Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ілля СЕРВЕТНИК
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Руслан БЕЛЗЕЦЬКИЙ
(прізвище та ініціали)
«04» червня 2025р.
Рецензент: к.т.н., проф. каф. АІІТ
Віктор МІЗЕРНИЙ
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Андрій ЯРОВИЙ
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"05" Травня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Серветнику Іллі Вячеславовичу

1. Тема роботи: Розробка WEB-ресурсу для онлайн консультацій.
Керівник роботи: доцент Белзецький Руслан Станіславович.
Затверджені наказом вищого навчального закладу "20" 05 2025 року № 97

2. Термін подання студентом роботи 30 Травня 2025 р.

3. Вихідні дані до роботи :
Мова програмування PHP;
Середовище розробки PhpStorm.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки WEB-ресурсу для онлайн консультацій; проектування WEB-ресурсу для онлайн консультацій; розробка WEB-ресурсу для онлайн консультацій; тестування WEB-ресурсу для онлайн консультацій; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема монолітної архітектури, діаграма таблиць зовнішніх календарів, діаграма таблиць дзвінків, форма вибору вільного часу, активний користувач в відео сесії.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	доцент Белзецький Р.С.		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обґрунтування доцільності розробки WEB-ресурсу для онлайн консультацій	05.05.25	07.05.25	
2	Проектування WEB-ресурсу для онлайн консультацій	08.05.25	11.05.25	
3	Розробка WEB-ресурсу для онлайн консультацій	12.05.25	15.05.25	
4	Тестування WEB-ресурсу для онлайн консультацій	16.05.25	26.05.25	
5	Розробка інструкції користувача	27.05.25	29.05.25	
6	Аналіз отриманих результатів та формування висновків	30.05.25	01.06.25	
7	Оформлення бакалаврської кваліфікаційної роботи	02.06.25	04.06.25	

Студент (підпис) **Серветник І.В.** (прізвище та ініціали)

Керівник роботи (підпис) **Белзецький Р.С.** (прізвище та ініціали)

АНОТАЦІЯ

Серветник І.В. Розробка WEB-ресурсу для онлайн консультацій. Бакалаврська кваліфікаційна робота складається з 81 сторінок формату А4, на яких є 54 рисунків, список використаних джерел містить 22 найменувань.

Метою даної бакалаврської кваліфікаційної роботи є дослідження застосування сучасних та класичних підходів для розробки WEB-додатку.

У загальній частині роботи, на основі аналізу сучасних технічних рішень, методів та технологій, обґрунтовано доцільність створення WEB-ресурсу для онлайн-консультацій. Розглядалися переваги й недоліки наявних рішень, а також визначалися ключові вимоги до системи, з урахуванням потреб користувачів і консультантів. У результаті проведеного аналізу було сформовано архітектурну концепцію системи та розроблено зв’язки між основними сутностями серверної частини всіх модулів.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції WEB-ресурсу для онлайн консультацій успішно реалізовано.

Ключові слова: PHP, PhpUnit, Laravel, Livewire, монолітна архітектура, консультація, сесія, відеодзвінок.

ABSTRACT

Servetnyk I.V. Development of a WEB Resource for Online Consultations.

The bachelor's qualification thesis consists of 81 A4 pages, including 54 figures. The list of references contains 22 sources.

The aim of this bachelor's qualification thesis is to explore the application of both modern and classical approaches to the development of a WEB application.

In the general part of the work, based on the analysis of current technical solutions, methods, and technologies, the feasibility of creating a WEB resource for online consultations is substantiated. The advantages and disadvantages of existing solutions were considered, and the key requirements for the system were identified, taking into account the needs of both users and consultants. As a result of the analysis, the architectural concept of the system was formed, and relationships between the main server-side entities of all modules were designed.

A software product was developed and tested. The testing results show that the core functions of the WEB resource for online consultations have been successfully implemented.

Keywords: PHP, PhpUnit, Laravel, Livewire, monolithic architecture, consultation, session, video call.

ЗМІСТ

ВСТУП

У сучасному світі спостерігається стрімке зростання попиту на консультаційні та експертні послуги в різних галузях - від освіти і психології до фінансів, маркетингу та бізнес-аналізу. Все більше фахівців переходять у цифровий простір, де можуть надавати свої послуги без географічних обмежень. Онлайн-формат консультування стає не лише популярним, а й ефективним завдяки швидкості, зручності та доступності. Особливо важливою складовою онлайн-консультацій є відеозв’язок, що дозволяє забезпечити повноцінну комунікацію між консультантом і клієнтом.

На сьогодні більшість фахівців просувають свої послуги через соціальні мережі або месенджери, а процес організації консультацій - включаючи запис клієнтів, оплату, нагадування та відеозустріч - здійснюється вручну або за допомогою різних несумісних інструментів. Такий підхід є неефективним як для спеціалістів, так і для клієнтів. Виникає потреба у створенні єдиної платформи, яка б об’єднала всі необхідні функції: реєстрацію фахівців, формування їхніх профілів, автоматизований запис клієнтів на сеанси, інтегровану систему відеозв’язку, а також безпечну та зручну оплату з можливістю повернення коштів у разі скасування зустрічі.

Аналіз існуючих рішень показує, що розробка сучасного WEB-ресурсу для онлайн-консультацій є актуальним і необхідним кроком, що відповідає потребам ринку та сприяє підвищенню якості дистанційної взаємодії між фахівцями та клієнтами в глобальному масштабі.

Метою дослідження є розширення функціональних можливостей WEB-ресурсу для проведення онлайн-консультацій, шляхом реалізації реєстрації спеціалістів, автоматизації запису клієнтів, інтеграції відеозв’язку та забезпечення зручної платіжної системи.

Об’єктом дослідження є процес створення WEB-ресурсу для організації онлайн-консультацій.

Предметом дослідження є архітектура веб-платформи, функціональність інтерфейсу користувача та логіка взаємодії з відеосервісом і платіжною системою.

Задачі дослідження:

1. Обґрунтувати доцільність створення єдиної онлайн-платформи для проведення консультацій;
2. провести аналіз сучасних технологій для реалізації додатку з можливістю інтеграції інших сервісів;
3. спроектувати та реалізувати функціональність платформи для консультантів та клієнтів;
4. провести тестування веб-платформи та оцінити її зручність і стабільність роботи.

Апробація роботи.

Результати досліджень було апробовано на LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (ВНТКП ВНТУ) у 2025 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1]; подано заявку на реєстрацію авторського права на твір у формі комп’ютерної програми - «WEB-ресурс для онлайн консультацій», номер заявки С202504912 від 21.05.2025 р.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ

1.1 Аналіз предметної області

Сучасний веб-ресурс для онлайн консультацій - це не просто засіб комунікації, а повноцінна система взаємодії між спеціалістами-консультантами та їхньою цільовою аудиторією. У світі, де цифрова взаємодія все більше замінює традиційні формати, такі платформи мають забезпечувати максимально зручний, ефективний і безпечний процес співпраці між сторонами.

Консультанти очікують від сервісу гнучкості, автоматизації рутинних процесів і можливості масштабувати свою діяльність. З іншого боку, клієнти прагнуть простоти запису, зрозумілої структури сервісу, надійних механізмів оплати й гарантій результату. Таким чином, платформа має однаково добре закривати потреби обох категорій користувачів - як з боку надання послуг, так і з боку їхнього споживання.

Основна мета такої системи - зробити процес онлайн консультації настільки ж легким і прозорим, як замовлення таксі чи їжі через мобільний застосунок.

Платформа для онлайн консультацій має мати наступні ключові функціональні можливості:

1. Індивідуальні сесії. Консультант має змогу налаштувати власний робочий графік та приймати запити на приватні онлайн-зустрічі. Це дозволяє проводити індивідуальні консультації в зручний для себе час, з можливістю гнучкого керування навантаженням;
2. Групові зустрічі. Платформа має підтримувати створення подій з кількома учасниками - для майстер-класів, семінарів або колективних консультацій. Клієнти можуть зареєструватися на таку подію заздалегідь і приєднатися в зазначений час;
3. Інтеграція з календарями. Важливою функцією є автоматична синхронізація з двома найпоширенішими календарними сервісами -

Google Calendar та Outlook Calendar, що допомагає уникнути конфліктів у розкладі та мінімізує ризик забути про зустріч;

4. Надійна платіжна система. Платформа повинна забезпечувати швидке та безпечне проведення фінансових операцій через популярні сервіси. Крім цього, варто передбачити механізми автоматичного повернення коштів у випадках скасування або переносу зустрічі;
5. Міжнародна доступність. Оскільки цільова аудиторія може бути географічно розподіленою, інтерфейс має бути реалізований англійською мовою, а сама платформа - підтримувати різні часові пояси, валюти та формати оплати;
6. Особистий кабінет консультанта. Це центр керування, де фахівець може налаштовувати свій профіль, керувати клієнтами та розкладом. Тут він редагує опис послуг і цін, додає фото та сертифікати, переглядає історію сесій і статуси оплат, а також виставляє зручний графік роботи.

Також потрібно сконцентруватися на технічних аспектах розробки системи:

1. Безпека та конфіденційність. Усі дані, що передаються через платформу - як особисті, так і фінансові - мають бути надійно захищені за допомогою сучасних протоколів шифрування. Платформа повинна відповідати вимогам міжнародних стандартів безпеки.
2. Масштабованість. Система повинна бути здатною обслуговувати велику кількість одночасних користувачів, не втрачаючи продуктивності. Це стосується як стабільності відеозв’язку, так і швидкодії бекенду при високому навантаженні.
3. Поштова розсилка. Система повинна автоматично надсилати повідомлення електронною поштою з нагадуванням про майбутні консультації, підтвердженням запису, деталями платежів або іншими важливими оновленнями як для клієнтів, так і для консультантів.

4. Система онлайн-зв’язку. Для проведення відеозустрічей необхідний стабільний, масштабований інструмент, що підтримує відео, аудіо, чат та передачу даних між кількома користувачами. Цей механізм має бути легко інтегрованим у веб-інтерфейс і мати зрозумілу документацію для розробників. Ідеальним рішенням для таких цілей є Zoom Video SDK [2], який забезпечує високу якість зв’язку, підтримку великої кількості учасників і широкі можливості кастомізації під потреби конкретного проєкту.

1.2 Огляд аналогів

При розробці WEB-ресурсу для онлайн консультацій важливим етапом є аналіз існуючих аналогів. Це дозволяє не лише ознайомитися з уже реалізованими рішеннями, а й оцінити їхню ефективність, функціональність та зручність використання. Аналіз аналогів дає можливість виявити сильні сторони конкурентів, запозичити найкращі практики та уникнути типових помилок, які можуть призвести до негативного досвіду користувача або обмеженої функціональності платформи. Крім того, вивчення існуючих рішень сприяє знаходженню нових ідей, технологій та підходів, які вже показали свою успішність у реальних умовах.

На сучасному ринку представлено велику кількість онлайн-сервісів, які надають консультаційні послуги у різних форматах — від відеодзвінків до обміну повідомленнями або попередньо записаними відповідями. Майже всі ці платформи містять сторінки консультантів із детальною інформацією про доступні послуги.

Нижче наведено приклади найпопулярніших та найцікавіших рішень, які варто взяти до уваги при створенні власного WEB-ресурсу. Розглянемо 3 найпопулярніших аналоги:

1. Clarity (рис. 1.1).

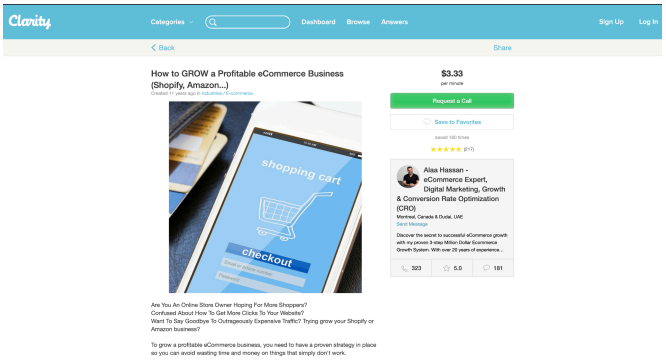

Рисунок 1.1 – Сторінка з описом дзвінка в додатку Clarity

Clarity є одним із найвідоміших сервісів онлайн консультування, який орієнтований переважно на бізнес-аудиторію. Його головна особливість - це можливість знайти експерта з конкретної теми, домовитись про телефонну розмову та отримати індивідуальну консультацію. Сервіс пропонує два способи взаємодії: клієнт може або залишити запит, на який відгукнуться фахівці, або самостійно обрати консультанта зі списку. Такий підхід дає змогу швидко знайти компетентного спеціаліста.

Проте, попри загальну зручність, платформа має ряд недоліків. Зокрема, консультанти не можуть заздалегідь налаштувати свій графік доступності, що ускладнює планування консультацій. Крім того, сервіс не підтримує групові дзвінки, що обмежує його використання в командній роботі або для навчання кількох осіб одночасно. Ще одним недоліком є модель оплати за хвилини, яка не завжди є зручною та прозорою для клієнтів, особливо в разі довших сесій або непередбачуваних затримок.

2. Wisio (рис. 1.2).

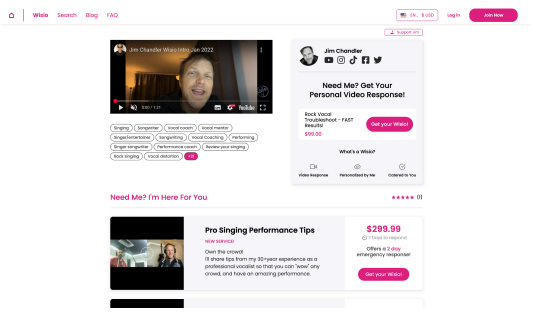

Рисунок 1.2 – Сторінка з описом дзвінка в додатку Wisio

Wisio позиціонує себе як інноваційна платформа для онлайн консультацій з акцентом на зручність і простоту використання. Її головною особливістю є асинхронний формат взаємодії: клієнти надсилають свої запитання консультантам, які у відповідь записують відео або аудіо та надсилають його назад. Це дозволяє фахівцям відповідати у зручний для них час, а клієнтам - переглядати відповідь стільки разів, скільки потрібно.

Однак такий підхід має свої обмеження. Відсутність можливості комунікації в реальному часі позбавляє користувачів живого діалогу, що є особливо важливим у ситуаціях, коли питання виникають у процесі обговорення. Це може знижувати ефективність консультації, особливо в сферах, де потрібне оперативне з'ясування деталей або уточнення нюансів.

3. Intro (рис. 1.3).

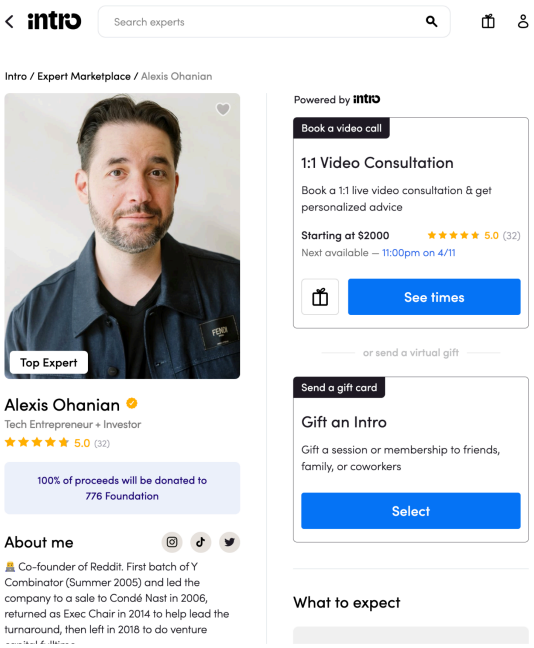

Рисунок 1.3 – Сторінка з описом дзвінка в додатку Intro

Intro - ще один приклад сервісу, який реалізує модель онлайн консультацій через бронювання часу для відеозв'язку з експертами. Платформа вирізняється сучасним дизайном, зручною навігацією та широким вибором фахівців у різних сферах. Користувачі можуть обрати консультанта, переглянути його профіль, відгуки, вільний час у календарі та забронювати дзвінок.

Попри позитивні сторони, платформа також має низку недоліків. Зокрема, відсутність інтеграції з популярними онлайн-календарями (такими як Google Calendar або Outlook Calendar) ускладнює синхронізацію подій. Крім того, так само як і Clarity, Intro не підтримує проведення групових сесій, що обмежує її функціональність у контексті командних консультацій чи навчальних заходів.

1.3 Постановка задачі та формулювання вимог

Для успішної реалізації необхідно чітко окреслити технічні та функціональні характеристики системи: від продуктивності та стабільності роботи до інтеграції сторонніх сервісів, зручності інтерфейсу та відповідності сучасним вимогам до захисту персональної інформації.

Функціональні вимоги для консультанта:

1. **Реєстрація та аутентифікація.** Консультант створює обліковий запис, підтверджує пошту, заповнює інформацію про себе та додає платіжні реквізити. У випадку втрати пароля може відновити доступ через лист на пошту. Зареєстровані користувачі входять за допомогою email і пароля.
2. **Головна сторінка зі статистикою.** Містить стислу інформацію про доходи, кількість сесій, а також список майбутніх дзвінків для швидкого доступу до них.
3. **Список дзвінків.** Консультант переглядає майбутні, завершені, скасовані та активні дзвінки. **Менеджмент пропозицій.** Консультант створює, редагує та видаляє індивідуальні або групові плани консультацій, визначаючи тематику, ціну та тривалість.
4. **Редагування профілю.** Дозволяє оновлювати ім’я, фото, опис послуг, досвід, сертифікати, мови тощо, аби підтримувати актуальність інформації для потенційних клієнтів.
5. **Під’єднання календарів.** Платформа дозволяє підключати Google Calendar та Outlook для автоматичної синхронізації графіка.
6. **Менеджмент оплати.** Доступ до розділу з інформацією про зароблені кошти, історію виплат та можливість виведення грошей.

Функціональні вимоги для клієнта:

1. **Купівля індивідуального дзвінка.** Клієнт обирає зручний час, залишає інформацію, та здійснює оплату через захищену платіжну систему.
2. **Купівля місця на груповий дзвінок.** Можна переглянути опис події, обрати кількість місць та зареєструватись на групову сесію.
3. **Проведення дзвінка.** Інтерфейс має бути інтуїтивним, з чітким аудіо та відео, підтримкою демонстрації екрана, вибору камери й мікрофона.

Вимоги до WEB-ресурсу:

1. **Безпека даних.** Персональні та фінансові дані користувачів повинні бути надійно захищені.
2. **Швидкість та надійність.** Система має працювати стабільно, без затримок чи збоїв, навіть при великій кількості користувачів.
3. **Зручний інтерфейс.** Інтерфейс має бути простим у використанні як для консультантів, так і для клієнтів.
4. **Міжнародна доступність.** Сервіс має бути англомовним, відкритим для користувачів з будь-якої країни.

1.4 Висновок

У першому розділі було проведено аналіз аналогів, що дозволило виявити ключові проблеми існуючих платформ для онлайн консультацій. Це допомогло визначити шляхи їх вирішення для створення більш ефективного продукту. На основі отриманих даних була сформульована чітка постановка задачі, що включає вимоги, які забезпечать максимальну ефективність як для консультантів, так і для клієнтів.

Створення такого сервісу передбачає оптимізацію робочих процесів консультантів, зручний інтерфейс для користувачів та інтеграцію з необхідними технологіями, що дозволить зробити платформу зручною, надійною та функціональною.

2 ПРОЕКТУВАННЯ WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ

2.1 Аналіз та вибір технологій розробки

При розробці веб-додатку, одним із найважливіших етапів є правильний вибір технологій. Від того, які саме інструменти та підходи будуть використані, залежить стабільність роботи системи, її продуктивність, масштабованість, легкість у підтримці, а також зручність розробки та подальшого оновлення функціоналу. Успішний вибір технологічного стеку дозволяє не лише зекономити час і ресурси, але й уникнути численних проблем, які можуть виникнути під час масштабування проєкту або його інтеграції з іншими сервісами.

Варто пам’ятати, що використання трендових, але ще нестабільних технологій може спричинити додаткові складнощі у реалізації. Часто такі інструменти не мають достатньої кількості документації або прикладів використання, що уповільнює процес розробки. Окрім того, нестабільні технології не завжди проходять достатню кількість перевірок у бойових умовах, тому приховані технічні обмеження можуть проявитися вже після запуску. Замість цього варто орієнтуватися на перевірені рішення, які вже давно присутні на ринку, активно підтримуються розробниками, мають широку спільноту користувачів і велику базу знань. Такі технології значно знижують ризик помилок, спрощують процес налагодження і відкривають більше можливостей для пошуку допомоги, якщо виникають технічні труднощі.

Розпочнемо аналіз і вибір технологій саме з бекенд-частини, оскільки саме на серверному боці буде зосереджена більшість бізнес-логіки нашого додатку. Правильно підібрана мова програмування та інструменти для бекенду дозволяють не лише спростити розробку, а й значно полегшити подальшу підтримку та розвиток проєкту. Для веб-додатку, яким є наш проєкт, одним із найлогічніших варіантів видається мова PHP. Ця мова створювалася з самого початку спеціально для роботи з вебом, і за багато років довела свою ефективність та надійність у тисячах реальних проєктів.

PHP [3] має велику перевагу у вигляді широкого та активного ком'юніті, що дозволяє швидко знаходити відповіді на питання, вирішувати проблеми та отримувати підтримку. Крім того, PHP постачається з багатим набором вбудованих бібліотек, орієнтованих саме на створення веб-сервісів. Величезним плюсом є також наявність сучасних і гнучких фреймворків, які спрощують реалізацію бізнес-логіки, інтеграцію з базами даних, зовнішніми API та забезпечують високий рівень безпеки.

Серед найпопулярніших PHP-фреймворків варто розглянути два головних претенденти - Laravel та Symfony. Laravel [4] сьогодні вважається одним із найзручніших та найшвидших у розробці рішень завдяки простій та зрозумілій структурі, сучасному підходу до розробки та наявності власної ORM - Eloquent [5]. Цей фреймворк має велику кількість готових пакетів, які легко інтегруються, а також дуже добре написану документацію, що робить його чудовим вибором для невеликих і середніх проєктів. Symfony [6], своєю чергою, є більш складним, але водночас гнучкішим і потужнішим інструментом. Його ORM - Doctrine [7] - ідеально підходить для створення складних, масштабованих систем із чіткою структурою коду та високою якістю реалізації. Проте, у нашому випадку, з огляду на обсяги проєкту та потребу у швидкому старті, доцільніше обрати Laravel як основний фреймворк, який забезпечить баланс між швидкістю розробки та можливістю подальшого розширення.

