Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка системи надання платних відеоконсультацій

Текстова частина до курсової роботи за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи
Калітовський Б.В
_____
(підпис)
"____" _____ 2021 р.

Виконав студент
Возбранний Р.С.
"____" _____ 2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доктор техн. наук, декан ФІ
_____А. М. Глибовець
(підпис)
“____”_____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Возбранному Роману
факультету інформатики 3 курсу бакалаврської програми

ТЕМА: Розробка системи надання платних відеоконсультацій

Вихідні дані:

клієнт-серверний застосунок для надання платних відеоконсультацій.

Зміст ТЧ до курсової роботи:

Вступ

Анотація

1. Аналіз існуючих систем та сервісів

2. Огляд використаних технологій та інструментів

3. Програмна реалізація системи платних консультацій

Висновки

Список використаної літератури та електронних ресурсів

Додатки

Дата видачі “_____” _____201_ р.

Керівник _____ Завдання отримано _____

Тема: Розробка системи надання платних відеоконсультацій

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	03.10.2020	
2.	Вивчення предметної області	15.10.2020	
3.	Огляд існуючих рішень	01.11.2020	
4	Огляд засобів реалізації	18.11.2020	
5.	Вивчення технологій для розробки	30.12.2020	
6.	Побудова технічного завдання	08.01.2021	
7.	Написання першої частини курсової роботи	20.01.2021	
8.	Написання другої частини курсової роботи	01.02.2021	
9.	Написання третьої частини курсової роботи	15.02.2021	
10.	Написання висновків курсової роботи	23.02.2021	
11.	Перегляд змісту роботи з керівником	01.03.2021	
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	25.03.2021	
13.	Створення презентації	01.04.2021	
14.	Захист роботи		

Студент Возбранний Р.С.
Керівник Калітовський Б.В

“____” _____

Зміст

Анотація

В роботі проаналізовано наявні на ринку системи для проведення платних онлайн-консультацій. Також, описуються особливості інструментів і сервісів для інтеграції онлайн-оплати та відеодзвінків. В результаті, розроблено систему для проведення платних онлайн-консультацій.

Ключові слова:

Розробка, система, відеодзвінки, онлайн-оплата, консультації.

Вступ

Розвиток технологій та епідемічна ситуації у світі сприяє перенесенню фізичних бізнес-процесів на онлайн-платформи. Кожного дня зростає попит та росте довіра людей до онлайн послуг. Системи для платних онлайн-консультацій можуть бути корисними широкому колу людей. Вони дозволяють спростити та пришвидшити процес пошуку потрібного спеціаліста, користувачі можуть заощадити свій час, не витрачаючи його на дорогу та очікування в черзі.

Онлайн-консультації дозволяють скоротити витрати на офіс для консультантів. Все що потрібно для проведення консультації - це комп'ютер або телефон та доступ до мережі. Це дозволяє знизити вартість консультації та розширити коло користувачів.

В роботі проводиться аналіз наявних на ринку систем для проведення онлайн-консультацій. Також розглядаються популярні інструменти для інтеграції онлайн-оплати та відеодзвінків у власному застосунку. В результаті роботи планується реалізувати систему для проведення платних відеоконсультацій, яка може слугувати основою для створення нових застосунків у сфері онлайн-консультацій.

Об’єкт дослідження: інструменти та сервіси для створення систем платних онлайн-консультацій.

Мета дослідження: порівняти підходи до створення систем платних онлайн-консультацій та створити власну систему.

Робота має три розділи:

1) В першому розділі проводиться огляд та аналіз існуючих на ринку застосунків для проведення онлайн-консультацій, а також огляд популярних інструментів та сервісів для інтеграції онлайн-оплати та відеодзвінків.
2) В другому розділі описуються технології, які в надалі будуть використовуватись для реалізації власної системи.

3) В третьому розділі формулюються вимоги та детально розглядаються важливі моменти реалізації власної системи.

Розділ 1. Аналіз існуючих систем

1.1 Опис функцій існуючих систем

Для кращого розуміння процесу надавання та отримання онлайн-консультацій, було вирішено проаналізувати наявні на ринку системи. Це допоможе визначити, які інструменти та сервіси для розробки активно використовуються в реальних великих проектах. Для аналізу було обрано застосунки з різних сфер, для визначення потреб користувачів та вимог до системи в залежності від теми проведення консультації.

1.1.1 Система Doctor Online

Doctor Online – онлайн сервіс моніторингу здоров’я та консультацій з лікарями. Система представлена у вигляді мобільного додатку. [1]

В розділі консультацій користувач має можливість:

- обрати спеціаліста відповідно до запиту та симптомів;
- прикріпити до звернення файл або фото;
- отримати попередній висновок лікаря та рекомендації на лікування, прийом препаратів, аналізи;
- оплатити послугу онлайн;
- перегляд історії чату;

Система пропонує декілька варіантів отримання консультацій:

- консультація телефоном;
- відеодзвінок;
- консультація в чаті.

Варіанти оплати:

- картою (використовуючи сервіс Fondy) (рисунок 1.1.1.1);
- з мобільного рахунку;
- за рахунок промокоду;
- оформлення підписки.

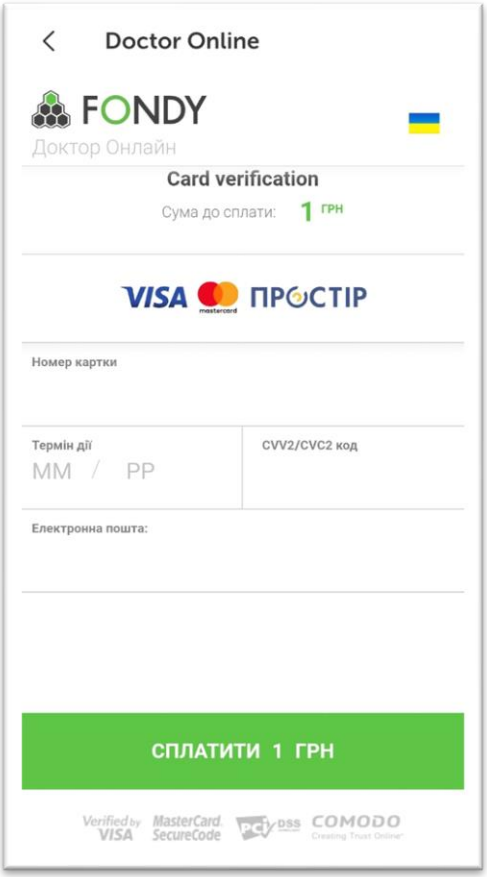


Рис. 1.1.1.1. Оплата картою в Doctor Online

1.1.2 Платформа Юристи.UA

Юристи.UA – це платформа, призначена для отримання швидкої юридичної допомоги. Система передбачає отримання допомоги лише у форматі консультації в чаті. Оплата реалізована за допомогою сервісу Interkassa (рисунок 1.1.2.1). [2]

В розділі консультацій користувач має можливість:

- знайти юриста;
- описати питання;
- прикріпити до звернення файл або фото;
- обрати тип консультації;
- оцінити вартість питання;
- здійснити оплату.

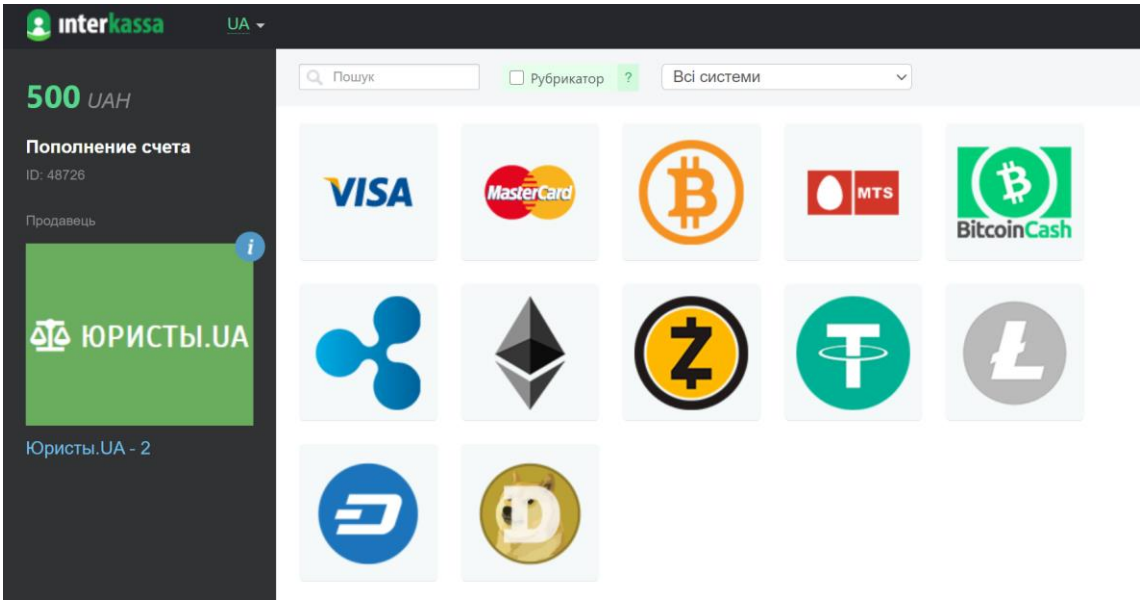


Рис. 1.1.2.1. Оплата реалізована за допомогою сервісу Interkassa

1.1.3 Платформа QITonline

QITonline – платформа, яка допомагає психотерапевтам здійснювати лікування онлайн. Одною з функцій системи є створення сесій для онлайн-контакту терапевта з клієнтом.

Консультант може надсилати тести користувачу для подальшого автоматичного аналізу та панувати онлайн зустрічі. Відеодзвінки реалізовані за допомогою системи Jitsi (рисунок 1.1.3.1). [3]

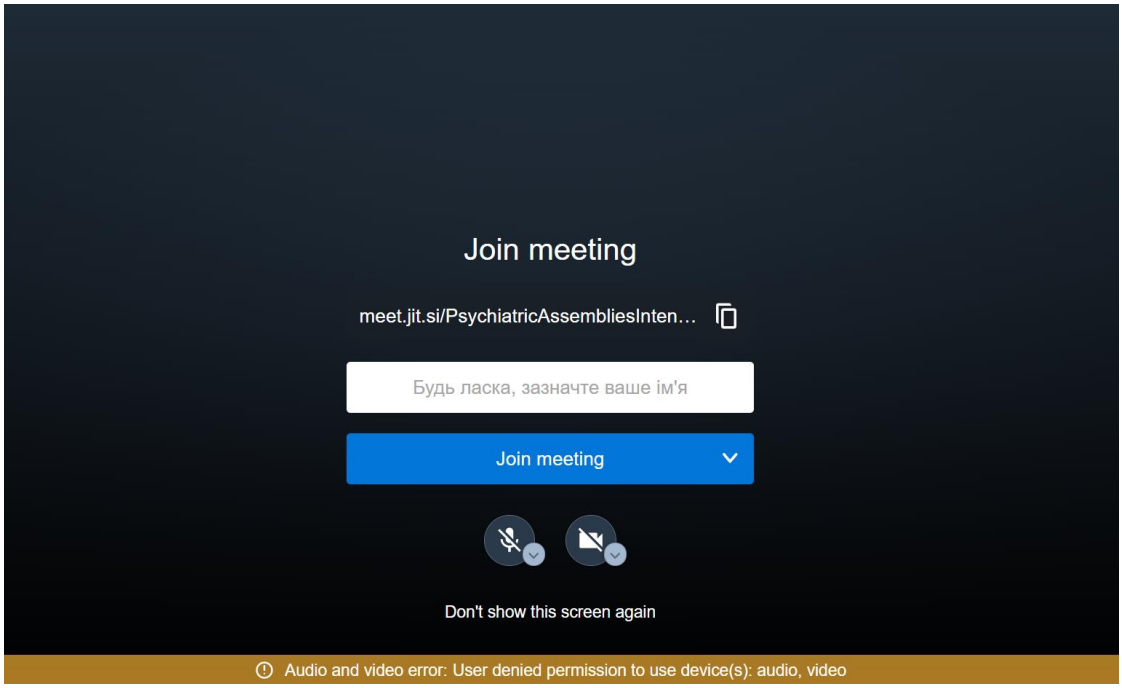


Рис. 1.1.3.1. Інтерфейс дзвінка системи Jitsi

1.2 Аналіз функцій існуючих систем

В результаті аналізу функцій описаних систем було визначено, що більшість з них використовує готові рішення для реалізації відеоконсультацій та здійснення оплати.

Зважаючи на попередній аналіз, було вирішено:

- порівняти процес інтеграції готових сервісів для реалізації відеодзвінків з написанням власної імплементації;
- порівняти процес інтеграції сервісів для реалізації онлайн-оплати.

1.2 Опис функцій сервісів для інтеграції відеодзвінків

На ринку існує багато сервісів, що дозволяють інтегрувати відеодзвінки у власний застосунок. Під час аналізу, буде розглянуто основні функції систем, вартість, зручність інтеграції та загальну архітектуру організації відеодзвінків.

1.3.1 Сервіс Twilio

Twilio – сервіс для розробників, який дозволяє легко інтегрувати різні методи комунікації, використовуючи SMS, WhatsApp, Voice, Video, email та IoT. Twilio дозволяє створювати відеододатки в режимі реального часу та пропонує безкоштовну послугу Video WebRTC Go Rooms для створення 1:1 чатів.

Послуга Video WebRTC Go Rooms

Послугу можна використовувати для індивідуальних відеодзвінків. Хвилини для учасників – безкоштовні, включено 25 ГБ використання TURN сервера на місяць. Go Rooms використовують peer-to-peer топологію. Одночасно на обліковий запис може бути не більше 100 одночасних учасників, наприклад, 50 кімнат з двома учасниками.

Послуга Video P2P Rooms

У P2P Room учасники обмінюються медіа напряму. Медіа шифруються наскрізним способом (E2E) за допомогою протоколів безпеки WebRTC. Twilio не є посередником в обміні даними. Єдиним винятком є ситуація, коли обмін вимагає TURN. В цьому випадку TURN сервер передаватиме зашифровані дані для забезпечення підключення (рисунок 1.3.1.1).

Ціна за використання Twilio "Programmable video" починається з $0.004/хв за користувача для малих групових дзвінків (до 4 учасників). Ціна за послугу "Peer-to-peer Room" стартує від $0.0015/хв за одного користувача. [4]

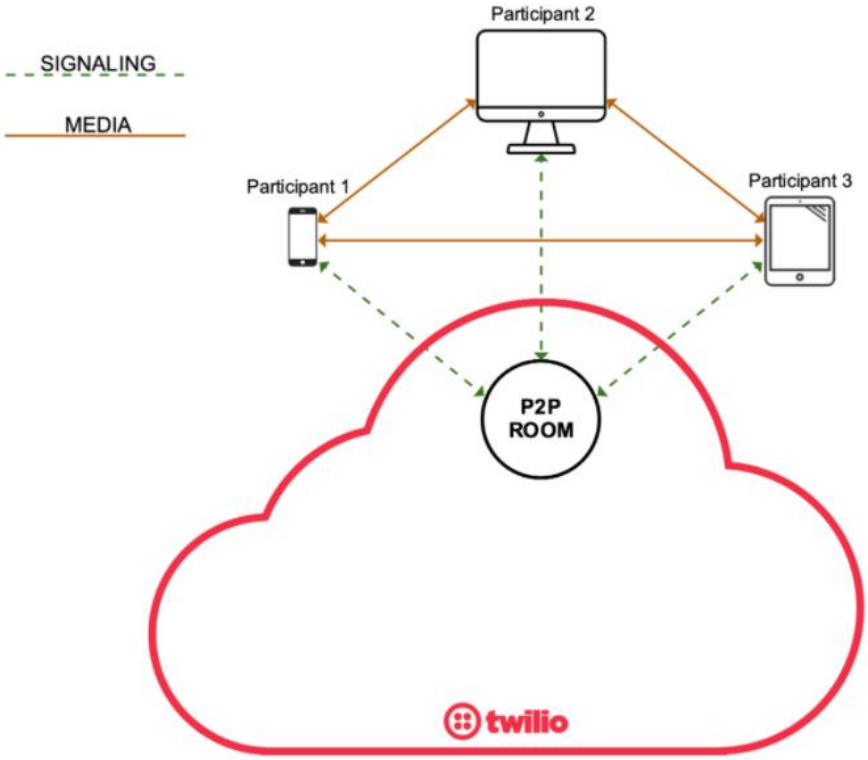


Рис. 1.3.1.1. Архітектура Twilio P2P Room

1.3.2 Сервіс Vonage Video API

Vonage Video API дозволяє спростити інтеграцію систем відеодзвінків, обміну повідомленнями, спільного доступу до екрана та інших. Платформа включає клієнтські бібліотеки для вебу, iOS, Android, Windows, та Linux. Також наявні серверні SDK і REST API. Vonage Video API використовує WebRTC для аудіо та відео зв'язку. Вартість використання Video API починається з € 0.0040 за хвилину. [5]

Основні функції системи:

- Підтримка one-to-one, групових відеочатів та великих трансляцій.

- Можливість ділитися екраном.
- Взаємодія WebRTC та телефонії з Voice API.
- Обмін повідомленнями та файлами між користувачами сесії.

Усі програми, побудовані на платформі Vonage Video API, потребують двох основних компонентів:

- Клієнт – клієнтський код, який працює у браузері або мобільному додатку, налаштований розробником за допомогою клієнтських бібліотек. Клієнтська сторона обробляє більшість функцій Vonage Video API.
- Сервер – серверний код, що виконується на веб-сервері, встановленому розробником за допомогою серверних SDK-файлів (доступні для Node, PHP, Java, .NET, Python та Ruby).

Функції клієнта та сервера (рисунок 1.3.2.1).

