

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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-сервісу "STUDINFO" для допомоги
абітурієнтам під час вступу до ЗВО»

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 мають посилання
на відповідне джерело*

_____ Денис ГАРКАВЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

_____ Денис ГАРКАВЕНКО

Керівник: _____ Ірина ЗАМРІЙ
д-р техн. наук, проф.

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Гаркавенку Денису Олеговичу

1. Тема кваліфікаційної роботи: «Розробка web-сервісу "STUDINFO" для допомоги абітурієнтам під час вступу до ЗВО»

керівник кваліфікаційної роботи д-р техн. наук, професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

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

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про процеси інформаційної підтримки абітурієнтів під час вступної кампанії, відкриті дані Єдиної державної електронної бази з питань освіти, технічна документація засобів розробки web-сервісів.

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

1. Аналіз предметної області інформаційної підтримки абітурієнтів під час вступної кампанії, огляд існуючих web-сервісів для допомоги абітурієнтам та визначення недоліків існуючих рішень.

2. Огляд та обґрунтування вибору засобів реалізації клієнтської і серверної частин web-сервісу STUDINFO.

3. Проектування архітектури та програмна реалізація web-сервісу STUDINFO – структура бази даних, парсинг відкритих даних ЄДЕБО, модуль "Помічник вступу", калькулятор конкурсного бала та панель адміністрування.

4. Тестування розробленого web-сервісу та демонстрація результатів розробки.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Схема зв'язків між сутностями бази даних.
6. Діаграма послідовності.
7. Діаграма діяльності користувача.
8. Екранні форми.
9. Апробація результатів дослідження.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-04.03.2026	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2026	
3	Огляд методик реалізації web-сервісів для допомоги абітурієнтам під час вступу до ЗВО	14.03-25.03.2026	
4	Проектування застосунку для допомоги абітурієнтам під час вступу до ЗВО	26.03-02.04.2026	
5	Програмна реалізація застосунку	03.04-19.04.2026	
6	Тестування застосунку	20.04-28.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	29.04-04.05.2026	
8	Розробка демонстраційних матеріалів	05.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти

_____ (підпис)

Денис ГАРКАВЕНКО

Керівник
кваліфікаційної роботи

_____ (підпис)

Ірина ЗАМРІЙ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 78 стор., 2 табл., 47 рис., 29 джерел.

Мета роботи – спрощення процесу підтримки абітурієнтів під час вступної кампанії через автоматизацію розрахунку конкурсних показників та надання інформаційної підтримки.

Об'єкт дослідження – процеси інформаційної підтримки абітурієнтів під час вступної кампанії та аналізу даних вступної кампанії.

Предмет дослідження – засоби та технології для розробки сервісу інформаційної підтримки вступників за допомогою автоматизації розрахунку конкурсних показників, анкетування для підбору конкурсних пропозицій.

Короткий зміст роботи: В роботі досліджено та проаналізовано сферу інформаційної підтримки абітурієнтів під час вступної кампанії до ЗВО за рахунок web-сервісів. Було визначено та аргументовано вибір основних технологій - Next.js, TypeScript, Chakra UI, Ruby, Ruby on Rails, PostgreSQL, Docker.

Реалізований сервіс допомагає обирати ЗВО, університет та конкурсну пропозицію на базі пройденої анкети (модуль "Помічник вступу"), переглядати результати вступної кампанії в реальному часі та отримувати всю необхідну інформацію для абітурєнта. Проведено тестування за допомогою сучасних методів тестування, підготовлено повну документацію та звітний матеріал.

Сферою використання застосунку є інформаційна підтримка абітурієнтів під час вступу до ЗВО.

КЛЮЧОВІ СЛОВА: WEB-SERVIS, TYPESCRIPT, NEXT.JS, RUBY, TYPESCRIPT, CHAKRA UI, POSTGRESQL АБІТУРІЄНТ, ВСТУПНА КАМПАНИЯ, КОНКУРСНИЙ ПОКАЗНИК, ЗВО, КОНКУРСНИЙ БАЛ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	9
ВСТУП	10
1 АНАЛІЗ WEB-СЕРВІСІВ ДЛЯ ПІДТРИМКИ АБІТУРІЄНТІВ ПІД ЧАС ВСТУПНОЇ КАМΠΑНІЇ	12
1.1 Особливості інформаційної підтримки абітурієнтів	12
1.2 Аналіз вступної кампанії та конкурсних показників	14
1.3 Огляд існуючих web-сервісів для допомоги абітурієнтам	15
1.3.1 Abitly	15
1.3.2 Osvita.ua	18
1.3.3 Education.ua	20
1.3.4 Abit-poisk	23
1.3.5 Висновки аналізу існуючих web-сервісів	24
1.4 Визначення недоліків існуючих рішень та постановка задачі	27
2 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ WEB-СЕРВІСУ	29
2.1 Клієнтська частина web-сервісу	29
2.1.1 Аналіз та вибір технологій клієнтської частини	29
2.1.2 Підходи до побудови клієнтської частини web-сервісу	32
2.2 Серверна частина web-сервісу	34
2.2.1 Аналіз та вибір технологій серверної частини	34
2.2.2 Підходи до організації серверної логіки та API	35
2.2.3 Забезпечення безпеки даних та контролю доступу	37
2.3 Система керування базою даних	37
2.4 Панель адміністрування системою	38
2.5 Засоби розгортання web-сервісу	39
3 РЕАЛІЗАЦІЯ WEB-СЕРВІСУ	40
3.1 Проектування загальної архітектури web-сервісу	40
3.1.1 Структура бази даних	43
3.1.2 Структура бекенд частини	46

3.1.3 Структура фронтенд частини	50
3.1.4 Проектування API-архітектури	52
3.1.5 Архітектура розгортання проєкту	55
3.2 Розробка основного лендінгу web-сервісу	55
3.3 Парсинг та підготовка відкритих даних з ЄДЕБО	60
3.4 Розробка модуля "Помічник вступу"	63
3.4.1 Реєстрація та профіль користувача	63
3.4.2 Анкета вступника та конкурсні пропозиції	65
3.4.3 Сторінки університетів	71
3.4.4 Розробка калькулятора конкурсного балу	73
3.5 Налаштування панелі адміністрування	75
4 РЕЗУЛЬТАТИ РОЗРОБКИ ТА ТЕСТУВАННЯ WEB-SERVICES	77
4.1 Демонстрація екранних форм web-сервісу	77
4.2 Тестування розробленого web-сервісу	84
ВИСНОВКИ	87
ПЕРЕЛІК ПОСИЛАНЬ	89
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	92
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ	100

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування застосунків)

CORS – Cross-Origin Resource Sharing (спільне використання ресурсів між різними джерелами)

CSR – Client Side Rendering (рендеринг на стороні клієнта)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

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

HTTPS – Hypertext Transfer Protocol Secure (захищений протокол передачі гіпертексту)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JWT – JSON Web Token (токен автентифікації)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

REST – Representational State Transfer (передача репрезентативного стану)

SEO – Search Engine Optimization (пошукова оптимізація)

SQL – Structured Query Language (мова структурованих запитів)

SSR – Server Side Rendering (рендеринг на стороні сервера)

СУБД – Система управління базами даних

UI – User Interface (інтерфейс користувача)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

UX – User Experience (користувацький досвід)

ЗВО – Заклад вищої освіти

НМТ – Національний мультипредметний тест

КБ – конкурсний бал (інтегральний показник що визначає позицію абітурієнта у рейтинговому списку)

КП – конкурсна пропозиція (пропозиція закладу вищої освіти щодо прийому на навчання за певною спеціальністю)

ВСТУП

Розвиток цифрових технологій та зростання обсягів даних вступної кампанії створюють запит на нові інструменти для абітурієнтів. Вибір університету і спеціальності – це не разове рішення, а процес: НМТ, коефіцієнти, прохідні бали, конкуренція, перспективи. Кожен з цих параметрів важливий, і кожен потребує аналізу. Наявні сервіси з цим не справляються – одні показують дані без пояснень, інші рахують бал і на цьому зупиняються. Абітурієнт лишається сам на сам з десятком вкладок і купою цифр.

З розвитком web-технологій з'явилась можливість об'єднати все це в одному місці. Платформа, яка одночасно рахує конкурсний бал, оцінює шанси, підбирає програми і пояснює результат має валикий потенціал в наших реаліях. Такий підхід дозволяє абітурієнту бачити повну картину, а не збирати її по шматках. Додатково – платформа працює з персональними даними, тому питання безпеки стоїть не менш гостро ніж питання функціональності.

STUDINFO – це саме такий інструмент. Розрахунок конкурсного бала за введеними параметрами, оцінка шансів на основі реальної статистики, пошук і фільтрація освітніх програм, персоналізовані рекомендації – все в одному інтерфейсі. Є особистий кабінет для збереження результатів і порівнянь, є адміністративна панель. Захист даних реалізовано на кількох рівнях: автентифікація, хешування паролів, розмежування прав, захист від типових web-загроз і передача через HTTPS.

Об'єктом дослідження є процеси інформаційної підтримки абітурієнтів під час вступної кампанії та аналізу даних вступної кампанії.

Предметом дослідження є засоби та технології для розробки сервісу інформаційної підтримки вступників за допомогою автоматизації розрахунку конкурсних показників, анкетування для підбору конкурсних пропозицій.

Метою дослідження є спрощення процесу підтримки абітурієнтів під час вступної кампанії через автоматизацію розрахунку конкурсних показників та надання інформаційної підтримки.

Для досягнення поставленої мети проаналізовано існуючі сервіси і визначено їх слабкі місця. Сформовано вимоги до системи, спроектовано архітектуру. Клієнтська частина – Next.js і TypeScript, серверна – Ruby on Rails, база даних – PostgreSQL, розгортання – Docker. Проведено тестування розрахункового модуля, системи рекомендацій і механізмів безпеки.

Наукова новизна полягає в поєднанні аналізу даних, персоналізація і захист інформації в одному українському сервісі для абітурієнтів – такого досі не було.

Практична значущість полягає у створенні готового web-сервісу, що замінює необхідність одночасного використання кількох розрізнених платформ – абітурієнт отримує розрахований конкурсний бал, персоналізований список конкурсних пропозицій і детальну інформацію про університети в одному місці. Це скорочує час на пошук і аналіз інформації та зменшує ймовірність помилок при прийнятті рішень щодо вибору закладу вищої освіти. Платформа розрахована на подальший розвиток – додавання нових модулів і адаптацію до змін у правилах вступної кампанії.

1 АНАЛІЗ WEB-SERVISІВ ДЛЯ ПІДТРИМКИ АБІТУРІЄНТІВ ПІД ЧАС ВСТУПНОЇ КАМПАНІЇ

1.1 Особливості інформаційної підтримки абітурієнтів

Інформаційна підтримка абітурієнтів під час вступної кампанії – це не лише просто доступ до даних, а і про те, наскільки людина може швидко зорієнтуватись, зрозуміти свої шанси і прийняти рішення. А зробити це складно, бо вступ – процес багатосаровий [1]. Кожен, хто проходив через це, знає: інформації багато, але зібрати її до купи і зробити з неї висновок – окреме завдання.

Рейтинги оновлюються щогодини. Кількість поданих заяв змінюється. Конкурсні списки коригуються. Дані, які були актуальні вранці, до вечора можуть застаріти. Абітурієнт змушений постійно щось відслідковувати – і в цьому режимі легко щось пропустити або прийняти рішення на основі вже неактуальної інформації. Особливо гостро це відчувається в останні дні подачі заяв, коли ситуація змінюється буквально в режимі реального часу.

Конкурсний бал – це лише один з параметрів. Насправді при виборі університету і спеціальності абітурієнт має враховувати одночасно:

- спеціальність і освітню програму;
- форму навчання – денна, заочна, дистанційна;
- регіон і місто розташування закладу;
- кількість бюджетних місць і рівень конкуренції;
- прохідні бали попередніх років;
- репутацію університету і перспективи працевлаштування;
- власні пріоритети і фінансові можливості.

Всі ці критерії мають різну вагу для кожної конкретної людини. Аналізувати їх одночасно без допоміжного інструменту – складно і довго.

Формула конкурсного бала включає результати тестування з різних предметів, вагові коефіцієнти, додаткові бали за певні досягнення, і все це може змінюватись залежно від спеціальності або року вступу, що вже створює комплексність в вирахуванні КБ. Для людини без досвіду роботи з такими розрахунками це реальна проблема. Помилка в обчисленні або неправильне трактування умов вступу – і висновки про власні шанси виявляються хибними. А на основі хибних висновків приймаються хибні рішення, що в подальшому призводить до певних неприємних наслідків.

Дані про вступну кампанію розміщені на різних платформах, і кожна виконує свою окрему функцію:

- офіційні портали – повна інформація, але складний інтерфейс і відсутність аналітики [2, 3];
- освітні агрегатори – зручніше, але переважно новини та загальні пояснення;
- калькулятори конкурсного бала – корисні, але вирішують лише одну задачу;
- університетські сайти – детальна інформація, але в різному форматі і незручна для порівняння.

В результаті абітурієнт тримає відкритими кілька платформ одночасно, переносить дані вручну і ризикує щось переплутати або пропустити. Це не проблема конкретного сервісу – це системна проблема предметної галузі. Відсутність персоналізації. Більшість сервісів працюють однаково для всіх. Результати пошуку не враховують рівень підготовки конкретного користувача, його пріоритети, попередні дії або індивідуальні обмеження. Між тим ефективна підтримка – це адаптація під конкретну людину. Абітурієнту потрібна не загальна таблиця університетів, а відповідь на питання, що і є головною технічною метою даної роботи.

Саме сукупність цих вимог і визначає необхідність розробки платформи, яка підходить до задачі комплексно, а не закриває лише одну її частину [4].

1.2 Аналіз вступної кампанії та конкурсних показників

Проведений аналіз показав, що за вступною кампанією стоїть ціла система: формування конкурсних пропозицій, обробка результатів тестування, побудова рейтингових списків. На перший погляд все зрозуміло, але за кожним з цих кроків стоїть велика кількість змінних, і в сукупності це складний організаційно-інформаційний процес.

Центральний елемент – конкурсний бал. Це інтегральний показник, який визначає позицію абітурієнта в рейтингу. До його складу входять:

- результати НМТ з кількох предметів;
- вагові коефіцієнти, встановлені для кожної спеціальності окремо;
- додаткові бали за певні досягнення або особливі умови вступу.

Важлива деталь: один і той же результат тестування має різну "цінність" для різних спеціальностей – через різні вагові коефіцієнти. Плюс правила розрахунку можуть змінюватись щороку. Тобто досвід минулого року не завжди допомагає – все треба перераховувати заново.

Після обчислення балу формується рейтинговий список для кожної конкурсної пропозиції. Місце в цьому списку визначає шанси на зарахування – бюджет чи контракт. І це є стартова точка для потенційних помилок: навіть незначна зміна одного показника – кількості заяв, їхньої пріоритетності, відмов інших вступників – може змінити позицію абітурієнта в списку. Рейтинг живий і постійно рухається.

Заяви подаються, пріоритети змінюються, частина вступників відмовляється від участі – рейтингові списки реагують на це одразу. Відстежити всі ці зміни вручну нереально, а рішення, прийняте на основі даних кількох годин давнини, може виявитись неактуальним.

Окремо варто згадати про статистику попередніх років. Прохідні бали минулих кампаній дають приблизне уявлення про рівень конкуренції – але не більше. Умови вступу змінюються, популярність спеціальностей коливається,

кількість абітурієнтів щороку різна. Тому історичні дані – це орієнтир, а не гарантія. Їх треба враховувати, але не абсолютизувати.

Додатково – невизначеність. Скільки людей подали заяви на ту саму спеціальність, які в них пріоритети, чи збираються вони відмовлятися – абітурієнт цього не знає. Конкурс за своєю природою є ймовірнісним, і точно передбачити результат неможливо навіть при повному доступі до даних.

Нарешті – обсяг. Університетів багато, конкурсних пропозицій ще більше, у кожної свої умови і коефіцієнти. Утримати всі параметри в голові і зробити з них коректний висновок одній людині не під силу. Вступна кампанія характеризується динамічністю даних, варіативністю правил і структурною складністю – і саме це обґрунтовує потребу в інструментах на кшталт STUDINFO, які автоматизують розрахунки і дають абітурієнту реальну картину його шансів [5, 6].

1.3 Огляд існуючих web-сервісів для допомоги абітурієнтам

1.3.1 Abitly

Abitly орієнтований на абітурієнтів, які тільки починають розбиратись у темі вступу [7]. Довідкова інформація про університети, загальні рейтинги, базовий пошук програм – цього тут вистачає. Навігація не викликає питань, розділи знаходяться швидко. Але вже на етапі пошуку під конкретні критерії з'являються обмеження: база даних неповна, частина спеціальностей і конкурсних пропозицій просто відсутня. Або результат виявляється неповним, або доводиться йти шукати деінде. В обох випадках це додатковий час і зусилля.

Рейтинг університетів – одна з ключових функцій сервісу. Корисна річ на старті, коли потрібно скласти загальне уявлення про заклади. Можна подивитись які університети вважаються сильнішими, порівняти їх між собою на базовому рівні. Але є принципова проблема: рейтинг загальний і безособовий. Він не враховує результати конкретного абітурієнта – його бали, пріоритети, обрану

спеціальність. Університет може займати високу позицію в рейтингу, але прохідний бал там такий, що конкретному абітурієнту він просто недоступний. Окремим випадком є відсутність демонстрації позиції в загальному рейтингу університетів, тому його практична цінність доволі обмежена і не є повноцінною.

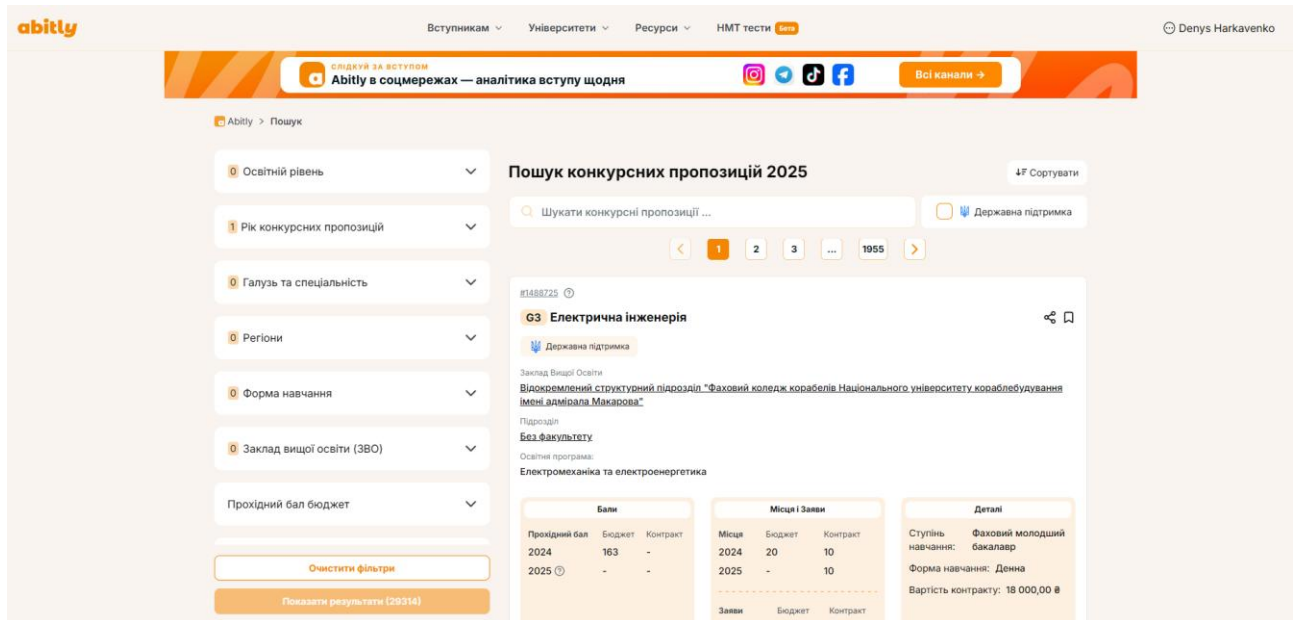


Рис. 1.1 Інтерфейс конкурсних пропозицій web-сервісу Abitly

Головне що відсутнє – розрахунок конкурсного бала відносно конкурсної пропозиції, а можливо лише на окремій сторінці на сайті. Без нього абітурієнт не розуміє де він знаходиться відносно інших вступників і які в нього реальні шанси. Рахувати вручну – довго і є ризик помилитись. Відкривати окремий калькулятор – ще один інструмент у списку. Дані про бал, кількість заяв, динаміку рейтингу і статистику попередніх років існують розрізнено і ніде не зводяться в єдину картину. Подивитись як всі ці фактори впливають один на одного – неможливо. Рішення в таких умовах приймається скоріше інтуїтивно, ніж на основі повного аналізу.

Персоналізація – ще одне слабке місце. Сервіс видає однаковий результат для всіх – незалежно від балів, пріоритетів і побажань конкретної людини. Підтримка яка не враховує хто перед нею – це не підтримка, а довідник. Не

формує рекомендацій на основі введених даних, не адаптує пошук під індивідуальні критерії. Всі бачать одне й те саме.

Особистий кабінет є, але його можливості базові. Зберегти результати пошуку – не можна. Сформувати список обраних конкурсних пропозицій – не можна. Порівняти кілька варіантів в одному місці – теж не можна. Кожен раз коли абітурієнт повертається до сервісу, він починає з нуля. Повторно вводить ті самі дані та заново шукає ті самі конкурсні пропозиції.

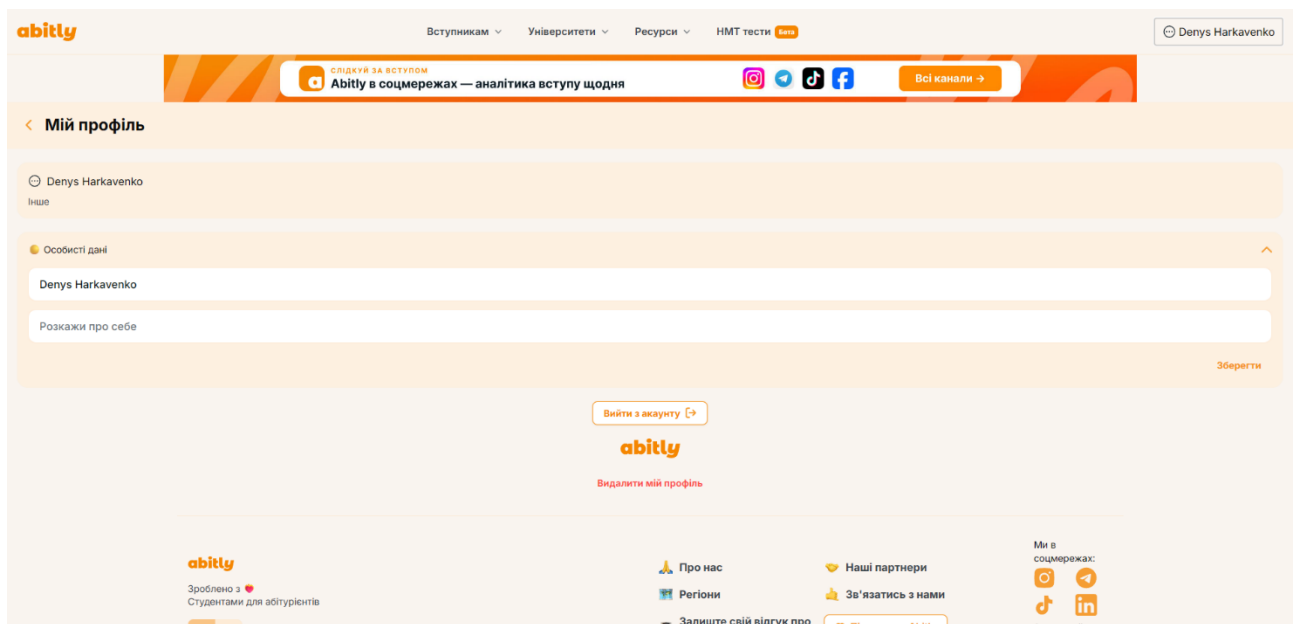


Рис. 1.2 Особистий кабінет web-сервісу Abitly

Дані оновлюються із затримкою. В пік кампанії рейтингові списки змінюються постійно – кожна нова заява зміщує позиції. Абітурієнт, який дивиться на застарілу картину, може прийняти рішення яке вже не відповідає реальній ситуації. Дані не оновлюються вчасно, результати не зберігаються – наступного разу все доводиться шукати заново.

Підсумок такий. Abitly може бути корисним на самому початку – коли потрібно просто зорієнтуватись, подивитись які університети існують і що вони пропонують. Для цієї задачі сервіс справляється. Але як повноцінний інструмент для аналізу вступної кампанії і прийняття обґрунтованих рішень – не працює. Немає розрахунку конкурсного бала відносно конкурсної пропозиції. Немає

персоналізації. Дані не оновлюються вчасно (використовуються за минулі роки). Немає комплексного підходу де всі фактори враховуються разом. Кожен з цих недоліків окремо – вже проблема. Разом вони формують системне обмеження, яке підтверджує потребу в більш інтегрованому рішенні.

1.3.2 Osvita.ua

Osvita.ua добре відомий серед тих, хто шукає інформацію про вступ [8]. Університети, спеціальності, умови прийому, освітні новини – все зібрано в одному місці і структуровано. Розібратись в інтерфейсі нескладно. Для тих хто тільки починає вивчати тему – непоганий старт: можна знайти університет, подивитись які спеціальності він пропонує, прочитати про умови вступу. Саме в цій ролі – інформаційного довідника – сервіс працює добре.

Але є принципова проблема: сервіс дає дані, а не аналіз. Розрахувати конкурсний бал тут можна, але оцінити шанси на вступ – ні. Дані є – але що з ними робити далі, платформа не пояснює. Рахувати, зіставляти, робити висновки – все лягає на самого користувача. Для тих хто вперше стикається з вступною кампанією це відчутна складність: інформації багато, але орієнтиру немає.

Пошук і фільтрація освітніх програм – окрема слабка точка. Інформації на платформі багато, але гнучких механізмів для її сортування немає. Знайти варіанти під кілька критеріїв одночасно – спеціальність, форма навчання, регіон, рівень конкуренції – складно. Немає можливості задати кілька параметрів і отримати відфільтрований результат. Доводиться переглядати вручну і самотійно відсівати те, що не підходить. При великому обсязі даних це займає значно більше часу ніж мало б.

Персоналізації немає взагалі. Osvita.ua подає інформацію однаково для всіх – незалежно від балів, пріоритетів і ситуації конкретної людини. Жодних рекомендацій на основі введених даних. Жодної адаптації під індивідуальні показники. Два абітурієнти з абсолютно різними результатами НМТ і різними

цілями бачать однаковий контент. Для загального ознайомлення це ще терпимо. Але для прийняття конкретного рішення – недостатньо.

Особистий кабінет є, але функціонально обмежений. Зберегти результати пошуку – не можна. Сформувати список обраних програм і повернутись до нього пізніше – теж. Повернувся через день – починай спочатку. Ті самі пошуки, ті самі сторінки, ті самі дії. Для процесу який розтягується на тижні це відчутний мінус.

The screenshot shows the 'Розрахунок конкурсного бала' (Calculation of competitive score) form on the Osvita.ua website. The form is titled 'Введіть дані для розрахунку' (Enter data for calculation). It includes the following fields and options:

- 2026 рік НМТ** (Year of the NMT exam) - dropdown menu.
- Спеціальність** (Specialty) - dropdown menu.
- Регіон для вступу** (Region for admission) - dropdown menu.
- Бали предметів** (Subject scores) - section with two tabs: **100-200** (selected) and **Тестові** (Test).
- Українська мова** (Ukrainian language) - input field with value **200**.
- Історія України** (History of Ukraine) - input field with value **200**.
- Математика** (Mathematics) - input field with value **200**.
- 4-й предмет** (4th subject) - dropdown menu with value **200**.
- Розрахувати конкурсний бал** (Calculate competitive score) - button.

Рис. 1.3 Розрахунок конкурсного бала за допомогою сайту Osvita.ua

Комплексного аналізу вступної кампанії немає. Дані про конкурсний бал, кількість заяв, динаміку рейтингу і статистику прохідних балів попередніх років існують, але розрізнено. Звести їх в єдину картину і подивитись як вони впливають одне на одного – платформа не дозволяє. Немає інструменту який поєднає всі ці параметри і дає на виході зрозуміле уявлення про шанси вступу. Користувач змушений сам поєднувати різні джерела – і знову відкривати кілька вкладок, переносити дані вручну і складати картину по шматках.

Актуальність даних – ще одне питання. Сервіс оновлює інформацію, але не в режимі реального часу. В активну фазу кампанії, коли рейтинги змінюються щогодини і кожна нова заява впливає на позиції, навіть кілька годинна затримка може мати значення. Рішення прийняте на основі даних які вже не відповідають ситуації – це ризик отримати несподіваний результат саме тоді, коли найменш хочеться сюрпризів.

Підсумок. Osvita.ua – сильний інформаційний ресурс. Якщо потрібно дізнатись про університет, прочитати новини про вступ або знайти загальну інформацію про спеціальність – сюди. З цим завданням платформа справляється добре. Але як інструмент для аналізу і прийняття рішень вона не працює. Немає розрахунків. Немає персоналізації. Немає гнучкого пошуку. Немає комплексного підходу де всі фактори враховуються разом. Великий обсяг даних без інструментів для їх обробки – це бібліотека без каталогу. Інформація є, але знайти в ній потрібне і зробити з цього конкретний висновок – окреме завдання, яке платформа не бере на себе.

1.3.3 Education.ua

Education.ua – ще один інформаційний ресурс для тих, хто шукає дані про університети і освітні програми [9]. Формат – каталог: можна шукати за спеціальністю, містом або закладом. Для початкового орієнтування підходить. Далі – починаються обмеження.

Пошук і фільтрація базові. Комбінувати кілька критеріїв одночасно або задавати розширені параметри відбору – не можна. Тобто якщо потрібно знайти денну бюджетну спеціальність у конкретному місті з певним рівнем конкуренції – платформа з цим не впорається. Доведеться фільтрувати вручну.

Рейтингу університетів у повному розумінні немає. Можна переглянути окрему інформацію про заклад, але порівняти кілька між собою в систематизованому вигляді – не вийде. Для абітурієнта, який хоче зрозуміти де який університет стоїть відносно інших – це мінус.

Розрахунку конкурсного бала немає. Це ключова проблема. Вся інформація статична – дані є, але адаптувати їх під свої результати неможливо. Абітурієнт бачить умови вступу, але не розуміє як він виглядає на їхньому фоні. Рахувати доводиться самостійно або йти на інший сервіс.

Особистий кабінет є і дозволяє зберігати заклади за категоріями – ЗВО, школи, коледжі. Для базового збереження цього достатньо. Але далі функціональність не йде: сформувати список обраних освітніх програм, зберегти результати пошуку або порівняти кілька варіантів в одному місці – не можна. Тобто кабінет є, але як повноцінний інструмент для роботи з інформацією він не працює.