Наступним кроком є вибір бази даних. Оскільки більшість інформації у додатку має чітку структуру і логічні зв'язки, найкращим рішенням буде використання реляційної бази даних. Найпопулярнішим варіантом є MySQL [8] - перевірене роками рішення, яке відзначається високою швидкістю роботи, надійністю та підтримкою з боку великої кількості інструментів та сервісів. До того ж, MySQL чудово інтегрується з Laravel, що також є важливою перевагою.

Втім, повністю уникнути використання NoSQL [9] рішень у сучасному веб-додатку практично неможливо. Для таких задач, як кешування даних, управління чергами задач (наприклад, відправка листів або сповіщень), оптимальним варіантом є Redis. Це високошвидкісна NoSQL база типу

“ключ-значення”, яка прекрасно справляється з обробкою великої кількості дрібних операцій у реальному часі. Redis забезпечує мінімальні затримки та високу стабільність, що критично важливо для плавної роботи користувацького інтерфейсу.

Переходячи до фронтенду, варто зазначити, що сучасні користувачі очікують від сайту не лише функціональність, а й швидкість, плавність та приємний досвід взаємодії. Саме тому більшість веб-застосунків сьогодні будуються як SPA (single-page application) [10], які забезпечують динамічне оновлення контенту без повного перезавантаження сторінки. Для створення таких інтерфейсів найчастіше використовують популярні JavaScript-фреймворки - React, Vue або Angular. Вони забезпечують високий рівень інтерективності та гнучкості, але разом із тим суттєво збільшують складність підтримки проєкту. Особливо це помітно, коли команда невелика, а фронтенд і бекенд розділені технологічно.

Проте Laravel має чудове рішення для таких випадків - фреймворк Livewire [11]. Це інструмент, який дозволяє створювати реактивні інтерфейси, використовуючи PHP, без необхідності писати великий обсяг JavaScript-коду. Livewire “під капотом” використовує JavaScript для обміну даними з сервером, але розробнику достатньо працювати лише з PHP. Це значно спрощує розробку, особливо коли бекенд і фронтенд тісно пов’язані між собою, як у нашому випадку. За рахунок Livewire можна побудувати сучасний, швидкий та зручний інтерфейс, залишаючись при цьому в межах єдиного технологічного стеку.

Окрему увагу слід приділити модулю відеодзвінків, де важлива стабільність і швидкість реакції інтерфейсу. У випадках, коли кількість запитів до сервера зростає, а навантаження стає значним, можливі затримки або підвисання. Щоб цього уникнути, доцільно використовувати легковісний JavaScript-фреймворк Alpine.js [12], який ідеально доповнює Livewire. Його інтеграція проста та ефективна, а продуктивність дозволяє обробляти навіть критичні для інтерфейсу сценарії без втрат якості.

2.2 Вибір архітектури додатку

Не менш важливим аспектом є вибір архітектури, на якій буде побудована система. Існує три найпоширеніші варіанти архітектурного підходу: монолітна, мікросервісна та сервісноорієнтована.

Монолітна архітектура [13] (рисунок 2.1) передбачає, що вся логіка додатку розміщена в одному єдиному кодовому блоці. Це найпростіший та найбільш зрозумілий підхід, який ідеально підходить для проєктів із невеликою або помірною складністю. Моноліт легко розгорнути, його простіше тестувати та підтримувати на ранніх етапах, і він не вимагає складної інфраструктури. Основним недоліком такого підходу є те, що зі зростанням додатку будь-які зміни можуть ускладнюватися, а масштабування окремих функцій стає неможливим без зміни всього додатку.

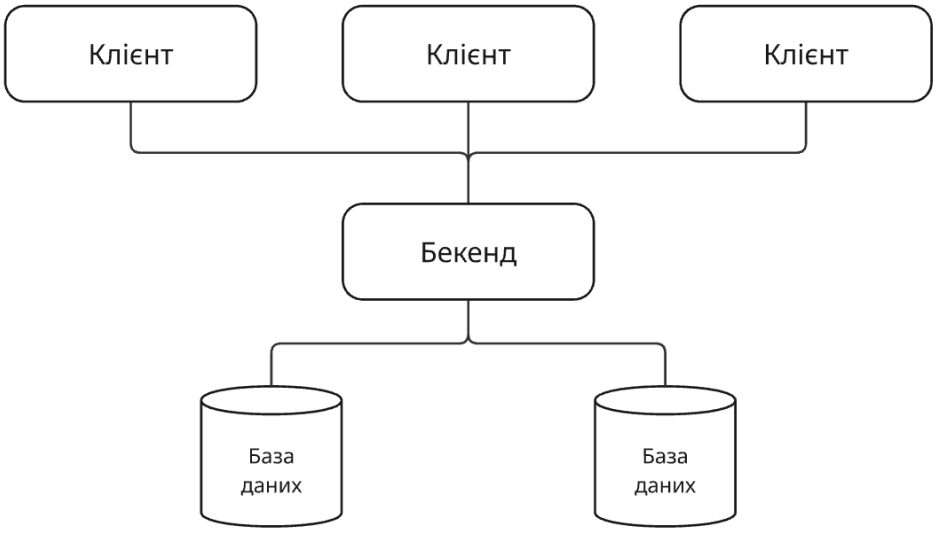

Рисунок 2.1 – Схема монолітної архітектури

Мікросервісна архітектура [14] (рисунок 2.2), навпаки, розділяє функціонал на незалежні сервіси, кожен з яких виконує окрему задачу. Це дозволяє масштабувати кожну частину системи окремо, використовувати різні технології для кожного сервісу та розподіляти навантаження між окремими командами. Такий підхід забезпечує високу гнучкість і підвищує відмовостійкість. Але при цьому він значно ускладнює початкову розробку,

вимагає налаштування взаємодії між сервісами, автоматизації розгортання, централізованого логування та моніторингу, а також потребує вищого рівня технічної компетенції від команди.

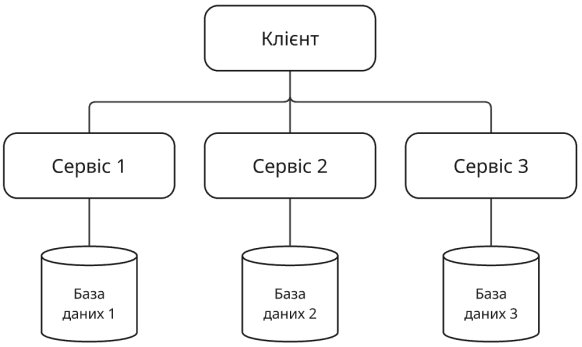

Рисунок 2.2 – Схема мікросервісної архітектури

Сервісноорієнтована архітектура [15] (рисунок 2.3) багато в чому подібна до мікросервісної, але замість повністю ізольованих компонентів вона орієнтується на повторне використання сервісів через спільну шину даних (ESB) або інші інструменти інтеграції. SOA була популярною у корпоративних рішеннях, де системи повинні обмінюватися даними через стандартизовані API або протоколи. Основна відмінність полягає в тому, що SOA менш гнучка, ніж мікросервіси, але краще підходить для інтеграції з великими існуючими системами. Водночас вона може створювати “вузькі місця” у системі через спільні точки обміну.

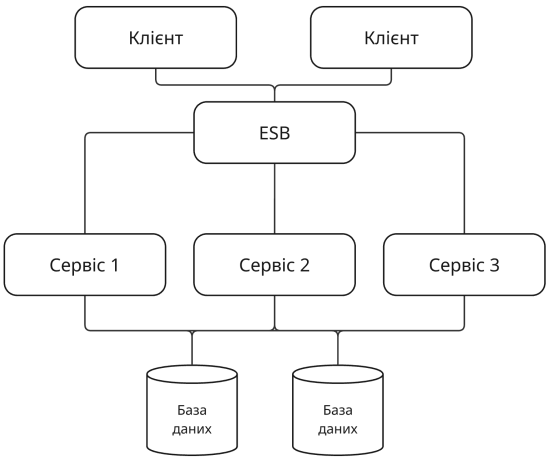

Рисунок 2.3 – Схема сервісноорієнтовної архітектури

У нашому випадку, враховуючи обсяг проєкту, логіку додатку, а також необхідність швидкої реалізації, найбільш доцільним вибором є монолітна архітектура. Вона дозволить швидко запустити основний функціонал, зменшити витрати на інфраструктуру та спростить підтримку проєкту на перших етапах. За потреби, у майбутньому можна буде поступово рефакторити окремі частини коду та виділяти їх в окремі сервіси, якщо зросте навантаження або функціональна складність системи.

2.3 Проектування модулів

У процесі побудови будь-якого програмного забезпечення особливу роль відіграє модульність. Практично кожен сучасний проєкт ділиться на окремі модулі - логічно виділені частини коду, що групують споріднені функції в межах однієї зони відповідальності. Такий підхід дозволяє значно спростити як розробку, так і подальшу підтримку системи. Завдяки чіткому розмежуванню обов’язків модулі допомагають уникнути хаосу в кодовій базі, зменшити кількість помилок та зробити систему більш зрозумілою для розробників, які будуть з нею працювати в майбутньому.

Одним із ключових підходів, який сприяє грамотному проєктуванню модульної архітектури, є Domain-Driven Design (DDD) [16] - методологія, зосереджена на глибокому розумінні предметної області. DDD передбачає, що структура коду повинна відображати структуру бізнесу. Замість того, щоб будувати систему навколо технічних деталей, ми будуємо її навколо доменної моделі - ключових понять, які реально існують у нашій бізнес-логіці. У DDD ми розділяємо систему на bounded contexts - відокремлені доменні області, кожна з яких охоплює певний аспект роботи системи. Саме ці bounded contexts і є основою для побудови модулів. Наприклад, у нашому випадку модулі “Оплата”, “Дзвінки”, “Сесія” - це окремі bounded contexts, які мають чіткі межі, власну бізнес-логіку, термінологію та навіть правила.

Окрім DDD, при проєктуванні модулів слід керуватися й принципами об’єктно-орієнтованого програмування. Зокрема, важливу роль тут відіграють GRASP-патерни (General Responsibility Assignment Software Patterns), які допомагають визначити, як найкраще розподілити відповідальність між класами та модулями. Два ключові патерни - High Cohesion та Low Coupling [17] - є фундаментальними для побудови якісної модульної архітектури.

High Cohesion означає, що модуль повинен бути сфокусований на виконанні однієї чітко визначеної задачі або групи пов’язаних задач. Іншими словами, всі його компоненти повинні працювати над спільною метою. Це сприяє кращій зрозумілості, простоті змін та повторному використанню коду. З іншого боку, Low Coupling передбачає, що залежність між модулями має бути мінімальною. Один модуль повинен мати змогу еволюціонувати без необхідності вносити зміни в інші модулі. Такий підхід зменшує ризик помилок, покращує масштабованість та дозволяє працювати над різними частинами системи паралельно.

Виходячи з цього, нижче наведено розподіл функціоналу нашого застосунку на логічні модулі.

Модуль, що відповідає за роботу з консультантами, охоплює всі процеси, пов’язані з їхнім життєвим циклом у системі. У ньому реалізується логіка реєстрації нового користувача з відповідною роллю, збереження та редагування персональних даних, а також управління файлами. Значну увагу в цьому модулі приділено безпеці обробки персональної інформації та перевірці даних.

Оплата - це окремий модуль, який забезпечує повну інтеграцію з платіжним шлюзом та підтримує весь платіжний цикл. Він відповідає за створення платіжних сесій, ініціацію та перевірку транзакцій, обробку результатів оплат через webhook-и, формування історії платежів і, що не менш важливо, за розрахунок винагороди консультантам. У разі виникнення проблем - таких як відхилення транзакції, недостатньо коштів, або збої при підтвердженні платежу - модуль обробляє помилки та генерує відповідні повідомлення.

Модуль дзвінків відповідає за організацію та управління індивідуальними або груповими зустрічами між користувачами. Він зберігає деталі дзвінків: дату, час, тривалість, учасників, статус, а також дозволяє бронювати вільні часові слоти. Додатково реалізовано можливості перенесення або скасування зустрічей з автоматичним сповіщенням обох сторін. Особливу увагу приділено перевірці конфліктів у розкладі, що унеможливлює накладання дзвінків. Цей модуль тісно взаємодіє з календарним інтерфейсом, що дозволяє зручно переглядати майбутні та минулі дзвінки в одному місці, зберігаючи цілісність розкладу.

Із модулем дзвінків тісно пов’язаний модуль інтеграції із зовнішніми календарями. Його завдання - надати користувачеві можливість синхронізувати події з такими сервісами, як Google Calendar або Outlook. Цей модуль реалізує підключення через протокол OAuth2 [18], автоматичне зчитування та оновлення подій, відображення зовнішніх івентів у внутрішньому розкладі системи, а також збереження синхронізованих даних у базі. Таким чином, модуль допомагає уникнути дублювання або накладень у розкладі, що особливо актуально для консультантів із щільним графіком.

Нарешті, модуль відео сесій є одним із ключових у проєкті, оскільки саме через нього відбувається реальна взаємодія між консультантом і клієнтом. У цьому модулі реалізовано інтеграцію з відео-платформою, створення віртуальних кімнат, авторизацію та доступ користувачів до дзвінка за ролями. Під час сесії здійснюється логування основних подій - таких як вхід, вихід, загальна тривалість, активність користувачів - для подальшого аналізу або вирішення суперечок. Також реалізовано захист від несанкціонованого доступу та обробку типових проблем: наприклад, повторне підключення у разі втрати зв’язку, перенаправлення користувача у разі закриття сесії або недопущення приєднання до вже завершеного дзвінка. Цей модуль формує основу взаємодії в режимі реального часу та забезпечує користувачеві максимально стабільний та зрозумілий досвід.

2.4 Висновок

У другому розділі було проаналізовано та обрано технології розробки. Для бекенд-частини було вибрано мову PHP разом із фреймворком Laravel, оскільки це надійне та швидке рішення для створення веб-додатків. Для фронтенд-частини обрано Livewire завдяки його тісній інтеграції з Laravel, що дозволяє зменшити складність коду та пришвидшити процес розробки. Також використовується легковісний JavaScript-фреймворк Alpine.js - переважно для реалізації функціоналу відеодзвінків. У якості системи керування базами даних було обрано MySQL - за її високу продуктивність, стабільність та широке поширення.

Також було визначено архітектуру додатку. Для реалізації цього проєкту обрано монолітну архітектуру, оскільки вона найкраще підходить для розробки швидкого та функціонально не надто складного рішення.

Окрім цього, було проаналізовано майбутню модульну структуру системи, визначено основні модулі, їх функціональність та ключові властивості. Зокрема, модулі повинні бути максимально незалежними, зосередженими на виконанні своїх завдань і мати мінімальний рівень взаємодії між собою. У наступних розділах буде представлено реалізацію та тестування цих модулів.

3 РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ

3.1 Реалізація модуля консультанта

Реалізацію модулів консультанта розпочнемо з базового та найважливішого кроку - створення моделі. У фреймворку Laravel модель є центральним елементом, що представляє певну сутність в додатку та взаємодіє з відповідною таблицею в базі даних. У нашому випадку - це модель консультанта, яка буде ключовою в побудові всієї системи. Саме ця модель стане основою для реалізації механізмів аутентифікації, авторизації, а також буде пов'язана з іншими сутностями додатку.

Laravel надає потужний інструмент для генерації базових компонентів - Artisan. Це вбудований командний інтерфейс, який суттєво спрощує розробку, дозволяючи автоматично створювати моделі, контролери, запити, міграції, компоненти та інші частини проєкту. Завдяки Artisan ми можемо уникнути рутинної роботи, зосередившись на логіці додатку.

Створення моделі відбувається за допомогою команди: *php artisan make:model User -m*. Ключ *-m* одночасно створює і міграцію - спеціальний файл, у якому описано структуру таблиці. Міграції в Laravel, як і в багатьох сучасних фреймворках, дозволяють синхронізувати базу даних між різними розробниками. Це надзвичайно зручно, оскільки виключає потребу передавати SQL-дампи чи вручну створювати таблиці - достатньо виконати команду, і база автоматично оновиться відповідно до описаних схем.

На рисунку 3.1 зображено модель User, яка вже має заповнені атрибути та наслідує ключові абстракції, зокрема класи та інтерфейси, необхідні для подальшої реалізації функціональності аутентифікації. Це забезпечує правильну інтеграцію з механізмами Laravel аутентифікації та іншими службами фреймворку.

```
class User extends Authenticatable implements MustVerifyEmail
{
    use HasApiTokens;
    use HasFactory;
    use Notifiable;
    use SoftDeletes;

    protected $fillable = [
        'uuid',
        'name',
        'slug',
        'email',
        'email_verified_at',
        'password',
        'onboarding_completed',
        'timezone_id',
        'time_format',
        'stripe_account_id',
        'validated_by_stripe',
        'currency_id',
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

    protected $casts = [
        'onboarding_completed' => 'boolean',
        'email_verified_at' => 'datetime',
        'validated_by_stripe' => 'boolean',
        'time_format' => TimeFormat::class,
    ];
```

Рисунок 3.1 - Модель User

Клас User наслідує абстрактний клас Authenticatable, який є частиною вбудованої системи аутентифікації Laravel. Цей клас забезпечує необхідну інтеграцію з механізмами авторизації та захисту маршрутів, дозволяючи фреймворку розпізнавати дану модель як таку, що підлягає аутентифікації. Окрім цього, клас User імплементує інтерфейс MustVerifyEmail. Це означає, що перед отриманням повного доступу до системи консультанти повинні підтвердити свій email. Такий підхід є важливим з точки зору безпеки та довіри.

Також модель User використовує трейт SoftDeletes. Його основна задача - реалізація механізму м’якого видалення. Замість фізичного видалення запису з таблиці, в полі deleted_at зберігається дата, коли запис був позначений як

видалений. Це дозволяє в подальшому відновити користувача при потребі, а також зберігати історичні дані для внутрішнього аналізу або логування.

Переходячи до змінної $fillable, ми бачимо масив, у якому перелічені поля, що дозволені для масового заповнення. Під капотом Laravel використовує власну ORM під назвою Eloquent, яка реалізує патерн ActiveRecord. Цей патерн передбачає, що кожен об'єкт моделі не лише представляє окремий запис у базі даних, а й безпосередньо містить методи для роботи з цими даними - збереження, оновлення, видалення або отримання пов’язаних записів. Таким чином, одна й та сама сутність у коді поєднує в собі як самі дані, так і логіку для їх обробки.

Завдяки цьому підходу, розробнику не потрібно явно оголошувати змінні у класі або писати додаткові SQL-запити. Достатньо визначити перелік полів у масиві $fillable, після чого Eloquent автоматично забезпечує правильну роботу з відповідною таблицею в базі даних. Це значно пришвидшує розробку, зменшуючи обсяг рутинного коду та дозволяючи зосередитися на бізнес-логіці проєкту.

Масив $fillable містить перелік полів, які дозволено масово заповнювати при створенні або оновленні моделі. Кожне з них відповідає за певну характеристику користувача. Поле uuid використовується як унікальний ідентифікатор користувача, відокремлений від звичайного числового id, що зручно для публічної ідентифікації. Ім’я користувача зберігається у полі name, а slug є URL-дружньою версією імені, яка може використовуватись, наприклад, у маршрутах або профілях. Email і password - стандартні поля для автентифікації, при цьому email_verified_at фіксує дату підтвердження електронної пошти.

Поле onboarding_completed вказує, чи користувач пройшов первинне налаштування профілю, timezone_id зберігає часову зону, а time_format відповідає за формат відображення часу (наприклад, 12 чи 24-годинний). Stripe_account_id використовується для прив'язки до платіжної системи Stripe [19], а validated_by_stripe показує, чи обліковий запис успішно верифіковано на стороні Stripe. Нарешті, currency_id вказує, у якій валюті консультант приймає

оплату за свої послуги. У сукупності ці поля дозволяють гнучко налаштовувати профіль консультанта та інтегрувати його в систему онлайн консультацій.

Перейдемо до реалізації аутентифікації. Laravel підтримує багато бібліотек для роботи з аутентифікацією, проте у нашому випадку найбільш відповідною є бібліотека Sanctum [20]. Вона генерує access-токени, зберігає їх у базі даних і автоматично додає у cookie браузера. Такий підхід ідеально відповідає нашим потребам, оскільки дозволяє забезпечити безпечну та зручну взаємодію між клієнтом і сервером без зайвих складнощів з авторизацією.

Розпочнемо з аутентифікації на стороні реєстрації користувача. Сторінка реєстрації реалізована як Livewire-компонент, що містить поля для введення імені, електронної пошти та пароля. У межах цього компонента реалізовано виведення верстки, валідацію вхідних даних і логіку створення та збереження нового користувача в системі. На рисунку 3.2 зображено метод реєстрації консультанта.

```
public function register()
{
    $this->validate();

    $user = app(UserService::class)->create(
        $this->name,
        $this->email,
        $this->password
    );

    Auth::login($user);

    return redirect()->route('onboarding.intro');
}
```

Рисунок 3.2 - Метод реєстрації консультанта

Метод register відповідає за обробку реєстрації. Після валідації введених даних викликається сервіс UserService, який створює новий запис у базі. У разі успішного виконання система автоматично авторизує новоствореного користувача та перенаправляє його на стартову сторінку онбордингу.

Метод register викликає сервіс UserService, який містить основну бізнес-логіку, пов’язану з обробкою даних консультанта. Метод create, представлений на рисунку 3.3, відповідає за створення нового користувача в таблиці users, передаючи необхідні параметри для збереження.

```
public function create(string $name, string $email, string $password): User
{
    $user = User::create([
        'email' => $email,
        'name' => $name,
        'password' => Hash::make($password),
    ]);

    return $this->updateTimezone($user);
}
```

Рисунок 3.3 - Створення нового консультанта

Окрім цього, у UserService також використовується метод updateTimezone, який відповідає за визначення часового поясу користувача та його збереження у відповідному полі таблиці. Для зручності часовий пояс визначається на основі IP-адреси, отриманої із запиту. Це дозволяє автоматично дізнатися приблизне місцезнаходження користувача, а отже - і його часову зону, без додаткових дій з його боку.

