МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА САЙТУ-ПОРТФОЛІО ДЛЯ ХУДОЖНИКІВ З ЕЛЕМЕНТАМИ СОЦІАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ LARAVEL ТА REACT»**

Виконала: студентка 2 курсу, групи 8.1211іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.І. Сичова
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)
« ____ » _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Сичовій Валерії Іванівні
(прізвище, ім'я та по-батькові)

1. Тема роботи «Розробка сайту-портфоліо для художників з елементами соціальної мережі з використанням Laravel та React»

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » травня 2022 року № 500-с

2. Строк подання студентом роботи 02.12.2022

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Збір даних про предметну область.
2. Розробка моделі програмного доданку.
3. Вибір технологій.
4. Реалізація доданку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 10.05.2022

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.07.2022	
2.	Збір вихідних даних.	22.07.2022	
3.	Обробка методичних та теоретичних	07.09.2022	
	джерел.		
4.	Розробка першого та другого розділу.	10.10.2022	
5.	Розробка третього розділу.	24.11.2022	
6.	Оформлення та нормоконтроль	02.12.2022	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	16 .12.2022	

Студент _____ В.І. Сичова
(підпис) (ініціали та прізвище)

Керівник роботи _____ М.О. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка сайту-портфоліо для художників з елементами соціальної мережі з використанням Laravel та React»: 50 с., 20 рис., 17 джерел, 2 додатка.

БАЗА ДАНИХ, КЛІЄНТ, СЕРВЕР, СХЕМА, ФРЕЙМВОРК, GRAPGQL.

Об’єкт дослідження – багатокористувацькі онлайн-майданчики пошуку роботи та їх створення.

Мета роботи: розробити серверну та клієнтську частини сайту-портфоліо для художників з елементами соціальної мережі з використанням бекенд фреймворку Laravel та фронтенд бібліотек React.

Метод дослідження: аналітичний (огляд існуючих подібних майданчиків та технологій для їх створення) практичний (створення веб-доданку).

У кваліфікаційній роботі розглянуто процес створення онлайн-майданчика пошуку роботи із розміщенням портфоліо на особистій сторінці.

Пошук роботи або робітника завжди є актуальною проблемою суспільстві. Методи цього пошуку еволюціонують разом із технологіями людини, і розвиток цей, здається буде нескінченним, як не бачиться кінця й для товарно-грошових відносин між людьми. Саме через це розробляємий сайт матиме два типи користувачів: роботодавці та шукачі роботи, але систему орієнтовано більшою мірою саме на художників, як шукачів роботи.

SUMMARY

Master’s qualifying paper «Development of the Artists' Portfolio Site with Elements of a Social Network using Laravel and React»: 50 pages, 20 figures, 17 references, 2 supplements.

DATABASE, CLIENT, SERVER, SCHEM, FRAMEWORK, GRAPGQL.

The object of research is multi-user online job search sites and their creation.

Aim of the study: to develop a server and client portfolio site for artists with social network elements using the Laravel backend framework and React frontend libraries.

Methods of research: analytical (review of existing similar interfaces and technologies for their creation) practical (creation of a web interface).

The qualifying work research the process of creating an online job search site with the placement of a portfolio on a personal page.

Finding a job or a worker is always a pressing problem for society. The methods of this finding evolve together with human technologies, and development seems to be endless, just as there is no end in sight for commodity-money relations between people. It is because of that the developed site will have two types of users: employers and job seekers, but the system is oriented to a greater extent precisely to artists as job seekers.

.

ЗМІСТ

ВСТУП

Дистанційною роботою у сьогоденні нікого не здівувати, які фрілансом. Пошук роботи теж дуже природно перейшов у майже безконтактне (фізично) середовище – інтернет. Майданчиків для розміщення відкритих вакансій вже є чимало; сервісів, що дозволяють опублікувати своє портфоліо – також. Навіть є цілі соціальні мережі, орієнтовані саме на роботодавців та шукачів роботи, але в усьому є свої складнощі і недоліки. Тема цієї роботи сформулювалася після аналізу таких майданчиків, орієнтованих на художників, різноманітних дизайнерів та інших гравців цієї сфери. Нажаль, початківці, щойно закінчивши складну програму свого ВИШу, або ще студенти, опиняються у скрутному становищі. Їх роботи, виконані за програмою, юридично належать ВИШу та не можуть бути використані, а своїх сторонніх проєктів високої ланки було ще замало, щоб отримати хоч якусь відомість. Тож першим кроком вони подаються до інтернету, де їх чекають більш аматорські майданчики для викладання своїх робіт (соціальні мережі на кшталт Instagram, Twitter або майже забутий Tumbler), більш професійні (наприклад, Artstation) та декілька фріланс-сайтів де є вакансії-запити для художників – типу Upwork або Frelans.ua. І все це орієнтовано на роботодавця або на вже «набившего руку» художника (запит на Upwork може закритись ще до того, як початківець, взявшийся за нього, завершить роботу). Тому за мету роботи взято розробку ще одної альтернативи, яка буде орієнтована виключно на аудиторію художників і дизайнерів.

Створення веб-доданку такого типу, з декількома ролями користувачів, вимагає ретельного вибору інструментів. Усталені фреймвоки та спеціалізовані бібліотеки сильно спрощують процес розробки, тому для цієї роботи обрано вже давно зарекомендувавший себе фреймворк Laravel у зв’язці із базою даних MySQL. Для фронт-енд сторони буде використано набір бібліотек React, який є дуже зручним для структурованої верстки сторінок та дозволяє змінювати

інформацію на сторінці не перезавантажуючи її. Для запитів, замість звичного API використаємо GraphQL у спеціально заточеному під обраний фреймворк інструменті Lighthous.

Тематика даної роботи дозволяє глибше опанувати сучасні технології для розробки складних веб-доданків.

Основним напрямком цієї роботи є створення веб-сервісу, який дозволить художникам викладати своє портфоліо просто у профілі, а роботодавцям – обирати та напряму звертатись до тих фахівців, чий стиль та напрямок робіт їм підходять.

Для досягнення поставлених цілей було сформовано наступні етапи роботи:

– розглянути вже існуючі альтернативні майданчики пошуку роботи та розміщення портфоліо;

– розробити модель роботи доданку та методи зв’язку між його серверною та клієнтською частинами;

– обрати технології для створення серверної та клієнтської частини доданку;

– розробити доданок та відладити його роботу;

– протестувати розроблений доданок.

Практичне значення проведеної роботи – буде розроблено сайт-портфоліо для художників з елементами соціальної мережі, який можна розвинути в комерційний продукт.

1 ПОСТАНОВКА ЗАДАЧІ, ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Вимоги

У роботі буде розроблено сайт-портфоліо для художників з елементами соціальної мережі.

Функціональні вимоги:

– реєстрація (двох типів, як роботодавець і як фахівець);
– авторизація;
– вихід з облікового запису;
– видалення акаунта;
– додання нового медіа;
– редагування акаунту;
– надання відгуку на фахівця;
– подання скарги на роботодавця;
– подання скарги на відгук;
– можливість листування.

Нефункціональні вимоги:

– інтуїтивна зрозумілість;
– зручність;
– гнучкість;
– розширюваність.

Розробка потребує використання наступного ПО:

– PhpStorm;
– NodeJS;
– модуля React;
– БД MySQL.

1.2 Огляд існуючих рішень

Instagram, Twitter, Tumbler. Соціальні мережі, разраховані на публікацію медіа-контенту з мінімальною кількістю тексту. Вони є чудовими джерелами, щоб зрозуміти хід думок, смаки та інші риси притаманні людині, але якщо і використовувати їх у якості портфоліо – то тільки як додатковий матеріал до основного, також, скоріш за все, для цього варто завести окремий, «робочий», акаунт.

Також, окремою складністю такого варіанту є набір аудиторії, серед якої із досить невисоким шансом може знайтись роботодавець або замовник.

Artstation. Тут знаходяться роботи найпопулярніших та високорівневих фахівців. Вершки світу майстрів і початківець просто загубиться за їх спинами. Хоча завести акаунт на Artstation ніколи не буде зайвим,оскільки він створений для демонстрації свої робіт.

Фріланс-сайти. Сайти створені для пошуку роботи та працюють за принципом «Хто встиг – той з'їв». Орієнтовані на роботодавця або на вже досвідченого художника-фрілансера. Дають змогу заробити собі рейтинг, але, початківцям зазвичай залишається найменш оплачувана вакансія та ненадійний замовник.

1.3 Огляд технологій для реалізації проєкту

Для серверної частини доданку обрано php-фреймворк Laravel, який побачив світ ще у 2011 році та й до цього часу не зупиняється у розвитку, майбутні оновлення розписані щонайменш на два роки уперед. В нього є популярні конкуренти, такі як Yii, Symfony, тощо; здебільшого – теж php-фреймворки, хоча у топ лідерів витнувся ще й Django, який є найбільш популярним фреймворком від Python. Однак Laravel все ще є найбільш звичним для веб-доданків такого роду.

Ось що ми можемо довідатись про конкурентів з їх офіційних джерел.

Yii – це високопродуктивний компонентний PHP-фреймворк, призначений для швидкої розробки сучасних веб-додатків. Слово Yii (вимовляється як Йі [ji:]) в китайській мові означає «простий та еволюційний». Також Yii може розшифровуватись як акронім для Yes It Is!

Yii – це універсальний фреймворк і може бути задіяний у всіх типах веб-додатків, що використовують PHP. Завдяки його компонентній структурі і відмінній підтримці кешування, фреймворк особливо підходить для розробки таких великих проектів як портали, форуми, системи керування вмістом (CMS), інтернет-магазини або RESTful-додатки.

Як і багато інших PHP-фреймворків, Yii втілює архітектурний шаблон MVC (Model-View-Controller) та сприяє організації коду відповідно до вимог шаблону.

Yii дотримується філософії простого й елегантного коду. Yii ніколи не буде намагатись пере-ускладнювати дизайн тільки заради слідування будь-яким шаблонам проектування.

Yii є full-stack фреймворком і включає в себе перевірені можливості, які добре себе зарекомендували: конструктори запитів та ActiveRecord для реляційних та NoSQL баз даних, підтримка REST API, багаторівневе кешування та інші.

Yii надзвичайно розширюваний. Завдяки надійній архітектурі розширень Yii, досить легко використовувати або розробляти поширюванні розширення.

Висока швидкодія завжди є головною ціллю Yii.

Yii – не проект однієї людини. Він підтримується і розвивається сильною командою і великою спільнотою розробників, які їй допомагають. Команда розробників фреймворку Yii стежать за тенденціями веб-розробки і розвитком інших проектів. Найбільш значимі можливості та кращі практики регулярно впроваджуються у фреймворк у вигляді простих й елегантних інтерфейсів [1].

Symfony є проєктом із відкритим вихідним кодом, керований спільнотою з тисячами учасників, до яких можна доєднатись, додаючи код або

документацію.

Symfony – це PHP-фреймворк для веб- і консольних додатків і набір повторно використовуваних компонентів PHP. Symfony використовується тисячами веб-додатків і більшістю популярних проектів PHP.

Розробкою проєкту керує основна команда Symfony за підтримки учасників Symfony.

Symfony суворо дотримується семантичного керування версіями, публікує версії «Довгострокової підтримки» (LTS) і має процес випуску, який є передбачуваним і зручним для бізнесу [2].

Django – це високорівневий веб-фреймворк Python, який заохочує швидку розробку та чистий, прагматичний дизайн. Створений досвідченими розробниками, він впорається з більшою частиною клопоту веб-розробки, тому можна зосередитися на написанні свого додатка без потреби винаходити колесо. Це безкоштовно та з відкритим кодом.

Django було розроблено, щоб допомогти розробникам якнайшвидше перевести додатки від концепції до завершення, серйозно ставиться до безпеки та допомагає розробникам уникати багатьох типових помилок безпеки. Деякі з найзавантаженіших веб-сайтів використовують здатність Django швидко та гнучко масштабуватися.

Проєкт містить десятки додаткових функцій, які можна використовувати для вирішення поширених завдань веб-розробки. Django піклується про автентифікацію користувачів, адміністрування вмісту, карти сайтів, RSS-канали та багато інших завдань – прямо з коробки. Компанії, організації та уряди використовували Django для створення різноманітних речей – від систем керування контентом до соціальних мереж і наукових обчислювальних платформ [3].

Laravel прагне надати розробнику неймовірний досвід, надаючи такі потужні функції, як ретельне впровадження залежностей, виразний рівень абстракції бази даних, черги та заплановані завдання, модульне та інтеграційне тестування тощо. Під час створення веб-додатку програмісту доступні

різноманітні інструменти та фреймворки, проте команда розробників вважає, що Laravel є найкращим вибором для створення сучасних повноцінних веб-додатків.

Laravel називають «прогресивним» фреймворком, розуміючи під цим, що Laravel росте разом із користувачем: якщо він тільки робить свої перші кроки у веб-розробці, величезна бібліотека документації, посібників і відеоуроків Laravel допоможе вам вивчити основи, не перевантажуючись, а якщо програміст вже senior розробник, Laravel надає надійні інструменти для впровадження залежностей, модульного тестування, черг, подій у реальному часі тощо. Laravel налаштований для створення професійних веб-додатків і готовий працювати з робочими навантаженнями підприємства.

Laravel неймовірно масштабований. Завдяки зручній для масштабування природі PHP і вбудованій підтримці Laravel для швидких систем розподіленого кешу, таких як Redis, горизонтальне масштабування з Laravel стає легким. Насправді програми Laravel легко масштабуються для обробки сотень мільйонів запитів на місяць [4].

