Висновки до розділу 3

1. Описано клієнт-серверну архітектуру програмної системи з Flutter mobile app, NestJS API, MongoDB, Redis, Socket.IO, Docker і Nginx.
2. Розглянуто серверні модулі, мобільні екрани, навігацію, репозиторій даних, локальний кеш і fallback-механізми.
3. Описано алгоритми генерації турнірної сітки, оновлення результату матчу, пошуку, повідомлень і realtime-синхронізації.
4. Визначено підхід до тестування: наявні smoke-тести, сценарна перевірка API/UI, Swagger і мобільний запуск на emulator або фізичному пристрої.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для локального запуску системи потрібні Node.js 20+, npm 10+, Flutter SDK 3.35+, Android Studio або Flutter Android toolchain, Docker Desktop, MongoDB і Redis. У Docker-режимі MongoDB, Redis, API та Nginx запускаються через docker-compose.yml. У локальному режимі API може запускатися через npm run start:dev, а MongoDB - окремим контейнером або локальним сервісом.

На Android emulator мобільний застосунок звертається до backend через 10.0.2.2. На фізичному пристрої потрібно використовувати LAN IP комп'ютера, наприклад <http://192.168.0.154:3000/api/v1>, за умови, що телефон і комп'ютер знаходяться в одній Wi-Fi мережі.

Для production-впровадження необхідно замінити JWT_SECRET, налаштувати CORS_ORIGIN, забезпечити HTTPS, підготувати Firebase service account, налаштувати backup MongoDB, обмежити адміністративні endpoint і визначити політику зберігання персональних даних.

Таблиця 26 - Вимоги до середовища запуску

Категорія	Вимоги
Backend runtime	Node.js 20+, npm 10+, NestJS CLI для розробки.
Mobile runtime	Flutter SDK 3.35+, Dart SDK, Android SDK, emulator або фізичний Android-пристрій.
Database	MongoDB 7 через Docker або локальний сервіс.
Cache/realtime support	Redis 7 Alpine, опційний для локального запуску.
Containers	Docker Desktop і docker compose.
Network	Доступ до localhost:3000 або LAN IP:3000, відкритий firewall port 3000.
Production security	Сильний JWT_SECRET, HTTPS, обмежений CORS, реальний Firebase setup.

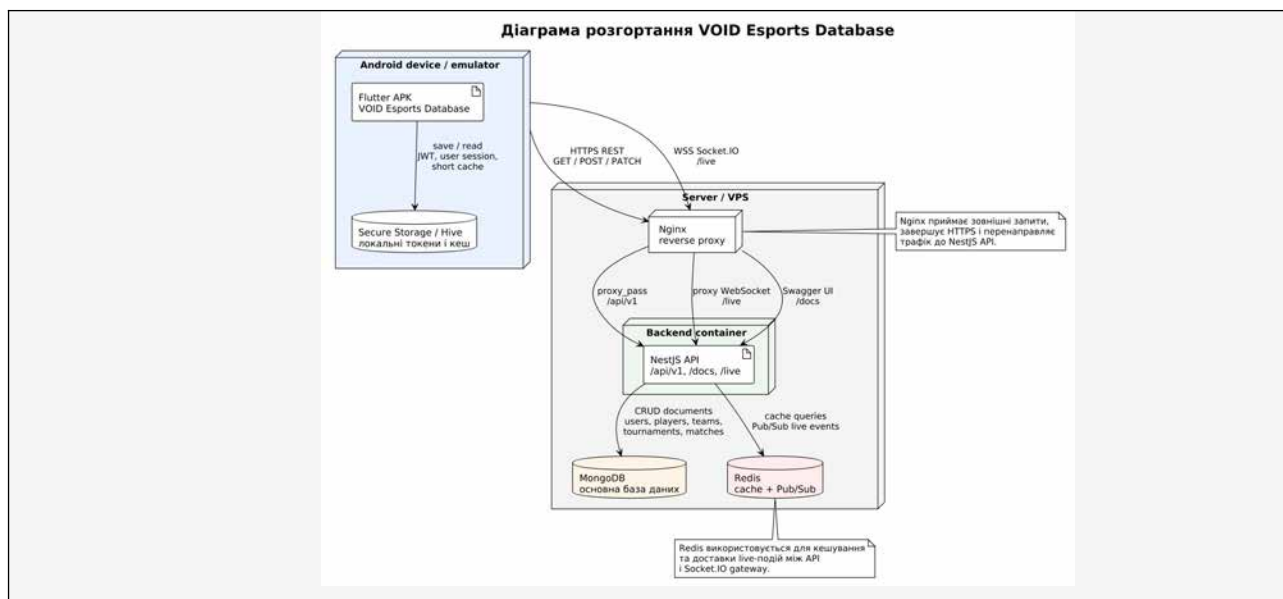


Рис. 27 - Діаграма розгортання: Android device/emulator, Nginx, NestJS API, MongoDB, Redis

4.2 Порядок локального запуску

Найшвидший шлях запуску складається з трьох етапів: підняти MongoDB, Redis і API через Docker; виконати seed CS2 HLTV top 10; запустити Flutter-застосунок на emulator або телефоні з правильним API_BASE_URL і SOCKET_URL.