Аналогічним чином функціонує компонент авторизації користувача. Метод login, представлений на рисунку 3.4, виконує валідацію вхідних даних і, у разі успішної перевірки, здійснює логін користувача в систему, створюючи нову сесію. Метод login створює екземпляр класу LoginRequest, куди передаються електронна пошта, пароль та параметр запам’ятовування користувача. Далі запускається валідація за встановленими правилами, після чого виконується автентифікація користувача через метод authenticate. У разі успіху сесія регенерується для захисту від атак типу session fixation, і користувача перенаправляють на головну сторінку системи.

```
public function login()
{
    $request = new LoginRequest([
        'email' => $this->email,
        'password' => $this->password,
        'remember' => $this->remember
    ]);

    $this->validate($request->rules());
    $request->authenticate();

    session()->regenerate();

    return redirect()->intended(RouteServiceProvider::HOME);
}
```

Рисунок 3.4 - Метод логіну консультанта

Так само, як і компоненти реєстрації та входу, необхідно реалізувати механізм відновлення пароля. У випадку, якщо користувач забув свій пароль, система надсилає листа з посиланням на його електронну пошту. Перейшовши за цим захищеним посиланням, користувач має змогу ввести новий пароль. Такий підхід є як безпечним, так і зручним способом скидання пароля без участі адміністратора чи підтримки.

Після успішного створення акаунта, консультант автоматично потрапляє до персонального кабінету. Одним із перших кроків, які він має виконати, є налаштування власного робочого календаря. Це необхідно для того, щоб під час бронювання дзвінків клієнти мали можливість обрати вільний час, зручний як для себе, так і для консультанта. Такий підхід не лише покращує взаємодію між сторонами, а й забезпечує ефективне планування консультацій.

У календарі доступності реалізовано дві основні моделі: AvailablePeriod та Vacation. Перша з них, AvailablePeriod, є ключовою сутністю, яка містить дані про години, коли консультант відкритий до прийому клієнтів. Кожен запис включає три головні поля: day, from та till. Поле day фіксує день тижня - наприклад, понеділок або середу. Це зроблено для зручності: замість того щоб щотижня вручну виставляти графік, консультант задає типовий розклад, який автоматично повторюється. У полях from і till зазначаються години початку та

завершення робочого відрізка в межах цього дня. Метод, що відповідає за створення запису графіка роботи, зображений на рисунку 3.5.

```
public function saveTimeSlot()
{
    $this->handleTime();

    $data = $this->withValidator(function (Validator $validator) {
        $validator->after(function ($validator) {
            if ($this->checkFromAndTillAreNotCorrect()) {
                $validator->errors()->add('from_bigger_than_till', 'Incorrect times');
            }

            if ($this->checkTimeSlotOverlapping()) {
                $validator->errors()->add('slot_overlapping', 'Overlapping timeslot');
            }
        });
    })
        ->validate([
            'from' => 'required',
            'till' => 'required'
        ]);

    $data['day'] = $this->date['day_of_the_week'];
    auth()->user()->availablePeriods()->create($data);

    $this->setData();
    $this->showStatus();
    $this->createCalendarChangedEvent();
}
```

Рисунок 3.5 - Метод створення вільного періоду консультанта

Метод saveTimeSlot відповідає за додавання нового інтервалу доступності у календар. Спочатку викликається допоміжний метод, що готує дані часу до перевірки. Далі відбувається валідація - перевіряється, чи час початку не більший за час завершення, а також чи новий інтервал не перетинається з уже наявними записами. У випадку помилки користувач отримає відповідне повідомлення, що дозволить йому скоригувати дані. Якщо ж усе коректно, новий часовий слот зберігається в базу даних, календар оновлюється, і створюється подія, яка оновлює календар без перезавантаження сторінки.

Друга модель - Vacation, що призначена для ситуацій, коли консультант планує відпустку або будь-яку іншу тривалу відсутність. Усі час від часу потребують відпочинку, і щоб не змінювати вручну кожен запис у календарі, достатньо просто вказати період відпустки. Після цього система автоматично

заблокує можливість бронювання дзвінків на ці дати, і клієнти не зможуть обрати недоступний час. У моделі використовуються два поля - from та till, які фіксують початок і завершення періоду відсутності. Метод, що створює такий запис, показано на рисунку 3.6.

```
public function saveVacation(): void
{
    $data = $this->validate([
        'from' => ['required', 'date', 'after_or_equal:today'],
        'till' => ['required', 'date', 'after_or_equal:from'],
    ]);

    auth()->user()->vacations()->create($data);

    $this->applyChanges();
}
```

Рисунок 3.6 - Метод створення відпустки для консультанта

Метод saveVacation спочатку перевіряє, чи обидві дати коректні: чи дата початку не лежить у минулому, і чи дата завершення не передує початковій. Після успішної перевірки дані зберігаються в таблицю відпусток, а система застосовує відповідні зміни до календаря консультанта.

Наступним важливим кроком у розвитку системи є реалізація інтеграції зі сторонніми календарями. Існує три основні платформи, які дозволяють користувачам зберігати та керувати подіями онлайн: Google Calendar, Outlook Calendar та Apple Calendar. На жаль, остання не надає публічного API для доступу до календарів своїх користувачів, що фактично унеможливлює технічну інтеграцію з нею. Натомість Google та Outlook відкриті до зовнішніх рішень - обидві системи мають повноцінну документацію та API, які дозволяють реалізовувати гнучку й стабільну інтеграцію.

Підключення зовнішніх календарів - це не просто додатковий функціонал, а справді критично важлива опція для консультантів. У більшості з них дуже насичений графік, і часто події розподілені між кількома платформами. Консультації можуть проходити не лише через наш сервіс, а й через інші канали, тому синхронізація дозволяє уникнути конфліктів у розкладі. Окрім зручності для самого консультанта, інтеграція вирішує важливе завдання -

автоматичне блокування слотів для дзвінків у випадку, якщо на цей самий час уже запланована інша подія в зовнішньому календарі.

Для реалізації цієї функції в системі передбачено три ключові сутності: ExternalCalendarOauth, ExternalCalendar та ExternalCalendarEvent. Їхня структура взаємозв'язків зображена на рисунку 3.7. Центральною точкою є модель ExternalCalendarOauth - вона зберігає інформацію про підключений акаунт (наприклад, Gmail або Outlook), через який консультант авторизується за допомогою протоколу OAuth 2.0. Саме ця модель відповідає за збереження токенів доступу, оновлення дозволів і підтвердження зв'язку з відповідною зовнішньою платформою.

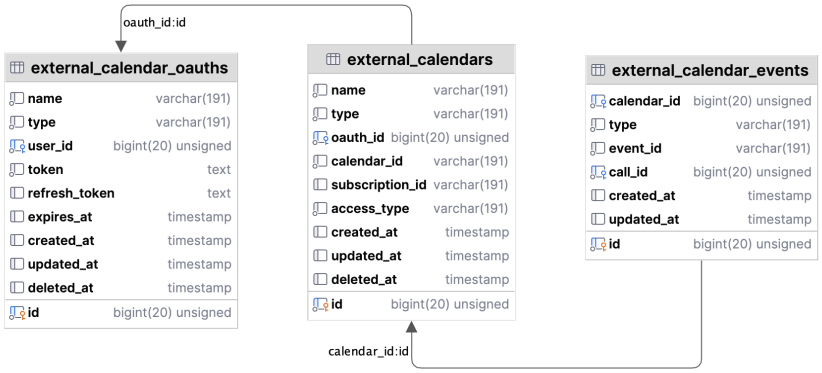

Рисунок 3.7 - Діаграма таблиць зовнішніх календарів

Кожен акаунт консультанта може містити кілька календарів, і для зберігання цієї інформації використовується модель ExternalCalendar. Вона представляє безпосередньо саму сутність календаря. Календарі поділяються за правами доступу на два основні типи: readable та writable. Readable-календарі призначені виключно для читання. Це означає, що система може переглядати події в такому календарі, але не має можливості вносити будь-які зміни чи додавати нові події. Класичним прикладом такого типу є календар державних свят, який присутній у багатьох акаунтах за замовчуванням - користувач може ознайомитися з подіями, однак не може їх редагувати чи доповнювати.

На відміну від цього, writable-календарі підтримують повний двосторонній обмін даними. Тобто, система не лише зчитує інформацію про вже

наявні події, а й має змогу записувати нові - наприклад, автоматично додавати туди консультації, заброньовані клієнтами. Це забезпечує зручність та автоматизацію процесу планування, зменшуючи ризики накладок у розкладі.

Кожен календар, у свою чергу, може містити велику кількість подій. За їх облік відповідає модель ExternalCalendarEvent. Вона зберігає назви подій, час їх початку та завершення, а також інші параметри, які можуть бути важливими для синхронізації. Під час бронювання нових дзвінків система має здійснювати запит на отримання вже запланованих подій із зовнішніх календарів, щоби перевірити доступність консультанта та запобігти накладенню зустрічей.

Розробку цієї частини функціоналу варто розпочати з отримання даних моделі ExternalCalendarOauth. Для цього використовується сучасний та безпечний протокол авторизації OAuth 2.0, який підтримується такими провідними постачальниками, як Google та Microsoft.

OAuth 2.0 - це відкритий стандарт авторизації, який дозволяє стороннім додаткам отримувати обмежений доступ до захищених ресурсів користувача без передачі й зберігання його пароля. У процесі авторизації користувач самостійно надає доступ до певних даних, таких як календарі, підтверджуючи свою згоду на спеціально призначеній сторінці сервісу-постачальника.

Процес починається з переадресації користувача на сторінку авторизації Google або Microsoft, де він обирає акаунт, який бажає підключити до нашого сервісу. Після цього відображається список дозволів, які додаток запитує для повноцінної роботи з календарем - наприклад, доступ до читання подій або можливість їх створення. Пройшовши всі етапи, користувач автоматично повертається назад на сторінку нашого веб-ресурсу, де система обробляє отримані токени й зберігає їх у базі даних, створюючи відповідний запис у таблиці ExternalCalendarOauth. Метод, який відповідає за обробку відповіді від Google, зображено на рисунку 3.8.

```
public function callback()
{
    try {
        $socialiteUser =
Socialite::driver('google')->redirectUrl(route('external-calendars.google.oauth.callback'))->user();

        $oauth = app(GoogleExternalCalendarOauthService::class)->create(
            $this->getUser(),
            $socialiteUser->email,
            $socialiteUser->token,
            $socialiteUser->refreshToken,
            now()->addSeconds($socialiteUser->expiresIn),
        );

        return $this->hasAllScopes($oauth) ? $this->redirectBackWithSuccess($oauth) :
$this->redirectBackWithError($oauth);
    } catch (Exception $exception) {
        return $this->redirectBack();
    }
}
```

Рисунок 3.8 - Обробка результату підключення календар аккаунта

Метод callback обробляє відповідь від Google після авторизації через OAuth 2.0. На першому етапі відбувається отримання об'єкта користувача через сервіс Socialite, зокрема витягується інформація про електронну пошту, токени доступу та час дії сесії. Далі ці дані передаються до сервісу GoogleExternalCalendarOauthService, який створює запис про підключення у базі даних. Після цього метод перевіряє, чи має обліковий запис всі необхідні дозволи для роботи з календарями. У разі успішної перевірки користувача перенаправляють із повідомленням про успішне підключення, інакше - з помилкою. У випадку винятків або помилок на будь-якому етапі користувач повертається на попередню сторінку без збереження змін.

Після успішного підключення акаунта, потрібно здійснити наступний крок - підключення календарів, які належать консультанту. Для цього необхідно спочатку отримати список доступних календарів з акаунта Google, після чого консультант самостійно обирає ті, які хоче використовувати у нашому сервісі. Обрані календарі записуються до бази даних, і в подальшому система може звертатись до них для зчитування або створення подій. Метод, що відповідає за отримання Google-календарів, зображений на рисунку 3.9.

```
private function requestGetAll(ExternalCalendarOauth $externalCalendarOauth): Collection
{
    $externalCalendarOauth =
app(GoogleExternalCalendarOauthService::class)->checkToken($externalCalendarOauth);

    $response = Http::withToken($externalCalendarOauth->token)->get(
        config('services.google.api_url') . '/calendar/v3/users/me/calendarList',
        [
            'fields' => 'items(id, summary, accessRole)',
        ],
    );

    if ($response->failed()) {
        $this->handleError($externalCalendarOauth);
        return collect();
    }

    return collect($response->json('items'));
}
```

Рисунок 3.9 - Отримання списку календарів користувача

Метод requestGetAll відповідає за отримання списку всіх календарів, підключених до акаунта консультанта через Google API. На початку відбувається перевірка актуальності токена доступу за допомогою сервісу GoogleExternalCalendarOauthService. Якщо токен застарілий, він автоматично оновлюється. Далі виконується HTTP-запит до Google API для отримання календарів, у якому вказано, що нас цікавлять лише ідентифікатор календаря, його назва та права доступу. У разі невдачі метод обробляє помилку та повертає порожню колекцію, щоб не порушити логіку програми. Якщо ж запит вдалий - система отримує і повертає колекцію календарів, з якими згодом може працювати консультант.

Коли клієнт бронює дзвінок, потрібно робити запит у зовнішній календар, щоб консультант бачив його та міг планувати майбутній графік, спираючись на розклад консультацій. Для цього, після створення дзвінка, потрібно робити запит. Приклад запиту до Google для створення нової події зображено на рисунку 3.10.

```
protected function requestCreateByCalendar(ExternalCalendar $externalCalendar, Call $call): ?array
{
    $externalCalendarOauth = app(GoogleExternalCalendarOauthService::class)->checkToken($externalCalendar->oauth);

    $response = Http::withToken($externalCalendarOauth->token)->post(
        config('services.google.api_url') . "/calendar/v3/calendars/{$externalCalendar->calendar_id}/events",
        $this->getCreationData($call),
    );

    if ($response->failed()) {
        $this->handleError($externalCalendar);
        return null;
    }

    return $response->json();
}

private function getCreationData(Call $call): array
{
    return [
        'start' => [
            'dateTime' => $call->getDateInTimezone('from')->toRfc3339String(),
            'timeZone' => $call->getDateInTimezone('from')->getTimezone(),
        ],
        'end' => [
            'dateTime' => $call->getDateInTimezone('till')->toRfc3339String(),
            'timeZone' => $call->getDateInTimezone('till')->getTimezone(),
        ],
        'summary' => $this->getTitle($call),
        'description' => $this->getCallDescription($call),
        'source' => [
            'title' => 'Online video call',
            'url' => $call->show_url,
        ],
        'location' => $call->show_url,
        'reminders' => [
            'useDefault' => false,
            'overrides' => [['method' => 'popup', 'minutes' => 60]],
        ],
    ];
}
```

Рисунок 3.10 - Створення події в календарі

Метод requestCreateByCalendar відповідає за створення події у зовнішньому календарі, зокрема у Google Calendar, після того як клієнт забронював дзвінок. На першому етапі перевіряється актуальність токена, пов’язаного з відповідним календарем, за допомогою сервісу GoogleExternalCalendarOauthService. Після успішної перевірки формується HTTP-запит типу POST до API Google, в якому передається необхідна інформація про подію. Якщо запит не проходить, викликається обробник помилок, і метод повертає null, щоб уникнути критичних збоїв. У випадку успіху метод повертає відповідь від API у вигляді масиву, що містить інформацію про створену подію.

Допоміжний метод getCreationData формує структуру даних, яку потрібно передати до API для створення події. Тут вказується дата та час початку і

завершення дзвінка з урахуванням часових поясів, назва події, її опис, посилання на відеозустріч, а також налаштування нагадувань. Це дозволяє створити подію, максимально наближену до реального формату зустрічі, яку зручно бачити в особистому календарі консультанта.

3.2 Реалізація модуля дзвінків

Наступним етапом розробки є створення модуля, пов’язаного з організацією дзвінків - одного з ключових елементів функціональності нашого додатку. Саме через цю частину системи відбувається основна взаємодія між клієнтом і консультантом: консультант створює дзвінки, клієнт має змогу їх бронювати, далі проходить сесія, після якої консультант отримує оплату. Таким чином, дзвінки є серцевиною усієї платформи.

Особливу увагу при проєктуванні цього модуля потрібно приділити гнучкості. Архітектура має однаково ефективно підтримувати як індивідуальні, так і групові сесії. Це дозволить у майбутньому масштабувати продукт без суттєвих переробок. Для досягнення цього важливо дотримуватись принципу DRY [21], який наголошує на необхідності мінімізації дублювання коду та повторення логіки.

Модуль дзвінків включає в себе велику кількість сутностей, кожна з яких виконує чітко визначену функцію та не дублює інформацію інших - що відповідає нормальним формам бази даних. Завдяки цьому система є стабільною, розширюваною та логічно структурованою. Структура сутностей модуля дзвінків і взаємозв’язки між ними детально відображені на рисунку 3.11.

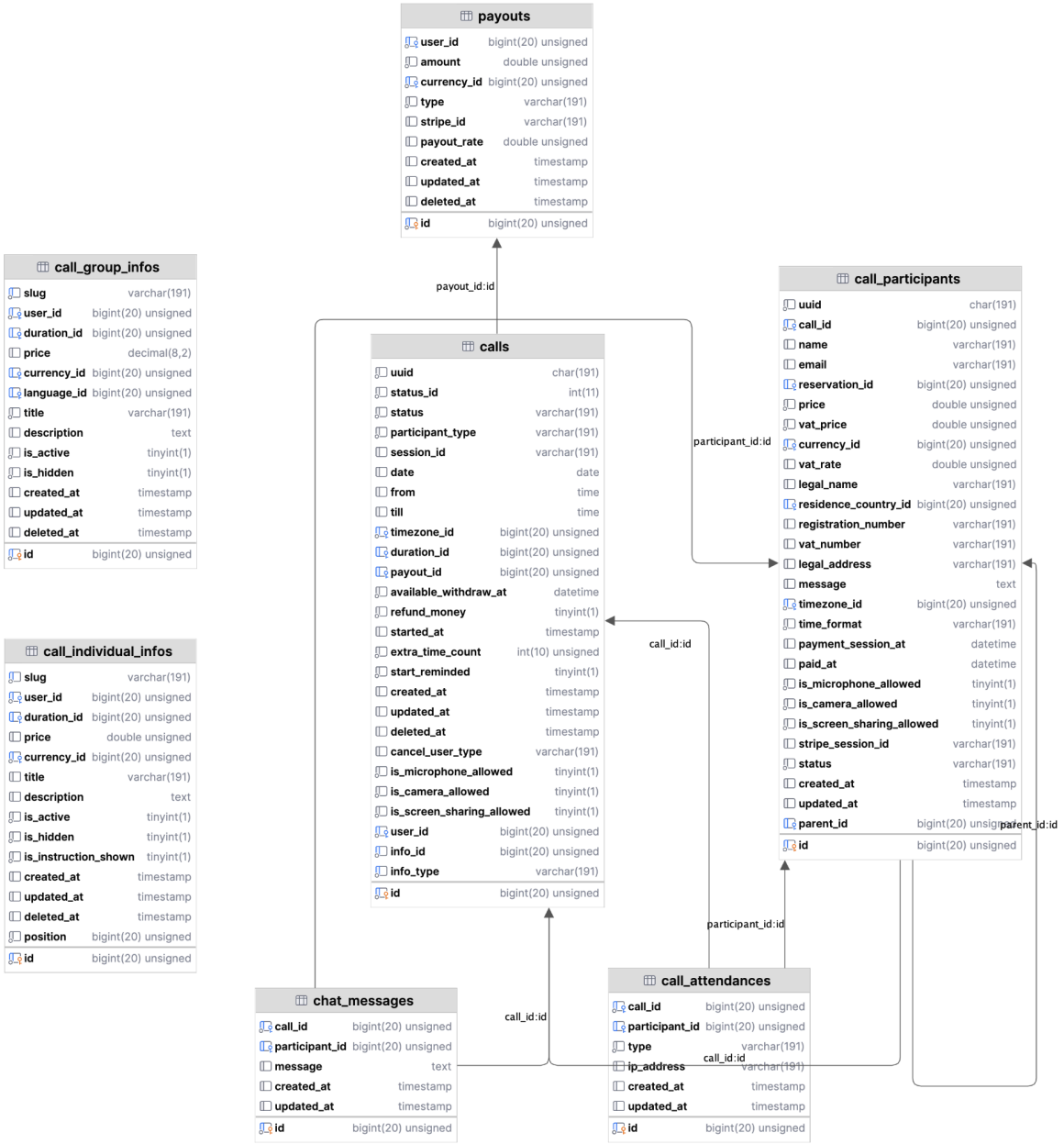

Рисунок 3.11 - Діаграма таблиць дзвінків

Сутність Call є центральною ланкою у модулі дзвінків і виступає так званою агрегатною сутністю за методологією DDD. Вона координує роботу багатьох пов’язаних сутностей, забезпечуючи узгодженість та цілісність бізнес-логіки. Саме в Call зберігається основна інформація про кожен дзвінок, його поточний стан, часові параметри, тип участі та фінансові дані.

Одним із ключових параметрів є статус дзвінка, який може мати одне з трьох значень: активний, завершений або відмінений. Активний статус означає, що дзвінок ще не відбувся або відбувається просто зараз. Завершений означає,

що сесія пройшла успішно, обидві сторони залишилися задоволені, і консультант має право на отримання винагороди. Відмінений статус застосовується у випадках, коли зустріч була скасована вручну або коли консультант не з’явився, і тоді кошти, як правило, повертаються клієнту.

Поліморфні поля info_id та info_type відіграють важливу роль у гнучкому зв’язуванні сутностей. Поліморфне поле – це такий тип зв’язку, який дозволяє одній сутності належати до кількох інших моделей одночасно через єдиний інтерфейс. У випадку Call, зв’язок може вести або до call_group_infos, або до call_individual_infos. Ці таблиці зберігають дані про послугу: її назву, тривалість, вартість та опис. Вони створюються консультантом у власному кабінеті. Проте після створення дзвінка інформація в Call залишається незмінною, навіть якщо консультант оновить відповідну інформацію про послугу. Це зроблено для збереження історичної достовірності та недопущення маніпуляцій.

Поле participant_type вказує на тип участі у дзвінку - груповий або індивідуальний. Від цього значення залежить поведінка коду: як саме обробляти дзвінок, які компоненти ініціалізувати та як формувати посилання на відеозв’язок.

Часові параметри зберігаються в полях date, from, till та timezone_id. Дата і час записуються у форматі UTC, що дозволяє уникнути проблем із часовими зонами при збереженні та обробці даних. А поле timezone_id зберігає інформацію про часову зону, яка потрібна для правильного відображення часу у звичному для користувача форматі.