Альтернативами технології React є фронтент-фреймворки Vue та Angular. Те, що саме власник Laravel став першим інвестором Vue, у свій час поєднало їх технології у стійкий стек, здається саме він мав би використовуватись, але React здається зрозумілішим через кількість туторіалів та активне ком’юніті. Angular же програє обом своїм конкурентам у продуктивності.

Vue (вимовляється як /vjuː/, як view) – це фреймворк JavaScript для створення інтерфейсів користувача. Він створений на основі стандартних HTML, CSS і JavaScript і забезпечує декларативну та компонентну модель програмування, яка допомагає вам ефективно розробляти інтерфейси користувача, прості чи складні.

Vue – це фреймворк та екосистема, яка охоплює більшість загальних функцій, необхідних для розробки інтерфейсу. Але Інтернет надзвичайно різноманітний – речі, які створюються в Інтернеті, можуть кардинально відрізнятися за формою та масштабом. Зважаючи на це, Vue розроблено таким

чином, щоб бути гнучким і поступово адаптуватися. Залежно від випадку, Vue можна використовувати різними способами:

– покращення статичного HTML без етапу створення;

– вбудовування як веб-компонентів на будь-якій сторінці;

– односторінкова програма (SPA);

– Fullstack / Server-Side Rendering (SSR);

– Jamstack / Генерація статичного сайту (SSG);

– орієнтація на комп’ютер, мобільний пристрій, WebGL і навіть термінал.

Незважаючи на гнучкість, основні знання про те, як працює Vue, поділяються на всі ці варіанти використання. Знання, отримані новачками на цьому шляху, залишаться корисними, коли вони зростатимуть для досягнення більш амбітних цілей у майбутньому, а ветеран можуть вибрати оптимальний спосіб використання Vue на основі проблем, які намагаються вирішити, зберігаючи ту саму продуктивність [5].

Angular – це платформа розробки, побудована на TypeScript. Як платформа Angular включає:

– компонентна основа для створення масштабованих веб-додатків;

– набір добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, зв’язок клієнт-сервер тощо;

– набір інструментів розробника, які допоможуть вам розробляти, будувати, тестувати й оновлювати код.

Angular може масштабуватися від проектів одного розробника до програм корпоративного рівня. Angular створено для того, щоб зробити оновлення максимально простим, тому використовувати переваги останніх розробок можна з мінімальними зусиллями. Екосистема Angular складається з різноманітної групи з понад 1,7 мільйона розробників, авторів бібліотек і творців контенту.

Компонентна модель Angular пропонує надійну інкапсуляцію та

інтуїтивно зрозумілу структуру програми. Компоненти також роблять програму безболісною для модульного тестування та можуть покращити загальну читабельність коду [6].

React робить створення інтерактивних інтерфейсів безболісним. Можна робити прості представлення для кожного стану програми, і React ефективно оновлюватиме та відтворюватиме потрібні компоненти, коли дані змінюватимуться.

Декларативні представлення роблять код більш передбачуваним і легшим для налагодження. React дозволяє створювати інкапсульовані компоненти, які керують власним станом, а потім комбінувати їх для створення складних інтерфейсів користувача.

Оскільки логіка компонентів написана на JavaScript, а не на шаблонах, можна легко передавати розширені дані через програму та не допускати стану в DOM.

Команда React не робить припущень щодо решти стеку технологій користувача, тому можна розробляти нові функції в React, не переписуючи існуючий код.

React також може рендерити на сервері за допомогою Node і розширювати мобільні програми за допомогою React Native.

Компоненти React реалізують метод render(), який приймає вхідні дані та повертає те, що потрібно відобразити. У цьому прикладі використовується XML-подібний синтаксис під назвою JSX. Вхідні дані, які передаються в компонент, можуть бути доступні за допомогою render() через this.props. Проте JSX необов’язковий і не потрібен для використання React.

Окрім отримання вхідних даних (до яких можна отримати доступ через this.props), компонент може підтримувати дані внутрішнього стану (до яких можна отримати доступ через this.state). Коли дані про стан компонента змінюються, відтворену розмітку буде оновлено шляхом повторного виклику render().

React дозволяє взаємодіяти з іншими бібліотеками та фреймворками,

наприклад Remarkable, зовнішня бібліотека Markdown, для перетворення значення <textarea> в реальному часі.

До обраних основних технологій добре доєднується GraphQL своєю простотою розуміння та написання запитів. Він навіть дозволяє у деяких місцях мінімізувати роботу контролерів та сервісів [7].

GraphQL – це мова запитів для API та середовище виконання для виконання цих запитів із наявними даними. GraphQL надає повний і зрозумілий опис даних у вашому API, дає клієнтам можливість вимагати саме те, що їм потрібно, і нічого більше, полегшує розвиток API з часом і надає потужні інструменти розробника.

Запити GraphQL завжди повертають передбачувані результати. Програми, які використовують GraphQL, є швидкими та стабільними, оскільки вони контролюють дані, які вони отримують, а не сервер.

Запити GraphQL отримують доступ не лише до властивостей одного ресурсу, але й плавно дотримуються посилань між ними. У той час як типові API REST вимагають завантаження з кількох URL-адрес, API GraphQL отримують усі дані, необхідні вашій програмі, в одному запиті. Програми, які використовують GraphQL, можуть працювати швидко навіть за повільного з’єднання з мобільною мережею.

GraphQL API організовано за типами та полями, а не за кінцевими точками. Отримайте доступ до всіх можливостей ваших даних з єдиної кінцевої точки. GraphQL використовує типи, щоб переконатися, що програми запитують лише те, що можливо, і надають чіткі та корисні помилки. Програми можуть використовувати типи, щоб уникнути написання коду розбору вручну.

API GraphQL дозволяє додавати нові поля та типи, не впливаючи на наявні запити. Застарілі поля можна приховати від інструментів. Завдяки використанню єдиної версії, що розвивається, API GraphQL надають додаткам безперервний доступ до нових функцій і заохочують чистіший і зручніший серверний код.

GraphQL створює єдиний API для всієї програми, не обмежуючись

певним механізмом зберігання. Напишіть GraphQL API, які використовують наявні дані та код за допомогою механізмів, доступних багатьма мовами. Користувач надає функції для кожного поля в системі типів, і GraphQL викликає їх з оптимальною паралельністю.

У якості бази даних обрано найбільш звичну оку реляційну MySQL, яка дуже добре працює із Laravel [8].

1.4 Додаткові технології, що будуть використані

Половина розробки доданку зосереджено на формуванні ТЗ, для чого використовується багато різноманітних інструментів, дозволяючи досконально описати важливі аспекти майбутнього продукту.

Для створення діаграм буде використано дуже зручний інструмент Drowio, який пропонує широкий асортимент готових патернів діаграм та ще більший перелік інструментів для створення схем з нуля. У ньому можна використовувати будь-яку нотацію, неймовірно гнучкі налаштування зовнішнього вигляду блоків, фігур та стрілок допоможуть це зробити без зайвого головного болю [9].

Для створення мокапів сторінок, які можна буде легко та зрозуміло читати при верстці, необхідно використання такого інструменту, який зміг би донести інформацію, закладену дизайнером, до програміста у повному обсязі. У якості такого інструменту виступає сервіс Figma, який дозволяє дивитися розмір блоків, номери використаних кольорів, підтримує широкий асортимент шрифтів, формулює базовий css блоку та навіть може зберігати додаткові нотатки. В одному проєкті можна зберігати майже необмежену кількість мокапів сторінок та додавати коментарі до кожної [10].

Для написання коду проєкту необхідна сучасна прогресивна IDE, яка б підтримувала як синтаксис мови серверної частини, так і javascript, html та css, необхідні для написання сторінок фронтенду. Для цього ідеально підходить

продукт компанії JetBrains PhpStorm. Він підтримує досить значний перелік фреймворків (Symfony, Drupal, WordPress, Zend Framework, Laravel, Magento, Joomla!, CakePHP, Yii), глибоко аналізує структуру коду і дійсно розуміє код, підтримуючи всі можливості мови PHP як у нових, так і у legacy-проектах. Редактор підтримує автодоповнення коду та рефакторинги, запобігає помилкам на льоту. У PhpStorm можна працювати з найсучаснішими технологіями: HTML5, CSS, Sass, Less, Stylus, CoffeeScript, TypeScript, Emmet та JavaScript. При цьому будуть доступні рефакторинг, налагодження та юніт-тестування. Завдяки функції Live Edit всі зміни можна відразу подивитися в браузері.

Доданок має вбудовані інструменти для розробників – IDE інтегрована із системами контролю версій, підтримує віддалене розгортання, бази даних та SQL, інструменти командного рядка, Docker, Composer, REST-клієнт та багато інших інструментів. PhpStorm включає всю функціональність WebStorm, а також повноцінну підтримку PHP, баз даних і SQL [11, 12].

Під час розробки будуть використовуватись підручники з відповідних технологій [14–17].

2 РОЗРОБКА МОДЕЛІ ТА ДІАГРАМ ВЗАЄМОДІЇ ПЗ

2.1 Вивчення предметної області

Розробка моделі майбутнього програмного забезпечення є дуже важливою частиною процесу створення кінцевого продукту і суттєво впливає на його подальший життєвий цикл. Відповідально виконана модель сильно полегшує роботу над програмою та сприяє глибшому розумінню задачі розробниками.

У першу чергу розробник має зібрати вимоги до майбутнього додатку та проаналізувати схожі проекти, що було зроблено у першому розділі.

Серед супутніх функцій було виділено такі необхідні:

– реєстрація;
– авторизація;
– вихід з акаунта;
– видалення акаунта;
– додання нового медіа;
– редагування акаунта;
– пошук фахівця або роботодавця;
– надання відгуку на фахівця;
– подання скарги на будь-якого користувача;
– можливість блокування акаунта користувача на підставі скарг (за замовчанням буде використовуватись границя у три скарги);
– можливість листування між роботодавцем і фахівцем;
– можливість користувача мати обидва типи акаунта.

Під час вивчення предметної області було виділено чотири актори – неавторизований користувач, роботодавець, фахівець та модератор, що виконував би обов’язки з врегулювання конфліктів на підставі скарг. Його основні можливості наступні:

– перевірка та підтвердження скарг;
– написання листів користувачам;
– блокування акаунтів.

Створити акаунт модератора має могти тільки адміністратор, перейти на його сторінку для перегляду має бути неможливим, а при листування з ним у вікні бесіди має світитися відповідне повідомлення, щоб запобігти шахрайству.

2.2 Формування вимог

На базі проведеного дослідження було сформовано списки необхідних для виконання умов.

Загальні вимоги:

– при реалізації виконати роботи у повному обсязі;
– з точки зору замовника: створення сайту-портфоліо для художників з елементами соціальної мережі;
– з точки зору користувача-роботодавця: можливість пошуку відповідного фахівця та спілкування з ним;
– з точки зору користувача-фахівця: можливість розміщення свого портфоліо із супутніми коментарями та спілкування із замовниками;
– розробити зрозуміле та розгорнуте технічне завдання.

Технічні вимоги:

– бек-енд частина сервісу має бути реалізована за допомогою фреймворку Laravel;
– фронт-енд частина сервісу має бути реалізована на основі бібліотек React;
– бек-енд і фронт-енд мають підтримувати зв'язок за допомогою технології GraphQL;
– автентифікація має виконуватися через електронну пошту користувача та його персональний пароль;

– користувач може мати два типи акаунтів на однакових ідентифікаційних даних.

Для вдалої розробки моделей доданку, розглянемо детальніше інформацію, необхідну для здорового функціонування. На підставі зібраних даних виділено щонайменш вісім сутностей: користувач (об’єднує в собі усіх, хто має облікові записи), роль (за допомогою якої будуть розрізнятися користувачі), лист, відгук, скарга, ілюстрація, блокування та технологія (щоб уникнути плутаниці та полегшити процедуру пошуку фахівців, варто виділити її у окремий лист, який зможуть поповнювати користувачі).

Так, сутності роль та технологія є тривіальними списками, тож їх детально розглядати немає сенсу. Перейдемо до більш складних і почнемо з сутності користувача, бо саме вона є центральною у системі. Кожен акаунт користувача має електронну адресу, на яку його зареєстровано, пароль входу та роль – Модератор, Роботодавець або Фахівець. В залежності від ролі, необхідні поля будуть розрізнятись. Модератор обмежиться тільки основними трьома, що вже були вказані вище. Акаунт роботодавця буде мати в собі додатково ім’я людини, аватар, назву його компанії (за наявності) та опис «About» у якому користувач може вказати особливості своєї діяльності, вподобання, тощо. Акаунт фахівця потребує найбільшого розширення: окрім імені та аватара, він обов’язково має перелік технологій, вказує теми та стилістику, яким він надає переваги при роботі, вказує теми які є для нього неприйнятними і тому за замовлення із ними він не береться. У секції «About» фахівець, подібно до роботодавця, може вказати особливості своєї діяльності.

Також, фахівець може завантажувати ілюстрації досить великих розмірів у якості портфоліо і додавати опис до них, це означає, що варто виділити ілюстрацію як окрему сутність, яка буде мати поле опису дату завантаження та поле із ідентифікаційним номером фахівця, який її завантажив.

Майже така сама за структурою є сутність відгуку – опис, ідентифікаційний номер фахівця, роботодавця, який залишив відгук та дата створення.

Ідентичною є сутність листа: опис, відправник, отримувач, дата відправки.

Наступна сутність – скарга. Вона також має містити опис (причину) скарги, ідентифікаційні номери користувача, на якого було подано скаргу та користувача, що її залишив, дату подання, але, оскільки в нас є можливість перевірки скарги модератором, з’являються ще два поля, які вказують на те, переглянута скарга, чи ні, а також, чи була вона погоджена.