Якщо потрібен hot reload backend, доцільно запускати лише MongoDB і Redis у Docker, а API - локально з apps/api. У такому разі MONGODB_URI необхідно змінити на mongodb://127.0.0.1:27017/void-esports, а Redis у локальному режимі можна вимкнути через REDIS_ENABLED=false.

Для перевірки працездатності backend слід відкрити `http://localhost:3000/api/v1/health` і `http://localhost:3000/docs`. Для телефону потрібно перевірити URL `http://<LAN-IP>:3000/api/v1/health` у браузері пристрою.

```
cd "H:\Void - Esports Database"
docker compose up --build
```

```
cd "H:\Void - Esports Database\apps\api"
npm run seed:cs2:hltv-top10
```

```
cd "H:\Void - Esports Database\apps\mobile"
flutter run --dart-define=API_BASE_URL=http://10.0.2.2:3000/api/v1 --dart-define=SOCKET_URL=http://10.0.2.2:3000/live
```

4.3 Склад інсталяційного пакета

Склад програмного продукту включає вихідні файли backend, мобільного клієнта, конфігурацію Docker, Nginx, environment-приклад, seed-скрипти і документацію запуску. Для передачі або захисту роботи важливо зберегти структуру каталогів, оскільки мобільна і серверна частини мають власні залежності та команди.

До інсталяційного пакета не доцільно включати node_modules, build, dist і .dart_tool як основні артефакти, якщо вони можуть бути відновлені командами npm install, npm run build, flutter pub get і flutter build. Натомість потрібно включити package.json, package-lock.json, pubspec.yaml, pubspec.lock і всі вихідні файли.

Таблиця 27 - Склад інсталяційного пакета

Файл або каталог	Призначення
apps/api/src	NestJS вихідний код: modules, common, integrations, config, seeds.
apps/api/package.json	Скрипти build, start, seed, lint, test і залежності backend.
apps/mobile/lib	Flutter вихідний код: screens, widgets, models, repository, services, navigation, theme.
apps/mobile/assets	Локалізація uk/en та растрові зображення.
docker-compose.yml	MongoDB, Redis, API, Nginx для локального розгортання.
infrastructure/nginx/default.conf	Reverse проху для /api, /docs, /live і /health.
.env.example	Шаблон змінних середовища без секретів production.
README.md	Інструкції запуску, seed, mobile run, troubleshooting.

4.4 Опис основних користувацьких сценаріїв

Реєстрація та вхід

Користувач відкриває LoginScreen, обирає режим входу або реєстрації, вводить email, пароль і displayName, за потреби вибирає роль. Клієнт надсилає запит до /auth/register або /auth/login. У разі успіху репозиторій зберігає токен, refresh token, sessionId і активний акаунт.

Створення профілю гравця

Гравець переходить до MyPlayerProfileScreen, заповнює nickname, discipline, країну, роль, додаткові поля та аватар. Перед збереженням може виконуватися валідація nickname через /players/validate-nickname. Після створення профіль стає доступним у списку гравців і деталях.

Керування командою

Team Admin створює чернетку команди, редагує назву, tag, дисципліну, регіон, логотип, опис і ростер. Команду можна публікувати, архівувати, відновлювати, дублювати, а гравців запрошувати через invitations. Player приймає або відхиляє інвайт у своєму профілі.

Створення турніру

Tournament Operator відкриває TournamentWizardScreen, вибирає template preset або вводить параметри вручну: формат, регіон, дати, capacity, check-in, seeding, roster lock і best-of. Після публікації турнір стає доступним для перегляду та реєстрації команд.

Проведення матчу

Оператор відкриває MatchDetailScreen, редагує score, карти, veto, live events, streamUrl, технічні паузи або forfeit. Сервер зберігає undo snapshot, оновлює матч, синхронізує bracket і надсилає realtime-подію підписаним клієнтам.

Адміністрування

Admin використовує AdminPanelScreen для перегляду користувачів, зміни ролей і статусів, модерації скарг, featured-контенту, оголошень, platform settings, analytics і audit log. Це забезпечує контроль операційної частини системи.

4.5 Обмеження та напрями подальшого розвитку

Поточна версія має функціонально широку, але навчальну реалізацію. Firebase/FCM hooks існують, однак production push-повідомлення потребують реального service account і налаштувань мобільного застосунку. Інші розділи, окрім Teams і Players, можуть використовувати sample data або часткову backend-інтеграцію залежно від конкретного сценарію запуску.

