Національний лісотехнічний університет України
(повна назва вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: *Розроблення інформаційної системи футбольного клубу «Оріон» засобами React*

Виконав: студент 2 курсу групи КНС-21
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Бушко М.Т.
(прізвище та ініціали)

Керівники Дендюк М.В., Габа О.О.
(прізвище та ініціали)

Рецензент Мокрицька О.В
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України

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

(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувачка кафедри КН

Борецька І. Б.
"10" червня 2025 року

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

Бушко Маркіян Тарасович
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інформаційної системи футбольного клубу «Оріон» засобами *React*
керівники роботи *Габа О.О., асистент кафедри комп'ютерних наук, Дендюк М.В., кандидат технічних наук, доцент кафедри комп'ютерних наук*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 15 листопада 2024 року №С-882
2. Термін подання студентом роботи 10 червня 2025 року
3. Вихідні дані до роботи Постановка завдання та його формалізація. Алгоритм побудови інформаційних систем. Вихідні дані та прототипи схожих систем. Графічне представлення вхідних та вихідних даних. Література за тематикою роботи.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ.
Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне та математичне забезпечення.
Розділ 3. Програмне та технічне забезпечення.
Висновки.
Список використаних джерел.
Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді (підготовка матеріалів для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи бакалаврської дипломної роботи	Термін виконання етапів роботи	Примітка
1.	1. Огляд проблемної області та методів її вирішення. 2. Формування функціональних вимог та постановка завдань. 3. Оформлення розділу 1 пояснювальної записки.	18.11.2024р. 15.12.2025р.	Виконано
2.	1. Аналіз інформаційного забезпечення. 2. Вибір технологій реалізації. 3. Проєктування бази даних та ОО-моделі. 4. Оформлення розділу 2.	16.01.2025р. 15.03.2025р.	Виконано
3.	1. Реалізація клієнтської та серверної частини. 2. Налаштування бази даних та API. 3. Завантаження зображень, авторизація. 4. Оформлення розділу 3.	16.03.2025р. 15.05.2025р.	Виконано
4.	1. Завершення пояснювальної записки. 2. Перевірка, оформлення і подання на кафедру.	16.05.2025р. 10.06.2025р.	Виконано

Студент _____ **Бушко М.Т.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Габа О.О.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Дендюк М.В.**
(підпис) (прізвище та ініціали)

ТЕХНІЧНЕ ЗАВДАННЯ

Розроблення інформаційної системи футбольного клубу засобами React

1. Провести аналіз існуючих вебресурсів футбольних клубів, визначити переваги та недоліки підходів до подання інформації.
2. Розробити архітектуру вебзастосунку: визначити основні сторінки, навігацію та структуру даних.
3. Реалізувати інтерфейс користувача за допомогою React з урахуванням сучасних UX/UI принципів.
4. Реалізувати серверну частину за допомогою Node.js та Express з підключенням до бази даних MongoDB.
5. Реалізувати функціональність:
   • відображення новин клубу;
   • перегляд розкладу та результатів матчів;
   • перегляд турнірної таблиці;
   • перегляд складу команди з детальною інформацією про гравців.
6. Створити систему реєстрації та авторизації користувачів із розмежуванням прав доступу (користувач/адмін).
7. Реалізувати адмін-панель для керування контентом (додавання, редагування, видалення новин, матчів, гравців).
8. Забезпечити адаптивність дизайну для мобільних пристроїв, планшетів та десктопів.
9. Провести тестування коректної роботи всіх функцій вебзастосунку.

АНОТАЦІЇ

Пояснювальна записка до дипломної роботи бакалавра містить 80 сторінок пояснювальної записки, 22 рисунків, 6 таблиць, 1 додаток, 12 джерел.

У дипломній роботі розроблено адаптивний вебзастосунок для футбольної команди “Оріон”. Проведено аналіз сучасних вебрішень для підтримки спортивних клубів, досліджено потреби вболівальників у доступі до актуальної інформації про команду, склад, новини, матчі та турнірне становище. Для реалізації вебзастосунку використано стек технологій MERN (MongoDB, Express.js, React, Node.js). Реалізовано функціонал новинної стрічки, сторінки матчів, динамічної турнірної таблиці, перегляду складу команди. Передбачено можливість реєстрації, авторизації та персонального кабінету користувача. Адміністратор має змогу керувати контентом. Забезпечено адаптивну верстку для різних пристроїв та базову перевірку на безпечність запитів. Сайт протестовано на коректність роботи основних функцій.

Ключові слова: React, Node.js, MongoDB, Express, футбольна команда, авторизація, REST API, адаптивна верстка, MERN стек.

ABSTRACT

The explanatory note to the bachelor's thesis contains 80 pages of explanatory note, 22 figures, 6 tables, 1 appendence, and 12 references.

In the thesis, an adaptive web application for the football team "Orion" was developed. The analysis of existing web solutions for supporting sports clubs was carried out, and the needs of fans for access to up-to-date information about the team, squad, news, matches, and standings were studied. The web application was implemented using the MERN stack (MongoDB, Express.js, React, Node.js). The system includes a news feed, match page, dynamic tournament table, and team roster view. Features such as user registration, authentication, and a personal profile were added. Administrators are provided with tools for managing content. The layout is responsive across different screen sizes, and basic request validation is ensured. The system was tested for proper functioning.

Keywords: React, Node.js, MongoDB, Express, football team, authentication, REST API, responsive design, MERN stack.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних

ІС — інформаційна система

ПЗ — програмне забезпечення

ОС — операційна система

ПК — персональний комп’ютер

ІТ — інформаційні технології

ФК — футбольний клуб

DOM — Document Object Model (об'єктна модель документа)

UI — User Interface (інтерфейс користувача)

UX — User Experience (досвід користувача)

API — Application Programming Interface (інтерфейс прикладного програмування)

CMS — Content Management System (система керування вмістом)

HTTP — HyperText Transfer Protocol (протокол передавання гіпертексту)

JSON — JavaScript Object Notation (формат обміну даними)

CRUD — Create, Read, Update, Delete (операції з даними)

SPA — Single Page Application (односторінковий застосунок)

ВСТУП

Актуальність теми полягає у збільшені кількості футбольних команд в Україні, з кожним роком їх стає все більше й більше. Кожній з них необхідній соціальні мережі й сайти, щоб їхні вболівальники могли бути з командою якнайближче, знали всі останні новини про свою улюблену команду, гравців, які виступають за неї, турнірне становище й розклад матчів.

Об'єктом дослідження є процес розробки вебзастосунку для футбольної команди з використанням сучасних вебтехнологій.

Предметом дослідження є структура, функціональність та дизайн вебсайту футбольної команди, що забезпечує зручний доступ до інформації про її діяльність.

Метою роботи є розробка адаптивного, функціонального та візуально привабливого вебсайту, який дозволяє відображати новини, склад команди, розклад матчів і турнірну таблицю.

Практичне значення роботи полягає у створенні повноцінного та зручного у використанні вебпродукту, готового до реального впровадження у діяльність футбольної команди. Розроблений вебзастосунок дозволяє систематизувати подачу інформації про команду, автоматизувати процес публікації новин, оновлення складу, результатів матчів та турнірної таблиці. Це сприяє підвищенню поінформованості вболівальників, формуванню позитивного іміджу клубу, а також забезпечує сучасну онлайн-присутність, що є важливим чинником у розвитку та популяризації команди.

1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Аналіз сучасного стану цифрових рішень для футбольних клубів

У сучасному світі цифровізація активно проникає в усі сфери діяльності, зокрема і в спортивну галузь. Футбольні клуби дедалі частіше впроваджують інформаційні системи для підвищення ефективності комунікації з вболівальниками, управління адміністративною діяльністю, просування бренду клубу та ведення статистики матчів.

Багато європейських клубів мають офіційні сайти та мобільні додатки, де представлено інформацію про гравців, календар матчів, результати, новини клубу, відеоогляди, продаж квитків та атрибутики. В Україні також спостерігається поступовий перехід до таких рішень, однак цей процес ще не на всіх рівнях є достатньо розвиненим, особливо серед аматорських чи молодіжних команд.

Основні переваги впровадження вебсистем для футбольного клубу:

- централізація інформації про команду, гравців, матчі та турнірні таблиці;
- покращення взаємодії з уболівальниками;
- можливість інтеграції новин, фотогалерей, коментарів та продажу квитків;
- підвищення іміджу клубу через сучасний онлайн-простір.

Проте є і певні виклики:

- необхідність технічної підтримки та оновлень;
- потреба в адаптивному дизайні під різні пристрої;
- забезпечення безпеки користувацьких даних та захист від атак;
- потреба в адмініструванні контенту та дотриманні принципів UX/UI.

Крім офіційних сайтів, деякі футбольні клуби впроваджують **аналітичні платформи**, які дозволяють відстежувати індивідуальні показники гравців, переглядати статистику в реальному часі, створювати персоналізовані тренувальні плани. Такі системи найчастіше доступні для професійних команд, однак із розвитком відкритих технологій дедалі більше аматорських клубів отримують змогу застосовувати подібні рішення.

Ще одним важливим напрямком є **інтеграція соціальних мереж** у платформу клубу — що дозволяє автоматично поширювати контент, посилювати взаємодію з аудиторією та залучати нових фанатів.

Відтак, поєднання адміністративного, інформаційного та інтерактивного функціоналу в єдиній вебсистемі — це не лише тренд, а й **необхідність для розвитку сучасного футбольного клубу**, зокрема на аматорському рівні.

1.2. Аналіз технологій для створення інформаційних систем

Для реалізації сучасного, швидкого та масштабованого вебзастосунку обрано технологію React — одну з найпопулярніших бібліотек для створення інтерфейсів користувача [1, 2], що базується на мові програмування JavaScript [3**Помилка! Джерело посилання не знайдено.**]. React активно використовується як у великих комерційних проєктах, так і у внутрішніх корпоративних системах.

Переваги використання React:

- компонентний підхід, що сприяє повторному використанню коду;
- висока продуктивність завдяки віртуальному DOM;
- активна спільнота та велика кількість готових бібліотек (React Router, Redux, Material UI тощо);
- можливість інтеграції з бекендом (Node.js [4, 5], Express [6], MongoDB [7] тощо) та API;
- адаптивна верстка, що дозволяє працювати на різних пристроях.

До можливих недоліків React-проєктів належать:

- складність для початківців у підтримці стану додатку при масштабуванні;
- потреба у налаштуванні середовища (Webpack, Babel).

З боку серверної частини (бекенду) для збереження даних та обробки запитів доцільно використати **Node.js** разом із **Express.js** — це дозволяє побудувати масштабоване RESTful API [8, 9], яке легко взаємодіє з фронтендом на React. Як база даних буде використовуватись **MongoDB**, що добре підходить для зберігання документно-орієнтованих структур, таких як профілі гравців, матчі, новини тощо.

Завдяки такому технічному стеку (**React + Node.js + Express + MongoDB**) система буде:

- **швидкою та асинхронною**, що важливо для взаємодії в реальному часі;
- **гнучкою в розширенні функціоналу**;
- **зручною для командної розробки** з поділом фронтенд/бекенд логіки;
- **відкритою до інтеграції** з іншими сервісами (наприклад, Telegram-ботами, платіжними шлюзами або сторонніми API з футбольною статистикою).

Таким чином, React у поєднанні з Node.js та MongoDB є **оптимальним вибором для створення сучасної інформаційної системи футбольного клубу**, яка забезпечує зручність, продуктивність і масштабованість.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз вимог до інформаційного забезпечення вебзастосунку

Інформаційне забезпечення вебзастосунку футбольного клубу охоплює всі дані, необхідні для коректної роботи системи, взаємодії з користувачами та забезпечення функціональності.

Основні вимоги до інформаційного забезпечення:

- **Повнота даних** — система повинна містити актуальну інформацію про гравців, матчі, турніри, новини та інше.
- **Структурованість** — дані мають бути логічно організовані, розділені за категоріями (гравці, новини, матчі, таблиці).
- **Актуальність** — усі відомості мають регулярно оновлюватись, особливо в контексті новин, складу команди та розкладу ігор.
- **Цілісність** — уникнення дублювання, втрати або спотворення інформації.
- **Доступність** — дані мають бути швидко доступними користувачам, незалежно від пристрою чи типу підключення.
- **Безпека** — обмеження доступу до адміністративної частини сайту, захист персональних даних зареєстрованих користувачів.

Інформаційна модель включає такі категорії даних:

- профілі гравців (ПІБ, позиція, фото, статистика);
- календар та результати матчів;
- новини клубу;
- турнірна таблиця;
- фотографії з матчів;
- дані користувачів (тільки для зареєстрованих: ім’я, email, роль).

2.2. Вибір інструментальних засобів та технологій для розробки вебсайту

Для створення інформаційної системи футбольного клубу було використано сучасні вебтехнології, що дозволяють створити повноцінний багатофункціональний вебзастосунок із чітким розподілом клієнтської та серверної частини.

Клієнтська частина (Frontend):

- **React** — бібліотека для створення динамічних інтерфейсів, що дозволяє працювати з компонентами та оновлювати сторінку без перезавантаження.
- **React Router** — реалізація маршрутизації в межах одного SPA-додатку.
- **React Icons** — бібліотека, що надає набір іконок (у проєкті використовувалися іконки соцмереж, зокрема FaFacebook, FaTwitter, FaInstagram).
- **useState, useEffect, useParams, useNavigate** — хуки React, що використовуються для роботи зі станом, виконання побічних ефектів, отримання параметрів маршруту та навігації між сторінками.
- **Fetch API** — вбудований інтерфейс браузера для здійснення HTTP-запитів до серверної частини. Застосовувався для отримання даних про матчі, гравців, новини тощо.
- **CSS** — для оформлення сторінок застосовувався звичайний CSS, без використання фреймворків стилізації типу Tailwind чи Bootstrap.

