МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія проєктування системи залучення інвесторів для просування стартапів»

Завідувач
випускаючої кафедри **А. С. Довбиш**

Керівник роботи **В. К. Ободяк**

Студентка групи ІН.м-02 **Н. Р. Титова**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет ЕлІТ Кафедра Комп'ютерних наук
Спеціальність «122 – Комп'ютерні науки»

Затверджую_____
Зав. кафедри Довбиш А. С.
«_____» _____20___ р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТЦІ

Титовій Надії Ростиславівні
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Інформаційна технологія проєктування системи залучення інвесторів для просування стартапів
затверджую наказом по університету від «____» _____ 20 ___ р.
№ _____

2. Термін здачі студентом закінченого проєкту (роботи) _____

3. Вхідні дані до проєкту (роботи)_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Огляд технологій розробки та існуючих рішень, що застосовуються для залучення інвесторів з метою просування стартапів. 2) Формування мети та постановки задачі, визначення етапів реалізації проєкту. 3) Вибір технологій та інструментів, що необхідні для розробки, огляд та аналіз баз даних. 4) Проєктування бази даних. 5) Розроблення інформаційного та програмного забезпечення системи. 6) Тестування готового програмного продукту та аналіз 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 _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Термін виконання проєкту (роботи)	Примітка
1.	Огляд технологій розробки та існуючих рішень	01.11–07.11	
2.	Формування мети та постановка задачі	08.11–18.11	
3.	Вибір технологій та інструментів для розробки	19.11–29.11	
4.	Проєктування та розробка системи	18.11–24.11	
5.	Створення бази даних	25.11–02.12	
6.	Тестування готового програмного продукту	03.12-17.12	
7.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи	08.12-12.12	

Студентка–дипломниця _____
(підпис)

Керівник проєкту _____
(підпис)

РЕФЕРАТ

Записка: 74 стор., 38 рис., 1 додаток, 24 джерела.

Об’єкт дослідження – процес проєктування та створення веб-систем для залучення інвесторів.

Мета роботи – розроблення та запровадження механізмів підвищення ефективності діяльності стартап-компаній.

Методи дослідження – під час розроблення веб-системи використані: база даних MongoDB, таблиці каскадних стилів CSS, мова програмування JavaScript, TypeScript, фреймворк Angular, NestJS, Bootstrap.

Результати – було проведено аналіз предметної області застосування та описана актуальність створення інформаційної технології проєктування системи; обґрунтований вибір середовища розробки; розроблений веб-додаток для залучення інвесторів з метою просування стартапів, що містить необхідну інформацію про інвесторів та бізнес; поєднано інформаційну систему з базою даних, щоб користувачі могли легко відфільтрувати та переглянути інформацію, яка є найбільш актуальною для їх бізнесу; після реалізації проєкту проведено тестування роботи веб-додатка.

СТАРТАП, ІНВЕСТУВАННЯ, ІННОВАЦІЙНІ ТЕХНОЛОГІЇ,
JAVASCRIPT, ANGULAR, TYPESCRIPT, NESTJS, BOOTSTRAP,
MONGODB, CSS

Зміст

ВСТУП

Ми живемо в епоху стартапів. На сьогоднішній день, кожен, хто має блискучу ідею, невелику команду ентузіастів та наполегливість, може стати наступним Стівом Джобсом та перетворитися в багатомільйонну компанію за лічені роки.

Але на шляху до успіху будь-який проєкт очікують труднощі. Окрім технічного характеру, засновники стикаються із такою основною проблемою як просування стартапу.

Зайвого часу на отримання результату немає, а виводити продукт на ринок, не маючи сформованого попиту – немає можливості. Тому у розробників продукту зазвичай виникає істотне обмеження бюджету.

Гроші породжують гроші. Тому потрібно пам’ятати, що коли збільшуються доходи – збільшуються і витрати. Тому стартапери нерідко покладаються на фінансову підтримку від інвесторів.

Тож, актуальним є проблема пошуку інвестора, котрий придбає ідею компанії-початківця на взаємовигідних умовах.

Метою кваліфікаційної магістерської роботи є розроблення інформаційної системи підтримки стартап-компаній при залученні інвесторів для просування стартапів. Для створення ефективної системи потрібно проаналізувати вітчизняний і міжнародний досвід успішного розвитку підприємств.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз предметної області застосування

Стартапи – це молоді компанії, засновані для того, щоб розробити унікальний продукт або послугу, вивести їх на ринок і зробити непереборними та незамінними для клієнтів.

Стартапи засновані на інноваціях, усуваючи недоліки існуючих продуктів або створюючи абсолютно нові категорії товарів і послуг, тим самим руйнуючи усталені способи мислення та ведення бізнесу для цілих галузей.

Звичайні компанії дублюють те, що було зроблено раніше. Потенційний власник ресторану може франшизувати існуючий ресторан. Тобто вони працюють на основі існуючого шаблону того, як має працювати бізнес. Стартап, на відміну, має на меті створити абсолютно новий шаблон.

Це також вказує на ще один ключовий фактор, який відрізняє стартапи від інших компаній: швидкість і зростання. Стартапи прагнуть дуже швидко розвивати ідеї. Вони часто роблять це за допомогою процесу, який називається ітерацією, під час якого вони постійно вдосконалюють продукти за допомогою відгуків і даних про використання. Часто стартап починається з базового скелета продукту, який називається мінімально життєздатним продуктом (MVP), який будуть тестувати та переглядати, поки проєкт не буде готовий вийти на ринок.

Стартапи зазвичай прагнуть швидко розширити свою клієнтську базу. Це допомагає їм закріплювати все більші частки ринку, що, у свою чергу, дозволяє їм залучати більше грошей, що дозволяє їм ще більше розвивати свої продукти та аудиторію.

Все це швидке зростання та інновації зазвичай, явно чи неявно, слугує для кінцевої мети: виходу на ринок. Коли компанія відкриває себе для державних інвестицій, це створює можливість для ранніх інвесторів отримати

готівку та пожинати свої винагороди, концепція на мові стартапів, відома як «вихід».

Не дивлячись на те, що не існує двох однакових стартап-проєктів, було узагальнено чинники успішності та провальності стартапів (рис. 1.1).

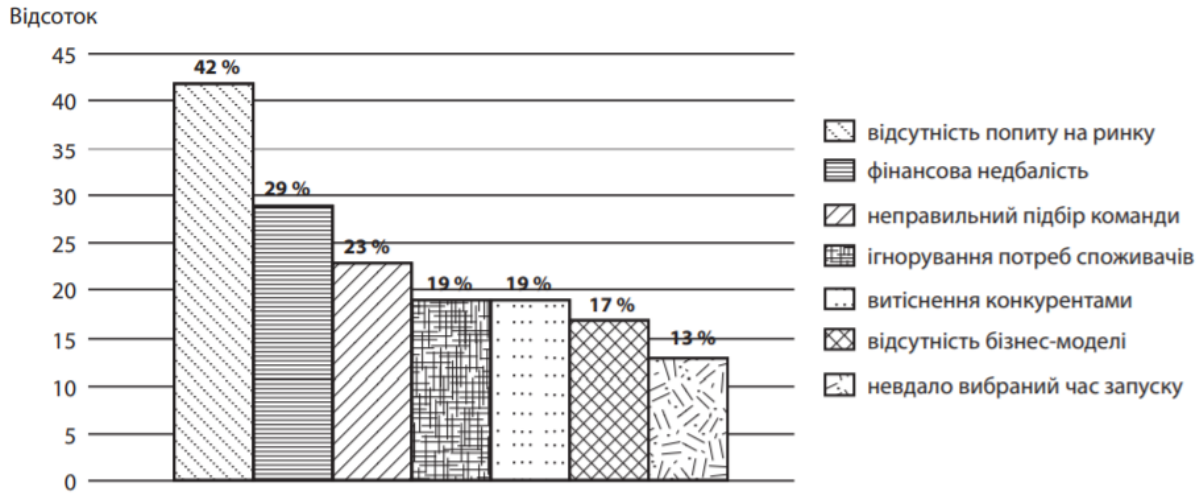

Рисунок 1.1 – Найпоширеніші причини провалу стартапів [1]

На успішність стартапів впливають різні внутрішні та зовнішні фактори, що першочергово мають бути врахованими перед виходом проєкту на ринок. Подальші кроки полягають в розробленні механізмів підвищення ефективності стартапів та їх запровадження в діяльність компаній. Важливої уваги потребує питання захисту результатів інтелектуальної власності.