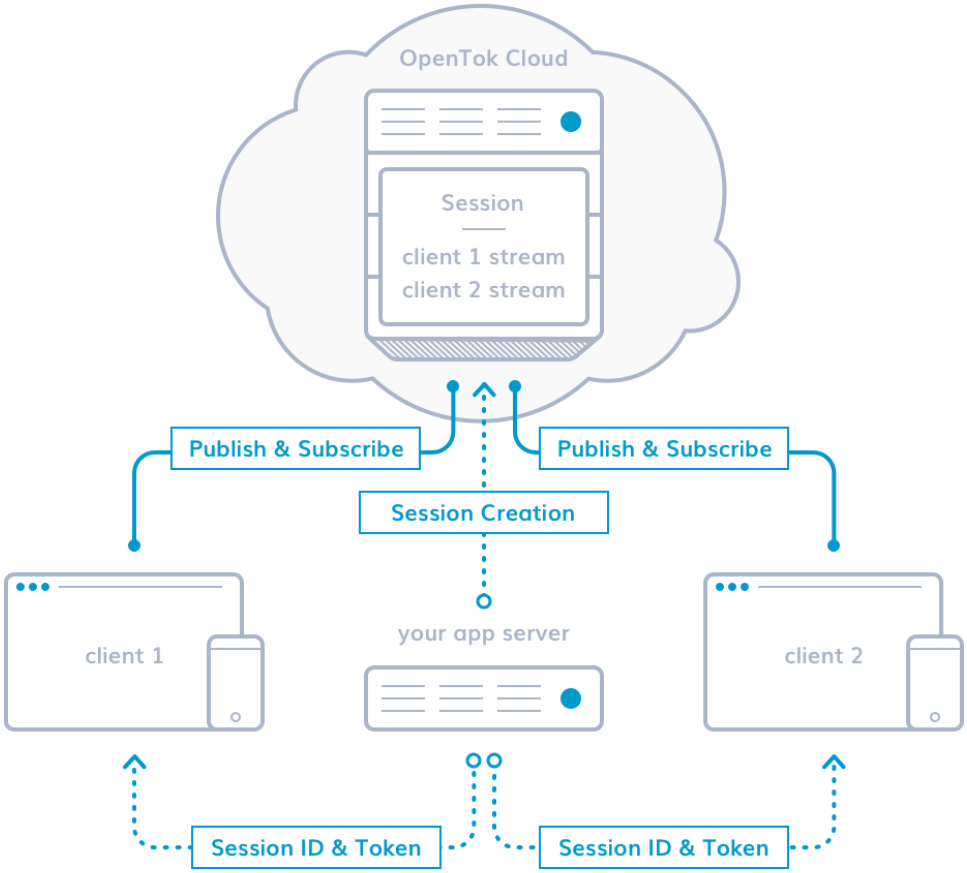


Рис. 1.3.2.1. Функції клієнта та сервера Vonage Video API

Клієнт:

- використовує клієнтські SDK;
- отримує ідентифікатор сесії та токен від сервера;
- підключається до сеансу за допомогою токена;
- віддає аудіо-відео стрім;
- слухає події сесії.

Сервер:

- використовує серверні SDK;
- створює сеанси в хмарі Vonage Video API;
- генерує токени для клієнтів;
- надсилає ідентифікатори сесій та токени клієнтам.

Сесія:

- "чат" у хмарі Vonage Video API;
- з’єднує клієнтів між собою;
- надсилає події клієнтам.

1.4 Аналіз існуючих сервісів для інтеграції відеодзвінків

API та SDK для створення відеочатів дозволяють зберегти час та гроші при створенні застосунку. Розробникам не доводиться писати всі функції відеодзвінків з нуля.

На ринку існує багато сервісів зі схожим функціоналом для інтеграції відеодзвінків. В роботі було проаналізовано функціонал та можливості двох сервісів: Twilio та Vonage. Для уникнення оплати за використання сторонніх сервісів, у власній системі було вирішено реалізувати відеодзвінки з нуля.

1.5 Опис функцій сервісів для інтеграції онлайн-оплати

1.5.1 Система LiqPay

LiqPay – платіжна система, призначена для спрощення проведення онлайн розрахунків. Містить модулі для популярних CMS. Пропонує базовий набір функціоналу розробки для інтеграції прийому платежів на сайті, який включає генерацію HTML-форми, для переходу на платіжну сторінку системи, надсилання запитів на сервер системи для проведення операції та формування підпису запиту, який використовується для перевірки відповіді сервера LiqPay. Також, система надає SDK для мобільних пристроїв IOS та AOS, що дозволяє приймати оплату картками Mastercard і Visa. [6]

Основні функції системи:

- Інтернет-еквайринг – набір інструментів для прийому платежів на сайті.
- P2P платежі – переказ коштів з картки на картку, телефон, email або розрахунковий рахунок.
- Платіжний Бот – прийом платежів в 1 клік від клієнтів в популярних месенджерах.
- Виплати – переказ грошових коштів з рахунку компанії будь-якій кількості одержувачів на картку, рахунок, email або телефон.

В рамках виконання роботи було розглянуто можливості створення платіжної сторінки (Checkout). Платіжна сторінка LiqPay (Додаток Р) дозволяє просто, швидко і безпечно отримувати платежі. Дані картки клієнта передаються безпосередньо на сервер LiqPay і не залишаються на власному сайті або сервері. [6]

Доступні методи оплати:

- Apple Pay;
- Google Pay;
- QR-код через мобільний додаток Privat24;

- картка;
- Приват24;
- LiqPay;
- розстрочка ПриватБанку;
- оплата готівкою у терміналах ПриватБанку;
- рахунок на пошту;
- MasterPass.

Схема роботи платіжної сторінки LiqPay (Додаток С) [6]:

- формування запиту до API Checkout;
- в результаті виконання запиту буде редірект клієнта на нову вкладку у браузері, де відкриється платіжна сторінка LiqPay.
- клієнт заповнює реквізити картки та підтверджує оплату на платіжній сторінці.
- сервер LiqPay надсилає інформації про статус оплати.

1.5.2 Система Stripe

Stripe – інтегрований набір платіжних продуктів. Система пропонує широкий набір варіантів оплати для різних варіантів оплати. [7]

Способи інтеграції:

- Stripe Checkout – сторінки оплати;
- Stripe Elements – створення власної процедури оплати;
- Flexible invoicing – створення кастомізованих інвойсів;
- Mobile apps – здійснення оплати використовуючи смартфон.

В рамках виконання роботи було розглянуто можливості створення платіжної сторінки (Checkout). Stripe дозволяє спростити введення інформації користувачем шляхом валідації даних та відображення помилок. Сторінка оптимізована для будь-якого пристрою. Checkout підтримує щонайменше 25 мов, 135 валют і динамічно відображає способи оплати. Також Stripe дозволяє

налаштувати кольори та брендинг торгової марки на сторінці оплати (Додаток Т). [7]

1.6 Аналіз існуючих сервісів для інтеграції онлайн-оплати

При виконанні роботи був проведений аналіз двох платіжних систем: Stripe та Liqpay. Хоч ці системи мають схожий список послуг, Stripe надає розробникам набагато більше можливостей, у порівнянні з LiqPay. Система LiqPay в свою чергу добре адаптована для українського ринку. Інфраструктура Приват банку дозволяє легко адмініструвати акаунт організації та надає можливості оплати за допомогою додатку Приват24, який є дуже розповсюдженим на території України. Зважаючи на це, для інтеграції у власну систему було обрано сервіс LiqPay.

1.7 Висновки до розділу 1

У розділі проаналізовано функції існуючих систем та визначено, що більшість використовує готові рішення. Проте, для застосування у власній майбутній системі було вирішено розглянути процес створення системи відеодзвінків з нуля, задля уникнення оплати за використання готових рішень.

Також в розділі здійснюється порівняння платіжних сервісів. Для інтеграції у власну платформу було обрано платіжний сервіс LiqPay, через широке розповсюдження та простоту використання на території України.

Розділ 2. Огляд використаних технологій та інструментів

2.1 Frontend

Для реалізації клієнтської частини системи для реалізації платних відеоконсультацій було обрано екосистему React. В цьому розділі наведений короткий опис бібліотеки React, та допоміжних бібліотек для розробки React SPA. Також, детально описується механізм роботи WebRTC та Socket.IO.

2.1.1 Бібліотека React.js

React.js – відкрита фронтенд JavaScript бібліотека, призначена для створення інтерфесів користувача та UI компонентів. React дозволяє створювати великі односторінкові застосунки з динамічним відображенням даних без перезавантаження сторінки. Зазвичай, створення React застосунку вимагає використання додаткових бібліотек для роутингу та управління сховищем. [8]

Основні особливості бібліотеки:

- Віртуальний DOM – бібліотека самостійно вирішує які компоненти сторінки треба оновити, це дозволяє забезпечити високу швидкість роботи застосунку.
- JSX – розширення синтаксису JavaScript, яке дозволяє комбінувати HTML з JavaScript.

2.1.2 Бібліотека Redux

Redux – відкрита JavaScript бібліотека, призначена для управління сховищем застосунку, яка часто використовується з React. Redux виник з ідеї Flux - архітектурного патерну, створеного інженерами з Facebook. Основною ідеєю Redux є використання одного об’єкта, в якому зберігається стан всього застосунку. Зберігання стану в одному місці

значною мірою спрощує масштабованість застосунку і дозволяє швидко знаходити проблеми або помилки. [9]

Основні концепції Redux:

- єдине джерело істини,
- стан тільки для читання,
- зміна стану за допомогою функцій.

Стан в Redux сховищі призначений тільки для читання. Єдиним спосіб його змінити є надсилання об’єкту action, який описує що сталося. Редьюсери визначають як повинен змінитись стан після того як відбулась подія. Redux не підтримує асинхронні дії, такі як отримання даних з сервера, тому було використано допоміжну бібліотеку Redux Thunk.

2.1.3 Бібліотека Styled Components

Styled Components – бібліотека для React і React Native, призначена для написання CSS за принципом CSS-in-JS. При використанні цієї бібліотека стилі обмежуються компонентом. Це дозволяє не перейматися про конфлікти іменування класів. [10]

2.1.4 Бібліотека Formik

Formik – бібліотека, яка допомагає працювати з формами в React застосунках. Вона спрощує отримання даних зі стану форми, валідацію даних, відображення помилок, сабміт форми та багато іншого. Разом з Formik використовується бібліотека Yup, яка допомагає створювати моделі для валідації. [11]

2.1.5 WebRTC (Simple-peer)

Simple-peer відповідає за реалізацію WebRTC у браузері та надає зручний api для роботи з ним. WebRTC (Web Real Time Communication) – стандарт, який описує передачу потокових даних між браузерами та іншими застосунками що його підтримують. [12]

WebRTC може бути використаний для:

- аудіо стріму,
- відео стріму,
- надсилання файлів,
- відео чату,
- створення багатокористувацьких ігор.

Популярні сервіси, що використовують технологію WebRTC:

- Google Meet,
- Jitsi Meet,
- BigBlueButton.

Переваги стандарту WebRTC:

- Не вимагає установки стороннього програмного забезпечення.
- Висока якість зв’язку (використання сучасних аудіо та відео кодеків, автоматичне підлаштування якості потоку під умови з’єднання, вбудована система шумозаглушення, автоматичне регулювання чутливості мікрофонів учасників).
- Високий рівень безпеки (з’єднання захищене і зашифроване згідно з протоколами DTLS і SRTP. WebRTC працює тільки по протоколу HTTPS).
- Вбудований механізм захоплення контенту, наприклад, робочий стіл.

При використанні WebRTC, для налагодження з’єднання, користувачі обмінюються SDP-діаграмами (Session Description Protocol) (рисунок 2.1.5.1). Діаграма являє собою текстовий файл, який містить всю необхідну інформацію для створення з’єднання. Файл містить інформацію про IP-адресу, про порти, які використовуються, про те, якою саме інформацією обмінюються клієнти, та про те, які кодеки використовуються.

```
v=0
o=- 1815849 0 IN IP4 192.168.0.15
s=Cisco SDP 0
c=IN IP4 194.67.15.181
t=0 0
m=audio 20062 RTP/AVP 99 18 101 100
a=rtpmap:99 G.729b/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=rtpmap:100 X-NSE/8000
a=fmtp:100 200-202
```

Рис. 2.1.5.1 SDP-діаграмама

Важливим зауваженням є те, що в діаграмі вказана IP-адреса пристрою, який знаходиться в якійсь локальній мережі. Тому якщо користувачі з різних локальних мереж захочуть налагодити з’єднання, в них це не вийде.

Для розв'язання цієї проблеми використовується фреймворк ICE (Internet Connectivity Establishment). Він описує способи обходу NAT (Network address translation). Цей фреймворк використовує приписування STUN-сервера (рисунок 2.1.5.2).

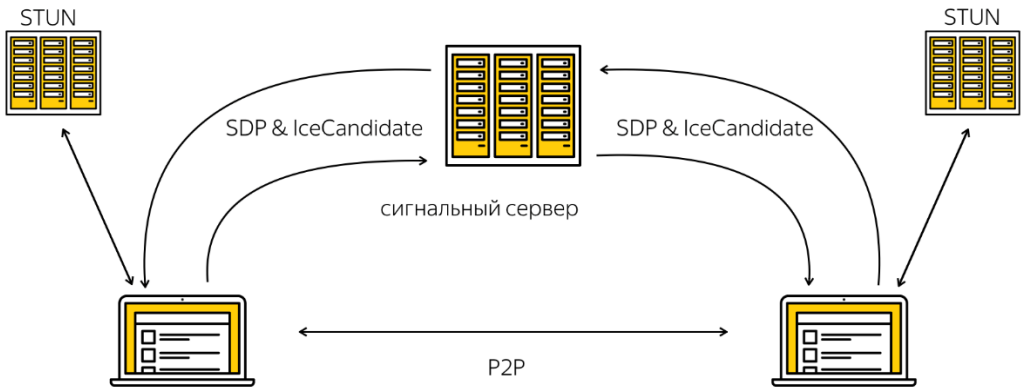


Рис. 2.1.5.2 Схема роботи відеодзвінків з використанням STUN-серверів

STUN-сервер – спеціальний сервер, при зверненні до якого, можна дізнатись свою зовнішню IP-адресу. Таким чином в процесі установки P2P з’єднання, кожен з клієнтів повинен зробити запит до цього STUN-сервера, для того щоб дізнатись свою IP-адресу та сформувати додаткову інформацію (IceCandidate) і за допомогою сигнального механізму обмінятися цим IceCandidate. Після цього клієнти будуть знати правильну IP-адресу співрозмовника, та зможуть встановити P2P з’єднання.

Окремим випадком є ситуація, коли пристрій схований за подвійним NAT. В цьому випадку фреймворк ICE передбачає використання TURN-сервера (рисунок 2.1.5.3). TURN-сервер – спеціальний сервер, який перетворює з’єднання клієнт – клієнт на з’єднання клієнт – сервер – клієнт, виступаючи у ролі ретранслятора.

Незалежно від того, яким з трьох сценаріїв відбувається з’єднання, API-технологія буде ідентичною. Для налаштування потрібно вказати конфігурацію ICE- і TURN-серверів. [13]

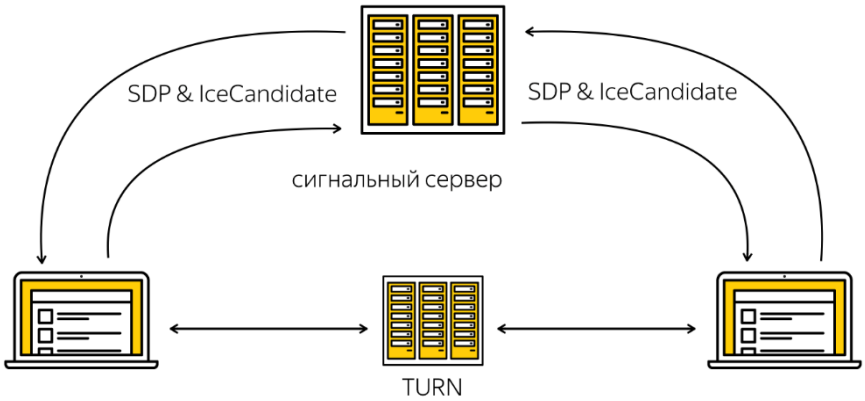


Рис. 2.1.5.3 Схема роботи відеодзвінків з використанням TURN-сервера

2.1.6 Socket.io

Socket.IO - бібліотека, яка забезпечує взаємодію між браузером та сервером у режимі реального часу, у двосторонньому режимі та на основі подій. Socket.IO складається з клієнтської та серверної частини. [14]

Варіанти передачі даних в Socket.IO:

- HTTP long-polling (складається з послідовності запитів (long-running GET запити для отримання даних з сервера та short-running POST запити, надсилання даних на сервер) (рисунок 2.1.6.1),
- WebSocket.

Основні особливості Socket.IO:

- надійність (якщо не вдається встановити з'єднання WebSocket, буде використано HTTP long-polling),
- автоматичне повторне підключення,

- трансляція для всіх клієнтів або для підмножини (кімнати).

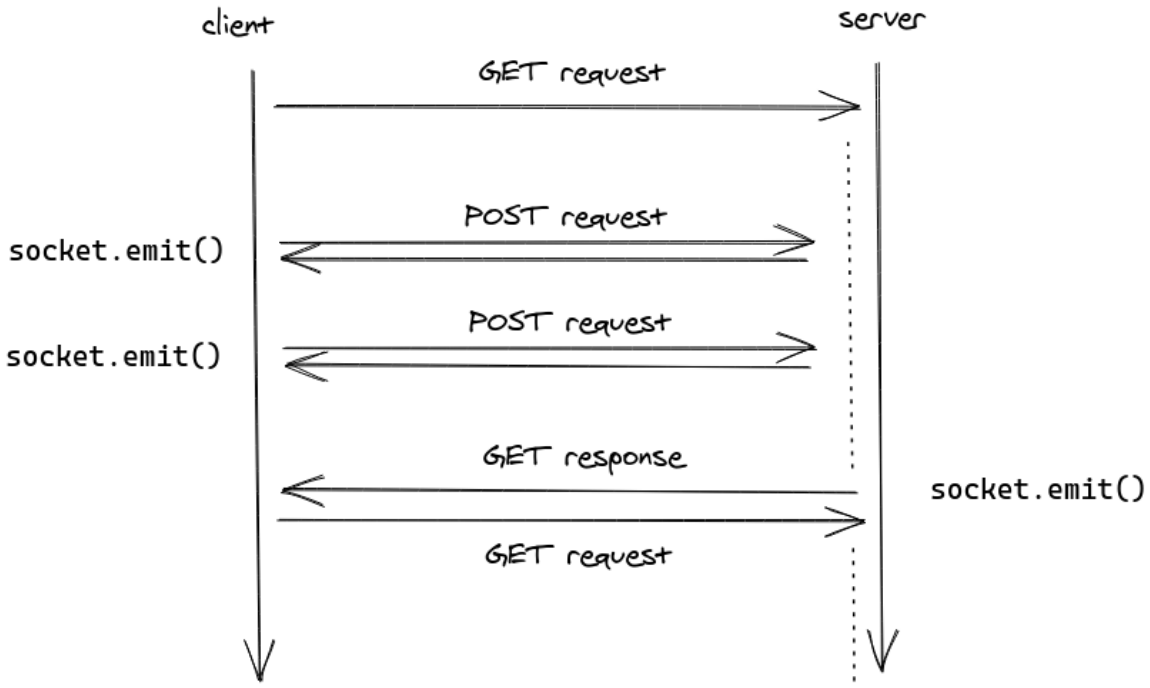


Рис. 2.1.6.1. Структура Socket.IO HTTP long-polling

2.2 Backend

Для реалізації Rest API було обрано платформу Node.js та фреймворк Express. Взаємодія з базою даних виконується за допомогою бібліотеки Sequelize. Дали наведено короткий опис технологій, використаних при реалізації власної системи.