Серверна частина (Backend):

- **Node.js** — середовище виконання JavaScript-коду на сервері.
- **Express.js** — мінімалістичний фреймворк для створення веб-серверів та API.
- **cors** — бібліотека для налаштування CORS (Cross-Origin Resource Sharing), яка забезпечує обмін даними між сервером та клієнтом.
- **dotenv** — дозволяє зберігати конфіденційні змінні середовища в .env файлі.
- **mongoose** — бібліотека для взаємодії з базою даних MongoDB, яка дозволяє працювати зі схемами та моделями [10].
- **bcrypt** — використовується для хешування паролів перед збереженням у базу даних.
- **jsonwebtoken (jwt)** — бібліотека для реалізації авторизації на основі токенів [11].
- **multer** — middleware для обробки multipart/form-data, який застосовується для завантаження файлів.
- **path** — модуль Node.js для роботи з файловими шляхами.

- **Власні middleware:** authMiddleware, adminMiddleware — для обробки прав доступу користувачів.

База даних:

- **MongoDB** — NoSQL-база даних, що зберігає дані у вигляді документів.
- **Mongoose** — ODM (Object Data Modeling) бібліотека для MongoDB, яка дозволяє працювати зі схемами для моделей, таких як News, Player, Team, Match, User.

Інструменти розробки:

- **Postman** — застосовувався для тестування REST API-запитів під час розробки.

Таким чином, обраний стек технологій дозволяє ефективно реалізувати повнофункціональну інформаційну систему для футбольного клубу із підтримкою ролей користувачів, новин, матчів, гравців та іншого контенту.

2.3. Проєктування бази даних

У процесі розробки інформаційної системи футбольного клубу було здійснено проєктування бази даних, яка дозволяє ефективно зберігати та обробляти ключову інформацію про користувачів, гравців, команди, матчі та новини. Для зберігання даних було обрано документно-орієнтовану базу **MongoDB**, що забезпечує гнучкість структури та зручну інтеграцію з Node.js за допомогою бібліотеки **Mongoose**.

Основні сутності бази даних:

1. **users** (Користувачі)
   - **firstName**: ім’я користувача
   - **secondName**: прізвище користувача
   - **email**: електронна пошта
   - **password**: хешований пароль
   - **role**: роль (наприклад, "admin", "user")
2. **players** (Гравці)
   - **name**: ім’я гравця
   - **position**: позиція на полі
   - **country**: країна гравця

- **photo**: шлях до зображення гравця

3. **teams** (Команди)
   - **name**: назва команди
   - **matches**: кількість матчів
   - **wins**: кількість перемог
   - **draws**: кількість нічиїх
   - **losses**: кількість поразок
   - **goalsFor**: кількість забитих голів
   - **goalsAgainst**: кількість пропущених голів
4. **matches** (Матчі)
   - **homeTeam**: об'єкт команди-господаря
   - **awayTeam**: об'єкт команди-гостя
   - **date**: дата проведення
   - **time**: час проведення
   - **result**: результат матчу
5. **News** (Новини)
   - **title**: заголовок новини
   - **content**: основний текст
   - **createdAt**: дата публікації
   - **image**: шлях до зображення

Проєктована структура бази даних є гнучкою, масштабованою та повністю відповідає вимогам сучасного вебзастосунку футбольного клубу. Кожна колекція відповідає логічній сутності в доменній області, а взаємозв’язки реалізовані через посилання або вкладені об'єкти в документах. Це дозволяє зручно оперувати інформацією як на фронтенді, так і на бекенді.

2.4. Побудова об’єктно-орієнтованої моделі

Для організації логіки взаємодії між об’єктами вебзастосунку було побудовано об’єктно-орієнтовану модель, яка відображає сутності предметної області футбольного клубу та взаємозв’язки між ними.

Основні об’єкти системи:

- **User** — представляє зареєстрованого користувача. Має властивості: ім’я, прізвище, email, пароль, роль (звичайний користувач або адміністратор). Можливі дії: реєстрація, авторизація, редагування профілю.
- **Player** — гравець футбольної команди. Містить інформацію про ім’я, позицію, країну походження, а також фотографію.
- **Team** — футбольна команда. Включає статистичну інформацію про матчі, перемоги, поразки, нічиї, забиті та пропущені голи. Є центральною сутністю для матчів.
- **Match** — модель матчу між двома командами. Має інформацію про дату, час, результат, а також об’єкти команд-господаря та команди-гостя.
- **News** — новинна публікація, що включає заголовок, зображення, зміст і дату створення. Відображається у відповідному розділі інтерфейсу користувача.

Взаємозв’язки між об’єктами:

- **Match** пов’язаний з **Team** як через домашню, так і гостьову команду (через зовнішні ключі).
- **User** має рольову модель доступу до функцій системи (перегляд, редагування контенту, додавання новин тощо).
- **Player** може бути асоційований з **Team**, якщо передбачити зв’язок у майбутньому для розширення функціональності.

Ця модель дає змогу гнучко масштабувати систему, додаючи нові ролі, сутності або типи взаємодії без порушення поточної архітектури. Використання Mongoose забезпечує об'єктно-документний підхід до роботи з базою даних MongoDB, що ідеально підходить для сучасних Node.js застосунків.

2.5. Архітектура вебзастосунку та структура API

Для реалізації вебзастосунку футбольного клубу було обрано **клієнт-серверну архітектуру**, де фронтенд і бекенд розділені та взаємодіють між собою через HTTP-запити.

Структура вебзастосунку:

Клієнтська частина (Frontend):

- Розроблена з використанням **React**.
- Відповідає за динамічне відображення сторінок, маршрутизацію та взаємодію з користувачем.
- Використовує **fetch** для надсилання запитів до бекенду.
- Компонентна структура дозволяє легко масштабувати інтерфейс.

Серверна частина (Backend):

- Реалізована на основі **Node.js** з використанням фреймворку **Express.js**.
- Працює з базою даних **MongoDB** через ORM-бібліотеку **Mongoose**.
- Підтримує обробку HTTP-запитів, аутентифікацію (через JWT), завантаження файлів (multer) та управління доступом до ресурсів (через middleware).

Основні маршрути API:

Таблиця 2.1 - Аутентифікація та авторизація

Метод	**Шлях**	**Опис**
POST	/api/register	Рєстрація нового користувача
POST	/api/login	Авторизація (видача JWT токена)

Таблиця 2.2 - Новини

Метод	**Шлях**	**Опис**
GET	/api/news	Отримання всіх новин
GET	/api/news/:id	Отримання однієї новини за ID
POST	/api/news	Створення новини (admin, з фото)
PUT	/api/news/:id	Оновлення новини (admin, з фото)
DELETE	/api/news/:id	Видалення новини (admin)
GET	/api/news/archive	Отримання новини з архіву(за датою)

Таблиця 2.3 - Гравці

Метод	**Шлях**	**Опис**
GET	/api/players	Отримання списку гравців
POST	/api/players	Додавання нового гравця (admin)

PUT	/api/players/:id	Редагування гравця (admin)

Продовження таблиці 2.3

Метод	**Шлях**	**Опис**
DELETE	/api/players/:id	Видалення гравця (admin)

Таблиця 2.4 - Турнірна таблиця

Метод	**Шлях**	**Опис**
GET	/api/table	Отримання таблиці чемпіонату

Таблиця 2.5 - Матчі

Метод	**Шлях**	**Опис**
GET	/matches	Отримання списку матчів за участю ФК «Оріон»

3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

У процесі створення інформаційної системи футбольного клубу «Оріон» було використано набір сучасних програмних інструментів, які забезпечують зручність, ефективність та масштабованість розробки. Засоби розробки можна умовно поділити на:

- інструменти для створення клієнтської частини (фронтенду),
- засоби для створення серверної частини (бекенду),
- допоміжні інструменти (бази даних, API-тестування тощо).

Фронтенд.

Фронтенд було реалізовано з використанням мови **JavaScript** та бібліотеки **React**. Додатково використовувався **React Router DOM** для маршрутизації між сторінками застосунку.

Основні бібліотеки та інструменти:

- **React** — бібліотека для побудови інтерфейсів користувача.
- **React Router DOM** — маршрутизація (перехід між сторінками SPA).
- **React Icons** — набір SVG-іконок, що дозволив швидко стилізувати інтерфейс.
- **Fetch API** — вбудований інструмент браузера для виконання HTTP-запитів до бекенду.
- **CSS (звичайний)** — використано для стилізації компонентів без фреймворків.

Ось, як виглядає вкладка з новинами (рис. 3.1):

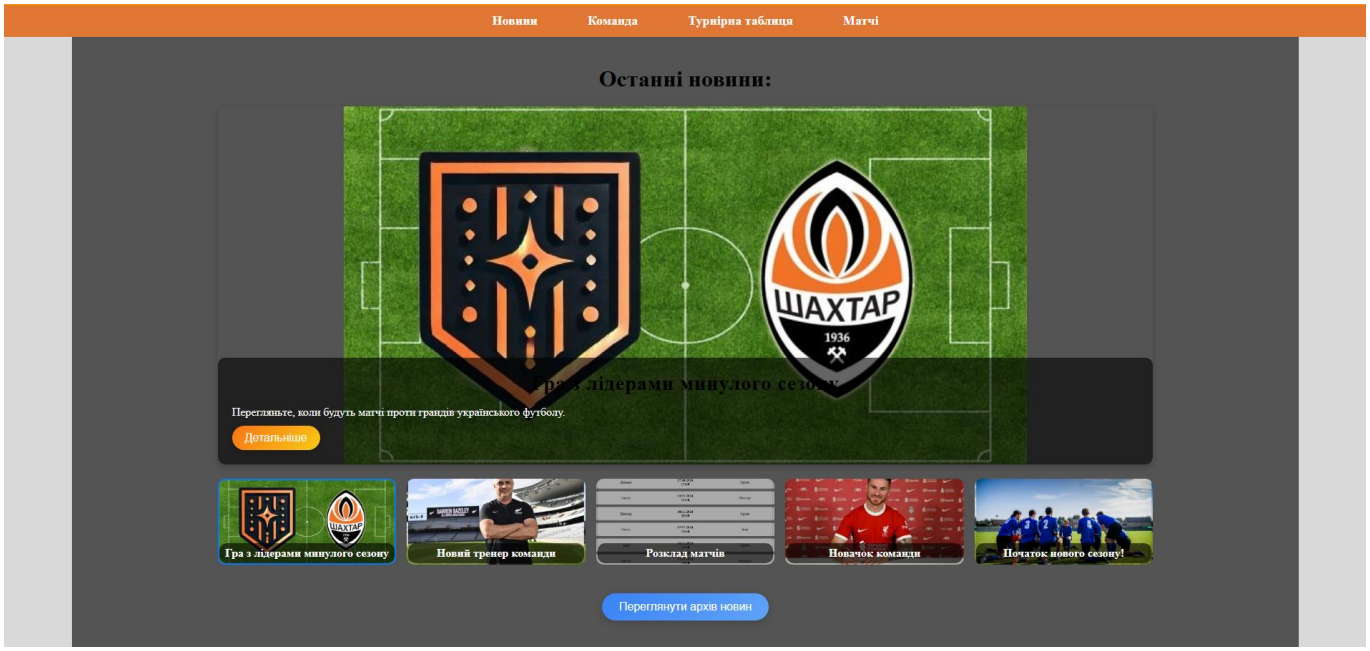

Рисунок 3.1 – Вкладка з новинами

Бекенд.

Серверну частину реалізовано на платформі **Node.js** з використанням фреймворку **Express.js**, що забезпечує створення REST API.

Основні бібліотеки та інструменти:

- **Express** — фреймворк для побудови серверних маршрутів та обробки запитів.
- **Mongoose** — бібліотека для взаємодії з MongoDB.
- **dotenv** — для зберігання конфіденційних змінних середовища в .env файлі.
- **cors** — забезпечує доступ до API з інших доменів (CORS-політика).
- **bcrypt** — використано для хешування паролів.
- **jsonwebtoken (JWT)** — для автентифікації та авторизації користувачів.
- **multer** — використовується для обробки завантажень файлів (зображень).
- **path** — вбудований модуль Node.js для роботи з файловими шляхами.

У проєкті також використовуються кастомні middleware:

- authMiddleware — для перевірки JWT токенів,
- adminMiddleware — для перевірки адміністративних прав користувачів.

База даних.

Для зберігання інформації про користувачів, гравців, команди, матчі та новини використовувалась **MongoDB** — документоорієнтована NoSQL база даних. Підключення до бази відбувалося за допомогою бібліотеки **Mongoose**.

Переваги MongoDB:

- гнучка структура даних (JSON-подібні документи),
- масштабованість,
- простота використання у веб-додатках.