Потужним каталізатором для успішного розвитку як економіки в цілому, так і кожного окремого підприємства є створення на основі аналізу вітчизняного і міжнародного досвіду зручного середовища для розбудови інноваційних технологій [1].

Існує п’ять поширених типів стартапів у різних галузях, кожен з яких має різні підходи до масштабування [2]:

1) Стартапи малого бізнесу.

2) Покупні стартапи.

3) Масштабовані стартапи.

4) Розгалужені стартапи.

5) Соціальні стартапи.

Розглянемо кожен тип стартапу докладно.

1. Стартапи малого бізнесу: незалежні компанії з невеликими командами.

У більшості стартапів є «велика» кінцева мета – бути викупленим чи отримати грошові вливання. Початківці малого бізнесу бувають різними. Від індивідуальних компаній і партнерства до невеликих команд – ці стартапи із задоволенням залишаються стартапами, продаючи свої продукти та послуги.

І хоча вони зацікавлені в зростанні, вони ростуть у власному темпі. Такі стартапи часто завантажуються або фінансуються самостійно, а це означає, що менше тиску, щоб якнайшвидше масштабувати або бути відповідальним до безпосередніх потреб інвесторів.

2. Покупні стартапи: бізнеси, створені для того, щоб їх викупили.

Концепція: невеликі команди будують бізнес з нуля і продають його більшому гравцеві у своїй галузі.

Викуп здається досить приємною угодою. Однак побудувати щось, що варто придбати за мільйони чи мільярди, легше сказати, ніж зробити. Спочатку потрібно усвідомити, що конкуренція дуже жорстка в будь-якій індустрії програмного забезпечення.

Треба мати на увазі, що стартапи необов’язково повинні бути прибутковими, щоб бути викуплені. Це являє собою значний ризик для інвесторів, але ще більший ризик для власників бізнесу, які намагаються продати компанію, яка витрачає гроші. Існує багато прикладів того, наскільки безладним може бути цей процес.

Тим не менш, існує багато незалежних розробників додатків і невеликих команд, які витрачають кілька років на бізнес (або навіть на побічну роботу), який продається більшій компанії.

3. Масштабовані стартапи: компанії, які шукають капітал (або самі масштабуються).

Спільним між всіма типами стартапів є потреба в масштабуванні.

Це відповідає дійсності, незалежно від того, наскільки великий чи маленький стартап, чи є бізнесом із десятками співробітників, чи дуетом, який працює в гаражі батьків.

Але деякі стартапи легше масштабувати, ніж інші. Більшість споживчих і бізнес-додатків є прикладами масштабованих стартапів: як тільки вони дадуть про себе знати і напрацюють базу користувачів, стає легше залучати нових клієнтів. Це свого роду ефект сніжної кулі.

Масштабовані стартапи роблять це шляхом залучення капіталу від зовнішніх інвесторів (уявіть: інвесторів-ангелів, венчурних капіталістів, ділових партнерів, друзів, родини). Маючи новознайдені гроші, вони можуть підтримувати ініціативи зростання, щоб отримати більше клієнтів і врешті-решт привернути увагу людей, які готові їх викупити.

4. Розгалужені стартапи: компанії, які відходять від більших корпорацій.

Не всі види стартапів створюються з нуля. Відгалуження стартапу цілком зрозуміло. Простіше кажучи, це стартапи, які відділяються від більших материнських компаній, щоб стати власними організаціями.

Наприклад, відгалуження бізнесу може бути створено для того, щоб більша компанія вийшла на новий ринок або порушила роботу меншого конкурента. Оскільки ці стартапи діють незалежно від своїх материнських компаній, вони мають свободу вести бізнес і експериментувати, не привертаючи до себе особливої уваги чи контролю.

Як підкреслює Investopedia [3], така компанія, як Sidewalk Labs (відгалуження материнської компанії Google Alphabet), є хорошим прикладом такої дочірньої компанії.

5. Соціальні стартапи: некомерційні та благодійні компанії.

Про стартапи іноді вважають, що вони одержимі зростанням і шкодують грошей. Тим не менш, деякі стартапи спеціально створені, щоб робити добро. Соціальні стартапи, до яких входять благодійні та некомерційні організації,

масштабуються заради філантропії. Вони працюють так само, як і будь-який інший стартап, але роблять це за допомогою грантів і донорів. Яскравим прикладом соціального стартапу є Code.org, організація, якій вдалося зібрати майже 60 мільйонів доларів (з таких компаній, як Google і Facebook), щоб допомогти студентам надати можливості в галузі інформатики.

Можна виділити найпоширеніші типи стартапів за галузями [4]. Хоча більшість із них насправді пов’язані з технологіями, безумовно, є можливості для запуску і в більш «нетрадиційних» сферах:

- Програмне забезпечення (SaaS) і технології.
- Маркетинг і реклама.
- Охорона здоров’я.
- IT.
- Страхування.
- Освіта.
- Нерухомість.
- Навколишнє середовище та енергетика.
- Роздрібна торгівля та електронна комерція.
- Блокчейн і криптовалюти.

1.2 Дослідження актуальності проблеми

Стартапи відіграють роль потужної рушійної сили в розвитку бізнесу. Це пояснюється тим, що якраз діяльність стартапів спонукає появу нових напрямів у виробництві, наукових та прикладних дослідженнях.

Актуальність досліджуваної проблеми полягає в тому, що завдяки розвитку інформаційних-комунікаційних технологій стартапи набувають більшого поширення у всьому світі, але при цьому необхідно зауважити, що на кожен успішний проєкт припадають десятки провальних, тобто навіть геніальна ідея не завжди приносить мільйони доларів від інвесторів. Започаткування стартапу в умовах сьогодення стає дедалі складнішим завданням. Для збереження та зміцнення своїх конкурентних позицій на ринку

значна частина підприємців прагне вирватися вперед у гострій боротьбі за венчурний капітал намагаючись запустити власний стартап, але далеко не завжди це завершується успіхом. Тому актуальним було і є вирішення проблеми чіткої побудови проєкту та його подальшого успіху. Пошук інвесторів є нагальною проблемою і в Україні, і в інших державах.

У сучасному світі тільки максимальне використання новаторських технологій та бізнес-рішень дозволяє бути конкурентоспроможним. Отже, стартапи стають основою нових думок і проривних рішень у великому бізнесі. Число вітчизняних стартапів збільшується з кожним роком, але, згідно з оцінкою фахівців, з усіх поданих проєктів інтерес представляють у середньому 3–5%, але на результат виходять не більше 1–2% [5]. Головною проблемою стартапів у нашій країні є не стільки нестача обдарованих фахівців, скільки відсутність матеріальної допомоги процесу формування новітніх товарів і послуг. Відповідно до статистики, небагато проєктів досягають етапу перемовин з капіталовкладниками, більшість – відхиляються ще на початку.

