**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веборієнтованої інформаційної системи для супроводу суддівської діяльності в баскетболі»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів має посилання на відповідне джерело*

_____ Євгеній ОЛЬХОВСЬКИЙ
*(підпис)*

Виконав: здобувач вищої освіти групи ПД-42
Євгеній ОЛЬХОВСЬКИЙ

Керівник: Тимур ДОВЖЕНКО
*канд. техн. наук*

Рецензент: _____

**Київ 2026**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерії програмного забезпечення

Освітньо-професійна програма «Інженерії програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерія програмного забезпечення
_____Ірина ЗАМРІЙ
« _____» ____2026 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Ольховському Євгенію Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка веборієнтованої інформаційної системи для супроводу суддівської діяльності в баскетболі»
керівник кваліфікаційної роботи канд. техн. наук Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості щодо організації роботи суддівського апарату в баскетболі, дослідження існуючих інформаційних систем спортивного менеджменту та недоліків ручного керування розкладами, технічний огляд інструментів веброзробки із супровідною документацією.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області, дослідження існуючих бізнес-процесів організації суддівства в баскетболі та обґрунтування необхідності створення спеціалізованої вебсистеми.
2. Проєктування клієнт-серверної архітектури, логічної структури реляційної бази даних та моделі безпеки інформаційної системи
3. Програмна реалізація, опис функціонування та комплексне тестування серверної і клієнтської частин вебзастосунку для супроводу суддівської діяльності

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма компонентів вебзастосунку.
5. Діаграма варіантів використання вебзастосунку.
6. Діаграма послідовності процесу призначення арбітра на матч
7. Екранні форми.
8. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Підбір та аналіз науково-технічної літератури | 25.02.-04.03.2026 | |
| 2 | Аналіз та дослідження існуючих аналогів систем спортивного менеджменту | 05.03-13.03.2026 | |
| 3 | Визначення вимог до веборієнтованої системи супроводу суддівської діяльності | 14.03-19.03.2026 | |
| 4 | Проєктування архітектури системи та реляційної бази даних | 20.03.-01.04.2026 | |
| 5 | Програмна реалізація клієнтської та серверної частин вебзастосунку | 02.04.-28.04.2026 | |
| 6 | Оформлення роботи: вступ, висновки, реферат | 29.04-09.05.2026 | |
| 7 | Розробка демонстраційних матеріалів | 12.05-18.05.2026 | |
| 8 | Попередній захист роботи | 12.05-22.05.2026 | |
| 9 | Подання роботи | 23.05-30.05.2026 | |

Здобувач вищої освіти _____ Євгеній ОЛЬХОВСЬКИЙ
*(підпис)*

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
*(підпис)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 1 табл., 27 рис., 20 джерел.

*Мета роботи* – оптимізація процесу призначення арбітрів на баскетбольні матчі за рахунок розробки спеціалізованої веборієнтованої інформаційної системи обліку та планування.

*Об'єкт дослідження* – процес організації та кадрового супроводу суддівської діяльності у баскетбольних змаганнях.

*Предмет дослідження* – методи, засоби та вебтехнології для розробки інформаційних систем керування розкладом спортивних арбітрів.

*Короткий зміст роботи*: У кваліфікаційній роботі проаналізовано поточний стан організації суддівства у вітчизняному баскетболі та виявлено критичні недоліки ручного керування розкладами через електронні таблиці. Досліджено функціонал закордонних аналогів (Assignr, ArbiterSports, HorizonWebRef). На базі проведеного аналізу було спроєктовано та програмно реалізовано клієнт-серверний вебзастосунок «RefMate».

Серверна частина написана мовою Java з використанням фреймворку Spring Boot, а клієнтська – на Vanilla JS з Fetch API. Збереження даних організовано у реляційній СУБД PostgreSQL. У системі реалізовано жорстке розмежування прав доступу на ролі «Адміністратор» та «Арбітр». Для забезпечення безпеки впроваджено stateless-авторизацію через JWT-токени та криптографічне хешування паролів за допомогою BCrypt.

Основний функціонал охоплює створення матчів, автоматичну фільтрацію вільних арбітрів для уникнення конфліктів у розкладі та миттєву генерацію Email-сповіщень через SMTP-протокол.

Хмарне розгортання виконано з використанням платформ Render, Vercel та Neon.tech.

Сферою використання розробленого програмного продукту є регіональні федерації баскетболу, аматорські ліги та комітети арбітрів, які потребують надійного інструменту для цифрового менеджменту персоналу.

КЛЮЧОВІ СЛОВА: СПОРТИВНИЙ МЕНЕДЖМЕНТ, АВТОМАТИЗАЦІЯ РОЗКЛАДІВ, JAVA, SPRING BOOT, POSTGRESQL, JWT, ВЕБЗАСТОСУНОК.

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

**ACID** (Atomicity, Consistency, Isolation, Durability) – набір властивостей транзакцій бази даних (атомарність, узгодженість, ізольованість, довговічність).

**AJAX** (Asynchronous JavaScript and XML) – підхід до побудови інтерактивних інтерфейсів користувача вебзастосунків.

**API** (Application Programming Interface) – програмний інтерфейс взаємодії між клієнтом і сервером.

**BEM** (Block, Element, Modifier) – методологія розробки та іменування CSS-класів.

**CDN** (Content Delivery Network) – географічно розподілена мережева інфраструктура, що забезпечує швидку доставку контенту користувачам. **CSS** (Cascading Style Sheets) – спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду.

**DOM** (Document Object Model) – об'єктна модель документа, яка дозволяє скриптам динамічно звертатися до вмісту вебсторінки.

**FIBA** (Fédération Internationale de Basketball) – Міжнародна федерація баскетболу.

**HTML** (HyperText Markup Language) – стандартизована мова розмітки документів у Всесвітній павутині.

**HTTP** (HyperText Transfer Protocol) – протокол передачі гіпертексту, що використовується для обміну даними у мережі.

**JAR** (Java ARchive) – формат файлу, що містить скомпільований Java-код та метадані. **JPA** (Java Persistence API) – специфікація Java, що надає можливість зберігати об'єкти в реляційній базі даних.

**JSON** (JavaScript Object Notation) – легкий текстовий формат для зберігання та обміну даними.

**JWT** (JSON Web Token) – стандарт для передачі інформації між сторонами у вигляді зашифрованого токена.

**ORM** (Object-Relational Mapping) – технологія програмування, яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов.

**PaaS** (Platform as a Service) – хмарна модель надання платформи (серверів, баз даних) як послуги.

**REST** (Representational State Transfer) – архітектурний стиль для проєктування мережевих вебзастосунків.

**SaaS** (Software as a Service) – бізнес-модель, при якій програмне забезпечення надається користувачу як послуга.

**SMTP** (Simple Mail Transfer Protocol) – мережевий протокол для передачі електронної пошти.

**SPA** (Single Page Application) – односторінковий вебзастосунок, що динамічно оновлюється без перезавантаження.

**SQL** (Structured Query Language) – структурована мова запитів для роботи з реляційними базами даних.

**UI** (User Interface) – інтерфейс користувача, що забезпечує взаємодію з вебзастосунком.

**UML** (Unified Modeling Language) – уніфікована мова моделювання в інженерії програмного забезпечення.

**UX** (User Experience) – користувацький досвід роботи з системою.

**UUID** (Universally Unique Identifier) – універсальний унікальний ідентифікатор.

**СУБД** – система управління базами даних.

**ФБУ** – Федерація баскетболу України.

# ВСТУП

Актуальність теми. Проведення будь-яких спортивних змагань, і особливо баскетбольних турнірів, пов'язане з необхідністю обробки значного масиву логістичної та кадрової інформації. Якість та своєчасність проведення матчу напряму залежать від того, чи буде вчасно сформована і чи прибуде на місце кваліфікована суддівська бригада. Сьогодні більшість регіональних та аматорських баскетбольних федерацій в Україні все ще спираються на ручні методи адміністрування. Графіки призначень формуються у звичайних електронних таблицях, а постійна комунікація щодо зайнятості арбітрів ведеться через розрізнені мобільні месенджери.

Такий децентралізований і неавтоматизований підхід неминуче призводить до помилок. Серед найпоширеніших проблем — одночасне призначення одного й того ж фахівця на різні ігри, плутанина в локаціях та втрата інформації у переповнених чатах. Вирішити цю організаційну кризу можна лише за допомогою цифровізації, створивши профільний програмний інструмент. Саме тому розробка спеціалізованої вебсистеми для супроводу суддівства є надзвичайно актуальною задачею.

Мета дослідження – оптимізація процесу призначення арбітрів на баскетбольні матчі за рахунок розробки спеціалізованої веборієнтованої інформаційної системи обліку та планування.

Для досягнення поставленої мети було сформульовано та вирішено такі задачі:

1. Провести огляд предметної області, проаналізувати існуючі системи спортивного менеджменту та виявити недоліки ручного керування розкладами суддів.
2. Сформулювати функціональні та нефункціональні вимоги до розроблюваної інформаційної системи, що дозволить визначити технічні характеристики та критерії успішності готового продукту.

3. Спроєктувати клієнт-серверну архітектуру вебзастосунку, модель безпеки та логічну структуру бази даних для зберігання турнірної інформації, що забезпечить цілісність, швидку обробку та надійний захист даних користувачів.
4. Розробити серверну логіку системи: реалізувати алгоритм автоматичного підбору вільних арбітрів, сервіс розсилки електронних сповіщень та модуль захищеної авторизації.
5. Створити адаптивний інтерактивний інтерфейс користувача із жорстким розмежуванням прав доступу для різних ролей.
6. Провести комплексне тестування розробленого програмного забезпечення для перевірки коректності роботи клієнтської та серверної частин.

Об’єкт дослідження – процес організації та кадрового супроводу суддівської діяльності у баскетбольних змаганнях.

Предмет дослідження – методи, засоби та вебтехнології для розробки інформаційних систем керування розкладом спортивних арбітрів.

Методи дослідження. У роботі застосовано методи системного аналізу для вивчення предметної області, структурного та об’єктно-орієнтованого проєктування – для визначення вимог, а також методи програмної інженерії – для реалізації та тестування клієнт-серверної архітектури.

Наукова новизна роботи полягає у формалізації вимог та створенні оптимізованого алгоритму призначення арбітрів, який автоматично відфільтровує зайнятий персонал, що повністю виключає ризик виникнення часових конфліктів у розкладі матчів.

Практична значущість отриманих результатів. Впровадження розробленого вебзастосунку дозволить організаторам змагань відмовитися від ручного збору даних, централізувати облік та автоматизувати сповіщення. Продукт значно скорочує час на адміністрування турнірів і готовий до використання суддівськими комітетами федерацій баскетболу.

# 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНІЧНИХ РІШЕНЬ

## 1.1 Оцінка стану та особливостей організації суддівства у сучасних баскетбольних змаганнях

Організація будь-якого баскетбольного турніру вимагає не лише підготовки команд та арен, а й безперебійної роботи суддівського апарату. Практика показує, що навіть регіональні чемпіонати чи студентські ліги налічують десятки ігор щотижня. Кожен такий матч потребує присутності повноцінної бригади: зазвичай це два або три арбітри на майданчику, а також від трьох до чотирьох суддів-секретарів за столом.

Згідно з офіційними правилами баскетболу[1], саме суддівська бригада несе повну відповідальність за контроль перебігу гри, ведення протоколу та управління ігровим часом. Відповідно, відсутність хоча б одного фахівця через плутанину в розкладі гарантовано призводить до затримки, а іноді й до скасування матчу.

Якщо на рівні загальнонаціональних професійних змагань працюють окремі комітети і процес адміністрування є більш стандартизованим, то на рівні міських та аматорських ліг ситуація значно складніша. Більшість таких організацій не мають доступу до централізованих баз даних або достатніх бюджетів для оплати ліцензій на дорогі закордонні програмні рішення для спортивного менеджменту.

Через це щоденне управління персоналом лягає безпосередньо на плечі головного судді турніру або призначаючого менеджера. Щотижня ця людина виконує об'ємну рутинну роботу. Спочатку потрібно зібрати інформацію про те, хто з арбітрів вільний у конкретні дні та години. Далі ці дані треба вручну зіставити з розкладом оренди спортивних залів. Також організатору необхідно постійно тримати в голові рівень кваліфікації кожного судді, щоб на принциповий і складний матч випадково не призначити бригаду з недосвідчених

новачків. Тільки після виконання всіх цих кроків сформований графік розсилається виконавцям.

На практиці цей процес найчастіше перетворюється на хаотичну роботу з розрізненими Excel-таблицями, телефонними дзвінками та особистим листуванням у кількох різних месенджерах одночасно. Такий підхід робить організаційну систему вкрай вразливою. Інформація про зайнятість людини може змінитися за кілька годин, локальні таблиці швидко втрачають актуальність, а людський фактор постійно провокує так звані «накладки» — ситуації, коли одного арбітра помилково призначають на дві різні ігри в один і той самий час.

Підсумовуючи, поточний стан організації суддівства на базовому та середньому рівнях характеризується високою часткою ручної праці та децентралізованістю даних. Це створює постійне організаційне перевантаження для менеджерів турнірів, що прямо вказує на гостру потребу в оптимізації цього процесу за допомогою вітчизняних спеціалізованих вебінструментів.

### 1.2 Дослідження бізнес-процесів призначення арбітрів та комунікації всередині суддівського корпусу

Для того щоб спроєктувати якісну інформаційну систему, необхідно детально розібрати поточний бізнес-процес призначення арбітра на матч (модель «AS-IS», тобто як це працює зараз без автоматизації). Цей процес є класичним прикладом багатоетапної комунікації, яка має кілька критичних "вузьких місць".

Усе починається з отримання розкладу туру від організаторів ліги. Головний суддя бачить перелік матчів, локації та час їх проведення. Наступним кроком є аналіз кадрового резерву. Формування бригади не зводиться до простого вибору вільних людей. Якщо матч має високий рівень складності і проводиться за потрійною механікою суддівства[2], головний суддя повинен підібрати одного старшого арбітра (Crew Chief) та двох суддів (Umpire), які

мають відповідну національну чи регіональну ліцензію і достатній досвід спільної роботи на майданчику.

Визначивши попередніх кандидатів, організатор переходить до фази комунікації. Зазвичай це виглядає як розсилка особистих повідомлень або повідомлень в групі: адміністратор пропонує судді відпрацювати конкретний матч і чекає на відповідь. Саме тут виникає найбільша затримка в часі. Арбітр може бути на основній роботі, у дорозі або судити іншу гру, тому відповідь іноді надходить через кілька годин. Увесь цей час процес формування бригади для конкретного матчу залишається замороженим.

Якщо арбітр відхиляє пропозицію (через хворобу чи інші плани), весь цикл пошуку та комунікації запускається наново. Якщо ж погоджується — адміністратор вручну вносить його прізвище до зведеної Excel-таблиці. Проте навіть узгоджений розклад не є гарантією стабільності. У разі раптового перенесення часу гри хоча б на одну годину, адміністратор змушений знову зв'язуватися з усією бригадою, перепитувати про їхню доступність на новий час і вносити ручні правки.

Аналіз цього комунікаційного ланцюга показує, що головна проблема полягає у відсутності превентивного збору даних про доступність (Availability). Головний суддя щоразу діє «наосліп», витрачаючи час на тих, хто свідомо не може працювати в цей день.

Цільова модель бізнес-процесу («TO-BE»), яку має реалізувати майбутній застосунок, передбачає інверсію цієї комунікації. Арбітри повинні заздалегідь самостійно відмічати в особистому кабінеті дати, коли вони готові судити. Завдяки цьому адміністратор під час створення матчу бачитиме випадаючий список виключно з тих фахівців, які точно вільні. Після натискання кнопки призначення система має автоматично відправити Email-сповіщення, позбавивши головного суддю необхідності вести особисті переписки. Це зведе комунікацію до двох кліків і повністю усуне ризик накладок у графіку.

### 1.3 Порівняльний аналіз існуючих систем спортивного менеджменту

Ринок програмного забезпечення для спортивного менеджменту пропонує декілька готових рішень, які покликані автоматизувати роботу ліг та федерацій. Більшість із них розроблені за кордоном і орієнтовані на західну модель управління спортом. Щоб зрозуміти, яким має бути майбутній вебзастосунок, доцільно проаналізувати найпопулярніші світові платформи: Assignr, ArbiterSports та HorizonWebRef.

Платформа Assignr (Рис.1.1) є одним із найвідоміших вебсервісів для асоціацій арбітрів. Її головна перевага полягає у зручному механізмі збору інформації про доступність персоналу та наявності інструментів масового призначення (Mass Assign). Система вміє автоматично розсилати Email- та Push-сповіщення, якщо в розкладі відбуваються зміни. Проте для українських реалій цей вебдодаток має два суттєві недоліки. По-перше, повністю відсутня українська локалізація інтерфейсу (Рис.1.2). По-друге, вебплатформа працює за моделлю платної підписки, вартість якої залежить від кількості суддів та матчів, що робить її економічно невигідною для регіональних змагань.


Рис. 1.1 Приклад екранної форми платформи Assignr

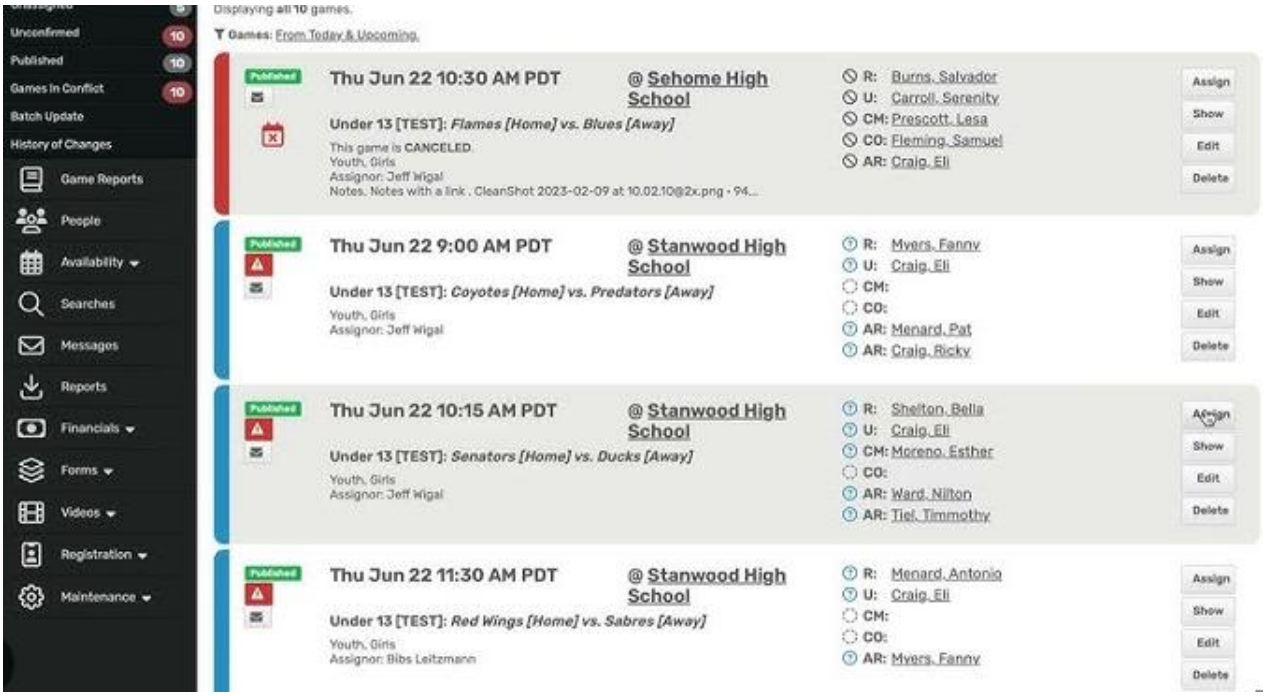



Рис. 1.2 Приклад екранної форми платформи Assignr

Наступний аналог — ArbiterSports (Рис.1.3). Це великий монополіст на ринку США, яким користуються школи, коледжі та професійні ліги. Це комплексна веборієнтована екосистема, яка охоплює планування турнірів, потужні алгоритми автоматичного призначення та навіть вбудований платіжний шлюз ArbiterPay для виплати гонорарів (Рис.1.4). Незважаючи на такий масштаб, система є надзвичайно дороговартісною.