Окрему категорію становлять поля, пов’язані з оплатою. Session_id зберігає ідентифікатор Stripe-сесії, який дозволяє дізнатись, скільки саме сплатив клієнт за дзвінок. Поле payout_id пов’язане з таблицею виплат і вказує, до якої конкретної виплати належить цей дзвінок. Одна виплата може включати кілька дзвінків. Поле available_withdraw_at позначає дату, коли консультант зможе вивести кошти. Щоб забезпечити захист прав клієнта, сервіс встановлює обмеження на виведення протягом п’яти днів після завершення дзвінка. Це дає

можливість користувачу звернутись у підтримку в разі проблем. Нарешті, поле refund_money є булевим і сигналізує, чи були повернуті кошти клієнту.

Таким чином, сутність Call виступає повноцінною бізнес-одиницею, яка поєднує логіку проведення дзвінків, взаємодію користувачів, фінансові механізми та часову координацію в єдиному цілісному елементі системи.

Розібравшись з усіма сутностями, перейдемо до реалізації створення сторінок з інформацією про індивідуальні та групові дзвінки. Як вже згадувалося раніше, ці сутності створює консультант, і саме вони зберігають усю важливу інформацію про майбутні послуги. На відповідних сторінках консультант може переглядати створені послуги, додавати нові, редагувати існуючі або видаляти ті, які більше не є актуальними. Такий функціонал забезпечує повну гнучкість та зручність для керування власним графіком і пропозиціями.

Індивідуальні послуги зазвичай містять мінімальний обсяг інформації. Це лише назва, короткий або розширений опис, тривалість консультації та її вартість. Всі параметри стосуються лише самої сутності послуги, адже дата і час проведення дзвінка визначаються пізніше - безпосередньо клієнтом під час бронювання. Клієнт бачить доступні вікна у графіку консультанта і самостійно обирає найзручніший час. Після цього в системі автоматично створюється запис дзвінка, пов’язаний із вибраною послугою. Метод, який відповідає за створення індивідуального плану, показаний на рисунку 3.12.

```
public function save()
{
    $data = $this->validate();

    $data['currency_id'] = auth()->user()->currency_id;
    auth()->user()->individualInfos()->create($data);

    $this->redirectRoute('home.manage.info.individual.index');
}
```

Рисунок 3.12 - Створення індивідуального плану

Метод save відповідає за збереження нової індивідуальної послуги, яку створює консультант у своєму кабінеті. Спочатку він проводить валідацію

введених даних, після чого додає до них ідентифікатор валюти, взятий з профілю консультанта. Потім за допомогою Eloquent-зв’язку створюється новий запис у таблиці індивідуальних послуг, який автоматично зв’язується з консультантом. Після успішного створення виконується перенаправлення на сторінку зі списком усіх індивідуальних послуг, де можна переглянути щойно додану.

Групові дзвінки, на відміну від індивідуальних, мають заздалегідь визначені дату та час проведення, оскільки неможливо погодити одну спільну дату з великою кількістю клієнтів. Консультант самостійно встановлює ці параметри, а клієнти просто бронюють місце на запропонований час. Крім того, групові дзвінки містять більше інформації, що допомагає привернути увагу потенційних клієнтів. Зокрема, вони можуть включати зображення та додаткові файли, які роблять опис послуги більш привабливим і зрозумілим. Метод, що відповідає за створення групового дзвінка, показано на рисунку 3.13.

```
public function save(array $data): void
{
    DB::transaction(function () use ($data) {
        $info = auth()->user()->groupInfos()->create($data);

        if ($data['date'] && $data['from'] && $data['till']) {
            $this->createCall($info, $data);
        }

        if ($data['cover'] || $this->duplicateInfo) {
            $this->handleMedia($info, 'cover', $data['cover']);
            OptimizeModelImagesJob::dispatch($info, 'cover');
        }

        if ($data['images'] || $this->duplicateInfo) {
            $this->handleMediaCollection($info, 'images', $data['images']);
            OptimizeModelImagesJob::dispatch($info, 'images');
        }

        if ($data['files'] || $this->duplicateInfo) {
            $this->handleMediaCollection($info, 'files', $data['files']);
        }

        $info = $info->refresh();
        if ($info->is_active) {
            CallGroupInfoActivatedEvent::dispatch($info);
        }

        $this->redirectRoute('home.manage.info.group.index');
    });
}
```

Рисунок 3.13 - Створення групового дзвінка

Метод save відповідає за збереження нової групової інформації про дзвінок у базі даних. У межах транзакції спочатку створюється запис про групову послугу, прив’язаний до поточного користувача. Якщо у вхідних даних задані дата, час початку та завершення - автоматично створюється дзвінок. Далі, якщо передано обкладинку або здійснюється дублювання інформації, до сутності прикріплюється медіафайл типу "cover", а також запускається фонове завдання на оптимізацію зображення. Аналогічні дії виконуються для колекцій зображень та додаткових файлів - вони зберігаються через відповідні методи обробки медіа та оптимізуються, якщо потрібно. Завершується процес перенаправленням користувача на сторінку з усіма груповими послугами для подальшого управління.

Після створення планів консультант поширює інформацію про дзвінки через соціальні мережі або безпосередньо серед своїх клієнтів. Клієнти переходять на персональну сторінку консультанта, де можуть обрати послугу, що їх зацікавила. У випадку індивідуального дзвінка, клієнт обирає зручний час із переліку доступних слотів. Ці слоти формуються на основі календаря, який консультант заповнює у своєму кабінеті, з урахуванням відпусток, вже заброньованих консультацій та подій у зовнішніх календарях. Якщо ж ідеться про груповий дзвінок, клієнт знайомиться з детальним описом послуги та бронює участь на запропоновану дату і час.

У процесі бронювання дзвінка система створює дві основні сутності: Call та CallParticipant. Сутність Call містить загальну інформацію про сам дзвінок, тоді як CallParticipant - деталі про конкретного клієнта, який бере участь. У випадку групового дзвінка одна сутність Call може об’єднувати багато CallParticipant, адже кілька клієнтів можуть приєднатися до однієї сесії. Метод створення Call зображено на рисунку 3.14, а метод створення CallParticipant - на рисунку 3.15.

```
private function createCall(): Call
{
    $timezone = $this->getTimezone();

    $from = Carbon::parse("{$this->selectedDate} {$this->selectedTimeSlotFrom}")->shiftTimezone($timezone->name)->utc();
    $till = Carbon::parse("{$this->selectedDate} {$this->selectedTimeSlotTill}")->shiftTimezone($timezone->name)->utc();

    return $this->info->calls()->create([
        'user_id' => $this->user->id,

        'participant_type' => CallParticipantType::INDIVIDUAL,
        'duration_id' => $this->info->duration_id,

        'date' => $from->toDateString(),
        'from' => $from->toTimeString(),
        'till' => $till->toTimeString(),
    ]);
}
```

Рисунок 3.14 - Метод створення Call

Метод createCall відповідає за створення нової сутності дзвінка на основі вибраної клієнтом дати та часу. Спочатку він визначає часовий пояс користувача та переводить вибрані дату і час початку й завершення дзвінка в UTC для коректного зберігання в базі даних. Далі створюється запис у таблиці calls, де зберігається інформація про користувача, тип дзвінка (у цьому випадку індивідуальний), тривалість, дата та час проведення консультації.

```
protected function createParticipant(Call $call, array $data): CallParticipant
{
    return $call->participants()->create([
        'name' => $data['userName'],
        'email' => $data['userEmail'],
        'price' => $this->info->price,
        'currency_id' => $this->info->currency_id,
        'message' => $data['message'],
        'timezone_id' => $this->getTimezone()->id,
        'time_format' => $this->getTimeFormat(),
        'payment_session_at' => now()->format('Y-m-d H:i:s'),
    ]);
}
```

Рисунок 3.15 - Метод створення CallParticipant

Метод createParticipant створює сутність CallParticipant, яка містить дані клієнта, що забронював дзвінок. Він прив’язується до раніше створеного дзвінка і зберігає такі дані як ім’я, email, повідомлення клієнта, ціна та валюта дзвінка, ідентифікатор часового поясу, формат часу, а також дату й час початку сесії оплати. Цей метод забезпечує персоналізацію та подальше опрацювання замовлення конкретного користувача.

Після створення сутностей Call та CallParticipant, наступним кроком є отримання оплати від клієнта. Для цього використовується функціонал платіжних сесій у Stripe. Ми ініціюємо створення сесії, передаючи всі необхідні дані, зокрема назву консультації, вартість, валюту та контактну інформацію. Після цього клієнт автоматично перенаправляється на сторінку Stripe, де бачить короткий опис послуги, її ціну та форму для введення платіжних реквізитів. Метод, що відповідає за створення Stripe-сесії, наведений на рисунку 3.16.

```
public function createSession(Call $call, CallParticipant $participant, string $cancelUrl = null): Session
{
    Stripe::setApiKey(config('services.stripe.secret'));

    return Session::create($this->getSessionData($call, $participant, $cancelUrl));
}

private function getSessionData(Call $call, CallParticipant $participant, string $cancelUrl = null): array
{
    $data = [
        'success_url' => $participant->getConfirmationUrl(),
        'cancel_url' => $cancelUrl ?? URL::signedRoute('participants.cancellation', ['participant' => $participant->id]),
        'mode' => 'payment',
        'customer_email' => $participant->email,
        'client_reference_id' => $participant->id,
        'payment_method_types' => [
            'card'
        ],
        'metadata' => ['participant_id' => $participant->id],
        'expires_at' => $this->getExpiresAt()->timestamp,
        'line_items' => [
            [
                'price_data' => [
                    'product_data' => [
                        'name' => $call->info->title,
                    ],
                    'unit_amount' => $participant->total_price * 100,
                    'currency' => $participant->currency->code,
                ],
                'quantity' => 1,
            ],
        ],
    ];

    $user = $call->user;
    if ($user->validated_by_stripe && $user->residence_country_stripe_supports) {
        $data['payment_intent_data'] = [
            'application_fee_amount' => ($participant->payout_commission * 100),
            'transfer_data' => [
                'destination' => $user->stripe_account_id,
            ],
        ];

        if (!$user->residence_country_stripe_supports_transfers) {
            $data['payment_intent_data']['on_behalf_of'] = $user->stripe_account_id;
        }
    }

    return $data;
}
```

Рисунок 3.16 - Метод створення Stripe сесії

Метод createSession відповідає за створення платіжної сесії Stripe для конкретного дзвінка та його учасника. Спершу він встановлює секретний ключ API Stripe, отриманий із конфігурації застосунку, після чого викликає метод getSessionData, який формує масив з усіма необхідними параметрами для ініціалізації платіжної сесії. Далі цей масив передається у Stripe для створення

сесії, яка повертається для подальшої роботи, наприклад, для переадресації користувача на сторінку оплати.

Метод getSessionData формує детальну конфігурацію платіжної сесії Stripe. Він задає URL-адреси для успішної оплати та скасування, режим оплати, email клієнта, ідентифікатор клієнта та інші метадані. Основним елементом є опис товару, його вартість і валюта, що передаються через параметр line_items. Додатково, якщо консультант має верифікований Stripe-акаунт і підтримується його країна проживання, до запиту додається інформація про комісію та передачу коштів на акаунт консультанта.

Після успішної оплати клієнт повертається на наш сайт, де відбувається підтвердження бронювання. Система надсилає повідомлення як консультанту, так і клієнту про успішне створення дзвінка та передає посилання для підключення. У випадку, якщо оплата не була завершена, створені раніше записи про дзвінок видаляються, звільняючи обране місце та дозволяючи клієнту повторно обрати послугу й здійснити оплату знову.

3.3 Реалізація модуля відео сесій

Після завершення реалізації ключових модулів консультанта та клієнта, настає черга одного з найважливіших етапів розробки платформи - створення модуля відео сесій. Саме тут відбувається безпосередня взаємодія між консультантом і клієнтом у форматі живого спілкування. Для забезпечення якісного та стабільного з'єднання було обрано Zoom Video SDK, який надає широкий спектр можливостей.

Серед них - підтримка великої кількості учасників у дзвінку, високоякісна передача відео й аудіо, трансляція екрана, мінімальні затримки та загалом висока надійність. При цьому важливо розуміти, що Zoom відповідає виключно за передачу даних між користувачами, тоді як усі інші аспекти - логіка сесії, інтерфейс та керування подіями реалізовуються на нашому боці. Це дозволяє нам повністю контролювати процес, адаптувати взаємодію під потреби платформи та забезпечити максимально зручний досвід користування.

Однією з переваг Zoom є його чудова технічна документація, яка дозволяє швидко інтегрувати SDK у будь-який проєкт. Існує декілька варіантів реалізації відео сесій: для платформ Android, iOS, Windows та WEB. У нашому випадку ідеальним вибором є саме WEB-рішення, оскільки ми створюємо WEB-додаток, доступний з будь-якого пристрою, що має сучасний браузер.

Zoom Video SDK надає два ключові інструменти для реалізації відео взаємодії: публічне API та JavaScript бібліотеку. API використовується для створення відео сесії на боці сервера, а також для логування процесу дзвінка. Це особливо корисно при виникненні помилок або для аналізу поведінки користувачів. JavaScript бібліотека дозволяє підключати учасників до сесії, передавати відео й аудіо потоки, обмінюватись текстовими повідомленнями у форматі JSON, а також реагувати на численні події - такі як підключення чи відключення користувачів, початок мовлення, надсилання повідомлень та інші.

Реалізація відео функціоналу починається зі створення сесії через API. Після цього її ідентифікатор зберігається в базі даних, щоб згодом використовуватись у JavaScript коді для підключення користувачів до відповідного дзвінка. Метод збереження Zoom-сесії наведено на рисунку 3.17, а приклад запиту до API - на рисунку 3.18.

```
public function handle(CallStartedEvent $event)
{
    $call = $event->call;

    $session = app(ZoomApiService::class)->createSession($call);
    $call->update(['session_id' => $session['session_id'] ?? null]);
}
```

Рисунок 3.17 - Метод зберігання Zoom сесії

Метод handle викликається у відповідь на подію CallStartedEvent, яка спрацьовує на початку дзвінка. У цьому методі отримується об’єкт дзвінка, після чого за допомогою сервісу ZoomApiService створюється нова Zoom-сесія для цього дзвінка. Після отримання відповіді від Zoom, ідентифікатор створеної сесії, session_id, зберігається у відповідному полі моделі Call. Це дозволяє

надалі пов’язати конкретний дзвінок з відповідною Zoom-сесією, щоб користувачі могли до неї приєднатись.

```
public function createSession(Call $call): array
{
    $response = $this->getHttpClient()->post('sessions', [
        'session_name' => $call->uuid,
    ]);

    return $response->json();
}

private function getToken(): string
{
    return JWT::encode([
        'iss' => config('services.zoom.api_key'),
        'iat' => now()->timestamp,
        'exp' => now()->addHours(48)->timestamp,
    ], config('services.zoom.api_secret'), 'HS256', head: ['alg' => 'HS256', 'typ' => 'JWT']);
}

private function getHttpClient()
{
    $apiUrl = config('services.zoom.api_url');
    return Http::withToken($this->getToken())->baseUrl("{$apiUrl}/v2/videosdk");
}
```

Рисунок 3.18 - Метод з запитом на створення Zoom сесії

Метод createSession відповідає безпосередньо за створення сесії у Zoom через HTTP-запит до їх API. Він надсилає POST-запит на адресу sessions, передаючи назву сесії як унікальний ідентифікатор дзвінка, uuid. Метод використовує HTTP-клієнт, який створюється за допомогою getHttpClient. У цьому клієнті в заголовок додається авторизаційний токен, згенерований методом getToken. Сам токен створюється у форматі JWT, де задається ключ API, час створення, а також час завершення дії токена - у цьому випадку через 48 годин. Метод getHttpClient формує з'єднання з базовим URL Zoom Video SDK, додаючи згенерований токен у заголовок. У підсумку, метод createSession повертає відповідь Zoom у вигляді масиву з інформацією про створену сесію, включно з її ідентифікатором.

Маючи ідентифікатор створеної Zoom-сесії, можемо переходити до розробки фронтенд компонента, який буде відповідати за підключення користувачів до відеодзвінка. Для реалізації цього інтерфейсу використаємо комбінацію технологій Livewire та JavaScript. Livewire забезпечить початкову структуру компонента та дозволить керувати логікою з боку сервера, наприклад,

передачею параметрів сесії або перевіркою прав доступу. JavaScript, у свою чергу, відповідатиме за обробку взаємодії користувача в реальному часі, що особливо важливо при великій кількості учасників, коли критичною стає швидкодія інтерфейсу. Адже Livewire, працюючи через HTTP-запити до сервера, не завжди може миттєво реагувати на всі дії користувачів.

Комбінація цих двох технологій дозволяє побудувати зручний і функціональний інтерфейс: Livewire забезпечує надійну серверну логіку, а JavaScript - плавну та динамічну роботу користувацького інтерфейсу. Розробку розпочнемо з побудови структури компонентів, яка наочно представлена на рисунку 3.19.

```
<div wire:ignore x-data id="video-room-container" class="flex bg-room-bg">
    <div class="w-full flex flex-col h-[100dvh]">
        <livewire:components.video.header :call="$call" :participant-id="$participantId" />

        <main class="flex-1 relative flex flex-row justify-center px-2 py-1 overflow-hidden">
            @include('components.video.room.info')

            <div class="w-full h-full" id="content">
                <livewire:components.video.video-room-content :call="$call" :participant-id="$participantId"
/>
            </div>
        </main>

        <livewire:components.video.footer :call="$call" :participant-id="$participantId" />

        @include('components.video.room.notifications')
        @include('components.video.room.host-attendance')
    </div>

    <div class="h-screen flex z-100 relative divide-x">
        <x-video.room.sections.devices-section />
        <x-video.room.sections.chat-section />
        <x-video.room.sections.participants-list-section />
    </div>

    <livewire:components.video.call-assessment-form :call="$call" :participant-id="$participantId"/>
</div>
```

Рисунок 3.19 - Структура компонентів дзвінка

Структура відео-кімнати реалізована у вигляді контейнера з ідентифікатором video-room-container, який ігнорується Livewire для того, щоб JavaScript міг безперешкодно працювати з DOM-елементами. Вся основна частина інтерфейсу знаходиться всередині блоку, який займає всю висоту екрану. У верхній частині розміщено компонент заголовка, що відображає інформацію про дзвінок. Центральна зона містить головний вміст кімнати, в якому знаходиться ще один Livewire-компонент video-room-content,

відповідальний за вивід відео, а також додаткову інформацію про дзвінок. У нижній частині розташовано компонент футера, де розміщені елементи управління сесією, такі як мікрофон, камера, вихід з кімнати тощо. Окремо також підключено шаблони для сповіщень та інформації про присутність хоста.

Поруч з основною частиною інтерфейсу знаходиться додаткова вертикальна панель, поділена на три секції: керування пристроями, чат і список учасників, які реалізовані за допомогою Blade-компонентів. Ці секції забезпечують зручну навігацію та взаємодію між учасниками дзвінка. Наприкінці шаблону також підключається Livewire-компонент call-assessment-form, який відповідає за оцінювання дзвінка після його завершення. Така структура дозволяє чітко розділити логіку та інтерфейс на окремі частини, забезпечуючи масштабованість та зручність у підтримці коду.

Продовжуючи реалізацію модуля відео сесій, наступним кроком є створення JavaScript-файлів, які забезпечують повноцінну інтеграцію з Zoom Video SDK. Основна мета цього етапу - організувати підключення користувача до дзвінка, забезпечити передачу аудіо та відео в реальному часі, а також отримувати інформацію про інших учасників сесії. Для цього необхідно ініціалізувати Zoom SDK, обробити підключення до сесії за допомогою наданого session id, налаштувати передачу медіа-даних і підписатися на ключові події, такі як вхід або вихід учасників, початок мовлення, зміни стану мікрофону та камери тощо.

Файл, що відповідає за безпосереднє підключення до Zoom, грає важливу роль - саме в ньому виконується ініціалізація Zoom-клієнта, передача автентифікаційного токену, налаштування параметрів сесії та старт основного потоку взаємодії з платформою. Цей скрипт забезпечує стабільне підключення до сервера Zoom, після чого розпочинається активна участь користувача у відео сесії. Саме цей файл, що реалізує базове під’єднання до Zoom Video SDK, зображено на рисунку 3.20.

```
export default {
    video: null,
    session: null,

    startSession() {
        this.video = ZoomVideo.createClient()

        return this.video.init('en-US', 'Global', { patchJsMedia: true })
            .then(async () => {
                await this.video.join(serverState.call.uuid, serverState.accessToken, serverState.user.name)
                this.session = this.video.getMediaStream();
            })
            .catch(() => {
                connection.handleReconnect();
            });
    },

    endSession() {
        this.video.leave();
    },

    getCurrentUser() {
        return this.video.getCurrentUserInfo();
    },

    getAllParticipants() {
        const participantsList = this.video.getAllUser();
        return participants.getUniqueParticipants(participantsList);
    },
    getOtherParticipants() {
        const participantsList = this.video.getAllUser()
            .filter(user => {
                return (user.userId !== this.video.getCurrentUserInfo().userId);
            });

        return participants.getUniqueParticipants(participantsList);
    },
    hasOtherUsers() {
        return (this.video.getAllUser().length > 1);
    },

    getParticipant(userId) {
        return this.video.getAllUser().find(user => user.userId === userId);
    }
}
```

Рисунок 3.20 - Файл з інтеграцією бібліотекою ZoomSdk

Цей JavaScript-файл відповідає за основну логіку підключення до відеосесії через Zoom Video SDK та управління учасниками дзвінка. У методі startSession ініціалізується клієнт Zoom, вказується мова інтерфейсу та параметри, після чого користувач приєднується до сесії, використовуючи унікальний ідентифікатор виклику, токен доступу та своє ім’я. Після успішного приєднання зберігається об’єкт медіа-потоку (тобто доступ до аудіо та відео). Якщо виникає помилка під час підключення, викликається функція повторного з’єднання handleReconnect, яка відповідає за стабільність зв’язку.

Також у файлі реалізовано низку допоміжних методів для взаємодії з учасниками відеосесії. Метод getCurrentUser повертає інформацію про поточного користувача, тоді як getAllParticipants та getOtherParticipants дозволяють отримати список усіх або лише інших учасників відповідно, з використанням додаткової обробки через функцію

participants.getUniqueParticipants. Метод hasOtherUsers визначає, чи є в дзвінку хтось, окрім поточного користувача, а getParticipant дозволяє знайти конкретного учасника за його ідентифікатором. Цей модуль формує фундамент для динамічної роботи відеозв'язку та реактивного відображення учасників у реальному часі.