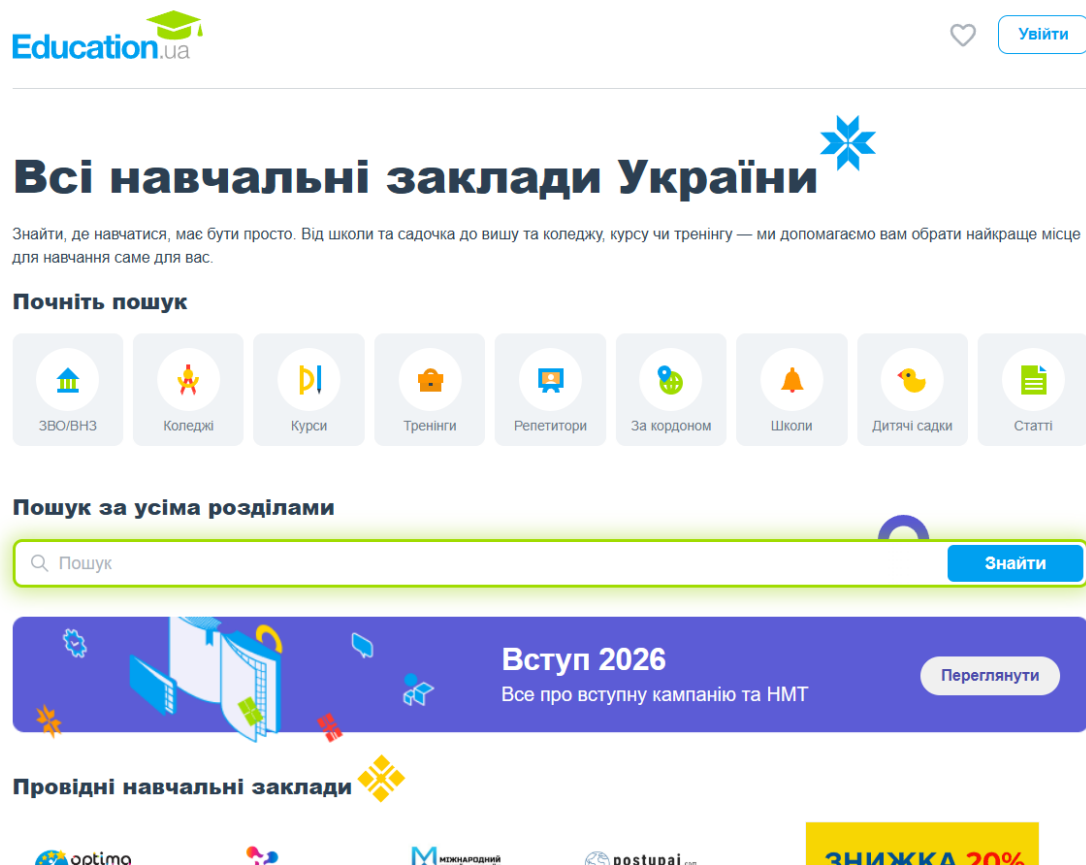


Рис. 1.4 Інтерфейс сервісу Education.ua

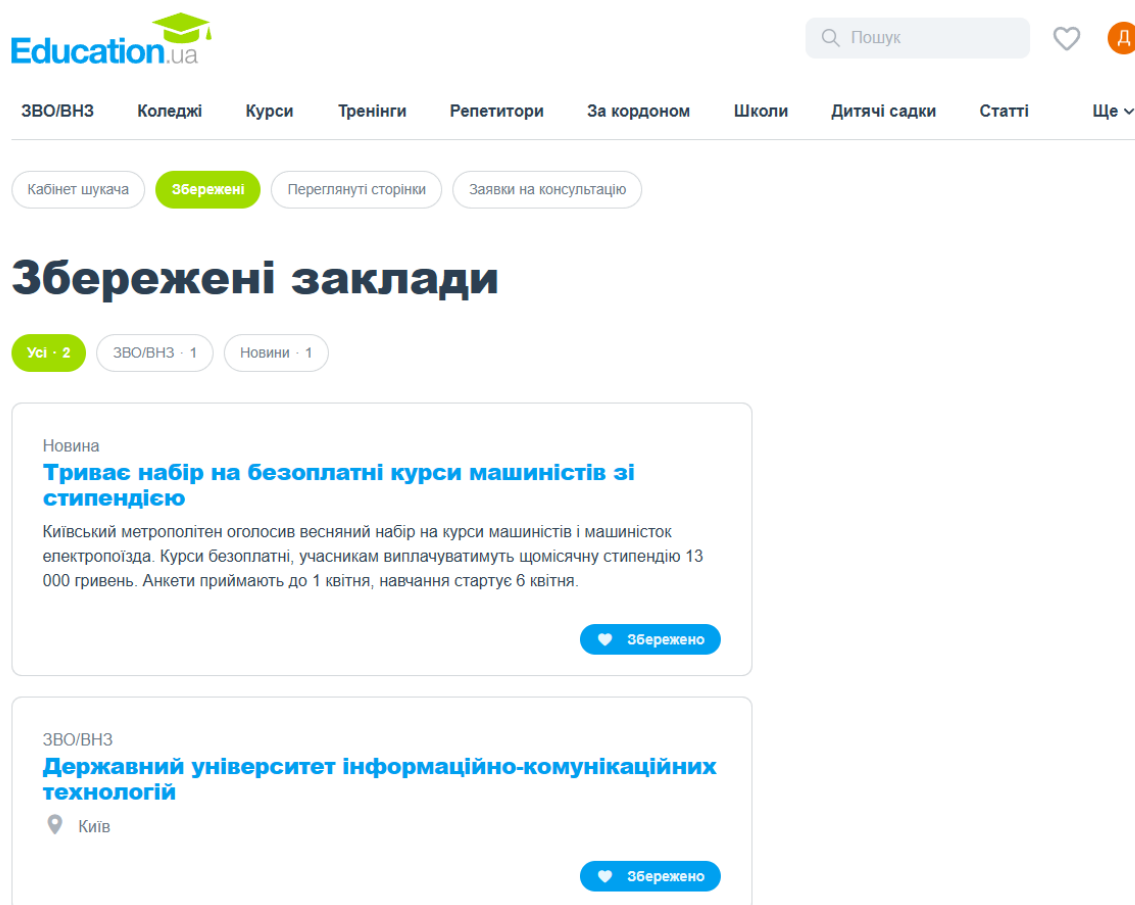


Рис. 1.5 Приклад збережених даних на сервісі Education.ua

Комплексного аналізу вступної кампанії немає. Конкурсний бал, статистика попередніх років, динаміка рейтингу, кількість заяв – все це існує розрізнено і ніде не зводиться разом. Повної картини вступного процесу платформа не дає. Користувач отримує шматки інформації і самостійно намагається скласти з них щось цілісне.

Дані оновлюються, але не в режимі реального часу. В активну фазу кампанії це відчутно – ситуація змінюється швидко, а платформа за нею не встигає. Рішення на основі даних які відстають від реальності – ризик.

Підсумок. Education.ua – довідник. Корисний коли потрібно знайти базову інформацію про університет або спеціальність. Але не більше. Немає розрахунків, немає персоналізації, немає комплексного аналізу. Для абітурієнта який хоче не просто переглянути інформацію, а прийняти обґрунтоване рішення – цього замало.

1.3.4 Abit-poisk

Abit-poisk – вузькоспеціалізований сервіс для відстеження заяв вступників в режимі реального часу [10]. На відміну від Abitly, Osvita.ua і Education.ua, цей інструмент не є інформаційним порталом – його єдина функція полягає в агрегації даних про заяви з ЄДЕБО і відображенні поточного стану конкурсу.

Головна задекларована перевага – відстеження заяв в реальному часі. Сервіс агрегує дані з ЄДЕБО і намагається показати поточну картину конкурсу та рейтингових списків. Проте є суттєве обмеження: пошук вступника в БД працює не завжди коректно – частина вступників просто не знаходиться в системі. Це знижує надійність сервісу як інструменту для прийняття рішень, оскільки користувач не може бути впевнений що бачить повну і актуальну картину. Приклад пошуку вступника Гаркавенко Д.О. 2022 року вступу за допомогою сервіру Abit-poisk зображено на рисунку 1.6.

The screenshot shows the Abit-poisk website interface. At the top, there is a yellow banner with the 'letyshops' logo and promotional text. Below the banner, the website header includes the URL 'ABIT-POISK.ORG.UA' and navigation links for 'Рейтингові списки 2025', 'Прохідні бали', 'Спеціальності', and 'Обране'. The main content area is titled 'Репетитор з математики • Чат • Канал' and features a search bar with the text 'Гаркавенко Д. О. 2022' and a 'ПОШУК :)' button. Below the search bar, a summary box shows 'Рік: 2022', 'Знайдено заяв: 1', and 'Показано: 1'. It also includes a link to the search results and options to 'СКОПЮВАТИ ЛІНК НА ЦЕЙ ПОШУК' and '☆ ДОДАТИ В ОБРАНЕ'. The main part of the page is a table with columns for 'ОКР', 'ПІБ', 'С', '№', 'П', 'Місця', 'Σ', 'СБО', 'Скл. кон. балу', 'ВНЗ', 'Ф', 'Спец.', 'Кв', and 'Д'. The table contains one row for 'Гаркавенко Д. О.' with various exam scores and a 'Прохідний бал на бюджет 2021: 184.273' note. At the bottom, there is a footer with copyright information and social media links.

ОКР	ПІБ	С	№	П	Місця	Σ	СБО	Скл. кон. балу	ВНЗ	Ф	Спец.	Кв	Д
Б	Гаркавенко Д. О.	До наказу (контракт)	7	1	ВМ 50 БМ _{МАХ} 6 К 44	140.400	—	Українська мова Математика Історія України	140 136 148	УНУС	ЕІП	071 Облік і оподаткування	—

Рис. 1.6 Пошук вступника Гаркавенко Д.О. 2022 року вступу за допомогою сервісу Abit-poisk

Проте вузька спеціалізація є одночасно і головним обмеженням. Сервіс працює виключно в період активної фази вступної кампанії – поза цим часом він практично не має цінності. Розрахунку конкурсного бала немає. Персоналізованих рекомендацій немає. Каталогів університетів і спеціальностей немає. Особистого кабінету немає. Користувач не може зберегти результати або повернутись до них пізніше.

Інтерфейс простий і функціональний – без зайвих елементів, орієнтований на одну задачу. Але саме через це він не може замінити повноцінний інструмент підтримки абітурієнта – він лише доповнює інші сервіси в конкретний момент кампанії.

Підсумок. Abit-poisk корисний як додатковий інструмент моніторингу під час активної фази вступу. Для комплексної підтримки абітурієнта – від вибору спеціальності до відстеження результатів – він не підходить через відсутність більшості необхідних функцій.

1.3.5 Висновки аналізу існуючих web-сервісів

Аналіз Abitly, Education.ua, Osvita.ua і Abit-poisk підтверджує одне: кожен сервіс закриває щось своє, але в сукупності картина не складається. Функції є – але розрізнені, неповні і не пов'язані між собою, що створює неповноцінну картину.

Розрахунок балів – є, але без прив'язки до конкурсної пропозиції. Абітурієнт отримує цифру, але не розуміє що вона означає в конкретному конкурсі. Рівень конкуренції, вагові коефіцієнти, кількість заяв – все це впливає на реальні шанси, але жоден із сервісів це не враховує в комплексі. Цифра є. Відповіді – немає.

Комплексного підходу немає ніде. Щоб зібрати повну картину – рейтинг, бал, статистику, актуальні дані – доводиться відкривати кілька платформ і зводити інформацію вручну. Це довго, незручно і підвищує ризик десь помилитись.

Персоналізації немає. Всі бачать одне й те саме – незалежно від балів, пріоритетів і конкретної ситуації. Рекомендацій під індивідуальні параметри немає в жодному з розглянутих рішень.

Пошук і фільтрація обмежені. Даних багато, але відібрати їх під кілька критеріїв одночасно – складно. Гнучких інструментів відбору немає, тому знайти оптимальний варіант швидко не виходить.

Збереження результатів – ще одна проблема. Накопичити дані, сформувати список обраних варіантів, порівняти їх і повернутись пізніше – цього немає. Кожен раз починаєш спочатку.

Актуальність даних теж під питанням. Вступна кампанія рухається швидко – рейтинги оновлюються, заяви подаються і відкликаються. Сервіси за цим не встигають. В найактивніший момент кампанії частина інформації вже застаріла.

Зведені результати аналізу функціональних можливостей розглянутих сервісів наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу функціоналу проаналізованих сервісів для допомоги абітурієнтам під час вступної кампанії

Критерій	Abitly	Education.ua	Osvita.ua	Abit-Poisk	STUDINFO
Розрахунок шансу на вступ до ЗВО	-	-	-	-	+
Рейтинг університетів	+	-	±	-	+
Пошук та фільтрація освітніх програм	±	±	-	-	+
Персоналізація	±	-	-	-	+
Особистий кабінет	+	-	±	-	+

Продовження таблиці 1.1

Зведені результати аналізу функціоналу проаналізованих сервісів для допомоги абітурієнтам під час вступної кампанії

Збереження даних користувача	-	-	±	±	+
Комплексний аналіз вступу (шанс на вступ, КБ, календар вступу)	-	-	-	-	+
Персоналізована рекомендація ЗВО та конкурсних пропозицій	±	-	-	-	+
Оновлення даних вступних кампанії в реальному часі	-	-	±	±	±
Час обробки запиту <2 сек.	-	+	+	-	±
Актуальність даних	±	±	±	±	+
Адаптивність інтерфейсу	+	+	+	±	+
Повнота наповнення бази даних	±	+	+	±	+

У підсумку можна зазначити, що існуючі рішення не дають абітурієнту повноцінного інструменту для роботи з даними вступної кампанії. Інформація є, але зібрати її до купи, проаналізувати і отримати конкретну відповідь – в жодному з них не вийде.

1.4 Визначення недоліків існуючих рішень та постановка задачі

Підсумовуючи аналіз конкурентів можна зазначити, що виявлених недоліків існуючих рішень недостатньо. Не тому що вони погані, а тому що кожне закриває лише частину задачі. Одні сервіси орієнтовані переважно на надання статистики вступу, інші – на пошук університетів або перегляд конкурсних пропозицій. У результаті абітурієнт змушений одночасно користуватися кількома різними платформами, вручну порівнювати дані, перевіряти прохідні бали, окремо розраховувати конкурсний бал та аналізувати шанси на вступ. Такий підхід ускладнює процес прийняття рішення та займає значну кількість часу. Особливо це відчутно під час активної фази вступної кампанії, коли інформація постійно оновлюється, а кількість можливих варіантів є дуже великою.

Саме це і визначає задачу розробки STUDINFO. Центральною ідеєю створення власного проєкту є спрощення процесу прийняття рішень під час вступу. Основна мета полягає не лише у наданні великого обсягу інформації, а у створенні єдиного середовища, де абітурієнт зможе отримати всі необхідні інструменти для аналізу та вибору. Система повинна не просто відображати дані, а допомагати працювати з ними: автоматизувати розрахунки, аналізувати результати та надавати зрозумілі рекомендації відповідно до індивідуальних параметрів користувача.

Важливо також враховувати, що сучасний вступник стикається з великою кількістю інформації, яка часто подається у складному та незручному форматі. Таблиці конкурсних пропозицій, коефіцієнти, рейтингові списки та прохідні бали потребують детального аналізу. Для багатьох користувачів, особливо тих, хто вперше проходить процес вступу, це створює додаткові труднощі та невизначеність. Саме тому STUDINFO має виконувати не лише інформаційну, але й консультативну функцію, допомагаючи користувачу краще орієнтуватися у вступній кампанії. Виходячи з цього, STUDINFO має реалізувати такі функції:

- розрахунок конкурсного бала з урахуванням введених результатів і параметрів конкретної конкурсної пропозиції;
- модуль «Помічник вступу»
- підбір університетів і спеціальностей під індивідуальні характеристики користувача;
- персоналізовані рекомендації щодо вибору закладу вищої освіти та освітніх програм;
- каталог університетів із деталями по кожній спеціальності: умови вступу, конкурсні показники, кількість місць;
- перегляд і аналіз конкурсних пропозицій із фільтрацією та сортуванням;
- рейтинги університетів і спеціальностей на основі актуальних даних;
- особистий кабінет для збереження розрахунків, обраних варіантів і їх порівняння;
- адаптація інформації під індивідуальні параметри користувача;
- адміністративна панель для керування даними і забезпечення роботи системи.

У підсумку можна зазначити, що STUDINFO не має стати ще одним довідником. Основною ідеєю є збір всіх даних та функцій в одному місці. Інструмент який не просто показує дані, а допомагає з ними працювати і приймати рішення.

Крім того, важливо що система розрахована не на разове використання. Вступна кампанія – це процес який розтягується на тижні. Абітурієнт повертається, перераховує, порівнює варіанти, змінює пріоритети. Особистий кабінет, збереження результатів, актуальні дані в режимі реального часу – все це не додаткові функції, а частина основної логіки продукту. STUDINFO має працювати не як сторінка яку відкрив і закрити, а як інструмент до якого повертаєшся протягом всієї кампанії.

2 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ WEB-SERVICE

2.1 Клієнтська частина web-сервісу

При розробці клієнтської частини web-сервісу STUDINFO критично важливим було обрати інструменти, що забезпечують швидке оновлення інтерфейсу при роботі з динамічними даними вступної кампанії, коректну обробку і валідацію введених користувачем параметрів, стабільну роботу при значній кількості одночасних запитів до серверної частини, інтуїтивну навігацію з мінімальною кількістю кроків для отримання результату, а також адаптивне відображення елементів інтерфейсу на різних типах пристроїв – як десктопних, так і мобільних.

2.1.1 Аналіз та вибір технологій клієнтської частини

За основу рендерингу клієнтською частиною було обрано Next.js - React-фреймворк з вбудованою маршрутизацією, автоматичною оптимізацією ресурсів і гнучкою організацією структури проєкту [11, 12]. App Router дозволяє будувати багатосторінкову архітектуру без додаткових бібліотек – кожна папка в директорії app автоматично стає маршрутом. Для застосунку з каталогами, сторінками конкурсних пропозицій, профілем користувача і модулем розрахунків це практичне рішення – кожен розділ організовується незалежно і масштабується окремо.

Додатковою перевагою Next.js є автоматична оптимізація ресурсів та гнучка організація компонентів. Фреймворк підтримує розділення компонентів на серверні та клієнтські, що спрощує побудову логіки застосунку та структурування проєкту. Також Next.js забезпечує зручну інтеграцію з API та підтримує масштабування застосунку. Це особливо важливо для системи STUDINFO, оскільки web-сервіс працює з великою кількістю динамічних даних та окремих функціональних модулів.

Детальну архітектуру Next.js зображено на рисунку 2.1.

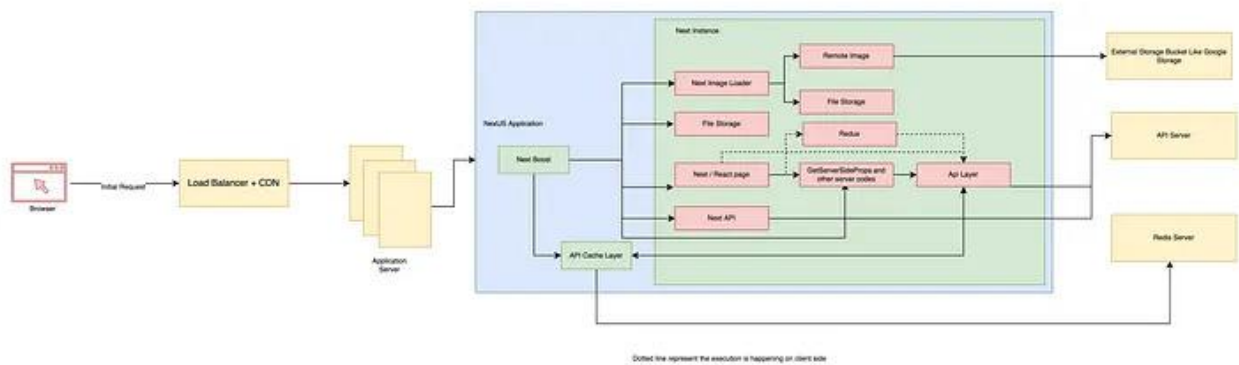


Рис. 2.1 Архітектури роботи Next.js

Не менш важливим фактором вибору Next.js став SSR (Server Side Rendering), який покращує швидкість завантаження сторінки в порівнянні з CSR (Client Side Rendering) [13, 14]. Схему роботи SSR зображено на рисунку 2.2.

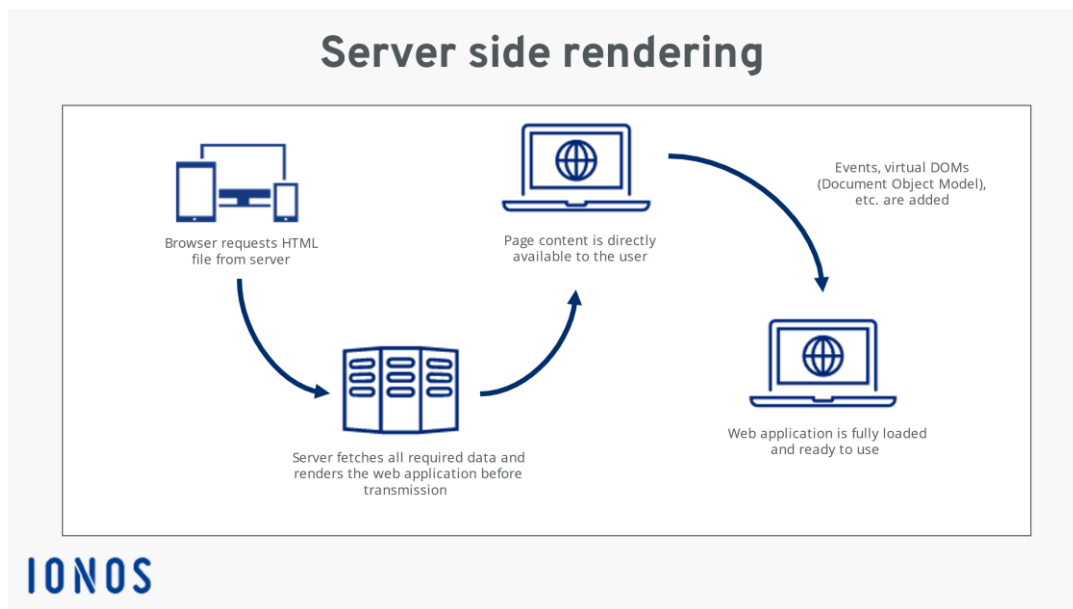


Рис. 2.2 Схему роботи SSR

Для підвищення надійності клієнтської частини доцільно використовувати TypeScript. TypeScript – надбудова над JavaScript з статичною типізацією [15, 16]. Компілюється у чистий JS, тому не вимагає змін у середовищі виконання.

Основна відмінність від JavaScript – перевірка типів відбувається на етапі компіляції, а не в runtime. Підтримує interface, generics, union types, type guards. Помилки невідповідності типів, некоректні звернення до властивостей і відсутні поля виявляються до запуску застосунку. У проєктах з великою кількістю сутностей і API-контрактів явна типізація суттєво зменшує кількість дефектів і спрощує рефакторинг.

Для стилізації інтерфейсу обрано Chakra UI – бібліотеку React-компонентів з вбудованою підтримкою тем, адаптивної верстки і доступності (accessibility) [17]. Кожен компонент – Button, Input, Modal, Table, Select, Drawer – має набір пропсів для налаштування зовнішнього вигляду без написання кастомного CSS. Підтримка responsive props дозволяє задавати різні значення для різних breakpoints безпосередньо в JSX – без медіа-запитів. Компоненти форм інтегруються з Formik і react-hook-form – валідаційні стани (error, success, loading) відображаються через вбудовані пропси FormControl, FormLabel і FormErrorMessage. Для складних UI-патернів – наприклад, фільтрації освітніх програм або відображення рейтингових списків – Chakra UI надає готові компоненти Stack, Grid, Flex і SimpleGrid, що спрощує верстку складних лейаутів.

Для керування серверним станом і взаємодії з API обрано RTK Query – інструмент, вбудований у Redux Toolkit [18]. RTK Query автоматично генерує хуки для отримання, кешування і оновлення даних – без написання boilerplate-коду для кожного запиту окремо. Кожен endpoint описується один раз через createApi – назва, метод, URL, параметри – і RTK Query автоматично генерує useGetQuery і useMutation хуки. Кешування працює "з коробки" – повторний запит до того самого endpoint з тими самими параметрами не йде на сервер, а повертає закешований результат. Інвалідація кешу реалізована через теги (providesTags, invalidatesTags) – після успішної мутації автоматично оновлюються всі пов'язані запити. Для платформи з рейтинговими списками, конкурсними пропозиціями і даними що постійно змінюються в період кампанії

– це критично важливо. Polling і refetchOnFocus дозволяють налаштувати автоматичне оновлення даних без додаткового коду.

Для розробки і документування UI-компонентів використано Storybook – інструмент для ізольованої розробки React-компонентів поза контекстом застосунку. Кожен компонент описується через stories – файли що визначають різні стани компонента: порожня форма, форма з помилкою, завантаження, успішний результат. Це дозволяє розробляти і тестувати компоненти незалежно від серверної частини і стану застосунку. Addon-essentials надає вбудовані інструменти для перегляду пропсів, зміни теми і перевірки адаптивності – без запуску повного застосунку.

Для керування формами обрано Formik – бібліотеку що обробляє стан форми, валідацію і сабміт. Відстежує touched, errors і values для кожного поля – без написання кастомних onChange і onBlur обробників. Інтегрується з Yup для schema-based валідації – правила описуються окремо від компонента і перевіряються автоматично. handleSubmit блокує сабміт при наявності помилок і викликає onSubmit лише при валідному стані форми.

Для анімацій інтерфейсу використано Framer Motion – бібліотеку декларативних анімацій для React. Анімації описуються через пропси initial, animate і exit безпосередньо на компоненті – без написання CSS keyframes. AnimatePresence обробляє exit-анімації при розмонтуванні компонентів – наприклад, при закритті модального вікна або переході між сторінками. layout prop автоматично анімує зміни позиції і розміру елемента при зміні лейауту.

2.1.2 Підходи до побудови клієнтської частини web-сервісу

Клієнтська частина STUDINFO побудована на основі компонентного підходу – UI розбитий на незалежні блоки, кожен з яких інкапсулює власну логіку, стан і розмітку. Компоненти діляться на presentational – отримують дані через пропси і відповідають лише за відображення, та container – містять логіку і взаємодіють зі стором. Такий підхід дозволяє змінювати кожен модуль незалежно – без впливу на решту системи.

Для маршрутизації використано App Router – підхід до організації сторінок у Next.js, що прийшов на заміну Pages Router. На відміну від Pages Router де маршрути визначались файлами у папці `pages/`, App Router використовує директорію `app/` де кожна папка з файлом `page.tsx` автоматично стає маршрутом. App Router підтримує серверні компоненти за замовчуванням – це дозволяє виконувати запити до бази даних і API безпосередньо на сервері без додаткового API-шару. `layout.tsx` визначає спільний лейаут для групи сторінок – header, footer, sidebar – і рендериться один раз без перемонтування при навігації між сторінками. `global-error.tsx` обробляє критичні помилки на рівні застосунку і відображає fallback UI замість порожнього екрану. Для SEO Next.js надає Metadata API – title, description, og-теги описуються через об'єкт `metadata` або функцію `generateMetadata` безпосередньо в `page.tsx` без додаткових бібліотек. Детальну схему App Router маршрутизації в Next.js наведено на рисунку 2.3.

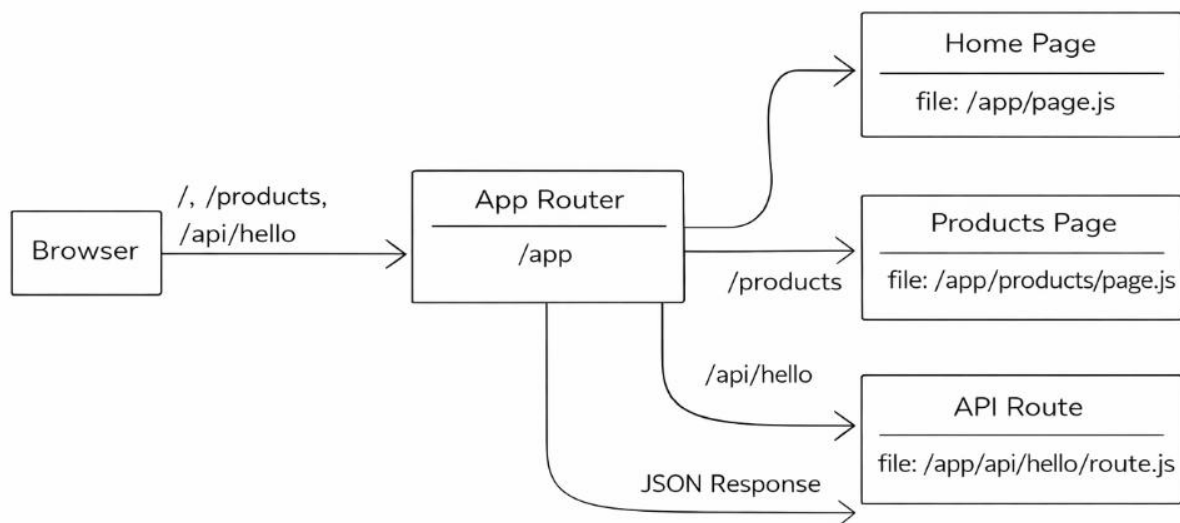


Рис. 2.3 Схема App Router маршрутизації в Next.js

Для отримання серверних даних і їх відображення на фронті використано RTK Query. `useGetQuery` повертає кілька станів одночасно – `data` містить отримані дані, `isLoading` спрацьовує при першому завантаженні, `isFetching` – при повторному запиті з наявним кешем, `isError` – при помилці мережі або сервера.

Це дозволяє відображати різні UI-стани – скелетони при завантаженні, повідомлення про помилку при збої, актуальні дані після успішного запиту – без додаткового boilerplate-коду.

2.2 Серверна частина web-сервісу

В процесі розробки серверної частини web-сервісу STUDINFO ключовим завданням був вибір інструментів, що забезпечують надійну обробку бізнес-логіки і API-запитів від клієнтської частини, безпечне зберігання і передачу персональних даних користувачів, асинхронне виконання ресурсоємних операцій без блокування основного потоку, коректну серіалізацію і валідацію даних на рівні сервера, а також стабільну роботу системи в умовах пікового навантаження в активну фазу вступної кампанії.

2.2.1 Аналіз та вибір технологій серверної частини

За основу серверної частини проєкту було обрано Ruby – динамічно типізована інтерпретована мова програмування з чистим об'єктно-орієнтованим синтаксисом [19, 20]. Версія 3.3.2 використана у проєкті – вона включає покращення продуктивності інтерпретатора YJIT і оновлений garbage collector. Ruby має розвинену екосистему бібліотек (gems) що дозволяє швидко підключати готові рішення для типових задач – автентифікація, серіалізація, пагінація, фонові задачі.

