МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

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

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система пошуку та спонсорування талантів SkillScope»
здобувачки групи ІН – 01 Грищенко Олександри Сергіївни

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

Олександра ГРИЩЕНКО
_____
(підпис)

Керівник
асистент кафедри комп'ютерних наук,
кандидат фізико-математичних наук Ольга ШУТИЛЄВА _____
(підпис)

Консультант
доцент кафедри іноземних мов та лінгводидактики
кандидат філологічних наук Людмила ГНАПОВСЬКА _____
(підпис)

Суми – 2024

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

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

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Грищенко Олександри Сергіївни

1. Тема роботи: «Інформаційна система пошуку та спонсорування талантів SkillScope» затверджую наказом по СумДУ від «23» травня 2024 року №0570-VI
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Розробка структури, функціоналу вебсайту та вибір програмних засобів реалізації. 3) Програмна реалізація інформаційної системи пошуку та спонсорування талантів 4) Аналіз та тестування результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «1» квітня 2024 р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	до 25.04	
2	*Огляд аналогічних продуктів та формування додаткових вимог до проєкту*	до 27.04	
3	*Проєктування та прототипування функціоналу та структури системи, розробка бізнкс-логіки*	до 1.05	
4	*Розробка інтелектуальної системи інформаційної системи пошуку та спонсорування талантів та бази даних до неї*	до 15.05	
5	*Аналіз та тестування отриманих результатів*	до 19.05	
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*	до 20.05	

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

АНОТАЦІЯ

Записка: 96 стор., 45 рис., 3 табл., 3 додатки, 25 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі підтримки українських та світових митців та творців контенту шляхом розробки відповідної вебплатформи, що об'єднує функціонал соціальної мережі та краудфандингової платформи.

Об’єкт дослідження — система для пошуку та спонсорування талантів.

Мета роботи — розробка вебсайту, що дозволяє талантам залучати кошти та отримувати визнання, використовуючи внутрішню валюту для оцінки та підтримки їх творчих доробків.

Методи дослідження — методи проєктування вебзастосунків, алгоритми взаємодії користувачів в соціальних мережах, технології краудфандингу та інструменти створення інтерфейсів користувача.

Результати — розроблено вебплатформу «SkillScope», яка об'єднує талантів та спонсорів, дозволяючи останнім оцінювати пости митців за допомогою внутрішньої валюти Coins. Система дозволяє створити профіль користувача та редагувати його, надає можливість створювати та поширювати пости, робити фінансові внески спонсорам, купуючи внутрішню валюту, та виводити внутрішню валюту талантам на банківську картку. Проведено тестування всіх можливих сценаріїв використання вебсайту різними категоріями користувачів.

ІНФОРМАЦІЙНА СИСТЕМА, СПОНСОРУВАННЯ, ДОНАТ, ТАЛАНТ, СПОНСОР, ВНУТРІШНЯ ВАЛЮТА, КРАУДФАНДИНГОВА ПЛАТФОРМА, JAVASCRIPT, REACT, FIRESTORE.

ЗМІСТ

ВСТУП

Сучасний світ стає все більше цифровим, і це відображається на усіх сферах життя, у тому числі культурі і мистецтві. Пандемія, війна та інші незгоди зовнішнього світу призвели до того, що багато робіт перейшли у онлайн, разом з замовниками і виконавцями. Тим паче системи для донатів разом з донатами стали незамінними для допомоги армії у ці сумні часи. Після повномасштабного вторгнення існує тенденція до відторгнення усього російського, в тому числі російських програмних продуктів, тому зараз в Україні з'явилась ніша, вільна для заповнення українськими додатками. Після того як творці контенту втратили значні контракти, зараз існує нагальна потреба у зручних інструментах для підтримки українських створювачів контенту: блогерів, художників, музикантів тощо.

Актуальність. У кінці 2022 року спостерігалась велика популярність NFT токенів (унікальних витворів цифрового-мистецтва, які можна купити за реальні кошти) та платформ для донатів, таких як Kickstarter, Buy Me A Coffee або українського аналогу Donatello – платформа донатів для підтримки українських авторів, творців, блогерів. Тому ідея створення такого програмного продукту досить не нова, але все ще актуальна.

З діджиталізацією світу та перенесенням багатьох професій у online-простір та створення нових професій таких як блогер, content-creator, з'явилася потреба у монетизації їх робіт та витворів через мережу Інтернет, у тому числі це дуже важливо для талантів-початківців, які тільки почали свій шлях у кар'єрі і шукають зацікавлену аудиторію, яка у перспективі зможе фінансово підтримувати їх роботу. Тому постають питання: як саме таланти можуть знайти платоспроможну аудиторію? Як артисту монетизувати свої роботи? Як поширити їх серед світу?

Об'єкт дослідження. Тому для створення дипломної роботи об'єктом дослідження було обрано вебсайт, що дозволяє артистам залучати кошти та отримувати визнання, має потенціал вплинути на життя багатьох молодих

талантів та сприяти розвитку культури в Україні і світі. Вебсайт, який реалізує найкращі практики сайтів зі схожою тематикою та врахує і виправить їх недоліки – «SkillScope». «SkillScope» – це вебплатформа (міні-соціальна мережа), яка має на меті залучити митців («талантів») та людей («спонсорів»), які зможуть оцінювати («спонсорувати») їх роботи («пости»), використовуючи внутрішню валюту («Coins»).

Гіпотеза. Створення вебплатформи для пошуку талантів та їх спонсорування сприятиме активізації культурного життя, фінансової підтримки талановитих авторів та розвитку креативних індустрій.

Новизна. Новизна роботи полягає в поєднанні функціоналу соціальної мережі та краудфандингової платформи, а також використання внутрішньої валюти для оцінки та підтримки творчого доробку творчих досягнень користувачів, яку в перспективі можна вивести на картку як реальні гроші. Крім того, унікальний підхід до створення інтерфейсу і взаємодії з користувачами може виділити проєкт на ринку та забезпечити йому конкурентну перевагу.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, розробки структури вебсайту та вибору програмних засобів реалізації, програмної реалізації, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Інформаційний огляд

Сучасний ринок дуже гнучкий до змін у суспільстві і здатний підлаштовуватись під попит і пропозицію. Ще зовсім нещодавно ніхто і не міг уявити, що витвори мистецтва, будь-то цифрова копія гри або картини, які зовсім не існують в реальності, зможуть продаватись за реальні гроші. Але який попит – такі і рішення.

Люди, робота яких пов’язана з культурою, мистецтвом та креативними індустріями були надзвичайно уразливою категорією осіб, що постраждали під час пандемії COVID-19. Це легко пояснити, бо саме їх праця напряму пов’язана зі схваленням суспільства, людей, які могли прийти подивитись на їх роботи або навіть придбати їх фізичні копії напряму з рук власника, як от під час арт-аукціонів. Тому почала відбуватись так звана «цифрова трансформація» діячів мистецтва – переведення здобутків артистів у цифровий онлайн формат, де їх аудиторією стають не тільки локальні поціновувачі їх творчості, а й люди з усього світу, що дало змогу залучити ще більше фанатів, а з ними досягнути ще більшого рівня монетизації. Навіть Україну не оминув цей тренд, і з 2020 року діє онлайн-виставка українських митців «Strange time», яку розпочали у відповідь на ізоляцію під час пандемії і де більшість робіт існують лише у цифровій копії [1].

Вже сьогодні ми можемо спостерігати, як контент доволі швидко трансформується в цифровий, який ми звикли сприймати кожен день: книги та музика стають електронними, художники більше можуть не брати до рук пензля, для того щоб намалювати картину. Тому постає питання як продавати та просувати те, чого не існує в матеріальному світі? Відповідь – глобальні платформи цифрового контенту, такі як YouTube – для розміщення відео, Instagram – для розміщення фото, Spotify – для розміщення музики, Steam – для продажу цифрових копій ігор тощо. Багато з подібних компаній – монополісти

та не соромляться брати великий відсоток з заробітків артиста, що ставить його у скрутне становище і окрім основної платформи діяльності створювачі контенту змушені звертатися до сторонніх – до платформ краудфандингу – такої як «SkillScope», де їх фанати можуть залишити грошову допомогу напряму: від «спонсора» до «таланта» без посередника.

1.1.1 Культура донатів

Донат (від англійського «donate») – це волонтерський внесок або пожертвування, яке зазвичай роблять в інтернеті. Термін широко використовується в контексті інтернет-сторінок, блогів, стрімінгових платформ та інших ресурсів, де користувачі можуть надіслати гроші для підтримки контент-творців, вебсайтів, благодійних організацій, проєктів тощо. Переказ коштів (донат) проходить за допомогою наявних онлайн-інструментів (залежно від платформи та людини чи організації, яка збирає донати) [2].

Онлайн-донати часто здійснюються з метою подяки за створений творцем контент. Зазвичай донат блогерам, музикантам, художникам та іншим творчим людям сприймається як матеріальна допомога задля підтримки їхньої подальшої творчої діяльності та створення нового контенту.

Існує вираз «маленьких донатів не буває». Навіть якщо кожний підписник, або хоча б більша частина аудиторії задонатить по одній гривні, контент-творець вже отримає достатню суму грошей, щоб продовжувати свою кар'єру. На такому правилі і ґрунтуються принципи роботи більшості краудфандингових платформ.

Оскільки вебдодаток написаний англійською мовою, увага у цьому розділі буде сконцентрована саме на іноземній допомозі. Збройна агресія Російської Федерації об’єднала людей по усьому світу у питаннях пов’язаних з фінансуванням потреб армії та українців у ці складні часи. Небайдужі іноземці вже другий рік продовжують допомагати українцям у їх боротьбі проти агресора, тому краудфандингові платформи стали майданчиком не лише для спонсорування створення якісного контенту, але і на потреби війни, біженців і окремих громадян.

Деякі іноземці, зокрема з США та країн Балтії, приєдналися до руху, збираючи вдома кошти для українських військових. У 2022 році, після років рекордних оборонних бюджетів і реформ, українська армія оснащена набагато краще, ніж у 2014 році, але постачання все ще далекі від ідеальних. Цивільні благодійні організації та волонтерські проєкти допомоги все ще існують, забезпечуючи війська тим, чого їм не дає офіційна військова система. Іноземні громадяни можуть допомогти, пожертвувавши гроші на такі проєкти [3]. З моменту початку повномасштабної війни лише на рахунки НБУ та у три найбільші фонди країни українці та іноземці задонатили понад 33,96 млрд грн через краудфандингові платформи [4].

1.1.2 Краудфандингові платформи

Краудфандингова платформа – онлайн платформа, на якій розміщуються ідеї, для яких відбувається збір коштів серед інтернет-користувачів. Як правило, спонсорам гарантується нематеріальна винагорода у вигляді готового продукту або інших подарунків, які мають безпосереднє відношення до проєкту. Майданчик бере на себе забезпечення багатьох аспектів, в тому числі фінансових і юридичних, а також допомагає в просуванні і полегшує взаємодію всіх учасників [5].

Концепція «краудфандингу» виникла в 2003 році, коли Браян Камеліо, бостонський музикант і комп'ютерний програміст, запустив платформу «ArtistShare», яка дозволяла колегам-музикантам шукати фінансування у своїх шанувальників і доброзичливців. Пізніше ця платформа перетворилася на форум для збору коштів для кіно/відео та фотографічних проєктів, на додачу до музики. Успіх цього форуму призвів до появи більш популярних сайтів на основі винагород, таких як Indiegogo та Kickstarter у 2008 та 2009 роках відповідно [6].

Гроші можуть збиратися на різноманітні цілі: фінансова підтримка артиста, музичний альбом, мобільна гра, створення нового відео для платформи «YouTube» тощо. У залежності від моделі краудфандингу, його поділяють на:

1. Тип «усе або нічого» – контент-творець встановлює суму-ціль, яку необхідно зібрати до певного строку. Якщо в повному обсязі суму зібрати не вдалось, усі вкладені гроші повертаються інвесторам;

2. Тип «вічне фінансування» – на відміну від типу «усе або нічого» кінцевого дедлайну(або навіть цілі) на збір немає і він залишається активним поки автор не збере повну суму або не закінчить збір на половині і не забере гроші (приклад платформи – «SkillScope», на якій дедлайнів на донати немає і збір може тривати вічно);

3. Тип «залишити все» – повністю повторює сенс типу «усе або нічого» з відмінністю, що автор забирає гроші, навіть якщо повна сума збору не досягнута.

У 2021-2022 роках всесвітні медіа заполонили заголовки про те, як на найпрестижніших аукціонах світу за велику кількість грошей продаються картини, яких не існує у реальному світі. Виглядає як утопія, але це і є нова течія NFT-мистецтва. Це твори сучасного мистецтва, які створені художниками, музикантами, письменниками тощо за допомогою цифрових технологій [7]. NFT – це унікальний токен, цифровий сертифікат авторської роботи, що підтверджує її унікальність та закріплює право власності. Такі токени допомагають залучити додаткові кошти шляхом продажу або запровадження монетизації робіт, які існують лише у цифровому світі, шляхом отримання відсотків з перепродажу. Якщо картині надано NFT-токен, її копії будуть нічого не варті, тому з появою таких токенів стало можливим закріплювати право власності, що дозволило творам цифрового мистецтва не відставати у ціні від предметів традиційного мистецтва.

Отже, ідея заробітку на цифрових копіях предметів мистецтва досить нова і все ще актуальна, люди готові платити реальні кошти, будь то повне придбання цифрової копії або лише підтримка автора у форматі донатів.

1.2 Огляд існуючих рішень

1.2.1 Платформа фінансування Kickstarter

Kickstarter [8] – це платформа фінансування, спрямована на допомогу творчим проєктам у запуску. Вона повністю заснована на краудфандингу, тож пожертви від широкої публіки підживлюють динамічні нові ідеї, що розміщені на сайті [9], головна сторінка якого зображена на Рисунку 1.1. Одна з рис платформи – вона створена лише для некомерційних організацій, і її політика не підтримує відкриття зборів на благодійність. Kickstarter – типовий приклад платформи, що діє за принципом «усе або нічого»: творець може зібрати свої кошти, лише якщо він досягне своєї цілі фінансування до кінцевого терміну. Якщо він не досягне мети вчасно, гроші повертаються назад спонсорам.

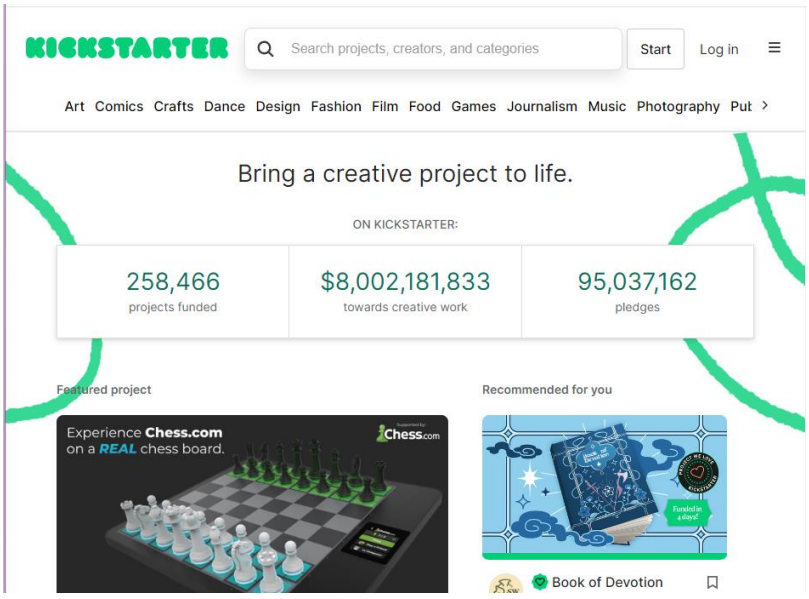

Рисунок 1.1 – Головна сторінка платформи Kickstarter

Творці спонсорованого проєкту створюють сторінку для відображення деталей свого творіння, де показують прототипи, свої напрацювання та ідеї, встановлюють мету фінансування та кінцевий термін. Також прийнято встановлювати нагороди для тих, хто донатить, щоб заохотити їх, наприклад такі

нагороди як: розміщення імені спонсора десь у готовому проєкті, одяг під стилістику проєкту або зустріч з творцем (рис. 1.2).

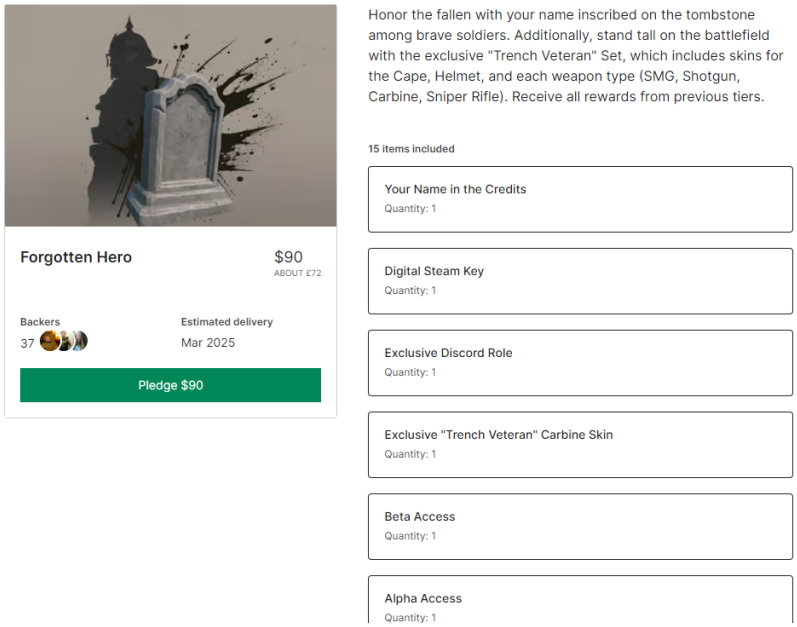

Рисунок 1.2– Типові нагороди спонсорам Kickstarter

1.2.2 Buy Me A Coffee

Buy Me A Coffee [10], головну сторінку якого можна побачити на Рисунку 1.3, пропонує впливовим особам, фрілансерам і різним артистам і творцям контенту платформу для отримання підтримки, прийняття пожертвувань, створення членства та кращого зв’язку зі своїми шанувальниками. Вебсайт використовує віртуальну валюту – чашку кави, яка буде коштувати стільки, скільки зазначить творець. Нажаль платформа не надає можливості обирати яку точну суму хоче задонатити спонсор: вона в будь-якому разі має бути кратна одній чашці кофе за визначеною творцем ціною. Автор має створити сторінку, додати описи, визначити ціну за одну каву та способи оплати. Прихильники автора або проєкту можуть купити скільки завгодно кави і збір не обмежений у часі - тип «вічне фінансування». Головна особливість сайту – можливість додавання щомісячних чи річних підписок на свою сторінку певного рівня за певну суму, наприклад «золото», «срібло» або «бронза». Творці можуть

додати винагороди, які можуть запропонувати покупцям підписок, додати їм привітання тощо. Також можна продавати цифрові продукти на своїй сторінці.

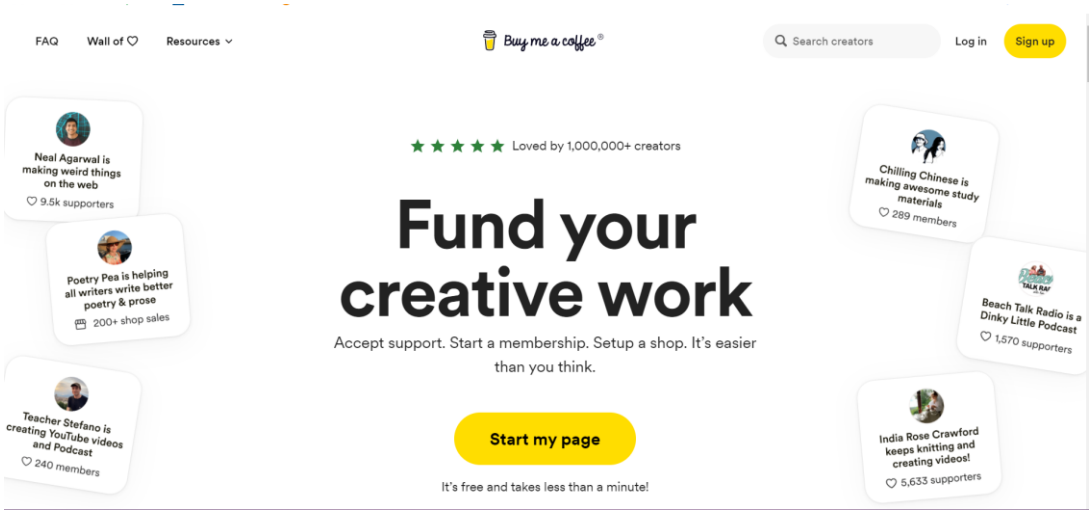

Рисунок 1.3 – Головна сторінка платформи Buy Me A Coffee

1.2.3 Donatello

Donatello [11] – платформа збору донатів від українських розробників. Це дуже простий вебсайт, який працює за принципом маркетплейсу. Автори мають можливість створити власні донат-сторінки на нашій платформі. Посилання на свою донат-сторінку автори публікують на інших онлайн-сервісах і у соцмережах [12]. На відміну від платформ, що були розглянуті вище, Donatello має дуже обмежений функціонал: відсутність підписок, внутрішньої валюти та можливості викладення авторського контенту на сторінці. Все, що може зробити автор – це обрати ціль збору та валюту донатів. Все, що доступно спонсору – простий грошовий донат з коротким текстовим повідомленням. Перевагою може стати приємний та лаконічний дизайн сайту (рис. 1.4).

Провівши аналіз трьох вебсайтів зі спорідненою тематикою до SkillScope, було знайдено як переваги, так і недоліки. Найбільш помітними недоліками виявились:

- Робота платформи за принципом «усе або нічого», що неуможливлює отримання грошей авторами у короткі сроки, що може призвести до закриття певного проєкту, на який йде збір, раніше, ніж збереться потрібна сума грошей.

Особливо цей недолік критичний для авторів-початківців, які тільки шукають свою аудиторію і не мають багатьох підписників та фанатів;

- Неможливість донатити стільки, скільки хоче сам спонсор, бо мінімальна сума донату виставляється кожним автором окремо;

- Перевантаженість інформацією та застарілий дизайн, велика кількість проєктів на головній сторінці, що відволікає спонсорів від їх головної мети – монетизувати проєкти;

- Неможливість виставити більше одного збору за раз;

- Відсутність віртуальної валюти, що ускладнює і збільшує час процесу спонсорування, оскільки кожен раз під час донату спонсор змушений вносити дані банківської карти, щоб зробити лише один донат за раз.

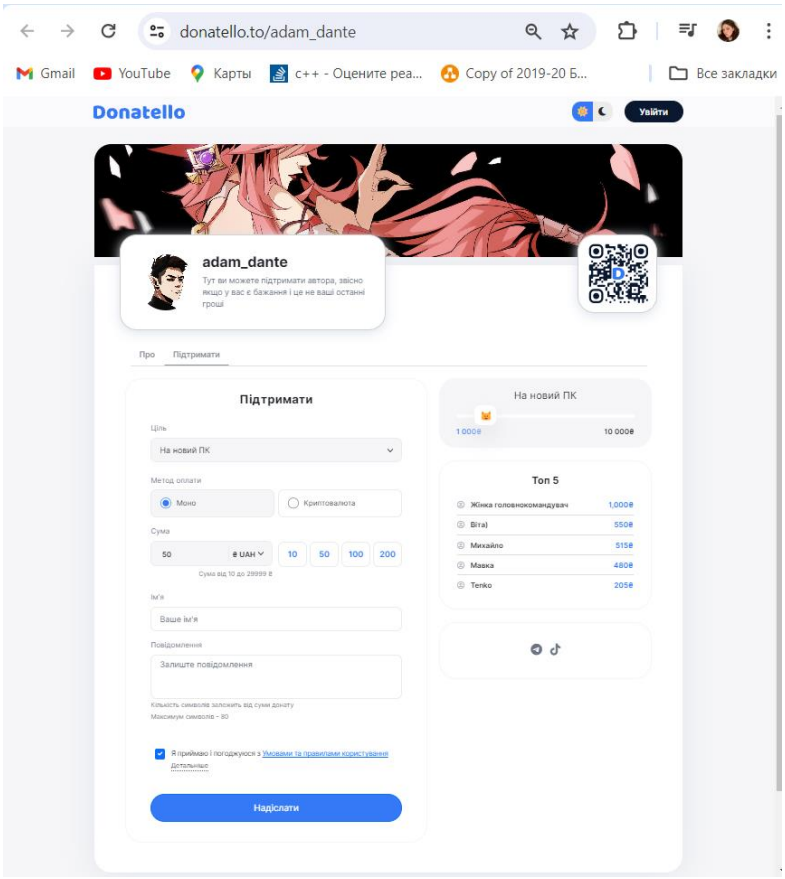

Рисунок 1.4 – Лаконічний дизайн профілю автора на платформі Donatello

Порівняємо детальніше інформацію про сайти краудфандингу та проведемо їх порівняльну характеристику (Таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика існуючих аналогів

Характеристика	**Kickstarter**	**Buy Me A Coffee**	**Donatello**
Тип спонсорування	«все або нічого»	«вічне фінансування»	«залишити все»
Наявність внутрішньої валюти	-	-	-
Відсутність комісії платформи	-	-	-
Простий і лаконічний дизайн	-	+	+
Відсутність обмеження в сумі збору або мінімального одиничного донату	-	-	+
Відсутність обмежень в кількості зборів одночасно	-	+	-
Відсутність обмежень для донатів з усього світу	+	+	-

Отже, після проведеного аналізу наявності певних важливих функцій серед популярних закордонних та вітчизняних платформ для спонсорування, було вирішено розробити сайт, який перейме найкращі практики та не буде містити недоліки, виявлені під час аналізу. Вебплатформа SkillScope матиме всі переваги даних сайтів, а також матиме приємний і унікальний дизайн та інтуїтивно зрозумілий інтерфейс користувача.