2.2.1 Node.js

Node.js – кросплатформенна програмна платформа для серверної розробки на мові JavaScript. В основі Node.js лежить JavaScript Google V8, який дозволяє транслювати JavaScript в машинний код. Node.js використовує керовану подіями, неблокуючу модель вводу та виводу, що робить її легкою і ефективною. [15]

2.2.2 Express

Express – фреймворк для створення веб застосунків та API на платформі Node.js.

Основні механізми, які надає Express [16]:

- написання обробників для запитів з різними HTTP методами в різних маршрутах,
- інтеграція з механізмами рендеренга «view», для генерації відповіді, вставляючи дані в шаблони.
- установка загальних параметрів веб застосунку, таких як порт для підключення і місце розміщення шаблонів, які використовуються для відображення відповіді.
- «middleware» для додаткової обробки запита в будь-який момент часу під час обробки запита.

2.2.3 Sequelize

Sequelize – ORM (Object–relational mapping) бібліотека для Node.js застосунків, яка здійснює зіставлення таблиць в базі даних і відношеннями між ними та класами. При роботі з Sequelize можна не писати SQL запити, а працювати з даними так, як зі звичайними об’єктами. [17]

Підтримувані СУБД:

- Postgres,
- MySQL,
- MariaDB,
- SQLite,
- Microsoft SQL Server.

2.2.4 Liqpay/sdk-nodejs

Liqpay/sdk-nodejs – набір базових засобів розробки, який спрощує інтеграцію прийому платежів через систему Liqpay на власному сайті.

Базовий набір функціоналу Liqpay sdk-nodejs [6]:

- генерація html-форми для переходу на сторінку оплати LiqPay,
- надсилання запиту на сервер LiqPay для проведення операції,
- формування підпису запита для перевірки справжності отриманої від LiqPay відповіді.

2.3 Висновки до розділу 2

Для реалізації Rest API було обрано фреймворк Express, який працює на платформі Node.js. Просту та швидку роботу з базою дозволяє забезпечити Sequelize ORM. Оновою клієнтського застосунку було обрано бібліотеку React. За управління станами в застосунку відповідає бібліотека Redux. Відеодзвінки здійснюються за допомогою технології WebRTC.

Описані інструменти та технології дозволяють швидко створити надійний сервер та гнучкий, легко масштабований клієнтський застосунок.

Розділ 3. Програмна реалізація системи платних консультацій

3.1 Технічне завдання

На основі аналізу існуючих рішень та інструментів розробки, виконується формування технічного завдання для реалізації власної системи. При визначенні необхідного функціоналу, було вирішено реалізувати лише основні функції для забезпечення проведення відеоконсультацій для забезпечення універсальності системи.

3.1.1 Основні вимоги

3.1.1.1 Ролі користувачів системи

Ролі користувачів системи: гість, авторизований користувач, консультант. Функціонал, доступний для неавторизованих користувачів (гостей сайту):

- перегляд активних консультантів;
- авторизація.

Для авторизованих користувачів:

- поповнення рахунку;
- перегляд історії транзакцій;
- перегляд оплачених консультацій (забезпечити можливість повернутися до консультації);
- перегляд активних консультантів;
- оплата консультації;
- отримання консультаційних послуг.

Для консультантів:

- перегляд активних консультантів;
- перегляд історії транзакцій;

- перегляд оплачених консультацій (забезпечити можливість повернутися до консультації);
- зміна статусу активності;
- зміна детальної інформації консультанта;
- надавання консультаційних послуг.

3.1.1.2 Опис функціоналу сторінок та елементів інтерфейсу

Головна панель навігації має бути присутньою на всіх сторінках. Не відображається під час проведення консультації. Відображення відповідно до ролі:

- Для гостей сайту містить посилання на головну сторінку, сторінку перегляду списку всіх активних консультантів та на сторінки реєстрації та логіну.
- Для авторизованого користувача – посилання на головну сторінку, сторінку перегляду списку всіх активних консультантів, сторінку перегляду історії оплачених замовлень, профіль користувача та кнопку перемикання на роль консультанта.
- Для консультанта – посилання на головну сторінку, сторінку зміни статусу активності, сторінку перегляду історії оплачених замовлень, сторінку перегляду списку всіх активних консультантів, профіль та кнопку перемикання на роль користувача.

Головна сторінка:

- Для гостей сайту містить загальну інформацію про систему, посилання для переходу на сторінки авторизації та сторінку зі списком всіх активних консультантів.
- Для авторизованих користувачів – загальна інформація про процес надавання та отримання консультацій, кнопки для переходу на сторінку перегляду списку всіх активних консультантів та для переходу в інтерфейс консультанта.

- Для консультантів – загальна інформація про процес надавання консультацій, кнопка для переходу на сторінку перегляду списку всіх активних консультантів, кнопка для переходу на сторінку профілю для заповнення детальної інформації про консультанта та кнопка переходу на сторінку зміни статусу активності консультанта.

Сторінка перегляду активних консультантів:

- Для гостей сайту та для консультантів – повинна містити список активних консультантів. Картка консультанта повинна містити відображення короткої інформації про консультанта (ім’я, короткий опис, ціна).
- Для авторизованого користувача – повинна містити список активних консультантів, який оновлюється динамічно, без перезавантаження сторінки. Картка консультанта повинна містити відображення короткої інформації про консультанта (ім’я, короткий опис, ціна). Для кожного елементу списку повинна бути кнопка переходу на сторінку оформлення та оплати замовлення.

Сторінка створення замовлення доступна лише для авторизованих користувачів та містить картку консультанта, в якій відображається ім’я та опис консультанта, правила проведення консультацій, вартість години консультацій з вибраним консультантом, форма для вибору тривалості консультації, кнопка для підтвердження замовлення та переходу до кімнати відеоконсультації.

На сторінці відео консультації повинна відображатися кнопка завершення дзвінка та запит на використання медіа пристроїв (якщо такий дозвіл ще не отримано). Якщо доступ до медіа пристроїв успішно отримано, відображається відео з власної веб-камери, відео співрозмовника, аудіо співрозмовника, відображення інформацію про статус відео дзвінка.

Сторінка профілю:

- Для авторизованого користувача та для консультанта – відображається інформація про користувача, кнопка виходу з системи, поточний баланс, кнопка поповнення балансу; історія транзакцій. Для кожного елемента у списку транзакцій повинна відображатися сума, опис та час здійснення транзакції.
- В інтерфейсі консультанта сторінка профілю доповнюється формою для введення детальної інформації про консультанта.

Сторінка поповнення рахунку доступна для авторизованого користувача та консультанта. Містить відображення балансу користувача, форму для вибору суми поповнення, відображення інформації про комісію для майбутньої транзакції, кнопку для генерації форми LiqPay, після натискання якої буде відображено модальне вікно з кнопкою оплати LiqPay. При натисканні на кнопку підтвердження оплати в модальному вікні відбувається редірект на сторінку оплати системи LiqPay.

Сторінку зміни статусу активності доступна лише для консультанта та повинна містити опис механізму надання консультацій на платформі, відображення обраної вартості консультацій, відображення статусу активності, посилання на профіль консультанта для зміни вартості консультацій, кнопка перемикання статусу активності консультанта, при натисканні якої, консультант з’являється у списку активних консультантів в інших користувачів системи. При покиданні цієї сторінки, статус консультанта повинен автоматично змінюватись на неактивний.

3.1.1.3 Опис функціоналу серверної частини застосунку

Вимоги до архітектури:

- сервер має бути реалізований з використанням REST архітектури;
- для роботи з динамічним контентом має бути використано socket.io;

- CRUD (create, read, update, delete) – 4 базові функції управління даними «створення, зчитування, зміна і видалення»;
- авторизація користувачів з використанням JWT (JSON Web Token).

Основні вимоги до функціонала:

- Реалізація CRUD для всіх моделей системи.
- При оформленні замовлення, оплата повинна здійснюватися автоматично: кошти знімаються з балансу користувача, та додаються до балансу консультанта.
- Реалізація отримання відповіді від сервера LiqPay про статус здійснення оплати, при поповненні рахунку в профілі користувача.
- Забезпечення можливості динамічного оновлення списку консультантів в інтерфейсі користувача.
- Забезпечення обміну інформацією між користувачами при налагодженні WebRTC з’єднання.

3.1.2 Заплановані допрацювання

- Текстовий чат:
  - новий вид консультування: текстовий чат;
  - можливість ділитися файлами під час текстової консультації;
  - збереження історії текстового чату.
- Планування консультацій:
  - можливість створення розкладу надавання послуг консультантом;
  - бронювання консультацій.
- Вивід коштів з системи.
- Реалізація інтерфейсу адміністратора.

3.1.3 Перспективи розвитку

Можливі покращення:

- моніторинг процесу консультування зі збереженням подій відео дзвінка (під'єднання, роз'єднання, проблеми зі з'єднанням), для подальшої обробки скарг на надані послуги від користувачів:
- створення системи рейтингу консультантів;
- можливість демонстрації екрану.

3.2 Розробка системи

3.2.1 Backend

3.2.1.1 Структура бази даних

Після аналізу вимог, перед початком розробки системи було створено орієнтовну ER-модель майбутньої бази даних (рисунок 3.2.1.1.1).

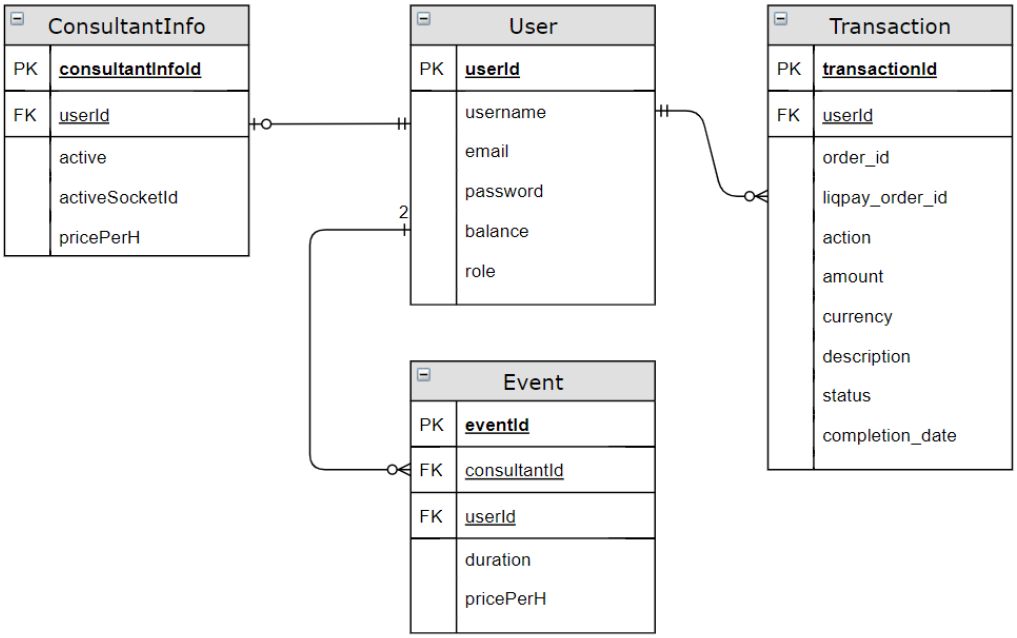


Рис. 3.2.1.1.1. ER-модель бази даних

3.2.1.2 Створення структури Express REST API

Фреймворк Express значно спрощує розробку на Node.js, однак він не пропонує способів організації проекту. Вибір архітектури проекту – дуже важливий етап розробки.

Для реалізації системи було вирішено використати трирівневу архітектуру. Ідея полягає у винесенні бізнес-логіки з API роутів (рисунок 3.2.1.2.1). [18]

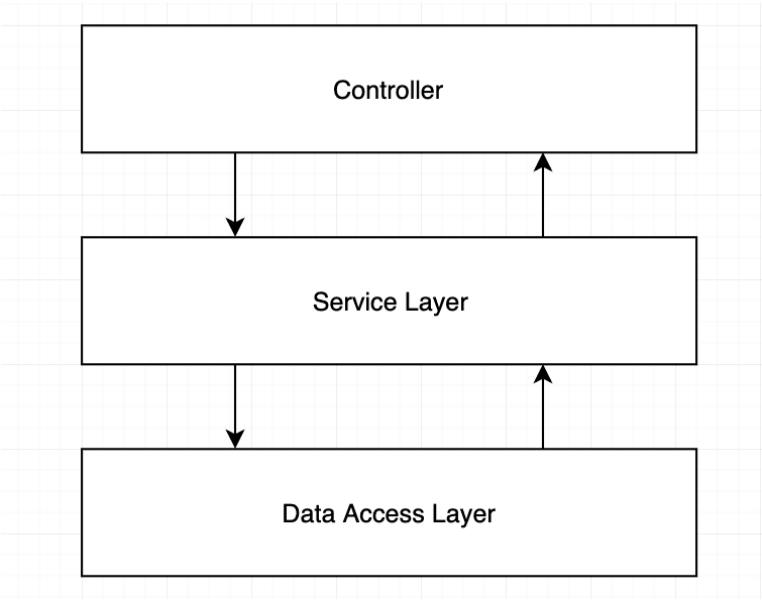


Рис. 3.2.1.2.1. Трирівнева архітектура Express

Для забезпечення модульної структури коду, код було розділено на наступні папки та файли [19]:

- models – визначення схем моделей.
- routes – API роути до контролерів.
- controllers – відповідають за валідацію параметрів запиту та за надсилання відповідей від сервера з правильними кодами.
- services – відповідають за отримання даних з бази та повернення об’єктів або надсилання помилок.

3.2.1.3 Налаштування Sequelize ORM

Після установки бібліотеки за допомогою команди «npm install –save sequelize», потрібно створити об’єкт Sequelize (рисунок 3.2.1.2.1). Для властивості dialect в залежності від СУБД можуть використовуватись такі значення: mysql, mariadb, sqlite, postgres, mssql.

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize({
    dialect: 'sqlite',
    storage: './db.sqlite',
});
```

Рис. 3.2.1.3.1. Створення об'єкта Sequelize

Ключовим компонентом в роботі Sequelize з базою даних є моделі, які описують структуру даних. Для спрощення та пришвидшення процесу створення нових моделей, було використано механізм автоматичного імпорту (рисунок 3.2.1.3.2).

```
fs.readdirSync(__dirname).filter((file : string ) =>
    file !== 'index.js',
).forEach((file : string ) => {
    const model = require(path.join(__dirname, file))(sequelize, Sequelize.DataTypes);
    db[model.name] = model;
});

Object.keys(db).forEach(function(modelName : string ) {
    if ('associate' in db[modelName]) {
        db[modelName].associate(db);
    }
});
```

Рис. 3.2.1.3.2. Автоматичний імпорт моделей Sequelize

Перед початком взаємодії з базою даних, потрібно впевнитись у відповідності таблиць в базі даних з визначеними моделями. Для синхронізації використовується метод sync().

3.2.1.4 Авторизація JWT

JWT (JSON Web Token) – JSON рядок, який формується шляхом кодування даних за допомогою секретного слова. JWT (рисунок 3.2.1.4.1) складається з трьох частин, розділених крапками:

- Header – рядок, який описує токен і використаний алгоритм хешування.
- Payload – дані, які зберігаються в токені.
- Signature – підпис, згенерований на основі заголовка та даних.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9          // header (заголовок)
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9     // payload (корисне навантаження)
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM  // signature (підпис)
```

Рис. 3.2.1.4.1. JSON Web Token

Згенерований JWT надсилається клієнту, де він зберігається в localStorage і буде надсилатися клієнтом серверу при кожному HTTP запиті всередині заголовка (рисунок 3.2.1.4.2).

```
Authorization: Bearer {token}
```

Рис. 3.2.1.4.2. Authorization header

Для створення сервісу для авторизації потрібно установити пакет jsonwebtoken та створити змінну для зберігання секретного ключа. При авторизації, клієнт надсилає два поля: email та password.

Якщо користувача з таким email знайдено та його пароль відповідає введеному, потрібно створити JWT та надіслати його у відповідь на запит. Метод jwt.sign() приймає payload, секретне слово та додаткові налаштування. В payload буде передано ідентифікатор та роль користувача (рисунок 3.2.1.4.3).

```
const user = await User.findOne({
    where: {
        email: email
    }
});

if (!user) {
    reject( reason: {message: "User not found"});
}

// check if the password is valid
let passwordIsValid = bcrypt.compareSync(password, user.password);
if (!passwordIsValid) {
    reject( reason: {message: "Wrong password"});
}

// if user is found and password is valid
// create a token
const token = jwt.sign(
     payload: { sub: user.id, role: user.role },
    config.secret,
     options: {expiresIn: 86400}
    );
```

Рис. 3.2.1.4.3. Авторизація користувача та генерація JWT

Для перевірки валідності токена та авторизації за ролями було створено Express.js middleware (рисунок 3.2.1.4.4). Пакет express-jwt дозволяє помістити об’єкт payload який містить JWT, в req параметри запиту для подальшого використання в контролерах.

```
const jwt = require('express-jwt');
const {secret} = require('../config');

function authorize(roles :any[] = []) {
    // roles param can be a single role string (e.g. Role.user or 'user')
    // or an array of roles (e.g. [Role.admin, Role.user] or ['admin', 'user'])
    if (typeof roles === 'string') {
        roles = [roles];
    }

    return [
        // authenticate JWT token and attach user to request object (req.user)
        jwt( options: { secret, algorithms: ['HS256'] }),

        // authorize based on user role
        (req, res, next) => {
            if (roles.length && !roles.includes(req.user.role)) {
                // user's role is not authorized
                return res.status(401).json({ message: 'Unauthorized' });
            }

            // authentication and authorization successful
            next();
        }
    ];
}

module.exports = authorize;
```

Рис. 3.2.1.4.4 Authorize middleware

Для використання створеної функції, необхідно помістити метод router.all вище методів, доступ до яких вимагає авторизації (рисунок 3.2.1.4.5).

```
router.route( prefix: '/:id')
    .all(authorize(Role.user))
    .post(eventController.create);
```

Рис. 3.2.1.4.5 Authorize middleware usage

3.2.1.5 Онлайн оплата LiqPay Checkout

Схема роботи LiqPay Checkout передбачає формування кнопки для оплати та обробку відповіді від сервера зі статусом оплати LiqPay на стороні власного сервера. Клієнт надсилає на сервер суму поповнення, після цього, використовуючи LiqPay Node.js SDK, виконується генерація (рисунок 3.2.1.5.1) кнопки (форми) для оплати (рисунок 3.2.1.5.2). Після натискання кнопки клієнт перенаправляється на сторінку оплати LiqPay.

```
const liqpay = new LiqPay(public_key, private_key);
const html = liqpay.cnb_form( params: {
    action: 'pay',
    amount: amount,
    currency: 'UAH',
    description: `Deposit`,
    order_id: v4(),
    version: '3',
    server_url: 'https://992a94372047.ngrok.io/liqpay/result',
    result_url: 'http://localhost:3000/user/profile',
    customer: userId,
    commission_payer: 'sender',
});
```

Рис. 3.2.1.5.1 Генерація кнопки оплати LiqPay Checkout


```
<form method="POST" action="https://www.liqpay.ua/api/3/checkout"
accept-charset="utf-8">
<input type="hidden" name="data" value="eyAidmVyc2lvbiIgOiAzLCAicHVibGljX2tleSIgOiAieW91cl9wdWJsaWNfa2V5IiwgImFjdGlv
biIgOiAicGF5IiwgImFtb3VudCIgOiAxLCAiY3VycmVuY3kiIDogIlVTRCIsICJkZXNjcmlwdGlv
biIgOiAiZGVzY3JpcHRpb24gdGV4dCIsICJvcmRlcl9pZCIgOiAib3JkZXJfaWRfMSIgfQ==">
<input type="hidden" name="signature" value="QvJD5u9Fg55PCx/Hdz6lzWtYwcI=">
<input type="image"
src="//static.liqpay.ua/buttons/p1ru.radius.png">
</form>
```


Рис. 3.2.1.5.2 Приклад згенерованої форми LiqPay Checkout

Після здійснення оплати користувачем, сервер LiqPay надсилає POST запит з двома параметрами data і signature:

- data - json рядок з інформацією про платіж, закодований функцією base64;
- signature – унікальний підпис кожного запиту.