Крім того, користувачі часто відзначають перевантажений і морально застарілий інтерфейс, який вимагає тривалого навчання перед початком роботи. Фінансовий модуль також орієнтований виключно на американську банківську систему, тому для вітчизняного ринку він не має практичної цінності.

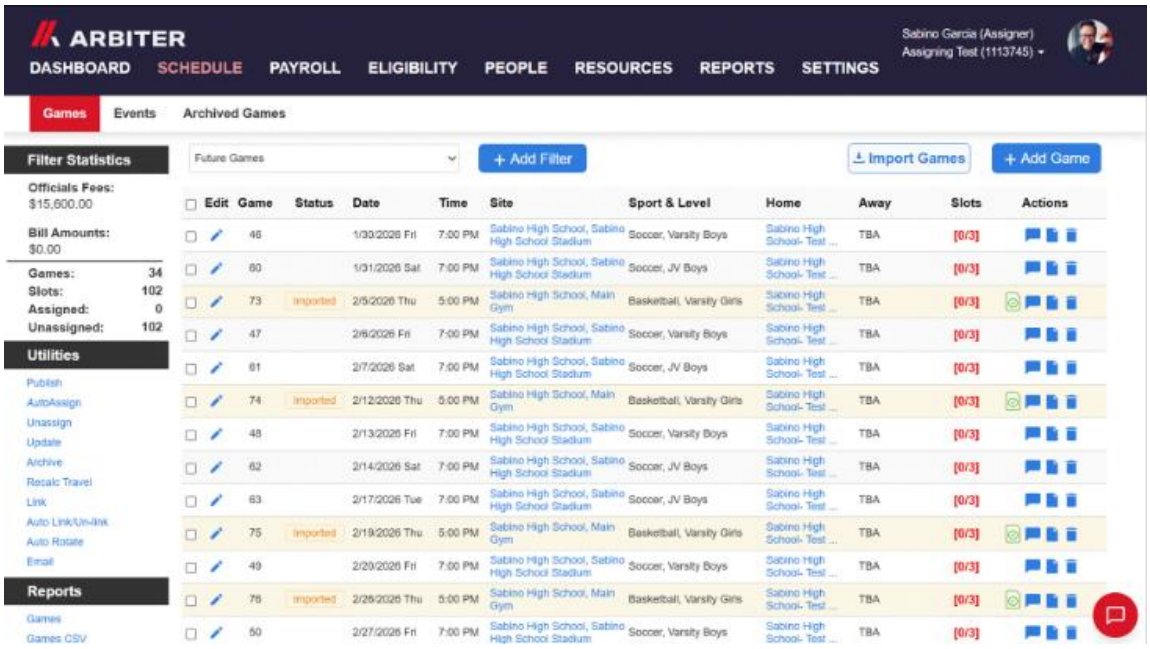



Рис. 1.3 Приклад екранної форми платформи ArbiterSports

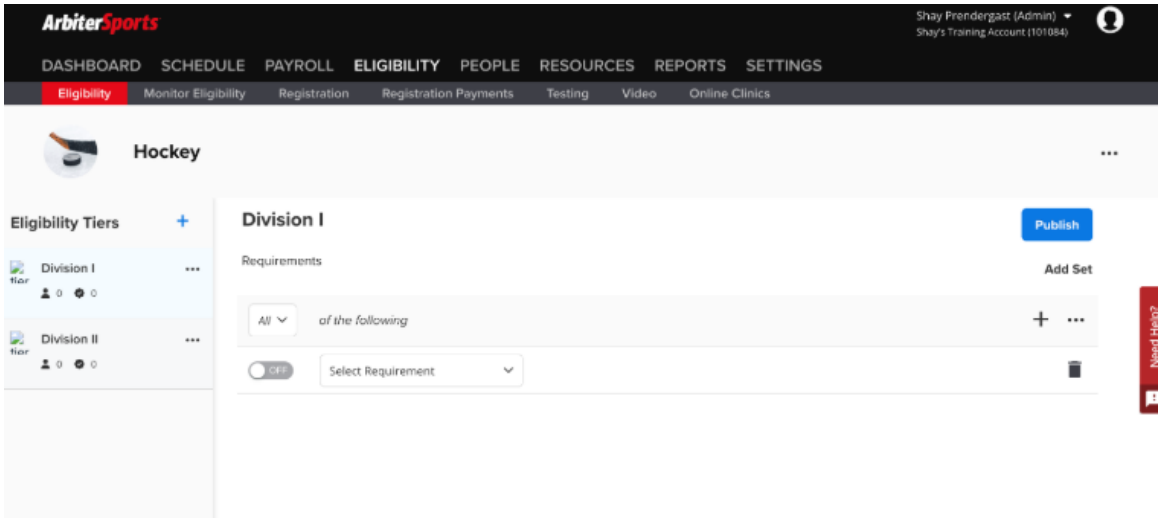



Рис. 1.4 Приклад екранної форми платформи ArbiterSports

Третій варіант — HorizonWebRef (Рис.1.5), який позиціонується як більш доступна альтернатива для ліг середнього розміру. Цей вебресурс пропонує гнучку систему ролей, внутрішню дошку оголошень та модуль відстеження ліцензій. Головним мінусом HorizonWebRef є його візуальна та технічна складова. Інтерфейс не відповідає сучасним стандартам UX-дизайну (Рис.1.6), а робота з вебсистемою через мобільний браузер є вкрай незручною, що критично для арбітрів, які постійно перебувають у русі.


Рис. 1.5 Приклад екранної форми платформи HorizonWebRef


Рис. 1.6 Приклад екранної форми платформи HorizonWebRef

Для наочності зведені результати аналізу характеристик розглянутих засобів представлено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз вебзастосунків для координації арбітрів

| **Показник** | **Assignr** | **ArbiterSports** | **HorizonWebRef** |
|---|---|---|---|
| **Цільова аудиторія** | Локальні ліги | Професійні ліги США | Ліги середнього розміру |
| **Безкоштовна версія** | Відсутня (лише Trial) | Відсутня | Відсутня |
| **Автоматизація сповіщень** | Присутня | Присутня | Присутня |
| **Українська локалізація** | Відсутня | Відсутня | Відсутня |
| **Сучасний UX/UI вебдизайн** | Присутня | Відсутня | Відсутня |
| **Контроль конфлікту розкладів** | Присутня | Присутня | Присутня |

Підсумовуючи результати огляду, можна зробити висновок, що жодне з існуючих SaaS-рішень не є ідеальним для вітчизняних аматорських та напівпрофесійних ліг. Вони або занадто дорогі та перевантажені зайвим функціоналом, або мають застарілий вебдизайн і проблеми з локалізацією.

Це підтверджує необхідність розробки власного вебзастосунку, який би поєднував сучасний зрозумілий інтерфейс, безкоштовність базового використання та чіткий фокус на автоматизації призначень без перевантаження системи зайвими модулями.

## 1.4 Виявлення недоліків ручного керування та обґрунтування необхідності автоматизації

Аналізуючи поточний стан організації змагань, можна чітко виділити критичні недоліки ручного формату управління. Перш за все, це відсутність єдиної інформаційної бази. Головний суддя змушений тримати в пам'яті або шукати в локальних файлах інформацію про поточну кваліфікацію та місцезнаходження десятків арбітрів. Якщо турнір масштабується і кількість ігор зростає, такий підхід неминуче дає збій.

Другим критичним фактором є низька швидкість комунікації. Процес узгодження лише однієї суддівської бригади через телефонні дзвінки чи розрізнені чати може тривати кілька годин. Якщо ж напередодні туру трапляється форс-мажор і хтось із фахівців повідомляє про хворобу, оперативна заміна перетворюється на стресову ситуацію для всього оргкомітету. Адміністратору доводиться терміново обдзвонювати весь резерв, не маючи перед очима актуальної картини їхньої зайнятості на цей день.

Окрім організаційних затримок, ручне ведення графіків сильно ускладнює збір статистики. Наприкінці ігрового місяця фінансовому відділу ліги необхідно отримати дані про кількість відпрацьованих матчів для кожного арбітра, щоб нарахувати гонорари. Без централізованої системи адміністратору доводиться піднімати архіви повідомлень та вручну зводити ці цифри, що призводить до математичних помилок.

Усе перелічене вище беззаперечно обґрунтовує гостру необхідність автоматизації. Впровадження веборієнтованої інформаційної системи дозволить перенести всю розрізнену інформацію у єдине безпечне середовище. Алгоритми системи зможуть самостійно аналізувати графіки та відкидати зайнятий персонал, а інтеграція поштових протоколів забезпечить миттєве інформування учасників процесу без втручання людини.

## 1.5 Визначення функціональних та нефункціональних вимог до проектованої системи

Спираючись на виявлені організаційні проблеми та проаналізований досвід існуючих комерційних платформ, було сформовано перелік архітектурних та логічних вимог до майбутнього вебзастосунку.

З точки зору базової функціональності, система повинна підтримувати жорстке розмежування прав доступу на основі ролей. Для ролі адміністратора необхідно реалізувати інструменти створення баскетбольних матчів, управління загальною базою суддів та безпосереднього призначення сформованих бригад на ігри. Зі свого боку, арбітр повинен отримати захищений доступ до особистого кабінету. У цьому просторі він зможе переглядати історію своїх минулих і майбутніх ігор, а також самостійно блокувати дні, коли він не може працювати (керування власною доступністю).

Критично важливою функціональною вимогою є наявність механізму алгоритмічної фільтрації під час призначення. Вебзастосунок має автоматично приховувати зі списку кандидатів тих фахівців, які вже отримали призначення на інший матч у цей самий час, усуваючи ризик логістичних накладок. Також система повинна вміти самостійно генерувати та відправляти персональні Email-сповіщення одразу після того, як адміністратор затвердив бригаду.

Серед нефункціональних вимог пріоритетними є продуктивність, адаптивність та безпека. Швидкодія системи вимагає відгуку API до 200 мс та рендерингу клієнтської частини до 2 секунд. Інтерфейс має коректно відображатися у всіх сучасних браузерах на екранах шириною від 320 пікселів. Захист персональних даних забезпечується криптографічним хешуванням паролів та безпечною stateless-авторизацією (JWT-токени). Крім того, модульна архітектура вебсервісу повинна стабільно обробляти навантаження від понад 100 одночасних користувачів без втрати швидкодії.

# 2. ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1 Вибір та обґрунтування архітектурного шаблону системи (Single Page Application та REST API)

Проєктування сучасного вебзастосунку вимагає чіткого розділення зон відповідальності між його компонентами. Для інформаційної системи супроводу суддівства «RefMate» було обрано класичну клієнт-серверну архітектуру (Рис.2.1) з використанням шаблону Single Page Application (SPA) для клієнтської частини та REST API для серверної [5].

Такий підхід означає, що система фізично та логічно розділена на два незалежні модулі, які спілкуються між собою за допомогою стандартизованих HTTP-запитів, обмінюючись даними у легкому текстовому форматі JSON. Серверна частина (Backend) виступає в ролі надійного обчислювального центру. Вона відповідає за підключення до бази даних, валідацію вхідної інформації, криптографічний захист, перевірку бізнес-правил (наприклад, чи не призначений арбітр на інший матч) та відправку Email-сповіщень. Сервер не генерує HTML-сторінки, а лише надає дані через чітко визначені кінцеві точки (ендпоінти).

Зі свого боку, клієнтська частина (Frontend) бере на себе всю відповідальність за побудову користувацького інтерфейсу. Концепція Single Page Application дозволяє завантажити основний HTML-документ та базові скрипти лише один раз під час першого входу у вебзастосунок. Усі подальші зміни на екрані (наприклад, перехід від списку матчів до профілю арбітра) відбуваються динамічно, без повного перезавантаження вебсторінки. Це критично важливо для цільової аудиторії продукту: арбітри часто перевіряють свій розклад зі смартфонів у дорозі або в спортивних залах із нестабільним мобільним інтернетом. SPA-архітектура забезпечує миттєвий відгук інтерфейсу, що створює досвід користування, наближений до нативних мобільних додатків.

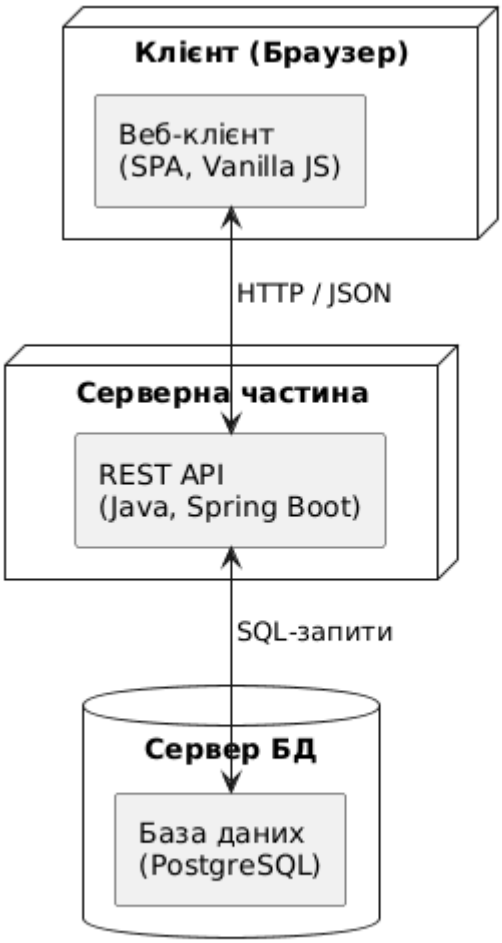



Рис. 2.1 Загальна клієнт-серверна архітектура вебзастосунку «RefMate»

Використання REST (Representational State Transfer) як архітектурного стилю для взаємодії гарантує, що вебсервер не зберігає інформацію про поточний стан клієнта (stateless). Кожен запит від браузера містить усю необхідну інформацію для його обробки, включаючи токен авторизації. Це робить серверну частину максимально незалежною і готовою до горизонтального масштабування у хмарному середовищі [10].

## 2.2 Проєктування логічної структури системи за допомогою UML-діаграм

Для того щоб візуалізувати вимоги до вебзастосунку та детально спланувати поведінку його компонентів до початку написання програмного коду, було використано уніфіковану мову моделювання (UML). UML-діаграми дозволяють описати систему з різних ракурсів: від загальних сценаріїв використання користувачами до низькорівневої структури програмних модулів.

### 2.2.1 Моделювання функціональних можливостей

Діаграма варіантів використання (Рис.2.2) описує взаємодію між зовнішніми акторами (користувачами) та системою, показуючи, які саме функції доступні кожній ролі. У вебзастосунку «RefMate» виділено дві основні дійові особи з кардинально різними правами доступу: «Адміністратор» (головний суддя або менеджер ліги) та «Арбітр».


Рис. 2.2 Діаграма варіантів використання (Use Case) вебзастосунку

Згідно з побудованою моделлю, Адміністратор є ініціатором більшості процесів у системі. До його ключових прецедентів належать:

- Створення та редагування матчів: внесення в базу інформації про дату, час, локацію та граючі команди.
- Перегляд загальної бази арбітрів: доступ до списку всього персоналу з контактними даними.
- Призначення суддівської бригади: вибір фахівців на конкретну гру. Цей прецедент обов'язково включає (зв'язок <<include>>) приховану системну функцію — автоматичну перевірку розкладу, яка не дозволить адміністратору призначити арбітра, якщо той вже зайнятий або самостійно заблокував цей день.

Роль Арбітра у системі має більш обмежений, але критично важливий для логістики функціонал. Його прецеденти охоплюють:

- Безпечну авторизацію: вхід у захищений особистий кабінет за допомогою логіна та пароля.
- Управління власною доступністю: можливість попередньо позначати у вбудованому календарі дні, коли арбітр не може працювати через особисті причини чи роботу в інших турнірах.
- Перегляд розкладу: отримання детальної інформації про свої майбутні призначення (хто напарники, де гра, який час).

Такий розподіл функцій на рівні прецедентів повністю задовольняє бізнес-вимоги, усуваючи дублювання роботи та автоматизуючи найскладніший етап — узгодження вільних дат.

### 2.2.2 Опис архітектурних компонентів системи

Для розуміння внутрішньої будови та взаємозв'язків модулів програмного продукту була розроблена діаграма компонентів (Рис.2.3). Цей вид моделювання дозволяє візуалізувати фізичну структуру вебзастосунку та показати, як саме розділена бізнес-логіка між різними шарами коду.

Клієнтська частина вебзастосунку складається з двох ключових блоків: модуля інтерфейсу користувача (UI), який безпосередньо відповідає за відображення даних та обробку подій у браузері, та менеджера API-запитів, який інкапсулює логіку взаємодії з сервером за допомогою технології Fetch.

Серверна частина, побудована на базі фреймворку Spring Boot, має класичну багатошарову архітектуру. Точкою входу для всіх зовнішніх запитів є REST Контролери. Вони приймають вхідні дані, делегуючи перевірку прав доступу спеціалізованому модулю безпеки, який працює з JWT-токенами. Якщо авторизація пройшла успішно, запит передається на шар бізнес-логіки (Services). Саме тут виконуються всі основні обчислення та перевірки алгоритмів призначення.

