МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна вебсистема для пошуку професійних репетиторів»
здобувачки групи ІН-01 Заїки Софії Олексіївни

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

Софія ЗАЇКА
(підпис)

Керівник,
старший викладач комп'ютерних
наук, кандидат технічних наук
Олег БЕРЕСТ

(підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Заїки Софії Олексіївни

1. Тема роботи: «Інформаційна веб-система для пошуку професійних репетиторів» затверджую наказом по СумДУ від *22 квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз схожих ресурсів у відповідній галузі, визначення та розробка завдань дослідження. 2) Огляд технологій, що використовуються для створення веб-додатків. 3) Створення дизайну для сайту для пошуку професійних репетиторів. 4) Розробка інформаційної веб-системи для пошуку репетиторів. 5) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз схожих ресурсів у відповідній галузі, визначення та розробка завдань дослідження*		
2	*Огляд технологій, що використовуються для створення веб-додатків*		
3	*Створення дизайну сайту для пошуку професійних репетиторів*		
4	*Розробка інформаційної веб-системи для пошуку репетиторів*		
5	*Аналіз отриманих результатів*		
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 104 стр., 81 рис., 2 додатки, 20 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки вона спрямована на створення інформаційної веб-системи для пошуку професійних репетиторів, що відповідає сучасним потребам освітнього середовища. Застосування такої системи сприяє ефективній комунікації студентів з кваліфікованими викладачами відповідно до їхніх учбових вимог та часових рамок**.**

Об’єкт дослідження — процес розробки інформаційної веб-системи для пошуку професійних репетиторів.

Мета роботи — розробка інформаційної веб-системи для пошуку професійних репетиторів, яка забезпечує швидкий, зручний та ефективний пошук репетиторів за різними дисциплінами та параметрами.

Методи дослідження — інструменти для розробки функціональної клієнтської сторони, інструменти для ефективного керування та збереження даних, інструменти, призначені для створення інформаційних веб-систем.

Результати — розроблено інформаційну веб-систему для пошуку професійних репетиторів, яка дозволяє користувачам знаходити викладачів відповідно до їх освітніх потреб. Система надає можливість фільтрації за предметами, ціновими категоріями та часовими рамками, забезпечує реєстрацію і управління розкладом репетиторів, а також дозволяє студентам записуватися на уроки.

ІНФОРМАЦІЙНА ВЕБ-СИСТЕМА, ПОШУК РЕПЕТИТОРІВ, REACT, NEST, POSTGRESQL, DOCKER, MATERIAL UI .

ЗМІСТ

ВСТУП

У сучасному суспільстві технології відіграють дедалі важливішу роль в освітньому процесі. Освіта онлайн набирає популярності, оскільки дозволяє людям з різних куточків світу отримувати знання без фізичного відвідування навчальних закладів. За даними Inside Higher Ed [1] близько 51,8% студентів пройшли принаймні один онлайн-курс у 2019-2020 роках, що свідчить про те, що частка студентів, які беруть участь в онлайн-освіті, вища, ніж вважалося раніше. В іншому звіті підкреслюється, що 6,6 мільйона студентів, або близько 33% з 19,7 мільйона студентів, навчаються в тій чи іншій формі онлайн-освіти [2]. Велика гнучкість у виборі часу занять, доступність широкого спектру програм, а також вартість, яка часто значно нижча, ніж у традиційних університетах, роблять цей формат особливо привабливим. У цьому контексті розробка веб-проектів, спрямованих на покращення освітніх можливостей, є актуальним і важливим завданням.

Ця робота присвячена створенню веб-сайту, який пропонує послуги репетиторства. Основною метою проекту є реалізація зручного та ефективного веб-сайту, на якому студенти зможуть знайти підходящих репетиторів з різних предметів, котрі піходять їм по ціновій категорії та знанням, а репетиторам розмістити свої послуги та керувати розкладом своїх занять.

Для реалізації цього проекту було обрано наступні технології:

- React JS – широко використовувана бібліотека для розробки користувацьких інтерфейсів. Вона дозволяє створювати ефективні та зручні користувацькі інтерфейси; використання React JS гарантує високий рівень інтерактивності та зручності для користувачів сайту.
- Nest JS – прогресивний фреймворк для розробки серверних додатків. Він має важливу особливість: можливість створювати надійні та масштабовані бекенди веб-сайтів. Його використання забезпечує стабільну та ефективну роботу сервера.
- PostgreSQL – це потужна реляційна база даних і досить поширений вибір для зберігання структурованих даних; PostgreSQL дозволяє

користувачам ефективно зберігати та керувати своїми даними. Це надає можливість оптимально організувати дані та забезпечити їх надійність.

- Docker – контейнеризаційна платформа, яка дозволяє ізолювати додаток з усіма його залежностями в окремому контейнері. Використання Docker забезпечує стабільне середовище для розробки та розгортання додатків, що дозволяє уникнути проблем з сумісністю різних середовищ.
- Material UI – набір компонентів інтерфейсу користувача для React, створений згідно з принципами Material Design. Material UI забезпечує сучасний та уніфікований вигляд інтерфейсу, що прискорює процес розробки інтерфейсу.
- Swagger – інструмент для документування та тестування REST API. Використання Swagger дозволяє створювати інтерактивну документацію для API, що полегшує розробку та інтеграцію різних компонентів системи, а також забезпечує прозорість та зручність використання API для розробників.

Таким чином, поєднуючи ці технології, основною метою даної роботи є створення веб-платформи, яка відповідає сучасним освітнім вимогам та допомагає покращити навчальний процес та взаємодію між викладачем та студентом.

1 ОГЛЯД АНАЛОГІВ ТА ТЕХНОЛОГІЙ

1.1 Онлайн-навчання та репетиторство

Сьогодення характеризується швидкими змінами в технологіях, економічній динаміці та споживчих запитах. Як наслідок, знання та навички потребують постійного оновлення. Люди шукають нові можливості для навчання та особистого розвитку, де важливу роль відіграють різноманітні навчальні платформи. Навчальні платформи пропонують кожному користувачеві можливість отримати нові знання від експертів у різних галузях, пристосовані до його індивідуальних потреб.

Репетитори завжди були цінним інструментом для покращення академічної успішності студентів. Взаємодія з викладачами допомагає студентам вирішувати конкретні проблеми та недоліки, з якими вони стикаються. Варто зазначити, що індивідуальний підхід та особиста увага роблять процес навчання більш ефективним. Однак традиційна форма репетиторства може бути обмежена географічними факторами, розкладом і високою вартістю.

З розвитком інтернет-технологій репетиторство перейшло в онлайн-формат. Платформи онлайн-навчання дозволяють студентам знайти репетитора будь-де, де вони мають доступ до інтернету, і взаємодіяти з ним у режимі реального часу. Це робить навчання більш гнучким, оскільки студенти можуть обирати програму і метод навчання, які найкраще їм підходять. Такий підхід також робить навчання доступнішим, особливо для тих, хто живе у віддалених районах або має обмежені можливості зустрічатися з викладачами віч-на-віч. Це особливо актуально в наш час, коли багато українців живуть за кордоном і потребують вивчення іноземної мови з україномовним викладачем.

Саме через релевантність сайтів для пошуку репетиторів для навчання їх існує досить багато. Тому, у наступному підрозділі буде розглянуто 3 популярних веб-сайти даного типу та буде проведено їх аналіз.

1.2 Аналоги веб-сайтів для пошуку репетиторів

Розглянемо три найпопулярніші сайти для пошуку репетиторів: Preply [3], Cerrera [4] та Buki [5].

Preply – це міжнародна платформа, яка дозволяє вивчати більше 45 іноземних мов з репетиторами з усього світу. Платформа зосереджена на мовних курсах, дозволяючи обирати викладачів на основі їхнього профілю, досвіду, цінового діапазону та відгуків інших студентів (рис. 1.2.1).

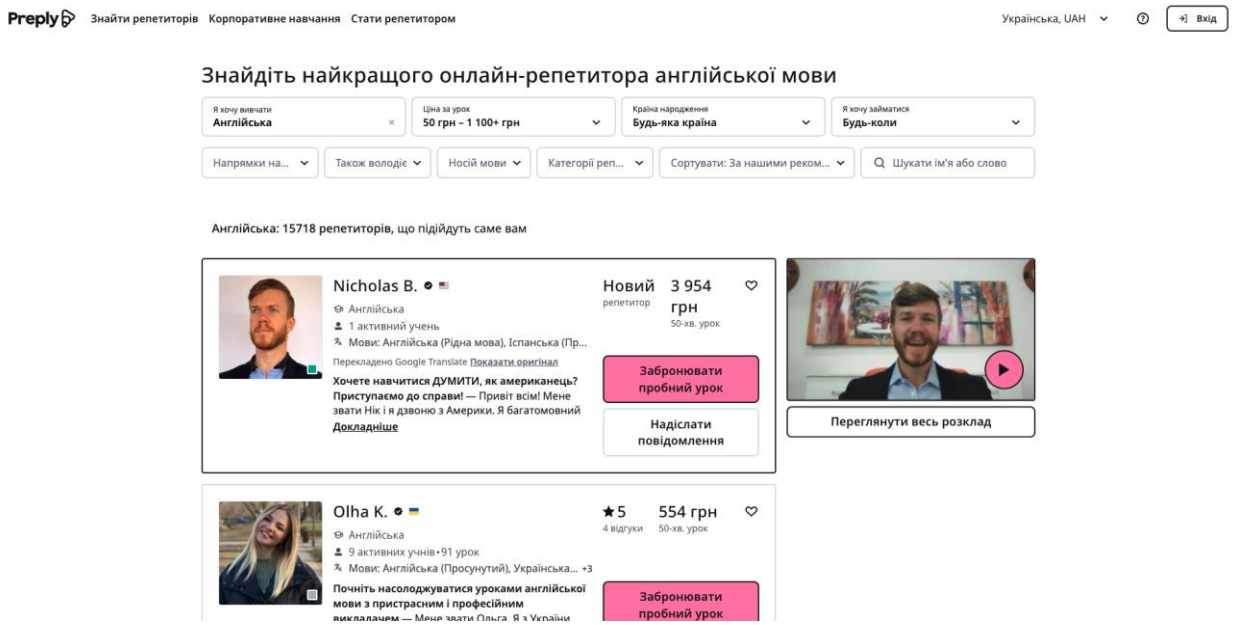

Рисунок 1.2.1 – Вигляд сторінки з пошуку репетитора платформи Preply

Головною перевагою даного ресурсу є широкий вибір мов та викладачів з можливістю обрати носія мови. Також сайт відзначається гнучкістю у плануванні занять.

Із недоліків можна відмітити, що основний акцент зроблено саме на мови, що може бути обмеженням для тих, хто шукає репетиторів з інших предметів. Також дуже часто цінник за заняття є доволі високим, але з цією проблемою гарно впорається фільтр ціни.

Cererra - українська платформа, яка пропонує широкий спектр освітніх послуг, не обмежуючись лише мовами. Вона забезпечує можливості для вивчення різноманітних предметів від математики до фітнесу (рис. 1.2.2).

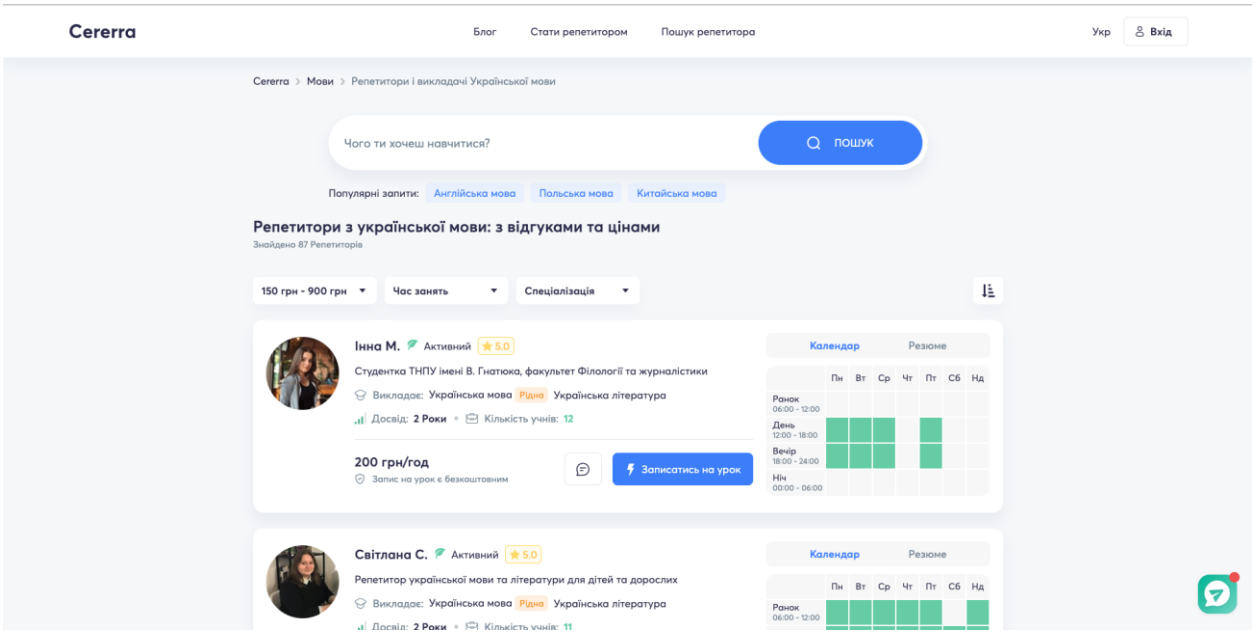

Рисунок 1.2.2 – Вигляд сторінки для пошуку репетитора платформи Cererra

Щодо переваг даного сайту, то це однозначно різноманітність предметів для вивчення. До того ж є функція яка дозволяє обрати між онлайн та офлайн навчанням. Також, варто зазначити, що це українська платформа, тому тут легше буде знайти репетитора для підготовки до НМТ та інших екзаменів на відмінну від міжнародних платформ.

Із недоліків можна відмітити, що там майже немає варіантів для вибору іноземних репетиторів порівняно з міжнародними платформами.

Buki – платформа, котра пропонує послуги репетиторів по всій Україні з більш ніж 100 предметами. Вона має одну з найбільших баз даних репетиторів у країні і забезпечує детальний пошук за містом, предметом, ціною та іншими параметрами (рис. 1.2.3).

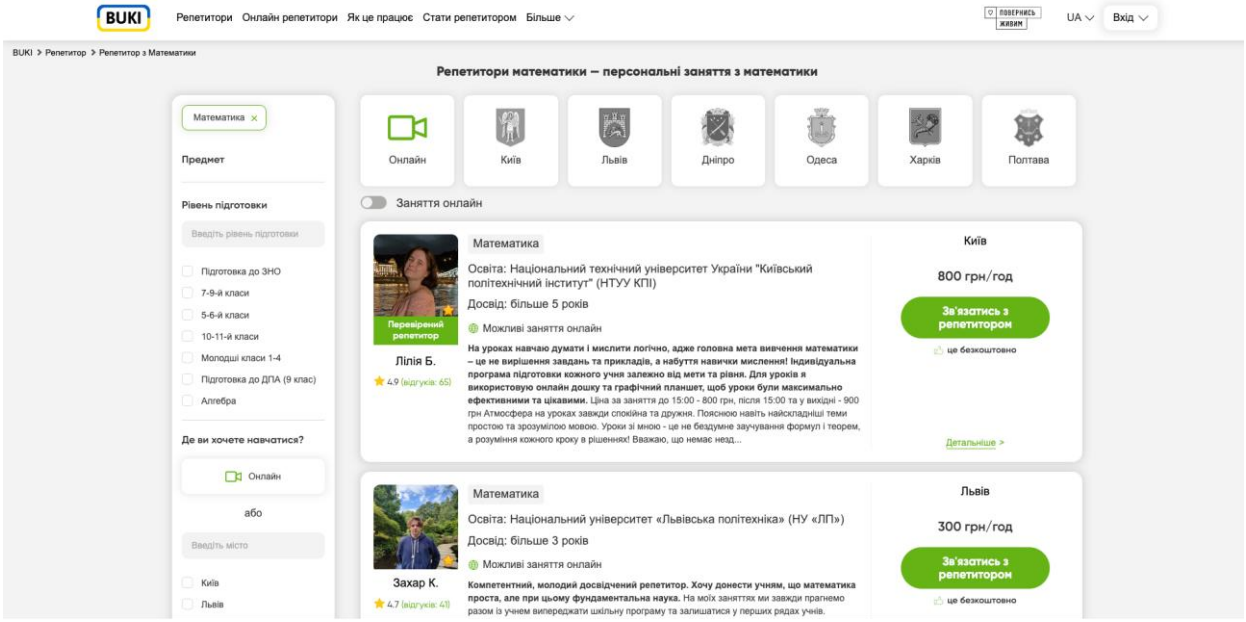

Рисунок 1.2.3 – Вигляд сторінки для пошуку репетитора платформи Buki

Перевагою даного ресурсу є велика кількість викладачів. Як і в Cererra тут також можна обрати між онлайн та офлайн навчанням, і в загальному дані платформи є дуже схожими між собою.

Слабкою стороною є відсутність іноземних репетиторів.

Що ж спільного мають дані три платформи?

- Можливість авторизації учня або вчителя. Є основою багатьох сайтів. За допомогою реєстрації репетитори можуть розмістити свої послуги, а учні зв’язатися з потрібним викладачем чи одразу записатися на заняття. Вигляд реєстрації на ресурсах Preply, Cererra, Buki вказано на рисунках 1.2.4-1.2.6 відповідно.

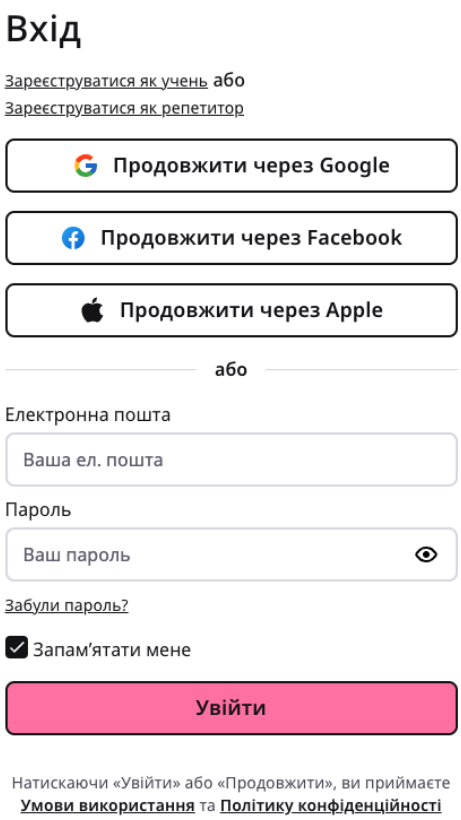

Рисунок 1.2.4 – Вигляд авторизації платформи Preply

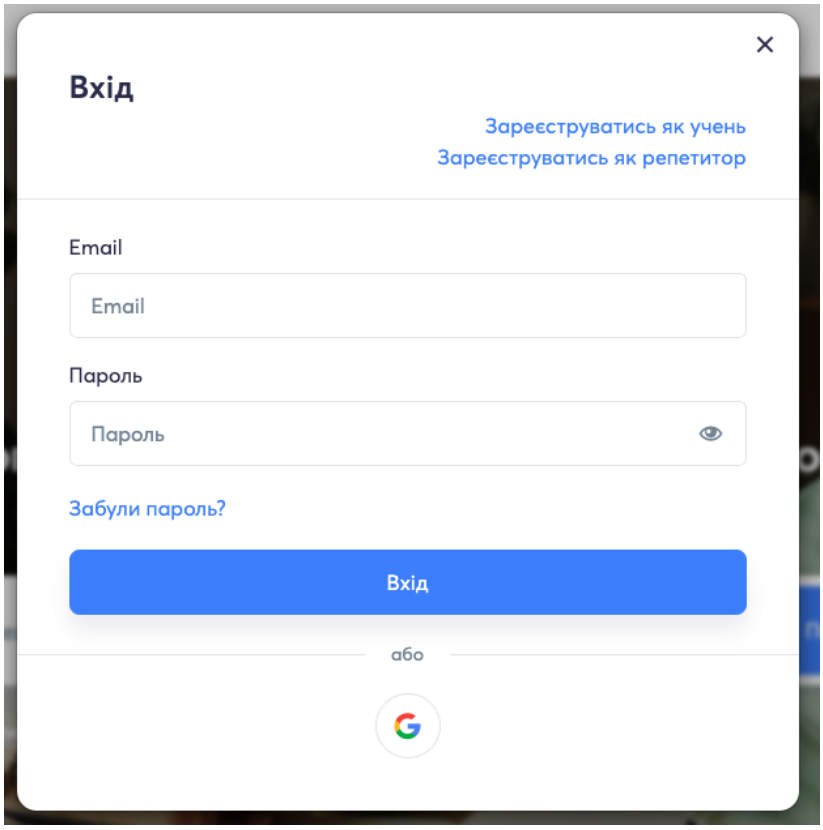

Рисунок 1.2.5 – Вигляд авторизації платформи Cererra

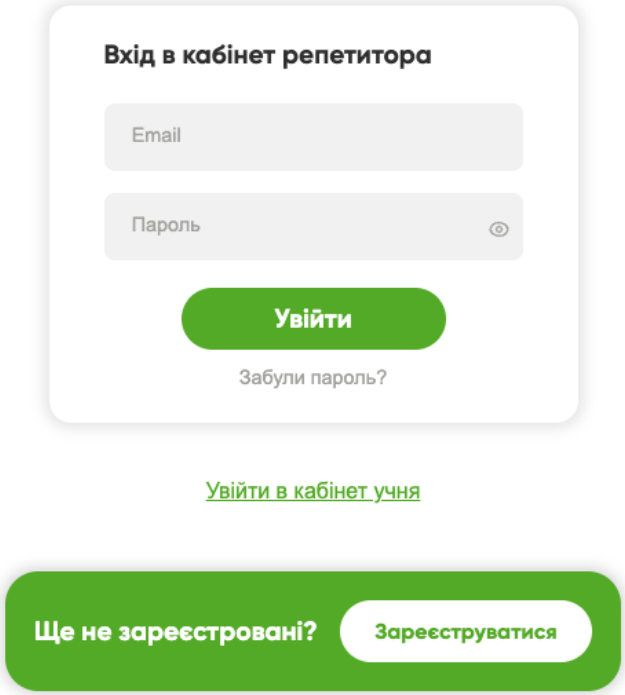

Рисунок 1.2.6 – Вигляд авторизації платформи Buki

- Фільтр для пошуку репетитора за предметом, ціною за урок та за іншими різноманітними параметрами. Допомагає знайти викладача згідно критеріям учня. Вигляд фільтрації на ресурсах Preply, Cererra, Buki вказано на рисунках 1.2.7-1.2.9 відповідно.

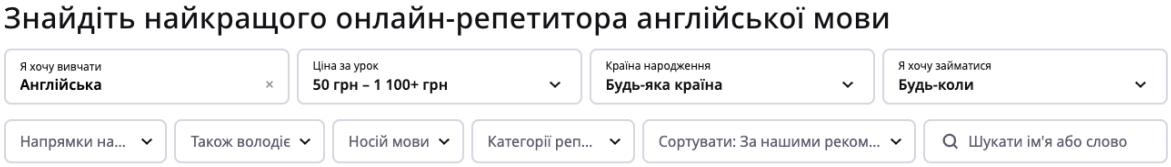

Рисунок 1.2.7 – Вигляд фільтру платформи Preply

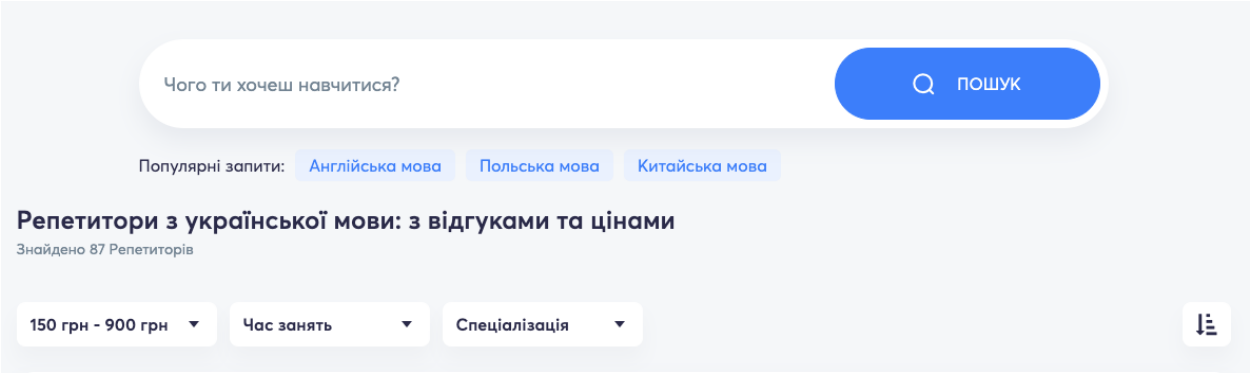

Рисунок 1.2.8 – Вигляд фільтру платформи Cererra

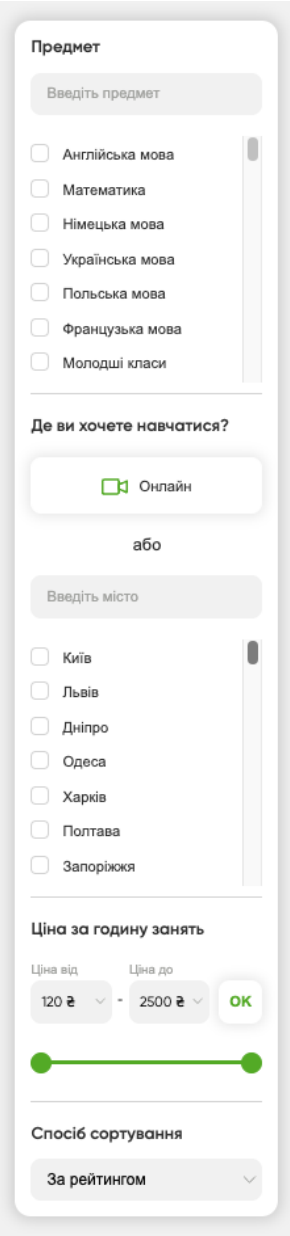

Рисунок 1.2.9 – Вигляд фільтру платформи Buki

- Рейтинг з відгуками. За допомогою рейтингу можна відсіяти некомпетентних викладачів, та, навпаки – знайти професіонального репетитора. Вигляд рейтингу з відгуками на ресурсах Preply, Cererra, Buki вказано на рисунках 1.2.10-1.2.12 відповідно.

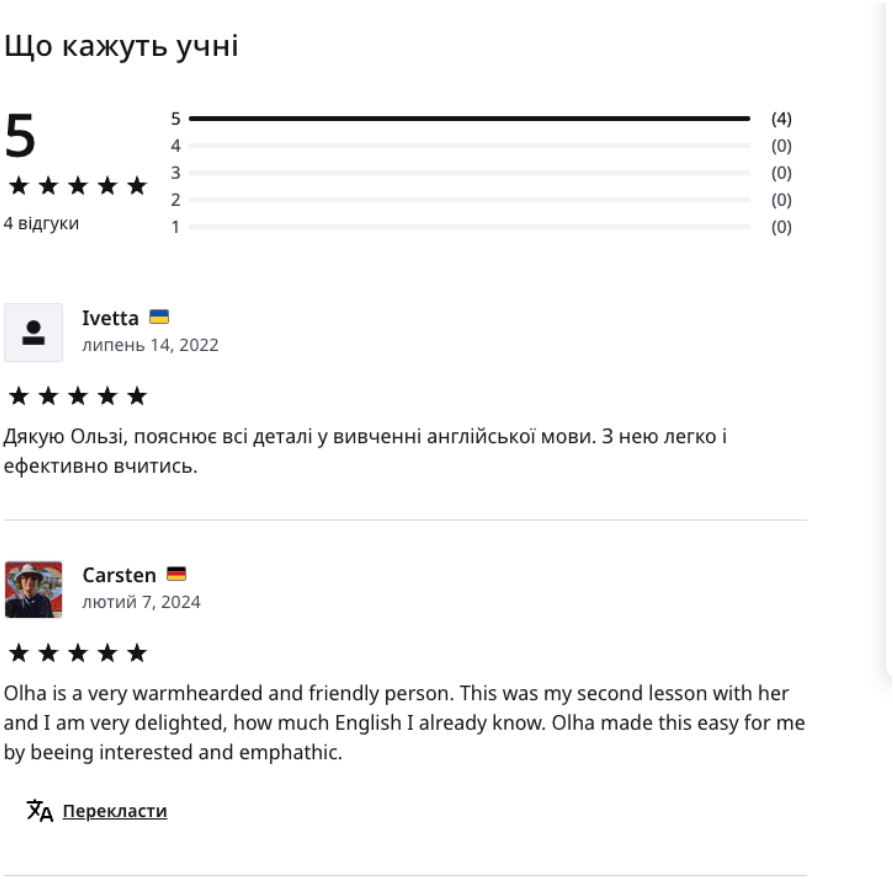

Рисунок 1.2.10 – Вигляд рейтингу платформи Preply

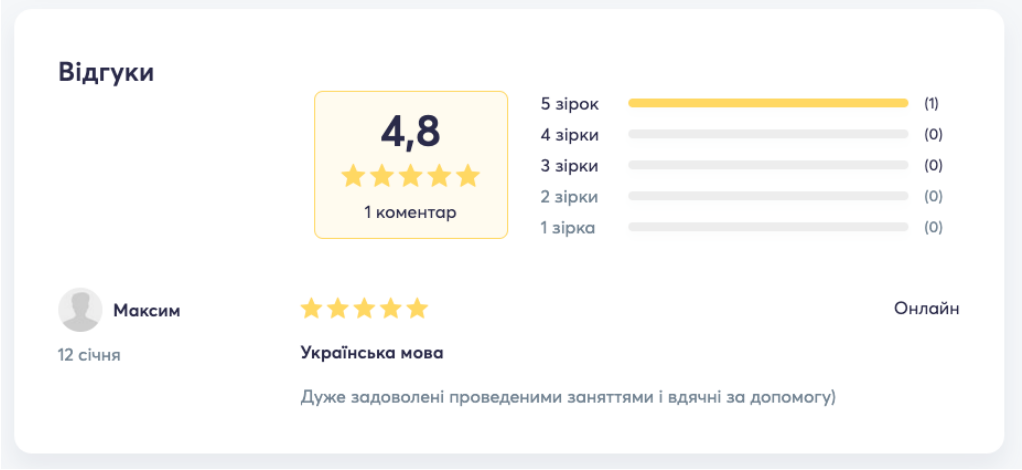

Рисунок 1.2.11 – Вигляд рейтингу платформи Cererra

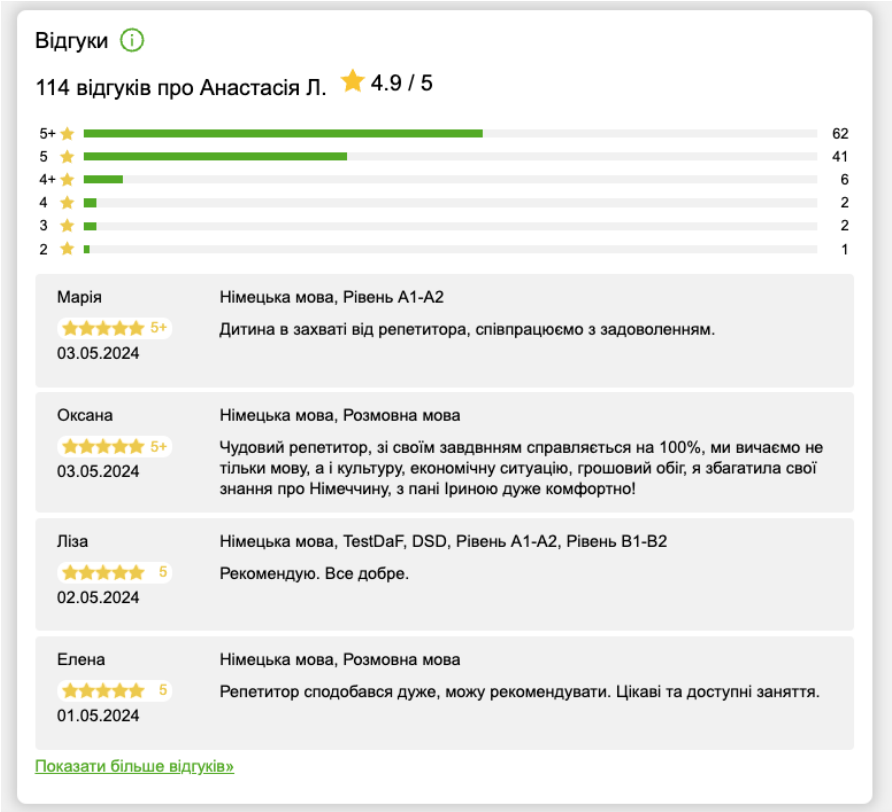

Рисунок 1.2.12 – Вигляд рейтингу платформи Buki

- Можливість переглядати розклад доступних уроків у репетитора та записатися на потрібний час. Даний функціонал допомагає уникнути витрати часу на знаходження репетитора зі зручним розкладом для учня. Вигляд перегляду розкладу на ресурсах Preply, Cererra, Buki вказано на рисунках 1.2.13-1.2.15 відповідно.

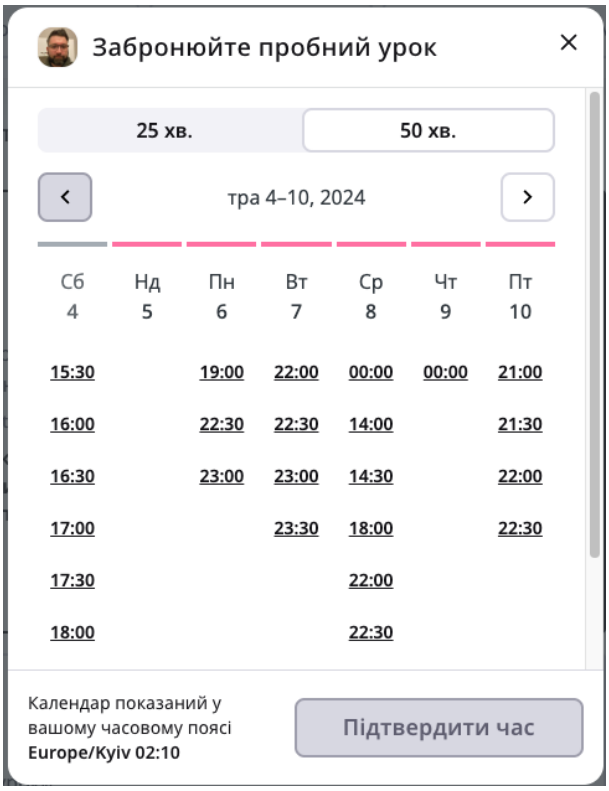

Рисунок 1.2.13 – Вигляд розкладу занять платформи Preply

Рисунок 1.2.14 – Вигляд розкладу занять платформи Cererra

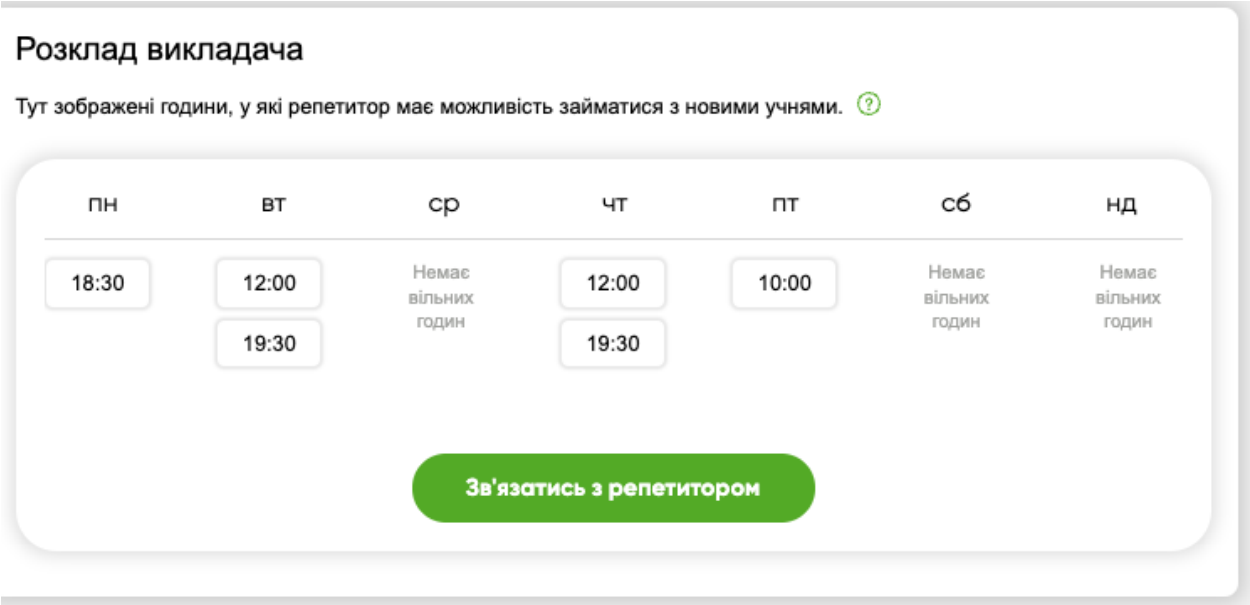

Рисунок 1.2.15 – Вигляд розкладу занять платформи Buki

Отже, кожна з цих платформ має свої особливості та переваги. Preply ідеально підходить для вивчення мов з носіями мови, Cererra надає широкий спектр предметів та гнучкість форматів навчання, а Buki вирізняється великою кількістю репетиторів і детальним пошуком по Україні.