Після реалізації базового підключення до Zoom Video SDK, наступним кроком є підписка на події (івенти), пов’язані з користувачами у дзвінку. Zoom надає зручний інтерфейс для відстеження змін у складі учасників сесії. Основними подіями, що стосуються користувачів, є user-added, user-removed та user-updated. Вони дозволяють відреагувати на зміни в режимі реального часу, забезпечуючи динамічне оновлення інтерфейсу відеозв’язку.

Коли спрацьовує подія user-added, це означає, що до дзвінка щойно приєднався новий учасник. У відповідь на це необхідно створити новий блок у DOM, в якому буде відображене ім’я користувача та трансляція його відео. Подія user-removed, навпаки, спрацьовує при виході користувача - в такому випадку його блок видаляється з інтерфейсу, а якщо він демонстрував екран, трансляцію потрібно зупинити. Подія user-updated є найбільш гнучкою - вона активується при будь-яких змінах налаштувань користувача: вмиканні або вимиканні мікрофона чи камери, зміні пристроїв, а також інших діях. Ці зміни потрібно оперативно відображати на екрані, щоб інтерфейс залишався актуальним і зрозумілим для всіх учасників.

```
zoom.video.on('user-added', (payload) => {
    if ((payload[0].userIdentity === zoom.getCurrentUser()?.userIdentity) && (payload[0].userId !== zoom.getCurrentUser()?.userId)) {
        connection.handleDisconnect();
        return;
    }

    this.setParticipants();
    this.handleConnect(payload[0]);

    reaction.handleCurrentReaction();
});

zoom.video.on('user-removed', (payload) => {
    this.setParticipants();
    this.handleDisconnect();
});

zoom.video.on('user-updated', (payload) => {
    const isCurrentUser = (payload[0].userId === zoom.getCurrentUser()?.userId);
    const isUserConnected = this.isParticipantConnected(payload[0].userId);

    if (!isCurrentUser && isUserConnected) this.render();
});
```

Рисунок 3.21 - Обробка івентів з участниками дзвінка

Цей фрагмент коду демонструє підписку на основні Zoom-івенти, пов’язані з учасниками дзвінка: user-added, user-removed та user-updated. При виклику події user-added перевіряється, чи не дублюється поточний користувач, що іноді може траплятись через баги з reconnection. Якщо це сталося, викликається метод handleDisconnect для повторного підключення. Далі оновлюється список учасників через setParticipants, викликається метод handleConnect для відображення нового користувача в інтерфейсі та запускається оновлення реакцій, якщо такі є.

Подія user-removed спрацьовує при відключенні учасника. У відповідь відбувається оновлення списку учасників і видалення його відеоблоку з екрана. Подія user-updated використовується для реагування на зміни у статусі користувача, наприклад, якщо він ввімкнув або вимкнув камеру чи мікрофон. Якщо змін зазнав не поточний користувач і він при цьому вже підключений, то викликається метод render, який перерисовує відповідну частину інтерфейсу. Такий підхід дозволяє оперативно відображати зміни й підтримувати актуальний стан відеосесії для всіх учасників.

3.4 Висновок

У третьому розділі було повністю реалізовано WEB-ресурс для онлайн консультацій, що включає всі необхідні функціональні модулі для зручної взаємодії між консультантами та клієнтами. Було створено особистий кабінет консультанта, який дозволяє вводити та редагувати особисту інформацію, керувати запланованими дзвінками, створювати нові консультації, налаштовувати свій графік роботи, а також інтегрувати зовнішні сервіси для дзвінків. Інтерфейс побудовано з урахуванням зручності та логіки, що дозволяє консультанту ефективно працювати з системою та швидко знаходити потрібну інформацію.

Крім цього, розроблено повноцінний модуль для бронювання та проведення консультацій. Клієнт може переглядати список доступних дзвінків, вивчати опис кожного з них і бронювати зручний для себе час. Оплата послуг

здійснюється через надійну міжнародну платіжну систему Stripe, що гарантує безпеку транзакцій. Консультант, у свою чергу, має можливість переносити чи скасовувати зустрічі, і у випадку відміни гроші автоматично повертаються клієнту. Усе це забезпечує прозору та зручну взаємодію між сторонами.

Особливу увагу було приділено реалізації відеосесій з використанням Zoom Video SDK. Завдяки цьому консультанти та клієнти можуть проводити онлайн зустрічі у високій якості з мінімальною затримкою. Передбачена можливість демонстрації екрана, зміни аудіо- та відеопристроїв, що робить процес спілкування максимально гнучким. Консультант, як головний учасник сесії, має додаткові повноваження - наприклад, може вимикати мікрофони або камери учасників при потребі. Такий підхід забезпечує стабільність та контроль над консультаційним процесом.

Наступним важливим кроком стане етап тестування розробленого продукту. Оскільки функціонал системи досить широкий, необхідно переконатися в його коректній роботі та надійності. Тестування дозволить виявити потенційні помилки, покращити юзер-досвід і підготувати платформу до повноцінного запуску для реальних користувачів.

4 ТЕСТУВАННЯ WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ

4.1 Тестування модуля консультанта

Тестування розпочнемо з перевірки модуля аутентифікації. Спершу переходимо на сторінку входу, де перевіряємо поведінку форми у різних сценаріях: якщо користувач не вводить жодних даних - система коректно показує повідомлення про помилки валідації. У разі введення неіснуючого користувача - з'являється повідомлення про некоректні облікові дані, як це видно на рисунку 4.1.

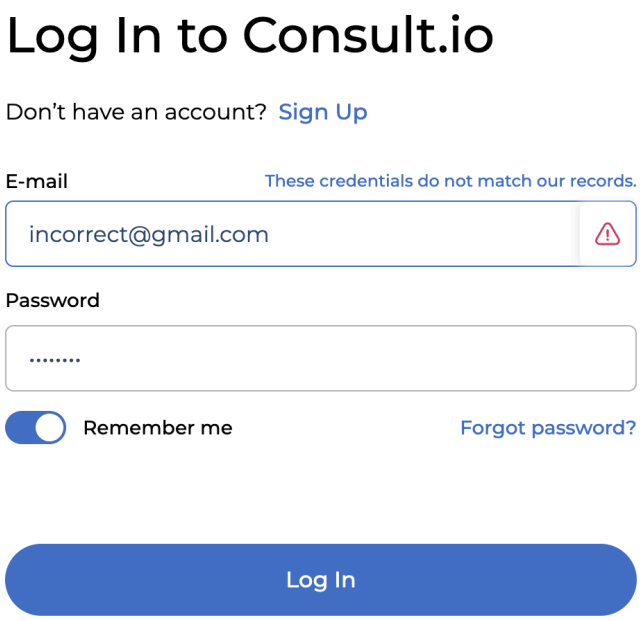

Рисунок 4.1 - Валідація форми логіну

Далі переходимо до тестування реєстрації нового користувача. Якщо натиснути кнопку реєстрації, не заповнивши жодного поля, відображаються відповідні повідомлення про необхідність заповнення обов'язкових полів, як показано на рисунку 4.2. При спробі зареєструватися з емейлом, який уже існує в системі, також з'являється зрозуміла помилка.

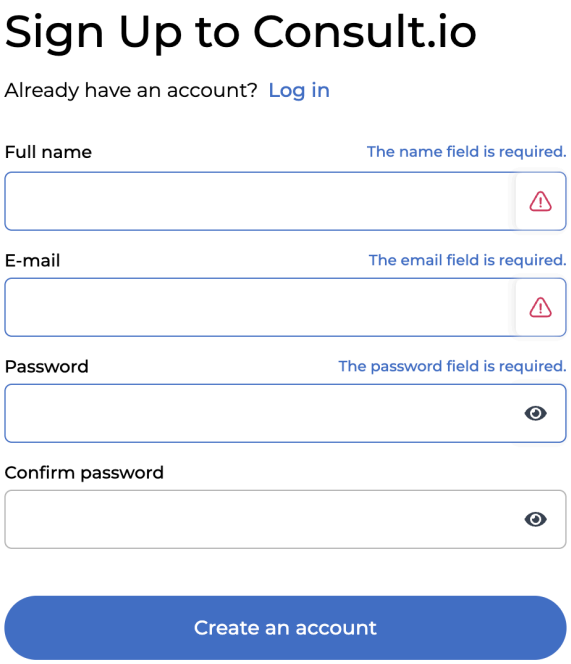

Рисунок 4.2 - Валідація форми реєстрації

Наступний крок - перевірка функції відновлення пароля. Якщо користувач натисне «Forgot password» на сторінці входу, відкривається модальне вікно з полем для введення електронної пошти. Після введення адреси, на неї надсилається лист із посиланням на сторінку для зміни пароля. Перейшовши за цим посиланням, користувач зможе ввести новий пароль, після чого отримає доступ до свого облікового запису. Процес відновлення пароля зображено на рисунку 4.3.

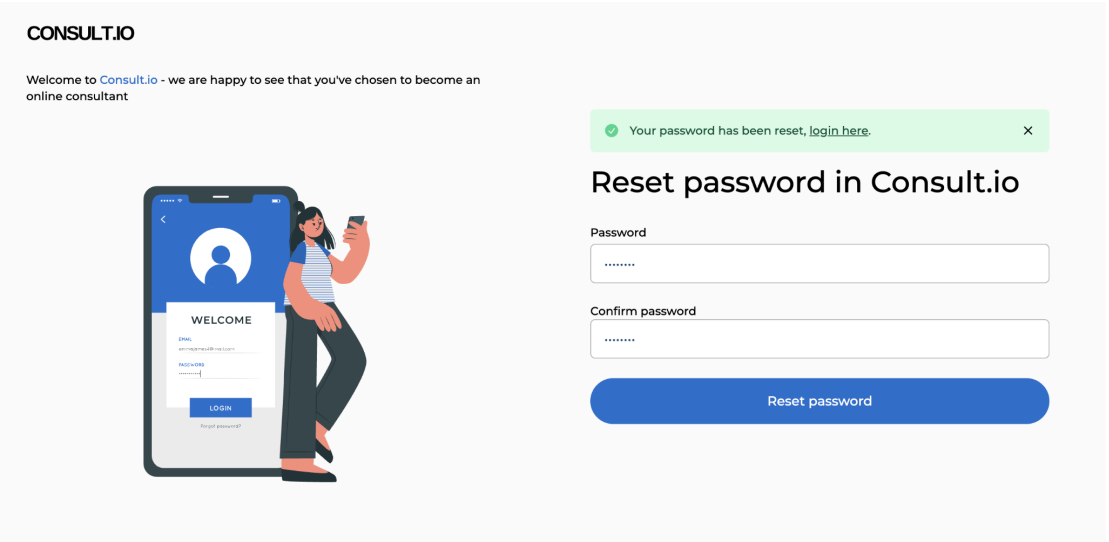

Рисунок 4.3 - Форма відновлення пароля

Завершивши перевірку аутентифікації, переходимо до тестування особистого кабінету консультанта. Однією з ключових функцій є налаштування календаря з доступним часом, у який клієнти можуть бронювати зустрічі. Якщо консультант намагається створити новий часовий інтервал, який перекривається з уже існуючим, система автоматично відображає помилку, як це видно на рисунку 4.4. Це дозволяє уникнути плутанини в розкладі та гарантує чітку організацію робочого часу.

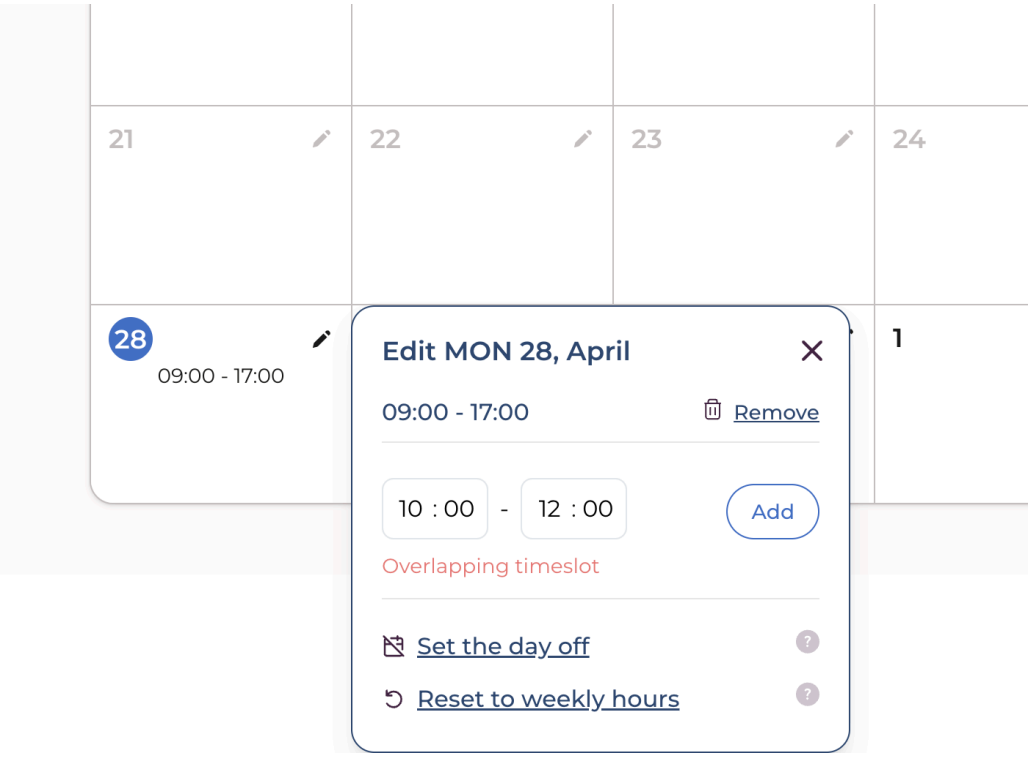

Рисунок 4.4 - Календар доступного часу

Наступним етапом тестування стане перевірка форми редагування особистих даних консультанта. Особливу увагу варто приділити завантаженню аватару чи інших медіа-файлів. Система повинна обмежувати можливість додавання надто великих файлів, оскільки це створює навантаження як на сервер, що має зберігати ці файли, так і на клієнтів, які змушені довше чекати на завантаження. Таким чином, якщо консультант намагається завантажити файл розміром понад 10 мегабайтів, відображається повідомлення про помилку - як показано на рисунку 4.5.

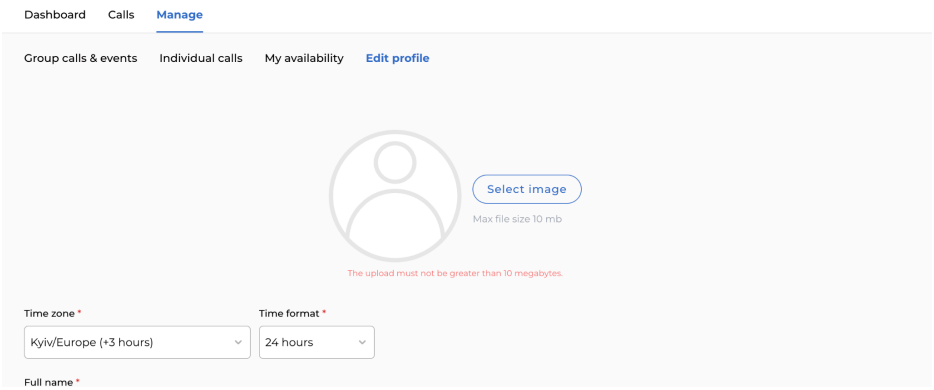

Рисунок 4.5 - Форма редагування особистої інформації

Далі тестуємо форму створення дзвінків, яку заповнює консультант. Тут надзвичайно важливо перевірити, чи всі поля мають правильну валідацію. Вона повинна враховувати тип даних, обов'язковість заповнення, а також логіку введених значень. Наприклад, система не повинна дозволяти створення дзвінка із від'ємною вартістю - така ситуація є абсурдною з погляду бізнес-логіки. У разі спроби введення недопустимих значень користувач отримає відповідне попередження, як це проілюстровано на рисунку 4.6. Така ретельна перевірка забезпечує стабільність функціонування системи й зменшує ймовірність людських помилок під час взаємодії з платформою.

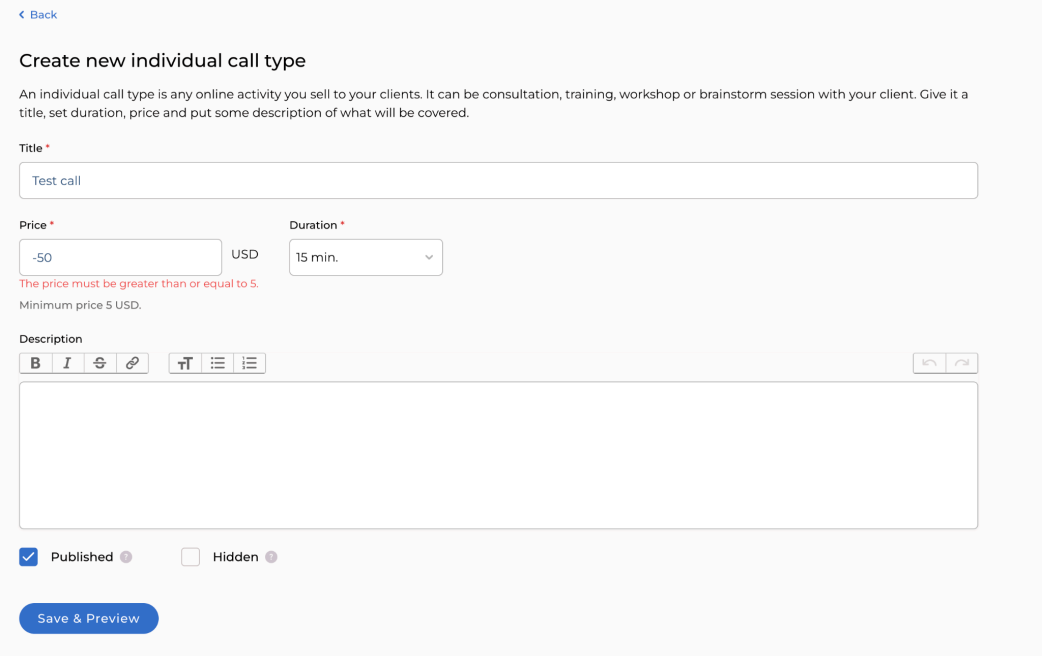

Рисунок 4.6 - Форма створення індивідуального дзвінка

4.2 Тестування модуля дзвінків

Розпочнемо тестування модуля дзвінків із ключового процесу - бронювання індивідуального дзвінка. У першу чергу перевіримо логіку вибору доступних дат і часу. Для цього потрібно створити індивідуальний план, у якому кожен дзвінок триває 15 хвилин. Консультант додає вільний часовий інтервал з 12:00 до 15:00. Додатково він підключає зовнішній календар, наприклад Google Calendar, де вже запланована подія з 13:00 до 14:00. У підсумку система повинна коректно обробити конфлікт часу та згенерувати 8 доступних для бронювання слотів. Це свідчить про правильну інтеграцію з зовнішніми сервісами та коректну логіку розрахунку доступного часу. Робота цієї функції наочно зображена на рисунку 4.7.

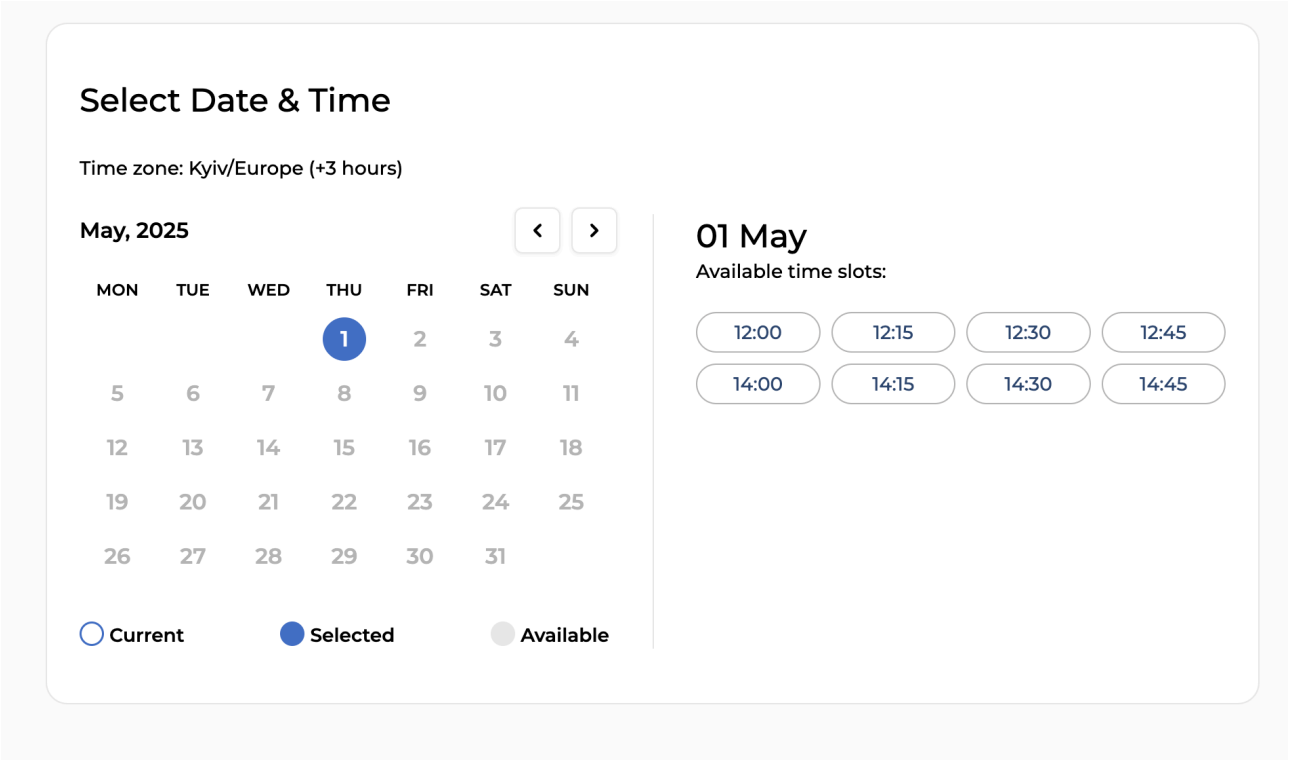

Рисунок 4.7 - Форма вибору вільного часу

Наступним кроком буде одна з найважливіших частин системи - тестування онлайн-оплати. Ми перевіримо коректність обробки платіжних даних через інтегровану платіжну систему Stripe. Якщо користувач вводить правильні реквізити, транзакція проходить успішно, і дзвінок бронюється. Якщо ж використовується невалідна або неіснуюча картка, система коректно

виводить повідомлення про помилку. Це забезпечує прозорість і безпечність платежів. Обробку оплати проілюстровано на рисунку 4.8.