І остання – блокування. Вона буде містити ідентифікаційний номер користувача, якого заблоковано, номер модератора, який прийняв це рішення, причину блокування, дату початку та дату кінця.

Тепер, коли предметна область достатньо вивчена, можна перейти до розробки технічного завдання.

2.3 Розробка діаграм

Оскільки повністю текстовий матеріал є складним для сприйняття та йому бракує наочності, скористаємося графічними формами подання для підвищення зрозумілості технічного завдання – таблицями та діаграмами.

На сьогодні існує багато нотацій та ще більше видів діаграм, кожна має свої переваги, недоліки та специфіку. Використаємо їх для розгляду складних дій розроблюваного доданку. Для розробки технічного завдання проєкту було виділено такий перелік діаграм:

– діаграма акторів і прецедентів (допоможе співвіднести ролі майбутніх користувачів системи та необхідні ім. функції);
– діаграма послідовності (відобразить порядок передачі даних);
– діаграма активності (конкретизує дії кожної частини системи під час конкретного прецеденту);
– діаграма компонентів (буде використана для побудування дерева компонентів React);

– діаграма бази даних;

– діаграма розміщення.

При вивчені було виділено чотири актора – неавторизований користувач (надалі User), фахівець (Artist), роботодавець (Employer) та модератор (Moderator). Кожен з них має певні особливості діяльності, відмінності у необхідній для оформлені особистої сторінки, тощо. Відобразимо всіх акторів та можливі прецеденти на діаграмі акторів і президентів, виконаній в нотації UML (див. рис. 2.1).

Рисунок 2.1 – Діаграма акторів і прецедентів

Діаграма вийшла доволі складною але все читається. Опишемо найбільш складні у реалізації прецеденти у форматі назва-актор-опис-події-умови.

Назва прецеденту: *Реєстрація.*

Актор: неавторизований користувач (User).

Стислий опис: створення нового облікового запису типу Employer або Artist.

Основний потік подій: Неавторизований користувач переходить на сторінку реєстрації, заповнює поля імені, електронної адреси, паролю, обирає, який тип облікового запису він бажає створити та натискає кнопку «Next step». Якщо форма заповнена валідними даними, відкривається наступна сторінка з полями необхідними для обраного типу акаунта. Для облікового запису Artist є обов’язковим поле технологій, тож користувач має обрати хоча б одну технологію з випадаючого списку. Після заповнення полів, користувач натискає клавішу «Sign up», дані підправляються на сервер і з’являється анімація лоадеру із повідомленням про необхідність веріфікації через лист у електронній скриньці, після якої відкривається сторінка особистого кабінету.

Альтернативний потік подій:

1) якщо перша сторінка була заповнена невалідними даними: поле із помилкою виділяється червоним маркером, при натисканні клавіши «Next step» сторінка не змінюється та з’являється відповідне повідомлення;

2) якщо на другій сторінці не заповненні обов’язкові поля: при натисканні клавіши «Sign up» з’являється повідомлення про те, что обов’язкові поля не можуть бути пустими та відповідне поле виділяється червоним маркером;

3) якщо на сервері виникла помилка під час обробки даних – виводиться відповідне сповіщення, яке також буде містити контакти підтримки сайту;

4) якщо на вказану електронну адресу вже зареєстровано такий самий тип акаунта, після натискання клавіши «Next step» сторінка не змінюється на наступну та з’являється відповідне повідомлення.

Постумови: веріфікація створеного акаунту. Доки це не зроблено, особистий кабінет буде недоступний.

Назва прецеденту: *Обробка скарги.*

Актор: модератор (Moderator).

Стислий опис: прогляд та погодження скарги на обліковий запис користувача.

Основний потік подій: Moderator отримує сповіщення про скаргу на користувача типу Employer або Artist, відкриває його. Якщо скарга є актуальною, Moderator за необхідністю дописує до підстави свій коментар та погоджує її. За необхідністю пише в особисті повідомлення попередження про скаргу.

Альтернативний потік подій: якщо скарга є невалідною, Moderator відмічає її як непогоджену. Запланована на певний день процедура бази даних видаляє проглянуті, але непогоджені скарги.

Передумови: на користувача відправлено скаргу.

Постумови: скарга зберігається у базі даних.

Назва прецеденту: *Видача бану (та обробка скарги)*.

Актор: модератор (Moderator).

Стислий опис: блокування облікового запису користувача на підставі трьох скарг.

Основний потік подій: Moderator отримує сповіщення про скаргу на користувача типу Employer або Artist, відкриває його. Якщо скарга є актуальною, Moderator погоджує її і активується дія блокування акаунта «Ban». Moderator заповнює текстове поле підставою, на якій заблоковано акаунт, вказати строк бану та натиснути кнопку «OK».

Альтернативний потік подій: Moderator вважає, що порушення у попередніх скаргах не є тяжкими та не видає блокування, але може написати у особисті повідомлення попередження про його можливість.

Передумови: на користувача вже є дві погоджених скарги.

Постумови: Заблокованому користувачу приходить електронний лист зі сповіщенням про бан його акаунта та поясненням його причин. При вході до особистого кабінету відображається відповідне сповіщення. Всі нові дані

зберігаються у БД.

Ці прецеденти все ще варто зобразити у графічному виді, щоб визначитись з послідовністю дій та тим, яка інформація буде надаватись актором та яку проходити обробку. Розглянемо спочатку діаграми послідовності реєстрації (див. рис. 2.2).

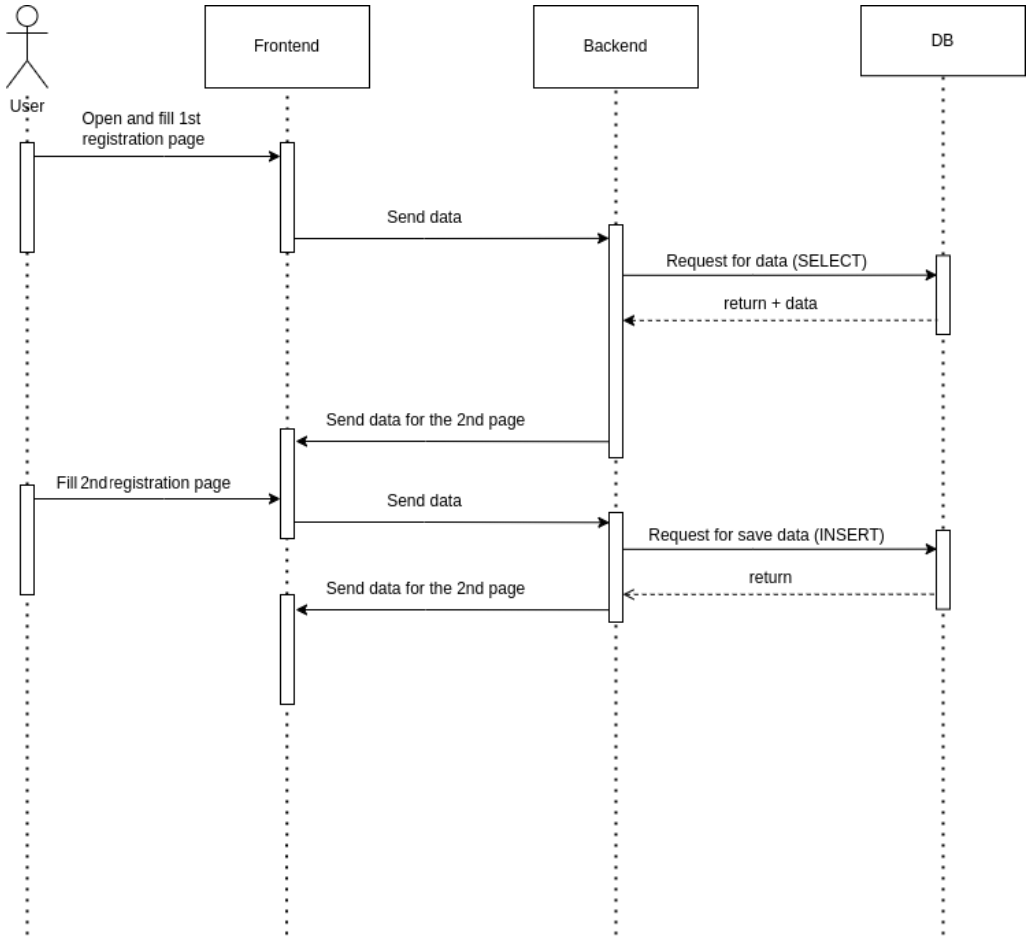

Рисунок 2.2 – Діаграма послідовності прецеденту Реєстрація

Оскільки користувач може зареєструвати два типи акаунта, було вирішено поділити процес на дві частини. У першій користувач вводить свою електронну адресу, створює пароль та обирає тип акаунта, після чого додаток перевіряє, чи існує вже такий обліковий запис. Якщо ні – реєстрація продовжується і користувач має можливість заповнити поля інформацією, яка буде відображатися безпосередньо на особистій сторінці.

Далі розглянемо прецеденти подання скарги та блокування акаунта. Для

зручності вони об’єднані в одній діаграмі, адже, при певних, умовах вони йдуть послідовно (див. рис. 2.3).

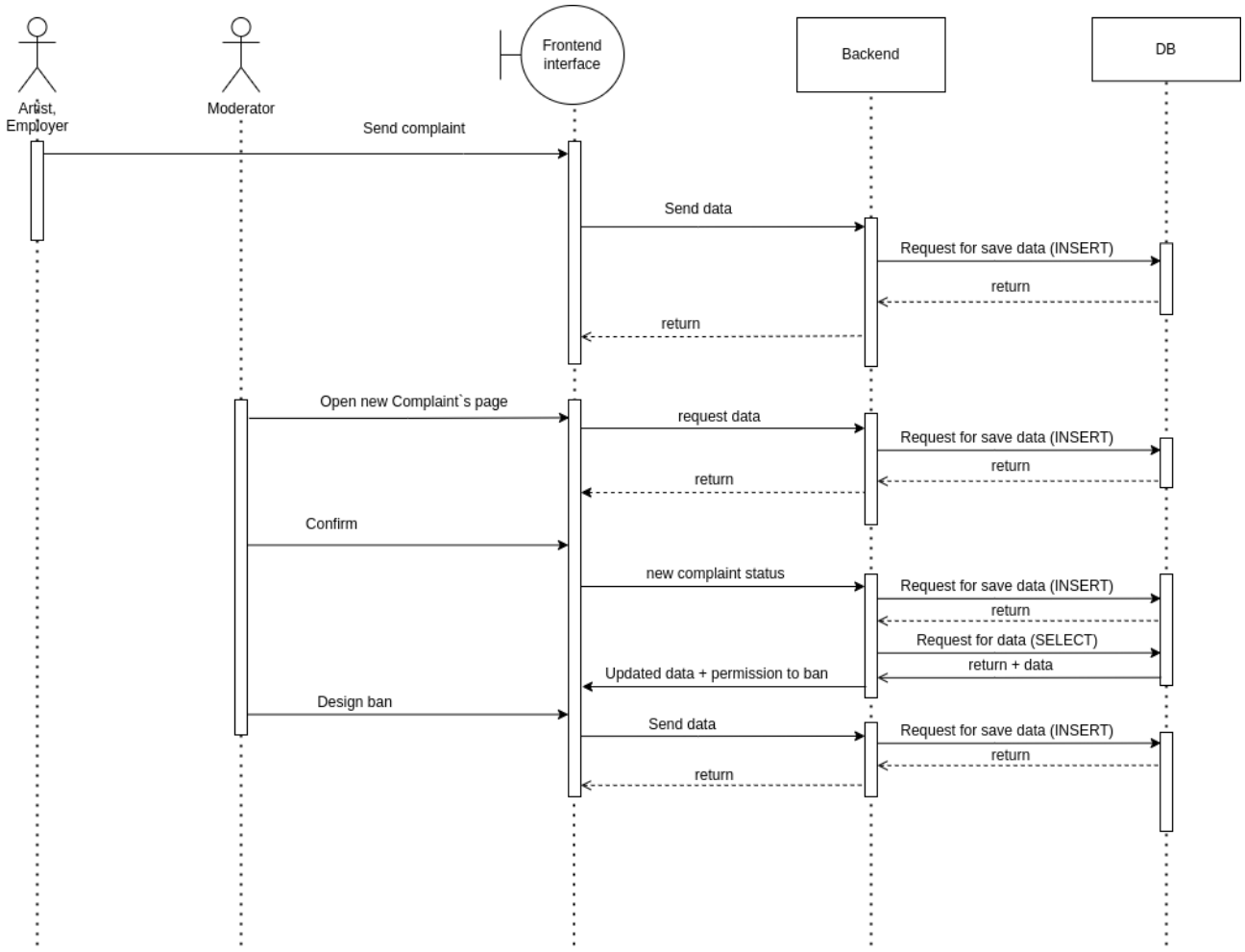

Рисунок 2.3 – Діаграма послідовності прецедентів Подання скарги та Блокування акаунта

Майже все зрозуміло, але треба трохи більше деталізації, щоб було легше працювати над дизайном сторінок Модератора, до ця послідовність не повністю відображає текстовий опис прецедента. Для цього спробуємо описати цей прецедент за допомогою діаграми активності (див. рис. 2.4).