Як основний фреймворк було обрано Ruby on Rails 7.1.2, що реалізує патерн MVC і принципи Convention over Configuration та DRY [19, 20]. ActiveRecord забезпечує ORM-шар для роботи з PostgreSQL – SQL-запити генеруються автоматично через Ruby-методи. Action Controller обробляє вхідні HTTP-запити і формує відповіді. Puma використовується як веб-сервер – підтримує многопоточну обробку запитів і є дефолтним для Rails 7.

Blueprinter – бібліотека для серіалізації Ruby-об'єктів у JSON. Кожен серіалізатор описує які поля моделі включати у відповідь – це дозволяє

контролювати структуру API-відповіді і не повертати зайвих даних клієнту. Pagy – бібліотека для пагінації з мінімальним споживанням пам'яті порівняно з will_paginate і kaminari. Faraday – HTTP-клієнт для виконання запитів до зовнішніх API з підтримкою middleware, retry-логіки і cookie jar через faraday-cookie_jar.

Для веб черг було обрано Sidekiq 8.0.3 – інструмент для виконання фонових задач через черги Redis [21]. sidekiq-scheduler додає підтримку cron-розкладу – задачі запускаються автоматично за розкладом без додаткових інструментів. Redis 5.2 використовується як сховище черг для Sidekiq. Nokogiri – бібліотека для парсингу HTML і XML – використовується для отримання даних з зовнішніх освітніх ресурсів. aws-sdk-s3 забезпечує інтеграцію з Amazon S3 для зберігання файлів. ruby-openai у поєднанні з neighbor і nemo-narray реалізує векторний пошук для модуля персоналізованих рекомендацій – neighbor забезпечує пошук найближчих векторів у базі даних, nemo-narray виконує математичні операції над масивами.

2.2.2 Підходи до організації серверної логіки та API

В процесі розробки взаємодію між клієнтською і серверною частинами вирішено організувати через REST API – архітектурний стиль обрано через його простоту, стандартизованість і широку підтримку в екосистемі Rails. HTTP-методи GET, POST, PATCH, DELETE відповідають за операції з ресурсами: університети, спеціальності, конкурсні пропозиції, користувачі. JSON-відповіді формуються через Blueprinter-серіалізатори – на відміну від Active Model Serializers, Blueprinter має простіший синтаксис і вищу продуктивність при серіалізації великих колекцій.

Такий підхід дозволяє чітко розділити відповідальність між Frontend і Backend та спрощує подальшу підтримку системи. REST API також забезпечує універсальність взаємодії між компонентами та можливість подальшого масштабування платформи без зміни серверної логіки.

Приклад архітектури та принципів роботи REST API продемонстровано на рисунку 2.4

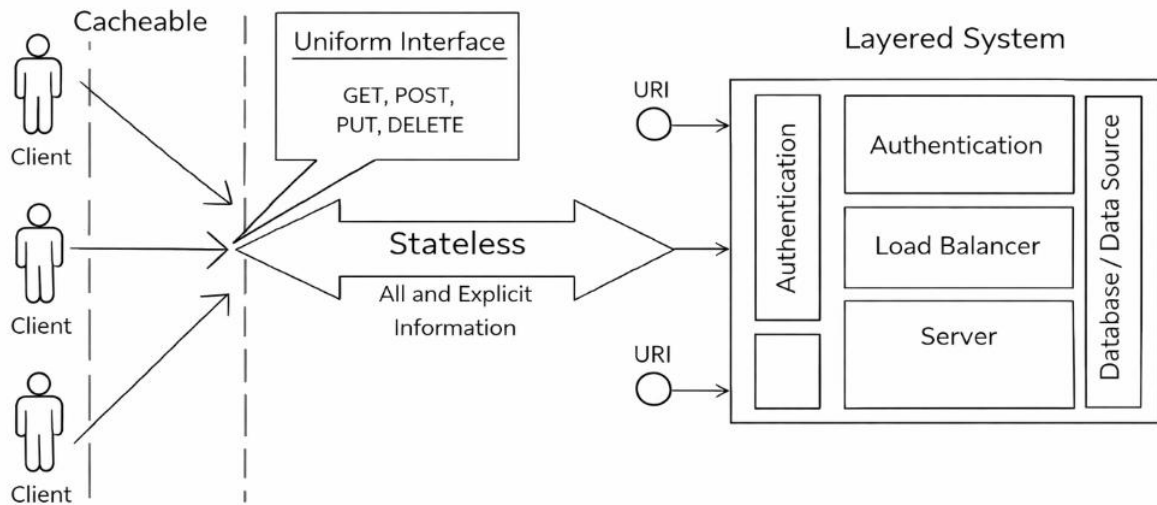


Рис. 2.4 Приклад архітектури та принципів роботи REST API

Для документування API обрано rswag – єдиний інструмент що дозволяє генерувати Swagger-документацію безпосередньо з RSpec-тестів без дублювання опису endpoints. Swagger UI доступний через окремий endpoint і дозволяє тестувати API без Postman чи інших зовнішніх інструментів.

Для обробки міжсайтових запитів від Next.js підключено rack-cors – стандартне рішення для Rails API що налаштовує дозволені домени, методи і заголовки. Альтернативи на рівні nginx потребують додаткової конфігурації інфраструктури – rack-cors вирішує задачу на рівні застосунку.

Бізнес-логіка організована за патерном MVC з виносом складних операцій у Service Objects – розрахунок конкурсного бала, формування рекомендацій через OpenAI, парсинг зовнішніх даних через Nokogiri. Такий підхід обрано щоб уникнути fat controllers і забезпечити можливість незалежного тестування кожної операції.

Для асинхронного виконання задач обрано Sidekiq – на відміну від Delayed Job і Resque, Sidekiq використовує потоки замість процесів що забезпечує нижче споживання пам'яті при великій кількості задач. Redis обрано як сховище черг

через швидкість операцій з пам'яттю і нативну підтримку в Sidekiq. sidekiq-scheduler додає cron-розклад – щоденне оновлення конкурсних пропозицій і генерація рекомендацій запускаються автоматично.

2.2.3 Забезпечення безпеки даних та контролю доступу

STUDINFO працює з персональними даними користувачів – результатами НМТ, анкетами, параметрами конкурсного відбору. Це визначає підвищені вимоги до захисту даних на всіх рівнях системи – автентифікація, авторизація, передача і зберігання. Проаналізувавши доступні підходи і бібліотеки, було реалізовано багаторівневу систему захисту.

Для автентифікації використано Devise 4.9 – gem що реалізує реєстрацію, вхід, відновлення паролю і управління сесіями через стратегії Warden [22]. Паролі хешуються bcrypt з унікальним salt – відновлення оригінального значення з хешу є обчислювально нерентабельним. Для контролю доступу обрано Pundit і CanCanCan – Pundit описує дозволи через policy-клас для кожної моделі, CanCanCan централізує права через Ability і перевіряє їх через authorize! в контролерах.

Захист від типових web-загроз реалізовано на кількох рівнях [23]. SQL-ін'єкції виключено через параметризовані запити Active Record. XSS-атаки блокуються на рівні React – DOM оновлюється через віртуальне дерево без прямої вставки HTML. CSRF-токен перевіряється Rails при кожному POST, PATCH і DELETE запиті [24]. Передача даних між клієнтом і сервером здійснюється виключно через HTTPS.

2.3 Система керування базою даних

PostgreSQL – об'єктно-реляційна СУБД з відкритим кодом, що підтримує ACID-транзакції, foreign keys, складні JOIN-запити, віконні функції, повнотекстовий пошук і JSON-поля [25]. На відміну від MongoDB що використовує документо-орієнтовану модель без жорстких зв'язків між

колекціями, PostgreSQL забезпечує referential integrity через foreign key constraints – критично для платформи з ієрархічною структурою даних: University → Specialty → CompetitiveOffer → Application [26]. MySQL не розглядався як альтернатива через обмежену підтримку віконних функцій і менш розвинений повнотекстовий пошук порівняно з PostgreSQL [27]. Схему архітектури та принцип роботи PostgreSQL продемонстровано на рисунку 2.5

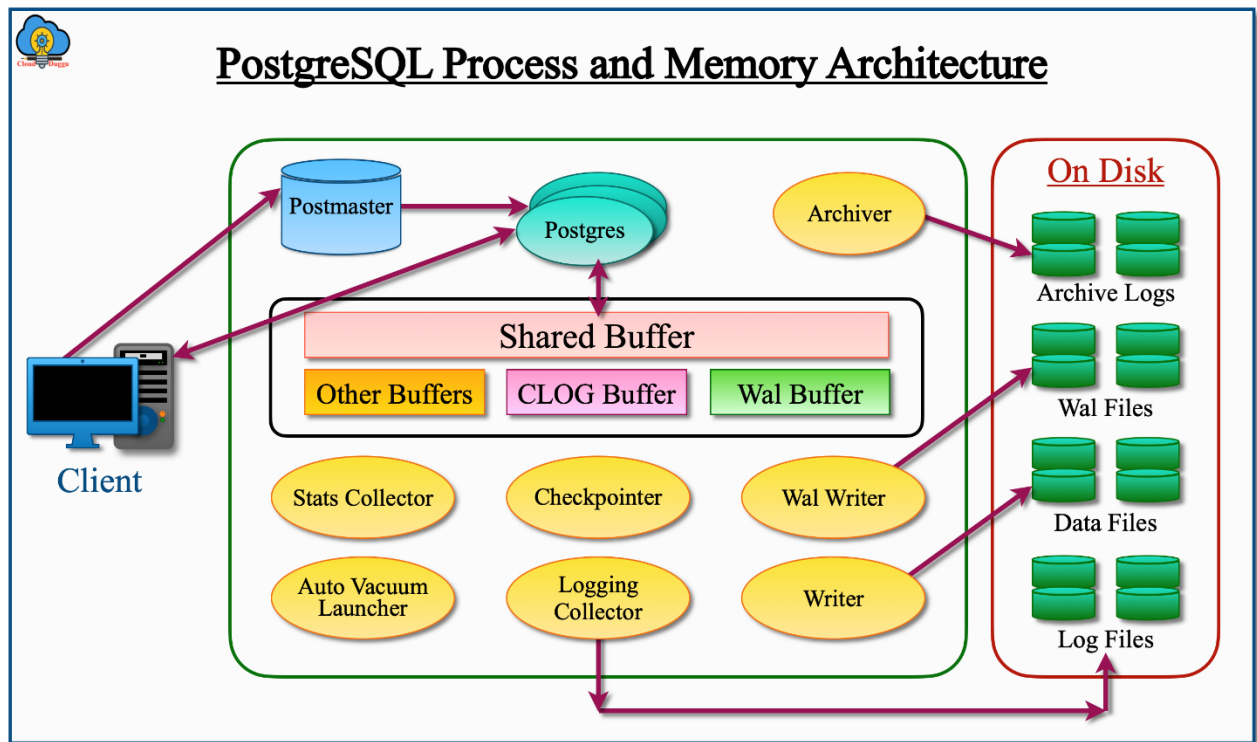


Рис 2.5 Архітектура та принцип роботи PostgreSQL

2.4 Панель адміністрування системою

Для адміністрування даних платформи використано Active Admin – gem для Rails що генерує адміністративний інтерфейс на основі моделей Active Record. Реєструється кожна модель через `ActiveAdmin.register` – автоматично створюється CRUD-інтерфейс з переліком записів, фільтрами, пошуком і формою редагування. У проєкті через адмін-панель керуються всі основні сутності бази даних – анкети користувачів, університети, спеціальності, конкурсні пропозиції. `activeadmin_json_editor` підключено для редагування

JSON-полів через візуальний інтерфейс – без прямого редагування рядка. Доступ до /admin обмежено через `Devise authenticate_admin!` – неавторизовані запити перенаправляються на сторінку входу.

2.5 Засоби розгортання web-сервісу

Для розгортання фронтенду обрано Vercel – платформу, оптимізовану під Next.js з автоматичним деплоєм при пуші в GitHub [28]. Налаштовано два середовища – staging на гілці development і production на гілці main. При кожному пуші Vercel автоматично виконує збірку та розгортання нової версії застосунку без необхідності ручного втручання. Це дозволяє значно спростити процес оновлення системи та швидко перевіряти внесені зміни. Крім того, Vercel забезпечує підтримку серверного рендерингу, автоматичну оптимізацію ресурсів та коректну роботу функціоналу Next.js без додаткового налаштування серверної інфраструктури. Також платформа дозволяє швидко створювати preview-версії застосунку для тестування нових функцій перед їх публікацією у production. Так як Vercel повністю орієнтований на Next.js, це є одним із найбільш практичних рішень для розгортання клієнтської частини проєкту.

Для розгортання бекенду обрано Railway з використанням Docker – серверна частина упакована в контейнер разом з усіма залежностями, що гарантує ідентичне середовище виконання на локальній машині і в production [29]. PostgreSQL підключено як окремий Railway-сервіс – налаштовано два інстанси бази даних: staging для гілки development і production для гілки main. Sidekiq запущено як окремий процес на Railway – він підключається до Redis та обробляє фонові задачі. Такий підхід дозволяє безпечно тестувати зміни перед деплоєм у production, спрощує підтримку системи та забезпечує стабільну роботу веб-сервісу.

Обрана інфраструктура дозволяє автоматизувати процес розгортання та зменшити кількість помилок під час оновлення системи.

3 РЕАЛІЗАЦІЯ WEB-SERVICE

3.1 Проектування загальної архітектури web-сервісу

Проектування архітектури STUDINFO передбачало визначення структури взаємодії між компонентами системи, розподілу відповідальності між клієнтською і серверною частинами та сценаріїв роботи користувачів з платформою. На основі функціональних вимог розроблено діаграму варіантів використання, яку зображено на рисунку 3.1.

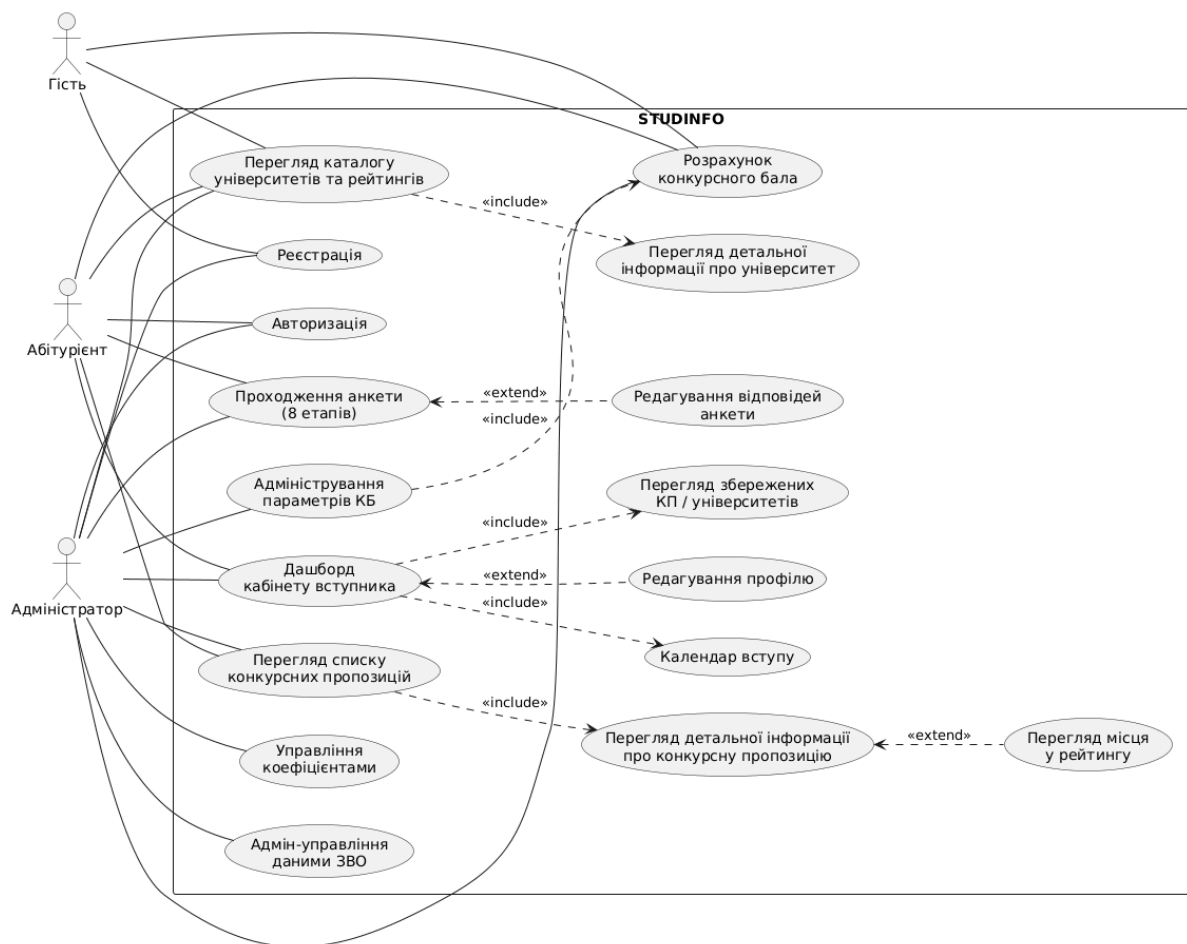


Рис.3.1 Діаграма варіантів використання web-сервісу

Діаграма варіантів використання визначає три типи акторів – зареєстрований абітурієнт, гість і адміністратор. Гість має доступ до публічних

функцій без автентифікації – каталог університетів, рейтинг університетів і пошук та фільтрація програм. Зареєстрований абітурієнт отримує доступ до повного функціоналу платформи – реєстрація і авторизація, проходження анкети з 8 етапів, перегляд дашборду, перегляд списку КП з можливістю розширення до детальної інформації через зв'язок *extend*, розрахунок конкурсного бала що включається в перегляд дашборду через зв'язок *include*, додавання університетів до обраних і перегляд обраних університетів. Адміністратор має доступ до окремих функцій керування – КБ, коефіцієнти і адміністрування даних ЗВО.

Після визначення варіантів використання системи було розроблено діаграму послідовності web-сервісу, яку зображено на рисунку 3.2.

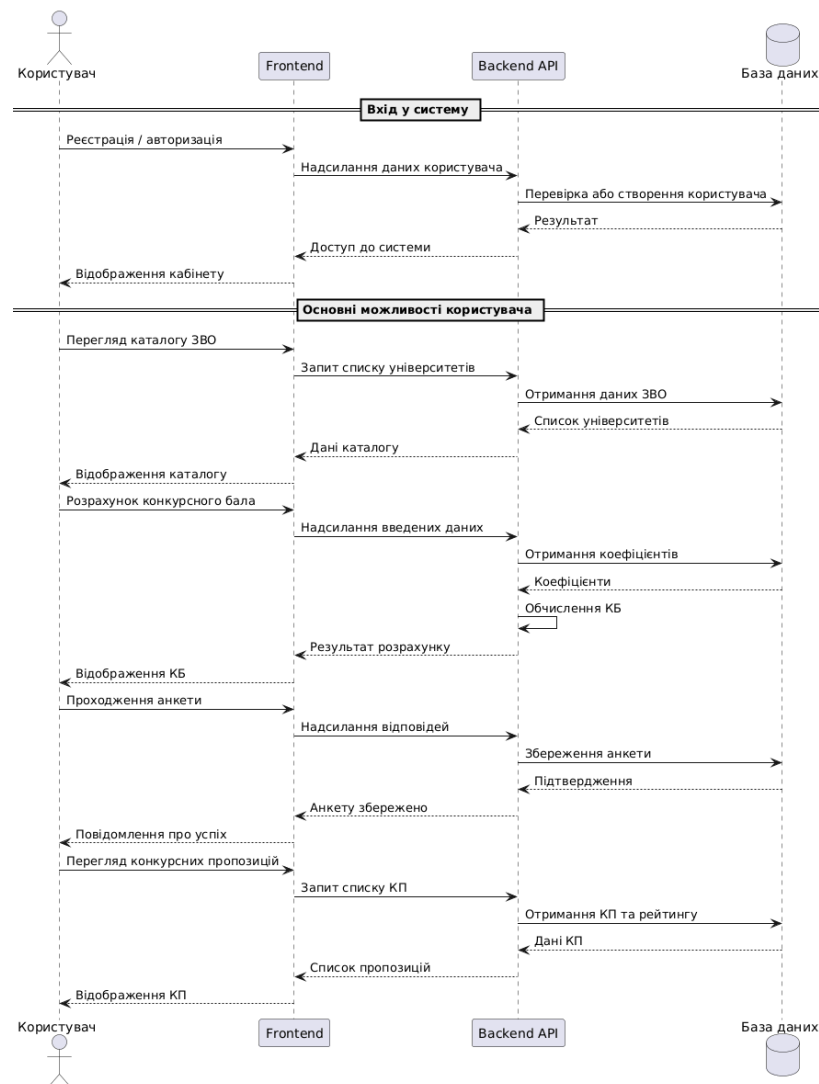


Рис. 3.2 Діаграма послідовності web-сервісу

Діаграма послідовності відображає порядок взаємодії між користувачем, клієнтською частиною, серверною логікою та базою даних. Діаграма демонструє процес обробки основних запитів системи – авторизацію, отримання даних університетів, розрахунок конкурсного бала, проходження анкети та роботу з конкурсними пропозиціями. Це дозволяє наочно показати механізм обміну даними між компонентами STUDINFO та послідовність виконання ключових операцій системи.

Також було розроблено діаграму діяльності користувача, яка описує повний користувацький флоу від відкриття платформи до роботи з результатами. При відкритті STUDINFO система перевіряє чи зареєстрований користувач. Незареєстрований користувач перенаправляється на сторінку реєстрації – заповнення форми і створення акаунту. Зареєстрований користувач проходить авторизацію і переходить до анкети. Анкета реалізована як цикл з 8 етапів – система перевіряє чи залишились питання і переходить до наступного етапу до завершення. Після завершення анкети результати зберігаються і виконується розрахунок конкурсного бала.

На дашборді формується персоналізований список конкурсних пропозицій – користувач може переглянути список, обрати конкретну КП, переглянути свій конкурсний бал, місце в рейтингу і всі необхідні деталі про університет. Паралельно доступний каталог університетів і перегляд рейтингу – авторизований користувач може додати університет до обраних. Такий підхід дозволяє об'єднати всі основні функції системи в межах одного інтерфейсу та забезпечує зручну навігацію між різними модулями платформи.

Крім того, діаграма діяльності демонструє послідовність взаємодії користувача із системою та логіку переходів між окремими етапами роботи застосунку. Це дозволяє краще зрозуміти структуру користувацького сценарію та визначити основні точки взаємодії між клієнтською та серверною частинами системи. Діаграму діяльності користувача зображено на рисунку 3.3.

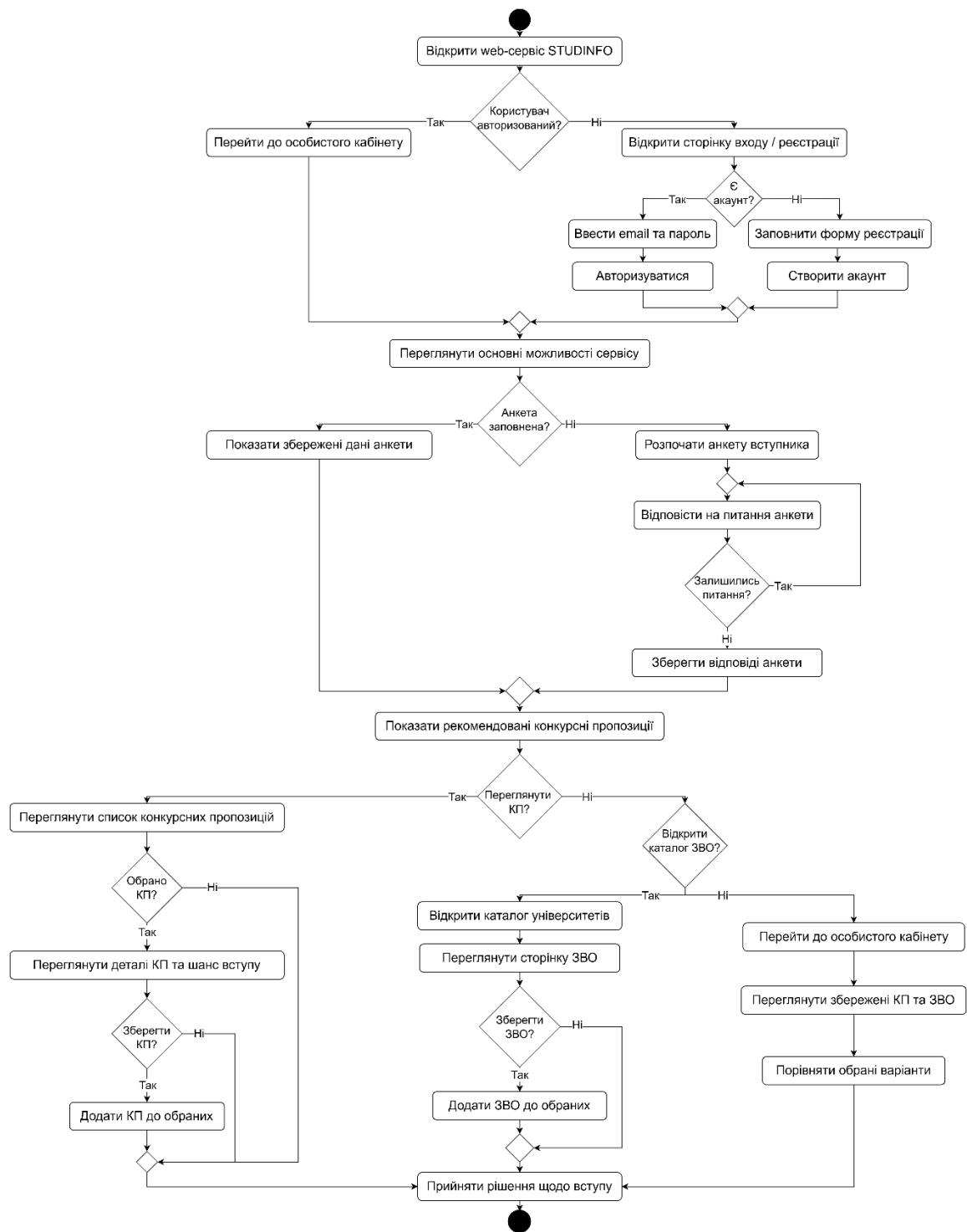


Рис.3.3 Діаграма діяльності користувача web-сервісу

3.1.1 Структура бази даних

База даних web-сервісу спроектована за реляційною моделлю та складається з набору взаємопов'язаних сутностей, що забезпечують збереження користувацьких даних, інформації про заклади освіти, освітні пропозиції, взаємодію користувачів із системою та аналітичні дані.

Користувачі та пов'язані сутності

Центральною сутністю системи є таблиця User, яка містить основну інформацію про користувачів (ідентифікатор, електронну пошту, ім'я та тип користувача). Тип користувача реалізований у вигляді переліку (enum) і визначає роль користувача в системі (абітурієнт, студент або інший). З таблицею User пов'язані наступні сутності:

AdminRole – реалізує розширення прав доступу користувача (зв'язок один до одного);

Questionnaire – анкета користувача, що містить відповіді у форматі JSON та статус проходження (зв'язок один до одного);

FavoriteUniversity – зберігає обрані користувачем університети (зв'язок один до багатьох);

ChatSession – сесії взаємодії користувача з чат-системою (зв'язок один до багатьох);

Ahoj::Visit – записи про відвідування системи (зв'язок один до багатьох, необов'язковий).

Сутність Ahoj::Visit пов'язана з таблицею Ahoj::Event, що зберігає події користувача (зв'язок один до багатьох).

Анкетування та рекомендації

Сутність Questionnaire використовується для збереження відповідей користувача, які застосовуються для формування рекомендацій. З нею пов'язані:

- QuestionnaireOffer – зв'язок між анкетною та освітніми пропозиціями (один до багатьох);

- FavoriteOffer – обрані користувачем пропозиції (один до багатьох);

- Offer – реалізовано зв'язок багато до багатьох через проміжну таблицю offers_questionnaires.

Заклади освіти

Сутність Region визначає регіон та має зв'язок один до багатьох із таблицею University. Таблиця University містить інформацію про заклади освіти та пов'язана з:

- UniversityInfo (один до одного) – розширена інформація;
- Offer (один до багатьох) – освітні пропозиції;
- Commission (один до багатьох) – контактна інформація приймальних комісій;
- Educator (один до багатьох) – викладачі;
- FavoriteUniversity (один до багатьох) – обрані користувачами університети.

Освітні пропозиції

Сутність Offer є ключовою для представлення освітніх програм. Вона містить атрибути, що описують тип фінансування, форму навчання та видимість. Особливості зв'язків:

- зв'язок один до багатьох із таблицею Request, яка зберігає заявки;
- зв'язок один до одного з таблицею OffersExamsId, що містить перелік ідентифікаторів вступних іспитів;
- рекурсивний зв'язок із самою собою через поле last_year_offer_id, що дозволяє відстежувати відповідні пропозиції попередніх років.

Додаткові сутності

GeneralData – окрема таблиця без зв'язків, призначена для збереження загальних даних системи.

Особливості проєктування

При проєктуванні бази даних були враховані наступні принципи:

- нормалізація структури для уникнення дублювання даних;
- використання зовнішніх ключів для забезпечення цілісності даних;
- застосування enum-полів для обмеження допустимих значень;
- використання JSONB-полів для збереження гнучких структур даних;
- реалізація як простих (1:1, 1:N), так і складних (M:N, рекурсивних) зв'язків.

Детальну структуру бази даних показано на рисунку 3.4.

