**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ЧЕРНІВЕЦЬКИЙ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТУ ІМЕНІ ЮРІЯ  
ФЕДЬКОВИЧА»**

**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ «ФАХОВИЙ  
КОЛЕДЖ ЧЕРНІВЕЦЬКОГО НАЦІОНАЛЬНОГО  
УНІВЕРСИТЕТУ ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА»**

**Природниче відділення Циклова комісія**

**комп’ютерних наук**

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр» зі спеціальності 1*22 Комп’ютерні науки*

підготовки за освітньо-професійною програмою «*Комп’ютерні науки*»

на тему:

«**Платформа для освіти «MindForge»»**

**Виконав (ла):**

студент (ка) 4-го курсу

*Кукуляк Дмитро Валерійович*

(прізвище, ім’я та по батькові) (підпис)

**Керівник:**

к. ф.-м. н., Ковдриш В.В.

(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

**Рецензент:**

к. техн. н., доц. Петренко П.В.

(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

***До захисту допущено:***

***на засідання циклової комісії***

***проткол №***  ***від „*** ***”*** ***2024 р.***

***Голова циклової комісії***  ***Василь КОРОПЕЦЬКИЙ***

Чернівці, 2025

**Анотація**

Дипломна робота досліджує можливості онлайн освіти як спосіб покращення успішності учнів та студентів у позашкільний час. Задачі включають аналіз ринку, визначення конкурентів, формування технічного стеку.

Виявлено прогалину на ринку освіти та обґрунтовано необхідність створення простішої платформи для досягнення учнями та студентами поставлених цілей. На основі вищезгаданого сформоване технічне завдання та цілі проєкту.

Створено мінімально життєздатний продукт з перспективами подальшого вдосконалення та виходу на ринок. Було виявлено задовільні показники по всім параметрам і очікується продовження розробки продукту.

Сторінок – 51, рисунків – 9, джерел літератури – 19, додатків – 1, таблиць - 2

**Ключові слова**: освіта, онлайн освіта, EdTech репетиторство, навчання, платформа, веб-застосунок, клієнтська частина, серверна частина.

\_\_\_\_\_\_\_\_\_\_ Кукуляк Д.В.

(підпис)

**Abstract**

The thesis explores the potential of online education as a means to improve the academic performance of pupils and students during extracurricular time. The objectives include market analysis, identification of competitors, and the formation of the technical stack. A gap in the education market has been identified, substantiating the need to create a simpler platform to help pupils and students achieve their goals. Based on the above, the technical requirements and project objectives have been formulated.

A minimum viable product has been developed with prospects for further improvement and market entry. Satisfactory results have been achieved across all parameters, and continued product development is anticipated.

Pages – 51, figures – 9, references – 19, appendices – 1, tables – 2

**Keywords**: education, online education, EdTech, tutoring, learning, platform, web application, client-side, server-side.

\_\_\_\_\_\_\_\_\_\_ Kukuliak D.V.

(sign here)

**ЗМІСТ**

[ВСТУП 6](#_Toc200812020)

[РОЗДІЛ І. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8](#_Toc200812021)

[1.1. Аналіз вимог студентів та менторів 8](#_Toc200812022)

[1.2. Ідея застосунку 9](#_Toc200812023)

[1.3. SWOT-аналіз ринку 10](#_Toc200812024)

[1.4. Технічне завдання та функціональні вимоги 12](#_Toc200812025)

[1.4.1.Основні ролі користувачів 12](#_Toc200812026)

[1.4.2. Основні функціональні вимоги 13](#_Toc200812027)

[1.4.3.Нефункціональні вимоги 14](#_Toc200812028)

[1.4.4. Додаткові (опціональні) функції 14](#_Toc200812029)

[1.5. Висновки до розділу 14](#_Toc200812030)

[РОЗДІЛ ІІ: ПРОЄКТУВАННЯ СИСТЕМИ 16](#_Toc200812031)

[2.1. Архітектура проєкту: клієнт + сервер 16](#_Toc200812032)

[2.2. Стек проєкту 17](#_Toc200812033)

[2.2.1 Стек клієнтської частини 18](#_Toc200812034)

[2.2.2. Стек серверної частини 20](#_Toc200812035)

[2.3. Проєктування бази даних 21](#_Toc200812036)

[2.4 Проектування API та взаємодії між сервісами 24](#_Toc200812037)

[2.4.1. Шляхи для API 25](#_Toc200812038)

[2.4.2. User flow 25](#_Toc200812039)

[2.4.3. Взаємодія фронтенду і бекенду 26](#_Toc200812040)

[2.4.4. Real-time функціонал 26](#_Toc200812041)

[2.4.5. Підсумки 26](#_Toc200812042)

[2.5 Проектування інтерфейсу користувача (UI/UX) 28](#_Toc200812043)

[2.6 Дизайн інтерфейсу: процес, інструменти, макети 29](#_Toc200812044)

[РОЗДІЛ ІІІ: РЕАЛІЗАЦІЯ СИСТЕМИ 31](#_Toc200812045)

[3.1 Розробка бекенду (Express.js, TypeScript, Prisma, Supabase) 31](#_Toc200812046)

[3.2 Розробка фронтенду (Next.js, Tailwind CSS, Redux, Shadcn/UI) 33](#_Toc200812047)

[3.3 Інтеграція фронтенду та бекенду, real-time функціонал 34](#_Toc200812048)

[3.4 Тестування системи (Frontend + Backend) 35](#_Toc200812049)

[3.5 Деплой, контейнеризація 36](#_Toc200812050)

[ВИСНОВОК 38](#_Toc200812051)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40](#_Toc200812052)

Нижче наведено перелік усіх умовних скорочень, специфічних термінів, які будуть зустрічатися в роботі:

**Скорочення:**

* **UI (User Interface)** – інтерфейс користувача, оцінює наскільки вебсайт візуально привабливий та інтуїтивний
* **UX (User Experience)** – досвід користувача при роботі з сайтом, позначає наскільки приємно взаємодіяти з сайтом, і як його функціонал себе показує
* **API (Application Programming Interface)** – інтерфейс програмування, найчастіше зв’язок між серверною та клієнтською частиною у вигляді програми
* **ORM (Object-Relational Mapping)** – об’єктно-реляційна передача, спосіб працювати з базами даних в об’єктно-орієнтованому стилі, навіть дял мов з динамічною типізацією.
* **SSR (Server Side Rendering) –** рендер(відмалювання) на стороні сервера усього HTML перед тим як відправити його на клієнт
* **SPA (Single Page Application)** – застосунок на одну сторінку, новий спосіб створення клієнтської частини застосунку шляхом рендеру лише елементів, які змінилися, без повного перезавантаження сторінки
* **CI/CD (Continuous Integration/Continuous Deployment) –** постійна інтеграція та розгортка, шлях автоматизації лінтування, тестування, розгротки, оновлення, та інших дій.
* **CRUD (Create, Read, Update, Delete)** – створення, читання, оновлення та видалення – чотири базові операції для взаємодії з БД, на яких будується серверна частина
* **JWT (JSON Web Token)** – токен, який зберігає інформацію про авторизацію користувача та передається між клієнтською та серверною частиною. Стандарт безпеки.
* **P2P (Peer to Peer)** – користувач користувачу, система коли користувач одночасно й надає послуга, і купує їх.

**Терміни:**

* **Студент** - учень, який бажає отримати знання. Шукає ментора для навчання
* **Ментор** — репетитор, фахівець, користувач який надає послуги студентам
* **Сесія** — окрема навчальна зустріч між учнем і ментором, яка може проходити онлайн або офлайн.
* **Замовлення** — заявка, створена користувачем на проведення навчальної сесії або заняття з ментором
* **Чат** — функціонал для обміну повідомленнями між користувачами платформи в реальному часі.
* **Рейтинг**— система оцінювання якості роботи менторів на основі відгуків учнів.
* **SWOT (Strenghts, Weaknesses, Opportunities and Threaths)** – аналіз технічний проєктів, який базується на 4 критеріях: сили, слабкості, можливості та загрози

# ВСТУП

Ми живемо в неймовірний час, коли інформація є легкодоступною, дізнатися щось нове не є проблемою. Насправді, коли можливо отримати відповідь на будь-яке питання за лічені хвилини, деякі люди стверджують, що немає потреби щось вчити самостійно. І в цьому є правда, але різниця між швидкою інформацією, яку можна отримати миттєво, та справжнім майстерством колосальна. Тому, незважаючи на сучасні можливості, вчити інформацію ретельно, та запам’ятовувати її для подальшого використання є необхідною навичкою. Якщо ж враховувати можливості ШІ, які з’явилися за останні 5 років, то питання загострюється.

Якщо ж зважати на більш комплексну інформацію – не простий факт або дату, а, наприклад, методи розв’язування систем лінійних рівнянь з вищої математики – то вивчення та засвоєння такої інформації стане проблематичнішим, незважаючи на всі можливості штучного інтелекту. Але важливо зазначити, що це лише в даний момент часу – технології ШІ рухаються з неймовірною швидкістю. Тож, можливе й таке, що через 5-10 років ШІ стане настільки потужним, що дійсно буде здатний справлятися з комплексною інформацією.

Для чого людині вчитися, якщо будь-яку інформацію можливо отримати миттєво за пошуком в інтернеті, де ШІ швидко дасть відповідь? Тому що людині потрібна освіта, незважаючи ні на що. Постійне покладання на ШІ задля допомоги затуплює здібності людини. Якщо не використовувати м’язи, вони атрофуються – зменшуються. Аналогічно поводить себе мозок – не використовуй його, і його здібності стрімко падають. Це демонструє цьогорічне дослідження, [1] де висновки вказують на значну негативну кореляцію між використанням інструментів ШІ та можливість критично думати. Автор застерігає, що, незважаючи на лічені переваги та можливість зберегти час, використання ШІ зменшує здатність користувачів думати критично, будувати логічні ланцюги та досліджувати глибокі думки власної підсвідомості [1].

Отже, можна зробити висновок, що задля сприятливого функціонування людського мозку та зберігання сили мислення, не варто надто сильно покладатися на ШІ. Саме тому освіта є важливою частиною людського життя, незважаючи на здатність миттєво отримувати інформацію. Зважаючи на вищеподану інформацію, можна сміливо сказати, що справжня освіта зараз є ще більш важливою, ніж колись. Грунтуючись на цьому, викладання є, і буде, актуальним, тому розвиток цієї індустрії перспективний.

Ще більш критичним воно стає в Україні. Починаючи з 2020 року освітяни всіх ступенів зіштовхнулися з ліченими проблемами. COVID-19 спричинив карантин, учні та студенти перейшли на дистанційне навчання та зіштовхнулися з незнайомими умовами навчання, через що якість освіти впала. Як тільки освітній процес почав повертатися до норми, у лютому 2022 рф напала на Україну, і це мало ще гірші наслідки для освіти. Здобувачі освіти мали тікати від війни, ховатися в бомбосховищах, та виживати, тому освіта переходила на задній план. Зважаючи на це, останні 5 років в Україні якість освіти страждає. Попит породжує пропозицію, тому ринки позашкільної освіти зараз популярні, як ніколи.

В основному діти та студенти займаються з репетиторами, або ж з вчителями напряму. Виникає проблема знаходження такого викладача, який покращить знання здобувача освіти з конкретного предмету. На жаль, ринок нестабільний: онлайн школи хоч і можуть забезпечувати швидкий пошук, але, зазвичай, за великими цінами, а знайти репетитора локально ще складніше, і доводиться йти очно. Одним словом, не найкращі варіанти. Так і виникла ідея MindForge – онлайн застосунку для освіти, який з’єднує учня з ментором напряму, за регульованою розцінкою, та робить це швидко.