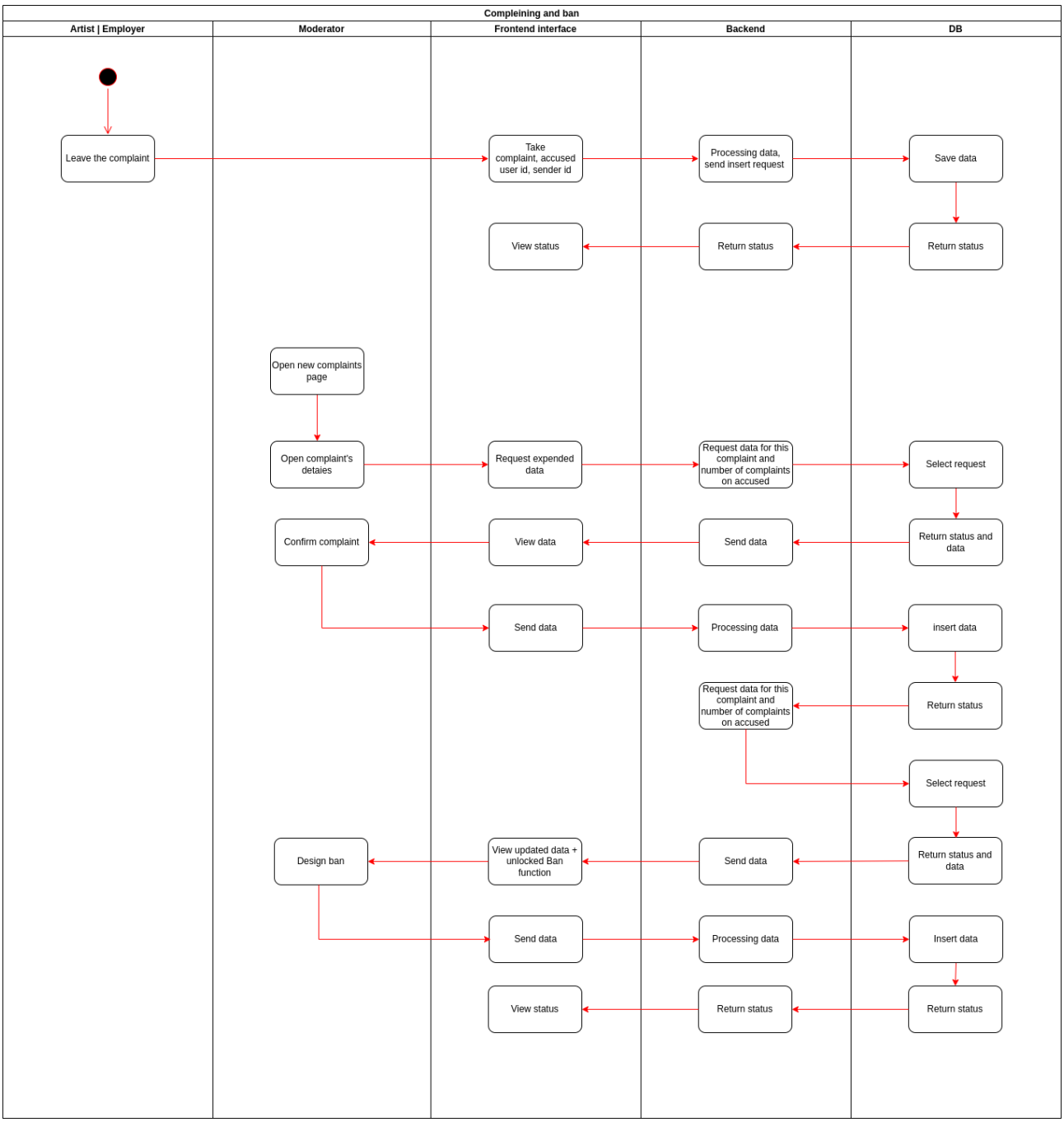

Рисунок 2.4 – Діаграма активності прецедентів Подання скарги та Блокування акаунта

Тепер при погляді на діаграми зрозуміло, які дії будуть виконуватись.

Завдяки попереднім діям можна визначитись із необхідними для роботи системи даними, цього достатньо для розробки моделі бази даних (див. рис. 2.5).

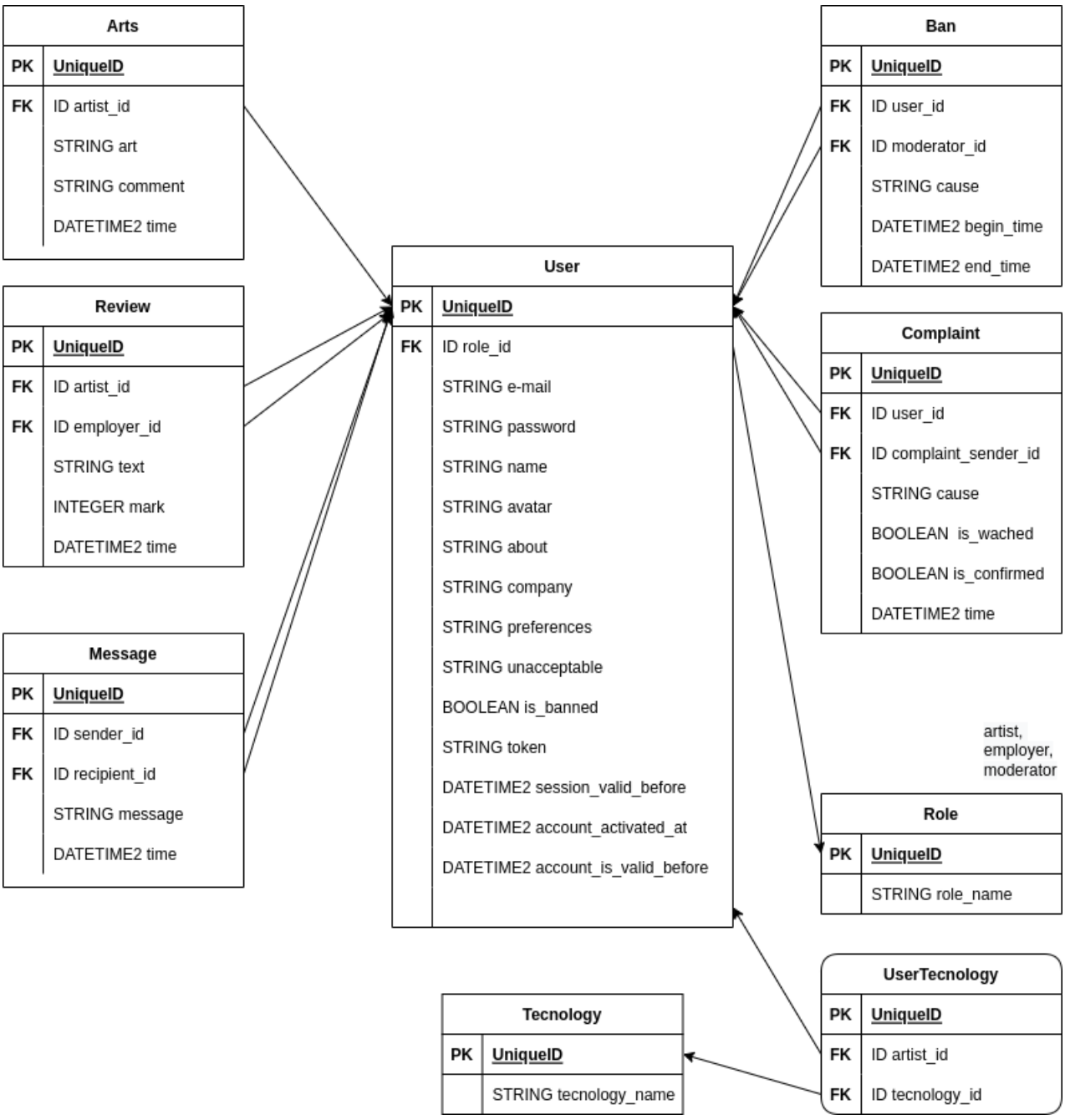

Рисунок 2.4 – Модель бази даних

Як можна побачити розроблена модель бази даних містить усі вісім виявлених у розділі 2.2 сутностей та додаткову таблицю, яка є повністю залежною і використана для реалізації зв’язку «багато до багатьох» між користувачем (а саме – фахівцем) та переліком технологій, що він використовує. Модель є досить гнучкою та сприятливою для розширення при подальшому розвитку системи.

Завершимо розділ діаграмою розміщення (див. рис. 2.5). Весь доданок

має бути розміщено на віддаленому сервері з підтримкою php, поряд із ним буде розташована база даних. Згенеровані сторінки проектуються на клієнтські пристрої і спілкуються із сервером за допомогою ajax-запитів, які містять у собі GraphQL query.

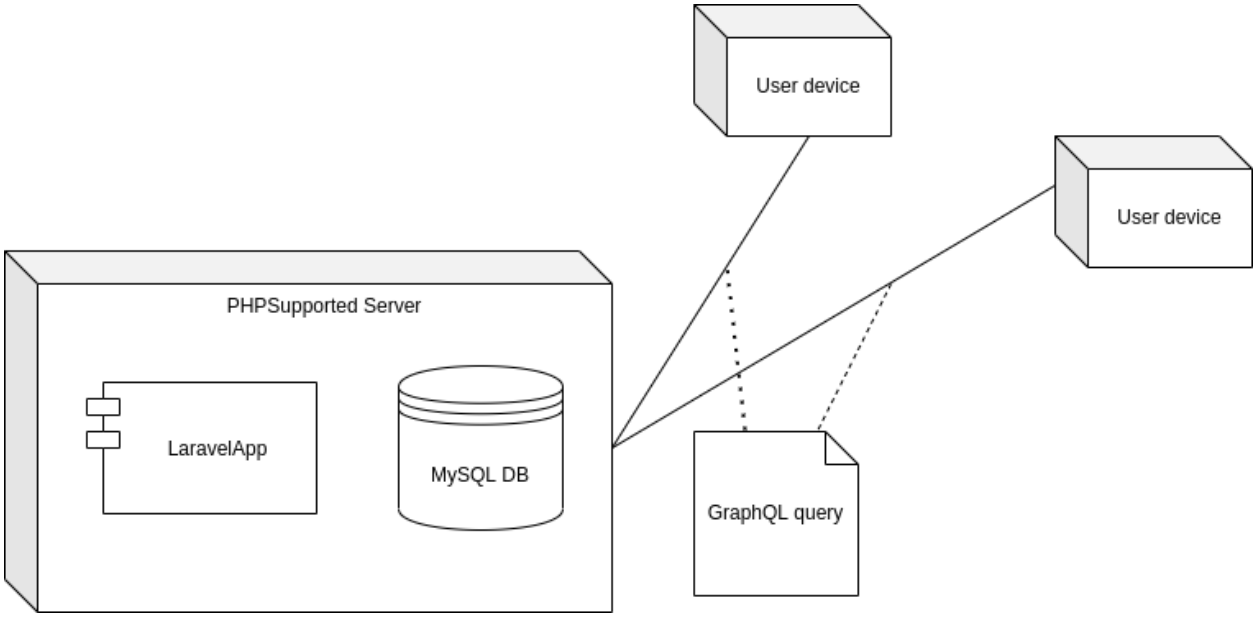

Рисунок 2.5 – Діаграма розміщення

Посилання на електронні версії діаграм на гугл-диску буде надано у Додатку А.

2.4 Розробка макетів інтерфейсу

Коли сформульовано основну логіку роботи програми, треба подбати, щоб нею можна було скористатись. Сучасний дизайн розвивається у напрямку інтуїтивної зрозумілості та мінімалістичності, дуже важливо притримуватись цих тенденцій, для зручності користування. Визначемо вимоги до інтерфейсу:

– інтуїтивно зрозумілий інтерфейс;
– простий, помірно строгий дизайн;
– спокійні кольори;

– можливість скористатися усіма необхідними функціями;

– здебільшого використовувати однотипні елементи.

Розробка макетів інтерфейсу виконана за допомогою хмарного сервісу Figma. У якості палітри обрано поєднання трьох помірно холодних відтінків синього кольору, для влучної назви сайту, яка б природно читалась в логотипі, обрано поєднання слів «Digital» та «Art» – DiArt.

Створено макети наступних сторінок:

1) лендінг сторінка;

2) три сторінки для реєстрації (перша та друга частини прецеденту з різними типами акаунта);

3) дві сторінки входу в обліковий запис (розширення на випадок, якщо користувач має два акаунта);

4) сторінка пошуку користувачів;

5) два вигляди особистої сторінки фахівця (очима авторизованого фахівця та очима роботодавця);

6) сторінка відгуків на фахівця;

7) особиста сторінка роботодавця;

8) робоча сторінка модератора;

9) сторінка жалоби та бану;

10) сторінка очікування.

Сторінка лендінгу (див. рис. 2.6) містить великий варіант логотипу, місце для інформації про сервіс, перехід до сторінок реєстрації та авторизації. Загальні атрибути, які будуть поширюватися на всі інші сторінки – хедер з логотипом, футер із номером та поштовою скринькою підтримки сайту та сайдбар із карточками акаунтів чотирьох випадкових фахівців.