1.3 Використання веб-технологій у навчальних платформах

Сучасні веб-технології широко використовуються для створення ефективних та зручних платформ для навчання та репетиторства. Їх використання дозволяє створювати веб-сервіси, які відповідають вимогам користувачів та забезпечують зручний і привабливий інтерфейс для взаємодії з платформою.

Ось деякі аспекти веб-технологій, які є популярними для навчальних платформ та в цілому у створенні сайтів:

Фронтенд технології: Фронтенд технології, такі як HTML, CSS, React та Angular, дозволяють розробляти динамічні та інтуїтивно зрозумілі інтерфейси. Завдяки цим технологіям користувачі можуть легко переміщатися по веб-сторінках, шукати викладачів або, навпаки, відстежувати свій прогрес у навчанні; реалізуючи інтерфейси за допомогою React JS, користувачі можуть

бачити зміни без необхідності перезавантажувати сторінку, тим самим покращувати свій користувацький досвід.

Серверні технології: серверні технології відіграють важливу роль у стабільній роботі платформи та обробці запитів користувачів. Дані технології включають в собі такі фреймворки, як NodeJS і Express.js, які використовуються для побудови бекенду веб-сайту. Сервер забезпечує надійну роботу платформи і виконує функції захисту даних, аутентифікації та авторизації користувачів. Вони також відповідають за управління базою даних та забезпечення ефективної взаємодії з нею.

Загалом, використання веб-технологій у навчальних платформах сприяє підвищенню якості освіти та робить її більш доступною для всіх, надаючи зручні та ефективні способи навчання, взаємодії з викладачами та вдосконалення навичок.

2 ВИБІР МЕТОДІВ РІШЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Визначення основних функцій веб-сайта

Для розв’язання поставленої задачі, а саме створення веб-платформи для надання послуг репетиторства, необхідно реалізувати такі основні функції:

- Авторизація репетиторів та студентів
- Відображення інформації про репетиторів, включаючи фотографію, Ім’я та фамілію, предмет викладання, ціну, досвід та опис
- Сторінка на якій репетитор може обрати зручний час для своїх занять, в той час як учні в подальшому можуть надіслати запит на заняття на обраний час
- Можливість сортування репетиторів по критеріям: предмет, ціна, досвід
- Можливість редагування профілю

2.2 Вибір програмних засобів реалізації

Для успішної реалізації веб-платформи для надання послуг репетиторства необхідно зробити важливий вибір програмних засобів. Цей вибір визначає технологічний стек, який використовуватиметься для розробки і підтримки проекту. Основні обрані програмні засоби включають:

1. React JS для розробки frontend-частини.

React - це "бібліотека" JavaScript. Це не зовсім "фреймворк". Він дуже часто потребує додаткового використання бібліотек.

Великі та малі компанії використовують його для створення своїх додатків. Популярність React пояснюється тим, що він базується на базових навичках веб-розробки. Проте, це не означає, що його можна вивчити за день чи те, що кожну функцію легко зрозуміти з першої спроби. Натомість, React має перевагу у тому, що мінімізує кількість необхідних специфічних знань. Отже, не потрібно вивчати шаблони, контролери чи складні патерни. Замість цього, більшість коду буде складатися з JavaScript у поєднанні зі стандартним HTML [6, с. 7].

React слідує філософії Unix, котру сформував Даг Макілрой: "Пишіть програми, які роблять щось одне і роблять це добре. Пишіть програми, які б працювали разом". Отже, React це невелика бібліотека, яка фокусується лише на одній речі, і робить її надзвичайно добре. Ця "одна річ" є частиною визначення React: створення користувацьких інтерфейсів.

Інтерфейс користувача (UI) - це все, що надається користувачам, щоб вони могли взаємодіяти з механізмом. UI є скрізь, від простих кнопок на мікрохвильовці до приладової панелі космічного шаттла. Якщо пристрій, з яким можна взаємодіяти, розуміє JavaScript, тоді React може бути використаний щоб описати інтерфейс для нього. Оскільки веб-браузери розуміють JavaScript, то React підходить для опису веб-інтерфейсів.

Долучно використовувати слово "описувати", тому що це те, що програмісти в основному роблять з React - просто говорять йому, що хочуть. Після цього React створить власний інтерфейс у веб-браузері. Без React або подібних бібліотек потрібно було б вручну створювати інтерфейс за допомогою нативних веб-аплікаторів та JavaScript, а це не так просто.

Коли ви чуєте твердження "React є декларативним", це саме те, що воно має на увазі. Програміст описує інтерфейси за допомогою React і говорить йому, що хоче (а не як це зробити). React подбає про "як" і перетворить декларативні описи (які пишуться мовою React) на реальні UI в браузері. React поділяє цю просту декларативну силу з самим HTML, але з React можна бути декларативними для HTML-інтерфейсів, які представляють динамічні дані, а не тільки статичні [7, с. 14].

2. Nest JS для розробки backend-частини.

NestJS - це фреймворк для створення ефективних, масштабованих серверних додатків Node.js. Він використовує прогресивний JavaScript, побудований на TypeScript і повністю підтримує його, але при цьому дозволяє розробникам кодувати на чистому JavaScript - і поєднує в собі елементи ООП (об'єктно-орієнтованого програмування), ОФП (функціонально-орієнтованого

програмування), ФП (функціональне програмування) та ФРП (функціонально-реактивне програмування) [8, с. 5].

NestJS є гарною відправною точкою для багатьох розробників, які хочуть використовувати сучасний веб-фреймворк. Багато розробників навчалися програмуванню на таких мовах, як Java або C/C++, які є строгими мовами, тому використання JavaScript може бути трохи незручним, і легко припуститися помилок через відсутність безпеки типів. Nest.js використовує TypeScript, який є золотою серединою. Ця мова поєднує в собі простоту та потужність JavaScript з безпекою типів інших мов, до яких більшість програмістів вже звикли. Безпека типів у Nest.js доступна лише під час компіляції, оскільки сервер Nest.js компілюється до сервера Node.js Express, який виконує JavaScript. Однак це все одно є великою перевагою, оскільки дозволяє краще розробляти програми без помилок щодо часу виконання.

Node.js має багату екосистему пакетів у NPM (Node Package Manager), яка налічує понад 350 000 пакетів і є найбільшим у світі реєстром пакетів. Завдяки тому, що Nest.js використовує Express, ви маєте доступ до кожного з цих пакунків при розробці Nest-додатків. Багато з них навіть мають визначення типів для своїх пакетів, що дозволяє IDE читати пакети і робити підказки/автозаповнення коду, які можуть бути неможливими при схрещуванні JavaScript-коду з кодом TypeScript. Однією з найбільших переваг Node.js є величезне сховище модулів, які можна використовувати замість того, щоб писати свої власні [9, с. 8].

3. PostgreSQL для зберігання даних про репетиторів.

PostgreSQL має велику аудиторію як серед шанувальників баз даних, так і серед розробників відкритого програмного забезпечення. Кожен, хто створює додаток з нетривіальними обсягами даних, може отримати вигоду від використання бази даних. PostgreSQL - це чудова реалізація реляційної бази даних, повнофункціональна, з відкритим вихідним кодом і безкоштовна у використанні. PostgreSQL можна використовувати практично з будь-якою мовою програмування, яку ви захочете C, C++, Perl, Python, Java, Tcl та PHP.

Вона дуже тісно пов'язана з галузевим стандарту для мов запитів, SQL92, і наразі впроваджує функції для підвищення відповідності до останньої версії цього стандарту [10, с. 1].

4. Docker для контейнеризації.

Згідно з даними IBM "Контейнеризація передбачає інкапсуляцію або пакування програмного коду та всіх його залежностей так, щоб він міг працювати рівномірно і послідовно на будь-якій інфраструктурі".

Іншими словами, контейнеризація дозволяє вам об'єднати ваше програмне забезпечення разом з усіма його залежностями в автономний пакет, щоб його можна було запустити, не проходячи через складний процес налаштування.

Docker – це платформа контейнеризації з відкритим вихідним кодом, яка дозволяє контейнеризувати ваші додатки, ділитися ними за допомогою публічних або приватних реєстрів. Дана платформа не єдиний інструмент контейнеризації на ринку, але він являється найпопулярнішим [11].

Docker був започаткований як внутрішній проект засновника dotCloud Соломона Хайкса. Він був випущений з відкритим вихідним кодом у березні 2013 року під ліцензією Apache 2.0. Маючи досвід роботи з платформою dotCloud як сервісом, засновники та інженери Docker усвідомлювали виклики, пов'язані з роботою з контейнерами. Тому в Docker вони розробили стандартний спосіб керування контейнерами. Docker використовує базові функції ядра операційної системи, які уможливлюють контейнеризацію [12].

5. Material UI для розробки інтерфейсів користувача.

Material UI – це бібліотека React-компонентів з відкритим вихідним кодом, яка реалізує Material Design від Google. Вона включає в себе повну колекцію готових компонентів, готових до використання у виробництві прямо з коробки, і має набір опцій кастомізації, які дозволяють легко реалізувати вашу власну систему дизайну на основі компонентів Material UI [13].

Використання Material UI може значно прискорити процес розробки, оскільки вона усуває необхідність створювати звичайні компоненти додатка з

нуля. Завдяки вже готовим компонентам, Material UI значно скорочує час, необхідний для розробки проектів, дозволяючи розробникам зосередитися на більш складних аспектах додатку. Також, вона добре інтегрується з іншими бібліотеками та фреймворками, що робить її універсальним вибором для багатьох веб-проектів.

6. Swagger для взаємодії з API через графічний інтерфейс.

Специфікація OpenAPI (OAS) визначає стандартний, мовно-діагностичний інтерфейс для HTTP API, який дозволяє як людям, так і комп'ютерам виявляти і розуміти можливості сервісу без доступу до вихідного коду, документації або через перевірку мережевого трафіку. При правильному визначенні, споживач може розуміти і взаємодіяти з віддаленим сервісом з мінімальною кількістю логіки реалізації. Визначення OpenAPI може використовуватися інструментами генерації документації для відображення API, інструментами генерації коду для створення серверів і клієнтів на різних мовах програмування, інструментами тестування та багатьма іншими випадками використання [14].

Swagger – це набір правил/специфікацій для формату, що описує REST API. Він забезпечує потужну екосистему інструментів навколо цієї формальної специфікації, таких як генератори та редактори коду, що активно розвиваються. Найкраща частина Swagger полягає в тому, що документація методів, параметрів і моделей тісно інтегрована в код сервера, що дозволяє API завжди залишатися синхронізованими [15].

Отже, обраний технологічний стек забезпечує високу продуктивність, масштабованість та стабільність веб-платформи. Він також дозволяє розробляти додаток з урахуванням сучасних та найкращих практик у галузі розробки веб-додатків. Цей вибір сприяє успішному втіленню ідеї проекту і задоволенню потреб користувачів у пошуку та наданні репетиторських послуг.

3 ДИЗАЙН МАЙБУТНЬОГО ДОДАТКУ

Перед тим як приступити до написання коду дуже важливою частиною є створення дизайну, це полегшить подальше розуміння загальної структури та взаємодії компонентів програми. Такий підхід дозволяє уникнути потенційних помилок та неузгодженостей у функціоналі продукту на більш пізніх етапах розробки.

Для створення дизайну було використано Figma — популярний онлайн інструмент для інтерфейсного дизайну та прототипування, який використовується дизайнерами по всьому світу для створення графічних інтерфейсів веб-сайтів та мобільних додатків. Однією з ключових особливостей Figma є те, що вона базується на хмарних технологіях, що дозволяє декільком користувачам одночасно працювати над одним проектом в реальному часі. Це сприяє співпраці та взаємодії команди, а також покращує загальний процес дизайну. Також, оскільки Figma є веб-базованою платформою, вона дозволяє користувачам працювати з будь-якого пристрою з інтернет-з’єднанням, без необхідності встановлювати складне програмне забезпечення.

Ознайомитись з повним дизайном можна за посиланням у списку літератури [16].

3.1 Головна сторінка

Головна сторінка сайту TutorMatch представляє собою інтерфейс для пошуку репетиторів, що дозволяє користувачам знайти ідеального викладача за різними параметрами. Дизайн сторінки чистий і сучасний, орієнтований на легкість використання і ефективність (рис. 3.1.1).

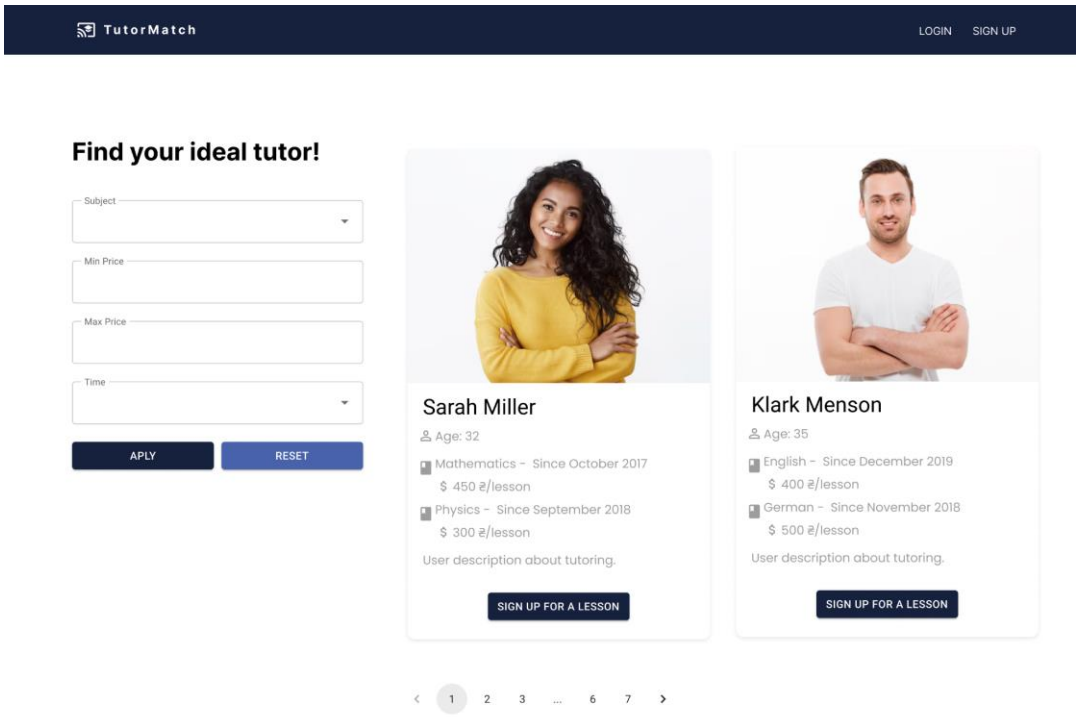

Рисунок 3.1.1 – Вигляд головної сторінки

Як можна побачити, верхня частина сторінки містить навігаційну панель із логотипом TutorMatch, а також кнопки входу та реєстрації для користувачів.

Основна частина сторінки включає панель для фільтрації, де можна обрати предмет, мінімальну і максимальну ціну за урок, а також зручний проміжок часу для занять. Кнопки "Apply" та "Reset" допомагають застосувати або скинути введені критерії пошуку.

Під пошуковою панеллю розміщені профілі репетиторів, кожен з яких містить фотографію, ім'я, вік, вартість занять і короткий опис досвіду. Кнопка "Sign Up for a Lesson" дозволяє зареєструватися на заняття безпосередньо з головної сторінки.

Також, варто зазначити, що нижня частина сторінки містить пагінацію, що дозволяє переглядати додаткові сторінки з профілями репетиторів.

Коли користувач натискає на кнопку для запису на урок, з'являється модальне вікно, яке дозволяє вибрати дату та час та предмет, якщо у репетитора їх декілька. Таким чином студент надсилає запит на заняття у викладача який йому найбільше підійшов по критеріям пошуку (рис. 3.1.2-3.1.4).

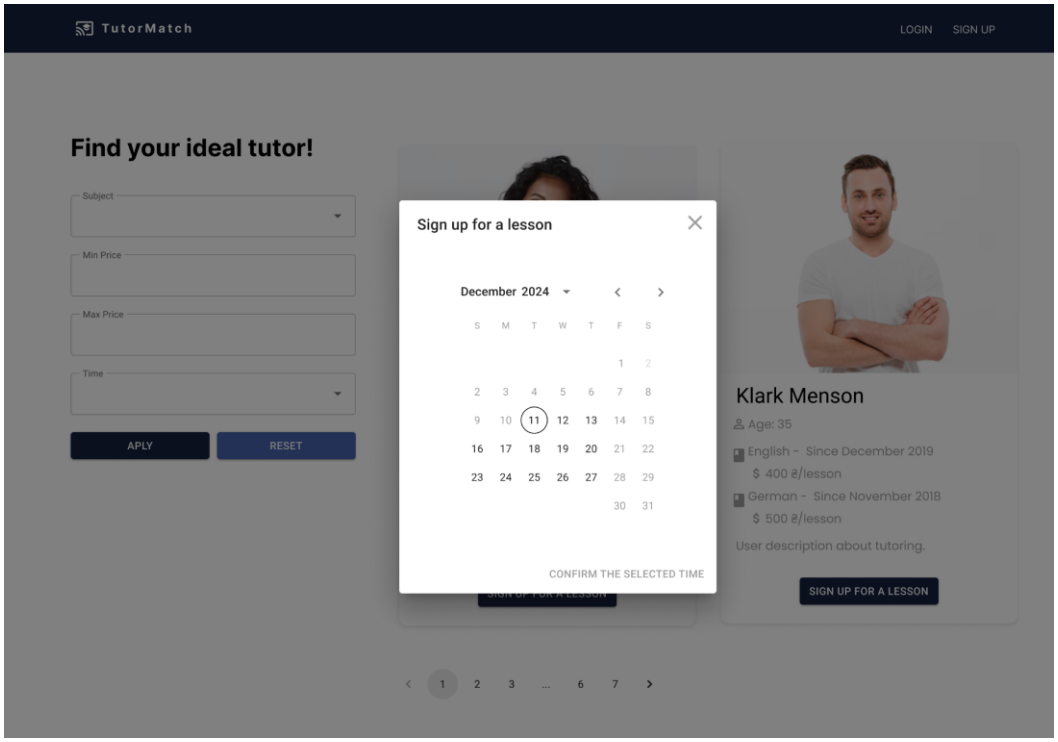

Рисунок 3.1.2 – Запис на заняття, вибір дати

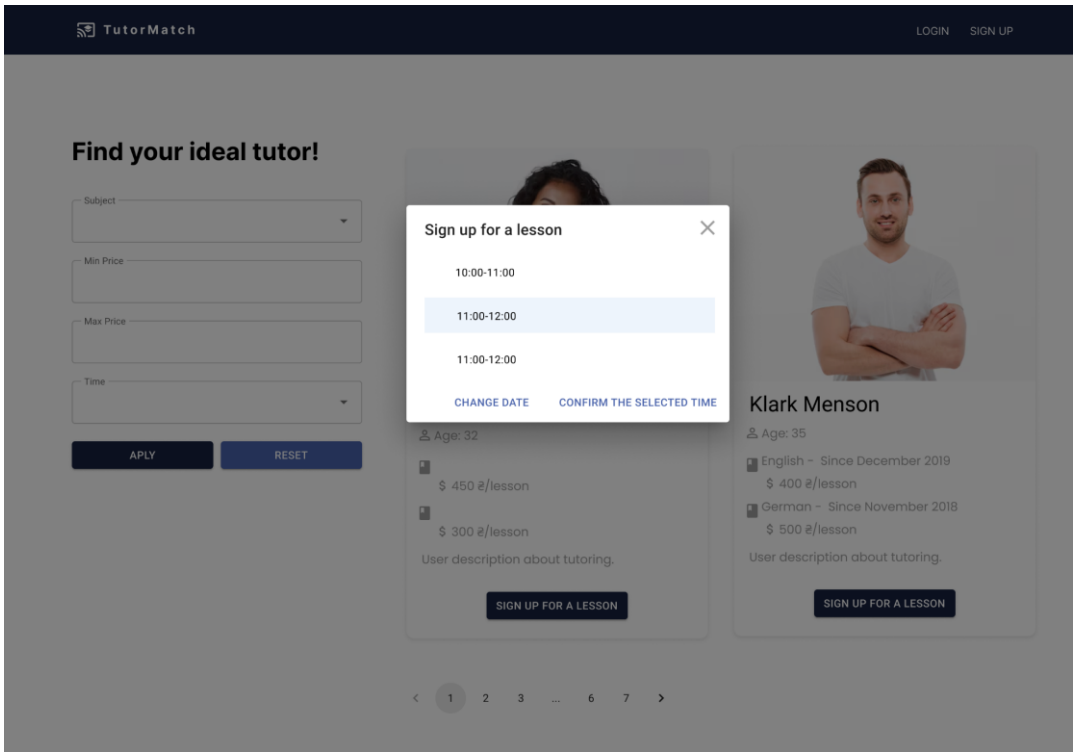

Рисунок 3.1.3 – Запис на заняття, вибір часу

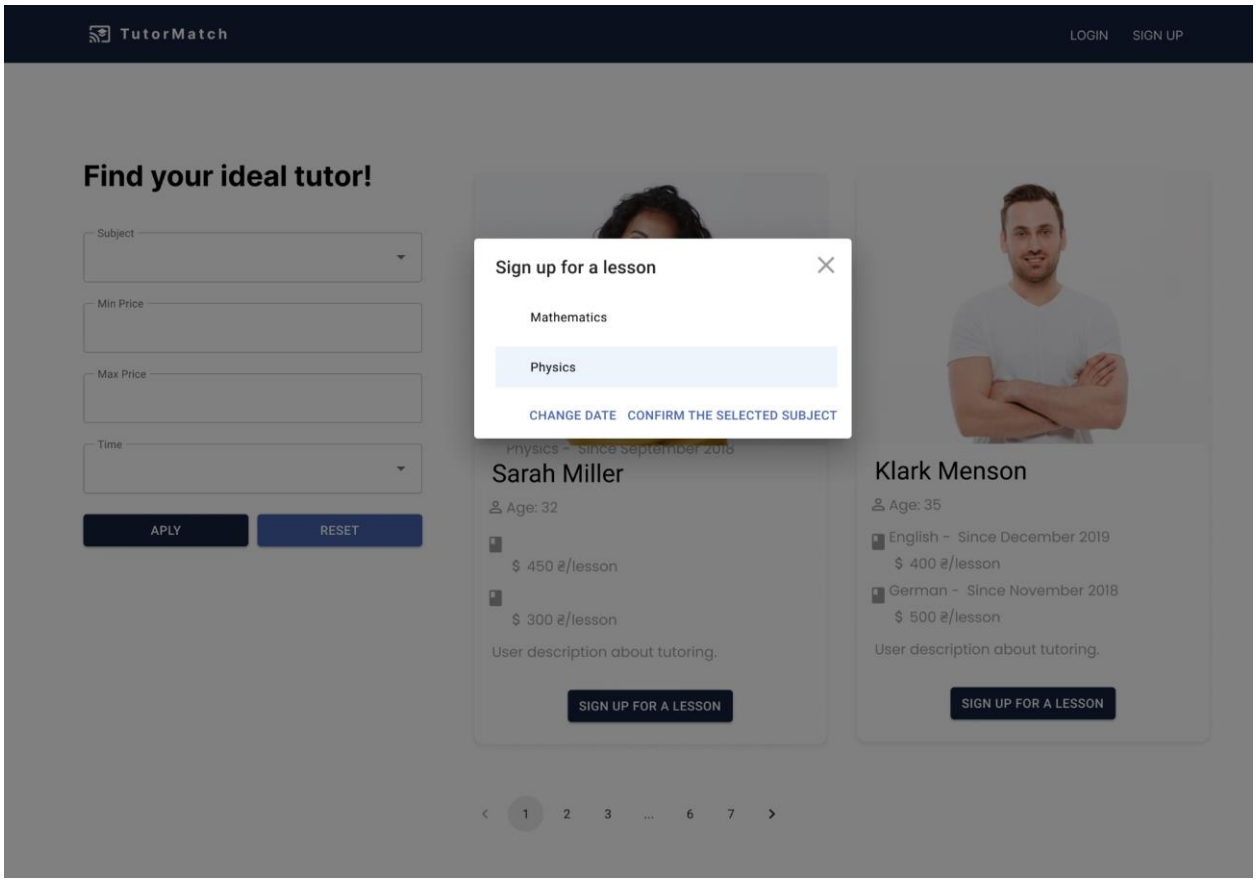

Рисунок 3.1.4 – Запис на заняття, вибір предмету

3.2 Сторінка входу

Сторінка входу на сайті TutorMatch має доволі простий і зрозумілий дизайн, який забезпечує чітку та ефективну взаємодію з користувачем (рис. 3.2.1).

Поле для введення електронної пошти має стандартний розмір і чітко вказує місце для введення.

Поле для пароля містить іконку для перемикання видимості пароля, що дозволяє користувачам контролювати відображення введених символів (рис. 3.2.2).

У випадку введення некоректної електронної адреси або пропуску поля для пароля, система відображає червоні підказки (рис. 3.2.3). Це допомагає користувачам швидко ідентифікувати проблеми і виправити їх без затримки.

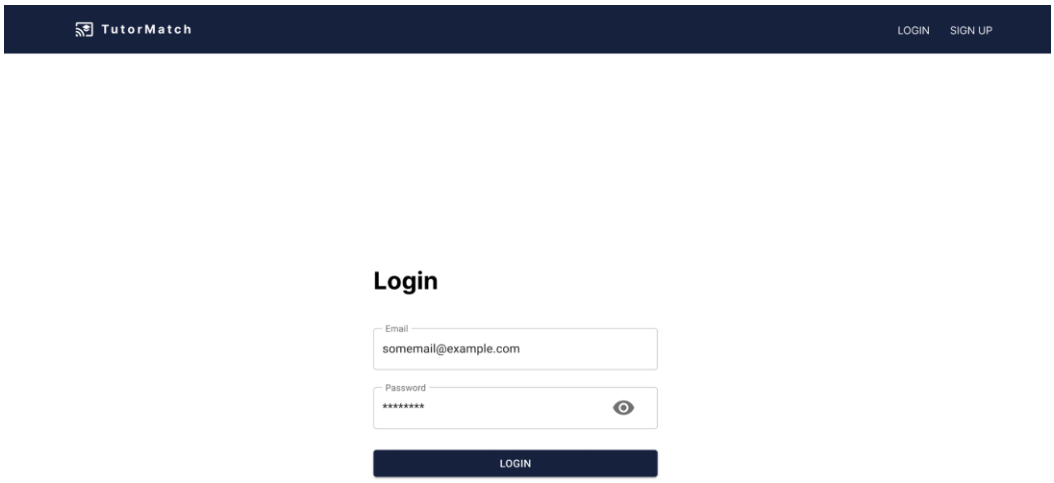

Рисунок 3.2.1 – Вигляд сторінки входу

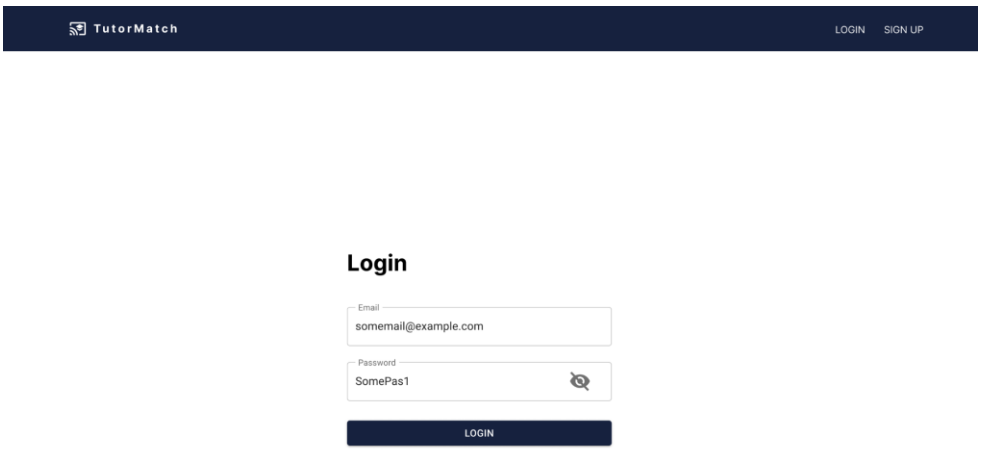

Рисунок 3.2.2 – Сторінка входу, відображення паролю

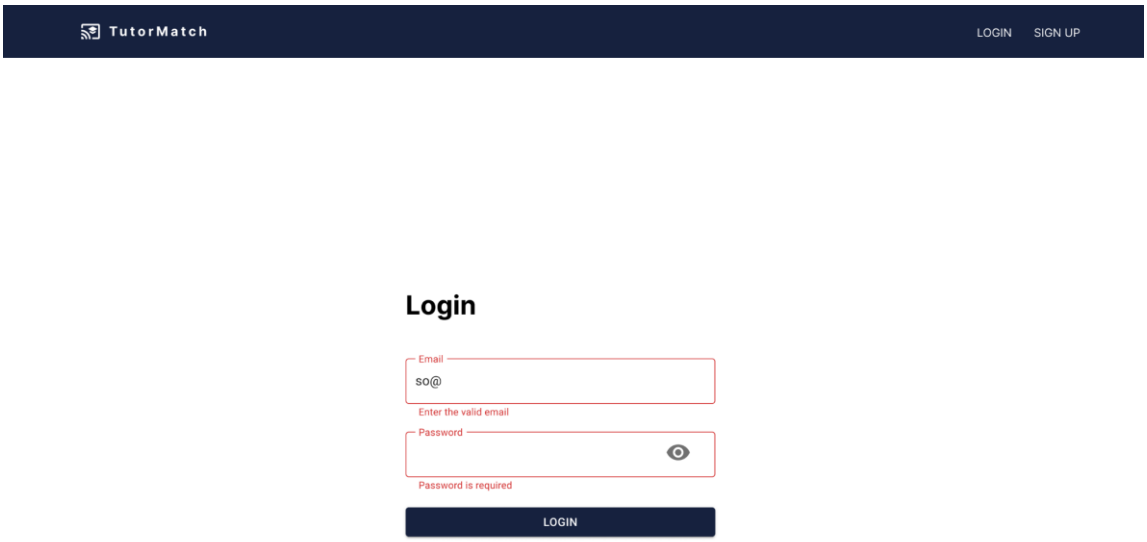

Рисунок 3.2.3 – Сторінка входу, валідація

3.3 Сторінка реєстрації

Сторінка реєстрації на сайті TutorMatch є частиною юзер-інтерфейсу, яка дозволяє користувачам реєструватися як студентам або репетиторам.

Перш за все, випадаюче меню дозволяє вибрати роль: "Student" або "Tutor", змінюючи форму відповідно до обраної ролі, вище даного меню знаходиться кнопка за допомогою якої можна завантажити фотографію.

Основними полями є ім'я, прізвище, дата народження, контакт для зв'язку котрий потрібен для подальшого зв’язку учня з вчителем, опис, електронна адреса та пароль (рис. 3.3.1).

Для репетиторів є додаткові поля, такі як: предмет, досвід у предметі, вартість за урок. Репетитори також мають можливість додати кілька предметів, якщо ж їх більше ніж один, то відповідно є кнопка для його або їх видалення, яка стає активною (рис. 3.3.2).

Всі поля мають чіткі місця для введення інформації з відповідними підказками в межах поля.

Форми включають валідацію на стороні клієнта, що вказує на помилки до відправлення форми, наприклад, необхідність введення правильної електронної адреси. Даний дизайн не відрізняється від валідації при вході.