Для перевірки справжності запита, отриманого від сервера LiqPay, потрібно сформувати signature на стороні власного сервера, за допомогою функції str_to_sign LiqPay Node.js SDK, використовуючи data з запиту та власний private_key. Після цього, отриману signature потрібно порівняти з тою, яка була отримана від сервера LiqPay. Якщо signature ідентичні, можна бути впевненим, що відповідь від сервера LiqPay справжня (рисунок 3.2.1.5.3). [6]

```
const liqpay = new LiqPay(public_key, private_key);
const sign = liqpay.str_to_sign(
    str: private_key +
    data +
    private_key
);

if (sign !== signature) {
    reject( reason: {message: "Signature is not valid"});
}
```

Рис. 3.2.1.5.3 Перевірка справжності відповіді від сервера LiqPay

3.2.1.6 Відеодзвінки (WebRTC, Socket.IO)

Сервер відповідає за створення дзвінка та налагодження P2P з’єднання між клієнтами. Сам дзвінок здійснюється напряму між браузерами. Для установки бібліотеки Socket.IO використовується команда npm i socket.io.

Для підключення socket.io до сервера express потрібно використати відмінну від звичайної процедуру створення сервера з використанням функції http.createServer (рисунок 3.2.1.6.1).

```
const app = express();
const http = require("http");
const server = http.createServer(app);
const socket = require("socket.io");
const io = socket(server, {
    cors: {
        origin: "http://localhost:3000",
        methods: ["GET", "POST"]
    }
});
```

Рис. 3.2.1.6.1 Створення Express сервера та підключення socket.io

При підключенні користувача до кімнати, на сервер надсилається подія "join-room". Сервер перевіряє допустиму кількість користувачів в кімнаті, та надсилає іншим користувачам кімнати інформацію про користувача, який приєднується до кімнати (рисунок 3.2.1.6.2).

```
socket.on("join-room", async roomID => {
    const event = await find(roomID);
    if (!event) {
        return;
    }
    io.to(event.ConsultantInfo.activeSocketId).emit('event-created', {roomID});

    if (rooms[roomID]) {
        const length = rooms[roomID].length;
        if (length === 2) {
            io.to(socket.id).emit('room-full');
            return;
        }
        io.to(rooms[roomID][0]).emit('user-join-req', socket.id);
        rooms[roomID].push(socket.id);
    } else {
        rooms[roomID] = [socket.id];
    }
    socketToRoom[socket.id] = roomID;
});
```

Рис. 3.2.1.6.2 Обробка події " join-room"

Для обробки надсилання та отримання сигналу frontend бібліотеки simple-peer, для ініціалізації обміну потоковою відео та аудіоінформацією, було створено обробники подій (рисунок 3.2.1.6.3): "sending-signal" та "returning-signal".

```
socket.on("sending-signal", payload => {
    io.to(payload.userToSignal).emit('user-joined', {signal: payload.signal, callerID: payload.callerID});
});

socket.on("returning-signal", payload => {
    io.to(payload.callerID).emit('receiving-returned-signal', {signal: payload.signal, id: socket.id});
});
```

Рис. 3.2.1.6.3 Обробка подій бібліотеки simple-peer

Для обробки завершення дзвінка та обробки втрати з’єднання з користувачем було створено обробник події "disconnect" (рисунок 3.2.1.6.4).

```
socket.on('disconnect', async () => {
    await deactivate(socket.id);
    socket.broadcast.emit( ev: 'consultant-list-updated');

    const roomID = socketToRoom[socket.id];
    let room = rooms[roomID];
    if (room) {
        room = room.filter(id => id !== socket.id);
        rooms[roomID] = room;
    }
    if (rooms[roomID]) {
        const usersInThisRoom = rooms[roomID].filter(id => id !== socket.id);
        socket.emit("all-users", usersInThisRoom);
        io.to(usersInThisRoom[0]).emit('user-disconnected', {id: socket.id});
    }
});
```

Рис. 3.2.1.6.4 Обробка події "disconnect"

3.2.2 Frontend

3.2.2.1 Створення проекту за допомогою Create React App

Для спрощення створення React.js SPA проекту було використано CLI (command line interface) create-react-app. Інструмент створює проект з налаштованими Webpack, Babel та іншими інструментами для розробки.

3.2.2.2 Підключення та налаштування Redux

Для забезпечення роботи бібліотеки Redux в React застосунку, необхідно встановити саму бібліотеку Redux командою "npm i redux". Також, потрібно встановити допоміжні модулі: React Redux, Redux Thunk та Redux devtools extension. Для цього потрібно виконати команду "npm i react-redux redux-thunk redux-devtools-extension". Модуль React Redux відповідає за роботу Redux з React. Модуль Redux Thunk – middleware для забезпечення можливості створення асинхронних actions в Redux (рисунок 3.2.2.2.1). Redux devtools extension – надає можливості моніторингу та відладки зміни станів в Redux сховищі.

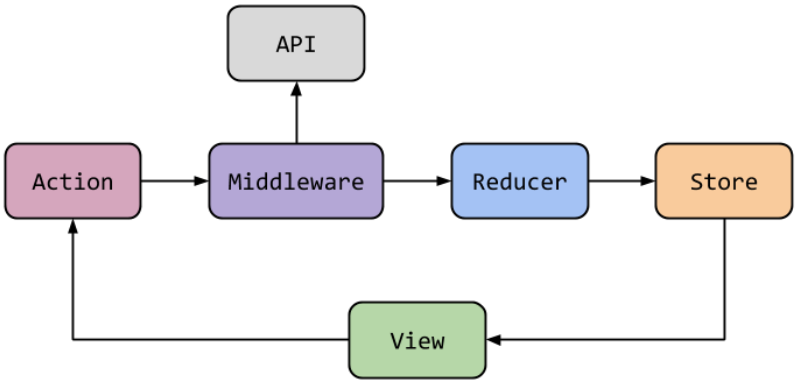


Рис. 3.2.2.2.1 Схема роботи React, Redux і Redux Thunk

Для організації файлової структури в Frontend застосунку було вирішено створити три папки: actions, actionTypes, reducers (рисунок 3.2.2.2.2).

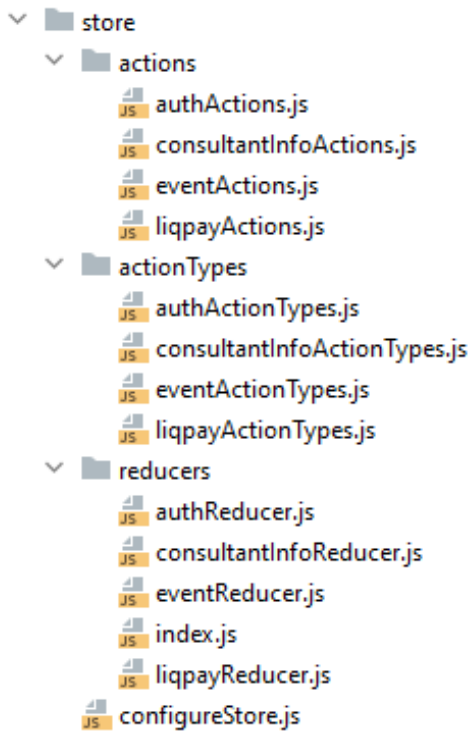


Рис. 3.2.2.2.2 Структура файлів для роботи з Redux сховищем

Функція для створення та налаштування сховища винесена в окремий файл: configureStore.js (рисунок 3.2.2.2.3). Для створення Redux сховища використовується функція createStore бібліотеки redux.

Для підключення Redux Thunk використано функцію applyMiddleware, імпортованої з бібліотеки redux. Для забезпечення підключення розширення Redux Devtools, було використано функцію composeWithDevTools з бібліотеки redux-devtools-extension.

```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

export default function configureStore() {
    return createStore(
        rootReducer,
        composeWithDevTools(applyMiddleware(thunk))
    );
}
```

Рис. 3.2.2.2.3 Функція для створення та налаштування сховища Redux

Функція createStore приймає rootReducer, створений для поєднання всіх редюсерів застосунку в один. Для його створення використовується функція combineReducers, імпортована з бібліотеки redux, в яку потрібно передати об’єкт, в якому визначені всі редюсери застосунку (рисунок 3.2.2.2.4).

```
import { combineReducers } from 'redux'
import authReducer from './authReducer';
import liqpayReducer from './liqpayReducer';
import consultantInfoReducer from './consultantInfoReducer';
import eventReducer from './eventReducer';

export default combineReducers( reducers: {
    auth: authReducer,
    liqpay: liqpayReducer,
    consultantInfo: consultantInfoReducer,
    event: eventReducer,
});
```

Рис. 3.2.2.2.4 Створення rootReducer

У файлі index.js потрібно обгорнути головний компонент App провайдером, імпортованим з бібліотеки react-redux (рисунок 3.2.2.2.5). Компоненту Provider потрібно передати об’єкт store, який створюється функцією configureStore.

```
import configureStore from './store/configureStore'
import {Provider} from 'react-redux';

ReactDOM.render(
    <React.StrictMode>
        <Provider store={configureStore()}>
            <Router history={history}>
                <App/>
            </Router>
        </Provider>
    </React.StrictMode>,
    document.getElementById( elementId: 'root')
);
```

Рис. 3.2.2.2.5 Використання компонента Provider бібліотеки react-redux

3.2.2.3 Підключення маршрутизації React Router

В будь-якому веб-застосунку повинна бути маршрутизація. Користувач повинен бачити де він знаходиться в застосунку в будь-який момент часу. А бачить він своє місцеперебування в адресному рядку браузера. Веб застосунок повинен зіставляти URL з відповідною йому сторінкою.

Також, важливим є забезпечення роботи історії. Користувач повинен мати можливість повертатися на попередню сторінки натискаючи кнопку відповідну кнопку в браузері. Сам React не надає такої можливості, тому для реалізації маршрутизації використовуються сторонні бібліотеки. Для використання було обрано бібліотеку React Router.

React Router – досить популярна і проста бібліотека для маршрутизації з гарною документацією. Основні можливості:

- навігація по кліку (компонент Link),
- перенаправлення (компонент Redirect),

- маршрутизація (компонент Route),
- історія (властивість history).

У п’ятій версії React Router з’явилася підтримка хуків. Через це було вирішено використовувати найновішу версію 5.2.0. Огляд основних хуків, які були використані при реалізації застосунку [20]:

- useHistory – надає доступ до параметра history, за допомогою якого можна здійснювати роутинг, використовуючи функції push, replace та інші.
- useLocation – надає доступ до параметра location, яка є незмінюваною. Це надає можливість відстежувати зміну роута та керувати параметрами пошуку.
- useParams – надає доступ до параметрів пошуку з URL.

Забезпечення можливості використання об’єкту history в Redux actions вимагає використання тільки визначених версій history. React Router (react-router-dom) 5.x працює з history 4.x. Для реалізації було використано бібліотеку history версії 4.10.1. Для забезпечення можливості доступу з Redux, потрібно створити власний об’єкт history (рисунок 3.2.2.3.1). [21]

```
import {createBrowserHistory} from 'history'

const history = createBrowserHistory();
export default history;
```

Рис. 3.2.2.3.1 Створення власного об’єкта history

Також, потрібно передати створений об’єкт до компонента Router, імпортованого з бібліотеки react-router-dom. Важливо не сплутати з BrowserRouter. Це дозолить синхронізувати роутинг, який виконується в React з Redux роутингом.

```
import {Router} from "react-router-dom";
import configureStore from './store/configureStore'
import {Provider} from 'react-redux';

ReactDOM.render(
    <React.StrictMode>
        <Provider store={configureStore()}>
            <Router history={history}>
                <App/>
            </Router>
        </Provider>
    </React.StrictMode>,
    document.getElementById( elementId: 'root')
);
```

Рис. 3.2.2.3.2 Підключення власного об’єкта history до маршрутизації застосунку

3.2.2.3 Авторизація JWT

Для реалізації JWT авторизації на стороні клієнта було створено authReducer.js (рисунок 3.2.2.3.1) і authActions.js. У файлі authActions.js відбуваються всі запити до сервера.

```
export default (state :{...} = initialState, action) => {
    switch (action.type) {
        case AUTH_USER:
            return {...state, user: action.payload};
        case AUTH_LOADING:
            return {...state, authLoading: action.payload};
        case LOGIN_LOADING:
            return {...state, loginLoading: action.payload};
        case LOGIN_ERROR:
            return {...state, loginError: action.payload};
        case SIGNUP_LOADING:
            return {...state, signupLoading: action.payload};
        case SIGNUP_ERROR:
            return {...state, signupError: action.payload};
        default:
            return state;
    }
};
```

Рис. 3.2.2.3.1 Вміст файлу authReducer.js

Функція loginUser викликається при сабміті форми на сторінці "Login" (Додаток В), функція signupUser – на сторінці Signup (Додаток Г) відповідно. Після успішної авторизації, сервер повертає JWT та об’єкт користувача.

Для запису та отримання токена з localStorge, використовуються функції setToken та getToken (рисунок 3.2.2.3.3). Для запису об’єкта користувача до сховища викликається функція "dispatch(authUser(response.data.user))". Після цього відбувається перенаправлення на головну сторінку авторизованого користувача (Додаток Д).

```
export const getToken = () => {
    return localStorage.getItem( key: "access_token");
}
export const setToken = (token) => {
    localStorage.setItem("access_token", token);
}
```

Рис. 3.2.2.3.3 Робота з localStorge

Для обробки стану завантаження, та відображення користувачу прелоадерів, використовується функції "dispatch(signupLoading(bool))" та "dispatch(signupLoading(bool))", які відповідають за перемикання стану loginLoading та signupLoading відповідно.

Для створення форм було використано бібліотеку Formik. Бібліотека yup відповідає за створення схем валідації (рисунок 3.2.2.3.4).

```
const SignupSchema = Yup.object().shape({
    username: Yup.string()
        .min(3, 'Too Short!')
        .max(10, 'Too Long!')
        .required('Required'),
    email: Yup.string().email('Invalid email').required('Required'),
    password: Yup.string().required('Required').min(8, 'Seems a bit short...'),
    confirmPassword: Yup
        .string()
        .required('Required')
        .test('passwords-match', 'Passwords must match', function (value) {
            return this.parent.password === value;
        }),
});
```

Рис. 3.2.2.3.4 Схема для валідації Signup форми

Бібліотека Formik дозволяє виконати валідацію введених користувачем даних, та запобігати надсиланню форми, якщо вони не відповідають схемі валідації, яку потрібно передати до компонента Formik (рисунок 3.2.2.3.5).

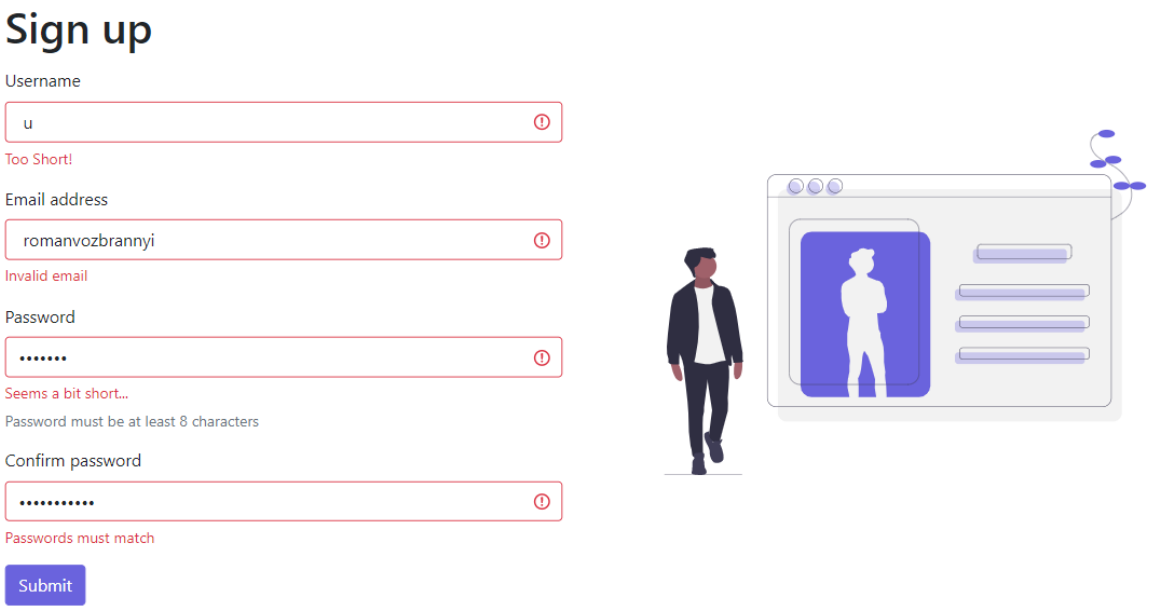


Рис. 3.2.2.3.5 Форма SignUp з відображенням помилок валідації

При завантаженні застосунку потрібно відновити сесію, надіславши запит до сервера для отримання інформації про авторизованого користувача. За це відповідає функція fetchCurrentUser (рисунок 3.2.2.3.6), яка викликається після заантаження головного компонента застосунку (App). В заголовку запиту передається токен, який отримується з localStorage браузера функцією getTocken.

```
export const fetchCurrentUser = () => {
    return (dispatch, getState) => {
        dispatch(authLoading( bool: true));
        api.get( url: '/auth/me',  config: {
            headers: {
                'Accept': 'application/json',
                'Authorization': `Bearer ${getToken()}`
            },
        }) Promise<AxiosResponse<any>>
            .then((response : AxiosResponse<any> ) => {
                dispatch(authUser(response.data.user));
            }) Promise<void>
            .catch(() => {
                dispatch(logoutUser());
            }) Promise<void>
            .finally( onFinally: () => {
                dispatch(authLoading( bool: false));
            })
    }
};
```

Рис. 3.2.2.3.6 Функція для отримання даних авторизованого користувача

Керування ролями консультанта та користувача відбувається лише на клієнтській стороні. Для переходу між інтерфейсами консультанта (Додаток Л) та користувача (Додаток А) в меню застосунку були створені відповідні кнопки (рисунок 3.2.2.3.7). Для виходу з системи, користувачу потрібно натиснути відповідну кнопку на сторінці профілю (Додаток И).

