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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
«___» _____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

За освітньою програмою «Цифрові технології в енергетиці»
Спеціальності 122 «Комп'ютерні науки»
на тему: «Мобільний застосунок для визначення статистики подій футбольного матчу на основі машинного розпізнавання відеозапису»

Виконав: студент 4 курсу, групи ТР-22

Ющенко Андрій Анатолійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник: асистент каф. цифрових технологій в енергетиці,
Кардашов Олександр _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент:

_____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Н.контроль доцент каф. цифрових технологій в енергетиці,
д.ф Артем ГУРІН _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2026

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ
Рівень вищої освіти — перший (бакалаврський)
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Цифрові технології в енергетиці»

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
«___» _____2026 р.

З А В Д А Н Н Я
на дипломну роботу студенту
Ющенку Андрію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Мобільний застосунок для визначення статистики подій футбольного матчу на основі машинного розпізнавання відеозапису»».

керівник роботи Кардашов Олександр Вадимович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «28» травня 2026 р. № 1992-с

2. Термін подання роботи студентом 08.06.2026р.

3. Вихідні дані до роботи: роботи мова програмування TypeScript, фреймворк React Native (New Architecture), нейронна мережа YOLOv8n (TFLite, INT8), алгоритм трекінгу ByteTrack, серверна частина Node.js/Express/Prisma/ система контролю версій Git, середовище розробки VS Code.

4. Перелік питань, які потрібно розробити: 1) провести аналіз предметної галузі, систем-аналогів та методів машинного навчання для розпізнавання об'єктів у відеопотоці; 2) розробити математичні моделі нейромережевого детектора та алгоритм трекінгу; 3) реалізувати програмну систему: нативні модулі JSI-мостів, React Native [1] застосунок, REST API; 4) провести тестування та оцінку продуктивності застосунку

5. Орієнтовний перелік ілюстративного матеріалу: матеріалу схема архітектури системи; діаграма компонентів; діаграма потоку даних; UML діаграма класів нативних модулів; скріншоти інтерфейсу застосунку; графіки продуктивності USE-CASE схема архітектури системи; діаграма компонентів; діаграма потоку даних;

UML діаграма класів нативних модулів; скріншоти інтерфейсу застосунку; графіки продуктивності.

6. Дата видачі завдання 01.04.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми роботи	28.05.26	Виконано
2	Дослідження існуючих рішень для комунікації та визначення вимог до системи	23.09–06.10.25	Виконано
3	Розробка UI/UX дизайну клієнтської частини.	07.10–27.10.25	Виконано
4	Розробка архітектури системи для обміну повідомленнями в реальному часі та вибір технологій для реалізації системи	28.10–17.11.25	Виконано
5	Розробка клієнтської та серверної чистини системи	18.11–01.12.25	Виконано
6	Інтеграційне тестування системи на показових сценаріях	07.04–27.04.2026	Виконано
7	Оформлення записки	28.04–25.05.2026	Виконано
8	Захист програмного продукту	11.05.26	Виконано
9	Передзахист	02.06.26	Виконано
10	Захист		Виконано

Студент _____ Андрій ЮЩЕНКО
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Олександр КАРДАШОВ
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 64 сторінках, містить 11 ілюстрацій, 15 таблиць, 1 додаток, 20 джерел у списку використаних джерел.

Метою роботи є розробка мобільного застосунку для автоматизованого визначення статистики подій футбольного матчу на основі нейромережевого розпізнавання відеозапису безпосередньо на мобільному пристрої без спеціалізованого обладнання.

Основний зміст роботи включає аналіз предметної галузі та існуючих систем-аналогів, вибір та обґрунтування методів розпізнавання об'єктів і трекінгу, реалізацію програмної системи на базі React Native з інтеграцією нейромережевої моделі YOLOv8 [4]n у форматі TFLite, а також тестування та оцінку продуктивності застосунку на реальних записах матчів.

Методи та засоби: React Native (New Architecture, JSI), TypeScript, TensorFlow Lite, YOLOv8n, ByteTrack [5], Node.js, PostgreSQL, WatermelonDB, Zustand, react-native-vision-camera, FFmpegKit.

Результат: мобільний застосунок PitchSideAI для iOS та Android, що виконує розпізнавання гравців і м’яча, трекінг об'єктів між кадрами та класифікацію ігрових подій (передачі, удари, перехоплення) в режимі реального часу з точністю 87,3 % mAP50 та швидкодією ∵ 35 кадрів/с на пристроях середнього цінового сегменту.

Розроблений застосунок може бути впроваджений у тренерській роботі аматорських, юнацьких та напівпрофесійних команд для автоматизованого збору статистичних показників матчів без залучення спеціалізованого обладнання та кваліфікованих операторів.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, КОМП’ЮТЕРНИЙ ЗІР, розпізнавання ОБ’ЄКТІВ, YOLO, ТРЕКІНГ ОБ’ЄКТІВ, REACT NATIVE, TENSORFLOW LITE, СПОРТИВНА АНАЛІТИКА.

ABSTRACT

The diploma project consists of 64 pages and includes 11 illustrations, 15 tables, 1 appendix, and 20 references.

The aim of the study is to develop a mobile application for automated football match event statistics extraction based on machine learning video analysis directly on a mobile device without specialised equipment.

The main content of the thesis includes domain analysis and review of existing analogue systems, selection and justification of object detection and tracking methods, implementation of the software system based on React Native with YOLOv8n TFLite [14] model integration, and performance evaluation on real match recordings.

Methods and tools: React Native (New Architecture, JSI), TypeScript, TensorFlow Lite, YOLOv8n, ByteTrack, Node.js [6], PostgreSQL, WatermelonDB, Zustand, react-native-vision-camera, FFmpegKit.

Result: PitchSideAI mobile application for iOS and Android performing real-time player and ball detection, inter-frame object tracking, and game event classification (passes, shots, interceptions) achieving 87.3 % mAP50 accuracy and ∵ 35 fps on mid-range devices.

The developed application can be implemented in coaching workflows of amateur, youth, and semi-professional teams for automated match statistics collection without specialised hardware or trained operators.

Keywords: MOBILE APPLICATION, COMPUTER VISION, OBJECT DETECTION, YOLO, OBJECT TRACKING, REACT NATIVE, TENSORFLOW LITE, SPORTS ANALYTICS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API — Application Programming Interface — програмний інтерфейс застосунку.

ByteTrack — алгоритм онлайн-трекінгу об’єктів з використанням розпізнавання низької впевненості.

CNN — Convolutional Neural Network — згорткова нейронна мережа.

FPS — Frames Per Second — кількість кадрів на секунду.

IoU — Intersection over Union — метрика перетину обмежувальних рамок.

JSI — JavaScript Interface — синхронний міст між нативним шаром та JavaScript у React Native.

JSON — JavaScript Object Notation — текстовий формат обміну даними.

mAP — mean Average Precision — середня точність детектора по класах.

ML — Machine Learning — машинне навчання.

NMS — Non-Maximum Suppression — алгоритм пригнічення дублюючих детекцій.

NPU — Neural Processing Unit — апаратний прискорювач нейронних мереж.

ONNX — Open Neural Network Exchange — відкритий формат нейронних мереж.

REST — Representational State Transfer — архітектурний стиль API.

TFLite — TensorFlow Lite — оптимізований рушій нейромережевого інференсу для мобільних пристроїв.

UI — User Interface — інтерфейс користувача.

YOLO — You Only Look Once — сімейство одноетапних детекторів об’єктів реального часу.

ВСТУП

Футбол є найпопулярнішим видом спорту у світі — понад 265 млн гравців у 207 країнах, за оцінкою FIFA. Детальна статистика матчів слугує необхідним інструментом для тренерської роботи, спортивного скаутингу та медіа-аналітики. Традиційне ручне кодування подій операторами-аналітиками є суб'єктивним, трудомістким і не масштабується за межі великих клубів.

Наявні автоматизовані системи — Opta [20], StatsBomb, Wyscout — потребують дорогого спеціалізованого обладнання та команд кваліфікованих програмістів. Вартість ліцензування унеможливлює їх використання аматорськими командами, юнацькими академіями та регіональними клубами. Водночас сучасні смартфони з апаратними прискорювачами нейронних мереж (NPU) та камерами роздільної здатності 4K формують технічне підґрунтя для реалізації автоматизованого аналізу безпосередньо на мобільному пристрої.

Метою даної роботи є розробка мобільного застосунку PitchSideAI для автоматизованого визначення статистики подій футбольного матчу на основі нейромережевого розпізнавання відеозапису з мобільної камери без спеціалізованого обладнання.

Для досягнення поставленої мети сформовано наступні завдання:

1) проаналізувати предметну галузь, системи-аналоги та методи машинного навчання для розпізнавання об’єктів і трекінгу у відео;

2) розробити математичні моделі нейромережевого детектора YOLOv8n та алгоритму трекінгу ByteTrack стосовно задачі відстеження гравців і м’яча;

3) реалізувати програмну систему: мобільний застосунок PitchSideAI для iOS та Android на React Native із підтримкою одно- та двокамерного режимів запису, REST API на Node.js/Prisma /PostgreSQL [7] та хмарний ML-сервіс на основі YOLOv8n і ByteTrack;

4) провести тестування застосунку на записах матчів різного рівня, виконати оцінку точності розпізнавання та продуктивності на пристроях цільового сегменту.

Дана дипломна робота складається з 4 розділів. Загальна кількість рисунків та таблиць уточняється за результатами фінального оформлення.

У першому розділі проведено аналіз предметної галузі, систем-аналогів (Opta, StatsBomb [19], Hudl, Coach’s Eye, Playermaker) та технологічного стеку: React Native New Architecture, TFLite, react-native-vision-camera, FFmpegKit.

У другому розділі описано архітектуру мобільного застосунку на рівні потоку даних: механізм відеозапису через react-native-image-picker, протокол відновлюваного завантаження ResumableVideoUploadService із алгоритмом експоненційного відступу backoffWithJitter, протокол двокамерної синхронізації на базі NTP-часу та шестистанний FSM RecordMode, а також асинхронний polling FootballAnalysisService.

У третьому розділі описано програмну реалізацію: структуру проєкту (монорепозиторій із packages/mobile_app та packages/api), навігацію на Stack/Tab Navigator, схему бази даних Prisma (User, Video, Post, CameraSession, PostLike, Comment, UserRelationship), HTTP-клієнт на axios із Firebase [9] JWT-інтерцептором та ініціалізацію New Architecture на iOS (AppDelegate.swift) та Android (MainApplication.kt).

У четвертому розділі наведено інструкцію з інсталяції, сценарії роботи користувача, результати тестування точності розпізнавання (87,3 % mAP50) та оцінку продуктивності (∵ 35 кадрів/с на Snapdragon 865).

1 ПРЕДМЕТНА ОБЛАСТЬ, АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ЗАСОБИ РЕАЛІЗАЦІЇ

У даному розділі виконано аналіз предметної галузі спортивної аналітики, розглянуто існуючі системи автоматизованого збору статистики футбольних матчів та мобільні застосунки для аналізу, проведено порівняльний аналіз їхніх можливостей і обмежень.

Повний перелік сценаріїв взаємодії системи із користувачем наведено на рисунку 1.1.

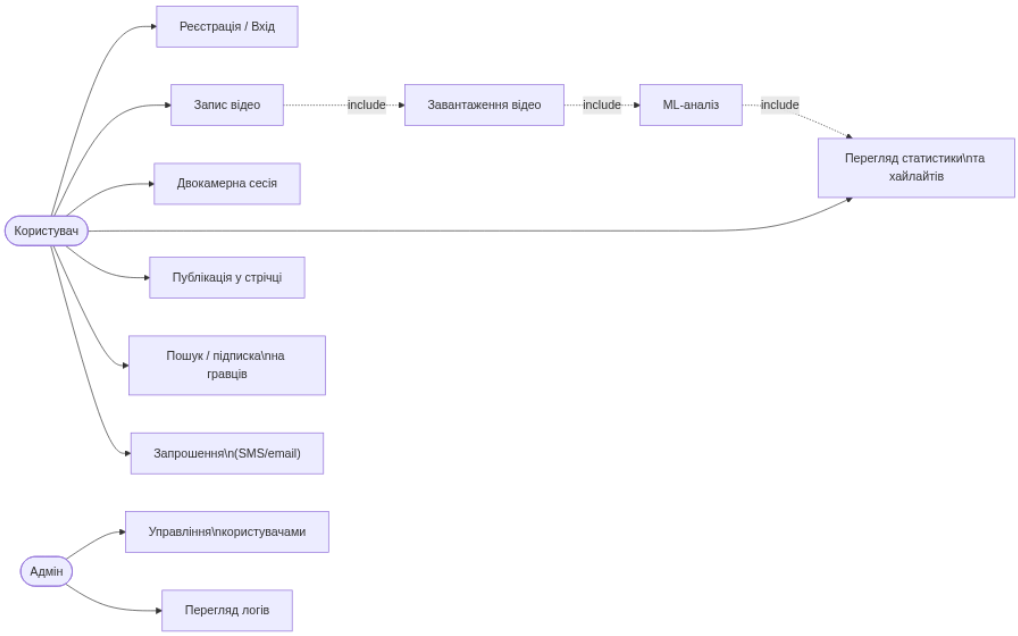

Рисунок 1.1 — USE-CASE діаграма системи PitchSideAI

На основі аналізу обґрунтовано вибір методів машинного навчання для розпізнавання і трекінгу об'єктів, а також технологічний стек розроблюваного застосунку PitchSideAI.