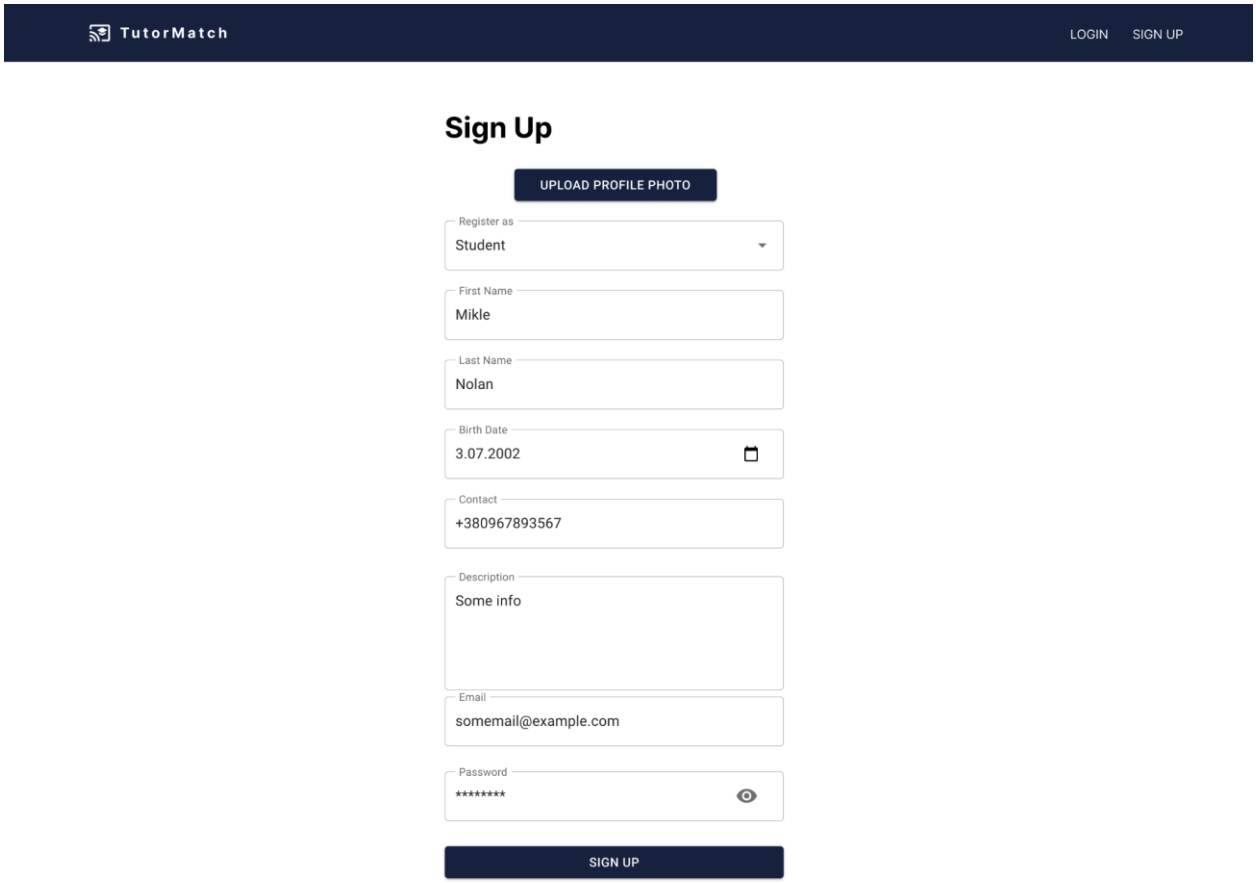

Рисунок 3.3.1 – Сторінка реєстрації в ролі студента

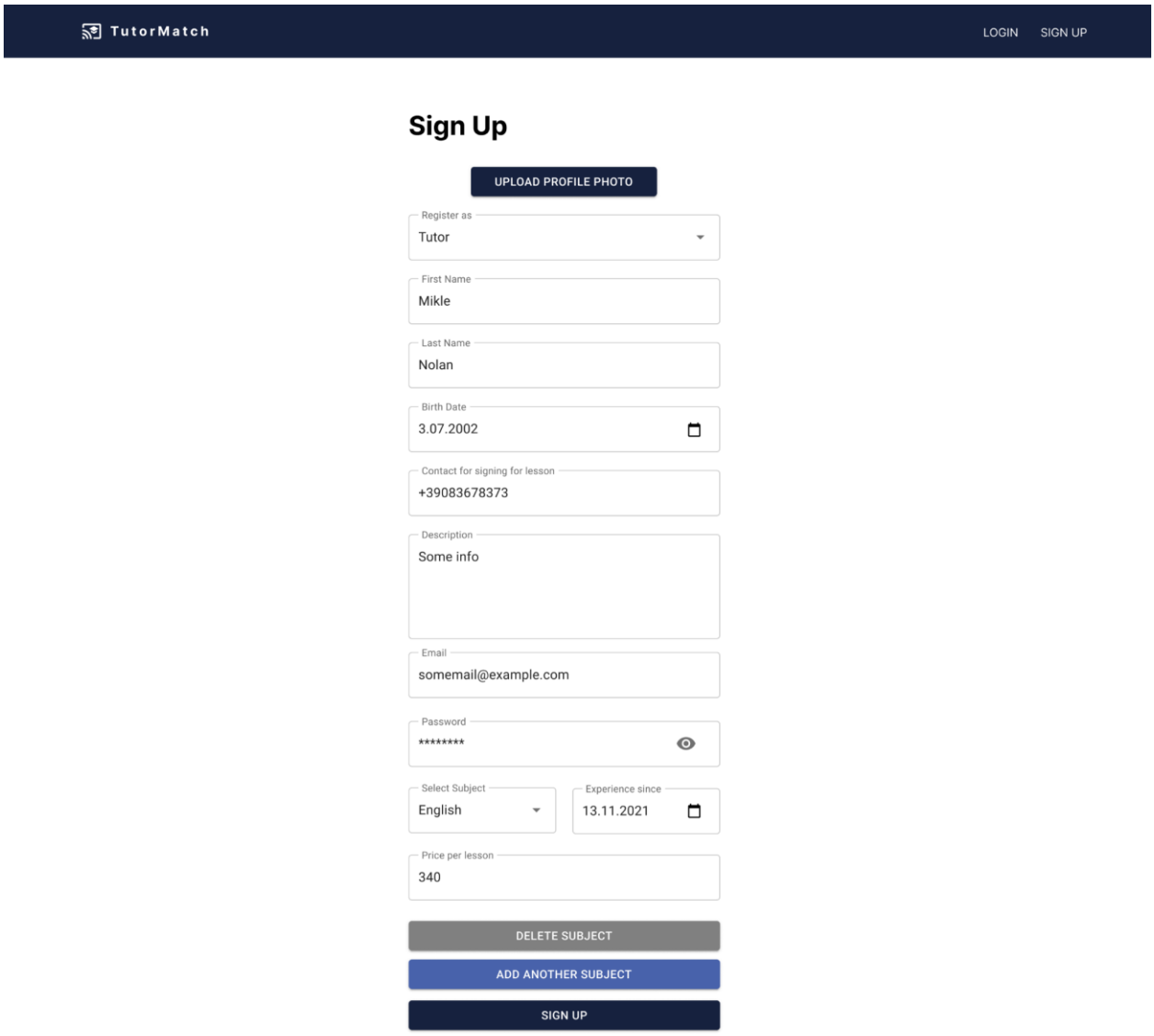

Рисунок 3.3.2 – Сторінка реєстрації в ролі репетитора

3.4 Сторінка розкладу

Сторінка "Work hours" на сайті TutorMatch призначена для налаштування робочих годин для репетиторів. Це дозволяє їм вказати свою доступність щодо графіку проведення занять (рис. 3.4.1).

Дні тижня відображені у вертикальному списку, з понеділка по п'ятницю, з чекбоксами для вибору активності в кожен конкретний день.

Кожен день має випадаючі меню для встановлення часу початку і закінчення роботи.

Чекбокси дозволяють легко вказати, чи є репетитор доступним в певний день.

Випадаючі меню для встановлення часу початку та закінчення робочого дня розташовані поряд з кожним днем, забезпечуючи зручність у налаштуванні графіка.

Також, варто зазначити, що через авторизацію кнопки навігації змінились на "Schedule", "Lessons" і "Log Out", вони знаходяться у верхній частині сторінки, даючи легкий доступ до інших секцій сайту.

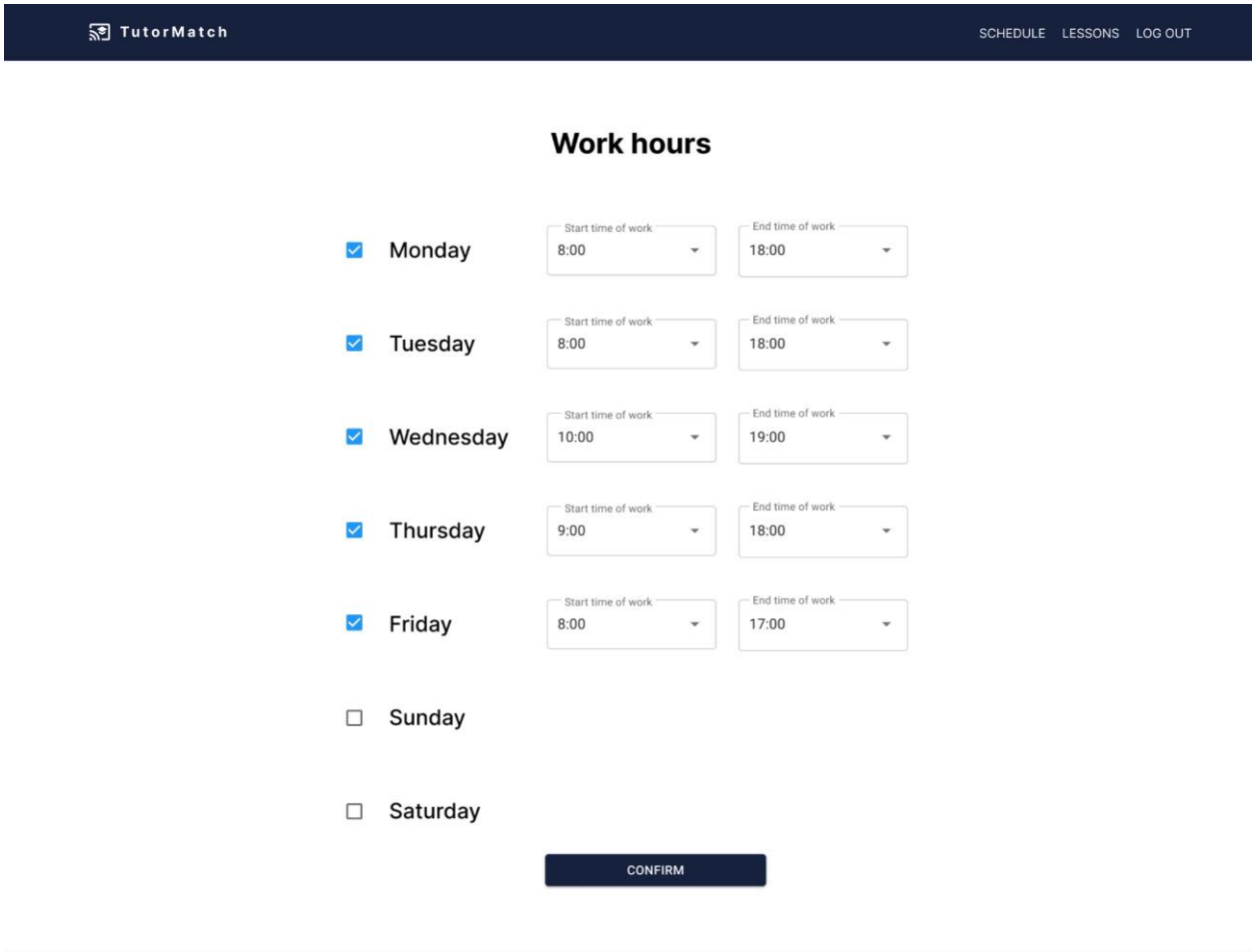

Рисунок 3.4.1 – Сторінка розкладу репетиторів

3.5 Сторінка занять

Сторінка "Lessons" на сайті TutorMatch дозволяє користувачам переглядати вже заплановані уроки та запити на них (рис. 3.5.1), а репетиторам додатково приймати запити на уроки (рис. 3.5.2).

Для юзера сторінка розділена на дві секції: "Lessons request" для запитів на уроки, які ще не є підтвердженими, та "Lessons" для вже підтверджених уроків.

Кожен запит на урок містить інформацію про предмет, дату, ім'я студента, вік, контактну інформацію, а також кнопки "Accept" та "Cancel" для швидкого реагування. У випадку студента вона містить ту ж інформацію про вчителя якому було надіслано запит та відстутня кнопка "Accept" відповідно.

Вже підтверджені уроки відображаються з інформацією про предмет, дату, час, ім'я студента/репетитора, вік, контактну інформацію, а також кнопкою "Cancel" для скасування уроку.

Ця сторінка відіграє важливу роль у забезпеченні ефективного управління часом та заняттями для репетиторів та учнів, дозволяючи їм з легкістю адаптуватися до запланованих та потенційних уроків.

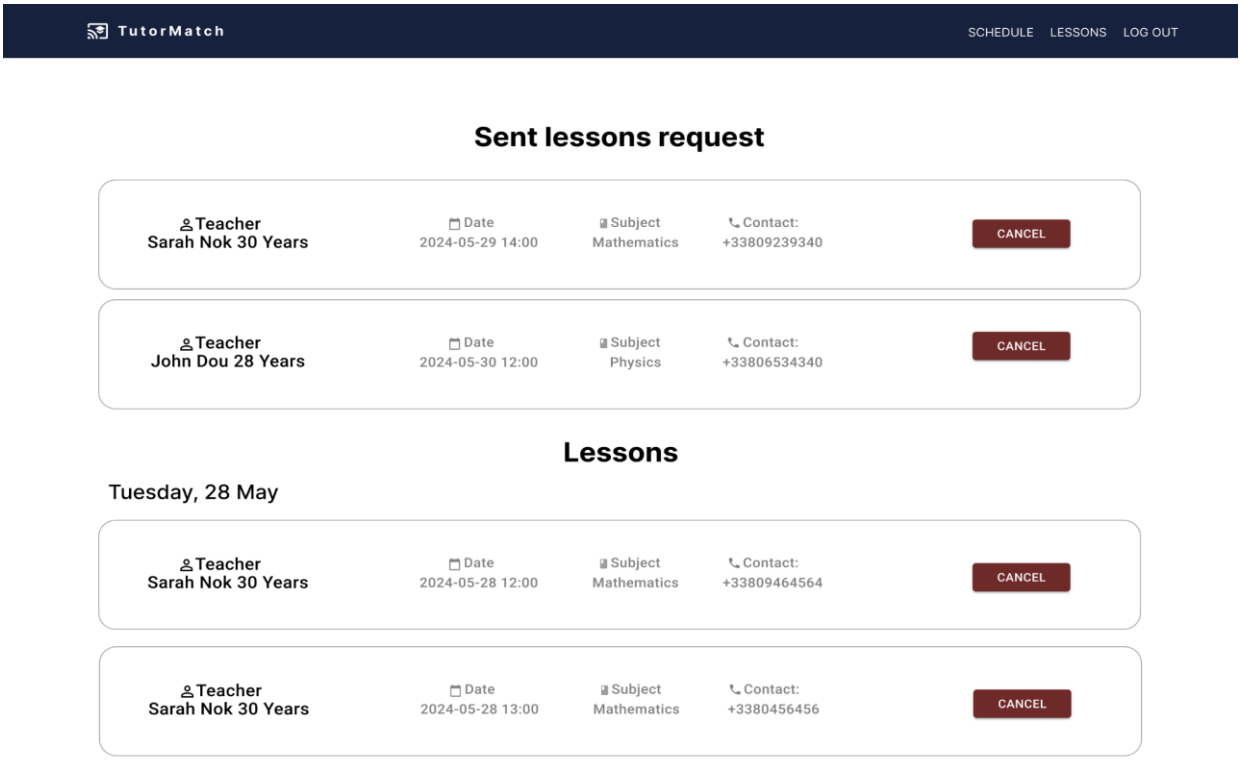

Рисунок 3.5.1 – Сторінка занять учнів

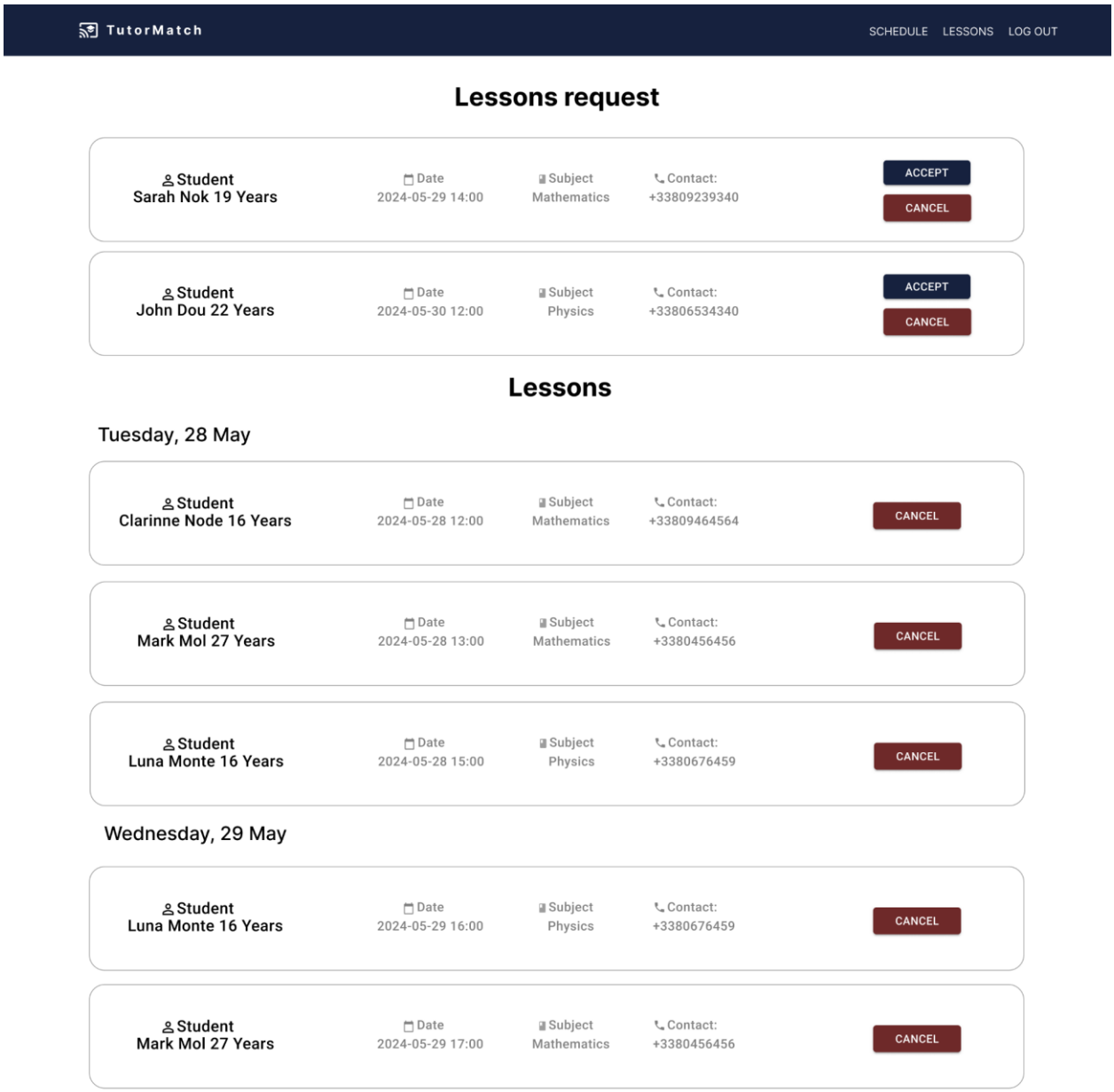

Рисунок 3.5.2 – Сторінка занять репетиторів

3.6 Сторінка профілю

Сторінка "Profile" на сайті TutorMatch надає користувачам можливість перегляду та редагування своїх особистих даних і предметів у випадку репетитора, які вони викладають. Для користувачів сторінка включає основну інформацію, таку як дата народження, електронна адреса, контактний номер телефону, а також кнопки для редагування профілю чи його видалення (рис. 3.6.1). Для репетиторів, крім основної інформації, присутня секція "Subjects", де відображені предмети, які вони викладають, з

можливістю редагування або додавання нових предметів (рис. 3.6.2). Вона включає інформацію про ціну за урок і дату початку викладання предмету. Така структура сторінки забезпечує легкий доступ до редагування освітнього профілю.

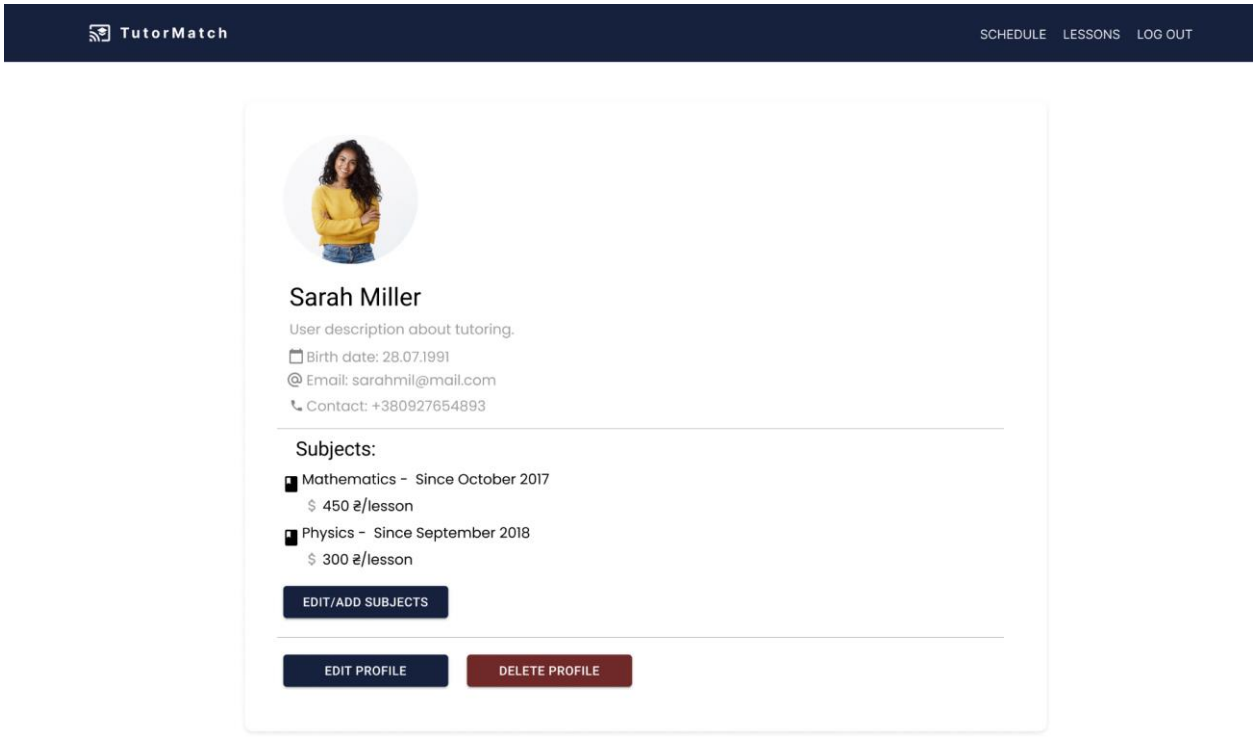

Рисунок 3.6.1 – Сторінка профілю репетитора

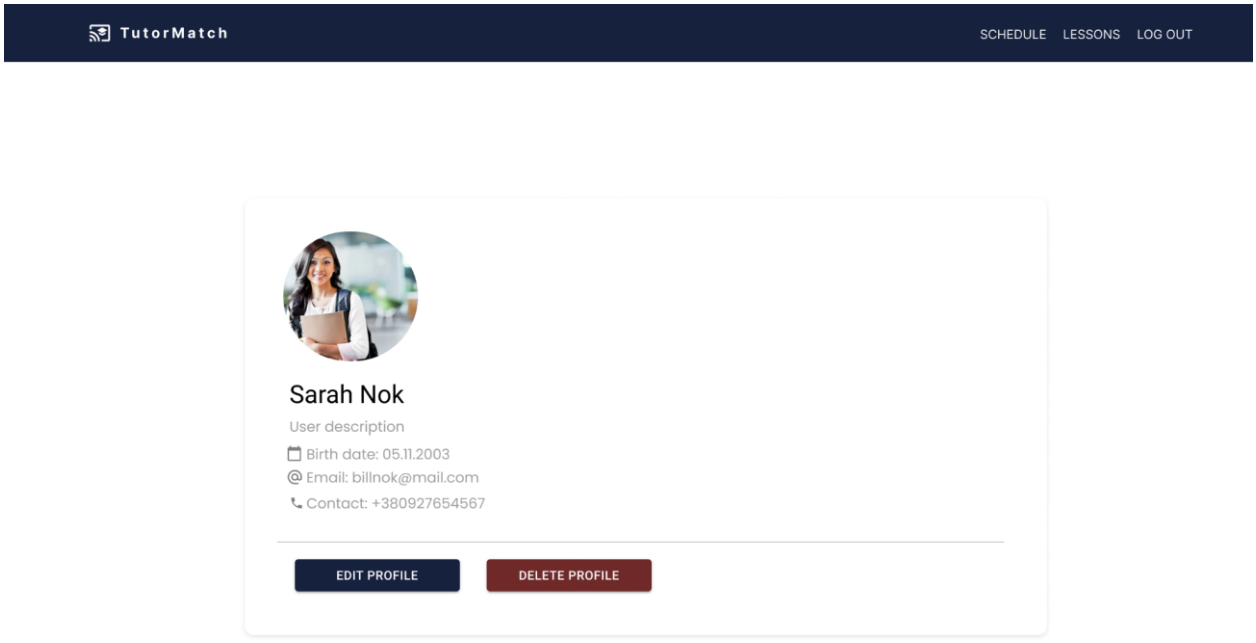

Рисунок 3.6.2 – Сторінка профілю студента

3.7 Сторінка редагування профілю

Сторінка "Edit Profile" на веб-сайті TutorMatch забезпечує можливість перегляду та подальшого редагування даних користувача (рис. 3.7.1). Юзер має можливість змінити свої особисті дані, такі як ім'я, прізвище, дату народження, контактні дані, електронну пошту та пароль (рис. 3.7.2). Також є опції для оновлення фотографії профілю або її видалення. Це дає змогу користувачеві підтримувати актуальність своєї інформації для потенційних учнів або репетиторів.

Сторінка "Edit/Add Subjects" забезпечує функціональність редагування та додавання предметів, які репетитор викладає (рис. 3.7.3). Користувач може вказати предмет, досвід викладання починаючи з конкретної дати та вартість одного уроку. Це дозволяє репетиторам ефективно адмініструвати свої предмети та адаптувати їх до потреб учнів. Опції додавання нового предмету та видалення існуючого розширюють можливості для налаштування та гнучкості управління своїм профілем (рис. 3.7.4).

Ці сторінки відіграють ключову роль у процесі управління особистими та професійними даними користувачів на платформі, забезпечуючи їм зручний спосіб оновлення інформації, що сприяє кращій організації та представленню їхніх послуг потенційним клієнтам.

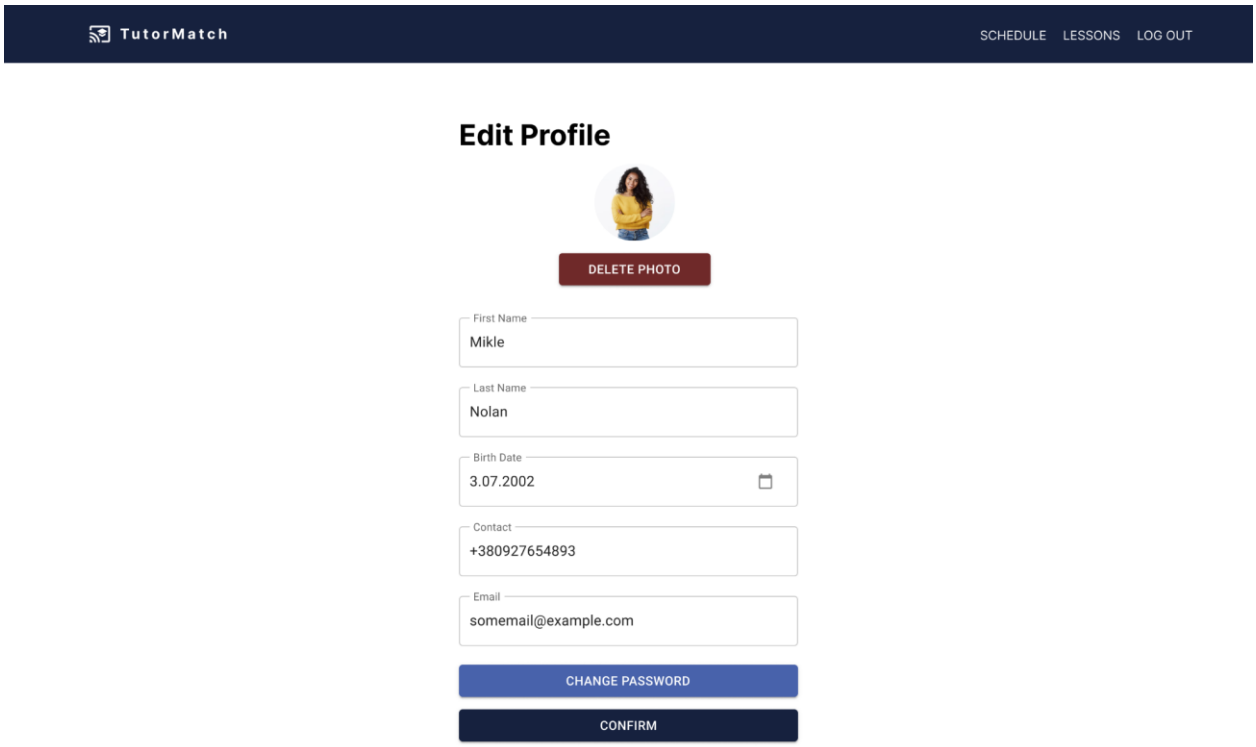

Рисунок 3.7.1 – Сторінка редагування профілю

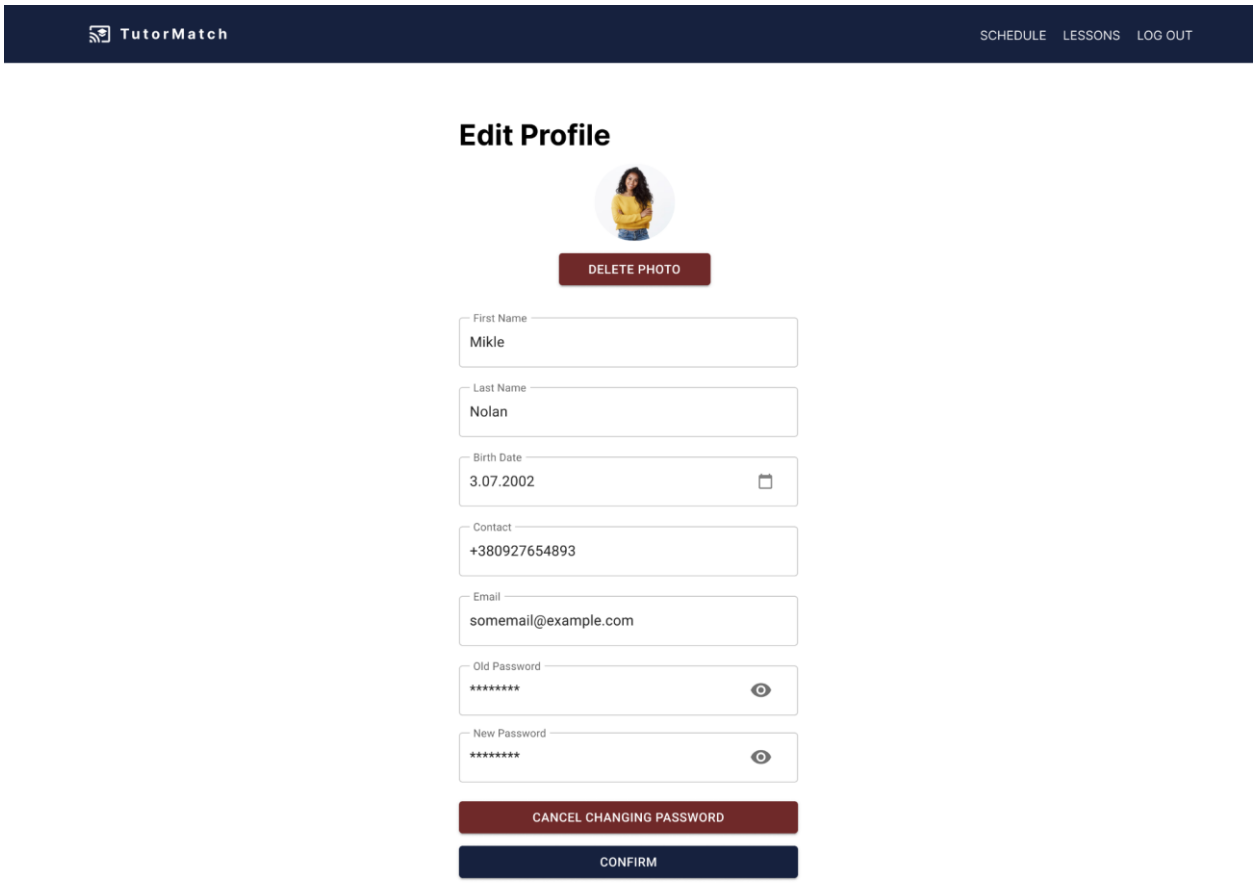

Рисунок 3.7.2 – Сторінка редагування профілю
зміна паролю

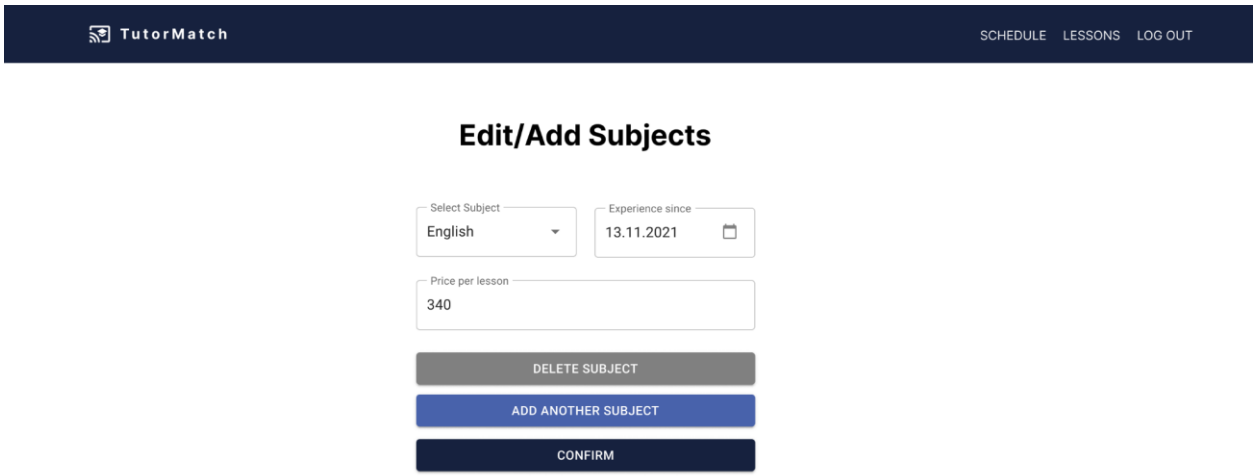

Рисунок 3.7.3 – Сторінка редагування предметів

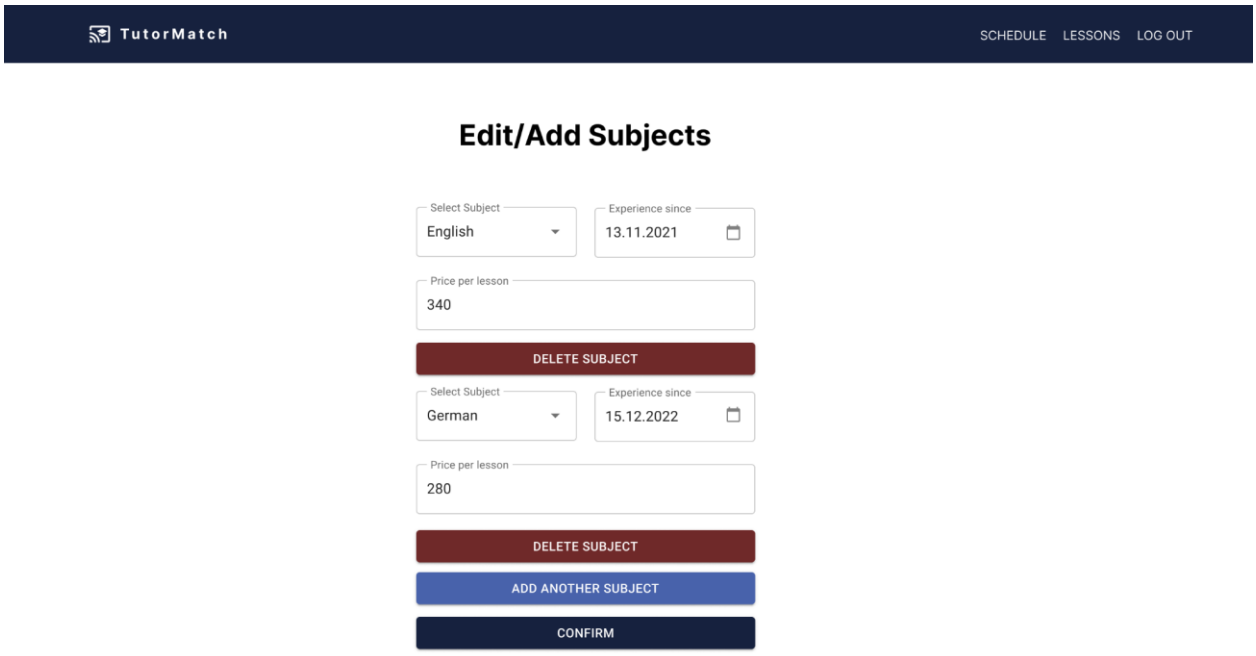

Рисунок 3.7.4 – Сторінка редагування предметів
коли їх декілька

4 ТЕОРЕТИЧНІ ЗАСАДИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

4.1 Entity relationship diagram

База даних є невід'ємною частиною програмних систем. Повноцінне використання ER-діаграми в розробці баз даних гарантує вам створення високоякісного проекту бази даних, який буде використовуватися при створенні, управлінні та підтримці бази даних. ER-модель також є засобом комунікації [17].

Entity Relationship Diagram (ERD) є графічним зображенням даних та їх взаємозв'язків у базі даних або інформаційній системі. Він використовує різні символи для позначення сутностей (об'єктів, які містять дані), атрибутів (інформація про сутності), і зв'язків між цими сутностями.

Отже, ERD використовується для аналізу, проектування та оптимізації структур даних, забезпечуючи легке візуальне розуміння структури та зв'язків даних. Це також сприяє кращому зрозумінню бізнес-процесів і спілкуванню між різними учасниками проекту, від аналітиків бізнесу до розробників.

На основі аналогів, які були проаналізовані було створено ERD, котра представляє структуру бази даних для платформи для пошуку професійних репетиторів (рис. 4.1.1). Вона містить три основні сутності: "Subjects", "Users", та "Lessons", кожна з яких має різні атрибути. Розглянемо дані атрибути:

1. Subjects (Предмети):
    - id: унікальний ідентифікатор предмета.
    - tutorID: ідентифікатор викладача, який веде предмет.
    - name: назва предмета.
    - price: ціна за урок з даного предмету.
    - experience: дата, з якої викладач почав викладати предмет.
2. Users (Користувачі):
    - id: унікальний ідентифікатор користувача.
    - role: роль користувача в системі (можливо обрати 2 ролі – студент або викладач).