Рис. 3.2.2.3.7 Меню з кнопками перемикання ролі користувача

Для контролю доступу користувачів до маршрутів за ролями, було створено компонент ProtectedRoute (рисунок 3.2.2.3.8). До нього передається компонент для відображення, ролі, яким доступний перехід за цим маршрутом, шлях для редіректу у разі невідповідності ролі.

```
export const ProtectedRoute = ({ component: Component, roles, redirectPath :string = '/', ...rest }) => {
    const currentUser = useSelector( selector: state => state.auth.user);

    return (
        <Route {...rest} render={props => {
            if (!currentUser || (roles?.length && !roles.includes(currentUser?.role))) {
                // role not authorised so redirect to home page
                return <Redirect to={{ pathname: redirectPath}} />
            }

            // authorised so return component
            return <Component {...props} />
        }} />
    )
}
```

Рис. 3.2.2.3.8 Компонент ProtectedRoute

3.2.2.4 Оплата LiqPay

Для поповнення рахунку в застосунку було створено сторінку підтвердежння замовлення (Додаток К), на якій розміщено форму для вводу суми поповнення (рисунок 3.2.2.4.1). Після введення суми попонення, виконується обчислення суми поповнення разом з комісією LiqPay.

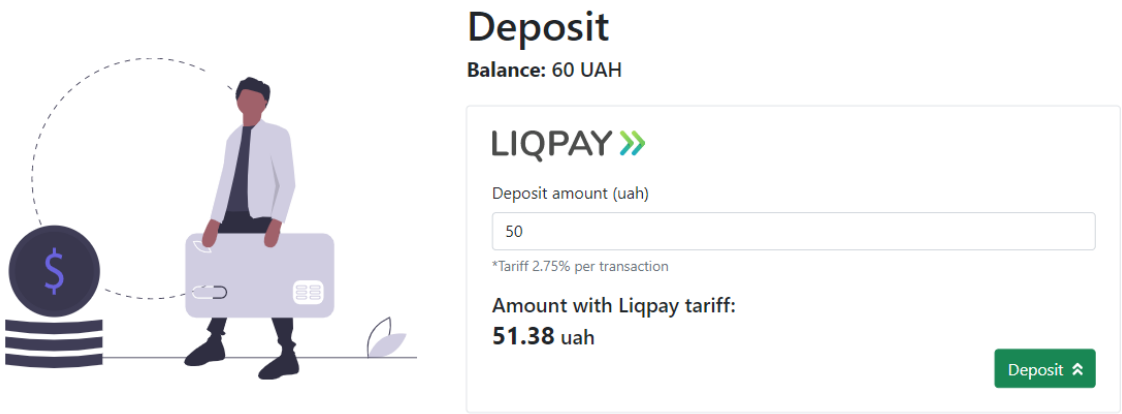


Рис. 3.2.2.4.1 Форма для поповнення рахунку

Після натискання кнопки сабміту, на сервер надсилається запит з інформацією про суму поповнення для отримання форми liqpay. Після отримання відповіді, користувачу відображається модальне вікно з формою оплати LiqPay (рисунок 3.2.2.4.2), отриманою з api.

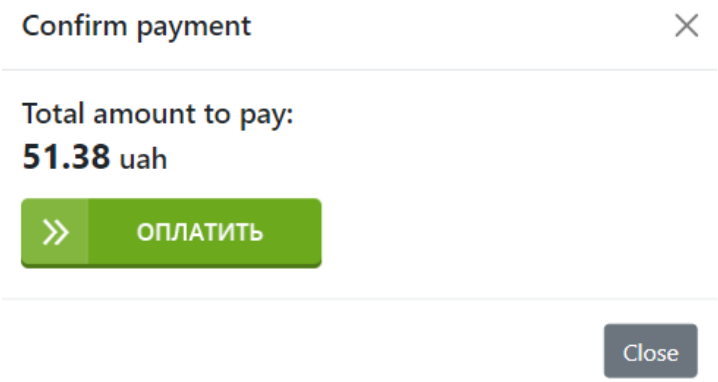


Рис. 3.2.2.4.2 LiqPay форма для оплати

При натисканні на форму оплати LiqPay відбувається перенаправлення користувача на сторінку оплати LiqPay Checout (Додаток Р). Після успішної оплати, користувачу буде запропоновано повернутися на сайт з консультаціями.

3.2.2.5 Інтерфейс консультанта

Консультант не може почати проводити консультації, якщо він не заповнив додаткову інформацію про консультанта на сторінці профілю (Додаток П) в інтерфейсі консультанта (рисунок 3.2.2.5.1). Після заповнення цієї інформації, користувач може перейти на сторінку активації консультанта (Додаток Н).

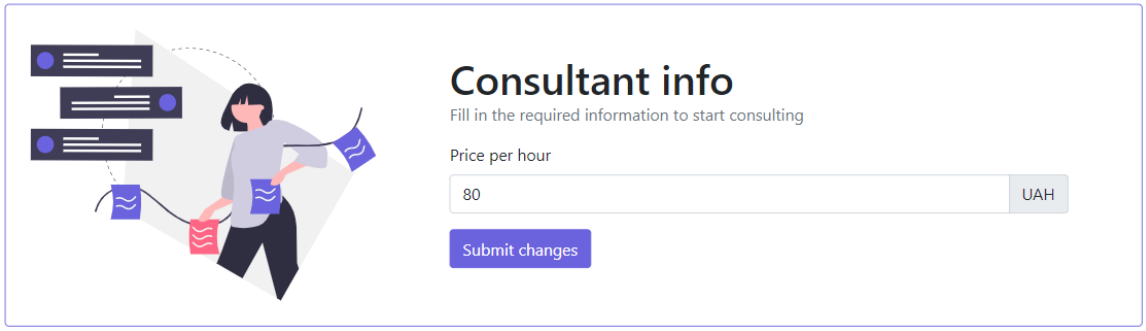


Рис. 3.2.2.5.1 Форма для введення інформації про консультанта

При переході на сторінку активації консультанта, встановлюється socket.io з'єднання (рисунок 3.2.2.5.2). Це відбувається для забезпечення можливості оновлення списку активних консультантів (Додаток Б) без перезавантаження сторінки.

```
useEffect( effect: () => {
    dispatch(fetchConsultantInfo());
    socketRef.current = io.connect(api.defaults.baseURL);
    socketRef.current.on("consultant-updated", () => {
        dispatch(fetchConsultantInfo());
        setUserInfoLoading( value: false);
    })
    socketRef.current.on("event-created", payload => {
        history.push(`/call/${payload.roomID}`)
    })
    return () => {
        handleDeactivate();
    }
}, deps: []);
```

Рис. 3.2.2.5.2 Створення socket.io з'єднання

При натисканні кнопки активації, викликається socket.io подія "activate-consultant" за допомогою функції "emit("activate-consultant", user?.id)". При цьому на сервері відбувається зміна статусу активності користувача, а в браузері відображається повідомлення про успішну активацію консультанта (рисунок 3.2.2.5.3).

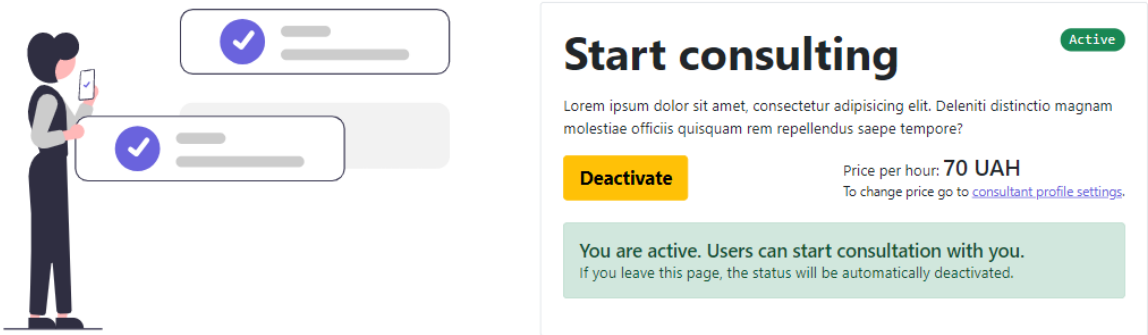


Рис. 3.2.2.5.3 Повідомлення про успішну активацію консультанта

Консультант залишається активним поки не натисне кнопку деактивації, або поки не залишить сторінку. Сервер слідкує за з’єднанням з сокетом, та автоматично змінить статус консультанта при розірванні з’єднання. Якщо користувач почне консультацію, перенаправлення консультанта на сторінку відеодзвінка відбудеться автоматично, за допомогою події socket.io "event-created", яка створюється сервером після успішного придбання консультації користувачем.

```
socketRef.current.on("event-created", payload => {
    history.push(`/call/${payload.roomID}`)
})
```

Рис. 3.2.2.5.4 Автоматичне перенаправлення на сторінку відеодзвінка

3.2.2.6 Відеоконсультація

В інтерфейсі користувача, на сторінці перегляду всіх активних консультантів (Додаток Е) також встановлюється socket.io з’єднання, для динамічного оновлення списку консультантів (рисунок 3.2.2.6.1).

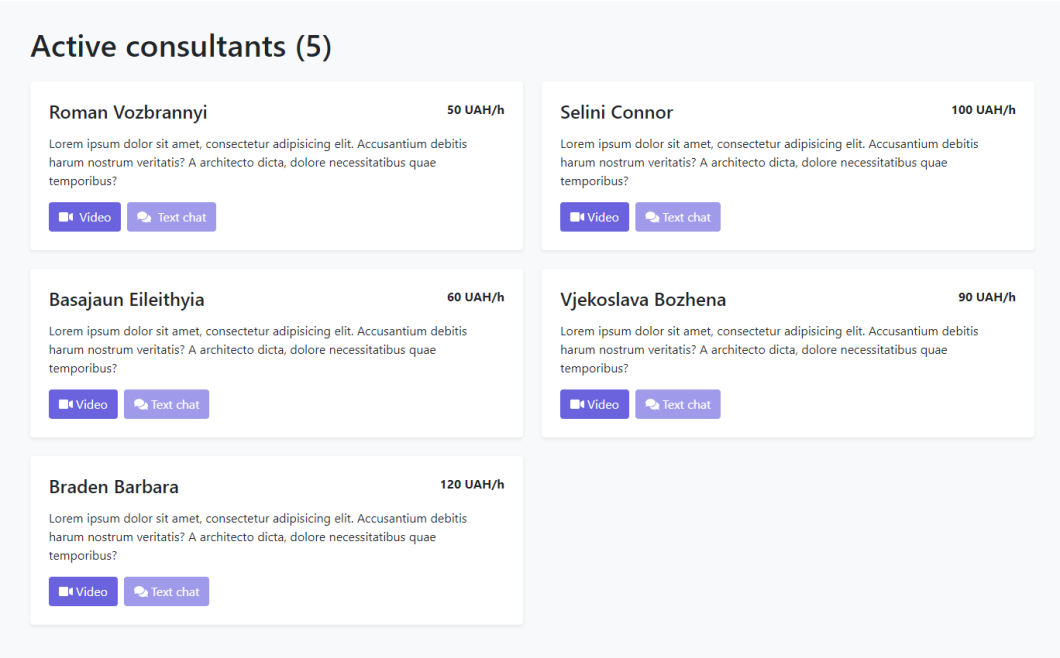


Рис. 3.2.2.6.1 Список активних консультантів

Після вибору консультанта та типу консультації, користувач переходить на сторінку підтвердження початку консультації (Додаток Ж). На цій сторінці користувач повинен обрати тривалість консультації (рисунок 3.2.2.6.2).

Рис. 3.2.2.6.2 Форма підтвердження початку консультації

При натисканні кнопки підтвердження початку консультації, автоматично починається консультація. На сторінці відеоконсультації, користувачу пропонується надати доступ до камери та мікрофона (рисунок 3.2.2.6.3).

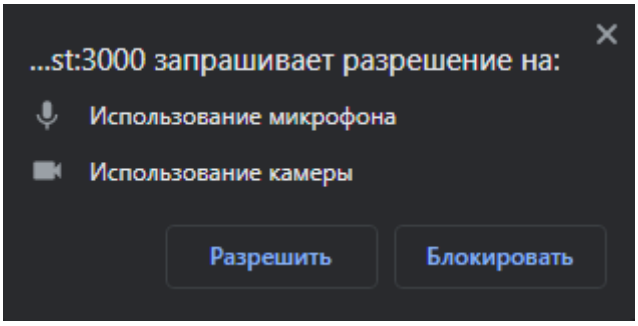


Рис. 3.2.2.6.3 Запит дозволу на використання камери та мікрофона

Запит на надання доступу відображається браузером автоматично, якщо користувач ще не надавав дозволу раніше. Для надсилання запиту на використання пристроїв та отримання потоку відео та аудіо використовується функція "navigator.mediaDevices.getUserMedia({video:

videoConstraints, audio: true})", яка приймає список девайсів, використання яких вимагається застосунком. Після успішного отримання відео та аудіо потоків, користувачу відображається власне зображення, та починається процес під'єднання до кімнати (рисунок 3.2.2.6.4).

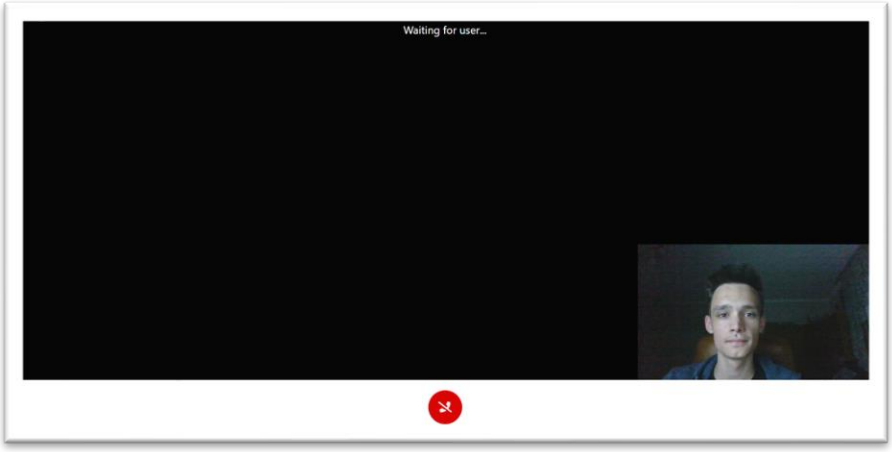


Рис. 3.2.2.6.4 Під’єднання до кімнати відеоконсультації

Для налагодження потокового обміну відео та аудіо даними використовується бібліотека simple-peer. При приєднанні користувача до кімнати, викликається функція createPeer (рисунок 3.2.2.6.5), яка приймає ідентифікатор нового користувача, ідентифікатор сокета, та медіа-стрім. Ця функція відповідає за створення об’єкта peer та створення socket.io події "sending-signal", для ініціалізації обміну даними між користувачами.

```
function createPeer(userToSignal, callerID, stream) {
    const peer = new Peer( opts: {
        initiator: true,
        trickle: false,
        stream,
    });

    peer.on("signal", signal => {
        console.log("sending-signal");
        socketRef.current.emit("sending-signal", {userToSignal, callerID, signal})
    })

    return peer;
}
```

Рис. 3.2.2.6.5 Функція createPeer

У свою чергу, новий користувач кімнати, надсилає дані та метаінформацію у відповідь. Після успішного обміну даними та метаінформацією, між користувачами починається обмін потоковими медіа-даними, а в інтерфейсі дзвінка відображається відео співрозмовника (рисунок 3.2.2.6.6).

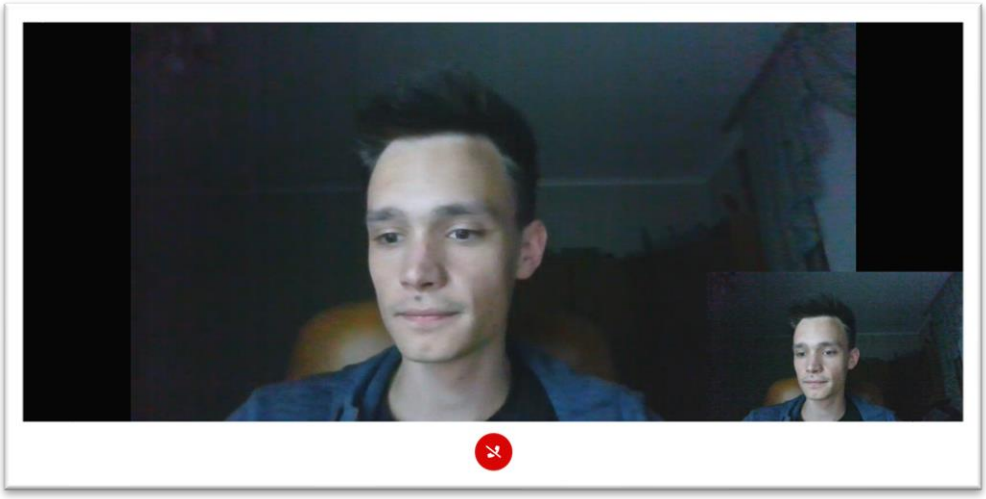


Рис. 3.2.2.6.5 Відеоконсультація

Користувач може залишити кімнату натиснувши відповідну кнопку в меню дзвінка, після цього відбудеться перенаправлення на головну сторінку.