Часто особи, відповідальні за проєкти, не можуть стратегічно грамотно привести в дію та подати стартап. Основним чинником цього є непроінформованість в усіх юридичних аспектах, недосконаліть законодавчої бази, порушення певних умов відповідно до розподілу капіталів, отриманих від інвесторів. Як наслідок, проєкт припиняє свій розвиток або взагалі існування. Наведені проблеми характерні для багатьох країн, але стартапи продовжують розвиватись. Доказом розвитку ринку стартапів є дослідження сервісу Startup Ranking, який проаналізував 136 країн в світу, нарахував близько 100 тис. стартапів і визначив рейтинг країн-лідерів за кількістю впроваджених стартапів. Перше місце, згідно зі Startup Ranking, належить США (47 887 стартапів), а Україна посіла 43 місце (262 стартапи). Дані щодо країн і кількості стартапів наведені на рис. 1.2.

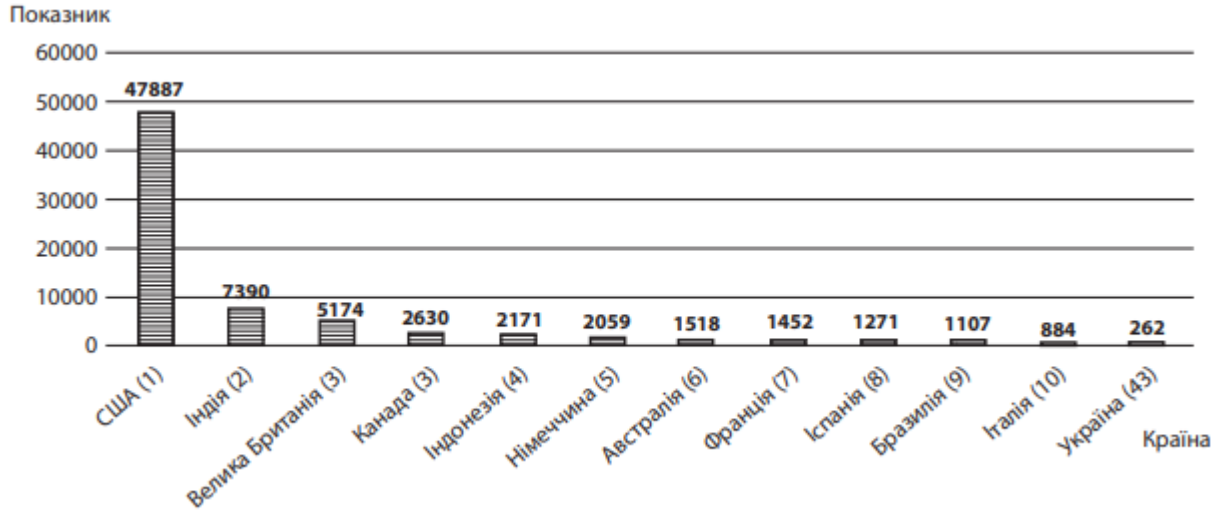

Рисунок 1.2 – Рейтинг країн за кількістю стартапів [1]

Необхідно підтримувати результативне функціонування стартапів в нашій країні, формувати механізми для зростання ефективності їх роботи в українському бізнес-середовищі, і впроваджувати популяризацію новітніх технологій для успішної розбудови вітчизняних компаній. Основною перешкодою розвитку стартапів в нашій країні є дефіцит клієнтів, які бажають придбати готовий продукт. З цієї причини стартапи створюються в нашій країні орієнтуючись на іноземних клієнтів, наприклад, з ринків Азії, Європи, Сполучених Штатів Америки тощо. Це дає можливість простіше зацікавити інвесторів і розвинути свою ідею аж до масового продукту, послуги або першого публічного розміщення приватною компанією своїх акції на фондовому ринку. По цій причині якщо йдеться мова про рівень університету або області, то не варто витрачати ресурси на змагання і різну підготовку з розвитку стартапу. Потрібно розвивати підприємливість. У вишах можлива модифікація навчальних програм, формування міжпредметних проєктних груп зі студентів різних спеціальностей – технічних, гуманітарних. На менторів, закріплених за ними, покладатимуться задачі створенню умов для розробки групою бізнес-ідей.

Із метою формування стартапів в нашій країні потрібна ґрунтовна підтримка держави малого бізнесу в сфері інновацій і у сфері венчурного інвестування відповідно до наступних позицій [6]:

1) створення системи гарантій та страхування інвестицій;

2) ведення державного реєстру структур, які працюють у сфері венчурного інвестування;

3) організація системи підготовки кадрів менеджерів інновацій;

4) забезпечення гарантій прав компаній на інтелектуальну власність;

5) розробка ефективного механізму порядку утворення та використання коштів венчурного фонду;

6) розроблення нормативно-правової бази, яка б регламентувала правові відносини у сфері інвестування в інновації;

7) розробка процедури допуску на український ринок іноземного капіталу;

8) розроблення методології оцінки ринкових перспектив комерціалізації науковотехнічної продукції в межах реалізації стартапів.

Українські науковці мають перспективні розробки, але:

– відсутні платформи та системні програми підтримки наукоємних стартап проєктів;

– немає системи ефективного витрачання коштів на фундаментальну науку;

– відсутні навички ведення переговорів із бізнесом та розуміння потреб ринку;

– немає створеної дієвої бізнес моделі взаємодії між науковцями – науковими інституціями – бізнесом – владою;

– відсутня система відбору креативних (підприємливих) студентів, молодих науковців.

Підтримка стартапів є комплексним процесом, реалізація якого включає:

– збільшення інвестування з боку держави, та як наслідок, забезпечення розвитку галузі на території країни та сплата податків, які дадуть можливість через певний час існування проєкту повернути інвестовані гроші;

– підтримку соціальної спрямованості інноваційних проєктів;

– розвиток підприємницької культури;

– посилення режиму інтелектуальної власності;

–розробку державними органами законопроєктів, які сприятимуть розвитку інноваційного підприємництва;

– посилення інформаційних можливостей участі українських новаторів в міжнародних програмах.

1.3 Моделі фінансування стартапів

Запорука успіху стартапу знаходиться у прямій залежності від поєднання таких складових: ідеї, співробітників і інвесторів. Безумовно, найважливішою умовою вважається процедура залучення інвестицій. Фінансування бізнес-проєктів є одним з головних питань з метою осмислення бізнесу і розвитку фінансової грамотності.

У випадку коли розробник бізнес-ідеї не має своїх коштів, він шукає зовнішні джерела фінансування.

Подальший розвиток проєкту залежить від доцільності вибору тієї чи іншої моделі фінансування. Важливе рішення вибору моделі приймається з урахуванням усіх факторів – можливостей, ресурсів, інвесторів та безпосередньо самої ідеї.

Існує безліч інвестиційних можливостей, які можна розглянути, намагаючись отримати фінансування свого стартапу. Залежно від того, на якому етапі розвитку знаходиться наш бізнес, деякі варіанти фінансування можуть мати більше сенсу, ніж інші (рис. 1.3).