- firstName та lastName: ім'я та прізвище.
- photo: шлях до фотографії користувача.
- birth_date: дата народження.
- description: короткий опис профілю.
- contact: контактні дані.
- email: пошта для подальшої авторизації.
- password: пароль для подальшої авторизації.

3. Lessons:

- id: унікальний ідентифікатор заняття.
- tutorID: ідентифікатор викладач.
- studentID: ідентифікатор студента.
- subjectID: ідентифікатор предмета, по якому проводиться заняття.
- date: дата та час проведення заняття.
- accepted: чи було заняття прийнято викладачем.

Розглянемо зв'язки між сутностями.

1. Зв'язок між Users та Subjects.

Викладачі (Users) пов’язані з предметами (Subjects), які вони викладають через атрибут tutorID у таблиці Subjects. Цей зв'язок вказує, що кожен предмет може бути пов'язаний з одним викладачем, а викладач може викладати кілька предметів. Це типічний приклад зв'язку один-до-багатьох, де один викладач може викладати декілька предметамів.

2. Зв'язок між Lessons та Subjects.

Заняття (Lessons) пов'язані з предметами (Subjects) через subjectID. Це вказує на те, що кожне заняття має конкретний предмет, і відповідно кожен предмет може бути частиною багатьох занять. Це ще один один-до-багатьох зв'язок, де один предмет може бути використаний у багатьох заняттях.

3. Зв'язок між Lessons та Users.

Заняття (Lessons) також пов'язані з користувачами (Users) через tutorID та studentID, tutorID поєднує заняття з викладачами, показуючи, хто веде

заняття, studentID з'єднує заняття з студентами, показуючи, кому викладають заняття.

Ці зв'язки ілюструють взаємодії між студентами та викладачами в контексті кожного заняття. Вони є прикладом зв'язку багато-до-багатьох, де багато студентів можуть взаємодіяти з багатьма викладачами через різні заняття.

Дана діаграма допомагає визначити та візуалізувати структуру бази даних, виявити ключові зв'язки між даними та спрощує розробку та управління базою даних.

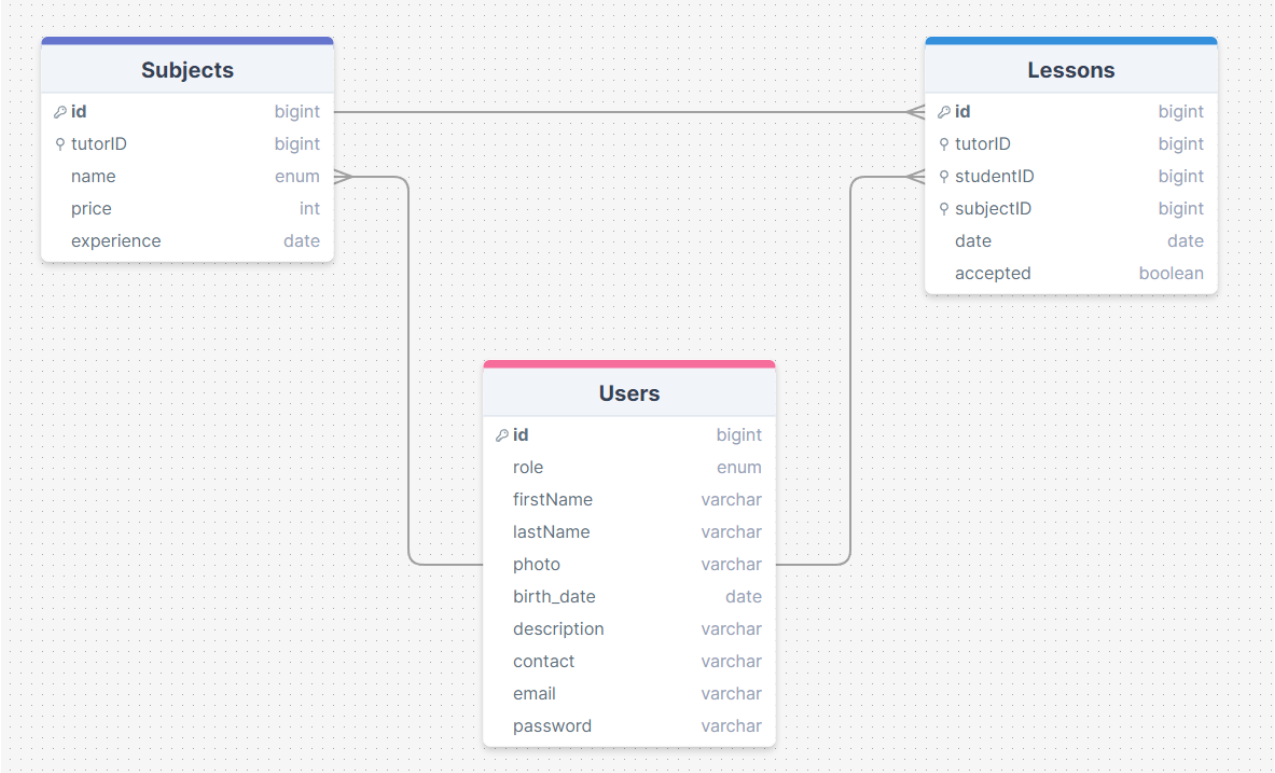

Рисунок 4.1.1 – ERD

4.2 Use Case Diagram

Use Case Diagram (Діаграма варіантів використання) — це важливий інструмент в моделюванні та аналізі систем з точки зору користувача. Вона належить до сімейства діаграм UML (Unified Modeling Language) і використовується для візуалізації функціональних вимог системи.

Use Case Diagram є важливим інструментом в процесі аналізу та дизайну систем, що використовується для відображення взаємодій між акторами (користувачами чи зовнішніми системами) та системою для досягнення певних цілей чи завдань. Цей інструмент забезпечує високорівневий огляд функціональності системи, дозволяючи розуміти, що система повинна робити, не вдаючись у внутрішні деталі її реалізації [18].

UCD допомагає визначити обов'язки та очікування акторів щодо системи, включаючи які функції системи вони ініціюють або беруть участь. Актори можуть бути зображені як користувачі або інші системи, які взаємодіють із моделюваною системою. Кожен Use Case описує послідовність взаємодій між акторами та системою для досягнення конкретної мети, представлені у вигляді овалів або еліпсів на діаграмі.

Діаграма також визначає межі системи, які включають усі випадки використання та допомагають розмежувати систему від зовнішніх елементів. Відносини між акторами та варіантами використання відображаються за допомогою твердих ліній, що показують, які актори залучені до кожного варіанту використання.

Крім того, Use Case Diagram є незамінною при визначенні вимог до системи, плануванні проектів, проектуванні систем, тестуванні та валідації, а також забезпечує важливу документацію, яка допомагає забезпечити успішну розробку програмного забезпечення, створюючи спільну мову для стейкхолдерів проекту.

На основі вимог була розроблена схема варіантів використання, в якій враховані всі функції (рис. 4.2.1):

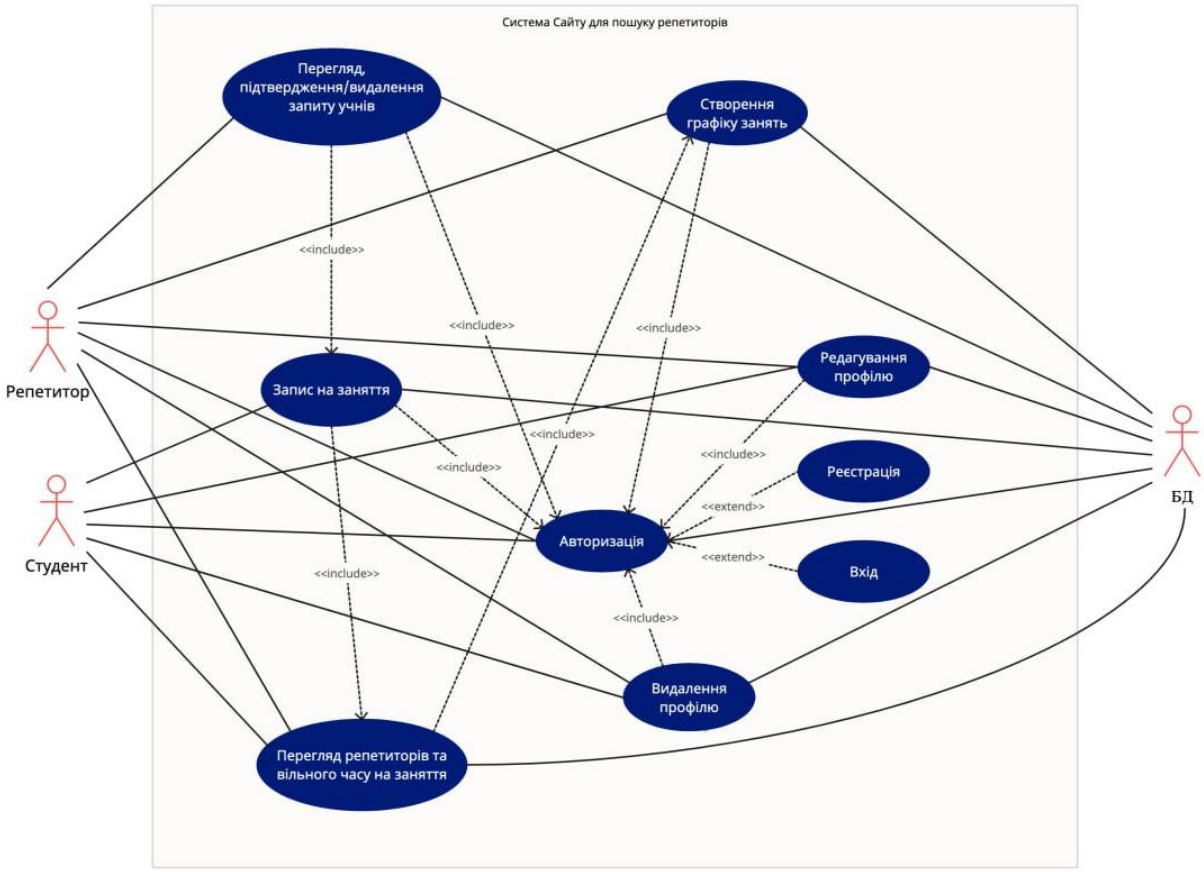

Рисунок 4.2.1 – UCD

Розглянемо основні випадки використання, які визначені на діаграмі:

- Реєстрація: дозволяє користувачам створити новий обліковий запис.
- Вхід: дозволяє зайти в систему після реєстрації.
- Авторизація: включає в себе як реєстрацію так і вхід.
- Редагування профілю: дозволяє користувачам оновлювати свої особисті дані.
- Видалення профілю: користувачі можуть видалити свій обліковий запис.
- Перегляд репетиторфів та вільного часу на заняття: дозволяє користувачам переглядати карточки репетиторів та подивитися їх графік занять вільний до запису.
- Створення графіку занять: дозволяє репетиторам створити графік за допомогою якого у визначений час студенти можуть подавати запити на заняття до них.

- Запис на заняття: студенти можуть відправити запиит на заняття до репетиторів.
- Перегляд, підтвердження/видалення запиту учнів: репетитори можуть підтвердити або видалити запит на заняття який надіслав їм студент.

Діаграма також показує зв'язки "include" і "extend", які вказують на взаємодію між різними випадками використання. Наприклад, авторизація включає в себе реєстрацію та вхід, а запис на заняття може включати перегляд і підтвердження або відмову заняття.

Ця діаграма корисна для розуміння взаємодії між користувачами та системою, дозволяючи розробникам зосередитись на створенні необхідних функціональних можливостей і забезпечуючи, що вимоги користувачів враховуються на етапі проектування.

5 РОЗРОБКА І ІМПЛЕМЕНТАЦІЯ

5.1 Структура проекту backend частини

Бекенд частина проекту організована у декілька основних директорій та файлів для розробки інформаційної веб-системи (рис. 5.1.1).

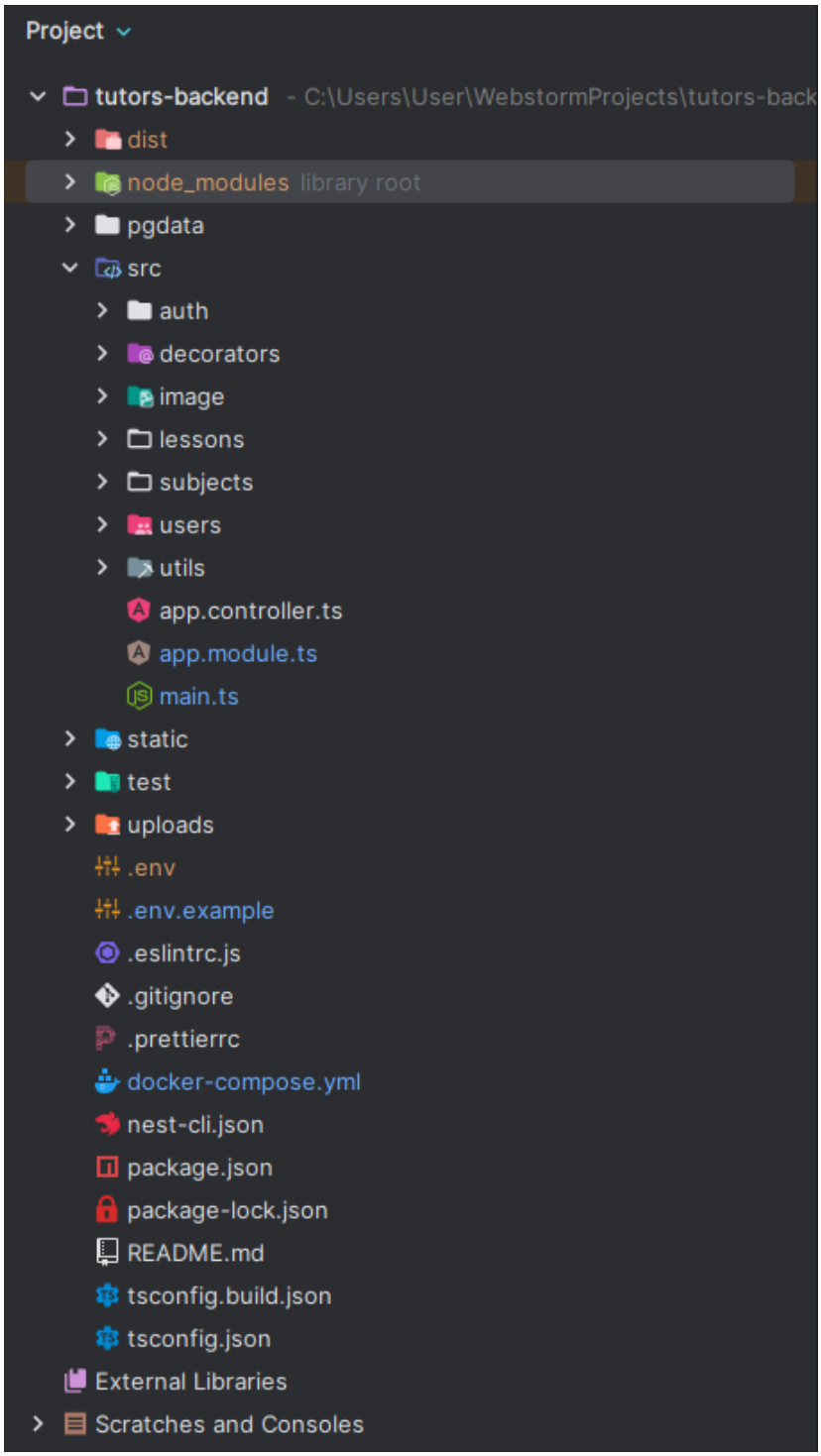

Рисунок 5.1.1 – Структура backend частини

Структура проекту складається з декількох ключових компонентів:

- dist: дана папка містить зкомпільовані JavaScript файли, які є результатом транспіляції TypeScript коду. Вона створюється автоматично під час білду проекту.
- node_modules: бібліотека, яка містить всі залежності проекту. Це стандартна папка для проектів Node.js, що використовуються для зберігання і керування внешніми бібліотеками та модулями.
- src: головна папка з вихідним кодом проекту, яка включає такі підпапки та файли:
    - auth: модуль аутентифікації, який відповідає за обробку авторизації користувачів.
    - decorators: містить декоратори TypeScript, які можуть бути використані для розширення функціональності класів або методів.
    - image: модуль для обробки зображень.
    - lessons, subjects, users: папки, що містять логіку та моделі для роботи з різними доменами бізнес-логіки проекту (уроки, предмети, користувачі).
    - utils: допоміжні інструменти та функції.
- app.controller.ts, app.module.ts, main.ts: основні файли, що запускають і конфігурують NestJS додаток.
- static: папка для статичних файлів, які можуть бути віддані клієнту напряму.
- test: папка для тестових файлів.
- uploads: папка для зберігання файлів, завантажених користувачами.
- .env та .env.example: файли для зберігання змінних оточення, .env.example використовується для подальшого завантаження в гіт як приклад даного файлу.

- .eslintrc.js, .gitignore, .prettierrc: конфігураційні файли для ESLint, Git та Prettier.
- docker-compose.yml, nest-cli.json, package.json, package-lock.json, README.md, tsconfig.build.json, tsconfig.json: конфігураційні і описові файли, що використовуються для керування залежностями, скриптами білду та іншими налаштуваннями проекту.

5.2 Структура проекту frontend частини

Фронтенд частина проекту організована у декілька основних директорій та файлів для розробки веб-додатку (рис. 5.2.1).

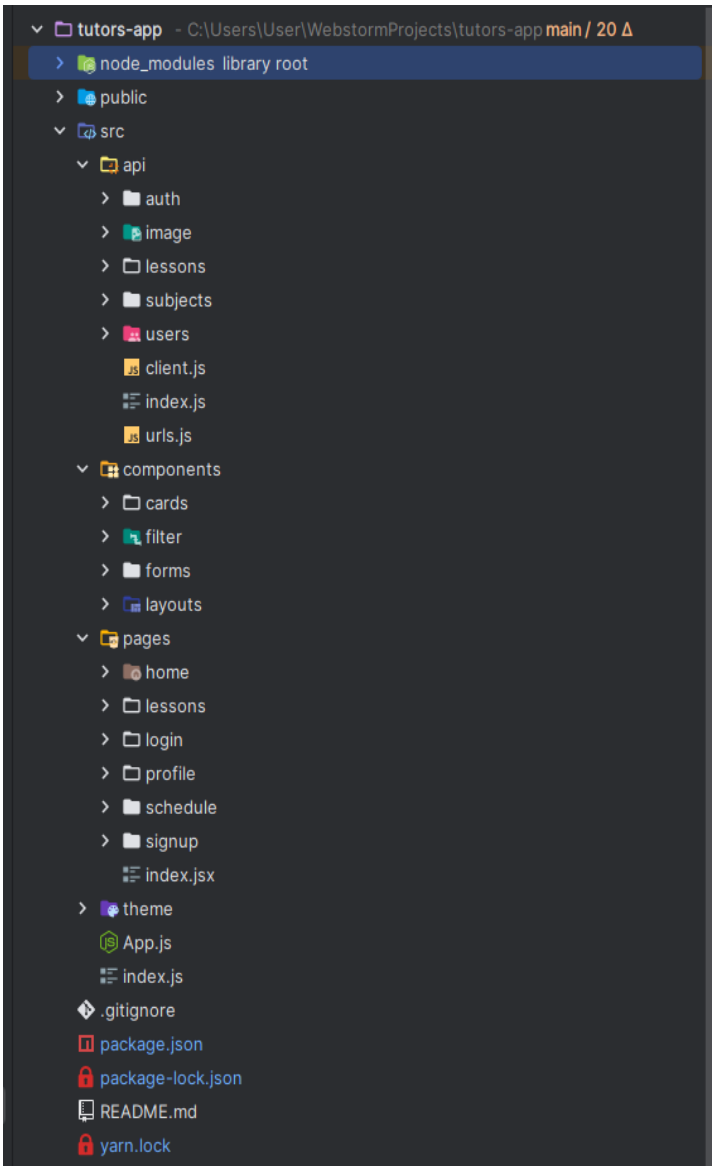

Рисунок 5.2.1 – Структура frontend частини

Розглянемо детальний опис структури проекту:

- node_modules: бібліотека з встановленими залежностями Node.js, які використовуються в проекті.
- public: директорія для статичних файлів, таких як зображення, JavaScript та CSS, які доступні публічно.
- src: головна папка з вихідним кодом проекту, яка включає такі підпапки та файли:
    - api: містить модулі для роботи з API:
        - auth: функціонал аутентифікації користувачів.
        - image: обробка зображень.
        - lessons, subjects, users: модулі для взаємодії з даними уроків, предметів та користувачів.
        - client.js, index.js, urls.js: Конфігураційні та допоміжні скрипти.
    - components: компоненти React для інтерфейсу:
        - cards: містить карточки репетиторів, профілю та предметів.
        - filter: містить UI компонент блоку фільтрації.
        - forms: містить форми для авторизації, редагування профілю та графіку репетитора.
        - layouts: містить головне меню сайту.
    - pages: сторінки додатку, такі як:
        - home: головна сторінка, що містить карточки репетиторів та блок фільтрації.
        - lessons: сторінка занять на якій відображаються запити на заняття та прийняті запити.
        - login: сторінка входу в аккаунт.
        - profile: сторінка профілю користувача.
        - schedule: сторінка графіку роботи репетитора.
        - signup: сторінка реєстрації користувача.
        - index.jsx: головний вхідний файл сторінок.

    - theme: тема оформлення додатку, що містить в собі головні кольори які використовуються при розробці.
- App.js: головний React компонент, який ініціалізує додаток.
- index.js: головний вхідний файл додатку.
- gitignore: визначає файли та директорії, які не слід включати в репозиторій git.
- package.json та package-lock.json: конфігураційні файли Node.js з переліком залежностей та іншими налаштуваннями.
- README.md: опис проекту та інструкції для розгортання.
- yarn.lock: файл залежностей для менеджера пакетів Yarn.

Така структура дозволяє систематизувати розробку додатку та забезпечує легкість управління та розширення проекту.

5.3 Розгортання backend частини

Для розгортання backend на вашому ПК першим кроком необхідно завантажити Docker [19] та Node JS [20], що має 18 версію.

Переконайтесь, що у директорії проекту є файл .env.example. Створіть з нього файл .env, який буде містити всі необхідні змінні середовища для проекту, при необхідності скоригуйте дані під себе.

Наступним кроком у консолі в директорії проекту необхідно написати команду: "`npm install`" (або "*`npm i`*"). Дана команда необхідна для встановлення залежностей проекту, які зазначені в файлі package.json.

Далі необхідно запустити команду "`docker-compose up -d`". Ця команда підніме всі служби, визначені у вашому файлі docker-compose.yml, і створить необхідні контейнери (рис. 5.3.1).

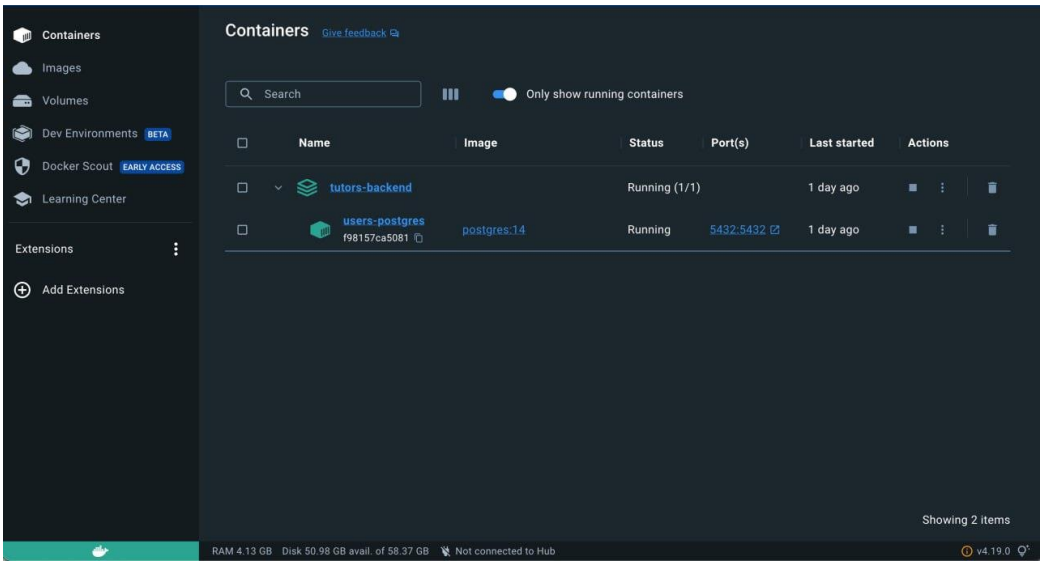

Рисунок 5.3.1 – Docker, розгорнуті контейнери

Для запуску проекту в режимі розробки вводимо команду "`npm run start:dev`" у консоль.

Щоб перевірити, чи бекенд працює коректно, відкрийте Swagger UI, який доступний за посиланням http://localhost:3000/api. Тут ви зможете переглянути всі доступні ендпойнти і виконати тестові запити до вашого API (рис. 5.3.2).

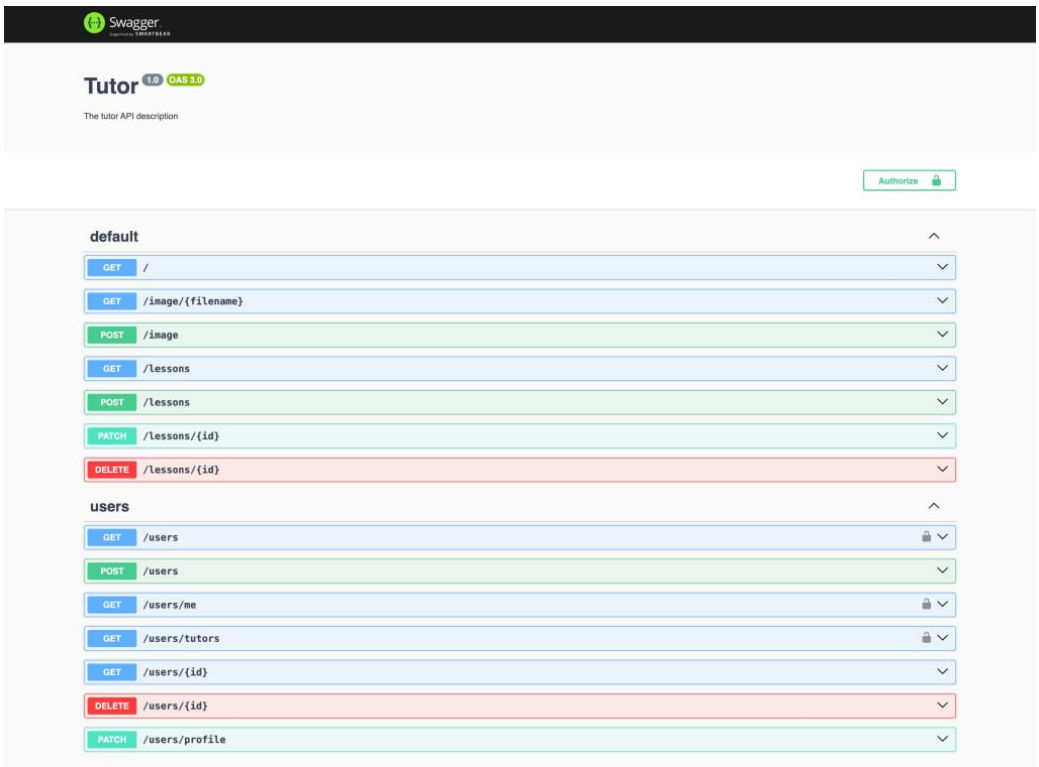

Рисунок 5.3.2 – Swagger, доступні ендпойнти

5.4 Розгортання frontend частини

Для розгортання frontend частини проекту, так само як і для backend, важливо спочатку встановити всі необхідні залежності. Це робиться за допомогою команди "`npm install`" або скорочено "*`npm i`*", яку слід виконати у кореневій директорії проекту. Файл package.json містить список усіх бібліотек та модулів, які потрібні для роботи проекту. Ці залежності включають як ті, що потрібні безпосередньо для виконання коду на сервері, так і ті, що використовуються в процесі розробки, наприклад, компілятори або тестові утиліти.

Наступним кроком після встановлення залежностей, для запуску проекту використовується команда "`npm run start`". Ця команда зазвичай визначена в секції scripts файла package.json і здебільшого просто запускає сервер Node.js, який обслуговує ваш frontend.

Після запуску сервера, веб-додаток буде доступний за адресою, зазвичай http://localhost:3001 або іншою портом, зазначеним у налаштуваннях сервера або у команді запуску. Це дозволяє перевірити роботу веб-додатку локально перед тим, як виконувати його розгортання на виробничому сервері (рис. 5.4.1).

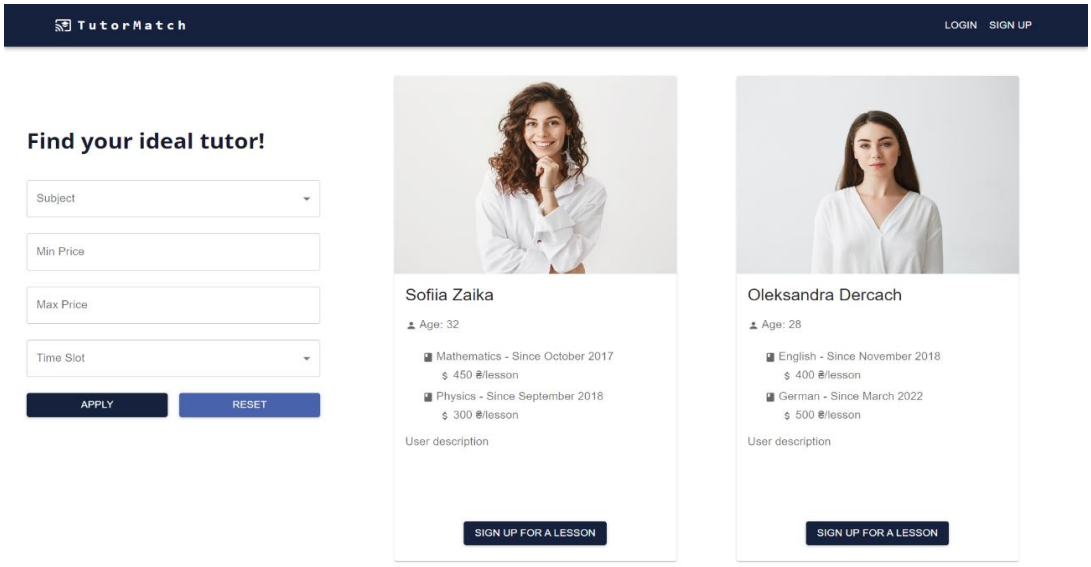

Рисунок 5.4.1 – Розгорнута frontend частина

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

При вході на головну сторінку сайту TutorMatch, користувач може бачити форму для пошуку репетитора та список доступних викладачів з їхніми профілями (рис. 6.1).

Розглянемо для початку хід дій для репетитора, котрий хоче розмістити свої послуги на сайті.

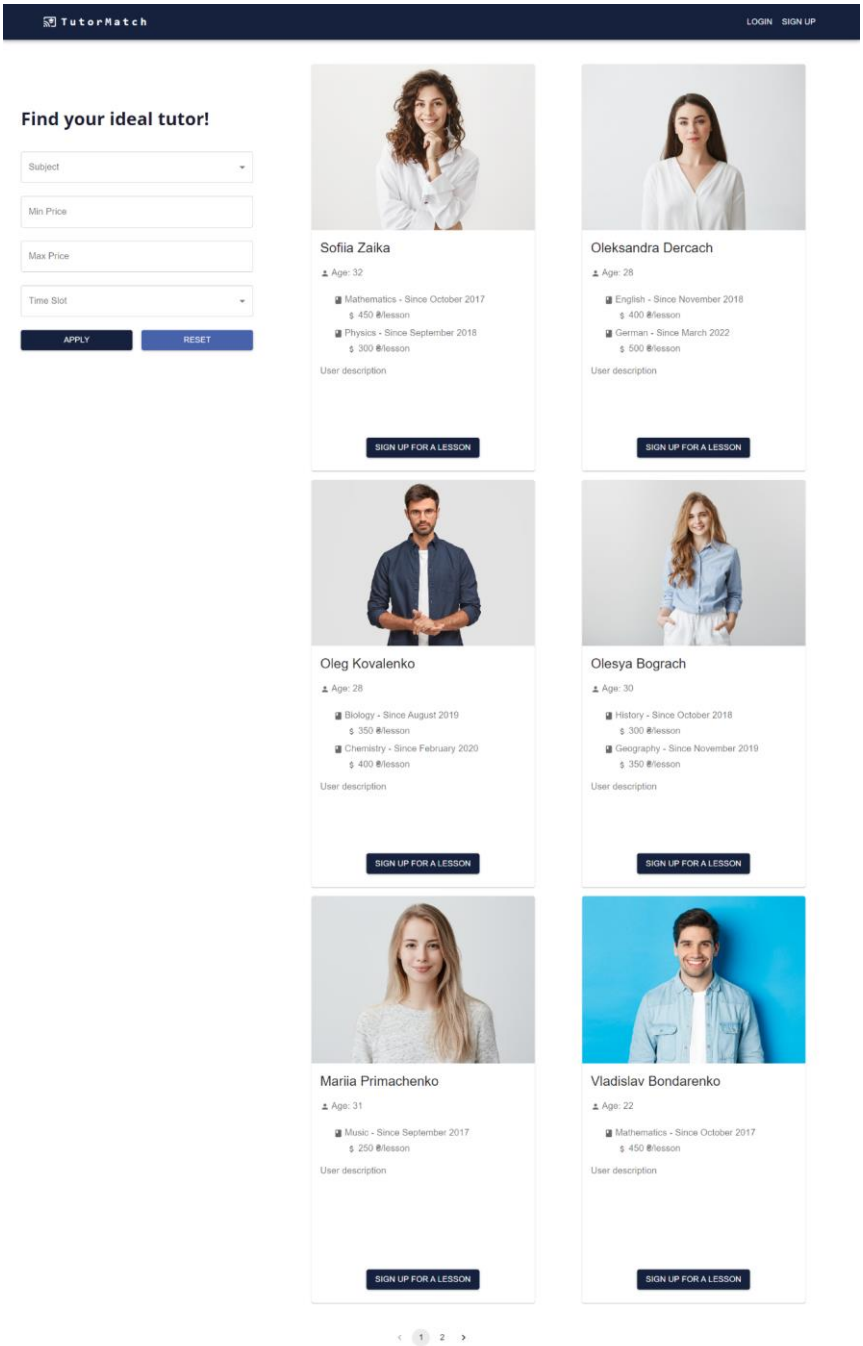

Рисунок 6.1 – Головна сторінка

У першу чергу репетитору потрібно зареєструватись. Для цього він заходить на сторінку "Sign up", де в якості ролі по замовчуванні стоїть роль студента (рис. 6.2).