В інтерфейсі користувача та консультанта доступна сторінка для перегляду оплачених консультацій (Додаток З) (Додаток О). Якщо з’єднання було розірвано, або вихід з дзвінка відбувся випадково, користувач та консультант може повернутися до дзвінка, натиснувши відповідну кнопку на цій сторінці.

3.3 Висновки до розділу 3

В процесі виконання роботи було визначено вимоги до функціонала майбутньої системи. Досліджено процес створення відеодзвінків та інтеграції

онлайн-оплати. В результаті виконання роботи було створено зручну та просту у використанні систему для проведення платних відеоконсультацій, яка відповідає вимогам та технічному завданню.

Висновки

Під час виконання курсової роботи був проведений аналіз функцій представлених на ринку систем для проведення онлайн-консультацій. Також, були дослідженні популярні інструменти та сервіси для інтеграції онлайн-оплати та онлайн-дзвінків при створенні власного застосунку. При розробці системи був досліджений процес створення відеодзвінків без використання сторонніх сервісів та процес інтеграції сторонньої системи для здійснення онлайн-оплати.

В результаті виконання роботи було реалізовано систему для проведення платних відеоконсультацій, яка може слугувати основою для створення нових застосунків у сфері онлайн-консультацій та онлайн-підтримки.

В майбутньому плануються допрацювання системи, шляхом додавання функціоналу для планування консультацій, обміну текстовими повідомленнями та виводу коштів з системи.

Перспективами розвитку проекту є забезпечення можливості моніторингу процесу консультування зі збереженням подій відеодзвінка для подальшої обробки скарг користувачів на надані послуги. Також, проект може бути доповнено системою рейтингу консультантів, яка допоможе забезпечити високий рівень якості консультацій. Ще одним можливим покращенням може бути можливість демонстрації екрану під час консультації.

Список використаної літератури та електронних ресурсів

1. Офіційна сторінка платформи «Doctor Online». [Електронний ресурс]. – Режим доступу: https://doctoronline.care/uk/
2. Офіційна сторінка платформи «Юристи.UA». [Електронний ресурс]. – Режим доступу: https://uristy.ua/ua
3. Офіційна сторінка платформи «QITonline». [Електронний ресурс]. – Режим доступу: https://www.qit.online/
4. Документація сервісу Twilio. [Електронний ресурс]. – Режим доступу: https://www.twilio.com/docs/video
5. Документація сервісу Vonage Video API. [Електронний ресурс]. – Режим доступу: https://tokbox.com/developer/
6. Документація сервісу LiqPay. [Електронний ресурс]. – Режим доступу: https://www.liqpay.ua/documentation/uk/api/home/
7. Документація сервісу Stripe. [Електронний ресурс]. – Режим доступу: https://stripe.com/docs/payments/checkout
8. Документація бібліотеки React.js. [Електронний ресурс]. – Режим доступу: https://uk.reactjs.org/
9. Документація бібліотеки Redux. [Електронний ресурс]. – Режим доступу: https://redux.js.org/
10. Документація бібліотеки Styled Components. [Електронний ресурс]. – Режим доступу https://styled-components.com/docs
11. Документація бібліотеки Formik. [Електронний ресурс]. – Режим доступу: https://jaredpalmer.com/formik/
12. Документація бібліотеки Simple-peer. [Електронний ресурс]. – Режим доступу: https://github.com/feross/simple-peer
13. Досвід використання WebRTC (лекція Яндекса). [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/yandex/blog/419951/

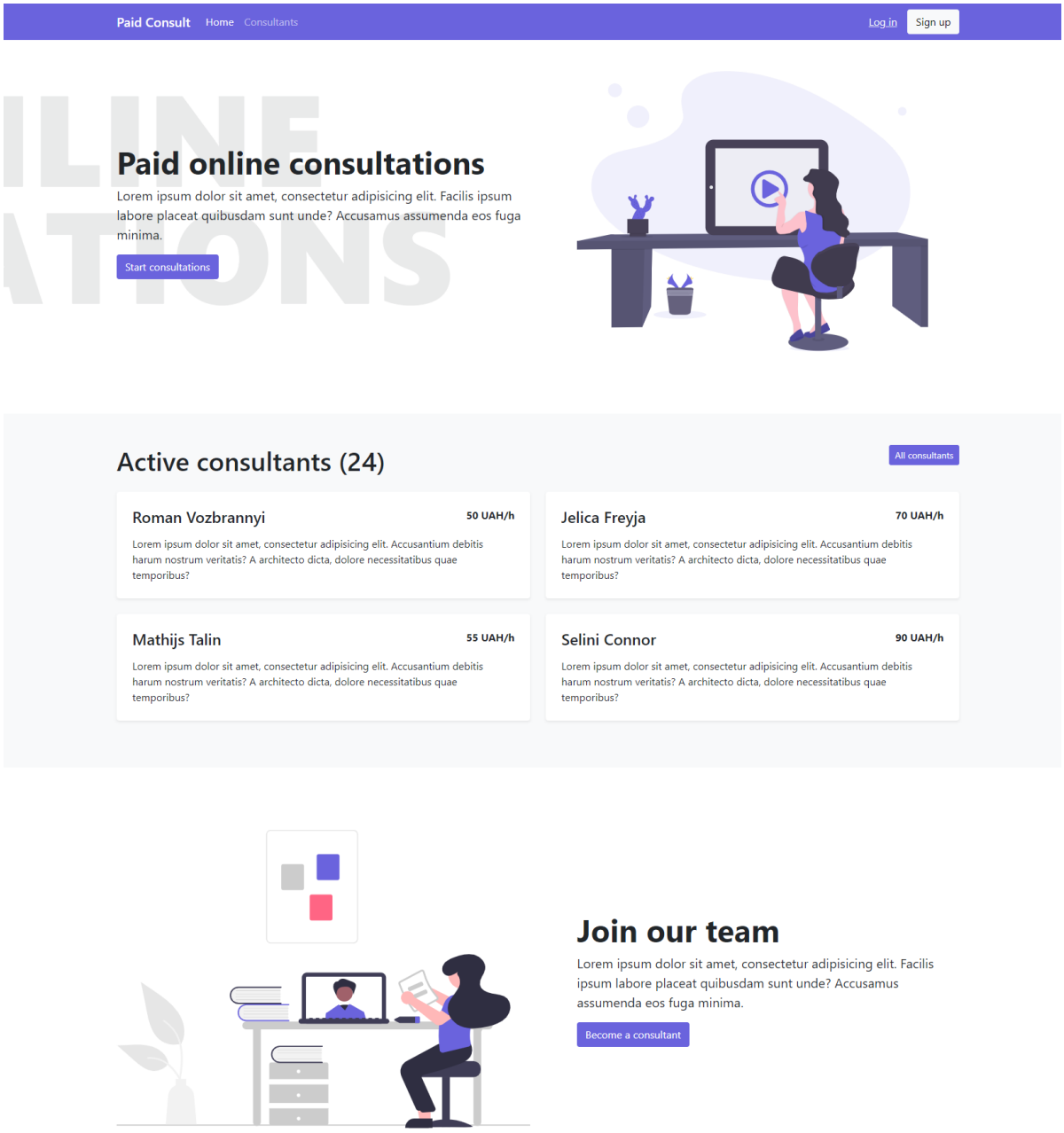
14. Документація бібліотеки socket.io. [Електронний ресурс]. – Режим доступу: https://socket.io/docs/v4/
15. Документація бібліотеки Node.js. [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/
16. Документація бібліотеки Express. [Електронний ресурс]. – Режим доступу: http://expressjs.com/en/api.html
17. Документація бібліотеки Sequelize. [Електронний ресурс]. – Режим доступу: https://sequelize.org/master/
18. Bulletproof node.js project architecture. [Електронний ресурс]. – Режим доступу: https://dev.to/santypk4/bulletproof-node-js-project-architecture-4epf
19. Route-Controller-Service structure for ExpressJS. [Електронний ресурс]. – Режим доступу: https://sodocumentation.net/node-js/topic/10785/route-controller-service-structure-for-expressjs
20. Документація бібліотеки React Router. [Електронний ресурс]. – Режим доступу: https://reactrouter.com/web/guides/quick-start
21. React Router v5 - Fix for redirects not rendering when using custom history. [Електронний ресурс]. – Режим доступу: https://jasonwatmore.com/post/2020/10/22/react-router-v5-fix-for-redirects-not-rendering-when-using-custom-history

Додатки

Додаток А

(Обов’язковий)

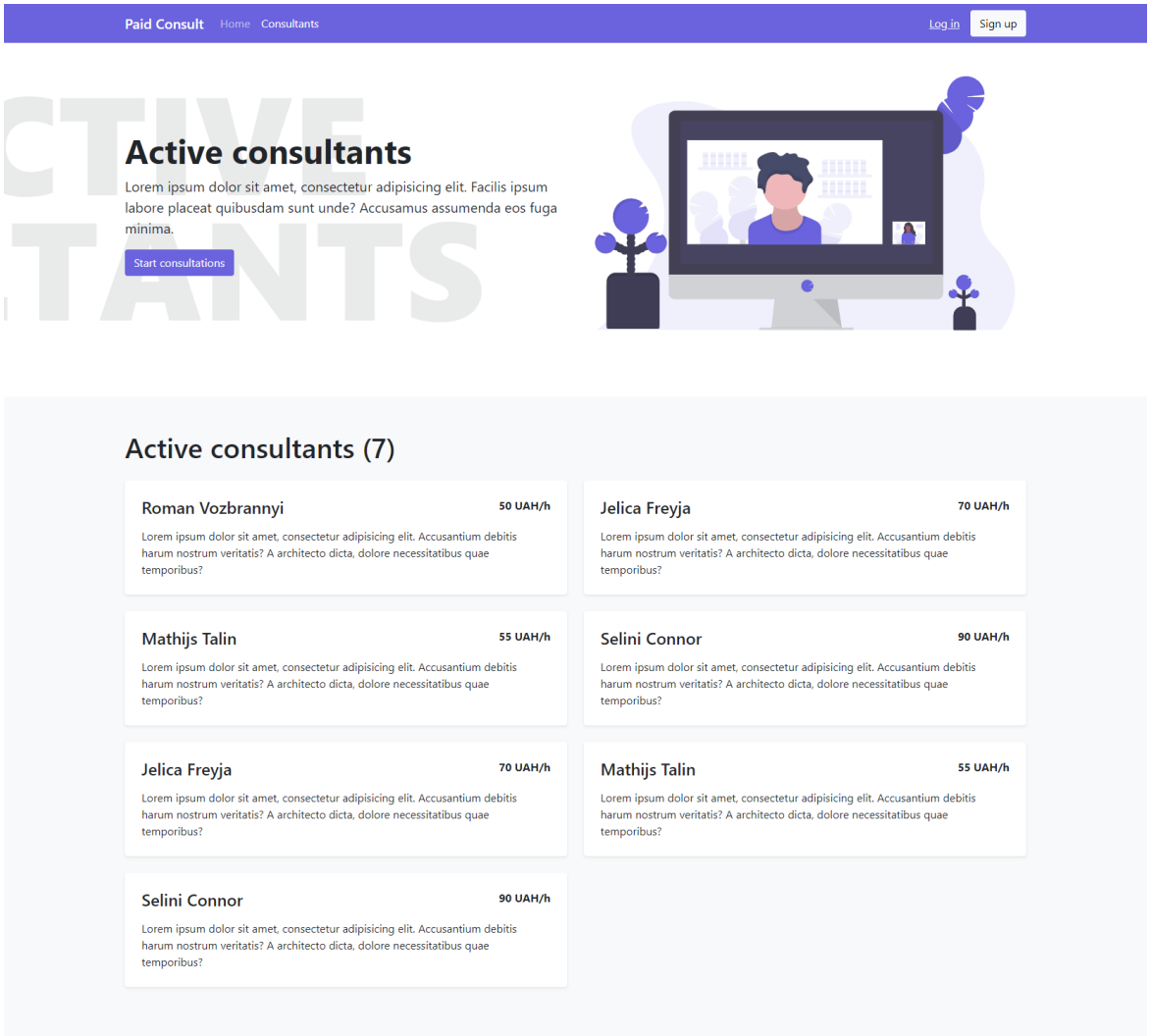
Головна сторінка для гостей сайту



Додаток Б

(Обов’язковий)

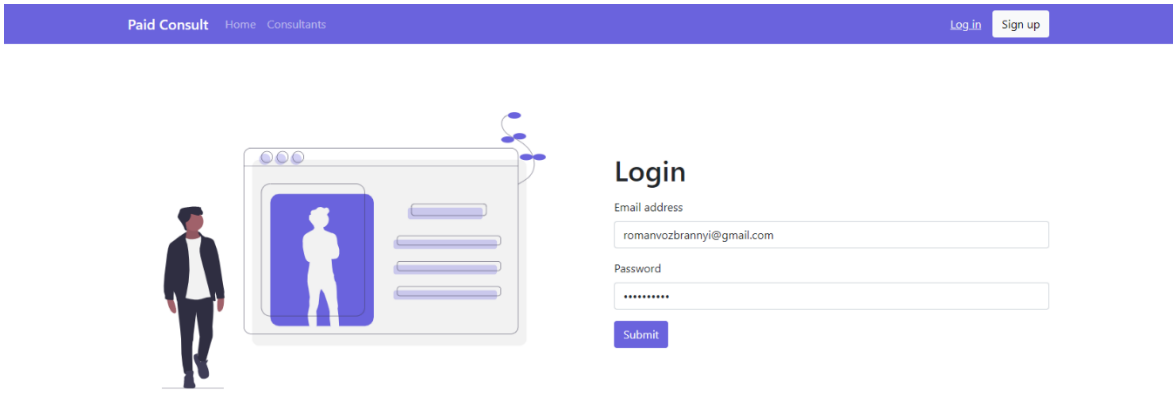
Сторінка зі списком активних консультантів для гостей сайту



Додаток В

(Обов’язковий)

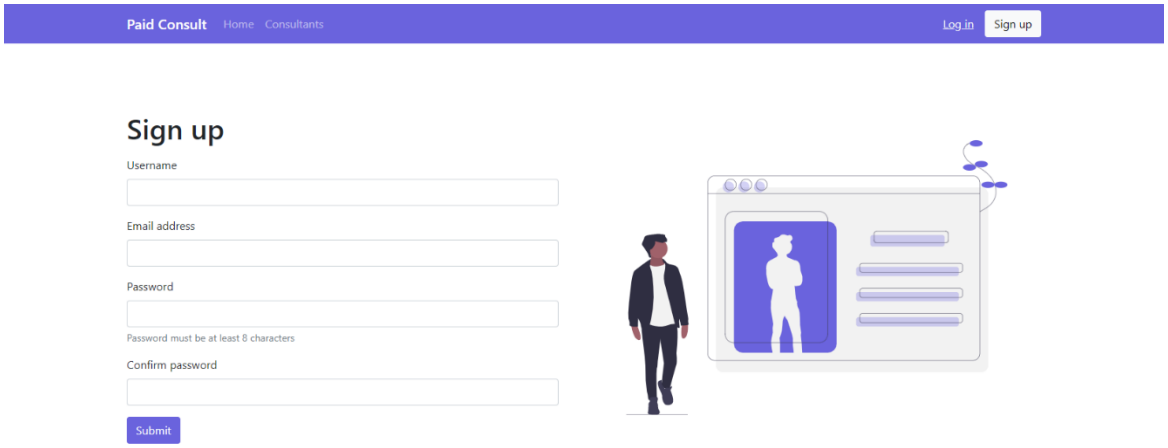
Сторінка Login



Додаток Г

(Обов’язковий)

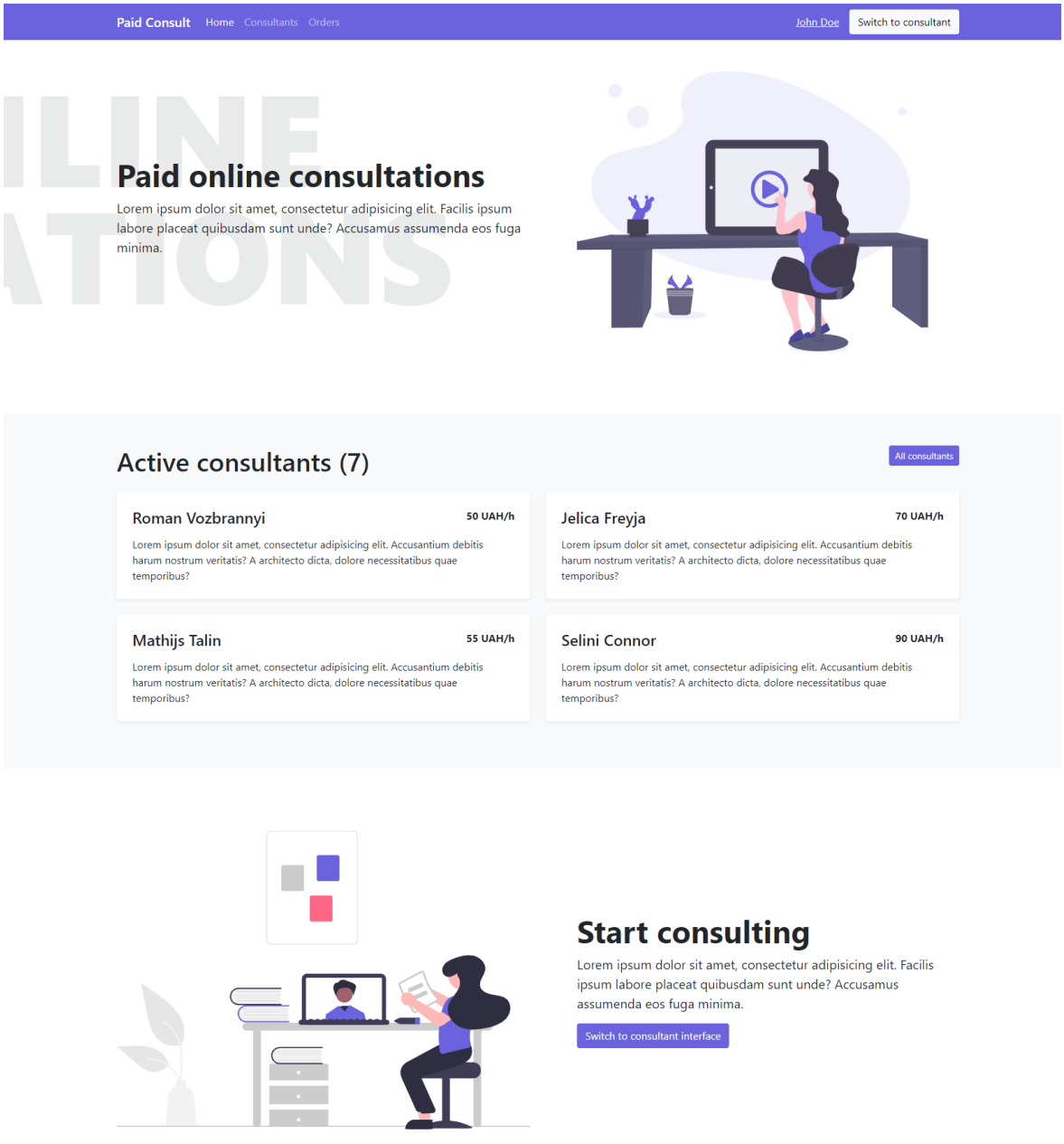
Сторінка Sign up



Додаток Д

(Обов’язковий)