Соціальний вхід у мобільному репозиторії частково представлений демо-сесіями, тому для production необхідна повна інтеграція Google/Apple OAuth на клієнті та сервері. Double-elimination bracket має спеціальну реалізацію для чотирьох команд і fallback для більших сіток, що також є напрямом розширення.

Подальший розвиток може включати повнотекстовий пошук MongoDB або окремий search engine, production push, веб-адмінку, розширення дисциплін, live scoreboard, імпорт зовнішніх матчів, автоматичні турнірні правила, докладнішу статистику гравців, coverage-тести, CI/CD і контроль ролей на рівні кожної операції.

Таблиця 28 - Напрями розвитку системи

Напря́м	Очікуваний результат
Повнотекстовий пошук	Додати MongoDB text indexes або окремий search сервіс для великих колекцій.
Production push	Підключити Firebase service account, Android config і перевірити доставку FCM.
OAuth	Реалізувати реальний Google/Apple login замість демо-логіки клієнта.
Double elimination	Розширити генерацію lower bracket для 8, 16 і більше команд.
Тести	Додати unit, integration, e2e і widget tests для основних сценаріїв.

Напря́м	Очікуваний результат
CI/CD	Автоматизувати lint, test, build API і Flutter APK.

4.6 Експлуатаційні ризики та заходи зменшення

Під час експлуатації клієнт-серверної системи виникають ризики, пов'язані з доступністю backend, якістю мережі, коректністю ролей, конфігурацією CORS, секретами JWT, станом бази даних і людськими помилками оператора. У турнірних сценаріях особливо критичним є неправильне оновлення результату матчу, оскільки воно може вплинути на сітку і наступні матчі.

Для зменшення ризику втрати даних потрібно регулярно створювати резервні копії MongoDB, контролювати змінні середовища, не зберігати production secrets у репозиторії, використовувати HTTPS і обмежувати адміністративні endpoint. Для матчів важливо зберігати audit logs і undo snapshots, що вже частково реалізовано в MatchesService.

Для зменшення ризику неправильного доступу потрібно підтримувати єдину рольову модель між backend і Flutter, не покладатися лише на приховування кнопок у UI, а обов'язково перевіряти ролі на сервері. Для production слід також додати e2e-тести на критичні сценарії: зміна ролі, створення турніру, оновлення score, модерація скарги.

Таблиця 29 - Ризики експлуатації та заходи

Ризик	Наслідок	Заходи
Неправильний JWT_SECRET	Компрометація токенів	Замінити secret, зберігати в secure env, обертати токени.
Відсутність backup MongoDB	Втрата турнірних і матчевих даних	План backup/restore і перевірка відновлення.
Неправильний CORS	Мобільний або web-клієнт не може звернутися до API	Явно налаштувати CORS_ORIGIN для середовища.
Помилка оператора	Неправильний bracket	Undo snapshot, audit log,

Ризик	Наслідок	Заходи
матчу	progression	підтвердження критичних дій.
Недостатня валідація DTO	Некоректні дані у БД	Whitelist, forbidNonWhitelisted, class-validator, Mongoose enum.
Немає Firebase setup	Push не доставляється	Підключити service account і перевірити push-token реєстрацію.

4.7 Детальний план тестування

Тестування системи доцільно виконувати поетапно. Спочатку перевіряється запуск backend і база даних, потім основні API-запити через Swagger, після цього мобільні сценарії на емуляторі або фізичному пристрої. Окремо потрібно перевіряти помилки, тому що дипломний застосунок має показувати не лише успішні стани, а й зрозумілу поведінку в разі недоступності сервера або недостатніх прав.

Наведений план тестування можна використовувати як чеклист перед захистом. Він не замінює повний набір автоматизованих тестів, але допомагає послідовно пройти функціональність і зафіксувати, які екрани та endpoint працюють.

Таблиця 30 - Детальний план функціонального тестування

ID	Модуль	Дія	Очікуваний результат
T-01	Health	Відкрити GET /health	Повертається status ok і timestamp.
T-02	Swagger	Відкрити /docs	Документація API завантажується.
T-03	Register	Створити нового користувача	Користувач створений, токени повернено.
T-04	Login	Увійти з правильним email/password	Відкривається DashboardScreen.
T-05	Invalid login	Ввести неправильний пароль	Показується помилка входу.
T-06	Session	Перезапустити застосунок	Активна сесія відновлюється.
T-07	Player create	Створити профіль гравця	Профіль збережено та видно у