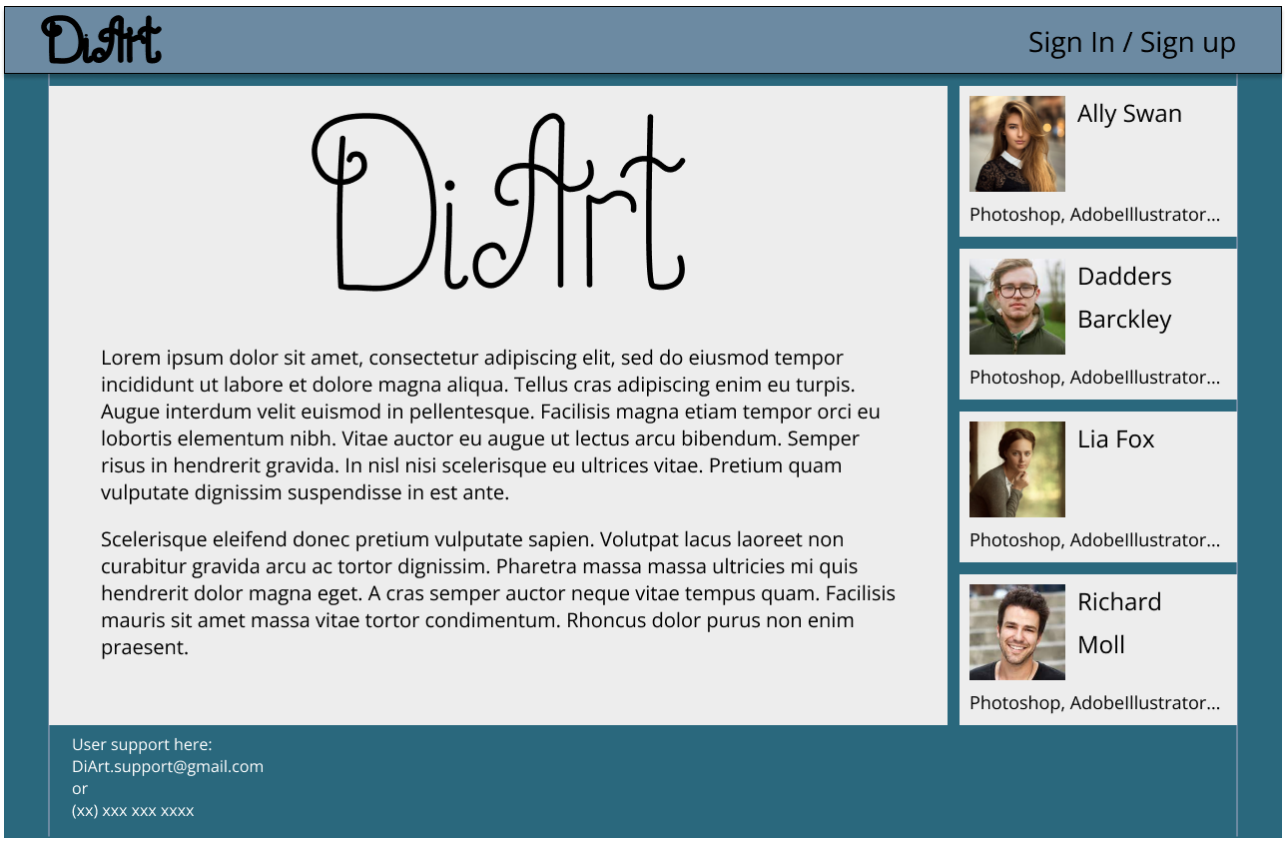

Рисунок 2.6 – Лендінг-сторінка сайту

Сторінка реєстрації (див. рис. 2.7) розроблена відповідно до описаного в попередньому пункті прецеденту Реєстрація. Перша частина містить в собі поле для вводу електронної адреси, два поля для створення паролю та дві радіо кнопки для вибору типу акаунта. При натисканні клавіши "Next step" передбачається, що дані про електронну пошту відправляється на сервер для перевірки наявності акаунта обраного типу у базі даних. Якщо такий обліковий запис вже існує, сторінка не змінюються на наступну та виводиться відповідне повідомлення на тілі форми. Також повідомлення про помилку буде з’являтись при некоректному заповненні полів, яке спровокую помилку валідації. Якщо пола заповнені валідними даними та перевірка бази даних показала, що створити новий акаунт можливо – відбудеться перехід до другої частини реєстрації. Наступна сторінка відрізняється залежно від типу акаунта, на ній являється форма заповнення необхідних для облікового запису полів (див. рис. 2.8). Аватар можна буде додати при редагування, як і змінити наповнення полів.

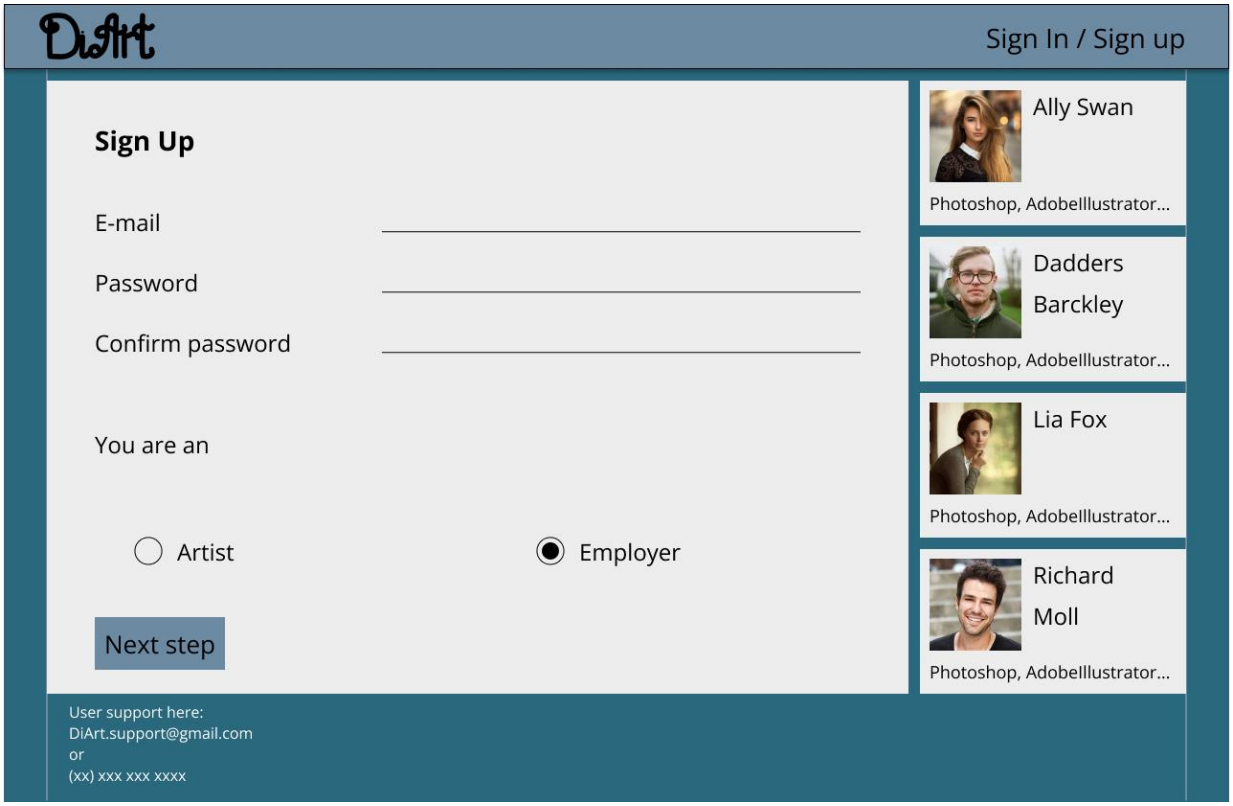

Рисунок 2.7 – Перша сторінка реєстрації

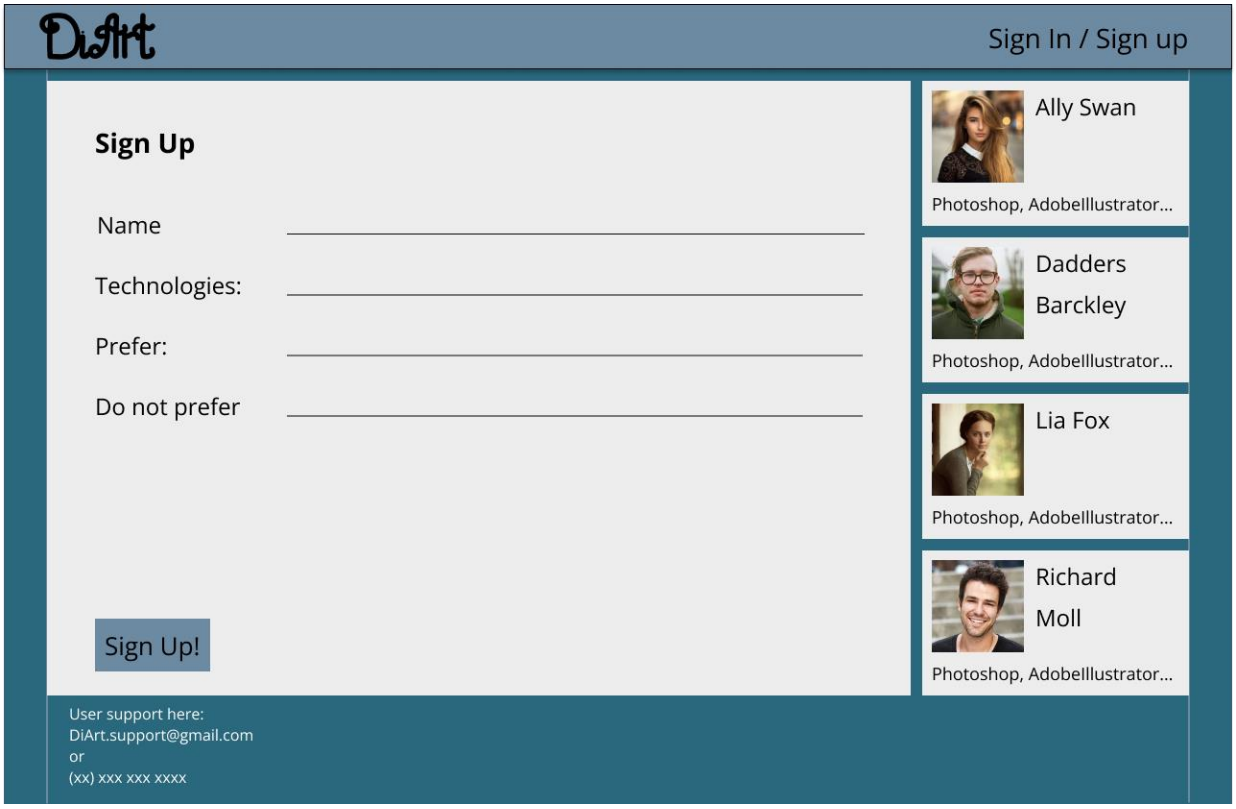

Рисунок 2.8 – Друга сторінка реєстрації, варіант роботодавця

Розглянемо особисту сторінку користувача на прикладі акаунта фахівця (див. рис. 2.9), оскільки вигляд кабінету роботодавця є надто тривіальним (див. рис. Б.2).

На мокапі можна побачити аватар користувача, інформацію про діяльність фахівця, іконку для виклику сторінки редактування інформації («олівець»), посилання на сторінку відгуків, поле для додавання нової ілюстрації та, безпосередньо, перелік мініатюр раніше викладених робіт. Функції пошуку, листування та виходу з акаунта знаходяться у випадаючому списку, прихованому під ім’ям користувача у хедері. Також цей список містить посилання на сторінку з правилами ресурсу.

З точки зору роботодавця, або користувача іншої сторінки, іконка «олівець» буде замінена на іконку початку листування (піктограма «конвертик») та подання скарги («прапорець»). Ці варіанти сторінок можна побачити у Додатку Б.

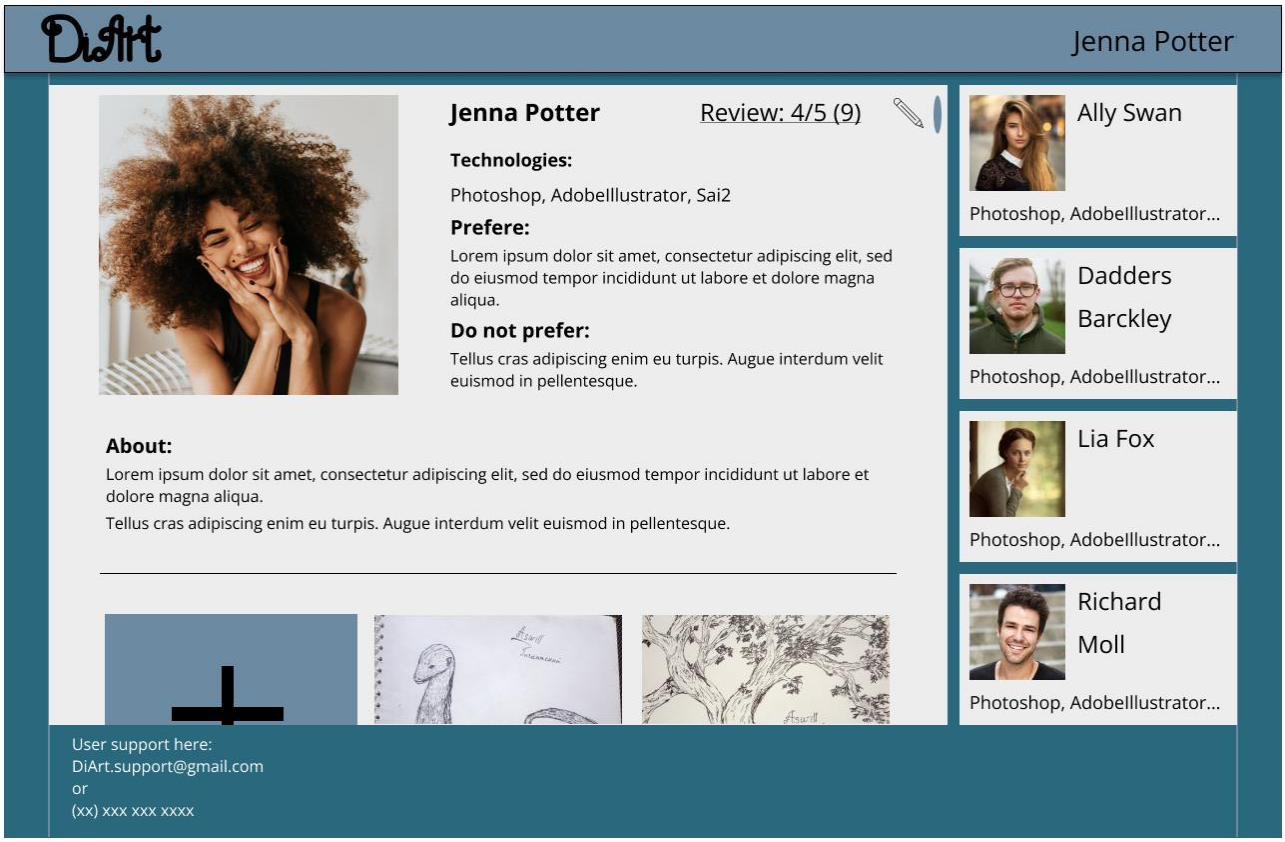

Рисунок 2.9 – Особиста сторінка авторизованого фахівця

Сторінка відгуків (див. рис. 2.10) має у собі перелік відгуків роботодавців, які скористались послугами фахівця. Кожен відгук має піктограму «прапорець», що дозволяє оскаржити його, не переходячи на сторінку людини, що залишила цей відгук. У причині скарги необхідна указати, що оскаржується саме відгук, зробити скрін та додати посилання на нього до тіла скарги. Оскаржити відгук може не тільки фахівець, якому було його надано, це функція також є доступною для будь-якого авторизованого користувача, який переглядає сторінку. Роботодавець може залишити відгук, натиснувши на іконку «чат», яка відкриває віконце із текстовим полем.