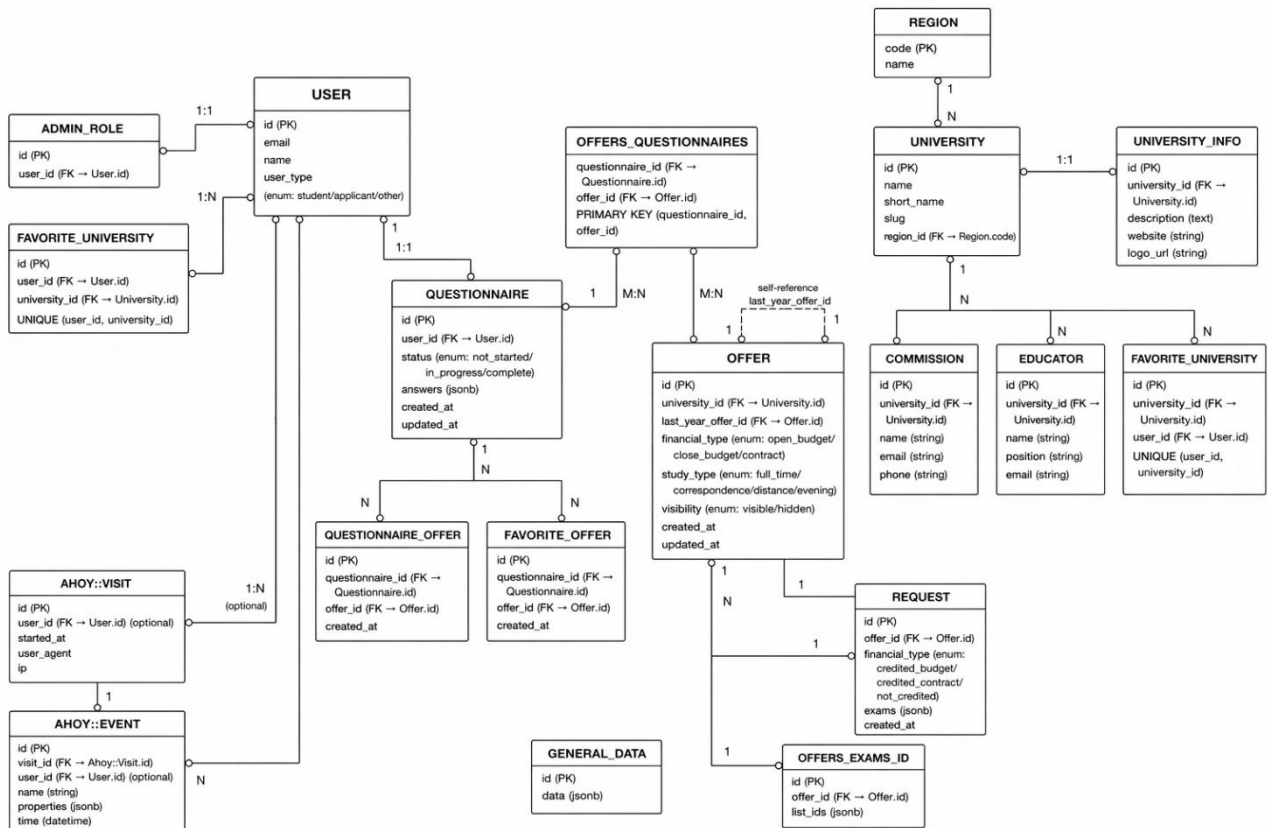


Рис. 3.4 Схема зв'язків між сутностями бази даних web-сервісу

3.1.2 Структура бекенд частини

Серверна частина STUDINFO організована за стандартною структурою Rails-застосунку з розширенням через Service Objects і Blueprinter-серіалізатори. Побудова серверної логіки за модульним принципом дозволила розділити відповідальність між окремими компонентами системи та спростити підтримку проекту. Основна бізнес-логіка винесена з контролерів у окремі сервіси, що покращує читабельність коду та забезпечує повторне використання функціональності в різних частинах системи. Окремо реалізовано структуру для роботи з API, адміністративною панеллю та серіалізацією даних. Такий підхід забезпечує кращу масштабованість серверної частини та спрощує подальше розширення функціоналу web-сервісу. Організацію та структуру серверної частини web-сервісу "STUDINFO" зображено на рисунку 3.5.



Рис. 3.5 Структура Rails у web-сервісу

Директорія `app/admin` містить конфігурацію `ActiveAdmin` для кожної моделі – `blogs`, `courses`, `offers`, `questionnaires`, `universities`, `users` та інші. Кожен файл реєструє відповідний ресурс в адмін-панелі і визначає доступні фільтри, колонки і форми редагування.

Директорія `app/blueprints` містить серіалізатори для формування JSON-відповідей API – `university_blueprint`, `offer_blueprint`, `questionnaire_blueprint`,

`user_blueprint` та інші. Кожен `blueprint` явно описує які поля моделі включаються у відповідь і дозволяє формувати різні представлення одної моделі – наприклад `university_blueprint` для списку і `university_full_blueprint` для детальної сторінки, зображено на рисунку 3.6.



Рис. 3.6 Структура `app/blueprints` директорії

Директорія `app/controllers` організована за `namespace`-ами – `api/` містить контролери для REST API, `nmt_calculator/` для калькулятора конкурсного бала, `universities/` для роботи з університетами і обраними програмами, `users/` для реєстрації і сесій, `tg/` для Telegram `webhook`-ів. На рівні `app/` розміщені основні контролери – `offers_controller`, `universities_controller`, `questionnaire_controller` та інші, зображено на рисунку 3.7.

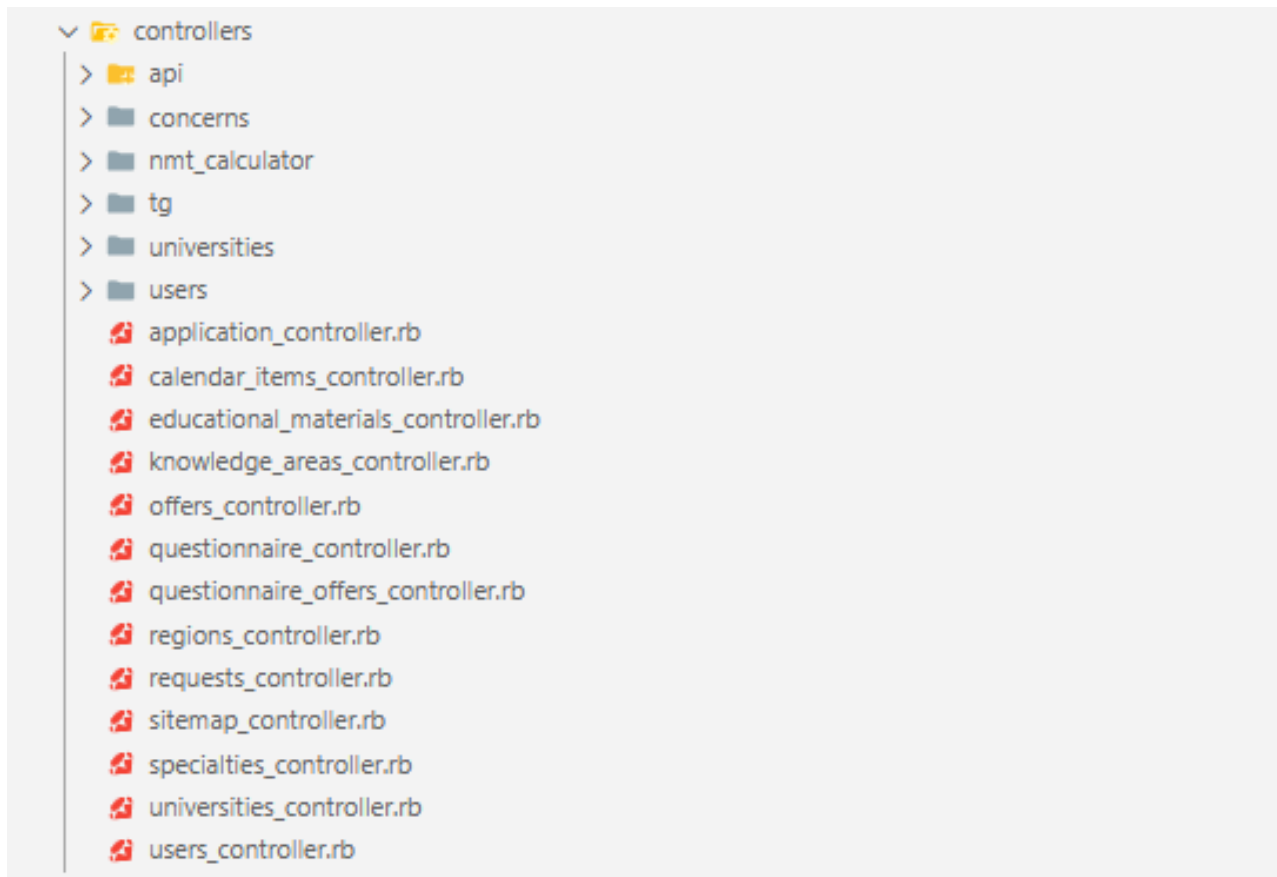


Рис. 3.7 Структура app/controllers директорії

Директорія app/jobs містить фонові задачі що виконуються через Sidekiq. Задачі організовані за namespace-ами – edebo/year2025/ для парсингу даних ЄДЕБО, questionnaire/ для оновлення анкет і розрахунку статистики. Серед ключових задач – parser_requests_job для парсингу заяв, import_specialties_coefficients_job для імпорту коефіцієнтів, update_offers_statistic_job для оновлення статистики конкурсних пропозицій.

Директорія app/models містить моделі що відповідають таблицям бази даних – University, Offer, Questionnaire, User, Specialty, Region, Commission та інші. Директорія app/services організована за функціональними namespace-ами – edebo/ для сервісів парсингу і оновлення даних ЄДЕБО, questionnaire/ для розрахунку балів і фільтрації, questionnaire_offer/ для розрахунку і збереження результатів, universities/ для оптимізації зображень і генерації slug-ів, зображено на рисунку 3.8.

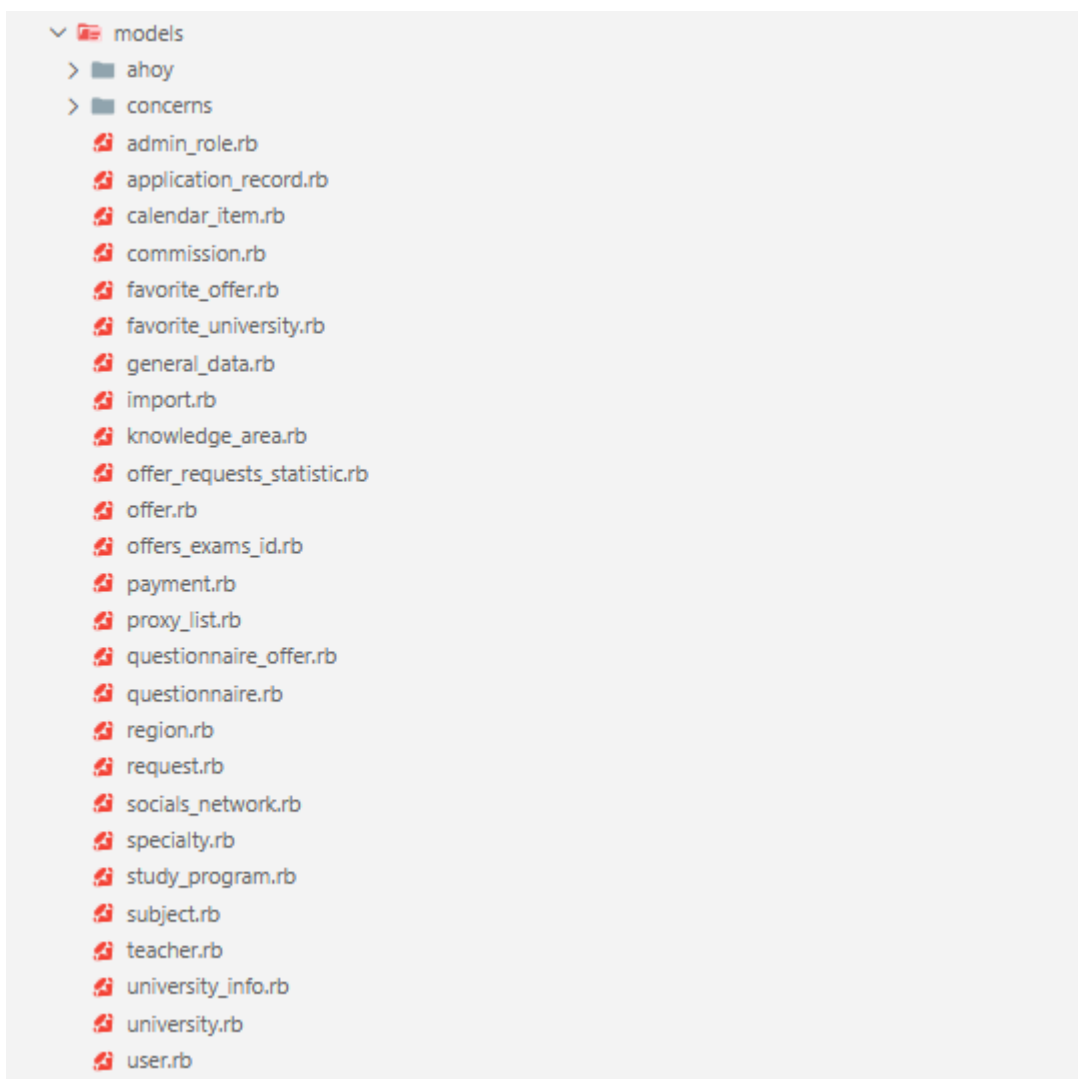


Рис. 3.8 Структура app/models директорії

3.1.3 Структура фронтенд частини

Клієнтська частина STUDINFO організована за структурою Next.js App Router з розділенням на сторінки, компоненти, хуки і утиліти, що дозволяє логічно розділити функціональні модулі застосунку та спростити підтримку проєкту при подальшому розширенні функціоналу. Окремо організовано структуру для роботи з API, перевикористовуваними UI-компонентами, валідацією форм та глобальним управлінням станом застосунку. Крім того, використання App Router забезпечує більш гнучку організацію маршрутизації та підтримку сучасного підходу до побудови React-застосунків. Детальну структуру клієнтської частини web-сервісу зображено на рисунку 3.9.

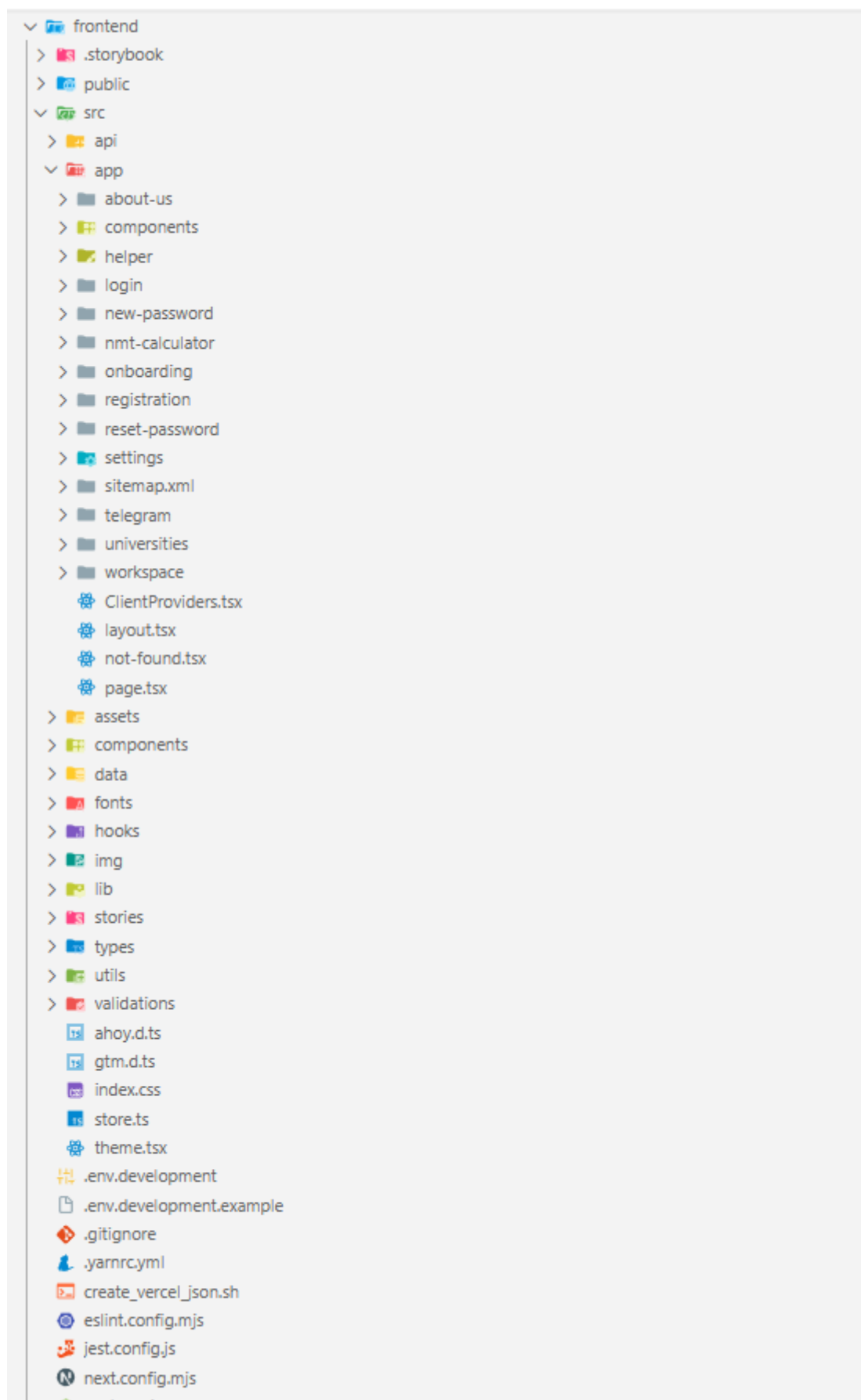


Рис. 3.9 Структура Next.js app routing у web-сервісу

Директорія `src/app` містить сторінки застосунку організовані за маршрутами – `about-us`, `login`, `registration`, `reset-password`, `new-password`, `onboarding`, `settings`, `nmt-calculator`, `universities`, `workspace`, `telegram`. Кожна папка містить `page.tsx` – основний компонент сторінки та `layout.tsx` – основний шаблон розмітки для директорії. Кореневі файли `layout.tsx`, `not-found.tsx` і `page.tsx` визначають загальний лейаут застосунку, сторінку 404 і головну сторінку відповідно. `ClientProviders.tsx` інкапсулює всі провайдери – `Redux store`, `Chakra UI` тема, інші контексти – і підключається в `layout.tsx`.

Директорія `src/api` містить `RTK Query endpoint`-и організовані за ресурсами – `helpers.ts` для допоміжних запитів, `nmtCalculator.ts` для калькулятора конкурсного бала, `user.ts` для роботи з профілем користувача.

Директорія `src/components` містить `shared`-компоненти що використовуються на кількох сторінках. `src/hooks` містить кастомні `React-хуки`. `src/types` містить `TypeScript`-типи і інтерфейси для всіх сутностей – університети, спеціальності, конкурсні пропозиції, користувачі. `src/validations` містить `Yup`-схеми для валідації форм. `src/utills` містить допоміжні функції. `src/lib` містить конфігурацію бібліотек.

Кореневі файли `store.ts` визначає `Redux store` з підключеними `slice`-ами і `RTK Query API`, `theme.tsx` містить кастомну `Chakra UI` тему, `index.css` – глобальні стилі.

3.1.4 Проектування API-архітектури

API серверної частини `STUDINFO` організовано через `scope '/api'` в `routes.rb` – всі `endpoint`-и згруповані під єдиним префіксом і розділені за ресурсами.

Автентифікація реалізована через `Devise` з кастомними контролерами – `users/sessions` для входу і виходу, `users/passwords` для відновлення паролю, `users/registrations` для реєстрації. Endpoint `GET /api/users/current` повертає поточного користувача через `UserBlueprint` з `view :public` – `id`, `email`, `name`, `user_type`, `questionnaire_id` і `questionnaire_status`.

Університети доступні через GET /api/universities з підтримкою param :slug – деталі університету запитуються через slug а не числовий id. Вкладені ресурси universities/:slug/favorite і universities/:slug/study_programs організовані через scope module: :universities. Конкурсні пропозиції – GET /api/offers/:id для детальної сторінки і GET /api/offers/popular для публічного списку без автентифікації. Анкета користувача – GET /api/questionnaire/:id з вкладеними offers, regions і questionnaire_offers.

Серіалізація відповідей реалізована через Blueprinter. UniversityBlueprint підтримує кілька view – базовий з основними полями, with_stats з фінансовою статистикою і ліцензійним обсягом, detailed що включає commissions і educators. QuestionnaireBlueprint через field з блоком рахує offers_count і calculated_offers_count – кількість пропозицій і кількість розрахованих через FilterService. OfferBlueprint підтримує view :registration, :card_public і :show – різні набори полів для різних контекстів використання.

На стороні фронтенду взаємодія з API реалізована через RTK Query в store.ts. userApi створено через createApi з baseUrl '/api' і tagTypes для кешування – CurrentUser, CurrentQuestionnaire, AllQuestionnaireOffers, Offers, AllUniversities та інші. Кожен endpoint описаний через build.query або build.mutation – getCurrentUser повертає дані поточного користувача і кешується через тег CurrentUser, authUser і createUser інвалідують CurrentUser після успішного виконання через invalidateOnSuccess helper. Мутації для роботи з паролем – requestPasswordReset, setNewPassword, changePassword – не інвалідують кеш бо не змінюють стан застосунку.

Приклад реалізації запитів через RTK Query:

```
requestPasswordReset: build.mutation({
  query: ({ email }) => ({
    url: 'password/forgot',
    method: 'POST',
    body: { email },
    responseHandler: 'text',
```

```

    }),
  }),
  setNewPassword: build.mutation({
    query: ({ token, email, password, password_confirmation }) => ({
      url: 'password/reset',
      method: 'POST',
      body: { token, email, password, password_confirmation },
      responseHandler: 'text',
    }),
  }),
  changePassword: build.mutation({
    query: ({ current_password, password, password_confirmation }) => ({
      url: 'password/change',
      method: 'POST',
      body: { current_password, password, password_confirmation },
    }),
  }),
  deleteCurrentUser: build.mutation({
    query: () => ({
      url: 'users',
      method: 'DELETE',
      responseHandler: 'text',
    }),
    invalidatesTags: ['CurrentUser'],
  }),
  getQuestionnaire: build.query({
    query: (id) => `questionnaire/${id}`,
    providesTags: ['CurrentQuestionnaire'],
  }),

```

3.1.5 Архітектура розгортання проєкту

Production середовище розгорнуто на Railway і складається з чотирьох сервісів, зображено на рисунку 3.10.

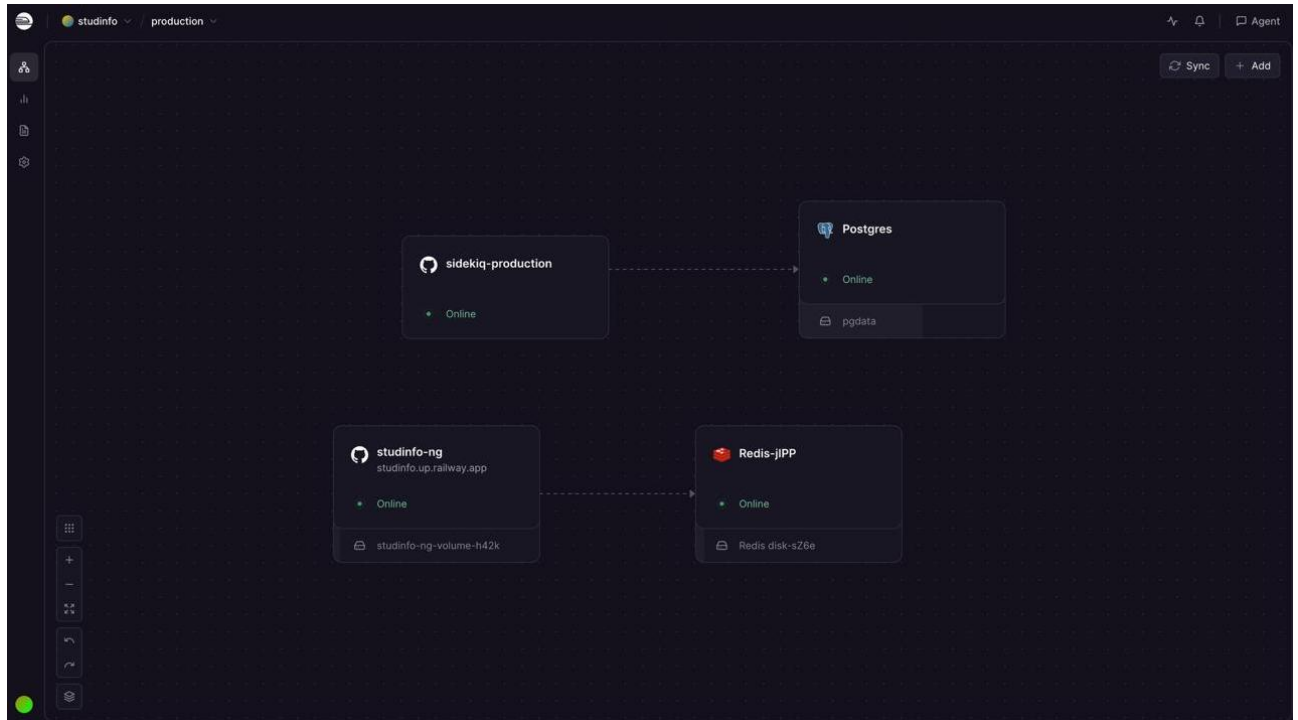


Рис. 3.10 Production середовище STUDINFO на Railway

studinfo-ng – основний Rails-сервіс доступний за адресою studinfo.up.railway.app. Розгорнуто через Docker-контейнер з персистентним volume studinfo-ng-volume-h42k для зберігання файлів. sidekiq-production – окремий процес що запускає Sidekiq worker і підключається до того ж Redis інстансу для обробки черг фонових задач. Postgres – окремий Railway сервіс з персистентним volume pgdata для зберігання основної бази даних. Redis-jlPP – окремий Redis інстанс з персистентним volume Redis disk-sz6e що використовується Sidekiq як сховище черг.

Sidekiq-production підключений до Postgres і Redis через внутрішні Railway змінні середовища – DATABASE_URL і REDIS_URL. studinfo-ng також підключений до Redis для кешування. Така архітектура дозволяє масштабувати

кожен сервіс незалежно і перезапускати окремі компоненти без впливу на решту системи.

Фронтенд розгорнуто на Vercel – production deployment прив'язаний до гілки main GitHub репозиторію, зображено на рисунку 3.11.

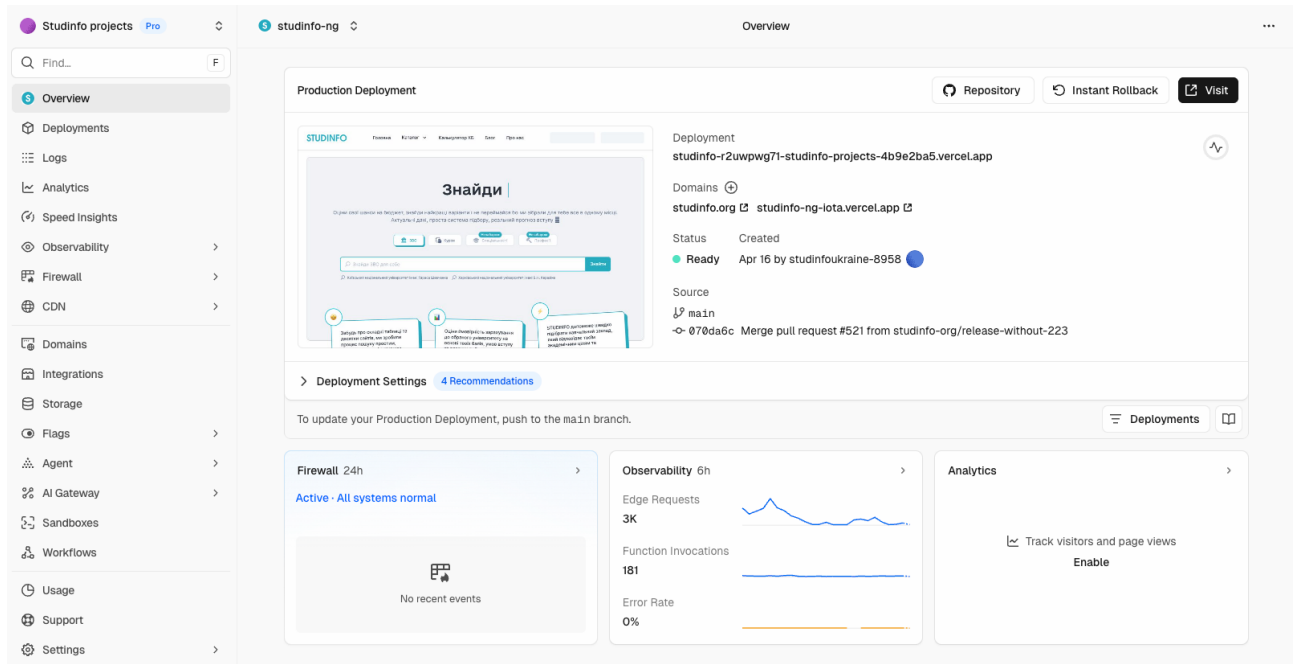


Рис. 3.11 Панель керування production deployment на Vercel

Кожен merge pull request в main автоматично тригерить новий деплой без ручного втручання. Платформа доступна за двома доменами – studinfo.org і studinfo-ng-iota.vercel.app. Firewall активний в режимі моніторингу – за останні 24 години критичних подій не зафіксовано. Observability показує 3K edge requests і 181 function invocation за останні 6 годин при Error Rate 0%.