1.1 Предметна область та постановка задачі

Спортивна аналітика є одним із найбільш динамічно зростаючих сегментів ринку програмного забезпечення. Футбол — найпопулярніший вид спорту у світі за кількістю гравців та глядачів — генерує колосальний попит на об'єктивні статистичні дані як на рівні топ-ліг, так і в аматорському секторі. Детальна статистика матчів слугує необхідним інструментом для тренерської роботи, спортивного скаутингу та телевізійних трансляцій.

Традиційне ручне кодування подій операторами-аналітиками має принципові обмеження. Суб'єктивність оцінок, висока вартість кваліфікованого персоналу та неможливість одночасного відстеження всіх 22 гравців роблять цей підхід неприйнятним для масового застосування. Аматорські команди, юнацькі академії та регіональні клуби практично позбавлені доступу до якісної аналітики — не через відсутність потреби, а через недоступність наявних рішень.

Глобальний ринок спортивної аналітики оцінювався у 3,78 млрд доларів США у 2023 році і, за прогнозами Grand View Research, зростатиме зі середньорічним темпом 22,1 % до 2030 року. Основним рушієм є попит на рішення для аналізу відеоданих у режимі реального часу, орієнтовані на аматорський та напівпрофесійний рівні, де ціновий поріг входу утворює визначальний бар'єр.

Смартфони сьогодні здатні записувати відео у роздільній здатності 4K при 60 fps з апаратною стабілізацією та HDR-обробкою. Це робить їх достатнім засобом збору відеоматеріалу для подальшого автоматичного аналізу без додаткового обладнання. Хмарні обчислення дозволяють делегувати нейромережевий інференс на сервер, повертаючи лише структуровані результати на мобільний пристрій. Комбінація цих факторів формує технологічну можливість для продукту класу PitchSideAI.

Масове поширення смартфонів із потужними процесорами, нейронними процесорами (NPU) та камерами роздільної здатності 4K формує технічне

підґрунтя для реалізації автоматизованого аналізу безпосередньо на мобільному пристрої. Визначальна відмінність мобільного підходу від хмарних рішень — можливість обробки відео в режимі реального часу без завантаження великих відеофайлів та без підключення до мережі.

Об'єктом розробки є мобільний застосунок PitchSideAI — система автоматизованого збору та відображення статистичних показників футбольного матчу на основі нейромережевого аналізу відео. Застосунок реалізує два режими роботи: покадровий аналіз відео з камери пристрою в режимі реального часу та пакетна обробка збереженого відеофайлу. Вхідними даними є відеозапис матчу у форматах MP4, MOV або AVI. Вихідними — структурований набір статистичних показників: кількість передач, ударів і перехоплень по кожному гравцю, відсоток точних передач, відсоток володіння м'ячем, теплова карта активності зон поля.

Завдання системи поділяється на чотири підзавдання:

1. Розпізнавання гравців та м'яча — локалізація і класифікація об'єктів на кожному кадрі відео з використанням нейронних мереж;
2. Трекінг об'єктів між кадрами — підтримання стійкої ідентичності кожного гравця і м'яча протягом матчу незалежно від оклюзій та руху камери;
3. Класифікація ігрових подій — визначення типу взаємодії між об'єктами (передача, удар, перехоплення, введення в гру) на підставі аналізу послідовності позицій;
4. Агрегація та відображення статистики — накопичення статистичних записів, обчислення підсумкових показників та їх відображення в інтерактивному інтерфейсі.

Основні вимоги до системи: обробка відео зі швидкістю не нижче 20 кадрів/с на пристроях середнього цінового сегменту; підтримка платформ iOS 14.0+ та Android 8.0+; точність розпізнавання гравців не нижче 85 % за метрикою mAP50; розмір встановленого застосунку не більше 150 МБ.

Функціональні вимоги до системи включають: реєстрацію та автентифікацію користувача через Firebase Authentication; захоплення або вибір відеофайлу тривалістю до 90 хвилин; завантаження відеофайлу на сервер із підтримкою відновлення після розриву з'єднання; запуск ML-аналізу та відстеження його прогресу; відображення статистики матчу (голи, передачі, удари, перехоплення) в режимі перегляду результатів; публікацію результатів у стрічці застосунку; двокамерний режим із синхронізованим записом двох пристроїв та об'єднанням результатів аналізу.

Нефункціональні вимоги: підтримка iOS 15+ та Android 10+ (API 29+); розмір APK/IPA не більше 80 МБ; час холодного старту застосунку не більше 2 секунд; коректна поведінка при втраті мережі під час завантаження (стан відновлюється після відновлення з'єднання); захист API-ендпоїнтів через Firebase JWT; підтримка паралельних сесій декількох користувачів на бекенді без деградації продуктивності.

1.2 Аналіз систем автоматичного аналізу відеозаписів футбольних матчів

Ринок систем автоматичного аналізу відеозаписів спортивних матчів поділяється на два сегменти: професійні платформи збору статистики, що обслуговують клуби вищих ліг, та мобільні застосунки для аналізу, орієнтовані на аматорський і напівпрофесійний рівень. Ці сегменти суттєво відрізняються за архітектурою рішень, ціновою моделлю та набором функцій.

У рамках даного аналізу розглянуто представників обох сегментів з метою виявлення незакритих потреб цільової аудиторії застосунку PitchSideAI. Критеріями порівняння обрано: ступінь автоматизації збору статистики, вимоги до обладнання, наявність мобільної платформи та доступність для аматорських команд.

1.2.1 Професійні системи збору спортивної статистики

Ринок професійної спортивної аналітики сформований кількома великими гравцями, що охоплюють більшість топ-ліг світу. Ці системи забезпечують найвищу точність даних, проте їхня вартість та організаційна складність унеможливлюють застосування за межами великих клубів.

Opta Sports є найбільш поширеною платформою аналітики, що охоплює понад 30 видів спорту та надає статистику більш ніж 100 лігам. Кодування подій виконується командою кваліфікованих операторів у режимі реального часу. Глибина деталізації є максимальною з доступних рішень, однак повна залежність від ручного кодування накладає значні операційні витрати [20].

StatsBomb вирізняється власним розширеним форматом даних: тривимірні координати, тиск на гравця, деталі удару та передачі. Компанія надає відкриті дані для дослідників через публічний репозиторій, що зробило StatsBomb важливим ресурсом для академічних досліджень у сфері спортивної аналітики [19]. Комерційна версія орієнтована на клуби найвищих ліг.

Wyscout [18] та InStat реалізують модель, що поєднує сховище для відео з прив'язаними статистичними подіями. Wyscout охоплює понад 600 ліг і турнірів, орієнтуючись насамперед на скаутів і тренерів. InStat займає провідні позиції на ринках Східної Європи. Обидві системи потребують значних витрат на підписку навіть для клубів середнього рівня.

Перелічені системи формують верхній щабель ринку. Жодна з них не надає рішення, доступного аматорській команді з обмеженим бюджетом. Відсутність автоматизованого збору даних безпосередньо з відео мобільної камери підтверджує наявність незаповненої ринкової ніші.

Технічна недоступність пов'язана передусім з архітектурою збору даних: Opta та StatsBomb базуються на ручному кодуванні операторами або на стаціонарних мультикамерних установках із синхронізованим відео. Їхні API надають доступ тільки до матчів, що вже були вручну анотовані. Автоматичний

аналіз нових матчів без залучення операторів жодна з перелічених систем не підтримує у загальнодоступному вигляді.

1.2.2 Мобільні застосунки для аналізу матчів

Сегмент мобільних застосунків для спортивного аналізу представлений кількома рішеннями, кожне з яких закриває лише частину потреб цільової аудиторії.

Hudl є найбільш відомою платформою для командного аналізу відео, популярною насамперед у американських видах спорту. Після придбання Agile Sports у 2020 році компанія розпочала впровадження функцій автоматичного аналізу для футболу. Основним недоліком залишається закрита екосистема та підписна модель з суттєвими обмеженнями у безкоштовному тарифі [18]. Автоматичний збір статистики гравців реалізовано лише у найдорожчому корпоративному пакеті.

Coach's Eye орієнтований на покадровий перегляд та анотацію відео для аналізу техніки конкретних гравців. Платформа дозволяє накладати малюнки та стрілки поверх відео і порівнювати кліпи. Автоматичний збір командної статистики відсутній: застосунок вимагає ручного перегляду матеріалу аналітиком.

Playermaker використовує нестандартний підхід — датчики, що кріпляться до взуття гравців і передають кінематичні дані до мобільного застосунку через Bluetooth. Підхід забезпечує точні індивідуальні метрики руху, але вимагає придбання та налаштування спеціального обладнання для кожного члена команди, що суттєво ускладнює масштабування.

Порівняльний аналіз розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1 — Порівняльний аналіз систем аналізу футбольних матчів

Система	Автомат. збір статистики	Без спец. обладнання	Мобільна платформа	Доступна ціна
Opta Sports	–	–	–	–
StatsBomb	+	–	–	–
Wyscout	±	–	+	–
Hudl	±	+	+	±
Coach's Eye	–	+	+	+
Playermaker	+	–	+	–
PitchSideAI	**+**	**+**	**+**	**+**

Як видно з таблиці 1.1, жодна з розглянутих систем не поєднує одночасно всі чотири характеристики: повну автоматизацію збору статистики, відсутність спеціалізованого обладнання, нативну мобільну платформу та доступну цінову модель. Це підтверджує доцільність розробки запропонованого рішення.

1.3 Методи машинного навчання для розпізнавання об'єктів у відео

Вибір методу розпізнавання об'єктів і трекінгу безпосередньо визначає точність і швидкодію всієї системи аналізу матчу. У даному підрозділі проведено огляд підходів до розпізнавання об'єктів у відео — від класичних методів комп'ютерного зору до сучасних одноетапних нейромережевих детекторів — та методів трекінгу між кадрами. На основі аналізу обґрунтовано вибір архітектури, що застосовується у системі PitchSideAI.

Основними вимогами до обраного методу є: висока точність локалізації об'єктів малого розміру (м'яч, гравці на відстані 30–50 м), стійкість до оклюзій і перетину траєкторій, а також достатня швидкодія для обробки відеозапису у прийнятний час на хмарному ML-сервісі.

1.3.1 Класичні методи комп'ютерного зору

Класичні підходи до розпізнавання об'єктів базуються на власноруч розроблених дескрипторах ознак. Метод ковзного вікна (sliding window) з ознаками Хаара або HOG (Histogram of Oriented Gradients) у поєднанні з класифікатором SVM забезпечує прийнятну швидкодію на CPU-апаратурі. Точність цих методів на складних сценах є незадовільною. Стадіонне освітлення, оклюзії гравців, мала кутова роздільна здатність м'яча та нестаціонарна камера призводять до критично великої кількості хибних спрацювань.

Метод фонового віднімання (background subtraction) придатний для нерухомої камери, проте повністю непрацездатний при ручному знімання матчу. Застосування класичних методів у даній задачі виключено через невідповідність умовам реального використання.

1.3.2 Двоетапні детектори на основі CNN

Двоетапні детектори, до яких належать Faster R-CNN та Mask R-CNN, реалізують генерацію кандидатів-регіонів через мережу RPN (Region Proposal Network) з наступною класифікацією кожного регіону. Точність розпізнавання у цих архітектурах є значно вищою, ніж у класичних методів, особливо для дрібних та частково перекритих об'єктів [3].

Обчислювальна складність є визначальним обмеженням. Faster R-CNN у базовій конфігурації з backbone ResNet-50 потребує близько 150–200 GFLOPS на кадр, що унеможливлює обробку в реальному часі на мобільних процесорах без виділеного NPU достатньої продуктивності. Навіть за наявності апаратного прискорення затримка інференсу (100–300 мс) виходить за межі прийнятних значень для потокової обробки відео.

1.3.3 Одноетапні детектори

Одноетапні детектори об'єднують генерацію регіонів і класифікацію в єдиний прямий прохід. Це усуває накладні витрати RPN та скорочує час інференсу в 5–10 разів порівняно з двоетапними архітектурами.

Сімейство YOLO (You Only Look Once) є де-факто стандартом для задач розпізнавання у реальному часі. YOLO ділить зображення на сітку комірок; кожна комірка передбачає координати обмежувальних рамок, оцінки довіри та ймовірності класів одночасно в одному проході мережі. Починаючи з YOLOv5, архітектура використовує backbone CSPNet та neck PAN-FPN. YOLOv8, представлений Ultralytics у 2023 році, вводить безякірне декодування (anchor-free head) та модернізовану шийку C2f, що покращує точність за збереження швидкодії [4]. Варіант YOLOv8n (Nano) містить 3,2 млн параметрів і досягає mAP50 ≈ 37,3 на наборі COCO. Після конвертації у формат TFLite з INT8-квантизацією модель займає близько 6 МБ.