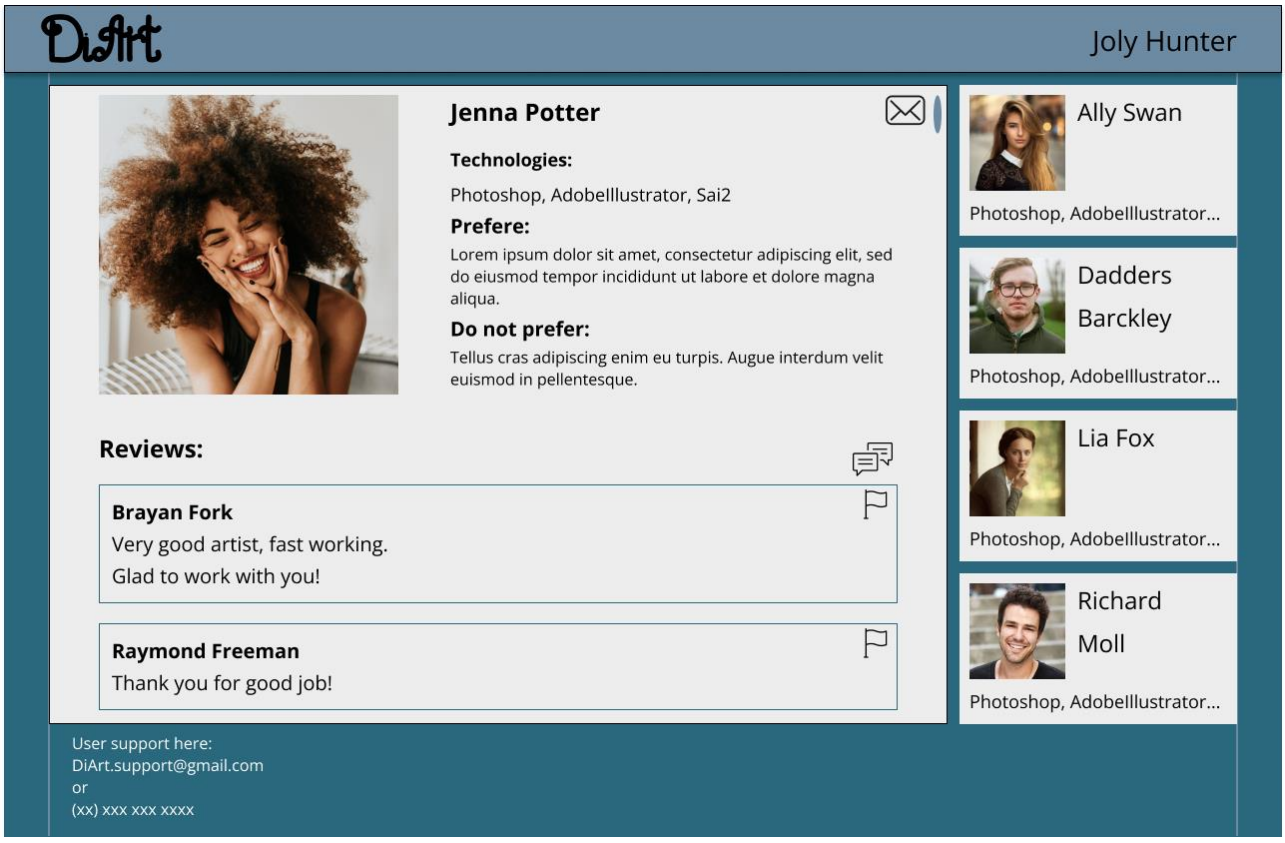

Рисунок 2.10 – Сторінка відгуків. Вигляд роботодавця

Наступним розглянемо профіль модератора, який суттєво відрізняється за структурою від акаунтів інших користувачів (див. рис. 2.11).

По-перше, обліковий запис модератора не має ані особистого імені, ані аватару, ані будь-яких полів опису. По-друге, він закритий для перегляду сторонніми користувачами.

Особистий кабінет модератора має дві вкладки: перша – з проглянутими, але не затвердженими і не відхиленими скаргами, друга – з новими скаргами. Переглянуті скарги, по яким було надане рішення, зникають з кабінету, але зберігаються у базі даних. Вони можуть бути переглянуті через сторінку користувача, на яких вони були надані, або через архів, який доступний з випадаючого списку у хедері.

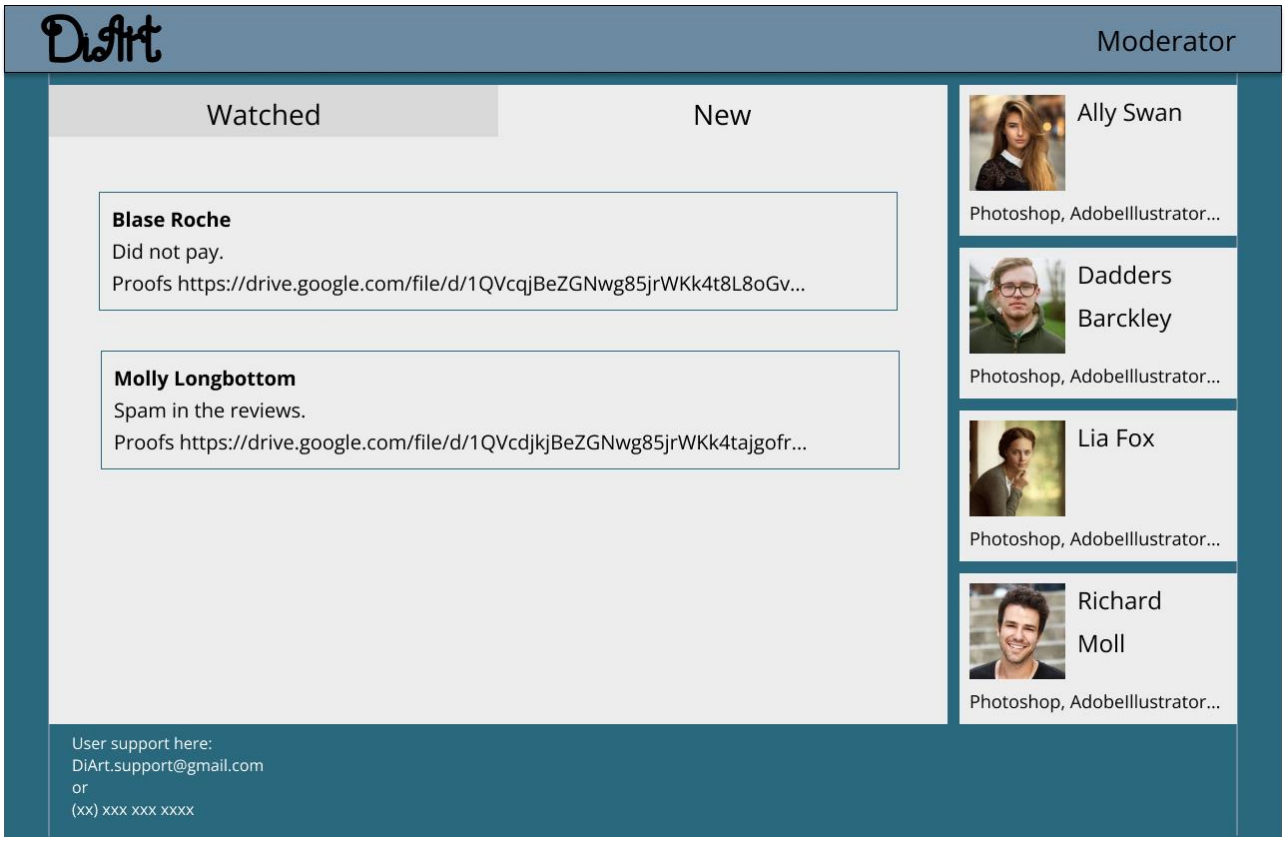

Рисунок 2.11 – Робочий кабінет модератора

Тепер розглянемо сторінку перегляду скарги об’єднану з видачею блокування (див. рис. 2.12).

На сторінці ми можемо бачити ім’я користувача, який надав скаргу, тіло скарги, яке містить у собі причину та посилання на скриншот із доказами, ім’я користувача, на якого було подано скаргу, та кнопки підтвердження та відхилення скарги. Цифра у дужках поряд із ім’я порушника є посиланням на список попередніх скарг і відображає їх кількість.

Якщо кількість скарг перевищує заданий у сервісі мінімум, то частина

сторінки, необхідна для видачі блокування стає активною. На ній можна клікнути по чек боксу з лейблом «Ban», що дозволить описати причини блокування та, натиснувши кнопку «Confirm», заблокувати користувача.

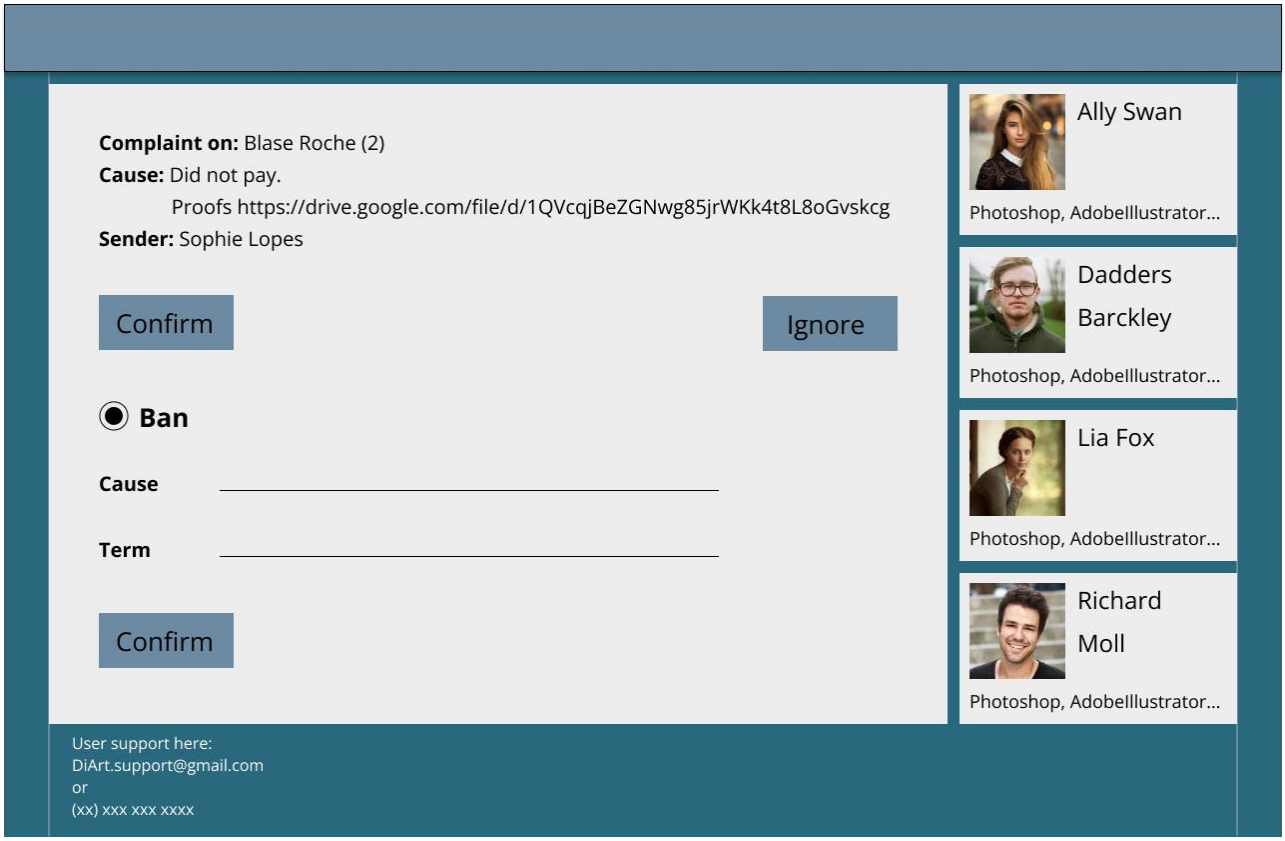

Рисунок 2.12 – Сторінка видачі перегляду скарги та видачі блокування

Ми розглянули найбільш цікаві мокапи сторінок інтерфейсу, інші можна переглянути за посиланням [2] у Додатку А.