Підключення виглядає ось так:

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("Підключено до бази:", mongoose.connection.name);
  } catch (err) {
    console.error("Помилка підключення:", err.message);
  }
};

module.exports = connectDB;
```

.env файл:

PORT=5000

JWT_SECRET=###*

MONGO_URI=###*

REACT_APP_API_URL=http://localhost:5000

EMAIL_SERVICE=gmail

MAIL_USER=###*

MAIL_PASS=###*

* реальні дані виглядають по-інакшому.

3.2. Вимоги до технічного забезпечення

Для коректної роботи інформаційної системи футбольного клубу визначено технічні вимоги як для кінцевого користувача, так і для розробника, який здійснює розробку, тестування та підтримку програмного продукту.

Мінімальні системні вимоги (для користувача):

- **Операційна система:** Windows 10 / Ubuntu 20.04 або новіша версія;
- **Процесор:** двоядерний (Intel Core i3 / AMD Ryzen 3);
- **Оперативна пам'ять:** 4 ГБ;
- **Місце на диску:** щонайменше 200 МБ для зберігання тимчасових файлів;
- **Інтернет-з'єднання:** необхідне для доступу до серверної частини системи;
- **Браузер:** Google Chrome, Mozilla Firefox або інший сучасний веб-браузер із підтримкою JavaScript.

Рекомендовані системні вимоги (для розробника):

- **Операційна система:** Windows 10/11 64-bit або Ubuntu 22.04 LTS;
- **Процесор:** чотириядерний (AMD Ryzen 5 / Intel Core i5 або кращий);
- **Оперативна пам'ять:** від 8 ГБ (оптимально — 16 ГБ);
- **Відеокарта:** дискретна графіка не є обов'язковою, але може покращити загальну продуктивність;
- **Накопичувач:** твердотільний накопичувач (SSD) об'ємом від 256 ГБ для зберігання вихідного коду, бібліотек, бази даних тощо. Додатковий жорсткий диск (HDD) може використовуватись для резервного копіювання;
- **Периферія:** клавіатура, миша, монітор з роздільною здатністю не менше 1920×1080.

Програмне забезпечення, необхідне для розробки:

- **Node.js** – для реалізації серверної частини проєкту;
- **MongoDB** – для зберігання структурованих даних;
- **MongoDB Compass** (опціонально) – для візуального перегляду та тестування бази даних;
- **Visual Studio Code** – як основне середовище розробки (IDE);
- **Google Chrome** – для перегляду результатів розробки;

- **Postman** (опціонально) – для тестування API-запитів;

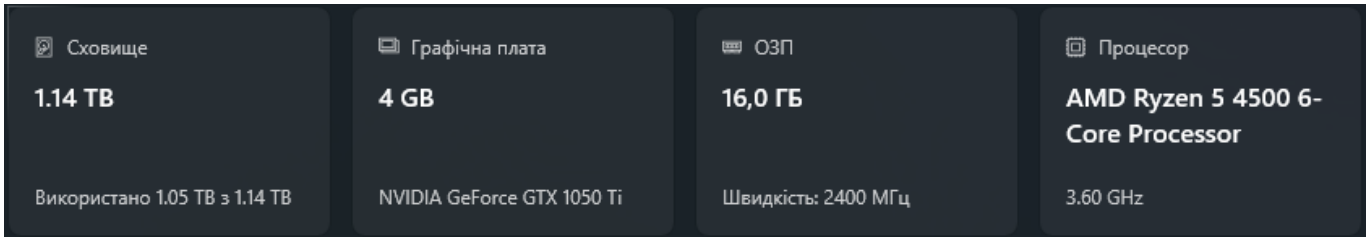

Рисунок 3.2 – ПК на якому проходив тест

3.3. Опис програмної реалізації

Цей підрозділ детально описує логіку роботи основних компонентів інформаційної системи ФК «Оріон». Програмна реалізація включає клієнтську частину (React), серверну частину (Node.js + Express), базу даних (MongoDB), а також механізми авторизації, завантаження зображень і контент-менеджменту.

1. Архітектура додатку

Інформаційна система реалізована за принципом **SPA (Single Page Application)**. Уся взаємодія користувача з інтерфейсом здійснюється без перезавантаження сторінки — динамічно підвантажуються лише необхідні дані з сервера. Це забезпечує високу швидкість роботи та кращий досвід користувача.

Взаємодія клієнт – сервер

Клієнтська частина надсилає запити до REST API, реалізованого на Express. Сервер опрацьовує запити, взаємодіє з базою даних MongoDB і повертає відповіді у форматі JSON. Дані надсилаються асинхронно за допомогою fetch.

Структура проєкту

Структура frontend (рис. 3.3):

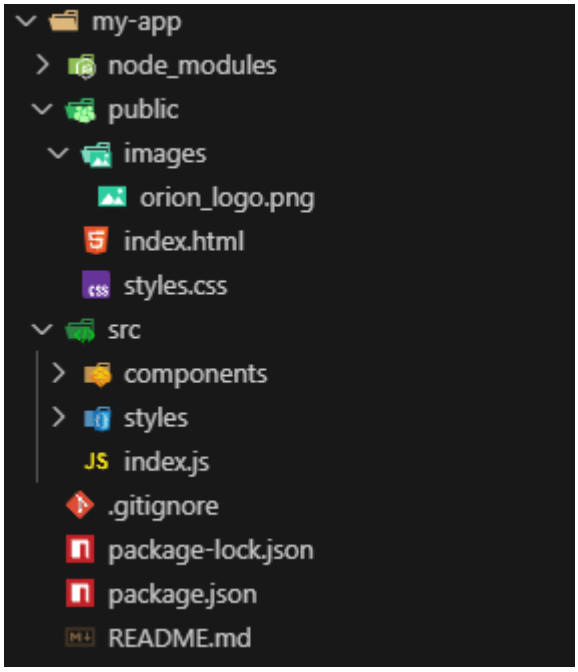

Рисунок 3.3 – Структура frontend

Структура backend (рис. 3.4):

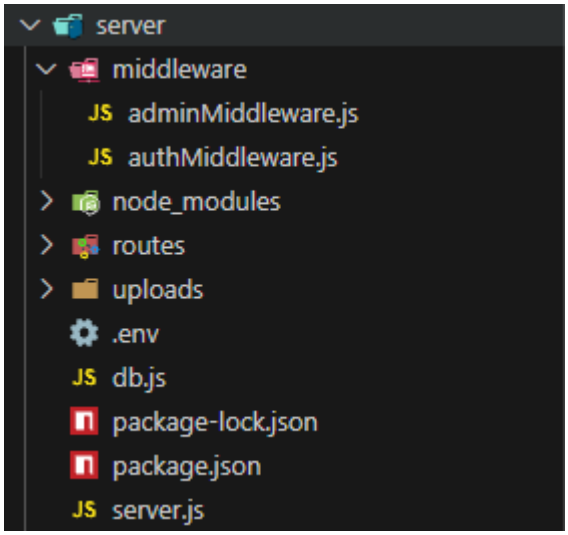

Рисунок 3.4 – Структура backend

2. Реалізація клієнтської частини

Клієнтська частина інформаційної системи реалізована за допомогою **React** — сучасної JavaScript-бібліотеки для побудови інтерфейсів користувача. Основна ідея — розбиття інтерфейсу на компоненти та динамічна робота з даними без перезавантаження сторінки.

Навігація за допомогою React Router

Для реалізації маршрутизації використано бібліотеку react-router-dom, що дозволяє створити SPA з кількома логічними сторінками: головна, новини, гравці, матчі, турнірна таблиця, реєстрація/авторизація тощо.

Приклад App.jsx:

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
# тут мають бути імпорти компонент

function App() {
  return (
    <Router>
      <Header />
      <Nav />
      <div id="content">
        <div id="main">
          <Routes>
            <Route path="/" element={<Main />} />
            <Route path="/news" element={<News />} />
            <Route path="/team" element={<Team />} />
            <Route path="/table" element={<Table />} />
            <Route path="/matches" element={<Matches />} />
            <Route path="/register" element={<Register />} />
            <Route path="/login" element={<Login />} />
            <Route path="/profile" element={<Profile />} />
            <Route path="/forgot-password" element={<ForgotPassword />} />
            <Route path="/reset-password/:token" element={<ResetPassword />} />
            <Route path="/news/:id" element={<NewsDetails />} />
            <Route path="/news/archive" element={<NewsArchive />} />
            <Route path="/news/create" element={<NewsForm />} />
            <Route path="/news/edit/:id" element={<NewsForm isEdit />} />
          </Routes>
        </div>
      </div>
      <Footer />
    </Router>
  );
}

export default App;
```

Робота з API та станом компонентів

Використовуються хуки useState та useEffect для керування станом і виконання запитів до сервера. Запити реалізовані через fetch.

Фрагмент із News.jsx:

```jsx
import { useEffect, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import "../styles/News.css";

const API_BASE_URL = "http://localhost:5000";

const News = () => {
  const [news, setNews] = useState([]);
  const [selectedNews, setSelectedNews] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    fetch(`${API_BASE_URL}/api/news`)
      .then((res) => res.json())
      .then((data) => {
        const sorted = data.sort(
          (a, b) => new Date(b.createdAt) - new Date(a.createdAt)
        );
        setNews(sorted);
        setSelectedNews(sorted[0]);
      })
      .catch((err) => console.error("Помилка завантаження новин:", err));
  }, []);

  const handleViewArchive = () => {
    navigate("/news/archive");
  };

  return (
    # html код
  );
};

export default News;
```

Відображення даних

Компоненти React автоматично оновлюють інтерфейс після отримання нових даних. Завдяки використанню JSX (суміш HTML + JS) реалізується зручний механізм виводу списків новин, гравців, матчів тощо.

Дані отримані з API (рис. 3.5):

Рисунок 3.5 – Список отриманих новин через API

3. Реалізація серверної частини

Серверна частина інформаційної системи ФК «Оріон» реалізована за допомогою платформи **Node.js** з використанням фреймворку **Express**. Основними задачами бекенду є обробка запитів з клієнтської частини, взаємодія з базою даних MongoDB, перевірка авторизації користувачів, а також забезпечення CRUD-функціоналу для новин, гравців, матчів і команд.

4. Структура REST API.

Сервер реалізує RESTful API, яке обробляє запити з фронтенду. Основні маршрути:

Таблиця 3.1 – Основні маршрути

Ресурс	HTTP	Опис
/api/news	GET	Отримати всі новини
/api/news	POST	Додати новину (admin)
/api/news/:id	PUT	Редагувати новину (admin)
/api/news/:id	DELETE	Видалити новину (admin)

Продовження таблиці 3.1

Ресурс	HTTP	Опис
/api/players	GET	Отримати список гравців
/api/players	POST	Додати гравця (admin)
/api/matches	GET	Отримати матчі
/api/matches	POST	Додати матч (admin)

Приклад CRUD-операції:

Додати новину(POST):

```
app.post('/api/news', authMiddleware, adminMiddleware, upload.single('image'), async
(req, res) => {
  try {
    const { title, content } = req.body;
    const image = req.file ? `/uploads/${req.file.filename}` : '';
    const news = new News({ title, content, image });
    const saved = await news.save();
    res.status(201).json(saved);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при створенні новини' });
  }
});
```

authMiddleware перевіряє, що користувач авторизований;

adminMiddleware переконується, що це адмін;

upload.single('image') — multer обробляє файл із поля image у форм;

Якщо файл наявний, то шлях до нього зберігається у полі image.

Об’єкт **news** створюється та зберігається в базу даних.

Якщо все добре — повертається 201 Created.

Ось, як це виглядає на фронтенді:

Для початку потрібно увійти в аккаунт адміна (про адмінів й простих користувачів буде дещо пізніше) через форму для входу в аккаунт(рис. 3.6):

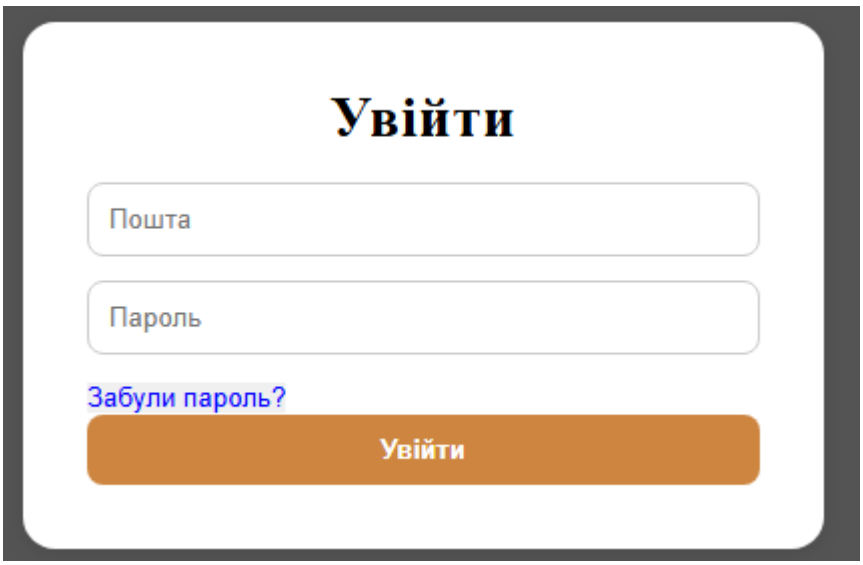

Рисунок 3.6 – Вхід в аккаунт

В разі неправильного вводу пошти чи пароль буде виведена помилка «Користувача не знайдено». Якщо все ж буде введено вірно, то користувача зразу перекине на вкладку з новинами й ще він зможе переглянути свій профіль, як показано на рис. 3.7.