# РОЗДІЛ І. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## Аналіз вимог студентів та менторів

Чим далі ми рухаємось, тим більш доступною стає інформація. Спочатку на заміну бібліотекам та книжкам прийшов інтернет, та вікі сторінки, збільшивши кількість інформації на кілька порядків. Потім почали з’являтися ресурси спрямовані на навчання: вебінари, курси, лекції, і раптом додаткова освіта в інтернеті стала легкодоступною. Але на фоні цього зростає попит на індивідуальне навчання, якість якого набагато перевищує решту джерел інформації.

Проєкт MindForge спрямований на таке індивідуальне навчання, але з обох боків. Для студента це можливість знайти ментора на будь-який смак: починаючи від предмету, і закінчуючи кваліфікацією та досвідом роботи й ціною. Для ментора це можливість швидко знаходити й набирати учнів на навчання, заробляючи собі на життя. Модель платформи передбачає надавання лише самого сервісу, а користувачі вже надалі будуть взаємодіяти між собою. Така модель бізнесу називається Peer to Peer Marketplace.

Ще одним складником, на якому варто сфокусуватися, є верифікація менторів. Реєстрація повинна відбуватися просто, без тривалих раундів співбесід та перевірок, де тебе можуть або прийняти, або відмовити. Тому варто зазначити, що вона проходитиме як і реєстрація на решті вебсайтів, без додаткових умов та дій. Ментор повинен самостійно забезпечити собі хороший профіль, який будуть помічати. Тому варто забезпечити можливість додавання секцій для освіти та досвіду роботи репетитора, і приєднати рейтингову систему, щоби наочно показати людині здібності ментора.

Варто також розглянути вимоги для студентів та менторів. Студенту необхідна сучасна та надійна, візуально приваблива платформа, що зможе забезпечити швидкий пошук серед зареєстрованих менторів на веб-сайті. Пошук має включати фільтрування, сортування, вибірку. Інтерфейс має бути простим та інтуїтивним, вимагати сучасним практикам та стандартам індустрії. Щодо ментора повинна бути забезпечена швидка й проста реєстрація, можливість детально заповнити свій профіль та забезпечити конкурентоспроможність ментора на фоні інших.

## Ідея застосунку

**Mind Forge** – це освітня платформа нового покоління, яка об’єднує студентів і викладачів (або менторів) без посередників, надаючи широкий спектр послуг: від репетиторства з різних шкільних предметів до допомоги з домашніми завданнями, курсовими роботами, дипломними проєктами. Ця модель не передбачає посередників, що спрощує розробку, запуск, та все інше. На відміну від інших платформ, від репетиторів не вимагатимуть проходити перевірки й готуватися, ідея схожа з фріланс платформами (Upwork, Fiverr), де твоя діяльність, тобто виконані замовлення, і є твоїм підтвердженням. Це забезпечить низький вхідний поріг для викладачів, що сприятиме збільшенню їхній кількості. Іншою перевагою буде маленька комісія, оскільки не потрібно підтримувати величезну команду менеджерів, які вимагають фінансування – ще більш привабливі умови для викладання.

Цільова аудиторія поділятиметься на дві групи: викладач, або ж репетитор, ментор, який може надавати послуги, і студент, який хоче ці послуги придбати. Від викладача не вимагається сувора перевірка знань перед тим як почати заробляти – варто лиш довести свої навички. Обмежень по віковим, статевим, расовим, або інших категоріях, нема. Що стосується студента, то тут обмежень також немає. Це не обов’язково має бути учень чи студент, може бути будь-яка людина, якій потрібна допомога.

Застосунок фокусується на простоті роботи та легкості використання. Планується, щоб він став місцем пошуку допомоги на довгострокову перспективу, або ж разової допомоги, і, головне, щоб це сталося швидко.

Основний функціонал системи для студента включатиме: реєстрацію, створення профілю користувача. Перегляд каталогу викладачів, з яких студент може собі знайти викладача, або ж створити замовлення і чекати на відгуки. Можливість пропонувати замовлення викладачу. Перегляд поточних замовлень студента, скасування, знаходження нового викладача, можливість лишити відгук у разі виконання, або ж розірвання замовлення. Що ж стосується викладача: редагування профілю, додавання освіти та місця попередньої роботи, або одразу кілька за них. Перегляд поточних замовлень, на які можна відгукнутися. Розрив замовлення. Також передбачено систему чатів, які будуть створюватися автоматично при відгуках на замовлення або пропозицію замовлення

## SWOT-аналіз ринку

Перед тим як переходити до опису функціональних вимог та технічих деталей, варто проаналізувати існуючі рішення. Для цього ми скористаємося моделлю SWOT, і почнімо з конкурентів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Платформа | Сильні сторони | Слабкі сторони | Можливості | Загрози |
| Preply | Глобальна платформа, велика база користувачів, система верифікації | Висока комісія, складна процедура реєстрації для викладачів | Розширення предметів, інтеграція AI | Конкуренція з новими платформами, перевантаженість популярних ніш |
| iTalki | Спеціалізація на мовах, активна спільнота | Обмежена тематика (в основному мови), не дуже гнучка система цін | Додати інші предмети, практичні навички | Нестача викладачів нетипових мов або напрямків |
| Wyzant | Локалізована підтримка, зручна система пошуку | Доступна лише в окремих країнах, складна реєстрація | Розширення на міжнародний ринок | Обмеження регіонального доступу |

**Табл.** 1.3.1. SWOT аналіз конкурентів

Отже, підсумувавши результати аналізу конкурентів, можна зібрати всі необхідні дані та підкреслити, що важливо впровадити в наш проєкт:

* На ринку є місце для нової платформи з простішою реєстрацією, нижчою комісією, гнучкою тематикою та сучасним підходом до UX.
* Важливо уникати складних процедур для менторів, забезпечити прозорість і простоту для обох сторін.
* Розширення предметів і інтеграція сучасних технологій (AI, real-time, мобільність) — це шлях до конкурентної переваги.
* Локалізація під український ринок і адаптація під його реалії — ще одна сильна сторона для старту.

|  |  |
| --- | --- |
| Сильні сторони | Слабкі сторони |
| Простота та низький поріг входу для викладачів  Широкий спектр послуг: шкільні предмети, допомога з ДЗ, КР, дипломами, категорія «інше»  Автоматизована взаємодія без посередників  Гнучкість ролей (викладач/учень) в одному профілі  Низькі або відсутні комісії | Відсутність верифікації знань  Початкова нестача користувачів і відгуків  Можливі випадки шахрайства без верифікації  Потреба у якісній модерації при широкій тематиці |
| Можливості | Загрози |
| Заповнення ніш, які ігноруються конкурентами  Локалізація для різних ринків  Створення швидкої допомоги – замовлення без очікування профілю | Конкуренція з великими гравцями Недовіра до неперевірених викладачів  Потенційні юридичні ризики (написання дипломів/робіт за інших)  Потреба у масштабованій модерації |

**Табл. 1.3.2** SWOT аналіз MindForge

* 1. **Технічне завдання та функціональні вимоги**

Метою цієї роботи є розробка сучасної платформи для позашкільної освіти, яка буде слугувати ефективним зв’язковим між студентом та учнем. Моделлю платформи слугуватиме P2P, де користувачі одночасно і надають послуги, й купують їх. Така модель забезпечує мінімальне втручання від сторони розробників, що зменшує витрати на проєкт як грошових, так і людських ресурсів. Платформа повинна забезпечити швидкий та інтуїтивно зрозумілий пошук для обох сторін, дозволяючи максимально ефективну взаємодію між ними. Основна ідея – максимально спростити процес реєстрації та пошуку, дозволивши найкращий досвід. Якщо коротко – швидко, просто, ефективно.

### 1.4.1.Основні ролі користувачів

1. **Учень (Student):**

* Реєстрація та створення профілю.
* Пошук менторів за предметом, ціною, рейтингом, відгуками.
* Створення замовлення на навчання (разове або регулярне).
* Перегляд профілю ментора, відгуків, рейтингу
* Комунікація з ментором через чат.
* Оцінювання та залишення відгуків після завершення сесії.
* Перегляд історії замовлень та сесій.

1. **Ментор (Tutor):**

* Реєстрація та створення розширеного профілю (освіта, досвід, предмети, ціни).
* Прийом замовлень від учнів, підтвердження або відхилення заявок.
* Ведення календаря занять, планування сесій.
* Комунікація з учнями через чат.
* Перегляд та управління відгуками, рейтингом.
* Можливість редагування профілю, додавання нових предметів.

1. **Адміністратор (опціонально):**

* Модерація контенту, перевірка профілів, вирішення спірних ситуацій.
* Управління категоріями, предметами, тарифами платформи.

### 1.4.2. Основні функціональні вимоги

* **Реєстрація** та автентифікація: Користувачі повинні мати можливість реєструватися через email або соціальні мережі, проходити верифікацію, відновлювати пароль.
* **Профілі користувачів:** Кожен користувач має особистий профіль із фото, контактною інформацією, описом, для менторів — додатково освіта, досвід, предмети, ціни.
* **Пошук і фільтрація:** Зручний пошук менторів за предметом, ціною, рейтингом, відгуками, досвідом, типом занять (онлайн/офлайн).
* **Замовлення та організація сесій:** Учень може створити замовлення на сесію, обрати тип (разова/регулярна), вказати кількість занять, дедлайни, залишити коментарі.
* **Календар та планування:** Інтегрований календар для планування та відстеження майбутніх і минулих сесій.
* **Чат та сповіщення:** Вбудований real-time чат для комунікації між учнем і ментором, система сповіщень про нові повідомлення, зміни статусу замовлення, нагадування про сесії.
* **Система відгуків і рейтингу:** Після завершення сесії учень може залишити відгук і оцінку ментору. Рейтинг формується на основі середнього балу та кількості відгуків.
* **Фінансові операції (опціонально):** Можливість здійснювати оплату через платформу, автоматичне нарахування комісії, історія транзакцій.
* **Безпека та захист даних:** Захист персональних даних користувачів, шифрування паролів, захист від несанкціонованого доступу.
* **Адаптивний інтерфейс:** Платформа повинна коректно працювати на різних пристроях (ПК, планшет, смартфон).

### 1.4.3.Нефункціональні вимоги

* **Масштабованість:** Система повинна підтримувати зростання кількості користувачів і навантаження без втрати продуктивності.
* **Продуктивність:** Всі основні операції (пошук, створення замовлення, чат) мають виконуватися швидко, із мінімальною затримкою.
* **Зручність використання (Usability):** Інтерфейс має бути інтуїтивно зрозумілим, з мінімальною кількістю кроків для виконання основних дій.
* **Можливість інтеграції:** Передбачити можливість інтеграції з платіжними системами, календарями, відеоплатформами (Zoom, Google Meet).

### 1.4.4. Додаткові (опціональні) функції

* Система рекомендацій менторів на основі попередніх замовлень і відгуків.
* Групові заняття.
* Мобільний застосунок.
  1. **Висновки до розділу**

Перший розділ цієї роботи слугував точкою входу в проєкт MindForge. Був проведений детальний аналіз предметної області , визначено основні проблеми, з якими стикаються студенти, наведено аргументи на користі створення платформи для організації взаємодії між учнями та менторами. Розглянуто поточний стан ринку, виявлено його прогалини, які ще не заповнені, проведено аналіз конкурентів, їх сильні та слабкі сторони на міжнародному та українському ринках.