1.3 Постановка задачі

У результаті проведеного інформаційного огляду теми, поставлено мету цієї роботи – проєктування та розробка інформаційної системи пошуку та спонсорування талантів зі зручним інтерфейсом користувача та механізмом

фінансування талантів. Для досягнення поставленої мети у ході виконання кваліфікаційної роботи необхідно виконати наступні завдання:

1. Вивчити теорію програмування вебсайту;

2. Провести аналіз тематичної літератури і інформації з теми монетизації цифрових витворів мистецтва та спонсорування творців контенту;

3. Провести аналіз сайтів-аналогів для формування додаткових вимог до проєкту;

4. Розробити користувацький інтерфейс (UI) та наповнення вебсайту;

5. Розробити бізнес-логіку, якій буде слідувати інформаційна система;

6. Створити базу даних з потрібної для роботи системи таблицями;

7. Виконати програмну реалізацію вебсайту разом з фінансовою; системою та механізмом спонсорування талантів спонсорами;

8. Провести тестування інформаційної системи користувачами, зареєстрованими під різними ролями.

2 РОЗРОБКА СТРУКТУРИ ВЕБСАЙТУ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Створення MindMap для проєктування структури вебсайту

MindMap – це графічний інструмент, який допомагає організувати і структурувати інформацію, ідеї та концепції. За допомогою деревовидної структури, можна зрозуміти зв’язки між компонентами або структурними одиницями.

За допомогою MindMap на Рисунок 2.1 було спроєктовано логіку і структуру вебсайту «SkillScope», яка буде імплементована під час розробки інформаційної системи. Один компонент карти – одна сторінка вебресурсу.

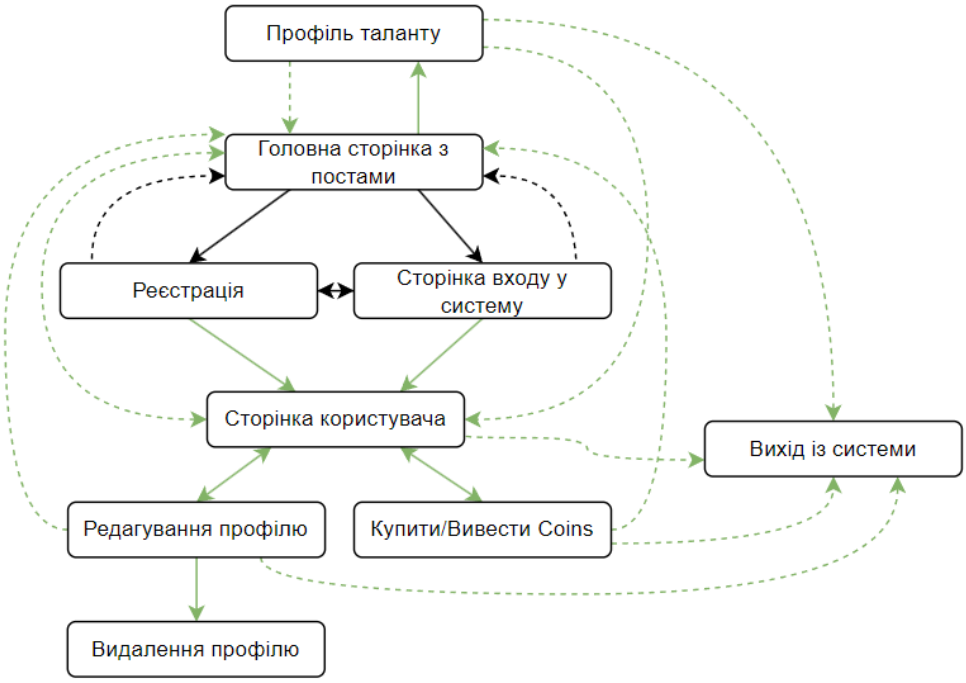

Рисунок 2.1 – MindMap навігації по інформаційній системі

Основна навігація здійснюється за допомогою відповідних кнопок, вона позначена цільними стрілками на зображенні. Додаткова навігація може здійснюватись через кнопки хедера вебсайту, такі переходи на MindMap

позначені пунктирними стрілками, з яких зелені – це ті, які потребують попереднього входу у систему.

Після переходу на сайт, відкривається головна сторінка інформаційної системи – «Main page», звідки і можна здійснювати усі переходи між сторінками. Щоб здійснювати будь-які дії в інформаційній системі, у тому числі перегляд профілів користувачів, редагування свого профілю та будь які фінансові операції, потрібно або зареєструватись або увійти у систему.

2.2 Створення діаграми варіантів використання на основі бізнес-логіки проєкту

Сайт містить різний функціонал та наповнення в залежності яку роль має користувач: талант чи спонсор.

Система діє за такими правилами:

1. Лише користувач з роллю «спонсор» може фінансувати користувачів з роллю «талант», за умови якщо «спонсор» має хоча б один «Coins» на рахунку;
2. Користувач з роллю «спонсор» може спонсорувати пост нескінченну кількість разів, витрачаючи одну одиницю внутрішньої валюти за один донат;
3. Лише користувач з роллю «талант» може виставляти пости;
4. Лише профілі користувачів з роллю «талант» можуть висвічуватись на головній сторінці;
5. Користувачі обох ролей можуть заходити на профілі користувачів з роллю «талант» за умови, що вони авторизовані;
6. Лише користувачі з роллю «спонсор» можуть купувати внутрішню валюту;
7. Лише користувачі з роллю «талант» можуть виводити внутрішню валюту на картку за умови, що їх баланс більше або дорівнює кількості внутрішньої валюти, яку вони прагнуть вивести;
8. Баланс користувача з роллю «талант» вираховується як сума усіх «Coins» з його постів; при донаті внутрішня валюта напряму додається як до

суми «Coins» на пості, так і до балансу користувача, що дозволяє не порушувати цілісність системи при виведені внутрішньої валюти «талантом» на картку;

9. Користувачі обох ролей можуть редагувати та видаляти свій профіль.

Оскільки існує дві ролі, під якими можуть бути зареєстровані користувачі, для того щоб продемонструвати залежність функціоналу від конкретної ролі було обрано діаграму варіантів використання (Рисунок 1.2):

Рисунок 1.2 – Діаграма варіантів використання вебресурсу SkillScope

Діаграма варіантів використання – діаграма, що дозволяє продемонструвати всі сценарії взаємодії між ролями (або акторами) та системою, причому зображення функціональних вимог (або бізнес-логіки) відбувається саме зі сторони користувача системи. Найпоширеніша ситуація, в якій можна застосувати цю діаграму – це моделювання поведінки підсистеми або системи в цілому, тому вона ідеально змогла допомогти спроєктувати логіку роботи інформаційної системи [13].

2.3 Реалізація механізму спонсорування в інформаційній системі

Основний функціонал інформаційної системи «SkillScope» побудований навколо процесу спонсорування талантів спонсорами, який зображений на Рисунок 2.2, де дії «спонсора» показані зеленим, а «таланта» помаранчевим.

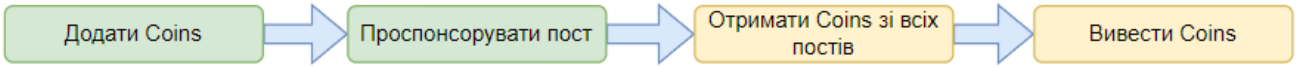

Рисунок 2.2 – Етапи переходів внутрішньої валюти «Coins»

Щоб залишити одну одиницю внутрішньої валюти на посту користувача з роллю «талант», спонсор перш за все має обміняти реальні гроші на внутрішню валюту «Coins» за допомогою сторінки «Payment» натиснувши кнопку «Buy Coins» в особистому профілі. Ця сторінка представляє собою симуляцію введення деталей банківської карти та введення бажаної кількості внутрішньої валюти. Тільки після успішного введення деталей картки, процедура вважається завершеною та «Coins» додаються на акаунт спонсора. Щоб проспонсорувати таланта, спонсор може зайти на його профіль та задонатити одну одиницю валюти на один пост таланта за раз. Внутрішня валюта напряму додається як до суми «Coins» на пості, так і до балансу користувача, що дозволяє не порушувати цілісність системи. Після цього валюта віднімається з акаунту спонсора та додається на акаунт таланта як сума усіх «Coins» з усіх його постів. Після цього у своєму профілі талант може натиснути кнопку «Вивести Coins» («Withdraw Coins»), ввести дані банківського рахунку та кількість валюти, що має бути виведена. Після цього зазначена кількість валюти віднімається з акаунту таланту, але не з його постів, що дозволяє залишити статистику спонсорування постів спонсорами.

2.4 Відображення ключових процесів у діаграмах діяльності

Додатково до діаграми варіантів використання використовуються в розрозробці діаграми діяльності, що представлені на Рисунках 2.3-2.10, щоб розкрити та описати основні процеси та їх послідовності, що відбуваються в інформаційній системі більш детально, оскільки діаграма варіантів використання показує лише процеси, що можуть відбуватися в системі, а не їх послідовність.

Діаграми діяльності є однією з п’яти діаграм в UML для моделювання динамічних аспектів систем. Діаграма діяльності – це блок-схема, яка показує потік від діяльності до діяльності [14].