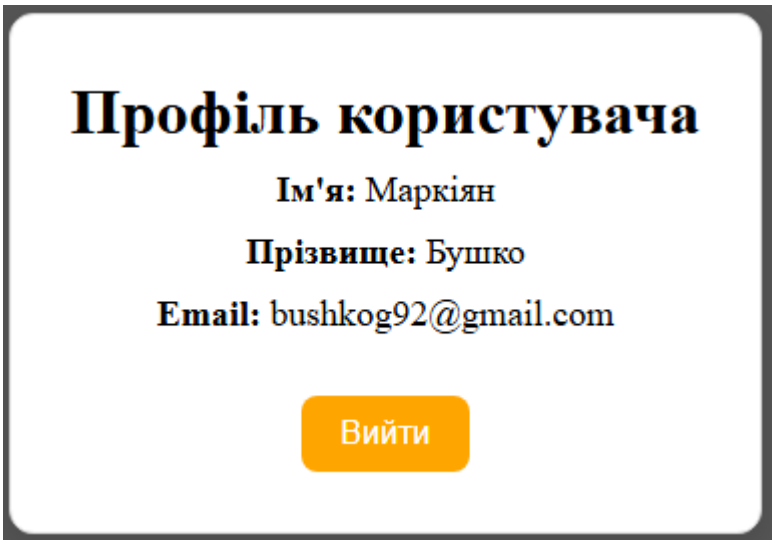

Рисунок 3.7 – Профіль користувача

Якщо у вас роль адміна, то на вкладці з новинами буде кнопка «Додати новину», після натискання на не адміна перекине на вкладку з формою для створення новини(рис. 3.8):

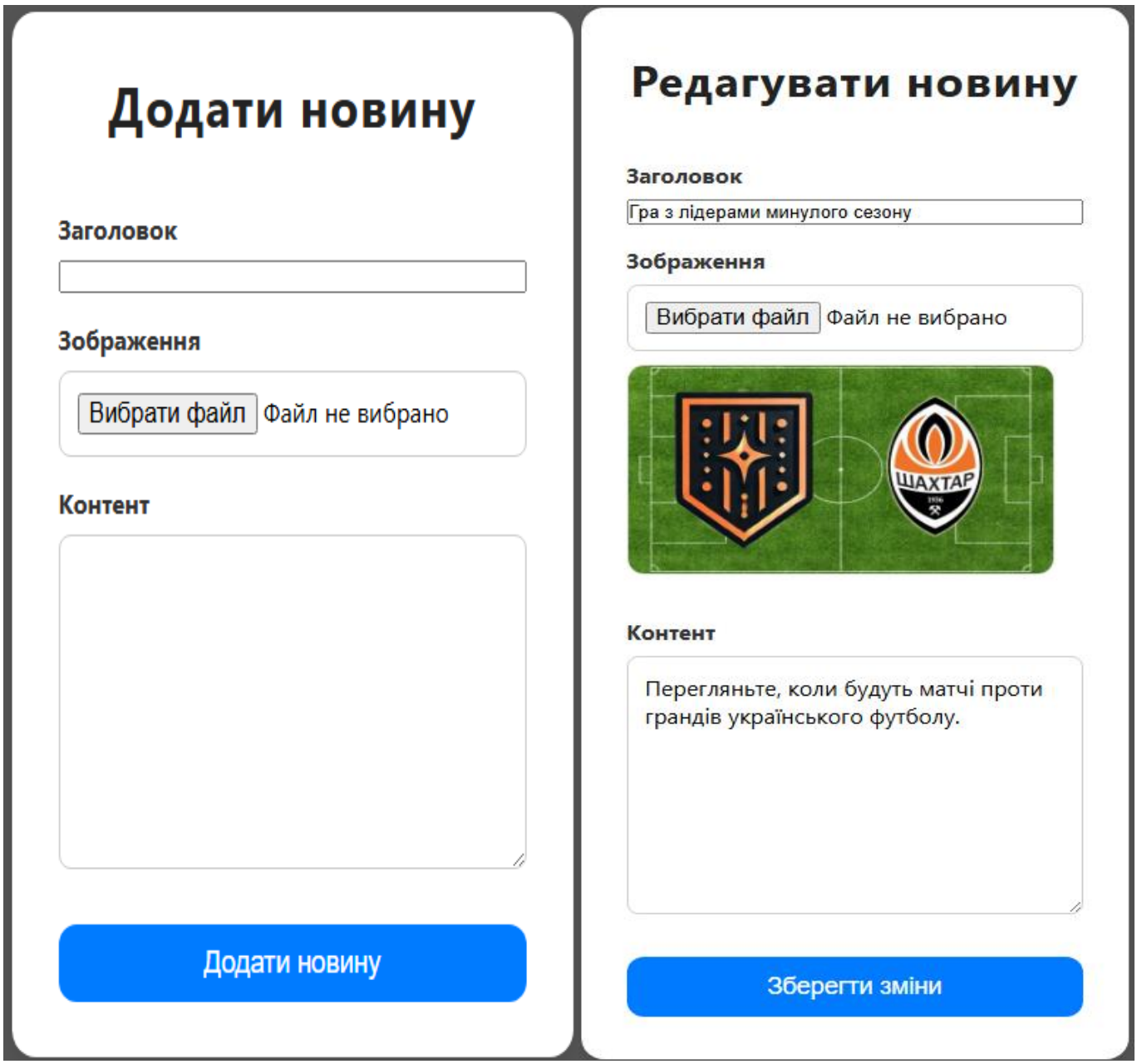

Рисунок 3.8 – Додати/редагувати новину

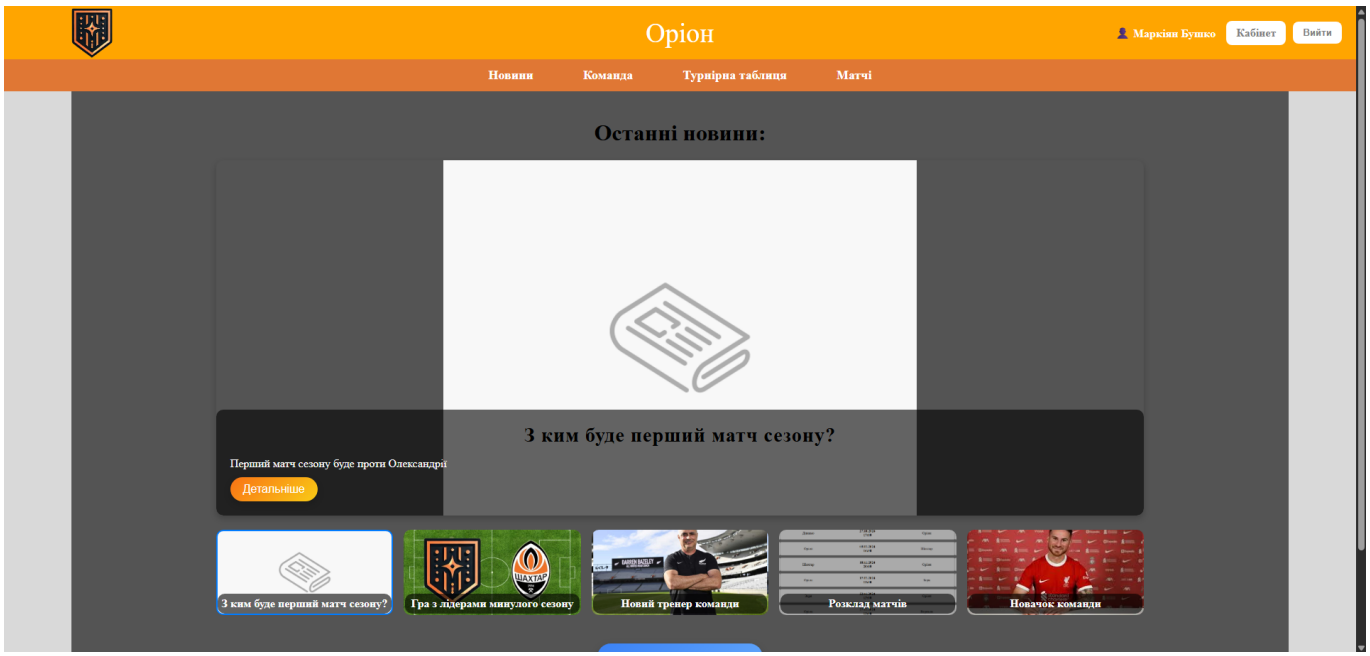

Рисунок 3.9 – Добавлена новина на фронтенді

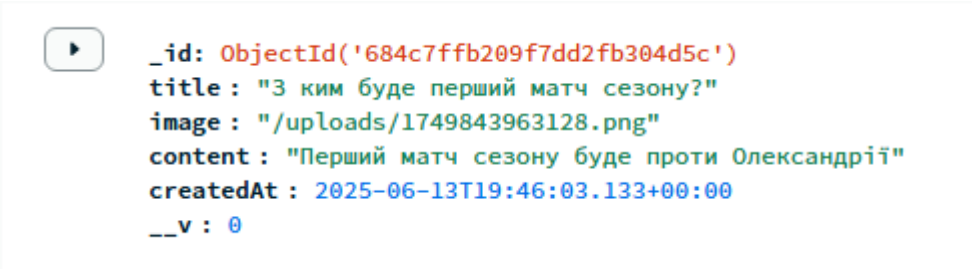

Рисунок 3.10 – Добавлена новина в БД

Отримати гравців(GET):

Для цього використаю Postman, як показано на рис. 3.11.

Рисунок 3.11 – Запит GET у Postman

Редагувати гравця(PUT)(може тільки адмін), як показано на рис. 3.13.**:**

Рисунок 3.12 – Картка гравця команди до/після редагування

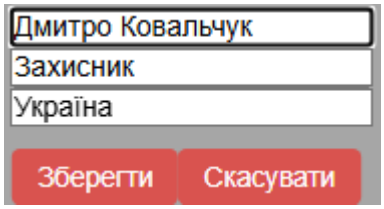

Рисунок 3.13 – Меню редагування гравця

Видалення новини(DELETE) (тільки адмін)**:**

Видалити новину можна на сторінці з архівом усіх новин, там же ж можна й редагувати й добавляти новини, як показано на рис. 3.14.:

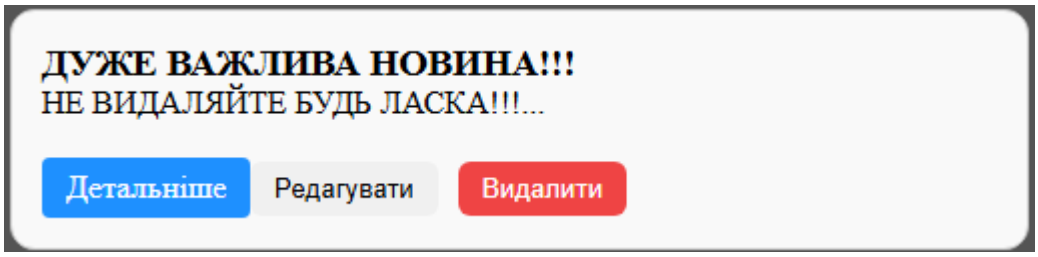

Рисунок 3.14 – Карточка новини в архіві

При натисканні на кнопку «Видалити» сайт спитає, чи впевнені ви в цьому, як показано на рис. 3.15.:

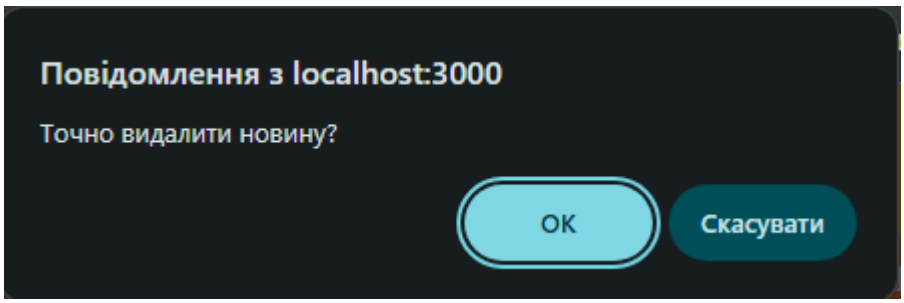

Рисунок 3.15 – Alert для підтвердження дії

Після підтвердження новина зникає як з фронту, так і з БД.

5. Middleware для авторизації.

authMiddleware.js

Забезпечує перевірку JWT-токена. Якщо токен валідний — користувач допускається до маршруту.

```
const jwt = require('jsonwebtoken');
const User = require('../../models/User');

module.exports = async function (req, res, next) {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Немає токена' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId).select('-password');
    if (!user) return res.status(401).json({ message: 'Користувача не знайдено' });

    req.user = user; // містить role
    next();
  } catch (err) {
    return res.status(403).json({ message: 'Недійсний токен' });
  }
};
```

adminMiddleware.js

Дозволяє доступ лише адміністраторам.

```
module.exports = function (req, res, next) {
  if (req.user && req.user.role === 'admin') {
    next();
  } else {
    return res.status(403).json({ message: 'Доступ заборонено: тільки для адміністратора'
});
```

```
  }
};
```

6. Завантаження зображень: Multer.

Для завантаження фото гравців або новин використовується multer.

```
const multer = require('multer');
const path = require('path');

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  },
});

const upload = multer({ storage });
```

7. Робота з БД.

У системі ФК «Оріон» зберігання даних реалізоване з використанням **MongoDB** — гнучкої документно-орієнтованої NoSQL бази даних. Для зручної взаємодії з нею на стороні Node.js використовується ODM-бібліотека **Mongoose**.

База даних містить основні сутності: **News**, **Players**, **Matches**, **Teams** та **Users**. Кожна з них має свою схему, яка описує структуру документа в колекції.