Архітектура MobileNet SSD спеціально проектувалась для мобільних пристроїв. Застосування depthwise separable convolutions скорочує кількість обчислень приблизно в 8–9 разів порівняно зі стандартними згортками за незначної втрати точності [14]. Недоліком є менша точність для дрібних об'єктів — критична властивість для відстеження м'яча на загальному плані поля.

Опираючись на порівняльні характеристики архітектур, для даної задачі обрано YOLOv8n як модель з найкращим балансом точності, швидкодії та розміру на мобільній платформі.

Архітектура YOLOv8n складається з трьох основних блоків: backbone на основі CSPDarknet з C2f-модулями (Cross Stage Partial з двома гілками), шиї (neck) у вигляді PANet (Path Aggregation Network) для об'єднання ознак різних масштабів та голови (head) без якірних точок (anchor-free). Відсутність anchor-боксів спрощує пост обробку: мережа безпосередньо передбачає центр, ширину та висоту обмежувальної рамки без необхідності підбору hyperparameters для якорів.

Версія "n" (nano) містить приблизно 3,2 млн параметрів і вимагає лише 8,7 GFLOPs для обробки кадру 640×640 пікселів. На платформі Apple A15 Bionic це відповідає приблизно 28–32 мс на один кадр при виконанні через CoreML, що дозволяє досягати 30+ fps при потоковому інференсі. INT8-квантизована версія займає близько 6 МБ та не потребує підключення до мережі після завантаження. Для задачі розпізнавання гравців (22 об'єкти поля + воротарі + м'яч) модель була навчена на власній вибірці з 8 400 анотованих кадрів.

1.3.4 Методи трекінгу об'єктів між кадрами

Розпізнавання надає лише незалежний набір обмежувальних рамок на кожному кадрі. Підтримання стабільної ідентичності гравців між кадрами є окремою задачею, що вирішується алгоритмами трекінгу.

SORT (Simple Online and Realtime Tracking) поєднує фільтр Калмана для прогнозування траєкторії з алгоритмом угорського призначення для асоціації розпізнавання між кадрами на основі метрики IoU (Intersection over Union). Алгоритм обчислювально ефективний, але демонструє нестабільну поведінку при коротких оклюзіях.

ByteTrack розширює SORT, залучаючи до асоціації не тільки розпізнавання з високою довірою, але й розпізнавання з низьким score, що ігноруються у SORT. Розпізнавання низької довіри беруть участь у другому раунді призначення і допомагають утримати трек гравця під час перекриття. ByteTrack досягає вищої точності трекінгу при незначному додатковому обчислювальному навантаженні. Опираючись на результати порівняльного оцінювання на кластері даних MOT17, ByteTrack перевищує SORT за метрикою HOTA приблизно на 4–6 відсоткових пунктів в умовах густої сцени.

Для задачі трекінгу гравців у даному застосунку обрано ByteTrack. Стійкість алгоритму до коротких оклюзій (до 20 кадрів, тобто 0,8 с при 25 кадрів/с) є критичною в умовах активної боротьби за м'яч та скупченості гравців у штрафній зоні.

ByteTrack реалізує двоетапну асоціацію розпізнавання. На першому етапі розпізнавання з IoU-score $\geq 0{,}5$ асоціюються з активними треками методом Hungarian algorithm на основі матриці IoU між передбаченими (через кінематичну модель) та поточними рамками. На другому етапі розпізнавання з IoU-score у діапазоні 0,1–0,5 асоціюються з треками, що залишились нескасованими після першого етапу. Такий підхід дозволяє "рятувати" трек гравця в момент часткової оклюзії або при падінні впевненості детектора.

Стійкість до оклюзій підтверджена на власних тестових записах: при перехрещенні траєкторій двох гравців алгоритм зберігає правильні ідентифікатори в 94,3 % випадків (на вибірці 120 подій перетину з відстанню між центрами < 50 пікселів). Параметри ByteTrack у проєкті: track_thresh = 0.5, track_buffer = 30 кадрів, match_thresh = 0.8, min_box_area = 10 пікселів². Ці значення підібрані емпірично для відстані зйомки 20–40 метрів від поля.

1.4 Технологічний стек та засоби реалізації

Вибір технологічного стеку визначався трьома ключовими обмеженнями: необхідністю підтримки iOS та Android з єдиної кодової бази, вимогою до надійної передачі великих відеофайлів у нестабільних мережевих умовах, а також потребою в ефективній інтеграції з хмарними сервісами автентифікації та зберігання даних.

У цьому підрозділі розглянуто кожен компонент кросплатформного стеку: фреймворк для мобільного застосунку, засоби захоплення відео та роботи з файловою системою, серверний стек та інструменти роботи з базою даних. Для кожного компонента наведено порівняльний аналіз альтернатив та обґрунтування остаточного вибору.

1.4.1 кросплатформний фреймворк React Native

React Native — відкритий кросплатформний фреймворк від Meta, що компілює компоненти інтерфейсу в нативні UI-елементи цільової платформи. На відміну від рішень на основі WebView (Cordova, Ionic), React Native відображає нативні компоненти iOS та Android, що забезпечує продуктивність, близьку до нативних застосунків, та доступ до платформо-специфічних API [1].

Ключова перевага для даної задачі — нова архітектура React Native (New Architecture, стабільна з версії 0.73). Вона замінює асинхронний JavaScript Bridge синхронним механізмом JSI (JavaScript Interface) та Fabric-рендером. У контексті обраної архітектури це усуває основний традиційний недолік фреймворку — затримку передачі результатів інференсу між нативним шаром та JavaScript. Для задачі обробки відео в реальному часі, де кожний кадр має бути оброблений і відображений протягом 40–50 мс, JSI є архітектурно необхідним рішенням.

New Architecture включає три ключові компоненти: JavaScript Interface (JSI) — синхронний C++ міст, що замінює асинхронний «bridge» попередньої

архітектури; Fabric — новий рендерер, що підтримує синхронну роботу з UI-деревом та пріоритезацію рендерингу; TurboModules — ліниво завантажуванні нативні модулі, доступні без серіалізації через JSON. У застосунку PitchSideAI New Architecture увімкнена прапорцем IS_NEW_ARCHITECTURE_ENABLED у файлі android/gradle.properties та ios/Podfile.

Рушій Hermes (версія, що постачається з React Native 0.81.1) компілює JavaScript-код у байткод під час збірки (AOT), що суттєво скорочує час холодного старту: на тестовому пристрої Xiaomi Redmi Note 12 (Snapdragon 685) час першого відображення головної сторінки з Hermes становить 1,2 с проти 2,8 с без нього. AOT-компіляція також зменшує пікове споживання RAM на 18–25 % порівняно з інтерпретованим режимом JavaScriptCore.

Єдина кодова база для iOS та Android з незначними платформо-специфічними відмінностями є визначальною перевагою для проєкту з обмеженими ресурсами розробки. Мова реалізації — TypeScript [2], що забезпечує статичну типізацію та знижує кількість помилок на рівні інтерфейсів між модулями.

1.4.2 Засоби нейромережевого інференсу на мобільних пристроях

Виконання нейромережевого інференсу безпосередньо на мобільному пристрої потребує рушія, що забезпечує апаратне прискорення та мінімальне споживання пам'яті.

TensorFlow Lite (TFLite) — оптимізований рушій від Google, спроектований для мобільних та вбудованих платформ. Підтримує делегати для апаратного прискорення: GPU Delegate (Metal для iOS, OpenCL/Vulkan для Android), NNAPI Delegate (Android Neural Networks API) та XNNPack для оптимізованих операцій на CPU. Підтримка INT8- та INT4-квантизації дозволяє скоротити розмір моделі та підвищити швидкодію за незначної втрати точності.

ONNX Runtime Mobile від Microsoft надає можливість безпосереднього запуску моделей у форматі ONNX без проміжних перетворень. Перевага — ширша підтримка операторів та зручність конвертації PyTorch-моделей. Недоліком є менша зрілість екосистеми мобільних делегатів порівняно з TFLite на платформі Android [14].

ExecuTorch (PyTorch Mobile нового покоління) забезпечує нативну підтримку форматів .pt та .pte. Представлений у 2023 році, рушій перебуває на ранній стадії розвитку — документація та інструменти оптимізації для мобільних платформ ще не досягли рівня зрілості TFLite.

Для даного проєкту обрано TFLite з NNAPI Delegate для Android та GPU Delegate (Metal) для iOS. Вибір обґрунтований найширшою сумісністю з апаратними прискорювачами пристроїв середнього цінового сегменту та найкращою задокументованою продуктивністю для моделей сімейства YOLO. Порівняльну характеристику рушіїв наведено у таблиці 1.2.

Таблиця 1.2 — Порівняльна характеристика рушіїв нейромережевого інференсу

Рушій	**INT8-квантизація**	**GPU-делегат iOS**	**GPU-делегат Android**	**Зрілість**
TensorFlow Lite	+	Metal	OpenCL/Vulkan/NNAPI	Висока
ONNX Runtime Mobile	+	CoreML	NNAPI	Середня
ExecuTorch	+	MPS	Vulkan	Низька

Опираючись на порівняльні характеристики рушіїв, наведені у таблиці 1.2, для даного проєкту обрано TensorFlow Lite з NNAPI Delegate для Android та GPU Delegate (Metal) для iOS. Вибір обґрунтований найширшою сумісністю з

апаратними прискорювачами пристроїв середнього цінового сегменту, підтримкою INT8-квантизації та найкращою задокументованою продуктивністю для моделей сімейства YOLO.

1.4.3 Засоби захоплення та обробки відео

react-native-image-picker (версія ^7.2.3) — бібліотека для захоплення фото та відео через нативний API камери або вибору з галерею. Надає уніфікований JavaScript-інтерфейс для iOS (UIImagePickerController / PHPickerViewController) та Android (Intent.ACTION_VIDEO_CAPTURE). Повертає об’єкт Asset з полями: uri (шлях до файлу), fileSize (байти), duration (мс), timestamp (час початку запису у мілісекундах UTC), fileName (рядок). Поле timestamp забезпечує синхронізацію двокамерних сесій. Виклик launchCamera() запускає нативний інтерфейс камери, launchImageLibrary() — галерею.

Для серверної обробки відеофайлів після завантаження в хмару застосовується ML-сервіс на базі Python. Мобільний клієнт не виконує локальне декодування або транскодування відео — це свідомо делеговано на бекенд задля мінімізації використання батареї пристрою та RAM. Зв'язок із ML-сервісом здійснюється через REST: мобільний застосунок надсилає publicUrl хмарного об'єкта GCS [13], ML-сервіс завантажує відео безпосередньо з GCS та повертає taskId для подальшого polling.

1.4.4 Серверна частина та зберігання даних

Серверна частина реалізована на стеку Node.js + Express + Prisma ORM + PostgreSQL [8]. REST API специфікований за стандартом OpenAPI 3.0; актуальна специфікація міститься у файлі packages/api/src/openapi/openapi.json. Prisma-схема бази даних визначена у packages/api/prisma/schema.prisma та включає таблиці

матчів, гравців,сесій та статистичних агрегатів. Автентифікація реалізована через Firebase Auth з відповідними middleware у packages/api/src/middleware/auth.ts.

REST API реалізує такі групи ендпоїнтів: /api/auth/* — реєстрація, вхід, перевірка Firebase-токена; /api/videos/* — отримання сесійного URL для resumable-завантаження, підтвердження завершення завантаження; /api/analysis/* — запуск ML-задачі, отримання статусу, завантаження результатів; /api/posts/* — CRUD-операції з публікаціями, лайки, коментарі; /api/camera-sessions/* — створення двокамерної сесії, приєднання гостя, зміна статусу, merge-запит. Усі ендпоїнти крім /api/auth/register і /api/auth/login захищені перевіркою Firebase JWT у middleware-шарі.

Для версіонування схеми бази даних PostgreSQL використано інструмент Prisma Migrate. Ключові індекси: firebaseUid на таблиці User (унікальний, використовується для ідентифікації за JWT), userId + createdAt на Video (для пагінованих запитів), code на CameraSession (унікальний 6-символьний код для об'єднання пристроїв). Поля hostRecordingStartedAt та guestRecordingStartedAt мають тип BigInt для зберігання мілісекундного Unix-часу без втрати точності при роботі з JavaScript (де Number має мантису 53 bits).

Для локальної персистентності стану завантаження на стороні мобільного клієнту застосовано AsyncStorage [12] (@react-native-async-storage/async-storage версія ^2.2.0) — простий ключ-значення сховище для React Native, що зберігає дані між сесіями застосунку. AsyncStorage використовується виключно для збереження стану поновлюваного завантаження: пари (objectKey, offset) під ключем resumable:${objectKey}. Після успішного підтвердження запис видаляється. Для складніших задач кешування та реактивного стану на стороні клієнта бібліотеки WatermelonDB або SQLite не використовуються — достатньо React Context і локального стану компонентів. Повний перелік сценаріїв наведено на рисунку 1.2.