3.2 Розробка основного лендінгу web-сервісу

Головна сторінка реалізована як React Server Component у файлі app/page.tsx – компонент рендериться на сервері і не містить директиви 'use client'. Структура сторінки побудована з незалежних секційних компонентів – Hero, UniversitiesBlock, ProductInfoBlock, CoursesBlock, CalculatorBlock, OffersBlock, ReviewsBlock – кожен з яких відповідає за окрему секцію і підтягує

власні дані незалежно. Адаптивні відступи між секціями налаштовані через responsive props Chakra UI – gap і padding змінюються залежно від breakpoint без медіа-запитів. Між секціями розміщені рекламні блоки GptAdSlot з різними ключами для десктопу і мобільного – slot_2/slot_8, slot_3/slot_9, slot_4/slot_7. Детальну структуру page.tsx зображено на рисунку 3.12.

```

1  import type { JSX } from 'react'
2  import { Flex, Box } from '@chakra-ui/react' 38.9k (gzipped: 12.5k)
3  import Header from '../components/Header'
4  import Footer from '../components/Footer'
5  import ContainerLarge from '../components/UI/Containers/ContainerLarge'
6  import Hero from '../components/HeroBlock/Hero'
7  import UniversitiesBlock from '../components/UniversitiesBlock'
8  import CoursesBlock from '../components/CoursesBlock'
9  import OffersBlock from '../components/OffersBlock'
10 import CalculatorBlock from '../components/CalculatorBlock'
11 import ProductInfoBlock from '../components/ProductInfoBlock/ProductInfoBlock'
12 import ReviewsBlock from '../components/ReviewsBlock/ReviewsBlock'
13 import GptAdSlot from '../components/gpt-ad/GptAdSlot'
14
15 const Page = (): JSX.Element => (
16   <>
17     <Header />
18     <Hero />
19     <Box>
20       <ContainerLarge>
21         <Flex
22           direction="column"
23           gap={{
24             base: '80px',
25             md: '90px',
26             xl: '100px',
27           }}
28           p={{
29             base: '80px 20px',
30             sm: '80px 22px',
31             md: '90px 24px',
32             lg: '90px 28px',
33             xl: '100px 32px',
34             '2xl': '100px 64px',
35           }}
36         >
37           <UniversitiesBlock />
38           <GptAdSlot desktopKey="slot_2" mobileKey="slot_8" />
39           <ProductInfoBlock />
40           <CoursesBlock />
41           <GptAdSlot desktopKey="slot_3" mobileKey="slot_9" />
42           <CalculatorBlock />
43           <OffersBlock />
44           <ReviewsBlock />
45           <GptAdSlot desktopKey="slot_4" mobileKey="slot_7" />
46         </Flex>
47       </ContainerLarge>
48     </Box>
49
50     <Footer />
51   </>
52 )
53
54 export default Page
55

```

Рис. 3.12 Структура page.tsx файлу головного лендінгу

Глобальний лейаут визначено в `app/layout.tsx` – тут підключаються ClientProviders з Redux store і Chakra UI темою, GTM через `next/script` з `strategy beforeInteractive` для ініціалізації до завантаження сторінки і Google AdSense з `strategy afterInteractive`. SEO налаштовано через об'єкт `metadata` – `title`, `description`, `robots`, `canonical URL`, `OpenGraph` і `Twitter Card` теги формуються статично. `metadataBase` встановлено як `studinfo.org` що дозволяє використовувати відносні шляхи для `og:image`. Приклад налаштування SEO за допомогою об'єкту `metadata` зображено на рисунку 3.13.

```
export const metadata = {
  title: 'STUDINFO: розрахунок ймовірності вступу легко та швидко 🚀',
  description:
    'Інноваційний освітній застосунок, який допоможе автоматично розрахувати конкурсний бал та шанси на вступ! Жоден вибір місця для вступу не обійдеться без STUDINFO.',
  metadataBase: new URL('https://studinfo.org/'),
  alternates: {
    canonical: 'https://studinfo.org/',
  },
  robots: 'index, follow',
  openGraph: {
    title: 'STUDINFO: розрахунок ймовірності вступу легко та швидко 🚀',
    description:
      'Інноваційний освітній застосунок, який допоможе автоматично розрахувати конкурсний бал та шанси на вступ! Жоден вибір місця для вступу не обійдеться без STUDINFO.',
    type: 'website',
    url: 'https://studinfo.org/',
    images: [
      {
        url: 'https://studinfo.org/img/metadata/main.png',
        width: 1200,
        height: 630,
        alt: 'STUDINFO: інноваційний освітній застосунок',
      },
    ],
  },
  twitter: {
    card: 'summary_large_image',
    title: 'STUDINFO: розрахунок ймовірності вступу легко та швидко 🚀',
    description:
      'Інноваційний освітній застосунок, який допоможе автоматично розрахувати конкурсний бал та шанси на вступ! Жоден вибір місця для вступу не обійдеться без STUDINFO.',
    url: 'https://studinfo.org/',
    images: ['https://studinfo.org/img/metadata/main.png'],
  },
  icons: [
    { url: '/favicon.ico', type: 'image/x-icon' },
    { url: '/favicon.svg', type: 'image/svg+xml' },
  ],
}
```

Рис. 3.13 Налаштування SEO за допомогою об'єкту `metadata` в Next.js

На стороні бекенду лендінг використовує два endpoint-и без автентифікації. `UniversitiesController#index` повертає список університетів через `UniversityBlueprint` – підтримує фільтрацію за регіоном, типом університету, формою навчання, ціновим діапазоном і середнім балом через SQL-умови з параметризованими запитами. При наявності параметрів `cost_min` або `cost_max` до `score` приєднується `with_offer_stats` – агрегована статистика по пропозиціях. Сортування реалізовано через `COLLATE "uk-UA-x-icu"` для коректного сортування кирилиці. Улюблені університети визначаються через

favorite_university_ids – окремий запит до бази що повертає id університетів доданих поточним користувачем до обраних.

OffersController#popular повертає 30 популярних конкурсних пропозицій без автентифікації – skip_before_action :authenticate_user! виключає цей endpoint з перевірки. Запит використовує left_joins(:requests) і group('offers.id') для підрахунку кількості заяв і сортування через order_by_order_expression. Сериалізація через OfferBlueprint з view :card_public – повертає мінімальний набір полів необхідний для відображення картки пропозиції на лендінгу. Детальну структуру контролера офферів зображено на рисунку 3.14.

```

1  # frozen_string_literal: true
2
3  class OffersController < ApplicationController
4    before_action :authenticate_user!
5    skip_before_action :authenticate_user!, only: %i[popular]
6
7    AVERAGE_SCORE = 154.13
8    ITEMS_PER_PAGE = 6
9
10   def index
11     pagy, records = pagy(offers, items: ITEMS_PER_PAGE)
12     render json: {
13       offers: OfferBlueprint.render_as_json(
14         records.includes(:university).includes(:specialty),
15         questionnaire:,
16         view: :registration
17       ),
18       count: offers.visible.count,
19       count_all: Offer.where(entry_year: 2025).visible.count,
20       average_score: AVERAGE_SCORE,
21       pagination: pagy_metadata(pagy)
22     }, status: :ok
23   end
24
25   def popular
26     offers = Offer
27       .visible
28       .preload(:university)
29       .left_joins(:requests)
30       .group('offers.id')
31       .order_by_order_expression
32       .limit(30)
33
34     render json: {
35       offers: OfferBlueprint.render_as_json(offers, view: :card_public)
36     }, status: :ok
37   end
38
39   def show
40     render json: OfferBlueprint.render(
41       Edebo::ShowOfferDetailsService.new(offer).call,
42       view: :show,
43       questionnaire: current_user.questionnaire
44     ), status: :ok
45   end
46
47   private
48
49   def offers
50     @offers ||= Questionnaire::FilterService.new(questionnaire:).call
51   end
52
53   def offer
54     Offer.find(params[:id])
55   end
56
57   def questionnaire
58     @questionnaire ||= Questionnaire.find(params[:questionnaire_id])
59   end
60 end

```

Рис. 3.14 Демонстрація offers_controller.rb файлу

3.3 Парсинг та підготовка відкритих даних з ЄДЕБО

Дані вступної кампанії отримуються з відкритого API порталу vstup.edbo.gov.ua – офіційного ресурсу ЄДЕБО. Парсинг реалізований через набір Service Objects організованих у namespace Edebo::Year2025, зображено на рисунку 3.15.

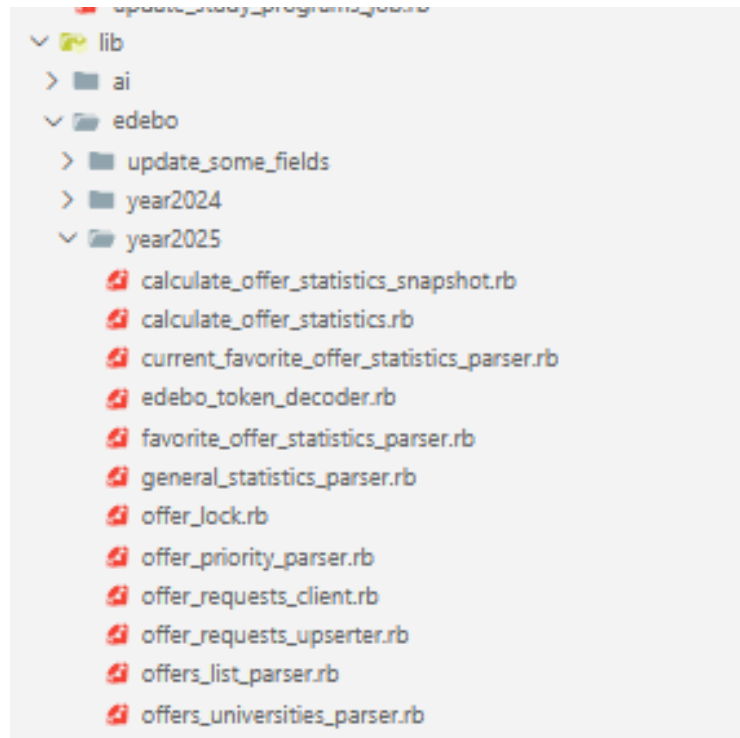


Рис. 3.15 Набір Service Objects організованих у namespace Edebo::Year2025 для парсингу даних з ЄДЕБО

Кожен клас відповідає за окрему задачу – отримання списку університетів, парсинг конкурсних пропозицій, збір заяв вступників, розрахунок статистики.

Точкою входу є `Edebo::Year2025::OffersUniversitiesParser` – клас що приймає параметри `qualification`, `education_base` і `specialty` і виконує POST-запит до <https://vstup.edbo.gov.ua/offers-universities>. Відповідь містить список університетів з їх `ids` – переліком ідентифікаторів конкурсних пропозицій. Для кожного університету викликається `University.find_or_initialize_by(code:)` – якщо університет вже існує в базі він оновлюється, якщо ні – створюється новий запис.

Після збереження університету в базу даних одразу запускається Edebo::Year2025::OffersListParser з отриманими ids. Детальну структуру парсеру для отримання оферів за 2025 рік зображено на рисунку 3.16.

```

3   class Edebo::Year2025::OffersListParser
12   def call
13   end
33
34   JSON.parse(response.body)['offers'].each do |edebo_offer|
35     code = edebo_offer['usid']
36     name = edebo_offer['usn']
37     educational_program = edebo_offer['spn']
38     faculty = edebo_offer['ufn']
39     specialty_id = edebo_offer['szc'] || edebo_offer['ssc']
40     name_of_university = edebo_offer['un']
41     financial_type = edebo_offer['ustn']
42     study_type = edebo_offer['efn']
43     enrollment_course = edebo_offer['cid']
44     period_of_study = edebo_offer['etrm']
45     application_deadline = edebo_offer['rtrm']
46     license_volume = edebo_offer['ol']
47     max_budget_size = edebo_offer['ox']
48     contract_size = edebo_offer['oc']
49     region_coefficient = edebo_offer['rk']
50     price_per_year = edebo_offer['price']
51     subject_coefficients = if edebo_offer['os']
52                           edebo_offer['os'].values.map do |val|
53                             {
54                               'name' => val['sn'],
55                               'coefficient' => val['k']
56                             }
57                           end
58     else
59       []
60     end
61
62     entry_year = 2025
63
64     offer = Offer.find_or_initialize_by(code:, entry_year:)
65     offer.assign_attributes(
66       name:,
67       educational_program:,
68       faculty:,
69       entry_year:,
70       university:,
71       specialty_id:,
72       name_of_university:,
73       financial_type:,
74       study_type:,
75       enrollment_course:,
76       period_of_study:,
77       application_deadline:,
78       license_volume:,
79       max_budget_size:,
80       contract_size:,
81       region_coefficient:,
82       price_per_year:,
83       subject_coefficients:
84     )
85     offer.save!
86   end
87 end
88
89 private
90
91 attr_reader :ids, :university

```

Рис. 3.16 Детальна структура парсеру для отримання оферів за 2025 рік

OffersListParser виконує POST-запит до <https://vstup.edbo.gov.ua/offers-list> з переліком ids і парсить відповідь. Кожна конкурсна пропозиція містить повний набір полів – код, назву, освітню програму, факультет, specialty_id, фінансовий тип, форму навчання, курс вступу, термін навчання, ліцензійний обсяг, кількість бюджетних і контрактних місць, регіональний коефіцієнт, вартість навчання і

коефіцієнти предметів. Поля зберігаються через Offer.find_or_initialize_by(code:, entry_year:) і offer.save! – ідемпотентний підхід що дозволяє повторно запускати парсер без дублювання даних.

HTTP-запити до ЄДЕБО реалізовані через FaradayWithProxy – обгортку над Faraday з підтримкою проксі і відключеною SSL-верифікацією. Заголовки імітують реальний браузер – User-Agent Firefox, X-Requested-With XMLHttpRequest, Referer і Origin встановлені на vstup.edbo.gov.ua. Це необхідно оскільки ЄДЕБО блокує запити без коректних заголовків. Конфігурація заголовків запиту до ЄДЕБО зображено на рисунку 3.17.

```
def call
  response = FaradayWithProxy.new(url: EDEBO_OFFERS_LIST_URL, ssl: { verify: false }).post do |req|
    req.headers['Referer'] = 'https://vstup.edbo.gov.ua/offers/'
    req.headers['User-Agent'] = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0'
    req.headers['Accept'] = 'application/json, text/javascript, */*; q=0.01'
    req.headers['Accept-Language'] = 'ru,uk;q=0.8,en;q=0.6,ru-RU;q=0.4,en-US;q=0.2'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded; charset=UTF-8'
    req.headers['X-Requested-With'] = 'XMLHttpRequest'
    req.headers['Origin'] = 'https://vstup.edbo.gov.ua'
    req.headers['DNT'] = '1'
    req.headers['Connection'] = 'keep-alive'
    req.headers['Sec-Fetch-Dest'] = 'empty'
    req.headers['Sec-Fetch-Mode'] = 'cors'
    req.headers['Sec-Fetch-Site'] = 'same-origin'
    req.headers['Sec-GPC'] = '1'
    req.headers['Priority'] = 'u=0'

    req.body = {
      ids:
    }
  end
end
```

Рис. 3.17 Конфігурація заголовків запиту до ЄДЕБО

Окремо реалізований модуль OfferRequestsClient для збору заяв вступників. Модуль містить механізм rate limiting через RATE_LIMIT_SLEEP_SCHEDULE – масив затримок від 60 до 1800 секунд що застосовуються послідовно при отриманні відповіді "Перевищено ліміт запитів". Retry-логіка реалізована через exponential backoff – затримка між повторними спробами збільшується від 2 до 10 секунд при помилках з'єднання, таймауті або парсингу JSON. Cookie jar підключається через faraday-cookie_jar – сесійні cookies зберігаються між запитами для коректної роботи з ЄДЕБО.

OffersUniversitiesParser також реалізує retry при Faraday::ConnectionFailed і Faraday::TimeoutError – при помилці з'єднання процес очікує 10 секунд і повторює запит. Це критично для стабільної роботи парсера в умовах нестабільного з'єднання або тимчасової недоступності ЄДЕБО.

3.4 Розробка модуля "Помічник вступу"

3.4.1 Реєстрація та профіль користувача

Реєстрація користувача реалізована через Users::RegistrationsController що успадковує Devise::RegistrationsController. Метод create будує ресурс через build_resource і одразу створює прив'язану анкету через resource.build_questionnaire – кожен новий користувач автоматично отримує порожню анкету без додаткових запитів. При успішному збереженні користувач одразу авторизується через sign_in і отримує відповідь зі статусом 201. При помилці повертається хеш errors з повідомленнями валідації і поле email_confirmed для обробки випадку коли email вже зареєстрований але не підтверджений. Додаткові параметри реєстрації – name, password, user_type – дозволені через configure_sign_up_params і devise_parameter_sanitizer. Деталі RegistrationsController зображено на рисунку 3.18.

```

1 # frozen_string_literal: true
2
3 class Users::RegistrationsController < Devise::RegistrationsController
4   respond_to :json
5   before_action :configure_sign_up_params, only: :create
6
7   def create
8     build_resource(sign_up_params)
9     resource.save
10
11     if resource.persisted?
12       sign_in(resource.name, resource)
13       return render json: resource, status: :created
14     end
15
16     error_result = {
17       errors: resource.errors.to_hash(true),
18       # TODO: this is temporary, long term don't report if email is confirmed,
19       # it can be used to scan emails
20       email_confirmed: User.find_by(email: resource_params[:email]).try(:email_confirmed?)
21     }
22     render json: error_result, status: :unprocessable_entity
23   end
24
25   protected
26
27   def configure_sign_up_params
28     devise_parameter_sanitizer.permit(:sign_up, keys: %i[name password user_type])
29   end
30
31   def build_resource(*)
32     super
33
34     resource.build_questionnaire
35     resource
36   end
37
38 end

```

Рис. 3.18 RegistrationsController для реєстрації користувача

Авторизація реалізована через `Users::SessionsController`. Метод `create` використовує `warden.authenticate` для перевірки облікових даних – при успіху виконується `sign_in` і повертається `UserBlueprint` з `view :public` разом з `tokens_used`. При невалідних даних повертається статус 401 з повідомленням про помилку. Метод `destroy` викликає `sign_out` і повертає статус 200.

Профіль користувача доступний через `GET /api/users/current` – `UsersController#current` повертає поточного користувача через `UserBlueprint :public`, `questionnaire_id` і `questionnaire_status`. Це єдиний endpoint що використовується на фронтенді для визначення стану автентифікації і прогресу анкети. Оновлення профілю – `PATCH /api/users` з параметрами `email`, `name`, `user_type`. Видалення акаунту – `DELETE /api/users` через `current_user.destroy!` і `sign_out`. Використання профілю користувача на стороні фронтенду наведено на рисунку 3.19.

```

27 const SelectionBlock = ({
28   onStart,
29   options,
30   selectedOption,
31   onOptionSelect,
32 }: SelectionBlockProps): JSX.Element => {
33
34   const { data } = useGetCurrentUserQuery(null)
35   const currentUser = data?.current_user
36
37   const {
38     isOpen: isReplenishmentOpen,
39     onOpen: onReplenishmentOpen,
40     onClose: onReplenishmentClose,
41   } = useDisclosure()
42
43   return (
44     <Box bg="#F1F3F6">
45       <ContainerLarge>
46         <Flex
47           direction="column"
48           align="center"
49           gap={{ base: '10px', sm: '12px', md: '16px' }}
50           p={{
51             base: '24px 20px',
52             sm: '26px 22px',
53             md: '32px 24px',
54             lg: '48px 28px',
55             xl: '100px 32px',
56             '2xl': '140px 64px',
57           }}
58         >
59           <CreditBalance
60             creditNumber={currentUser?.messages_balance ?? 0}
61             onReplenishmentOpen={onReplenishmentOpen}
62           />
63

```

Рис. 3.19 Використання профілю користувача на стороні фронтенду

На стороні фронтенду всі операції з користувачем реалізовані через RTK Query в `store.ts`. `getCurrentUser` кешується через тег `CurrentUser` і використовується в усіх компонентах що потребують даних поточного користувача. `authUser`, `createUser` і `updateCurrentUser` інвалідують тег `CurrentUser` після успішного виконання через `invalidateOnSuccess` – це автоматично оновлює всі компоненти що підписані на `getCurrentUser` без додаткового коду.

3.4.2 Анкета вступника та конкурсні пропозиції

Анкета вступника – центральний модуль платформи що збирає індивідуальні параметри абітурієнта і використовується для формування персоналізованого списку конкурсних пропозицій. Модель `Questionnaire` зберігає відповіді у `jsonb` полі `answers` – гнучка структура що дозволяє зберігати різнотипні дані без зміни схеми бази даних. Статус анкети відстежується через `enum` – `not_started`, `in_progress`, `complete`. Поточний крок зберігається в полі `step_id` з константою `LAST_STEP_ID = 9`.

Анкета складається з 8 кроків – рік складання НМТ, результати іспитів, регіони, спеціальності, фінансові пріоритети, форма навчання, додаткові курси і фінальний перехід до дашборду. Відповіді зберігаються як `jsonb` через `accessor-` методи – `answers_regions`, `answers_study_type`, `answers_specialties`, `answers_financial_mode` – кожен з яких нормалізує дані через `normalize_array` перед збереженням.

На бекенді оновлення анкети реалізовано через `QuestionnaireController#update` що делегує логіку до `Questionnaire::UpdateService`. Контролер приймає `step_id` і `answers` через `questionnaire_params` з `permit` і передає їх в сервіс. Доступ до анкети через `current_user.questionnaire` – кожен користувач має рівно одну анкету створену при реєстрації.

Реалізацію модель `Questionnaire` наведено на рисунку 3.20.

```

1  # frozen_string_literal: true
2
3  class Questionnaire < ApplicationRecord
4    LAST_STEP_ID = 9
5    attr_accessor :answers_year,
6                  :exam_math,
7                  :exam_history,
8                  :exam_ukr_lang,
9                  :exam_tk,
10                 :exam_additional_subject_name,
11                 :exam_additional_subject_value
12
13    enum :status, %i[not_started in_progress complete]
14    belongs_to :user
15    has_and_belongs_to_many :offers
16    has_many :questionnaire_offers, dependent: :destroy
17    has_many :favorite_offers, dependent: :destroy
18
19    def self.ransackable_attributes(_auth_object = nil)
20      %w[answers created_at id status step_id updated_at user_id]
21    end
22
23    def answers_regions
24      = Array(answers&.<('regions')).map(&:to_s)
25    def answers_study_type
26      = Array(answers&.<('studyType')).map(&:to_s)
27    def answers_specialties
28      = Array(answers&.<('specialties')).map(&:to_s)
29    def answers_additional_course_university_id
30      = answers&.dig('additionalCourseInfo', 'selectedUniversity', 'value')
31    def answers_additional_course_score
32      = answers&.dig('additionalCourseInfo', 'courseScore')
33
34    def answers_regions=(value)
35      self.answers ||= {}
36      self.answers['regions'] = normalize_array(value)
37    end
38
39    def answers_study_type=(value)
40      self.answers ||= {}
41      self.answers['studyType'] = normalize_array(value)
42    end
43
44    def answers_specialties=(value)
45      self.answers ||= {}
46      self.answers['specialties'] = normalize_array(value)
47    end
48
49    def answers_financial_mode
50      values = Array(answers&.<('financialType')).map(&:to_s)
51      values.include?('Небюджетна') ? 'all' : 'budget_only'
52    end
53
54    def answers_financial_mode=(mode)
55      self.answers ||= {}
56      base = %w[Відкрита Фіксована]
57      self.answers['financialType'] = mode.to_s == 'all' ? (base + ['Небюджетна']) : base
58    end
59
60    def answers_additional_course
61      v = answers&.dig('additionalCourse')
62      v.present? ? v.to_s : nil
63    end
64
65    private
66    def normalize_array(value) = Array(value).map { |v| v.to_s.strip }.reject(&:blank?).uniq
67  end

```

Рис. 3.20 Реалізація моделі Questionnaire

На фронтенді компонент Sidebar визначає що показувати залежно від стану авторизації – незареєстрованому користувачу показується Registration, авторизованому SidebarForUser. SidebarForUser отримує questionnaire_id через useGetCurrentUserQuery і завантажує анкету через useGetQuestionnaireQuery. Switch по step_id рендерить відповідний компонент кроку – CompleteExamsYear для кроку 1, ExamsYear2024 або ExamsYear2023 для кроку 2 залежно від обраного року, Regions для кроку 3 і так далі. При step_id = 9 виконується redirect

на `/helper` – перехід до дашборду після завершення анкети. Реалізацію компоненту `Sidebar.tsx` наведено на рисунку 3.21.

```

14 const SidebarForUser = () => {
15   const { data: { questionnaire_id: questionnaireId } } = useGetCurrentUserQuery({ refetchOnMount: false })
16   const { data, isLoading, isError } = useGetQuestionnaireQuery(questionnaireId)
17
18   if (isLoading) {
19     return <Spinner />
20   }
21
22   if (isError) {
23     return 'Error'
24   }
25
26   const { step_id: stepId, answers } = data
27   const year = answers?.year || null
28
29   switch(stepId) {
30     case 1:
31       return (<CompleteExamsYear />)
32     case 2:
33       if(year === '2024' || year === '2025' || year === '2026'){
34         return (<ExamsYear2024 />)
35       }
36       return (<ExamsYear2023 />)
37     case 3:
38       return (<Regions />)
39     case 4:
40       return (<Specialties />)
41     case 5:
42       return (<FinancialPriority />)
43     case 6:
44       return (<StudyPriority />)
45     case 7:
46       return (<CheckAdditionalCourses />)
47     case 8:
48       return <AdditionalCourses />
49     case 9:
50       window.location.href = '/helper'
51       return <Spinner />
52   }
53
54   return null
55 }
56
57 const Sidebar = () => {
58   const { data, isError, isLoading } = useGetCurrentUserQuery({ refetchOnMount: false })
59
60   if (isLoading) {
61     return <Spinner />
62   }
63
64   if (isError || !data.current_user) {
65     return <Registration />
66   }
67
68   return <SidebarForUser />
69 }
70
71 export default Sidebar
72

```

Рис. 3.21 Реалізація компоненту `Sidebar.tsx`

Кожен крок анкети реалізований як окремий компонент що викликає `useUpdateCurrentQuestionnaireMutation` при переході до наступного кроку. Компонент `Regions` демонструє типовий патерн – локальний стан `checkedItems`

зберігає вибрані регіони, `handleNextClick` валідує що хоча б один регіон обраний через `showToast` і викликає `updateQuestionnaire` з `answers` і `step_id`. Кнопка "Назад" через `stepBack` оновлює тільки `step_id` без зміни `answers` – користувач може повернутись і переглянути попередній крок без втрати даних. Приклад реалізації кроку №3 "Обери регіон, в якому плануєш вступати до ЗВО" в компоненті `Regions.tsx` наведено на рисунку 3.22.

```

1  import React, { useState } from 'react' 6.9k (gzipped: 2.7k)
2  import { Flex } from '@chakra-ui/react' 38.5k (gzipped: 12.3k)
3  import { useGetCurrentUserQuery, useUpdateCurrentQuestionnaireMutation } from '../../api/user.ts'
4  import SidebarsButtons from '../../UI/Buttons/SidebarsButtons.tsx'
5  import useFillFieldsToast from '../../hooks/useFillFieldsToast.tsx'
6  import RegionsInput from './InputFields/RegionsInput.tsx'
7
8  interface RegionsProps {
9    isLoading?: boolean
10 }
11
12 const Regions: React.FC<RegionsProps> = () => {
13   const { data: { questionnaire_id: questionnaireId } } = useGetCurrentUserQuery(null)
14   const [updateQuestionnaire, { isLoading }] = useUpdateCurrentQuestionnaireMutation()
15   const [checkedItems, setCheckedItems] = useState<{ [key: string]: boolean }>({})
16   const showToast = useFillFieldsToast()
17
18   const selectedRegions = Object.keys(checkedItems).filter(key => checkedItems[key])
19
20   const handleNextClick = () => {
21     if (selectedRegions.length === 0) {
22       showToast()
23     } else {
24       updateQuestionnaire({ answers: { regions: selectedRegions }, step_id: 3, id: questionnaireId })
25     }
26   }
27
28   const stepBack = () => {
29     updateQuestionnaire({ step_id: 1, id: questionnaireId })
30   }
31
32   return (
33     <Flex gap="12px" direction="column" h="100%">
34       <RegionsInput value={checkedItems} onChange={setCheckedItems} />
35       <SidebarsButtons nextStep={handleNextClick} previousStep={stepBack} isLoading={isLoading} />
36     </Flex>
37   )
38 }
39
40 export default Regions
41

```

Рис. 3.22 Приклад реалізації компоненту `Regions.tsx`

Список конкурсних пропозицій формується на основі заповненої анкети користувача і є центральним елементом модуля "Помічник вступу". `QuestionnaireOffersController#index` повертає пагінований список `questionnaire_offers` відсортованих за `recommendation_score` і `competitive_score`. Підтримується фільтрація за університетами через параметр `university_ids` і

перемикання між усіма пропозиціями і відстежуваними через параметр `favorite`. Сортування обмежене через `ALLOWED_SORTS` – лише `asc` або `desc` для захисту від SQL-ін'єкцій через динамічні `ORDER BY`.

`QuestionnaireOffersController#show` реалізує нестандартний патерн – `params[:id]` може бути як `id questionnaire_offer` так і `id offer`. Метод `find_questionnaire_offer_for_params` спочатку шукає через `questionnaire.questionnaire_offers.find_by(id:)`, якщо не знаходить – шукає `Offer` і при `allow_create: true` створює новий `questionnaire_offer` через `find_or_initialize_by`. Це дозволяє фронтенду використовувати `offer.id` для переходу до детальної сторінки навіть якщо `questionnaire_offer` ще не існує.

При додаванні КП до відстежуваних через `QuestionnaireOffersController#update` виконується ланцюжок дій – `questionnaire_offer.update!(favorite: true)`, створення `favorite_offer` через `find_or_create_by!`, запуск `CurrentFavoriteOfferStatisticsParserJob` для оновлення статистики і виклик `QuestionnaireOffer::CalculateAndPersistService` для розрахунку конкурсного бала якщо КП переглядалась через `view_offer`.

Серіалізація відповідей реалізована через `OfferBlueprint` з кількома `view`. `View :registration` обчислює `score` через `CalculateCompetitiveScore` і `rating_position` через `QuestionnaireOffer::CalculateService` для кожної пропозиції – ці поля залежать від параметрів анкети конкретного користувача. `View :show` включає `offer_requests_statistic` через `OfferStatisticBlueprint`, `university` через `UniversityBlueprint` і `last_year_offer` для порівняння зі статистикою попереднього року. `View :card_public` повертає мінімальний набір полів для відображення картки без автентифікації.

На фронтенді сторінка `HelperPage` реалізована як `Client Component` з директивою `'use client'`. Паралельно виконуються кілька `RTK Query` запитів – `getCurrentUser`, `getQuestionnaire`, `getAllSortingUniversities`, `getAllSortingRegions` і `getAllQuestionnaireOffers`. При статусі анкети `in_progress` запускається інтервал що кожні 5 секунд інвалідує теги `CurrentQuestionnaire` і `AllQuestionnaireOffers` через `dispatch(userApi.util.invalidateTags)` – це забезпечує автоматичне оновлення

списку КП поки система розраховує бали у фоні. Сторінка підтримує URL-параметри page і tab – при зміні сторінки або переключенні між вкладками URL оновлюється через router.push без перезавантаження сторінки. Приклад реалізації фронтенд логіки сторінки КП наведено на рисунку 3.23.

```

42 const HelperPage = () : JSX.Element => {
84
85   const {
86     data: currentQuestionnaire,
87     isLoading: isCurrentQuestionnaireLoading,
88   } = useQuestionnaire()
89
90   const questionnaireId = currentUserData?.questionnaire_id
91
92   const { data: universitiesData, isLoading: isUniversitiesLoading } =
93     useGetAllSortingUniversitiesQuery(questionnaireId, {
94       skip: !questionnaireId,
95     })
96
97   const { data: regionsData, isLoading: isRegionsLoading } =
98     useGetAllSortingRegionsQuery(questionnaireId, {
99       skip: !questionnaireId,
100     })
101
102   const dispatch = useDispatch()
103   useEffect(() => {
104     if (isCurrentUserError) {
105       router.push('/login')
106     }
107   }, [isCurrentUserError, router])
108
109   useEffect(() => {
110     if (
111       !currentQuestionnaire ||
112       currentQuestionnaire.status !== 'in_progress'
113     ) {
114       return
115     }
116
117     const intervalId = setInterval(() => {
118       dispatch(
119         userApi.util.invalidateTags([
120           'CurrentQuestionnaire',
121           'AllQuestionnaireOffers',
122         ])
123       )
124     }, 5000)
125
126     return () => clearInterval(intervalId)
127   }, [currentQuestionnaire, dispatch])
128
129   const [sortOrder, setSortOrder] = useState('asc')
130   const favoriteTabFromQuery = searchParams?.get('tab') === 'favorites'
131   const [isFavorite, setIsFavorite] = useState(favoriteTabFromQuery)
132   const [selectedUniversityIds, setSelectedUniversityIds] = useState<number[]>([])
133
134   const { data: allOffersCountData, isLoading: isAllOffersCountLoading } =
135     useGetAllQuestionnaireOffersQuery(
136       questionnaireId
137       ? { id: questionnaireId, sortOrder, page: '1', favorite: false, universityIds: selectedUniversityIds }
138       : skipToken,
139     )
140
141   const {
142     data: favoriteOffersCountData,
143     isLoading: isFavoriteOffersCountLoading,

```

Рис. 3.23 Фрагмент реалізації фронтенд логіки сторінки КП

Компонент OffersSection рендерить два таби – "Усі пропозиції" з загальним лічильником і "Відстеження" з обмеженням до 10 пропозицій. Фільтрація за університетами реалізована через UniversityFilter що передає selectedUniversityIds в запит. Сортування за рекомендацією керується через Select з двома опціями що змінюють sortOrder між asc і desc.