Основні моделі Mongoose:

News.js

```
const mongoose = require('mongoose');

const newsSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  image: String,
  content: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
```

```
});

module.exports = mongoose.model('News', newsSchema);
```

Player.js

```
const mongoose = require('mongoose');

const playerSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  position: {
    type: String,
    required: true
  },
  country: {
    type: String,
    required: true
  },
  photo: {
    type: String,
    default: ''
  },
}, {timestamps: true});

module.exports = mongoose.model('Player', playerSchema);
```

Match.js

```
const mongoose = require('mongoose');

const matchSchema = new mongoose.Schema({
  date: Date,
  time: String,
  homeTeam: { type: mongoose.Schema.Types.ObjectId, ref: 'Team', required: true },
  awayTeam: { type: mongoose.Schema.Types.ObjectId, ref: 'Team', required: true },
  result: {
    homeGoals: { type: Number, default: null },
    awayGoals: { type: Number, default: null }
  }
});

module.exports = mongoose.model('Match', matchSchema);
```

Team.js

```
const mongoose = require('mongoose');

const teamSchema = new mongoose.Schema({
  name: { type: String, required: true, unique: true },
  matches: { type: Number, default: 0 },
  wins: { type: Number, default: 0 },
```

```
  draws: { type: Number, default: 0 },
  losses: { type: Number, default: 0 },
  goalsFor: { type: Number, default: 0 },
  goalsAgainst: { type: Number, default: 0 }
});

module.exports = mongoose.model('Team', teamSchema);
```

User.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  firstName: String,
  secondName: String,
  password: { type: String, required: true },
  role: { type: String, enum: ['admin', 'user'], default: 'user' },
  resetToken: String,
  resetTokenExpiry: Date
});

module.exports = mongoose.model('User', userSchema);
```

Список таблиць в MongoDB Atlas (рис. 3.16):

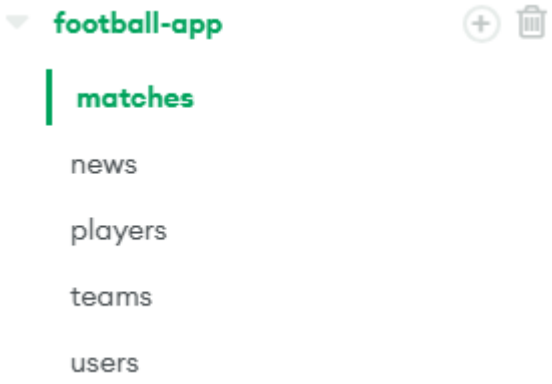

Рисунок 3.16 – Список таблиць

Приклад таблиці в MongoDB Atlas (рис. 3.17):

Рисунок 3.17 – Таблиця гравців команди

8. Авторизація та реєстрація

Для забезпечення доступу користувачів до персоналізованих функцій інформаційної системи футбольного клубу реалізовано механізм реєстрації та авторизації з використанням хешування паролів (bcrypt) і токенів (JWT).

Реєстрація користувача.

Для реєстрації користувач має вказати свою електронну пошту, ім’я, прізвище й ввести пароль два рази, як це показано на рис. 3.18.

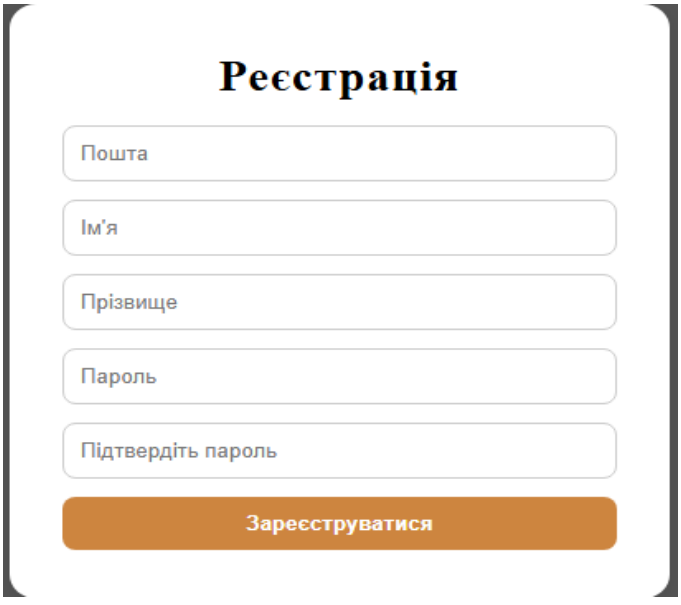

Рисунок 3.18 – Реєстрація

При реєстрації система перевіряє, чи існує вже користувач з вказаною електронною адресою. Якщо ні — створюється новий обліковий запис з хешованим паролем.

```
app.post('/api/register', async (req, res) => {
  const { email, firstName, secondName, password } = req.body;
  try {
    console.log("Реєстрація", req.body);
    const existing = await User.findOne({ email });
    if (existing) {
      return res.status(400).json({ message: "Користувач уже існує" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ email, firstName, secondName, password: hashedPassword });
    const saved = await newUser.save();
    console.log("Збережено користувача:", saved);

    res.status(201).json({ message: "Користувач створений", user: { email } });
  } catch (err) {
    console.error("Помилка при реєстрації:", err);
    res.status(500).json({ message: "Помилка сервера", error: err.message });
  }
});
```

Авторизація користувача (вхід).

Для входу в аккаунт користувач має вказати елетронну пошту й ввести пароль, як показано на рис. 3.19.

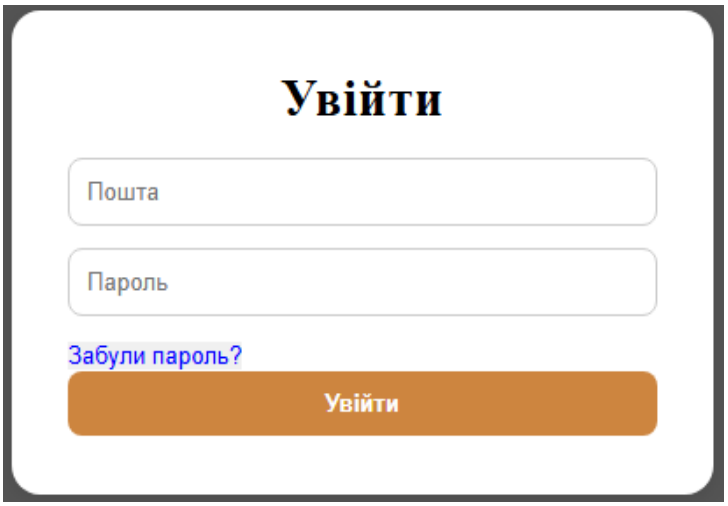

Рисунок 3.19 – Вхід в аккаунт

Після введення email та пароля система перевіряє їхню відповідність, а у разі успіху — видає JWT токен, який використовуватиметься для доступу до захищених маршрутів:

```
app.post('/api/login', async (req, res) => {
```

```
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: "Користувача не знайдено" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(401).json({ message: "Невірний пароль" });

    // Генерація токена
    const token = jwt.sign(
      { userId: user._id, email: user.email },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );


    res.status(200).json({
      message: "Успішний вхід",
      token,
      user: {
        firstName: user.firstName,
        secondName: user.secondName,
        email: user.email,
        role: user.role
      }
    });
  } catch (err) {
    res.status(500).json({ error: "Помилка авторизації" });
  }
});
```

JWT-токен

Токен зберігає ідентифікатор користувача та його роль, і використовується на клієнтській частині для подальшої авторизації запитів до захищених маршрутів.

Відновлення паролю.

У системі реалізовано механізм відновлення паролю через електронну пошту. У разі втрати доступу користувач може запросити листа для скидання пароля, в якому буде унікальне посилання для встановлення нового пароля.

```
router.post("/forgot-password", async (req, res) => {
  const { email } = req.body;

  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(404).json({ message: "Користувача не знайдено" });
```

```
    const token = crypto.randomBytes(32).toString("hex");
    user.resetToken = token;
    user.resetTokenExpiry = Date.now() + 3600000; // 1 година
    await user.save();

    const resetLink = `http://localhost:3000/reset-password/${token}`;

    // Надсилання email
    const transporter = nodemailer.createTransport({
      service: "gmail",
      auth: {
        user: process.env.MAIL_USER,
        pass: process.env.MAIL_PASS,
      },
    });

    await transporter.sendMail({
      to: user.email,
      subject: "Відновлення паролю",
      html: `
        <h2>Відновлення паролю</h2>
        <p>Натисніть на посилання нижче, щоб відновити пароль:</p>
        <a href="${resetLink}">${resetLink}</a>
      `,
    });

    res.json({ message: "Інструкції для відновлення паролю надіслано на пошту" });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error: err.message });
  }
});
```

При успішному запиті генерується токен (crypto.randomBytes(32)) і відправляється email з посиланням на сторінку скидання паролю.

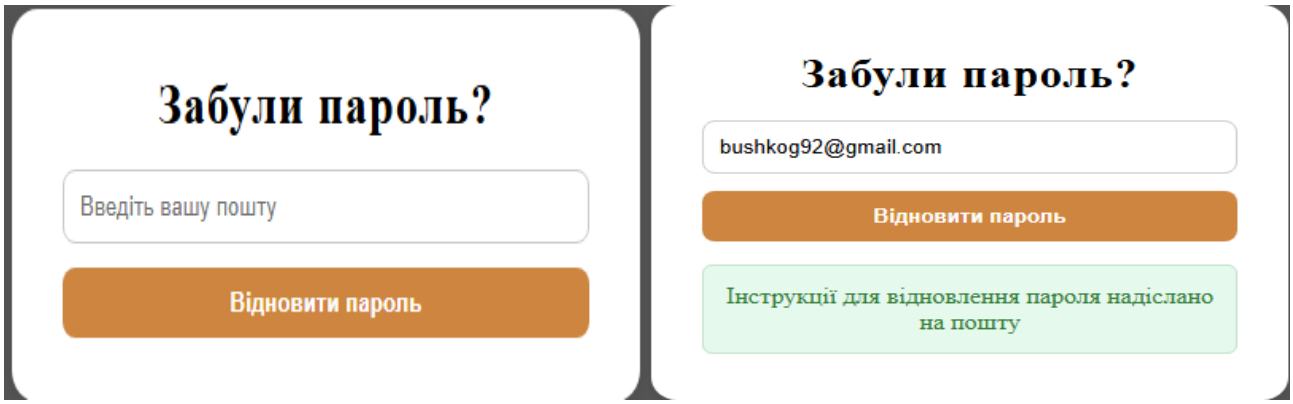

Рисунок 3.20 – Форма для відправки листа на пошту для відновлення паролю

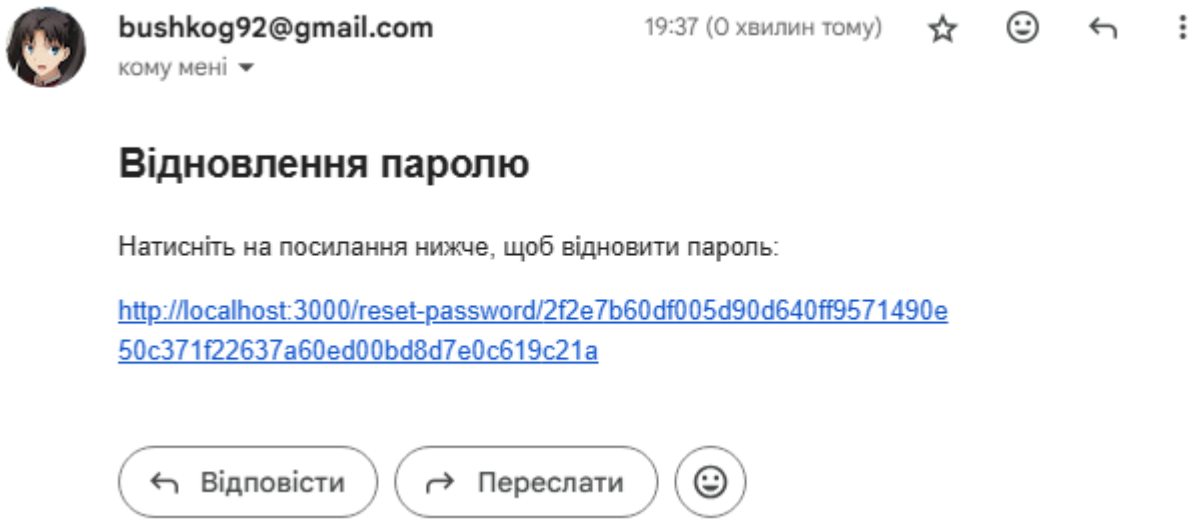

Рисунок 3.21 – Лист з відновленням паролю

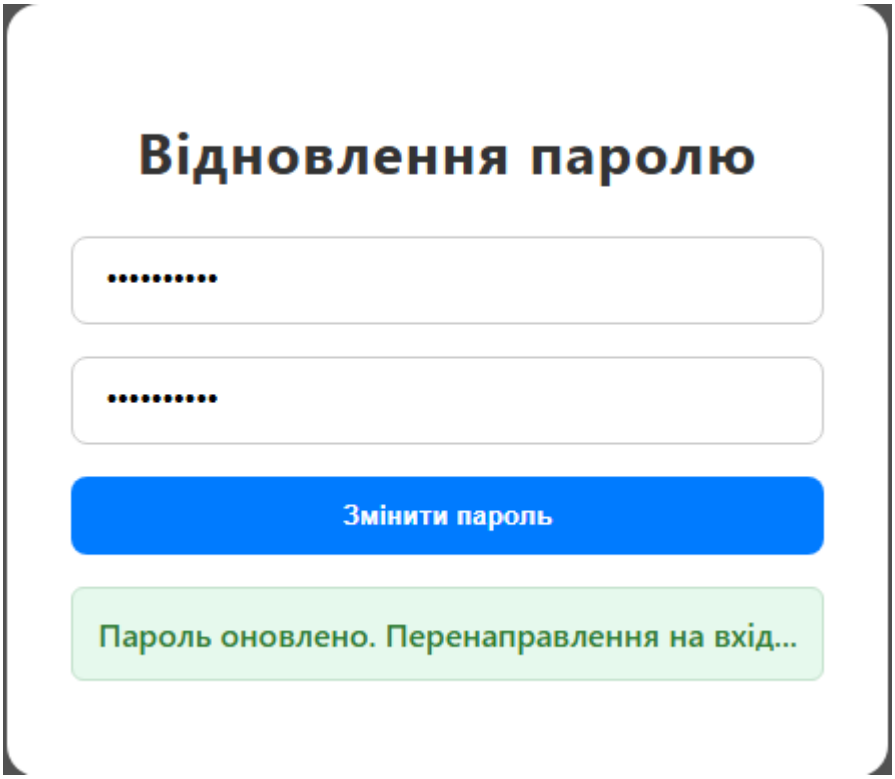

Рисунок 3.22 – Форма для відновлення паролю

Скидання паролю.