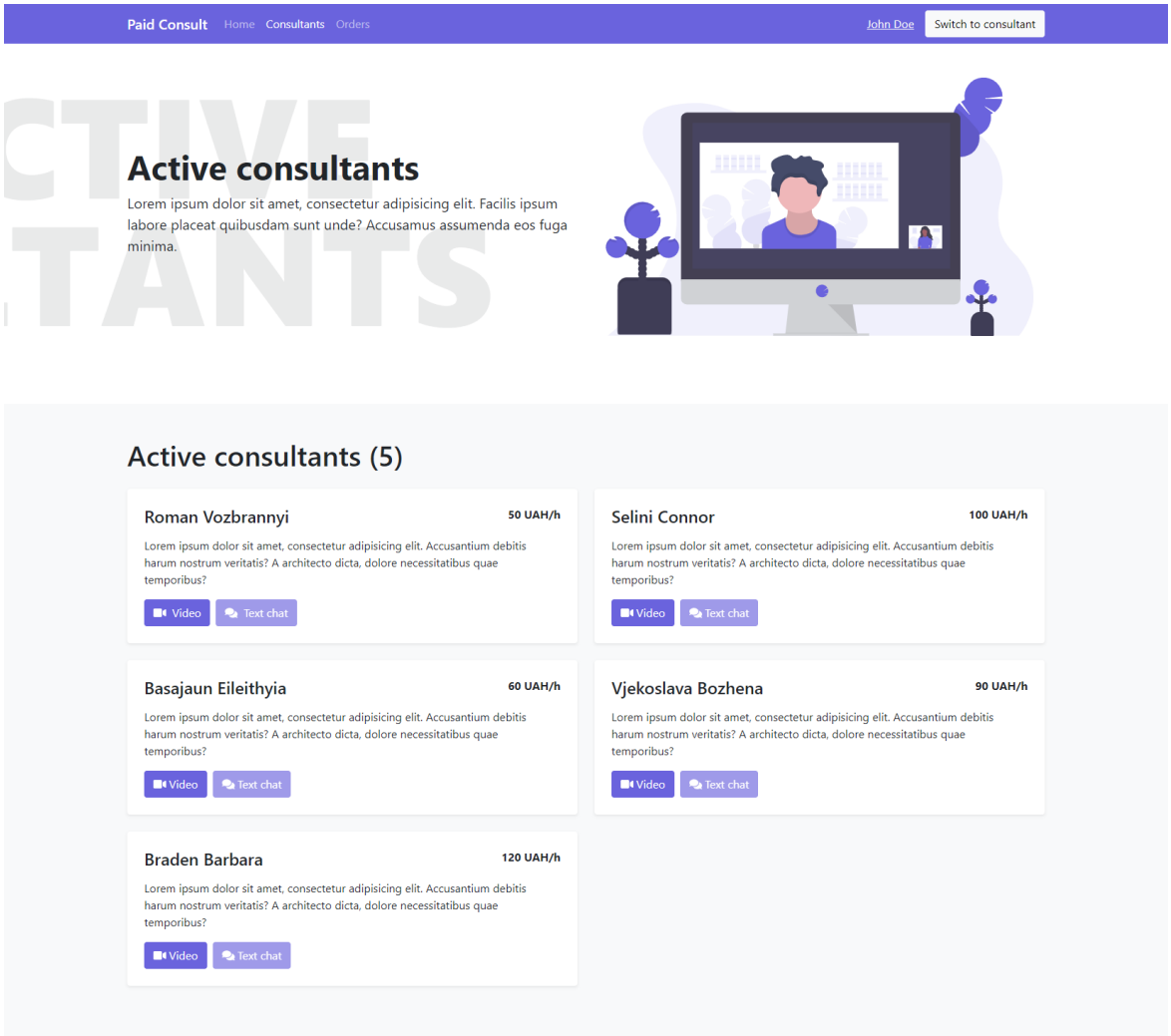
Головна сторінка для авторизованого користувача



Додаток Е

(Обов’язковий)

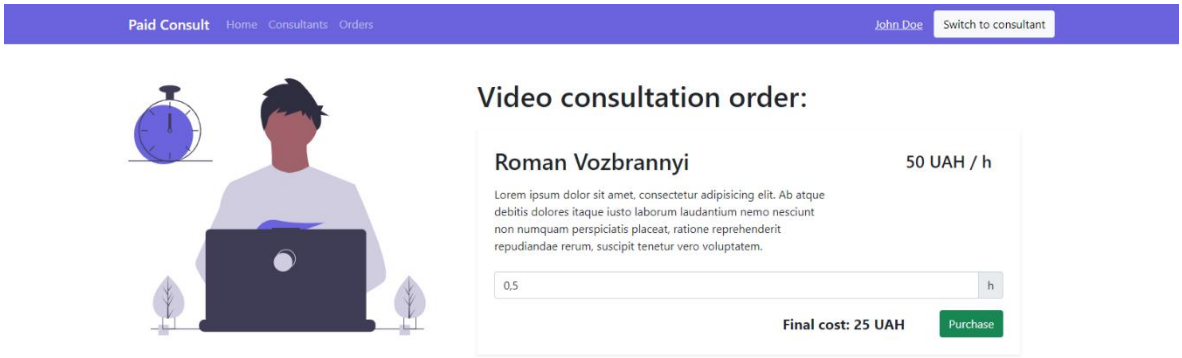
Сторінка зі списком активних консультантів для авторизованих користувачів



Додаток Ж

(Обов’язковий)

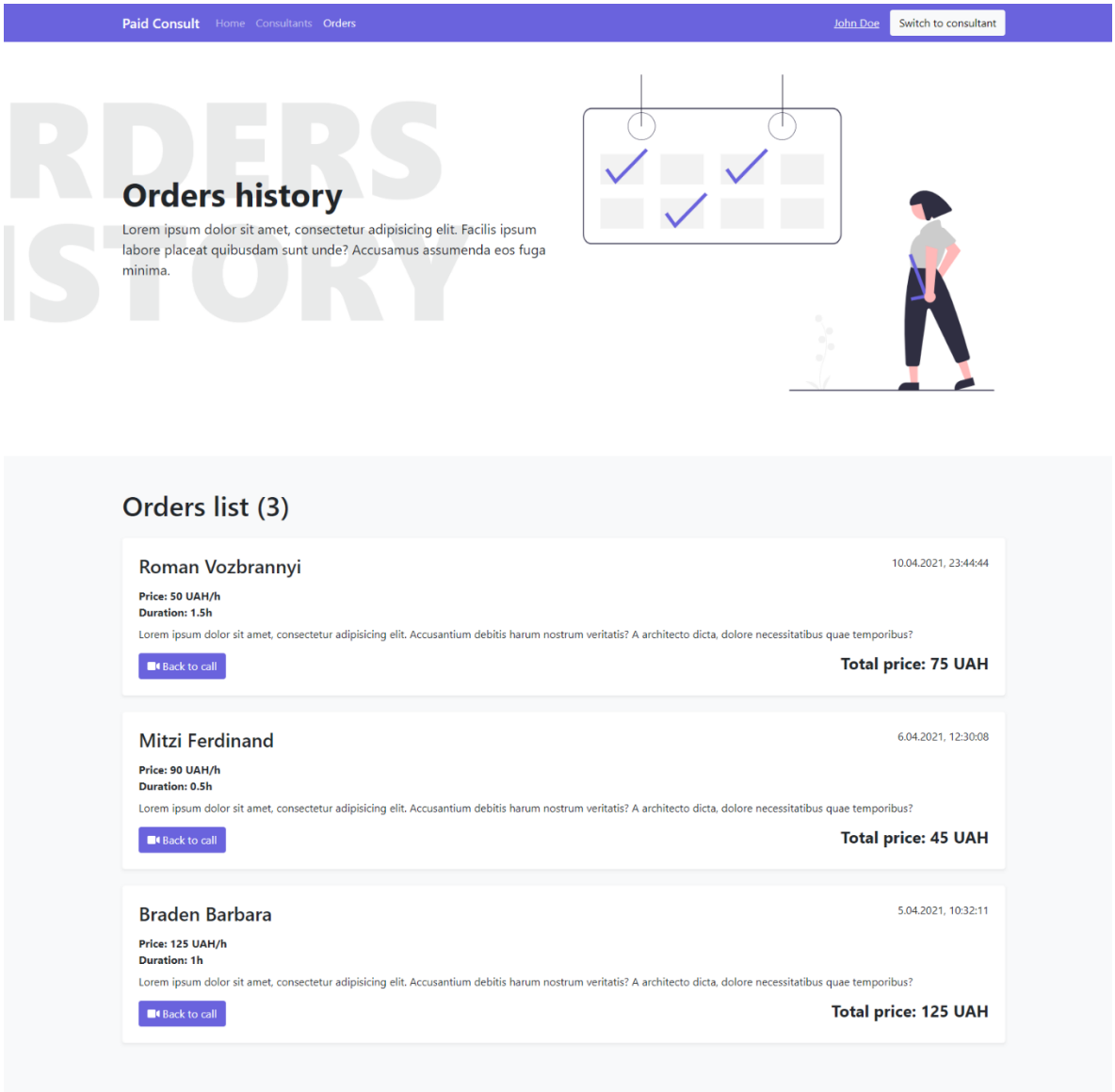
Сторінка підтвердження початку консультації



Додаток 3

(Обов’язковий)

Сторінка зі списком оплачених консультацій



Додаток И

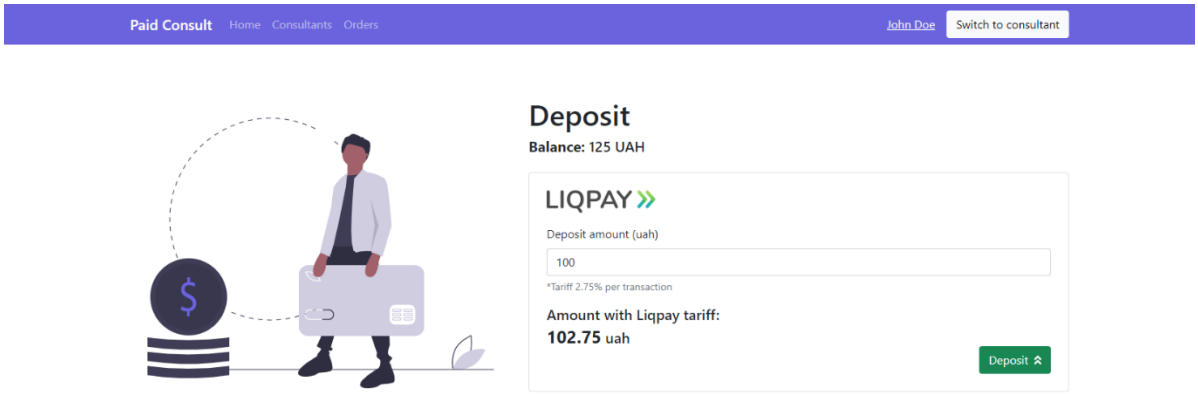
(Обов’язковий)

Сторінка профілю користувача

Додаток К

(Обов’язковий)

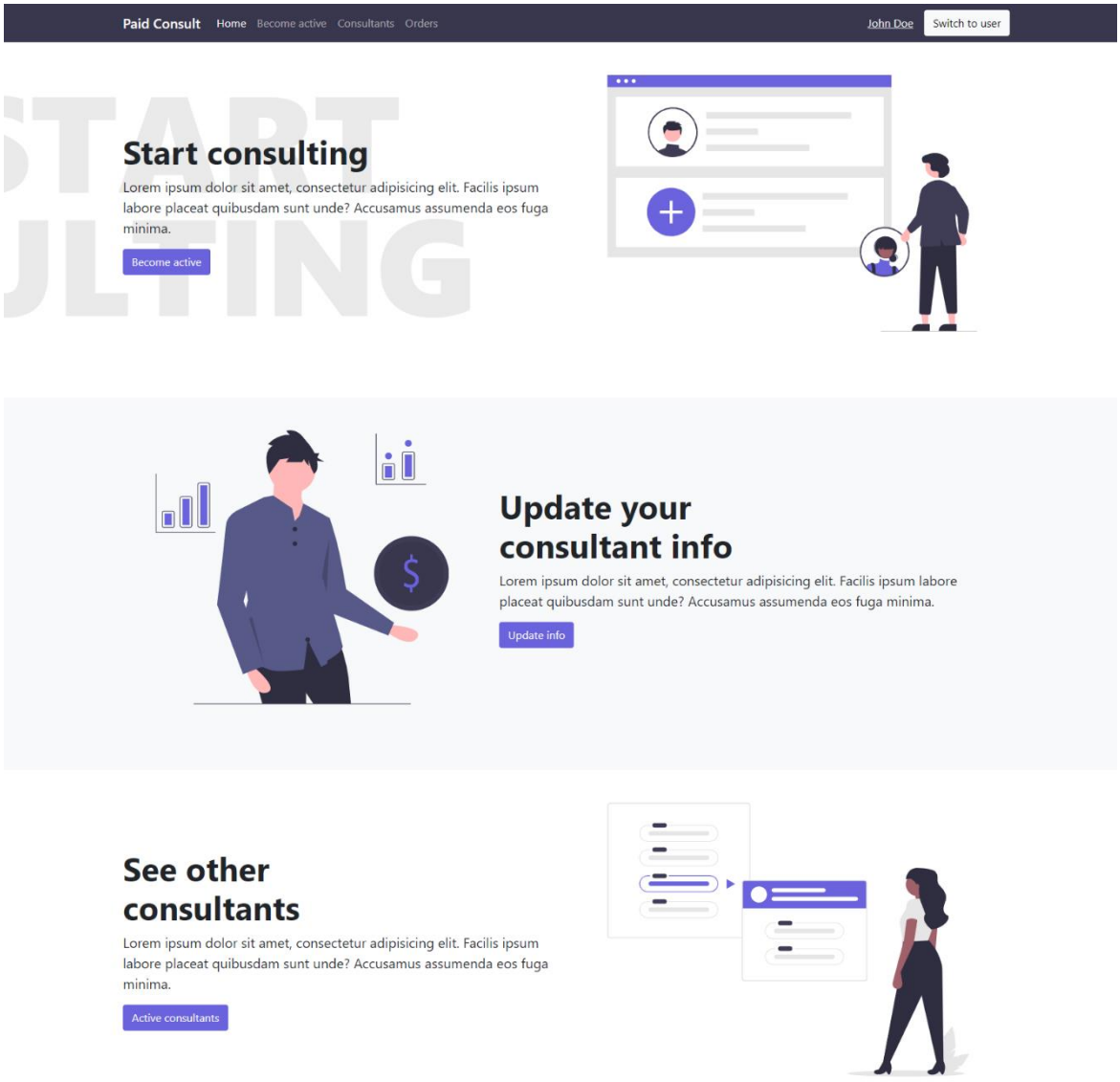
Сторінка поповнення рахунку



Додаток Л

(Обов’язковий)

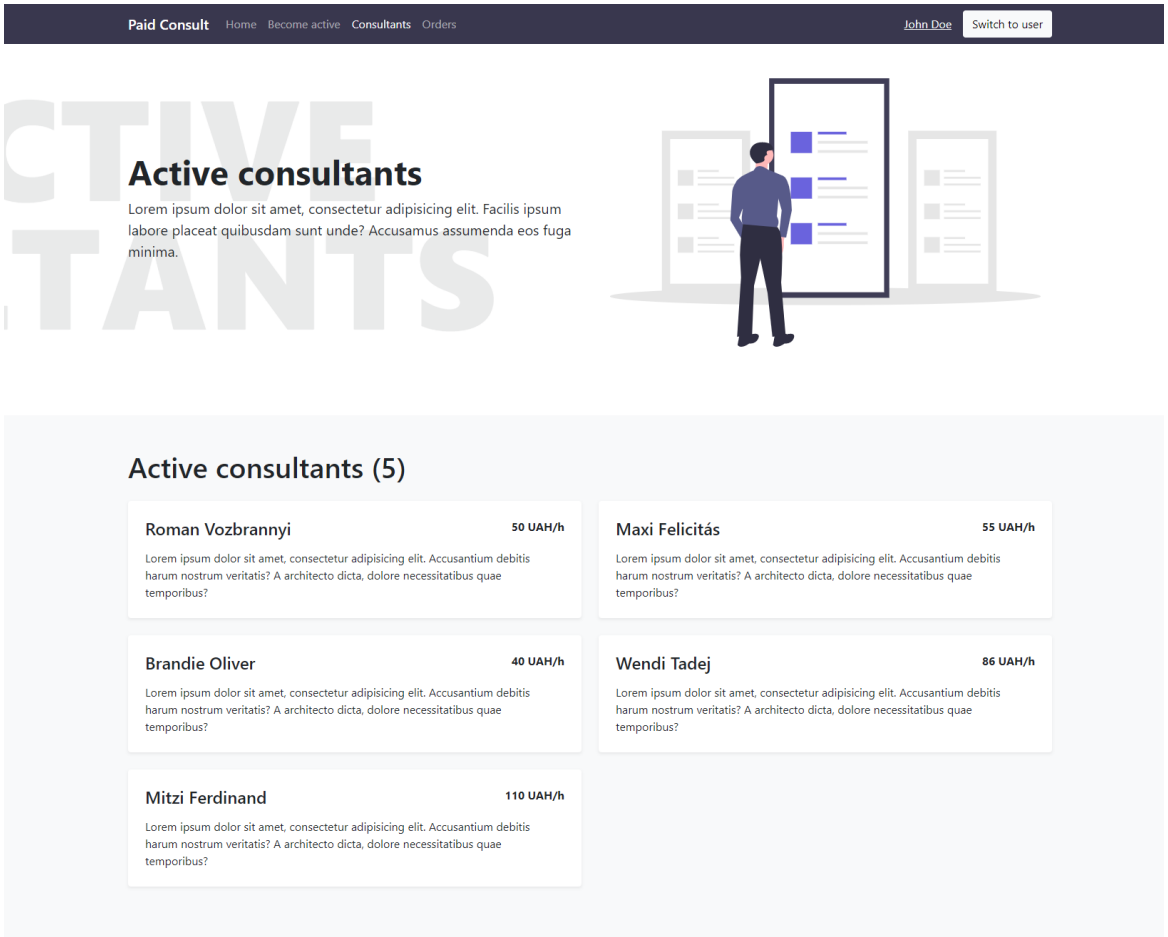
Головна сторінка інтерфейсу консультанта