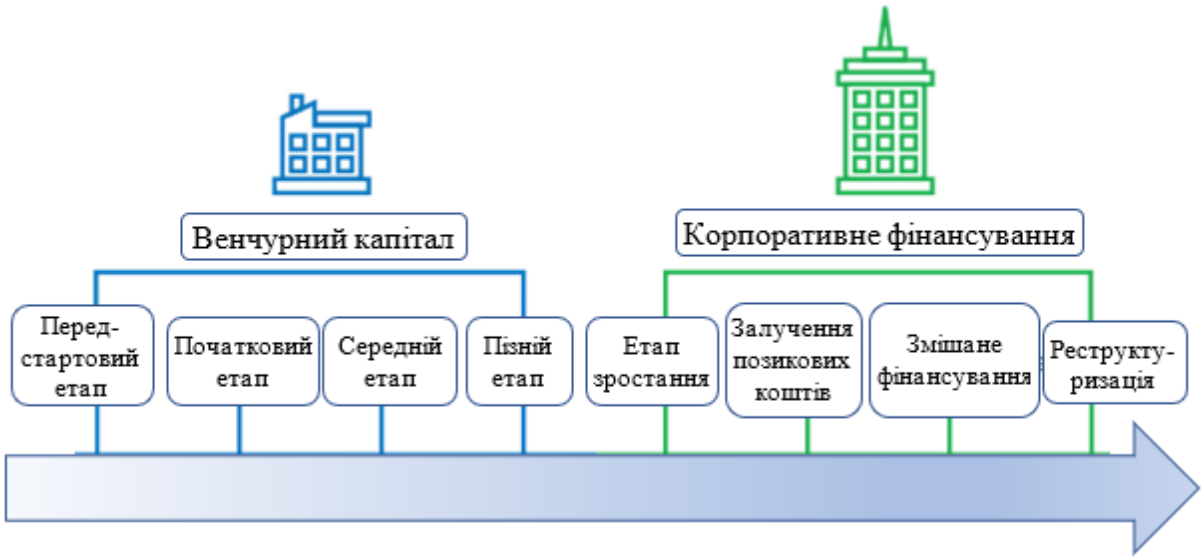

Рисунок 1.3 – Типи фінансування на різних етапах розвитку компанії (адаптовано з [6])

Є різні методи залучення інвестицій до фінансування бізнес-проєктів. Джерела фінансування стартапів класифікують таким чином [7]:

1) Особисті заощадження стартаперів.

Застосовуються на стадіях виникнення і розвитку, якщо концепція продукту, обслуговування, технологічні процеси, і також бізнес-план самого стартапу перебувають на етапі зародження, і на поточному етапі інвесторам пропонувати ще просто нема чого.

2) Друзі та родина.

На перших стадіях формування нашого стартапу відбір належного виду фінансування може бути стресом. У деяких випадках найкращим виходом є фінансування від друзів чи членів сім'ї. Запозичення у них часто означає найнижчу відсоткову ставку на відміну від банків. Це варіант здатний допомогти зменшити рівень заборгованості, щоб ми мали змогу інвестувати більше грошей у своє підприємництво. Але треба пам'ятати, що через позик у друзів і членів сім'ї можуть виникнути інциденти, оскільки часто виникають проблеми у розподілі товариських, домашніх і ділових взаємин.

3) Кредитування.

Кредитні напрями – це найбільш простий спосіб отримання фінансування для перших кроків у розвитку малого бізнесу, звісно якщо вони мають високонадійний ресурс доходу, хороший кредитний показник і переконливо позитивну кредитну хроніку.

Але стартап – підприємство зі серйозними ризиками, які часто складно прорахувати, крім цього, важко визначити рівень прибутковості проєкту.

Слідуючи цього фактору більш логічно брати позику на розвиток бізнесу на останніх стадіях, якщо ризики зменшились, а рентабельність зросла.

4) Венчурні фонди.

Це фонди приватного капіталу, які інвестують на ранніх стадіях свого розвитку.

Венчурні фонди – це приватні гроші, що фінансуються у підприємства, які перебувають на стадії свого формування. Вони свідомо ризикують, коли вкладають в підприємницьку ідею відносно невеликі кошти з метою заробити значні відсотки.

Група, яка вирішує підтримати стартап, надає йому гроші в обмін на частку у капіталі компанії. Члени комітету зазвичай шукають стартапи, готові до запуску своїх продуктів, що пройшли стадію ідеї, тому необхідно показати інвесторам, чим ви кращі за інші подібні компанії.

5) Бізнес-ангели.

Бізнес-ангел – це приватний інвестор, який фінансує стартап на ранній стадії замість того отримуючи долю в подальшому бізнесі. Це підвищує шанси стартаперів на успіх, але спричиняє втрату частини контролю над своєю компанією. Це обумовлено тим, що інвестори-ангели хочуть допомогти у ухваленні бізнес-рішень. Вони також отримають частину прибутку у разі продажу підприємцем свого бізнесу.

Одним із найкращих способів пошуку інвесторів-ангелів може бути нетворкінг. Також можна скористатися онлайн-сервісами, наприклад LinkedIn для їх пошуку.

6) Бізнес-інкубатори.

Для тих, хто хотів би реалізувати бізнес-ідею, але не має підприємницького досвіду, найбільш прийнятним є бізнес-інкубатори. Початківців інкубатори забезпечують приміщеннями, обладнанням, засобами комунікації тощо, організовують навчання, залучають менторів та консультантів. Тобто, бізнес-інкубатори допомагають проаналізувати ринок збуту, спланувати діяльність та довести проєкт до того рівня, коли ним зможуть зацікавитись майбутні інвестори.

Умови, на яких стартапери співпрацюють з бізнес-інкубаторами, розрізняються. Учаснику інкубатор свої послуги може надати безкоштовно за умови подальшого отримання долі (до 25%) при успішній реалізації проєкту. Інший підхід полягає в наданні інкубатором клієнту платних послуг. У такому разі бізнес-інкубатор не претендує на спільне володіння компанією.

За статистикою, близько 70% проєктів, що отримали такі попередні інвестиції, банкротують або реорганізуються в іншу структуру і з іншим керівником. Наступного етапу фінансування досягають 20-25% проєктів, з яких тільки 5% стартапів роблять дійсно потужний ривок [8].

Наша країна має успішний практичний досвід функціонування бізнес-інкубаторів, найпотужніші з яких забезпечують сферу інформаційних технологій.

7. Краудфандинг.

Організується в мережі інтернет-майданчик, на якому реєструються члени – суспільство, які прагнуть приєднатися до здійснення інновацій, але яким не достатньо індивідуальних фінансів для інвестування в проєкти як бізнес-ангелів або венчурного капіталіста. Стартапи, що притягають інвестування, утворюють рекламне промо-відео, у якому висвітлюється задумка проєкту, його мета і необхідна сума вкладень. Визначається необхідна сума коштів та час, що обмежений для збирання коштів, та у якості вдячності за вкладення в план, стартапери пропонують різні приємності, якимось чином

пов’язані з проєктом. Безумовно, частіше подібним способом мають намір збору коштів для формування музичних альбомів, кінозйомку, видання, громадські чи благодійні фонди.

Завдяки краудфандингу у молодих компаній з’являється можливість залучити гроші з мінімальними для себе витратами. Звичайно, у кожної платформи є свої системи оплати послуг, але в середньому це від 3% до 9% від залученої суми.

Крім цього, краудфандингові платформи допомагають отримати відповідь у питання – чи справді майбутній результат потрібний користувачам і це означатиме, чи є у нього ринок і блискуче майбутнє.

8. Грантові організації.

Початкову фінансову допомогу в досить скромних умовах, порівняно з іншими інвесторами, стартапам можуть виявляти також грантові компанії. Сутність роботи грантових установ полягає у наданні економічної допомоги молодим творчим людям із прогресивними ідеями. Якщо фінансованих план стає ефективним, підприємство отримує «хороше ім’я» і безкоштовний PR.

9. Держава.

Найчастіше, замислюючись про пошук ключа фінансування свого проекту, підприємець в останню чергу згадує про державу і про ту підтримку, яку вона здатна надати у формуванні його бізнесу.

Звичайно, для цього є причини: оскільки наш уряд, на жаль, не дуже шанує маленьких підприємців і стартаперів своїм інтересом, а також для того щоб отримати від нього матеріальну підтримку, потрібно застосувати досить старань.

Але безпосередньо з метою цього створюється колектив, в якій кожен захоплюється своїм заняттям: одні люди розробляють продукт, хтось захоплюється рекламою і вивченням торгу, але хтось може займатися оформленням юридичної сторони процесу і завданнями пошуку економічної допомоги.