Для збереження результатів шар сервісів звертається до шару доступу до даних (Repositories), який за допомогою ORM-технології автоматично трансформує об'єкти мови Java у SQL-запити до реляційної бази даних PostgreSQL. Окремим модулем виділено сервіс Email-сповіщень, який активується асинхронно після успішного збереження інформації про призначення бригади.

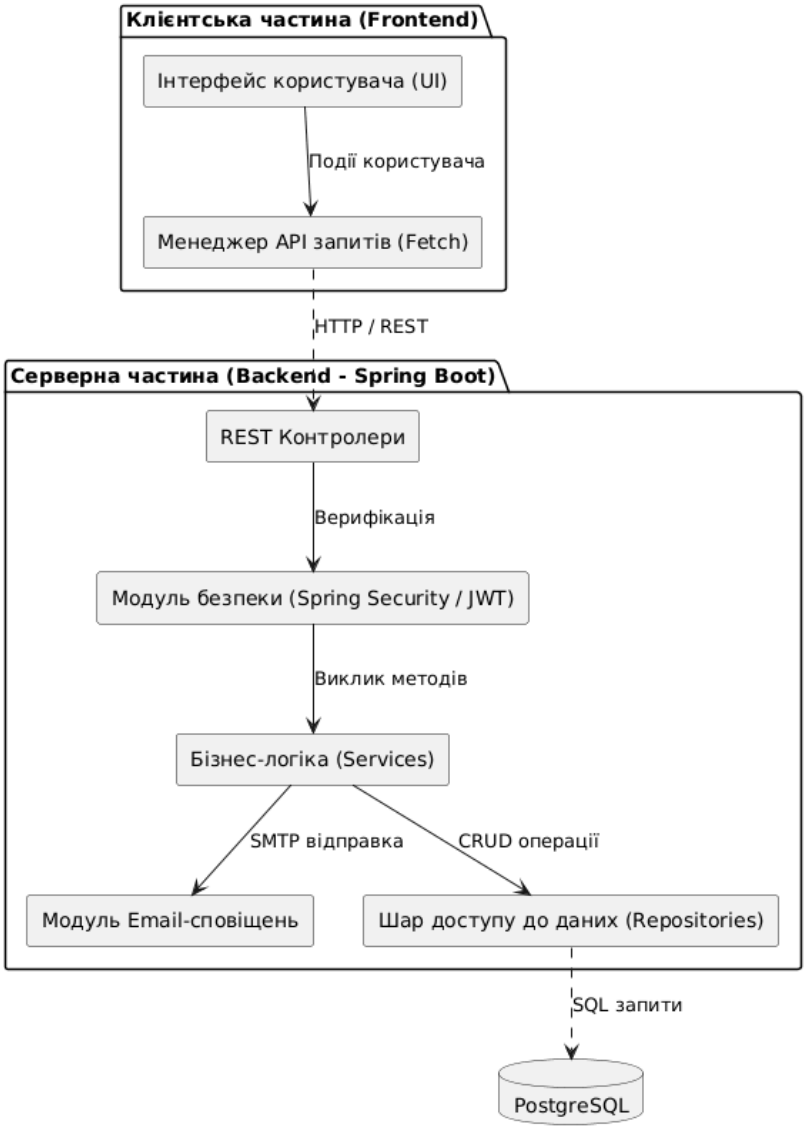



Рис. 2.3 Діаграма компонентів вебзастосунку

### 2.2.3 Моделювання послідовності взаємодії при призначенні арбітра

Щоб продемонструвати динаміку роботи розробленої системи у часі, було побудовано діаграму послідовності[20] (Рис.2.4). За основу взято найбільш критичний для предметної області бізнес-процес — призначення арбітра на конкретний баскетбольний матч. Ця модель детально показує хронологічний обмін повідомленнями між адміністратором, вебінтерфейсом, серверною частиною та базою даних.

Процес ініціюється, коли адміністратор відкриває сторінку створення матчу та задає необхідні параметри (дату і час). Вебклієнт миттєво формує HTTP-запит до серверної частини для отримання списку вільних фахівців. Сервер, отримавши запит, звертається до бази даних, де відбувається фільтрація: SQL-запит відкидає тих арбітрів, які мають статус недоступності на цей день або вже залучені до інших ігор. Відфільтрований список повертається на клієнт у форматі JSON і відображається на екрані.

Після того як адміністратор робить свій вибір і підтверджує призначення, вебзастосунок генерує POST-запит на збереження бригади. Серверна частина фіксує нові дані у PostgreSQL. Щойно транзакція успішно завершується, сервер автономно звертається до SMTP-шлюзу для генерації та відправки персонального електронного листа арбітру з деталями матчу.

Завершальним етапом є повернення HTTP-статусу успіху (200 OK) до веббраузера, що ініціює показ візуального сповіщення адміністратору про вдале завершення операції (Рис. 2.4).

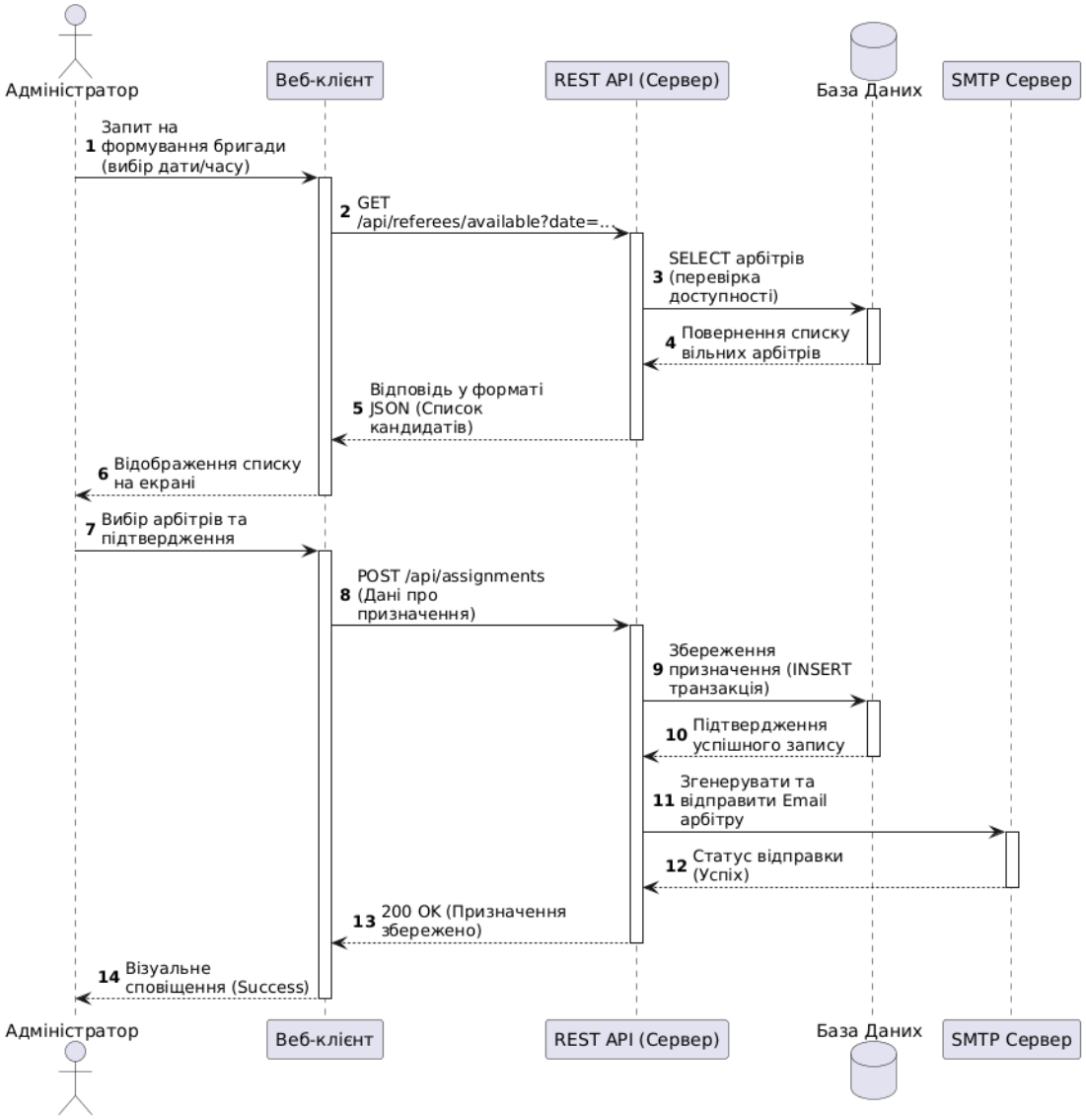



Рис. 2.4. Діаграма послідовності процесу призначення арбітра

## 2.3 Проєктування бази даних та моделі зберігання інформації

Зберігання та обробка структурованої інформації є критичним аспектом для будь-якого вебзастосунку. Оскільки предметна область спортивного суддівства передбачає наявність чітко визначених зв'язків між фахівцями, баскетбольними матчами та призначеннями, логічним і найбільш надійним вибором стала реляційна модель зберігання даних. Для фізичної реалізації бази даних було обрано систему управління реляційними базами даних (СУБД) PostgreSQL[12]. Вона відрізняється високою надійністю, суворо відповідає

стандартам ACID (атомарність, узгодженість, ізольованість, довговічність) та дозволяє ефективно будувати складні запити для агрегації і фільтрації інформації.

### 2.3.1 Побудова інфологічної моделі

Розробка структури бази даних вебзастосунку починається з інфологічного моделювання, яке дозволяє візуалізувати сутності предметної області та кардинальність зв'язків між ними. Основними бізнес-об'єктами системи «RefMate» є користувачі (арбітри та адміністратори), матчі, графіки доступності персоналу та безпосередні призначення на ігри (Рис.2.5).

Взаємодія між сутностями реалізується переважно через зв'язки типу «один до багатьох» (1:N) та «багато до багатьох» (M:N). Наприклад, один арбітр може мати багато записів про свою доступність у персональному календарі, а також бути призначеним на кілька різних ігор протягом ігрового сезону. Для вирішення зв'язку «багато до багатьох» між таблицею користувачів та таблицею матчів вводиться проміжна сутність «Призначення» (Assignment). Ця сутність фіксує не лише сам факт участі фахівця у грі, а й його конкретну роль на баскетбольному майданчику (наприклад, старший суддя або суддя).

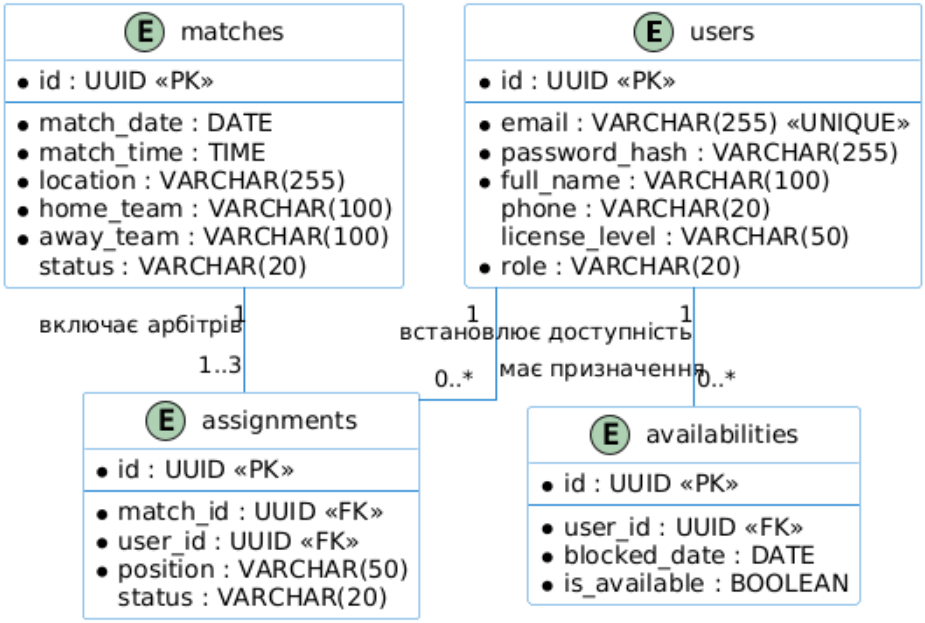



Рис. 2.5 Інфологічна модель бази даних вебзастосунку

### 2.3.2 Даталогічне проєктування та опис таблиць бази даних PostgreSQL

На основі розробленої інфологічної моделі було проведено даталогічне проєктування, яке визначає конкретні типи даних для кожного атрибута та правила цілісності у середовищі PostgreSQL[12]. Усі первинні ключі таблиць генеруються у форматі UUID, що значно підвищує рівень безпеки та захищає дані від перебору ідентифікаторів у вебзапитах.

Головна таблиця users (Користувачі) призначена для зберігання облікових записів. Вона містить поле email, яке виступає унікальним індексом для авторизації в системі, та password_hash для безпечного зберігання криптографічного хешу пароля. Також до таблиці включені поля з персональними даними (ім'я, телефон) та рівнем ліцензії ФБУ. Спеціальне поле role (типу VARCHAR) визначає права доступу до функцій вебзастосунку і може набувати значень ADMIN або REFEREE.

Таблиця matches (Матчі) зберігає всю логістичну інформацію про турнір. До її складу входять поля match_date та match_time, які ізолюють хронологічні дані для зручнішого пошуку. Також таблиця включає назву спортивної арени location, інформацію про граючі команди home_team та away_team. Додаткове поле status дозволяє відстежувати поточний стан гри (запланована, зіграна або скасована).

Таблиця assignments (Призначення) є ключовою сполучною ланкою бізнес-логіки. Вона містить зовнішні ключі match_id та user_id, які жорстко посилаються на відповідні батьківські таблиці. Задля забезпечення каскадної цілісності даних тут налаштовано правило ON DELETE CASCADE: якщо матч через певні причини скасовується і видаляється з бази даних вебсервісу, автоматично знищуються і всі пов'язані з ним призначення арбітрів. Також ця таблиця включає поле position (наприклад, Crew Chief, Umpire 1, Umpire 2), яке фіксує обов'язки судді в рамках потрійної механіки суддівства.

Остання базова таблиця availabilities (Доступність) формує персональні календарі арбітрів. Вона поєднує зовнішній ключ user_id, конкретну дату

blocked_date та логічне поле is_available (типу BOOLEAN). Завдяки цій структурі бекенд вебзастосунку може за частки секунди відфільтрувати список потенційних кандидатів під час планування нового ігрового туру, використовуючи стандартні операції SQL-з'єднання (JOIN).

### 2.4 Розробка моделі безпеки та стратегії авторизації

Враховуючи те, що інформаційна система «RefMate» оперує персональними даними суддів (телефони, рівні ліцензій, особисті розклади), питання кібербезпеки та надійної авторизації є одним із найважливіших етапів проєктування. Оскільки архітектура застосунку передбачає розділення на клієнтську (SPA) та серверну (REST API) частини, традиційний механізм сесій із використанням файлів cookie (Stateful) стає неефективним і важко масштабованим.

Замість цього було прийнято рішення реалізувати архітектуру авторизації без збереження стану (Stateless Authentication). У такій парадигмі сервер взагалі не запам'ятовує, чи увійшов користувач у систему. Вся інформація, необхідна для ідентифікації та перевірки прав доступу, передається у кожному HTTP-запиті. Інструментом для реалізації цієї стратегії було обрано відкритий стандарт JSON Web Token (JWT) [13].

Процес автентифікації у розробленій системі складається з кількох логічних кроків. На першому етапі користувач (арбітр або адміністратор) вводить свої облікові дані (електронну пошту та пароль) на сторінці входу. Клієнтський застосунок надсилає їх у вигляді JSON-об'єкта POST-запитом на відповідний відкритий ендпоінт сервера.

Серверна частина, використовуючи модуль безпеки, порівнює введений пароль із криптографічним хешем, який зберігається у базі даних PostgreSQL. Для хешування застосовується алгоритм BCrypt з динамічною генерацією криптографічної солі [15]. Це гарантує, що навіть у разі гіпотетичної

компрометації бази даних зловмисники не зможуть відновити оригінальні паролі користувачів.

Якщо облікові дані збігаються, сервер генерує два JWT-токени: Access Token (для короткострокового доступу до ресурсів) та Refresh Token (для оновлення доступу). Access Token містить зашифроване інформаційне навантаження (Payload), куди зашито ідентифікатор користувача (ID), його роль (ADMIN або REFEREE) та час закінчення дії токена (Expiration Time). Цей токен підписується на сервері за допомогою секретного ключа з використанням алгоритму HMAC-SHA256, що унеможливлює його підробку або зміну на клієнтській стороні.

Сформовані токени повертаються клієнту і зберігаються у захищеному сховищі браузера. При кожному наступному запиті до захищених ресурсів (наприклад, спробі призначити бригаду або переглянути розклад) клієнтський застосунок додає Access Token до HTTP-заголовка Authorization: Bearer <token>. Серверна частина отримує запит, валідує цифровий підпис токена, перевіряє його строк дії і, лише якщо він дійсний, витягує з нього роль користувача. На основі цієї ролі Spring Security дозволяє або блокує доступ до конкретного REST-ендпоінта.

Такий підхід забезпечує не лише надійний захист від несанкціонованого доступу, а й високу швидкодію, оскільки серверу не потрібно щоразу звертатися до бази даних для перевірки статусу сесії користувача.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ REFMATE

## 3.1 Обґрунтування вибору технологічного стеку розробки

Створення надійного та масштабованого вебзастосунку вимагає ретельного підбору технологічного стеку. Враховуючи специфіку інформаційної системи «RefMate», яка передбачає обробку великої кількості логістичних даних, безпечну авторизацію та високу швидкість відгуку, було прийнято рішення базувати розробку на перевірених та корпоративних стандартах програмування[7].

### 3.1.1 Серверна частина: Java 21 та переваги екосистеми Spring Boot

Для реалізації серверної частини (Backend) було обрано мову програмування Java, а саме її актуальну версію — Java 21 [3]. Головною перевагою цієї версії є повноцінна підтримка віртуальних потоків (Virtual Threads), що дозволяє серверу обробляти тисячі одночасних HTTP-запитів без значного навантаження на оперативну пам'ять сервера. Це особливо актуально під час масової розсилки Email-сповіщень про призначення арбітрів.

Фундаментом для побудови REST API виступив фреймворк Spring Boot 3 [14]. Він забезпечує швидке розгортання мікросервісів та автоматичну конфігурацію базових компонентів (Auto-Configuration). Екосистема Spring ідеально підходить для розробки корпоративних вебсистем завдяки вбудованим модулям: Spring Data JPA для зручної роботи з базою даних, Spring Security для налаштування правил доступу та Spring Web для обробки вебзапитів[16]. Додатково було підключено інструмент Lombok, який за допомогою спеціальних анотацій (наприклад, @Data або @Builder) дозволив значно скоротити кількість шаблонного коду (геттерів, сеттерів, конструкторів) у доменних моделях бази даних.