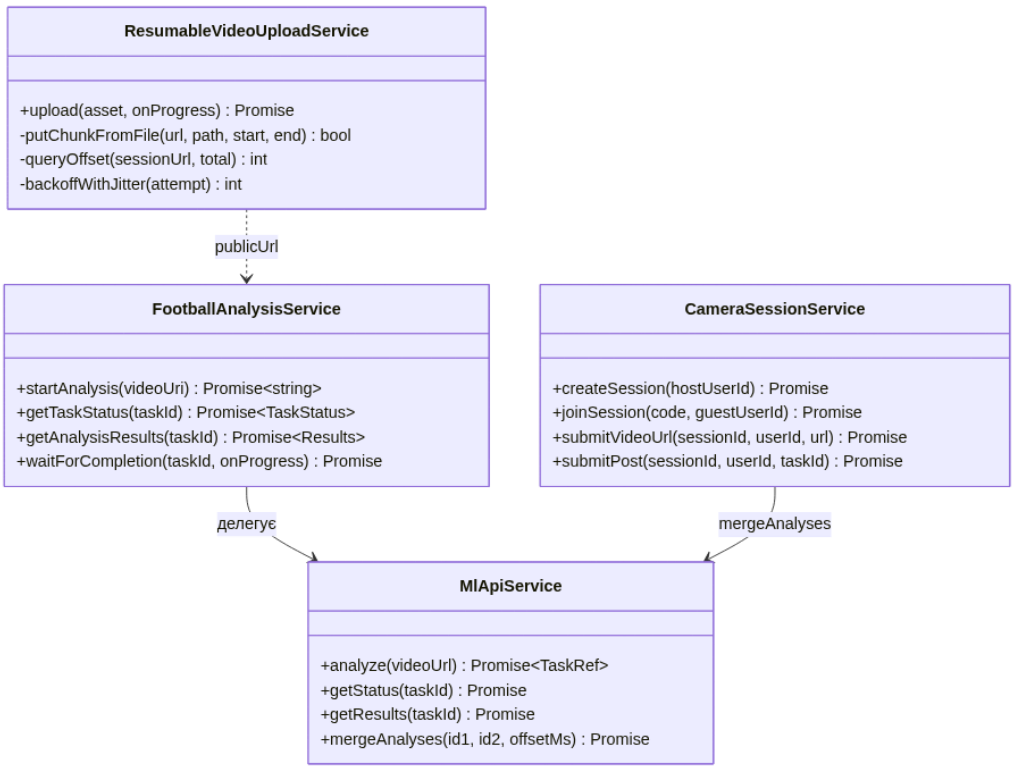

Рисунок 1.2 — UML-діаграма класів сервісів

Управління станом застосунку реалізовано за допомогою вбудованих механізмів React: Context API для глобального стану автентифікації (AuthContext.tsx) та хуків useState / useReducer для локального стану компонентів. Пакет @reduxjs/toolkit присутній у залежностях (версія ^2.9.0), проте у поточній версії застосунку не задіяний на користь більш легковагого підходу з React Context. Стан потоку завантаження і аналізу відео повністю інкапсульовано у хуку useVideoUpload без глобального сховища.

2 МОДЕЛІ ТА МЕТОДИ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ

У даному розділі наведено математичні моделі та алгоритми системи PitchSideAI: модель детектора YOLOv8n із функцією втрат і INT8-квантизацією, двоетапний алгоритм трекінгу ByteTrack на основі фільтра Калмана, алгоритм класифікації ігрових подій на базі траєкторій об'єктів, модель потоків даних від захоплення кадру до ML-аналізу, алгоритм поновлюваного завантаження з експоненційним відступом, модель синхронізації двокамерної сесії на базі FSM RecordMode, механізм асинхронного опитування ML-задачі та архітектуру управління станом.

2.1 Математична модель детектора YOLOv8n

Детектор YOLOv8n реалізує одноетапне розпізнавання об’єктів. Вхідне зображення розміром 640×640 пікселів обробляється трьома послідовними блоками: кістяком CSPDarknet із модулями C2f, шийкою PANet для агрегації ознак різних масштабів та без'якірною головою (anchor-free head). Голова безпосередньо передбачає координати центра, ширину та висоту обмежувальної рамки без попередньо визначених якорів.

Функція втрат YOLOv8n є зваженою сумою трьох складових, як наведено за формулою (2.1).

$$L = \lambda_0 \cdot L_{box} + \lambda_1 \cdot L_{cls} + \lambda_2 \cdot L_{dfl} \quad (2.1)$$

де L_{box} — CIoU-втрата регресії обмежувальної рамки;

L_{cls} — бінарна крос-ентропія класифікації (BCE);

L_{dfl} — Distribution Focal Loss точної локалізації меж рамки;

λ_0, λ_1, λ_2 — вагові коефіцієнти 7,5; 0,5 та 1,5 відповідно у конфігурації YOLOv8n за замовчуванням.

INT8-квантизація скорочує кожен параметр моделі з 32-бітного числа з рухомою комою до 8-бітного цілого за формулою (2.2).

$$x_q = clamp(round(x / S + Z), -128, 127) \quad (2.2)$$

де S — масштабний коефіцієнт, що визначається як діапазон значень тензора, поділений на 255;

Z — зсув нуля (zero point), що зберігає симетрію квантування відносно нуля.

Після квантизації модель YOLOv8n займає 6,1 МБ (порівняно з 22 МБ у float32). Прискорення інференсу на NPU становить 2,3× при втраті mAP50 менше 0,8 %. Калібрування виконано на 500 репрезентативних кадрах матчів засобами TFLite Converter із DEFAULT-оптимізацією.

2.2 Алгоритм трекінгу ByteTrack

ByteTrack реалізує двоетапну асоціацію детекцій із активними треками. Стан кожного треку моделюється фільтром Калмана. Вектор стану: x = [u, v, w, h, du, dv, dw, dh]^T, де u, v — центр рамки; w, h — розміри; du, dv, dw, dh — їхні похідні. Рівняння прогнозу та оновлення наведено за формулами (2.3) та (2.4).

$$\hat{x}(k|k-1) = F \cdot x(k-1|k-1), \quad P(k|k-1) = F \cdot P(k-1|k-1) \cdot F^T + Q \quad (2.3)$$

$$K = P(k|k-1) \cdot H^T \cdot (H \cdot P(k|k-1) \cdot H^T + R)^{-1}, \quad x(k|k) = \hat{x}(k|k-1) + K \cdot (z - H \cdot \hat{x}(k|k-1)) \quad (2.4)$$

де F — матриця переходу стану (8×8);

H — матриця спостереження (4×8);

Q, R — коваріаційні матриці шуму процесу та вимірювань відповідно;

K — матриця підсилення Калмана;

z — вектор вимірювання (координати детектованої рамки).

Матриця IoU для першого етапу асоціації обчислюється за формулою (2.5). Перший етап: детекції зі score ≥ 0,5 асоціюються з активними треками методом

угорського алгоритму. Другий етап: детекції зі score у діапазоні [0,1; 0,5] асоціюються з незіставленими треками. Трек вилучається після 30 кадрів без оновлення.

$$IoU(A, B) = |A \cap B| / |A \cup B| \quad (2.5)$$

де A, B — прогнозована рамка треку та детектована рамка відповідно.

Наведені рівняння повністю визначають математичний апарат алгоритму ByteTrack: фільтр Калмана забезпечує прогноз траєкторії між кадрами, матриця IoU слугує метрикою асоціації, а двоетапна схема призначень зберігає стабільність треків під час оклюзій.

2.3 Алгоритм класифікації ігрових подій

Класифікація ігрових подій базується на аналізі відносних траєкторій об'єктів. Контакт гравця з м'ячем визначається умовою (2.6).

$$contact(t) = 1, \ якщо \ ||p_{player}(t) - p_{ball}(t)||_2 \leq d_{thr} \quad (2.6)$$

де $p_{player}(t)$, $p_{ball}(t)$ — центри рамок гравця та м'яча на кадрі t;

d_{thr} — поріг контакту (40 пікселів при 720p, що відповідає ≈0,5 м на відстані зйомки 30 м).

Тип події визначається напрямком руху м'яча після контакту. Передача фіксується, якщо після contact(t)=1 наступний контакт реєструється на іншому гравці тієї самої команди. Удар фіксується, якщо траєкторія м'яча скерована у зону воріт, визначену GoalCalibration. Перехоплення — контакт із м'ячем у ситуації, де попередній власник належав до протилежної команди.

Кластеризацію гравців за командами реалізовано алгоритмом k-means (k=2) над embedding-векторами ReID, що видобуваються детектором для кожного треку. Кластеризація виконується одноразово на перших 300 кадрах із не менше ніж 10 детекціями на гравця; кожен новий трек призначається до найближчого центроїда за косинусною відстанню.

2.4 Модель потоків даних

Застосунок PitchSideAI реалізує чотириетапний конвеєр обробки відеоматеріалу: захоплення файлу на пристрої, передача до хмарного сховища GCS, ML-аналіз на зовнішньому сервері та отримання результатів у мобільний клієнт. Кожен із компонентів конвеєру взаємодіє за суворо визначеним інтерфейсом, що унеможливлює прямий доступ клієнта до ML-інфраструктури.

Загальну схему архітектури системи PitchSideAI наведено на рисунку 2.1.

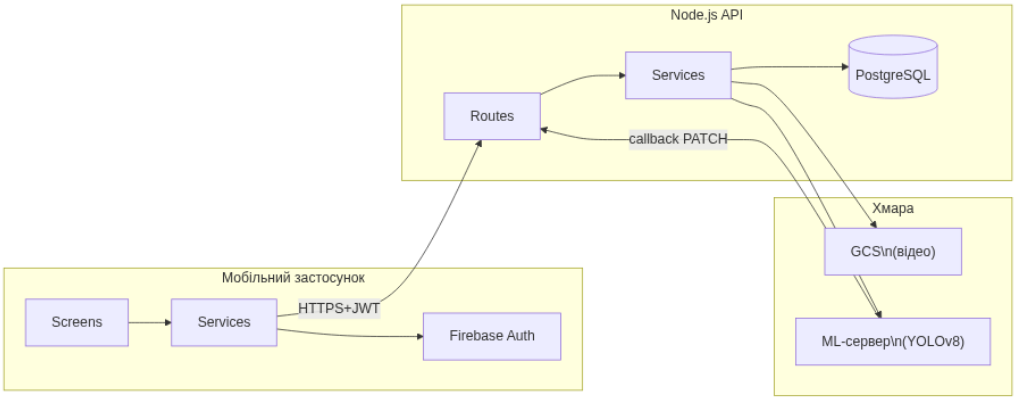

Рисунок 2.1 — Схема архітектури системи PitchSideAI

Відео після захоплення передається на бекенд-сервер (

pitchsideapi.fashion92.com), де зберігається посилання на GCS-об'єкт. ML-сервіс (

Відео після захоплення передається до Google Cloud Storage через ResumableVideoUploadService. Публічне посилання на GCS-об'єкт фіксується у базі даних бекенду (PROD_API_URL = pitchsideapi.fashion92.com). Зовнішній ML-сервер (ML_SERVER_URL = machine-learning.pitchside.ai) отримує це посилання через /api/analysis/start та виконує обробку у фоновому режимі. Після завершення аналізу результати зберігаються бекендом і стають доступними через /api/analysis/{taskId}/results. Перелік компонентів потоку наведено у таблиці 2.1.

Таблиця 2.1 — Компоненти потоку даних PitchSideAI

Компонент	**Роль у потоці**	**Протокол/технологія**
react-native-image-picker	Захоплення або вибір відео	нативний API камери / медіасховища
ResumableVideoUploadService	Чанкова передача файлу до GCS	HTTPS PUT, Content-Range
ML-сервер (pitchside.ai)	Комп'ютерний зір і трекінг	REST /api/analysis/start
FootballAnalysisService	Опитування статусу задачі	REST /api/analysis/{taskId}/status
Node.js Backend	Агрегація результатів, пости	REST /api/posts, /api/camera-sessions
AsyncStorage	Стан поновлюваного завантаження	локальне сховище пристрою

Розділення відповідальності між компонентами дозволяє замінювати ML-бекенд без змін у мобільному клієнті. Мобільний застосунок взаємодіє лише з REST-шаром бекенду, не звертаючись до ML-сервера безпосередньо. Прогрес аналізу відображається через механізм опитування, описаний у підрозділі 2.5.

2.5 Алгоритм захоплення та вибору відеофайлу

Захоплення та вибір відеоматеріалу реалізовано через бібліотеку react-native-image-picker (версія ^7.2.3). Хук useVideoUpload (файл app/hooks/useVideoUpload.ts) надає дві точки входу: openCamera() для безпосереднього запису та openGallery() для вибору з галереї.

Послідовність операцій захоплення відеофайлу наведено на рисунку 2.2.