На основі аналізу було сформульовано технічне завдання та визначено функціональні вимоги для мінімально життєздатного продукту (MVP) платформи MindForge. Особливий акцент було зроблено на простоті платформи та прозорості знань, доступності та гнучкості для обох сторін — як для студентів, так і для менторів(викладачів). Планується додати подальші функції відповідно до вимог ринку та користувачів.

# РОЗДІЛ ІІ: ПРОЄКТУВАННЯ СИСТЕМИ

## 2.1. Архітектура проєкту: клієнт + сервер

Архітектура платформи Mind Forge розділена на клієнтську (Frontend) та серверну (Backend) частини, що дозволяє досягти максимальної масштабованості, гнучкості та простоти підтримки системи. Клієнтська частина написана на основі фреймворку Next.js, який поєднує SSR, SSG та SPA. Це забезпечує швидке завантаження сторінок, SEO-оптимізацію та зручну маршрутизацію. Нижче додається структура проєкту:

* **src/app/** — маршрути для різних ролей (учень, ментор), сторінки реєстрації, профілів, пошуку;
* **src/components/** — багаторазові UI-компоненти (картки, кнопки, форми, модальні вікна);
* **src/lib/** — утиліти, кастомні хуки, API-клієнти для взаємодії із серверною частиною;
* **src/styles/** — глобальні стилі, теми;
* **src/types/** — типи TypeScript для забезпечення типізації.

Серверна частина написана на Express.js з TypeScript, що вкупі уможливлює створення масштабованих REST API з чіткою структурою та типізацією. Додаткова типобезпека досягнута за рахунок Prisma ORM, що дозволяє працювати з базою даних об’єктно-орієнтованою шляхом.

Серверна частина відповідає за

* обробку запитів від фронтенду;
* автентифікацію та авторизацію користувачів;
* роботу з базою даних через Prisma ORM;
* бізнес-логіку (створення/редагування профілів, пошук, обробка заявок, відгуків).

База даних — PostgreSQL, розгорнута через Supabase, має багато функціоналу: безкоштовна розгортка, генерація REST API, вбудовані біліотеки для авторизації, ролі, і багато іншого. Взаємодія між клієнтом і сервером здійснюється через REST API. Такий підхід дозволяє легко масштабувати систему.

Переваги обраної архітектури:

* Масштабованість: можна окремо масштабувати клієнтську, і серверну частину
* Гнучкість: завдяки модульній архітектурі легко додавати нові функції, інтегрувати сторонні сервіс
* Безпека: чітке розмежування доступу, централізована автентифікація.
* Можливість подальшого розвитку: додавання real-time, мобільних додатків, мікросервісів

## 2.2. Стек проєкту

При створенні проєкту вибір технологій є вирішальним. Перед тим, як йти напряму й використовувати те, до чого всі звикнули, або що є популярним, варто проаналізувати можливі технології та чітко зрозуміти, які інструменти найкраще підійдуть для конкретного завдання. Наприклад, якщо будується система для управління клієнтами, де дизайн не має вирішального значення, то комбінація біблітек компонентів та стилів скоротить витрати ресурсів на візуал. Так само якщо застосунок це проста лендінг-сторінка, немає сенсу обирати масивні фреймворки для цього та розробляти комплексні рішення.

На основі цього базувався й вибір стеку для поточного проєкту. Оскільки платформа має бути комплексною, великою, включати велику кількість операцій, стек має бути сучасним та підібраним спеціально під нагальні потреби застосунку.

### 2.2.1 Стек клієнтської частини

Можна з впевненістю сказати, що за роки свого існування те шлях, яким створюється фронт-енд частина змінився кардинально. У ранні дні свого існування кожна сторінка відповідала одному гіпертекстовому документу, стилів ще не було, і веб-сайти виглядали зовсім недружньо – відкривши зараз такий, можна це зрозуміти миттєво.

Як і решта технологій, інструменти створення клієнтською частини починалися з чогось дуже базового: жодних стилів, інтерактивності, динамічності, лише чиста статика. Лиш через кілька років з’явилися стилі, показавши що веб-сайти можуть бути не тільки холодними й ефективними, але й живими. Навіть JavaScript - мова клієнтської частини – була створена лише через 5 років після виходу HTML1. До того, як ці інструменти стали доступними, веб- сайти були тільки наборами незмінного тексту.

З появою вищезгаданих стандартів – CSS та JS, веб- розробка почала кардинально змінюватись. Завдяки CSS сайти стали красивими, а JavaScript – живими. Доступ до інтернету почав з’являтися в більшості, через це більше попиту та популярності – настала ера найбільш стрімкого розвитку для клієнтської частини

Згодом цей підхід теж починали змінювати. Проблемою стало те, що кожній сторінці на сайті відповідав окремий документ, і чим більше сторінок, тим більше документів. Також до кожної сторінки імпортувався сайт зі стилями, що могло швидко збільшити розміри проєкту. Рішенням цього стала верста за допомогою бібліотек та фреймворків, які кардинально змінили стан справ. Найбільш зручною річчю, яку вони ввели, стали компоненти – функції, які можна повторно використовувати. Найкращим прикладом цього є кнопка: на будь-якому сайті вони будуть повторюватися багато разів, і щоразу копіювати HTML та CSS код для вставлення є нудною роботою. Компоненти дозволяють оголошувати свої власні теги, які неймовірно зручно використовувати. Також ще кардинально змінився підхід до написання коду. HTML – це не мова програмування, а розмітки гіпертекстового документу, а сторінка – просто сайт, який ми відкриваємо, немає необхідності компіляції або інтерпретації коду. Але нові фреймворки дозволяють писати JS код з вставками HTML, що кардинально відрізняється від стандартного підходу. Всі компоненти представляють собою функції, які можна експортувати/імпортувати, і повторно використовувати. Також, варто зазначити, що такі файли написані у форматі .js, і інтерпретуються у статичні файли – HTML, CSS та JS, які вже згодом показуються користувачу. Цей підхід є найбільш сучасним, і більшість нових сайтів, які створюються, використовуватимуть якийсь фреймворк або бібліотеку.

Для поточного проєкту було вирішено обрати Nextjs, який є фреймворком над React, який, у свою чергу, і бібліотекою для створення SPA. Нижче наведено інформацію про них.

React — це популярна JavaScript-бібліотека для створення користувацьких інтерфейсів, розроблена Facebook. Ця бібліотека революціонізувала напичання клієнтських частин, увівший новий станарт - компонентний підхід. Інтерфейс розбивається на невеликі частини – компоненти, які можна повторно використовувати. React дозволяє писати HTMLу вигляді JS коду з використанням JSX, що робить розробку швидкою та зрозумілою. React ідеально підходить для створення сучасних, динамічних веб-додатків, але не вирішує всі задачі “з коробки”.

Next.js — це фреймворк для React, який значно покращує його можливості. Віннадає можливості SSR, SSG, а також має зручну маршрутизацію на основі файлової структури. Next.js вирішує більшість типових задач сучасної веб-розробки, поєднуючи зручність React із потужними можливостями для створення production-ready застосунків.

Чому саме ці технолїі для MindForge? React Next.js забезпечують масштабованість, продуктивність, сучасний вигляд і простоту розробки, що ідеально підходить для створення платформи на кшталт MindForge.

### 2.2.2. Стек серверної частини

Перші сайти були статичними: кожен документ був окремим HTML-файлом, і взаємодія з користувачем була неможлива, окрім читання контенту. З появою Всесвітньої павутини люди почали усвідомлювати, що недостатньо просто представляти інформацію споживачам; натомість додатки повинні бути динамічними, отримувати вхідні дані від користувачів, обробляти цю інформацію та надавати зворотний зв'язок у реальному часі. Так народилися серверні мови програмування: спочатку CGI-скрипти на Perl, потім PHP, ASP, Java Servlets. У 2000-х роках класичні фреймворки, такі як Laravel, Django, Ruby on Rails, почали набувати популярності. Це значно полегшило розробку складних веб-додатків. З появою Node.js JavaScript почав "виходити" з браузера і використовуватися в серверній розробці. Завдяки цьому фронтенд і бекенд могли бути написані однією мовою, усуваючи потенційний бар'єр у розробці реальних веб-додатків. На основі Node.js з'явився Express.js — легкий, гнучкий фреймворк REST API, який став стандартом для сучасних проектів на JavaScript. З часом зросла потреба в безпеці типів і зручних засобах роботи з даними. З появою таких технологій, як TypeScript, і нових ORM, як Prisma, стало можливим ефективно використовувати реляційні бази даних, автоматизувати міграції та зберігати цілісність даних. Останніми роками хмарні платформи, такі як Supabase, стали все більш популярними, пропонуючи готову інфраструктуру для зберігання даних, автентифікації та можливостей реального часу, щоб розробники могли зосередитися на логіці додатків, а не на обслуговуванні серверів.

Express.js, який є легким і адаптивним фреймворком для Node.js — створення RESTful API з Node.JS — це надзвичайно простий спосіб швидко створити і запустити REST API. Він має легку для розуміння синтаксис, активну спільноту, багато бібліотек і чудово підходить для створення масштабованих серверних додатків.

TypeScript — це надмножина JavaScript зі статичною типізацією. Використовуючи TypeScript, покращується якість коду, запобігається безліч помилок під час розробки, легше підтримувати і розвивати проект, а автозавершення коду покращується в редакторі коду.

Prisma — це найшвидший спосіб створити що завгодно в базах даних: від прототипів до готових до виробництва веб-додатків. Варто лиш повідомити Prisma про структуру даних, міграції і Prisma все зробить.

Supabase — це хмарна база даних, яка абстрагує і надає базу даних PostgreSQL, автентифікацію користувачів з RBAC, зберігання файлів і підписки в реальному часі для додатків. Supabase може допомогти швидко запустити надійну базу даних без налаштування сервера і полегшує масштабування в майбутньому. Воно також дозволяє купувати їх рішення або самостійно розміщувати сервіс.

Вибір стеку Express.js, TypeScript, Prisma та Supabase для бекенд-частини Mind Forge зумовлений прагненням поєднати швидкість розробки, надійність, масштабованість і сучасний підхід до роботи з даними. Така комбінація дозволяє створити безпечну, гнучку та легко підтримувану платформу, яка відповідає сучасним вимогам до онлайн-сервісів і забезпечує просту інтеграцію з іншими технологіями, що особливо важливо для динамічного розвитку освітнього проєкту.

## 2.3. Проєктування бази даних

Проектування бази даних — це, без перебільшення, один із найважливіших етапів у розробці будь-якої сучасної платформи. Саме від структури БД залежить, наскільки зручно буде працювати з даними, як швидко виконуватимуться запити, і чи можна буде масштабувати систему у майбутньому. Для MindForge було обрано комбінацію Prisma ORM та хмарної платформи Supabase, що дозволяє поєднати зручність розробки, гнучкість, безпеку та сучасний підхід до зберігання даних.

База даних будується на PostgreSQL, а для взаємодії з нею використовується Prisma — сучасна ORM, яка дозволяє описувати структуру даних декларативно, у вигляді schema.prisma. Це не просто зручно, а й реально економить час: достатньо змінити схему — і Prisma сама згенерує всі необхідні міграції та клієнтський код для роботи з БД. Supabase, у свою чергу, забезпечує хмарну інфраструктуру, авторизацію, зберігання файлів і real-time можливості.