### 3.1.2 Клієнтська частина: Vanilla JavaScript та концепція розробки без фреймворків

На противагу популярній тенденції використання важких фронтенд-фреймворків (на кшталт React чи Angular), клієнтську частину вебзастосунку «RefMate» було прийнято рішення реалізовувати виключно на базі класичних технологій: HTML5[17], CSS3[18] (з використанням препроцесора SCSS) та Vanilla JavaScript (ES6+)[4].

Такий підхід має кілька суттєвих переваг. По-перше, він забезпечує максимальну швидкодію роботи інтерфейсу на слабких мобільних пристроях, оскільки браузеру не потрібно завантажувати і парсити мегабайти бібліотечного коду. По-друге, розробка на чистому JavaScript дає повний контроль над DOM-деревом (об'єктною моделлю документа) та управлінням пам'яттю.

Взаємодія з сервером організована за допомогою сучасного стандарту Fetch API[8], який дозволяє асинхронно відправляти HTTP-запити у фоновому режимі та динамічно оновлювати вміст сторінки (концепція AJAX). Для стилізації інтерфейсу було використано методологію BEM (Block, Element, Modifier), що зробило CSS-код структурованим, ізольованим та незалежним від вкладеності HTML-тегів. Для інтерактивних діалогових вікон (спливаючих сповіщень про успіх чи помилку) інтегровано легку бібліотеку SweetAlert2[19], яка покращує користувацький досвід без перевантаження сторінки.

### 3.1.3 Засоби хмарного розгортання та контейнеризації

Оскільки система проєктувалася як веборієнтований сервіс, критично важливо було забезпечити її доступність у мережі Інтернет 24/7. Для вирішення цього завдання було використано сучасні хмарні платформи типу PaaS (Platform as a Service).

Для розміщення клієнтської частини (статичних файлів HTML, CSS, JS) обрано платформу Vercel, яка автоматично інтегрується з репозиторієм GitHub та забезпечує миттєве розгортання оновлень через глобальну мережу доставки

контенту (CDN). Серверну частину на базі Spring Boot розгорнуто на платформі Render. Перед деплоєм Java-додаток було скомпільовано у виконуваний JAR-файл, що дозволило Render автоматично підняти віртуальний контейнер і забезпечити стабільну роботу REST API. Реляційна база даних PostgreSQL розміщена на спеціалізованому хмарному сервісі Neon.tech, який пропонує безсерверну (serverless) архітектуру та високу швидкість обробки SQL-запитів.

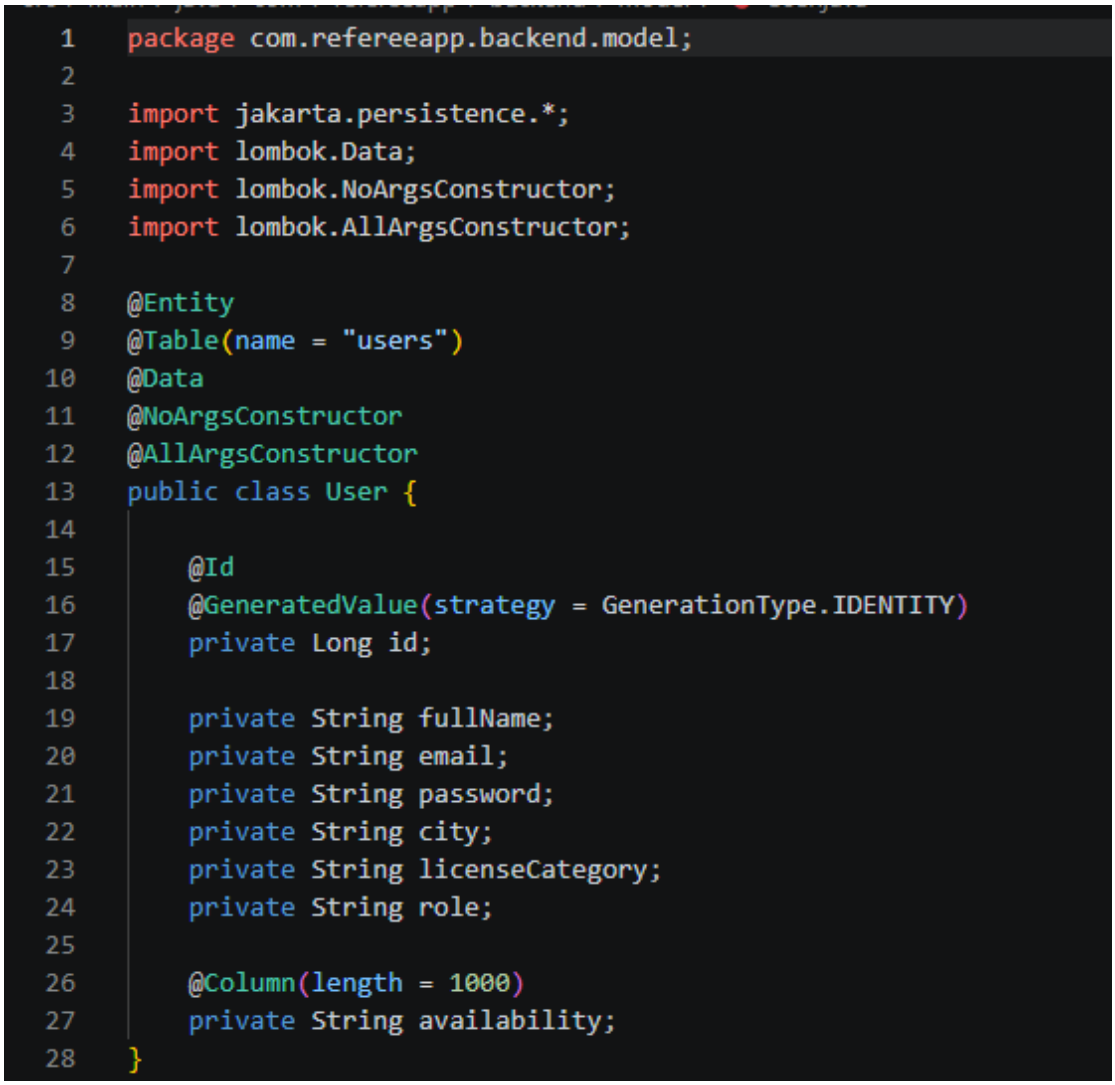
## 3.2 Реалізація серверної логіки (Backend)

Розробка серверної частини вебзастосунку «RefMate» велася за принципами багатошарової архітектури, де кожен пакет коду відповідає за конкретну задачу: взаємодію з даними, бізнес-логіку або обробку HTTP-мережі.

### 3.2.1 Побудова шару доступу до даних за допомогою Spring Data JPA та Hibernate

Обмін даними між Java-додатком та базою PostgreSQL реалізовано за допомогою специфікації JPA (Java Persistence API), а в ролі конкретного ORM-провайдера виступає Hibernate[16]. Це означає, що замість написання прямих SQL-запитів, розробник оперує звичайними Java-класами (сутностями), які автоматично мапляться на таблиці в базі даних (Рис. 3.1).

Для кожної сутності створено окремий інтерфейс-репозиторій, який успадковується від JpaRepository. Це дало змогу автоматично отримати базові методи (збереження, видалення, пошук по ID) без написання жодного рядка коду. Для більш складних вибірок (наприклад, пошуку арбітрів, які вільні у конкретну дату) використовувалися так звані Derived Queries — методи, назва яких автоматично генерує SQL-запит.



```java
package com.refereeapp.backend.model;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String fullName;
    private String email;
    private String password;
    private String city;
    private String licenseCategory;
    private String role;

    @Column(length = 1000)
    private String availability;
}
```

Рис. 3.1 Фрагмент програмного коду сутності бази даних із використанням JPA-анотацій

### 3.2.2 Розробка контролерів та реалізація REST-ендпоінтів

Взаємодія з клієнтським вебзастосунком відбувається через класи, помічені анотацією @RestController. Відповідно до архітектури проєкту, обробка мережевих запитів розділена між двома основними контролерами: UserController та MatchController (Рис.3.2).

Перший контролер (UserController) відповідає за управління обліковими записами, обробляючи запити на реєстрацію, отримання профілів арбітрів та перевірку їхньої доступності. Другий контролер (MatchController) бере на себе

всю логіку, пов'язану з турнірною сіткою: від створення нової гри до безпосереднього збереження суддівської бригади на конкретний матч.

Усі дані, які надходять з фронтенду, автоматично перетворюються з формату JSON у Java-об'єкти за допомогою вбудованих парсерів. Перед тим як передати інформацію на виконання до шару сервісів, дані проходять жорстку валідацію, що унеможливлює збереження в базу некоректних записів.

```
@PostMapping
public Match createMatch(@RequestBody Match match) {
    //  Встановлюємо статус PENDING для всіх призначених арбітрів
    if (match.getReferees() != null) {
        for (User ref : match.getReferees()) {
            match.getRefereeStatuses().put(ref.getId(), value: "PENDING");
        }
    }

    //  Зберігаємо матч у базу
    Match savedMatch = matchRepository.save(match);

    //  Розсилаємо листи всім призначеним суддям
    if (savedMatch.getReferees() != null) {
        for (User ref : savedMatch.getReferees()) {
            // Дістаємо повну інформацію про суддю з бази (щоб отримати його email)
            userRepository.findById(ref.getId()).ifPresent(fullReferee -> {
                emailService.sendMatchAssignmentEmail(
                        fullReferee.getEmail(),
                        fullReferee.getFullName(),
                        savedMatch.getTeamA(),
                        savedMatch.getTeamB(),
                        savedMatch.getDateTime().toString(),
                        savedMatch.getLocation()
                );
            });
        }
    }
```

Рис. 3.2 Фрагмент коду контролера MatchController для обробки вебзапитів

### 3.2.3 Програмна реалізація модуля безпеки (Spring Security, BCrypt)

Модуль безпеки є однією з найскладніших та найважливіших частин бекенду, яка повністю ізольована у пакеті security. Він базується на потужній екосистемі Spring Security[15]. Головним конфігураційним класом виступає SecurityConfig, де прописані правила доступу до вебресурсів: які ендпоінти є

відкритими (наприклад, для логіну), а які вимагають обов'язкової авторизації з правами адміністратора чи арбітра.

Для забезпечення архітектури без збереження стану (stateless) було розроблено спеціальний клас JwtUtil, який відповідає за криптографічну генерацію та валідацію токенів доступу[13]. Щоб система розуміла, хто саме робить запит, імплементовано JwtAuthenticationFilter. Цей фільтр перехоплює кожен вхідний вебзапит, шукає токен у HTTP-заголовках, розшифровує його і встановлює контекст безпеки для поточного користувача. Для зв'язування механізмів Spring Security з таблицею користувачів у базі даних PostgreSQL реалізовано класи CustomUserDetails та CustomUserDetailsService. Паролі у базі даних ніколи не зберігаються у відкритому вигляді — вони проходять одностороннє хешування алгоритмом BCrypt під час реєстрації.

### 3.2.4 Налаштування системи автоматизованих Email-сповіщень через SMTP

Для усунення ручної комунікації між організаторами та суддями у пакеті service було створено окремий клас EmailService (Рис.3.3). Цей модуль використовує бібліотеку Spring Boot Starter Mail для відправки електронних листів через стандартний протокол SMTP.

Логіка роботи цього сервісу тісно пов'язана з життєвим циклом матчу. Коли адміністратор через вебзастосунок успішно призначає арбітра на гру, після фіксації транзакції у базі даних викликається метод EmailService. Він динамічно формує персоналізований лист, підставляючи туди змінні з об'єкта Match (дату, час, локацію та назви команд), і відправляє його на електронну пошту фахівця (Рис.3.4). Асинхронна природа виконання цього процесу гарантує, що головний потік програми не блокується під час очікування відповіді від поштового сервера, завдяки чому вебінтерфейс реагує на дії адміністратора миттєво.

```
}

public void sendMatchAssignmentEmail(String toEmail, String refereeName, String teamA, String te
    try {
        SimpleMailMessage message = new SimpleMailMessage();

        // Вкажи тут ту саму пошту, що і в application.properties
        message.setFrom("jekaolhovskii@gmail.com");
        message.setTo(toEmail);
        message.setSubject("🏀 RefMate: Нове призначення на матч!");

        // Формуємо красивий текст листа
        String emailText = "Вітаємо, " + refereeName + "!\n\n" +
                "Вас призначено на новий баскетбольний матч у системі RefMate.\n\n" +
                "Деталі матчу:\n" +
                "Команди: " + teamA + " vs " + teamB + "\n" +
                "Дата і час: " + dateTime.replace(target: "T", replacement: " ") + "\n" +
                "Локація: " + location + "\n\n" +
                "Будь ласка, зайдіть у свій кабінет (розділ 'Мої призначення'), щоб підтвердити
                "З повагою,\nГоловна суддівська колегія";

        message.setText(emailText);
        mailSender.send(message);

        System.out.println("Лист успішно відправлено на пошту: " + toEmail);
```

Рис. 3.3 Фрагмент коду для реалізація сервісу автоматизованих Email-сповіщень

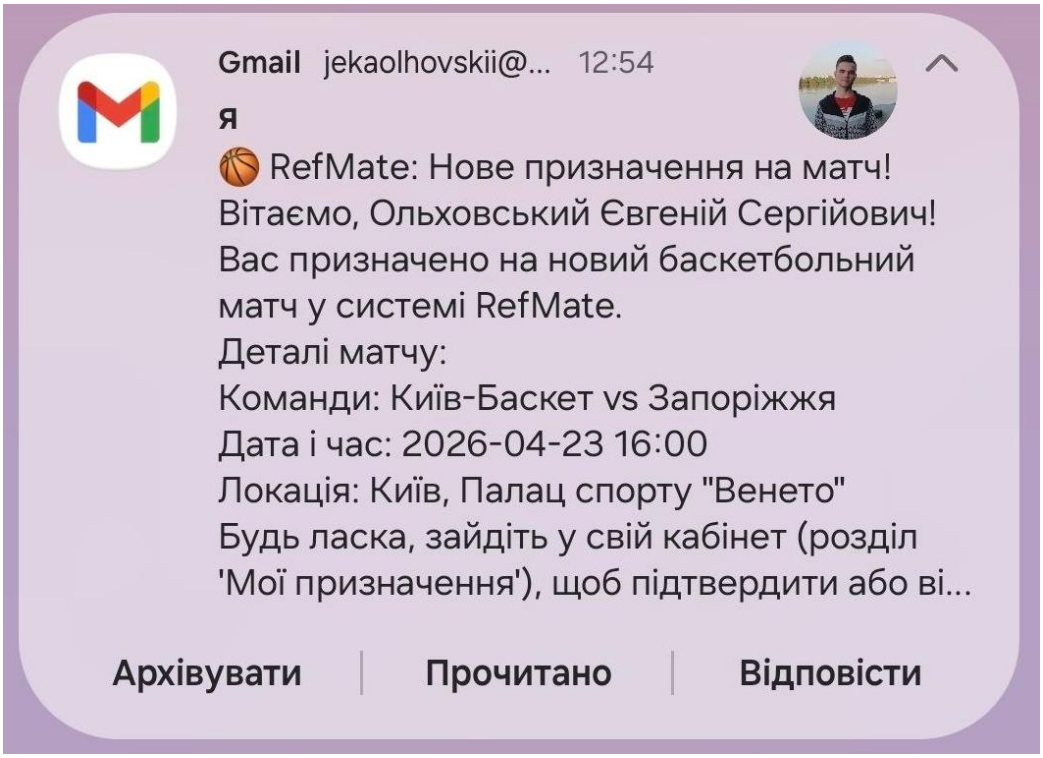



Рис. 3.4 Сповіщення у готовому вигляді, яке приходить на пошту

### 3.3 Розробка інтерфейсу користувача (Frontend)

Створення клієнтської частини вебзастосунку без використання важких реактивних фреймворків вимагає чіткої організації коду. Для інформаційної системи «RefMate» було побудовано легкий та високопродуктивний фронтенд, який логічно розділяє розмітку (HTML5), візуальне оформлення (CSS3) та логіку поведінки (Vanilla JS). Цей підхід забезпечує максимальну швидкість завантаження вебсторінок, оскільки браузер не витрачає ресурси на парсинг сторонніх бібліотек.

#### 3.3.1 Організація клієнтського коду та адаптивна верстка на CSS3

Для стилізації інтерфейсу було використано стандартні каскадні таблиці стилів (CSS) без застосування препроцесорів. Увесь візуальний код зібрано у файлі style.css. Структура стилів базується на логічному іменуванні класів, що дозволяє легко ідентифікувати елементи на сторінці (наприклад, login__container, form__input, button--primary).

Особливу увагу під час розробки було приділено адаптивності вебінтерфейсу (Responsive Web Design), адже користувачі (особливо арбітри) часто використовують мобільні пристрої. Для гнучкого позиціонування елементів широко застосовується технологія Flexbox (display: flex). Наприклад, центрування форм логіну та реєстрації на екрані досягається за допомогою властивостей justify-content та align-items.

Щоб забезпечити коректне відображення панелі управління (Dashboard) на невеликих екранах смартфонів, у CSS-коді прописані медіа-запити (@media (max-width: 768px)) (Рис.3.5). Вони динамічно змінюють напрямок блоків (flex-direction: column), зменшують відступи та розтягують кнопки на всю ширину екрана. Крім того, для відображення великих обсягів турнірних даних реалізовано клас table-responsive, який додає горизонтальну прокрутку для таблиць на мобільних пристроях, зберігаючи цілісність макета.

```css
@media (max-width: 768px) {
    .header {
        flex-direction: column;
        gap: 15px;
        padding: 1rem;
    }

    .header__user {
        flex-wrap: wrap;
        justify-content: center;
    }

    .dashboard-content {
        padding: 1rem;
    }

    .form-row {
        flex-direction: column;
        gap: 10px;
    }

    .button--small {
        width: 100%;
        text-align: center;
    }

    .dashboard-card {
        padding: 1rem;
    }
}
```

Рис. 3.5 Фрагмент CSS-коду для забезпечення адаптивності вебінтерфейсу

### 3.3.2 Асинхронна взаємодія з REST API та маніпуляції з DOM-деревом

Логіка клієнтської частини реалізована мовою JavaScript (ES6+)[9]. Оскільки система працює за клієнт-серверною архітектурою, обмін даними між веббраузером та бекендом (Spring Boot) відбувається в асинхронному режимі за допомогою нативного інтерфейсу Fetch API[8]. Усі скрипти підключені безпосередньо до HTML-документів (наприклад, app.js для логіну та register.js для реєстрації).