По-перше, кожен уродженець України має можливість використати безкоштовну дотацію з метою формування і розвитку свого бізнесу. Звичайно, що з метою отримання дотації потрібно забезпечити безліч документів, але вже після отримання коштів - відомостей відповідно до їх застосування, але загалом це цілком реально.

По-друге, кожен регіон країни розробляє свої проєкти підтримки маленькому і середньому бізнесу, в тому числі і безпосередньо стартапам, особливо якщо спрямованість проєкту відповідає потребам місцевих жителів.

1.4 Постановка задачі

Мета кваліфікаційної магістерської роботи полягає в розробленні інформаційної системи залучення інвесторів для просування стартапів.

Головним призначенням веб-додатка буде вирішення проблеми пошуку інвестора, котрий придбає ідею компанії-початківця на взаємовигідних умовах.

Для досягнення поставленої мети було проведено аналіз предметної області застосування та описана актуальність задачі зі даного веб-додатка.

Визначено наявність проблем, сформульована мета та поставлено такі задачі:

- обґрунтувати вибір середовища розробки;
- визначити порядок розроблення;
- створити концептуальну модель бази даних;
- розробити веб-додаток для залучення інвесторів з метою просування стартапів, що буде містити цінну інформацію про інвесторів та бізнес, поєднуючи її з базою даних, щоб користувачі могли легко відфільтрувати та переглянути інформацію, яка є найбільш актуальною для їх бізнесу;
- після реалізації проєкту провести тестування роботи веб-додатка.

Практична значимість роботи полягає у підвищенні ефективності діяльності стартап-компаній та запровадження необхідних механізмів.

2 МЕТОДИ РОЗРОБЛЕННЯ ВЕБ-ДОДАТКА

2.1 Середовище розробки

Всесвітня комп’ютерна мережа World Wide Web – найпопулярніша служба мережі Інтернет, яка надає користувачам доступ до широкого набору документів, які пов’язані один з одним за допомогою гіперпосилань, електронних з’єднань, що пов’язують частини інформації, щоб користувачі могли легко отримати до них доступ. Саме гіпертекст дає можливість, щоб користувач мав змогу з тексту обрати слово або фразу, завдяки чому для користувача відкривається доступ до інших документів, що містять інформацію щодо цього слова чи фрази. Гіпермедійні документи містять посилання на зображення, звуки, анімацію та фільми. Web працює в основному форматі клієнт-сервер Інтернету; сервери – це комп'ютерні програми, які зберігають і передають документи на інші комп'ютери в мережі, коли їх запитують, тоді як клієнти – це програми, які запитують документи з сервера, коли їх запитує користувач. Програмне забезпечення браузера дозволяє користувачам переглядати отримані документи [9].

Гіпертекстовий документ з відповідним текстом і гіперпосиланнями написаний мовою розмітки гіпертекста (HTML) і йому призначається онлайн-адреса, яка називається Uniform Resource Locator (URL).

HTML є скороченням від Hypertext Markup Language – використовується для створення електронних документів (так звані сторінки), які відображаються у всесвітній мережі. Кожна сторінка містить серію підключень до інших сторінок, які називаються гіперпосиланнями. Кожна веб-сторінка, яку ми бачимо в мережі Інтернет була написана з використанням однієї версії HTML.

У прикладі тегу HTML на рис. 2.1 можна побачити, що компонентів не так багато. Більшість тегів HTML мають початковий тег, що містить назву тегу, атрибути тегу, закриваючий тег, що містить пряму косу риску, і назву

тегу, що закривається. Для тегів, які не мають закривального тегу, наприклад <img>, найкраще закінчувати тег косою рискою.

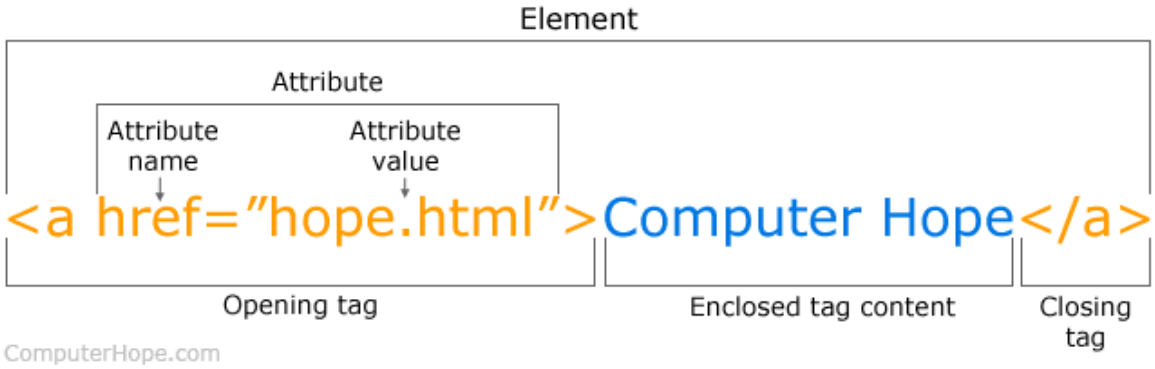

Рисунок 2.1 – Розбивка тегу HTML [10]

Більшість тегів містяться в кутових дужках менше і більше, і все між відкритим і закритим тегом відображається або впливає на тег. У наведеному вище прикладі тег <a> створює посилання під назвою «Computer Hope», яке вказує на файл hope.html.

Структура статичної HTML – сторінки:

```
<!doctype HTML>
<html>
<head>
<title>Мій перший HTML-документ</title>
</head>
<body>
Hello world!
</body>
</html></source>
```

HTML-код забезпечує правильне форматування тексту та зображень для вашого Інтернет-браузера. Без HTML браузер не знав би, як відображати текст, різні елементи або завантажувати зображення. HTML також забезпечує базову структуру сторінки, на яку накладаються каскадні таблиці стилів, щоб змінити її зовнішній вигляд. Можна було б думати про HTML як про кістки (структуру) веб-сторінки, а CSS як про її шкіру (зовнішній вигляд).

Каскадні таблиці стилів, які скорочено називають CSS – це проста мова дизайну, призначена для спрощення процесу створення презентаційних веб-сторінок.

CSS обробляє зовнішній вигляд та функціональність веб-сторінки. Використовуючи CSS, ви можете керувати кольором тексту, стилем шрифтів, інтервалом між абзацами, розміром та розташуванням колонок, використанням фонових зображень або кольорів, дизайном макета, варіаціями відображення для різних пристроїв та розмірів екрану, а також безліччю інших ефектів.

CSS легко вивчити і зрозуміти, але забезпечує потужний контроль над поданням HTML-документа. Найчастіше CSS використовується у поєднанні з мовами розмітки HTML або XHTML.

Основними перевагами CSS є такі [11]:

1) CSS заощаджує час – ви можете написати CSS один раз і потім повторно використовувати той самий аркуш на декількох HTML-сторінках. Ви можете визначити стиль для кожного елемента HTML та застосувати його до будь-якої кількості веб-сторінок.
2) Сторінки завантажуються швидше – якщо ви використовуєте CSS, вам не потрібно щоразу написати атрибути тегів HTML. Достатньо написати одне правило CSS для тега і застосувати його до всіх введень цього тегу. Таким чином, менше коду означає швидший час завантаження.
3) Простота обслуговування – для внесення глобальних змін достатньо змінити стиль і всі елементи на всіх веб-сторінках будуть оновлені автоматично.
4) Перевага стилів над HTML – CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати HTML-сторінці набагато кращий вид у порівнянні з HTML-атрибутами.