У MindForge реалізовано чітку структуру сутностей, які відображають реальні процеси на платформі:

* + **User** (користувач): основна таблиця, яка містить інформацію про всіх користувачів (студентів і менторів), їхні ролі, контактні дані, аватар, біо, тощо. Кожен користувач може бути як студентом, так і ментором (роль визначається полем role).
  + **TutorEducation**, TutorExperience: окремі таблиці для зберігання освіти та досвіду роботи менторів. Це дозволяє гнучко формувати профіль викладача, додавати кілька місць роботи чи навчання.
  + **Subject, Category:** таблиці для предметів і категорій занять. Категорії можуть бути разовими або регулярними (isRecurring), що дозволяє реалізувати як разові консультації, так і довготривалі курси.
  + **TutorSubject**: зв'язує ментора з предметом і категорією, а також містить ціну за заняття. Це дозволяє одному ментору викладати кілька предметів у різних категоріях.
  + **Order**: замовлення на заняття. Містить інформацію про студента, ментора (опціонально), предмет, категорію, статус, ціну, кількість сесій, тощо. Саме тут відбувається основна взаємодія між учнем і ментором.
  + **Session**: окремі сесії в рамках замовлення. Дозволяє планувати, відстежувати і зберігати історію занять.
  + **Review**: відгуки студентів про менторів. Кожен відгук містить рейтинг, коментар, дату, і зв'язаний із замовленням, студентом і ментором.
  + **OrderInvite**: система запрошень для взаємодії між студентами і менторами (наприклад, коли студент хоче запросити конкретного ментора на замовлення).
  + **SavedOrder, SavedTutor**: дозволяють зберігати улюблені замовлення та менторів для швидкого доступу.

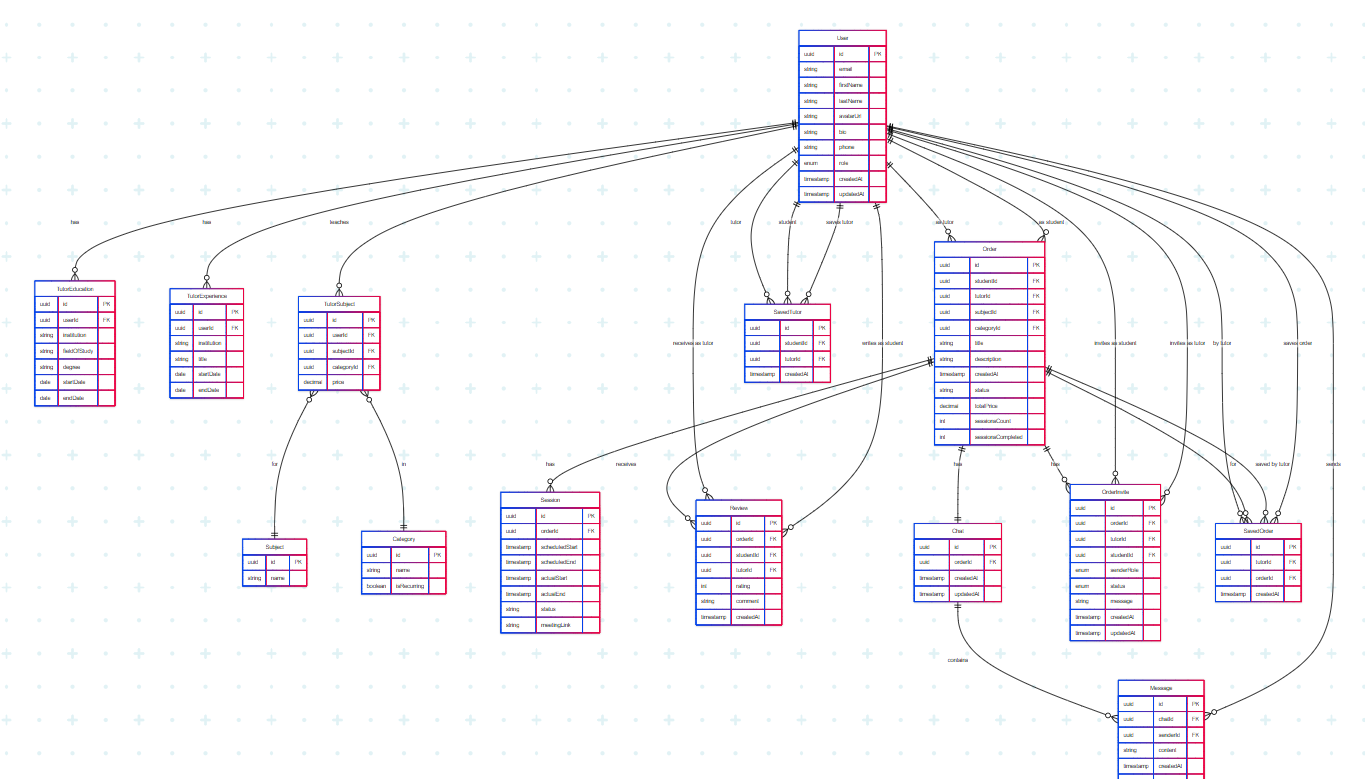


Рис 2.3.1. mmd діаграма БД

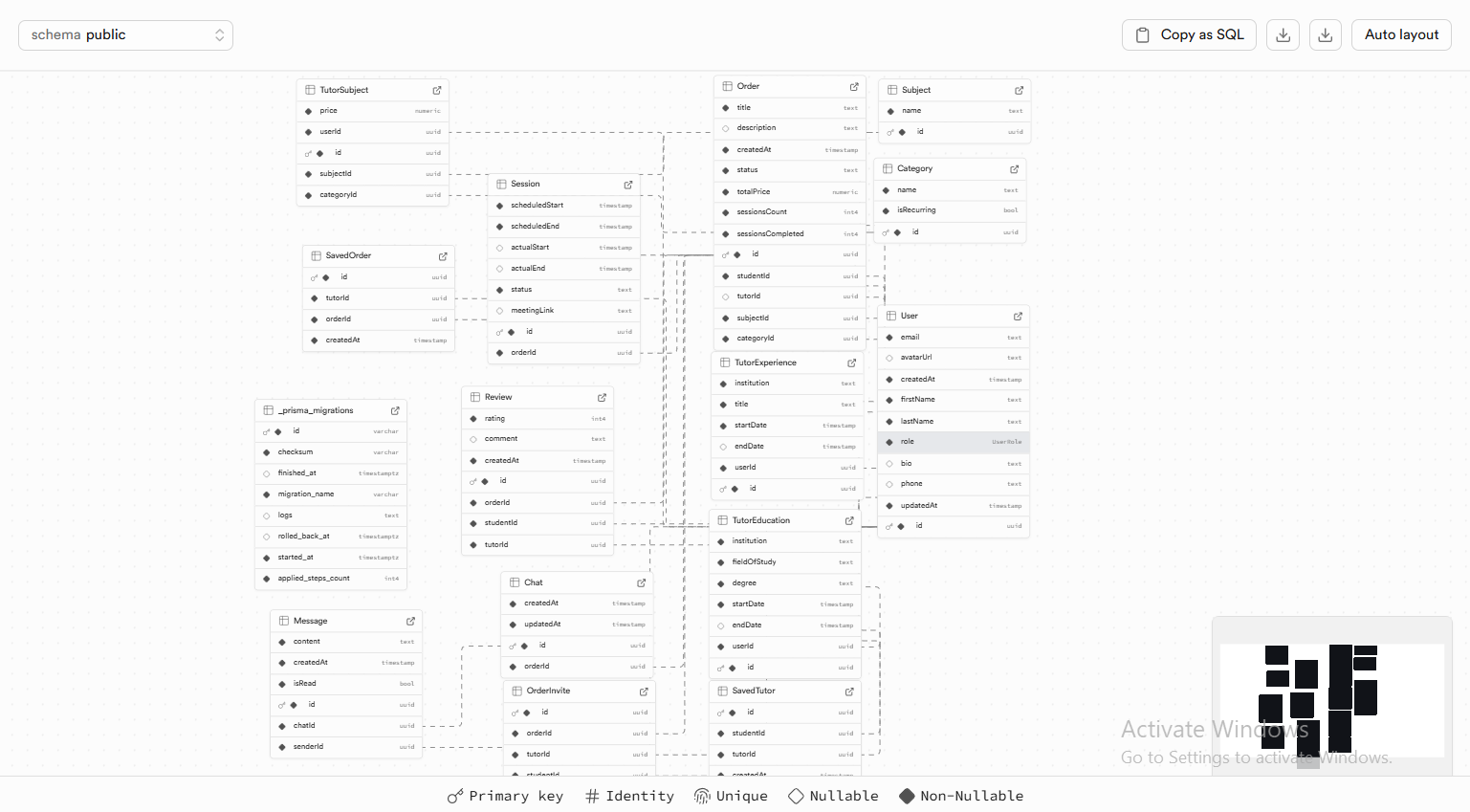


Рис. 2.3.2. SQL таблиці

- Всі зв'язки між таблицями реалізовані через зовнішні ключі, що забезпечує цілісність даних.

- Використання UUID як основного типу ідентифікаторів — це сучасний підхід, який підвищує безпеку і дозволяє уникати колізій при масштабуванні.

- Гнучка система ролей (UserRole), яка дозволяє розширювати функціонал у майбутньому (наприклад, додати адміністратора).

- Можливість зберігати історію освіти, досвіду, предметів для кожного ментора — це реально зручно для користувачів і підвищує довіру до платформи.

- Окремі таблиці для збережених замовлень і менторів — must-have для сучасних платформ, де користувачі хочуть швидко повертатися до улюблених викладачів чи занять.

Варто зазначити, що через обмеження часу не вдалося повністю реалізувати функціонал чатів (таблиці Chat, Message вже є у схемі, але реалізація на рівні API та фронтенду ще не завершена). Це залишено як перспективу для подальшого розвитку проєкту.

Загалом, структура БД MindForge — це сучасний, гнучкий і масштабований фундамент, який дозволяє розвивати платформу у будь-якому напрямку: від класичних занять до групових курсів, real-time сповіщень і мобільних застосунків.

## 2.4 Проектування API та взаємодії між сервісами

API — це серце будь-якої сучасної платформи. Саме через API відбувається вся магія: фронтенд спілкується з бекендом, користувачі отримують дані, створюють замовлення, залишають відгуки, а система працює як єдиний злагоджений механізм. Для MindForge було обрано класичний підхід — REST API, який забезпечує простоту, масштабованість і зрозумілість для розробників.

Взаємодія між клієнтською та серверною частиною відбувається через HTTP-запити до REST API. Кожен ресурс (користувач, замовлення, сесія, відгук, предмет, ментор) має власний набір ендпоінтів, які дозволяють виконувати CRUD-операції (Create, Read, Update, Delete). Це не просто стандарт, а must-have для будь-якої сучасної платформи, де важлива гнучкість і простота інтеграції.

### 2.4.1. Шляхи для API

* + **/api/auth** — реєстрація, вхід, вихід, відновлення паролю, перевірка токена (JWT).
  + **/api/users** — отримання інформації про користувачів, оновлення профілю, зміна ролі (студент/ментор).
  + **/api/tutors** — отримання списку менторів, фільтрація за предметами, категоріями, рейтингом.
  + /**api/subjects, /api/categories** — отримання списку предметів і категорій, додавання нових (для адміністратора).
  + **/api/orders** — створення замовлення, перегляд, оновлення статусу, додавання коментарів, скасування.
  + **/api/sessions** — створення та керування сесіями в рамках замовлення.
  + **/api/reviews** — додавання відгуків, перегляд відгуків для ментора чи замовлення.
  + **/api/invites** — система запрошень для взаємодії між студентами і менторами.
  + **/api/saved** — збереження улюблених замовлень і менторів.
  + **/api/chats, /api/messages** — (структура вже закладена, але реалізація ще не завершена; це залишено на майбутнє).