ID	Модуль	Дія	Очікуваний результат
			списку.
T-08	Nickname duplicate	Повторити nickname	Система блокує дублювання.
T-09	Player avatar	Додати аватар	Зображення відображається або fallback працює.
T-10	Players search	Ввести пошуковий запит	Список фільтрується.
T-11	Team draft	Створити чернетку команди	Команда має draft/active status.
T-12	Team roster	Додати гравця до ростеру	Ростер оновлено.
T-13	Invitation	Надіслати запрошення	Гравець бачить invitation.
T-14	Invitation accept	Прийняти запрошення	Гравець прив'язаний до команди.
T-15	Tournament wizard	Створити турнір	Турнір з'являється у списку.
T-16	Tournament publish	Опублікувати турнір	Статус змінено, турнір видимий.
T-17	Register team	Додати команду до турніру	Команда у списку учасників.
T-18	Generate bracket	Згенерувати сітку	Створені матчі/вузли bracket.
T-19	Match details	Відкрити MatchDetailScreen	Показано команди, рахунок, карти.
T-20	Score update	Оновити рахунок	Score і winnerTeamId збережено.
T-21	Bracket sync	Завершити матч у сітці	Переможець переходить далі.
T-22	Veto	Додати ban/pick	Крок з'являється у veto list.
T-23	Pause	Додати технічну паузу	Матч має paused state або pause event.
T-24	Notifications	Створити подію матчу	Користувач отримує повідомлення.
T-25	Read notification	Позначити повідомлення прочитаним	isRead змінюється.
T-26	Admin users	Відкрити список користувачів	Admin бачить таблицю користувачів.
T-27	Role guard	Викликати admin endpoint без	Доступ заборонено.

ID	Модуль	Дія	Очікуваний результат
		ролі	
T-28	Reports	Створити і закрити скаргу	Report змінює статус.
T-29	Announcements	Створити оголошення	Оголошення збережено.
T-30	Offline fallback	Вимкнути API і відкрити app	Застосунок показує fallback/cache.

4.8 Чеклист демонстрації програмного продукту

Перед захистом або демонстрацією доцільно підготувати короткий сценарій показу. Він має бути не надто довгим, але повинен охопити всі ключові частини системи: backend, Swagger, мобільний клієнт, дані, ролі, турнір, матч і повідомлення.

Чеклист нижче сформовано так, щоб під час демонстрації було видно зв'язок між текстом записки та реальною програмою. Якщо якийсь пункт не вдається показати через середовище запуску, його варто зафіксувати як обмеження поточної версії.

Таблиця 31 - Чеклист демонстрації

Крок	Дія	Що показати
1	Запустити docker compose	MongoDB, Redis, API, Nginx стартують без критичних помилок.
2	Відкрити /api/v1/health	Показати status ok.
3	Відкрити Swagger /docs	Показати групи endpoint.
4	Виконати seed	Показати тестові команди/гравців.
5	Запустити Flutter	Показати Splash/Login/Dashboard.
6	Увійти в систему	Показати збереження сесії.
7	Відкрити Players	Показати пошук і профіль.
8	Відкрити Teams	Показати команду і ростер.
9	Створити або відкрити турнір	Показати wizard або деталі турніру.
10	Показати bracket	Пояснити генерацію сітки.

Крок	Дія	Що показати
11	Відкрити матч	Показати рахунок, карти, veto, timeline.
12	Оновити score	Показати зміну стану або realtime-подію.
13	Відкрити Notifications	Показати повідомлення і actionRoute.
14	Відкрити Admin	Показати ролі, reports, settings або analytics.
15	Пояснити fallback	Показати поведінку без backend або описати кеш.

4.9 Контроль якості та критерії готовності

Критерії готовності для такого проєкту повинні враховувати не тільки факт запуску, а й цілісність сценаріїв. Якщо користувач може увійти, але не може перейти до свого профілю або створити команду, то система формально запускається, але функціонально ще не готова. Тому критерії краще формулювати за модулями.

Для backend критеріями є успішне збирання, health endpoint, Swagger, підключення до MongoDB, проходження базових тестів і коректні відповіді API. Для мобільної частини - запуск на emulator/пристрої, навігація, відображення даних, обробка помилок, відсутність критичних layout-помилки і робота з локальною сесією.

Окремим критерієм є відповідність тексту записки реальному функціоналу. Усі заявлені функції повинні бути або реалізовані, або чесно позначені як обмеження чи напрям розвитку. Це важливо для академічної доброчесності та нормального захисту роботи.

Таблиця 32 - Критерії готовності системи

Критерій	Перевірка	Примітка
Backend build	npm run build завершується без помилок.	Обов'язково перед демонстрацією.
Health	GET /health повертає ok.	Швидка перевірка доступності API.
Swagger	/docs відкривається.	Підтверджує структуру endpoint.