5) Сумісність із кількома пристроями – таблиці стилів дозволяють оптимізувати вміст для кількох типів пристроїв. Використовуючи один і той же документ HTML, можна надати різні версії веб-сайта для портативних пристроїв, таких як КПК та стільникові телефони, або для друку.
6) Глобальні веб-стандарти – зараз атрибути HTML старіють, і рекомендується використовувати CSS. Тому гарною ідеєю почне використовувати CSS на всіх HTML-сторінках, щоб зробити їх сумісними з майбутніми браузерами.

Із розвитком технологій функціональність веб-сторінок постійно зростає й наближається до функціональності настільних прикладних програм. Ця зростаюча функціональність реалізується за допомогою JavaScript.

JavaScript – це мова сценаріїв, що використовується для створення та керування динамічним вмістом веб-сайта, тобто. всім тим, що рухається, оновлюється або інакше змінюється на екрані, не вимагаючи ручного перезавантаження веб-сторінки. Є такі функції [12]:

- анімована графіка;
- слайд-шоу з фотографій;
- автозаповнення текстових пропозицій;
- інтерактивні форми.

Насправді додавання коду JavaScript на веб-сторінку – досить простий процес (і знайомий, якщо ви займалися кодуванням в HTML і CSS). JavaScript можна додати безпосередньо в код сторінки, використовуючи теги <script> і задаючи атрибут типу text/javascript. JavaScript виглядає дуже схоже на додавання CSS на сайт. На рис. 2.2 наведений приклад для порівняння.

Код JavaScript також може бути доданий на сторінку у вигляді окремого файлу заголовка з розширенням .js (зазвичай це робиться, якщо йдеться про код, який необхідно включити відразу на кілька сторінок). Потім сценарій

завантажується та обробляється у веб-браузері кожного користувача, перетворюючись на динамічні об'єкти та ефекти, які він бачить на екрані.

```
CSS:
<style>
CSS goes here
</style>
```

```
JavaScript:
<script type="text/javascript">
JavaScript code goes here
</script>
```

Рисунок 2.2 – Приклад додавання коду JavaScript

JavaScript – це «мова сценаріїв». Мови сценаріїв – це мови кодування, що використовуються для автоматизації процесів, які інакше користувачам довелося б виконувати самостійно, крок за кроком. Без скриптів будь-які зміни на відвідуваних веб-сторінках вимагають або ручного перезавантаження сторінки, або переходу по ряду статичних меню, щоб дістатися до потрібного вам контенту.

Мова сценаріїв, такий як JavaScript (JS, для тих, хто в курсі), виконує важку роботу, вказуючи комп'ютерні програми, такі як веб-сайти або веб-додатки, "зробити щось". У випадку JavaScript це означає, що динамічні функції, описані раніше, повинні робити все, що вони роблять, наприклад, змушувати зображення анімувати себе, фотографії – циклічно показувати слайд-шоу, а пропозиції автозаповнення – відповідати на підказки. Саме «сценарій» JavaScript змушує ці речі відбуватися, здавалося б, самі по собі.

Тим часом, оскільки JavaScript є такою невід'ємною частиною веб-функціональності, всі основні веб-браузери оснащені вбудованими механізмами, які можуть відтворювати JavaScript. Це означає, що команди JS можна вводити безпосередньо в HTML-документі, і веб-браузери зможуть їх зрозуміти. Іншими словами, використання JavaScript не вимагає завантаження додаткових програм або компіляторів [13].

У зв’язку з тим, що JavaScript має такий недолік як динамічна типізація, була створена мова програмування TypeScript, що компілюється в JavaScript і розроблена спеціально для великих додатків.

TypeScript – це строго типізована, об'єктно-орієнтована, компільована мова. Вона була розроблена Андерсом Хейлсбергом (розробником C#) у компанії Microsoft. TypeScript – це і мова, і набір інструментів (рис. 2.3).

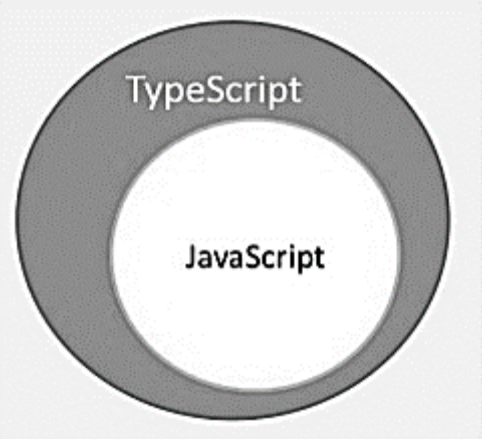

Рисунок 2.3 – Уявлення про TypeScript [14]

TypeScript починається і закінчується JavaScript. Аби використовувати TypeScript потрібно лише мати знання з мови програмування JavaScript. Весь код TypeScript перетворюється на його еквівалент JavaScript з метою виконання.

TypeScript підтримує інші бібліотеки JS. Скомпільований TypeScript можна використовувати з будь-якого коду JavaScript. JavaScript, згенерований TypeScript, може повторно використовувати всі існуючі рамки, інструменти та бібліотеки JavaScript. Будь-який дійсний файл .js можна перейменувати на .ts і скомпілювати з іншими файлами TypeScript.

TypeScript є портативним тому є можливість переносу між браузерами, пристроями та операційними системами. Він може працювати в будь-якому середовищі, в якому працює JavaScript. На відміну від своїх аналогів, TypeScript не потребує спеціальної віртуальної машини або конкретного середовища виконання для виконання.

TypeScript перевершує інші свої аналоги, такі як мови програмування CoffeeScript і Dart, оскільки TypeScript є розширеним JavaScript. На противагу цьому, такі мови, як Dart, CoffeeScript, самі по собі є новими мовами і вимагають специфічного для мови середовища виконання.

Переваги TypeScript включають:

1) Компіляція – JavaScript є інтерпретованою мовою. Тому його потрібно запустити, щоб перевірити його дійсність. Це означає, що ви пишете всі коди лише для того, щоб не знайти виведення, на випадок помилки. Отже, вам доведеться годинами намагатися знайти помилки в коді. Transpiler TypeScript забезпечує функцію перевірки помилок. TypeScript компілює код і генерує помилки компіляції, якщо виявить якісь синтаксичні помилки. Це допомагає виділити помилки перед запуском сценарію.

2) Сильна статична типізація – JavaScript не сильно типізований. TypeScript поставляється з додатковою системою статичного введення і виведення типу через TLS (TypeScript Language Service). Тип змінної, оголошеної без типу, може бути визначений TLS на основі її значення.

3) TypeScript підтримує визначення типів для існуючих бібліотек JavaScript. Файл визначення TypeScript (з розширенням .d.ts) надає визначення для зовнішніх бібліотек JavaScript. Отже, код TypeScript може містити ці бібліотеки.

4) TypeScript підтримує такі концепції об’єктно-орієнтованого програмування, як класи, інтерфейси, успадкування тощо.

Розробники front-end можуть просунутися на крок далі, використовуючи інструменти, які називаються фреймворками JavaScript.

JS-фреймворки надають розробникам JavaScript повні шаблони для веб-сайтів або додатків. Потім JS-фреймворки створюють у цих шаблонах місця, куди рекомендується помістити JS-код, а також попередньо написаний код, який можна вставити в ці місця.

Для виконання дипломного проєкту було обрано фреймворк Angular JS, котрий повністю заснований на HTML і JavaScript, тому немає необхідності вивчати інший синтаксис або мову.

AngularJS змінює статичний HTML в динамічний HTML. Він розширює можливості HTML, додаючи вбудовані атрибути та компоненти, а також надає можливість створювати атрибути користувача за допомогою простого JavaScript [15].