**2.4.2. User flow**

1. Студент реєструється через /api/auth/register, отримує JWT-токен.

2. За допомогою токена отримує доступ до /api/tutors, фільтрує менторів за предметом.

3. Створює замовлення через /api/orders, вказує предмет, категорію, опис, кількість сесій.

4. Ментор отримує сповіщення (через /api/invites або email), приймає замовлення.

5. Всі подальші дії (додавання сесій, відгуків, збереження улюблених) — через відповідні ендпоінти.

### 2.4.3. Взаємодія фронтенду і бекенду

Фронтенд (Next.js) надсилає HTTP-запити до бекенду (Express.js) через fetch/axios. Для захищених ендпоінтів використовується JWT-авторизація: токен зберігається у localStorage або httpOnly cookie, і додається до кожного запиту. Всі відповіді API стандартизовані: у разі успіху повертається об'єкт із даними, у разі помилки — об'єкт із кодом і повідомленням про помилку.

### 2.4.4. Real-time функціонал

Для реалізації real-time функцій (сповіщення, чати) планується використання можливостей Supabase (real-time subscriptions) або WebSocket. На даний момент структура для чатів і повідомлень вже закладена у БД та API, але повна реалізація цього функціоналу ще не завершена через обмеження часу. Це залишено як перспективу для подальшого розвитку проєкту.

### 2.4.5. Підсумки

- Чітка структура ендпоінтів, яка дозволяє легко масштабувати API, додавати нові ресурси без порушення існуючої логіки.

- Використання JWT для авторизації — сучасний стандарт безпеки, який дозволяє легко інтегрувати фронт і бекенд.

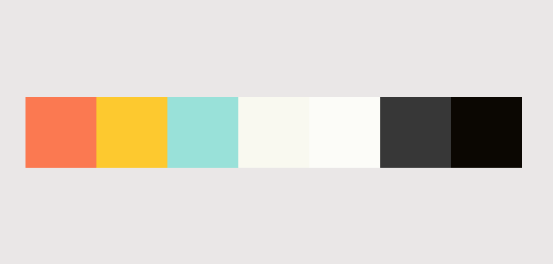
- Всі основні операції (реєстрація, створення замовлення, додавання відгуків, збереження улюблених) реалізовані через REST API, що спрощує тестування і підтримку.

- Можливість розширення API для мобільних застосунків, інтеграції з іншими сервісами (наприклад, платіжними системами, календарями, відеоплатформами).

Загалом, API MindForge — це сучасний, гнучкий і масштабований інтерфейс, який дозволяє платформі працювати швидко, безпечно і зручно для всіх користувачів. А реалізація real-time чатів — це наступний крок, який зробить платформу ще більш інтерактивною.

## 2.5 Проектування інтерфейсу користувача (UI/UX)

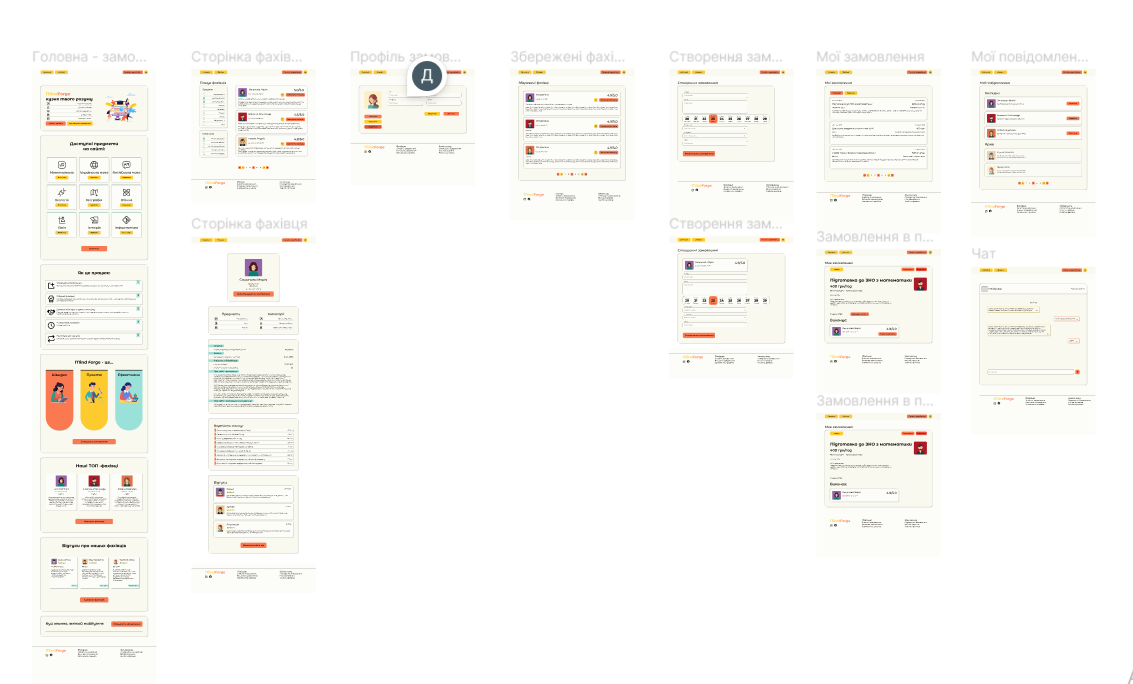
UI/UX є визначальним чинником успішності сучасного онлайн-сервісу. Саме від зручності, логіки та естетики інтерфейсу залежить, чи залишиться користувач на платформі, чи обере альтернативу. У процесі розробки MindForge було поставлено завдання створити інтерфейс, який поєднує сучасний вигляд, інтуїтивність та адаптивність. Практика показує, що навіть найфункціональніша система не буде затребуваною, якщо користувач не може швидко знайти потрібну дію або губиться у навігації.



**Рис 2.5.1**. Колірна палітра проєкту

В основі проектування інтерфейсу лежить принцип мінімалізму: відсутність зайвих елементів, максимальна увага до простору та акцентування на ключових функціях. Кожен елемент — кнопка, форма чи меню — має бути зрозумілим з першого погляду. Особливу увагу приділено адаптивності: MindForge коректно відображається як на ноутбуках, так і на мобільних пристроях, а мобільна версія є повноцінною, а не спрощеною копією. Важливим аспектом є доступність: дотримано контрастності, оптимального розміру шрифтів, логічної структури для користувачів з різними потребами.

Проектування інтерфейсу розпочиналося з аналізу найкращих практик у сфері онлайн-освіти. Було досліджено функціонал і дизайн провідних платформ (Coursera, Udemy, MentorCruise), визначено сильні та слабкі сторони їхніх рішень. Далі здійснювалося прототипування у Figma: створювалися wireframes для основних сторінок (головна, профіль, замовлення, чат, відгуки). Figma дозволила оперативно змінювати макети, тестувати різні варіанти та отримувати зворотний зв'язок. Для кольорової гами обрано спокійні відтінки синього та сірого з акцентами для кнопок і важливих елементів. Макети тестувалися на фокус-групах, збиралися відгуки, вносилися корективи.



**Рис 2.5.2** Wireframes для сторіное

Для реалізації інтерфейсу використовувалися Figma (макети), Tailwind CSS (швидка, уніфікована верстка), а також Shadcn/UI — бібліотека сучасних компонентів, які легко адаптуються під потреби платформи. Всі елементи (кнопки, поля, картки) виконані в єдиному стилі з однаковими відступами, шрифтами, тінями. Навігація організована максимально просто: головне меню, швидкий доступ до профілю, замовлень, чатів. Для форм реалізовано автозаповнення, підказки, валідацію у реальному часі. Використання SVG-іконок забезпечує чіткість відображення на будь-яких екранах. Реалізовано перемикач темної теми, що є актуальним для користувачів, які працюють у вечірній час.

Наприклад, сторінка замовлення містить інформацію про замовлення (предмет, опис, статус) ліворуч, праворуч розташовані кнопки для дій (прийняти, скасувати, написати ментору), а внизу — історія сесій, відгуки, можливість додати нову сесію.

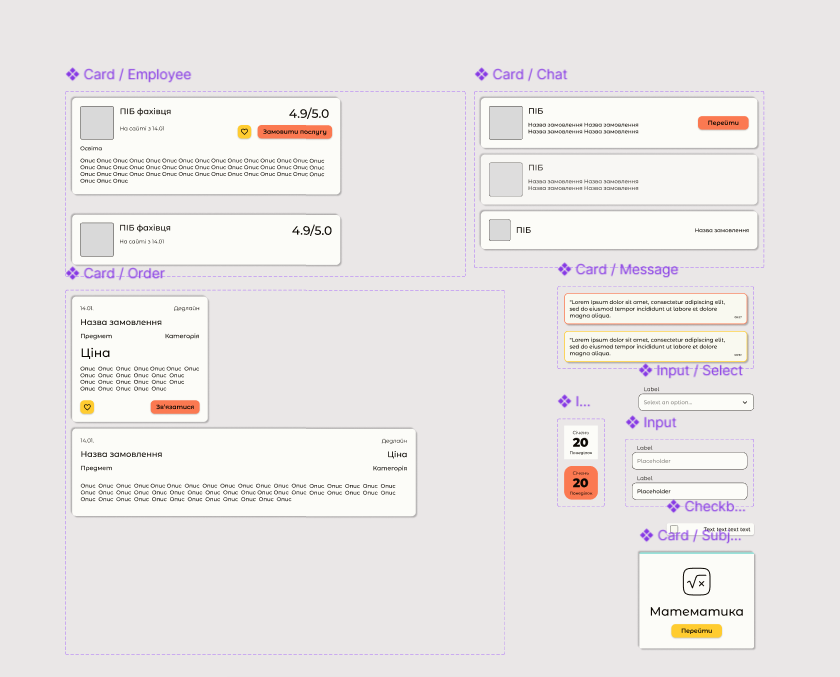
Інтерфейс MindForge є результатом пошуку балансу між естетикою та зручністю. Завдяки увазі до деталей, адаптивності та використанню сучасних підходів забезпечується позитивний досвід користувача, що є необхідною умовою для успіху онлайн-сервісу.

## 2.6 Дизайн інтерфейсу: процес, інструменти, макети

Дизайн інтерфейсу є ключовим етапом у створенні сучасної онлайн-платформи, оскільки саме візуальне оформлення визначає перше враження користувача та впливає на подальший досвід взаємодії із системою. У процесі розробки MindForge особлива увага приділялася не лише функціональності, а й естетичній складовій, що забезпечує привабливість та зручність використання.

Процес дизайну розпочинався з постановки задач, які визначали основні вимоги до інтерфейсу: простота, інтуїтивність, адаптивність та відповідність сучасним тенденціям UI/UX. Для реалізації цих вимог було обрано Figma як основний інструмент для створення макетів і прототипів. Figma дозволяє ефективно організовувати командну роботу, швидко вносити зміни та тестувати різні варіанти дизайну без значних витрат часу.