Тепер, можна виділити компоненти, у яких прописано вигляд інтерфейсу:

1) main;
2) header;
3) dropdownListCreator;
4) navbar;
5) navbarItem;
6) registerSecond;
7) registerEmployer;
8) login;

9) userProfile;

10) imageItem;

11) imageFull;

12) reviewPage;

13) reviewItem;

14) addImageForm;

15) addReviewForm;

16) sendComplaintForm;

17) moderatorProfile;

18) complaintAndBan;

19) messenger;

20) personMessenger;

21) messageItem;

22) search;

23) personSerchItem.

Компонента main є неявною, у ній виводяться або компонуються необхідні частини сторінок, реалізовані у різних компонентах.

Компоненти з «Item» на кінці є компонентами, які формують елементи списків.

Компоненти з «Form» на кінці реалізують спливаючі вікна з відповідними до назви формами.

У цілому, назви компонент дають достатнє уявлення про свій функціонал.

3 СТВОРЕННЯ ДОДАТКУ

3.1 Створення зв’язку із GraphQL та Lighthouse

Для реалізації доданку створимо новий Laravel проєкт із назвою DiArt та підключимо до нього чисту базу даних. Після написання міграцій, які відповідають, наведеній у пункті 2.3, моделі бази даних, за допомогою команди php artisan make створимо всі необхідні моделі. Після цього можна починати створення основних запитів, спираючись на перелік прецедентів, виділених під час розробки попереднього розділу.

GraphQL є дуже зручною альтернативою звичному REST API. До нього дуже легко перейти і підтримується він багатьма платформами, для чого існують спеціальні фреймворки. У випадку із Laravel для використання graphql-схем було створено розширюючий фреймворк Lighthouse [13]. Скориставшись офіційною документацією, встановимо у свій проєкт Lighthouse та сам GraphQL.

Після вдалої установки та налаштувань конфігурації у корні проєкту з’являється нова тека з назвою graphql, яка буде мати у собі файл schema.graphql. У цьому файлі буде прописана більша частина роботи із запитами.

При використанні GraphQL функції методів POST, PUT та DELETE виконують мутації (mutation), а функції методу GET – запити (query). Усі вони працюють із типами, описаними програмістом для кожної із сутностей. Поля описаних сутностей відповідають полям сутностей у базі даних, але дозволяють замість зовнішнього ключа вказувати відразу сутність (або список екземплярів сутності), на яку він посилається, що є дуже зручним у формуванні запитів. Обов’язкові поля позначаються знаком оклику на кінці.

Розглянемо оформлення сутностей на прикладі проєкту.

```
type User {
    id: ID
    name: String
    email: String
    email_verified_at: DateTime
    created_at: DateTime!
    updated_at: DateTime
    role: Role!
    technologies: [Technology]
    avatar: String
    about: String
    company: String
    preferences: String
    unacceptable: String
    api_token: String
    is_banned: Boolean!}

type Art {
    id: ID!
    artist: User!
    art: String!    comment: String
    created_at: DateTime!}

type Review {
    id: ID!
    artist: User!
    employer: User!
    text: String
    mark: Int
    created_at: DateTime!}

type Message {
    id: ID!
    sender: User!
    recipient: User!
    message: String
    created_at: DateTime!}

type Complaint {
    id: ID!
    sender: User!
    cause: String
    is_watched: Boolean!
    is_confirmed: Boolean!
    created_at: DateTime!}

type Ban {
    id: ID!
    user: User!
    moderator: User!
    cause: String
    created_at: DateTime!    end_at: DateTime!}

type Role {
    id: ID!    role_name: String!}

type Technology {
    id: ID!    technology_name: String!}
```

Рисунок 3.1 – Код сутностей у schema.graphql

Query повертає екземпляр (або список екземплярів) сутності і в запиті можна задавати всі поля, які були описані для цієї сутності. Наприклад, для сутності користувача у нашій схемі, оформлення type query буде виглядати так, як на рисунку 3.2.

```
      user(
            "Search by primary key."
            id:    ID    @eq    @rules(apply:    ["prohibits:email",
"required_without:email"])
            "Search by email address."
            email:   String   @eq   @rules(apply:   ["prohibits:id",
"required_without:id", "email"])
          ): User @find
          "List multiple users."
          users(
            id: ID @where(operator: "like")
          ): [User!]! @paginate(defaultCount: 10)
```

Рисунок 3.2 – Код запитів на користувача у schema.graphql

Mutation обов’язково запитує у користувача поля та повертає (це може варіюватися залежно від необхідної програмісту логіки) створений або оновлений екземпляр сутності. Розглянемо на прикладі створення нового користувача (див. рис. 3.3).

```
      type Mutation {
          registerEmployer(
          input: RegisterEmployerInput! @spread
          ): User!

      input RegisterArtistInput{
          name: String!
          email: String!
          password: String!
          password_confirmation: String!
          role_id: Int!    technologies: [Int]!
          avatar: String
          about: String
          preferences: String
          unacceptable: String
      }
```

Рисунок 3.3 – Код додавання нового користувача типу «Роботодавець»

Для перевірки запитів у GraphQL існує графічна оболонка. Після запуску серверу вона є доступною за адресою /graphql. Спробуємо створити нового користувача типу «Роботодавець». Для повернення полів нового екземпляру класу їх треба вказати. В даному випадку, ми запитала ідентифікатор, ім’я та ідентифікаційний токен тільки що створеного користувача.

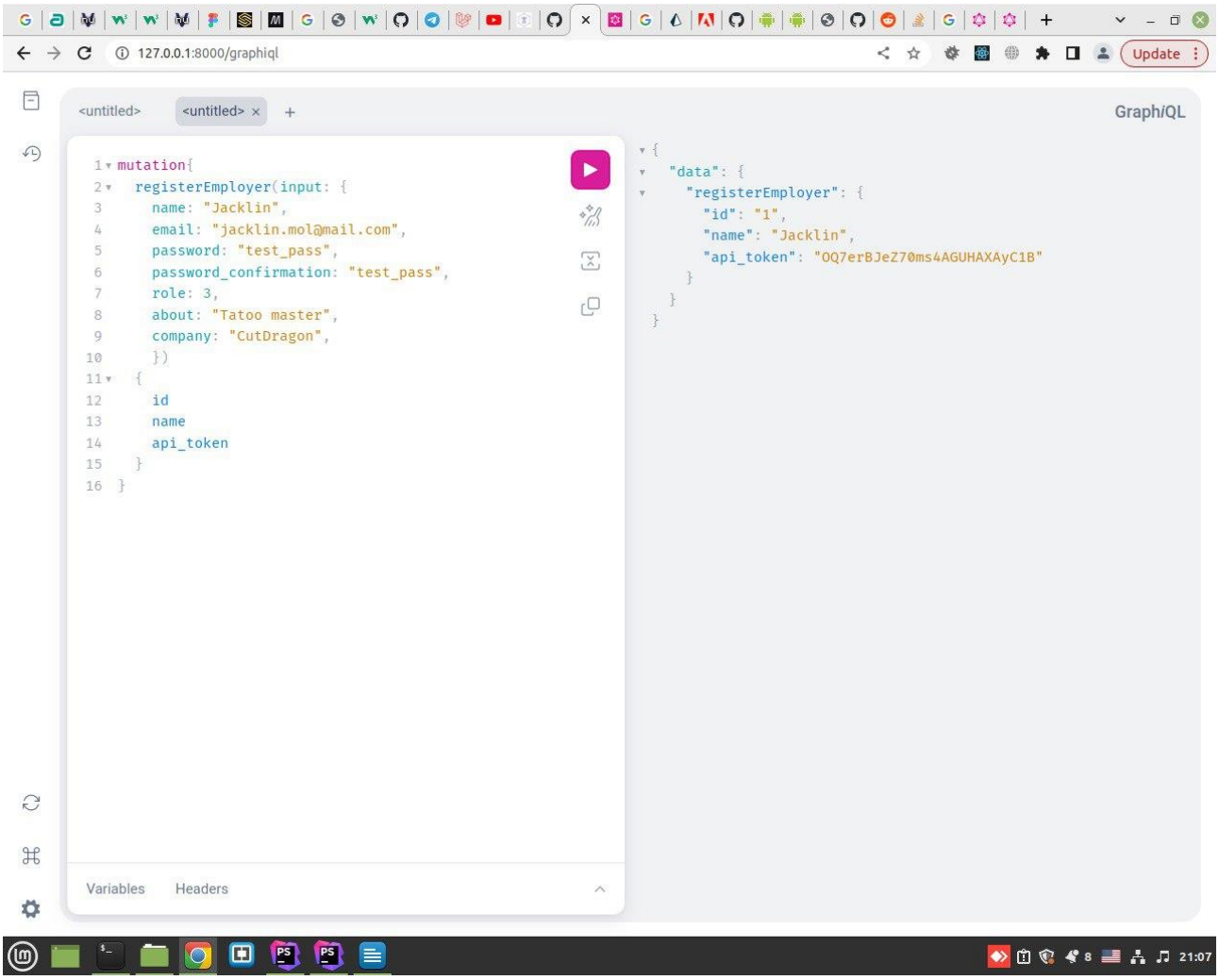

Рисунок 3.4 – Виконання запиту на створення користувача типу «Роботодавець» з графічної оболонки GraphQL

Але код мутації, вказаний у схемі, є лише інтерфейсом для створення, або оновлення сутності. Робота з базою даних описується у файлі мутації, який необхідно створити додатково (див. рис. 3.5). В цьому файлі створюється новий клас із функцією, яка приймає на вхід масив даних args, який містить надану користувачем у запиті інформацію. У тілі функції, за необхідністю, виконується

валідація, перехоплюються помилки та присвоюються полям нової сутності (або старої, що редактується) нові значення. Повертає функція екземпляр класу, поля з якого передаються користувачу у якості відповіді.

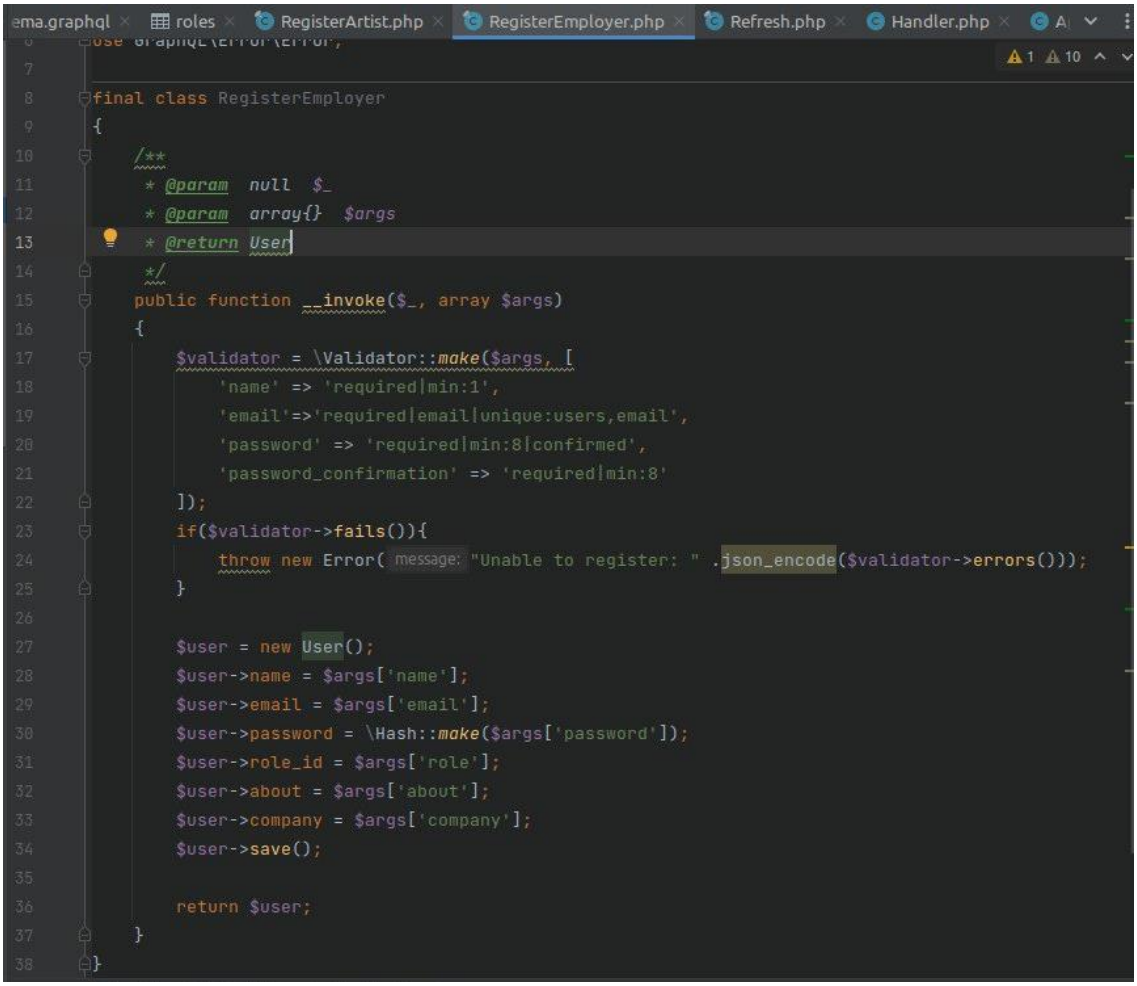

Рисунок 3.5 – Код файлу мутації RegisterEmployer

Кількість створених файлів мутацій відповідає кількості сутностей.