```
router.post("/auth/reset-password/:token", async (req, res) => {
  const { token } = req.params;
  const { password } = req.body;

  try {
    console.log('Запит на скидання паролю, токен:', token);
    const user = await User.findOne({
      resetToken: token,
      resetTokenExpiry: { $gt: Date.now() },
```

```
    });

    if (!user) {
        console.log("Не знайдено користувача або токен прострочено");
        return res.status(400).json({ message: "Недійсний або прострочений токен" });
    }


    user.password = await bcrypt.hash(password, 10);
    user.resetToken = undefined;
    user.resetTokenExpiry = undefined;
    await user.save();
    console.log("Пароль змінено для", user.email);

    res.json({ message: "Пароль успішно змінено" });
  } catch (err) {
    res.status(500).json({ message: "Помилка зміни паролю", error: err.message });
  }
});
```

Якщо токен валідний і ще не прострочений, пароль хешується та зберігається. Токен обнуляється.

ВИСНОВОК

У дипломній роботі було розроблено повнофункціональну інформаційну систему для футбольного клубу, яка включає вебінтерфейс для взаємодії з користувачами, адміністративну панель для керування вмістом та серверну частину з базою даних. Реалізація проєкту виконувалась із застосуванням сучасного технологічного стеку: React на фронтенді, Node.js та Express на бекенді, а також MongoDB для зберігання даних.

На етапі аналізу було досліджено проблемну область, визначено основні функціональні вимоги до системи: управління новинами, гравцями, матчами, турнірною таблицею, реєстрація та авторизація користувачів, а також можливості для адміністрування даних.

У рамках розробки реалізовано структуру бази даних з використанням Mongoose-схем, налаштовано маршрути API та інтегровано механізми автентифікації й авторизації користувачів з токенами JWT. На фронтенді створено зручну та інтуїтивну навігацію з використанням React Router, а також реалізовано окремі сторінки для кожного функціонального модуля системи.

Особливу увагу приділено реалізації завантаження зображень гравців і новин за допомогою Multer, а також безпечній обробці запитів на сервер. Було створено адміністративну панель із захищеним доступом, що дозволяє керувати вмістом сайту безпосередньо через вебінтерфейс.

Завдяки адаптивному дизайну інтерфейс системи коректно відображається як на настільних ПК, так і на мобільних пристроях, що підвищує зручність користування.

Результати тестування підтвердили коректну роботу всіх модулів проєкту. Реалізована інформаційна система відповідає поставленим вимогам і є готовою до практичного впровадження у ФК. Вона дозволить покращити організацію роботи з інформацією, підвищити ефективність управління діяльністю команди та покращити комунікацію з вболівальниками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація React [Електронний ресурс]. – Режим доступу: https://react.dev/learn
2. React: Початок роботи – MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Client-side_JavaScript_frameworks/React_getting_started
3. Сучасний підручник JavaScript [Електронний ресурс]. – Режим доступу: https://uk.javascript.info/
4. Node Cookbook: рішення та найкращі практики для Node.js 18 / Девід Марк Клементс. – Бірмінгем: Packt Publishing, 2022. – 408 с.
5. Node.js: Повний довідник з серверного JavaScript / Себастьян Спрінгер. – Бонн: Rheinwerk Computing, 2022. – 834 с.
6. Express – документація MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
7. MongoDB University – освітня платформа MongoDB [Електронний ресурс]. – Режим доступу: https://learn.mongodb.com/
8. REST API Design Rulebook / Марк Річардс. – Бостон: O'Reilly Media, 2020. – 120 с.
9. RESTful Web APIs / Ленард Річардсон, Майк Мішра. – Бостон: O'Reilly Media, 2023. – 448 с.
10. Офіційна документація Mongoose [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/guide.html
11. JSON Web Token Handbook – Auth0 [Електронний ресурс]. – Режим доступу: https://auth0.com/learn/json-web-tokens/
12. Nodemailer – офіційна документація [Електронний ресурс]. – Режим доступу: https://nodemailer.com/about/

ДОДАТОК А

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <link rel="stylesheet" href="styles.css" />
    <title>Футбольний клуб "Оріон"</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './components/App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

App.jsx

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Header from "./Header";
import Footer from "./Footer";
import Main from "./Main";
import Team from "./Team";
import Nav from "./Nav";
import Table from "./Table";
import Matches from "./Matches";
import Login from "./Login";
```

```
import Register from "./Register";
import Profile from "./Profile";
import ForgotPassword from './ForgotPassword';
import ResetPassword from './ResetPassword';
import News from "./News.jsx";
import NewsDetails from "./NewsDetails";
import NewsArchive from "./NewsArchive";
import NewsForm from "./NewsForm";

function App() {
  return (
    <Router>
      <Header />
      <Nav />
      <div id="content">
        <div id="main">
          <Routes>
            <Route path="/" element={<Main />} />
            <Route path="/news" element={<News />} />
            <Route path="/team" element={<Team />} />
            <Route path="/table" element={<Table />} />
            <Route path="/matches" element={<Matches />} />
            <Route path="/register" element={<Register />} />
            <Route path="/login" element={<Login />} />
            <Route path="/profile" element={<Profile />} />
            <Route path="/forgot-password" element={<ForgotPassword />} />
            <Route path="/reset-password/:token" element={<ResetPassword />} />
            <Route path="/news/:id" element={<NewsDetails />} />
            <Route path="/news/archive" element={<NewsArchive />} />
            <Route path="/news/create" element={<NewsForm />} />
            <Route path="/news/edit/:id" element={<NewsForm isEdit />} />
          </Routes>
        </div>
      </div>
      <Footer />
    </Router>
  );
}

export default App;
```

Header.jsx

```
import "../styles/Header.css";
import React from 'react';
import { Link, useNavigate } from 'react-router-dom';

function Header() {
  const navigate = useNavigate();
  const user = JSON.parse(localStorage.getItem('user'));
```

```
  const handleLogout = () => {
    localStorage.removeItem("user");
    localStorage.removeItem("token");
    localStorage.removeItem("role");
    navigate("/login");
  };

  return (
    <header>
      <img id="logo" src="/images/orion_logo.png" alt="logo" />
      <p id="teamName">Оріон</p>

      <div className="auth-buttons">
        {user ? (
          <>
            <span className="user-info">
              👤 {user.firstName} {user.secondName}
            </span>
            <Link to="/profile" className="profile-link">Кабінет</Link>
            <button onClick={handleLogout} className="logout-button">Вийти</button>
          </>
        ) : (
          <>
            <Link to="/login">Увійти</Link>
            <Link to="/register">Зареєструватись</Link>
          </>
        )}
      </div>
    </header>
  );
}

export default Header;
```

Main.jsx

```
import React from "react";
import News from "./News";
import "../styles/Main.css";

function Main() {
    return <News />;
}

export default Main;
```

Footer.jsx

```jsx
import React from "react";
import { FaFacebook, FaTwitter, FaInstagram } from "react-icons/fa";
import "../styles/Footer.css";

const year = new Date().getFullYear();

function Footer() {
    return (
        <footer>
            <p>Copyright © {year} "Оріон"</p>
            <div className="social-icons">
                <a href="/facebook"><FaFacebook /></a>
                <a href="/twitter"><FaTwitter /></a>
                <a href="/instagram"><FaInstagram /></a>
            </div>
        </footer>
    );
}

export default Footer;
```

Nav.jsx

```jsx
import React from "react";
import { Link } from "react-router-dom";
import "../styles/Nav.css";

function Nav() {
  return (
    <nav id="nav">
      <ul>
        <li><Link to="/">Новини</Link></li>
        <li><Link to="/team">Команда</Link></li>
        <li><Link to="/table">Турнірна таблиця</Link></li>
        <li><Link to="/matches">Матчі</Link></li>
      </ul>
    </nav>
  );
}

export default Nav;
```

News.jsx

```jsx
import { useEffect, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import "../styles/News.css";
```

```
const API_BASE_URL = "http://localhost:5000";

const News = () => {
  const [news, setNews] = useState([]);
  const [selectedNews, setSelectedNews] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    fetch(`${API_BASE_URL}/api/news`)
      .then((res) => res.json())
      .then((data) => {
        console.log("Отримані новини:", data);
        const sorted = data.sort(
          (a, b) => new Date(b.createdAt) - new Date(a.createdAt)
        );
        setNews(sorted);
        setSelectedNews(sorted[0]);
      })
      .catch((err) => console.error("Помилка завантаження новин:", err));
  }, []);

  const handleViewArchive = () => {
    navigate("/news/archive");
  };

  return (
    <div className="news-container">
      <h1>Останні новини:</h1>

      {selectedNews && (
        <div
          className="news-main"
          style={{ backgroundImage: `url(${API_BASE_URL}${selectedNews.image})` }}
        >
          <div className="news-content">
            <h2>{selectedNews.title}</h2>
            <p>{selectedNews.content}</p>
            <button
              className="read-more-btn"
              onClick={() => navigate(`/news/${selectedNews._id}`)}
            >
              Детальніше
            </button>
          </div>
        </div>
      )}

      <div className="news-row">
        {news.slice(0, 5).map((item) => (
          <div
```

```
            key={item._id}
            className={`news-item ${selectedNews?._id === item._id ? "selected" : ""}`}
            style={{ backgroundImage: `url(${API_BASE_URL}${item.image})` }}
            onClick={() => setSelectedNews(item)}
            role="button"
            aria-label={`Обрати новину: ${item.title}`}
            tabIndex={0}
            onKeyDown={(e) => e.key === "Enter" && setSelectedNews(item)}
          >
            <div className="news-content">
              <h3>{item.title}</h3>
            </div>
          </div>
        ))}
      </div>

      <div className="news-footer">
        <button className="archive-button" onClick={handleViewArchive}>
          Переглянути архів новин
        </button>
      </div>
    </div>
  );
};

export default News;
```

PlayerCards.jsx

```
import React, { useEffect, useState } from "react";
import "../styles/PlayerCards.css";

function PlayerCards() {
  const [players, setPlayers] = useState([]);
  const [newPlayer, setNewPlayer] = useState({ name: "", position: "", country: "" });
  const [editingPlayerId, setEditingPlayerId] = useState(null);
  const [editedPlayer, setEditedPlayer] = useState({ name: "", position: "", country: ""
});

  const token = localStorage.getItem("token");
  const userRole = localStorage.getItem("role");
  const isAdmin = userRole === "admin";

  useEffect(() => {
    fetch("http://localhost:5000/api/players")
      .then(res => res.json())
      .then(data => setPlayers(data))
      .catch(err => console.error("Помилка при завантаженні гравців:", err));
  }, []);
```

```
  const handleAddPlayer = async () => {
    const res = await fetch("http://localhost:5000/api/players", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      },
      body: JSON.stringify(newPlayer)
    });

    if (res.ok) {
      const data = await res.json();
      setPlayers([...players, data]);
      setNewPlayer({ name: "", position: "", country: "" });
    } else {
      alert("Помилка при додаванні (можливо, не авторизовано або немає прав)");
    }
  };

  const handleDelete = async (id) => {
    const res = await fetch(`http://localhost:5000/api/players/${id}`, {
      method: "DELETE",
      headers: { Authorization: `Bearer ${token}` }
    });

    if (res.ok) {
      setPlayers(players.filter(p => p._id !== id));
    } else {
      alert("Помилка при видаленні (можливо, не авторизовано або немає прав)");
    }
  };

  const handleEditClick = (player) => {
    setEditingPlayerId(player._id);
    setEditedPlayer({ name: player.name, position: player.position, country:
player.country });
  };

  const handleSaveEdit = async (id) => {
    const res = await fetch(`http://localhost:5000/api/players/${id}`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      },
      body: JSON.stringify(editedPlayer)
    });

    if (res.ok) {
      const updated = await res.json();
```

```
      setPlayers(players.map(p => p._id === id ? updated : p));
      setEditingPlayerId(null);
    } else {
      alert("Помилка при редагуванні (можливо, не авторизовано або немає прав)");
    }
  };

  return (
    <div className="player-cards-container">
      {isAdmin && (
        <form className="add-player-form" onSubmit={e => { e.preventDefault();
handleAddPlayer(); }}>
          <input
            type="text"
            placeholder="Ім'я"
            value={newPlayer.name}
            onChange={e => setNewPlayer({ ...newPlayer, name: e.target.value })}
            required
          />
          <input
            type="text"
            placeholder="Позиція"
            value={newPlayer.position}
            onChange={e => setNewPlayer({ ...newPlayer, position: e.target.value })}
            required
          />
          <input
            type="text"
            placeholder="Країна"
            value={newPlayer.country}
            onChange={e => setNewPlayer({ ...newPlayer, country: e.target.value })}
            required
          />
          <button type="submit">Додати</button>
        </form>
      )}

      <div className="player-cards-grid">
        {players.map(player => (
          <div key={player._id} className="player-card">
            <div className="player-photo">
              {player.photo ? <img src={player.photo} alt={player.name} /> : <span
className="placeholder">Фото</span>}
            </div>
            <div className="player-info">
              {editingPlayerId === player._id ? (
                <>
                  <input
                    value={editedPlayer.name}
```

```
                  onChange={e => setEditedPlayer({ ...editedPlayer, name:
e.target.value })}
                />
                <input
                  value={editedPlayer.position}
                  onChange={e => setEditedPlayer({ ...editedPlayer, position:
e.target.value })}
                />
                <input
                  value={editedPlayer.country}
                  onChange={e => setEditedPlayer({ ...editedPlayer, country:
e.target.value })}
                />
                <button onClick={() => handleSaveEdit(player._id)}>Зберегти</button>
                <button onClick={() => setEditingPlayerId(null)}>Скасувати</button>
              </>
            ) : (
              <>
                <h3>{player.name}</h3>
                <p>Позиція: {player.position}</p>
                <p>Країна: {player.country}</p>
                {isAdmin && (
                  <>
                    <button onClick={() =>
handleEditClick(player)}>Редагувати</button>
                    <button onClick={() => handleDelete(player._id)}>Видалити</button>
                  </>
                )}
              </>
            )}
          </div>
        </div>
      ))}
    </div>
  </div>
  );
}

export default PlayerCards;
```

Team.jsx

```
import React from "react";
import PlayerCards from "./PlayerCards";
import "../styles/Team.css"

function Team() {
  return (
    <div id="team-page">
      <h2>Список гравців команди:</h2>
```

```
      <PlayerCards />
    </div>
  );
}

export default Team;
```