Критерій	Перевірка	Примітка
Database	MongoDB доступна, seed виконується.	Потрібно для реальних списків.
Auth	Register/login працюють.	Базовий вхід у систему.
Roles	Admin endpoint закриті для звичайного користувача.	Безпека доступу.
Mobile run	Flutter запускається на emulator або телефоні.	Головна умова демонстрації.
Navigation	Основні маршрути відкриваються.	Перевіряє GoRouter.
Data display	Списки гравців, команд, турнірів і матчів не порожні.	Потрібно для захисту.
Error states	Помилки API показуються зрозуміло.	Не повинно бути crash.
Screenshots	Підготовлені знімки ключових екранів.	Для остаточного оформлення.
Documentation	README, .env.example і команди запуску актуальні.	Для відтворення проєкту.

4.10 Файли та конфігурація, які потрібно зберегти

Для передачі проєкту важливо зберегти не лише вихідний код, а й конфігураційні файли, які описують запуск. У дипломному контексті це має практичне значення: інша людина повинна мати змогу відкрити репозиторій, встановити залежності та відтворити базову демонстрацію.

Секрети production-середовища не слід зберігати у відкритому вигляді. Натомість потрібно залишити .env.example із назвами змінних і поясненнями. Реальні JWT_SECRET, Firebase service account і production URI мають передаватися окремо або налаштовуватися в середовищі запуску.

Таблиця 33 - Ключові файли для відтворення проєкту

Файл або каталог	Призначення	Коментар
docker-compose.yml	Запуск MongoDB, Redis, API і	Зберегти обов'язково.

Файл або каталог	Призначення	Коментар
	Nginx.	
apps/api/package.json	Backend scripts і залежності.	Потрібний для npm install/build.
apps/api/src	Вихідний код NestJS API.	Основна серверна частина.
apps/api/src/database/seeds	Seed-наповнення.	Потрібне для демонстрації.
apps/mobile/pubspec.yaml	Flutter залежності та assets.	Потрібний для flutter pub get.
apps/mobile/lib	Вихідний код Flutter.	Основна клієнтська частина.
apps/mobile/assets	Локалізація і зображення.	Потрібне для коректного UI.
infrastructure/nginx/default.conf	Reverse proxy.	Корисно для локального стенду.
.env.example	Шаблон змінних середовища.	Без реальних секретів.
README.md	Інструкція запуску.	Бажано актуалізувати перед задачею.

4.11 Матриця трасування вимог

Щоб показати зв'язок між вимогами, реалізацією та перевіркою, у роботі сформовано матрицю трасування. Вона допомагає уникнути ситуації, коли у вступі заявлено одну функцію, у розділі реалізації описано іншу, а під час тестування перевіряється третя. Для дипломної роботи така матриця корисна ще й тим, що по ній легко готуватися до захисту.

Матриця не є повною специфікацією всього коду, але вона фіксує основні функціональні блоки: автентифікацію, профілі, команди, турніри, матчі, повідомлення, адміністративну частину, аналітику та запуск. Кожен пункт має модуль реалізації і спосіб перевірки.

Таблиця 34 - Матриця трасування вимог

ID	Вимога	Реалізація	Перевірка
FR-0 1	Реєстрація та вхід користувача	AuthModule, LoginScreen	T-03, T-04, T-05
FR-0 2	Збереження і відновлення сесії	SessionsModule, Secure Storage	T-06
FR-0 3	Керування ролями користувача	UsersModule, RolesGuard, AdminUsersScreen	T-26, T-27
FR-0 4	Створення профілю гравця	PlayersModule, MyPlayerProfileScreen	T-07
FR-0 5	Перевірка унікальності nickname	PlayersService, MongoDB index	T-08
FR-0 6	Пошук і фільтрація гравців	PlayersScreen, EsportsRepository	T-10
FR-0 7	Створення та публікація команди	TeamsModule, TeamWizardScreen	T-11
FR-0 8	Керування ростером	TeamsModule, TeamAdminScreen	T-12
FR-0 9	Запрошення гравця до команди	Teams invitations, Notifications	T-13, T-14
FR-1 0	Створення турніру	TournamentsModule, TournamentWizardScreen	T-15
FR-1 1	Публікація турніру	Tournament lifecycle	T-16
FR-1 2	Реєстрація команди на турнір	TournamentParticipant	T-17
FR-1 3	Генерація турнірної сітки	BracketGeneratorService	T-18
FR-1 4	Перегляд деталей матчу	MatchesModule, MatchDetailScreen	T-19
FR-1 5	Оновлення рахунку	MatchesService, RealtimeGateway	T-20
FR-1 6	Синхронізація bracket	TournamentsService, MatchesService	T-21

ID	Вимога	Реалізація	Перевірка
FR-1 7	Керування veto і картами	MatchesModule	T-22
FR-1 8	Обробка технічної паузи	MatchesModule	T-23
FR-1 9	Повідомлення користувачів	NotificationsModule, NotificationsScreen	T-24, T-25
FR-2 0	Модерація скарг	ReportsModule, AdminReportsScreen	T-28
FR-2 1	Оголошення платформи	AnnouncementsModule	T-29
FR-2 2	Локальний fallback	EsportsRepository, Hive/cache	T-30

4.12 Сценарій підготовки до захисту

Підготовку до захисту доцільно починати не з презентації, а з перевірки робочого середовища. Спочатку потрібно переконатися, що Docker Desktop запущений, порти не зайняті іншими процесами, а в .env немає застарілих значень. Після цього можна запускати docker compose і перевіряти health endpoint.