На першому етапі створювалися Wireframes для основних сторінок платформи, що дозволяло визначити структуру розміщення елементів, логіку навігації та взаємодії користувача із системою. Після затвердження структури розпочиналася робота над деталізацією макетів: підбір кольорової гами, шрифтів, іконок, відступів та інших візуальних елементів. Особлива увага приділялася адаптивності — усі макети одразу розроблялися для різних типів пристроїв, що гарантує коректне відображення інтерфейсу як на десктопах, так і на мобільних пристроях.



**Рис 2.6.1**. Компоненти в Figma

Важливим етапом було тестування макетів на фокус-групах, що дозволило отримати зворотний зв'язок від потенційних користувачів і виявити недоліки на ранніх стадіях. На основі отриманих відгуків у макети вносилися корективи, що сприяло підвищенню зручності та зрозумілості інтерфейсу. У процесі роботи використовувалися сучасні підходи до дизайну: модульність, повторне використання компонентів, дотримання принципів контрастності та ієрархії, що позитивно вплинуло на цілісність і впізнаваність платформи.

Приклади макетів, створених у Figma, включають головну сторінку, профіль користувача, сторінку замовлення, чат, форму відгуків. Кожен макет демонструє послідовність і логічність розміщення елементів, а також відповідність загальній стилістиці MindForge. Завдяки ретельно продуманому дизайну інтерфейсу платформа виглядає сучасно, привабливо та забезпечує позитивний користувацький досвід, що є важливою конкурентною перевагою на ринку онлайн-освіти.

# РОЗДІЛ ІІІ: РЕАЛІЗАЦІЯ СИСТЕМИ

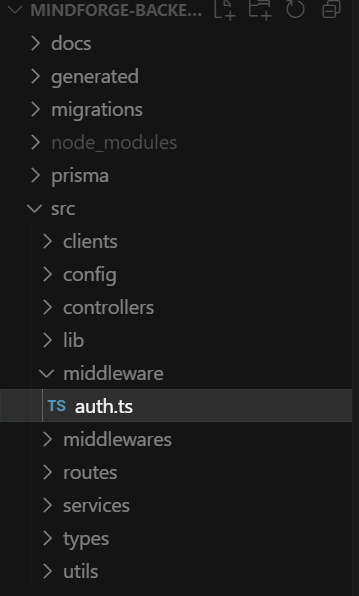
## 3.1 Розробка бекенду (Express.js, TypeScript, Prisma, Supabase)

Бекенд MindForge — це не просто набір серверних скриптів, а справжній «мозковий центр» платформи, який непомітно для користувача керує всіма процесами. Тут відбувається вся магія: від обробки запитів і перевірки прав до збереження історії замовлень і профілів. Саме бекенд відповідає за те, щоб кожен студент і ментор відчували себе у безпеці, а дані залишалися цілісними та захищеними.

В основі лежить Express.js у парі з TypeScript — це дає змогу писати чистий, передбачуваний код, де кожна помилка ловиться ще до запуску. Для роботи з базою даних обрано Prisma ORM: міст між кодом і PostgreSQL, який дозволяє працювати з таблицями через зручні типізовані методи. А ще — Supabase, який не просто зберігає дані, а й дає автентифікацію, real-time.

Архітектура побудована за принципом шарів (layered architecture), щоб не було каші з відповідальностей. Кожен шар — це окрема зона відповідальності:

* + **Контролери (controllers/)** — приймають HTTP-запити, викликають потрібні операції, формують відповіді. Наприклад, коли студент створює замовлення, контролер перевіряє права, передає дані далі й повертає результат.
  + **Операції (operations/)** — це такі собі «диригенти», які об'єднують сервіси, валідують дані, керують транзакціями. Вони відповідають за цілісний сценарій: наприклад, створення замовлення або оновлення профілю.
  + **Сервіси (services/)** — тут живе бізнес-логіка. Вони не знають про HTTP, але знають усе про правила платформи: як створити замовлення, як порахувати рейтинг ментора, як зберегти відгук.
  + **Мідлвари (middlewares/)** — перевіряють токени, права доступу, валідність даних ще до того, як запит потрапить у контролер.
  + **Типи та утиліти (types/, utils/)** — для типізації, валідації, логування, пагінації тощо.



**Рис 3.1.1.** Структура бекенду

Уявімо, студент хоче знайти ментора. Він створює замовлення через фронт. Запит летить на бекенд, де:

1. Контролер приймає запит, перевіряє токен через мідлвар, передає дані в операцію.

2. Операція валідує дані, викликає сервіс створення замовлення, обробляє можливі помилки.

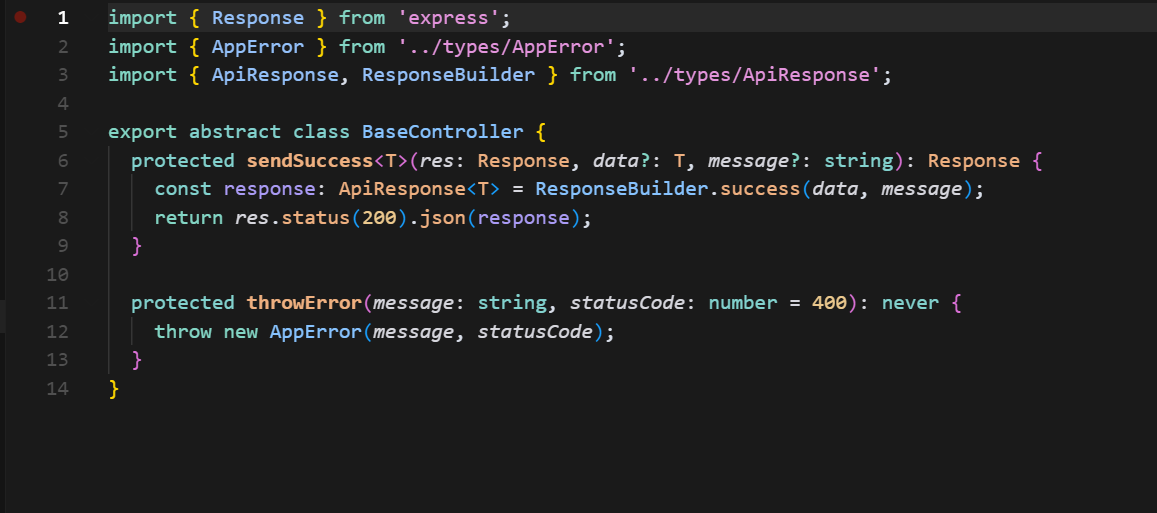
3. Сервіс перевіряє, чи всі поля заповнені, створює запис у базі через репозиторій, підбирає релевантних менторів.

4. Репозиторій через Prisma створює новий запис у таблиці orders.

5. Після успіху операція повертає результат контролеру, а той — фронту.

Вся авторизація — через Supabase Auth. Кожен користувач отримує унікальний токен, який перевіряється на кожному запиті. Ролі (студент, ментор, адмін) визначають, до яких розділів є доступ. Наприклад, студент не зможе видалити чужий відгук, а ментор — змінити чужий профіль. Це все контролюється на рівні middleware.

Щоб фронту було зручно працювати, бекенд завжди повертає структуровані об'єкти: у разі успіху — дані, у разі помилки — зрозуміле повідомлення з поясненням.



**Рис 3.1.2.** Базовий контролер, від якого наслідують наступні

Бекенд MindForge — це не просто сервер, а справжній «організатор» усіх процесів. Завдяки продуманій архітектурі, сучасним технологіям і чіткій структурі, він забезпечує надійність, безпеку й зручність для кожного користувача. А ще — дає змогу легко розвивати платформу, додаючи нові функції без страху зламати щось старе.

## 3.2 Розробка фронтенду (Next.js, Tailwind CSS, Redux, Shadcn/UI)

Фронтенд MindForge — це не просто «обличчя» платформи, а справжній майданчик для експериментів із сучасними підходами до розробки. Тут усе крутиться навколо зручності, швидкості й того самого «вау»-ефекту, коли навіть новачок відчуває себе впевнено.

В основі — Next.js, який дає змогу поєднати серверний рендеринг (SSR) із класичним SPA-досвідом. Це означає, що сторінки завантажуються блискавично, а SEO-боти не плачуть у кутку. Весь код живе у папці `src/`, а далі — чітка ієрархія:

* + **`app/`** — маршрути, сторінки, layout-и. Тут усе, що відповідає за навігацію: від головної до профілю ментора чи сторінки замовлення. Для студентів і менторів — окремі підпапки, щоб не плутати сценарії.
  + **`components/`** — бібліотека UI-елементів. Є все: кнопки, картки, інпути, layout-компоненти (header, footer, page-wrapper), навіть кастомні аватари. Більшість — на базі Shadcn/UI, але з купою власних фішок і стилів.
  + **`lib/`** — утиліти, хуки, API-клієнти. Наприклад, тут лежать кастомні React-хуки для роботи з формами, профілем, а також функції для запитів до бекенду.
  + **`types/`** — типи для TypeScript. Завдяки цьому код не розвалюється від однієї помилки в імені поля, а автодоповнення працює як швейцарський годинник.
  + **`services/`** — бізнес-логіка, наприклад, робота з профілями, предметами, менторськими даними.

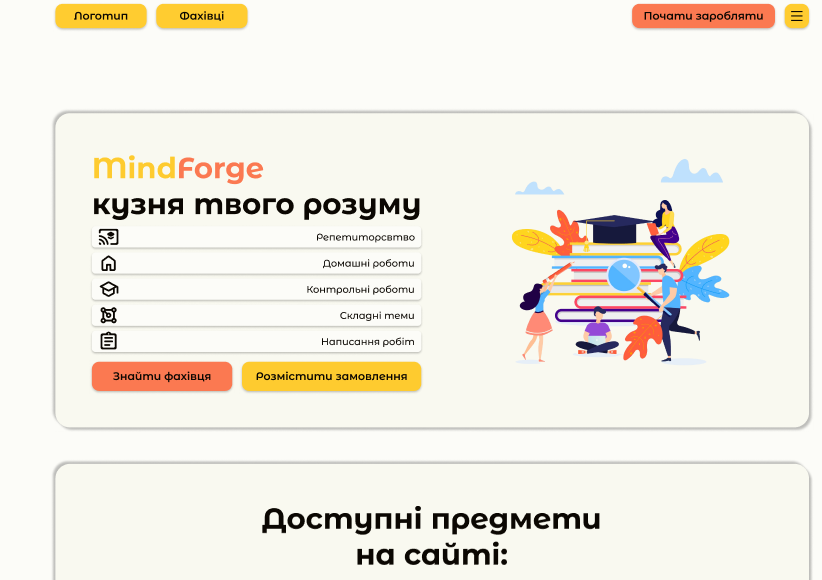
Вся стилізація — на Tailwind CSS. Це не просто модний фреймворк, а справжній рятівник, коли треба швидко змінити дизайн або підлаштувати щось під адаптив. Кожен компонент — від кнопки до складної форми — виглядає гармонійно й адаптивно.

Особливо реалізовані картки (order-card, employee-card, review-card тощо): вони не просто відображають дані, а й підлаштовуються під сценарій (наприклад, для студента чи ментора). Є навіть окремі компоненти для відгуків, цін, кроків реєстрації — усе, щоб користувач не губився.