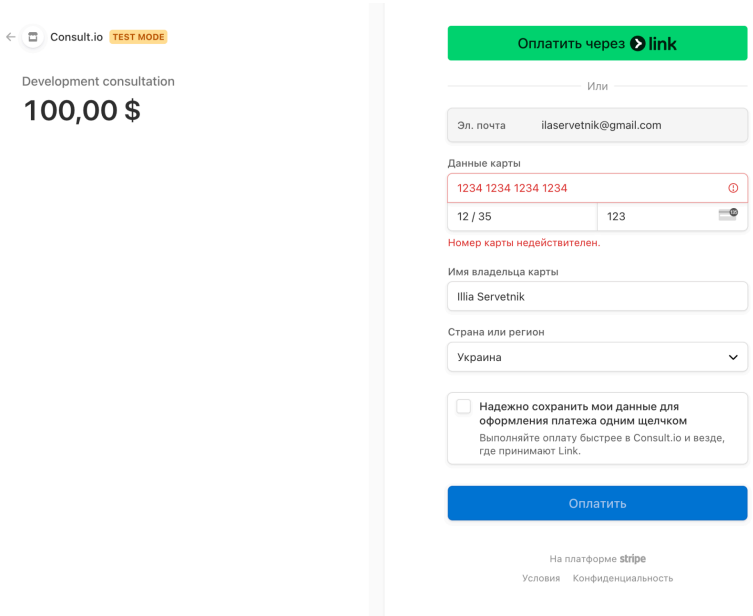

Рисунок 4.8 - Сторінка оплати дзвінка

Після успішного бронювання дзвінка, за умови підключеного зовнішнього календаря, наприклад Google Calendar, у консультанта автоматично створюється подія на відповідну дату та час. Ця подія містить інформацію про дзвінок і зберігається у календарі без необхідності ручного втручання. Це суттєво спрощує організацію графіку консультацій. Автоматично згенерована подія показана на рисунку 4.9.

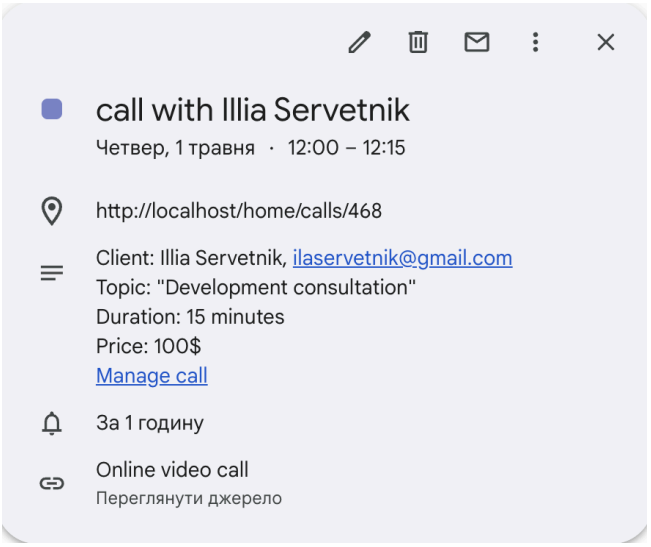

Рисунок 4.9 - Подія дзвінка в Google Calendar

Система також дозволяє перенести вже заброньований дзвінок. Для цього консультант заходить у свій особистий кабінет, обирає потрібний дзвінок, задає нову дату та час, і натискає кнопку для перенесення. Після цього дзвінок оновлюється в системі, а клієнт отримує сповіщення про зміну часу. Це забезпечує гнучкість і зручність для обох сторін. Приклад перенесення відображено на рисунку 4.10.

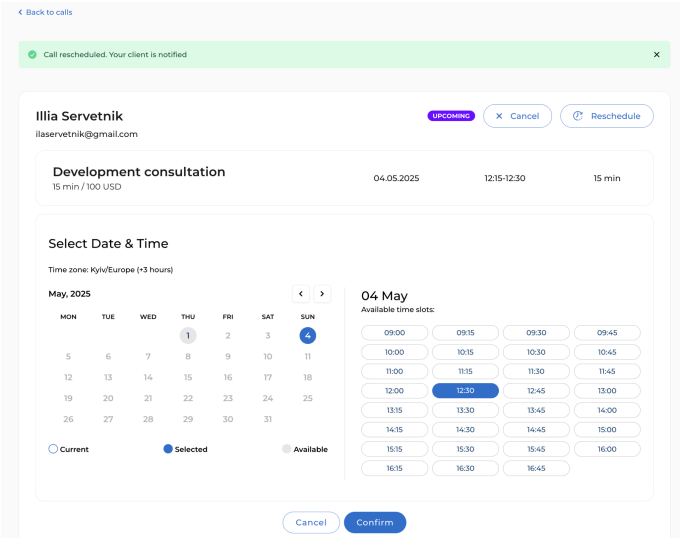

Рисунок 4.10 - Перенесення дзвінка

Окрім перенесення, консультант має можливість скасувати дзвінок. Для цього він також обирає відповідну зустріч і натискає кнопку скасування. У результаті клієнт отримує повідомлення електронною поштою з відповідною інформацією. Крім того, система ініціює процес повернення коштів за скасовану консультацію. Цей процес зображено на рисунку 4.11. Завдяки цим функціям платформа залишається гнучкою, зручною та дружньою до користувачів.

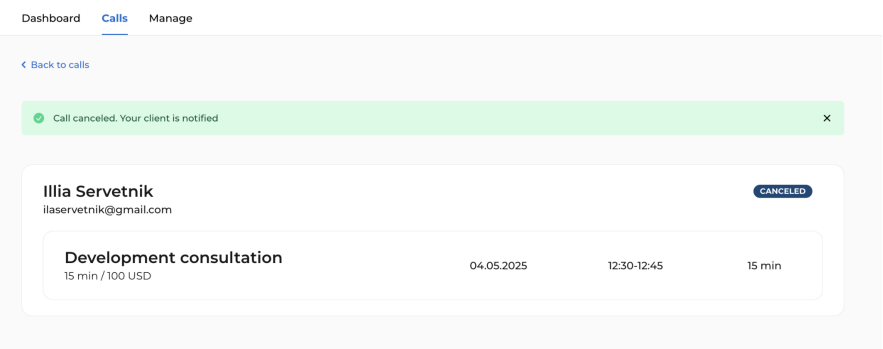

Рисунок 4.11 - Відміна дзвінка

4.3 Тестування модуля відео сесій

Після того як клієнт успішно забронював дзвінок і настав відповідний час, обидві сторони отримують можливість під’єднатися до онлайн-зустрічі. Тестування модуля відеозв’язку розпочинається з перевірки передачі аудіо та відеоданих. Під час активної розмови блок користувача, який говорить, автоматично підсвічується - це дозволяє легко ідентифікувати, хто саме бере участь у розмові в конкретний момент. Така візуальна підказка робить спілкування значно зручнішим, особливо у багатокористувацьких дзвінках, як показано на рисунку 4.12. Якість переданого відео та звуку залишається стабільно високою, з мінімальною затримкою, що критично важливо для комфортної взаємодії в режимі реального часу.

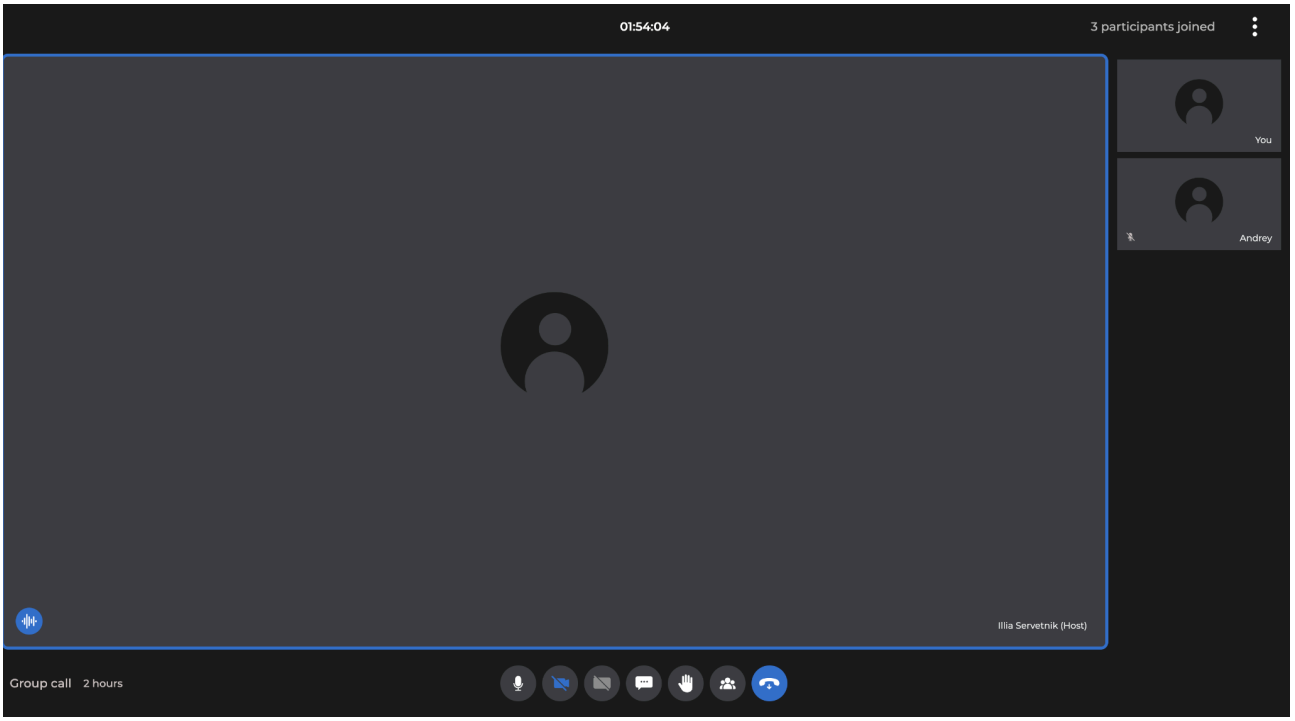

Рисунок 4.12 - Активний користувач в відео сесії

Окремо тестувалася функція демонстрації екрану. Кожен учасник має можливість обрати окреме вікно або повний екран для трансляції. При цьому важливою є не лише чіткість картинки, а й коректна передача звуку. Як продемонстровано на рисунку 4.13, функція працює належним чином. Однак під час тестування було виявлено, що користувачі з повільним

інтернет-з’єднанням можуть стикатись із проблемами при перегляді екрану - зокрема, зниженням якості відеопотоку. Це зумовлено особливостями рендерингу в Zoom Video SDK. Проте якість аудіо при демонстрації залишається стабільно високою, що дозволяє зберігати зміст спілкування навіть за умов низької пропускної здатності мережі.

Рисунок 4.13 - Демонстрація екрану

Для зручного обміну повідомленнями або файлами між учасниками реалізовано текстовий чат. Це особливо зручно, коли в дзвінку бере участь велика кількість користувачів, і не всі можуть одразу висловитись голосом. Через чат можна швидко написати запитання, уточнення або надіслати потрібні документи. На рисунку 4.14 показано приклад використання чату під час зустрічі.

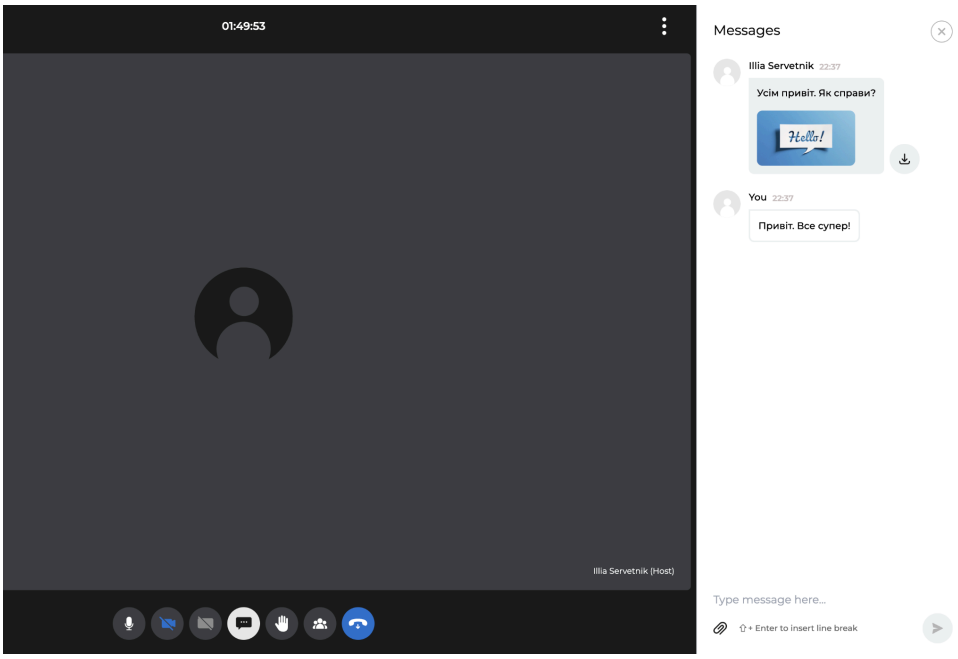

Рисунок 4.14 - Секція чату в дзвінку

Ще однією корисною функцією є можливість підняти руку. Це дозволяє користувачеві привернути увагу без переривання інших. Користувач із піднятою рукою автоматично відображається вище у списку учасників, і його реакція стає помітнішою для консультанта або модератора. Як показано на рисунку 4.15, ця функція підтримує структуру й порядок під час сесій із кількома клієнтами.

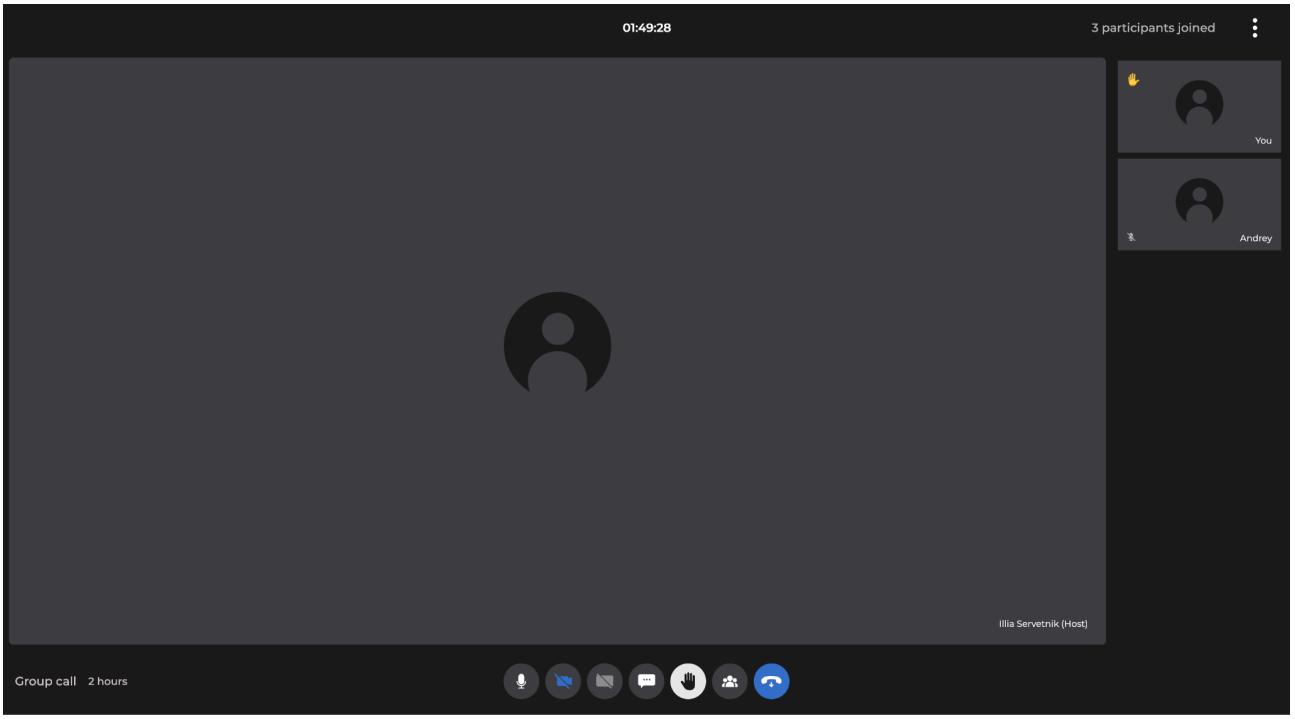

Рисунок 4.15 - Підняття руки

Консультант, як головний учасник дзвінка, має розширені можливості для модерації. Він може переглядати повний список учасників, вимикати мікрофони чи камери окремих клієнтів або всієї групи, припиняти трансляцію екрану, а також скидати всі активні підняті руки. Це дозволяє ефективно керувати порядком у дзвінку та уникати хаосу. Наприклад, після вимкнення мікрофона в інтерфейсі клієнта змінюється іконка в нижній панелі, а передача звуку припиняється. Всі інструменти модерації детально показано на рисунку 4.16.

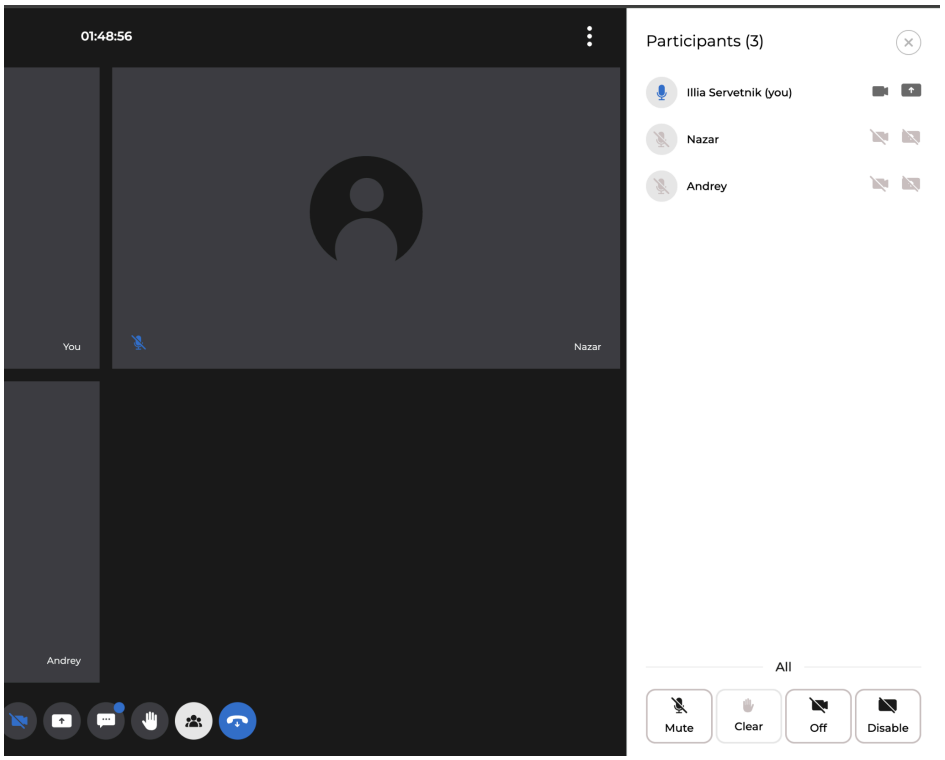

Рисунок 4.16 - Секція з модеруванням клієнтів

4.4 Висновок

Отже, в четвертому розділі було проведено повне тестування всіх основних модулів та функціоналу розробленого WEB-ресурсу. Було перевірено роботу модуля консультанта, модулю організації дзвінків, а також відеосесій - кожен із них пройшов ретельну перевірку на коректність, зручність і стабільність.

У рамках тестування модуля консультанта були протестовані всі етапи аутентифікації: вхід у систему, реєстрація нового користувача, механізм скидання пароля через електронну пошту. В особистому кабінеті консультанта

перевірено функціонал редагування особистих даних, налаштування графіку роботи, інтеграцію сторонніх календарів, а також створення індивідуальних і групових послуг - усе працює згідно з очікуваннями.

У модулі дзвінків було успішно протестовано бронювання консультаційних сесій клієнтами. Особливу увагу приділено рендерингу календаря доступного часу, який динамічно враховує зовнішні події з під'єднаних календарів. Також перевірено інтеграцію з платіжною системою: здійснення транзакцій, обробку помилок оплати, а також функціонал перенесення та скасування дзвінків з боку консультанта з автоматичним сповіщенням клієнтів і поверненням коштів.

Завершальним етапом стало тестування модуля відеосесій. Перевірка показала, що передача аудіо- та відеоданих відбувається стабільно, із мінімальними затримками та високою якістю. Продемонстровано ефективну роботу функції демонстрації екрану, а також можливість спілкування через чат, обмін файлами і використання інтерактивних реакцій, таких як підняття руки. Окремо протестовано інструменти модерації для консультантів, які дозволяють ефективно керувати перебігом зустрічі.

Результати тестування підтверджують, що система працює стабільно, логіка додатку реалізована правильно, а інтерфейс - інтуїтивно зрозумілий і зручний для користувачів. Усі ключові сценарії взаємодії відпрацьовуються коректно. Отже, WEB-ресурс готовий до використання реальними користувачами та подальшого масштабування.

ВИСНОВОК

У результаті виконання бакалаврської кваліфікаційної роботи, було успішно розроблено WEB-ресурс для онлайн консультацій між клієнтами та консультантами. Проєкт реалізує повний цикл взаємодії - від реєстрації користувачів до відеозв’язку в режимі реального часу.

У першому було проведено детальний аналіз предметної області, вивчено наявні аналоги та конкурентні рішення. Це дозволило виявити ключові переваги та недоліки подібних сервісів, а також сформулювати вимоги до функціональності майбутнього ресурсу - як для консультантів, так і для клієнтів.

В другому розділі було проектування системи. Обґрунтовано вибір оптимального набору технологій для створення сучасного та надійного WEB-додатку. Окремо було підібрано супутні бібліотеки й сторонні сервіси для інтеграції, що забезпечило масштабованість та розширюваність ресурсу. З урахуванням вимог до швидкості розробки та стабільності, було обрано монолітну архітектуру, яка найкраще відповідає поставленим завданням.