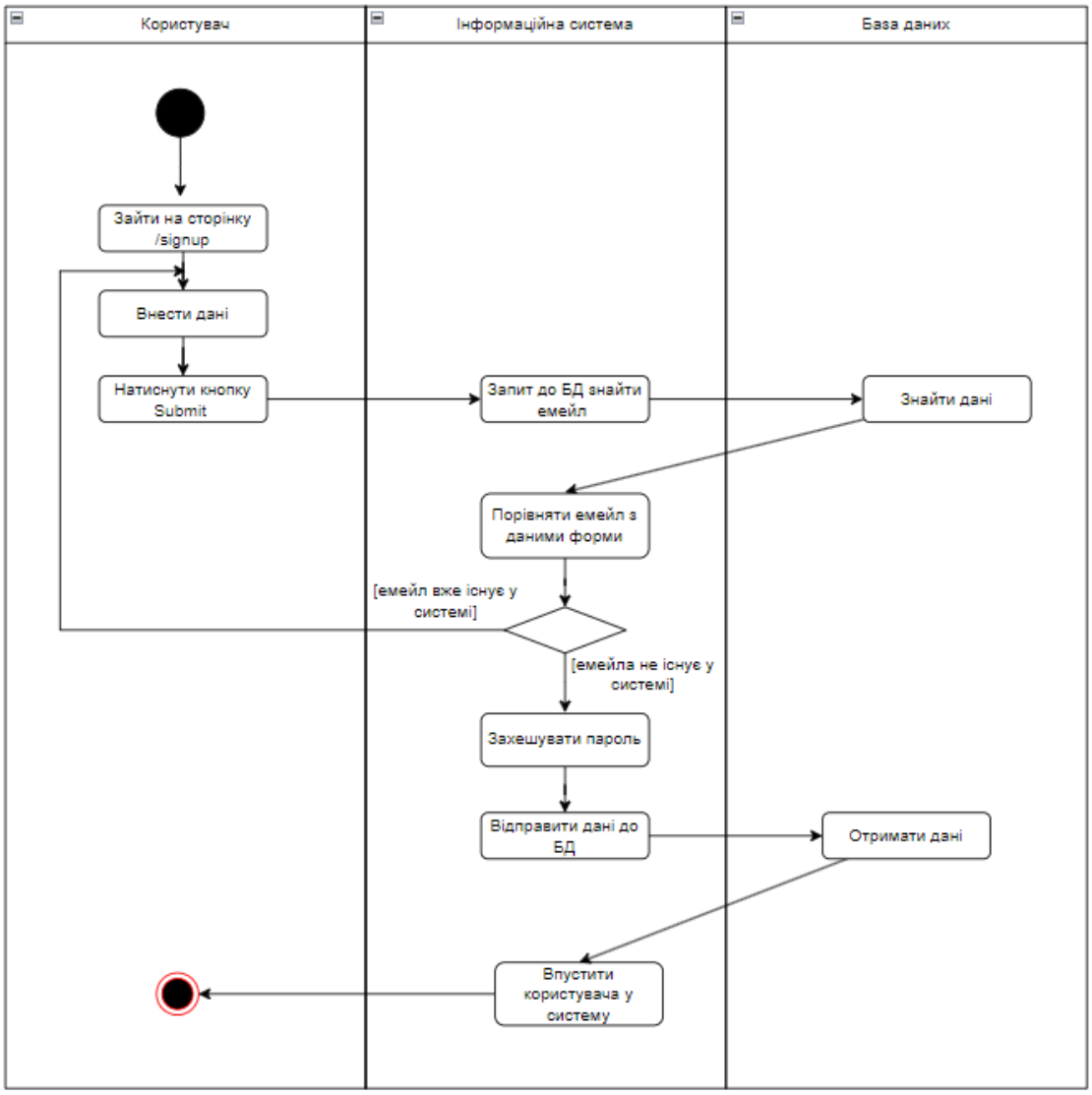

Рисунок 2.3 – Діаграма діяльності реєстрації у системі

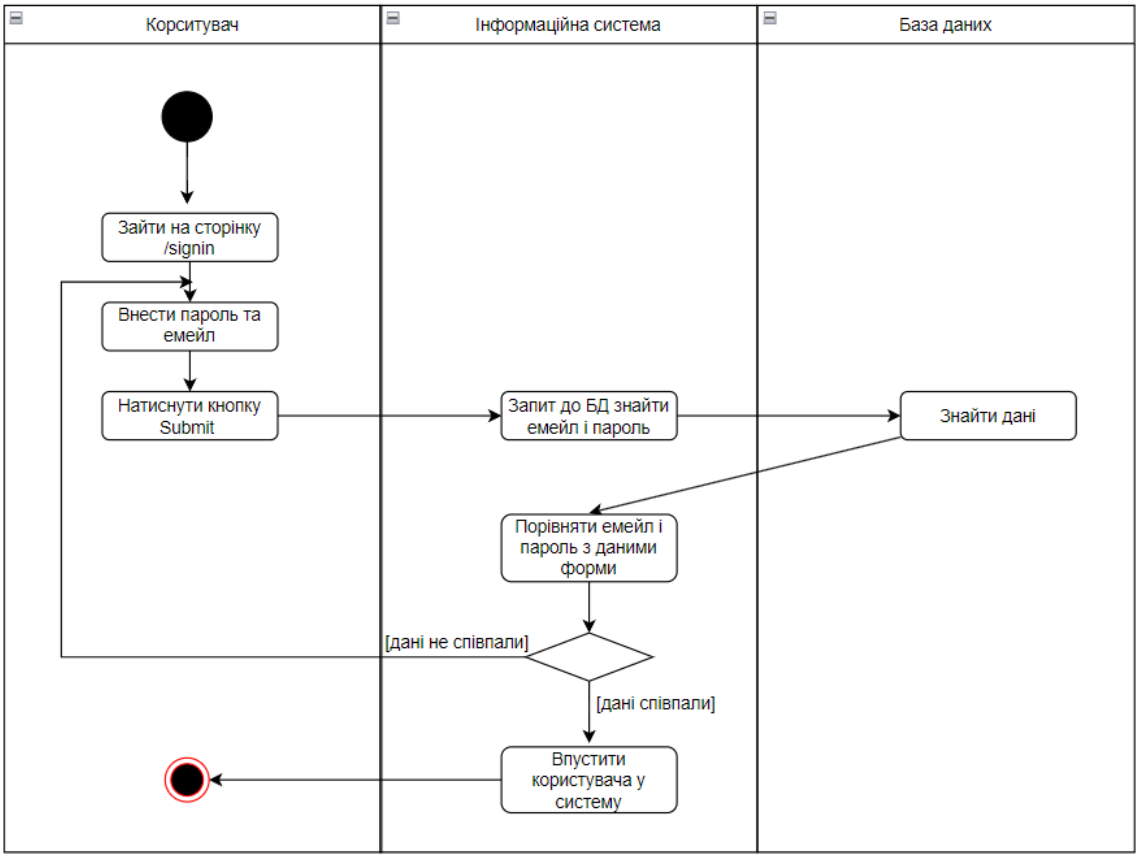

Рисунок 2.4 – Діаграма діяльності входу у систему

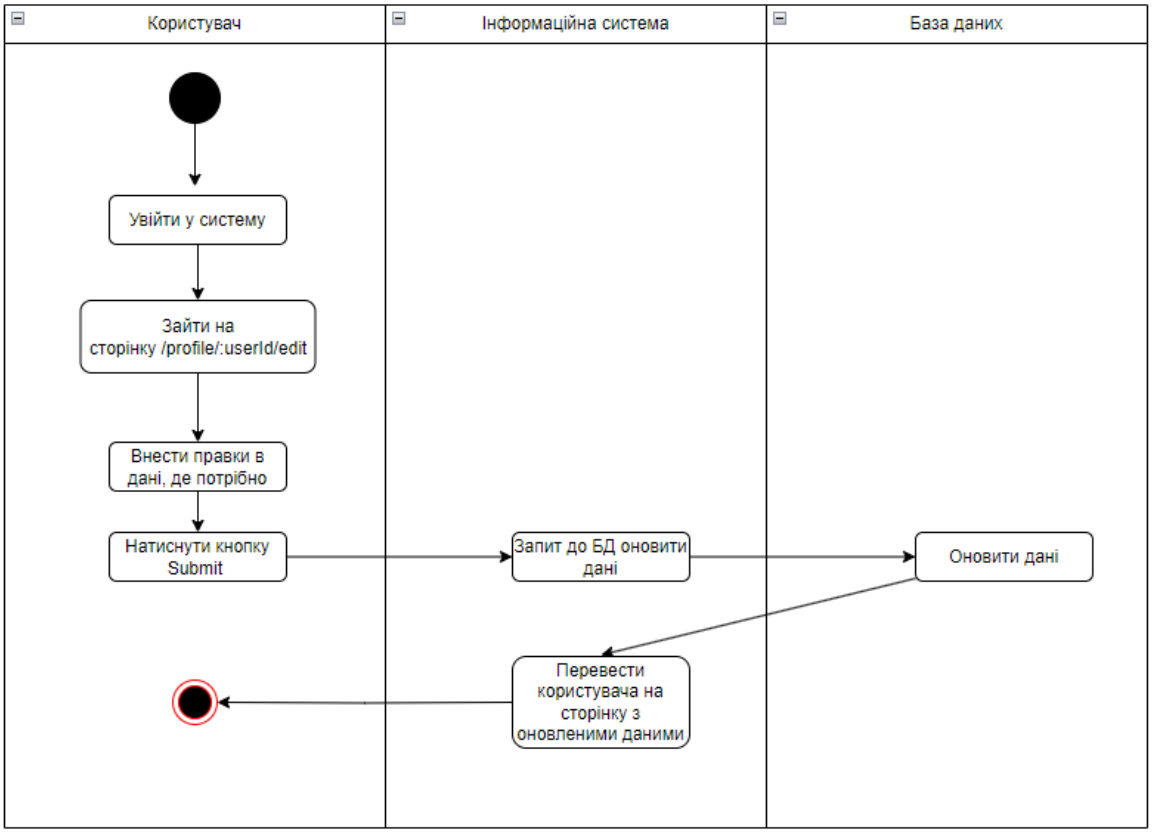

Рисунок 2.5 – Діаграма діяльності редагування профілю користувача

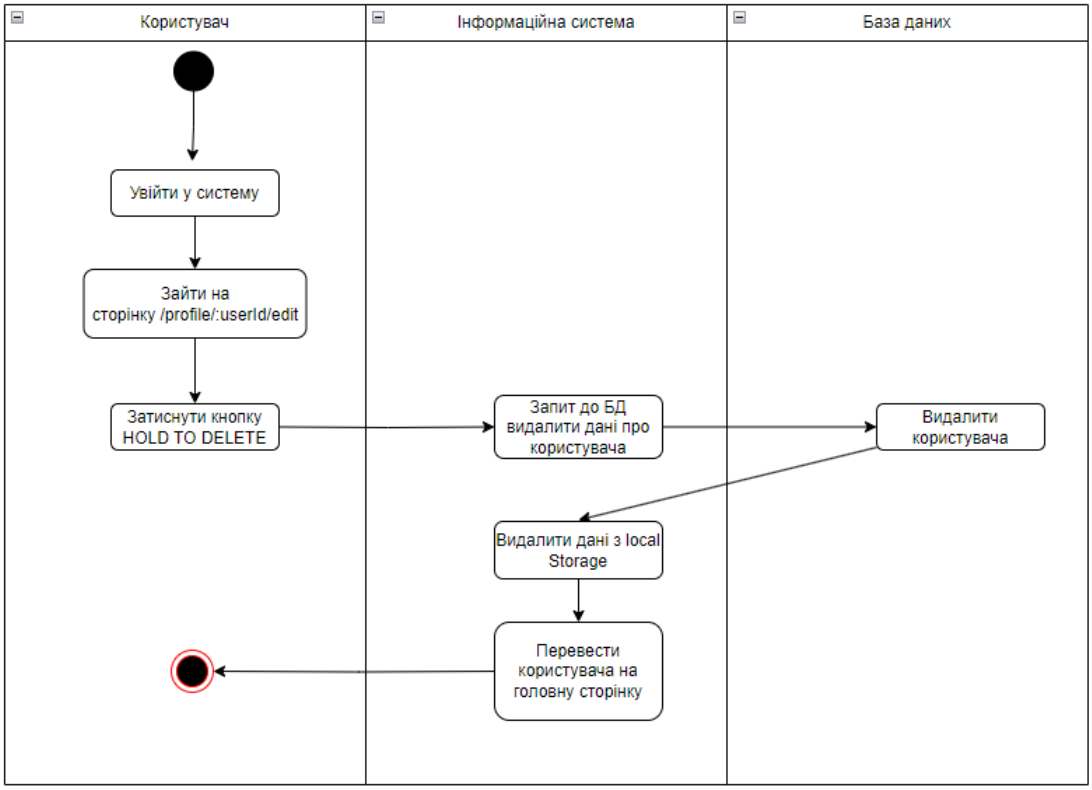

Рисунок 2.6 – Діаграма діяльності видалення користувача із системи

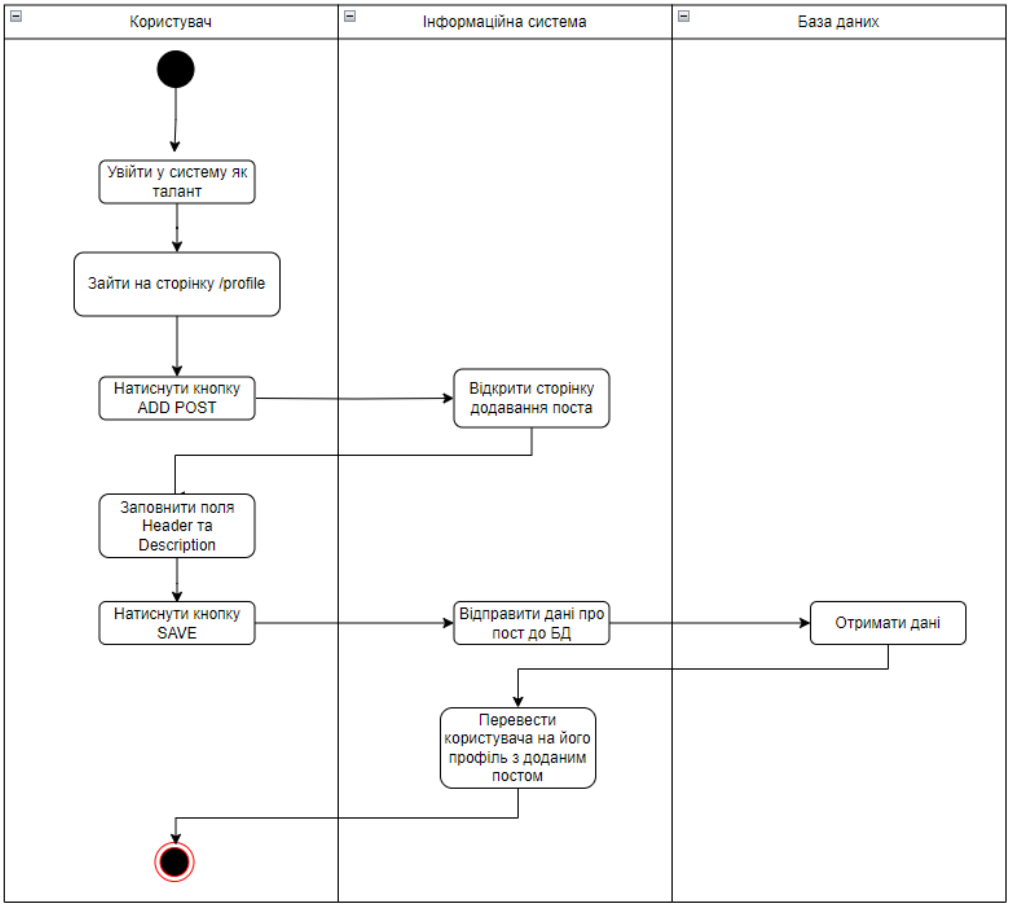

Рисунок 2.7 – Діаграма діяльності додавання нового посту талантом

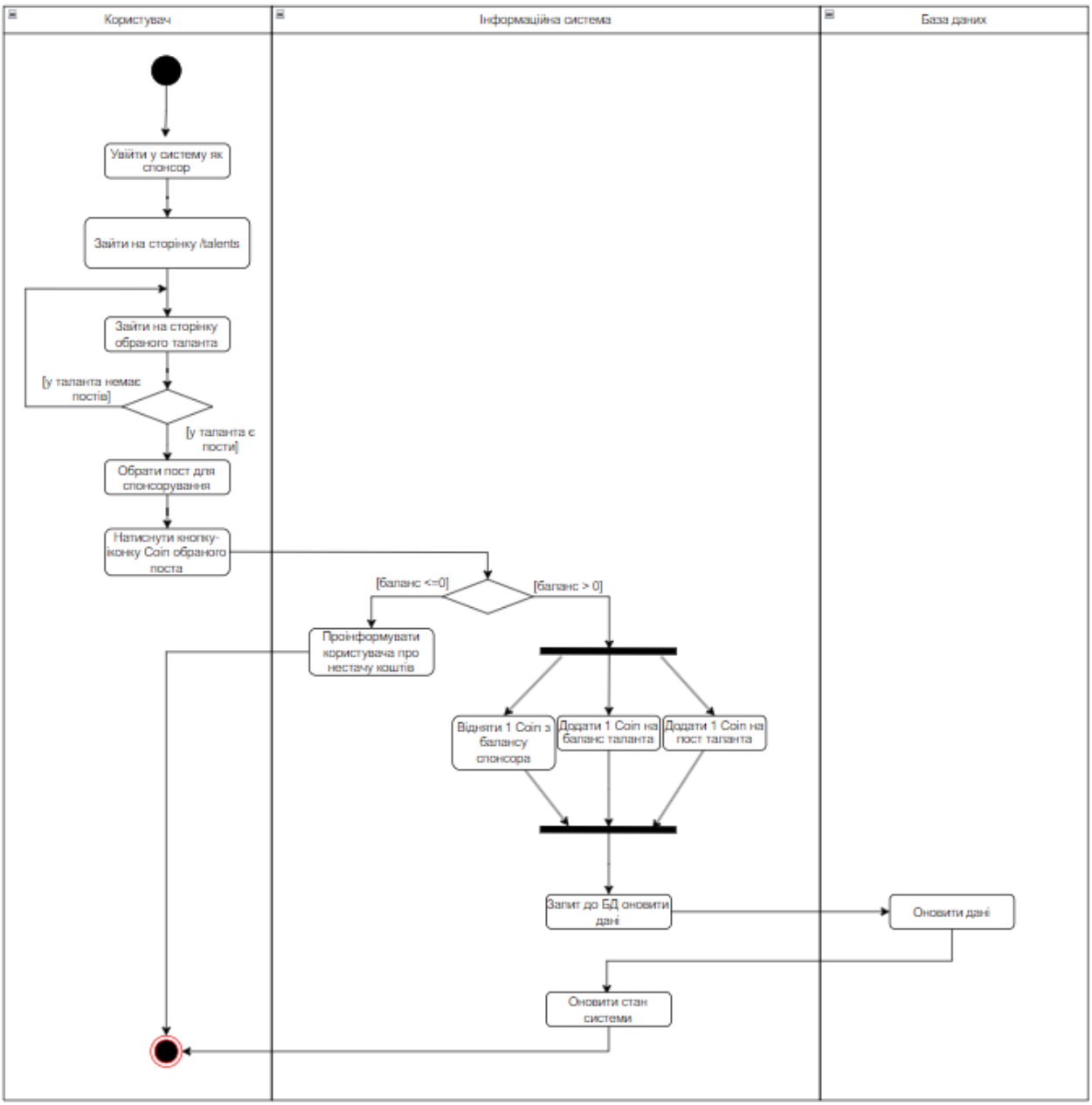

Рисунок 2.8 – Діаграма діяльності спонсорування таланта спонсором

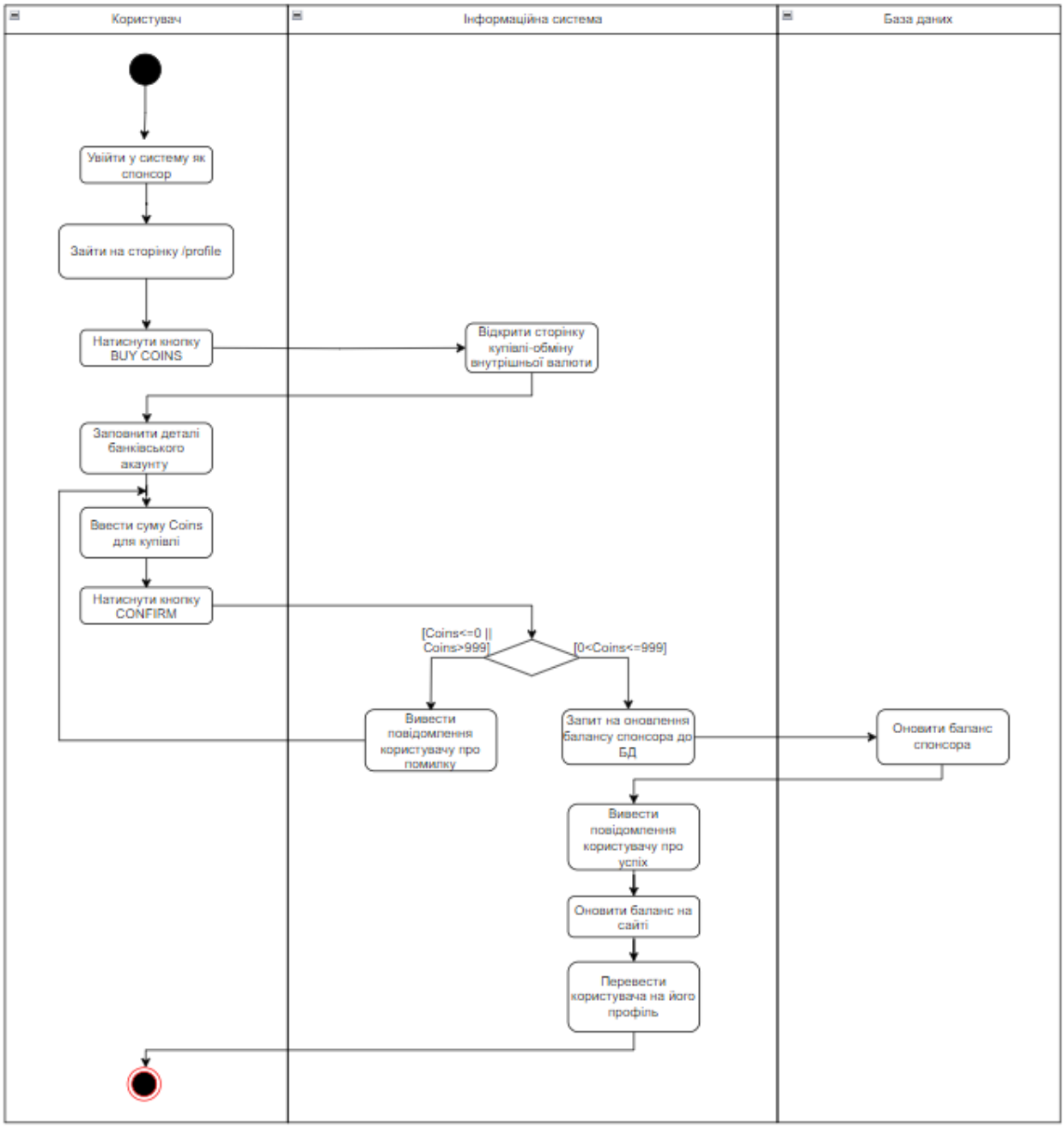

Рисунок 2.9 – Діаграма діяльності купівлі Coins спонсором

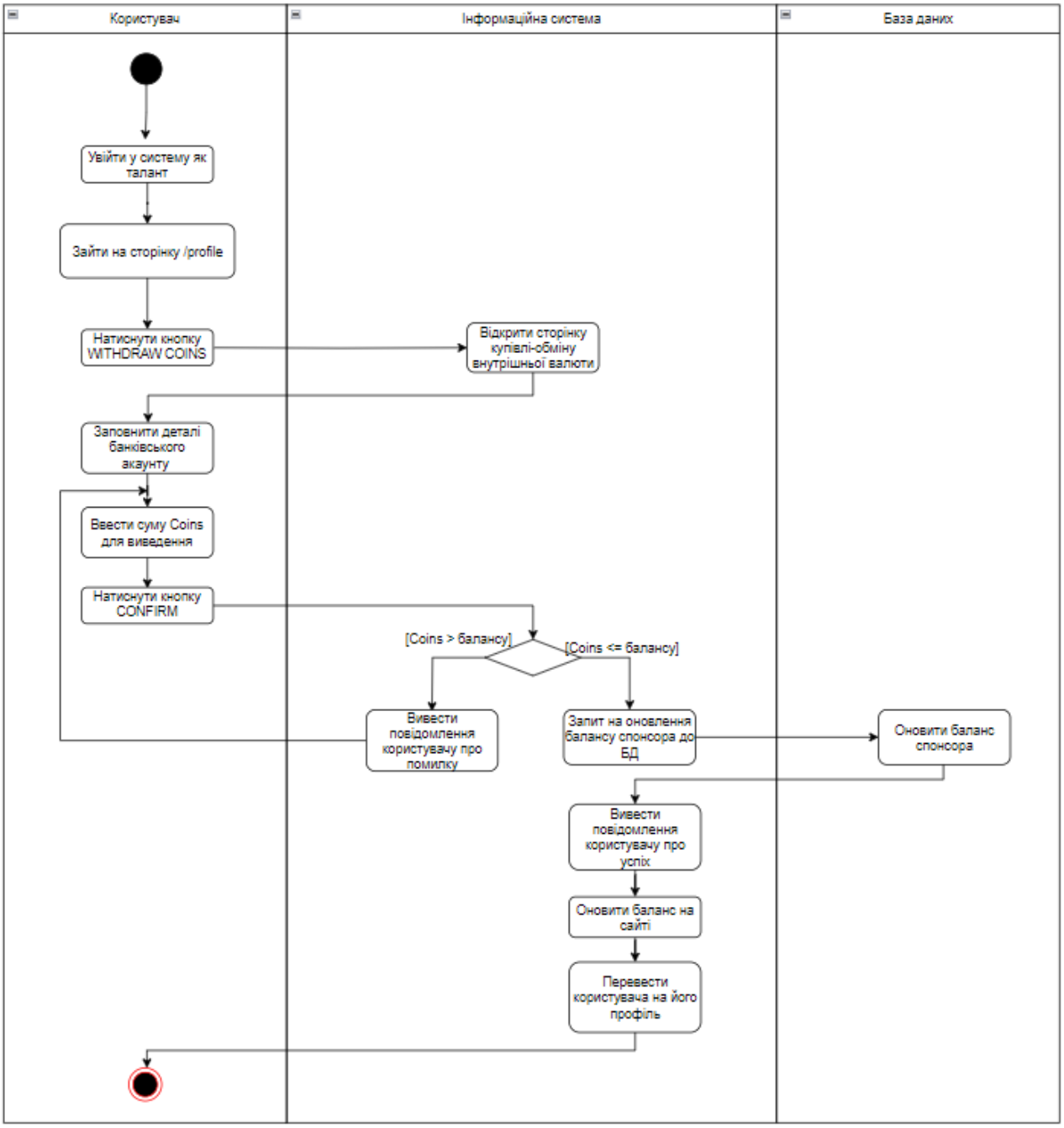

Рисунок 2.10 – Діаграма діяльності виведення Coins талантом на картку

2.5 Створення макетів головних сторінок у Figma

Наступний етап проєктування інформаційної системи – створення макетів головних сторінок вебсайту у Figma [15]. Figma – онлайн-сервіс для створення макетів, який зручно використовувати для розмітки основних сторінок вебсайтів або навіть створення простих анімацій. Він дозволяє реалізувати переходи між

сторінками та елементами інтерфейсу (інтерактивне прототипування), що допомагає краще зрозуміти функціонал і потік роботи і даних у межах продукту.

Щоб розпочати роботу над вебсайтом було створено макети основних сторінок (Рисунки 2.11-2.15).

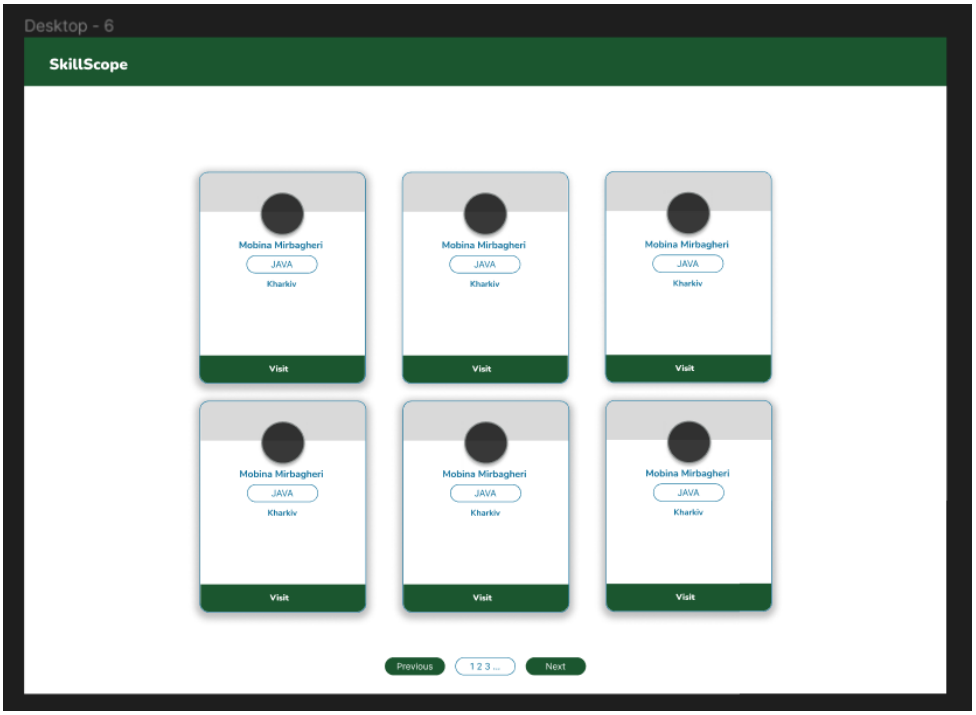

Рисунок 2.11 – Прототип головної сторінки

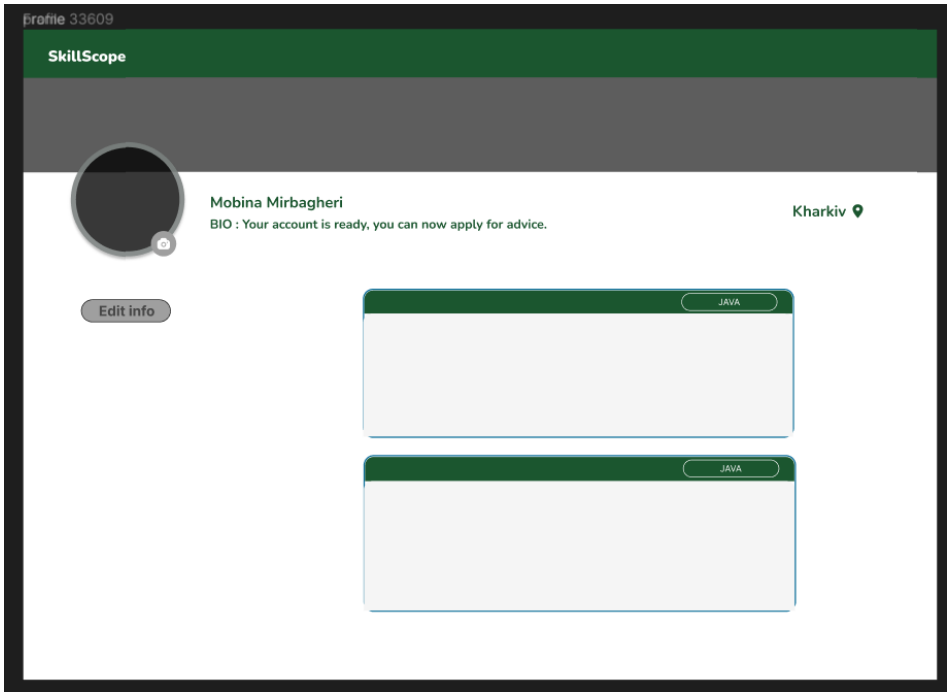

Рисунок 2.12 – Прототип профілю користувача

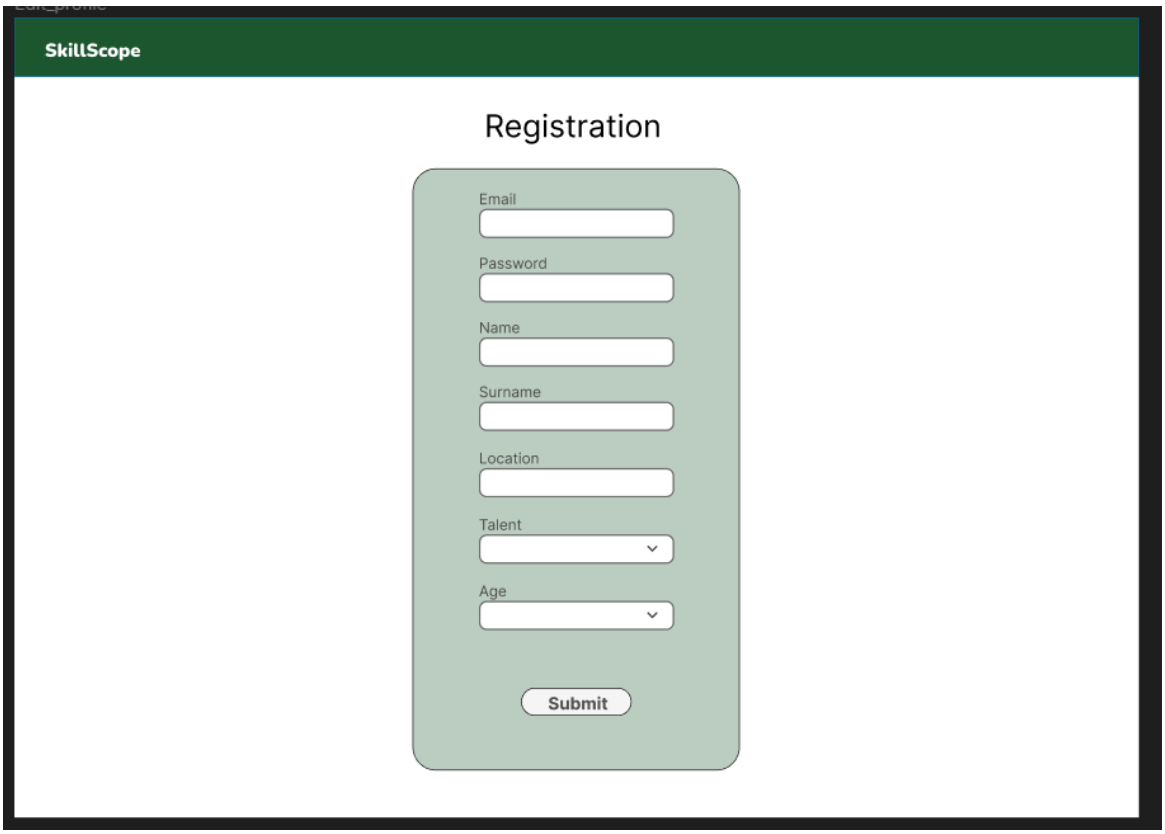

Рисунок 2.13 – Прототип сторінки реєстрації

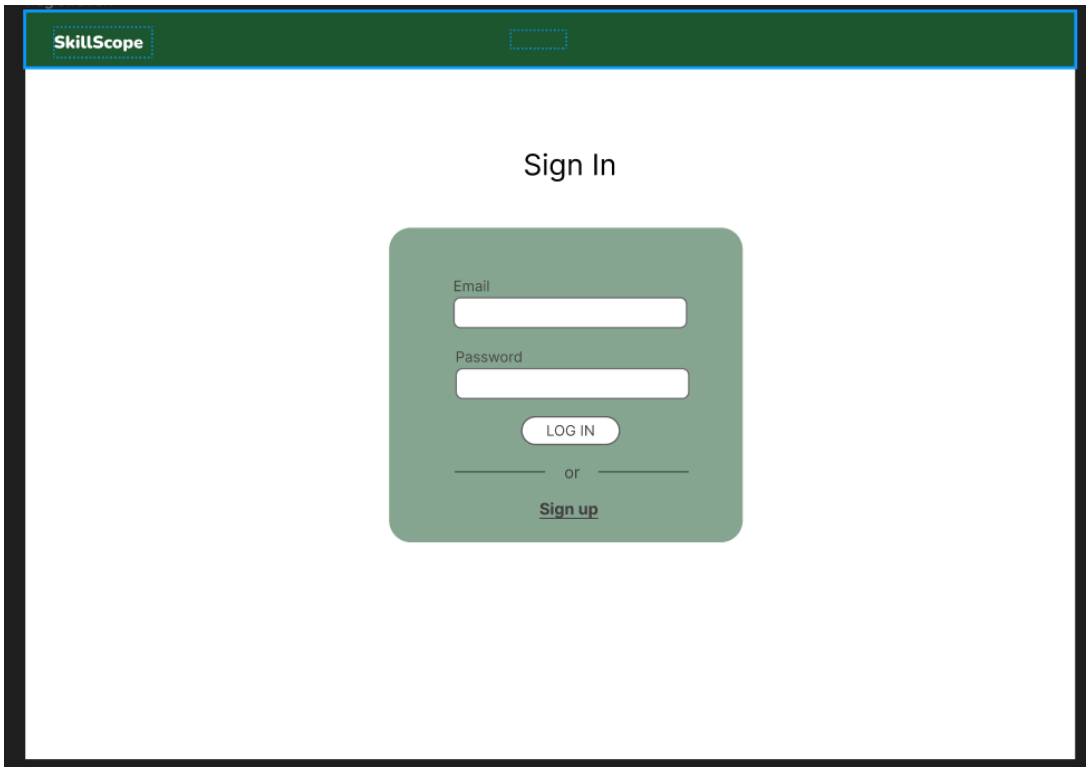

Рисунок 2.14 – Прототип сторінки входу у систему

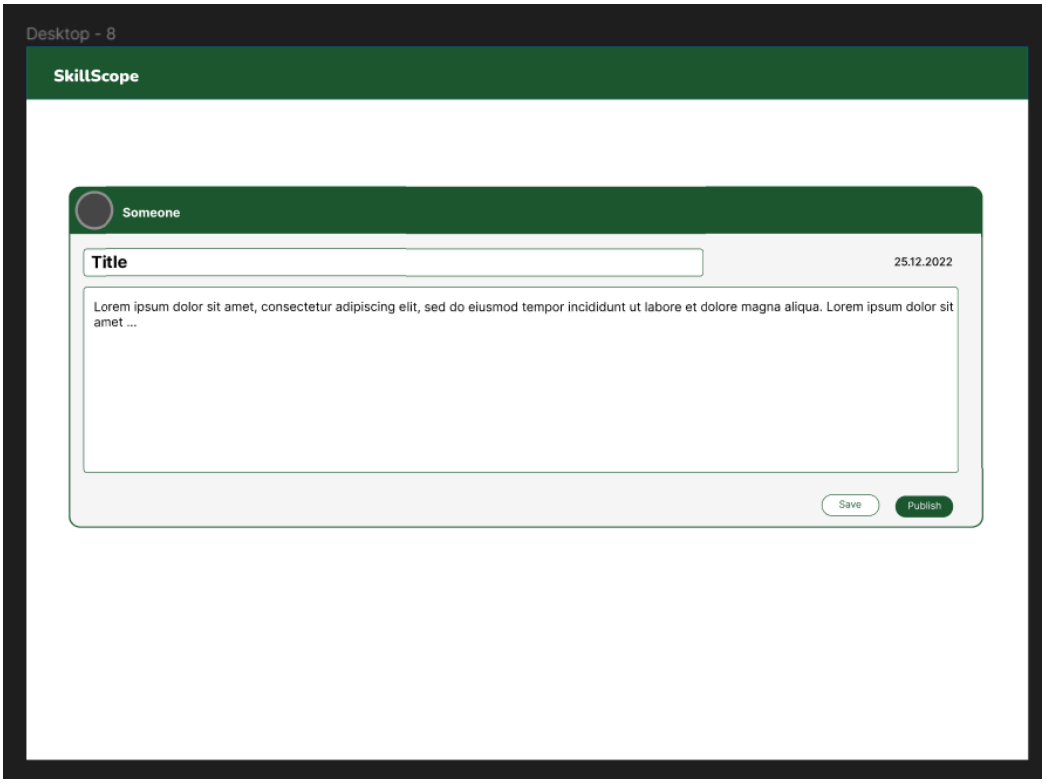

Рисунок 2.15 – Прототип сторінки додавання поста на сторінку таланту

У процесі написання коду ці сторінки були дещо змінені та допрацьовані в залежності від додаткового функціоналу і наповнення сайту, яким знадобилось доопрацювання у ході виконання роботи.

Створення прототипів головних сторінок сайту перед початком роботи над ним, значно полегшило написання коду і зменшило час, витрачений на нього.

2.6 Вибір мови програмування

Існує декілька шляхів втілити потрібну функціональність вебсайту:

- Java – досить популярна для веброзробки, оскільки вона надійно забезпечує безпеку, кросптлатформенність та масштабованість проєктів, а також має великий набір бібліотек. Суттєвим недоліком в контексті розробки цього проєкту може стати те, що ця мова потребує більше часу на розробку.
- Ruby та фреймворк Ruby on Rails, який прискорює розробку вебдодатків, має зрозумілий та елегантний синтаксис [16]. Але не зважаючи на переваги має

менш широке використання порівняно з іншими мовами, може бути складно знайти підтримку або рішення для конкретних проблем.

- Python – це універсальна мова програмування, широко використовується для розробки вебдодатків з використанням фреймворків, таких як Django та Flask [17]. Для розробки нашого проєкту ця мова не мала достатньої кількості бібліотек, які були необхідні для додавання всього функціоналу.

- JavaScript широко використовується для розробки клієнтської та серверної частини вебдодатків, має велику кількість бібліотек і фреймворків для різних потреб [18].