Другим кроком є підготовка тестових даних. Якщо база порожня, мобільний застосунок виглядатиме бідно, навіть якщо функціональність реалізована. Тому перед демонстрацією потрібно виконати seed і переконатися, що в системі є гравці, команди, турніри та матчі.

Третій крок - запуск мобільного клієнта. На Android emulator потрібно використовувати 10.0.2.2 для доступу до backend на локальному комп'ютері. На фізичному телефоні потрібно вказувати LAN IP комп'ютера і перевірити, що пристрій знаходиться в тій самій мережі.

Четвертий крок - підготовка акаунтів різних ролей. Для демонстрації зручно мати окремого адміністратора, турнірного оператора, менеджера

команди і звичайного гравця. Це дає змогу показати не лише різні екрани, а й обмеження доступу.

П'ятий крок - короткий показ основного сценарію. Наприклад: вхід, список гравців, профіль гравця, команда, турнір, bracket, матч, оновлення рахунку, повідомлення, адміністративна панель. Такий порядок добре показує, як дані рухаються системою.

Шостий крок - пояснення технічної архітектури. Тут важливо не заглиблюватися в кожен файл, а показати головні частини: Flutter-клієнт, NestJS API, MongoDB, Redis, Socket.IO, Docker і Swagger. Для цього в записці залишено місця під схеми і скріншоти.

Сьомий крок - чесно назвати обмеження. У проєкті є основа для Firebase push і production-розгортання, але для реальної експлуатації потрібні додаткові налаштування. Таке пояснення виглядає краще, ніж спроба представити навчальний прототип як повністю готовий комерційний продукт.

Восьмий крок - підготувати відповіді на типові питання. Наприклад, чому вибрано MongoDB, як працюють ролі, чому потрібен Redis, як формується bracket, що буде при недоступному backend, як захищені адміністративні маршрути і де зберігається токен користувача.

Окремо перед друком записки потрібно вставити реальні скріншоти в позначені місця. Скріншоти бажано робити з одного середовища запуску, щоб назви команд, гравців і турнірів збігалися між різними екранами.

Після вставлення скріншотів необхідно оновити нумерацію рисунків, таблиць і зміст. Це останній крок оформлення, але він важливий, бо після додавання зображень може змінитися кількість сторінок і розташування підрозділів.

4.13 План супроводу та подальшої підтримки

Після завершення дипломної роботи програмний продукт можна підтримувати як навчальний проєкт або поступово розвивати до повнішої платформи. Для цього варто мати простий план супроводу: як оновлювати

залежності, як перевіряти працездатність, як додавати нові модулі і як не втратити сумісність між мобільним клієнтом та API.

Першим напрямом супроводу є оновлення залежностей. Flutter, NestJS, Mongoose, Firebase-бібліотеки та інші пакети змінюються досить швидко. Перед оновленням потрібно перевіряти changelog, запускати build і проходити основні сценарії. Не бажано оновлювати всі великі залежності одночасно без проміжної перевірки.

Другим напрямом є стабілізація тестів. Наявні smoke-тести варто розширити unit-тестами для сервісів Auth, Players, Teams, Tournaments і Matches. Для найбільш важливих сценаріїв доцільно додати integration або e2e-тести, які піднімають тестову базу і перевіряють реальний HTTP-запит.

Третій напрям - покращення спостережуваності. У production-версії потрібно мати журнал помилок, basic metrics, перевірку health і зрозумілий механізм аналізу інцидентів. У дипломному проєкті вже є health endpoint і audit logs, тому їх можна розвивати далі.

Четвертий напрям - робота з даними. Для реальної платформи потрібно продумати резервне копіювання MongoDB, політику видалення даних, імпорт зовнішніх матчів і правила верифікації профілів. Без цього база може швидко стати неузгодженою.

П'ятий напрям - розширення UI. Мобільний застосунок уже має головні екрани, але надалі можна додати більш докладну статистику, live scoreboard, календар матчів, сторінку організатора, веб-адміністративну панель або адаптивний web-клієнт.

Шостий напрям - покращення безпеки. Потрібно додати реальну інтеграцію OAuth, refresh-token rotation, production HTTPS, обмеження CORS, захист секретів, rate limit для чутливих маршрутів і регулярний перегляд адміністративних ролей.