3.4.3 Сторінки університетів

Каталог університетів реалізований як гібридна сторінка – `page.tsx` є `async Server Component` що виконує початковий запит до API на сервері і передає результат як `initialData` в клієнтський компонент `UniversitiesCatalog`. Такий підхід забезпечує швидке відображення даних без додаткового клієнтського запиту.

Модель `University` використовує `pg_search` з `tsvector` колонкою `search_vector` і GIN індексом для пошуку. `search_vector` формується з `name`, `short_name` і назви регіону через `to_tsvector('simple')` і оновлюється через метод `update_search_vector` при зміні даних. `Slug` генерується автоматично при створенні через `SlugGenerator` і має унікальний індекс – університети доступні за URL адресами замість числових `id`. `University` включає асоціації `university_info` з детальною інформацією (`address`, `head`, `staff`, `website`, `ownership`), `commissions`, `educators`, `socials_networks` і `study_programs` через `foreign_key` `university_code`. Реалізацію моделі `University` зображено на рисунку 3.24.

```

1  # frozen_string_literal: true
2
3  class University < ApplicationRecord
4    include University::OfferStatistics
5    include PgSearch::Model
6
7    belongs_to :region, primary_key: :code, foreign_key: :region_id
8
9    has_one :university_info, dependent: :destroy
10
11    has_many :offers, dependent: :destroy
12    has_many :favorite_universities, dependent: :destroy
13    has_many :commissions, dependent: :delete_all
14    has_many :educators, dependent: :delete_all
15    has_many :socials_networks, dependent: :delete_all
16    has_many :study_programs, primary_key: :code, foreign_key: :university_code,
17      class_name: :StudyProgram, inverse_of: :university, dependent: :nullify
18
19    accepts_nested_attributes_for :socials_networks, allow_destroy: true
20
21    validates :slug, uniqueness: true, presence: true
22
23    pg_search_scope :search_fulltext,
24      against: [],
25      using: { tsearch: { tsvector_column: 'search_vector', prefix: true } }
26
27    before_validation :generate_slug, on: :create
28
29    def update_search_vector
30      text = [name, short_name, region&.name].compact.join(' ')
31
32      vector = self.class.send(:sanitize_sql_array, ["to_tsvector('simple', ?)", text])
33      self.class.where(id:).update_all("search_vector = #{vector}")
34    end
35
36    private
37
38    def generate_slug
39      return if slug.present? || name.blank?
40
41      self.slug = SlugGenerator.new(name, University).call
42    end
43  end
44
```

Рис. 3.24 Реалізація моделі `University`

`UniversitiesController#index` підтримує розширену фільтрацію – за ключовим словом через `LOWER(universities.name) LIKE` з параметризованим запитом, за регіоном через `region_id`, за типом університету через `university_infos.ownership`, за ціновим діапазоном і середнім балом через SQL-умови з `joins` до `offers`. Фільтр `has_budget` і `has_grants` реалізовані через `EXISTS subquery` – це ефективніше ніж `JOIN` для перевірки наявності пов'язаних записів. Сортування підтримує три режими – за назвою через `COLLATE "uk-UA-x-icu"` для коректної кирилиці, за рейтингом через `CASE WHEN rating_position IS NULL THEN 1 ELSE 0 END` для коректного розміщення університетів без рейтингу в кінці списку.

`UniversitiesController#show` завантажує університет через `find_by!(slug:)` з `includes` для `university_info`, `commissions`, `educators` і `socials_networks` – всі дані для детальної сторінки завантажуються одним запитом через `eager loading`. Сериалізація через `UniversityFullBlueprint` включає повну статистику по пропозиціях, контакти приймальної комісії, дані викладачів і соціальні мережі університету.

На фронтенді `UniversitiesCatalog` є `Client Component` що керує станом фільтрів, пошуку і сортування через кастомний хук `useUniversitiesCatalog`. Компонент має адаптивну верстку – на мобільних пристроях фільтри відкриваються через `Drawer`, на десктопі – відображаються як бічна панель через `Show above="md"` і `Show below="md"` від `Chakra UI`. `UniversitiesCards` отримує `queryParams` і виконує запит через `useGetAllUniversitiesQuery` – при зміні фільтрів або сортування `useEffect` скидає сторінку на першу через `updatePageParam('1')`. Дані університетів нормалізуються через `toNumber` для коректної обробки числових полів що можуть приходити як рядки. Сортування на фронтенді підтримує `nameAsc`, `nameDesc` і сортування за рейтингом через `localeCompare` з `locale 'uk'`.

Детальна сторінка університету організована через чотири вкладки – Основна інформація з популярними освітніми програмами і конкурсними пропозиціями, Спеціальності та програми, Контакти з даними приймальної

комісії і Соцмережі зі списком Telegram-каналів і Instagram-сторінок університету.

3.4.4 Розробка калькулятора конкурсного балу

Калькулятор конкурсного бала реалізований як окремий модуль на фронтенді без звернень до серверного API – всі дані спеціальностей, регіонів і предметів завантажуються з локального файлу `calculatorMockData.ts`. Це забезпечує миттєву відповідь без мережових затримок і дозволяє використовувати калькулятор без автентифікації.

Логіка калькулятора зосереджена в хуку `useCalculator` що керує трьома станами через `enum CalculatorStep` – `STEP1`, `STEP2` і `RESULT`. `formData` зберігає всі введені параметри, `result` – підсумковий розрахований бал. При зміні року НМТ через `handleNmtYearChange` автоматично скидаються залежні поля – для 2023 року "Історія України" стає предметом на вибір тому `historyOfUkraine` обнуляється. При зміні спеціальності через `handleSpecialtyChange` скидаються `creativeCompetitionScore` і `prepCoursesScore` якщо нова спеціальність не має відповідних прапорців `hasCreativeCompetition` або `hasSpecialSupport`.

Крок 1 збирає базові дані – спеціальність через `React Select` з пошуком, рік НМТ від 2023 до 2026, бали з української мови, математики, історії України і предмету на вибір. Коефіцієнти `k1-k3` автоматично підтягуються з обраної спеціальності і відображаються у `readonly`-полях. Коефіцієнт `k4` для предмету на вибір визначається через `getCoefficient()` – функція повертає відповідний коефіцієнт зі структури `specialty.coefficients.k4` залежно від назви предмету. Валідація `isStep1Valid` перевіряє заповненість усіх полів і діапазон балів 100-200.

Крок 2 збирає додаткові параметри – творчий конкурс що показується тільки якщо `specialty.hasCreativeCompetition === true`, регіон з регіональним коефіцієнтом від 1.0 до 1.07, і бал за підготовчі курси від 1 до 10 що показується тільки якщо `specialty.hasSpecialSupport === true` і користувач активував чекбокс.

Функція `calculateConcoursScore` реалізує різні формули залежно від року вступу.

Для калькуляції КБ 2025 і 2026 року використовується формула (3.1):

$$КБ = \frac{K1 * П1 + K2 * П2 + K3 * П3 + K4 * П4 + КТ * ТК}{(K1 + K2 + K3 + \left(\frac{K4 + K4_{\max}}{2} + КТ\right))} + ОУ, \quad (3.1)$$

де КБ – конкурсний бал;

Р1, Р2, Р3, Р4 – оцінки з чотирьох предметів НМТ;

К1, К2, К3, К4 – вагові коефіцієнти відповідних предметів;

К4_{max} – максимальне значення коефіцієнта для четвертого предмета;

КТ – коефіцієнт творчого конкурсу;

ТК – оцінка за творчий конкурс;

ОУ – додаткові (бонусні) бали.

Для 2024 знаменник спрощений – без k4_{max}. Для 2023 формула залежить від того чи обрано "Історія України" як предмет на вибір – якщо так, Р4 обнуляється і k4 не враховується. Кінцевий бал множиться на регіональний і галузевий коефіцієнти, обмежується діапазоном 100-200 через Math.min і Math.max і округлюється до 2 знаків.

Результат відображає підсумкову картку з усіма введеними даними у вигляді таблиці предмет/бал/коефіцієнт, всіма застосованими коефіцієнтами і великим блоком з кінцевим балом. getFeedbackMessage повертає текстовий фідбек залежно від діапазону балу – від песимістичного при балі нижче 130 до "ти в еліті вступників" при балі вище 195.

Детальну калькуляцію КБ для різних років здачі НМТ за допомогою мови TypeScript продемонстровано на рисунку 3.25.

```

13 export const calculateConcoursScore = (
44   const regionalCoefficient = region.coefficient
45
46   const electiveCoeff = electiveSubject
47     ? getCoefficient(specialty, electiveSubject.name) || 0
48     : 0
49
50   const p1 = ukrainianLanguage
51   const p2 = mathematics
52   const p3 =
53     nmtYear === 2023
54     ? electiveSubject?.name === 'Історія України'
55     ? electiveSubjectScore || 0
56     : 0
57     : historyOfUkraine || 0
58   const p4 =
59     nmtYear === 2023
60     ? electiveSubject?.name !== 'Історія України'
61     ? electiveSubjectScore || 0
62     : 0
63     : electiveSubjectScore || 0
64
65   const tk = creativeCompetitionScore || 0
66   const ou =
67     hasPrepCourses && prepCoursesScore && specialty.hasSpecialSupport
68     ? prepCoursesScore
69     : 0
70
71   let kb = 0
72
73   if (nmtYear === 2025 || nmtYear === 2026) {
74     kb =
75       (k1 * p1 + k2 * p2 + k3 * p3 + electiveCoeff * p4 + kt * tk) /
76       (k1 + k2 + k3 + (k4max + electiveCoeff) / 2 + kt) +
77       ou
78   } else if (nmtYear === 2024) {
79     kb =
80       (k1 * p1 + k2 * p2 + k3 * p3 + electiveCoeff * p4 + kt * tk) /
81       (k1 + k2 + k3 + electiveCoeff + kt) +
82       ou
83   } else if (nmtYear === 2023) {
84     if (electiveSubject?.name === 'Історія України') {
85       kb = (k1 * p1 + k2 * p2 + k3 * p3 + kt * tk) / (k1 + k2 + k3 + kt) + ou
86     } else {
87       kb =
88         (k1 * p1 + k2 * p2 + electiveCoeff * p4 + kt * tk) /
89         (k1 + k2 + electiveCoeff + kt) +
90         ou
91     }
92   } else {
93     kb = (k1 * p1 + k2 * p2 + k3 * p3 + kt * tk) / (k1 + k2 + k3 + kt) + ou
94   }
95
96   kb = kb * regionalCoefficient * sectorCoefficient
97
98   kb = Math.min(200, Math.max(100, kb))
99   kb = Math.round(kb * 100) / 100
100
101   const percentileBetter = Math.floor(Math.random() * 60) + 30
102
103   return { concoursScore: kb, percentileBetter, formData }
104 }

```

Рис. 3.25 Калькуляція КБ для різних років мовою TypeScript

3.5 Налаштування панелі адміністрування

Адміністративна панель STUDINFO реалізована на базі Active Admin з namespace home – всі URL мають вигляд /home/*. Dashboard автоматично перенаправляє на OfferAnalytics#index. Навігація організована через menu priority – аналітичні розділи мають найвищий пріоритет, CRUD-ресурси розміщені нижче.

User – базовий CRUD з `permit_params` для `email`, `password`, `tokens_balance`, `ai_assistant_enabled` і `admin`. Форма дозволяє редагувати `confirmed_at` напряму і `admin` флаг – прямий контроль доступу без окремої ролевої моделі.

Offer – найскладніший ресурс з `scopes` по `financial_types` і `study_types`, а також `visible/hidden` для `visibility` enum. Форма має кастомний `partial` `subject_coefficients_table` для редагування JSON-масиву коефіцієнтів через динамічну таблицю з JavaScript. `Concern Admin::OfferParamsProcessor` обробляє вкладену JSON-структуру перед збереженням.

Questionnaire – форма з кастомним контролером. Actions `:new` і `:destroy` заблоковані – анкети створюються тільки через API при реєстрації. Форма парсить JSON-поле `answers` у окремі поля і зберігає назад через `build_answers`. Після `update` викликається `Questionnaires::ProgressDeriver.call` для перерахунку `step_id` і `status`. Member action `reset_to_first_step` скидає анкету до початкового стану.

`OfferAnalytics` і `UniversityAnalytics` – read-only ресурси що будують LEFT JOIN з таблицею `ahoy_events` через raw SQL і агрегують `views_count`, `clicks_count` і `favorites_count` через COALESCE для відображення статистики взаємодії користувачів з платформою.

4 РЕЗУЛЬТАТИ РОЗРОБКИ ТА ТЕСТУВАННЯ WEB-SERVICE

Результатом виконання роботи є SPA (Single Page Application) web-сервіс для допомоги абітурієнтам, який дозволяє абітурієнтам отримувати всю актуальну інформацію щодо вступної кампанії та отримувати персоналізовані рекомендації на основі зібраної інформації про користувача.

4.1 Демонстрація екранних форм web-сервісу

На головному лендінгу розміщено всю необхідну інформацію про web-сервіс, за рахунок чого користувач може дізнатись все необхідне про функціонал сервісу переглянувши всього одну сторінку і витративши мінімальну кількість часу. Головний акцент робився на Него-секції, звідки користувач може потрапити на, наприклад, каталог університетів і почати свій шлях користування розробленою системою. На рисунку 4.1 зображено Него-секцію головного лендінгу web-сервісу.

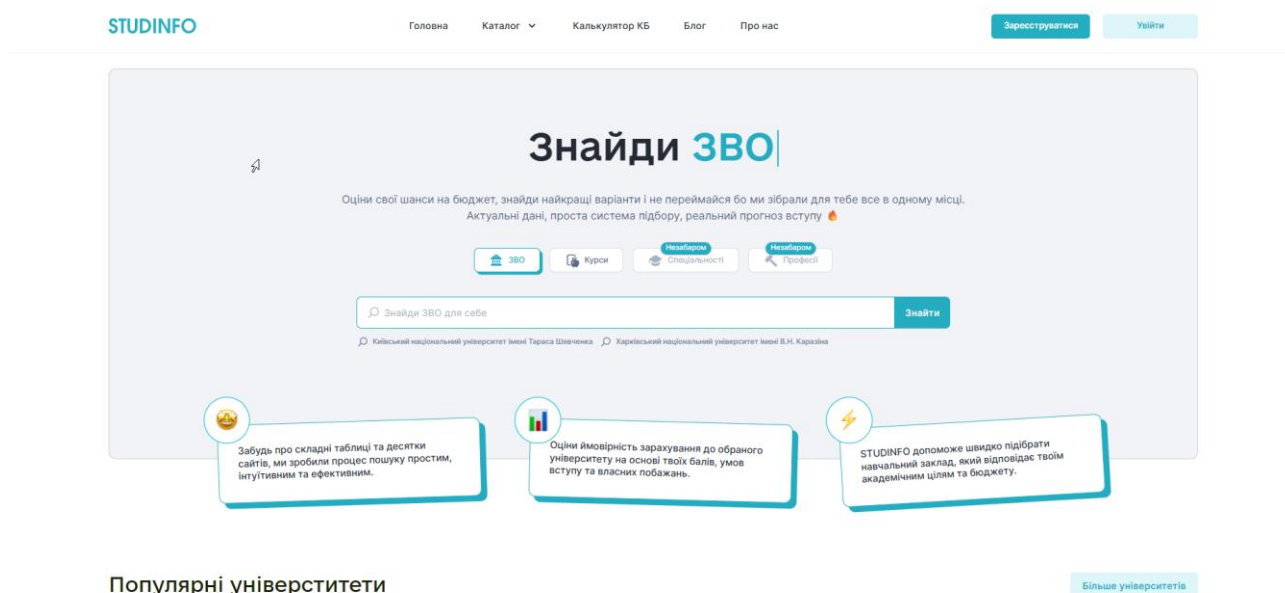


Рис. 4.1 Демонстрація Него-секції головного лендінгу

Секція "Що ти зможеш зробити зі STUDINFO?" демонструє основні функціональні можливості платформи через інтерактивні вкладки, а також функції які наразі знаходяться в розробці, що дає розуміння перспектив системи. На рисунку 4.2 зображено секцію функціональних можливостей головного лендінгу.

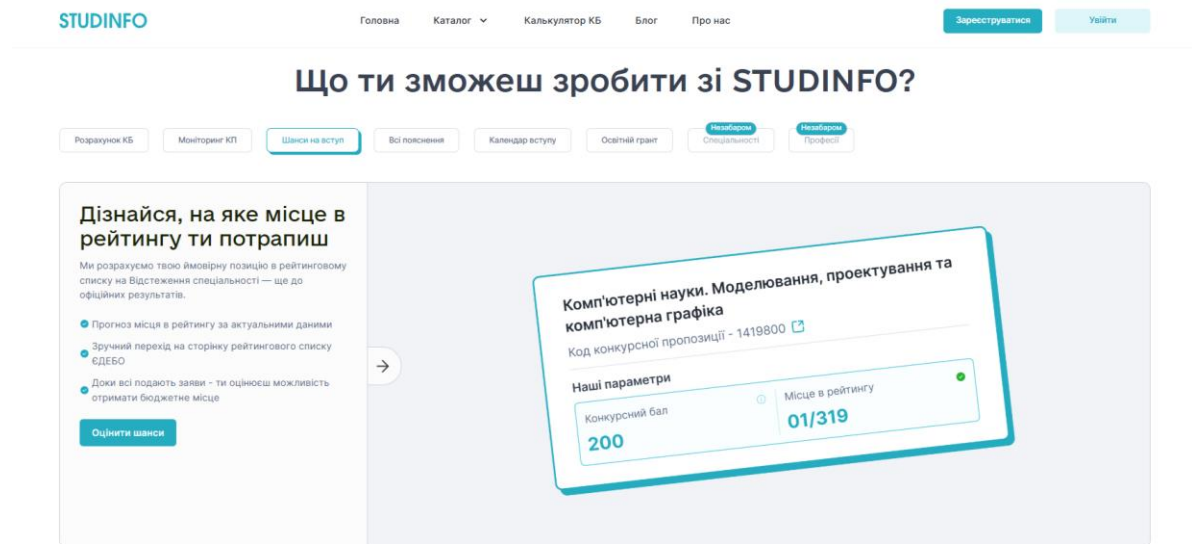


Рис. 4.2 Секція функціональних можливостей головного лендінгу STUDINFO

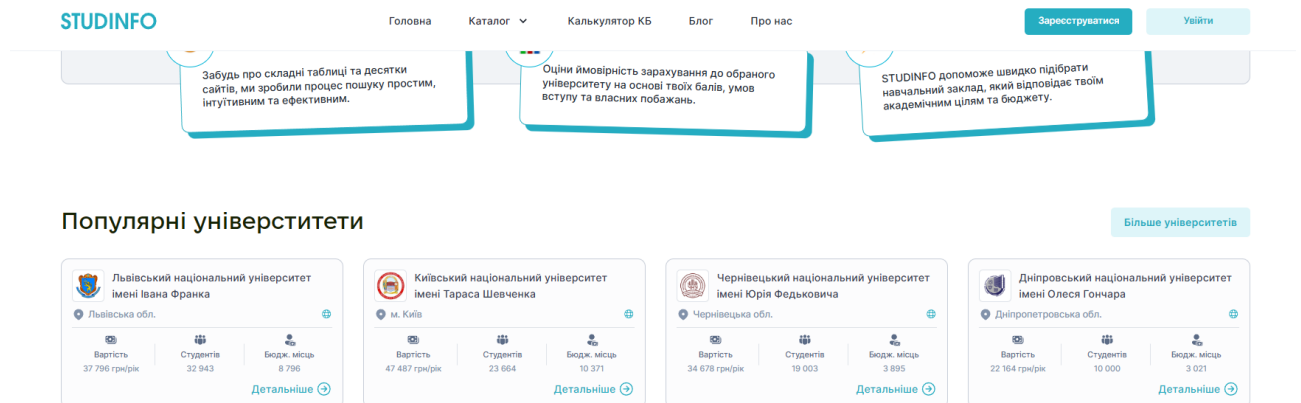


Рис. 4.3 Демонстрація секції головного лендінгу з популярними університетами

Сторінка реєстрації це стандартна форма реєстрації з вибором типу користувача – Студент, Абітурієнт або Інше та усіма стандартними полями. Також є можливість реєстрації через Telegram. На рисунку 4.4 зображено сторінку реєстрації STUDINFO.

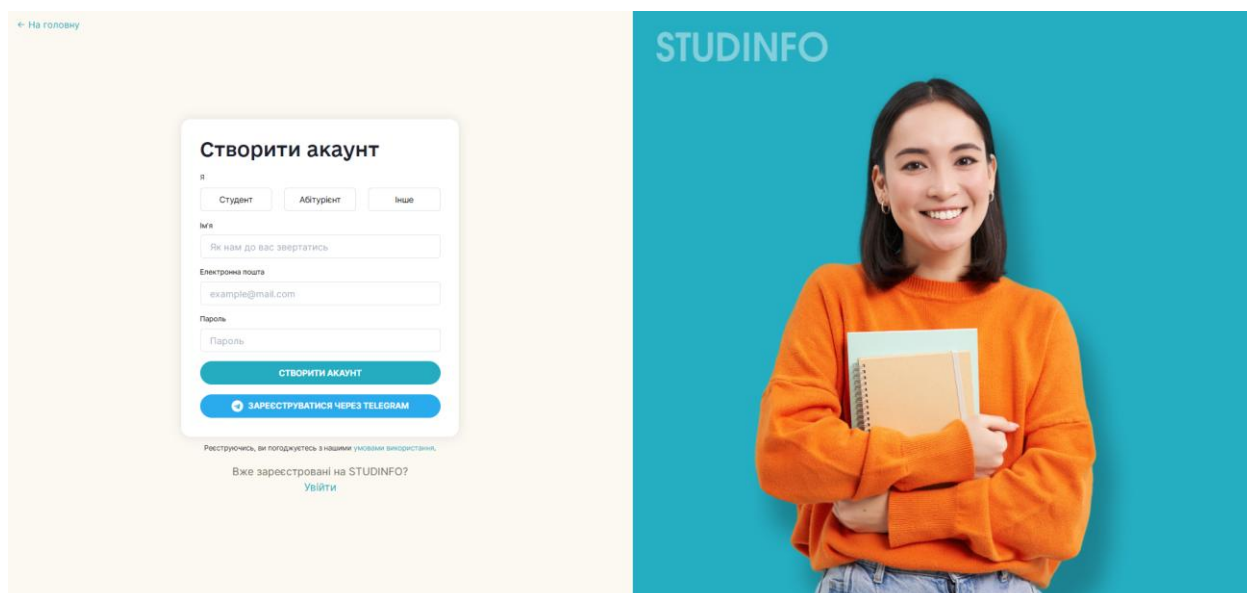


Рис. 4.4 Сторінка реєстрації STUDINFO

Особистий кабінет користувача створений як головний хаб всієї необхідної інформації. Відображається календар вступу з усіма етапами, статистика по ЗВО та КП в Україні, а також збережені університети та конкурсні пропозиції. На рисунку 4.5 зображено особистий кабінет вступника STUDINFO.

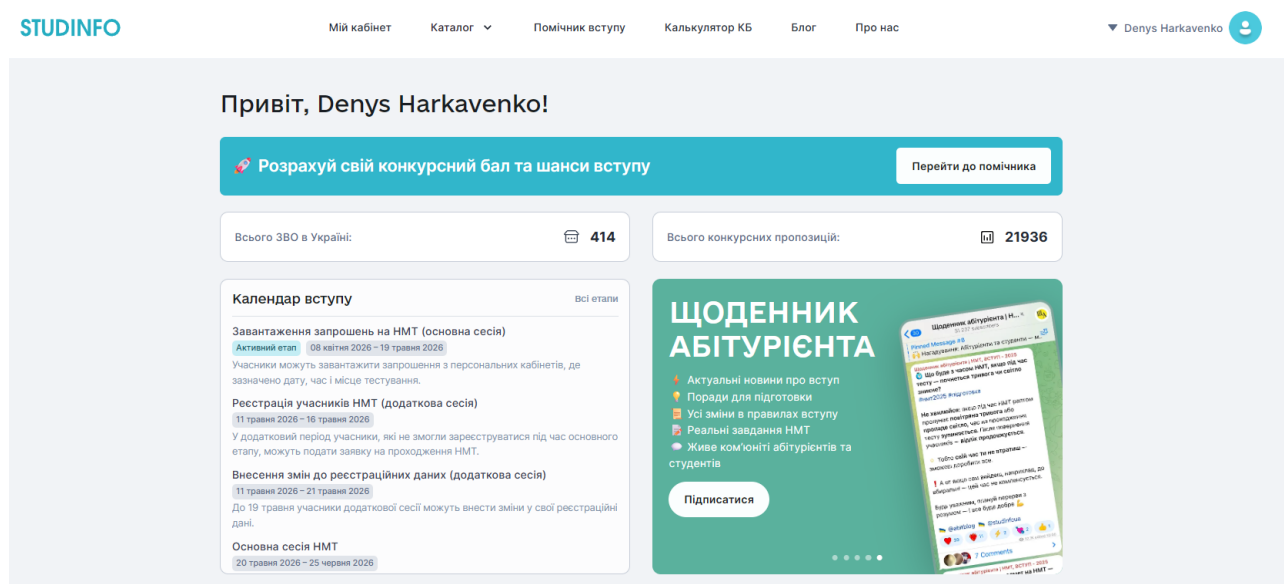


Рис. 4.5 Особистий кабінет вступника STUDINFO

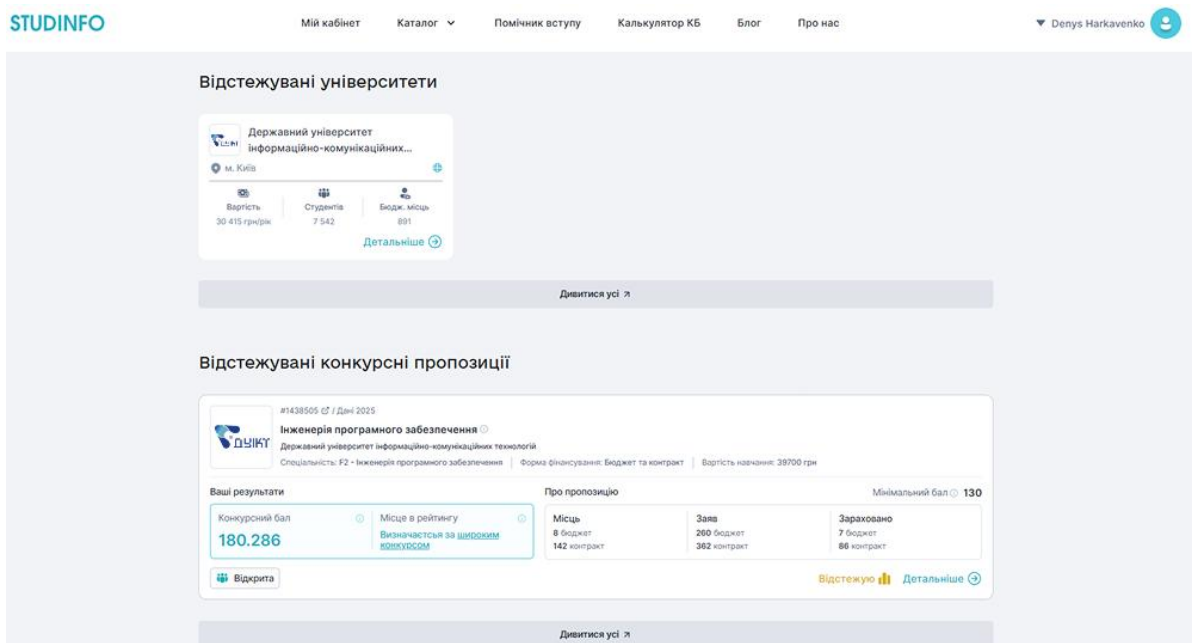


Рис. 4.6 Особистий кабінет вступника STUDINFO

Анкета вступника є найважливішим етапом для користувача. Після відповідей на всі запитання стає доступним основний функціонал системи - формується система рекомендацій ЗВО та конкурсних пропозицій, а також можливість зберігати КП та ЗВО в особистий кабінет. На рисунку 4.7 продемонстровано одне з дев'яти питань в анкеті вступника.

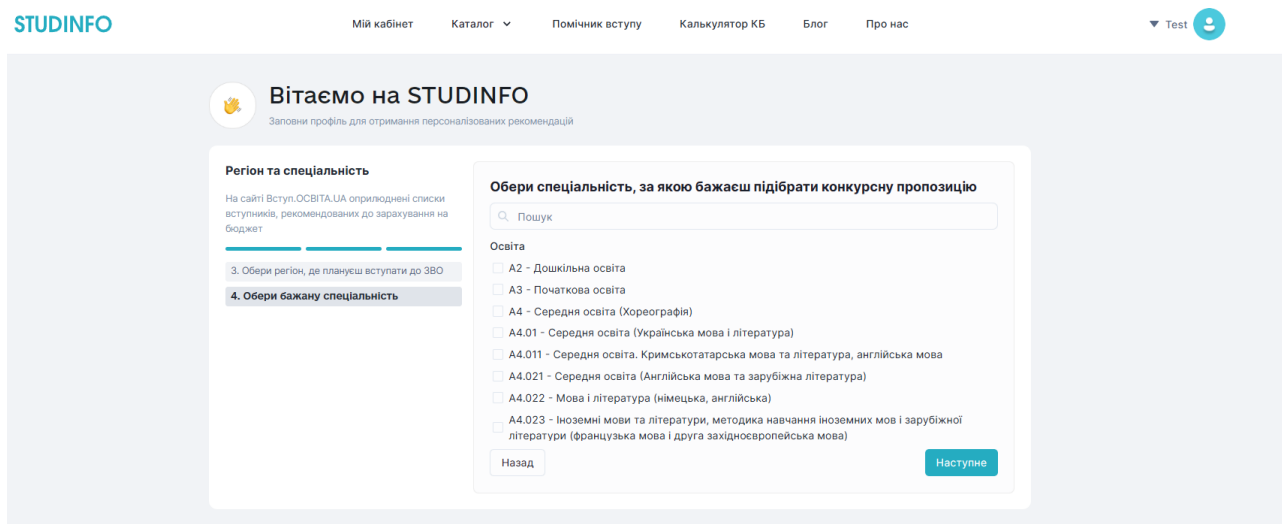


Рис. 4.7 Приклад питання в анкеті вступника

Після заповнення анкети вступника, формується список рекомендованих конкурсних пропозицій на базі відповідей користувача. Є можливість перегляду та редагування всіх введених даних, а також сортування за університетами та фільтрація за рекомендацією (шансом на вступ включно). На рисунку 4.8 продемонстровано каталог запропонованих конкурсних пропозицій, які відповідають побажанням користувача.