Після детального аналізу чотирьох мов програмування (Java, Ruby, Python, JavaScript), всю інформацію було підсумовано у Таблиці 2.1 за такими критеріями, як: розповсюдженість, швидкодія, рівень підтримки спільноти, достатність рівня знання мови на початок написання проєкту тощо.

Таблиця 2.1– Порівняльний аналіз мов програмування для створення інформаційної системи SkillScope

Характеристика	**Java**	**Ruby**	**Python**	**JavaScript**
Тип	Об'єктно-орієнтована	Об'єктно-орієнтована	Об'єктно-орієнтована	Об'єктно-орієнтована
Використання	Широко використовується для побудови вебдодатків та сервісів	Використовується для швидкого та легкого розвитку вебдодатків	Часто використовуєтьс я для веброзробки, особливо для прототипування	Використовується як основна мова для фронтенду та бекенду вебдодатків
Фреймворки	Spring, Hibernate	Ruby on Rails	Django, Flask	React, Angular, Vue
Синтаксис	Статичний, досить строгий	Динамічний, гнучкий	Динамічний, простий	Динамічний, гнучкий
Швидкодія	Вище середнього	Нижче середнього	Вище середнього	Вище середнього

Спільнота	Велика	Середня	Велика	Дуже велика
Використання у великих проєктах	Так	Так	Так	Так
Знання мови на момент початку проєкту	+	-	-	+

Після аналізу найпопулярніших можливих підходів для створення вебсайтів у Таблиця 2.1 було обрано мову програмування JavaScript через велику кількість бібліотек, доступних фреймворків, що зменшить час на написання коду. Оскільки JavaScript слабо типізована та динамічна мова, то необхідно бути уважним, оскільки типи даних змінних можуть змінюватися під час виконання програми, а також мова дозволяє виконувати різні операції навіть зі змінними різних типів даних без явного перетворення, що може призводити до помилок, особливо на великих та складних проєктах.

Порівнявши різні фреймворки для мови JavaScript у Таблиці 2.2, було прийнято рішення використати React. Ця бібліотека значно полегшує роботу з кодом, шляхом дробіння інтерфейсу на невеликі компоненти, що легко інтегруються та використовуються в залежності від умов і значно зменшує кількість дубльованого коду. Синтаксис дозволяє вбудовувати HTML-подібний код безпосередньо в JavaScript, що робить код React більш зрозумілим та легким для читання.

Таблиця 2.2 – Порівняльний аналіз фреймворків для мови програмування JavaScript

Фреймворк	**React**	**Angular**	**Vue**
Мова	JavaScript	TypeScript	JavaScript
Компоненти	JSX (розширений синтаксис JavaScript)	HTML з вбудованими директивами Angular	Шаблони HTML з вкладеними директивами Vue

Здатність реагувати на зміни	Стан та Props	Two-way binding	Директиви
Переваги	Гнучкість, швидкість рендерингу, масштабованість	Комплексність, документація, інтеграція зі стеком Google	Простота використання, легкість входження в проєкт, швидкість розробки
Вивчення	Легко вивчити, але потрібно зрозуміти React-особливості	Вимагає поглибленого розуміння TypeScript та Angular концепцій	Легко вивчити та почати, але є потреба у дослідженні для складних завдань
Недоліки	Необхідність в додаткових бібліотеках для повноцінного функціоналу	Великий об'єм коду, невелика швидкодія	Обмежений функціонал, менша підтримка від компаній

2.7 Огляд бази даних Cloud Firestore

Для коректного функціонування інформаційної системи та збереження даних усіх користувачів, що зареєстровані у ній, звісно необхідна база даних. Оскільки як бекенд частини у процесі розробки вебсайту не було передбачено, тому вибір припав на такі СУБД, які б змогли працювати напряму з вебзастосунком, без встановлення та налаштування серверу, реалізації власного API та були б сумісні з застосунками, що написані за допомогою React. Було обрано Firestore від Google.

Firestore – це хмарна база даних, що дає доступ до збережених у ній даних у реальному часі. Використання Firestore разом з React дозволяє зберігати, отримувати та оновлювати дані без ручного налаштовування серверу [19].

Перевагами використання Firestore у моєму проєкті стали:

1. Легкість використання та простота налаштування;

2. База даних легко масштабується в залежності від кількості даних та динамічно може розширювати кількість атрибутів в таблиці, якщо у процесі розробки таблиця або документ були розширені;

3. Firestore має простий API, який дозволяє взаємодіяти з базою даних без зайвого складності. Запити та оновлення даних виглядають лаконічно та зрозуміло, такий «API з коробки», значно допомагає зменшити об’єм коду;

4. Бібліотека Firestore для React інтегрується з легкістю в React-додатки;

5. Firestore використовує мову запитів на основі документів (у класичному розумінні записи в таблицях) та колекцій (в класичному розумінні таблиці), яка називається Firestore Query Language. Ця мова запитів орієнтована на роботу з документами та колекціями даних у форматі NoSQL, має свій унікальний синтаксис для виконання операцій, що набагато простіший і ближчий до роботи з об'єктами у порівнянні зі стандартними SQL-запитами, що дозволяє легше розуміти та використовувати його при роботі з Firestore;

6. Використовує ієрархічну модель даних, схожу на формат JSON, що значно полегшує обробку даних з бази даних у моєму вебзастосунку [20].

2.8 Огляд використаної технології React.js

React – це бібліотека JavaScript для створення інтерфейсів користувача простим і ефективним способом. У React можна створювати окремі компоненти інтерфейсу користувача. Поняття про компоненти є центральними для React. Компонент – це клас або функція JavaScript, яка може приймати вхідні дані та виводити елемент React, який описує, як саме повинна виглядати певна частина інтерфейсу користувача. Усі компоненти в поєднанні створюють складний інтерфейс користувача, а React візуалізує цей інтерфейс.

DOM (Document Object Model) – це структуроване відображення елементів HTML, що присутні на вебсторінці. Він визначає деревовидну структуру документів і надає програмам, таким як JavaScript, можливість доступу та зміни цих документів. Замість того, щоб прямо маніпулювати DOM, React

використовує віртуальний DOM і реалізує алгоритми для ефективного оновлення реального DOM: React рендерить компоненти на віртуальному DOM, а потім порівнює його з реальним DOM, визначаючи мінімальні зміни, які потрібно виконати для оновлення [21].

Основні концепти React.js:

1. Використання компонентів: React розглядає UI як набір повторюваних ізольованих компонентів, що використовуються для організації та структурування коду;

2. Використання JSX (JavaScript XML) для React-компонентів замість звичайної розмітки HTML;

3. Декомпозиція компонентів: спочатку складний інтерфейс користувача розбивається на кілька менших компонентів. Після завершення роботи над кожним із них, вони знову об’єднуються в цільний проєкт, що зменшує складність дебагінгу та збільшує читабельність коду (Рисунок 2.16);

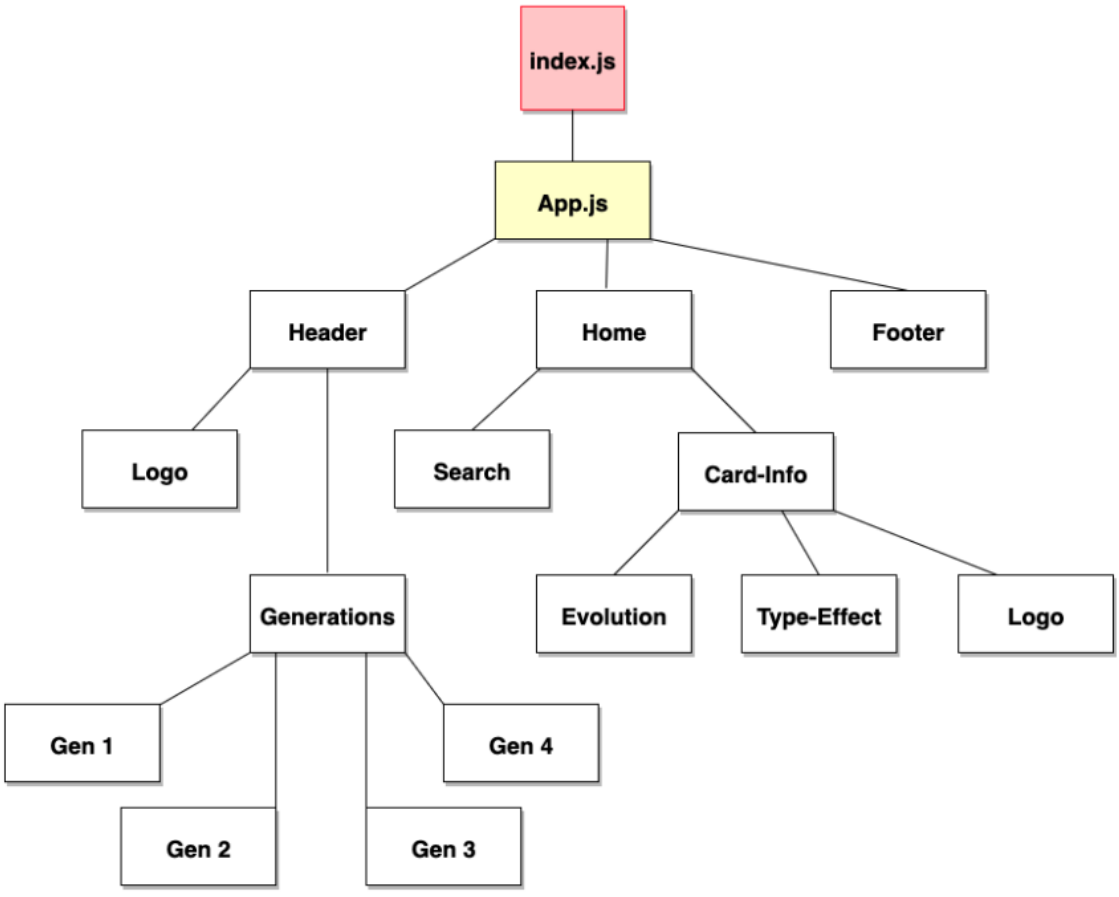

Рисунок 2.16 – Приклад декомпозиції в React

4. Перевикористання компонентів: розбиття інтерфейсу користувача на окремі компоненти, які можна використовувати в кількох частинах програми для створення кількох варіантів інтерфейсу користувача, що зменшує кількість дубльованого коду;

5. Компонент може мати стан: метод setState() може змінювати стан компонента. Коли стан змінюється, компонент перемальовується [22];

6. Компонент може мати стан (props): props схоже на state, але props передається в компонент та дають можливість налаштувати компоненти ззовні, в той час як state знаходиться всередині компонента. На відміну від стану, передані властивості від батьківського до дочірнього компоненту не можуть бути змінені. Props є одним з основних механізмів, які роблять компоненти в React перевикористовуваними та конфігурованими;

7. Умовний рендеринг: показування чи приховування деяких даних чи частин розмітки користувачеві за певних умов. Якщо умова виконується, користувач може бачити дані або ні;

8. React Hooks: функції, за допомогою яких ви можете «підчепитися» до стану та методів життєвого циклу React із функціональних компонентів, наприклад хук useState(), який був згаданий вище, здатний змінювати стан компонента;.

9. Обробники подій: авторські функції, які запускатимуться у відповідь на такі взаємодії користувача з інтерфейсом, як клацання, наведення курсора, фокусування введених даних у формі тощо;

10. Наявність «життєвого циклу»: у кожного компоненту є «життєвий цикл», яким можна маніпулювати під час трьох фаз: Mounting(монтування), Updating(оновлення) та Unmounting(демонтування).

2.9 Вибір середовища розробки для обраних технологій

Для написання проєкту SkillScope з використанням бібліотеки React.js було обрано такий текстовий редактор як Visual Studio Code (VS Code).

Visual Studio Code – редактор вихідного коду від компанії Microsoft для Windows, в якому є підтримка JavaScript, HTML/CSS та формату JSON, що знадобиться мені в процесі розробки інформаційної системи [23].

При виборі середовища Visual Studio Code як основного середовища розробки мого проєкту головними стали його переваги, такі як:

- Легкість використання, приємний інтерфейс та вбудовані функції і «гарячі клавіші»;
- Магазин плагінів, які полегшують роботу з бібліотекою React.js, наприклад Prettier та ESLint, які допоможуть форматувати вигляд коду та знаходити помилки;
- Інтеграція з Git, що було критично важливим для мене, оскільки проєкт великий за розміром і передбачається його зберігання у віддаленому репозиторії GitHub:
- Технологія Life Server, що дозволяє оновлювати сторінку браузера автоматично з кожною зміною в коді.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Проєктування бази даних

Важливою частиною проєктування інформаційної системи SkillScope є проєктування бази даних, де будуть зберігатися дані користувачів [24].

База даних інформаційної системи «Skillsope» досить проста, але містить в собі усі таблиці, що потрібні для роботи платформи. Вона складається із двох таблиць: users, posts. На Рисунок 3.1- 3.2 зображена структура бази даних сайту, усі таблиці (каталоги) і поля детально описані у Таблиці 3.1.

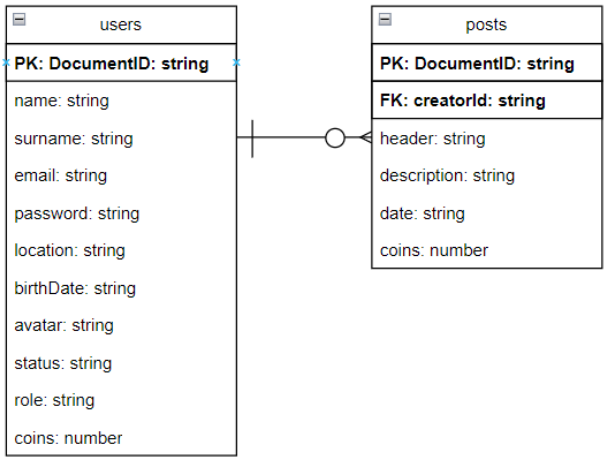

Рисунок 3.1 – ERD-діаграма бази даних вебсайту

Таблиця 3.1 – Опис полів бази даних

Таблиця	Поле	Зміст	Тип	Ключі / обмеження
users	name	Ім’я користувача	String	Primary Key
	surname	Прізвище користувача	String	not null
	email	Поштова скринька користувача	String	Unique, not null
	password	Пароль користувача	String	not null
	location	Країна проживання користувача	String	not null
	bitrhDate	Дата народження користувача	String	not null
	avatar	Посилання на зображення профілю користувача	String	

	status	«Про себе» користувача	String	
	role	Роль користувача (талант/спонсор)	String	not null
	coins	Баланс користувача	Number	Default 0
posts	creatorId	Ідентифікатор користувача, що створив пост	String	Foreign Key
	header	Заголовок поста	String	not null
	description	Опис поста, основна частина	String	not null
	data	Дата створення поста	String	not null
	coins	Кількість задоначеної внутрішньої валюти на пост	Number	Default 0

Підключення до бази даних та робота з нею відбуватиметься за допомогою бібліотеки Firebase, що дозволить не відходити від синтаксису мови JavaScript та забезпечувати ефективну роботу між вебсайтом та базою даних.

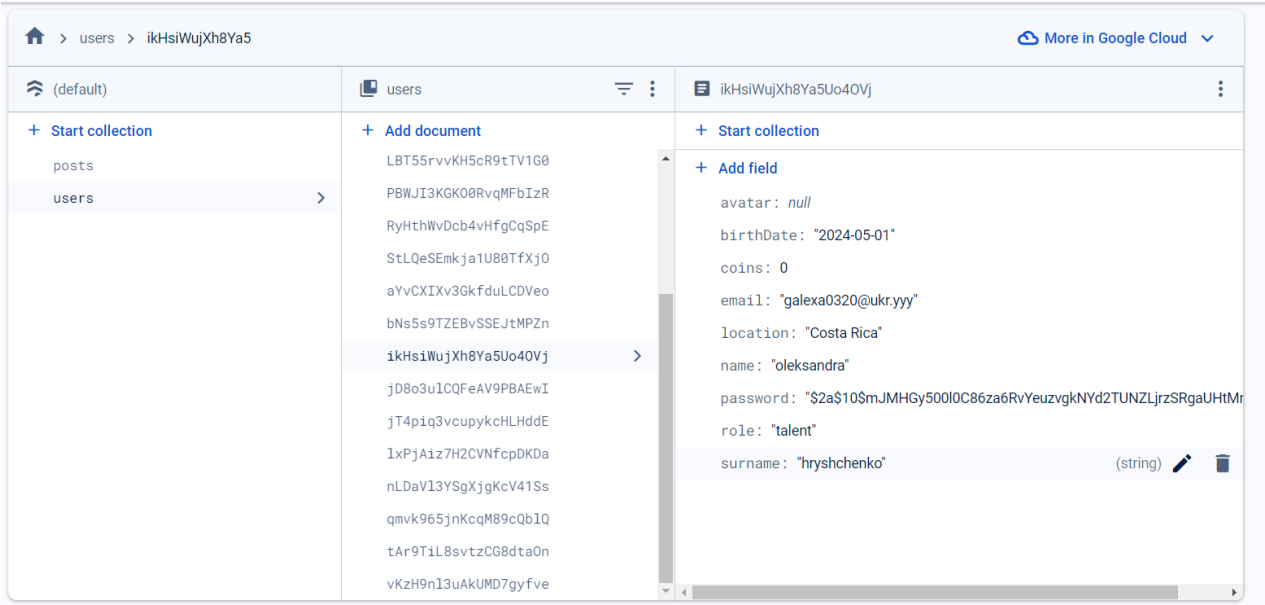

Рисунок 3.2 – Структура даних в хмарному сховищі Firestore

База даних Firestore складається з:

- Колекцій, які можна уявити як таблиці в традиційних реляційних базах даних. Наприклад колекції «users» і «posts». Кожна колекція містить документи;

- Документів, які можна уявити як записи в таблиці: один документ – один набір даних про користувача. Кожен документ у колекції містить конкретний набір полів.
- Полів, які містять фактичні дані. Наприклад, у колекції "users" документ може містити поля, такі як "name", "email", "age" та інші.
- Ідентифікаторів документів, які можуть бути згенеровані Firestore. Кожен документ має унікальний ідентифікатор.

3.2 Програмна реалізація

3.2.1 Основні відомості

1. Фронтенд (Frontend):

- Фронтенд сайту реалізований з використанням бібліотеки React.js, яка є популярною бібліотекою для створення інтерактивних інтерфейсів користувача.
- У компонентах React використовуються хуки, такі як useState і useEffect, для управління станом компонентів і виконання певних дій при їх монтуванні та оновленні.
- Для стилізації використовується CSS модулі з використанням класів, які імпортуються в компоненти React.
- Було застосовано готові до використання компоненти з бібліотеки Material UI, для зменшення кількості коду та допомогла досягти більш адаптивного макету.

2. Бекенд (Backend):

- Бекенд сайту було реалізовано за допомогою сервісу Firebase, який надає хмарні сервіси, такі як зберігання даних у базі даних Firestore.
- Взаємодія з базою даних Firestore відбувається через бібліотеку firebase/firestore, яка дозволяє виконувати різноманітні операції, такі як додавання, оновлення та видалення документів.

3. Маршрутизація (Routing):

- Для маршрутизації між сторінками сайту використовується бібліотека react-router-dom, яка дозволяє визначати маршрути та відображати відповідні компоненти для кожного маршруту.

4. Взаємодія з користувачем:

- Сайт має форми для введення даних користувачем, які валідуються за допомогою правил валідації, що наведені у файлі validationRules.js.
- Користувач може взаємодіяти зі створеними компонентами за допомогою кнопок, що дозволяє виконувати певні дії або надсилати дані до бази даних.

5. Допоміжні інструменти:

- Figma
- GitHub

Для початку розробки необхідно виконати команди:

npx create-react-app skillscope

cd skillscope,

що дозволять створити новий застосунок на базі React. Після виконання цієї команди буде створена нова директорія skillscope з усіма необхідними файлами та налаштуваннями для роботи з React. Після цього встановлюємо необхідні бібліотеки за допомогою команди npm install:

- @mui/material: Надає набір готових до використання компонентів і стилів, які відповідають дизайну Material Design від Google;
- @emotion/react і @emotion/styled: Бібліотека для стилізації React-компонентів;
- react-router-dom: забезпечує навігацію в додатках React. Вона дозволяє визначати маршрути та управляти переходами між сторінками додатка;
- firebase: надає набір інструментів для реалізації функціональності роботи з базою даних;
- react-hook-form: дозволяє керувати формами, пропонує простий спосіб зберігати стан форми та валідувати введені дані.

- @mui/icons-material: Містить набір іконок, які можна застосувати у компонентах Material-UI;
- styled-components: використовується для стилізації React-компонентів та дозволяє вбудовувати CSS стилі безпосередньо в код React. Використовується у компоненті ProgressButton.jsx;
- react-credit-cards-2: використовується для створення красивих та реалістичних компонентів кредитних карток;
- Bcryptjs: бібліотека Node.js для хешування та перевірки паролів. Вона використовує bcrypt, алгоритм хешування паролів, який вважається безпечним та надійним.

У кореневій директорії src створюємо папки components, db, shared, components і починаємо роботу на проєктом.

Структура інформаційної системи зображена на Рисунок 3.3 і виглядає так:

—frontend – містить папку components, де розміщені усі компонети, що необхідні для роботи вебсайту. Кожен компонент знаходиться в індивідуальній папці, що полегшує навігацію по структурі проєкту. Кожна папка містить файл з розширенням .jsx, що і містить основну розмітку і функціонал компоненту; файл з розширення .css містить стилі, що імпортуються у головний файл; файл index.js містить експорт компонента, що полегшує імпортувати його в інші компоненти, якщо таке передбачає логіка. Якщо у компонента є логічно пов’язані з ним підкомпоненти – вони виносяться разом в окрему папку components, яка зберігається всередині папки головного компоненту;

—shared – містить файли, які можуть використовуватись усіма компонентами системи, наприклад таким є файл validation.js який використовується мінімум чотирма компонентами: SignIn.jsx, SignUp.jsx, AddPostProfile.jsx, EditProfile.jsx. Також містить файли з функціями для хешування паролю hashPassword.js та verifyPassword.js для порівняння простого текстового пароля, введеного користувачем при вході у систему із хешованим паролем, що зберігається в базі даних;

—public – містить статичні зображення проєкту, такі як фавікони, банер профілю та файл з метаданими;

—db – містить увесь функціонал для роботи з базою даних. Містить файл з налаштуванням бази даних Firestore та папку service, де зберігаються файли з функціями для роботи з базою даних.

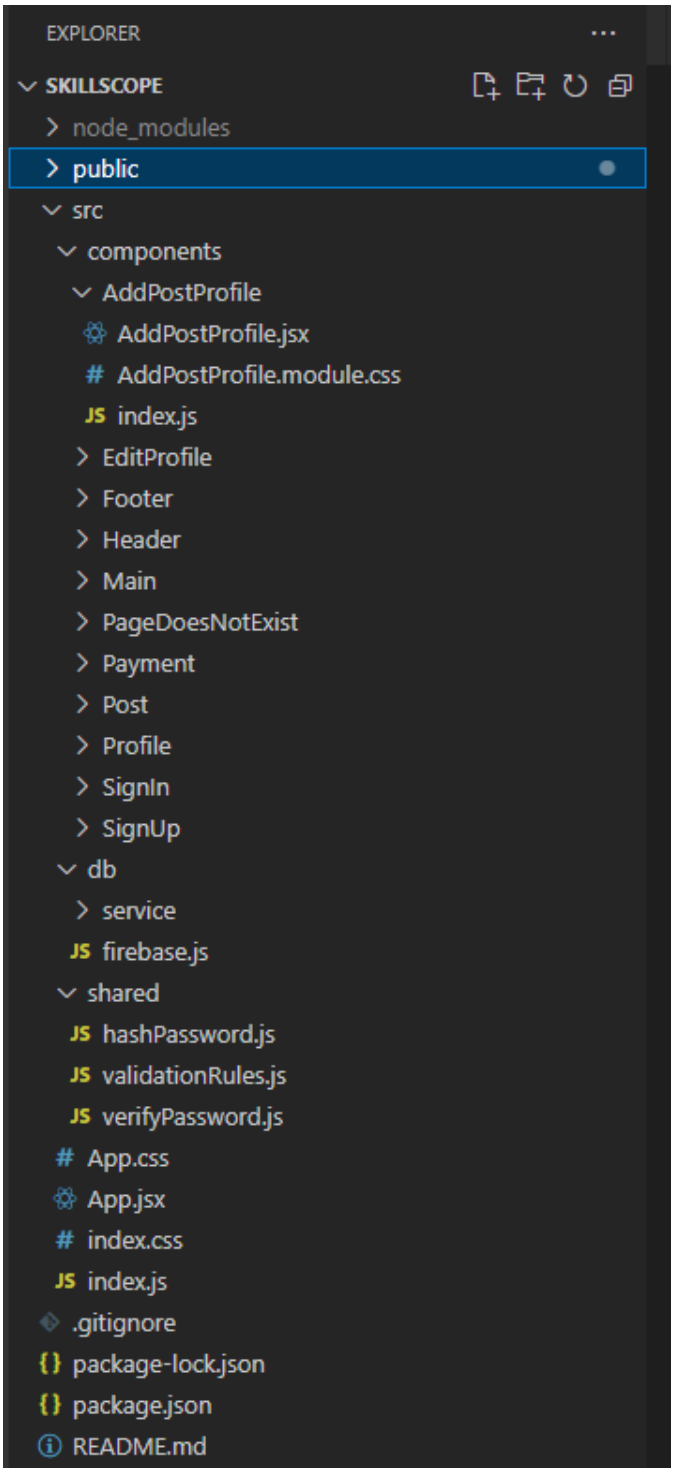

Рисунок 3.3 – Структура проєкту SkillScope

Кожен React компонент будувався з можливістю його повторного використання за різних умов. Один компонент може виглядати по різному в залежності від умови (умовний рендерінг), від того, на якій сторінці він знаходиться, або яка інформація передається у нього.