У третьому було створено всі необхідні модулі - від форм аутентифікації та особистого кабінету консультанта до механізмів створення консультаційних дзвінків, інтеграції з платіжною системою та сторонніми календарями. Особливу увагу було приділено модулю відео сесій, який забезпечує якісний аудіо та відеозв’язок.

У четвертому розділі було проведено комплексне тестування усіх компонентів системи. Було перевірено правильність бізнес-логіки, стабільність роботи сервісу, якість відеозв’язку та надійність інтеграцій із зовнішніми сервісами. Таким чином, поставлена мета бакалаврської кваліфікаційної роботи була повністю досягнута: розроблено повноцінний WEB-ресурс для онлайн консультацій, що відповідає всім заявленим вимогам і є готовим до подальшого впровадження та розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Матеріал LIV науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025) – https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23689/19567 (PDF, 3 с.).
2. Zoom Video SDK URL: https://developers.zoom.us/docs/video-sdk/web/ (дата звернення 22.05.2025).
3. PHP URL: https://www.php.net/ (дата звернення 22.05.2025).
4. Laravel URL: https://laravel.com/docs/master (дата звернення 22.05.2025).
5. Документація Eloquent ORM URL: https://laravel.com/docs/master/eloquent (дата звернення 22.05.2025).
6. Symfony URL: https://symfony.com/doc/current/index.html (дата звернення 22.05.2025).
7. Документація Doctrine Object Relational Mapper URL: https://www.doctrine-project.org/projects/doctrine-orm/en/3.3/index.html (дата звернення 22.05.2025).
8. MySql URL: https://dev.mysql.com/doc/ (дата звернення 22.05.2025).
9. NoSql URL: https://www.oracle.com/ua/database/nosql/what-is-nosql/ (дата звернення 22.05.2025).
10. SPA URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення 22.05.2025).
11. Livewire URL: https://livewire.laravel.com/docs/quickstart (дата звернення 22.05.2025).
12. Alpine.js URL: https://alpinejs.dev/ (дата звернення 22.05.2025).
13. Стаття про монолітну архітектуру в розробці додатків URL: https://www.vpnunlimited.com/ua/help/cybersecurity/monolithic-architecture (дата звернення 22.05.2025).

14. Стаття про мікросервісну архітектуру в розробці додатків URL: https://wezom.com.ua/ua/blog/scho-take-mikroservisna-arhitektura-znachennya-skladovi-perevagi (дата звернення 22.05.2025).
15. Стаття про сервісно орієнтовану архітектуру в розробці додатків URL: https://robotdreams.cc/uk/blog/520-vse-pro-servisno-oriyentovanu-arhitekturu (дата звернення 22.05.2025).
16. DDD - https://redis.io/glossary/domain-driven-design-ddd/ (дата звернення 22.05.2025).
17. GRASP принципи: об’єктно орієнтовані дизайн патерни URL: https://patrickkarsh.medium.com/object-oriented-design-with-grasp-principles-8049fa63e52 (дата звернення 22.05.2025).
18. OAuth2 URL: https://auth0.com/intro-to-iam/what-is-oauth-2 (дата звернення 22.05.2025).
19. Stripe URL: https://docs.stripe.com/api (дата звернення 22.05.2025).
20. Sanctum URL: https://laravel.com/docs/master/sanctum (дата звернення 22.05.2025).
21. Стаття про принцип розробки додатків DRY URL: https://dev.to/ralphcone/please-do-repeat-yourself-dry-is-dead-1jbg (дата звернення 22.05.2025).
22. Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп'ютерні науки» [Електронний ресурс]. - https://iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=46522&id=124285 (PDF, 55 с.) .

ДОДАТКИ

ДОДАТОК А. ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-ресурсу для онлайн консультацій

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,26 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Белзецький Р.С.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Серветник І.В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б. ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```php
<?php

namespace App\Services\Calls;

use App\Enums\Calls\CallStatus;
use App\Enums\UserType;
use App\Events\Calls\CallStartedEvent;
use App\Models\Calls\Call;
use App\Models\Calls\CallParticipant;
use App\Services\Payments\StripeService;
use Carbon\Carbon;
use Illuminate\Support\Str;

class CallService
{
    public function cancel(Call $call, UserType $userType = null): Call
    {
        return tap($call)->update([
            'status' => CallStatus::CANCELED,
            'cancel_user_type' => $userType,
        ]);
    }

    public function cancelWithRefund(Call $call, UserType $userType = null): Call
    {
        return tap($call)->update([
            'status' => CallStatus::CANCELED,
            'refund_money' => true,
            'cancel_user_type' => $userType,
        ]);
    }

    public function complete(Call $call): Call
    {
        return tap($call)->update(['status' => CallStatus::COMPLETED]);
    }

    public function reschedule(Call $call, Carbon $date, Carbon $from, Carbon $till): Call
    {
        $from = $date->clone()->setTime($from->format('H'), $from->format('i'));
        $till = $date->clone()->setTime($till->format('H'), $till->format('i'));

        return tap($call)->update([
            'date' => $from->utc()->toDateString(),
            'from' => $from->utc()->toTimeString(),
            'till' => $till->utc()->toTimeString(),
        ]);
    }
```

```
    public function start(Call $call): Call
    {
        if ($call->started_at) {
            return $call;
        }

        $call->update(['started_at' => now()->toDateTimeString()]);
        event(new CallStartedEvent($call));

        return $call;
    }

    public function getRefundCase(Call $call, CallParticipant $participant): ?string
    {
        if (($call->status !== CallStatus::CANCELED) || !$call->refund_money) {
            return null;
        }

        if ($call->cancel_user_type == UserType::HOST) {
            return 'Host canceled the call';
        }

        if ($call->cancel_user_type == UserType::PARTICIPANT) {
            return 'Client canceled the call';
        }

        if (!app(CallAttendanceService::class)->hostAttendedRequiredTimes($call)) {
            return app(CallAttendanceService::class)->participantAttended($call, $participant)
                ? "Host didn't join the call"
                : "Nobody joined the call";
        }

        return null;
    }

    public function getAvailableWithdrawAt(Call $call, CallParticipant $participant = null): Carbon
    {
        $minimalAvailableWithdrawAt =
Carbon::create($call->from)->addDays(config('services.payouts.freezing_days_count'));

        if (!$participant) {
            return $minimalAvailableWithdrawAt;
        }

        $balanceTransaction =
app(StripeService::class)->getBalanceTransactionFromParticipant($participant);
        if (!$balanceTransaction) {
            return $minimalAvailableWithdrawAt;
        }

        $balanceTransactionAvailableOn =
Carbon::createFromTimestamp($balanceTransaction->available_on);
```

```
        return $balanceTransactionAvailableOn->gte($minimalAvailableWithdrawAt) ?
$balanceTransactionAvailableOn : $minimalAvailableWithdrawAt;
    }

    public function getUniqueUuid(): string
    {
        $uuid = Str::uuid();

        if (Call::where('uuid', $uuid)->exists()) {
            return $this->getUniqueUuid();
        }

        return $uuid;
    }
}
```

```
<?php

namespace App\Services;

use App\Events\Users\UserDataChangedEvent;
use App\Events\Users\UserOnboardCompletedEvent;
use App\Models\User;
use App\Services\Payments\StripeService;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UserService
{
    public function create(string $name, string $email, string $password, string $socialiteDriver =
null): User
    {
        $user = User::create([
            'email' => $email,
            'name' => $name,
            'password' => Hash::make($password),
            'socialite_driver' => $socialiteDriver,
            'email_verified_at' => $socialiteDriver ? now() : null,
        ]);

        return $this->updateTimezone($user);
    }

    public function getUniqueUuid(): string
    {
        $uuid = Str::uuid();

        if (User::where('uuid', $uuid)->exists()) {
            return $this->getUniqueUuid();
        }
```

```
        return $uuid;
    }

    public function updateTimezone(User $user): User
    {
        $timezone = (app(TimezoneService::class)->getCurrentTimezone() ??
app(TimezoneService::class)->getDefault());

        return tap($user)->update(['timezone_id' => $timezone->id]);
    }

    public function completeStripe(User $user): bool
    {
        $account = app(StripeService::class)->retrieveAccount($user->stripe_account_id);

        if ($account && $account->charges_enabled) {
            $user->update(['validated_by_stripe' => true]);

            event(new UserOnboardCompletedEvent(auth()->user()));
            return true;
        }

        return false;
    }

    public function completeOnboarding(User $user): void
    {
        $user->update(['onboarding_completed' => true]);
        app(UserService::class)->updateTimezone($user);