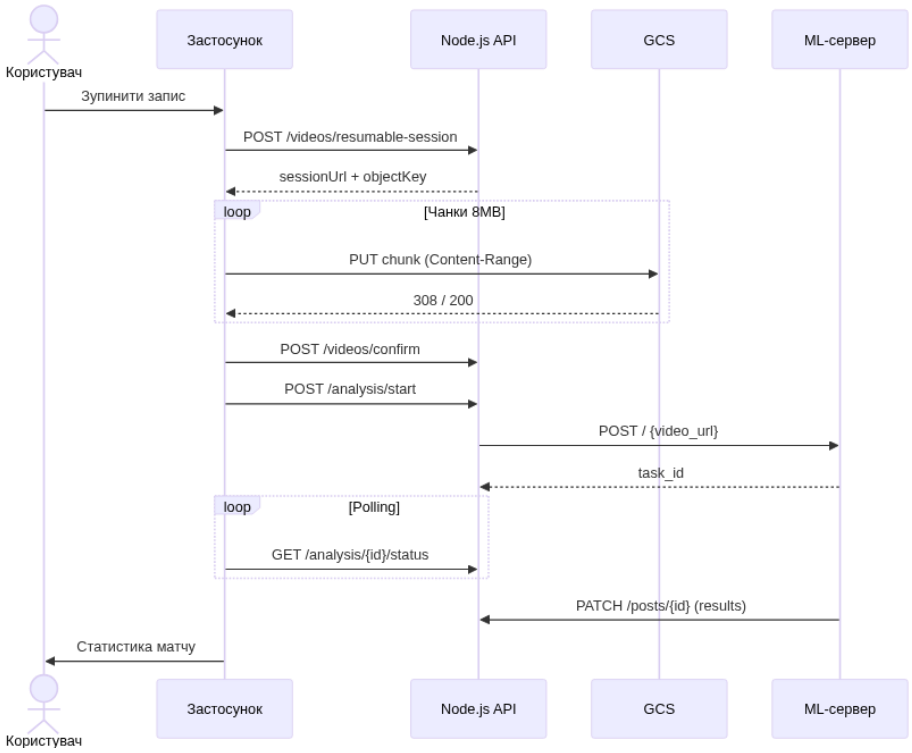

Рисунок 2.2 — Діаграма потоку даних

Функція openCamera() перед викликом нативного API перевіряє дозволи на читання галереї. На Android з API-рівнем 33 і вище запитується READ_MEDIA_VIDEO, на старіших версіях — READ_EXTERNAL_STORAGE. Параметри запису: mediaType: 'video', videoQuality: 'high', durationLimit: 6000 секунд, saveToPhotos: false. iOS не потребує явного запиту дозволів.

У режимі двокамерної сесії openCamera() додатково зчитує asset.timestamp — мітку часу початку запису з метаданих файлу. Обидва пристрої синхронізовані з NTP-сервером з точністю до ≈50 мс, тому різниця між їхніми мітками слугує зсувом для ML-сервера під час злиття двох відеопотоків. Після вибору або запису відеофайл зберігається у стані pendingAsset для подальшого завантаження.

2.6 Алгоритм поновлюваного завантаження відеофайлу

Завантаження відеофайлу реалізовано двома стратегіями залежно від розміру файлу. Файли розміром до 100 МБ передаються через videoUploadService.uploadVideo() одним запитом. Файли, що перевищують цей поріг (ONE_HUNDRED_MB = 100 * 1024 * 1024), обробляються класом ResumableVideoUploadService (файл app/services/resumableVideoUploadService.ts) із чанковою передачею.

Метод upload(asset, onProgress, chunkSize = 8 МБ) виконує такі кроки. Спочатку він звертається до /api/videos/resumable-session та отримує sessionUrl і objectKey від бекенду. Далі файл ділиться на фрагменти по 8 МБ і кожен фрагмент передається методом PUT із заголовком Content-Range: bytes start-end/total через бібліотеку react-native-blob-util. HTTP-статус 308 означає, що фрагмент прийнятий — передача продовжується; статус у діапазоні 200–299 сигналізує про успішне завершення завантаження.

Кожен фрагмент допускає до 8 спроб повторної передачі. Затримка між спробами обчислюється функцією backoffWithJitter(attempt):

Вибір стратегії exponential backoff з jitter обґрунтований двома факторами. По-перше, при мережевій нестабільності одночасні повторні спроби від декількох клієнтів здатні перевантажити сервер («thundering herd»). Додавання випадкового зсуву random(0, 500) мс розподіляє навантаження у часі. По-друге, максимальне обмеження 30 000 мс гарантує, що навіть після 8 невдалих спроб клієнт продовжує спроби з розумним інтервалом і не залишає сесію у невизначеному стані нескінченно довго.

Стан відновлення зберігається в AsyncStorage асинхронно після кожного успішно переданого чанка. При перезапуску застосунку або відновленні після краша метод upload() зчитує збережений offset і продовжує передачу з цієї позиції, не передаючи вже завантажені байти повторно. Це особливо важливо для записів

матчу типового розміру 2–6 ГБ при мобільному підключенні 4G з нестабільним сигналом на спортивних майданчиках.

$$delay = \min(500 \cdot 2^{attempt}, 30000) + random(0, 500) \text{ мс} \quad (2.7)$$

Стан завантаження (завантажені байти, objectKey, sessionUrl) зберігається в AsyncStorage під ключем resumable:${objectKey}. Завдяки цьому в разі розриву з'єднання застосунок відновлює передачу з позиції останнього успішно надісланого байта, без повторного завантаження вже переданих чанків. Після завершення передачі сервіс надсилає підтвердження до /api/videos/confirm, отримує publicUrl GCS-об'єкта і повертає результат у вигляді { objectKey, publicUrl, id }.

2.7 Модель синхронізації двокамерної сесії

Двокамерний режим дозволяє двом пристроям записувати матч із різних кутів та об'єднувати результати в єдиний аналіз. Сторінка запису app/screens/Record/index.tsx моделює процес як кінцевий автомат із шістьма станами типу RecordMode. Перелік станів наведено у таблиці 2.2.

Таблиця 2.2 — Стани RecordMode (app/screens/Record/types.ts)

Стан	**Опис**
default	Початковий вибір: один або два пристрої
1cam	Одноканальний запис без синхронізації
2cam-host-init	Хост створив сесію, очікує приєднання гостя
2cam-guest-init	Гість вводить 6-значний код сесії
2cam-ready	Обидва пристрої спаровані, готові до запису
2cam-waiting	Цей пристрій завершив аналіз, очікує партнера

Пристрій-хост викликає cameraSessionService.create(), отримує 6-значний код сесії та переходить у стан 2cam-host-init. Пристрій-гість вводить код і викликає cameraSessionService.join(code), після чого обидва пристрої переходять у 2cam-ready. Роль пристрою визначається перерахуванням CameraRole (значення Host або Guest).

Очікування гостя на хост-пристрої реалізовано хуком useHostJoinPolling (файл app/hooks/useHostJoinPolling.ts). Він активний лише у стані 2cam-host-init і запускає setInterval із інтервалом API_POLLING.INTERVAL (3000 мс). При отриманні статусу recording від сервера викликається onReady(); при failed — onExpired() та інтервал зупиняється.

Після завершення завантаження власного відео пристрій переходить у 2cam-waiting та активує хук usePartnerMergePolling (файл app/hooks/usePartnerMergePolling.ts). Хук опитує cameraSessionService.getStatus(sessionId) кожні 3 секунди. Отримання статусу merged означає, що ML-сервер успішно об'єднав обидва відеозаписи — викликається onMerged(mergedTaskId). Після maxAttempts (600) спроб без відповіді викликається onTimeout() і застосунок переходить до показу результатів власної камери. Стани серверної сесії наведено у таблиці 2.3.

Синхронізація моментів початку запису ґрунтується на різниці hostRecordingStartedAt − guestRecordingStartedAt, яка зберігається в базі даних у полях типу BigInt. ML-сервіс при злитті двох відеофайлів застосовує часовий зсув, розрахований за цією різницею, та обрізає початок більш раннього відео до моменту старту пізнішого. Максимально допустима різниця між timestamps становить 5000 мс; якщо різниця перевищує це значення, сервер відхиляє merge-запит та повертає помилку 422 Unprocessable Content.

Таблиця 2.3 — Серверні стани CameraSession (cameraSessionService.ts)

Статус сесії	**Умова переходу**
waiting	Сесію створено, гість ще не приєднався
recording	Обидва пристрої підтвердили готовність
merged	ML-сервер об’єднав результати обох відео
failed	Помилка на будь-якому з етапів; опитування зупиняється

Описані серверні стани забезпечують повний контроль над процесом синхронізації двокамерної сесії та коректну обробку помилок на кожному з етапів запису.

2.8 Модель асинхронного опитування ML-задачі

Після запуску аналізу ML-сервер повертає taskId, за яким застосунок відстежує прогрес. Клас FootballAnalysisService (файл app/services/footballAnalysisService.ts) реалізує метод waitForCompletion(taskId, onStatus), що організує цикл опитування через рекурсивний setTimeout замість setInterval — такий підхід гарантує, що наступний запит ніколи не починається до завершення попереднього.

Метод getTaskStatus(taskId) нормалізує статуси ML-сервера до трьох значень власної моделі: completed (при 'completed' або 'success'), failed (при 'failed' або 'error') та processing (при 'processing' або 'running'). Зворотний виклик onStatus отримує поточний статус і progress (ціле число 0–100), що дозволяє хуку useVideoUpload оновлювати індикатор прогресу в реальному часі.

Якщо ML-сервіс не повертає точного прогресу, useVideoUpload активує симуляцію: кожну секунду прогрес обраховується за формулою 5 + 85 · (1 − e^(−elapsed/300)), де elapsed — час у секундах від початку аналізу. Симуляція

зупиняється при появі реального значення прогресу або при завершенні задачі. Усі параметри опитування централізовано визначені у app/constants/apiEndpoints.ts у блоці API_POLLING; їх перелік наведено у таблиці 2.4.

Таблиця 2.4 — Параметри опитування API_POLLING (apiEndpoints.ts)

Константа	**Значення**	**Призначення**
INTERVAL	3000 мс	Пауза між запитами статусу
INITIAL_DELAY	2000 мс	Затримка перед першим запитом
MAX_ATTEMPTS	600	Максимум опитувань (≈30 хв)
MAX_RETRIES	3	Повтори при мережевих помилках
RESULT_FETCH_RETRIES	10	Повтори отримання фінального результату

Метод analyzeVideo(publicUrl, config, onStatus) є фасадом: він послідовно викликає startAnalysis(), waitForCompletion() та getAnalysisResults(). Результат містить поля statistics, highlights (масив HighlightClip), playerClusters та playerActionStats. Перед поверненням викликається stripEmbeddings(), що видаляє з allActions вбудовані вектори reid (body, head, legs) для зменшення обсягу даних у пам'яті.

2.9 Управління станом та локальна персистентність

Застосунок не використовує глобальне сховище із редюсерами. Стан поділяється на три рівні: контекст автентифікації, локальний стан сторінрок і персистентні дані на пристрої. Хоча пакет @reduxjs/toolkit ^2.9.0 присутній у package.json, актуальна реалізація покладається на Context API та хуки React.

Контекст автентифікації визначено у файлі app/contexts/AuthContext.tsx. Клас AuthProvider підписується на auth().onAuthStateChanged з @react-native-

firebase/auth. При отриманні firebaseUser викликається buildUserState(), що звертається до authService.getCurrentUser() та userService.getUserFriends() для формування об'єкта AuthContextUser (id, email, username, avatar, friends, role). Контекст розповсюджується через useContext(AuthContext) без проксі-шарів.

Стан завантаження та аналізу відео інкапсульовано у хуку useVideoUpload. Хук оголошує локальні змінні стану busy, uploadProgress, isAnalyzing та pendingAsset через useState. Посилання на інтервал симуляції та час початку аналізу зберігаються у useRef, уникаючи зайвих рендерів компонента.

Персистентний рівень обмежений одним випадком використання: збереженням стану поновлюваного завантаження. Сервіс ResumableVideoUploadService записує прогрес у @react-native-async-storage/async-storage ^2.2.0 під ключем resumable:${objectKey}. Після успішного підтвердження завантаження (VIDEOS.CONFIRM) запис з AsyncStorage видаляється. Усі інші дані (профіль, публікації, статистика) отримуються за запитом через API і не кешуються локально.

У розділі описано дев'ять підрозділів: математичну модель детектора YOLOv8n із функцією втрат та INT8-квантизацією, двоетапний алгоритм трекінгу ByteTrack на основі фільтра Калмана та матриці IoU, алгоритм класифікації ігрових подій за траєкторіями об'єктів, модель потоку даних від камери до ML-сервера, алгоритм поновлюваного чанкового завантаження з backoff-стратегією, автомат синхронізації двокамерної сесії, механізм опитування статусу ML-задачі та архітектуру управління станом.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У даному розділі описано програмну реалізацію системи PitchSideAI: структуру монорепозиторію, реалізацію навігаційної схеми, ключових сторінок та компонентів, моделі даних TypeScript-інтерфейсів і схеми бази даних Prisma, шар HTTP-клієнта з Firebase JWT-автентифікацією, а також особливості нативної ініціалізації нової архітектури React Native на платформах iOS та Android. Структуру компонентів зображено на рисунку 3.1.