The screenshot shows the 'Конкурсні пропозиції' (Competitive Proposals) section of the STUDINFO application. On the left is a sidebar with the user's profile, including their name (Denys Harkavenko) and a list of subjects with scores. The main area displays a list of proposals. The first proposal is from the National Technical University of Ukraine 'Kyiv Polytechnical Institute' (NTU 'KPI'), Faculty of Informatics and Computer Engineering, Specialty F2 'Engineering of software security'. The user's score is 180.286, and they are ranked 220th. The second proposal is from the National University 'Kyiv-Mohyla Academy', Faculty of Engineering, Specialty F2 'Engineering of software security'. The user's score is 180.286, and they are ranked 61st. The third proposal is from the Institute of Atomic and Thermal Energy, Faculty of Engineering, Specialty F2 'Engineering of software security, Bachelor's degree'. The user's score is 180.286, and they are ranked 61st.

Рис. 4.8 Каталог конкурсних пропозицій

Є можливість перегляду детальної інформації для кожної конкурсної пропозиції, де нада вся необхідна інформація - гранти, загальна інформація про університет, умови вступу та навчання, статистика вступу. Також одразу розрахований конкурсний бал користувача і розраховано місце в рейтингу. Присутній функціонал для збереження КП в обрані, що полегшить її відстеження. На рисунку 4.9 зображено інформацію про конкурсну пропозицію "Інженерія програмного забезпечення" в Державний університеті інформаційно-комунікаційних технологій, як приклад сторінки з детальною інформацією про конкурсну пропозицію.

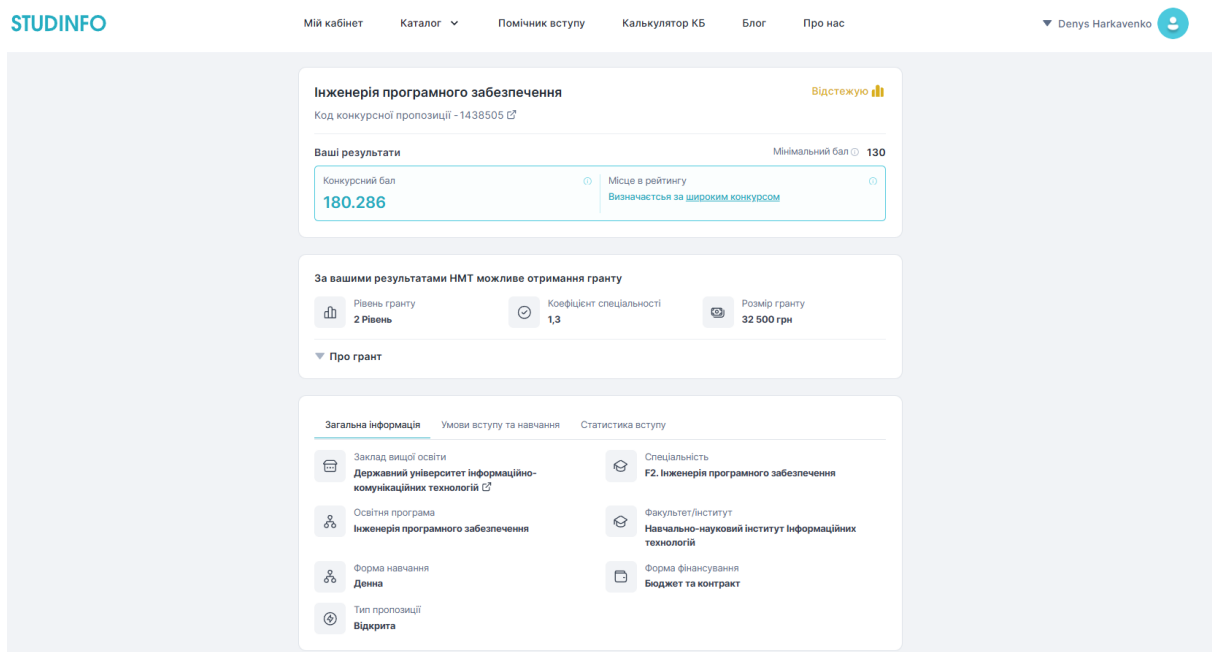


Рис. 4.9 Сторінка детальної інформації конкурсної пропозиції

Зі сторінки конкурсної пропозиції або за допомогою основної навігації сайту можна перейти на каталог університетів або конкретний університет. Є можливість переглянути всі університети в Україні, переглянути обрані університети, а також відсортувати каталог за допомогою різних фільтрів. На рисунку 4.10 зображено каталог університетів.

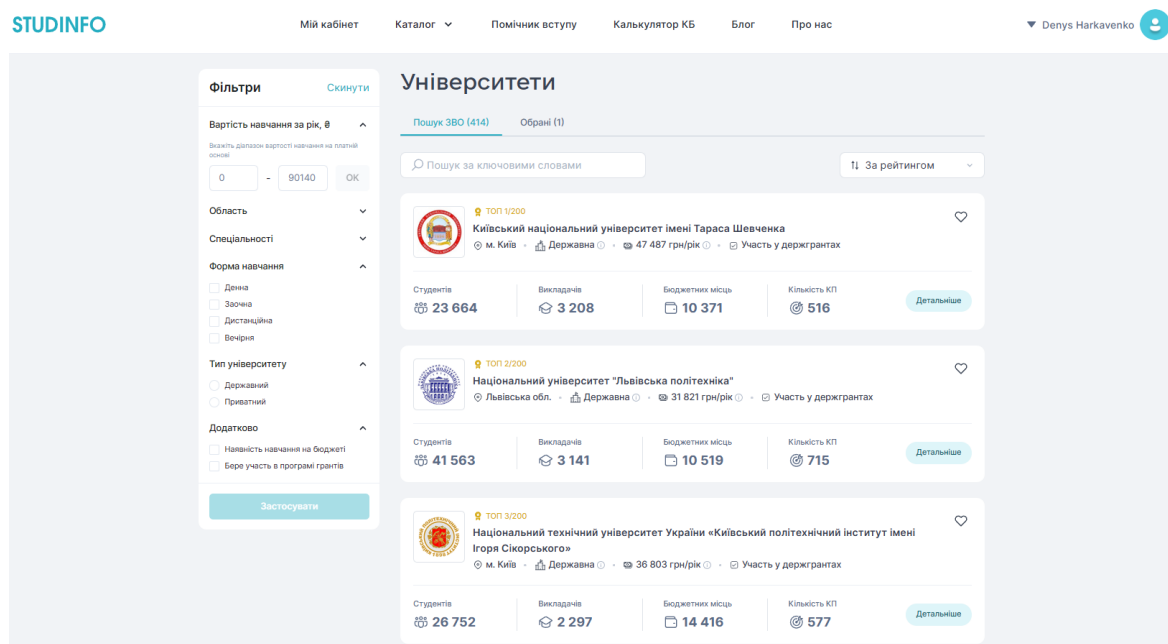


Рис. 4.10 Каталог університетів

Також є можливість переглянути сторінку університету з детальною інформацією про нього, рейтингом в Україні, спеціальностями та освітніми програмами, контактами та соцмережами. На рисунку 4.11 зображено сторінку Державного університету інформаційно-комунікаційних технологій.

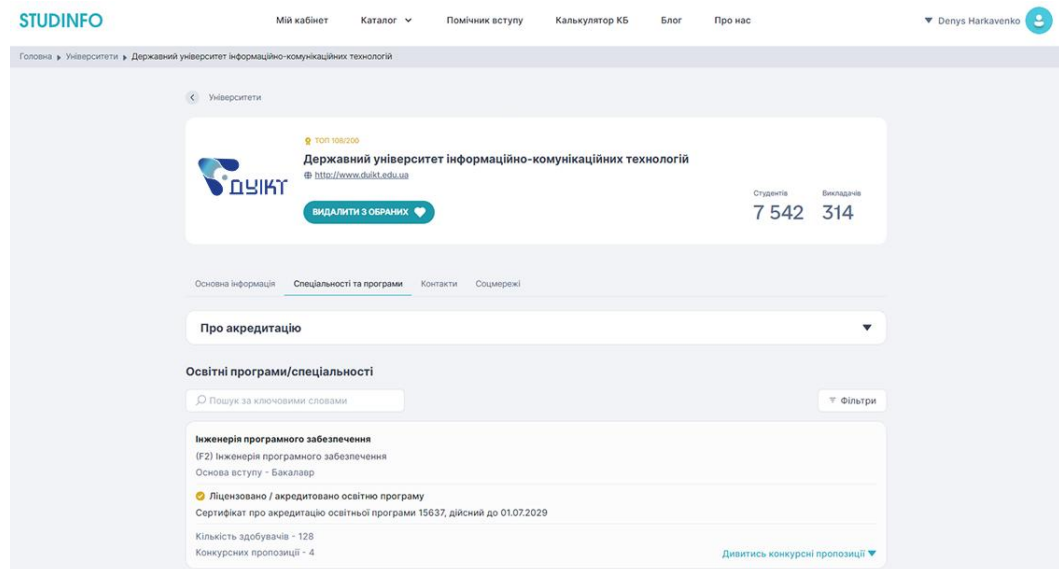


Рис. 4.11 Перегляд сторінки ДУІКТ за допомогою web-сервісу STUDINFO

Розрахувати свій КБ можна за допомогою сторінку розрахунку конкурсного балу. На рисунку 4.12 показано розрахунок КБ для спеціальності "F2 - Інженерія програмного забезпечення".

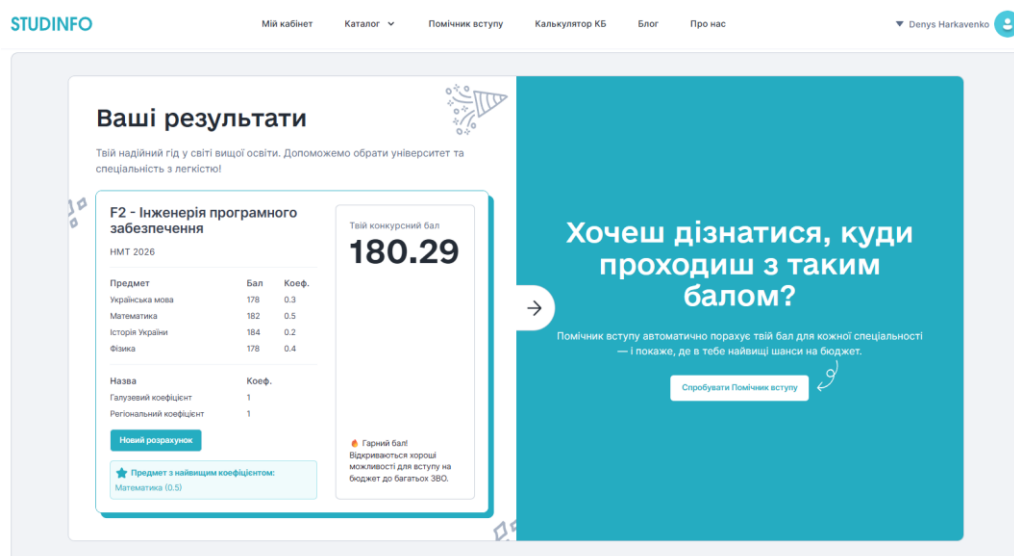


Рис. 4.12 розрахунок КБ за допомогою STUDINFO

4.2 Тестування розробленого web-сервісу

Тестування web-сервісу проводилось методом мануального тестування – перевірка основних функціональних сценаріїв через інтерфейс платформи. Для кожного тест-кейсу визначався очікуваний результат і фіксувався статус виконання. Тестування охоплювало основні модулі системи – реєстрацію і авторизацію, анкету вступника, модуль помічника вступу, каталог університетів і калькулятор конкурсного бала.

Таблиця 4.1

Тест кейси для web-сервісу STUDINFO

№	Тест-кейс	Очікуваний результат	Статус
1	Реєстрація з коректними даними	Акаунт створено, користувач авторизований, анкета ініціалізована	Працює
2	Реєстрація з існуючим email	Відображення повідомлення про помилку	Працює
3	Авторизація з коректними даними	Успішний вхід, перехід до кабінету	Працює
4	Авторизація з невірним паролем	Відображення помилки "Invalid Email or password"	Працює
5	Проходження анкети вступника (8 кроків)	Збереження відповідей на кожному кроці, перехід до помічника після завершення	Працює
6	Повернення на попередній крок анкети	Відображення попереднього кроку без втрати введених даних	Працює
7	Перехід до наступного кроку без заповнення полів	Відображення toast-повідомлення про необхідність заповнення	Працює

Продовження таблиця 4.1

Тест кейси для web-сервісу STUDINFO

8	Відображення списку конкурсних пропозицій	Список КП відфільтрований відповідно до параметрів анкети з розрахованим КБ	Працює
9	Додавання КП до відстежуваних	КП додається до вкладки "Відстеження", запускається розрахунок статистики	Працює
10	Фільтрація КП за університетом	Список оновлюється відповідно до обраного університету	Працює
11	Перегляд детальної інформації про КП	Відображення загальної інформації, умов вступу і статистики	Працює
12	Пошук університету за ключовим словом	Список оновлюється відповідно до введенного запиту	Працює
13	Фільтрація університетів за регіоном	Відображаються університети обраного регіону	Працює
14	Додавання університету до обраних	Університет з'являється у вкладці "Обрані"	Працює
15	Розрахунок конкурсного бала в калькуляторі	Коректний розрахунок КБ відповідно до введених балів і коефіцієнтів	Працює
16	Зміна року НМТ в калькуляторі	Автоматичне скидання залежних полів	Працює
17	Перегляд детальної сторінки університету	Відображення основної інформації, спеціальностей, контактів і соцмереж	Працює

Всі 17 тест-кейсів мають статус «Працює». Критичних помилок не виявлено. Реєстрація і авторизація працюють коректно – система повертає відповідні повідомлення при невалідних даних і успішно створює акаунт з автоматичною ініціалізацією анкети. Також перевірено обробку помилок при введенні неправильного пароля та повторній реєстрації з уже існуючим email. У всіх випадках система коректно реагує на дії користувача та відображає відповідні повідомлення.

Анкета вступника стабільно працює на всіх 8 етапах. Дані зберігаються при переході між кроками, а повернення до попередніх етапів не призводить до втрати введеної інформації. При спробі переходу далі без заповнення обов'язкових полів система відображає toast-повідомлення з попередженням. Це дозволяє уникнути некоректного заповнення анкети та забезпечує цілісність даних користувача.

Помічник вступу формує список конкурсних пропозицій відповідно до параметрів анкети та розрахованого конкурсного бала. Було протестовано роботу фільтрації за університетом, перегляд детальної інформації про конкурсну пропозицію та додавання КП до вкладки «Відстеження». Після додавання система коректно запускає розрахунок статистики та оновлює дані в інтерфейсі.

Калькулятор конкурсного бала виконує правильний розрахунок КБ відповідно до введених результатів та коефіцієнтів. При зміні року НМТ система автоматично скидає залежні поля, що запобігає появі помилкових результатів. Також перевірено коректність взаємодії між клієнтською та серверною частинами системи – всі API-запити обробляються стабільно, а дані відображаються без затримок та помилок.

Отримані результати тестування підтверджують стабільність роботи основного функціоналу веб-сервісу STUDINFO та готовність системи до використання в умовах реальної вступної кампанії.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи розроблено web-платформу STUDINFO для інформаційної підтримки абітурієнтів під час вступної кампанії. Проаналізовано існуючі web-сервіси та виявлено спільні проблеми та недоліки. Виходячи з цього сформульовано вимоги до системи і спроектовано архітектуру за клієнт-серверною моделлю. Стек: Next.js 14 і TypeScript на фронтенді, Ruby on Rails 7.1 на бекенді, PostgreSQL, Redis, Sidekiq, Docker. Клієнтська частина – App Router, RTK Query, Chakra UI. Серверна – REST API через rswag, Devise, Pundit і CanCanCan для захисту, Blueprinter для серіалізації.

Реалізовано парсинг відкритих даних з ЄДЕБО через Faraday – з підтримкою rate limiting і retry-логіки, бо ЄДЕБО блокує запити без коректних заголовків і обмежує частоту звернень. Модуль "Помічник вступу" включає 8-крокову анкету вступника, після заповнення якої формується персоналізований список конкурсних пропозицій з розрахованим КБ і місцем в рейтингу. Каталог університетів підтримує розширену фільтрацію і повнотекстовий пошук через pg_search. Калькулятор КБ реалізує різні формули для 2023-2026 років. Адміністративна панель побудована на Active Admin з аналітикою через Ahoy.

Платформу розгорнуто на Vercel і Railway – з окремими staging і production середовищами. Тестування проводилось методом мануального тестування основних функціональних сценаріїв – реєстрація і авторизація, анкета вступника, помічник вступу, каталог університетів і калькулятор КБ. Протестовано 17 тест-кейсів – всі успішно виконуються. Критичних помилок не виявлено, система працює відповідно до визначених вимог.

Розроблена платформа закриває задачу комплексно – один інструмент замість кількох розрізнених сервісів. З можливих напрямків подальшого розвитку – розширення модуля рекомендацій через OpenAI, порівняння університетів і мобільний застосунок.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Гаркавенко Д.О., Замрій І.В. Розробка веб-платформи STUDINFO для автоматизації розрахунку конкурсних показників та інформаційної підтримки абітурієнтів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії» 26 листопада 2025р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.114-116.

2. Гаркавенко Д.О., Замрій І.В. Забезпечення безпеки інформації у web-платформі STUDINFO. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» 23 квітня 2026р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С.235-237.

ПЕРЕЛІК ПОСИЛАНЬ

1. Биков В. Ю., Спірін О. М., Пінчук О. П. Проблеми та завдання сучасного етапу інформатизації освіти. Наукове забезпечення розвитку освіти в Україні: актуальні проблеми теорії і практики. Київ, 2017. С. 191–198.
2. Вступна кампанія. Офіційний портал вступу [Електронний ресурс]. URL: <https://vstup.edbo.gov.ua/>.
3. ЄДЕБО. Єдина державна електронна база з питань освіти [Електронний ресурс]. URL: <https://info.edbo.gov.ua/>.
4. Morze N., Varchenko-Trotsenko L. Modern approaches to the design of educational information environment of the university. *Information Technologies and Learning Tools*. 2021. Vol. 81, No. 1. P. 1–18. URL: <https://doi.org/10.33407/itlt.v81i1.3751>.
5. Patel D., Patel F., Chauhan U. Recommendation Systems: Types, Applications, and Challenges. *International Journal of Computing and Digital Systems*. 2023. Vol. 13, no. 1. P. 851–868. URL: <https://doi.org/10.12785/ijcds/130168>.
6. Zhang Q., Lu J., Zhang G. Recommender Systems in E-learning. *Journal of Smart Environments and Green Computing*. 2022. No. 1(2). P. 76–89.
7. Abitly – платформа для абітурієнтів [Електронний ресурс]. URL: <https://abitly.org/>.
8. Osvita.ua – інформаційний портал для абітурієнтів [Електронний ресурс]. URL: <https://osvita.ua/>.
9. Education.ua – освітній портал України [Електронний ресурс]. URL: <https://www.education.ua/>.
10. Abit-poisk – сервіс для відстеження заяв вступників [Електронний ресурс]. URL: <https://abit-poisk.org.ua/>.
11. Wieruch R. *The Road to React: Your journey to master plain yet pragmatic React.js*. S. l. : s. n., 2022. 497 p.

12. Next.js Documentation [Электронный ресурс]. Vercel Inc. 2024. URL: <https://nextjs.org/docs>.

13. Server-side and client-side scripting: the differences [Электронный ресурс]. IONOS Digital Guide. 2024. URL: <https://www.ionos.co.uk/digitalguide/websites/web-development/server-side-and-client-side-scripting-the-differences/>.

14. Sushin P. V. Next.js Application Architecture for Best Performance [Электронный ресурс]. Medium. 2023. URL: <https://medium.com/@sushinpv/nextjs-application-architecture-for-best-performance-8f1d22e33ba1>.

15. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Sebastopol : O'Reilly Media, 2019. 352 p.

16. TypeScript Documentation [Электронный ресурс]. Microsoft. 2024. URL: <https://www.typescriptlang.org/docs/>.

17. Chakra UI Documentation [Электронный ресурс]. 2024. URL: <https://chakra-ui.com/docs/getting-started>.

18. Redux Toolkit Documentation [Электронный ресурс]. 2024. URL: <https://redux-toolkit.js.org/introduction/getting-started>.

20. Ruby on Rails Guides [Электронный ресурс]. 2024. URL: <https://guides.rubyonrails.org>.

21. Hartl M. Ruby on Rails Tutorial: Learn Web Development with Rails. 7th ed. New York : Addison-Wesley, 2022. 816 p.

20. Sidekiq Documentation [Электронный ресурс]. 2024. URL: <https://github.com/sidekiq/sidekiq/wiki>.

22. Devise Documentation [Электронный ресурс]. Heartcombo. 2024. URL: <https://github.com/heartcombo/devise>.

23. OWASP Foundation. OWASP Top 10: The Ten Most Critical Web Application Security Risks [Электронный ресурс]. 2021. URL: <https://owasp.org/www-project-top-ten/>.

24. Ruby on Rails. Securing Rails Applications [Электронный ресурс]. Ruby on Rails Guides. 2024. URL: <https://guides.rubyonrails.org/security.html>.

25. PostgreSQL Architecture [Электронный ресурс]. Cloud Duggu. 2023. URL: <https://www.cloudduggu.com/postgresql/architecture/>.
26. Copeland R. PostgreSQL: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2023. 362 p.
27. PostgreSQL Documentation [Электронный ресурс]. PostgreSQL Global Development Group. 2024. URL: <https://www.postgresql.org/docs>.
28. Vercel Documentation [Электронный ресурс]. Vercel Inc. 2024. URL: <https://vercel.com/docs>.
29. Kane S., Matthias K. Docker: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2023. 354 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



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



Розробка web-сервісу "STUDINFO" для допомоги абітурієнтам під час вступу до ЗВО

Виконав студент 4 курсу
групи ПД-41
Гаркавенко Денис Олегович
Керівник роботи
д-р техн. наук, професор Замрій Ірина Вікторівна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу підтримки абітурієнтів під час вступної кампанії через автоматизацію розрахунку конкурсних показників та надання інформаційної підтримки.
- **Об'єкт дослідження** – процеси інформаційної підтримки абітурієнтів під час вступної кампанії та аналізу даних вступної кампанії.
- **Предмет дослідження** - засоби та технології для розробки сервісу інформаційної підтримки вступників за допомогою автоматизації розрахунку конкурсних показників, анкетування для підбору конкурсних пропозицій.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область інформаційної підтримки абітурієнтів під час вступної кампанії та дослідити існуючі web-сервіси для допомоги абітурієнтам.
2. Визначити недоліки існуючих рішень та сформулювати вимоги до розроблюваної системи.
3. Обґрунтувати вибір засобів реалізації клієнтської і серверної частин web-сервісу.
4. Спроекувати архітектуру web-сервісу STUDINFO, структуру бази даних та API.
5. Реалізувати web-сервіс STUDINFO — парсинг відкритих даних ЄДЕБО, модуль "Помічник вступу", калькулятор конкурсного бала та панель адміністрування.
6. Провести тестування розробленого web-сервісу.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Abitly	Education.ua	Osvita.ua	Abit-Poisk	STUDINFO
Розрахунок шансу на вступ до ЗВО	-	-	-	-	+
Рейтинг університетів	+	-	±	-	+
Пошук та фільтрація освітніх програм	±	±	-	-	+
Персоналізація	±	-	-	-	+
Особистий кабінет	+	-	±	-	+
Збереження даних користувача	-	-	±	±	+
Комплексний аналіз вступу (шанс на вступ, КБ, календар вступу)	-	-	-	-	+
Персоналізована рекомендація ЗВО та конкурсних пропозицій	±	-	-	-	+
Оновлення даних вступних кампаній в реальному часі	-	-	±	±	±
Час обробки запиту <2 секунд	-	+	+	-	±
Актуальність даних	±	±	±	±	+
Адаптивність інтерфейсу	±	+	±	±	+
Повнота наповнення БД	±	+	+	±	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Реєстрація та авторизація користувача через email і пароль з валідацією формату даних
2. Особистий кабінет: збереження результатів розрахунків, відкладених університетів і спеціальностей, персональних параметрів
3. Розрахунок КБ на основі результатів НМТ, вагових коефіцієнтів предметів і додаткових балів
4. Модуль «Помічник вступу»: підбір університетів і спеціальностей під індивідуальні параметри користувача з персоналізованими рекомендаціями
5. Оновлення даних про конкурсні пропозиції і рейтингові списки в реальному часі
6. Каталог університетів із фільтрацією за спеціальністю, регіоном, формою навчання, КБ
7. Перегляд конкурсних пропозицій із деталізацією умов вступу і кількістю бюджетних місць
8. Адміністративна панель для керування даними

Нефункціональні вимоги

1. Час відповіді API — до 500 мс для стандартних запитів, до 2 секунд для важких розрахунків
2. Захист даних користувача через JWT-авторизацію, HTTPS і розмежування прав доступу через Pundit
3. Система має витримувати одночасне навантаження від щонайменше 500 користувачів без деградації продуктивності — актуально в пік вступної кампанії
4. Контейнеризація через Docker для відтворюваності середовища на різних платформах
5. Моніторинг помилок у реальному часі через Sentry з автоматичними сповіщеннями

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



Ruby



TypeScript



Vercel



Docker



PostgreSQL



Sentry



RTK



Ruby on Rails



Chakra UI



Sidekiq

Sidekiq



Next.js



Railway app



Devise



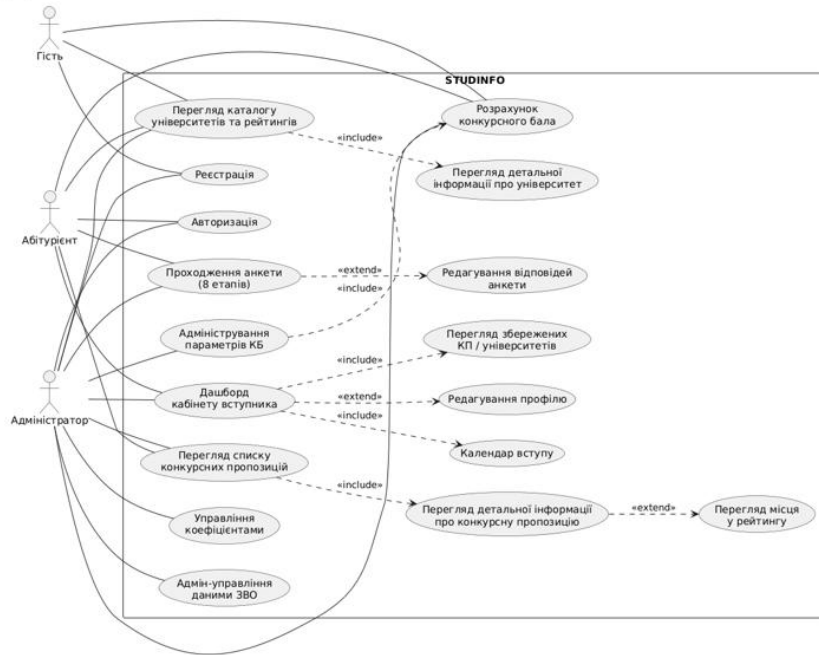
GitHub



Jest

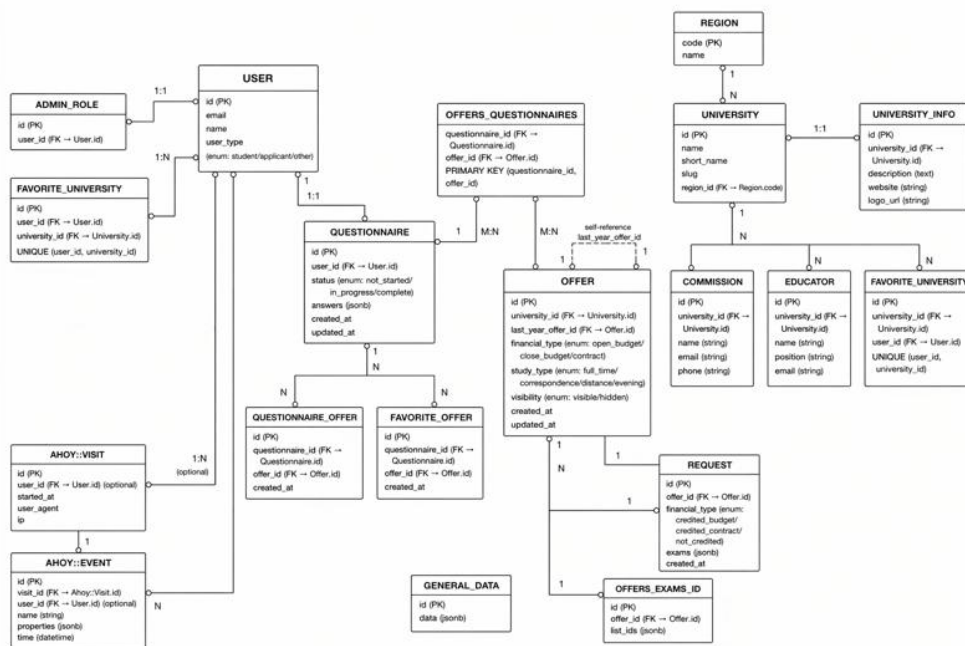
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