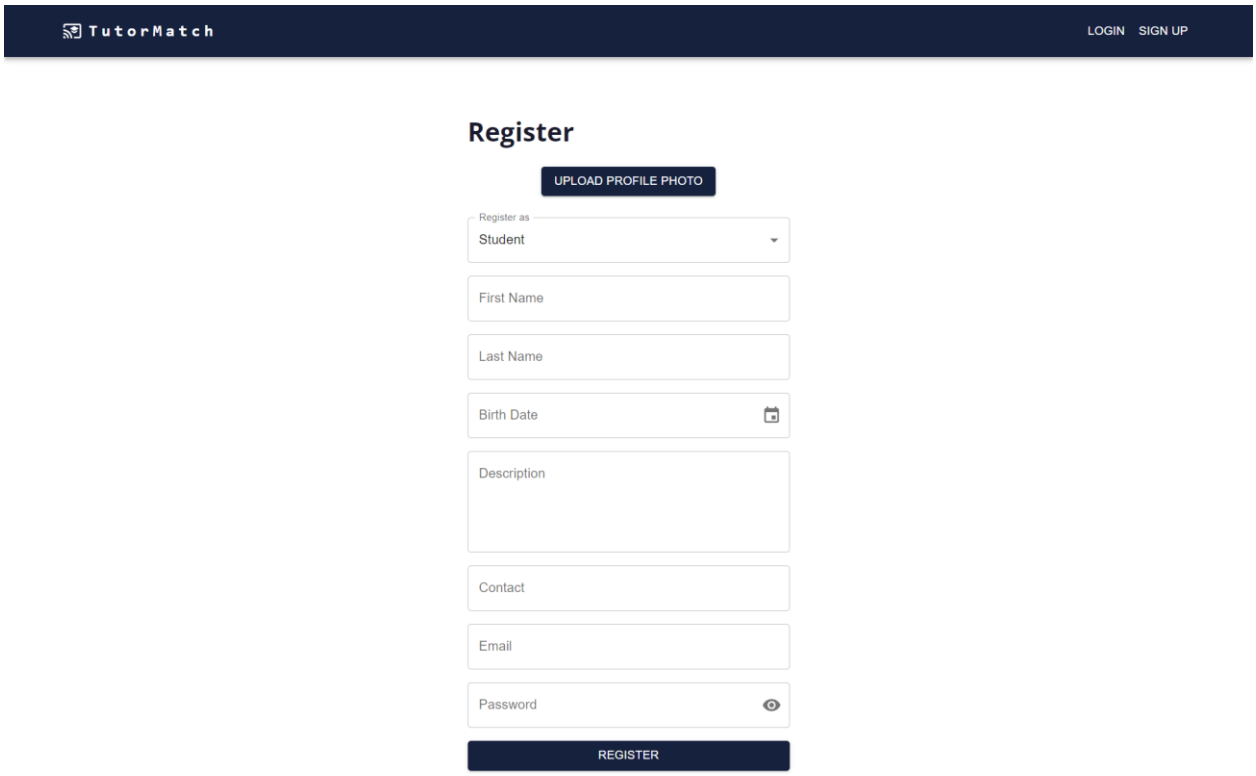

Рисунок 6.2 – Сторінка реєстрації у ролі студента

Наступним кроком репетитору потрібно змінити роль на відповідну своїй – "Tutor". Таким чином з’являються додаткові поля для заповнення необхідні викладачам, а саме: предмет викладання, досвід з цього предмету та ціна за заняття (рис. 6.3). За допомогою кнопки "Add subject" можна додати ще необхідні предмети викладання.

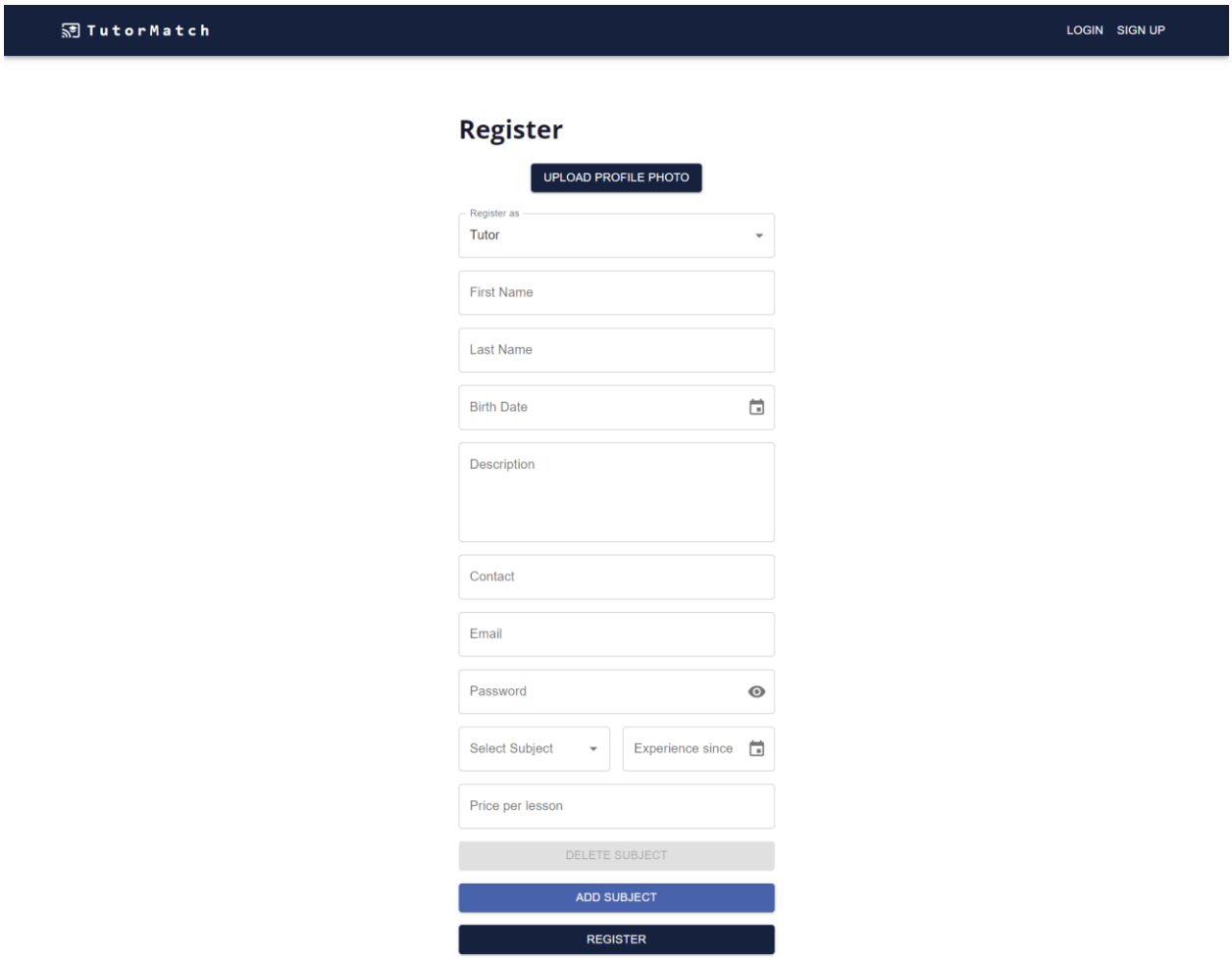

Рисунок 6.3 – Сторінка реєстрації у ролі вчителя

Зареєструватись з пустими полями є неможливим, оскільки на даному сайті є валідація (рис. 6.4).

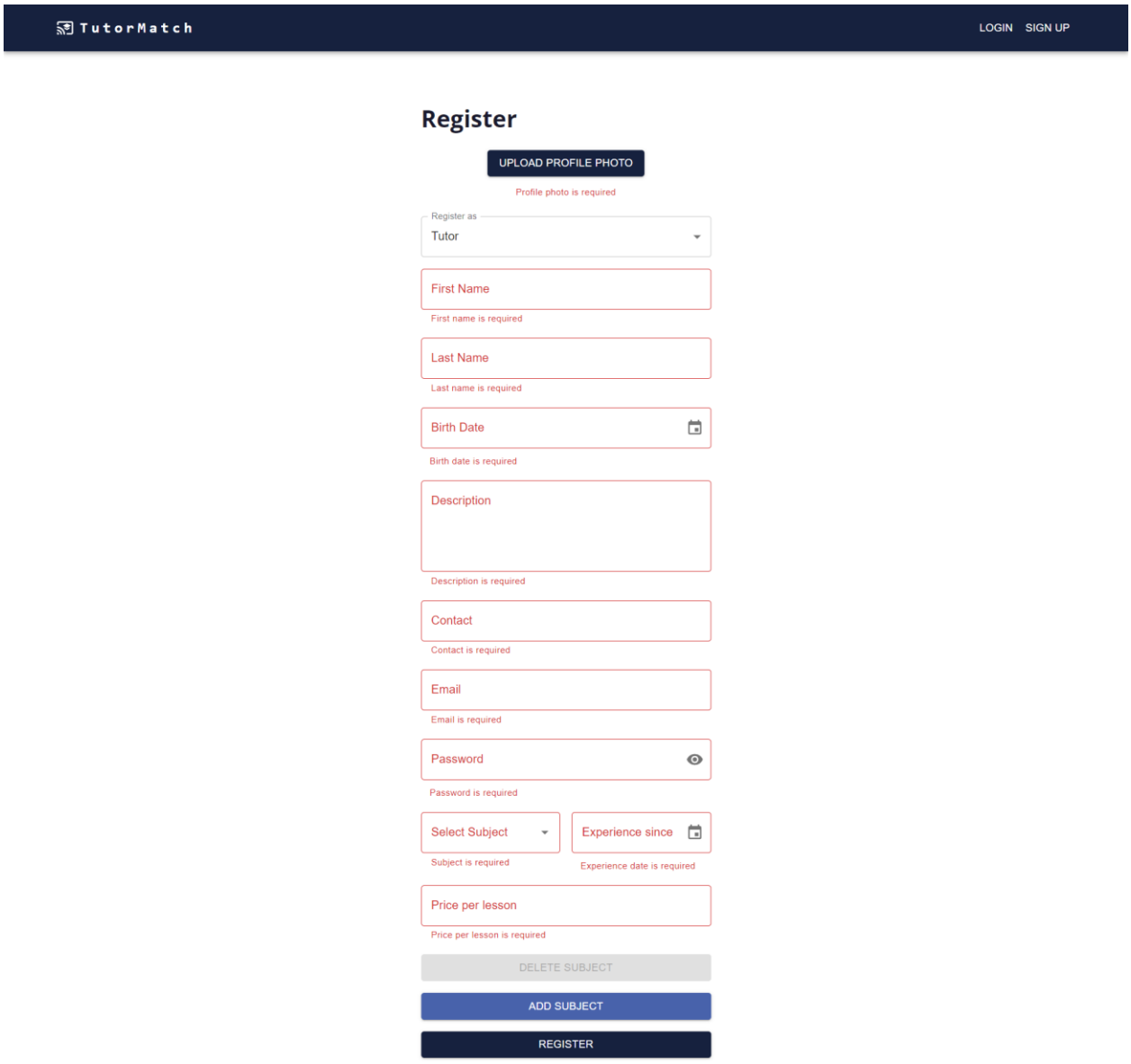

Рисунок 6.4 – Сторінка реєстрації валідація для репетитора

Аналогічно із вчителем валідація працює і для студента, проте з деякими відмінностями. Студент може залишити поле опису пустим та не завантажувати фотографію профілю (рис. 6.5).

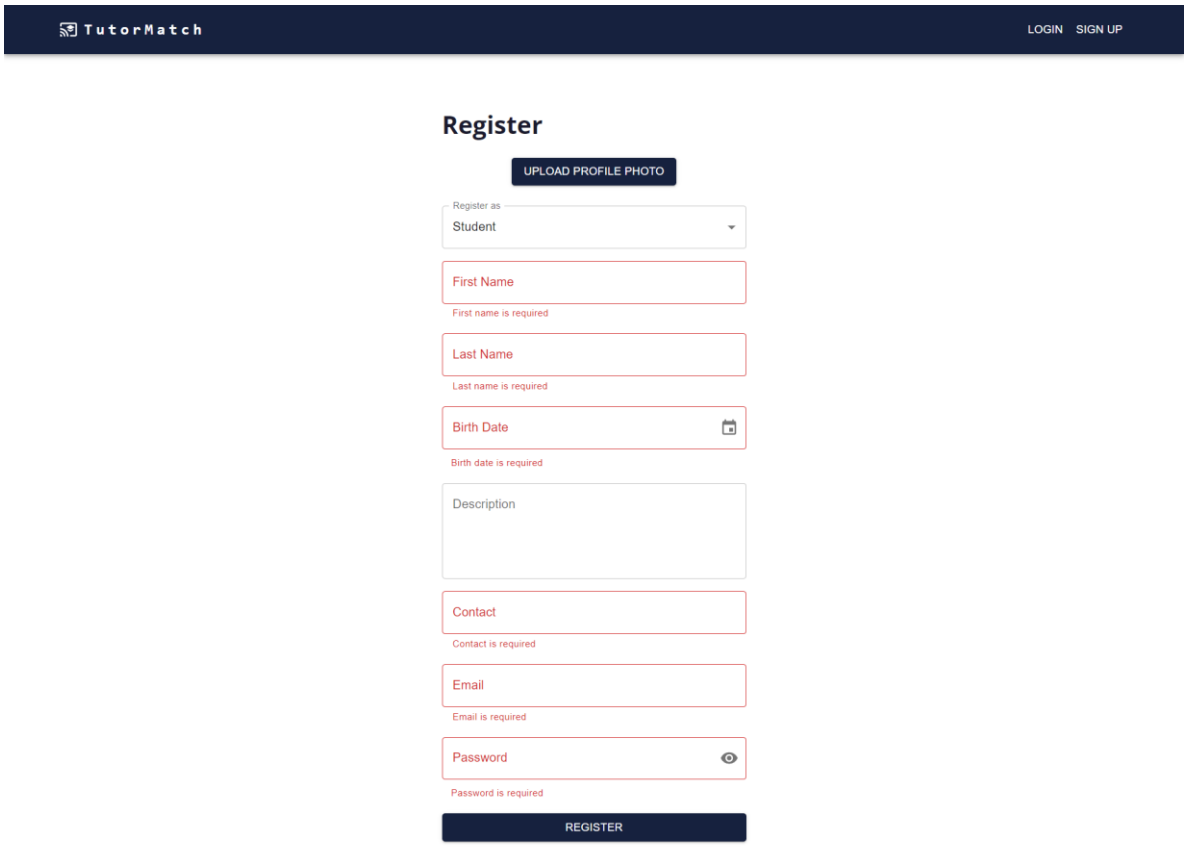

Рисунок 6.5 – Сторінка реєстрації валідація для студента

Дати у полях реєстрації користувач обирає ха допомогою календаря, що є зручним користувацьким досвідом (рис. 6.6).

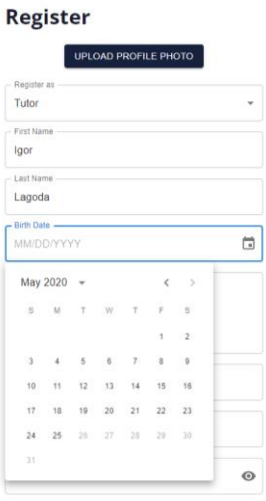

Рисунок 6.6 – Сторінка реєстрації, заповнення дати народження

Варто зазначити, що діють також такі умови валідації як (рис. 6.7):

- Поле опису не може бути більше 200 символів.
- Поле контакту має бути у форматі +38000000000.

- Поле пошти має відповідати формату електронної адреси.
- Пароль має бути мінімум 6 символів (рис. 6.8).
- Пароль має містити мінімум 1 цифру (рис. 6.9).
- Ціна не може бути нижче 0 (рис. 6.10).

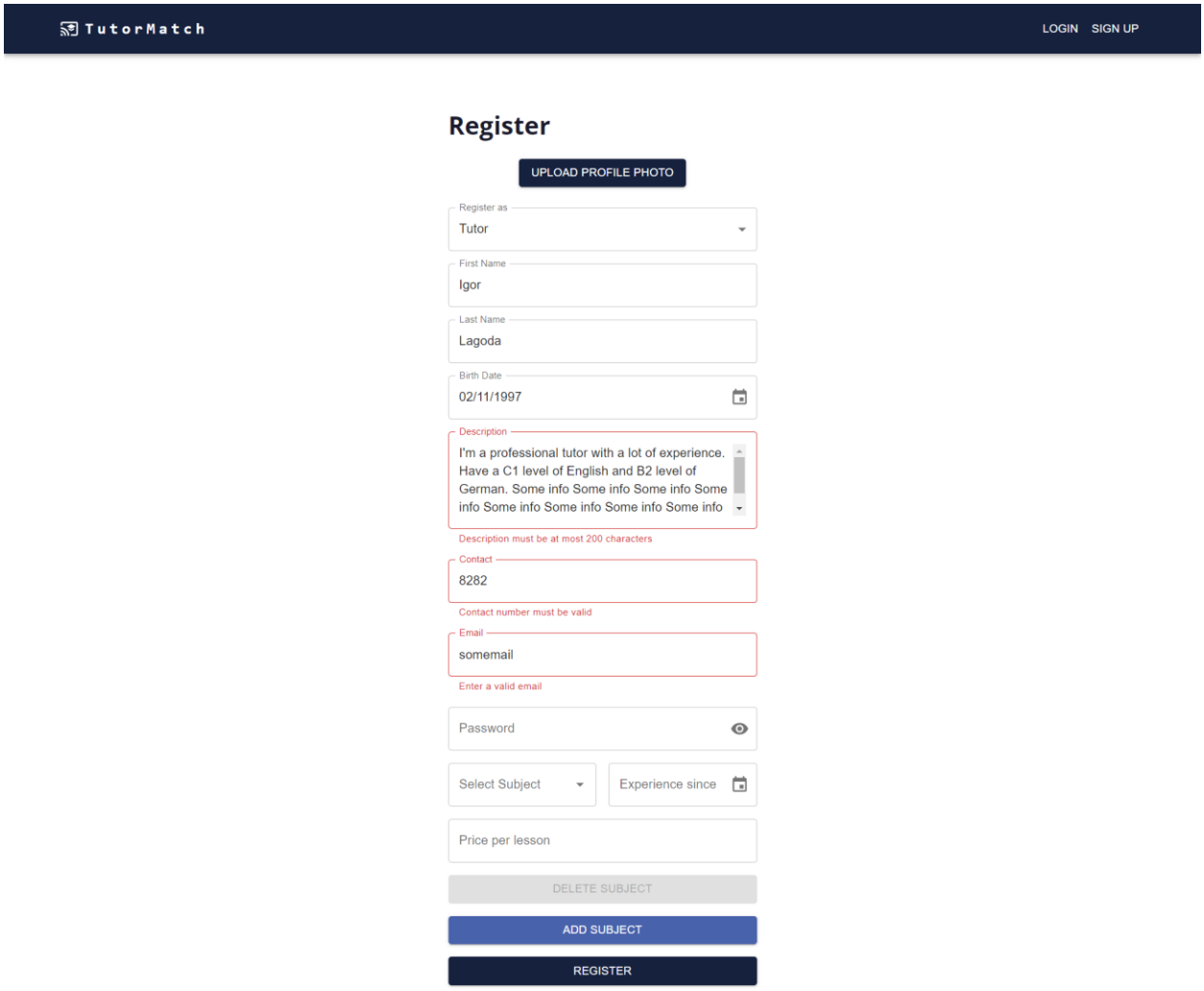

Рисунок 6.7 – Сторінка реєстрації, додаткові умови валідації (опис, контакт, пошта)

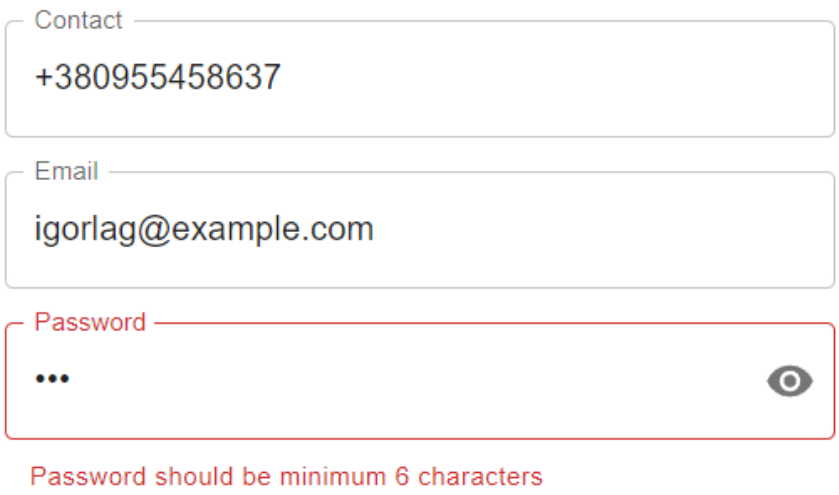

Рисунок 6.8 – Сторінка реєстрації, додаткові умови валідації (пароль - мінімально 6 символів)

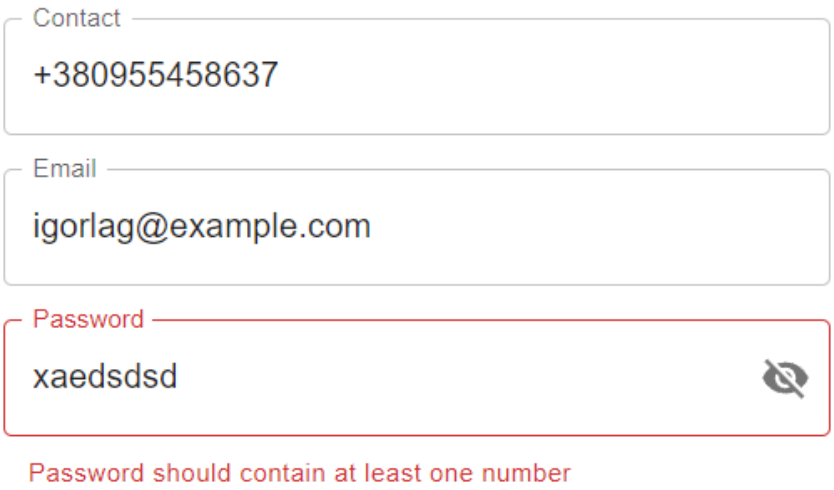

Рисунок 6.9 – Сторінка реєстрації, додаткові умови валідації (пароль - мінімум 1 цифра)

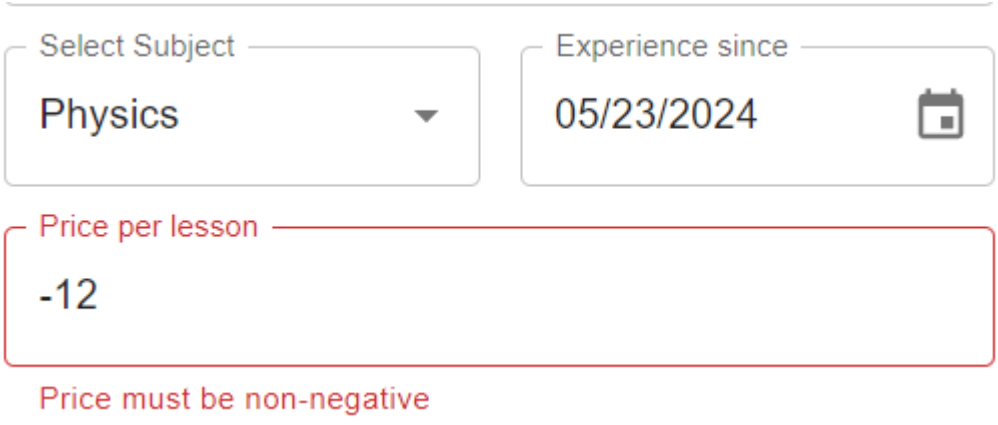

Рисунок 6.10 – Сторінка реєстрації, валідація ціни

Предмет обирається за допомогою випадаючого списку (рис. 6.11). При додаванні ще предметів викладання неможливо обрати предмет, який було обрано до цього, він зникає зі списку.

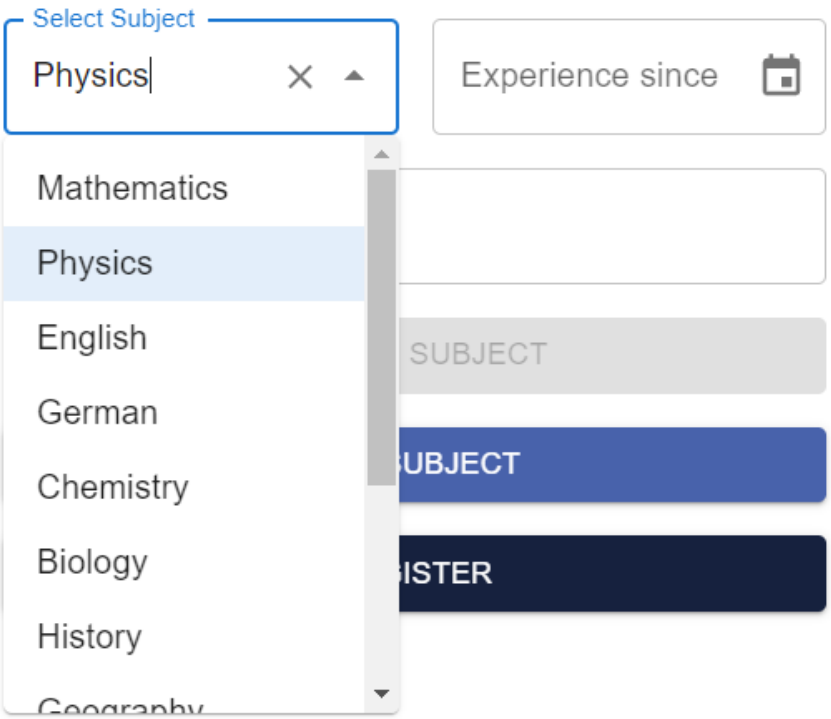

Рисунок 6.11 – Сторінка реєстрації, вибір предмета викладання

Для репетитора є необхідним вибір фотографії, тому після натискання на кнопку "Upload Profile Photo" користувач може обрати необхідну світлину для свого профіліо (рис. 6.12).

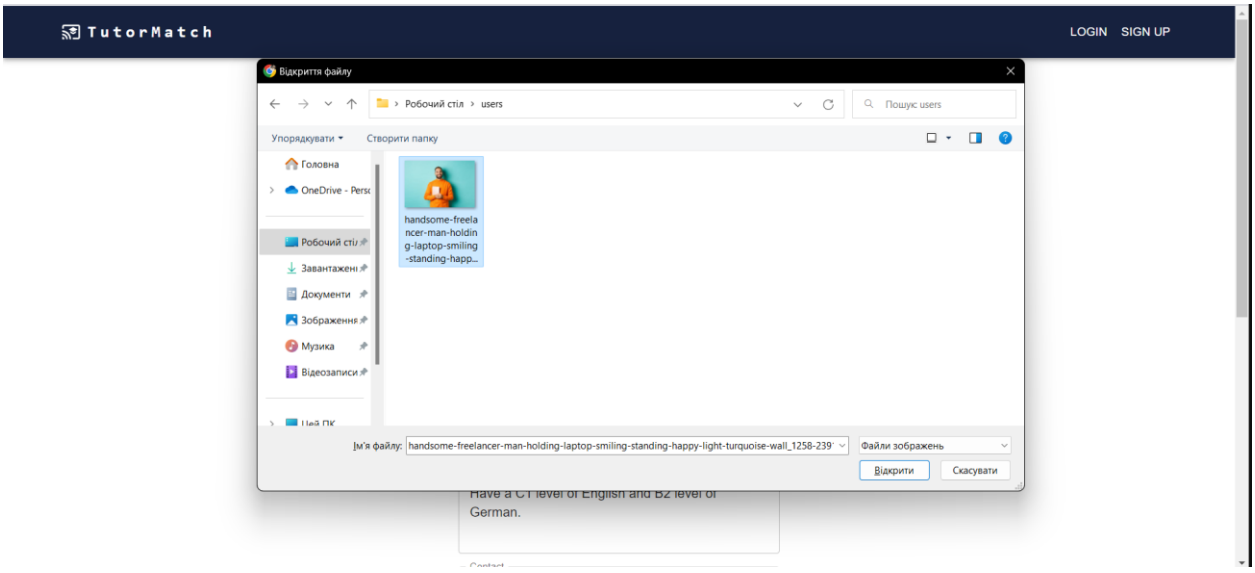

Рисунок 6.12 – Сторінка реєстрації, вибір фотографії

У результаті звповнена форма реєстрації має вигляд відповідно до рисунку 6.13.

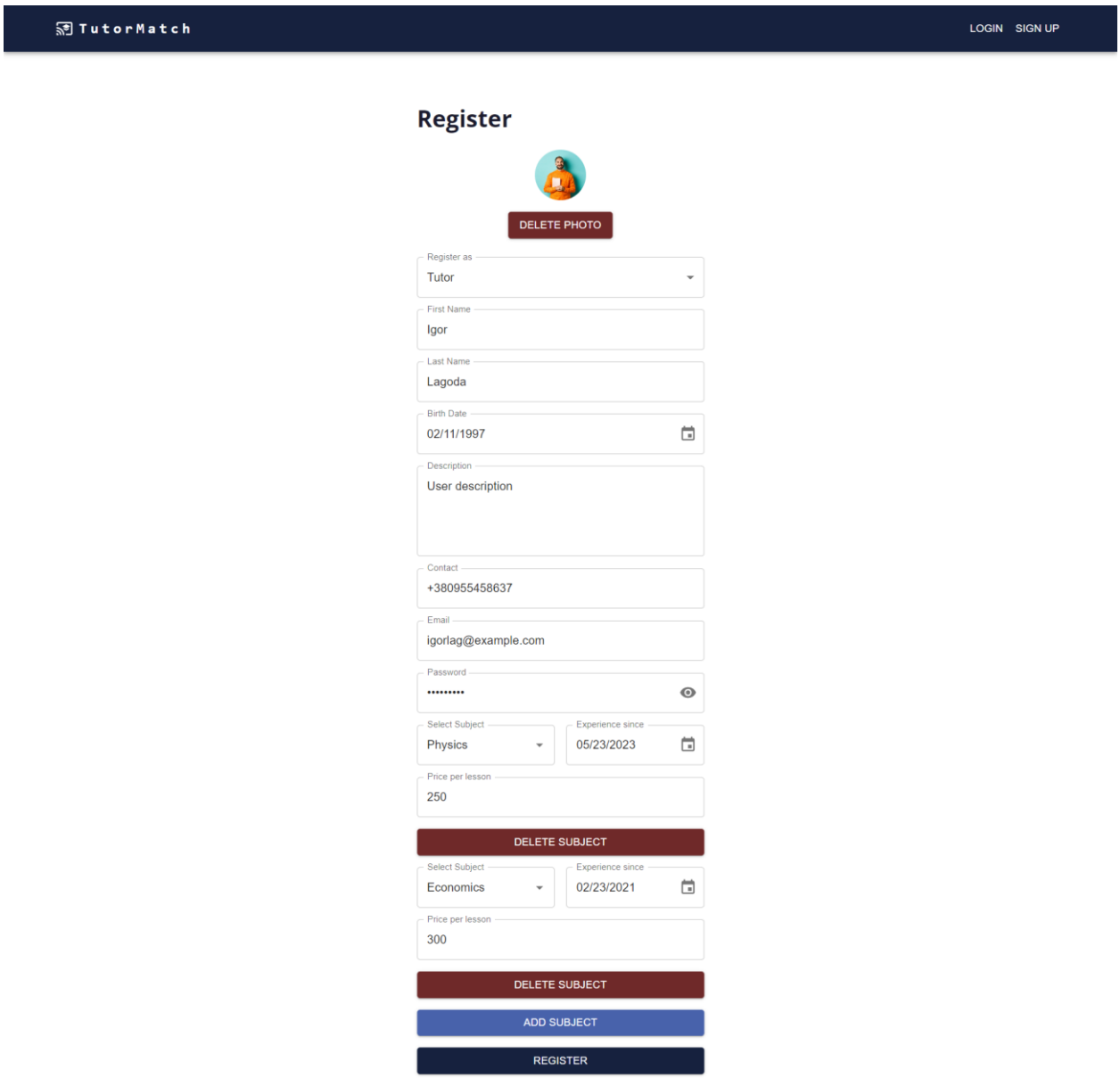

Рисунок 6.13 – Сторінка реєстрації з заповненими даними

Після успішної реєстрації здійснюється перехід на сторінку профіля користувача (рис. 6.14). На сторінці профілю можна не тільки переглянути свої дані, але й редагувати їх. Таккож є можливість видалити свій аккаунт.

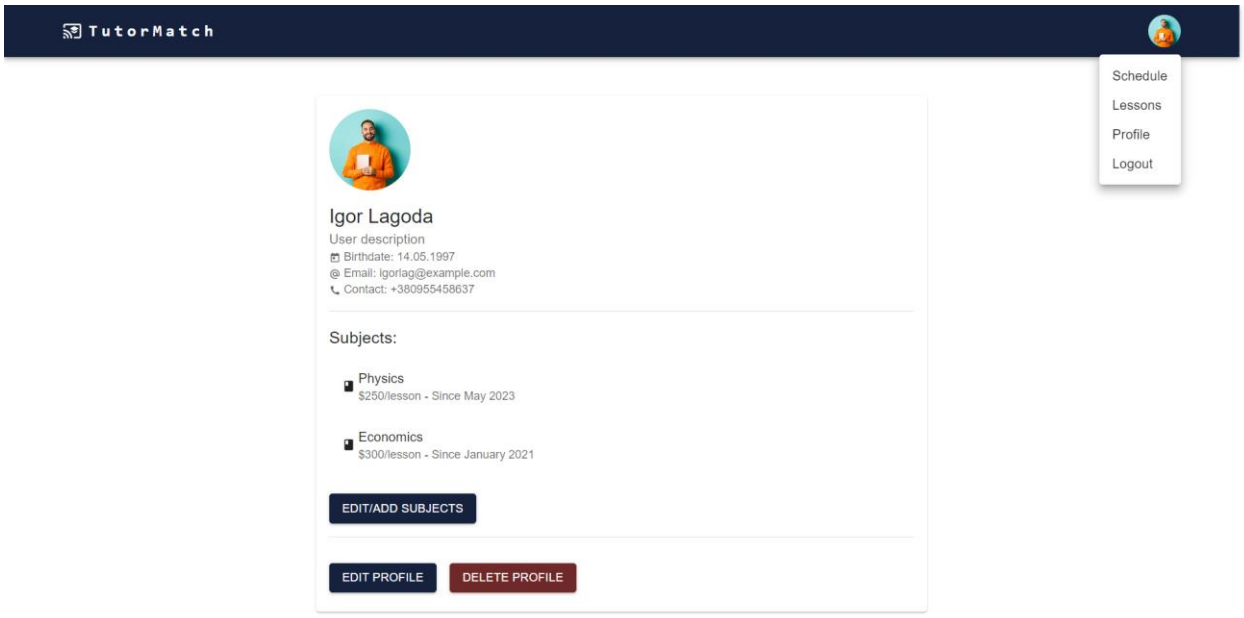

Рисунок 6.14 – Сторінка профілю користувача та вигляд випадаючого меню для репетитора

При необхідності можна редагувати предмети якщо в майбутньому ціна за викладання виросте чи якщо репетитор підвищить свою кваліфікацію та буде викладати нові предмети (рис. 6.15).

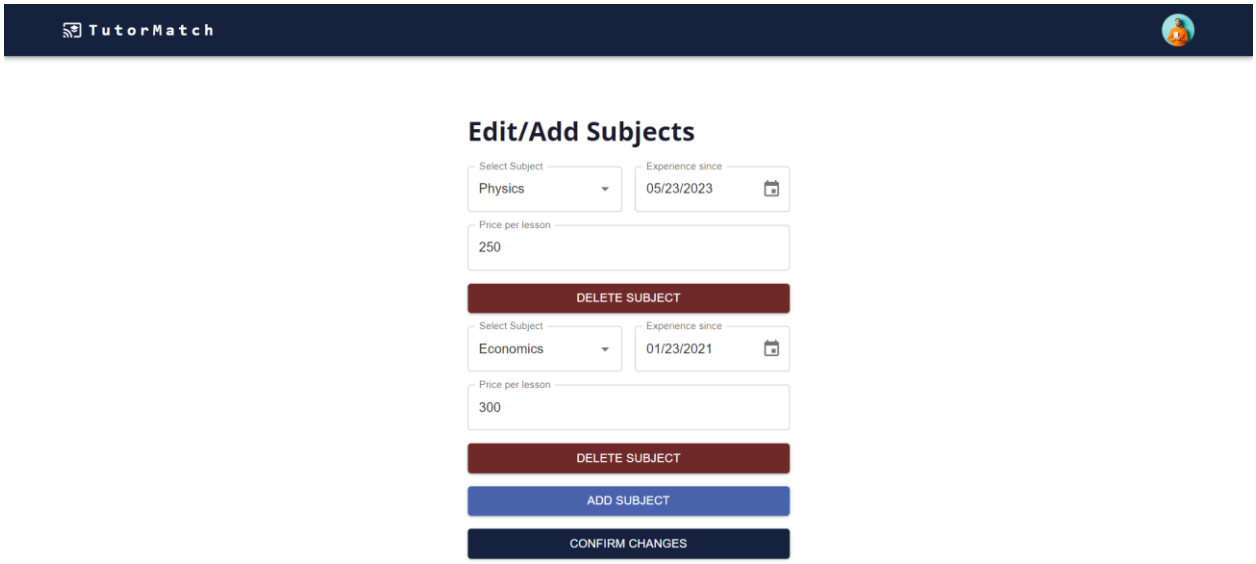

Рисунок 6.15 – Сторінка редагування предметів викладання репетитора

Аналогічно можна змінити дані профілю, а точніше особисту інформацію (рис. 6.16).