Взаємодія з користувачем починається з перехоплення подій форми. Наприклад, під час натискання кнопки «Увійти» або «Зареєструватися»,

JavaScript-код зупиняє стандартну поведінку браузера (e.preventDefault()), зчитує значення з відповідних полів вводу (через document.getElementById) і формує JSON-об'єкт. Цей об'єкт передається у Fetch-запит методом POST на відповідний ендпоінт сервера.

Під час виконання запитів до захищених ресурсів (наприклад, отримання списку матчів), скрипт обов'язково витягує JWT-токен із локального сховища браузера (localStorage) та додає його до HTTP-заголовка Authorization: Bearer. Після отримання відповіді від сервера у форматі JSON, JavaScript динамічно модифікує DOM-дерево[9]. Для таблиць з розкладом скрипт програмно створює нові рядки та комірки, наповнює їх даними та інтегрує у вебсторінку.

### 3.3.3 Інтерактивні елементи та обробка користувацьких подій

Сучасні вебзастосунки вимагають високого рівня інтерактивності. Для інформування користувача про помилки валідації (наприклад, невірно введений пароль) безпосередньо у HTML-розмітці передбачені спеціальні контейнери (div class="form__error"), куди JavaScript динамічно вписує текст помилки.

Для реалізації більш складних діалогових вікон та спливаючих сповіщень про успішні дії у проєкт було інтегровано легковагову бібліотеку SweetAlert2 (підключену через CDN). Цей інструмент дозволив створити кастомні модальні вікна, які органічно вписуються у загальний дизайн вебінтерфейсу (Рис.3.6). Наприклад, успішна реєстрація нового арбітра супроводжується красивим візуальним підтвердженням від SweetAlert2, після чого відбувається автоматичний редирект на сторінку входу.

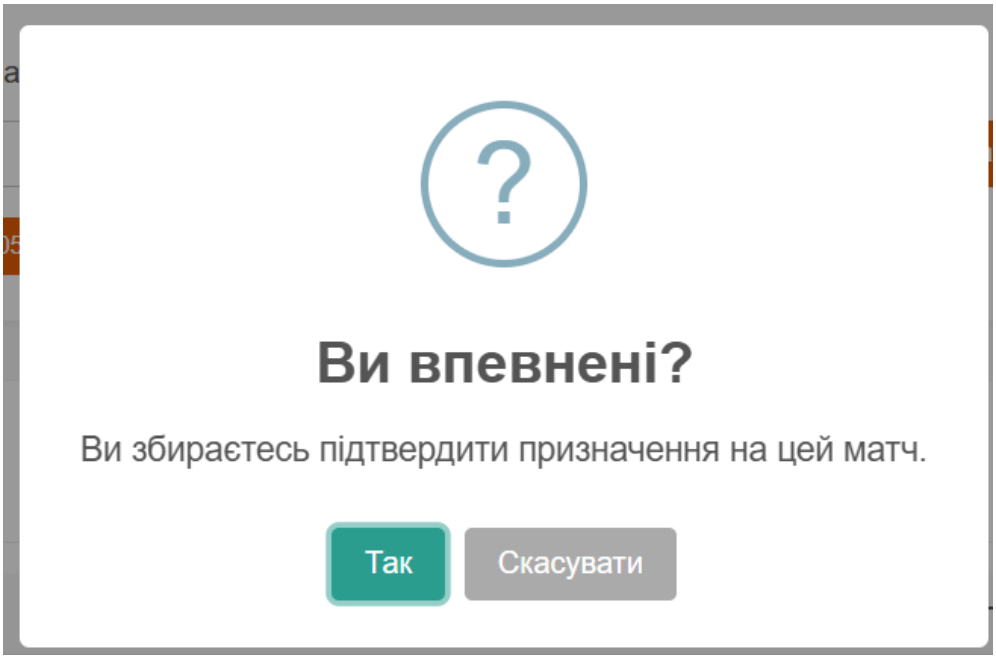



Рис 3.6 Приклад діалогового вікна, реалізованого за допомогою SweetAlert2

### 3.4 Опис екранних форм вебзастосунку

Для підтвердження працездатності розробленої системи та демонстрації її функціоналу було створено ключові екранні форми. Увесь інтерфейс спроєктовано з фокусом на мінімалізм, відсутність зайвих візуальних деталей та інтуїтивну зрозумілість елементів управління, що є критичним для користувачів із різним рівнем технічної грамотності.

#### 3.4.1 Екранні форми авторизації та реєстрації користувачів

Перше, з чим стикається користувач під час роботи з системою — це сторінка авторизації. Вона містить класичну форму для вводу електронної пошти та пароля. Дизайн виконано у світлих тонах із застосуванням акцентного помаранчевого кольору для кнопок і посилань, що візуально асоціюється з баскетбольною тематикою (Рис.3.9).

Сторінка реєстрації має значно складнішу логічну структуру. Вона збирає розширені персональні дані: прізвище, ім'я, по батькові, місто проживання, категорію суддівської ліцензії (від Національної до категорії D) та безпосередню роль у системі (Арбітр або Головний суддя) (Рис.3.7). Важливою архітектурною

особливістю цієї форми є використання динамічного вебінтерфейсу на базі JavaScript. Якщо користувач обирає у випадаючому списку роль адміністратора, клієнтський скрипт миттєво змінює CSS-властивість відображення для прихованого блоку, активуючи додаткове поле для вводу секретного ключа ліги (Рис.3.8). Це є першим базовим рубежем захисту від несанкціонованої реєстрації керівного складу.

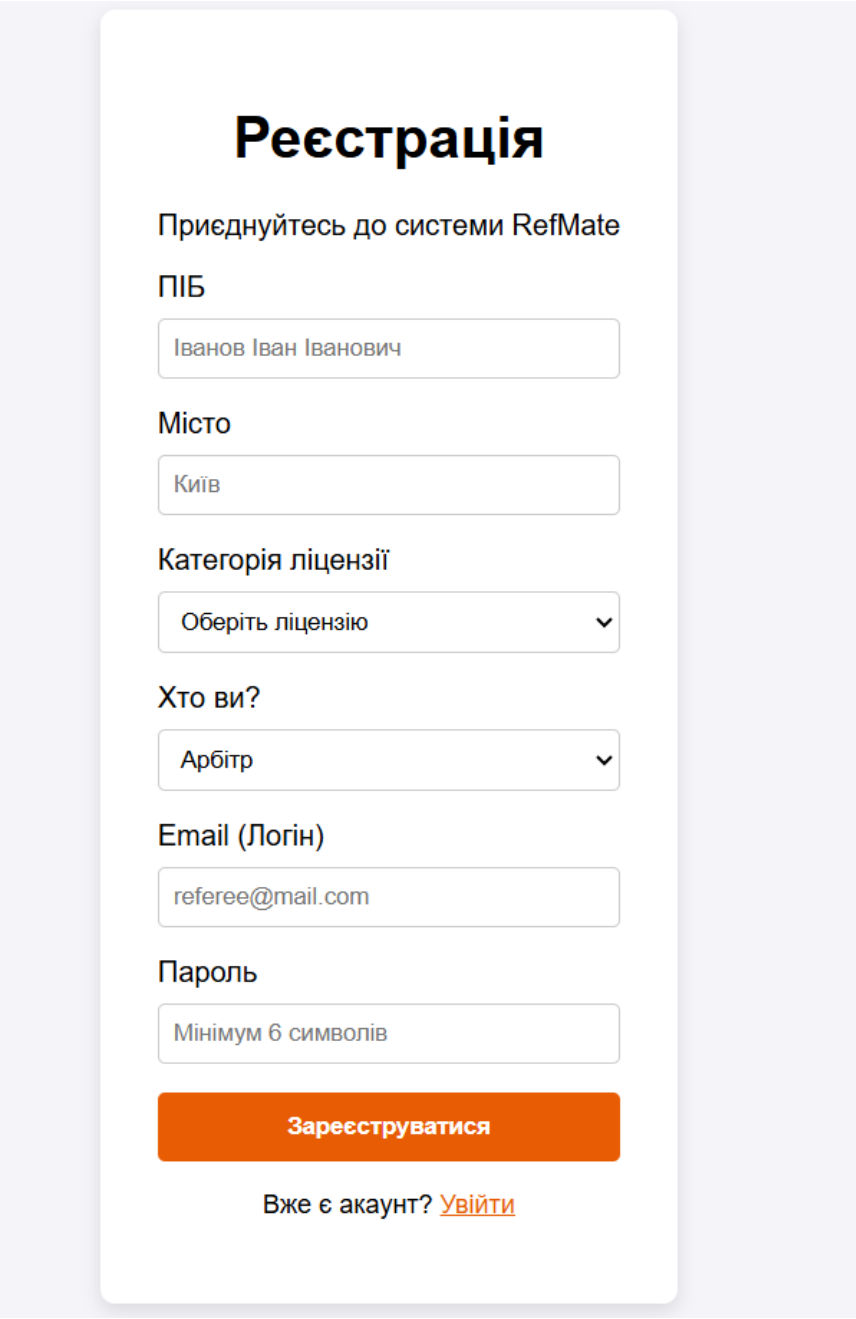



Рис. 3.7 Форма реєстрації

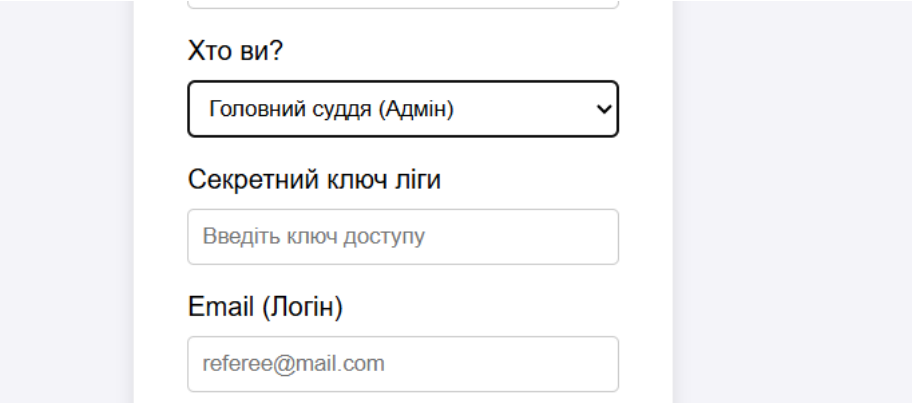



Рис. 3.8 Демонстрація можливості ввести ключ доступу, при реєстрації як адміністративної системи

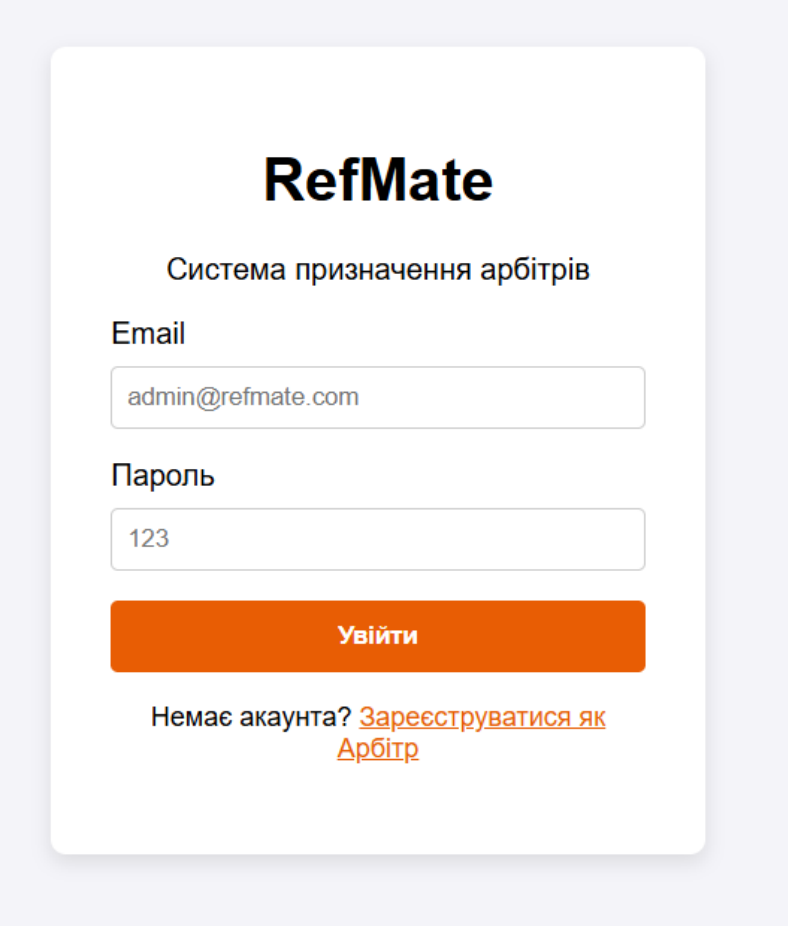



Рис. 3.9 Форма авторизації

### 3.4.2 Головна панель (Dashboard) та управління розкладом

Після успішної перевірки облікових даних серверною частиною, користувач перенаправляється на головну панель управління (Dashboard). Цей вебінтерфейс складається з навігаційного блоку у верхній частині сторінки, де

відображається логотип системи, ім'я поточного користувача з відповідним бейджем ролі та кнопка виходу, а також основної робочої області (Рис.3.10).

Уся турнірна інформація організована у вигляді ізольованих карток. Центральним елементом виступає таблиця матчів. Для адміністратора ця таблиця відображає весь розклад ліги і містить інтерактивні елементи управління для редагування даних та призначення суддівських бригад. Як було зазначено у попередніх підрозділах, для забезпечення комфортної роботи на мобільних пристроях таблиця розміщена у спеціальному контейнері з підтримкою горизонтальної прокрутки, що зберігає цілісність макета сторінки навіть за наявності великої кількості колонок.

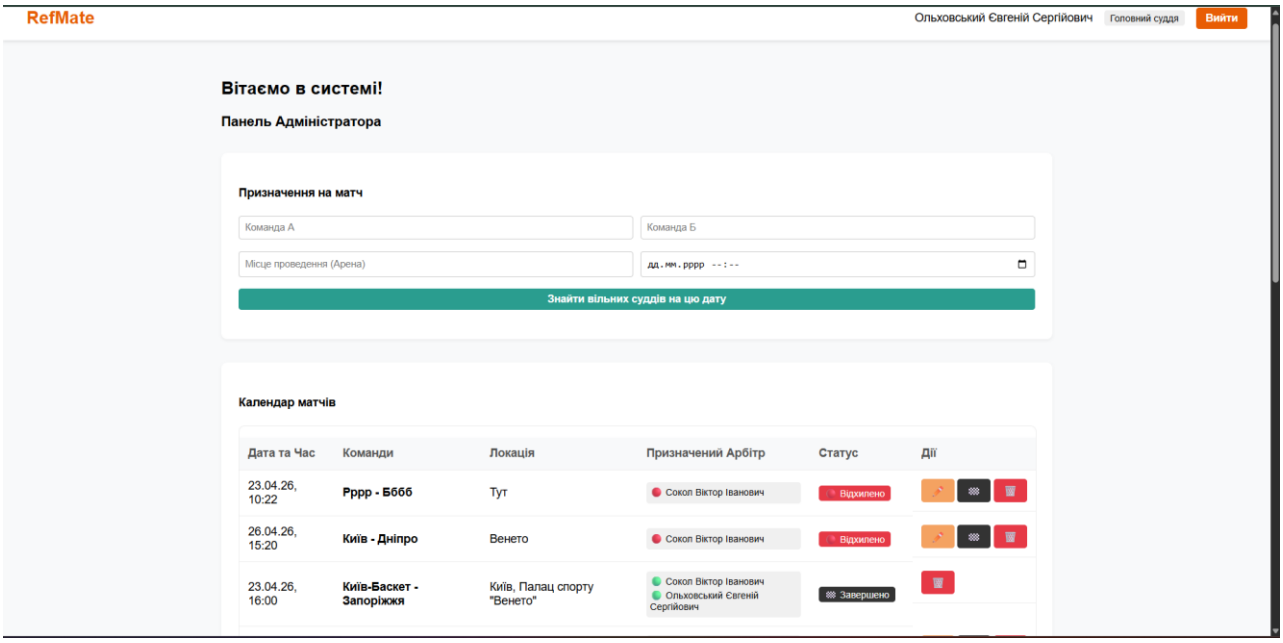



Рис. 3.10 Інтерфейс головної панелі управління з таблицею розкладу

### 3.4.3 Модуль призначення арбітрів та особистий кабінет фахівця

Найважливішою функціональною частиною вебзастосунку є модуль формування бригади на конкретний матч. Цей інтерфейс реалізовано за допомогою стандартних HTML-елементів випадаючих списків, які відповідають за вибір Старшого судді та двох його асистентів. Ці списки наповнюються даними динамічно: сервер повертає лише тих фахівців, які не мають конфліктів у розкладі. Таким чином, система на програмному рівні страхує адміністратора від створення логістичних накладок (Рис.3.12).

Зі свого боку, вебінтерфейс особистого кабінету арбітра сфокусований виключно на його персональних завданнях. Арбітр бачить відфільтровану таблицю лише зі своїми іграми (Рис.3.11). Додатково на цій сторінці розміщено елементи управління доступністю, за допомогою яких фахівець самостійно фіксує в базі даних дні, коли він не може обслуговувати матчі. Це мінімізує необхідність телефонних дзвінків та повністю автоматизує процес збору інформації про вільний персонал.