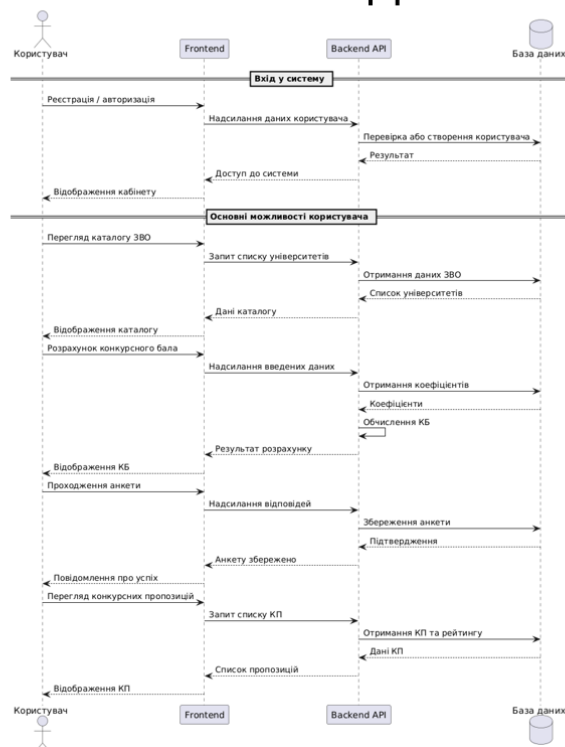
7

СХЕМА ЗВ'ЯЗКІВ МІЖ СУТНОСТЯМИ БАЗИ ДАНИХ



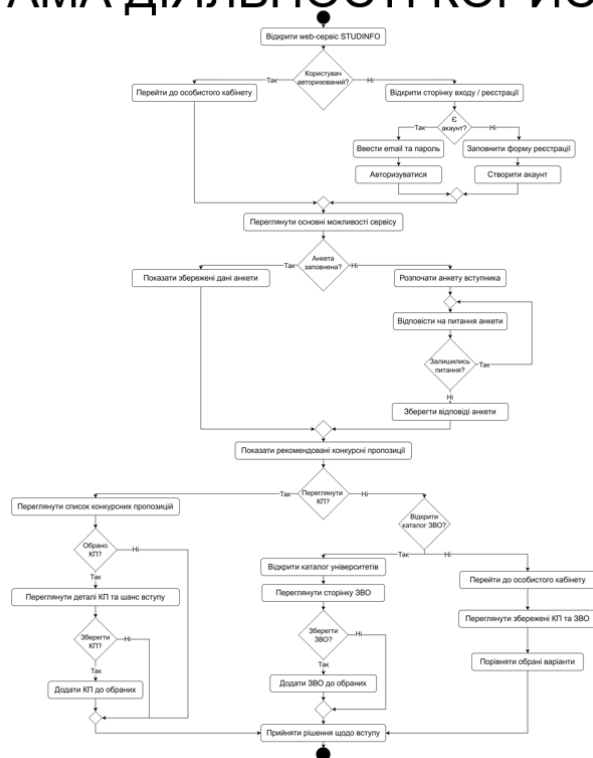
8

ДІАГРАМА ПОСЛІДОВНОСТІ



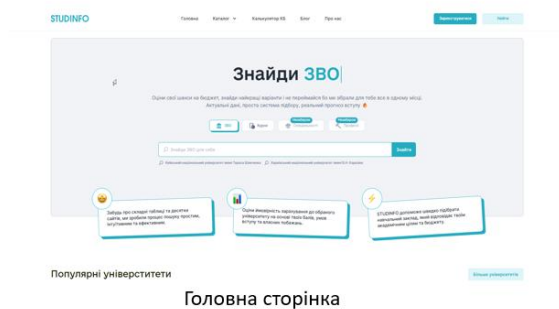
9

ДІАГРАМА ДІЯЛЬНОСТІ КОРИСТУВАЧА

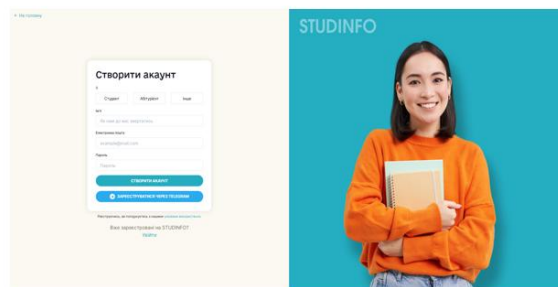


10

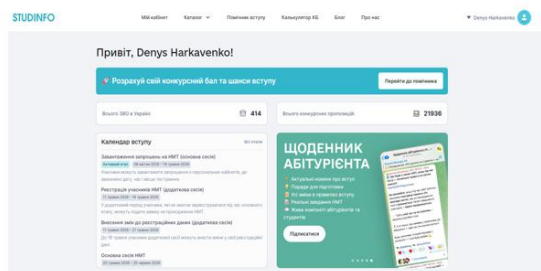
ЕКРАННІ ФОРМИ



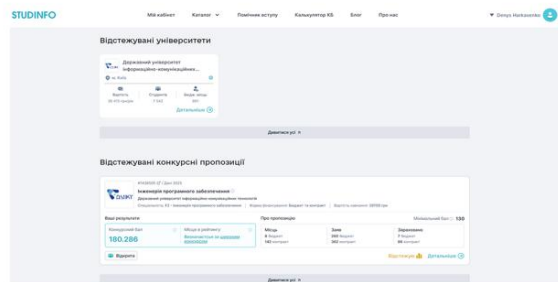
Головна сторінка



Форма реєстрації



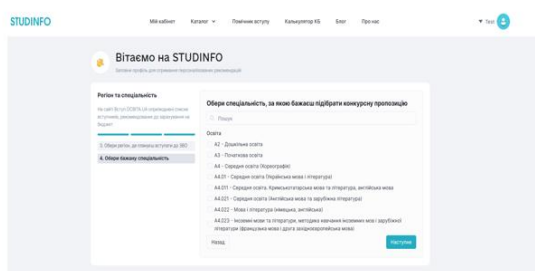
Кабінет вступника



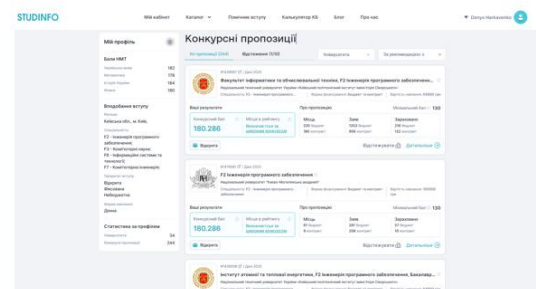
Обрані КП та ЗВО в кабінеті вступника

11

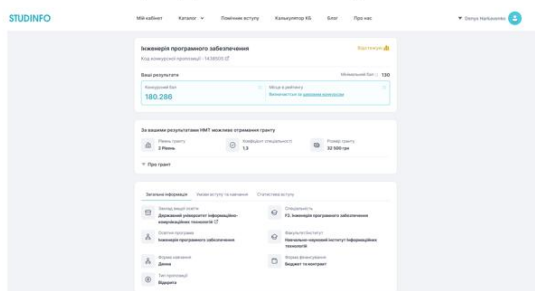
ЕКРАННІ ФОРМИ



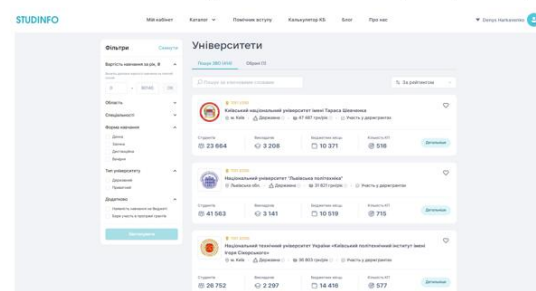
Проходження анкети вступника



Каталог КП з конкурсним балом та місцем в рейтингу



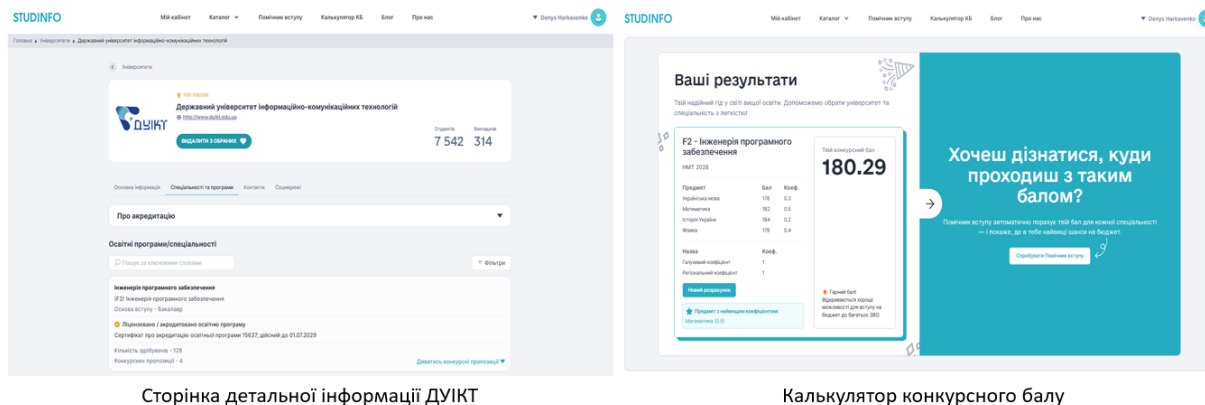
Сторінка детальної інформації КП



Каталог університетів відсортованих за рейтингом

12

ЕКРАННІ ФОРМИ



Сторінка детальної інформації ДУІКТ

Калькулятор конкурсного балу

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Гаркавенко Д.О., Замрій І.В. Розробка веб-платформи STUDINFO для автоматизації розрахунку конкурсних показників та інформаційної підтримки абітурієнтів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії» 26 листопада 2025р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.114-116.

2. Гаркавенко Д.О., Замрій І.В. Забезпечення безпеки інформації у web-платформі STUDINFO. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» 23 квітня 2026р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С.235-237.

14

ВИСНОВКИ

1. Проаналізовано предметну область інформаційної підтримки абітурієнтів та досліджено існуючі web-сервіси – Abitly, Osvita.ua, Education.ua, abit-poisk.org.ua. Виявлено спільні недоліки: відсутність розрахунку КБ відносно конкурсної пропозиції, відсутність персоналізації, розрізненість даних між платформами.
2. Визначено недоліки існуючих рішень та сформульовано функціональні і нефункціональні вимоги до системи STUDINFO.
3. Обґрунтовано вибір технологічного стеку – Next.js 14 і TypeScript на фронтенді, Ruby on Rails 7.1 на бекенді, PostgreSQL, Redis, Sidekiq, Docker.
4. Спроєктовано архітектуру web-сервісу за клієнт-серверною моделлю, структуру бази даних та REST API.
5. Реалізовано web-сервіс STUDINFO – парсинг даних ЄДЕБО, модуль "Помічник вступу" з 8-кроковою анкетною і персоналізованим списком КП, калькулятор КБ для 2023-2026 років, каталог університетів з фільтрацією і пошуком, адміністративна панель на Active Admin.
6. Проведено мануальне тестування 17 тест-кейсів – усі сценарії виконуються коректно та відповідають функціональним вимогам.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
# frozen_string_literal: true

class Users::RegistrationsController <
  Devise::RegistrationsController
    respond_to :json
    before_action :configure_sign_up_params,
    only: :create

    def create
      build_resource(sign_up_params)
      resource.save

      if resource.persisted?
        sign_in(resource_name, resource)
        return render json: resource, status:
:created
      end

      error_result = {
        errors: resource.errors.to_hash(true),
        email_confirmed: User.find_by(email:
resource_params[:email]).try(:email_confirme
d?)
      }
      render json: error_result, status:
:unprocessable_entity
    end

    protected

    def configure_sign_up_params
      devise_parameter_sanitizer.permit(:sign_up,
keys: %i[name password user_type])
    end

    def build_resource(*)
      super

      resource.build_questionnaire
      resource
    end
  end

# frozen_string_literal: true

class Users::SessionsController <
  Devise::SessionsController
    respond_to :json

    # POST /api/auth
    def create
      self.resource =
warden.authenticate(auth_options)
      if resource
        sign_in(resource_name, resource)
        render json: {
          message: 'Logged in successfully',
```

```
import GptProvider from '../components/gpt-ad/GptProvider'
import '../index.css'
import { ClientProviders } from './ClientProviders'
import Script from 'next/script'

export const metadata = {
  title: 'STUDINFO: розрахунок ймовірності вступу легко та
швидко 🚀',
  description:
    'Інноваційний освітній застосунок, який допоможе
автоматично розрахувати конкурсний бал та шанси на вступ!
Жоден вибір місця для вступу не обійдеться без STUDINFO.',
  metadataBase: new URL('https://studinfo.org/'),
  alternates: {
    canonical: 'https://studinfo.org/',
  },
  robots: 'index, follow',
  openGraph: {
    title: 'STUDINFO: розрахунок ймовірності вступу легко та
швидко 🚀',
    description:
      'Інноваційний освітній застосунок, який допоможе
автоматично розрахувати конкурсний бал та шанси на вступ!
Жоден вибір місця для вступу не обійдеться без STUDINFO.',
    type: 'website',
    url: 'https://studinfo.org/',
    images: [
      {
        url: 'https://studinfo.org/img/metadata/main.png',
        width: 1200,
        height: 630,
        alt: 'STUDINFO: інноваційний освітній застосунок',
      },
    ],
  },
  twitter: {
    card: 'summary_large_image',
    title: 'STUDINFO: розрахунок ймовірності вступу легко та
швидко 🚀',
    description:
      'Інноваційний освітній застосунок, який допоможе
автоматично розрахувати конкурсний бал та шанси на вступ!
Жоден вибір місця для вступу не обійдеться без STUDINFO.',
    url: 'https://studinfo.org/',
    images: ['https://studinfo.org/img/metadata/main.png'],
  },
  icons: [
    { url: '/favicon.ico', type: 'image/x-icon' },
    { url: '/favicon.svg', type: 'image/svg+xml' },
  ],
}

export default function RootLayout({
  children,
}): {
  children: React.ReactNode
} {
  return (
```

```

      user:
      UserBlueprint.render_as_hash(resource, view:
:public),
      tokens_used: resource.tokens_used
    }, status: :ok
  else
    render json: { error: 'Invalid Email or
password.' }, status: :unauthorized
  end
end

# DELETE /api/sign_out
def destroy
  sign_out(resource_name)
  render json: { message: 'Logged out
successfully' }, status: :ok
end

protected

def respond_to_on_destroy
  head :no_content
end

def set_flash_message!(*); end

# frozen_string_literal: true

class UsersController < ApplicationController
  def current
    return head :unauthorized if
current_user.nil?

    render json: {
      current_user:
UserBlueprint.render_as_json(current_user,
view: :public),
      tokens_used: current_user.tokens_used,
      questionnaire_id:
current_user.questionnaire.id,
      questionnaire_status:
current_user.questionnaire.status
    }, status: 200
  end

  def update
    return head :unauthorized if
current_user.nil?

    current_user.update!(user_params)

    head :ok
  end

  def destroy
    return head :unauthorized if
current_user.nil?

    current_user.destroy!
    sign_out

    head :ok

```

```

<html lang="uk">
<body>
  <GptProvider>
    <noscript>
      <iframe
        src="{`https://www.googletagmanager.com/ns.html?id=GT
M-${process.env.NEXT_PUBLIC_ANALYTICS_ID_1}`}"
        height="0"
        width="0"
        style={{ display: 'none', visibility: 'hidden' }}
      ></iframe>
    </noscript>

    <ClientProviders>
      <div id="root">{children}</div>
    </ClientProviders>

    <Script
      id="gtm-config"
      strategy="beforeInteractive"
      dangerouslySetInnerHTML={{
        __html: `
        (function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
        new Date().getTime(),event:'gtm.js'});var
        f=d.getElementsByTagName(s)[0],
        j=d.createElement(s),dl=!!'dataLayer'?&l='+l:'';j.async=t
        rue;j.src=
        'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.pa
        rentNode.insertBefore(j,f);
        })(window,document,'script','dataLayer','${process.env.N
        EXT_PUBLIC_ANALYTICS_ID_1}');
        `
      }}
    />

    <Script
      strategy="afterInteractive"
      src="{`https://www.googletagmanager.com/gtag/js?id=${pr
        ocess.env.NEXT_PUBLIC_ANALYTICS_ID_2}`}"
    />
    <Script id="data-layer" strategy="lazyOnload">
      {`
      window.dataLayer = window.dataLayer || [];
      function gtag(){dataLayer.push(arguments);}
      gtag('js', new Date());
      gtag('config',
      '${process.env.NEXT_PUBLIC_ANALYTICS_ID_2}', {
        page_path: window.location.pathname,
      });
      `}
    </Script>
    /* <Script
      id='cookieyes'
      strategy='beforeInteractive'
      src='https://cdn-
      cookieyes.com/client_data/cff13e66f3c065f05dfb78bc/script.js'
    /> */
    <Script
      id="adsense"
      strategy="afterInteractive"
      async
      src="https://pagead2.googlesyndication.com/pagead/js/adsb
      ygoogle.js?client=ca-pub-1451582410812539"

```

```

end

def confirmation
  if
    User.confirm_by_token(params[:confirmation_token])
    redirect_to
    "#{Settings.application_host}/workspace",
    allow_other_host: true
  else
    redirect_to
    "#{Settings.application_host}/app",
    allow_other_host: true
  end
end

private

def user_params
  params.permit(:email, :name, :user_type)
end

end

# frozen_string_literal: true

class RequestsController <
  ApplicationController
  protect_from_forgery with: :null_session

  def statistics
    offer = Offer.find(params[:offer_id])

    stat = OfferRequestsStatistic.find_by!(
      offer_id: offer.id,
      campaign_year: offer.entry_year.to_i
    )

    include_raw =
      ActiveModel::Type::Boolean.new.cast(params[:include_raw])

    render json:
      OfferRequestsStatisticBlueprint.render(
        stat,
        view: :api,
        include_raw:
      )
  end
end

# frozen_string_literal: true

class QuestionnaireController <
  ApplicationController
  include Devise::Controllers::SignInOut

  before_action :authenticate_user!

  def show
    render json:
      QuestionnaireBlueprint.render(questionnaire),
      status: :ok
  end
end

```

```

      crossOrigin="anonymous"
    />
  </GptProvider>
</body>
</html>
)
}

import { Heading, Flex } from '@chakra-ui/react'
import Logo from '../components/UI/svg/Logo'
import OurButton from '../components/UI/Buttons/OurButton'
import ZeroLetterSVG from
'../components/UI/svg/ZeroLetterSVG'

export const metadata = {
  title: '404 – Сторінку не знайдено | StudInfo',
}

const NotFoundPage = () => {
  const currentYear = new Date().getFullYear()

  return (
    <Flex bg="#FBF9F1" minH="100vh" direction="column"
      justify="space-between">
      <Flex
        direction="column"
        align="center"
        justify="center"
        maxW="1260px"
        p="0 20px"
        m="0 auto"
        flex="1"
      >
        <Heading display="flex" alignItems="center"
          fontSize="230px">
          4<ZeroLetterSVG />4
        </Heading>

        <Heading mt="50px" mb="38px" textAlign="center">
          Упс, щось пішло не по плану...
        </Heading>

        <OurButton href="/">
          На головну
        </OurButton>
      </Flex>

      <Flex
        justify="space-between"
        align="center"
        maxW="1260px"
        m="0 auto"
        p="0 20px 20px"
        w="100%"
      >
        <Heading fontSize="16px" color="#25ADC1">
          © {currentYear}, StudInfo
        </Heading>
        <Logo />
      </Flex>
    </Flex>
  )
}

```

```

def update
  Questionnaire::UpdateService.new(questionnaire:,
    params:
questionnaire_params)
    .call

  head :ok
end

private

def questionnaire
  current_user.questionnaire
end

def questionnaire_params
  params.permit(:step_id, answers: {})
end

# frozen_string_literal: true

class OffersCoefficientsUpdate
  def call
    Offer.where(coefficients_updated_at: nil).each do |offer|
      offer.update(
        coefficients_general_list: ProbabilityCalculator.new(offer:).call.to_a,
        coefficients_updated_at: Time.current
      )
    rescue
      ProbabilityCalculator::NoSuitableDataError
    end
  end

  # frozen_string_literal: true

  class UpdateQuestionnaireJob < ApplicationJob
    queue_as :default

    def perform
      Questionnaire.find_each do |questionnaire|
        questionnaire.in_progress!
        Questionnaire::CalculateScoreAndProbabilityJob.perform_later(questionnaire:)
      end
    end
  end

  class OfferAsQuestionnaireOfferBlueprint < Blueprinter::Base
    identifier :id

    field :competitive_score do |offer, options|
      QuestionnaireOffer::CalculateService.new(questionnaire: options.fetch(:questionnaire), offer:).competitive_score
    end
  end

```

```

export default NotFoundPage

'use client'

import React, { useEffect, useState } from 'react'
import { Flex, Box, Heading, Text, Link } from '@chakra-ui/react'
import {
  useCreateUserMutation,
  useGetCurrentUserQuery,
} from '../api/user.ts'
import InputField from '../login/components/InputField.tsx'
import { validateEmail, validatePassword } from '../validations/auth.tsx'
import CreateAccountButton from './CreateAccountButton.tsx'
import AccountSectionRegistration from './AccountSectionRegistration.tsx'
import TypeSelectionButtons from './TypeSelectionButtons.tsx'
import LoginImageSection from '../components/LoginFlow/LoginImageSection.tsx'
import MessDone from '../components/LoginFlow/MessDone.tsx'
import TelegramAuthButton from '../login/components/TelegramLoginButton.tsx'
import { useRouter } from 'next/navigation'
import NextLink from 'next/link'

const RegPage: React.FC = () => {
  const [selectedType, setSelectedType] = useState<'student' | 'applicant' | 'other' | null>(null)
  const [name, setName] = useState('')
  const [password, setPassword] = useState('')
  const [email, setEmail] = useState('')

  const [nameError, setNameError] = useState<string | null>(null)
  const [emailError, setEmailError] = useState<string | null>(null)
  const [passwordError, setPasswordError] = useState<string | null>(null)
  const [typeError, setTypeError] = useState<string | null>(null)
  const [registrationSuccess, setRegistrationSuccess] = useState(false)

  const [createUser, { isLoading: isCreating }] = useCreateUserMutation()

  const router = useRouter()

  const { data: currentUser, isLoading: userIsLoading } = useGetCurrentUserQuery(undefined, {
    refetchOnMountOrArgChange: true,
  })

  const [isFromAiAssistant, setIsFromAiAssistant] = useState(false)

  useEffect(() => {
    const params = new URLSearchParams(window.location.search)
    setIsFromAiAssistant(params.get('from') === 'ai-assistant')
  }, [])

  useEffect(() => {

```

```

    field :recommendation_score do |offer,
options|
    QuestionnaireOffer::CalculateService.new(q
uestionnaire: options.fetch(:questionnaire),
offer:).recommendation_score
    end

    field :rating_position do |offer, options|
    QuestionnaireOffer::CalculateService.new(q
uestionnaire: options.fetch(:questionnaire),
offer:).rating_position
    end

    field :minimal_score do |offer, options|
    QuestionnaireOffer::CalculateService.new(q
uestionnaire: options.fetch(:questionnaire),
offer:).minimal_score
    end

    field :open_offer?

    field :favorite? do |offer, options|
    questionnaire = options.fetch(:questionnaire)
    questionnaire.questionnaire_offers.find_by(
offer_id: offer.id)&.favorite? || false
    end

    field :calculation2023 do |_offer|
    nil
    end

    field :probability do |_offer, options|
    questionnaire = options.fetch(:questionnaire)
    questionnaire.in_progress? ? nil : '-'
    end

    association :offer, blueprint: OfferBlueprint,
view: :show do |offer|
    Edebo::ShowOfferDetailsService.new(Offer
Decorator.new(offer)).call
    end
end

# frozen_string_literal: true

class SpecialtyBlueprint < Blueprinter::Base
  identifier :id

  fields :name, :code, :description, :content,
:slug

  field :knowledge_area do |specialty|
    knowledge_area = specialty.knowledge_area
    knowledge_area ? { id: knowledge_area.id,
code: knowledge_area.code, name:
knowledge_area.name } : nil
    end
end

# frozen_string_literal: true

```

```

    if (!userIsLoading && currentUser) {
      router.push('/workspace')
    }
  }, [userIsLoading, currentUser, router])

  if (userIsLoading || currentUser) {
    return null
  }

  const onSubmit = async () => {
    setNameError(null)
    setEmailError(null)
    setPasswordError(null)
    setTypeError(null)

    let hasError = false

    if (!selectedType) {
      setTypeError('Оберіть тип користувача')
      hasError = true
    }

    if (!name.trim()) {
      setNameError('Ім\'я обов\'язкове')
      hasError = true
    }

    const emailErrorMessage = validateEmail(email)
    if (emailErrorMessage) {
      setEmailError(emailErrorMessage)
      hasError = true
    }

    const passwordErrorMessage = validatePassword(password)
    if (passwordErrorMessage) {
      setPasswordError(passwordErrorMessage)
      hasError = true
    }

    if (hasError) return

    try {
      await createUser({
        name,
        email,
        password,
        user_type: selectedType,
      }).unwrap()

      if (isFromAiAssistant && typeof window !== 'undefined') {
        window.dataLayer = window.dataLayer || []
        window.dataLayer.push({
          event: 'user_registered_from_ai_assistant',
          registration_source: 'ai-assistant',
        })
      }

      setRegistrationSuccess(true)
    } catch (error: any) {
      type BackendError = {
        errors?: {
          email?: string[]
          name?: string[]

```

```
class OffersController < ApplicationController
  before_action :authenticate_user!
  skip_before_action :authenticate_user!, only:
%i[popular]
```

```
AVERAGE_SCORE = 154.13
ITEMS_PER_PAGE = 6
```

```
def index
  pagy, records = pagy(offers, items:
ITEMS_PER_PAGE)
  render json: {
    offers: OfferBlueprint.render_as_json(
      records.includes(:university).includes(:sp
ecialty),
      questionnaire:,
      view: :registration
    ),
    count: offers.visible.count,
    count_all: Offer.where(entry_year:
2025).visible.count,
    average_score: AVERAGE_SCORE,
    pagination: pagy_metadata(pagy)
  }, status: :ok
end
```

```
def popular
  offers = Offer
    .visible
    .preload(:university)
    .left_joins(:requests)
    .group('offers.id')
    .order_by_order_expression
    .limit(30)
```

```
  render json: {
    offers:
OfferBlueprint.render_as_json(offers, view:
:card_public)
  }, status: :ok
end
```

```
def show
  render json: OfferBlueprint.render(
    Edebo::ShowOfferDetailsService.new(offe
r).call,
    view: :show,
    questionnaire: current_user.questionnaire
  ), status: :ok
end
```

```
private
```

```
def offers
  @offers ||=
Questionnaire::FilterService.new(questionnaire
:).call
end
```

```
def offer
  Offer.find(params[:id])
end
```

```
  password?: string[]
}
  email_confirmed?: boolean | null
}
```

```
let data: BackendError = {}
```

```
try {
  data =
    typeof error.data === 'string' ? JSON.parse(error.data) :
error.data
} catch (e) {
  console.error('Помилка при розпарсуванні JSON:', e)
}
```

```
const emailErrorText = data?.errors?.email?.[0]
```

```
if (emailErrorText?.toLowerCase().includes('taken')) {
  setEmailError('Ця електронна адреса вже зареєстрована')
} else if (emailErrorText) {
  setEmailError(emailErrorText)
} else {
  setEmailError('Сталася помилка. Спробуйте ще раз')
}
}
```

```
if (registrationSuccess) {
  return <MessDone action="confirm" />
}
```

```
return (
  <Flex flex="1" minH="100vh" position="relative">
    <Flex w="100%" minH="100vh">
      <Link
        href="/"
        as={NextLink}
        position="absolute"
        color="#25ACC1"
        top={4}
        left={4}
        _hover={{ textDecoration: 'none' }}
      >
        ← На головну
      </Link>
      <Flex
        bg="#FBF9F1"
        w={['100%', null, '50%']}
        justify="center"
        align="center"
      >
        <Flex direction="column" w="100%" align="center"
m="20px">
          <Box
            bg="#fff"
            borderRadius="16px"
            p="32px"
            h="fit-content"
            w="100%"
            maxW="474px"
            boxShadow="0 0 20px rgba(0, 0, 0, 0.1)"
          >
            <Heading
```

```

def questionnaire
  @questionnaire ||=
  Questionnaire.find(params[:questionnaire_id])
end
end
# frozen_string_literal: true

class RegionsController <
  ApplicationController
  def index
    if params[:questionnaire_id].present?
      render json:
      RegionBlueprint.render(Region.where(code:
      region_codes)), status: :ok
    else
      render json: { regions:
      RegionBlueprint.render_as_json(Region.all) },
      status: :ok
    end
  end

  private

  def region_codes
    Questionnaire.find(params[:questionnaire_id
  ]).answers_regions
  end
end

---
name: 'studinfo'

x-backend-app: &backend-app
build:
  context: .
  dockerfile: Dockerfile.dev
environment:
  EDITOR: vim
  REDIS_URL: redis://redis:6379
stdin_open: true
tty: true
volumes:
  - ./rails:rw
  - /rails/tmp/cache/bootsnap
  - /rails/vendor/bundle
  - rails_storage:/rails/storage

services:
  postgres:
    image: postgres:17.2-alpine
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - 5432:5432/tcp
    volumes:
      - /var/lib/postgresql/data

  redis:
    image: redis:7.0.15-alpine
    command: redis-server
    ports:

```

```

as="h1"
fontFamily="e-Ukraine-Medium"
fontSize={['24px', null, null, null, '32px']}
textAlign="start"
>
  Створити акаунт
</Heading>
<Text fontSize="12px" fontWeight="bold"
color="#191D24" mt="20px">
  Я
</Text>
<TypeSelectionButtons
  selectedType={selectedType}
  handleTypeChange={(type) => {
    setSelectedType(type)
    setTypeError(null)
  }}
/>
{typeError && (
  <Flex align="center" mt="4px">
    <Box
      w="20px"
      h="20px"
      bg="red.500"
      borderRadius="50%"
      display="flex"
      alignItems="center"
      justifyContent="center"
      mr="8px"
      p="10px"
    >
      <Text color="white" fontSize="12px"
fontWeight="bold">
        i
      </Text>
    </Box>
    <Text fontSize="12px" color="red.500" mt="4px">
      {typeError}
    </Text>
  </Flex>
)}
<InputField
  label="Ім'я"
  type="text"
  value={name}
  error={nameError}
  onChange={(e) => setName(e.target.value)}
  placeholder="Як нам до вас звертатись"
/>
<InputField
  label="Електронна пошта"
  type="email"
  value={email}
  error={emailError}
  onChange={(e) => setEmail(e.target.value)}
/>
<InputField
  label="Пароль"
  type="password"
  value={password}
  error={passwordError}
  onChange={(e) => setPassword(e.target.value)}
/>

```

```

- 6379:6379/tcp
volumes:
- ./tmp/redis:/data

sidekiq:
<<: *backend-app
command: bundle exec sidekiq
depends_on:
- redis
volumes:
- ./rails:rw
- rails_storage:/rails/storage

backend:
<<: *backend-app
entrypoint: ./dev-entrypoint.sh
command: bundle exec rails server -b
0.0.0.0
ports:
- "3000:3000/tcp"
depends_on:
- postgres
- redis

volumes:
rails_storage:

```

```

<CreateAccountButton isLoading={isCreating}
onClick={onSubmit} />
{ /*<OrDivider /> */ }
<TelegramAuthButton
text="зарегіструватися через telegram"
/>
</Box>
<Box pt={'12px'} fontSize={'12px'} textAlign="center">
Реєструючись, ви погоджуєтесь з нашими{' '}
<Link
href={'/terms_of_use.html'}
color="#25ACC1"
target={'_blank'}
>
умовами використання
</Link>
.
</Box>
<AccountSectionRegistration />
</Flex>
</Flex>
<LoginImageSection />
</Flex>
</Flex>
)
}

```

```
export default RegPage
```

```

import type { JSX } from 'react'
import { List, ListItem } from '@chakra-ui/react'
import SpecialtyCard from './SpecialtyCard'

```

```

type Specialty = {
  code: string
  name: string
  description?: string
  slug: string
}

```

```

interface SpecialitiesCardsProps {
  specialities: Specialty[]
}

```

```

const SpecialitiesCards = ({
  specialities,
}: SpecialitiesCardsProps): JSX.Element => {
  const listItems = specialities.map((specialty) => (
    <ListItem key={specialty.code}>
      <SpecialtyCard specialty={specialty} />
    </ListItem>
  ))
}

```

```

return <List flex="1" spacing={{ base: '12px', md: '14px', lg:
'16px' }}>{listItems}</List>
}

```

```
export default SpecialitiesCards
```