Table.jsx

```
import React, { useEffect, useState } from "react";
import "../styles/Table.css";

function Table() {
  const [tableData, setTableData] = useState([]);

  useEffect(() => {
    fetch("http://localhost:5000/api/table")
      .then((res) => res.json())
      .then((data) => {
        const processed = data
          .sort((a, b) => b.points - a.points)
          .map((team, index) => ({
            position: team.position,
            name: team.name,
            matches: team.matches || 0,
            wins: team.wins || 0,
            draws: team.draws || 0,
            losses: team.losses || 0,
            goalsFor: team.goalsFor || 0,
            goalsAgainst: team.goalsAgainst || 0,
            goalDifference: team.goalDifference,
            points: team.points,
          }));
        setTableData(processed);
      })
      .catch((err) => console.error("Error loading table data:", err));
  }, []);

  return (
    <div className="table-container">
      <h2>Турнірна таблиця</h2>
      <table className="league-table">
        <thead>
          <tr>
            <th>№</th>
            <th>Команда</th>
            <th>М</th>
            <th>П</th>
            <th>Н</th>
```

```
            <th>ПО</th>
            <th>З</th>
            <th>ПР</th>
            <th>ЗП</th>
            <th>О</th>
          </tr>
        </thead>
        <tbody>
          {tableData.map((team) => (
            <tr key={team.position}>
              <td>{team.position}</td>
              <td>{team.name}</td>
              <td>{team.matches}</td>
              <td>{team.wins}</td>
              <td>{team.draws}</td>
              <td>{team.losses}</td>
              <td>{team.goalsFor}</td>
              <td>{team.goalsAgainst}</td>
              <td>{team.goalDifference}</td>
              <td>{team.points}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}

export default Table;
```

Matches.jsx

```
import React, { useEffect, useState } from 'react';
import "../styles/Matches.css";

const Matches = () => {
  const [matches, setMatches] = useState([]);

  useEffect(() => {
    fetch('http://localhost:5000/matches')
      .then((res) => {
        if (!res.ok) {
          throw new Error('Помилка при завантаженні матчів');
        }
        return res.json();
      })
      .then((data) => {
        const orionMatches = data.filter(
          (match) =>
            match.homeTeam?.name?.toLowerCase() === "оріон" ||
```

```
            match.awayTeam?.name?.toLowerCase() === "оріон"
        );
        setMatches(orionMatches);
      })
      .catch((error) => console.error('Помилка:', error));
  }, []);


  return (
    <div className="matches-container">
      <h2 className="matches-title">Матчі ФК "Оріон"</h2>
      {matches.map((match, index) => (
        <div key={index} className="match-card">
          <div className="match-datetime">
            <span>{new Date(match.date).toLocaleDateString('uk-UA', {
              year: 'numeric',
              month: 'long',
              day: 'numeric'
            })}</span>
            <span className="match-time">{match.time}</span>
          </div>
          <div className="match-teams">
            <span
              className={`match-team ${
                match.homeTeam.name.toLowerCase() === "оріон" ? "orion" : ""
              }`}
            >
              {match.homeTeam.name}
            </span>
            <span className="match-vs">vs</span>
            <span
              className={`match-team ${
                match.awayTeam.name.toLowerCase() === "оріон" ? "orion" : ""
              }`}
            >
              {match.awayTeam.name}
            </span>
          </div>
        </div>
      ))}
    </div>
  );
};

export default Matches;
```

Register.jsx

```
import React, { useState } from "react";
import "../styles/Login.css";
```

```
function Register() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [firstName, setFirstName] = useState("");
  const [secondName, setSecondName] = useState("");
  const [errorMsg, setErrorMsg] = useState("");
  const [successMsg, setSuccessMsg] = useState("");

  const handleRegister = async (e) => {
    e.preventDefault();
    setErrorMsg("");
    setSuccessMsg("");

    if (password !== confirmPassword) {
      setErrorMsg("Паролі не співпадають");
      return;
    }

    try {
      const res = await fetch("http://localhost:5000/api/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, firstName, secondName, password }),
      });

      const data = await res.json();

      if (!res.ok) {
        setErrorMsg(data.message || "Помилка реєстрації");
      } else {
        setSuccessMsg("Реєстрація успішна! Тепер увійдіть.");
        setEmail("");
        setFirstName("");
        setSecondName("");
        setPassword("");
        setConfirmPassword("");
      }
    } catch (err) {
      setErrorMsg("Сервер недоступний");
    }
  };

  return (
    <div className="login-container">
      <h2>Реєстрація</h2>
      <form onSubmit={handleRegister}>
        <input
          type="email"
```

```
          placeholder="Пошта"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
        />

        <input
          type="text"
          placeholder="Ім'я"
          value={firstName}
          onChange={(e) => setFirstName(e.target.value)}
          required
        />

        <input
          type="text"
          placeholder="Прізвище"
          value={secondName}
          onChange={(e) => setSecondName(e.target.value)}
          required
        />

        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />

        <input
          type="password"
          placeholder="Підтвердіть пароль"
          value={confirmPassword}
          onChange={(e) => setConfirmPassword(e.target.value)}
          required
        />

        <button type="submit" id="enter">Зареєструватися</button>
      </form>

      {errorMsg && <p className="error">{errorMsg}</p>}
      {successMsg && <p className="success">{successMsg}</p>}
    </div>
  );
}

export default Register;
```

Login.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "../styles/Login.css";

function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errorMsg, setErrorMsg] = useState("");
  const [successMsg, setSuccessMsg] = useState("");
  const [loading, setLoading] = useState(false);

  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setErrorMsg("");
    setSuccessMsg("");

    if (!email || !password) {
      setErrorMsg("Будь ласка, заповніть усі поля.");
      return;
    }

    setLoading(true);
    try {
      const res = await fetch("http://localhost:5000/api/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
      });

      const data = await res.json();

      if (!res.ok) {
        setErrorMsg(data.message || "Помилка входу");
      } else {
        setSuccessMsg("Успішний вхід!");
        localStorage.setItem("token", data.token);
        localStorage.setItem("user", JSON.stringify(data.user));
        localStorage.setItem("role", data.user.role);
        setTimeout(() => {
          navigate("/");
          window.location.reload();
        }, 500);
      }
    } catch (err) {
      setErrorMsg("Сервер недоступний. Спробуйте пізніше.");
    } finally {
```

```
      setLoading(false);
    }
  };

  return (
    <div className="login-container">
      <h2>Увійти</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="email"
          placeholder="Пошта"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
        />

        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />

        <button
          id="forgotPassword"
          type="button"
          onClick={() => navigate("/forgot-password")}
        >
          Забули пароль?
        </button>

        <button type="submit" id="enter" disabled={loading}>
          {loading ? "Зачекайте..." : "Увійти"}
        </button>
      </form>

      {errorMsg && <p className="error">{errorMsg}</p>}
      {successMsg && <p className="success">{successMsg}</p>}
    </div>
  );
}

export default Login;
```

ForgotPassword.jsx

```
import React, { useState } from "react";
import "../styles/Login.css";
```

```
function ForgotPassword() {
  const [email, setEmail] = useState("");
  const [message, setMessage] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage("");
    setError("");

    try {
      const res = await fetch("http://localhost:5000/api/forgot-password", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email }),
      });

      const data = await res.json();
      if (!res.ok) {
        setError(data.message || "Помилка надсилання листа");
      } else {
        setMessage("Інструкції для відновлення пароля надіслано на пошту");
      }
    } catch (err) {
      setError("Сервер недоступний");
    }
  };

  return (
    <div className="login-container">
      <h2>Забули пароль?</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="email"
          placeholder="Введіть вашу пошту"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
        />
        <button type="submit" id="enter">Відновити пароль</button>
      </form>

      {error && <p className="error">{error}</p>}
      {message && <p className="success">{message}</p>}
    </div>
  );
}
```

```
export default ForgotPassword;
```

ResetPassword.jsx

```
import React, { useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import "../styles/ResetPassword.css";

function ResetPassword() {
  const { token } = useParams();
  const navigate = useNavigate();

  const [password, setPassword] = useState('');
  const [confirm, setConfirm] = useState('');
  const [message, setMessage] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage('');
    setError('');

    if (password !== confirm) {
      setError('Паролі не співпадають');
      return;
    }

    try {
      const res = await fetch(`http://localhost:5000/api/auth/reset-password/${token}`,
{
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ password }),
      });

      const data = await res.json();

      if (!res.ok) {
        setError(data.message || 'Помилка при зміні паролю');
      } else {
        setMessage('Пароль оновлено. Перенаправлення на вхід...');
        setTimeout(() => navigate('/login'), 2000);
      }
    } catch (err) {
      setError('Сервер недоступний');
    }
  };

  return (
    <div className="reset-password-container">
```

```
      <h2>Відновлення паролю</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="password"
          placeholder="Новий пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />
        <input
          type="password"
          placeholder="Підтвердіть пароль"
          value={confirm}
          onChange={(e) => setConfirm(e.target.value)}
          required
        />
        <button type="submit">Змінити пароль</button>
      </form>
      {error && <p className="error">{error}</p>}
      {message && <p className="success">{message}</p>}
    </div>
  );
}

export default ResetPassword;
```

Profile.jsx

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import "../styles/Profile.css";

function Profile() {
  const navigate = useNavigate();
  const [user, setUser] = useState(null);

  useEffect(() => {
    const savedUser = localStorage.getItem("user");
    if (savedUser) {
      setUser(JSON.parse(savedUser));
    } else {
      navigate("/login");
    }
  }, [navigate]);

  const handleLogout = () => {
    localStorage.removeItem("user");
    localStorage.removeItem("token");
    localStorage.removeItem("role");
    navigate("/login");
```

```
    window.location.reload();
  };


  if (!user) return null;

  return (
    <div className="profile-container">
      <h1>Профіль користувача</h1>
      <div className="profile-info">
        <p><strong>Ім'я:</strong> {user.firstName}</p>
        <p><strong>Прізвище:</strong> {user.secondName}</p>
        <p><strong>Email:</strong> {user.email}</p>
        <button onClick={handleLogout}>Вийти</button>
      </div>
    </div>
  );
}

export default Profile;
```

NewsDetails.jsx

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import "../styles/NewsDetails.css";

const NewsDetails = () => {
  const { id } = useParams();
  const [news, setNews] = useState(null);

  useEffect(() => {
    fetch(`http://localhost:5000/api/news/${id}`)
      .then((res) => res.json())
      .then((data) => setNews(data))
      .catch((err) => console.error("Помилка завантаження новини:", err));
  }, [id]);

  if (!news) return <div>Завантаження...</div>;

  return (
    <div className="news-details">
      <h1>{news.title}</h1>
      <img src={`http://localhost:5000${news.image}`} alt={news.title} />
      <p>{news.content}</p>
      <p className="meta">🕒 {new Date(news.createdAt).toLocaleString()}</p>
    </div>
```

```
  );
};

export default NewsDetails;
```

NewsArchive.jsx

```
import { useEffect, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import "../styles/NewsArchive.css";

const NewsArchive = () => {
  const [newsList, setNewsList] = useState([]);
  const [role, setRole] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    const storedRole = localStorage.getItem("role");
    setRole(storedRole);

    fetch("http://localhost:5000/api/news")
      .then((res) => res.json())
      .then((data) =>
        setNewsList(data.sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt)))
      )
      .catch((err) => console.error("Помилка завантаження новин:", err));
  }, []);

  const handleDelete = (id) => {
    if (!window.confirm("Точно видалити новину?")) return;

    fetch(`http://localhost:5000/api/news/${id}`, {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${localStorage.getItem("token")}`,
      },
    })
      .then(() => setNewsList((prev) => prev.filter((item) => item._id !== id)))
      .catch((err) => console.error("Помилка видалення:", err));
  };

  return (
    <div className="news-archive">
      <div className="archive-header">
        <h2>Архів новин</h2>
        {role === "admin" && (
          <Link to="/news/create" className="add-news-btn">+ Додати новину</Link>
        )}
      </div>
```

```
      {newsList.map((item) => (
        <div key={item._id} className="archive-item">
          <h3>{item.title}</h3>
          <p>{item.content.slice(0, 150)}...</p>

          <div className="archive-actions">
            <Link to={`/news/${item._id}`} className="details-btn">
              Детальніше
            </Link>

            {role === "admin" && (
              <>
                <button onClick={() =>
navigate(`/news/edit/${item._id}`)}>Редагувати</button>
                <button onClick={() => handleDelete(item._id)}>Видалити</button>
              </>
            )}
          </div>
        </div>
      ))}
    </div>
  );
};

export default NewsArchive;
```

NewsForm.jsx

```
import { useState, useEffect } from "react";
import { useNavigate, useParams } from "react-router-dom";
import "../styles/NewsForm.css";

const NewsForm = ({ isEdit = false }) => {
  const { id } = useParams();
  const navigate = useNavigate();
  const [formData, setFormData] = useState({ title: "", content: "" });
  const [imageFile, setImageFile] = useState(null);
  const [existingImage, setExistingImage] = useState("");

  useEffect(() => {
    if (isEdit && id) {
      fetch(`http://localhost:5000/api/news/${id}`)
        .then((res) => res.json())
        .then((data) => {
          setFormData({ title: data.title, content: data.content });
          setExistingImage(data.image); // для попереднього перегляду
        })
        .catch((err) => console.error("Помилка завантаження новини:", err));
    }
  }, [isEdit, id]);
```

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setFormData((prev) => ({ ...prev, [name]: value }));
};

const handleFileChange = (e) => {
  setImageFile(e.target.files[0]);
};

const handleSubmit = (e) => {
  e.preventDefault();

  const form = new FormData();
  form.append("title", formData.title);
  form.append("content", formData.content);
  if (imageFile) {
    form.append("image", imageFile);
  }

  const method = isEdit ? "PUT" : "POST";
  const url = isEdit
    ? `http://localhost:5000/api/news/${id}`
    : "http://localhost:5000/api/news";

  fetch(url, {
    method,
    headers: {
      Authorization: `Bearer ${localStorage.getItem("token")}`,
    },
    body: form,
  })
    .then((res) => res.json())
    .then(() => navigate("/news"))
    .catch((err) => console.error("Помилка при збереженні новини:", err));
};

return (
  <div className="news-form">
    <h2>{isEdit ? "Редагувати новину" : "Додати новину"}</h2>
    <form onSubmit={handleSubmit}>
      <label>Заголовок</label>
      <input name="title" value={formData.title} onChange={handleChange} required />

      <label>Зображення</label>
      <input type="file" name="image" onChange={handleFileChange} />
      {imageFile ? (
        <img
          src={URL.createObjectURL(imageFile)}
          alt="Прев'ю"
```

```
            style={{ maxWidth: "300px", marginTop: "10px" }}
          />
        ) : existingImage ? (
          <img
            src={`http://localhost:5000${existingImage}`}
            alt="Поточне зображення"
            style={{ maxWidth: "300px", marginTop: "10px" }}
          />
        ) : null}
        <label>Контент</label>
        <textarea
          name="content"
          value={formData.content}
          onChange={handleChange}
          required
          rows={8}
        />
        <button type="submit">{isEdit ? "Зберегти зміни" : "Додати новину"}</button>
      </form>
    </div>
  );
};

export default NewsForm;
```