Додаток М

(Обов’язковий)

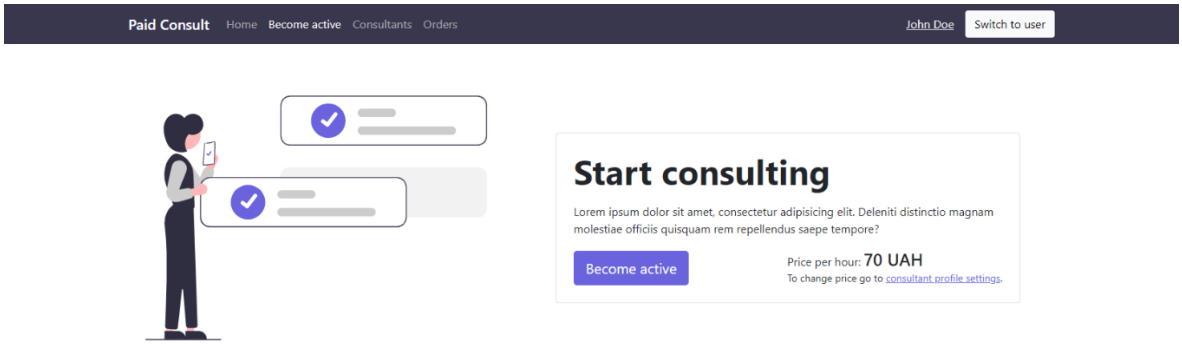
Сторінка зі списком активних консультантів в інтерфейсі консультанта

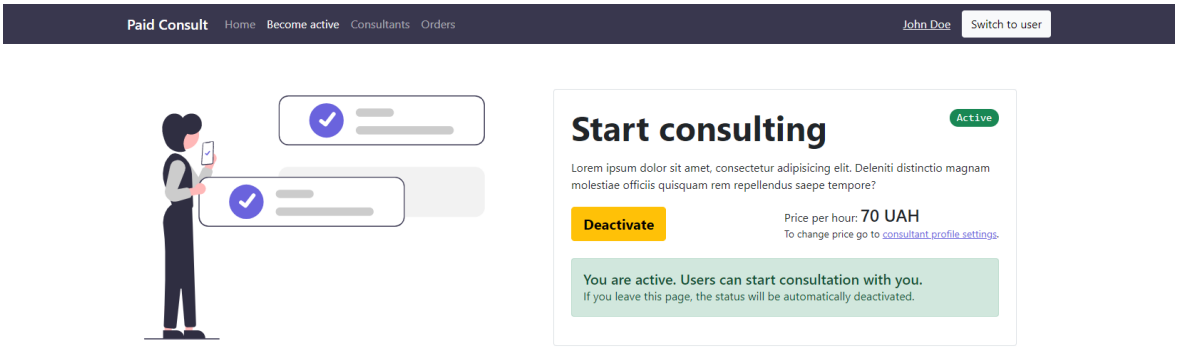


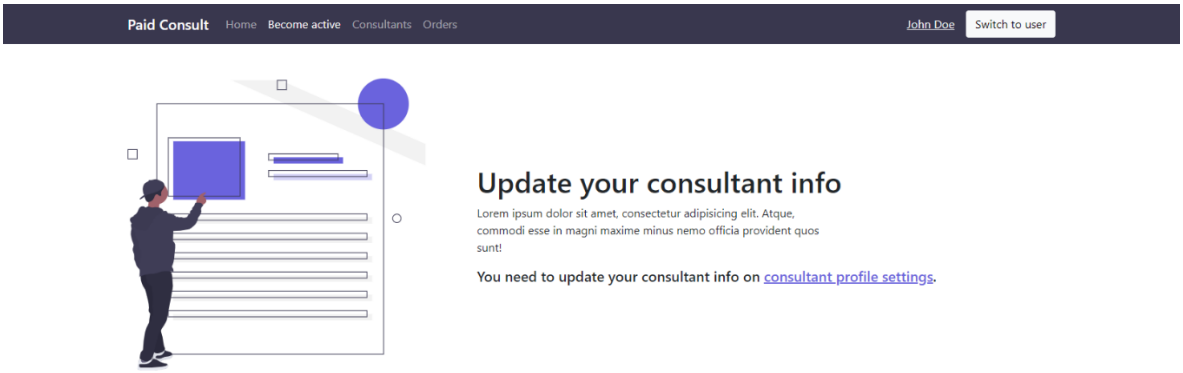
Додаток Н

(Обов’язковий)

Сторінка для зміни статусу активності в інтерфейсі консультанта



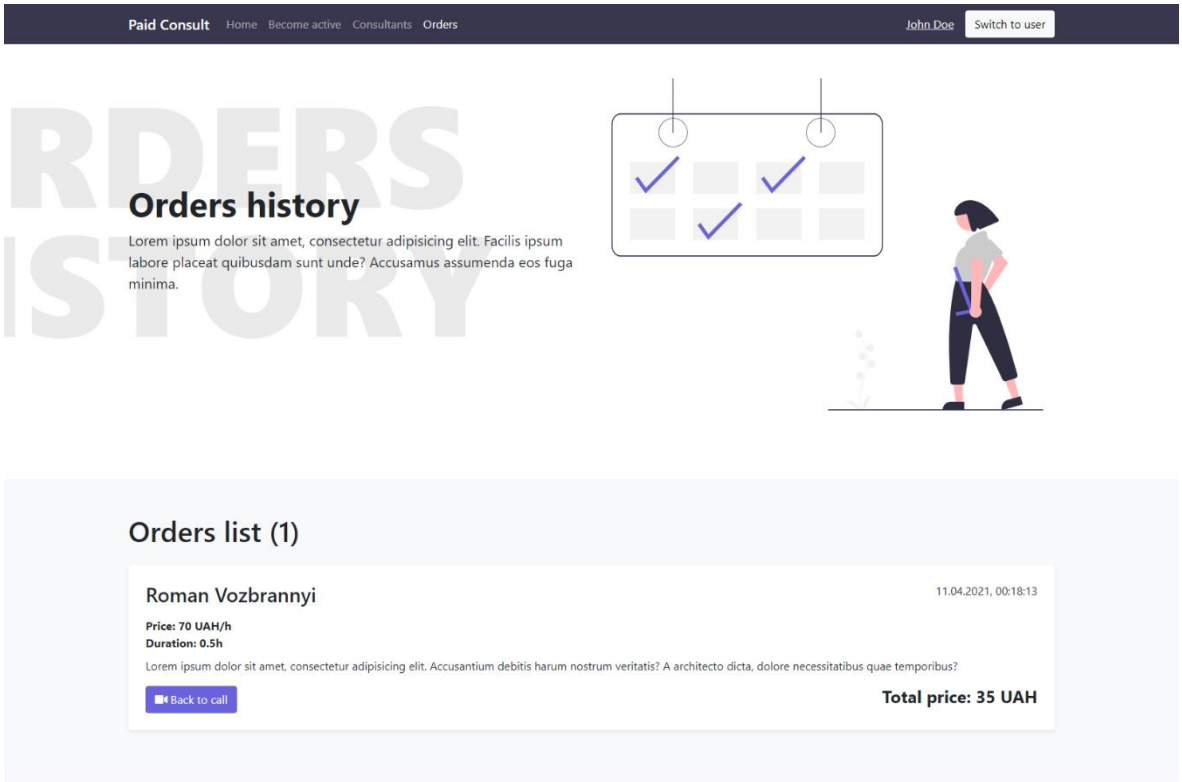




Додаток О

(Обов’язковий)

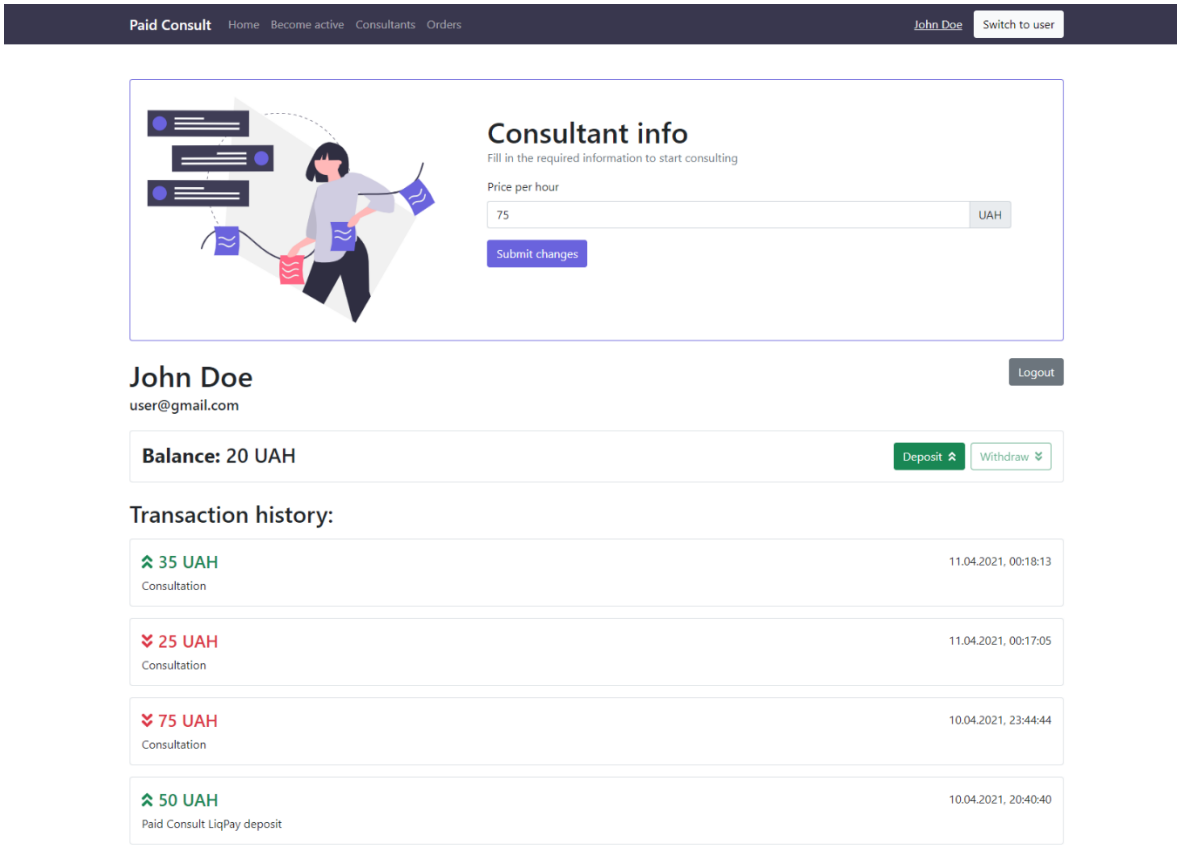
Сторінка зі списком оплачених консультацій в інтерфейсі консультанта



Додаток П

(Обов’язковий)

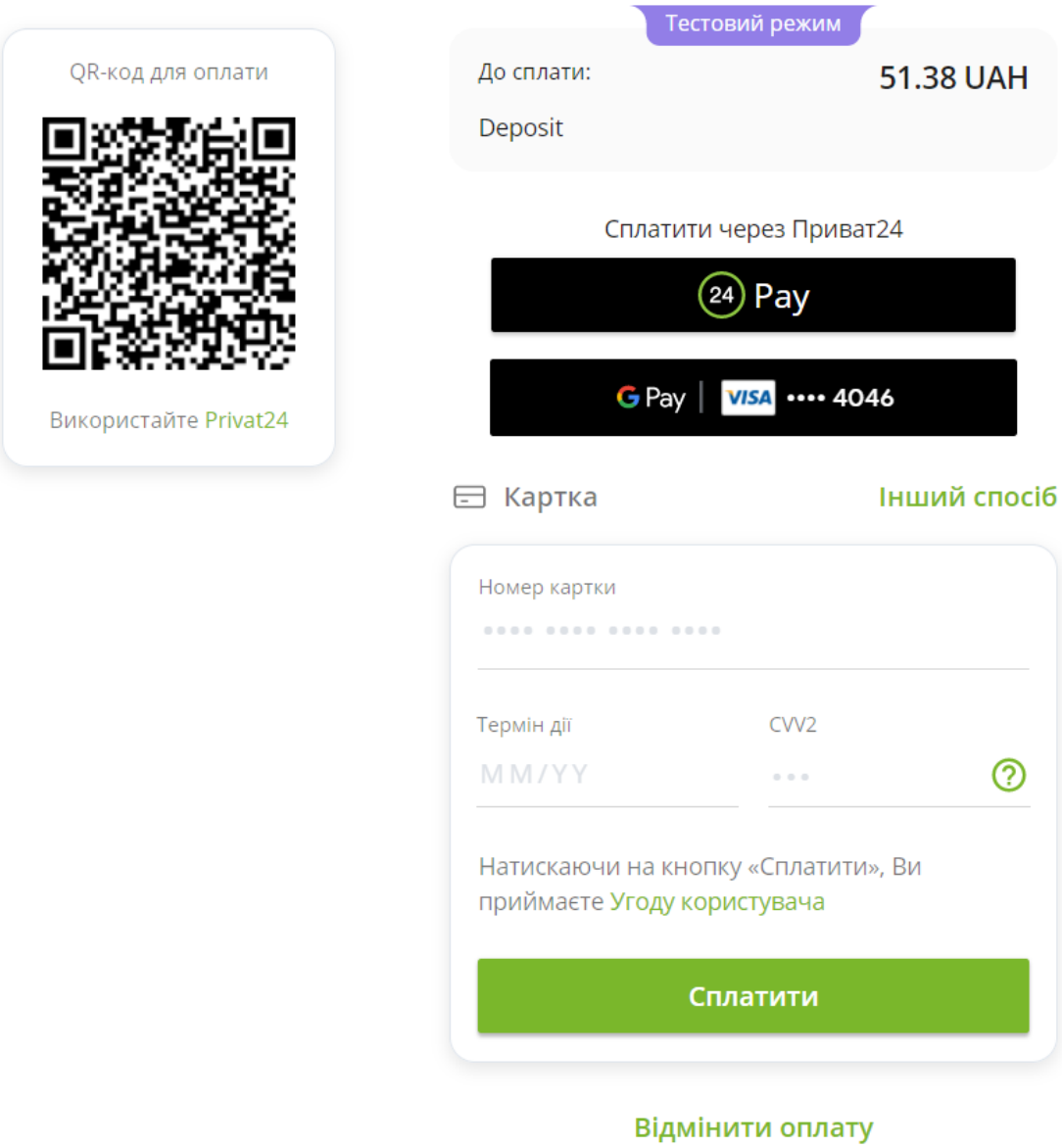
Профіль консультанта



Додаток Р

(Обов’язковий)

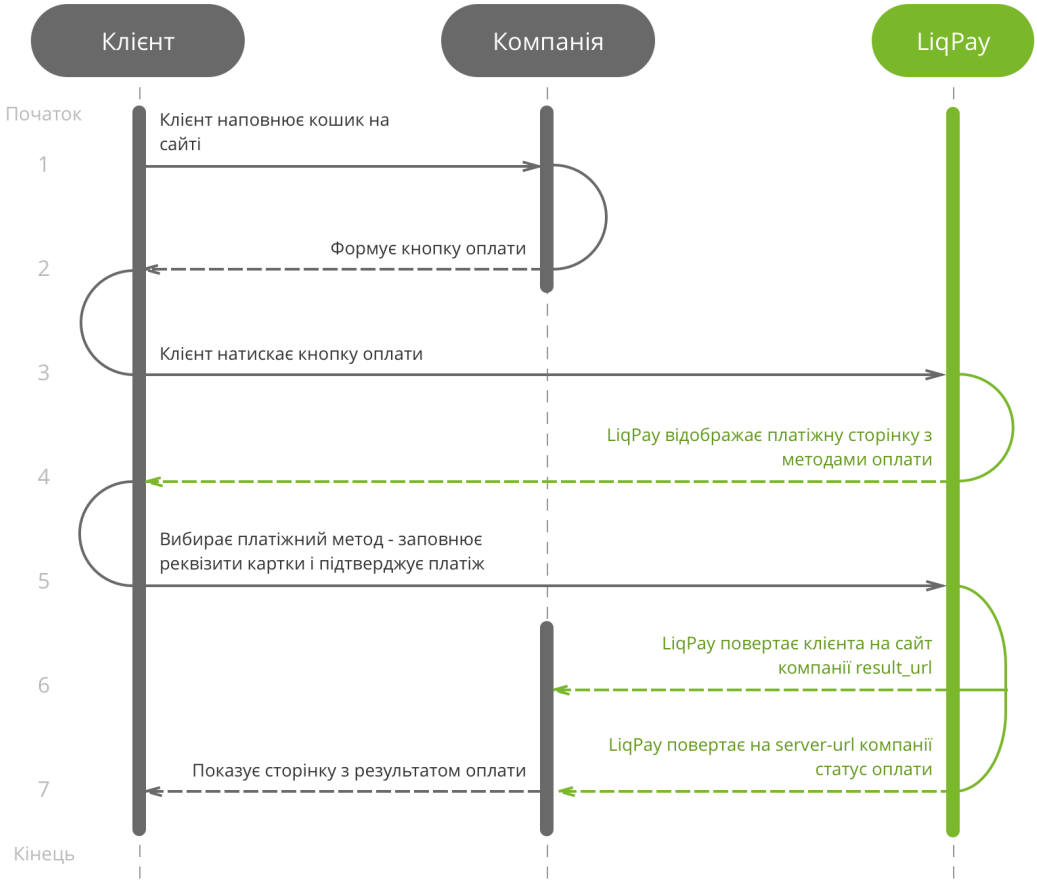
Сторінка LiqPay Checkout



Додаток С

(Обов’язковий)

Схема роботи LiqPay checkout



Додаток Т

(Обов’язковий)

Сторінка оплати Stripe Checkout