Таблиця 35 - План супроводу системи

Напрямок	Дія	Примітка
Оновлення	Раз на 1-2 місяці перевіряти Flutter,	Після оновлення

Напря́м	Дія	Примі́тка
залежностей	NestJS, Mongoose, Firebase, Dio.	виконувати build і smoke-тести.
Резервне копіювання	Створити регламент backup/restore MongoDB.	Перевіряти не лише створення, а й відновлення копії.
Автоматизовані тести	Розширити unit та integration tests.	Почати з Auth, Tournaments і Matches.
CI/CD	Додати pipeline lint/test/build.	Окремо для backend і Flutter APK.
Логи та моніторинг	Збирати помилки API і health-метрики.	Потрібно для production.
Безпека	Оновити JWT/refresh policy, OAuth, HTTPS, CORS.	Не зберігати секрети в репозиторії.
Дані	Описати правила імпорту, верифікації і видалення.	Зменшує хаос у базі.
UI/UX	Перевіряти мобільні екрани на різних розмірах.	Особливо форми wizard і таблиці admin.
Документація	Підтримувати README, .env.example і Swagger.	Документація має відповідати коду.
Релізи	Фіксувати версії і зміни.	Зручно для захисту і майбутнього розвитку.

4.14 Практична цінність результатів

Практична цінність роботи полягає в тому, що результатом є не окремий макет і не набір непов'язаних екранів, а повна клієнт-серверна система. У ній є мобільний інтерфейс, backend API, база даних, авторизація, ролі, локальне зберігання, запуск через Docker і документація API. Це дає змогу використовувати проєкт як основу для подальших навчальних або прикладних робіт.

Для організатора невеликого турніру система може бути корисною як прототип інструмента, який замінює розрізнені таблиці, повідомлення в чатах і ручне ведення результатів. Навіть якщо частину функцій потрібно доробити до production-рівня, загальна структура вже показує, як мають бути пов'язані команди, турніри, матчі та повідомлення.

Для команди або гравця практична користь полягає в централізованому профілі. Гравець може мати власний опис, дисципліну, країну, статистику і прив'язку до команди. Команда, у свою чергу, має ростер, менеджера, статус і можливість участі в турнірах.

Для адміністратора система демонструє, як поєднати керування користувачами, ролями, скаргами, оголошеннями та платформними налаштуваннями. Це важливо, тому що будь-який відкритий інформаційний сервіс потребує не тільки публічних сторінок, а й внутрішніх інструментів контролю.

Для навчального процесу робота корисна тим, що об'єднує кілька сучасних технологій в одному проєкті. Flutter показує побудову мобільного UI, NestJS - модульну серверну архітектуру, MongoDB - документну модель даних, Redis - кешування, Socket.IO - realtime-оновлення, Docker - відтворюване середовище запуску.

Окремою цінністю є те, що проєкт має місця для розвитку. Його можна розширювати web-клієнтом, повною production push-інтеграцією, імпортом зовнішніх матчів, складнішою статистикою, CI/CD і повнішим набором автоматизованих тестів. Тобто робота не закінчується лише захистом, а може бути базою для наступних ітерацій.

Висновки до розділу 4

1. Наведено вимоги до середовища запуску backend, мобільного застосунку, бази даних, Redis, Docker і мережевої доступності.
2. Описано порядок локального запуску через Docker і окремий режим для hot reload backend.

3. Визначено склад інсталяційного пакета і файли, які потрібно зберегти для відтворення проєкту.
4. Описано основні користувацькі сценарії та обмеження поточної версії, а також сформовано напрями подальшого розвитку.

ВИСНОВКИ

1. У бакалаврській кваліфікаційній роботі проаналізовано предметну область кіберспортивних інформаційних систем, визначено потребу в мобільній клієнт-серверній платформі для керування гравцями, командами, турнірами, матчами, повідомленнями, модерацією та аналітикою.
2. Виконано постановку завдання, сформовано об'єкт, предмет, мету, задачі, функціональні та нефункціональні вимоги. Визначено ролі Admin, Tournament Operator, Organizer, Team Admin, Player і Viewer та описано їх відповідальність.
3. Розроблено інформаційне забезпечення системи на базі MongoDB/Mongoose. Описано ключові колекції, поля, ObjectId-зв'язки, enum-статуси, timestamps та індекси для users, players, teams, tournaments, matches, notifications, reports та інших модулів.
4. Обґрунтовано вибір технологій Flutter, NestJS, MongoDB, Redis, Socket.IO, Docker і Nginx. Показано, як ці технології взаємодіють у межах локального та потенційного production-розгортання.
5. Реалізовано серверну частину з модульною структурою, глобальними guards, DTO validation, Swagger-документацією, JWT-автентифікацією, рольовим доступом, кешуванням, realtime gateway, seed-скриптами і адміністративними endpoint.
6. Реалізовано мобільний Flutter-клієнт з навігацією GoRouter, станом Riverpod, HTTP-шаром Dio, локальним збереженням сесій, fallback-даними, екранами dashboard, players, teams, tournaments, matches, profile, notifications, team admin, tournament operator і admin panel.
7. Описано алгоритми текстового пошуку, генерації турнірної сітки, оновлення результатів матчів, просування переможців, формування повідомлень, локального кешу і realtime-синхронізації змін.
8. Підготовлено рекомендації щодо локального запуску, складу інсталяційного пакета, тестування, вимог до середовища і напрямів