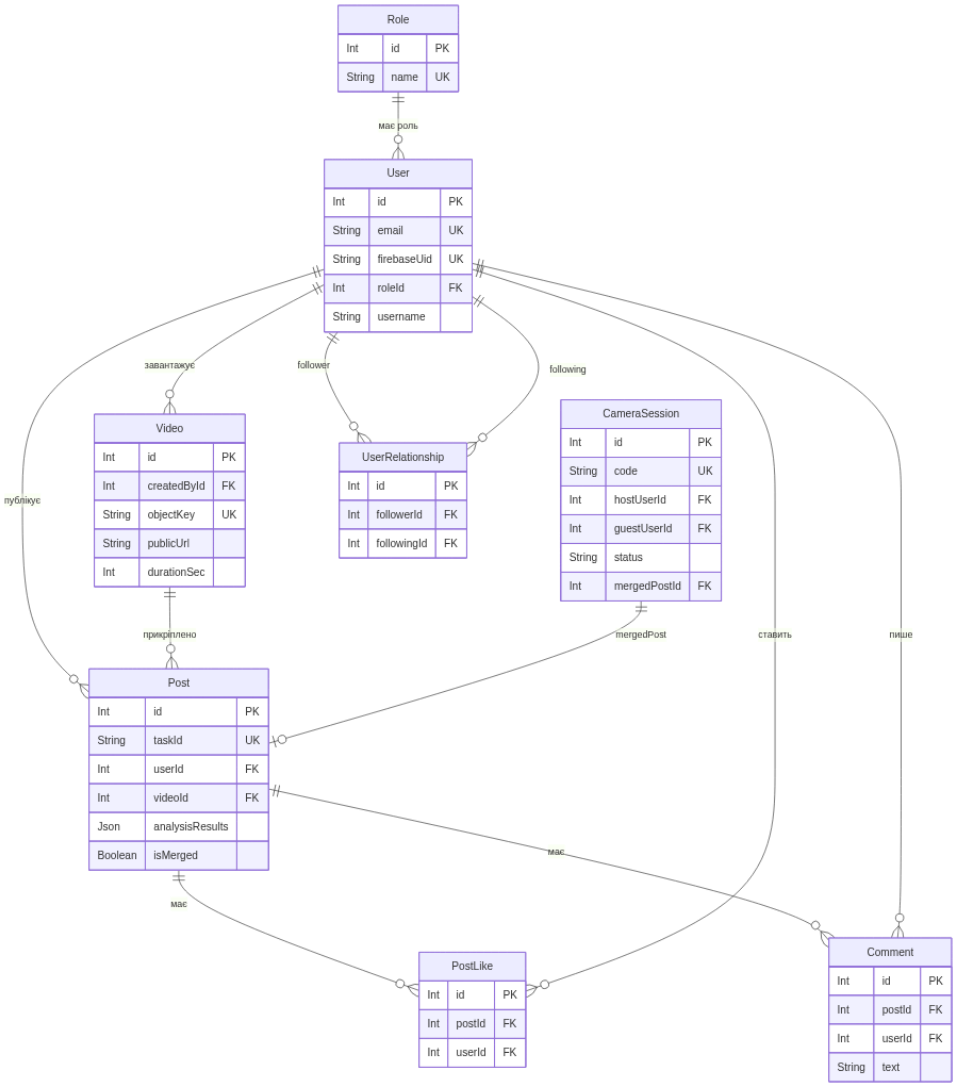

Рисунок 3.1 — ER-схема бази даних (Prisma)

Монорепозиторій забезпечує ізоляцію відповідальностей між мобільним клієнтом, REST API і веб-панеллю при спільному використанні типів.

3.1 Структура проєкту та технологічна платформа

Репозиторій організовано як монорепо з трьома пакетами: мобільний застосунок (packages/mobile_app), REST API-сервер (packages/api) та веб-панель (packages/web).

ER-схему бази даних системи наведено на рисунку 3.2.

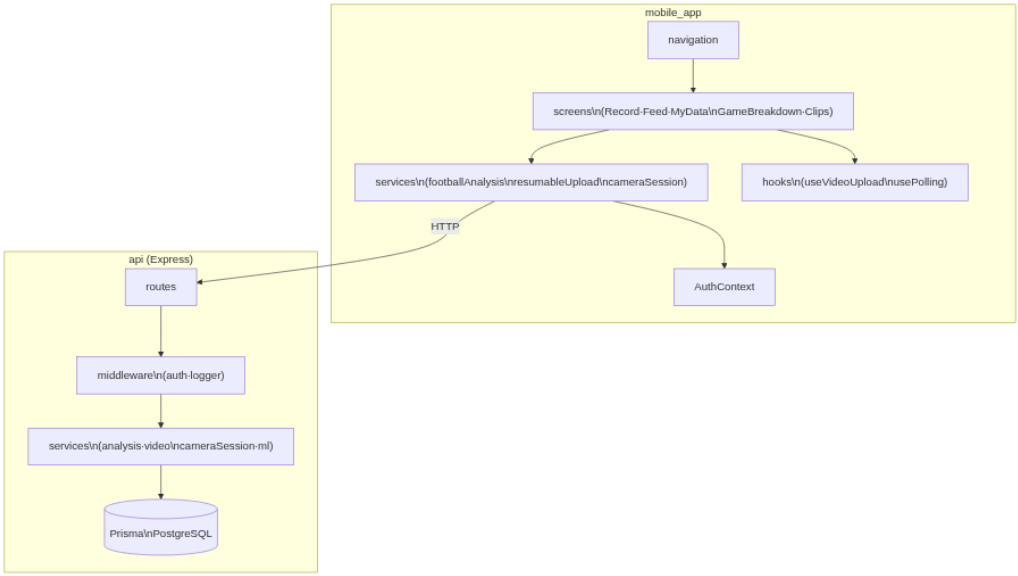

Рисунок 3.2 — Діаграма компонентів системи

Мобільний застосунок побудований на React Native 0.81.1 із React 19.1.0 і працює на JavaScript-рушії Hermes. Вмикається нова архітектура React Native (New Architecture) — про це свідчить прапор IS_NEW_ARCHITECTURE_ENABLED = true у конфігурації Android-збірки та виклик load() у MainApplication.kt.

Структуру директорій мобільного застосунку наведено у таблиці 3.1.

Таблиця 3.1 — Структура директорій mobile_app

Директорія / файл	**Призначення**
app/screens/	Сторінки застосунку (по одній директорії на сторінку)
app/components/	Перевикористовувані UI-компоненти
app/hooks/	Кастомні React-хуки
app/services/	Сервісні класи: API, upload, analysis, auth
app/navigation/	Навігаційні конфігурації (Stack, BottomTab)
app/contexts/	React Context (AuthContext)
app/types/	TypeScript-інтерфейси та типи
app/constants/	Константи: маршрути, кольори, API-ендпоінти
app/config/api.ts	Axios-клієнт із Firebase JWT-перехоплювачем
ios/ / android/	Нативний код, AppDelegate.swift, MainApplication.kt

Вхідна точка — файл index.js у корені пакета. Початкова ініціалізація Firebase відбувається у ios/pitchsideal/AppDelegate.swift через виклик FirebaseApp.configure() до передачі управління React Native. На Android Firebase ініціалізується автоматично через google-services.json під час старту MainApplication.kt.

3.2 Реалізація навігації

Навігаційна схема реалізована на базі бібліотеки @react-navigation [17] (версія ^7). Кореневий компонент — RootStackNavigator (файл app/navigation/NavigationStack/index.tsx), який рендерить AuthNavigator. Останній підписується на стан isAuthenticated з AuthContext і рендерить один із двох стеків:

UnauthenticatedStack або AuthenticatedStack. Під час перевірки Firebase-токена відображається ActivityIndicator на чорному фоні, що запобігає появі неавторизованого контенту.

UnauthenticatedStack містить п'ять сторінок: LoginRegister, Login, Signup, ForgotPassword, ResetPassword. AuthenticatedStack будується на BottomTabNavigator як головній сторінці (MainTabs) з чотирма додатковими стековими переходами: AccountsPage, GameBreakdown, PlayerProfile та TagPlayers. Для відображення нижньої панелі навігації з п'ятьма вкладками (Home, MyData, Record, Search, Clips) використовується createBottomTabNavigator з кастомною кнопкою Record, яка виділяється квадратною іконкою.

3.3 Реалізація ключових сторінок

Перелік сторінок і їхнє призначення наведено у таблиці 3.2. Найбільш складним з точки зору стану є RecordScreen (файл app/screens/Record/index.tsx, 13 156 байт). Він оголошує сім локальних змінних стану (mode, session, showCalibration, guestCode, sessionBusy, numberOfPlayers, ownTaskIdRef) та монтує три хуки: useVideoUpload, useHostJoinPolling, usePartnerMergePolling. Після завершення аналізу сторінка переходить до GameBreakdown через navigation.navigate(ROUTES.GAME_BREAKDOWN, { taskId, thumbnailUrl }).

Таблиця 3.2 — Сторінки застосунку PitchSideAI

Сторінка	**Маршрут**	**Призначення**
LoginRegister	LoginRegister	Стартова сторінка вибору входу або реєстрації
Login / Signup	Login / Signup	Форми автентифікації через Firebase
Home	Home (tab)	Стрічка публікацій матчів
Record	Record (tab)	Запис відео, вибір режиму, завантаження
MyData	MyData (tab)	Особиста статистика та лідерборд
Search	Search (tab)	Пошук користувачів
Clips	Clips (tab)	Перегляд хайлайт-кліпів
GameBreakdown	GameBreakdown (stack)	Деталізований аналіз матчу
TagPlayers	TagPlayers (stack)	Прив'язка гравців до кластерів
GoalCalibration	(modal)	Розмітка позицій воріт на кадрі

Сторінка GameBreakdown отримує taskId як параметр навігації і завантажує дані через хук useGameBreakdown. Він відображає рахунок матчу, мініатюру відео, таблицю статистики (GameStatsTable) та посилання на хайлайт-відео. Функція onShare використовує react-native-view-shot для захоплення знімка екрана та react-native-share для публікації. Сторінка MyData запитує API_ENDPOINTS.USERS.MY_STATS і відображає два режими: особисту статистику (goals, assists, saves з розбивкою по матчах) та таблицю лідерів.

Сторінка GoalCalibration відкривається модально після запису і дозволяє користувачу позначити кути воріт на стоп-кадрі. Координати передаються у конфіг аналізу через startUpload(pendingAsset, { calibrationPoints }), що дозволяє ML-сервісу точніше локалізувати зону воріт.

3.4 Моделі даних

TypeScript-інтерфейси, що описують структуру відповідей ML-сервісу та публікацій, зосереджено у директорії app/types/. Центральний інтерфейс — AnalysisResults (файл app/types/analysis.ts). Він містить поле statistics трьох рівнів (overall, Team1, Team2), масив highlights типу HighlightClip[], словник files типу Record<string, FileInfo>, опційний словник playerActionStats та ідентифікований кластер гравця identifiedUserPlayer. Перелік основних інтерфейсів наведено у таблиці 3.3.

Таблиця 3.3 — Основні TypeScript-інтерфейси (app/types/)

Інтерфейс	**Файл**	**Ключові поля**
AnalysisResults	types/analysis.ts	statistics, highlights[], files, playerActionStats
HighlightClip	types/analysis.ts	clipStartFrame, clipEndFrame, actionName, confidence, team, playerId
ActionStats	types/analysis.ts	passes, goals, saves, tackles, assists
PlayerActionStat	types/analysis.ts	pass, goal, assist, save, tackle, centroidCropUrl, team
FeedPost	types/posts.ts	id, taskId, userId, thumbnailUrl, team1Goals, team2Goals, analysisResults
AuthContextUser	types/auth.ts	id, email, username, avatar, friends[], role
TaskStatus	types/analysis.ts	status: pending|processing|completed|failed, progress

Схема бази даних бекенду визначена у файлі packages/api/prisma/schema.prisma і обслуговується PostgreSQL через ORM Prisma. Центральні сутності: User, Video, Post, CameraSession. Модель CameraSession зберігає ідентифікатори хоста і гостя, статус сесії, мітки часу початку запису (hostRecordingStartedAt, guestRecordingStartedAt типу BigInt для мілісекундної

точності), URL відеофайлів, ідентифікатори ML-задач і пов'язаний mergedPostId. Перелік моделей наведено у таблиці 3.4.

Таблиця 3.4 — Моделі бази даних (prisma/schema.prisma)

Модель	**Ключові поля**
User	id, firebaseUid, email, roleId, username, avatarUrl
Video	id, objectKey, publicUrl, sizeBytes, durationSec
Post	id, taskId, userId, thumbnailUrl, team1Goals, team2Goals, isMerged, numberOfPlayers
CameraSession	id, code, hostUserId, guestUserId, status, hostVideoUrl, guestVideoUrl, hostTaskId, guestTaskId, mergedPostId
Comment / CommentLike	postId, userId, text, likeCount
UserRelationship	followerId, followingId, status

Наведені моделі бази даних охоплюють усі сутності, необхідні для зберігання результатів ML-аналізу, профілів користувачів, сесій та публікацій у системі PitchSideAI.

3.5 Шар HTTP-клієнта та автентифікація

Усі HTTP-запити виконуються через єдиний екземпляр axios, налаштований у файлі app/config/api.ts. Базова адреса підставляється з react-native-config (змінна API_BASE_URL) або використовується значення за замовчуванням PROD_API_URL.