Один із способів використати компонент повторно – передати в нього props від батьківського компоненту. Ці значення дозволяють створювати динамічні компоненти, які не залежать від жорстко закодованих статичних даних. Ми можемо це побачити на прикладі компоненту картки профілю таланта TalentCard.jsx:

```
const TalentCard = ({ talentData }) => {
  return (
    <div className={styles.card}>
      <div className={styles.background}>
        <Avatar
          src={talentData.avatar ? talentData.avatar : "/broken-image.jpg"}
          sx={{ width: 60, height: 60, mt: 7 }}
          style={{
            border: "2px solid rgba(215, 227, 224, 0.5)",
          }}
        ></Avatar>
      </div>
      <div className={styles.content}>
        <div
          className={styles.name}
        >{`${talentData.name} ${talentData.surname}`}</div>
        <div className={styles.location}>
          <Place color="success" sx={{ fontSize: 16 }} />
{talentData.location}{" "}
        </div>
        <div className={styles.proof}>{talentData.status}</div>
      </div>)
};
```

Параметр props, що передається у компонент, інкапсулює властивості об'єкта. Зокрема, властивості avatar, name, surname та status, що були передані від батьківського компонента Main.jsx у об’єкті talentData. При рендерингу ми можемо створити набір компонентів карточок таланту, але передати в кожну з них різні дані і таким чином отримати набір однотипної розмітки html, наповненої різними даними. Залежно від того яку інформацію про певного

користувача буде передано у компонент TalentCard.jsx через props, відобразиться певна карточка профілю.

Умовний рендеринг — відображення компонента або його частин тільки якщо вони задовільняють певній умові. Розглянемо приклад Header.jsx, коли різні кнопки відображаються в залежності чи користувач увійшов в систему чи ні:

```
{isLogged ? (
      <div className={styles.button_wrap}>
        <NavLink className={styles.button_in} to={`profile/${id}`}>
          PROFILE
        </NavLink>
        <NavLink
          className={styles.button_in}
          onClick={handleSignOut}
          to="/talents"
        >
          SIGN OUT
        </NavLink>
      </div>
) : (
      <div className={styles.button_wrap}>
        <NavLink className={styles.button_in} to="/signin">
          SIGN IN
        </NavLink>
        <NavLink className={styles.button_in} to="/signup">
          SIGN UP
        </NavLink>
      </div>
    )}
```

Якщо користувач авторизувався в системі, у хедері він зможе побачити кнопки переходу у профіль та кнопку виходу із системи, а якщо ні, то побачить кнопки авторизації та реєстрації.

3.2.1 Маршрутизація

У React існує своя система маршрутизації вебсторінок на вебсайті, яка побудована на принципі співставлення з запитами до застосунку з певними компонентами. За допомогою маршрутизації ми можемо створювати

односторінкові вебпрограми з навігацією без необхідності оновлювати сторінку під час навігації. Щойно користувач натискає посилання, URL-адреса змінюється, і на сторінці відображається новий компонент [25]. Для цього в поєднанні з React.js зазвичай застосовують бібліотеку react-router-dom.

Вихідний код App.jsx разом з маршрутизацією можна переглянути у Додатку А.

У App.jsx було імпортовано BrowserRouter, Route і Routes з react-router-dom. BrowserRouter — це тип компонента Router, який використовується як контейнер для всієї вашої додаткової маршрутизації. Компонент Router створює маршрути. Ми повинні додати його в кореневий каталог нашої програми, щоб увімкнути маршрутизацію, тому ми загортаємо наш кореневий компонент App.jsx у маршрутизатор.

Routes — це компонент відповідності маршруту, кожен маршрут може мати свою власну URL-адресу та компонент, який буде відображатися, коли URL відповідає цьому маршруту. Маршрути по інформаційній систему SkillScope зображені на Рисунку 3.4.

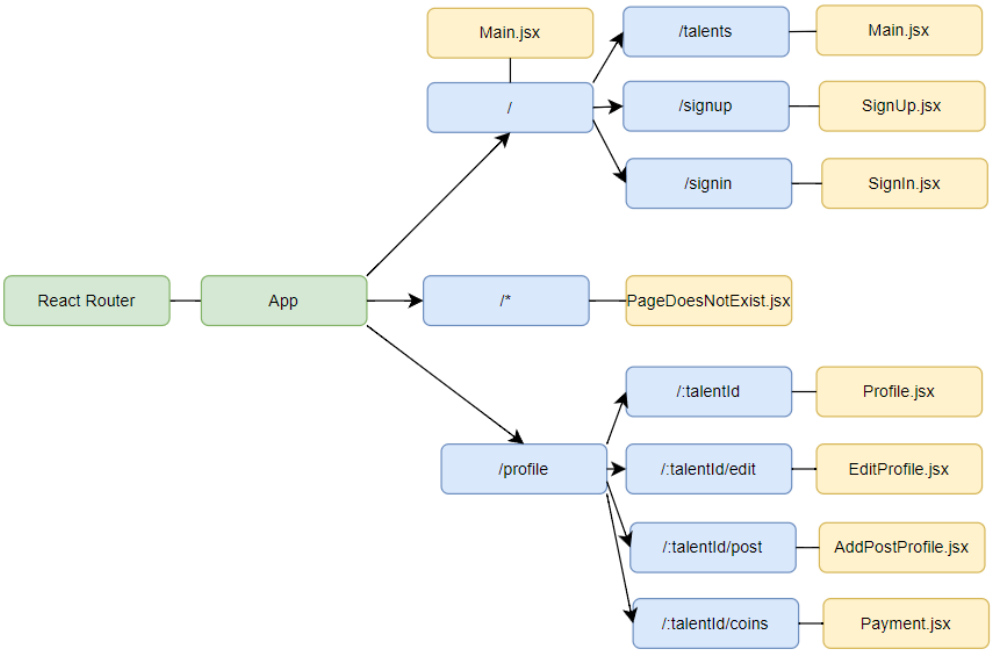

Рисунок 3.4 – Маршрутизація по вебсайту SkillScope

3.2.3 Засоби безпеки

Для того, щоб інформаційна система SkillScope залишалася стабільною та цілісною, а дані користувача залишалися недоторканими і не зазнавали несанкціонованого доступу, було вжито додаткових заходів:

1) Аутентифікація та авторизація: необхідність авторизації до доступу до деяких сторінок, зокрема профілів талантів, та аутентифікація на основі ролей задля доступу до певного функціоналу, доступному тільки певній ролі, наприклад доступ до спонсорування тільки для користувачів з роллю «спонсор»:

```
<IconButton disabled={role !== "sponsor"} onClick={handleAddCoin}>
          <img
            className={styles.coin}
            src={require("./img/coin.png")}
            alt="coin"
 />;
```

2) Валідація форм за допомогою вбудованих засобів бібліотеки react-hook-form та кастомної валідації з файлу validationRules.js, наприклад, валідація пошти:

```
email: {
      required: "Email is required",
      pattern: {
         value: /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
         message: "Not valid email. Must be examp@gmail.com",
      },
      minLength: {
         value: 5,
         message: "Email must have at least 5 characters",
      },
      maxLength: {
         value: 254,
         message: "Email must have maximum 254 characters",
      },
   };
```

3) Використання NoSQL бази даних Firestore, що автоматично захищає від SQL-ін’єкцій;

4) Хешування паролю перед його відправкою до бази даних за допомогою функції hashPassword.js. Замість дехешування для перевірки відповідності

паролю, що ввів користувач при вході у систему, з тим, що в базі даних, використовується функція порівняння verifyPassword.js з вбудованою bcrypt.compare(password, hashedPassword), оскільки процес хешування односторонній, що забезпечує високу безпеку;

5) Для запобігання несанкціонованого доступу до сторінок та обходу авторизації шляхом вставки підготовлених посилань у адресний рядок браузера, використовується перевірка на наявність ідентифікатора користувача idUser в localStorage, яка перенаправляє користувача на сторінку входу, якщо користувач не залогінений:

```
useEffect(() => {
   if (!idUser) {
     navigate("/signin");
   }
  }, [idUser, navigate]);
```

6) Обробка помилок та логування, щоб дізнатись про можливі вразливості інформаційної системи SkillScope. Наприклад, так оброблюються та логуються помилки при додаванні нового поста у базу даних:

```
export const addPost = async (data, formattedDate, id) => {
  try {
    await addDoc(collection(db, "posts"), {
      creatorId: id,
      header: data.header,
      description: data.description,
      date: formattedDate,
      coins: 0,
    });
    console.log("Data added to Firestore successfully!");
  } catch (error) {
    console.error("Error adding data to Firestore: ", error);
  }
};
```

3.3 Тестування

3.3.1 Тестовий сценарій «Реєстрація»

Після того як користувач зайшов на головну сторінку вебсайту SkillScope (Рисунок 3.5) за посиланням «/talents» для того, щоб зареєструватись у системі, користувач має натиснути кнопку «SIGN UP» у хедері.

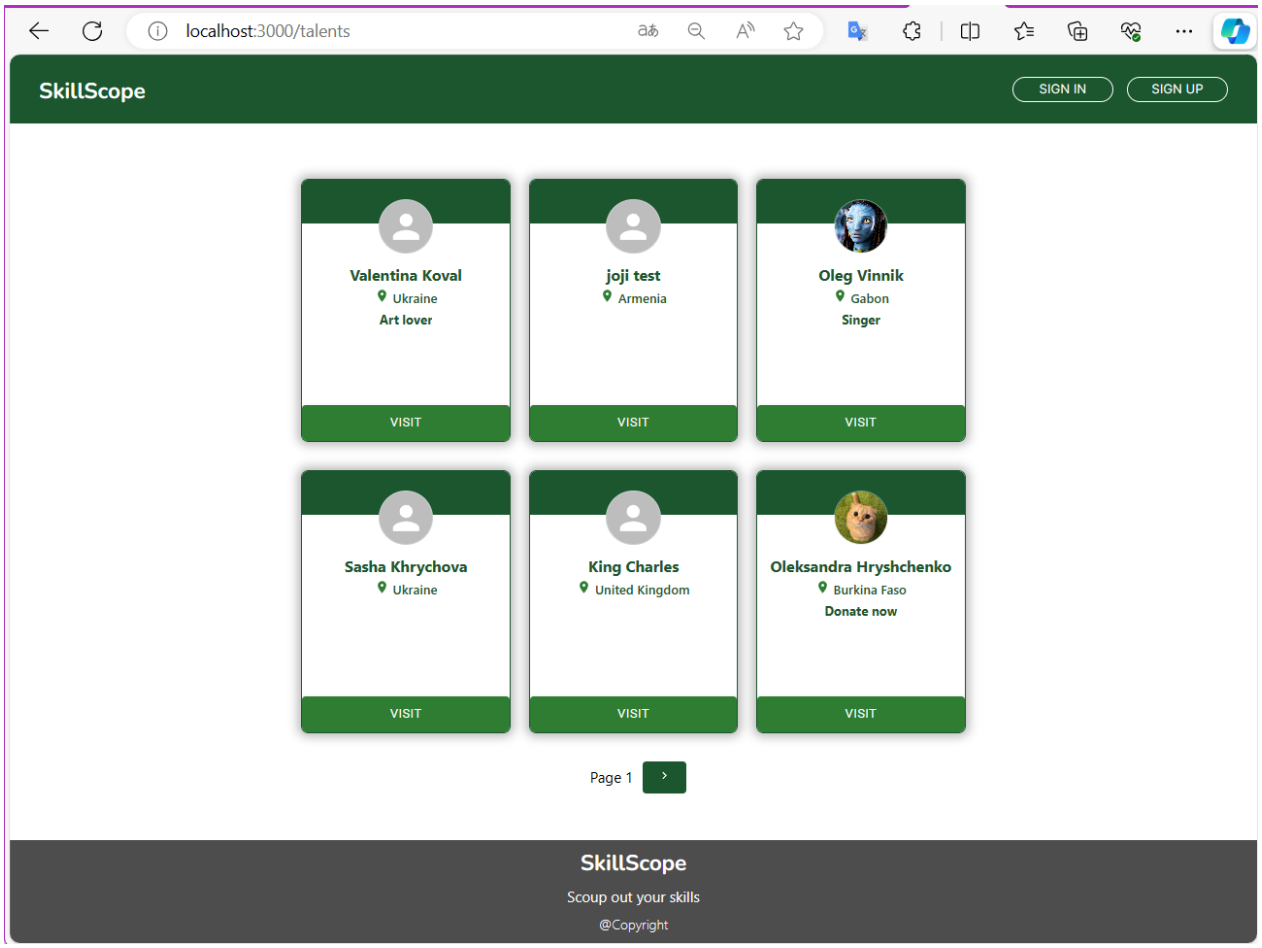

Рисунок 3.5 – Головна сторінка

Після цього відкриється сторінка реєстрації за посиланням «/signup» (Рисунок 3.6). Поля потрібно заповнити, слідуючи підказкам валідації полів, вибрати роль користувача («спонсор» чи «талант») та натиснути кнопку «SIGN UP» у формі.

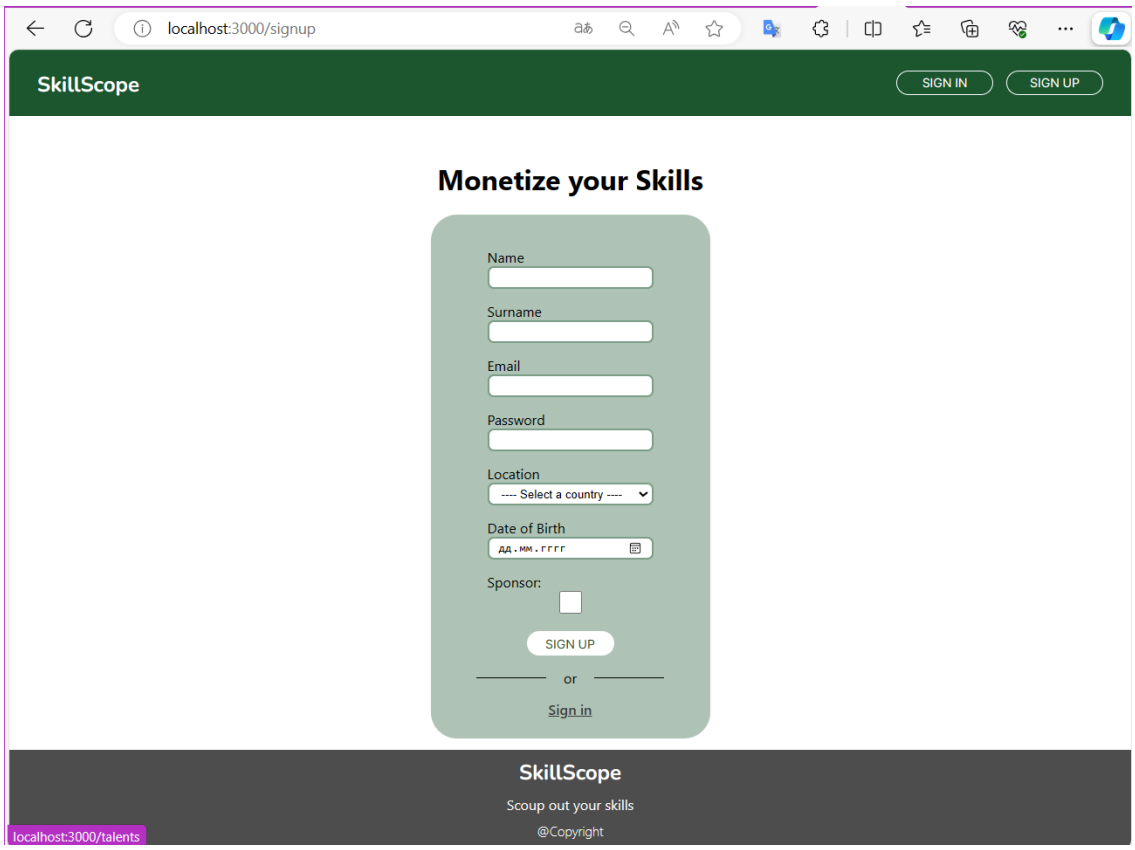

Рисунок 3.6 – Сторінка реєстрації нового користувача

Після цього, користувача автоматично перенаправить на його профіль з заповненими ним раніше даними за посиланням «/profile/:userId». В залежності від обраної користувачем ролі, він отримає різне наповнення профілю функціоналом: сторінка спонсора зображена на Рисунок 3.7, а таланта – на Рисунок .

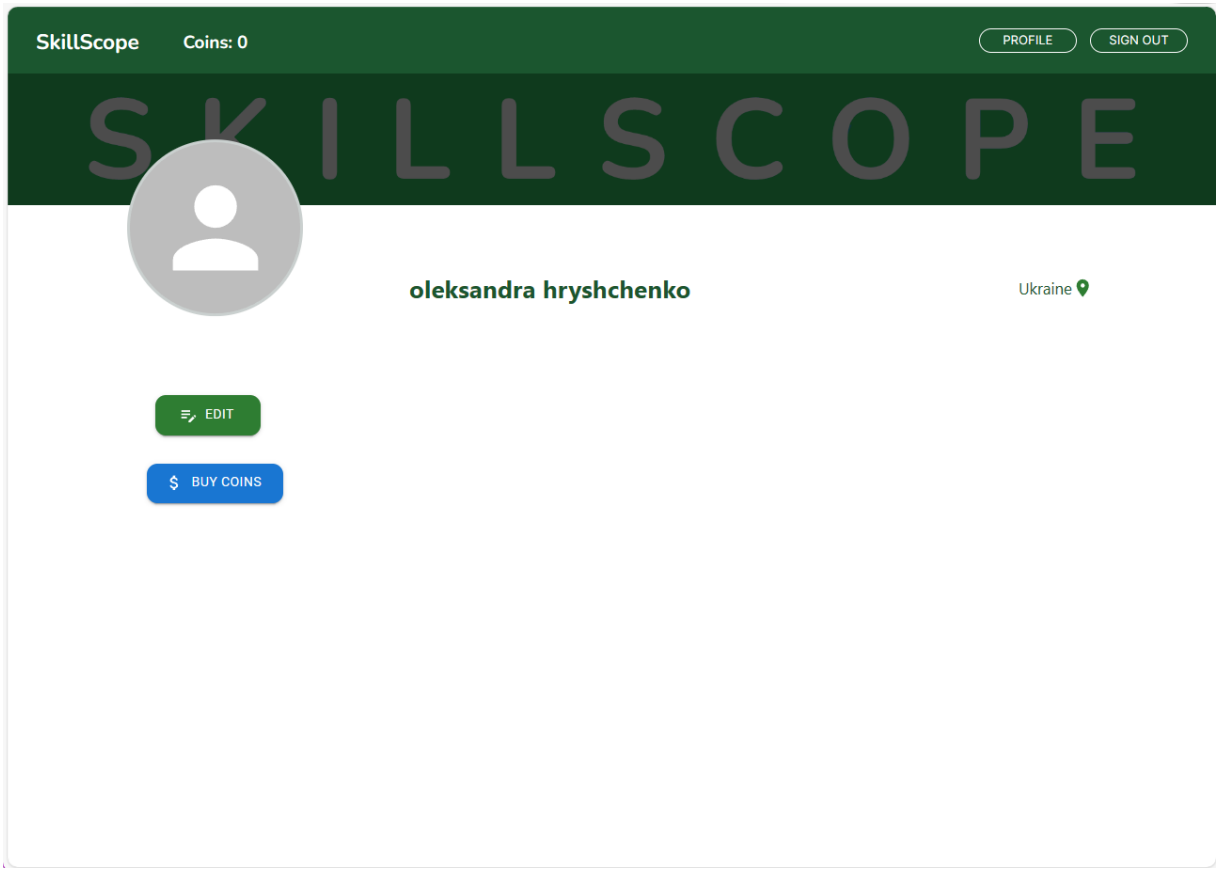

Рисунок 3.7 – Профіль спонсора

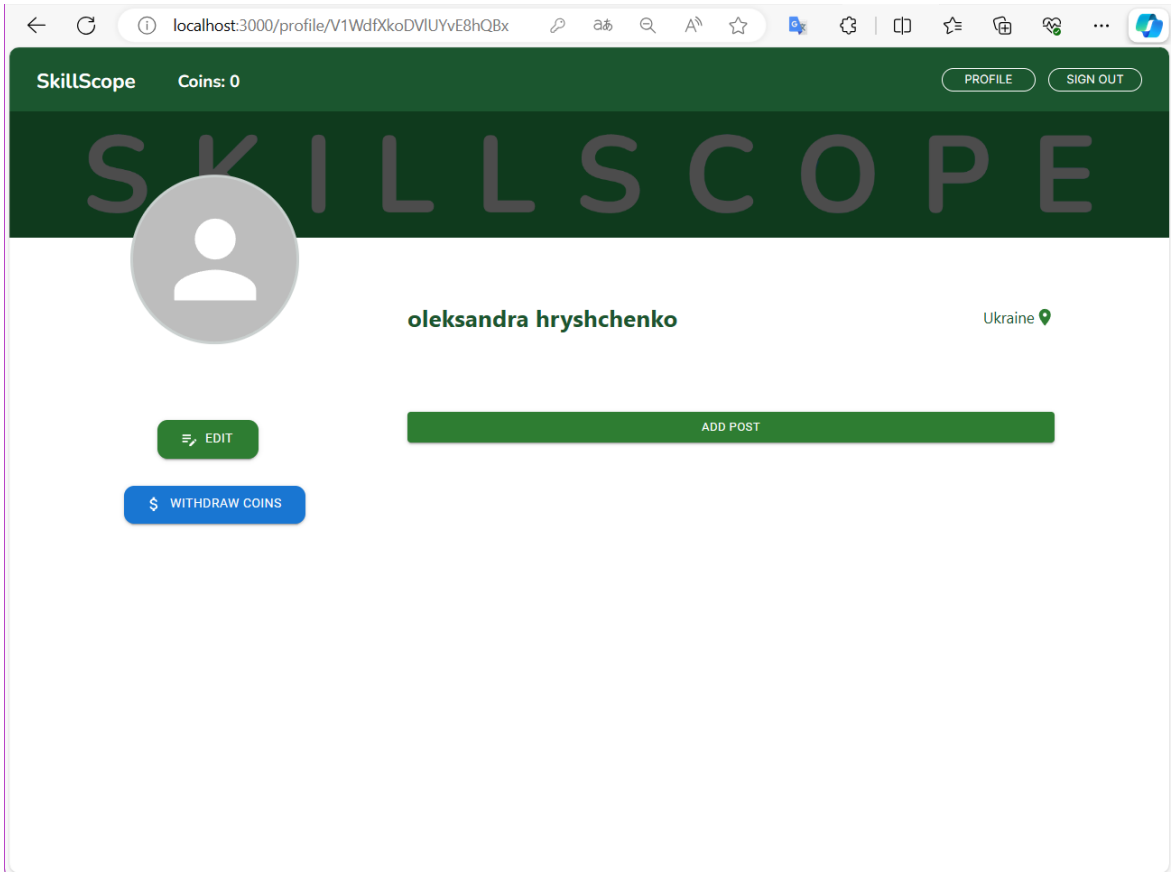

Рисунок 3.8 – Профіль таланта

3.3.2 Тестовий сценарій «Вхід у систему»

Цей сценарій схожий на реєстрацію, за винятком, що після входу на головну сторінку за посиланням «/talents» у хедері необхідно натиснути кнопку «SIGN IN». Після цього відкриється сторінка входу у систему за посиланням «/signin» (Рисунок 3.9), де потрібно ввести пошту, під якою зареєстрований користувач, та пароль, і якщо вони знайдені у системі, користувач буде перенаправлений на свою сторінку за посиланням «/profile/:userId».

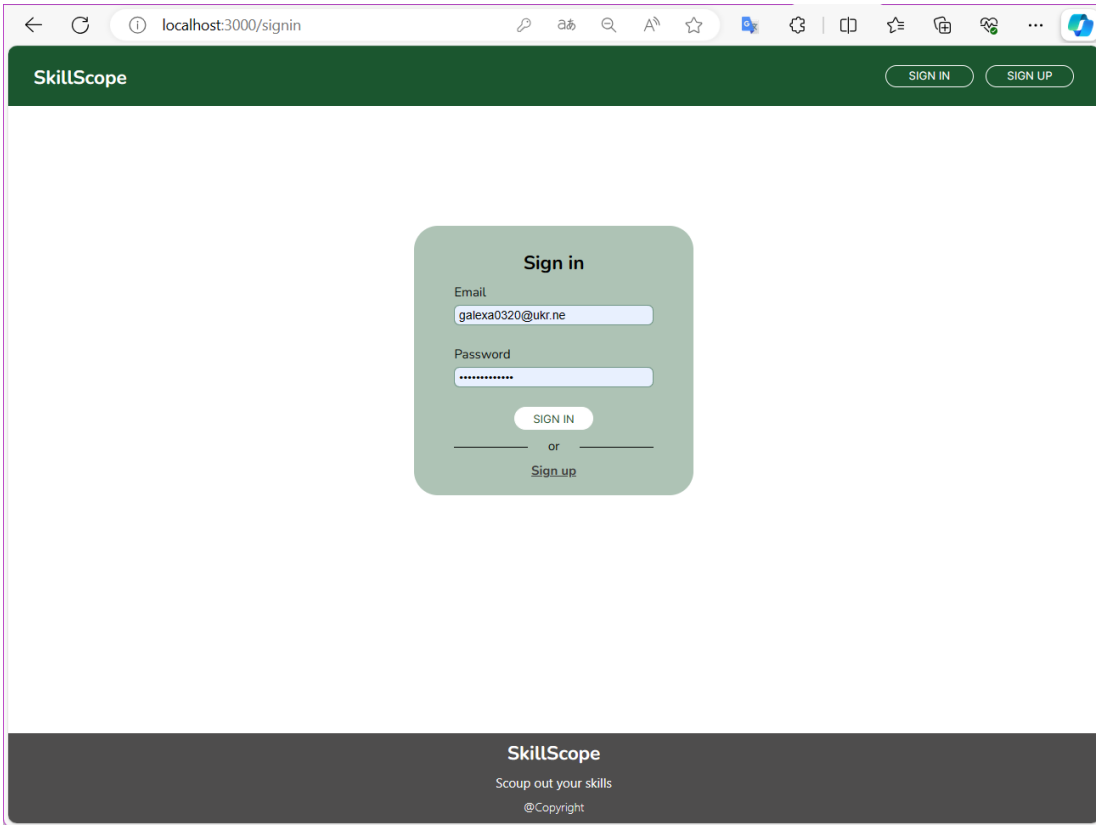

Рисунок 3.9 – Сторінка входу у систему

3.3.3 Тестовий сценарій «Редагування профілю»

Після того як користувач зареєструвався чи увійшов у систему, він може відредагувати дані, що були надані при реєстрації чи додати фото або статус свого профіля, якщо користувач під роллю «талант». Для цього потрібно перейти за посиланням «/profile/:userId» та натиснути кнопку «EDIT», що перенесе користувача на сторінку редагування профілю за посиланням «/profile/:userId/edit». Наповнення буде відрізнятись в залежності від ролі:

сторінка редагування профілю для користувача з роллю «талант» зображена на Рисунок .