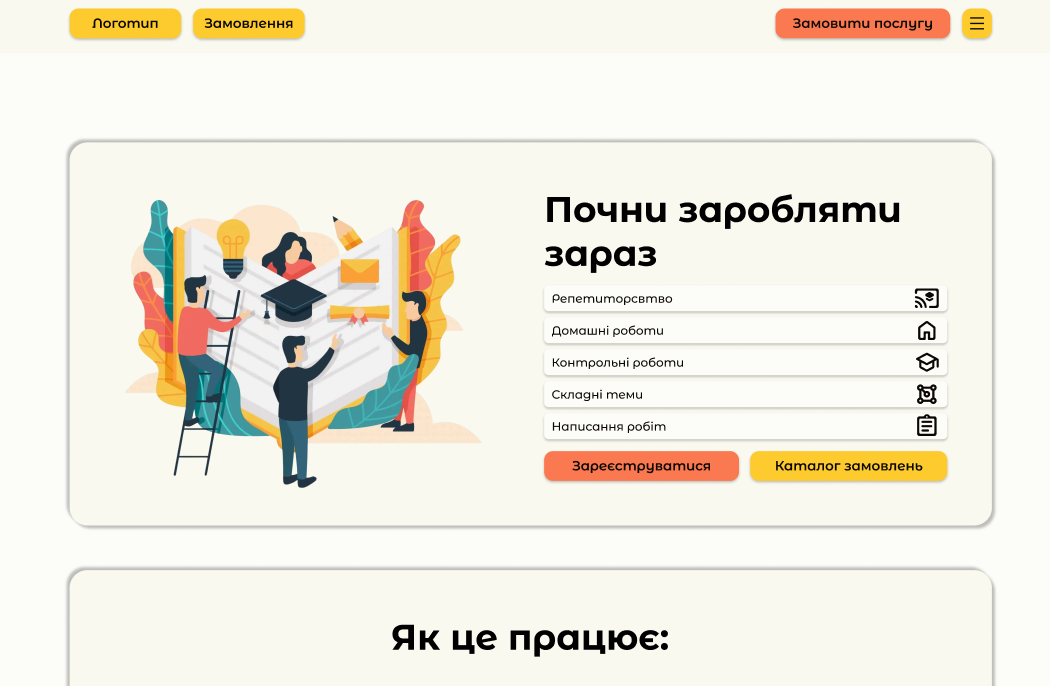
Навігація побудована так, щоб студент одразу потрапляв у потрібний розділ: каталог менторів, створення замовлення, перегляд своїх заявок чи збережених фахівців. Для менторів — окремий кабінет із розкладом, списком студентів, повідомленнями. Кожна сторінка — це окремий файл у відповідній папці, що спрощує підтримку й розширення.

MindForge виглядає однаково круто і на великому моніторі, і на смартфоні. Tailwind дозволяє швидко підлаштовувати компоненти під будь-який розмір екрана. Є перемикач темної теми (дуже зручно для тих, хто працює вечорами), автозаповнення форм, підказки, валідація у реальному часі.

Всі запити до бекенду — через окремі API-клієнти. Це дозволяє легко змінювати ендпоінти або додавати нові сценарії без тотального рефакторингу. Типи для відповідей прописані окремо, тому помилитися складно.



**Рис 3.2.1**. Головна сторінка для студента



**Рис 3.2.2**. Головна сторінка ментора

Загалом, фронтенд MindForge — це про турботу, сучасність і бажання зробити платформу максимально дружньою для кожного. І, чесно кажучи, працювати над цим було не менш цікаво, ніж користуватися готовим продуктом!

## 3.3 Інтеграція фронтенду та бекенду, real-time функціонал

Інтеграція фронтенду та бекенду у MindForge — це той міст, який поєднує всі частини платформи в єдине ціле. Саме завдяки цій взаємодії користувачі отримують можливість реєструватися, створювати замовлення, знаходити менторів, залишати відгуки та керувати своїм навчальним процесом у зручному інтерфейсі. Всі дії на клієнтській стороні миттєво відображаються у базі даних, а відповіді сервера повертаються у зрозумілому та структурованому вигляді.

Обмін даними між фронтендом і бекендом організовано через REST API. Кожен запит — це окрема історія: студент надсилає заявку на пошук ментора, бекенд перевіряє дані, підбирає релевантних кандидатів і повертає результат. Всі запити захищені: для доступу до особистої інформації чи створення замовлення потрібен авторизаційний токен, який перевіряється на сервері. Такий підхід гарантує безпеку та конфіденційність для кожного користувача.

Особливу увагу приділено стандартизації форматів запитів і відповідей. Це дозволяє уникати непорозумінь між клієнтом і сервером, а також спрощує розширення функціоналу у майбутньому. Наприклад, при створенні замовлення студент бачить актуальний список менторів, а ментор — отримує сповіщення про нові заявки. Всі ці процеси відбуваються швидко й непомітно для користувача, створюючи відчуття цілісності платформи.

Щодо real-time функціоналу, то його впровадження заплановано на наступні етапи розвитку MindForge. Уже зараз у структурі бази даних і API закладено основу для реалізації чатів і миттєвих сповіщень, які дозволять студентам і менторам спілкуватися у реальному часі, отримувати оновлення про статус замовлень, нові відгуки чи зміни у розкладі. Це стане важливим кроком до ще більш інтерактивної та динамічної платформи.

Таким чином, інтеграція фронтенду та бекенду у MindForge забезпечує не лише технічну цілісність, а й комфорт, безпеку та сучасний досвід для кожного користувача. А перспектива впровадження real-time функцій відкриває нові горизонти для розвитку платформи й підвищення її конкурентоспроможності.

## 3.4 Тестування системи (Frontend + Backend)

Тестування MindForge — це невід’ємна частина розробки, яка дозволяє впевнено стверджувати: платформа працює стабільно, а користувачі можуть довіряти кожному її елементу. Саме завдяки ретельній перевірці функціоналу вдається уникнути неприємних сюрпризів, підвищити якість сервісу та забезпечити позитивний досвід для всіх учасників освітнього процесу.

Тестування охоплює як фронтенд, так і бекенд. На клієнтській стороні особлива увага приділяється перевірці зручності інтерфейсу, коректності відображення елементів на різних пристроях, а також логіці переходів між сторінками. Для автоматизованого тестування інтерфейсу використовується Playwright — сучасний інструмент, який дозволяє моделювати дії реального користувача, перевіряти роботу форм, навігацію, адаптивність і навіть складні сценарії взаємодії. Це дає змогу швидко знаходити й виправляти помилки ще до того, як їх помітять користувачі.

На бекенді тестування зосереджене на перевірці коректності обробки запитів, захисту даних, стабільності роботи з базою та відповідності бізнес-логіки. Для автоматизації цього процесу використовується Jest — популярний фреймворк для тестування JavaScript і TypeScript. Завдяки Jest можна швидко перевіряти окремі функції, контролери, сервіси, а також імітувати різні сценарії взаємодії з API. Це дозволяє підтримувати високу якість коду та впевненість у стабільності бекенду

Тестування проводиться як вручну, так і автоматизовано. Для основних сценаріїв створюються тест-кейси: реєстрація, вхід, створення замовлення, пошук ментора, залишення відгуку, зміна профілю. Кожен кейс описує послідовність дій, очікуваний результат і критерії успішності. Наприклад, якщо студент вводить неправильний email, система має одразу повідомити про помилку, а не пропустити далі.

Особливу увагу приділено тестуванню безпеки: перевіряється, чи не можна отримати доступ до чужих даних, чи коректно працює авторизація, чи захищені персональні дані. Також тестується стійкість до навантажень — платформа має залишатися стабільною навіть при великій кількості одночасних користувачів.

Завдяки такому підходу тестування MindForge не є формальністю, а стає запорукою якості, довіри та впевненості у кожному релізі. Це дозволяє користувачам не замислюватися про технічні нюанси, а зосередитися на головному — навчанні та розвитку.

## 3.5 Деплой, контейнеризація

Розгортання MindForge — це завершальний етап, який перетворює проєкт із локальної розробки на доступний для користувачів онлайн-сервіс. Для забезпечення стабільної роботи, масштабованості та зручності обслуговування платформи обрано сучасний підхід із використанням контейнеризації та хмарного хостингу.

В основі деплою лежить Docker — технологія, яка дозволяє «запакувати» всю інфраструктуру проєкту у незалежні контейнери. Завдяки цьому серверна частина, база даних та допоміжні сервіси працюють ізольовано, не заважаючи одне одному, а розгортання нових версій відбувається швидко й безболісно. У репозиторії проєкту міститься Dockerfile, який описує всі необхідні кроки для збирання та запуску бекенду у контейнері.

Для хостингу обрано платформу Render, яка дозволяє автоматично запускати Docker-контейнери у хмарі. Це забезпечує високу доступність сервісу, автоматичне масштабування та простоту адміністрування. Завдяки такому підходу платформа MindForge завжди залишається доступною для користувачів, незалежно від навантаження чи технічних оновлень.

Всі репозиторії проєкту розміщені на GitLab — популярній платформі для командної розробки, яка підтримує зручний контроль версій, спільну роботу над кодом і можливість впровадження CI/CD у майбутньому. Окремо варто відзначити, що розробка MindForge ведеться українським розробником, що додає проєкту особливого національного колориту та актуальності в сучасних реаліях.

На даному етапі CI/CD ще не впроваджено, але у планах — налаштування автоматичного тестування, збірки та деплою через GitLab CI/CD. Це дозволить ще більше підвищити якість і стабільність платформи, а також спростить процес оновлення для всієї команди.

Завдяки такому підходу деплой MindForge є не лише технічно грамотним, а й максимально зручним для подальшого розвитку, масштабування та підтримки платформи.

# ВИСНОВОК

Платформа MindForge стала результатом багатомісячної роботи, численних експериментів, пошуків оптимальних рішень і постійного вдосконалення. На момент завершення цієї дипломної роботи проєкт перебуває на стадії MVP — мінімально життєздатного продукту, який уже дозволяє користувачам взаємодіяти, створювати замовлення, знаходити менторів, залишати відгуки та будувати власний освітній шлях у сучасному онлайн-середовищі.

Створення MindForge стало відповіддю на актуальні виклики ринку онлайн-освіти. Сучасний студент і ментор потребують не просто платформи для пошуку одне одного, а цілісної екосистеми, яка забезпечує зручність, безпеку, гнучкість і можливість розвитку. Саме ці принципи лягли в основу архітектури, дизайну та функціоналу MindForge. Платформа поєднує сучасний технологічний стек (Next.js, Express.js, Prisma, Supabase, Tailwind CSS, Redux, Shadcn/UI), що дозволяє забезпечити швидкість, масштабованість і простоту підтримки.

Особливу увагу приділено безпеці та захисту персональних даних. Використання Supabase Auth, JWT-токенів, перевірки ролей і прав доступу гарантує, що кожен користувач може бути впевнений у конфіденційності своєї інформації. Всі основні сценарії — від реєстрації до створення замовлення чи залишення відгуку — реалізовані так, щоб користувач отримував позитивний досвід незалежно від пристрою чи рівня технічної підготовки.

Важливо підкреслити, що MindForge — це лише перший крок на шляху до створення повноцінної освітньої платформи. На даному етапі реалізовано багатий функціонал: реєстрація, профілі студентів і менторів, створення та пошук замовлень, система відгуків, збереження улюблених менторів і замовлень, система запрошень, адаптивний інтерфейс, захищена авторизація, тестування, деплой через Docker і Render. Проте, як і будь-який MVP, проєкт має свої обмеження: частина функціоналу (зокрема, real-time чати, розширене керування сесіями, інтеграція платіжних систем, мобільний застосунок) ще перебуває у планах і стане наступним етапом розвитку.