Запити GraphQL від клієнта за допомогою JS майже не відрізняються від запитів REST API.

3.2 Проблемні питання реалізації

Найбільш проблемним у цьому проєкті є питання завантаження ілюстрацій. Є три можливих способи реалізувати завантаження та зберігання

ілюстрацій.

Перший спосіб. Завантажувати файл через GraphQL у поле типу file та зберігати у хмарному сховищі посилання на яке зберігати у відповідне поле таблиці бази даних. Або на сервері переводити файл у бінарний вид та зберігати у полі із типом Blobe. Але це провокує зростання трафіку щонайменше на 30 %.

Другий спосіб. Створити для завантаження файлів REST API ендпоінт. Через нього грузити файл на сервер де вже зберігати його в такий самий спосіб,як у першому варіанті. Завдяки особливостям реалізації REST API це економить трафік та є більш звичним варіантом.

Але оскільки у даній роботі розглядався саме GraphQL, було обрано третій спосіб.

За допомогою пакетів JS завантажене зображення переводилось до виду base64 та передавалось а сервер у виді строки, якай зберігалась у відповідному полі типу String. Однак, ця реалізація підходить лише для текстових проєктів, а для повноцінних краще скористатися REST API та хмарним сховищем.

ВИСНОВКИ

У першому розділі кваліфікаційної роботи було проаналізовано схожі проєкти, які вже існують у мережі Інтернет, та технології , за допомогою яких можна реалізувати сайт-портфоліо із елементами соціальної мережі. За результатами цього аналізу у якості серверного фреймворку було обрано Laravel, для реалізації фронтенду – набір бібліотек React, а для запитів від клієнту до сервера – технології GraphQL.

У другому розділі кваліфікаційної роботи було детально розглянуто предметну область, сформовані вимоги та розроблено ряд діаграм, які відображають основні функції доданку та моменти реалізації. На підставі цих діаграм розроблено низку макетів користувацького інтерфейсу, та виділено перелік компонент для його реалізації.

У третьому розділі кваліфікаційної роботи висвітлено процес створення схеми зв’язку між клієнтом та сервером GraphQL, детально розглянуто користування цією технологією, та розглянуто проблеми реалізації доданку.

Створений сервіс реалізував тільки базові функції та має перспективу розширення. Можливі такі оновлення: поліпшення та полегшення модерації, реалізація сторінки за замовченням після входу в обліковий запис (наразі цією сторінкою являється профіль користувача), можливий розвиток функціональності листування – можливість відправляти файли, підтримка смаглів, розширення функціональності спілкування відео та аудіо зв’язком, можливість користувача оскаржити блокування, тощо.

У ході праці над проектом було підвищено навички роботи із обраними технологіями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація фреймворку Yii. URL: https://www.yiiframework.com /doc/guide/2.0/uk/intro-yii (дата звернення: 06.09.2022).

2. Офіційний репозиторій фреймворку Simfony. URL: https://github.com/symfony/symfony (дата звернення: 06.09.2022).

3. Офіційна документація фреймворку Django. URL: https://www.djangoproject. com/start/overview/ (дата звернення: 06.09.2022).

4. Офіційна документація фреймворку Laravel. URL: https://laravel.com/docs/9.x (дата звернення: 06.09.2022).

5. Офіційна документація JS фреймворку VueJS. URL: https://vuejs.org/guide/introduction.html#the-progressive-framework (дата звернення: 06.09.2022).

6. Офіційна документація JS фреймворку Angular. URL: https://angular.io/guide/what-is-angular (дата звернення: 06.09.2022).

7. Офіційний сайт проєкту React. URL: https://reactjs.org/ (дата звернення: 06.09.2022).

8. Офіційний сайт проєкту GraphQL. URL: https://graphql.org/ (дата звернення: 10.11.2022).

9. Офіційний репозиторій десктопного доданку draw.io. URL: https://github.com/jgraph/drawio (дата звернення: 15.10.2022).

10. Офіційний сайт проєкту Figma. URL: https://www.figma.com/ (дата звернення: 09.10.2022).

11. Офіційний сайт проєкту PHPStorm. URL: https://www.jetbrains.com/phpstorm/ (дата звернення: 03.11.2022).

12. Офіційний сайт розробника JetBrains. URL: https://www.jetbrains.com/ (дата звернення: 06.09.2022).

13. Офіційна документація фреймворку Lighthouse. URL: https://lighthouse-php.com/5/getting-started/installation.html (дата звернення:

10.11.2022).

14. Correa D., Vallejo P. Practical Laravel: Develop clean MVC web applications. Independently published, 2022. 177 p.

15. Lyon W. Full Stack GraphQL Applications: With React, Node.js, and Neo4j. Manning, 2022. 264 p.

16. Buna S., Kendrick (Narrator) K. GraphQL in Action. Manning Publications, 2022. Audiobook.

17. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Independently published, 2018. 288 p.

ДОДАТОК А

Електронна версія діаграм

Посилання на електронну версію діаграм – https://drive.google.com/drive/folders/1zqmZ5ExRUogcqGpe1Wmvx-_GbgLtZlWj?usp=sharing.

ДОДАТОК Б

Альтернативний вигляд сторінок додатку

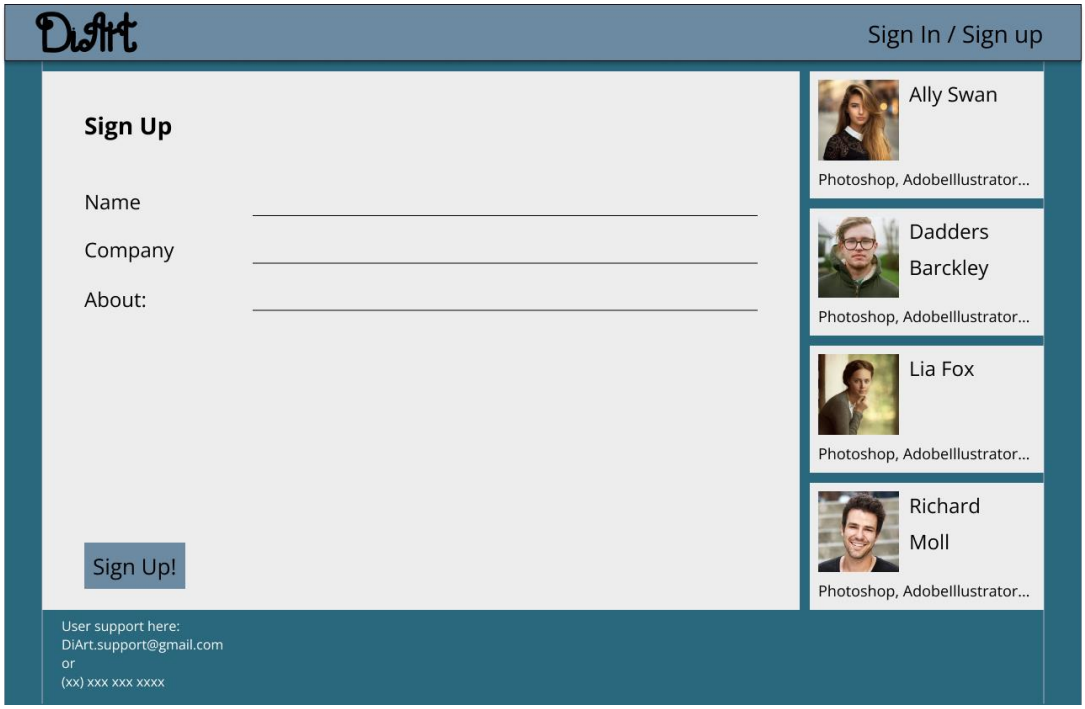

Рисунок Б.1 – Друга сторінка реєстрації роботодавця

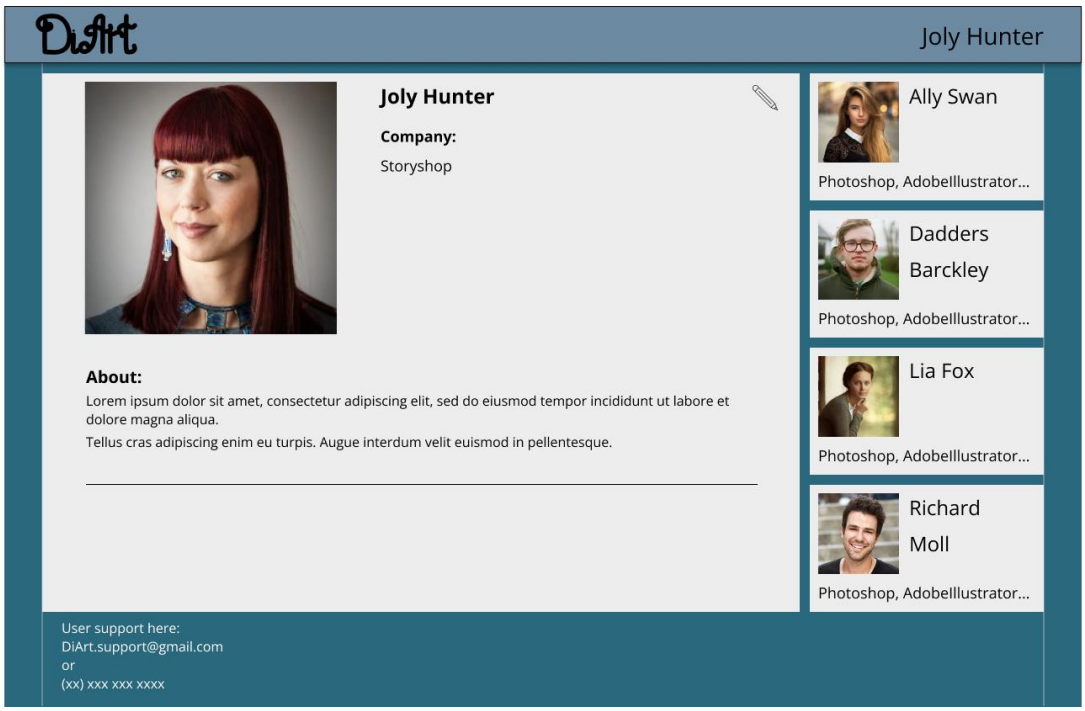

Рисунок Б.2 – Особистий кабінет роботодавця

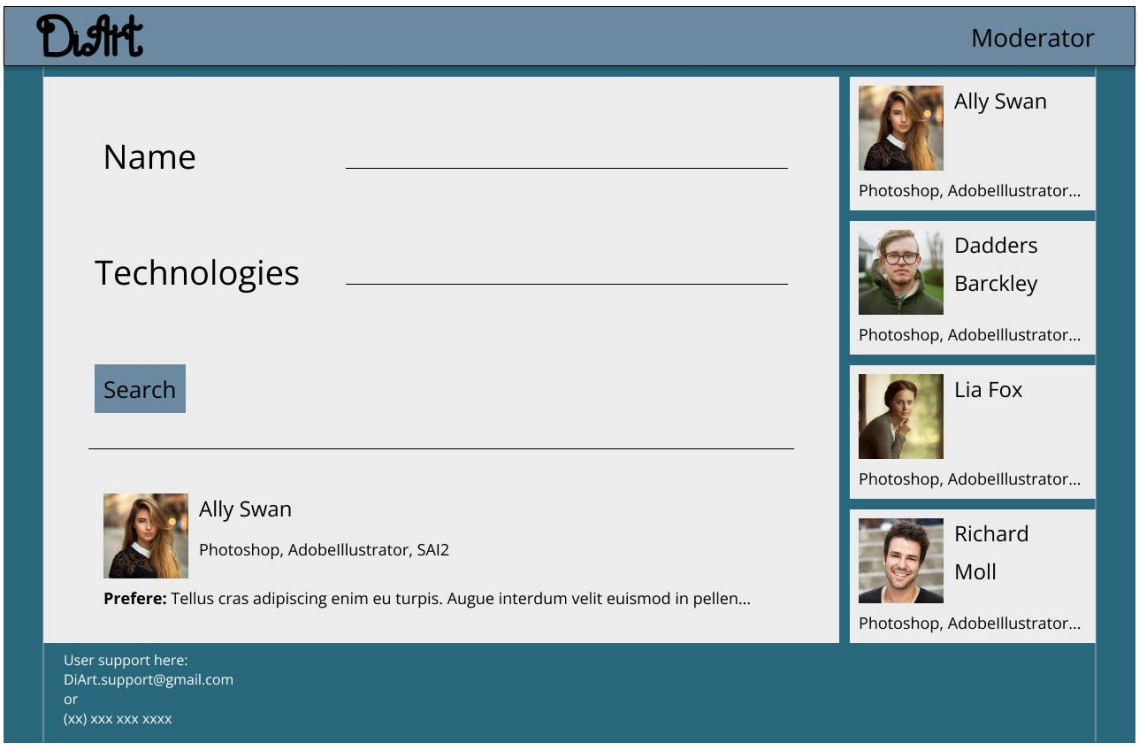

Рисунок Б.3 – Сторінка пошуку

Посилання на документ із мокапами у Figma – https://www.figma.com/file/oQh3UcMUjh6mYhVJrjywJb/Lab2-3?node-id=244%3A260&t=KKjvPOxOHYYLin7u-0.