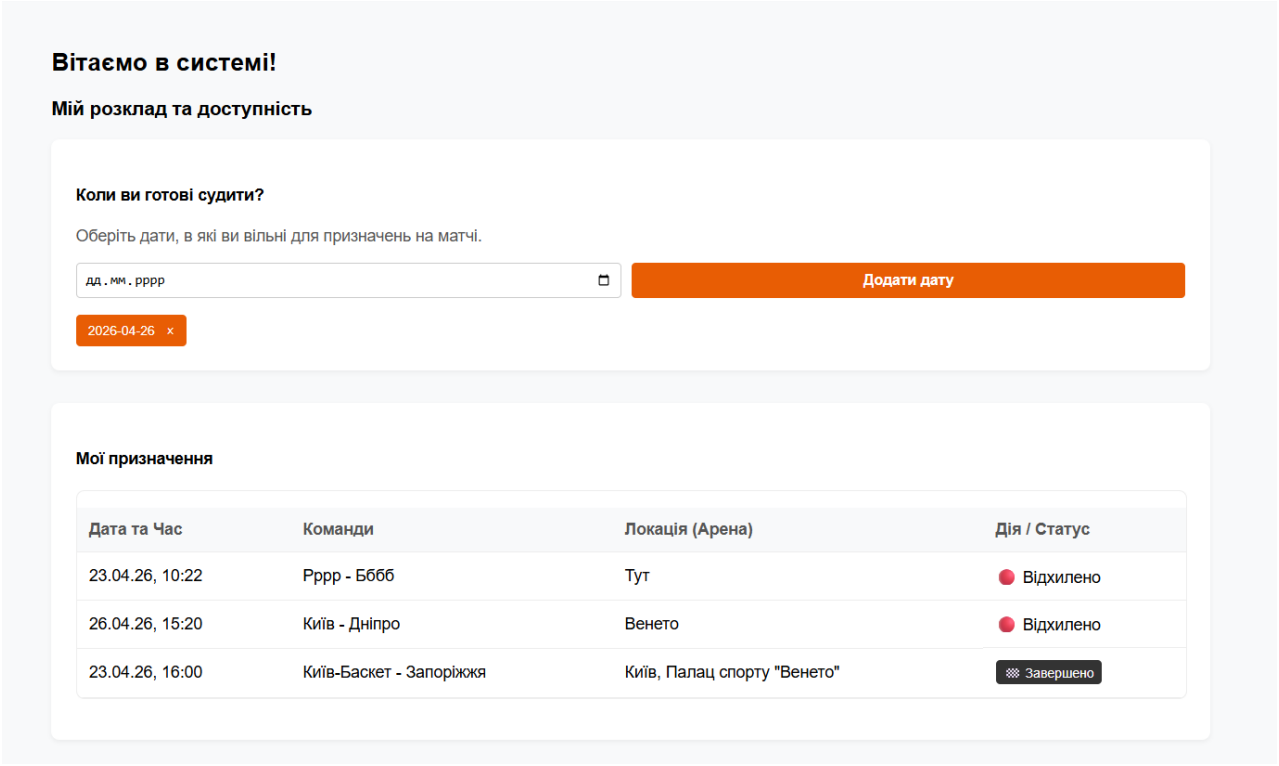



Рис. 3.11 Інтерфейс особистого кабінету арбітра

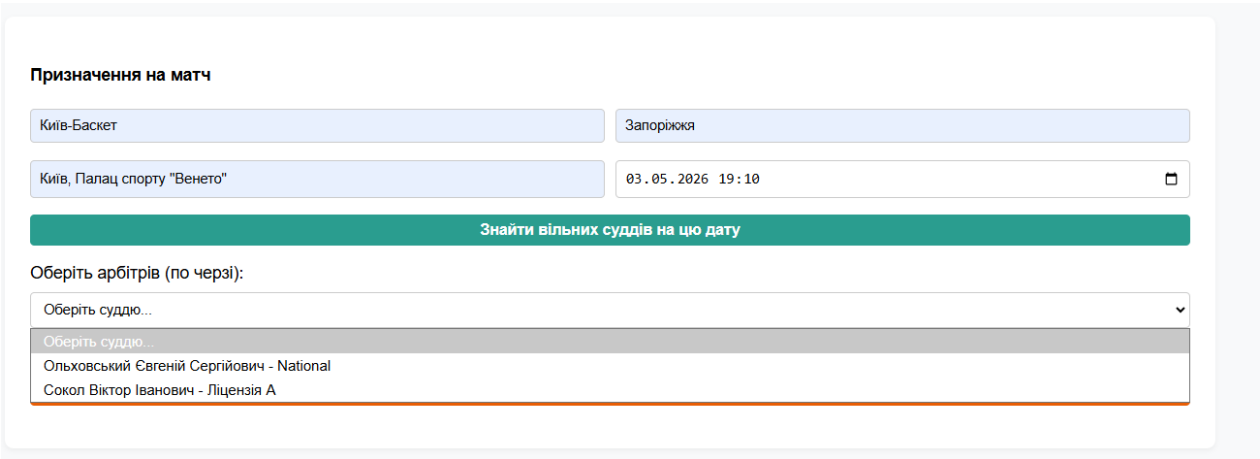



Рис. 3.12 Інтерфейс призначення бригад арбітрів на ігри

### 3.4.4 Програмна реалізація клієнтських скриптів та алгоритм авторизації

Логіка взаємодії клієнтської частини з сервером реалізована через ізольовані JavaScript-файли. Для демонстрації загального архітектурного підходу до написання клієнтських скриптів доцільно детально розглянути алгоритм обробки авторизації користувача, який міститься у базовому файлі app.js. Цей сценарій охоплює всі ключові аспекти фронтенд-розробки: перехоплення подій об'єктної моделі документа (DOM), асинхронну мережеву взаємодію, управління станом та обробку виняткових ситуацій (Рис.3.13).

Виконання скрипта починається з прослуховування події DOMContentLoaded, що гарантує повне завантаження HTML-розмітки перед початком будь-яких маніпуляцій. При спробі користувача відправити форму входу, стандартна поведінка веббраузера (перезавантаження сторінки) блокується за допомогою методу preventDefault(). Натомість скрипт зчитує введені значення електронної пошти та пароля, паралельно активуючи візуальний індикатор завантаження через бібліотеку SweetAlert2, щоб користувач розумів, що процес обробки даних розпочався.

Головна логіка інкапсульована в асинхронному блоці try...catch. Скрипт формує HTTP-запит методом POST до розгорнутого у хмарі бекенду (на платформі Render), передаючи облікові дані у форматі JSON-строки. Якщо серверна частина успішно валідує користувача та повертає статус 200 OK, клієнтський застосунок витягує з відповіді криптографічний JWT-токен та об'єкт з даними користувача.

Критично важливим етапом є збереження цих даних у локальному сховищі браузера (localStorage). Саме наявність токена у цьому сховищі дозволяє системі підтримувати архітектуру без збереження стану (stateless): під час усіх наступних переходів по сторінках вебзастосунку інші скрипти будуть автоматично додавати цей токен до заголовків запитів. Після успішного збереження даних відбувається примусове перенаправлення на сторінку головної панелі (dashboard.html).

У випадку, якщо сервер відхиляє авторизацію (наприклад, через невірний пароль) або якщо виникає помилка мережевого з'єднання (сервер тимчасово недоступний), виконання коду переходить до блоку обробки помилок. Замість виведення технічної інформації у консоль, скрипт генерує зрозумілі модальні вікна SweetAlert2 із відповідними повідомленнями, зберігаючи високий рівень користувацького досвіду (UX). Усі інші інтерактивні модулі системи, такі як реєстрація чи призначення суддівських бригад, побудовані за аналогічним патерном асинхронної взаємодії.

```
document.addEventListener('DOMContentLoaded', () => {
    const loginForm = document.getElementById('loginForm');

    if (loginForm) {
        loginForm.addEventListener('submit', async (e) => {
            e.preventDefault();

            const email = document.getElementById('email').value;
            const password = document.getElementById('password').value;

            Swal.fire({
                title: 'Вхід...',
                allowOutsideClick: false,
                didOpen: () => { Swal.showLoading(); }
            });

            try {
                const response = await fetch('https://refmate-api.onrender.com/api/users/login', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ email: email, password: password })
                });

                if (response.ok) {
                    const data = await response.json();
                    localStorage.setItem('jwtToken', data.token);
                    localStorage.setItem('currentUser', JSON.stringify(data.user));

                    Swal.close();
                    window.location.href = 'dashboard.html';
```

Рис. 3.13 Фрагмент програмного коду обробки процесу авторизації користувача

## 3.5 Методика тестування вебзастосунку

Заключним етапом програмної реалізації інформаційної системи «RefMate» стало комплексне тестування, необхідне для підтвердження надійності архітектури та коректності роботи бізнес-логіки. Враховуючи специфіку предметної області, де будь-яка логістична помилка може призвести до зриву баскетбольного матчу, перевірка проводилася на кількох рівнях: від візуального відображення до стрес-тестів серверних алгоритмів.

### 3.5.1 Тестування інтерфейсу користувача та кросбраузерна сумісність

Тестування клієнтської частини (Frontend) проводилося за сценаріями «чорного ящика», імітуючи реальну поведінку адміністраторів та арбітрів. Основний фокус був спрямований на перевірку адаптивності вебінтерфейсу, оскільки цільова аудиторія часто використовує смартфони безпосередньо у спортивних залах.

За допомогою інструментів розробника (Google Chrome DevTools) симулювалася роздільна здатність різних мобільних пристроїв. Результати підтвердили, що медіа-запити у CSS відпрацьовують коректно: блоки форм шикуються у колонку, а таблиця матчів успішно активує горизонтальну прокрутку (клас table-responsive), не руйнуючи загальний макет сторінки (Рис.3.14).

Окремим етапом стала перевірка обробки виняткових ситуацій на стороні клієнта. Моделювалися сценарії введення некоректних даних під час реєстрації та спроби авторизації з недійсним паролем. Тестування показало, що JavaScript-код успішно перехоплює HTTP-помилки від сервера та коректно ініціює виклик бібліотеки SweetAlert2, відображаючи користувачеві інформативні червоні модальні вікна замість системних збоїв.

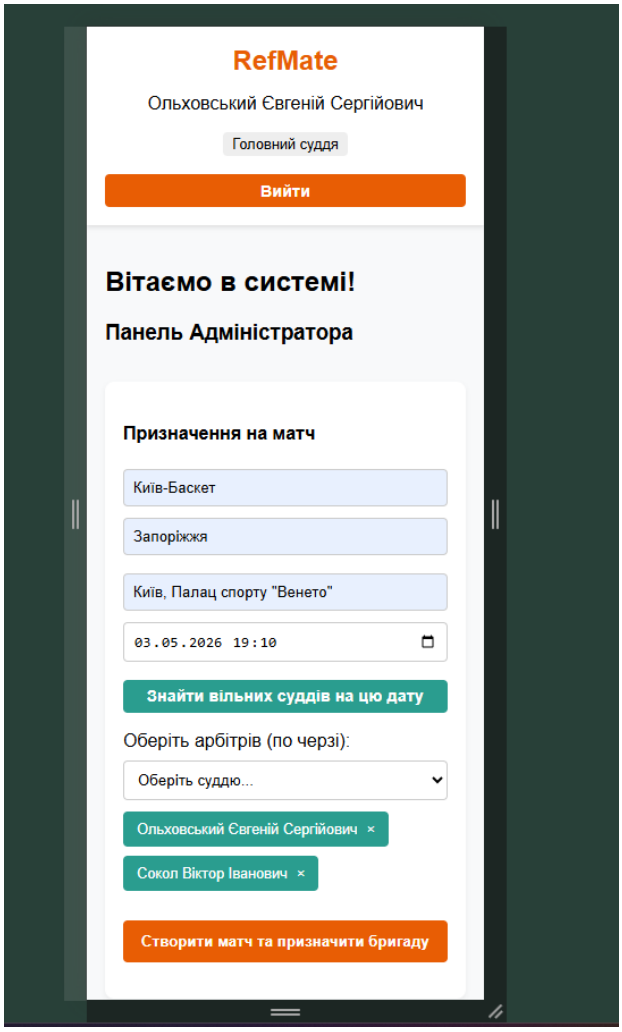



Рис. 3.14 Перевірка адаптивності вебінтерфейсу за допомогою інструментів розробника

### 3.5.2 Тестування REST API та бізнес-логіки серверної частини

Для тестування серверної частини (Backend) та перевірки коректності роботи кінцевих точок (endpoints) без необхідності взаємодії з клієнтським інтерфейсом було використано стандарт OpenAPI [11]. У середовищі Spring Boot цей функціонал реалізовано за допомогою конфігураційного класу OpenApiConfig, який автоматично генерує інтерактивну вебдокументацію за допомогою інструменту Swagger UI.

Цей підхід дозволив проводити модульне тестування бізнес-логіки безпосередньо з вікна браузера. У спеціальному інтерфейсі Swagger відображаються всі доступні REST-контролери (наприклад, UserController та MatchController) та методи HTTP-запитів до них.

Під час тестування перевірялася надійність механізму безпеки (Spring Security) [15]. Використовуючи вбудовану в Swagger функцію авторизації, тестувалася передача JWT-токена у заголовку запиту. Якщо запит на захищений маршрут (наприклад, створення матчу) відправлявся без токена, сервер коректно повертав статус 401 Unauthorized, що підтвердило надійність захисту бекенду.

Окремо тестувалися алгоритми доступу до бази даних PostgreSQL. Через інтерфейс Swagger відправлялися POST-запити з тестовими JSON-об'єктами для імітації призначення арбітрів (Рис.3.15). Перевірялося, чи коректно сервер обробляє ці дані, чи не створюються дублікати записів у базі та чи відпрацьовує перехоплення виняткових ситуацій (наприклад, якщо передано невалідний формат дати). Завершальним етапом стала перевірка модуля EmailService — після успішного виконання запиту через Swagger фіксувався факт автономної доставки електронного листа на тестову поштову скриньку.

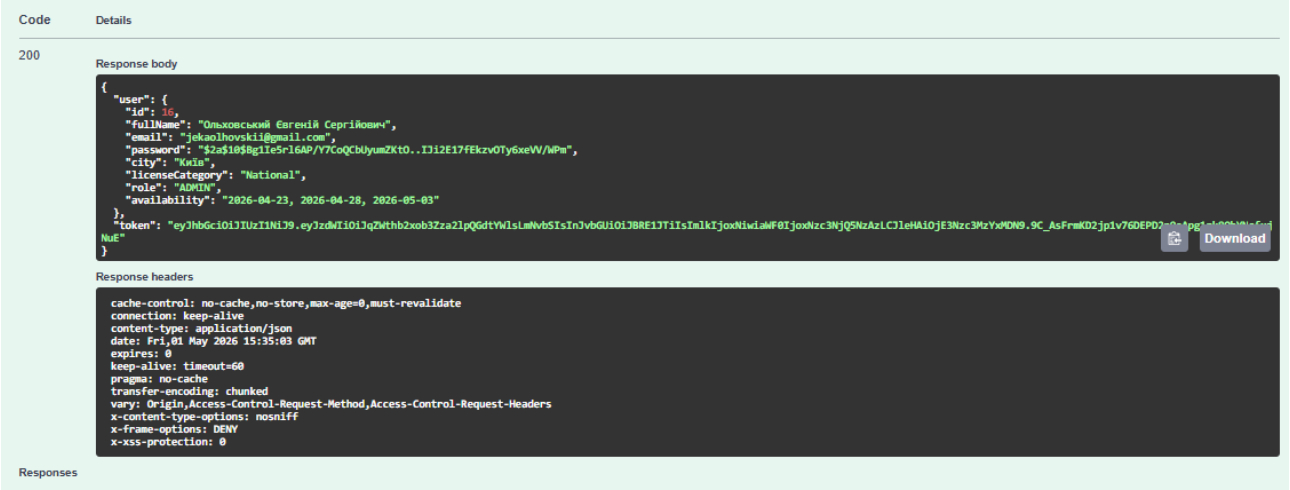



Рис. 3.15 Результат інтерактивного тестування ендпоінту авторизації у Swagger UI

### 3.5.3 Автоматизоване модульне тестування бізнес-логіки (Unit Testing)

Для забезпечення високої стабільності серверної частини та ізольованої перевірки архітектурних алгоритмів було застосовано підхід автоматизованого модульного тестування (Unit Testing). Написання тестових сценаріїв здійснювалося за допомогою сучасного фреймворку JUnit 5 [6] у поєднанні з бібліотекою Mockito (Рис.3.16).

Головна перевага такого підходу полягає у використанні механізму імітації (Mocking) за допомогою анотацій @Mock та @InjectMocks. Це дозволило тестувати логіку контролерів в абсолютній ізоляції від бази даних PostgreSQL та зовнішніх мережевих запитів. Наприклад, у класі UserControllerTest було покрито автоматичними тестами критичні аспекти безпеки: перевірялося, чи коректно викликається залежність PasswordEncoder для одностороннього хешування пароля користувача перед його передачею до репозиторію. Окремим надзвичайно важливим сценарієм стала перевірка захисту від несанкціонованого доступу: тест програмно імітує спробу реєстрації користувача з роллю «ADMIN» за допомогою невірного секретного ключа ліги. Завдяки методу assertThrows успішно підтверджується, що система перехоплює цю спробу і генерує відповідну виняткову ситуацію (RuntimeException).

У класі MatchControllerTest модульне тестування було застосовано для перевірки складного ланцюжка подій під час призначення арбітрів. Під час симуляції створення нового матчу інструмент Mockito не лише імітує збереження об'єкта, але й відстежує поведінку суміжних сервісів. За допомогою конструкції verify(emailService, times(1)) тестовий сценарій автоматично підтверджує, що метод відправки електронного листа був викликаний рівно один раз і прийняв правильні аргументи (Email арбітра, назви команд, локацію). Це гарантує безперебійну роботу логіки системи сповіщень без необхідності здійснювати реальну SMTP-розсилку під час виконання автоматизованих тестів.

```
    @Test
    void createUser_WithAdminRole_AndWrongKey_ShouldThrowException() {
        User hacker = new User();
        hacker.setRole("ADMIN");

        Exception exception = assertThrows(RuntimeException.class, () -> {
            userController.createUser(hacker, "WRONG-KEY");
        });

        assertEquals("Неправильний секретний ключ адміністратора!",
exception.getMessage());
```

Рис. 3.16 Фрагмент програмного коду модульного тестування логіки безпеки за допомогою JUnit 5

# ВИСНОВКИ

У кваліфікаційній бакалаврській роботі вирішено актуальне науково-практичне завдання, яке полягає у розробці веборієнтованої інформаційної системи для автоматизації логістичних процесів та супроводу діяльності суддівського апарату в баскетболі.

За результатами проведеного дослідження та програмної реалізації проєкту зроблено наступні висновки:

1. Здійснено комплексний системний аналіз бізнес-процесів призначення спортивних арбітрів. Виявлено критичні недоліки існуючих ручних методів планування (залежність від месенджерів та електронних таблиць), що обґрунтувало потребу у створенні спеціалізованого програмного рішення для усунення логістичних накладок.

2. Сформовано перелік функціональних та нефункціональних вимог до інформаційної системи. Спроєктовано клієнт-серверну архітектуру застосунку, яка гарантує високу продуктивність, масштабованість та можливість незалежного обслуговування серверної і клієнтської частин програмного комплексу.

3. Розроблено та оптимізовано реляційну модель бази даних під управлінням СУБД PostgreSQL. Створено надійну структуру взаємопов'язаних таблиць для зберігання персональних даних персоналу, турнірних розкладів та індивідуальних графіків доступності арбітрів із забезпеченням цілісності інформації.