        event(new UserDataChangedEvent($user));
    }
}
```

ДОДАТОК В. ІНСТРУКЦІЯ КОРИСТУВАЧА

Для того, щоб запустити проєкт, потрібно перейти до папки проєкту в терміналі, де він розташований та запустити команду *cp .env.example .env*. Таким чином буде створено файл конфігурації, який потрібно заповнити.

Далі потрібно виконати команду *docker compose up -d*. Якщо Docker Compose не встановлений на комп'ютері, його слід встановити з офіційних джерел. У результаті буде створено docker-контейнери, які можна переглянути в інтерфейсі. Список створених контейнерів зображено на рисунку В.1.

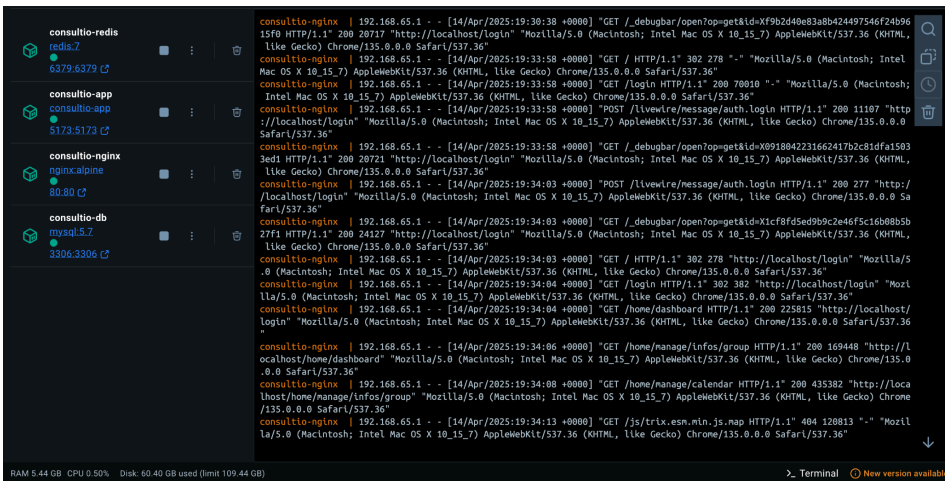

Рисунок В.1 - Список створених docker-контейнерів

Після створення контейнерів потрібно перейти в головний контейнер, ввівши команду *docker exec -it consultio-app /bin/bash*. Далі слід ввести команди, які завантажать усі необхідні бібліотеки та зберуть фронтенд-частину. Список усіх команд наведено у файлі README.md.

За потреби, для пришвидшеного тестування, можна згенерувати тестові дані. Для цього потрібно виконати команду *php artisan db:seed --class=MockDatabaseSeeder*.

Інструкція використання:

Після переходу на головну сторінку відображається сторінка автентифікації. Щоб створити нового консультанта, потрібно перейти на сторінку реєстрації, натиснувши посилання «Sign Up». Форму реєстрації зображено на рисунку В.2.

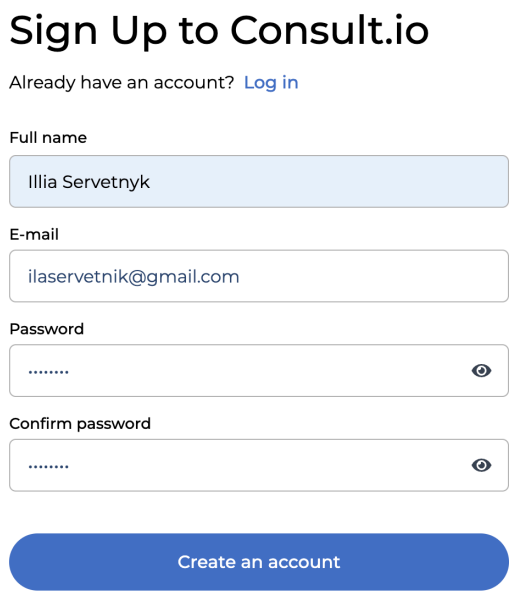

Рисунок В.2 – Форма реєстрації консультанта

Після створення облікового запису та верифікації пошти потрібно заповнити інформацію про консультанта та завершити процес онбордингу. На рисунку В.3 зображено сторінку для заповнення інформації консультанта.

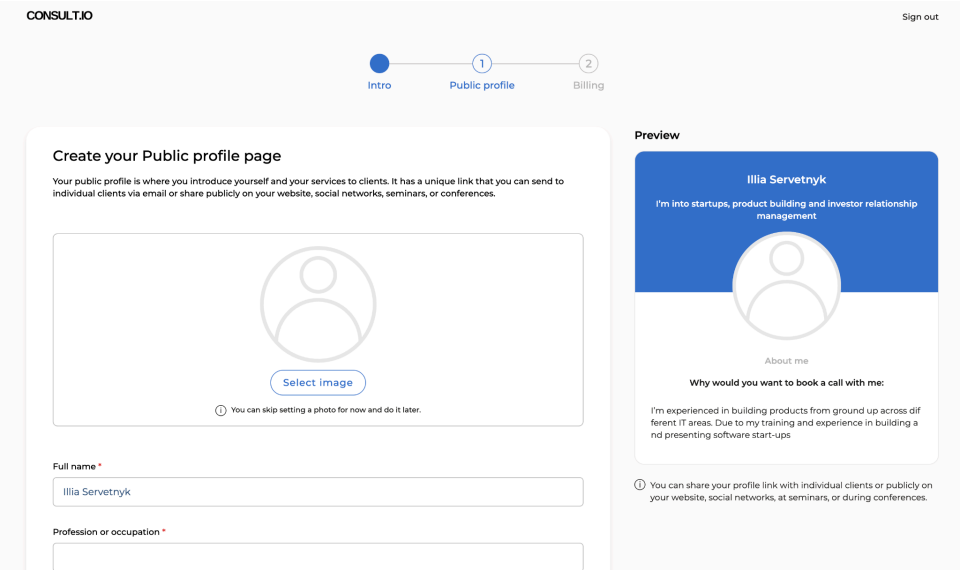

Рисунок В.3 – Форма публічної інформації консультанта

Після проходження процесу реєстрації варто заповнити інформацію про доступність, перейшовши на сторінку «My availability». Календар з розкладом зображено на рисунку В.4.

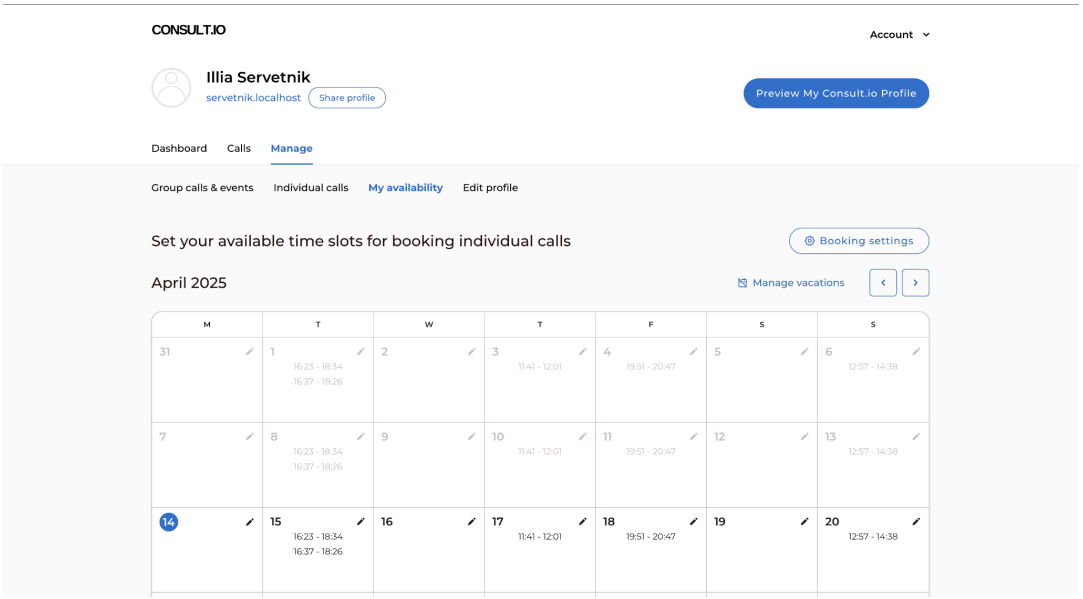

Рисунок В.4 – Календар доступності консультанта

Потім варто під'єднати онлайн календарі. Це дозволить синхронізувати дзвінки в додатку та події в календарі. Менеджмент календарів зображений на рисунку В.5.

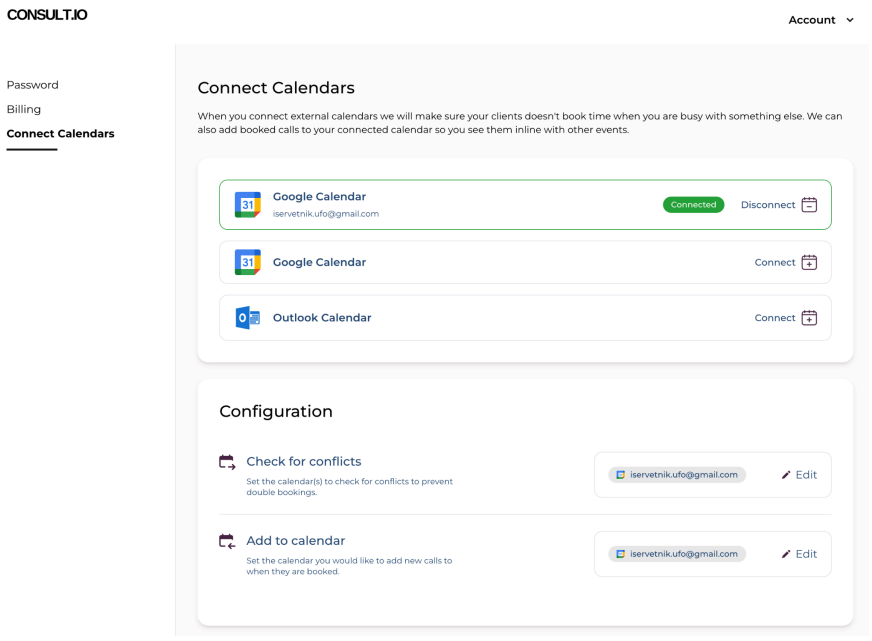

Рисунок В.5 - Менеджмент календарів консультанта

Щоб створити індивідуальний дзвінок, який придбає клієнт, потрібно перейти на сторінку «Individual calls». На цій сторінці відображатимуться всі 1-на-1 послуги, які надає консультант. Потім потрібно натиснути кнопку «Create new» та заповнити інформацію щодо дзвінка. Форму створення дзвінка зображено на рисунку В.6.

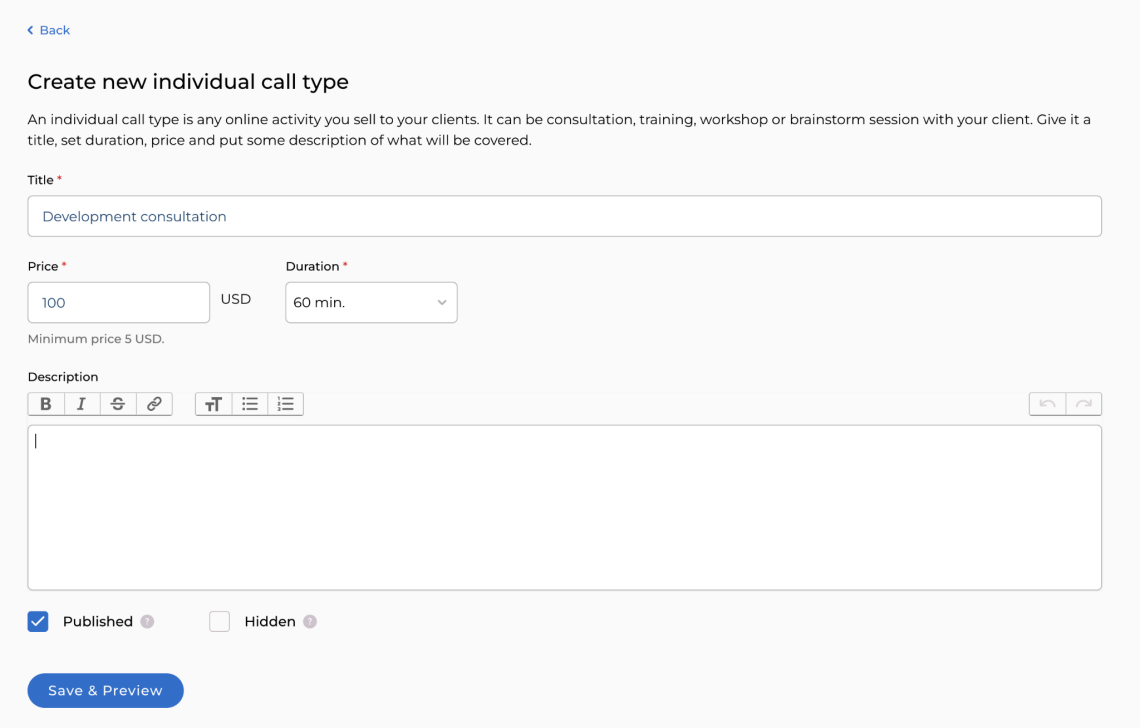

Рисунок В.6 - Форма створення дзвінка

Створивши перший дзвінок, консультант має поширити посилання на свій профіль, щоб клієнти мали змогу забронювати сесію. Публічний профіль консультанта зображено на рисунку В.7.

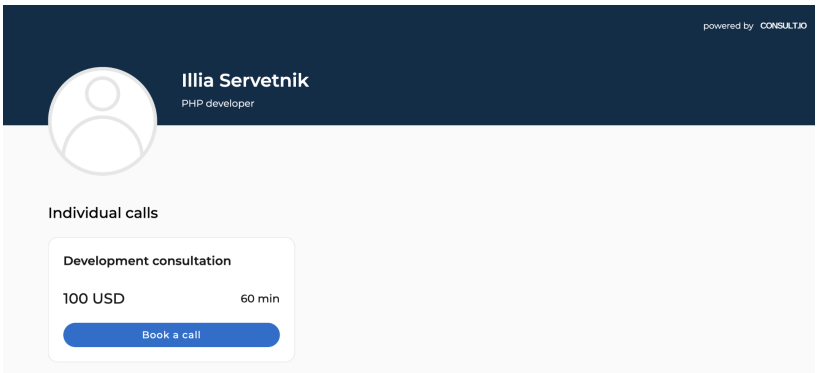

Рисунок В.7 - Публічна сторінка консультанта

Обравши послугу до вподоби, клієнт має вибрати потрібні дату та час, а також заповнити коротку інформацію про себе. Форму бронювання дзвінка зображено на рисунку В.8.

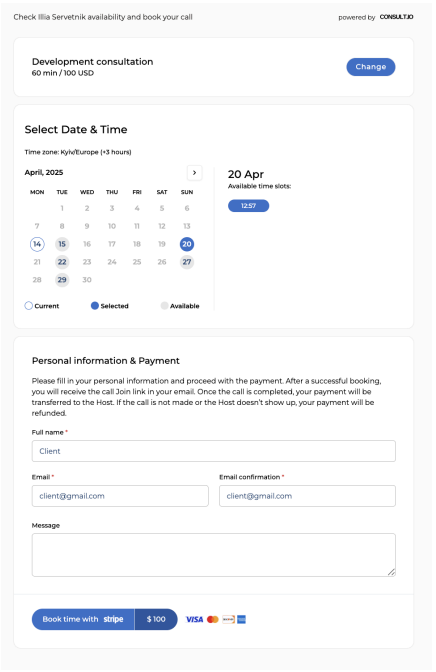

Рисунок В.8 - Сторінка бронювання дзвінка

Після цього клієнт автоматично переходить на сторінку оплати Stripe, де має ввести реквізити банківської картки. Сторінку оплати зображено на рисунку В.9.

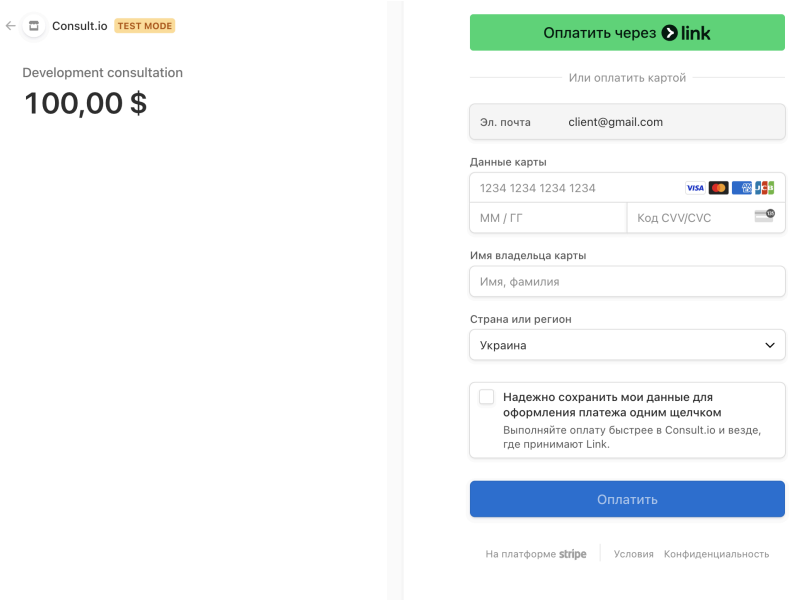

Рисунок В.9 - Сторінка оплати консультації

Коли клієнт оплачує дзвінок, у консультанта в особистому кабінеті створюється відповідний запис. Список майбутніх дзвінків, а також дзвінків, що вже відбулись або були скасовані, можна знайти на сторінці «Calls». За потреби консультант може перенести або скасувати дзвінки. Список усіх дзвінків зображено на рисунку В.10.

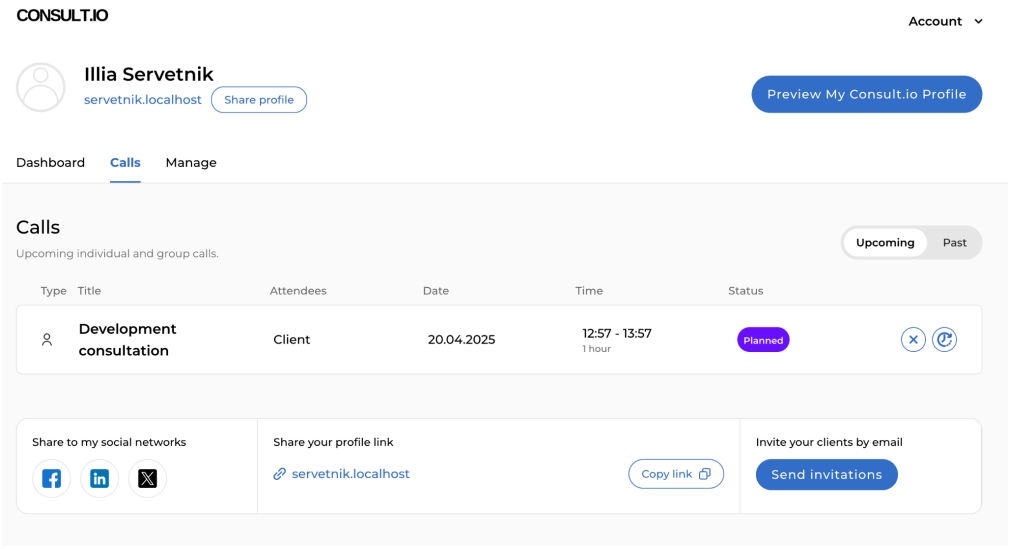

Рисунок В.10 - Список всіх дзвінків консультанта

Дочекавшись початку дзвінка, консультант і клієнт підключаються до сесії. Вони мають можливість змінювати пристрої введення та виведення, показувати екран і транслювати відео з вебкамери. Онлайн-дзвінок зображено на рисунку В.11.

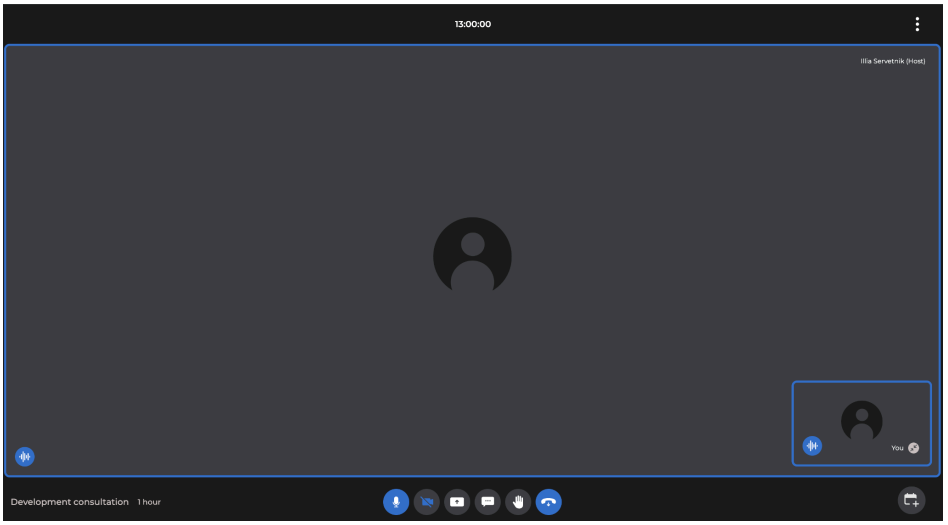

Рисунок В.11 - Онлайн дзвінок консультанта та клієнта

ДОДАТОК Г. ГРАФІЧНА ЧАСТИНА

РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ОНЛАЙН КОНСУЛЬТАЦІЙ.

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Серветник Ілля Вячеславович
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Белзецький Р.С.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

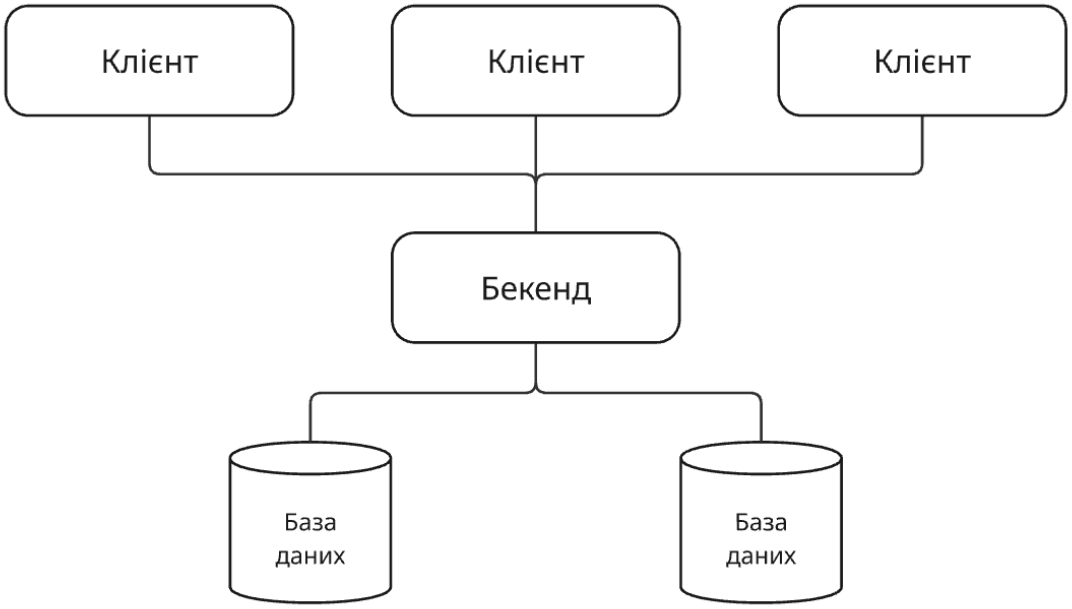

Рисунок Г.1 − Схема монолітної архітектури

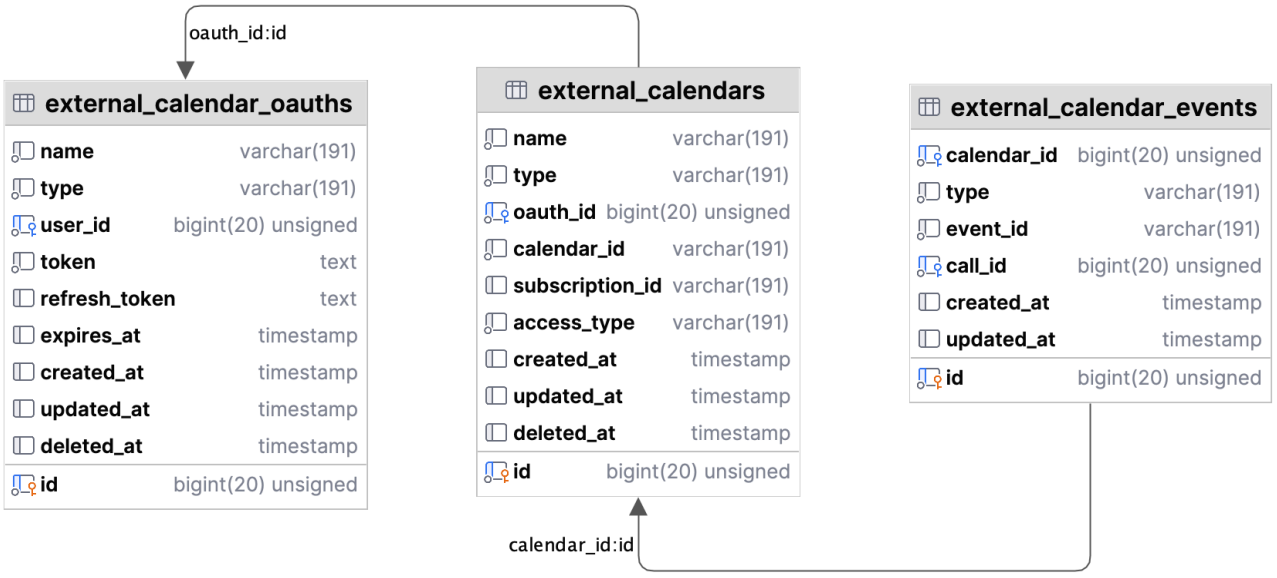

Рисунок Г.2 − Діаграма таблиць зовнішніх календарів

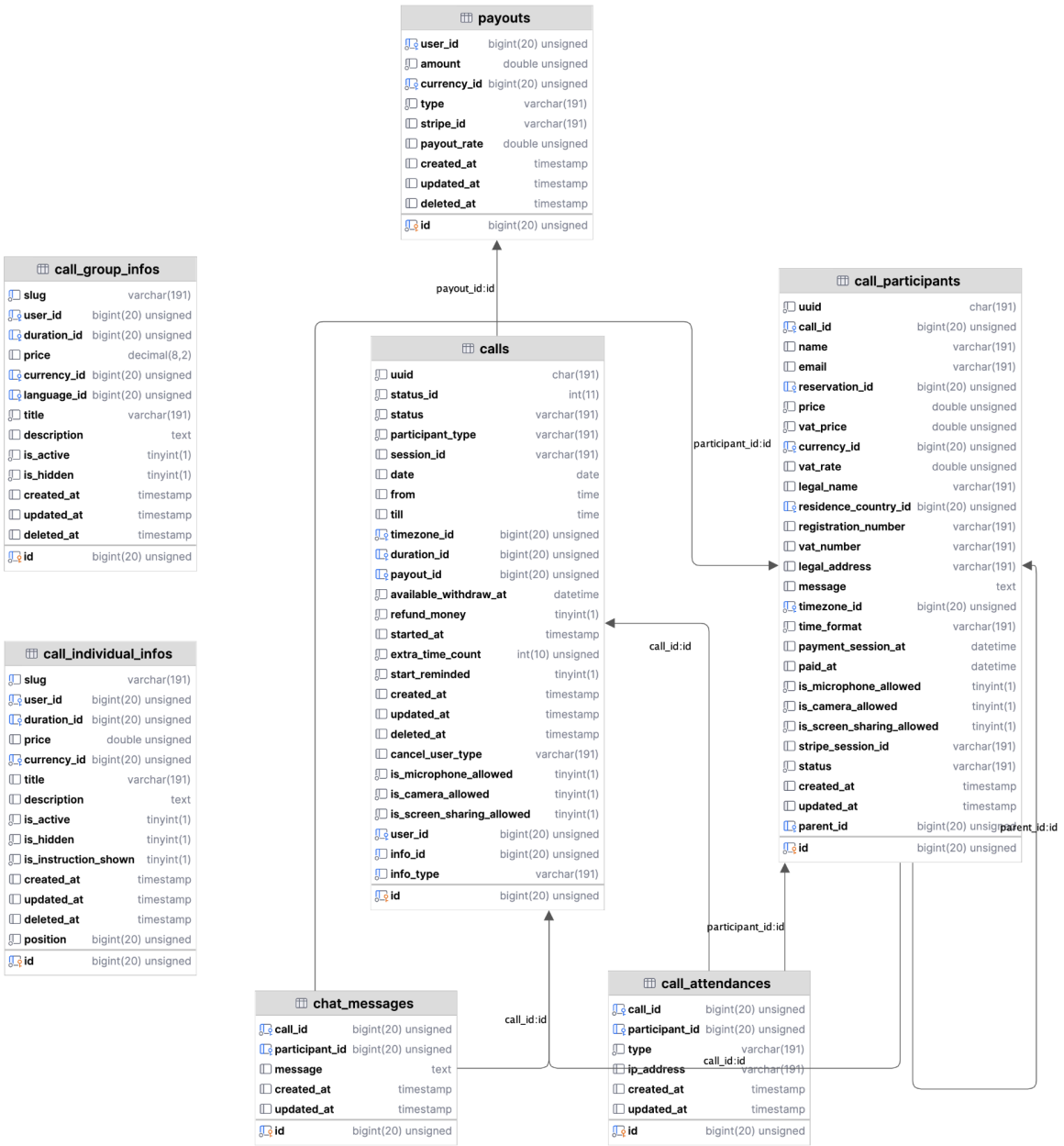

Рисунок Г.3 – Діаграма таблиць дзвінків

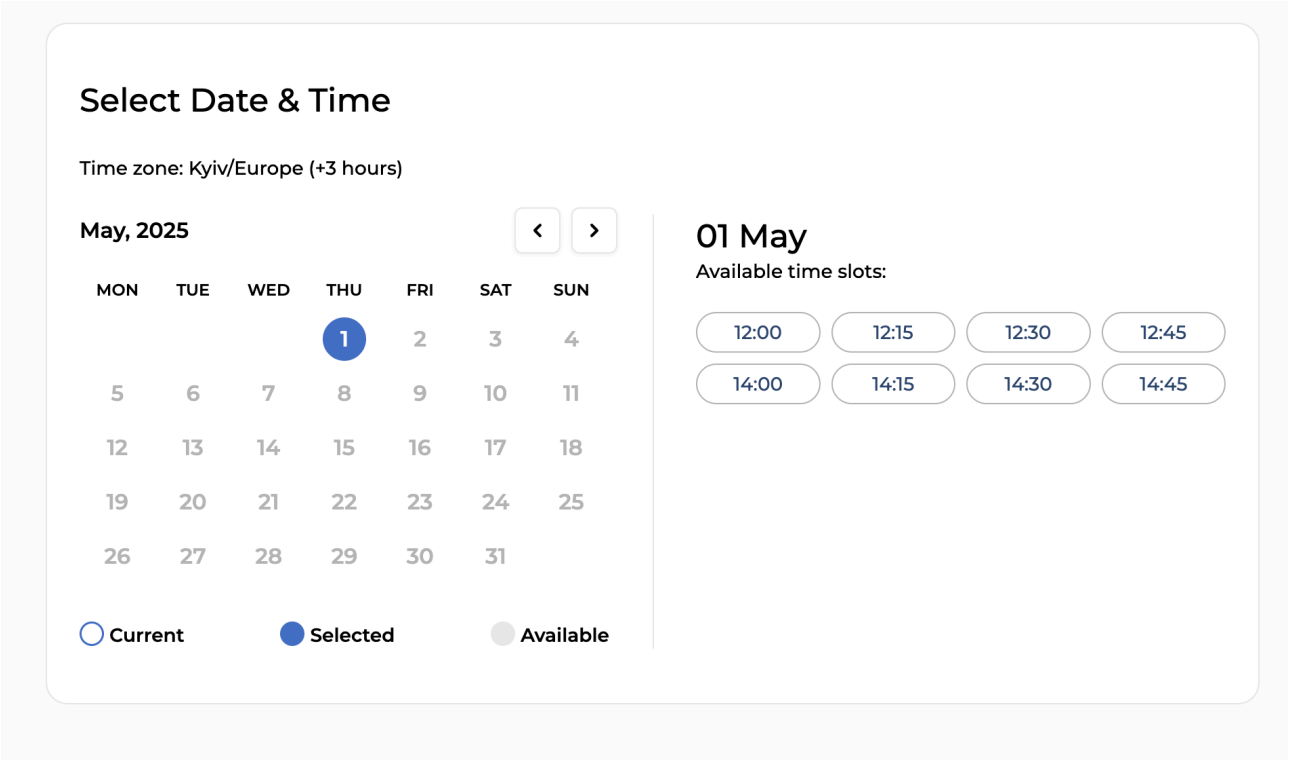

Рисунок Г.4 – Форма вибору вільного часу

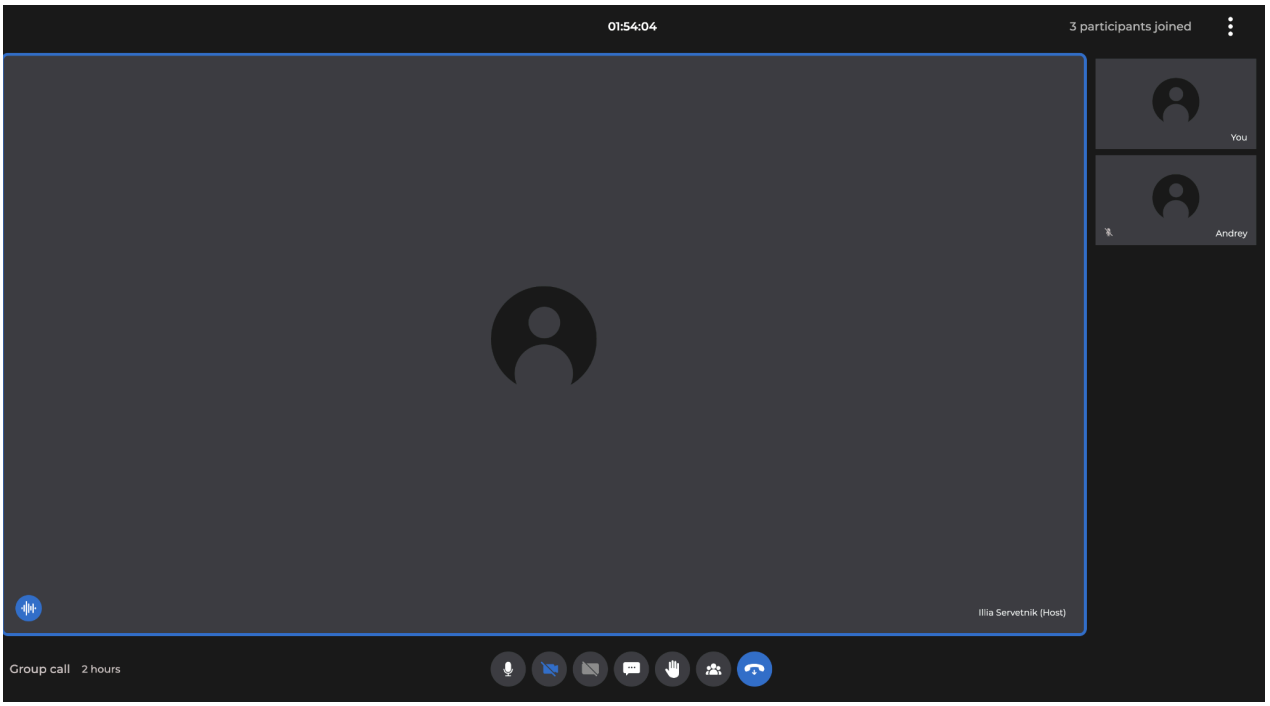

Рисунок Г.5 – Активний користувач в відео сесії