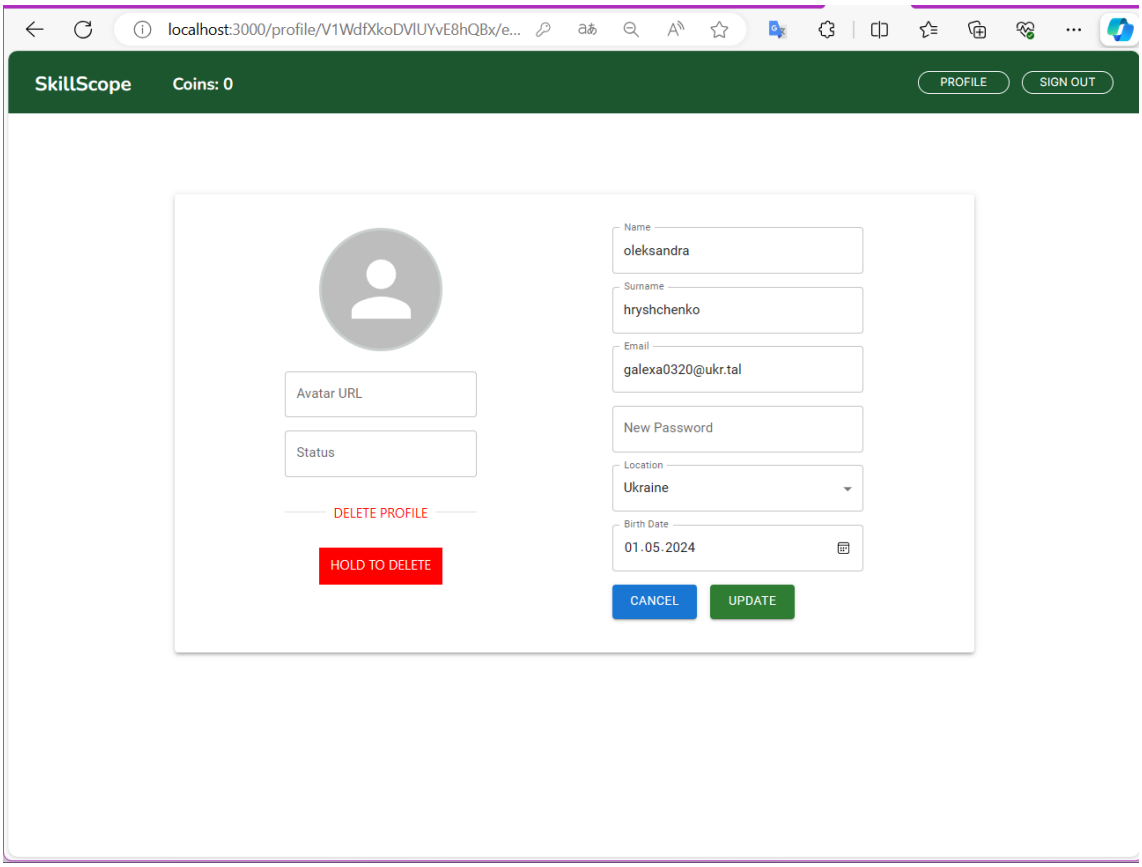

Рисунок 3.10 – Сторінка редагування профілю для таланта

Наповнення цієї сторінки буде дещо відрізнятися для спонсора, а саме вона буде бракувати поля додавання аватару та поля додавання статусу, оскільки їх сторінки все одно ніхто не буде бачити, окрім самого користувача (Рисунок).

Після заповнення усіх деталей, що хоче змінити користувач, необхідно натиснути кнопку «UPDATE», що збереже нові дані у базу даних, якщо вони пройдуть валідацію на фронтенді.

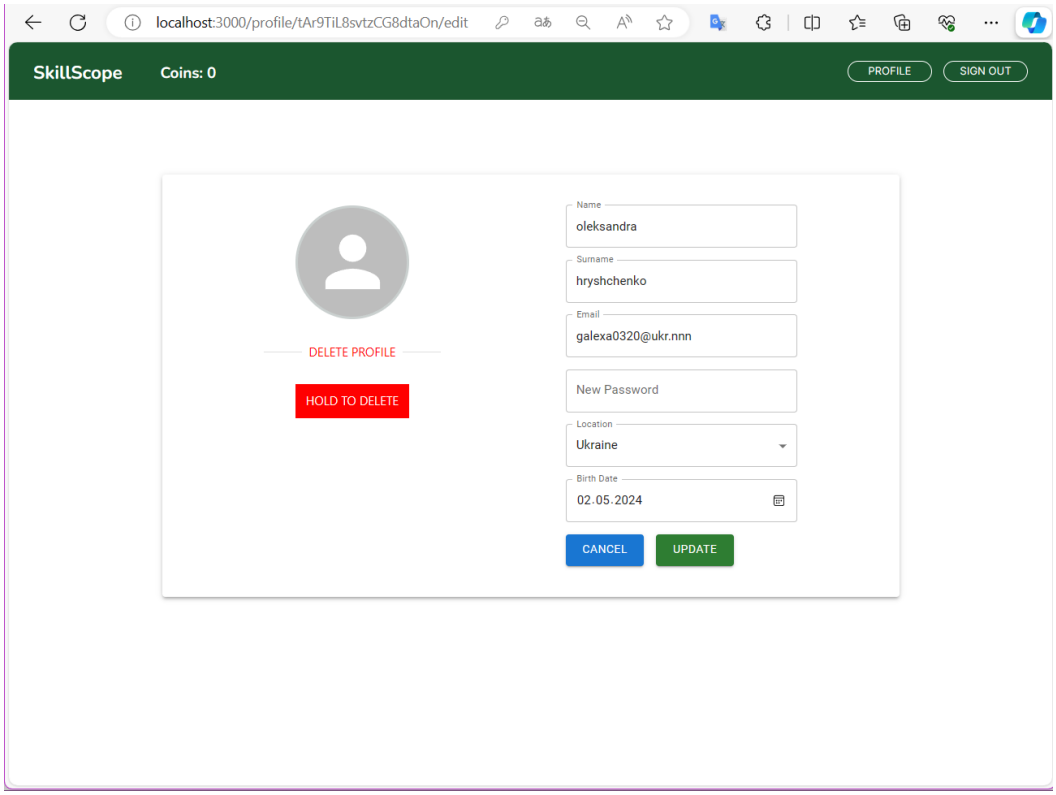

Рисунок 3.11 – Сторінка редагування профілю для спонсора

3.3.4 Тестовий сценарій «Видалення профілю»

Після того як користувач зареєструвався чи увійшов у систему, він може видалити профіль за власним бажанням на сторінці редагування профілю «/profile/:userId/edit». Необхідно натиснути та затиснути кнопку «HOLD TO DELETE» на 10 секунд (Рисунок 3.12).

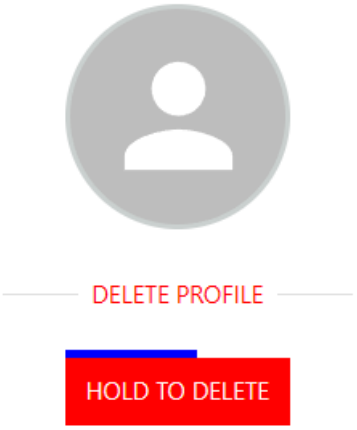

Рисунок 3.12 – Процес видалення профілю користувача

Після цього сторінка користувача та дані про нього видаляться із бази даних, користувач автоматично вийде із системи та перенесеться на головну сторінку «/talents».

3.3.5 Тестовий сценарій «Вихід із системи»

Після того, як користувач зайшов у систему, він може з неї вийти з будь-якої сторінки системи. Для цього кнопка «SIGN OUT» завжди розміщена у хедері. Після її натиснення, користувач автоматично вийде із системи та перенесеться на головну сторінку «/talents» (Рисунок 3.13).

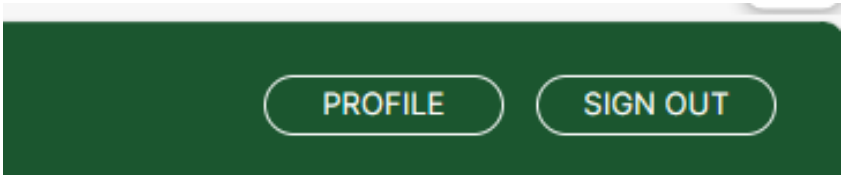

Рисунок 3.13 – Кнопка виходу із системи у хедері вебсайта

3.3.6 Тестовий сценарій «Додавання поста»

Для того, щоб додати пост на сторінку, користувач має зареєструватись чи увійти під роллю «талант» та зайти на свою сторінку «/profile/:userId», де натиснути кнопку «ADD POST», що перенесе користувача на сторінку «/profile/:userId/post», де талант може створити новий пост від свого імені (Рисунок 3.14).

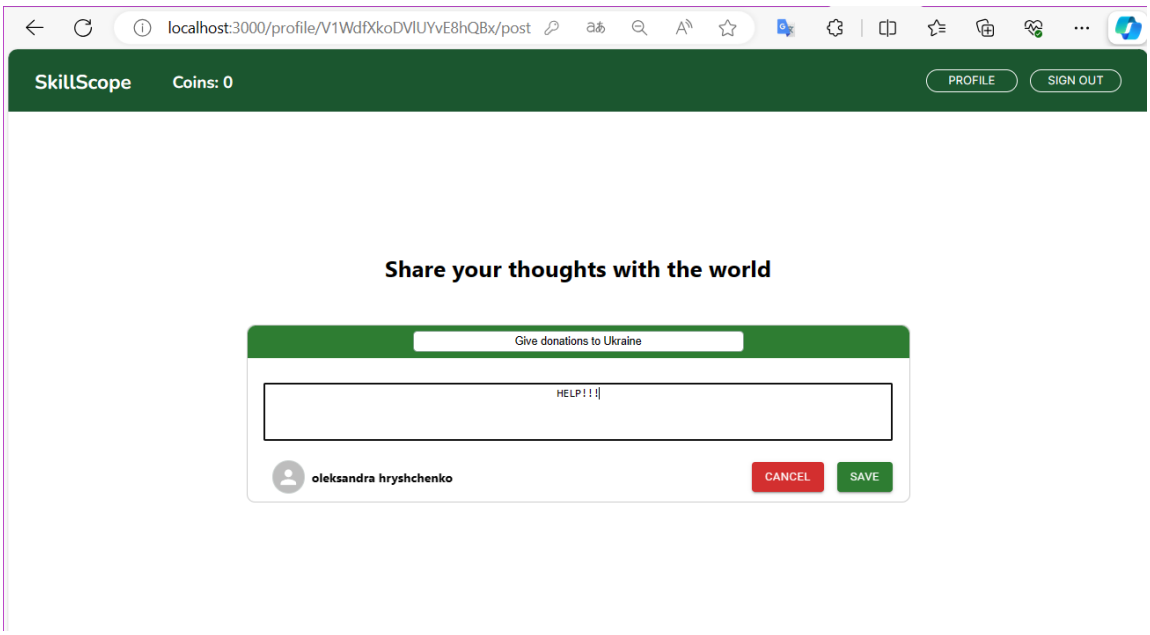

Рисунок 3.14 – Сторінка додавання поста

Після цього талант має ввести назву поста, що буде уособлювати його головну думку чи ціль, та опис, де буде задана головна інформація, та натиснути кнопку «SAVE». Якщо пост пройшов валідацію на фронтенді, він додається у систему та відобразиться на сторінці таланта (Рисунок 3.15).

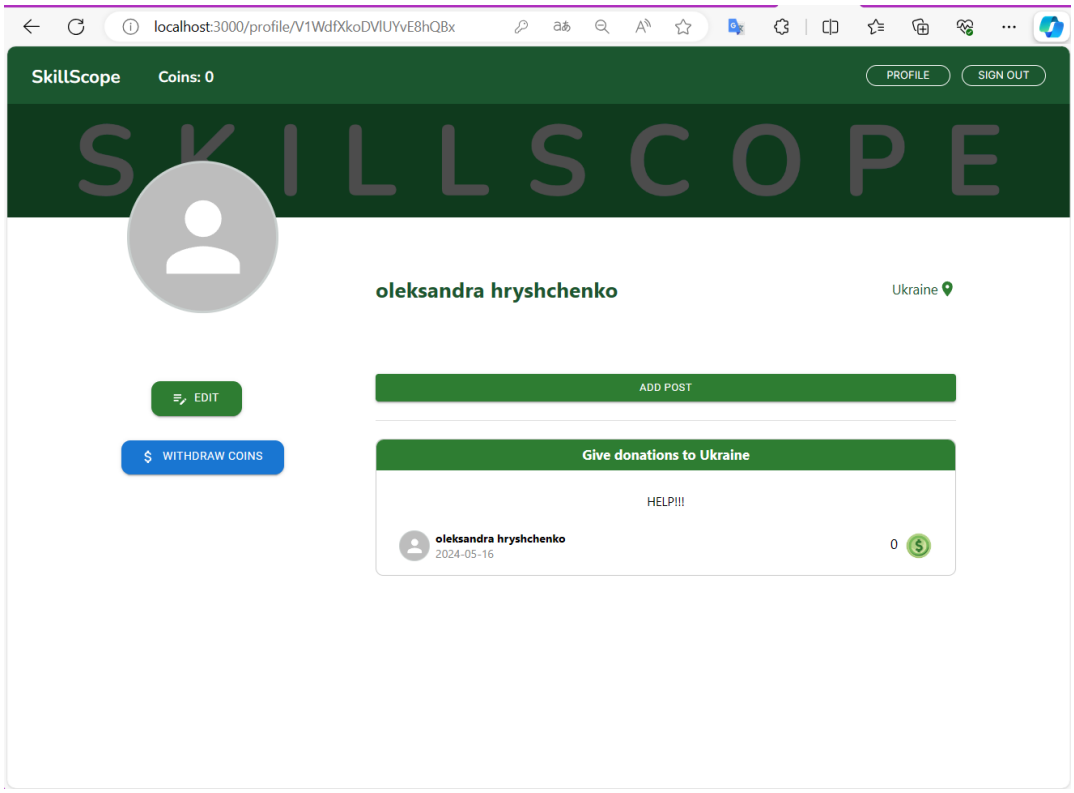

Рисунок 3.15 – Сторінка таланта з доданим постом

3.3.7 Тестовий сценарій «Спонсорування таланта»

Для того, щоб проспонсорувати таланта, користувач має увійти під роллю «спонсор». Оскільки при реєстрації спонсора, акаунт створюється з нульовим балансом, перед спонсоруванням спонсор має купити внутрішню валюту Coins. Для цього він має перейти на свій профіль за посиланням «/profile/:userId» та натиснути кнопку «BUY COINS», що на Рисунок 3.16.

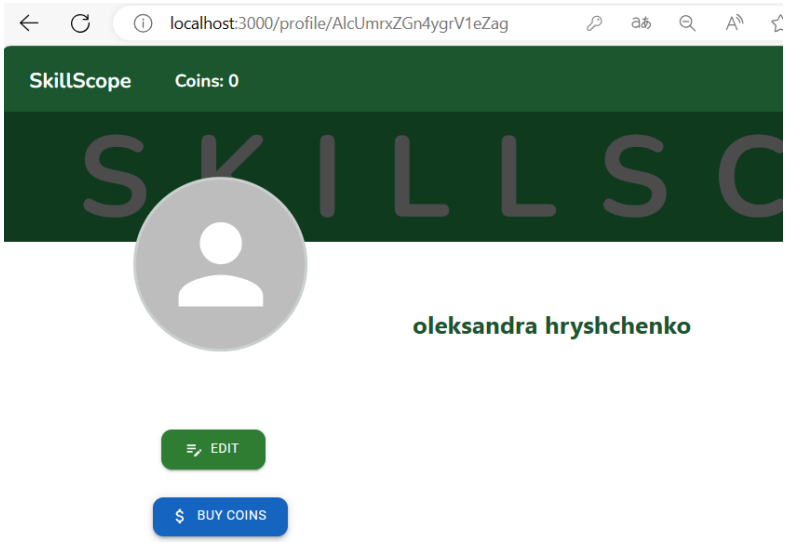

Рисунок 3.16 – Конопка купівлі внутрішньої валюти Coins

Після цього користувача буде перенесена на спеціальну сторінку «/profile/:userId/coins», де від спонсора вимагається введення деталей його банківського рахунку та кількості Coins (від 0 до 999), які користувач хоче купити (Рисунок 3.17).

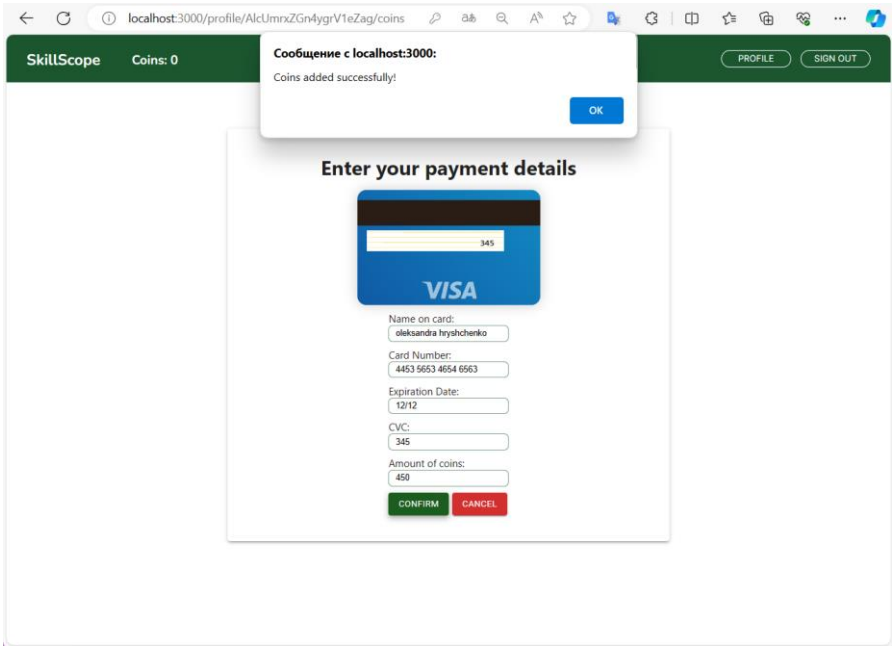

Рисунок 3.17 - Процес купівлі Coins

Після заповнення усіх полів, необхідно натиснути кнопку «CONFIRM» – це додасть введену кількість внутрішньої валюти до балансу, що відкриє можливість до спонсорування (Рисунок 3.18).

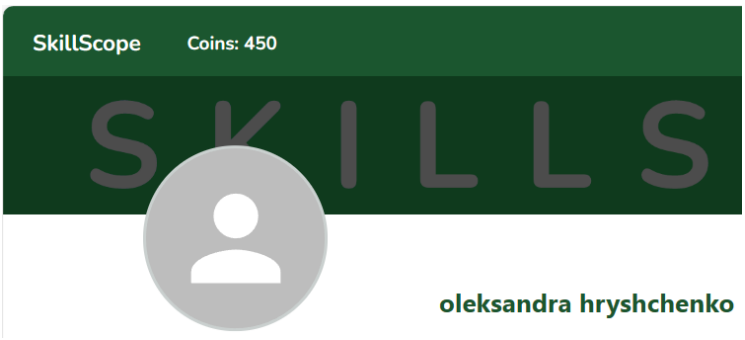

Рисунок 3.18 - Поповнений баланс спонсора

Тепер необхідно перейти на головну сторінку вебсайту натиснувши лого в хедері або за посиланням «/talents» та обрати профіль таланта, що сподобався та, натиснувши кнопку «VISIT» на необхідній карточці таланту, перейти на профіль цього таланту (Рисунок 3.19-3.20).

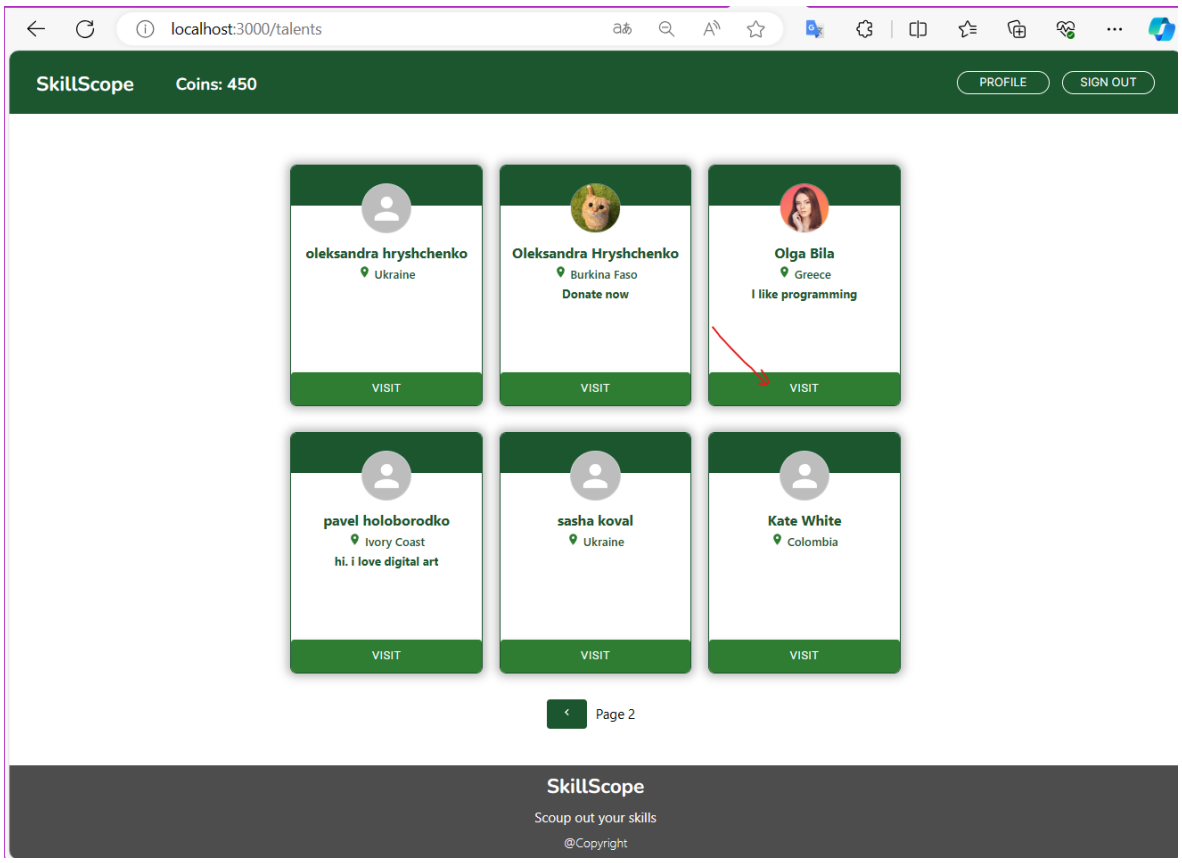

Рисунок 3.19 - Процес вибору таланта для спонсорування

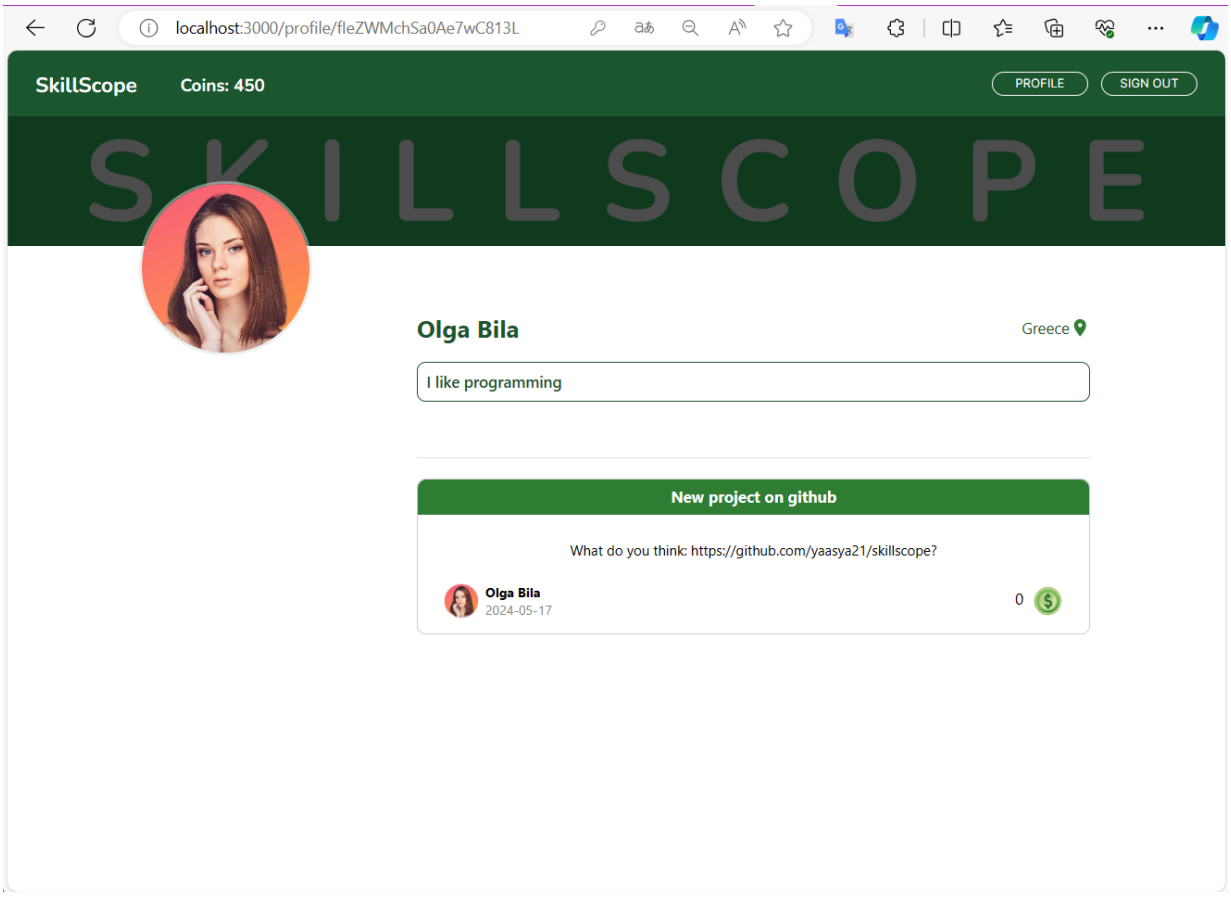

Рисунок 3.20 – Сторінка обраного таланта

Проспонсорувати таланта – значить спонсорувати пости, які є на його сторінці. Після натискання кнопки «Coin» на карточці поста (Рисунок 3.21), з балансу спонсора віднімається 1 одиниця внутрішньої валюти. Спонсорувати пости можна, поки вистачає валюти на рахунку спонсора: буде відніматись по 1 одиниці Coin за один клік по кнопці. Після цього гроші від спонсора перейдуть до проспонсорованого таланта (Рисунок 3.22).