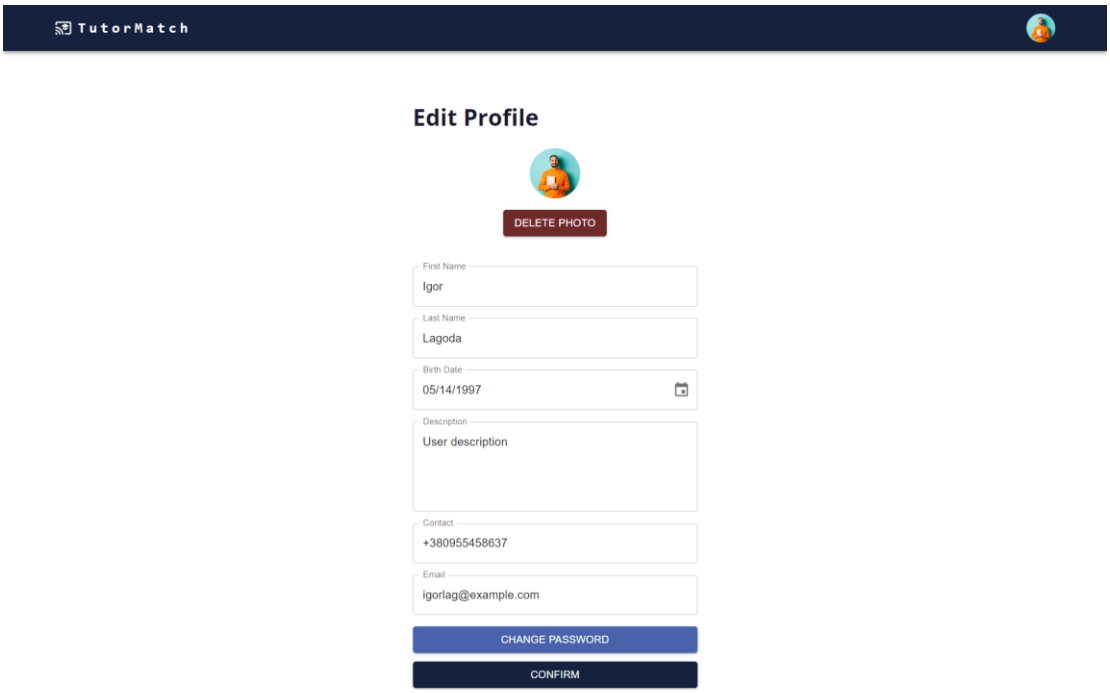

Рисунок 6.16 – Сторінка редагування профілю

Для змінення паролю необхідно натиснути кнопку "Change passport" після натисканні якої з’являться поля паролів для вводу – старого та нового відповідно (рис. 6.17).

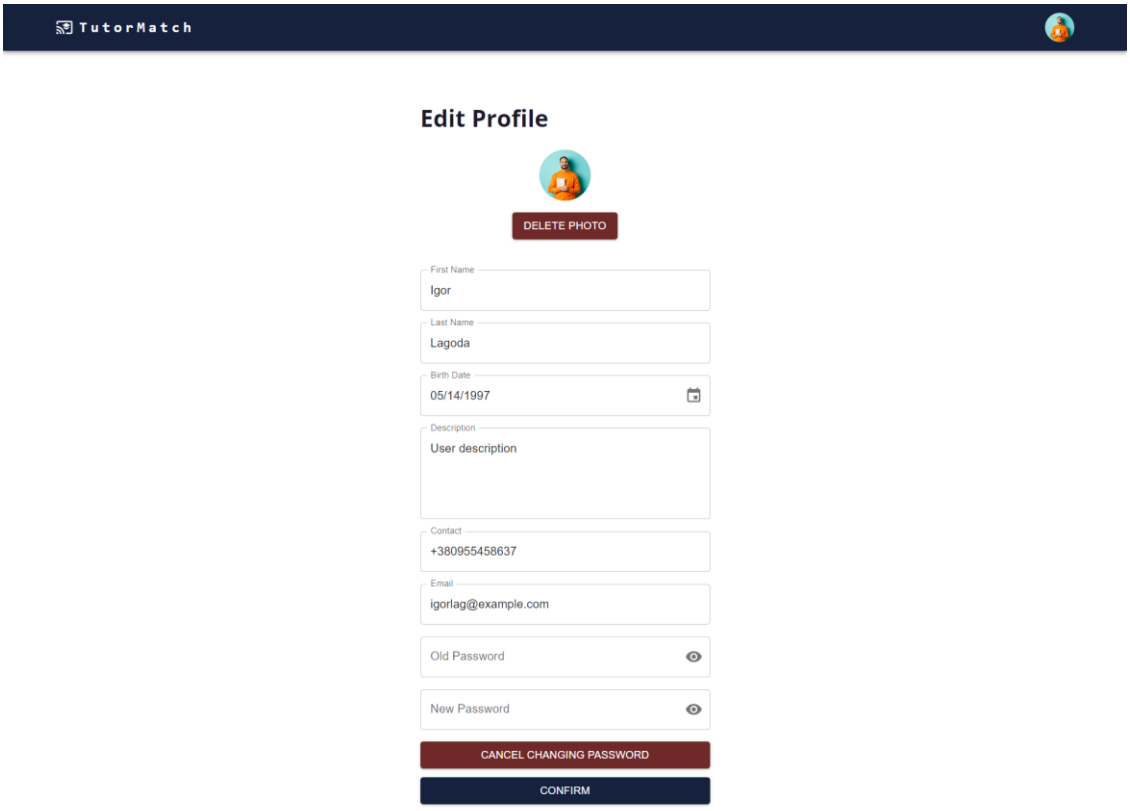

Рисунок 6.17 – Сторінка редагування профілю
із опцією змінення паролю

Для даних полей також працює валідація (рис. 6.18):

- старий пароль має співпадати з існуючим діючим паролем.
- новий пароль має містити мінімум 6 символів.
- новий пароль має містити хоча б 1 цифру.

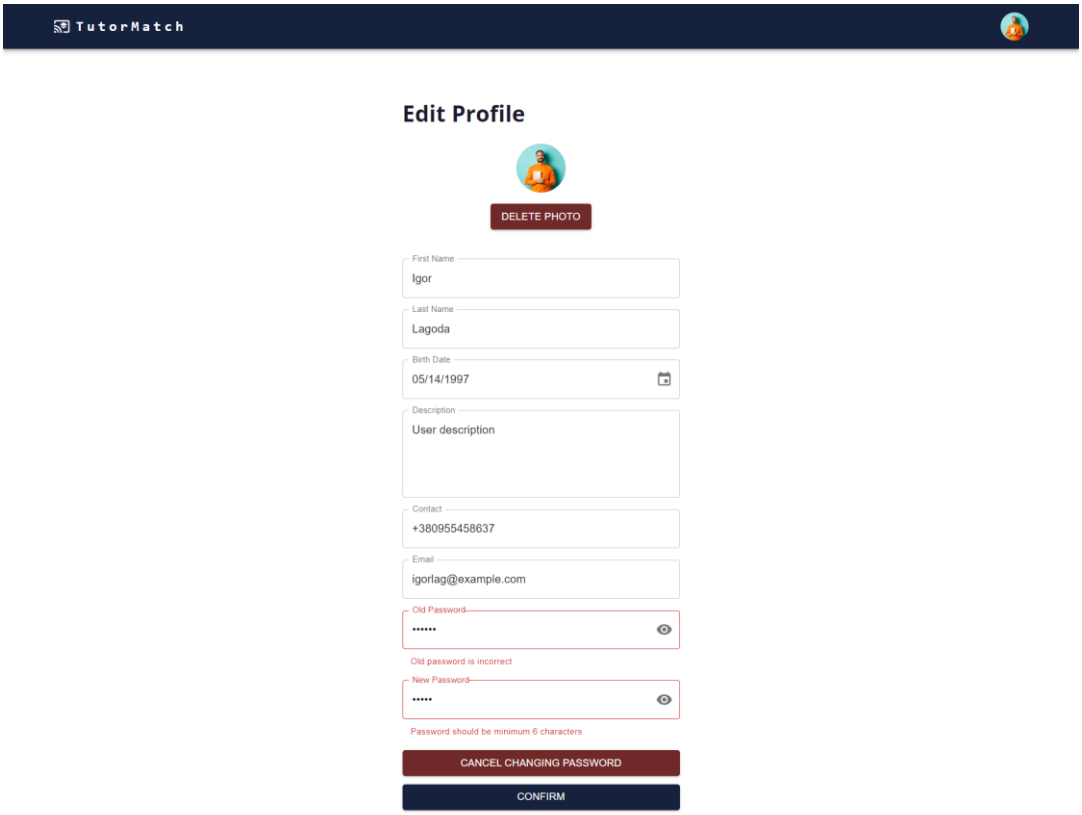

Рисунок 6.18 – Сторінка редагування профілю
із опцією змінення паролю - валідація

При вірному заповненню паролів валідація не виникає (рис. 6.19).

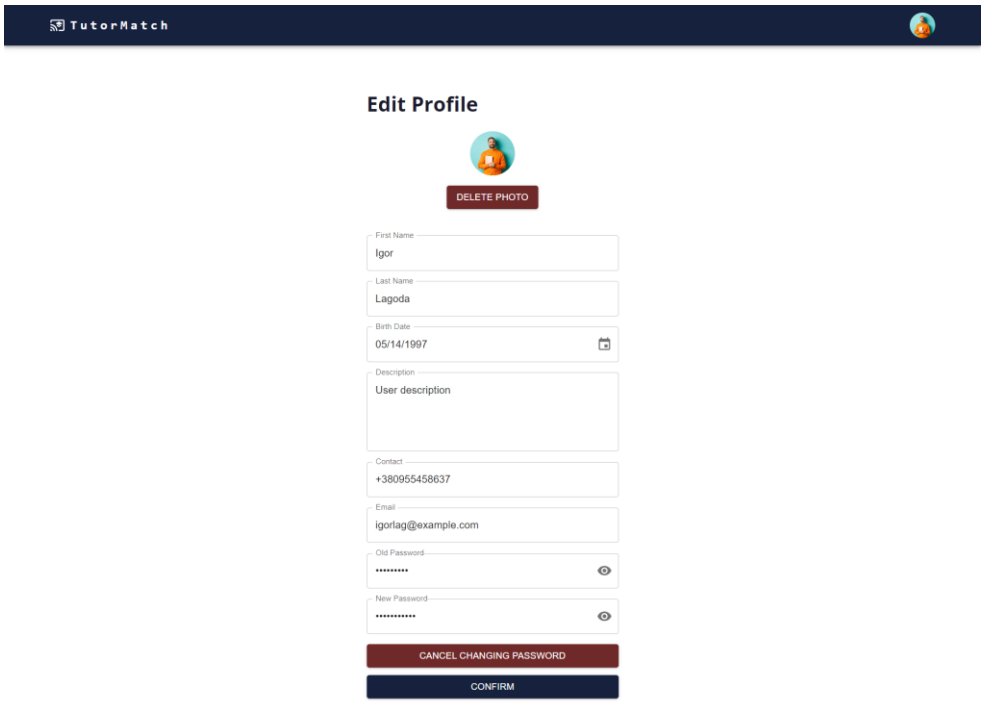

Рисунок 6.19 – Сторінка редагування профілю
із опцією змінення паролю з вірно заповненими даними

Наступним кроком необхідно перейти на сторінку графіку роботи (рис. 6.20).

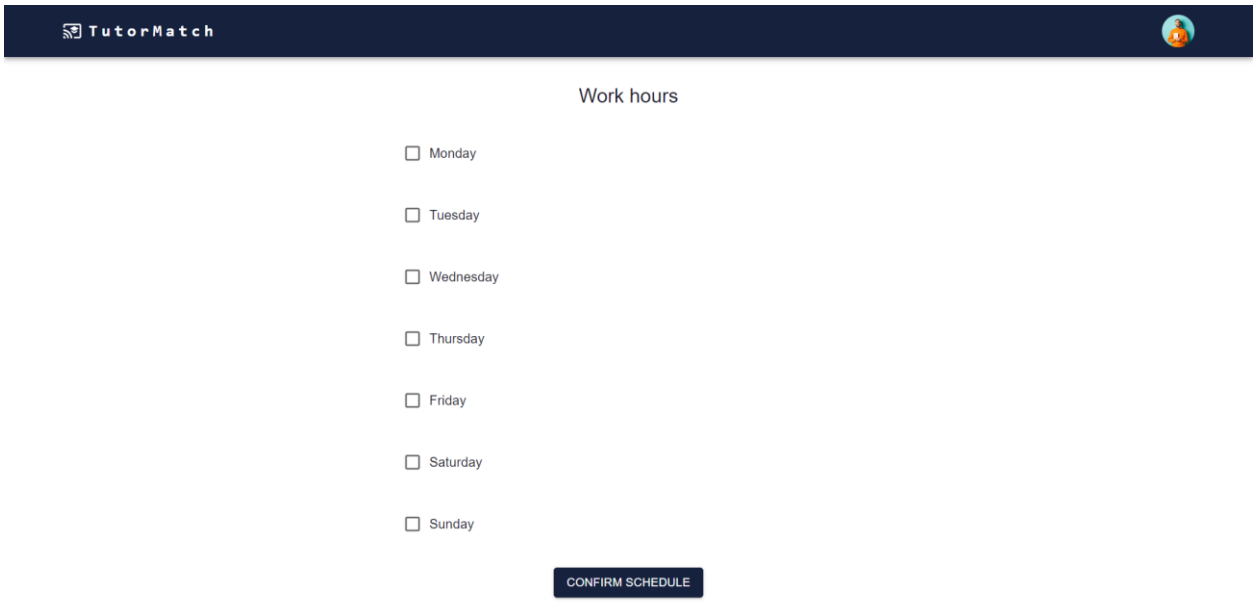

Рисунок 6.20 – Сторінка графіку роботи репетитора

На даній сторінці репетитор обирає робочі дні та час. За допомогою даному розкладу утворюються уроки на які студенти в подальшому можуть надіслати запит на заняття (рис. 6.21).

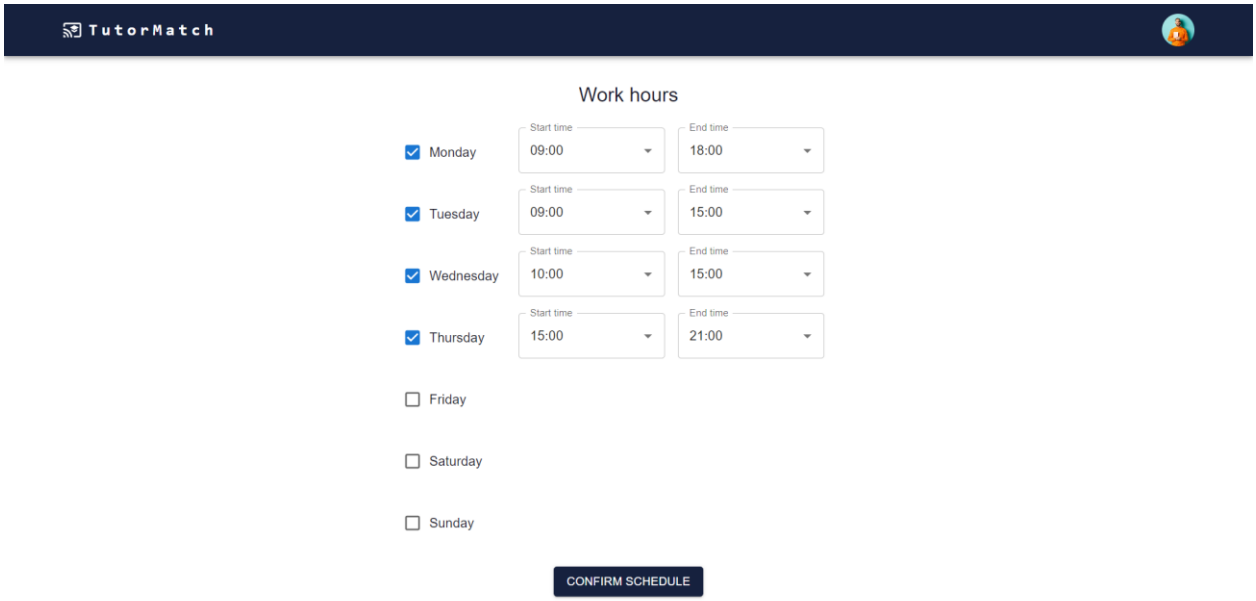

Рисунок 6.21 – Сторінка графіку роботи репетитора
із заповненми значенням

Перейшовши на сторінку занять бачимо два розділа: "Lessons Request" на якій знаходяться запити учнів та "Scheduled Lessons" де знаходяться вже прийняті запити на заняття (рис. 6.22). Як бачимо ніяких запитів та прийнятих занять немає. Тож наступним кроком зайдемо на сторінку студента щоб подати запити на заняття.

Рисунок 6.22 – Сторінка занять репетитора

Для того, щоб зайти на сторінку студента потрібно вийти зі сторінки репетитора та перейти на "Login" (рис. 6.23).

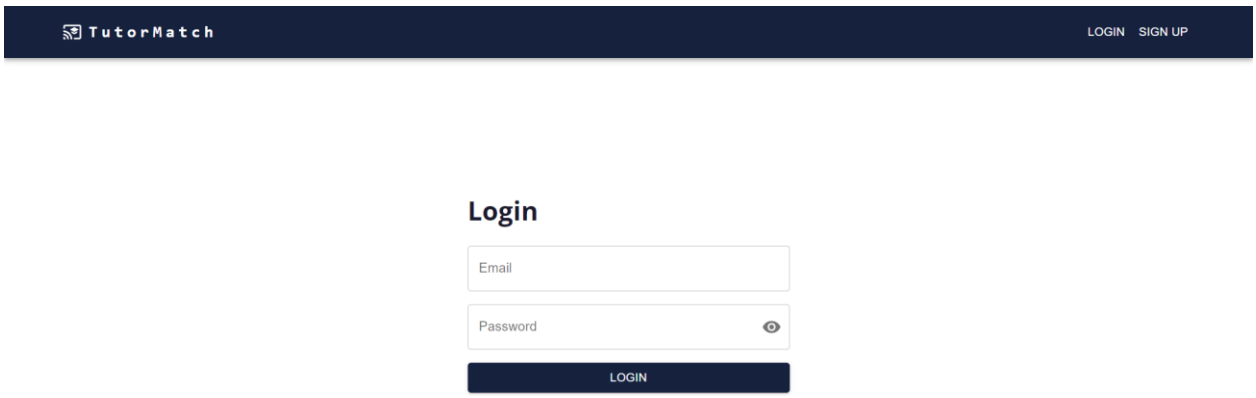

Рисунок 6.23 – Сторінка авторизації

Як можна побачити, валідація працює коректно і на сторінці авторизації (рис. 6.23).

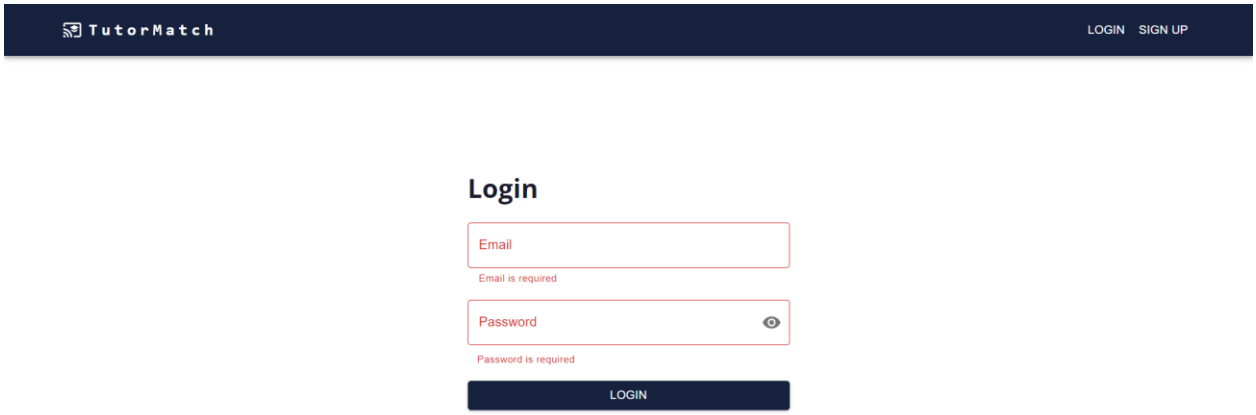

Рисунок 6.24 – Сторінка авторизації та її валідація

Вводимо валідні дані студента та тиснемо кнопку "Login" для авторизації на аккаунт студента (рис. 6.24).

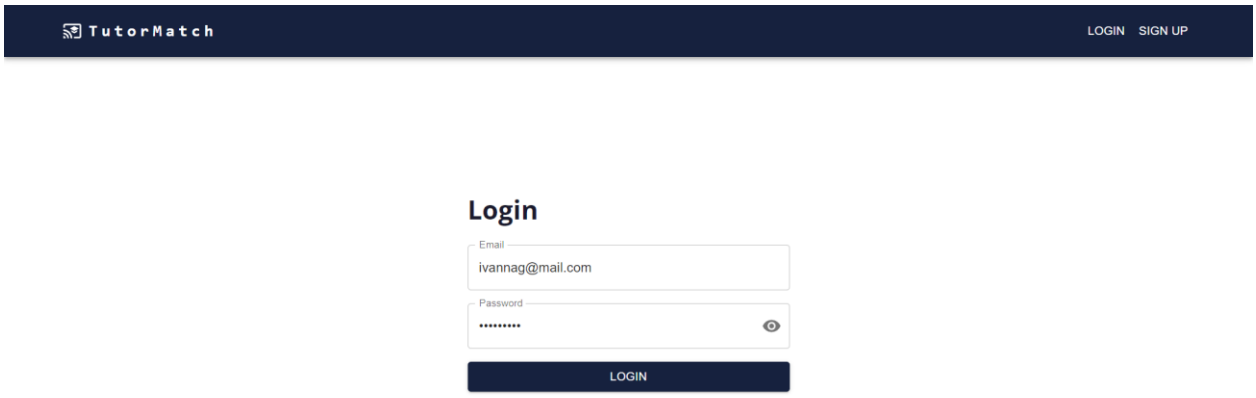

Рисунок 6.25 – Сторінка авторизації з веденими валідними даними

Після авторизації аналогічно до реєстрації користувача перекидує на сторінку його профіля (рис. 6.26). Редагування профілю та видалення аккаунту аналогічне до тих же функцій, як і у репетитора.

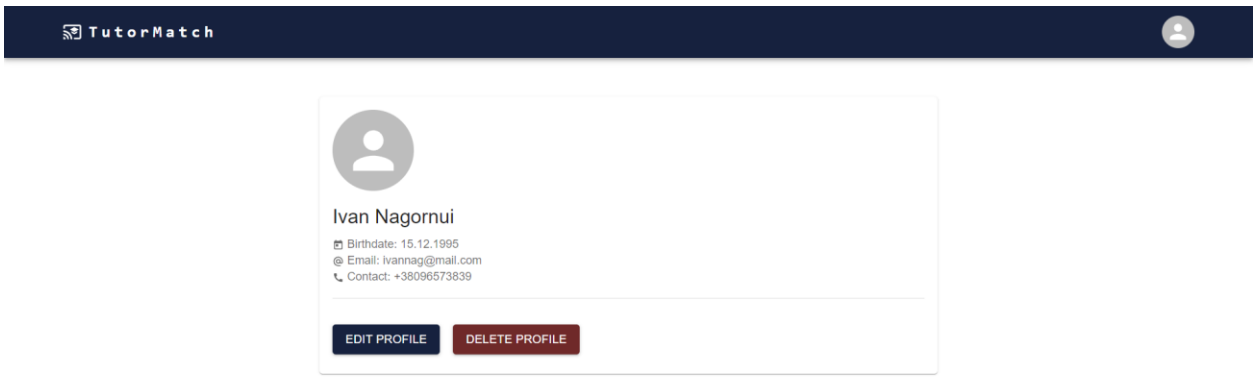

Рисунок 6.26 – Сторінка профілю студента

Перейдемо на головну сторінку (рис. 6.27).

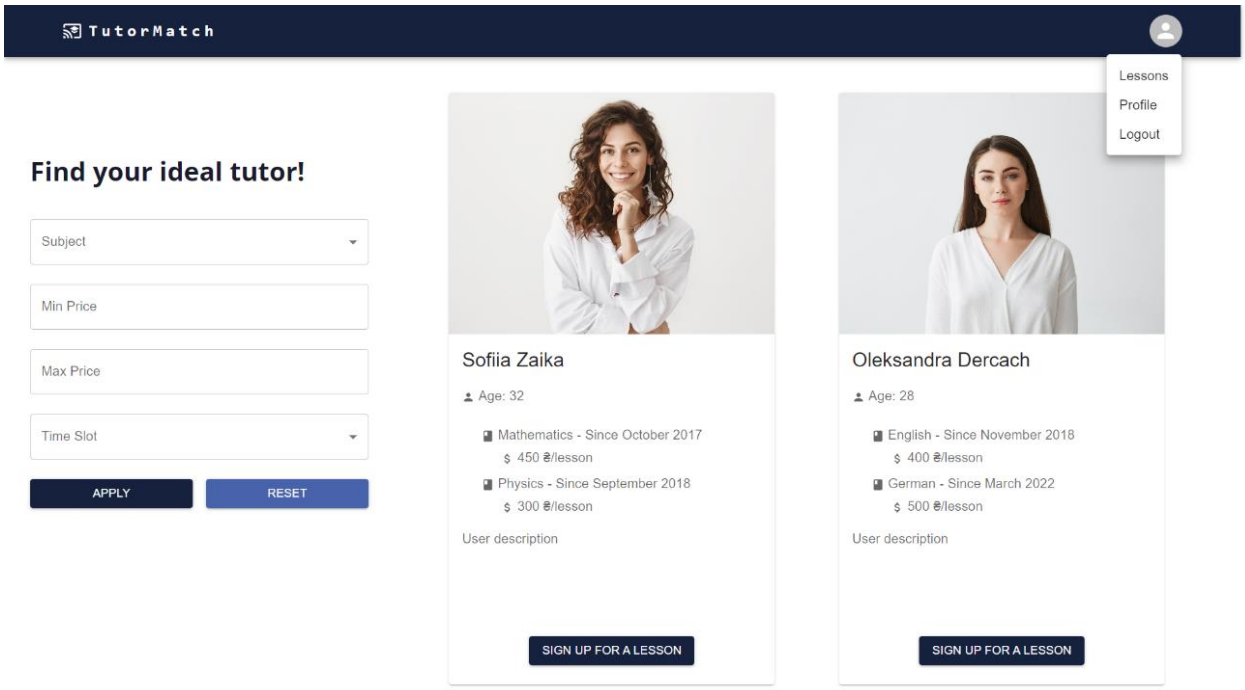

Рисунок 6.27 – Головна сторінка та
випадаюче меню студента

Скористаймося фільтрацією. У списку предметів оберемо "History" за необхідний предмет для студента та натиснемо "Apply" (рис. 6.28).

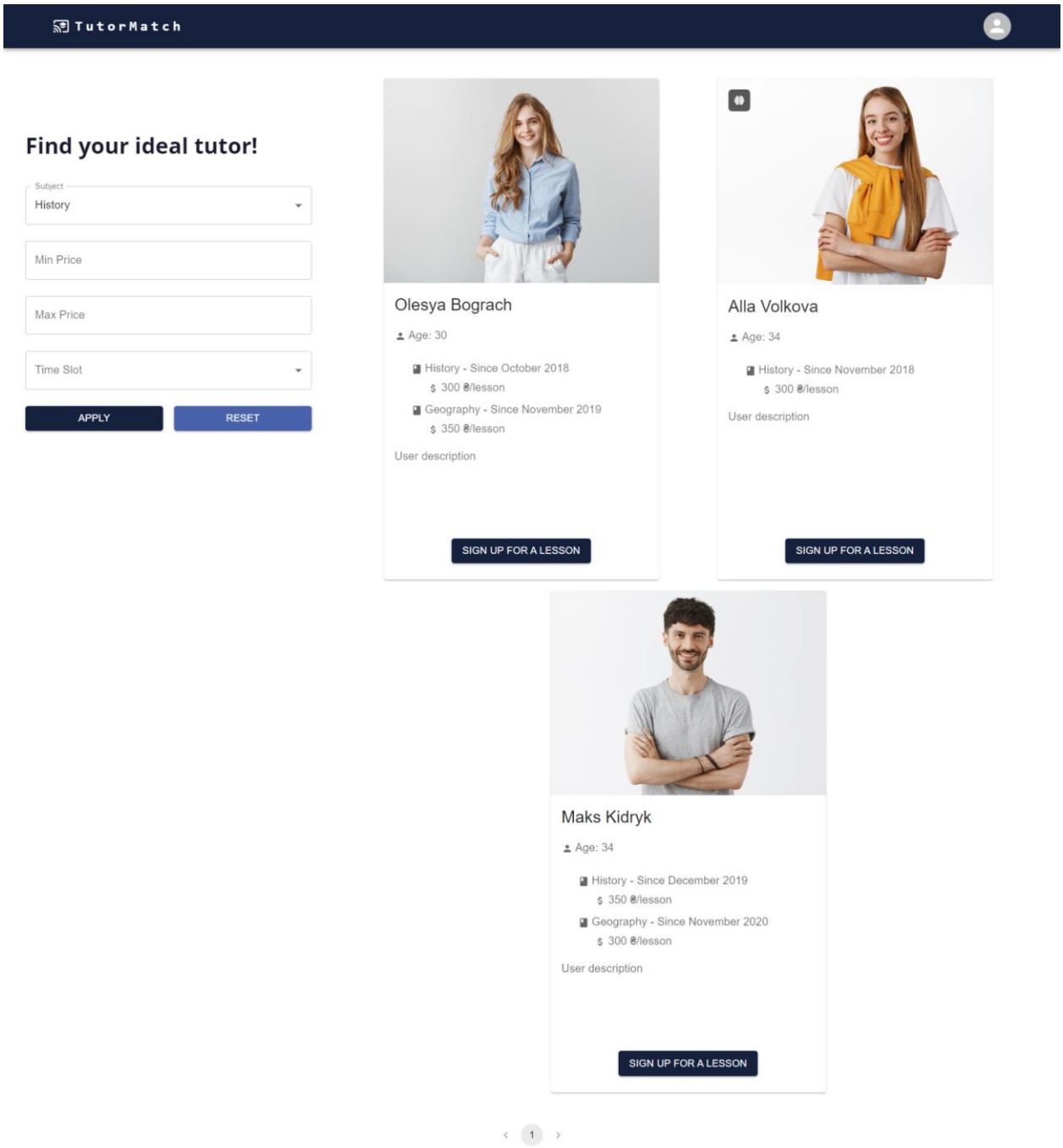

Рисунок 6.28 – Головна сторінка, фільтрація по предмету

Як можна побачити результат фільтрування є успішним. Тепер поставимо максимальну ціну предмету – 300 (рис. 6.29).

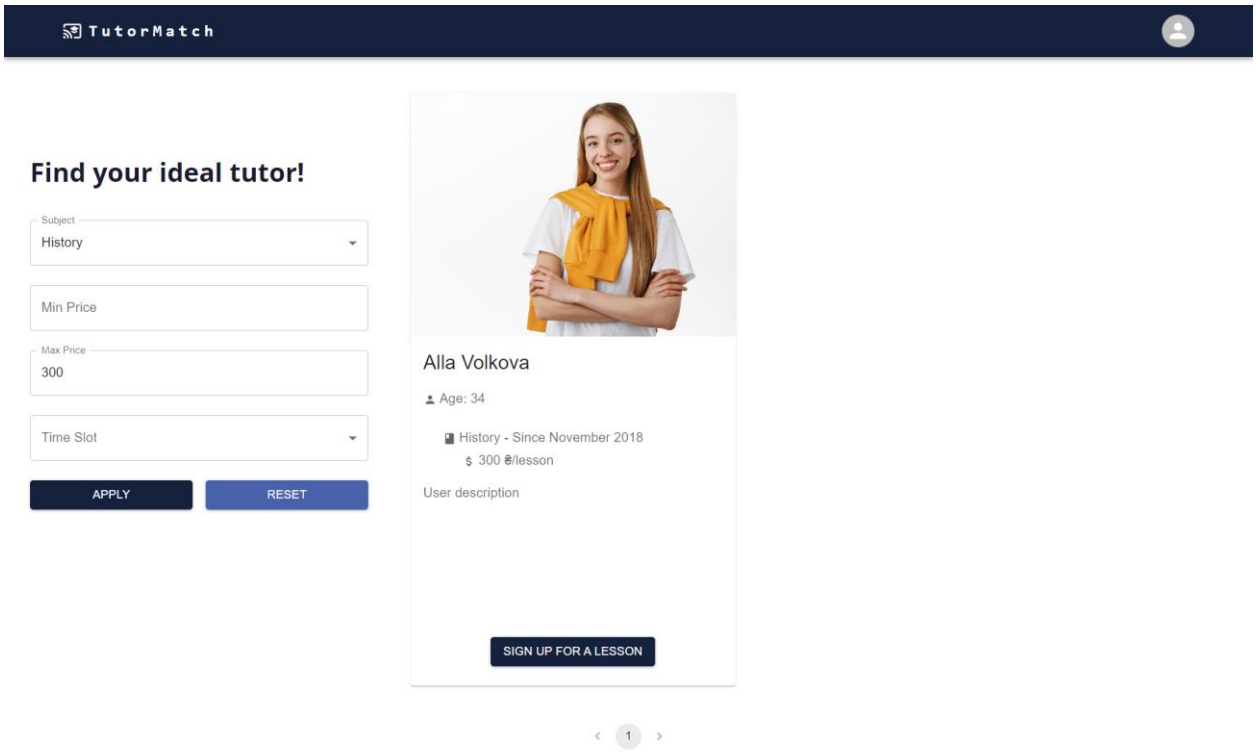

Рисунок 6.29 – Головна сторінка, фільтрація по предмету та максимальній ціні

Останнім кроком розглянемо фільтрацію за проміжками часу (рис. 6.30).

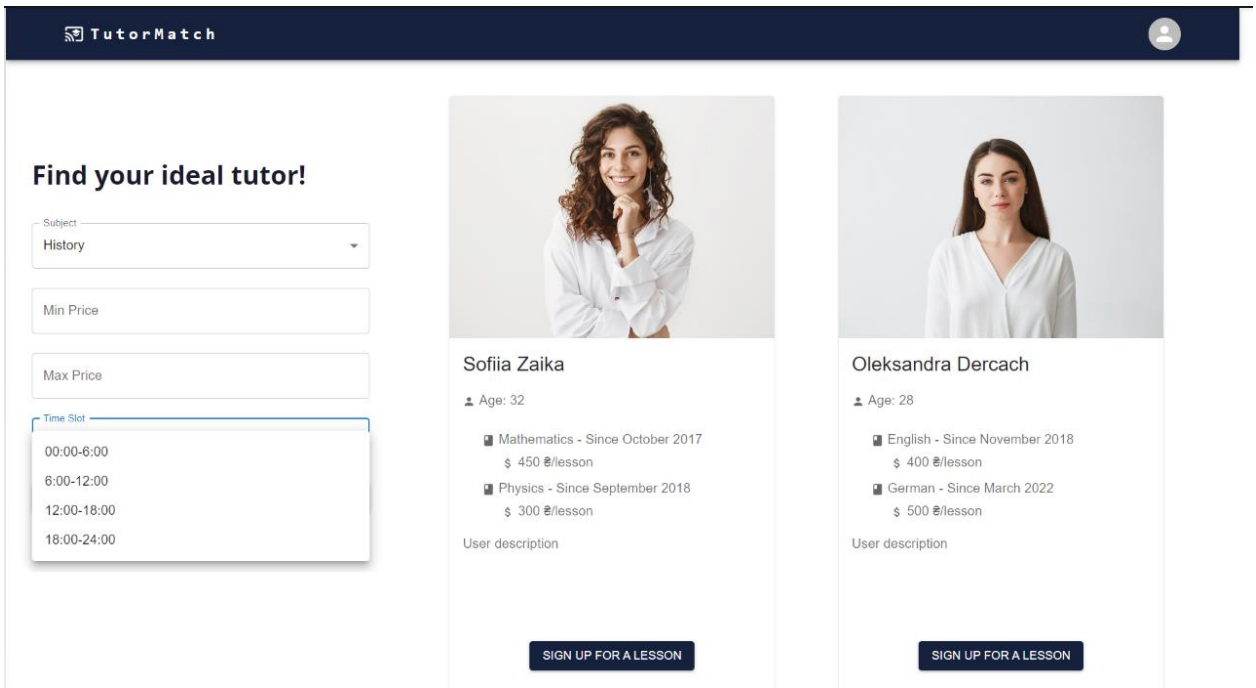

Рисунок 6.30 – Головна сторінка, вибір фільтрації по часу

Оберемо проміжки часу "12:00-18:00" та "00:00-06:00" та натиснемо "Apply" (рис. 6.31).