4. Спроєктовано та імплементовано серверну частину (Backend) з використанням мови Java 21 та екосистеми Spring Boot. Створено захищений REST API для обробки мережевих запитів. Завдяки інтеграції Spring Security та використанню JWT-токенів забезпечено багаторівневий захист даних і чітке розмежування прав доступу між адміністраторами ліги та арбітрами.

5. Розроблено алгоритм автоматизованого підбору суддівських бригад, який унеможливлює призначення фахівця за наявності конфліктів у його

розкладі. Додатково імплементовано програмний модуль для автоматичної генерації та розсилки персоналізованих електронних сповіщень через протокол SMTP

6. Створено інтерактивний інтерфейс користувача (Frontend), побудований на базі нативних вебтехнологій (HTML5, CSS3, Vanilla JavaScript) без використання надлишкових фреймворків. Забезпечено повну адаптивність вебсторінок для роботи на мобільних пристроях та налаштовано асинхронний обмін даними з сервером за допомогою Fetch API.

7. Проведено комплексне тестування програмного забезпечення через автоматизовані модульні тести (JUnit 5, Mockito) та перевірку кінцевих точок за допомогою Swagger UI. Успішне проходження всіх розроблених тест-кейсів та отримання очікуваних HTTP-відповідей підтвердили коректність роботи бізнес-логіки та відповідність функціоналу вимогам технічного завдання.

8. Робота пройшла апробацію :

1. Ольховський Є.С., Довженко Т.П. Визначення вимог до веборієнтованої інформаційної системи супроводу суддівської діяльності в баскетболі. Матеріали II Всеукраїнської науково-технічної конференції « Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез.- К.:ДУІКТ. С. 84-86.

2. Ольховський Є.С., Довженко Т.П. Аналіз та проєктування інформаційної системи для координації суддівського персоналу в спорті. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез.- К.:ДУІКТ. С.343-347.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційні правила баскетболу 2024 [Електронний ресурс] Міжнародна федерація баскетболу (FIBA). – URL: https://fbu.ua/dokumenty (дата звернення: 01.05.2026).

2. Посібник для арбітрів з баскетболу: Механіка суддівства [Електронний ресурс] FIBA Referee Operations. – URL: https://refereeing.fiba.basketball/en/game-official-licensing (дата звернення: 01.05.2026).

3. Evans B., Flanagan D. Java in a Nutshell: A Desktop Quick Reference. 8th ed. O'Reilly Media, 2023. 550 p.

4. Flanagan D. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020. 706 p.

5. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.

6. JUnit 5 User Guide [Електронний ресурс] JUnit. – URL: https://junit.org/junit5/docs/current/user-guide/ (дата звернення: 01.05.2026).

7. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008. 464 p.

8. MDN Web Docs: Fetch API [Електронний ресурс] Mozilla Foundation. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 29.04.2026).

9. MDN Web Docs: JavaScript [Електронний ресурс] Mozilla Foundation. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 30.04.2026).

10. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 612 p.

11. OpenAPI Specification [Електронний ресурс] Swagger. – URL: https://swagger.io/specification/ (дата звернення: 01.05.2026).

12. PostgreSQL 16 Documentation [Електронний ресурс] The PostgreSQL Global Development Group. – URL: https://www.postgresql.org/docs/16/index.html (дата звернення: 28.04.2026).

13. RFC 7519: JSON Web Token (JWT) [Електронний ресурс] IETF. – URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 01.05.2026).

14. Spring Boot Reference Documentation [Електронний ресурс] Spring. – URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 01.05.2026).

15. Spring Security Reference [Електронний ресурс] Spring. – URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 01.05.2026).

16. Walls C. Spring in Action. 6th ed. Manning Publications, 2022. 520 p.

17. HTML Standard [Електронний ресурс] WHATWG. – URL: https://html.spec.whatwg.org/ (дата звернення: 01.05.2026).

18. CSS: Cascading Style Sheets [Електронний ресурс] MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 01.05.2026).

19. SweetAlert2: Beautiful, responsive popup boxes [Електронний ресурс]. – URL: https://sweetalert2.github.io/ (дата звернення: 01.05.2026).

20. Unified Modeling Language (UML) [Електронний ресурс] Object Management Group. – URL: https://www.omg.org/spec/UML/ (дата звернення: 01.05.2026).

# ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

## Розробка веборієнтованої інформаційної системи для супроводу суддівської діяльності в баскетболі

Виконав студент 4 курсу
групи ПД-42
Ольховський Євгеній Сергійович
Керівник роботи
кандидат технічних наук, доцент Довженко Тимур Павлович

Київ - 2026

## МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета роботи** - спрощення процесу управління суддівської діяльності шляхом розробки веборієнтованої інформаційної системи планування та призначення арбітрів на матчі.

**Об'єкт дослідження -** процес управління суддівською діяльностю в баскетболі.

**Предмет дослідження** – методи, програмні засоби та вебтехнології для автоматизації управління суддівською діяльністю.

# ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести огляд предметної області, проаналізувати існуючі системи спортивного менеджменту та виявити недоліки ручного керування розкладами суддів.
2. Сформулювати функціональні та нефункціональні вимоги до розроблюваної інформаційної системи, що дозволить визначити технічні характеристики та критерії успішності готового продукту
3. Спроєктувати клієнт-серверну архітектуру вебзастосунку, модель безпеки та логічну структуру бази даних для зберігання турнірної інформації, що забезпечить цілісність, швидку обробку та надійний захист даних користувачів.
4. Розробити серверну логіку системи, що включає алгоритм автоматичного підбору вільних арбітрів, сервіс розсилки електронних сповіщень та модуль захищеної авторизації.
5. Створити адаптивний інтерфейс користувача із жорстким розмежуванням прав доступу для різних ролей.
6. Провести комплексне тестування розробленого програмного забезпечення для перевірки коректності роботи клієнтської та серверної частин.



# АНАЛІЗ АНАЛОГІВ

| Критерій порівняння | Excel + Месенджери | Assignr | ArbiterSports | HorizonWebRef | RefMate |
|---|---|---|---|---|---|
| **Безкоштовність** | + | - | - | - | + |
| **Українська локалізація** | + | - | - | - | + |
| **Автоматизація Email-сповіщень** | - | + | + | + | + |
| **Контроль конфлікту розкладів** | - | + | + | + | + |
| **Адаптивний UI/UX** | - | + | + | - | + |
| **Спрощена процедура призначення** | - | +/- | - | - | + |
| **Трекінг статусу в реальному часі** | - | +/- | + | - | + |

# ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Розділення прав доступу та забезпечення персональних кабінетів (Адміністратор / Арбітр).
2. Створення матчів з автоматичною фільтрацією спортивного персоналу за критеріями вільних дат у графіку та ліцензії.
3. Генерація та відправка email-сповіщень про призначення на гру в реальному часі.
4. Забезпечення можливісті підтвердження/відхилення матчу арбітром.
5. Підтримка повної української локалізації всіх елементів інтерфейсу користувача

Нефункціональні вимоги:

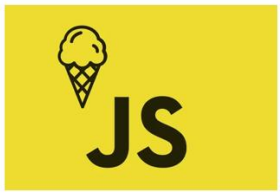
1. Час відгуку API не повинен перевищувати 200 мс, а повний рендеринг клієнтської частини має займати менше 2 секунд.
2. Архітектура та UI повинні забезпечувати виконання будь-якого головного сценарію взаємодії не більше ніж за 3-4 кліки.
3. Інтерфейс має підтримувати стабільну роботу та бути адаптивним у всіх сучасних браузерах на екранах із шириною від 320 пікселів.
4. Для надійного захисту персональних даних та підтримки понад 100 одночасних користувачів система повинна використовувати алгоритм Bcrypt для хешування паролів та JWT-токени для безпечної авторизації без збереження стану сесії на сервері.



# ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ





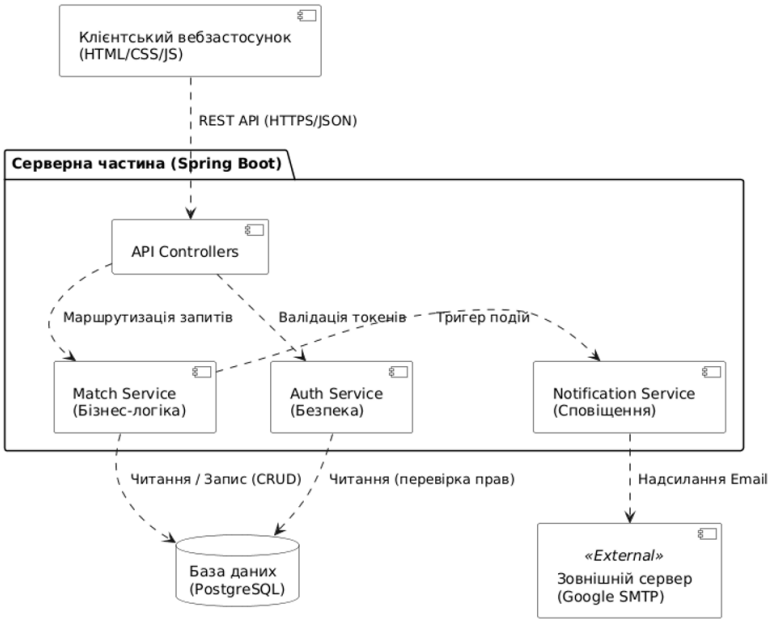









## Діаграма компонентів вебзастосунку RefMate

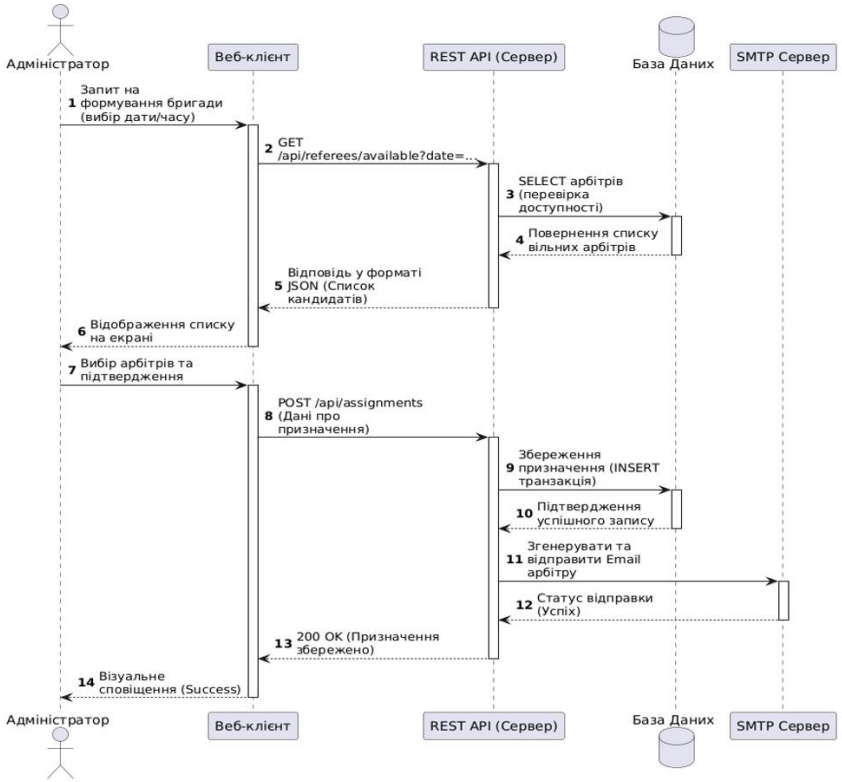



7

## Діаграма варіантів використання вебзастосунку RefMate

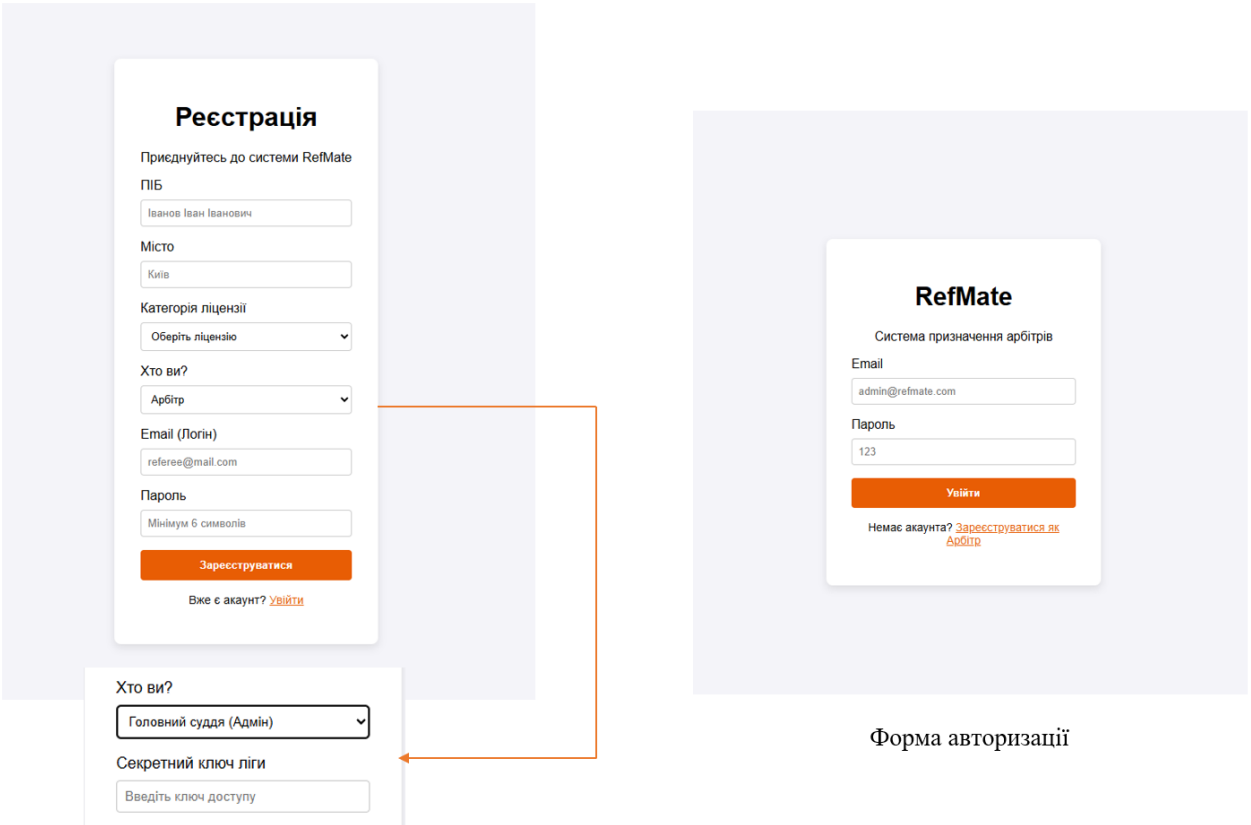

8

## Діаграма послідовності процесу призначення арбітра на матч




9

## ЕКРАННІ ФОРМИ ЗАСТОСУНКУ

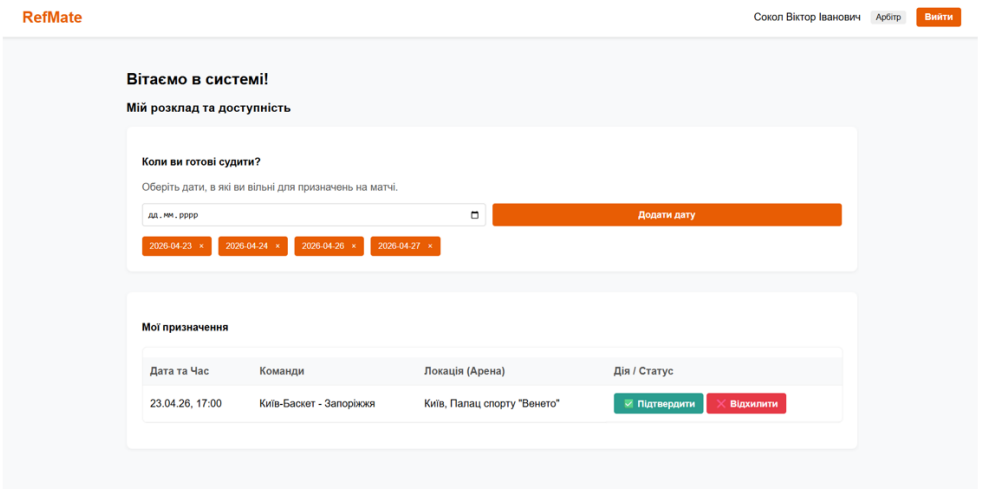



Форма авторизації

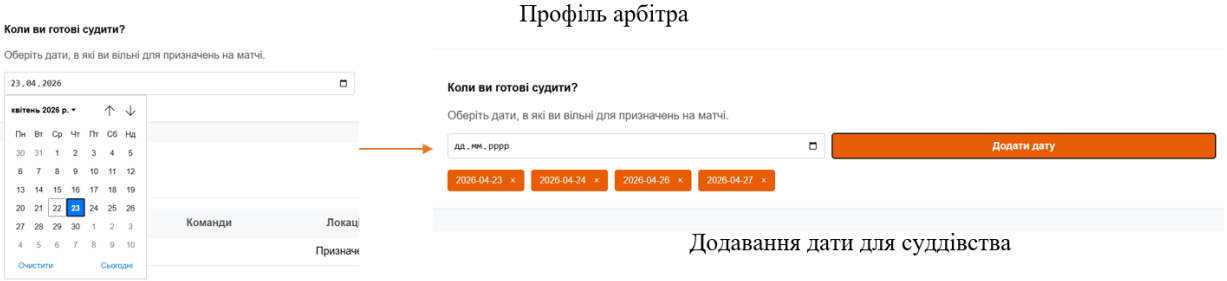
Форма реєстрації та валідація адміна

10

# ЕКРАННІ ФОРМИ ЗАСТОСУНКУ




Профіль арбітра

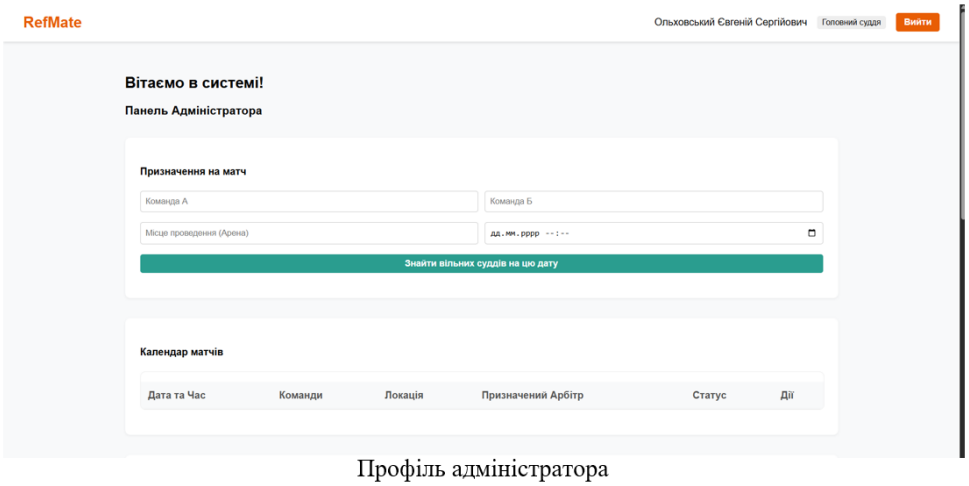



Додавання дати для суддівства

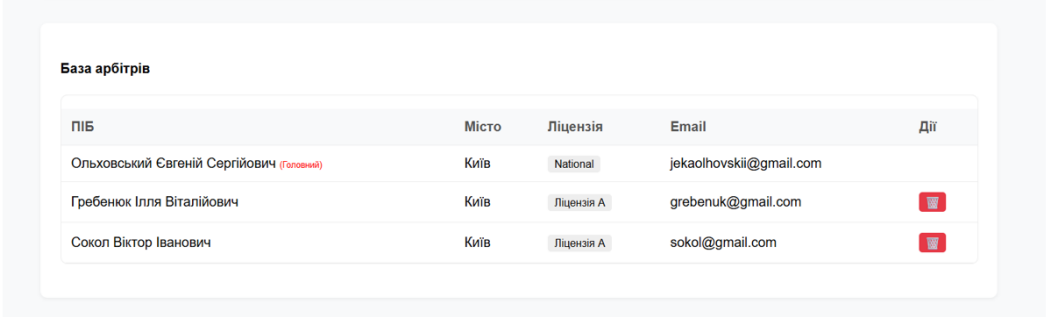
Вибір дати для суддівства



# ЕКРАННІ ФОРМИ ЗАСТОСУНКУ

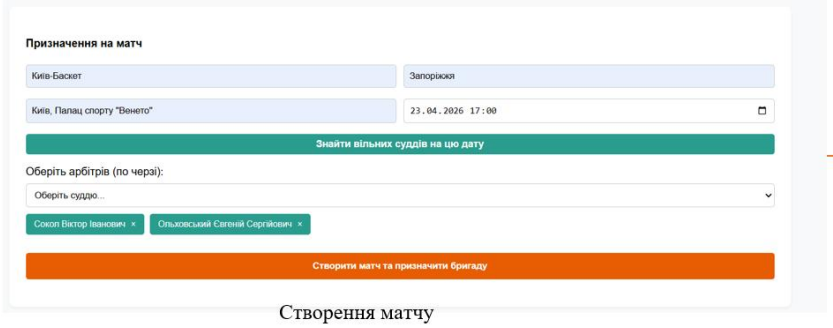



Профіль адміністратора

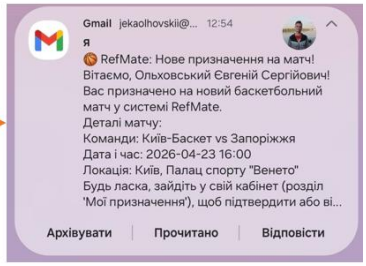



База арбітрів

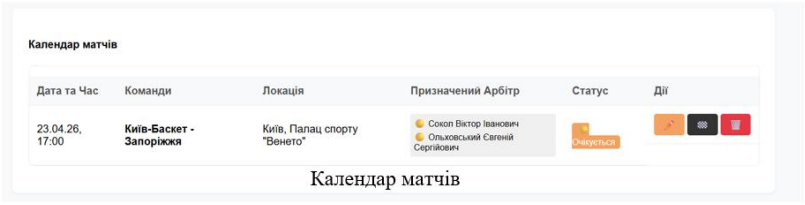
# ЕКРАННІ ФОРМИ ЗАСТОСУНКУ



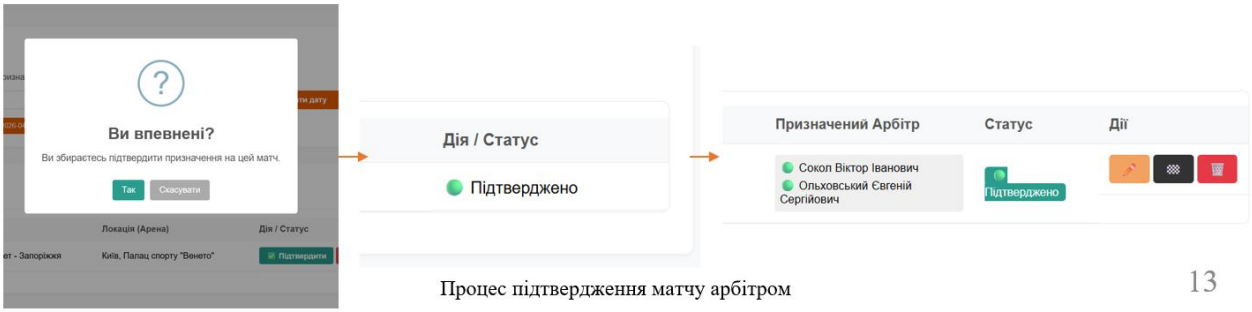

Створення матчу




Email повідомлення про призначення




Календар матчів




Процес підтвердження матчу арбітром



# АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

**Тези доповідей:**

1. Ольховський Є.С., Довженко Т.П. Визначення вимог до веборієнтованої інформаційної системи супроводу суддівської діяльності в баскетболі. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії»», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез.- К.:ДУІКТ. С. 84-86
2. Ольховський Є.С., Довженко Т.П. Аналіз та проєктування інформаційної системи для координації суддівського персоналу в спорті. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез.- К.:ДУІКТ. С.343-347.

# ВИСНОВКИ

1. Проаналізовано існуючі процеси організації суддівства та аналоги систем спортивного менеджменту, на основі чого сформовано точні функціональні та нефункціональні вимоги до програмного продукту.
2. Обрано стек вебтехнологій, а саме Java, Spring Boot, PostgreSQL, клієнтський JavaScript, для оптимальної реалізації серверної та клієнтської частин системи супроводу суддівської діяльності.
3. Спроєктовано клієнт-серверну архітектуру, логічну структуру бази даних та рольову модель доступу (Адміністратор/Арбітр) для забезпечення надійного зберігання інформації та криптографічного захисту персональних даних.
4. Розроблено та інтегровано на рівні серверної логіки алгоритм автоматичного фільтрування арбітрів, що дозволило системі самостійно відсікати недоступних кандидатів, а на практиці повністю усунуло ризик часових накладок..
5. Розроблено та впроваджено адаптивний користувацький інтерфейс із системою миттєвих Email-сповіщень через зовнішній SMTP-сервер, що звільнило організаторів від рутинної роботи.
6. Проведено тестування розробленого програмного забезпечення, у ході якого успішно пройдено 100% написаних модульних тестів (JUnit) для перевірки алгоритму призначень, а інструментальні заміри швидкодії зафіксували стабільний відгук API в межах 200 мс. Ці фактичні показники доводять коректність роботи системи та виконання поставлених вимог.

# ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

## Backend частина

### MatchController

```
package com.refereeapp.backend.controller;
import com.refereeapp.backend.model.Match;
import com.refereeapp.backend.model.User;
import
com.refereeapp.backend.repository.MatchRepositor
y;
import
com.refereeapp.backend.repository.UserRepository;
import
com.refereeapp.backend.service.EmailService;
import org.springframework.web.bind.annotation.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
@RestController
@RequestMapping("/api/matches")
@CrossOrigin(origins = "*")
public class MatchController {
   private final MatchRepository matchRepository;
   private final EmailService emailService;
   private final UserRepository userRepository;
   public MatchController(MatchRepository
matchRepository, EmailService emailService,
UserRepository userRepository) {
      this.matchRepository = matchRepository;
      this.emailService = emailService;
      this.userRepository = userRepository;
   }
   @GetMapping
   public List<Match> getAllMatches() {
      return matchRepository.findAll();
   }
   @GetMapping("/{id}")
   public Match getMatchById(@PathVariable Long
id) {
      return
matchRepository.findById(id).orElseThrow(() ->
new RuntimeException("Матч не знайдено"));
   }

   @PostMapping
   public Match createMatch(@RequestBody Match
match) {
      // 1. Встановлюємо статус PENDING для
всіх призначених арбітрів
      if (match.getReferees() != null) {
         for (User ref : match.getReferees()) {
            match.getRefereeStatuses().put(ref.getId(),
"PENDING");
         }
      }
      // 2. Зберігаємо матч у базу
      Match savedMatch =
matchRepository.save(match);
      // 3. Розсилаємо листи всім призначеним
суддям
      if (savedMatch.getReferees() != null) {
         for (User ref : savedMatch.getReferees()) {
            // Дістаємо повну інформацію про
суддю з бази (щоб отримати його email)
            userRepository.findById(ref.getId()).ifPres
ent(fullReferee -> {
               emailService.sendMatchAssignmentEm
ail(
                     fullReferee.getEmail(),
                     fullReferee.getFullName(),
                     savedMatch.getTeamA(),
                     savedMatch.getTeamB(),
                     savedMatch.getDateTime().toStrin
g(),
                     savedMatch.getLocation()
               );
            });
         }
      }

      return savedMatch;
   }

   @PutMapping("/{id}/status")
   public Match updateMatchStatus(@PathVariable
Long id, @RequestParam Long refereeId,
@RequestParam String status) {
      return
matchRepository.findById(id).map(match -> {
         match.getRefereeStatuses().put(refereeId,
status);
         return matchRepository.save(match);
      }).orElseThrow(() -> new
RuntimeException("Матч не знайдено"));
```

```
    }

    @PutMapping("/{id}")
    public Match updateMatch(@PathVariable Long
id, @RequestBody Match updatedMatch) {
        return
matchRepository.findById(id).map(match -> {
            match.setTeamA(updatedMatch.getTeamA()
);
            match.setTeamB(updatedMatch.getTeamB())
;
            match.setLocation(updatedMatch.getLocatio
n());
            match.setDateTime(updatedMatch.getDateTi
me());

            // Оновлюємо бригаду суддів
            match.setReferees(updatedMatch.getReferee
s());

            // Зберігаємо статуси старих суддів, а
новим ставимо PENDING
            Map<Long, String> currentStatuses =
match.getRefereeStatuses();
            Map<Long, String> newStatuses = new
HashMap<>();
            if (updatedMatch.getReferees() != null) {
                for (User ref :
updatedMatch.getReferees()) {
                    newStatuses.put(ref.getId(),
currentStatuses.getOrDefault(ref.getId(),
"PENDING"));
                }
            }
            match.setRefereeStatuses(newStatuses);

            return matchRepository.save(match);
        }).orElseThrow(() -> new
RuntimeException("Матч не знайдено"));
    }

    @DeleteMapping("/{id}")
    public void deleteMatch(@PathVariable Long id)
{
        matchRepository.deleteById(id);
    }

    @PutMapping("/{id}/finish")
    public Match finishMatch(@PathVariable Long
id) {
        return matchRepository.findById(id).map(match -> {
            match.setFinished(true);
            return matchRepository.save(match);
        }).orElseThrow(() -> new RuntimeException("Матч не знайдено"));
    }
}
```

## UserController

```
package com.refereeapp.backend.controller;
import com.refereeapp.backend.model.User;
import
com.refereeapp.backend.repository.UserRepository;
import com.refereeapp.backend.service.UserService;
import
org.springframework.security.crypto.password.Pass
wordEncoder;
import org.springframework.web.bind.annotation.*;

import com.refereeapp.backend.security.JwtUtil;
import org.springframework.http.ResponseEntity;
import java.util.HashMap;
import java.util.Map;

import java.util.List;

@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "*")
public class UserController {

    private final UserRepository userRepository;
    private final UserService userService;
    private final PasswordEncoder passwordEncoder;
    private final JwtUtil jwtUtil;

    public UserController(UserRepository
userRepository, UserService userService,
PasswordEncoder passwordEncoder, JwtUtil
jwtUtil) {
        this.userRepository = userRepository;
        this.userService = userService;
        this.passwordEncoder = passwordEncoder;
```

```
        this.jwtUtil = jwtUtil;
    }

    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    @PostMapping("/register")
    public User createUser(@RequestBody User user,
@RequestParam(required = false) String secretKey)
{
        // Логіка перевірки ролей та ключів
        if ("ADMIN".equals(user.getRole())) {
            if (!"REFMATE-2026".equals(secretKey)) {
                throw new
RuntimeException("Неправильний секретний
ключ адміністратора!");
            }
        } else {
            user.setRole("REFEREE");
        }

        // ШИФРУЄМО ПАРОЛЬ перед тим, як
зберегти в базу
        user.setPassword(passwordEncoder.encode(use
r.getPassword()));

        return userRepository.save(user);
    }

    @GetMapping("/available")
    public List<User>
getAvailableReferees(@RequestParam String date)
{
        return userService.getAvailableReferees(date);
    }

    @PostMapping("/login")
    public ResponseEntity<Map<String, Object>>
login(@RequestBody User loginData) {
        User user =
userRepository.findByEmail(loginData.getEmail())
```

## JwtAuthenticationFilter

```
package com.refereeapp.backend.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
                .orElseThrow(() -> new
RuntimeException("Користувача не знайдено"));

        if
(!passwordEncoder.matches(loginData.getPassword(
), user.getPassword())) {
            throw new
RuntimeException("Неправильний пароль");
        }

        // ГЕНЕРУЄМО ТОКЕН
        String token =
jwtUtil.generateToken(user.getEmail(),
user.getRole(), user.getId());

        // Формуємо відповідь: віддаємо і токен, і
дані користувача
        Map<String, Object> response = new
HashMap<>();
        response.put("token", token);
        response.put("user", user);

        return ResponseEntity.ok(response);
    }

    @PutMapping("/{id}")
    public User updateUser(@PathVariable Long id,
@RequestBody User updatedUser) {
        return userRepository.findById(id).map(user ->
{
            user.setAvailability(updatedUser.getAvailabi
lity());
            return userRepository.save(user);
        }).orElseThrow(() -> new
RuntimeException("Користувача не знайдено"));
    }

    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) {
        userRepository.deleteById(id);
    }


import
org.springframework.security.authentication.Userna
mePasswordAuthenticationToken;
import
org.springframework.security.core.context.Security
ContextHolder;
```

```
import
org.springframework.security.core.userdetails.UserD
etails;
import
org.springframework.security.web.authentication.W
ebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import
org.springframework.web.filter.OncePerRequestFilt
er;

import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends
OncePerRequestFilter {

   private final JwtUtil jwtUtil;
   private final CustomUserDetailsService
userDetailsService;

   public JwtAuthenticationFilter(JwtUtil jwtUtil,
CustomUserDetailsService userDetailsService) {
      this.jwtUtil = jwtUtil;
      this.userDetailsService = userDetailsService;
   }

   @Override
   protected void
doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
      throws ServletException, IOException {

      // 1. Шукаємо заголовок Authorization
      final String authorizationHeader =
request.getHeader("Authorization");

      String email = null;
      String jwt = null;

      // 2. Якщо заголовок є і починається з
"Bearer " (стандарт JWT)
      if (authorizationHeader != null &&
authorizationHeader.startsWith("Bearer ")) {
         jwt = authorizationHeader.substring(7); //
Відрізаємо слово "Bearer "
         try {
            email = jwtUtil.extractEmail(jwt); //
Дістаємо email з токена
         } catch (Exception e) {
            System.out.println("Токен недійсний або
його час вийшов");
         }
      }

      // 3. Якщо email є, а в системі користувач ще
не авторизований
      if (email != null &&
SecurityContextHolder.getContext().getAuthenticati
on() == null) {
         UserDetails userDetails =
this.userDetailsService.loadUserByUsername(email)
;

         // 4. Перевіряємо, чи токен не підроблений
         if (jwtUtil.validateToken(jwt)) {
            UsernamePasswordAuthenticationToken
authToken = new
UsernamePasswordAuthenticationToken(
                  userDetails, null,
userDetails.getAuthorities());
            authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(req
uest));
            // 5. Пускаємо користувача!
            SecurityContextHolder.getContext().setAu
thentication(authToken);
         }
      }
      // Передаємо запит далі по ланцюжку
      chain.doFilter(request, response);
   }
}
```

## Frontend частина

### Register JS

```
document.addEventListener('DOMContentLoaded',
() => {
   const registerForm =
document.getElementById('registerForm');
```

```
  const roleSelect =
document.getElementById('regRole');
  const adminKeyGroup =
document.getElementById('adminKeyGroup');
  if (roleSelect) {
    roleSelect.addEventListener('change', (e) => {
      if (e.target.value === 'ADMIN') {
        adminKeyGroup.style.display = 'block';
      } else {
        adminKeyGroup.style.display = 'none';
        document.getElementById('adminKey').va
lue = '';
      }
    });
  }
  if (registerForm) {
    registerForm.addEventListener('submit', async
(e) => {
      e.preventDefault();

      const selectedRole =
document.getElementById('regRole').value;
      const secretKey =
document.getElementById('adminKey').value;
      const newUser = {
        fullName:
document.getElementById('regFullName').value,
        city:
document.getElementById('regCity').value,
        licenseCategory:
document.getElementById('regLicense').value,
        email:
document.getElementById('regEmail').value,
        password:
document.getElementById('regPassword').value,
        role: selectedRole,
        availability: ""
      };
      Swal.fire({
        title: 'Реєстрація...',
        text: 'Будь ласка, зачекайте.',
        allowOutsideClick: false,
        didOpen: () => {
          Swal.showLoading();
        }
      });
      try {
        const response = await
fetch(`https://refmate-
api.onrender.com/api/users/register?secretKey=${se
cretKey}`, {
          method: 'POST',
          headers: { 'Content-Type':
'application/json' },
          body: JSON.stringify(newUser)
        });
        if (response.ok) {
          await Swal.fire({
            icon: 'success',
            title: 'Успіх!',
            text: 'Реєстрація пройшла успішно.
Тепер ви можете увійти.',
            confirmButtonColor: '#e85d04'
          });
          window.location.href = 'index.html';
        } else {
          Swal.fire({
            icon: 'error',
            title: 'Помилка!',
            text: 'Перевірте дані. Можливо,
така пошта вже існує або введено неправильний
секретний ключ.',
            confirmButtonColor: '#e63946'
          });
        }
      } catch (error) {
        console.error('Помилка:', error);
        Swal.fire({
          icon: 'error',
          title: 'Помилка з\'єднання',
          text: 'Не вдалося підключитися до
сервера.',
          confirmButtonColor: '#e63946'
        });
      }
    });
  }
});
```