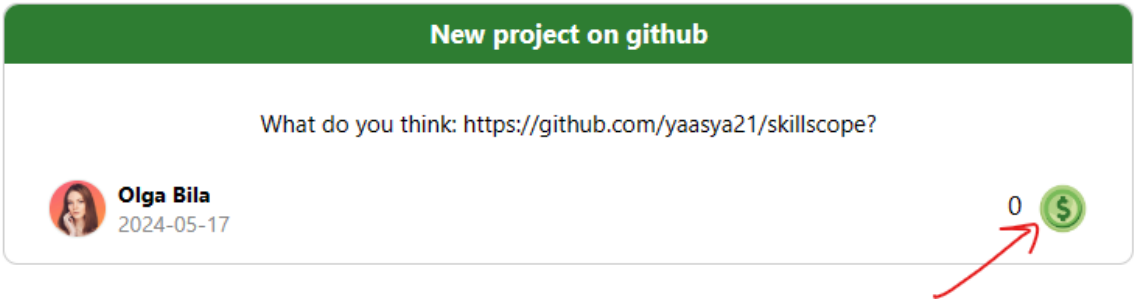

Рисунок 3.21 – Кнопка «Coin»

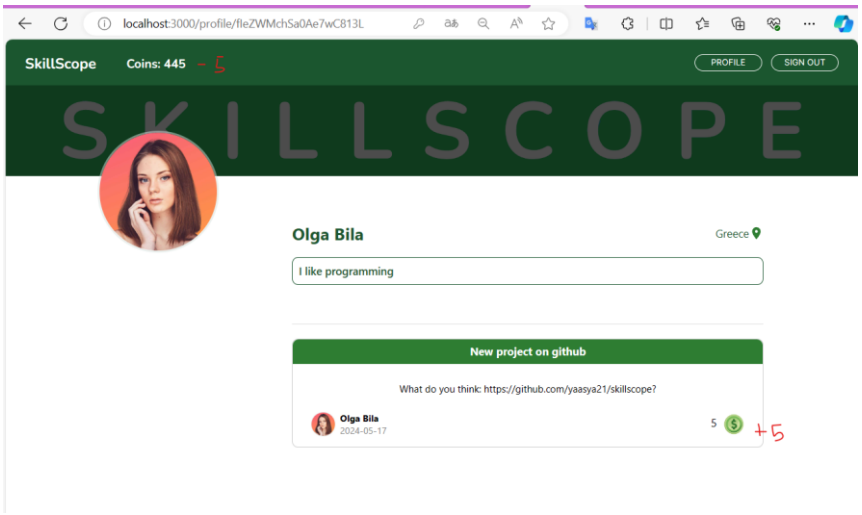

Рисунок 3.22 - Процес спонсорування таланта спонсором

3.3.8 Тестовий сценарій «Виведення внутрішньої валюти із системи талантом»

Після того, як на акаунт таланта надійшли Coins, він може вивести ці гроші на свою банківську карту, перейшовши на свій профіль «/profile/:userId» та натиснувши кнопку «WITHDRAW COINS» (Рисунок 3.23).

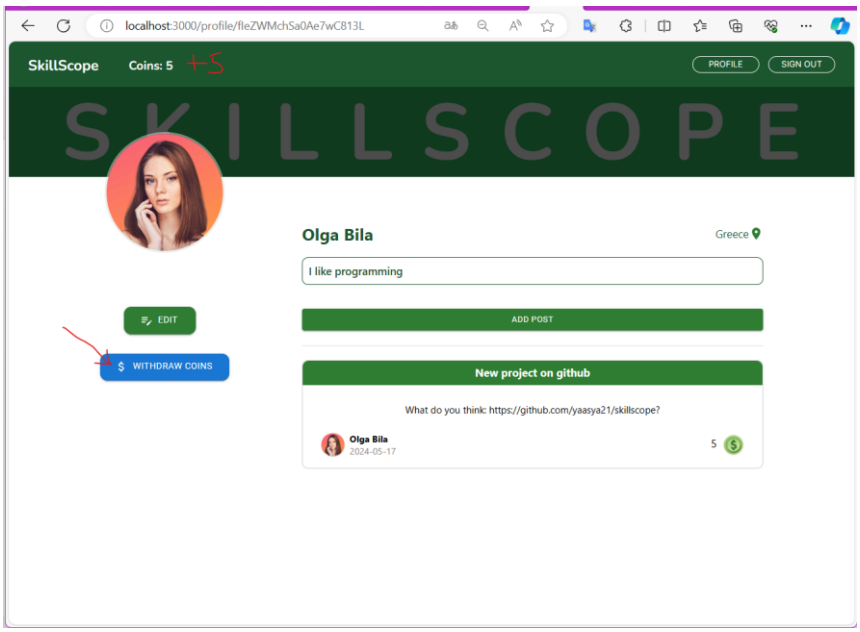

Рисунок 3.23 – Кнопка виводу грошей на банківську картку

Після цього таланта перенаправить на сторінку «/profile/:userId/coins», де він має ввести дані про свою банківську картку та кількість Coins, яку він хоче вивести на картку, що має бути менше або дорівнювати кількості внутрішньої валюти на балансі (Рисунок 3.24).

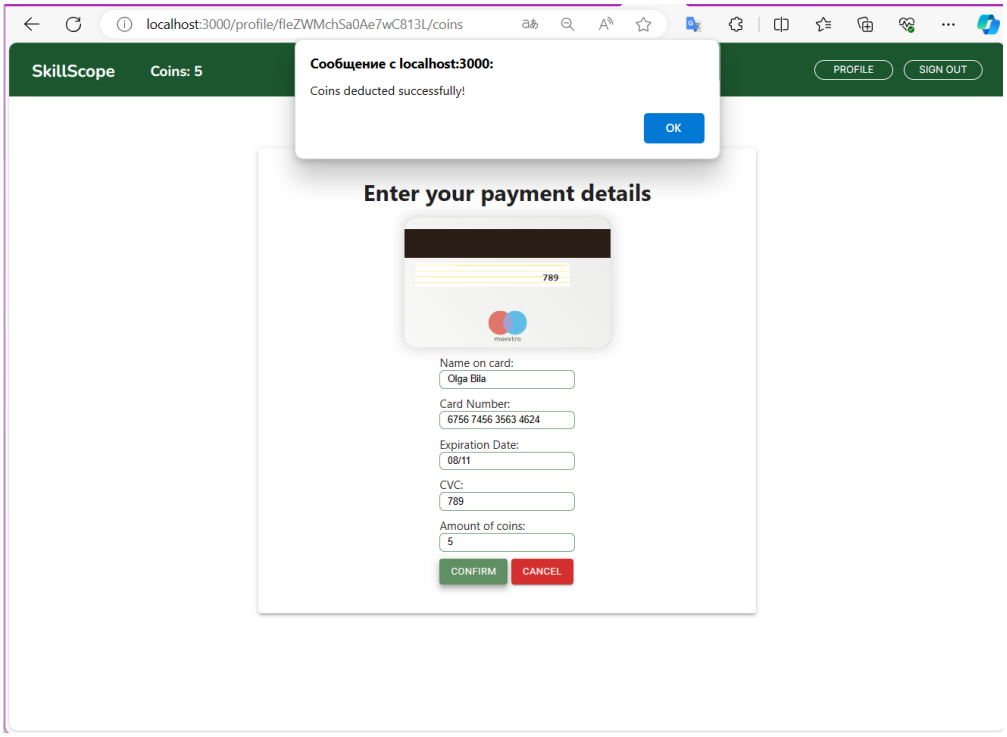

Рисунок 3.24 - Процес виведення Coins талантом

Після цього введена кількість валюти відніметься з балансу таланта (Рисунок 3.25).

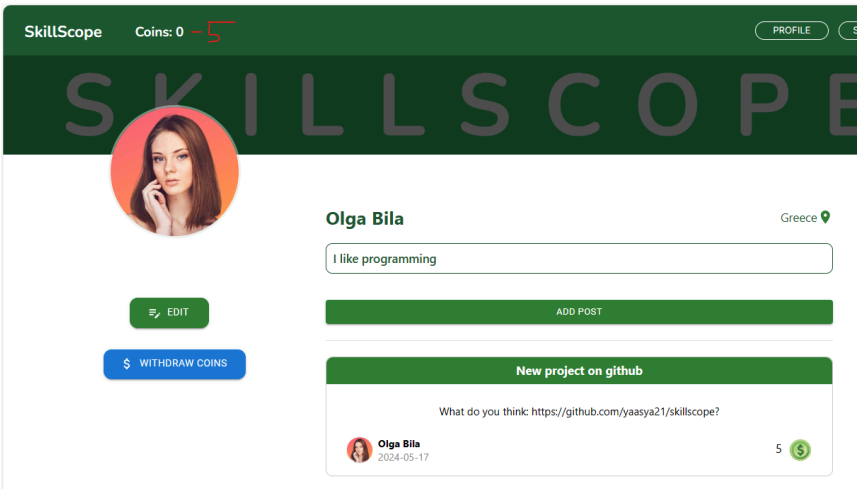

Рисунок 3.25 - Баланс таланта після виведення Coins

Як видно з результатів тестових сценаріїв вебзастосунок працює успішно і на цьому можна вважати процес розробки успішно закінченим. Надалі планується додати наступне:

—адаптувати вебсайт до екранів мобільних телефонів;

—додати навички, які може собі додавати талант, а спонсор може фільтрувати профілі по них на головній сторінці;

—замість симуляції проведення платежу, додати реальну оплату та виведення коштів через систему PayPal;

—вбудувати JWT-токени та авторизацію через сторонні сервіси (Google, Facebook тощо) для збільшення безпеки даних та зниження ризику несанкціонованого доступу.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було вирішено наступні задачі, що були зазначені в постановці задачі:

—вивчено теорію програмування вебсайту, розглянуто сучасні мови програмування для його створення;

—проведено аналіз тематичної літератури і інформації з заданої теми, на основі цього було розроблено модель і логіку роботи інформаційної системи;

—проведено аналіз сайтів-аналогів та визначено слабкі та сильні сторони конкурентів;

—створено прототипи головних сторінок вебсайту, на основі яких робилась його розмітка;

—продумано бізнес-логіку, якій слідує інформаційна система;

—спроєктовано базу даних на основі денормованої UML-діаграми, успішно імплементовано її у проєкт;

—дотримано best-practices при розробці дружнього до користувача інтерфейсу та наповнення вебсайту, створено адаптивні до розміру екрану макети, реалізовано функціонал, що зазначений бізнес-логікою, в тому числі функціонал спонсорування;

—проведено тестування користувачами з різними ролями та за різними тестовими сценаріями.

У результаті було отримано робочу та стабільну версію продукту – інформаційної системи SkillScope для пошуку та спонсорування талантів, яка допоможе контент-творцям монетизувати свої роботи у цифровому просторі.

Розроблений вебсайт має ряд цікавого функціоналу як:

—інтуїтивно зручний, продуманий, стильний та зручний інтерфейс;

—можливість реєстрації та авторизації;

—пагінація головної сторінки;

—робота з динамічною базою даних, що постійно оновлюється;

—зручна і зрозуміла навігація сайту;

—валідація полів, що збільшує зрозумілість системи та покращує User Experience;

—залежність функціональності сайту від категорії користувача;

—можливість кастомізації власного профілю для вираження індивідуальності користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Exhibition - strange time. *Strange Time*. URL: https://strangetime.art/exhibition/ (date of access: 03.05.2024).

2. Що таке донат (donate)? Де і для чого люди донатять? Хто може збирати донати?. *tseivo.com*. URL: https://tseivo.com/b/LocalWeirdo/t/kgdeea7gmr (дата звернення: 03.05.2024).

3. Want to help Ukraine's military as a foreigner? Here's what you can do. *The Kyiv Independent*. URL: https://kyivindependent.com/want-to-help-ukraines-military-as-a-foreigner-heres-what-you-can-do/ (date of access: 03.05.2024).

4. У найбільші фонди країни за рік задонатили майже $1 мільярд | Київщина 24/7. *Київщина 24/7*. URL: https://kyivschina24.com/news/u-najbilshi-fondy-krayiny-za-rik-zadonatyly-majzhe-1-milyard/ (дата звернення: 03.05.2024).

5. Що таке краудфандингова платформа?. *Дія.Бізнес*. URL: https://business.diia.gov.ua/handbook/finansovij-menedzment/so-take-kraudfandingova-platforma (дата звернення: 02.05.2024).

6. Freedman D. M., Nutting M. R. A brief history of crowdfunding including rewards, donation, debt, and equity platforms in the USA. John Wiley & Sons, Inc, 2015. URL: https://www.slideshare.net/slideshow/history-of-crowdfunding-46960230/46960230 (date of access: 02.05.2024).

7. Ринок цифрового мистецтва: як працює NFT і чи зможуть на ньому розбагатіти українські художники. *Суспільне Культура*. URL: https://suspilne.media/culture/112941-rinok-cifrovogo-mistectva-ak-vin-pracue-i-ci-zmozut-na-nomu-rozbagatiti-ukrainski-hudozniki/ (дата звернення: 02.05.2024).

8. Kickstarter. *Kickstarter*. URL: https://www.kickstarter.com/?& (date of access: 03.05.2024).

9. Moreau E. Not sure what kickstarter is all about? Here's what you need to know. *Lifewire*. URL: https://www.lifewire.com/what-is-kickstarter-3486258 (date of access: 02.05.2024).

10. Buy me a coffee. *Buy Me a Coffee*. URL: https://buymeacoffee.com/ (date of access: 02.05.2024).

11. Донателло / donatello. *Donatello - підтримка українського контенту*. URL: https://donatello.to/ (дата звернення: 02.05.2024).

12. Донателло / donatello. *Donatello - підтримка українського контенту*. URL: https://donatello.to/about (дата звернення: 02.05.2024).

13. Booch G. The unified modeling language user guide. 2nd ed. Upper Saddle River, NJ : Addison-Wesley, 2005. 185 p.

14. Booch G. The unified modeling language user guide. 2nd ed. Upper Saddle River, NJ : Addison-Wesley, 2015. 216 p.

15. Figma: the collaborative interface design tool. Figma. URL: https://www.figma.com/ (date of access: 03.05.2024).

16. DiLeo C., Cooper P. Beginning ruby 3: from beginner to pro. Apress L. P., 2020. 129 p.

17. Uzayr S. B. Mastering python for web: a beginner's guide. Taylor & Francis Group, 2022. 5 p.

18. Abelson H., Sussman G.J., Henz M., Wrigstad T., Sussman J. Structure and Interpretation of Computer Programs: JavaScript Edition. The MIT Press, 2022. 2 p.

19. Firestore | firebase. *Firebase*. URL: https://firebase.google.com/docs/firestore (date of access: 04.05.2024).

20. Biswas N. Beginning react and firebase: create four beginner-friendly projects using react and firebase. Apress L. P., 2021.

21. Narayn H. Just react!: learn react the react way. Apress L. P., 2022. 1-7 p.

22. State: a component's memory – react. *React*. URL: https://react.dev/learn/state-a-components-memory (date of access: 04.05.2024).

23. Microsoft. Visual studio code - code editing. redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (date of access: 04.05.2024).

24. Coronel Carlos, Morris Steven. Database Systems: Design, Implementation, and Management. 14th Edition. Cengage Learning, 2023. 36-47 p.

25. Narayn H. Just react!: learn react the react way. Apress L. P., 2022. 299 p.

ДОДАТОК А

Основні компоненти інтерфейсу

Решта коду розміщена на GitHub за посиланням

https://github.com/yaasya21/skillscope

App.jsx

```
const App = () => {
  return (
    <>
      <BrowserRouter>
        <Header />
        <Routes>
          <Route path="/">
            <Route index element={<Main />} />
            <Route path={"talents"} element={<Main />} />
            <Route path={"signup"} element={<SignUp />} />
            <Route path={"signin"} element={<SignIn />} />
          </Route>
          <Route path={"/profile"}>
            <Route path={":talentId"} element={<Profile />} />
            <Route path={":talentId/edit"} element={<EditProfile />} />
            <Route path={":talentId/post"} element={<AddPostProfile />} />
            <Route path={":talentId/coins"} element={<Payment />} />
          </Route>
          <Route path="*" element={<PageDoesNotExist />} />
        </Routes>
        <Footer />
      </BrowserRouter>
    </>
  );
};
```

AddPostProfile.jsx

```
const AddPostProfile = () => {
  const location = useLocation();
  const navigate = useNavigate();
  const idUser = location.pathname.split("/profile/")[1].split("/")[0];
  const isAddPost = location.pathname.includes("/post");
  const id = localStorage.getItem("id");

  const [userData, setUserData] = useState(null);
```

```
  useEffect(() => {
    const fetchData = async () => {
      if (id) {
        const userData = await getUserById(id);
        setUserData(userData);
      } else {
        navigate("/signin");
      }
    };

    fetchData();
  }, [id, navigate]);

  useEffect(() => {
    if (id !== idUser) {
      navigate(`/profile/${id}/post`);
    }
  }, [id, idUser, navigate]);

  return (
    <div className={styles.wrapper}>
      <h1>Share your thoughts with the world</h1>
      {userData && <Post id={id} isAddPost={isAddPost} userData={userData} />}
    </div>
  );
};
```

EditProfile.jsx

```
const EditProfile = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const idUser = location.pathname.split("/profile/")[1].split("/")[0];
  const role = localStorage.getItem("role");
  const id = localStorage.getItem("id");
  const [formData, setFormData] = useState(null);
  const [errors, setErrors] = useState({});
  const [originalPassword, setOriginalPassword] = useState("");

  useEffect(() => {
    const fetchData = async () => {
      const userData = await getUserById(id);
      if (userData) {
        setFormData(userData);
        setOriginalPassword(userData.password);
        userData.password = "";
      }
    };
```

```
  fetchData();
}, [id]);

useEffect(() => {
  if (!id) {
    navigate("/signin");
  }
  if (id !== idUser) {
    navigate(`/profile/${id}/edit`);
  }
}, [id, idUser, navigate]);

const handleChange = (e) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleSubmit = async (e) => {
  e.preventDefault();

  const formErrors = {};
  for (const field in formData) {
    if (registerOptions[field]) {
      const fieldRules = registerOptions[field];
      if (field === "password" && formData.password === "") {
        continue;
      }
      for (const rule in fieldRules) {
        if (rule === "required" && !formData[field]) {
          formErrors[field] = fieldRules[rule];
        } else if (
          rule === "pattern" &&
          !fieldRules[rule].value.test(formData[field])
        ) {
          formErrors[field] = fieldRules[rule].message;
        } else if (
          rule === "minLength" &&
          formData[field].length < fieldRules[rule].value
        ) {
          formErrors[field] = fieldRules[rule].message;
        } else if (
          rule === "maxLength" &&
          formData[field].length > fieldRules[rule].value
        ) {
          formErrors[field] = fieldRules[rule].message;
        } else if (
          rule === "validate" &&
          typeof fieldRules[rule].message === "function"
        ) {
          const validationResult = fieldRules[rule].message(formData[field]);
          if (validationResult) {
            formErrors[field] = validationResult;
```

```
          }
        }
      }
    }
  }

  if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
    return;
  }

  try {
    if (formData.password !== "") {
      formData.password = await hashPassword(formData.password);
    } else {
      formData.password = originalPassword;
    }
    await updateUser(id, formData);
    navigate(`/profile/${id}`);
  } catch (error) {
    console.error("Error updating user:", error);
  }
};

const handleDelete = async () => {
  try {
    await deleteUser(id);
    localStorage.removeItem("id");
    localStorage.removeItem("role");
    navigate("/talents");
  } catch (error) {
    console.error("Error deleting user:", error);
  }
};

return (
  <Box
    display="flex"
    justifyContent="center"
    alignItems="center"
    minHeight="750px"
    sx={{ p: 10, py: 5 }}
  >
    {formData && (
      <Paper
        elevation={3}
        style={{ display: "flex" }}
        sx={{ px: 10, py: 5 }}
        minWidth="300px"
      >
        <Grid container spacing={7}>
```

```
<Grid item xs={4} sx={{ mx: "auto" }} minWidth="300px">
  <Avatar
    src={formData.avatar ? formData.avatar : "/broken-image.jpg"}
    sx={{ width: 150, height: 150, mb: 3, mx: "auto" }}
    style={{
      border: "4px solid rgba(215, 227, 224, 0.5)",
    }}
  ></Avatar>
  {role === "talent" && (
    <>
      <TextField
        name="avatar"
        label="Avatar URL"
        value={formData.avatar}
        onChange={handleChange}
        fullWidth
        inputProps={{ maxLength: 300 }}
        sx={{ mb: 2 }}
      />
      <TextField
        name="status"
        label="Status"
        value={formData.status}
        onChange={handleChange}
        fullWidth
        multiline
        inputProps={{ maxLength: 300 }}
      />
    </>
  )}
  <Divider sx={{ my: 4, color: "red" }}>DELETE PROFILE</Divider>
  <ProgressButton
    longPressBackspaceCallback={() => handleDelete()}
  />
</Grid>
<Grid item xs={5} sx={{ mx: "auto" }} minWidth="300px">
  <TextField
    name="name"
    label="Name"
    value={formData.name}
    onChange={handleChange}
    fullWidth
    sx={{ mb: 2 }}
    error={!!errors.name}
    helperText={errors.name}
  />
  <TextField
    name="surname"
    label="Surname"
    value={formData.surname}
    onChange={handleChange}
```

```
  fullWidth
  sx={{ mb: 2 }}
  error={!!errors.surname}
  helperText={errors.surname}
/>
<TextField
  name="email"
  label="Email"
  value={formData.email}
  onChange={handleChange}
  fullWidth
  sx={{ mb: 2 }}
  error={!!errors.email}
  helperText={errors.email}
/>
<TextField
  name="password"
  label="New Password"
  value={formData.password}
  onChange={handleChange}
  type="password"
  fullWidth
  sx={{ mb: 2 }}
  error={!!errors.password}
  helperText={errors.password}
/>
<TextField
  name="location"
  label="Location"
  value={formData.location}
  onChange={handleChange}
  fullWidth
  select
  sx={{ mb: 2 }}
  error={!!errors.location}
  helperText={errors.location}
>
  {countryList.map((option) => (
    <MenuItem key={option.cca2} value={option.name}>
      {option.name}
    </MenuItem>
  ))}
</TextField>
<TextField
  name="birthDate"
  label="Birth Date"
  type="date"
  value={formData.birthDate}
  onChange={handleChange}
  fullWidth
  sx={{ mb: 2 }}
```

```
                error={!!errors.birthDate}
                helperText={errors.birthDate}
              />
              <Button
                type="submit"
                variant="contained"
                color="primary"
                size="large"
                sx={{ mr: 2 }}
                component={Link}
                to={`/profile/${id}`}
              >
                Cancel
              </Button>
              <Button
                type="submit"
                variant="contained"
                color="success"
                size="large"
                onClick={handleSubmit}
              >
                Update
              </Button>
            </Grid>
          </Grid>
        </Paper>
      )}
    </Box>
  );
};
```

Footer.jsx

```
const Footer = () => {
  const location = useLocation();
  const isProfilePage = location.pathname.includes("/profile");

  if (isProfilePage) {
    return null;
  }

  return (
    <footer className={styles.footer_wrap}>
      <div className={styles.footer_column}>
        <Logo />
        <p className={styles.moto}>Scoup out your skills</p>
        <p className={styles.main_text}>@Copyright</p>
      </div>
    </footer>
  );
```

```
};
```