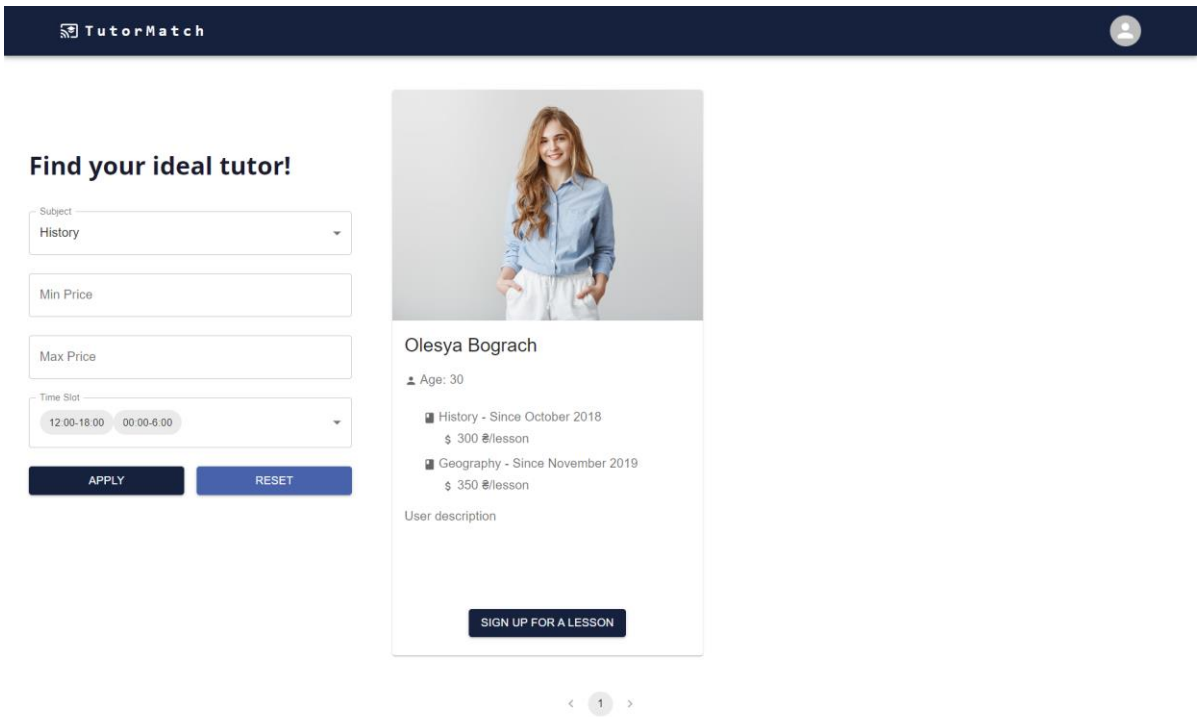

Рисунок 6.31 – Головна сторінка, фільтрація по предмету
та проміжку часу

Основною метою студента є запис на заняття, тож шукаємо репетитора якого створили раніше та натискаємо "Sign Up for a lesson" і обираємо необхідну дату (рис. 6.32).

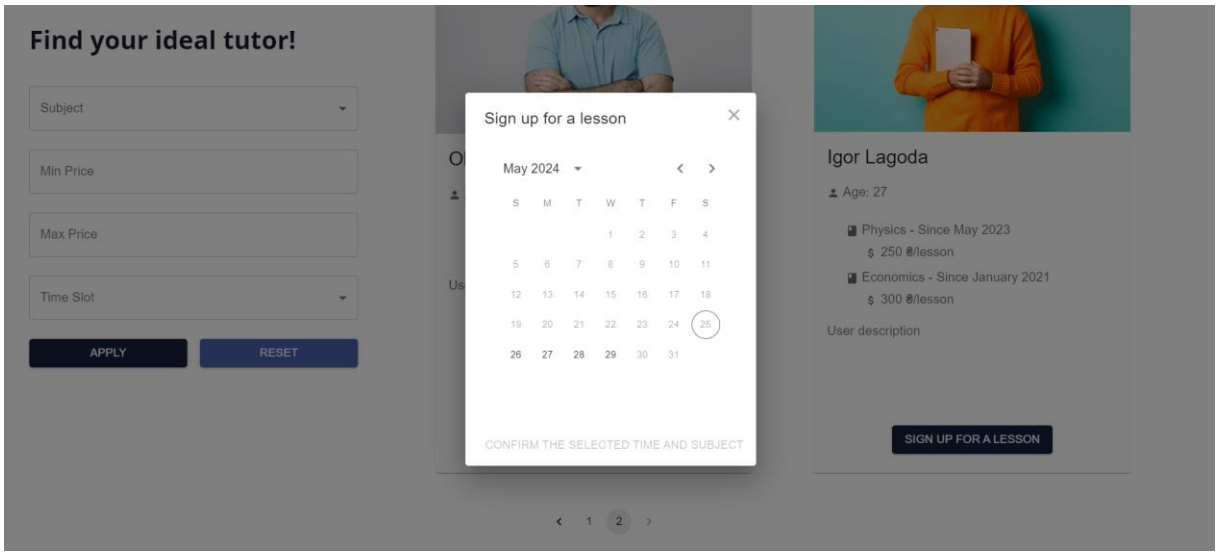

Рисунок 6.32 – Головна сторінка, запис на заняття,
вибір дати

Після вибору дати необхідно обрати час заняття (рис. 6.33).

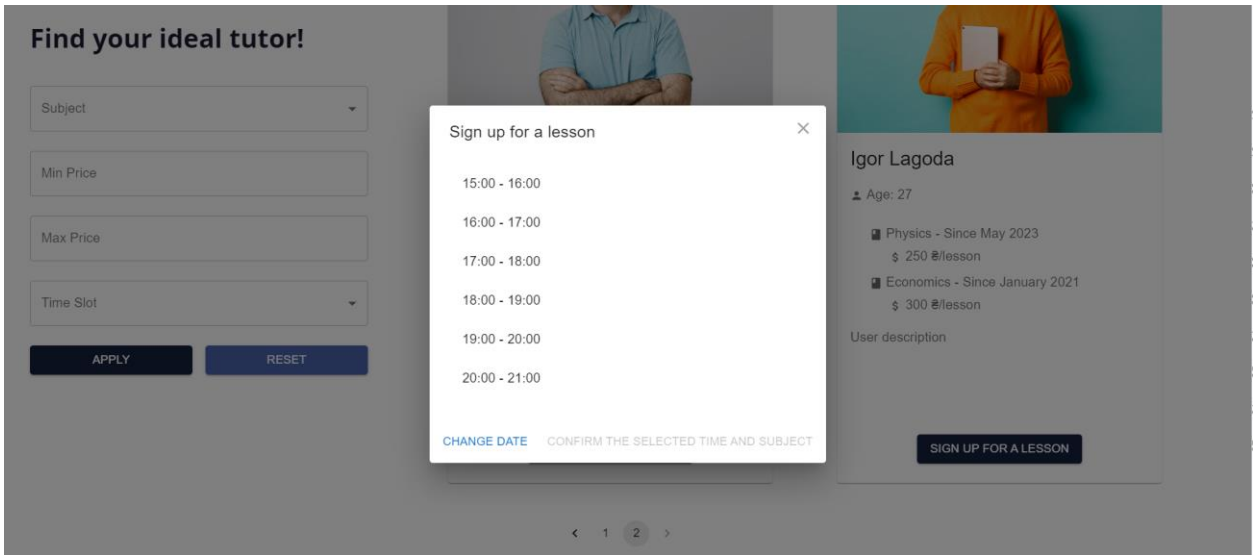

Рисунок 6.33 – Головна сторінка, запис на заняття,
вибір часу

Останнім кроком обирається предмет з якого буде проводитись заняття (рис. 6.34).

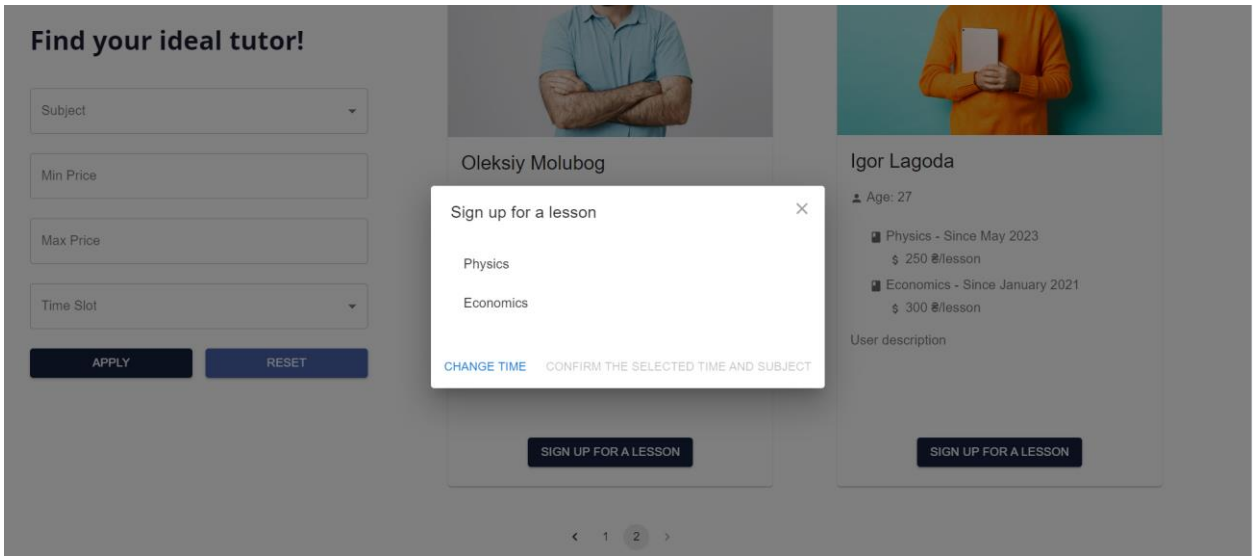

Рисунок 6.34 – Головна сторінка, запис на заняття,
вибір предмету

Після підтвердження предмету модальне вікно закривається та з’являється повідомлення про успішно надісланий запит на заняття (рис. 6.35). Аналогічно надішлемо ще 3 запити на заняття на: 2, 12 та 16 червня.

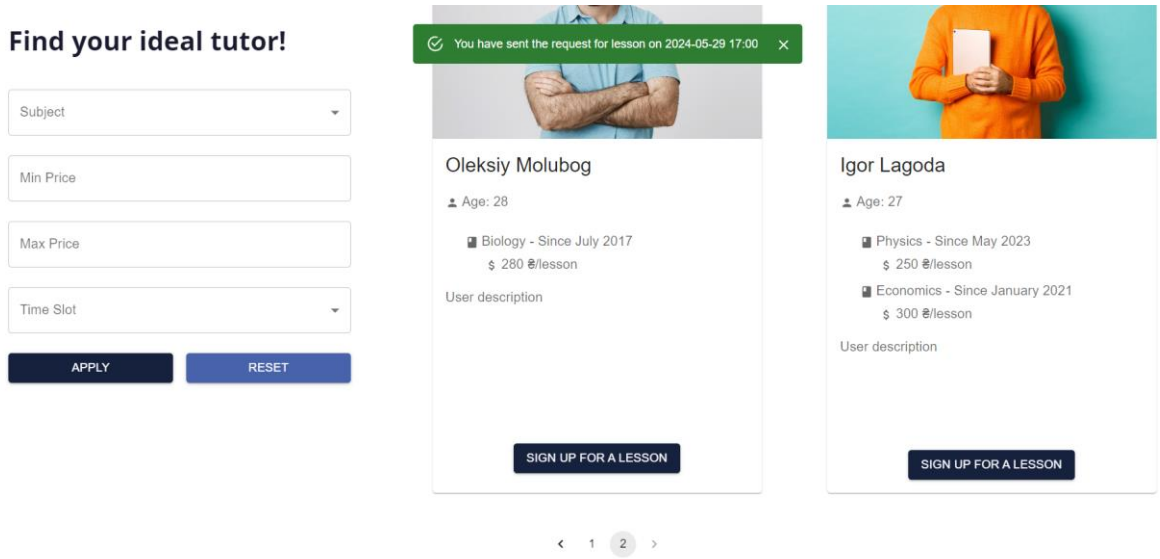

Рисунок 6.35 – Головна сторінка, запис на заняття, повідомлення про успішно відправлений запит на запис

Перейдемо на сторінку занять. На даній сторінці відображаються надіслані запити та вся необхіднна іннформація про викладача з можливістю відмінити запит (рис. 6.36).

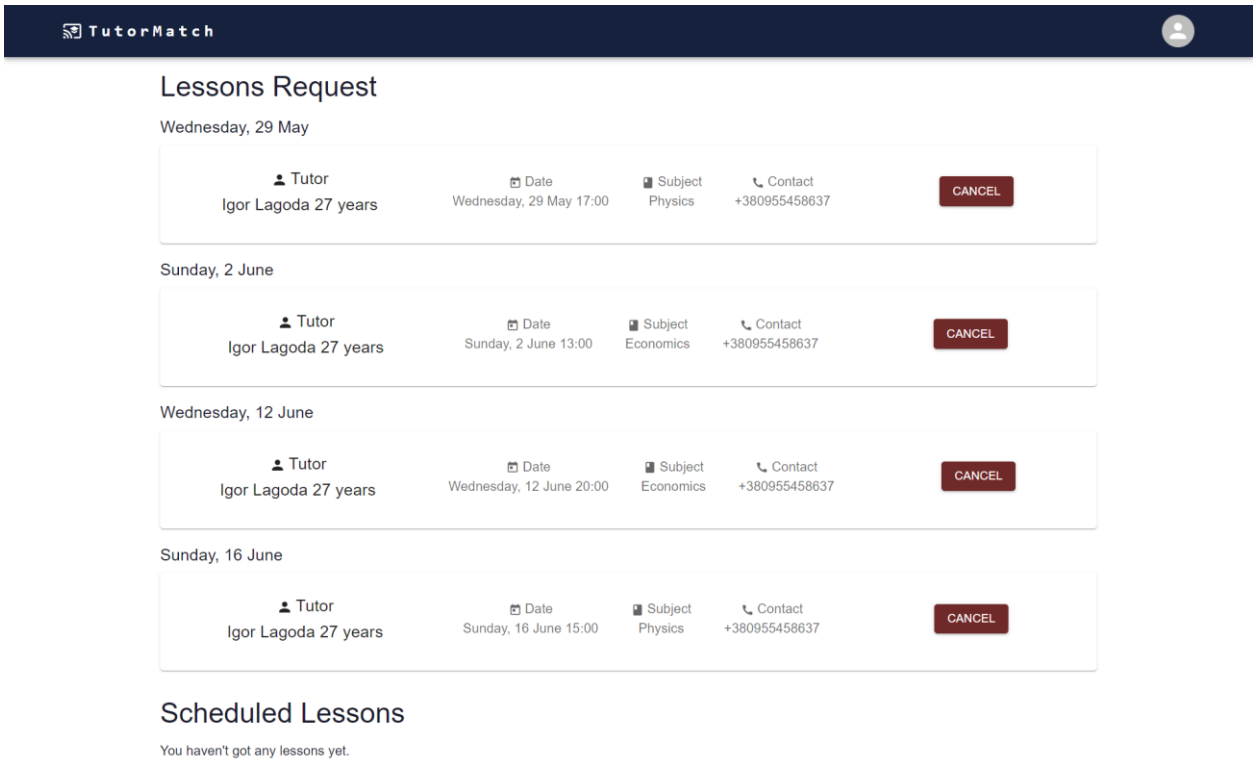

Рисунок 6.36 – Сторінка занять, список відправлених запитів на заняття студента

Авторизуємось як репетитор, якого було створено раніше та перейдемо на сторінку занять (рис. 6.37). Тепер на даній сторінці відображаються запити які було надіслано студентом.

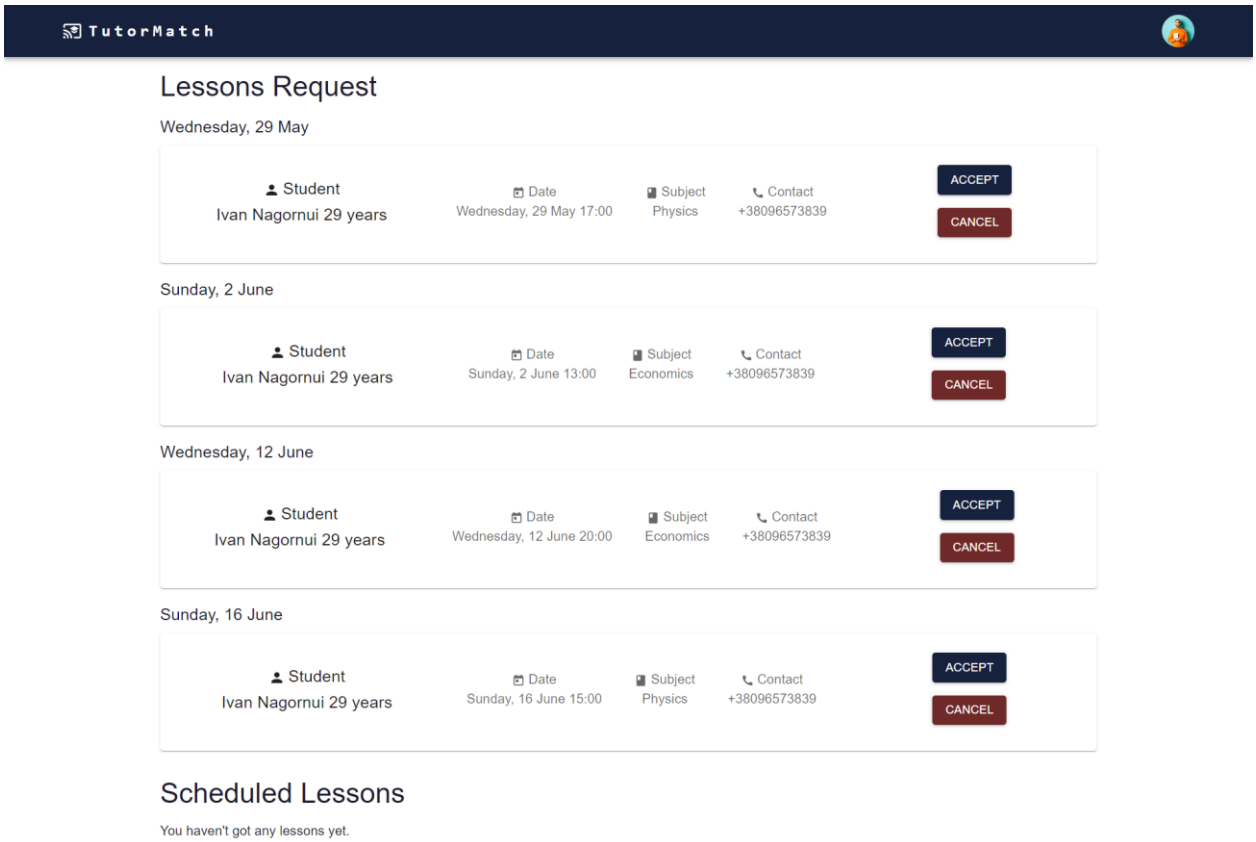

Рисунок 6.37 – Сторінка занять,
список запитів на заняття репетитора

Приймемо два запита за 29 травня та 12 червня (рис. 6.38). Дані запити перемістились до списку вже прийнятих занять.

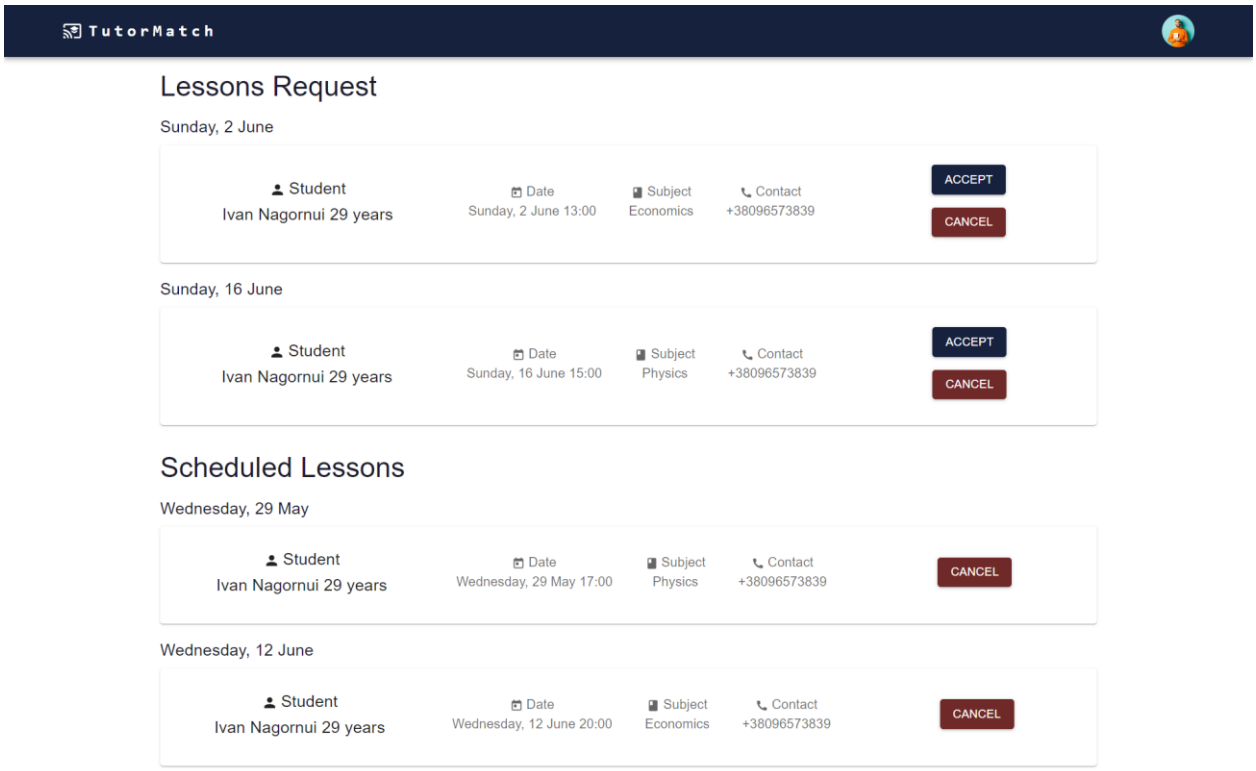

Рисунок 6.38 – Сторінка занять,
список запитів на заняття репетитора з двома прийнятими запитами

Запит на заняття за 2 червня видалимо (рис. 6.39).

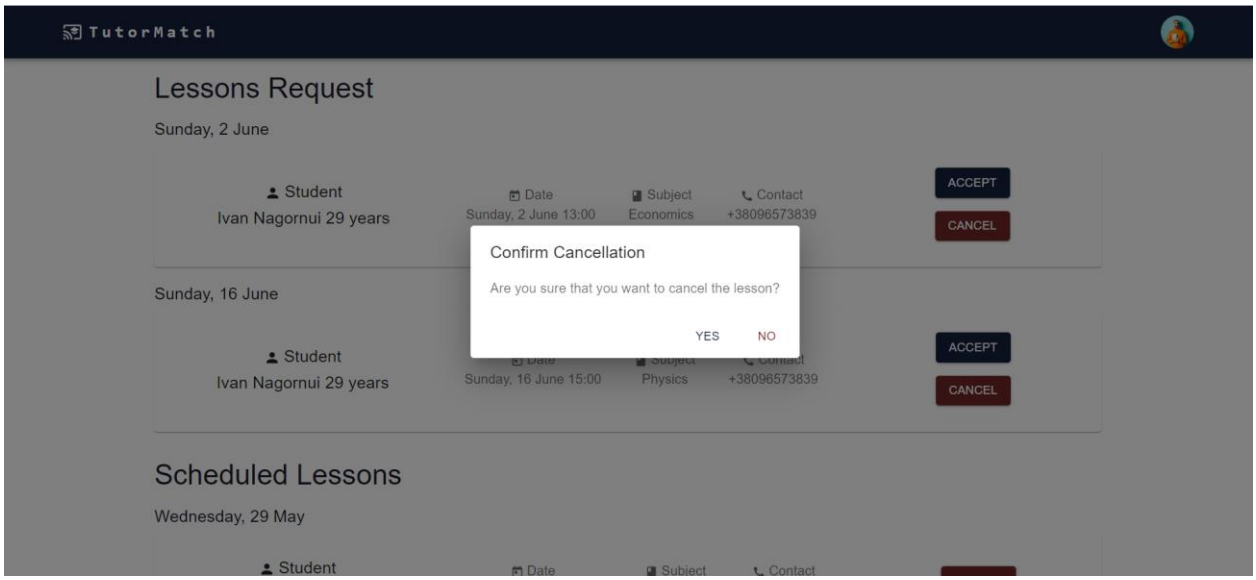

Рисунок 6.39 – Сторінка занять,
видалення запиту на заняття

Як результат даного запиту більше немає в переліку занять (рис. 6.40).

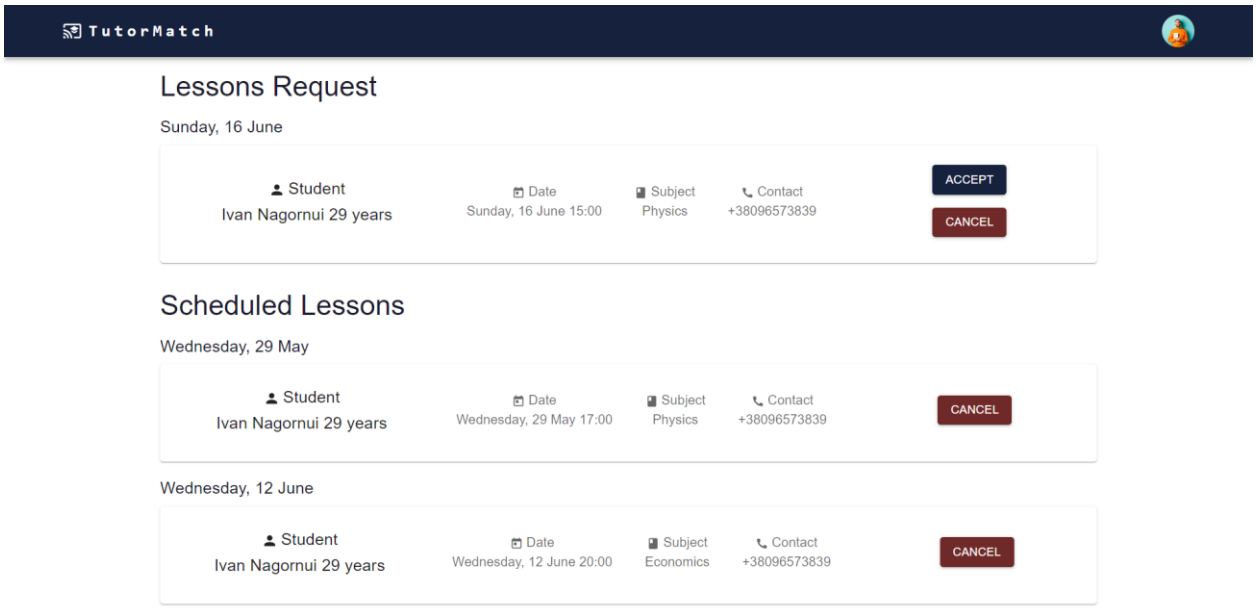

Рисунок 6.40 – Сторінка занять,
успішне видалення запиту на заняття

Останнім кроком авторизовуємось як студент та перевіряємо сторінку занять (рис. 6.41).

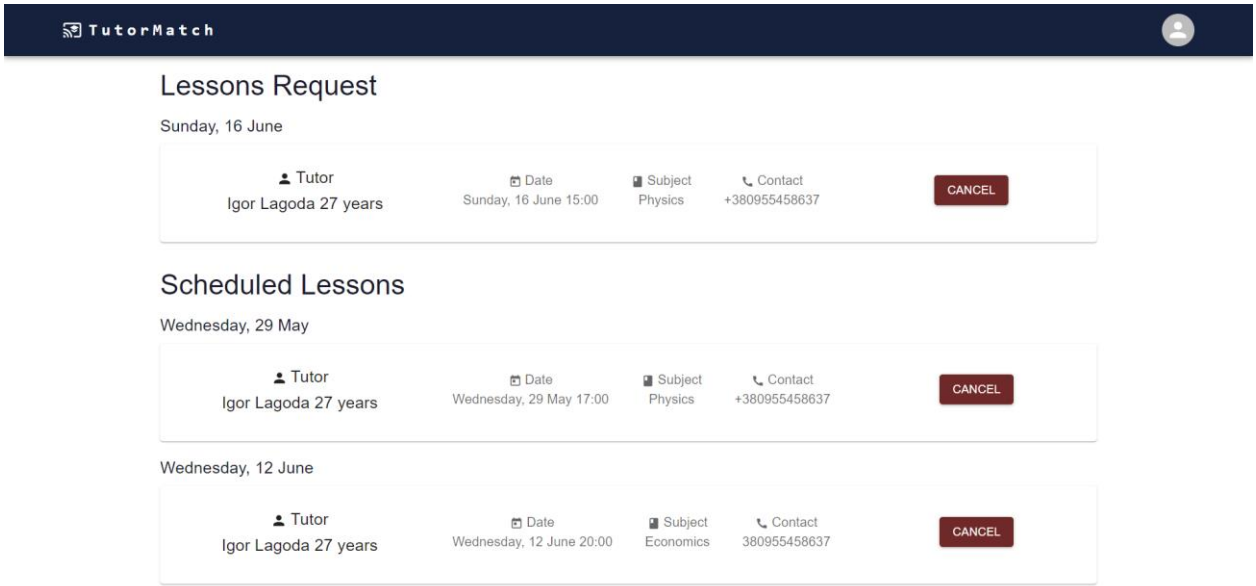

Рисунок 6.41 – Сторінка занять студента,
після прийняття та видалення запиту репетитором

Отже, всі запити та дії з ними відображаються коректно як для репетитора та студента. Даний сайт виконує необхідний функціонал як і передбачалось в умовах.

ВИСНОВКИ

Під час виконання даної роботи була розроблена інформаційна система для надання послуг репетиторства, реалізація якої представлена у вигляді веб-платформи. Основна мета проекту полягала у створенні зручного та ефективного інструменту, який сприятиме взаємодії між репетиторами та учнями, а також полегшить пошук репетиторів за предметами та критеріями ціни та досвіду.

У ході дипломної роботи було розглянуто численні джерела інформації з розробки веб-додатків, методологій, шаблонів і технологій. Було проведено аналіз цих даних, який визначив вибір мови програмування, методології та допоміжних інструментів. Було проаналізовано найбільш популярні аналоги для врахування всіх переваг і недоліків конкурентів та визначити сильні сторони продуктів, що розроблялись в той час.

Були розроблені такі діаграми Entyty relationship diagram та Use case diagram для подальшої реалізації логіки.

Для реалізації проекту були обрані ефективні програмні засоби: React JS для frontend-частини та Nest JS для backend-частини, PostgreSQL для зберігання даних, Docker для контейнеризації, Material UI для розробки інтерфейсів користувача та Swagger для взаємодії з API через графічний інтерфейс. Вибір цих технологій допоміг створити надійну, масштабовану та продуктивну веб-платформу. Для створення дизайну було використано такий інструмент як Figma, що допомогло в розробці сайту.

Завдяки створенню даного проекту були набуті важливі навички у розробці веб-додатків, роботі з реляційними базами даних, використанні сучасних технологій розробки, а також у розробці ефективних алгоритмів для обробки даних та покращення користувацького досвіду.

Отже, результатом даної роботи є розроблений веб-додаток, який надає ефективні та зручні інструменти для взаємодії між репетиторами та учнями.

СПИСОК ЛІТЕРАТУРИ

1. New U.S. data show jump in college students' learning online. Inside Higher Ed | Higher Education News, Events and Jobs. [Електронний ресурс] — Режим доступу: https://www.insidehighered.com/news/2021/10/13/new-us-data-show-jump-college-students-learning-online (дата звернення: 03.04.2024).
2. Online Education Statistics - 2024 (Before & After Covid-19). Admissonsly. [Електронний ресурс] — Режим доступу: https://admissionsly.com/online-education-statistics/ (дата звернення: 03.04.2024).
3. Preply. Сайт з пошуку репетиторів. [Електронний ресурс] — Режим доступу: https://preply.com/ua/ (дата звернення: 04.04.2024).
4. Cererra. Cererra | Сайт Репетиторів України | Знайти Професійного Вчителя та Репетитора. [Електронний ресурс] — Режим доступу: https://cererra.com/ (дата звернення: 04.04.2024).
5. Букі | Buki - ваш репетитор з будь-якого предмету. Репетитори України. [Електронний ресурс] — Режим доступу: https://buki.com.ua/ (дата звернення: 04.04.2024).
6. Morgan J. How to code in React.js. New York, 2021. - 961 p. (1)
7. Buna S. React succinctly. Morrisville, 2019. - 119 p.
8. Ritter K. NestJS Build a RESTful CRUD API. 2021. - 37 p.
9. Nest.js: A Progressive Node.js Framework / J. Bell et al. 2018. - 408 p.
10. Stones R., Matthew N. Beginning Databases with PostgreSQL From Novice to Professional. 2nd ed. 2005. - 637 p.
11. Introduction to Containerization and Docker · The Docker Handbook. Choose a language · The Docker Handbook. [Електронний ресурс] — Режим доступу: https://docker-handbook.farhan.dev/en/introduction-to-containerization-and-docker.html (дата звернення: 21.04.2024).
12. Docker Cookbook - Second Edition. Packt Subscription | Advance your knowledge in tech. [Електронний ресурс] — Режим доступу:

https://subscription.packtpub.com/book/cloud-and-networking/9781788626866/1/ch01lvl1sec02/introduction (дата звернення: 21.04.2024).

13. Overview - Material UI. MUI: The React component library you always wanted. [Електронний ресурс] — Режим доступу: https://mui.com/material-ui/getting-started/ (дата звернення: 23.04.2024).
14. OpenAPI Specification - Version 3.1.0 | Swagger. API Documentation & Design Tools for Teams | Swagger. [Електронний ресурс] — Режим доступу: https://swagger.io/specification/ / (дата звернення: 23.04.2024).
15. Swagger eBook. Learn programming languages with books and examples. [Електронний ресурс] — Режим доступу: https://riptutorial.com/ebook/swagger (дата звернення: 01.05.2024).
16. Figma. Дизайн роботи для сайту TutorMatch [Електронний ресурс] — Режим доступу: https://www.figma.com/design/S8gwAqc4JPD6t82J0pgxk0/Tutors?node-id=0:1&t=wC1qVTaOnFI2qPkc-1 (дата звернення: 05.05.2024).
17. What is Entity Relationship Diagram (ERD)?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. [Електронний ресурс] — Режим доступу: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/;WWWSESSIONID=A96C04A8AD1F3C8846BB07D4EB16CFA0.www1 (дата звернення: 07.05.2024).
18. UML Use Case Diagram Tutorial. Lucidchart. [Електронний ресурс] — Режим доступу: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 07.05.2024).
19. Docker: Accelerated Container Application Development. Docker. [Електронний ресурс] — Режим доступу: https://www.docker.com/ (дата звернення: 09.05.2024).
20. Node.js – Download Node.js®. Node.js – Run JavaScript Everywhere. [Електронний ресурс] — Режим доступу:

https://nodejs.org/en/download/package-manager (дата звернення: 11.05.2024).