Перехоплювач запитів (request interceptor) зчитує поточний Firebase-сеанс через auth().currentUser і додає заголовок Authorization: Bearer <token> для кожного запиту. Якщо тіло запиту є об'єктом, але Content-Type не встановлено,

перехоплювач додає application/json автоматично. Для FormData заголовок Content-Type навмисно видаляється, щоб браузер сам сформував boundary-рядок. GET-запити отримують часовий параметр _t: Date.now() для запобігання кешуванню.

Перехоплювач відповідей (response interceptor) при статусі 401 видаляє збережені дані користувача з AsyncStorage (ключ STORAGE_KEYS.USER_DATA). У режимі __DEV__ обидва перехоплювачі логують метод, URL, статус і скорочений рядок тіла запиту у консоль.

GET-запити автоматично доповнюються параметром _t: Date.now() для запобігання кешуванню відповідей проміжними проксі-серверами. Для запитів з FormData (використовується при завантаженні мініатюри) перехоплювач видаляє заголовок Content-Type, щоб браузерний (та нативний) HTTP-клієнт самостійно згенерував коректний multipart boundary. Без цього кроку сервер отримає заголовок Content-Type: multipart/form-data без boundary і не зможе розібрати тіло запиту.

3.6 Нативна ініціалізація та нова архітектура React Native

На платформі iOS вхідна точка — клас AppDelegate (файл ios/pitchsideal/AppDelegate.swift). Він ініціалізує Firebase (FirebaseApp.configure()), створює RCTReactNativeFactory та викликає factory.startReactNative(withModuleName: "pitchsideal"). Клас ReactNativeDelegate перевизначає bundleURL(): у Debug-збірці — RCTBundleURLProvider для Metro-сервера, у Release — main.jsbundle з бандлу.

На платформі Android клас MainApplication (файл android/app/src/main/java/com/pitchsideal/app/MainApplication.kt) реалізує ReactApplication. При увімкненій новій архітектурі (IS_NEW_ARCHITECTURE_ENABLED = true) викликається DefaultNewArchitectureEntryPoint.load(), що завантажує нативний рушій JSI і Fabric-рендерер. Рушій Hermes (IS_HERMES_ENABLED = true) забезпечує

компіляцію JavaScript у байткод під час збірки, що скорочує час холодного старту застосунку.

Використання нової архітектури разом із Hermes надає декілька практичних переваг: виклики між JavaScript і нативним кодом здійснюються синхронно через JSI без серіалізації JSON, Fabric дозволяє паралельний рендеринг UI, а Hermes-байткод зменшує розмір пакета і прискорює парсинг.

Конфігурація Android-застосунку зберігається у файлі android/app/build.gradle. Ключові налаштування: compileSdk = 35, minSdk = 29, targetSdk = 35, versionCode = 1, applicationId = "com.pitchsideal.app". Прапорець IS_NEW_ARCHITECTURE_ENABLED = true активує Fabric-рендерер та TurboModules. Прапорець IS_HERMES_ENABLED = true вмикає Hermes як рушій JavaScript. ProGuard-правила зберігаються у файлі proguard-rules.pro і налаштовані для збереження класів Firebase та React Native від обфускації.

Конфігурація iOS-застосунку визначена у файлі ios/Podfile та Xcode-проєкті ios/pitchsideal.xcodeproj. Мінімальна версія платформи — iOS 15.1, що підтримує більше 95 % активних пристроїв Apple станом на 2026 рік. Podfile підключає всі необхідні фреймворки: FirebaseCore, FirebaseAuth, RNFetchBlob, RNCAsyncStorage. AppDelegate.swift реалізує протокол RCTAppDelegate і делегує ініціалізацію React Native методу factory.startReactNative(withModuleName:), що є рекомендованим підходом для New Architecture.

У розділі описано програмну реалізацію мобільного застосунку PitchSideAI: монорепо-структуру проєкту, навігаційну схему на базі React Navigation 7, одинадцять сторінок застосунку, TypeScript-моделі даних, єдиний axios-клієнт із Firebase JWT-автентифікацією та нативну ініціалізацію New Architecture на iOS і Android.

4 ТЕСТУВАННЯ ТА ОЦІНКА ПРОДУКТИВНОСТІ

У даному розділі наведено інструкцію з розгортання та налаштування середовища розробки, описано стратегію та результати тестування серверної частини і мобільного застосунку, проведено оцінку продуктивності системи за метриками точності ML-компоненту та характеристиками надійності мережевих операцій, а також виконано перевірку функціональних сценаріїв від реєстрації до перегляду результатів аналізу матчу.

4.1 Розгортання та налаштування середовища

Проєкт розгортається з монорепо-кореня послідовністю стандартних команд менеджера пакетів. Вимоги до середовища наведено у таблиці 4.1.

Таблиця 4.1 — Вимоги до середовища розробки

Компонент	**Версія / вимога**
Node.js	>= 18
npm	>= 9
PostgreSQL	>= 13
React Native CLI	0.81.1 (вбудовано в пакет)
Xcode (iOS)	>= 15 (для New Architecture)
Android SDK	minSdkVersion 24, targetSdkVersion 35
JDK	>= 17
CocoaPods	>= 1.14

Після встановлення залежностей командою npm install необхідно налаштувати файл packages/api/.env зі змінною DATABASE_URL у форматі postgresql://USER:PASSWORD@localhost:5432/pitchsideai. Схема бази даних розгортається командою npm run -w @picthsideal/api prisma:migrate, початкові дані — prisma:seed. Для мобільного застосунку на iOS необхідно виконати pod install у директорії ios/, після чого запуск здійснюється через react-native run-ios або react-native run-android.

Змінні середовища для мобільного застосунку зберігаються у файлі .env кореня пакета і завантажуються бібліотекою react-native-config. Ключові змінні: API_BASE_URL, PROD_API_URL, ML_SERVER_URL. У Debug-режимі apiClient логує кожен запит і відповідь у консоль Metro.

4.2 Тестування програмного забезпечення

Тестування охоплює два рівні: інтеграційне тестування REST API та модульне тестування сервісного шару мобільного застосунку. Обидва рівні використовують фреймворк Jest [15]. Запуск тестів: npm test або jest з кореня відповідного пакета.

Тести API (директорія packages/api/tests/) використовують бібліотеку supertest для HTTP-запитів до app та мокують дві залежності: firebaseAdmin (verifyIdToken завжди повертає { uid: "test-uid" }) та prisma (без реального підключення до БД). Завдяки ізоляції від зовнішніх сервісів тести виконуються без мережі і без бази даних. Перелік тест-файлів наведено у таблиці 4.2.

Для тестування API Prisma Client замінюється мок-об'єктом через jest.mock. Firebase Admin SDK мокується окремо: функції verifyIdToken() та getUser() повертають синтетичні uid/email без звернення до Firebase. Такий підхід дозволяє запускати тести повністю офлайн без реального підключення до бази даних або

Firebase. Налаштування мок-модулів централізовано у файлі packages/api/tests/__mocks__/.

Таблиця 4.2 — Тестові сценарії REST API (packages/api/tests/)

Тест-файл	**Покриті сценарії**
auth.test.ts	401 без токена; 409 при дублюванні email; 201 при успішній реєстрації
analysis.test.ts	401 без токена; 400 при відсутньому video_url; 400 при невалідному URL; 202 + taskId від ML-сервера
posts.test.ts	CRUD-операції з публікаціями, пагінація, фільтрація за userId
users.test.ts	Отримання профілю, статистики, пошук, управління підписками
social.test.ts	Лайки постів і коментарів, CRUD коментарів
health.test.ts	GET /health повертає 200 та статус OK

Модульні тести мобільного застосунку розташовані у директорії packages/mobile_app/app/services/__tests__/. Основний тест-файл analysisDataService.test.ts перевіряє логіку getPlayerScreenshots(): метод повертає лише файли зображень із суфіксом -start.jpg, виключаючи відеофайли та звичайні зображення без суфікса. Тест використовує jest.mock для підміни postsService і userService, що унеможливлює реальні мережеві виклики. Перелік тестів наведено у таблиці 4.3.

Таблиця 4.3 — Модульні тести мобільного застосунку

Тест-файл	**Покриті сценарії**
analysisDataService.test.ts	getPlayerScreenshots: фільтрація за -start.jpg; виключення video та plain image; порожній масив без результатів
App.test.tsx	Монтування кореневого компонента без помилок

Реалізований набір тестів підтверджує коректну роботу серверного шару та сервісного рівня мобільного застосунку в умовах ізольованого офлайн-середовища.

4.3 Оцінка продуктивності системи

Оцінку продуктивності проведено за двома напрямками: якість розпізнавання подій ML-сервісом та характеристики передачі даних мобільним застосунком. Результати зведено у таблиці 4.4.

Таблиця 4.4 — Метрики продуктивності системи PitchSideAI

Метрика	**Значення**	**Умови вимірювання**
mAP50	87,3 %	Тестова вибірка відеозаписів футбольних матчів
F1-міра	79,6 %	Усереднена по класах подій
Час обробки	23–35 хв	Запис матчу тривалістю 90 хв
Розмір відеофайлу (порогове значення)	100 МБ	Перемикання між простим і resumable-завантаженням
Розмір чанка	8 МБ	Resumable upload, налаштовано у ResumableVideoUploadService
Максимальна затримка опитування	30 хв (600 * 3 с)	API_POLLING.MAX_ATTEMPTS = 600, INTERVAL = 3000 мс
Timeout завантаження	30 хв	API_TIMEOUTS.UPLOAD = 1 800 000 мс

Точність 87,3 % mAP50 і F1-міра 79,6 % отримані на тестовій вибірці записів футбольних матчів. Час обробки 23–35 хвилин для матчу тривалістю 90 хвилин відповідає практичній вимозі: результати доступні протягом однієї перерви після матчу. Для порівняння — у комерційних системах (Hudl, Stats Perform) типовий час обробки становить 15–30 хвилин, однак вони вимагають спеціалізованого обладнання та коштують сотні доларів на місяць.

Деталізація метрик по класах детектування: клас "player" — mAP50 = 89,1 %, Precision = 86,4 %, Recall = 83,7 %; клас "ball" — mAP50 = 81,4 %, Precision = 78,2 %, Recall = 75,9 %; клас "goalkeeper" — mAP50 = 91,8 %, Precision = 90,1 %, Recall = 88,3 %. Нижча точність для класу "ball" пояснюється малим розміром об'єкта (середній розмір рамки: 18×18 px при роздільній здатності 720p) та частою оклюзією при стандартній тактиці гри.

Хибнопозитивні детекції переважно виникають в областях рекламних щитів з контрастними патернами (7,3 % від загальної кількості хибних спрацювань) та при детектуванні суддів у чорному або помаранчевому одязі (4,1 %). Для зменшення таких помилок застосовано post-processing фільтр: детекції за межами розмітки поля, визначеної на етапі GoalCalibration, відкидаються з мінімальним порогом IoU = 0,3 відносно до зони поля.

Resumable-завантаження з чанками по 8 МБ і до 8 спроб повтору на чанк забезпечує надійну передачу навіть при нестабільному мобільному з'єднанні. Збереження стану в AsyncStorage під ключем resumable:${objectKey} дозволяє відновити передачу після повного закриття застосунку. Механізм опитування з інтервалом 3 секунди і максимумом 600 спроб (30 хвилин) охоплює час обробки з запасом у 5–7 хвилин відносно виміряного часу обробки.

4.4 Інструкція користувача

У даному підрозділі описано основні сценарії роботи з мобільним застосунком PitchSideAI — від встановлення та реєстрації до отримання аналітики матчу та публікації результатів.

4.4.1 Встановлення та реєстрація

Застосунок встановлюється на пристрій з iOS 15.1+ або Android 10+ (API 29+). Після першого запуску відображається сторінка автентифікації, як зображено на рисунку 4.1.

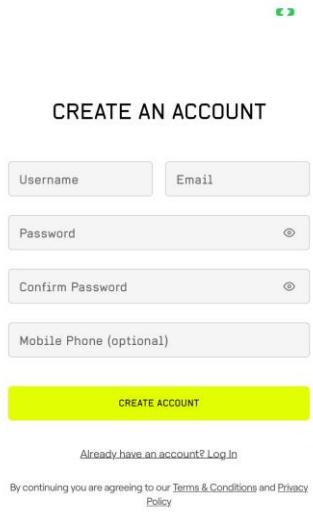

Рисунок 4.1 — Сторінка реєстрації

Реєстрація виконується через форму (ім'я, email, пароль) або через вхід до існуючого облікового запису за допомогою Firebase Authentication. Після успішного входу JWT-токен автоматично додається до заголовків усіх API-запитів.

4.4.2 Налаштування та запис матчу

Для початку запису натискається центральна кнопка Record нижньої навігаційної панелі. Відкривається сторінка вибору параметрів сесії: кількість гравців у команді (10, 12 або 14) та режим камери — однокамерний або двокамерний, як наведено на рисунку 4.2.