Header.jsx

```
const Header = () => {
  const navigate = useNavigate();
  const id = localStorage.getItem("id");
  const isLogged = localStorage.getItem("id");
  const [coins, setCoins] = useState(0);

  useEffect(() => {
    if (!id) {
      return;
    }

    const userDocRef = doc(db, "users", id);

    const unsubscribe = onSnapshot(userDocRef, (docSnapshot) => {
      if (docSnapshot.exists()) {
        const userData = docSnapshot.data();
        setCoins(userData.coins);
      }
    });

    return () => {
      unsubscribe();
    };
  }, [id]);

  const handleSignOut = () => {
    localStorage.removeItem("id");
    localStorage.removeItem("role");
    navigate("/talents");
  };

  return (
    <header className={styles.header}>
      <div className={styles.logo_wrap}>
        <div className={styles.logo}>
          <Logo />
        </div>
        {isLogged && (
          <div className={styles.coins}>
            <span>Coins: {coins} </span>
          </div>
        )}
      </div>
      {isLogged ? (
        <div className={styles.button_wrap}>
          <NavLink className={styles.button_in} to={`profile/${id}`}>
```

```
            PROFILE
          </NavLink>
          <NavLink
            className={styles.button_in}
            onClick={handleSignOut}
            to="/talents"
          >
            SIGN OUT
          </NavLink>
        </div>
      ) : (
        <div className={styles.button_wrap}>
          <NavLink className={styles.button_in} to="/signin">
            SIGN IN
          </NavLink>
          <NavLink className={styles.button_in} to="/signup">
            SIGN UP
          </NavLink>
        </div>
      )}
    </header>
  );
};
```

Main.jsx

```
const Main = () => {
  const [talentDataList, setTalentDataList] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      const fetchedTalentData = await getTalents();
      setTalentDataList(fetchedTalentData);
    };

    fetchData();
  }, []);

  return (
    <div className={styles.wrapper}>
      <TalentList talentDataList={talentDataList} />
    </div>
  );
};
```

Payment.jsx

```
const Payment = () => {
  const navigate = useNavigate();
```

```
const location = useLocation();
const idUser = location.pathname.split("/profile/")[1].split("/")[0];
const userId = localStorage.getItem("id");

const [formData, setFormData] = useState({
  number: "",
  name: "",
  expiry: "",
  cvc: "",
  focused: "",
  coins: 0,
});

useEffect(() => {
  if (!userId) {
    navigate("/signin");
  }
  if (userId !== idUser) {
    navigate(`/profile/${userId}/coins`);
  }
}, [userId, idUser, navigate]);

const formRef = useRef(null);

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData({ ...formData, [name]: value });
  if (name === "number") {
    e.target.value = formatCreditCardNumber(value);
  } else if (name === "expiry") {
    e.target.value = formatExpirationDate(value);
  } else if (name === "cvc" || name === "coins") {
    e.target.value = formatCVC(value);
  } else if (name === "name") {
    e.target.value = formatName(value);
  }
};

const handleInputFocus = (e) => {
  setFormData({ ...formData, focused: e.target.name });
};

const handleSubmit = async (e) => {
  e.preventDefault();

  if (
    !formData.name ||
    !formData.number ||
    !formData.expiry ||
    !formData.cvc ||
    !formData.coins
```

```
  ) {
    alert("Please fill in all required fields.");
    return;
  }

  const role = localStorage.getItem("role");
  const amountOfCoins = parseInt(formData.coins || 0);

  try {
    const userData = await getUserById(userId);
    if (userData) {
      const userCoins = userData.coins;
      let updatedCoins = userCoins;

      if (role === "sponsor") {
        if (amountOfCoins === 0) {
          alert("Insufficient coins!");
          return;
        }
        updatedCoins += amountOfCoins;
        alert("Coins added successfully!");
      } else {
        if (userCoins >= amountOfCoins && amountOfCoins !== 0) {
          updatedCoins -= amountOfCoins;
          alert("Coins deducted successfully!");
        } else if (amountOfCoins === 0) {
          alert("Insufficient coins!");
          return;
        } else {
          alert("Not enough coins!");
          return;
        }
      }

      await updateCoins(userId, updatedCoins);
      navigate(`/profile/${userId}`);
    } else {
      console.log("User not found in the database");
    }
  } catch (error) {
    console.error("Error updating coins:", error);
  }
};

return (
  <Box
    display="flex"
    justifyContent="center"
    alignItems="center"
    minHeight="800px"
  >
```

```
<Paper elevation={3} style={{ width: "50%" }} sx={{ px: 10, py: 5 }}>
  <div key="Payment">
    <div className={styles.wrapper}>
      <h1>Enter your payment details</h1>

      <Cards
        number={formData.number}
        name={formData.name}
        expiry={formData.expiry}
        cvc={formData.cvc}
        focused={formData.focused}
      />

      <form ref={formRef} onSubmit={handleSubmit}>
        <div className={styles.input}>
          <span>Name on card:</span>
          <input
            type="text"
            name="name"
            className="form-control"
            placeholder="Name"
            pattern="[a-z A-Z]+"
            required
            onChange={handleInputChange}
            onFocus={handleInputFocus}
          />
        </div>
        <div className={styles.input}>
          <span>Card Number:</span>
          <input
            type="tel"
            name="number"
            className="form-control"
            placeholder="Card Number"
            pattern="[\d| ]{16,22}"
            maxLength="19"
            required
            onChange={handleInputChange}
            onFocus={handleInputFocus}
          />
        </div>
        <div className={styles.input}>
          <span>Expiration Date:</span>
          <input
            type="tel"
            name="expiry"
            className="form-control"
            placeholder="Valid Thru"
            pattern="\d\d/\d\d"
            maxLength="5"
            required
```

```
    onChange={handleInputChange}
    onFocus={handleInputFocus}
  />
</div>
<div className={styles.input}>
  <span>CVC:</span>
  <input
    type="tel"
    name="cvc"
    className="form-control"
    placeholder="CVC"
    pattern="\d{3}"
    maxLength="3"
    required
    onChange={handleInputChange}
    onFocus={handleInputFocus}
  />
</div>
<div className={styles.input}>
  <span>Amount of coins:</span>
  <input
    type="tel"
    name="coins"
    className="form-control"
    placeholder="Amount of coins"
    pattern="\d{3}"
    maxLength="3"
    required
    onChange={handleInputChange}
  />
</div>
<Button
  type="submit"
  variant="contained"
  color="success"
  size="medium"
  onClick={handleSubmit}
  style={{
    marginRight: "5px",
  }}
>
  confirm
</Button>
<Button
  type="submit"
  variant="contained"
  color="error"
  size="medium"
  component={Link}
  to={`/profile/${userId}`}
>
```

```
              cancel
            </Button>
          </form>
        </div>
      </div>
    </Paper>
  </Box>
 );
};
```

Post.jsx

```
const Post = ({ id, postId, idLocal, isAddPost, userData, postData }) => {
  const navigate = useNavigate();
  const today = new Date();
  const formattedDate = today.toISOString().split("T")[0];
  const role = localStorage.getItem("role");

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  const [coins, setCoins] = useState(postData?.coins || 0);

  useEffect(() => {
    if (!isAddPost) {
      const postDocRef = doc(db, "posts", postData.id);
      const unsubscribe = onSnapshot(postDocRef, (docSnapshot) => {
        if (docSnapshot.exists()) {
          const updatedCoins = docSnapshot.data().coins;
          setCoins(updatedCoins);
        }
      });

      return () => unsubscribe();
    }
  });

  const handleAddCoin = async () => {
    await addCoin(idLocal, postId, id);
  };

  const onSubmit = async (data) => {
    await addPost(data, formattedDate, id);
    navigate(`/profile/${id}`);
  };

  if (isAddPost) {
```

```
return (
  <div className={styles.addpost_container}>
    <form onSubmit={handleSubmit(onSubmit)}>
      <div className={styles.post_header}>
        <input
          type="text"
          name="header"
          placeholder="Header"
          {...register("header", registerOptions.header)}
          style={{
            width: "50%",
            borderRadius: "5px",
            padding: "2px",
          }}
        />
      </div>
      <div className={styles.post_description}>
        <textarea
          name="description"
          placeholder="Description"
          {...register("description", registerOptions.description)}
          style={{
            resize: "none",
            textAlign: "center",
            padding: "4px",
            width: "100%",
            height: "100%",
          }}
        />
        {errors.header && (
          <p className={styles.error}>{errors.header.message}</p>
        )}
        {errors.description && (
          <p className={styles.error}>{errors.description.message}</p>
        )}
      </div>
      <div className={styles.author_container}>
        <div className={styles.namedate_container}>
          <Avatar
            src={userData.avatar ? userData.avatar : "/broken-image.jpg"}
            sx={{ width: 40, height: 40, mr: 1 }}
            style={{
              border: "2px solid rgba(215, 227, 224, 0.5)",
            }}
          ></Avatar>
          <div>
            <div
              className={styles.author_name}
            >{`${userData.name} ${userData.surname}`}</div>
          </div>
        </div>
```

```
          <div>
            {" "}
            <Button
              variant="contained"
              size="medium"
              color="error"
              sx={{ mr: 2 }}
              component={Link}
              to={`/profile/${id}`}
            >
              CANCEL
            </Button>
            <Button
              variant="contained"
              size="medium"
              type="submit"
              color="success"
            >
              SAVE
            </Button>
          </div>
        </div>
      </form>
    </div>
  );
} else {
  return (
    <div className={styles.post_container}>
      <div className={styles.post_header}>{postData.header}</div>
      <div className={styles.post_description}>{postData.description}</div>
      <div className={styles.author_container}>
        <div className={styles.namedate_container}>
          <Avatar
            src={userData.avatar ? userData.avatar : "/broken-image.jpg"}
            sx={{ width: 40, height: 40, mr: 1 }}
            style={{
              border: "2px solid rgba(215, 227, 224, 0.5)",
            }}
          ></Avatar>
          <div>
            <div
              className={styles.author_name}
            >{`${userData.name} ${userData.surname}`}</div>
            <div className={styles.post_date}>{postData.date}</div>
          </div>
        </div>
        <div className={styles.counter}>
          {coins}
          <IconButton disabled={role !== "sponsor"} onClick={handleAddCoin}>
            <img
              className={styles.coin}
```

```
                src={require("./img/coin.png")}
                alt="coin"
              />
            </IconButton>
          </div>
        </div>
      </div>
    );
  }
};
```

Profile.jsx

```
const Profile = () => {
  const location = useLocation();
  const navigate = useNavigate();
  const id = location.pathname.replace("/profile/", "");
  const idUser = localStorage.getItem("id");
  const role = localStorage.getItem("role");

  const [userData, setUserData] = useState(null);

  useEffect(() => {
    const fetchUserData = async () => {
      try {
        const userData = await getUserById(id);
        setUserData(userData);
      } catch (error) {
        console.error("Error fetching user data:", error);
      }
    };

    fetchUserData();
  }, [id]);

  useEffect(() => {
    if (!idUser) {
      navigate("/signin");
    }
  }, [idUser, navigate]);

  return (
    <>
      {userData && (
        <>
          <div className={styles.plug}></div>
          <div className={styles.wrapper}>
            {userData && (
              <>
                <ProfileSide
                  userData={userData}
```

```
                  idLocal={id}
                  idUser={idUser}
                  role={role}
                />
                <ProfileMain
                  userData={userData}
                  idLocal={id}
                  idUser={idUser}
                  role={role}
                />
              </>
            )}
          </div>
        </>
      )}
    </>
  );
};
```

SignIn.jsx

```
const SignIn = () => {
  const navigate = useNavigate();
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();
  const [errorMessage, setErrorMessage] = useState("");

  const onSubmit = async (data) => {
    try {
      const emailExists = await checkEmailExists(data.email);

      if (emailExists) {
        const userData = await getUserByEmail(data.email);

        if (userData) {
          const { password, role } = userData;
          const passwordCorrect = await verifyUserPassword(
            data.password,
            password
          );

          if (passwordCorrect) {
            const id = await getUserId(data.email);
            localStorage.setItem("id", id);
            localStorage.setItem("role", role);
            navigate(`/profile/${id}`);
          } else {
            setErrorMessage("Incorrect password");
```

```
        }
      } else {
        setErrorMessage("User not found");
      }
    } else {
      setErrorMessage("User not found");
    }
  } catch (error) {
    console.error("Error signing in:", error.message);
    setErrorMessage("Error signing in");
  }
};

const verifyUserPassword = async (enteredPassword, hashedPassword) => {
  try {
    return await verifyPassword(enteredPassword, hashedPassword);
  } catch (error) {
    console.error("Error verifying password:", error.message);
    throw new Error("Error verifying password");
  }
};

return (
  <>
    <form className={styles.signin_form} onSubmit={handleSubmit(onSubmit)}>
      <h2 className={styles.signin_form_elem}>Sign in</h2>
      <div className={styles.signin_form_elem}>
        <label>Email</label>
        <br />
        <input
          type="text"
          {...register("email", {
            ...registerOptions.email,
            validate: async (value) =>
              (await checkEmailExists(value)) || "Email not found",
          })}
          className={styles.signin_form_elem}
        />
        {errors.email && (
          <p className={styles.error}>{errors.email.message}</p>
        )}
      </div>
      <div className={styles.signin_form_elem}>
        <label>Password</label>
        <br />
        <input
          type="password"
          {...register("password")}
          className={styles.signin_form_elem}
        />
        {errorMessage && <p className={styles.error}>{errorMessage}</p>}
```

```
        </div>
        <button className={styles.signin_form_elem} type="submit">
          SIGN IN
        </button>
        <p className={styles.signin_form_elem}>or</p>
        <p>
          <NavLink className={styles.signin_form_elem} to={"/signup"}>
            <b>Sign up</b>
          </NavLink>
        </p>
      </form>
    </>
  );
};
```

SignUp.jsx

```
const SignUp = () => {
  const navigate = useNavigate();
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  const onSubmit = async (data) => {
    const role = data.role ? "sponsor" : "talent";
    try {
      const hashedPassword = await hashPassword(data.password);
      const userId = await addUser({
        name: data.name,
        surname: data.surname,
        email: data.email,
        password: hashedPassword,
        location: data.location,
        birthDate: data.birthDate,
        role: role,
        avatar: null,
        coins: 0,
      });
      console.log("Data added to Firestore successfully!");
      localStorage.setItem("role", role);
      localStorage.setItem("id", userId);
      navigate(`/profile/${userId}`);
    } catch (error) {
      console.error("Error adding data to Firestore: ", error);
    }
  };
  return (
    <div className={styles.signup}>
      <h1>Monetize your Skills</h1>
```

```
<div className={styles.form_wrap}>
  <form onSubmit={handleSubmit(onSubmit)}>
    <div className={styles.input_wrap}>
      <label htmlFor="name">Name</label>
      <input type="text" {...register("name", registerOptions.name)} />
      {errors.name && (
        <p className={styles.error}>{errors.name.message}</p>
      )}
    </div>
    <div className={styles.input_wrap}>
      <label htmlFor="surname">Surname</label>
      <input
        type="text"
        {...register("surname", registerOptions.surname)}
      />
      {errors.surname && (
        <p className={styles.error}>{errors.surname.message}</p>
      )}
    </div>
    <div className={styles.input_wrap}>
      <label htmlFor="email">Email</label>
      <input
        type="email"
        {...register("email", {
          ...registerOptions.email,
          validate: async (value) =>
            !(await checkEmailExists(value)) || "Email already taken",
        })}
      />
      {errors.email && (
        <p className={styles.error}>{errors.email.message}</p>
      )}
    </div>
    <div className={styles.input_wrap}>
      <label htmlFor="password">Password</label>
      <input
        type="password"
        {...register("password", registerOptions.password)}
      />
      {errors.password && (
        <p className={styles.error}>{errors.password.message}</p>
      )}
    </div>
    <div className={styles.input_wrap}>
      <label htmlFor="location">Location</label>
      <select {...register("location", registerOptions.location)}>
        <option value="">---- Select a country ----</option>
        {countryList.map((element) => (
          <option key={element.cca2} value={element.name}>
            {element.name}
          </option>
```

```
              ))}
            </select>
            {errors.location && (
              <p className={styles.error}>{errors.location.message}</p>
            )}
          </div>
          <div className={styles.input_wrap}>
            <label htmlFor="birthDate">Date of Birth</label>
            <input
              type="date"
              min="1900-01-01"
              max={new Date().toISOString().split("T")[0]}
              {...register("birthDate", registerOptions.birthDate)}
            />
            {errors.birthDate && (
              <p className={styles.error}>{errors.birthDate.message}</p>
            )}
          </div>
          <div className={styles.input_wrap}>
            <label htmlFor="role">Sponsor:</label>
            <input
              className={styles.checkbox}
              type="checkbox"
              {...register("role", {})}
            />
          </div>
          <button type="submit">SIGN UP</button>
        </form>
        <p className={styles.or}>or</p>
        <p className={styles.signin_check}>
          <NavLink className={styles.signin_form_elem} to={"/signin"}>
            Sign in
          </NavLink>
        </p>
      </div>
    </div>
  );
};
```

ДОДАТОК Б

Конфігурація та запити до бази даних

Firebase.js

```
const firebaseConfig = {
  apiKey: "AIzaSyA0PYhAKM7OXaECpQeNKFW6tqPIG7hL6kY",
  authDomain: "skillscope-70bd9.firebaseapp.com",
  projectId: "skillscope-70bd9",
  storageBucket: "skillscope-70bd9.appspot.com",
  messagingSenderId: "427267857498",
  appId: "1:427267857498:web:a7b03d0475bc3eaf7f0c63",
  measurementId: "G-ZQTFVFEC56"
};

const app = initializeApp(firebaseConfig);
```

addCoin.js

```
export const addCoin = async (userId, postId, sponsorId) => {
  try {
    const userDocRef = doc(db, "users", userId);
    const postDocRef = doc(db, "posts", postId);
    const sponsorDocRef = doc(db, "users", sponsorId);

    const sponsorDocSnapshot = await getDoc(sponsorDocRef);
    const sponsorCoins = sponsorDocSnapshot.data().coins;

    if (sponsorCoins >= 1) {
      await Promise.all([
        updateDoc(userDocRef, { coins: increment(1) }),
        updateDoc(postDocRef, { coins: increment(1) }),
        updateDoc(sponsorDocRef, { coins: increment(-1) }),
      ]);
    } else {
      alert("Insufficient coins!");
    }
  } catch (error) {
    console.error("Error adding coin:", error);
  }
};
```

addPost.js

```
export const addPost = async (data, formattedDate, id) => {
  try {
    await addDoc(collection(db, "posts"), {
      creatorId: id,
      header: data.header,
      description: data.description,
      date: formattedDate,
      coins: 0,
    });
    console.log("Data added to Firestore successfully!");
  } catch (error) {
    console.error("Error adding data to Firestore: ", error);
  }
};
```

addUser.js

```
export const addUser = async (userData) => {
  try {
    const docRef = await addDoc(collection(db, "users"), userData);
    return docRef.id;
  } catch (error) {
    console.error("Error adding user to Firestore:", error);
    throw new Error("Error adding user to Firestore");
  }
};
```

checkEmailExists.js

```
export const checkEmailExists = async (email) => {
  const q = query(collection(db, "users"), where("email", "==", email));
  const querySnapshot = await getDocs(q);
  return !querySnapshot.empty;
};
```

deleteUser.js

```
export const deleteUser = async (id) => {
  try {
    const userDocRef = doc(db, "users", id);
    await deleteDoc(userDocRef);
    console.log("User deleted successfully!");
  } catch (error) {
    console.error("Error deleting user:", error);
  }
};
```

getPosts.js

```
export const getPosts = async (creatorId) => {
    try {
      const q = query(
        collection(db, "posts"),
        where("creatorId", "==", creatorId)
      );
      const querySnapshot = await getDocs(q);
      const fetchedPosts = [];
      querySnapshot.forEach((doc) => {
        fetchedPosts.push({ id: doc.id, ...doc.data() });
      });
      fetchedPosts.sort((a, b) => new Date(b.date) - new Date(a.date));
      return fetchedPosts;
    } catch (error) {
      console.error("Error fetching posts:", error);
      return [];
    }
  };
```

getTalants.js

```
export const getTalents = async () => {
  try {
    const q = query(collection(db, "users"), where("role", "==", "talent"));
    const querySnapshot = await getDocs(q);
    const fetchedTalentData = [];
    querySnapshot.forEach((doc) => {
      fetchedTalentData.push({ id: doc.id, ...doc.data() });
    });
    return fetchedTalentData;
  } catch (error) {
    console.error("Error fetching talent data:", error);
    return [];
  }
};
```

getUserByEmail.js

```
export const getUserByEmail = async (email) => {
  const q = query(collection(db, "users"), where("email", "==", email));
  const querySnapshot = await getDocs(q);
  return querySnapshot.empty ? null : querySnapshot.docs[0].data();
};
```

getUserById.js

```
export const getUserById = async (id) => {
  try {
    const userDocRef = doc(db, "users", id);
    const userDocSnap = await getDoc(userDocRef);
```

```
    if (userDocSnap.exists()) {
      const userData = userDocSnap.data();
      return userData;
    } else {
      console.log("No such user!");
      return null;
    }
  } catch (error) {
    console.error("Error fetching document:", error);
    return null;
  }
};
```

getUserId.js

```
export const getUserId = async (email) => {
  const q = query(collection(db, "users"), where("email", "==", email));
  const querySnapshot = await getDocs(q);
  return querySnapshot.empty ? null : querySnapshot.docs[0].id;
};
```

updateCoins.js

```
export const updateCoins = async (userId, updatedCoins) => {
  try {
    const userDocRef = doc(db, "users", userId);
    await updateDoc(userDocRef, { coins: updatedCoins });
  } catch (error) {
    console.error("Error updating user coins:", error);
  }
};
```

updateUser.js

```
export const updateUser = async (id, formData) => {
  try {
    const userRef = doc(db, "users", id);
    await updateDoc(userRef, formData);
    console.log("User information updated successfully!");
  } catch (error) {
    console.error("Error updating user information:", error);
  }
};
```

ДОДАТОК В

Допоміжні функції

hashPassword.js

```
export const hashPassword = async (password) => {
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    return hashedPassword;
  } catch (error) {
    console.error("Error hashing password:", error);
    throw new Error("Error hashing password");
  }
};
```

verifyPassword.js

```
export const verifyPassword = async (password, hashedPassword) => {
  try {
    const isMatch = await bcrypt.compare(password, hashedPassword);
    return isMatch;
  } catch (error) {
    console.error("Error verifying password:", error);
    throw new Error("Error verifying password");
  }
};

const validatePassword = (value) => {
    // Check for invalid passwords - 8 identical characters
    if (/(\w)\1\1\1\1\1\1\1/.test(value)) {
        return "Must not contain 8 identical characters"
    }
    // Check for password requirements
    const requirementsRegex =
        /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!"#$%&'()*+,-./:;<=>?\\@[\]^_`{|}~]).{6,64}$/
    if (!requirementsRegex.test(value)) {
        return "Use [a-z], [A-Z], [0-9], special char"
    }
}

const validateDate = (value) => {
    const today = new Date()
    const birthDate = new Date(value)
    if (birthDate.getTime() > today.getTime()) {
        return "Birth date cannot be later than today"
```

```
    }
    if (birthDate.getYear() < 0) {
        return "This date is too old. Please select a more recent date"
    }
}
```

validationRules.js

```
const registerOptions = {
    name: {
        required: "Name is required",
        pattern: {
            value: /^[A-Za-z]+$/,
            message: "Name must be written in Latin",
        },
        minLength: {
            value: 1,
            message: "Name must have at least 1 characters",
        },
        maxLength: {
            value: 64,
            message: "Name must have maximum 64 characters",
        },
    },
    surname: {
        required: "Surname is required",
        pattern: {
            value: /^[A-Za-z]+$/,
            message: "Surname must be written in Latin",
        },
        minLength: {
            value: 1,
            message: "Surname must have at least 1 characters",
        },
        maxLength: {
            value: 64,
            message: "Surname must have maximum 64 characters",
        },
    },
    email: {
        required: "Email is required",
        pattern: {
            value: /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
            message: "Not valid email. Must be examp@gmail.com",
        },
        minLength: {
            value: 5,
            message: "Email must have at least 5 characters",
        },
        maxLength: {
```

```
        value: 254,
        message: "Email must have maximum 254 characters",
    },
},
password: {
    required: "Password is required",
    validate: {
        message: validatePassword,
    },
    minLength: {
        value: 8,
        message: "Password must have at least 8 characters",
    },
    maxLength: {
        value: 64,
        message: "Password must have maximum 64 characters",
    },
},
birthDate: {
    required: "Birth date is required",
    validate: {
        message: validateDate,
    },
},
status: {
    maxLength: {
        value: 500,
        message: "Status must have maximum 500 characters",
    },
},
link: {
    maxLength: {
        value: 200,
        message: "Link must have maximum 200 characters",
    },
},
header: {
    required: "Header field is required",
    maxLength: {
        value: 100,
        message: "Header should have maximum 100 characters",
    },
    pattern: {
        value: /^[a-zA-Z0-9\s!"#$%&№'()*+,./:;<=>?@[\]^_`{|}~\\-]*$/,
        message: "Not a valid header",
    },
},
description: {
    required: "Description field is required",
    maxLength: {
        value: 2000,
```

```
            message: "Description should have maximum 2000 characters",
        },
        pattern: {
            value: /^[a-zA-Z0-9\s!"#$%&№'()*+,./:;<=>?@[\]^_`{|}~\\-]*$/,
            message: "Not a valid description",
        },
    },
    onlyLatin: {
        required: "This field is required",
        pattern: {
            value: /^[a-zA-Z0-9]+$/,
            message: "Must be written in Latin",
        },
    },
}

export {registerOptions}
```