Server.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const connectDB = require('./db');
const News = require('../models/News');
const Player = require('../models/Player');
const Team = require('../models/Team');
const Match = require('../models/Match');
const bcrypt = require('bcrypt');
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const authMiddleware = require('./middleware/authMiddleware');
const adminMiddleware = require('./middleware/adminMiddleware');
const multer = require('multer');
const path = require('path');

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
```

```
  },
});

const upload = multer({ storage });

const PORT = process.env.PORT || 5000;

const app = express();

app.use(cors());
app.use(express.json());
const authRoutes = require('./routes/auth');
app.use('/api', authRoutes);
app.use('/uploads', express.static('uploads'));

// Підключення до БД
connectDB();

// Новини
app.get('/api/news', async (req, res) => {
  try {
    const news = await News.find();
    res.json(news);
  } catch (err) {
    res.status(500).json({ error: 'Помилка отримання новин' });
  }
});

app.get('/api/news/:id', async (req, res) => {
  try {
    const news = await News.findById(req.params.id);
    if (!news) return res.status(404).json({ message: 'Новину не знайдено' });
    res.json(news);
  } catch (err) {
    res.status(500).json({ message: 'Помилка сервера' });
  }
});

// Створення новини (тільки адміну)
app.post('/api/news', authMiddleware, adminMiddleware, upload.single('image'), async
(req, res) => {
  try {
    const { title, content } = req.body;
    const image = req.file ? `/uploads/${req.file.filename}` : '';
    const news = new News({ title, content, image });
    const saved = await news.save();
    res.status(201).json(saved);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при створенні новини' });
  }
```

```
});

// Редагування новини (тільки адміну)
app.put('/api/news/:id', authMiddleware, adminMiddleware, upload.single('image'), async
(req, res) => {
  try {
    const { title, content } = req.body;
    const updateData = { title, content };

    if (req.file) {
      updateData.image = `/uploads/${req.file.filename}`;
    }

    const updated = await News.findByIdAndUpdate(req.params.id, updateData, { new: true
});
    if (!updated) return res.status(404).json({ message: 'Новину не знайдено' });

    res.json(updated);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при оновленні новини' });
  }
});

// Видалення новини (тільки адміну)
app.delete('/api/news/:id', authMiddleware, adminMiddleware, async (req, res) => {
  try {
    const deleted = await News.findByIdAndDelete(req.params.id);
    if (!deleted) return res.status(404).json({ message: 'Новину не знайдено' });
    res.json({ message: 'Новину видалено' });
  } catch (err) {
    res.status(500).json({ message: 'Помилка при видаленні новини' });
  }
});

app.get('/api/news/archive', async (req, res) => {
  try {
    const news = await News.find().sort({ createdAt: -1 }); // найновіші новини є
першими
    res.json(news);
  } catch (err) {
    res.status(500).json({ error: 'Помилка отримання архіву новин' });
  }
});

// Гравці
// Отримати всіх гравців
app.get('/api/players', async (req, res) => {
  try {
    const players = await Player.find();
    res.json(players);
```

```
  } catch (err) {
    res.status(500).json({ message: 'Помилка при завантаженні гравців' });
  }
});

// Добавити гравця — тільки адміну
app.post('/api/players', authMiddleware, adminMiddleware, async (req, res) => {
  try {
    const player = new Player(req.body);
    const saved = await player.save();
    res.status(201).json(saved);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при створенні гравця' });
  }
});

// РЕДАГУВАННЯ гравця — тільки адміну
app.put('/api/players/:id', authMiddleware, adminMiddleware, async (req, res) => {
  try {
    const player = await Player.findByIdAndUpdate(req.params.id, req.body, { new: true
});
    res.json(player);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при оновленні гравця' });
  }
});

// ВИДАЛЕННЯ гравця — тільки адміну
app.delete('/api/players/:id', authMiddleware, adminMiddleware, async (req, res) => {
  try {
    await Player.findByIdAndDelete(req.params.id);
    res.json({ message: 'Гравця видалено' });
  } catch (err) {
    res.status(500).json({ message: 'Помилка при видаленні' });
  }
});

// Таблиця
app.get('/api/table', async (req, res) => {
  try {
    const teams = await Team.find();

    const table = teams.map(team => ({
      ...team._doc,
      goalDifference: team.goalsFor - team.goalsAgainst,
      points: team.wins * 3 + team.draws,
    }))
    .sort((a, b) =>
      b.points - a.points ||
      b.goalDifference - a.goalDifference ||
```

```
      b.goalsFor - a.goalsFor
    )
    .map((team, index) => ({ position: index + 1, ...team }));

    res.json(table);
  } catch (err) {
    res.status(500).json({ error: 'Помилка отримання таблиці' });
  }
});

// Матчі
app.get('/matches', async (req, res) => {
  try {
    const matches = await Match.find()
      .populate('homeTeam', 'name logo')
      .populate('awayTeam', 'name logo');
    res.json(matches);
  } catch (error) {
    res.status(500).json({ error: 'Помилка при завантаженні матчів' });
  }
});

// Реєстрація
app.post('/api/register', async (req, res) => {
  const { email, firstName, secondName, password } = req.body;
  try {
    console.log("Реєстрація", req.body);
    const existing = await User.findOne({ email });
    if (existing) {
      return res.status(400).json({ message: "Користувач уже існує" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ email, firstName, secondName, password: hashedPassword
});
    const saved = await newUser.save();
    console.log("Збережено користувача:", saved);

    res.status(201).json({ message: "Користувач створений", user: { email } });
  } catch (err) {
    console.error("Помилка при реєстрації:", err);
    res.status(500).json({ message: "Помилка сервера", error: err.message });
  }
});

// Логін
app.post('/api/login', async (req, res) => {
  try {
    const { email, password } = req.body;
```

```
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: "Користувача не знайдено" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(401).json({ message: "Невірний пароль" });

    // Генерація токена
    const token = jwt.sign(
      { userId: user._id, email: user.email },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );


    res.status(200).json({
      message: "Успішний вхід",
      token,
      user: {
        firstName: user.firstName,
        secondName: user.secondName,
        email: user.email,
        role: user.role
      }
    });
  } catch (err) {
    res.status(500).json({ error: "Помилка авторизації" });
  }
});

// Запуск
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

db.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  firstName: String,
  secondName: String,
  password: { type: String, required: true },
  role: { type: String, enum: ['admin', 'user'], default: 'user' },
  resetToken: String,
  resetTokenExpiry: Date
});

module.exports = mongoose.model('User', userSchema);
```

Match.js

```
const mongoose = require('mongoose');

const matchSchema = new mongoose.Schema({
  date: Date,
  time: String,
  homeTeam: { type: mongoose.Schema.Types.ObjectId, ref: 'Team', required: true },
  awayTeam: { type: mongoose.Schema.Types.ObjectId, ref: 'Team', required: true },
  result: {
    homeGoals: { type: Number, default: null },
    awayGoals: { type: Number, default: null }
  }
});

module.exports = mongoose.model('Match', matchSchema);
```

News.js

```
const mongoose = require('mongoose');

const newsSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  image: String,
  content: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('News', newsSchema);
```

Player.js

```
const mongoose = require('mongoose');

const playerSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  position: {
    type: String,
```

```
    required: true
  },
  country: {
    type: String,
    required: true
  },
  photo: {
    type: String,
    default: ''
  },
}, {timestamps: true});

module.exports = mongoose.model('Player', playerSchema);
```

Team.js

```
const mongoose = require('mongoose');

const teamSchema = new mongoose.Schema({
  name: { type: String, required: true, unique: true },
  matches: { type: Number, default: 0 },
  wins: { type: Number, default: 0 },
  draws: { type: Number, default: 0 },
  losses: { type: Number, default: 0 },
  goalsFor: { type: Number, default: 0 },
  goalsAgainst: { type: Number, default: 0 }
});

module.exports = mongoose.model('Team', teamSchema);
```

User.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  firstName: String,
  secondName: String,
  password: { type: String, required: true },
  role: { type: String, enum: ['admin', 'user'], default: 'user' },
  resetToken: String,
  resetTokenExpiry: Date
});

module.exports = mongoose.model('User', userSchema);
```

authMiddleware.js

```
const jwt = require('jsonwebtoken');
const User = require('../../models/User');

module.exports = async function (req, res, next) {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Немає токена' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId).select('-password');
    if (!user) return res.status(401).json({ message: 'Користувача не знайдено' });

    req.user = user;
    next();
  } catch (err) {
    return res.status(403).json({ message: 'Недійсний токен' });
  }
};
```

adminMiddleware.js

```
module.exports = function (req, res, next) {
  if (req.user && req.user.role === 'admin') {
    next();
  } else {
    return res.status(403).json({ message: 'Доступ заборонено: тільки для
адміністратора' });
  }
};
```

auth.js

```
const express = require("express");
const crypto = require("crypto");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const User = require("../../models/User");
const nodemailer = require("nodemailer");

const router = express.Router();

router.post("/forgot-password", async (req, res) => {
  const { email } = req.body;
```

```
  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(404).json({ message: "Користувача не знайдено" });

    const token = crypto.randomBytes(32).toString("hex");
    user.resetToken = token;
    user.resetTokenExpiry = Date.now() + 3600000;
    await user.save();

    const resetLink = `http://localhost:3000/reset-password/${token}`;

    const transporter = nodemailer.createTransport({
      service: "gmail",
      auth: {
        user: process.env.MAIL_USER,
        pass: process.env.MAIL_PASS,
      },
    });

    await transporter.sendMail({
      to: user.email,
      subject: "Відновлення паролю",
      html: `
        <h2>Відновлення паролю</h2>
        <p>Натисніть на посилання нижче, щоб відновити пароль:</p>
        <a href="${resetLink}">${resetLink}</a>
      `,
    });

    res.json({ message: "Інструкції для відновлення паролю надіслано на пошту" });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error: err.message });
  }
});

router.post("/auth/reset-password/:token", async (req, res) => {
  const { token } = req.params;
  const { password } = req.body;

  try {
    console.log('Запит на скидання паролю, токен:', token);
    const user = await User.findOne({
      resetToken: token,
      resetTokenExpiry: { $gt: Date.now() },
    });

    if (!user) {
        console.log("Не знайдено користувача або токен прострочено");
        return res.status(400).json({ message: "Недійсний або прострочений токен" });
    }
```

```
    user.password = await bcrypt.hash(password, 10);
    user.resetToken = undefined;
    user.resetTokenExpiry = undefined;
    await user.save();
    console.log("Пароль змінено для", user.email);

    res.json({ message: "Пароль успішно змінено" });
  } catch (err) {
    res.status(500).json({ message: "Помилка зміни паролю", error: err.message });
  }
});

module.exports = router;
```

MatchGenerator.js

```
const Team = require('../models/Team');

const getRandomTime = () => {
  const hour = Math.floor(Math.random() * 11) + 12;
  return `${hour}:00`;
};

const scheduleRound = (roundMatches, startDate, endDate, orionId) => {
  let date = new Date(startDate);
  const matches = [];
  let homeStreak = 0;
  let awayStreak = 0;

  while (roundMatches.length > 0) {
    let index = roundMatches.findIndex(({ home }) => {
      const isHome = home.toString() === orionId.toString();
      return (isHome && homeStreak < 3) || (!isHome && awayStreak < 3);
    });

    if (index === -1) {
      date.setDate(date.getDate() + 7);
      homeStreak = 0;
      awayStreak = 0;
      index = 0;
    }

    const match = roundMatches.splice(index, 1)[0];
    const isHome = match.home.toString() === orionId.toString();

    matches.push({
        date: new Date(date),
        time: getRandomTime(),
```

```
        homeTeam: match.home,
        awayTeam: match.away,
        result: {
            homeGoals: null,
            awayGoals: null
        }

    });

    do {
      date.setDate(date.getDate() + Math.floor(Math.random() * 3) + 5);
    } while (date > new Date(endDate));

    if (isHome) {
      homeStreak++;
      awayStreak = 0;
    } else {
      awayStreak++;
      homeStreak = 0;
    }
  }

  return matches;
};

const generateMatches = async () => {
  const teams = await Team.find().lean();
  const orion = teams.find(t => t.name === 'Оріон');
  if (!orion) throw new Error("Команда 'Оріон' не знайдена в базі");

  const opponents = teams.filter(t => t._id.toString() !== orion._id.toString());

  const firstRound = opponents.map(opponent => {
    const isHome = Math.random() < 0.5;
    return {
      home: isHome ? orion._id : opponent._id,
      away: isHome ? opponent._id : orion._id,
    };
  });

  const secondRound = firstRound.map(({ home, away }) => ({
    home: away,
    away: home,
  }));

  const shuffledFirst = [...firstRound].sort(() => Math.random() - 0.5);
  const shuffledSecond = [...secondRound].sort(() => Math.random() - 0.5);

  const firstRoundMatches = scheduleRound(
    shuffledFirst,
```

```
    new Date(2024, 8, 1),
    new Date(2024, 10, 30),
    orion._id
  );

  const secondRoundMatches = scheduleRound(
    shuffledSecond,
    new Date(2025, 2, 1),
    new Date(2025, 5, 30),
    orion._id
  );

  return [...firstRoundMatches, ...secondRoundMatches];
};

module.exports = generateMatches;
```