розвитку: production push, OAuth, повнотекстовий пошук, розширення double-elimination, CI/CD і ширше тестове покриття.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до розробки бакалаврської кваліфікаційної роботи для студентів спеціальності 122 «Комп'ютерні науки» освітнього ступеня «Бакалавр» / уклад. Б. Л. Голуб, Р. А. Руденський. Київ : НУБіП України, 2025. 57 с.
2. Загальні вимоги до змісту магістерської (бакалаврської) роботи : методичний файл кафедри комп'ютерних наук НУБіП України.
3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.
4. ДСТУ ГОСТ 7.1:2006. Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання. Київ : Держспоживстандарт України, 2007. 47 с.
5. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.
6. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2020. 705 p.
7. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.
8. HLTV.org. Counter-Strike matches and livescore. URL: <https://www.hltv.org/matches> (дата звернення: 19.05.2026).
9. Liquipedia Counter-Strike Wiki. URL: https://liquipedia.net/counterstrike/Main_Page (дата звернення: 19.05.2026).
10. FACEIT. Beginners Guide. URL: <https://support.faceit.com/hc/en-us/articles/14996562458268-FACEIT-Beginners-Guide> (дата звернення: 19.05.2026).
11. FACEIT. Joining CS2 tournaments. URL: <https://support.faceit.com/hc/en-us/articles/9936015291036-Joining-CS2-tournaments> (дата звернення: 19.05.2026).

12. Toornament Knowledge Base. Structures available on Toornament. URL: <https://help.toornament.com/structures/which-format-for-my-tournament> (дата звернення: 19.05.2026).
13. Battlefy Help Center. Tournament Management. URL: <https://help.battlefy.com/en/collections/1517897-tournament-management> (дата звернення: 19.05.2026).
14. start.gg. Organize an event. URL: <https://www.start.gg/create> (дата звернення: 19.05.2026).
15. Flutter documentation. URL: <https://docs.flutter.dev/> (дата звернення: 19.05.2026).
16. Dart documentation. URL: <https://dart.dev/guides> (дата звернення: 19.05.2026).
17. NestJS Documentation. URL: <https://docs.nestjs.com/> (дата звернення: 19.05.2026).
18. MongoDB Manual. URL: <https://www.mongodb.com/docs/manual/> (дата звернення: 19.05.2026).
19. Mongoose Documentation. URL: <https://mongoosejs.com/docs/> (дата звернення: 19.05.2026).
20. Redis Documentation. URL: <https://redis.io/docs/> (дата звернення: 19.05.2026).
21. Socket.IO Documentation. URL: <https://socket.io/docs/> (дата звернення: 19.05.2026).
22. Docker Documentation. URL: <https://docs.docker.com/> (дата звернення: 19.05.2026).
23. Nginx Documentation. URL: <https://nginx.org/en/docs/> (дата звернення: 19.05.2026).
24. JSON Web Token (JWT), RFC 7519. URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 19.05.2026).

25. OWASP Application Security Verification Standard. URL: <https://owasp.org/www-project-application-security-verification-standard/> (дата звернення: 19.05.2026).
26. Riverpod documentation. URL: <https://riverpod.dev/> (дата звернення: 19.05.2026).
27. Dio package documentation. URL: <https://pub.dev/packages/dio> (дата звернення: 19.05.2026).
28. Firebase Cloud Messaging documentation. URL: <https://firebase.google.com/docs/cloud-messaging> (дата звернення: 19.05.2026).
29. VOID Esports Database : вихідний код проєкту, README.md, README-no-docker.md, docs/application-description-ua.md, apps/api, apps/mobile. Локальний репозиторій H:\Void - Esports Database.

ДОДАТОК А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Чех Сергій Олександрович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система обліку кіберспортивних турнірів» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - ☐ інструменти генеративного ШІ не використовувалися.
 - ☒ інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____ інші (вказати назву)) були використані для:
 - ☐ перекладу іншомовних джерел;
 - ☒ стилістичного редагування та перевірки граматики;
 - ☒ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____
(підпис)

Сергій ЧЕХ