Не можна оминути увагою і наявність багів та недопрацювань, які неминучі на стадії MVP. Деякі сценарії потребують додаткового тестування, оптимізації чи рефакторингу. Однак гнучка архітектура, модульність коду та використання сучасних інструментів дозволяють швидко виявляти й виправляти помилки, а також додавати нові можливості без ризику для стабільності платформи.

Важливим досягненням є організація процесу розробки: використання GitLab для контролю версій, командної роботи та планування CI/CD, впровадження автоматизованого тестування (Playwright для фронтенду, Jest для бекенду), а також контейнеризація через Docker, що спрощує деплой і масштабування. Всі ці підходи відповідають сучасним стандартам розробки й забезпечують надійність та зручність підтримки проєкту.

Окремо варто відзначити українське походження MindForge. Проєкт створювався з урахуванням потреб українських студентів і менторів, а також із розумінням сучасних реалій та викликів, які стоять перед освітньою сферою країни. Це додає платформі особливого змісту й актуальності, а також відкриває перспективи для подальшого розвитку на локальному та міжнародному ринках.

Підсумовуючи, можна стверджувати, що MindForge — це не просто ще одна освітня платформа, а жива, гнучка й перспективна екосистема, яка здатна змінити підхід до навчання, зробити його більш персоналізованим, доступним і ефективним. Попереду — інтеграція нових сервісів, розширення функціоналу, впровадження real-time комунікацій, мобільних застосунків, платіжних систем, а також подальше вдосконалення інтерфейсу та користувацького досвіду.

MindForge — це лише початок великої історії. Попереду — нові виклики, амбітні цілі та безліч можливостей для зростання. Платформа вже сьогодні готова стати надійним помічником для студентів і менторів, а завтра — перетворитися на справжній центр сучасної онлайн-освіти.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gerlich, Michael. "AI Tools in Society: Impacts on Cognitive Offloading and the Future of Critical Thinking." *Societies*, vol. 15, no. 1, 2025, article 6, <https://doi.org/10.3390/soc15010006>.
2. Christensen, C. M., Horn, M. B., & Johnson, C. W. Disrupting Class: How Disruptive Innovation Will Change the Way the World Learns. — New York: McGraw-Hill, 2011. — 288 p.
3. Bates, A. W. Teaching in a Digital Age: Guidelines for designing teaching and learning. — 3rd Edition. — Vancouver, B.C.: Tony Bates Associates Ltd., 2022. — 526 p.
4. OECD. The State of School Education: One Year into the COVID Pandemic. — OECD Publishing, 2021. — 128 p. — Режим доступу: https://www.oecd.org/education/state-of-school-education-one-year-into-covid.htm
5. World Economic Forum. The Future of Jobs Report 2020. — Geneva: WEF, 2020. — 163 p. — Режим доступу: https://www.weforum.org/reports/the-future-of-jobs-report-2020
6. McKinsey & Company. How technology is shaping learning in the digital age. — 2022. — Режим доступу: https://www.mckinsey.com/industries/education/our-insights/how-technology-is-shaping-learning-in-the-digital-age
7. EdTech Ukraine. Огляд ринку EdTech в Україні. — 2023. — Режим доступу: https://edtechua.org/market-overview
8. Nielsen, J. Usability 101: Introduction to Usability // Nielsen Norman Group. — 2020. — Режим доступу: https://www.nngroup.com/articles/usability-101-introduction-to-usability/
9. Coursera. About Coursera. — 2024. — Режим доступу: https://about.coursera.org/
10. Udemy. About Udemy. — 2024. — Режим доступу: https://about.udemy.com/
11. MentorCruise. How MentorCruise works. — 2024. — Режим доступу: https://mentorcruise.com/how-it-works/
12. Supabase. The Open Source Firebase Alternative. — 2024. — Режим доступу: https://supabase.com/docs
13. Next.js Documentation. The React Framework for Production. — 2024. — Режим доступу: https://nextjs.org/docs
14. Docker Documentation. What is a Container? — 2024. — Режим доступу: https://docs.docker.com/get-started/overview/
15. Prisma. Next-generation Node.js and TypeScript ORM. — 2024. — Режим доступу: https://www.prisma.io/docs
16. Tailwind CSS Documentation. Rapidly build modern websites. — 2024. — Режим доступу: https://tailwindcss.com/docs
17. Playwright. Fast and reliable end-to-end testing for modern web apps. — 2024. — Режим доступу: https://playwright.dev/docs/intro
18. Jest. Delightful JavaScript Testing. — 2024. — Режим доступу: https://jestjs.io/docs/getting-started
19. Shadcn/UI. Beautifully designed components. — 2024. — Режим доступу: https://ui.shadcn.com/docs

# ДОДАТКИ

**Схема Prisma:**

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider  = "postgresql"

  url       = env("DATABASE\_URL")

  directUrl = env("DIRECT\_URL")

}

enum UserRole {

  STUDENT

  TUTOR

}

model User {

  id             String            @id @default(uuid()) @db.Uuid

  email          String            @unique

  firstName      String

  lastName       String

  avatarUrl      String?

  createdAt      DateTime          @default(now())

  updatedAt      DateTime          @updatedAt

  role           UserRole          @default(STUDENT)

  bio            String?

  phone          String?

*// Common relations*

  sentMessages   Message[]

  studentOrders  Order[]           @relation("StudentOrders")

  studentReviews Review[]          @relation("StudentReviews")

*// Tutor specific relations*

  tutorOrders    Order[]           @relation("TutorOrders")

  tutorReviews   Review[]          @relation("TutorReviews")

  education      TutorEducation[]

  experiences    TutorExperience[]

  subjects       TutorSubject[]

*// Relations for invites*

  tutorInvites   OrderInvite[]      @relation("TutorInvites")

  studentInvites OrderInvite[]      @relation("StudentInvites")

*// Relations for saving*

  savedOrders      SavedOrder[]        @relation("SavedOrdersByTutor")

  savedTutorsAsStudent SavedTutor[]    @relation("SavedTutorsStudent")

  savedTutorsAsTutor   SavedTutor[]    @relation("SavedTutorsTutor")

}

model TutorEducation {

  id           String    @id @default(uuid()) @db.Uuid

  userId       String    @db.Uuid

  institution  String

  fieldOfStudy String

  degree       String

  startDate    DateTime

  endDate      DateTime?

  user         User      @relation(*fields*: [userId], *references*: [id])

}

model TutorExperience {

  id          String    @id @default(uuid()) @db.Uuid

  userId      String    @db.Uuid

  institution String

  title       String

  startDate   DateTime

  endDate     DateTime?

  user        User      @relation(*fields*: [userId], *references*: [id])

}

model TutorSubject {

  id         String    @id @default(uuid()) @db.Uuid

  userId     String    @db.Uuid

  subjectId  String    @db.Uuid

  categoryId String    @db.Uuid

  price      Decimal

*// Relations*

  user       User      @relation(*fields*: [userId], *references*: [id])

  subject    Subject   @relation(*fields*: [subjectId], *references*: [id])

  category   Category  @relation(*fields*: [categoryId], *references*: [id])

  @@unique([userId, subjectId, categoryId])

}

model Subject {

  id            String         @id @default(uuid()) @db.Uuid

  name          String         @unique

  tutorSubjects TutorSubject[]

  orders        Order[]

}

model Category {

  id            String         @id @default(uuid()) @db.Uuid

  name          String         @unique

  isRecurring   Boolean        @default(false)

  tutorSubjects TutorSubject[]

  orders        Order[]

}

model Order {

  id                String    @id @default(uuid()) @db.Uuid

  studentId         String    @db.Uuid

  tutorId           String?   @db.Uuid

  subjectId         String    @db.Uuid

  categoryId        String    @db.Uuid

  title             String

  description       String?

  createdAt         DateTime  @default(now())

  status            String

  totalPrice        Decimal

  sessionsCount     Int

  sessionsCompleted Int       @default(0)

  chat              Chat?

*// Relations*

  category          Category  @relation(*fields*: [categoryId], *references*: [id])

  student           User      @relation("StudentOrders", *fields*: [studentId], *references*: [id])

  subject           Subject   @relation(*fields*: [subjectId], *references*: [id])

  tutor             User?     @relation("TutorOrders", *fields*: [tutorId], *references*: [id])

  reviews           Review[]

  sessions          Session[]

*// Relations for invites*

  invites           OrderInvite[]

*// Relations for saving*

  savedByTutors SavedOrder[] @relation("SavedOrdersByOrder")

}

model Session {

  id             String    @id @default(uuid()) @db.Uuid

  orderId        String    @db.Uuid

  scheduledStart DateTime

  scheduledEnd   DateTime

  actualStart    DateTime?

  actualEnd      DateTime?

  status         String

  meetingLink    String?

  order          Order     @relation(*fields*: [orderId], *references*: [id])

}

model Review {

  id        String    @id @default(uuid()) @db.Uuid

  orderId   String    @db.Uuid

  studentId String    @db.Uuid

  tutorId   String    @db.Uuid

  rating    Int

  comment   String?

  createdAt DateTime  @default(now())

*// Relations*

  order     Order     @relation(*fields*: [orderId], *references*: [id])

  student   User      @relation("StudentReviews", *fields*: [studentId], *references*: [id])

  tutor     User      @relation("TutorReviews", *fields*: [tutorId], *references*: [id])

}

model Chat {

  id        String    @id @default(uuid()) @db.Uuid

  orderId   String    @unique @db.Uuid

  createdAt DateTime  @default(now())

  updatedAt DateTime  @updatedAt

*// Relations*

  order     Order     @relation(*fields*: [orderId], *references*: [id])

  messages  Message[]

}

model Message {

  id        String    @id @default(uuid()) @db.Uuid

  chatId    String    @db.Uuid

  senderId  String    @db.Uuid

  content   String

  createdAt DateTime  @default(now())

  isRead    Boolean   @default(false)

*// Relations*

  chat      Chat      @relation(*fields*: [chatId], *references*: [id])

  sender    User      @relation(*fields*: [senderId], *references*: [id])

}

model OrderInvite {

  id         String    @id @default(uuid()) @db.Uuid

  orderId    String    @db.Uuid

  tutorId    String    @db.Uuid

  studentId  String    @db.Uuid

  senderRole UserRole

  status     InviteStatus

  message    String?

  createdAt  DateTime  @default(now())

  updatedAt  DateTime  @updatedAt

*// Relations*

  order      Order     @relation(*fields*: [orderId], *references*: [id])

  tutor      User      @relation("TutorInvites", *fields*: [tutorId], *references*: [id])

  student    User      @relation("StudentInvites", *fields*: [studentId], *references*: [id])

}

enum InviteStatus {

  PENDING

  ACCEPTED

  REFUSED

}

model SavedOrder {

  id        String   @id @default(uuid()) @db.Uuid

  tutorId   String   @db.Uuid

  orderId   String   @db.Uuid

  createdAt DateTime @default(now())

  tutor     User     @relation("SavedOrdersByTutor", *fields*: [tutorId], *references*: [id])

  order     Order    @relation("SavedOrdersByOrder", *fields*: [orderId], *references*: [id])

  @@unique([tutorId, orderId])

}

model SavedTutor {

  id        String   @id @default(uuid()) @db.Uuid

  studentId String   @db.Uuid

  tutorId   String   @db.Uuid

  createdAt DateTime @default(now())

  student   User     @relation("SavedTutorsStudent", *fields*: [studentId], *references*: [id])

  tutor     User     @relation("SavedTutorsTutor", *fields*: [tutorId], *references*: [id])

  @@unique([studentId, tutorId])

}