Переваги AngularJS [16]:

- MVC-фреймворк JavaScript із відкритим вихідним кодом.
- Підтримується Google.
- Немає необхідності вивчати іншу скриптову мову. Це просто чистий JavaScript та HTML.
- Підтримує поділ проблем завдяки використанню патерну проектування MVC.
- Вбудовані атрибути (директиви) роблять HTML динамічним.
- Легко розширюється та налаштовується.
- Підтримує односторінкові програми.
- Використовує ін'єкцію залежностей.
- Легко піддається модульному тестуванню.
- Дружній REST.

AngularJS слідує архітектурі MVC, діаграмі фреймворку MVC, як показано на рис. 2.4.

Контролер (Controller) представляє собою шар, що містить бізнес-логіку. Здійснення користувача запускають функції, які зберігаються всередині вашого контролера. Користувачі події є частиною контролера.

Представлення (View) використовуються для представлення презентаційного кінцевого шару, який надається користувачам.

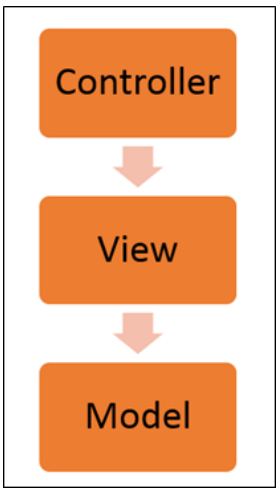

Рисунок 2.4 – Схема архітектури AngularJS [17]

Моделі (Model) використовуються для представлення даних. Дані у вашій моделі можуть бути простими і складатися з примітивних декларацій. Наприклад, якщо ви підтримуєте додаток для студентів, ваша модель даних може містити лише ідентифікатор студента та його ім’я. Або ж вона може бути складною, що має структуровану модель даних. Якщо ви підтримуєте додаток для власників автомобілів, ви можете мати структуру для визначення самого автомобіля з точки зору його потужності двигуна, кількості місць і т.д.

Для зосередження на роботі з розробки, не турбуючись про дизайн, і швидко створити гарний веб-сайт веб-розробники можуть використовувати фреймворк Bootstrap.

Bootstrap – це потужний набір інструментів — набір інструментів HTML, CSS та JavaScript для створення та побудови веб-сторінок та веб-додатків. Це безкоштовний проект з відкритим вихідним кодом, розміщений на GitHub і спочатку створений (і для) Twitter.

Веб-дизайнери та веб-розробники люблять Bootstrap за його гнучкість та простоту в роботі. Його головними перевагами є чуйність дизайну, широка сумісність з браузерами, послідовний дизайн за рахунок використання компонентів, що повторно використовуються, а також простота у використанні і швидкість освоєння. За рахунок JavaScript він досягає розширюваності, має вбудовану підтримку плагінів jQuery та програмний

JavaScript API. Використовують Bootstrap з довільним IDE або редактором, а також з будь-якою серверною технологією та мовою, від ASP.NET до PHP та Ruby on Rails. Bootstrap – це найпопулярніший фреймворк HTML, CSS та JavaScript для розробки чуйного та зручного для мобільних пристроїв веб-сайта.

Нижче наведені переваги Bootstrap [18].

- Він абсолютно безкоштовний для завантаження та використання.
- Це front-end фреймворк, який використовується для більш простої та швидкої розробки веб-сайтів.
- Він включає шаблони дизайну на основі HTML і CSS для типографіки, форм, кнопок, таблиць, навігації, модулів, каруселей зображень і багато іншого.
- Він також може використовувати плагін JavaScript.
- Він полегшує створення чуйних дизайнів.

NestJS – це фреймворк для створення ефективних та масштабованих серверних програм Node.js, побудованих з використанням та повною підтримкою TypeScript. Він використовує надійні фреймворки HTTP-серверів, такі як Express або Fastify. Nest забезпечує рівень абстракції над звичайними фреймворками Node.js та розкриває їх API розробнику. Це дає більшу свободу використання сторонніх модулів.

Гарною причиною вибрати NestJS замість ExpressJS (одного з найпопулярніших фреймворків Node.js) є той факт, що при запуску нового проекту на Node.js він являється чіткою архітектурою, заснованою на декількох простих компонентах (контролери, модулі та провайдери). Це дозволяє легко розділяти програми на мікросервіси.

Як уже згадувалося, NestJS – це розширений, універсальний, прогресивний Node.js фреймворк з відкритим вихідним кодом для створення переконливих та вимогливих бекенд-систем. У даний час це найшвидший фреймворк Node.js на TypeScript.

NestJS використовується для написання масштабованих, тестованих та слабозв'язаних додатків. Він виводить масштабовані сервери Node.js на новий рівень. Він підтримує такі бази даних, як PostgreSQL, MongoDB, MySQL. На NestJS сильно вплинули Angular, React та Vue, і він пропонує ін'єкцію залежностей прямо з коробки.

На січень 2020 року у нього понад 23 тисячі зірок на GitHub, а щотижневе завантаження npm становить майже 180 тисяч. Він заохочує розробників пробувати, вивчати та використовувати деякі відомі парадигми розробки програмного забезпечення, а його документація містить безліч прикладів, рецептів та джерел коду.

NestJS легко розширюється, тому що може використовуватися з іншими бібліотеками; універсальний завдяки своїй адаптивній повнофункціональній екосистемі та прогресивний, вносячи можливості JavaScript та патерни проектування.

Побудова блоків NestJS [19]:

1. Модулі: використовуються для організації коду та поділу функцій на логічні одиниці багаторазового використання. Згруповані файли TypeScript прикрашаються декоратором «@Module», який надає метадані, що використовуються Nest для організації структури програми.
2. Провайдери: також звані сервісами, які призначені для абстрагування будь-якої форми складності та логіки. Провайдери можуть бути створені та впроваджені в контролери або інші провайдери.
3. Контролери: відповідають за обробку вхідних запитів та повертають відповідні відповіді на клієнтську сторону програми (наприклад, виклик API).

Для обміну даними між сервером та клієнтською частиною веб-додатка у магістерській дипломній роботі використовується ApiRest.

REST є одним із найпопулярніших підходів до побудови архітектури API з передачею даних за протоколом HTTP і означає «передача стану уявлення».

Інтерфейси прикладного програмування (API) є всюди. Вони дозволяють програмному забезпеченню взаємодіяти з іншими частинами програмного забезпечення – внутрішніми чи зовнішніми – послідовно, що є ключовим компонентом масштабованості, не кажучи вже про можливість повторного використання.

Сьогодні досить поширеним є те, що онлайн-сервіси мають загальнодоступні API. Це дозволяє іншим розробникам легко інтегрувати такі функції, як вхід у соціальні мережі, платежі кредитними картками та відстеження поведінки.

REST API використовуються для доступу до даних і керування ними за допомогою загального набору операцій без стану. Ці операції є невід'ємною частиною протоколу HTTP і представляють важливу функціональність створення, читання, оновлення та видалення (CRUD), хоча і не в чистому вигляді один на один [20]:

- POST (створити ресурс або взагалі надати дані);
- GET (отримання індексу ресурсів або окремого ресурсу);
- PUT (створити або замінити ресурс);
- PATCH (оновити/змінити ресурс);
- DELETE (видалити ресурс).

Використовуючи ці операції HTTP та ім’я ресурсу як адресу, ми можемо створити REST API, створивши кінцеву точку для кожної операції. А завдяки реалізації шаблону ми матимемо стабільну і легко зрозумілу основу, що дозволить нам швидко розвивати код і підтримувати його надалі. Як згадувалося раніше, та сама основа буде використовуватися для інтеграції сторонніх функцій, більшість з яких також використовують REST API, що робить таку інтеграцію швидшою [21].