ДОДАТОК. БЕКЕНД

Реалізація App module:

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { RolesGuard } from './auth/roles.guard';
import { APP_GUARD } from '@nestjs/core';
import { SubjectsModule } from './subjects/subjects.module';
import { LessonsModule } from './lessons/lessons.module';
import { MulterModule } from '@nestjs/platform-express';
import { ImageModule } from './image/image.module';

@Module({
 imports: [
  TypeOrmModule.forRoot({
   type: 'postgres',
   host: '127.0.0.1',
   port: 5432,
   username: 'postgres',
   password: 'postgres',
   database: 'tutors',
   entities: [__dirname + '/**/*.entity.{js,ts}'],
   synchronize: true,
  }),
  UsersModule,
  AuthModule,
  MulterModule.register({
   dest: './upload',
  }),
  ImageModule,
  ConfigModule.forRoot({
   isGlobal: true,
   envFilePath: '.env',
  }),
  SubjectsModule,
  LessonsModule,
 ],
 controllers: [AppController],
 providers: [
  {
   provide: APP_GUARD,
   useClass: RolesGuard,
  },
 ],
})
export class AppModule {}
```

Реалізація сутностей:

user.entity.ts

```
import { Entity, Column, PrimaryGeneratedColumn, OneToMany } from 'typeorm';
import { Lesson } from '../lessons/lesson.entity';
import { Subject } from '../subjects/subject.entity';
import { Exclude } from 'class-transformer';
```

```
export enum UserRole {
  Tutor = 'Tutor',
  Student = 'Student',
}

interface DaySchedule {
  active: boolean;
  from: string;
  to: string;
}

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({
    type: 'enum',
    enum: UserRole,
  })
  role: UserRole;

  @Column()
  firstName: string;

  @Column()
  lastName: string;

  @Column()
  birthDate: Date;

  @Column()
  description: string;

  @Column({ unique: true })
  email: string;

  @Column({ nullable: true })
  profilePhoto?: string;

  @Column()
  @Exclude()
  password: string;

  @Column({ type: 'jsonb', nullable: true })
  schedule: {
    monday: DaySchedule;
    tuesday: DaySchedule;
    wednesday: DaySchedule;
    thursday: DaySchedule;
    friday: DaySchedule;
    saturday: DaySchedule;
    sunday: DaySchedule;
  };

  @OneToMany(() => Lesson, (lesson) => lesson.tutor)
  tutorLessons: Lesson[];

  @OneToMany(() => Lesson, (lesson) => lesson.student)
  studentLessons: Lesson[];
```

```
  @OneToMany(() => Subject, (subject) => subject.tutor)
  tutorSubject: Lesson[];

  @OneToMany(() => Subject, (subject) => subject.tutor, { cascade: true })
  subjects: Subject[];
}
```

subject.entity.ts

```
import { IsNotEmpty, IsEnum, IsNumber, IsDate } from 'class-validator';
import { SubjectName } from '../subject.entity';

export class CreateSubjectDto {
  @IsNotEmpty()
  @IsEnum(SubjectName)
  name: SubjectName;

  @IsNotEmpty()
  @IsNumber()
  pricePerLesson: number;

  @IsNotEmpty()
  @IsDate()
  experienceSince: Date;
}
```

lesson.entity.ts

```
import {
  Entity,
  Column,
  PrimaryGeneratedColumn,
  ManyToOne,
  JoinColumn,
} from 'typeorm';
import { User } from '../users/user.entity';
import { Subject } from '../subjects/subject.entity';

export enum LessonStatus {
  WaitingApproval = 'WaitingApproval',
  Scheduled = 'Scheduled',
  Canceled = 'Canceled',
}

@Entity()
export class Lesson {
  @PrimaryGeneratedColumn()
  id: number;

  @ManyToOne(() => User, (user) => user.tutorLessons)
  @JoinColumn({ name: 'tutorId' })
  tutor: User;

  @ManyToOne(() => User, (user) => user.studentLessons)
  @JoinColumn({ name: 'studentId' })
  student: User;

  @ManyToOne(() => Subject)
  @JoinColumn({ name: 'subjectId' })
  subject: Subject;
```

```
  @Column()
  date: Date;

  @Column()
   accepted: boolean;

  @Column({
    type: 'enum',
    enum: LessonStatus,
    default: LessonStatus.WaitingApproval,
  })
  status: LessonStatus;
}
```

Реалізація авторизації:

login.dto.ts

```
import { IsEmail, IsNotEmpty, IsString, MinLength } from 'class-validator';
import { ApiProperty } from '@nestjs/swagger';

export class LoginDto {
  @ApiProperty({ default: 'example1@gmail.com' })
  @IsNotEmpty()
  @IsEmail({}, { message: 'Please enter correct email' })
  email: string;

  @ApiProperty({ default: 'SomePass123' })
  @IsNotEmpty()
  @IsString()
  @MinLength(6)
  password: string;
}
```

signup.dto.ts

```
import {
  IsArray,
  IsEmail,
  IsEnum,
  IsNotEmpty,
  IsOptional,
  IsString,
  MinLength,
  ValidateNested,
} from 'class-validator';
import { ApiProperty } from '@nestjs/swagger';
import { RolesEnum } from '../../users/interfaces/roles.enum';
import { CreateSubjectDto } from '../../subjects/dto/create-subject.dto';
import { Type } from 'class-transformer';
import { Binary } from 'typeorm';

export class SignUpDto {
  @ApiProperty({ default: 'Sonya' })
  @IsNotEmpty()
  firstName: string;

  @ApiProperty({ default: 'Zaika' })
  @IsNotEmpty()
  lastName: string;
```

```
  @ApiProperty()
  @IsOptional()
  profilePhoto?: string;

  @ApiProperty({ default: new Date().toString() })
  @IsNotEmpty()
  @IsString()
  birthDate: Date;

  @ApiProperty({ default: 'some description' })
  @IsNotEmpty()
  @IsString()
  description: string;

  @ApiProperty({ default: 'example1@gmail.com' })
  @IsNotEmpty()
  @IsEmail({}, { message: 'Please enter correct email' })
  email: string;

  @ApiProperty({ default: RolesEnum.Tutor })
  @IsNotEmpty()
  @IsEnum(RolesEnum, { message: 'Role must be either Tutor or Student' })
  role: RolesEnum;

  @ApiProperty({ default: 'SomePass123' })
  @IsNotEmpty()
  @IsString()
  @MinLength(6, { message: 'Password must be at least 6 characters long' })
  password: string;

  @ApiProperty()
  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => CreateSubjectDto)
  subjects: CreateSubjectDto[];
}
```

auth.controller.ts

```
import {
  Body,
  Controller,
  Post,
  UploadedFile,
  UseGuards,
  UseInterceptors,
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { LoginDto } from './dto/login.dto';
import { SignUpDto } from './dto/signup.dto';
import { AuthGuard } from '@nestjs/passport';
import { ApiTags } from '@nestjs/swagger';

@ApiTags('auth')
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post('/signup')
  signUp(@Body() signUpDto: SignUpDto): Promise<{ token: string }> {
    return this.authService.signUp(signUpDto);
```

```
  }

  @UseGuards(AuthGuard('local'))
  @Post('/login')
  login(@Body() loginDto: LoginDto): Promise<{ token: string }> {
    return this.authService.login(loginDto);
  }
}
```

auth.module.ts

```
import { Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from '../users/user.entity';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { JwtStrategy } from './jwt.strategy';
import { LocalStrategy } from './local.strategy';

@Module({
  imports: [
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule.registerAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: (config: ConfigService) => {
        return {
          secret: config.get<string>('JWT_SECRET'),
          signOptions: {
            expiresIn: config.get<string | number>('JWT_EXPIRES'),
          },
        };
      },
    }),
    TypeOrmModule.forFeature([User]),
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy, LocalStrategy],
  exports: [JwtStrategy, PassportModule],
})
export class AuthModule {}
```

auth.service.ts

```
import {
  HttpException,
  HttpStatus,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User, UserRole } from '../users/user.entity';
import * as bcrypt from 'bcryptjs';
import { JwtService } from '@nestjs/jwt';
import { SignUpDto } from './dto/signup.dto';
import { LoginDto } from './dto/login.dto';

@Injectable()
export class AuthService {
```

```
constructor(
  @InjectRepository(User)
  private usersRepository: Repository<User>,
  private jwtService: JwtService,
) {}

public async validateUser(decoded: any): Promise<User> {
  return this.usersRepository.findOne(decoded.id);
}

async validate(token: string): Promise<boolean | never> {
  const decoded: unknown = this.jwtService.verify(token);

  if (!decoded) {
    throw new HttpException('Forbidden', HttpStatus.FORBIDDEN);
  }

  const user: User = await this.validateUser(decoded);

  if (!user) {
    throw new UnauthorizedException();
  }

  return true;
}

async signUp(signUpDto: SignUpDto): Promise<{ token: string }> {
  const dbUser = await this.usersRepository.findOne({
    where: { email: signUpDto.email },
  });
  console.log(dbUser);
  if (dbUser) {
    throw new HttpException('User already exists', HttpStatus.BAD_REQUEST);
  }

  console.log(signUpDto.subjects);

  const hashedPassword = await bcrypt.hash(signUpDto.password, 10);
  const user = await this.usersRepository.save({
    ...signUpDto,
    role: UserRole.Student,
    password: hashedPassword,
    subjects: signUpDto.subjects,
  });

  const token = this.jwtService.sign({ id: user.id });

  return { token };
}

async login(loginDto: LoginDto): Promise<{ token: string }> {
  const { email, password } = loginDto;

  const user = await this.usersRepository.findOne({
    where: { email },
  });

  if (!user) {
    throw new UnauthorizedException('Invalid email or password');
  }

  const isPasswordMatched = await bcrypt.compare(password, user.password);
```

```
    if (!isPasswordMatched) {
      throw new UnauthorizedException('Invalid email or password');
    }

    const token = this.jwtService.sign({ id: user.id });

    return { token };
  }
}
```

jwt.strategy.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy, ExtractJwt } from 'passport-jwt';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from '../users/user.entity';
import { Repository } from 'typeorm';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    @InjectRepository(User)
    private userRepository: Repository<User>,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: process.env.JWT_SECRET,
    });
  }

  async validate(payload) {
    const { id } = payload;

    const user = await this.userRepository.findOne({
      where: {
        id: id,
      },
    });

    if (!user) {
      throw new UnauthorizedException('Login first to access endpoint');
    }
    return user;
  }
}
```

local.strategy.ts

```
import { Strategy } from 'passport-local';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { AuthService } from './auth.service'; // Припустимо, що ви маєте такий сервіс

@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private authService: AuthService) {
    super({
      usernameField: 'email',
      passwordField: 'password',
    });
```

```
  }

  async validate(email: string, password: string): Promise<any> {
    const user = await this.authService.login({ email, password });
    if (!user) {
      throw new UnauthorizedException('Invalid credentials');
    }
    return user;
  }
}
```

roles.decorator.ts

```
import { SetMetadata } from '@nestjs/common';
import { RolesEnum } from '../users/interfaces/roles.enum';
import { ROLES_KEY } from '../utils/constants';

export const Roles = (...roles: RolesEnum[]) => SetMetadata(ROLES_KEY, roles);
```

roles.guard.ts

```
import { ExecutionContext, Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { RolesEnum } from '../users/interfaces/roles.enum';
import { ROLES_KEY } from '../utils/constants';
import { AuthGuard, IAuthGuard } from '@nestjs/passport';
import { User } from '../users/user.entity';

@Injectable()
export class RolesGuard extends AuthGuard('jwt') implements IAuthGuard {
  constructor(private reflector: Reflector) {
    super();
  }

  public handleRequest(err: unknown, user: User): any {
    return user;
  }

  async canActivate(context: ExecutionContext): Promise<boolean> {
    await super.canActivate(context);
    const request = context.switchToHttp().getRequest();
    const { user } = request;
    request.currentUser = user;

    const requiredRoles = this.reflector.getAllAndOverride<RolesEnum[]>(
      ROLES_KEY,
      [context.getHandler(), context.getClass()],
    );
    if (!requiredRoles) {
      return true;
    }

    if (requiredRoles.includes(RolesEnum.Any)) {
      return !!user;
    }
    return requiredRoles.some((role) => user.roles?.includes(role));
  }
}
```

ДОДАТОК. ФРОНТЕНД

Реалізація реєстрації:

```
import React, {useState} from 'react';
import {Field, FieldArray, Form, Formik} from 'formik';
import * as yup from 'yup';
import {
  Autocomplete,
  Avatar,
  Button,
  Container,
  FormControl,
  Grid,
  InputLabel,
  MenuItem,
  Select,
  TextField,
  Typography
} from '@mui/material';
import {PasswordField} from "../login-forms";
import {DesktopDatePicker} from '@mui/x-date-pickers/DesktopDatePicker';
import {AdapterDateFns} from '@mui/x-date-pickers/AdapterDateFns';
import {LocalizationProvider} from '@mui/x-date-pickers';
import {enUS} from 'date-fns/locale';
import dayjs from "dayjs";

const descriptionSchema = yup.string()
  .max(200, 'Description must be at most 200 characters')

export const RegistrationForm = () => {
  const [role, setRole] = useState('student')
  const initialValues = {
    firstName: '',
    lastName: '',
    profilePhoto: null,
    birthDate: null,
    description: '',
    contact: '',
    email: '',
    password: '',
    role: 'student',
    subjects: []
  };

  const registrationValidationSchema = yup.object({
    role: yup.string().required('Role is required'),
    firstName: yup.string().required('First name is required'),
    lastName: yup.string().required('Last name is required'),
    profilePhoto: role === 'student' ? yup.mixed().nullable() : yup.mixed().required('Profile photo is required for tutors'),
    description: role === 'student' ? descriptionSchema : descriptionSchema.required('Description is required'),
    birthDate: yup.date().max(new Date(), "Birth date must be in the past").required('Birth date is required'),
    contact: yup.string()
      .required('Contact is required')
```

```
            .matches(
              /^(\+\d{1,3}[- ]?)?\d{10}$/,
              'Contact number must be valid'
            ),
        email: yup.string().email('Enter a valid email').required('Email is required'),
        password: yup.string()
          .min(6, 'Password should be minimum 6 characters')
          .matches(/\d/, 'Password should contain at least one number')
          .required('Password is required'),
        subjects: yup.array().of(
          yup.object({
            name: yup.string().required('Subject is required'),
            experienceSince: yup.date().max(new Date(), "Date must be in the past").required('Experience date is required'),
            price: yup.number().min(0, "Price must be non-negative").required('Price per lesson is required')
          })
        ).when('role', (role, schema) => role === 'tutor' ? schema.required() : schema.notRequired())
      });

      const availableSubjects =  [
        "Mathematics", "Physics", "English", "German", "Chemistry", "Biology", "History",
        "Geography", "Computer Science", "Economics", "Business Studies", "Accounting",
      ];

      const isSubjectAdded = (selectedSubjects, currentSubject) => {
        return selectedSubjects.includes(currentSubject);
      };

      const [photoPreview, setPhotoPreview] = useState(null);

      return (
        <LocalizationProvider dateAdapter={AdapterDateFns} locale={enUS}>
          <Container component="main" maxWidth="xs" sx={{
            display: 'flex',
            flexDirection: 'column',
            alignItems: 'center',
            height: '100vh',
            mt: 6
          }}>
            <Formik
              initialValues={initialValues}
              validationSchema={registrationValidationSchema}
              enableReinitialize={true}
              onSubmit={(values, {setSubmitting}) => {
                console.log(values);
                setSubmitting(false);
              }}
              validateOnMount
            >
              {({
                  handleChange,
                  handleBlur,
                  handleSubmit,
                  isSubmitting,
                  touched,
                  errors,
                  values,
```

```
    setFieldValue,
    setFieldTouched,
}) => {
  const handlePhotoChange = (event) => {
    if (event.target.files[0]) {
      setFieldValue("profilePhoto", event.target.files[0]);
      setPhotoPreview(URL.createObjectURL(event.target.files[0]));
    }
  };

  const handleRemovePhoto = () => {
    setFieldValue("profilePhoto", null);
    setPhotoPreview(null);
  };

  return (
    <Form onSubmit={handleSubmit} noValidate>
      <h1>Register</h1>
      <Grid container spacing={2}>
        <Grid item xs={12}>
          {photoPreview ? (
            <div style={{
              display: 'flex',
              flexDirection: 'column',
              alignItems: 'center',
              marginBottom: '10px'
            }}>
              <Avatar src={photoPreview} alt="Profile Photo"
                  sx={{mb: 1, width: 70, height: 70}}/>
              <Button
                variant="contained"
                onClick={handleRemovePhoto}
                sx={{
                  mt: 1,
                  mb: 2,
                  backgroundColor: 'var( --cancel-element-color)',
                  '&:hover': {backgroundColor: 'var(--cancel-secondary-color)'}
                }}
              >
                Delete Photo
              </Button>
            </div>
          ) : (
            <Field>
              {({field}) => (
                <div style={{
                  display: 'flex',
                  flexDirection: 'column',
                  alignItems: 'center',
                  marginBottom: '10px'
                }}>
                  <input
                    accept="image/*"
                    id="profilePhoto"
                    name="profilePhoto"
                    type="file"
                    hidden
```

```
                        onChange={handlePhotoChange}
                      />
                      <label htmlFor="profilePhoto">
                        <Button variant="contained" component="span"
                                sx={{
                                  mb: 2,
                                  backgroundColor: 'var(--secondary-dark-color)',
                                  '&:hover': {backgroundColor: 'var(--primary-element-color)'}
                                }}
                        >
                          Upload Profile Photo
                        </Button>
                      </label>
                      {touched.profilePhoto && errors.profilePhoto && (
                        <Typography variant="caption" display="block" color="error"
                                    sx={{mb: 1.5, mt: -0.5}}>
                          Profile photo is required
                        </Typography>
                      )}
                    </div>
                  )}
                </Field>
              )}
              <Grid item xs={12}>
                <FormControl fullWidth>
                  <InputLabel id="role-select-label">Register as</InputLabel>
                  <Select
                    labelId="role-select-label"
                    id="role"
                    name="role"
                    value={values.role}
                    label="Register as"
                    onChange={e => {
                      handleChange(e);
                      setRole(e.target.value)
                      setFieldValue("subjects", e.target.value === "tutor" ? [{
                        name: '',
                        experienceSince: null,
                        price: ''
                      }] : []);
                    }}
                    onBlur={handleBlur}
                    error={touched.role && Boolean(errors.role)}
                  >
                    <MenuItem value="student">Student</MenuItem>
                    <MenuItem value="tutor">Tutor</MenuItem>
                  </Select>
                  {touched.role && errors.role && (
                    <Typography variant="caption" display="block" color="error"
                                sx={{mt: 1, ml: 1.5}}>
                      {errors.role}
                    </Typography>
                  )}
                </FormControl>
              </Grid>
            </Grid>
            <Grid item xs={12}>
```

```
                    <TextField
                        id="firstName"
                        name="firstName"
                        label="First Name"
                        variant="outlined"
                        fullWidth
                        onChange={handleChange}
                        onBlur={handleBlur}
                        error={touched.firstName && Boolean(errors.firstName)}
                        helperText={touched.firstName && errors.firstName}
                    />
                </Grid>
                <Grid item xs={12}>
                    <TextField
                        id="lastName"
                        name="lastName"
                        label="Last Name"
                        variant="outlined"
                        fullWidth
                        onChange={handleChange}
                        onBlur={handleBlur}
                        error={touched.lastName && Boolean(errors.lastName)}
                        helperText={touched.lastName && errors.lastName}
                    />
                </Grid>
                <Grid item xs={12}>
                    <DesktopDatePicker
                        label="Birth Date"
                        inputFormat="DD/MM/YYYY"
                        value={values.birthDate ? dayjs(values.birthDate, "YYYY-MM-DD").toDate()
: null}
                        onChange={date => {
                            setFieldValue('birthDate', date ? dayjs(date).format("YYYY-MM-DD") :
null);
                            setFieldTouched('birthDate', true, false);
                        }}
                        sx={(theme) => ({
                            width: '100%',
                            ...(touched.birthDate && errors.birthDate && {
                                label: {
                                    color: theme.palette.error.main,
                                },
                                fieldset: {
                                    borderColor: theme.palette.error.main,
                                },
                            }),
                        })}
                        maxDate={dayjs().subtract(4, 'year').toDate()}
                    />
                    {touched.birthDate && errors.birthDate && (
                        <Typography variant="caption" display="block" color="error"
                                    sx={{mt: 1, ml: 1.5}}>
                            {errors.birthDate}
                        </Typography>
                    )}
                </Grid>
                <Grid item xs={12}>
```

```
                    <TextField
                      id="description"
                      name="description"
                      label="Description"
                      variant="outlined"
                      fullWidth
                      multiline
                      rows={4}
                      onChange={handleChange}
                      onBlur={handleBlur}
                      error={touched.description && Boolean(errors.description)}
                      helperText={touched.description && errors.description}
                    />
                  </Grid>
                  <Grid item xs={12}>
                    <TextField
                      id="contact"
                      name="contact"
                      label="Contact"
                      variant="outlined"
                      fullWidth
                      onChange={handleChange}
                      onBlur={handleBlur}
                      error={touched.contact && Boolean(errors.contact)}
                      helperText={touched.contact && errors.contact}
                    />
                  </Grid>
                  <Grid item xs={12}>
                    <TextField
                      id="email"
                      name="email"
                      label="Email"
                      variant="outlined"
                      fullWidth
                      onChange={handleChange}
                      onBlur={handleBlur}
                      error={touched.email && Boolean(errors.email)}
                      helperText={touched.email && errors.email}
                    />
                  </Grid>
                  <Grid item xs={12}>
                    <PasswordField name="password" label="Password"/>
                  </Grid>
                  {values.role === 'tutor' && (
                    <FieldArray name="subjects">
                      {({push, remove}) => (
                        <>
                          {values.subjects.map((subject, index) => (
                            <React.Fragment key={index}>
                              <Grid item xs={12} sm={6}>
                                <FormControl fullWidth>
                                  <Autocomplete
                                    disablePortal
                                    id={`subject-autocomplete-${index}`}
                                    options={availableSubjects.filter(subject =>
!isSubjectAdded(values.subjects.map(sub => sub.subject), subject))}
                                    value={subject.name}
```

```
                                    onChange={(event, newValue) => {
                                        setFieldValue(`subjects[${index}].name`, newValue);
                                    }}
                                    onBlur={() => setFieldTouched(`subjects[${index}].name`,
true)}
                                    renderInput={(params) => (
                                        <TextField
                                            {...params}
                                            label="Select Subject"
                                            error={touched.subjects?.[index]?.name &&
Boolean(errors.subjects?.[index]?.name)}
                                            helperText={touched.subjects?.[index]?.name &&
errors.subjects?.[index]?.name}
                                        />
                                    )}
                                />
                            </FormControl>
                        </Grid>
                        <Grid item xs={12} sm={6}>
                            <DesktopDatePicker
                                label="Experience since"
                                inputFormat="DD/MM/YYYY"
                                value={subject.experienceSince ?
dayjs(subject.experienceSince, "YYYY-MM-DD").toDate() : null}
                                onChange={date => {
                                    setFieldValue(`subjects[${index}].experienceSince`, date ?
dayjs(date).format("YYYY-MM-DD") : null);
                                    setFieldTouched(`subjects[${index}].experienceSince`, true,
false);
                                }}
                                sx={(theme) => ({
                                    width: '100%',
                                    ...(touched.subjects?.[index]?.experienceSince &&
errors.subjects?.[index]?.experienceSince && {
                                        label: {
                                            color: theme.palette.error.main,
                                        },
                                        fieldset: {
                                            borderColor: theme.palette.error.main,
                                        },
                                    }),
                                })}
                                maxDate={new Date()}
                                error={touched.subjects?.[index]?.experienceSince &&
Boolean(errors.subjects?.[index]?.experienceSince)}
                            />
                            {touched.subjects?.[index]?.experienceSince &&
errors.subjects?.[index]?.experienceSince && (
                                <Typography variant="caption" display="block"
                                        color="error" sx={{mt: 1, ml: 1.5}}>
                                    {errors.subjects?.[index]?.experienceSince}
                                </Typography>
                            )}
                        </Grid>
                        <Grid item xs={12}>
                            <TextField
                                name={`subjects[${index}].price`}
```

```
                                    label="Price per lesson"
                                    type="number"
                                    fullWidth
                                    onChange={handleChange}
                                    onBlur={handleBlur}
                                    error={touched.subjects?.[index]?.price &&
Boolean(errors.subjects?.[index]?.price)}
                                    helperText={touched.subjects?.[index]?.price &&
errors.subjects?.[index]?.price}
                                />
                            </Grid>
                            <Grid item xs={12}>
                                <Button
                                    onClick={() => remove(index)}
                                    fullWidth
                                    variant="contained"
                                    disabled={values.subjects.length <= 1}
                                    sx={{
                                        backgroundColor: 'var( --cancel-element-color)',
                                        '&:hover': {backgroundColor: 'var(--cancel-secondary-
color)'}
                                    }}
                                >
                                    Delete Subject
                                </Button>
                            </Grid>
                        </React.Fragment>
                    ))}
                    <Grid item xs={12}>
                        <Button
                            onClick={() => push({
                                name: '',
                                experienceSince: null,
                                price: ''
                            })}
                            fullWidth
                            variant="contained"
                            sx={{
                                backgroundColor: 'var(--reset-element-color)',
                                '&:hover': {backgroundColor: 'var(--primary-element-color)'}
                            }}
                        >
                            Add Subject
                        </Button>
                    </Grid>
                </>
            )}
        </FieldArray>
    )}
    <Grid item xs={12}>
        <Button
            type="submit"
            variant="contained"
            disabled={isSubmitting}
            fullWidth
            sx={{
                mb: 8,
```

```
                                backgroundColor: 'var(--secondary-dark-color)',
                                '&:hover': {backgroundColor: 'var(--primary-element-color)'}
                            }}
                        >
                            Register
                        </Button>
                    </Grid>
                </Grid>
            </Form>
        )
    }
    }
    </Formik>
</Container>
</LocalizationProvider>
);
};
```

Реалізація головної сторінки та фільтрації:

```
import React, {useState} from 'react';
import styles from './styles.module.scss';
import {users} from "../../../mockData";
import {TutorCard} from "../../components/cards/tutor-card";
import {DefaultFilter} from "../../components/filter";
import Pagination from '@mui/material/Pagination';
import Stack from '@mui/material/Stack';

export const HomePage = () => {
  const [criteria, setCriteria] = useState({
    subject: '',
    minPrice: '',
    maxPrice: '',
    timeSlots: []
  });

  const [page, setPage] = useState(1);
  const tutorsPerPage = 6;

  const tutors = users.filter(user => user.role === 'tutor' && applyCriteria(user, criteria));
  const count = Math.ceil(tutors.length / tutorsPerPage);
  const displayedTutors = tutors.slice((page - 1) * tutorsPerPage, page * tutorsPerPage);

  const handleChange = (event, value) => {
    setPage(value);
  };

  return (
    <>
      <div className={styles.catalog}>
        <div className={styles.container}>
          <DefaultFilter setCriteria={setCriteria}/>
        </div>
        <div className={styles.cards}>
          {displayedTutors.map(tutor => (
            <TutorCard key={tutor.id} tutor={tutor}/>
          ))}
```

```
            </div>
          </div>
          <Stack spacing={2} alignItems="center" sx={{marginY: 3}}>
            <Pagination count={count} page={page} onChange={handleChange}/>
          </Stack>
        </>
    );
};


function applyCriteria(tutor, criteria) {
  let subjectMatch = criteria.subject ? tutor.tutorSubject.some(subject => subject.name === criteria.subject) : true;
  let priceMatch = true;
  if (criteria.minPrice || criteria.maxPrice) {
    priceMatch = tutor.tutorSubject.every(subject => {
      return (!criteria.minPrice || subject.pricePerLesson >= parseFloat(criteria.minPrice)) &&
        (!criteria.maxPrice || subject.pricePerLesson <= parseFloat(criteria.maxPrice));
    });
  }

  let timeSlotMatch = true;
  if (criteria.timeSlots && criteria.timeSlots.length) {
    timeSlotMatch = tutor.lessons.some(lesson => {
      if (lesson.accepted) {
        return false;
      }
      const lessonStart = new Date(lesson.date);
      const lessonEnd = new Date(lesson.date);
      lessonEnd.setHours(lessonEnd.getHours() + 1);

      return criteria.timeSlots.some(slot => {
        const [start, end] = slot.split('-').map(time => {
          const [hours, minutes] = time.split(':').map(Number);
          const date = new Date(lessonStart);
          date.setHours(hours, minutes, 0, 0);
          return date;
        });

        return (lessonStart >= start && lessonEnd <= end);
      });
    });
  }

  return subjectMatch && priceMatch && timeSlotMatch;
}
```

Реалізація модального вікна запису на заняття:

```
import React, { useState } from 'react';
import {
  Alert,
  Button,
  Dialog,
  DialogActions,
  DialogContent,
  DialogTitle,
```

```
  IconButton,
  List,
  ListItem,
  ListItemText,
  Snackbar
} from '@mui/material';
import CloseIcon from '@mui/icons-material/Close';
import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';
import { LocalizationProvider, StaticDatePicker } from '@mui/x-date-pickers';
import dayjs from 'dayjs';

const LessonDialog = ({ open, onClose, lessons, tutorSubjects, studentId, tutorId}) => {
  const [selectedDate, setSelectedDate] = useState(null);
  const [selectedTime, setSelectedTime] = useState(null);
  const [selectedSubject, setSelectedSubject] = useState(null);
  const [alertInfo, setAlertInfo] = useState({ show: false, message: '', severity: 'info' });


  const availableLessons = lessons.filter(lesson =>
    dayjs(lesson.date).isAfter(dayjs().startOf('day')) && !lesson.accepted
  );

  const availableTimes = availableLessons.filter(lesson =>
    dayjs(selectedDate).isSame(dayjs(lesson.date), 'day')
  );

  const disablePastAndUnavailableDates = (date) => {
    return dayjs(date).isBefore(dayjs().startOf('day')) || !availableLessons.some(lesson =>
      dayjs(date).isSame(dayjs(lesson.date), 'day')
    );
  };

  const handleTimeSelection = (lesson) => {
    setSelectedTime(lesson.date);
    setSelectedSubject(null);
  };

  const handleConfirm = () => {
    const subject = tutorSubjects.find(sub => sub.name === selectedSubject);
    if (selectedTime && subject) {
      const lessonConfirmation = {
        date: selectedTime,
        studentId: studentId,
        tutorId: tutorId,
        subjectId: subject.id,
        accepted: false
      };
      console.log('Confirmed Lesson:', lessonConfirmation);
      setAlertInfo({
        show: true,
        message: `You have sent the request for lesson on ${dayjs(selectedTime).format('YYYY-MM-
DD HH:mm')}`,
        severity: 'success'
      });
      setTimeout(handleClose, 500);
    } else {
      setAlertInfo({
```

```
            show: true,
            message: 'Please select a valid date, time, and subject.',
            severity: 'error'
        });
    }
};

const handleClose = () => {
    onClose();
    setSelectedDate(null);
    setSelectedTime(null);
    setSelectedSubject(null);
};

return (
    <>
        <Dialog open={open} onClose={handleClose} aria-labelledby="lesson-dialog-title">
            <DialogTitle id="lesson-dialog-title">
                Sign up for a lesson
                <IconButton aria-label="close" onClick={handleClose} sx={{
                    position: 'absolute',
                    right: 8,
                    top: 8,
                    color: (theme) => theme.palette.grey[500]
                }}>
                    <CloseIcon />
                </IconButton>
            </DialogTitle>
            <DialogContent>
                <LocalizationProvider dateAdapter={AdapterDayjs}>
                    {selectedDate && !selectedTime ? (
                        <List component="nav" aria-label="available times">
                            {availableTimes.map((lesson, index) => (
                                <ListItem button key={index} onClick={() => handleTimeSelection(lesson)}>
                                    <ListItemText primary={`${dayjs(lesson.date).format('HH:mm')} -
${dayjs(lesson.date).add(1, 'hour').format('HH:mm')}`} />
                                </ListItem>
                            ))}
                        </List>
                    ) : selectedTime && !selectedSubject ? (
                        <List component="nav" aria-label="select subject">
                            {tutorSubjects.map(subject => (
                                <ListItem button key={subject.id} onClick={() =>
setSelectedSubject(subject.name)}>
                                    <ListItemText primary={subject.name} />
                                </ListItem>
                            ))}
                        </List>
                    ) : (
                        <StaticDatePicker displayStaticWrapperAs="desktop" openTo="day"
value={selectedDate}
                            onChange={setSelectedDate}
shouldDisableDate={disablePastAndUnavailableDates}
                            minDate={dayjs().startOf('year')} maxDate={dayjs().add(1,
'year').endOf('year')} />
                    )}
                </LocalizationProvider>
```

```
            </DialogContent>
            <DialogActions>
              {selectedDate && !selectedTime && (
                <Button onClick={() => setSelectedDate(null)} color="primary">
                  Change Date
                </Button>
              )}
              {selectedTime && (
                <Button onClick={() => setSelectedTime(null)} color="primary">
                  Change Time
                </Button>
              )}
              <Button onClick={handleConfirm} disabled={!selectedTime || !selectedSubject}>
                Confirm the selected time and subject
              </Button>
            </DialogActions>
          </Dialog>
          <Snackbar open={alertInfo.show} autoHideDuration={6000}
                anchorOrigin={{ vertical: 'top', horizontal: 'center' }}
                onClose={() => setAlertInfo({ show: false, message: '', severity: 'info' })}>
            <Alert onClose={() => setAlertInfo({ show: false, message: '', severity: 'info' })}
                 severity={alertInfo.severity} variant="filled" sx={{ width: '100%' }}>
              {alertInfo.message}
            </Alert>
          </Snackbar>
        </>
    );
};

export default LessonDialog;
```

Підключення API авторизації:

client.js

```
import axios from "axios";
import { BASE_URL } from "./urls";

const TOKEN_KEY = "auth-token";

export const client = axios.create({
 baseURL: BASE_URL,
});

client.interceptors.request.use(
 (config) => {
   return {
     ...config,
     headers: {
       ...(config.headers || {}),
      Authorization: localStorage.getItem(TOKEN_KEY)
        ? `Bearer ${localStorage.getItem(TOKEN_KEY)}`
        : undefined,
     },
   };
 },
 (error) => {
```

```
  return Promise.reject(error);
 }
);

export const overfetch = (request) => {
 request({ client })
  .then((response) => {
   return response;
  })
  .catch(({ response }) => {
   return response.data.message || "Unknown error";
  });
};
```

users – index.jsx

```
import { overfetch } from "../client";
import { USERS_URL } from "../urls";

export const getAllUsers = () =>
 overfetch(({ client }) => client.get(USERS_URL));

export const getUsersTutors = () =>
 overfetch(({ client }) => client.get(`${USERS_URL}/tutors`));

export const getUserByID = (id = "") =>
 overfetch(({ client }) => client.get(`${USERS_URL}/${id}`));

export const getCurrentUser = () =>
 overfetch(({ client }) => client.get(`${USERS_URL}/me`));

const BASE_CREATE_USER_PARAMS = {
 firstName: "",
 lastName: "",
 photo: "",
 birthDate: "",
 description: "",
 email: "",
 password: "",
 roles: [],
};
```

auth – index.jsx

```
export const createUser = (params = BASE_CREATE_USER_PARAMS) =>
 overfetch(({ client }) => client.post(USERS_URL, params));

export const deleteUserByID = (id = "") =>
 overfetch(({ client }) => client.delete(`${USERS_URL}/${id}`));

export const pathProfile = (params) =>
 overfetch(({ client }) => client.post(USERS_URL, params));

import { overfetch } from "../client";
import { LOGIN_URL, SIGNUP_URL } from "../urls";

const BASE_LOGIN_PARAMS = {
```

```
 email: "someuser@gmail.com",
 password: "password1",
};

export const login = (params = BASE_LOGIN_PARAMS) =>
 overfetch(({ client }) => client.post(LOGIN_URL, params));

const BASE_SIGNUP_PARAMS = {
 firstName: "",
 lastName: "",
 photo: "",
 birthDate: new Date(),
 description: "",
 email: "",
 password: "",
};

export const signup = (params = BASE_SIGNUP_PARAMS) =>
 overfetch(({ client }) => client.post(SIGNUP_URL, params));
```

image – index.jsx

```
import { overfetch } from "../client";
import { IMAGE_URL } from "../urls";

export const getFile = (name = "") =>
 overfetch(({ client }) => client.get(`${IMAGE_URL}/${name}`));

export const uploadFile = (file) =>
 overfetch(({ client }) => client.post(IMAGE_URL, file));
```