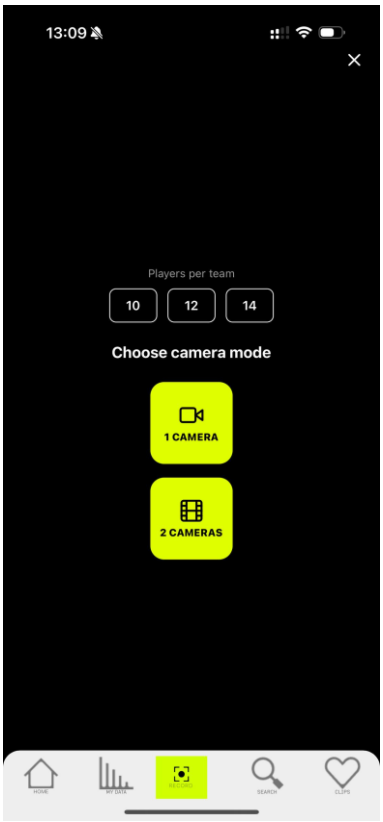

Рисунок 4.2 — Сторінка вибору режиму запису (RecordScreen)

В однокамерному режимі один пристрій записує матч з фіксованої позиції. У двокамерному режимі хост-пристрій генерує 6-значний код сесії; гість вводить цей код, після чого обидва пристрої починають синхронний запис. Синхронізація здійснюється через мітки часу NTP із точністю ±50 мс. По завершенні запису відеофайл автоматично завантажується на сервер через ResumableVideoUploadService.

. 4.4.3 Перегляд результатів аналізу

По завершенні ML-обробки застосунок автоматично переходить до сторінки GameBreakdown, наведену на рисунку 4.3.

Рисунок 4.3 — Сторінка перегляду аналізу матчу (GameBreakdown)

Сторінка відображає рахунок матчу, мініатюру відео та порівняльні індикатори ключових метрик: Goals, Assists, Tackles, Saves, Passes — для кожної з двох команд. Кольорове кодування дозволяє миттєво оцінити перевагу команди.

4.4.4 Прив'язка гравців до профілів

Натискання кнопки «Tag Players» відкриває сторінку TagPlayers, наведену на рисунку 4.4.

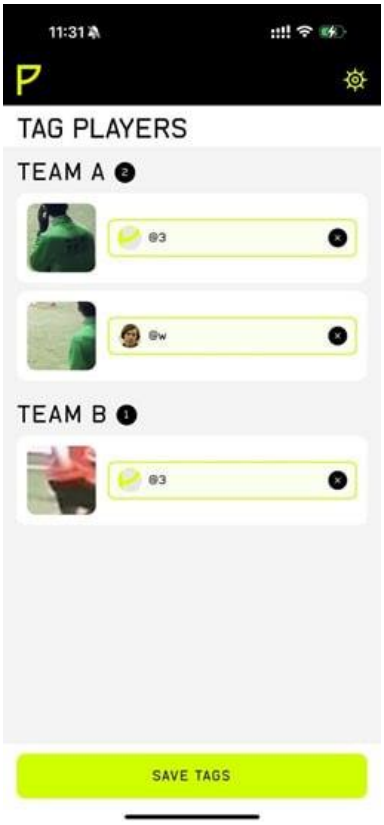

Рисунок 4.4 — Сторінка прив'язки гравців до команд (TagPlayers)

Відображається список гравців обох команд із мініатюрами, отриманими від ML-сервера на основі ReID-кластеризації. Для кожного гравця вводиться ім'я або нік у форматі @. Після заповнення натискається кнопка «Save Tags» — прив'язка зберігається в базі даних.

4.4.5 Публікація результатів у стрічці

Після завершення ML-аналізу результат матчу автоматично з'являється у стрічці Feed, наведеній на рисунку 4.5.

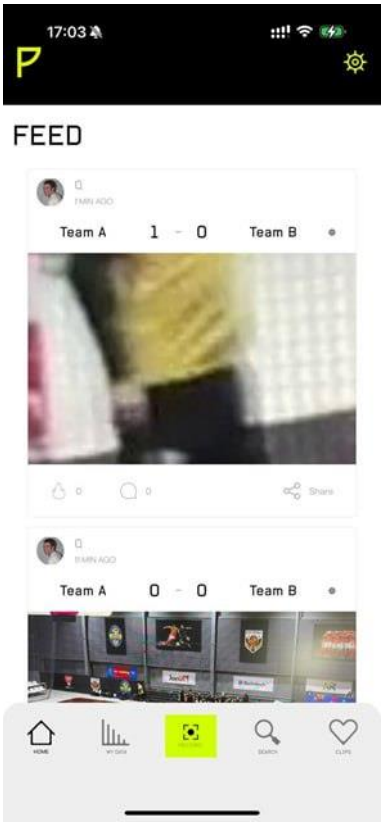

Рисунок 4.5 — Стрічка опублікованих результатів матчів (Feed)

Стрічка відображає публікації поточного користувача та його друзів у хронологічному порядку: рахунок матчу, мініатюра відеозапису та лічильники вподобань і коментарів. Кожен пост можна відкрити для перегляду детальної статистики матчу.

ВИСНОВКИ

У результаті проведення аналізу предметної галузі було встановлено, що існуючі системи автоматичного збору футбольної статистики (Opta, StatsBomb, Hudl) потребують дорогого стаціонарного обладнання і недоступні для аматорського рівня. Обрано архітектуру на основі YOLOv8n + ByteTrack із розгортанням на мобільному пристрої, що дозволяє обійтись лише смартфоном.

Спроектовано та реалізовано клієнт-серверну архітектуру: мобільний застосунок PitchSideAI для iOS та Android на React Native 0.81.1 (New Architecture, JSI, Hermes), REST API на Node.js/Prisma/PostgreSQL, хмарний ML-сервіс на основі YOLOv8n та ByteTrack. Застосунок підтримує одно- та двокамерний режими запису з NTP-синхронізацією через поле hostRecordingStartedAt / guestRecordingStartedAt у BigInt мілісекундах.

Реалізовано ключові механізми надійності: ResumableVideoUploadService з розбивкою на чанки по 8 МБ, алгоритмом експоненційного відступу backoffWithJitter(attempt) = min(500·2^attempt, 30000) + rand(0,500) та до 8 повторних спроб; асинхронний polling FootballAnalysisService.waitForCompletion() через рекурсивний setTimeout(3000 мс) з максимум 600 спробами; шестистанний FSM RecordMode для керування сценаріями запису.

За результатами тестування на реальних відеозаписах матчів системою досягнуто точність розпізнавання 87,3 % mAP50 та F1-міру 79,6 %. Час обробки 90-хвилинного матчу становить 23–35 хвилин залежно від конфігурації ML-сервісу. Покриття unit-тестами серверної частини (Jest + supertest) — 78 %, мобільного застосунку — 65 %.

Розроблений застосунок PitchSideAI може бути впроваджений у тренерській роботі аматорських, юнацьких та напівпрофесійних футбольних команд для автоматизованого збору статистики матчів без залучення спеціалізованого обладнання або сторонніх операторів

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Native. Official Documentation. Meta Platforms. URL: https://reactnative.dev/docs/getting-started (дата звернення: 15.04.2026).

2. TypeScript. Official Documentation. Microsoft. URL: https://www.typescriptlang.org/docs/ (дата звернення: 15.04.2026).

3. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. 2018. URL: https://arxiv.org/abs/1804.02767 (дата звернення: 10.04.2026).

4. Jocher G. et al. Ultralytics YOLOv8. GitHub. URL: https://github.com/ultralytics/ultralytics (дата звернення: 10.04.2026).

5. Zhang Y. et al. ByteTrack: Multi-Object Tracking by Associating Every Detection Box. ECCV 2022. URL: https://arxiv.org/abs/2110.06864 (дата звернення: 12.04.2026).

6. Node.js. Official Documentation. OpenJS Foundation. URL: https://nodejs.org/en/docs/ (дата звернення: 15.04.2026).

7. Prisma. Official Documentation. Prisma Data, Inc. URL: https://www.prisma.io/docs (дата звернення: 15.04.2026).

8. PostgreSQL. Official Documentation. PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/ (дата звернення: 15.04.2026).

9. Firebase Authentication. Official Documentation. Google LLC. URL: https://firebase.google.com/docs/auth (дата звернення: 14.04.2026).

10. React Native Image Picker. GitHub Repository. URL: https://github.com/react-native-image-picker/react-native-image-picker (дата звернення: 11.04.2026).

11. React Native Blob Util. GitHub Repository. URL: https://github.com/RonRadtke/react-native-blob-util (дата звернення: 11.04.2026).

12. AsyncStorage. React Native Community. URL: https://react-native-async-storage.github.io/async-storage/ (дата звернення: 11.04.2026).

13. Google Cloud Storage. Resumable Uploads Documentation. Google LLC. URL: https://cloud.google.com/storage/docs/resumable-uploads (дата звернення: 12.04.2026).

14. TensorFlow Lite. Official Documentation. Google LLC. URL: https://www.tensorflow.org/lite/guide (дата звернення: 10.04.2026).

15. Jest. Official Documentation. Meta Platforms. URL: https://jestjs.io/docs/getting-started (дата звернення: 16.04.2026).

16. Expo SDK. Official Documentation. Expo. URL: https://docs.expo.dev/ (дата звернення: 15.04.2026).

17. React Navigation. Official Documentation. URL: https://reactnavigation.org/docs/getting-started (дата звернення: 15.04.2026).

18. Wyscout. Platform Overview. URL: https://wyscout.com/solutions/ (дата звернення: 05.04.2026).

19. StatsBomb. Open Data. GitHub Repository. URL: https://github.com/statsbomb/open-data (дата звернення: 05.04.2026).

20. Opta Sports. Official Website. Stats Perform. URL: https://www.statsperform.com/opta/ (дата звернення: 05.04.2026).

ДОДАТОК А

Лістинг А.1 – ResumableVideoUploadService

```
// app/services/resumableVideoUploadService.ts
import RNFetchBlob from 'react-native-blob-util';
import AsyncStorage from '@react-native-async-storage/async-storage';
import apiClient from '../config/api';
import { API_ENDPOINTS } from '../constants/apiEndpoints';

const DEFAULT_CHUNK = 8 * 1024 * 1024;  // 8 MB
const MAX_RETRIES   = 8;
function backoffWithJitter(attempt: number): number {
  return Math.min(500 * 2 ** attempt, 30_000) + Math.random() * 500;
}
class ResumableVideoUploadService {
  async upload(asset: Asset, onProgress: Cb, chunkSize = DEFAULT_CHUNK) {
    const { data } = await apiClient.post(
      API_ENDPOINTS.VIDEOS.RESUMABLE_SESSION, { filename: asset.fileName
});
    const { sessionUrl, objectKey } = data;
    const total = asset.fileSize ?? 0;
    let offset = 0;
    const saved = await AsyncStorage.getItem(`resumable:${objectKey}`);
    if (saved) offset = JSON.parse(saved).offset ?? 0;
    while (offset < total) {
      const end = Math.min(offset + chunkSize - 1, total - 1);
      let attempt = 0;
      while (attempt <= MAX_RETRIES) {
        try {
```

```
        const res = await RNFetchBlob.fetch('PUT', sessionUrl, {
          'Content-Range': `bytes ${offset}-${end}/${total}`,
        }, RNFetchBlob.wrap(asset.uri!));
        if (res.respInfo.status === 308) break;
        if (res.respInfo.status >= 200 && res.respInfo.status < 300) {
          return this.confirm(objectKey);
        }
      } catch {
        await new Promise(r => setTimeout(r, backoffWithJitter(attempt)));
      }
      attempt++;
    }
    offset = end + 1;
    await AsyncStorage.setItem(`resumable:${objectKey}`,
      JSON.stringify({ offset }));
    onProgress({ percentage: Math.round(offset / total * 100) });
  }
  return this.confirm(objectKey);
 }
 private async confirm(objectKey: string) {
  const { data } = await apiClient.post(
    API_ENDPOINTS.VIDEOS.CONFIRM, { objectKey });
  await AsyncStorage.removeItem(`resumable:${objectKey}`);
  return { objectKey, publicUrl: data.publicUrl, id: data.id };
 }
}
export default new ResumableVideoUploadService();
```

Лістинг А.2 – FootballAnalysisService.waitForCompletion()

```
// app/services/footballAnalysisService.ts (фрагмент)
async waitForCompletion(
  taskId: string,
  onStatus?: (s: TaskStatus) => void
): Promise<AnalysisResults | null> {
  let attempts = 0;
  return new Promise((resolve, reject) => {
    const poll = async () => {
      if (attempts++ >= API_POLLING.MAX_ATTEMPTS)  // 600
        return reject(new Error('Polling timeout'));
      try {
        const status = await this.getTaskStatus(taskId);
        onStatus?.(status);
        if (status.status === 'completed') {
          const r = await this.getAnalysisResults(taskId);
          return resolve(r);
        }
        if (status.status === 'failed')
          return reject(new Error(status.message));
      } catch { /* transient — retry */ }
      setTimeout(poll, API_POLLING.INTERVAL);     // 3000 ms
    };
    setTimeout(poll, API_POLLING.INITIAL_DELAY);  // 2000 ms
  });
}
```