Також важливою частиною веб-додатка є Mongo DB – це документно-орієнтована база даних NoSQL на основі колекцій та документів замість таблиць і рядків, розроблена для простоти розробки, масштабування додатків та зберігання величезної кількості даних [22]. MongoDB не є системою управління реляційними базами даних (RDBMS). Її називають базою даних "NoSQL". Вона протилежна баз даних на основі SQL, де дані не нормалізуються відповідно до схем і таблиць, де кожна таблиця має фіксовану структуру. Натомість вона зберігає дані в колекціях у вигляді документів на основі JSON і не застосовує схем. У ній немає таблиць, рядків та стовпців, як у інших базах даних SQL (RDBMS).

У базі даних RDBMS таблиця може мати кілька рядків та стовпців. Аналогічно в MongoDB колекція може мати кілька документів, еквівалентних рядкам. Кожен документ має кілька "полів", які еквівалентні стовпцям. Документи однієї колекції можуть мати різні поля.

Переваги MongoDB [23]:

− MongoDB зберігає дані як документ на основі JSON, який не вимагає дотримання схеми. Це дозволяє зберігати ієрархічні дані у документі. Це спрощує зберігання та вилучення даних ефективним чином;

− легко масштабувати архітектуру веб-додатку відповідно до вимог, оскільки база даних на основі документів. MongoDB також дозволяє розподіляти дані на кілька серверів;

− MongoDB надає багато можливостей, таких як індексування, агрегування, зберігання файлів і т.д.;

− MongoDB швидко працює з величезними даними;

− MongoDB надає драйвери для зберігання та отримання даних з різних програм, розроблених на різних технологіях, таких як C#, Java, Python, Node.js та ін.;

− MongoDB надає інструменти управління базами даних MongoDB.

2.2 Проєктування структури веб-сайта

Для зручного користування веб-додатком та пошукової оптимізації необхідна правильно організована структура сайта.

Якщо не структурувати матеріал з розумом, ваші історії, ваші дописи в блоці, ваші сторінки продуктів будуть втрачені. Ваші відвідувачі не зможуть знайти те, що вони шукають, і, що важливо для вашого SEO: Google також загубиться.

Структура дасть Google підказки про те, де знайти найважливіший вміст. Структура вашого сайта визначає, чи зможе пошукова система зрозуміти, про що ваш сайт і що ви продаєте. Google сканує веб-сайти за допомогою внутрішніх і зовнішніх посилань за допомогою бота під назвою Googlebot. Переходячи за цими посиланнями, Google визначає взаємозв’язок між різними сторінками. Структура вашого сайта є довідником для Google і тому дуже важлива.

Тож гідна структура сайта допоможе нашому сайту зайняти високі місця в Google. Але не забувайте, що структура сайта також важлива для досвіду користувача (UX). Структура вашого веб-сайта має бути відображена в навігації вашого веб-сайта. Якщо ця структура зрозуміла, ваша аудиторія легко знатиметься на вашому сайті. Хороший UX підвищить ваші шанси на конверсію: купіть ваші продукти; підпишіться на вашу розсилку або поверніться ще раз.

Розрізняють зовнішню і логічну структуру.

Зовнішня структура зумовлює розташування основних значущих елементів на кожній сторінці (меню, пошук, основний зміст, авторизація, блок з основним контентом тощо).

На кожному веб-сайті є три розділи, які залишаються незмінними:

− Header («шапка»): розташований у верхній частині веб-сайта, заголовок зазвичай містить елементи, які включають логотип компанії, навігацію по веб-сайту та контактну інформацію.

– Body (тіло): тіло відображає основний вміст веб-сторінки.

– Footer (нижній колонтитул): нижній колонтитул розташований у нижній частині веб-сторінки, вміщує в себе деяку службову інформацію (наприклад, авторські права).

Зовнішня структура веб-додатка розміщена на рис. 2.5.

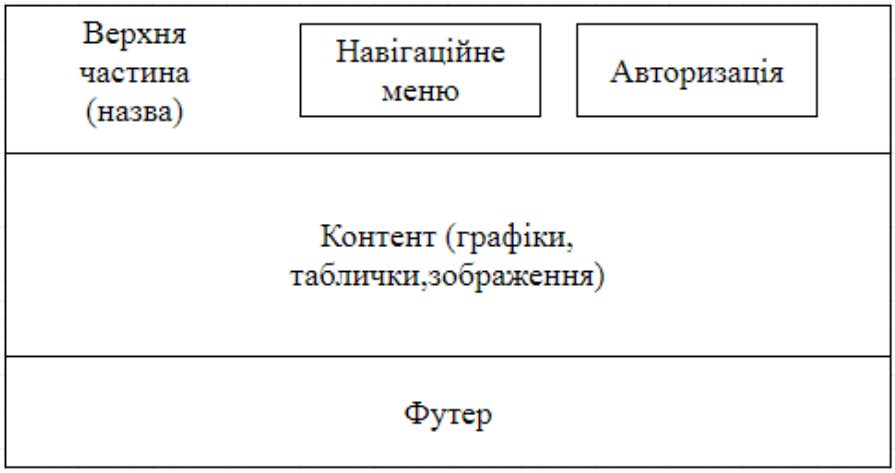

Рисунок 2.5 – Зовнішня структура веб-додатка

Простота важлива в дизайні інтерфейсу користувача. Чим більше елементів керування ви відображаєте на екрані в будь-який час, тим більше часу доведеться витратити вашим користувачам, щоб зрозуміти, як використовувати ваш інтерфейс. Коли вибір менший, доступні функції стають більш очевидними і їх легше сканувати. Проте спростити інтерфейс непросто, особливо якщо ви не хочете обмежувати функціональність програми.

Логічна організація сайта з послідовними назвами дозволяє користувачам робити успішні прогнози щодо того, де знайти шукану інформацію. Якщо розробник вводить користувачів в оману, використовуючи структуру, яка не є ні логічною, ні передбачуваною, або постійно використовує різні або неоднозначні терміни для опису функцій сайта, користувачі будуть розчаровані труднощами орієнтації та розуміння того, що може бути запропоновано.

Усі веб-сайти мають базову організаційну структуру, яка поділяється на один із чотирьох типів. Ці структури веб-сайта (або їх комбінація) можуть допомогти вам розпочати організацію сайта будь-якого розміру [24].

1. Ієрархічна модель. Це, напевно, найпоширеніший тип структур веб-сайтів. Починається з широкого набору інформації (батьківські сторінки), яка фільтрується до більш детальної інформації (дочірні сторінки). Іноді ці структури називають деревами, і вони дуже схожі на організаційні схеми в корпораціях.
2. Послідовна (лінійна) модель. Цей тип структур точно так само, як і звучить назва – ведуть відвідувачів сайта через послідовність. У той час як ієрархічна структура може спрямовувати відвідувачів сайта вниз або перейти на іншу батьківську сторінку, послідовні структури переводять відвідувача лише назад або вперед від одного кроку до іншого.
3. Матрична модель. Хоча ця структура може бути нетрадиційною, у перші роки існування Інтернету вона була досить популярною. Структура матричного типу дозволяє відвідувачам сайта вибирати, куди вони хотіли б перейти далі. Замість того, щоб створювати послідовність або обмежувати навігацію стосунками між батьками та дітьми, ця структура надає багато посилань у тематичних групах з тими, хто потрапляє на сторінку, вибираючи, куди їм рухатися далі.
4. Модель бази даних. Цей динамічний підхід до структурування веб-сайта інтегрує базу даних із пошуком. Щоб створити подібний сайт, потрібно думати знизу вгору – ретельно позначати метадані потрібного вмісту на основі принципів інформаційної архітектури. Якщо все зроблено правильно, ця структура створює сайт, на якому відвідувачі можуть створювати враження на основі того, що вони шукають.

Для розроблення сайта обрана мережна структура. Структура веб-системи показана на рис. 2.6.

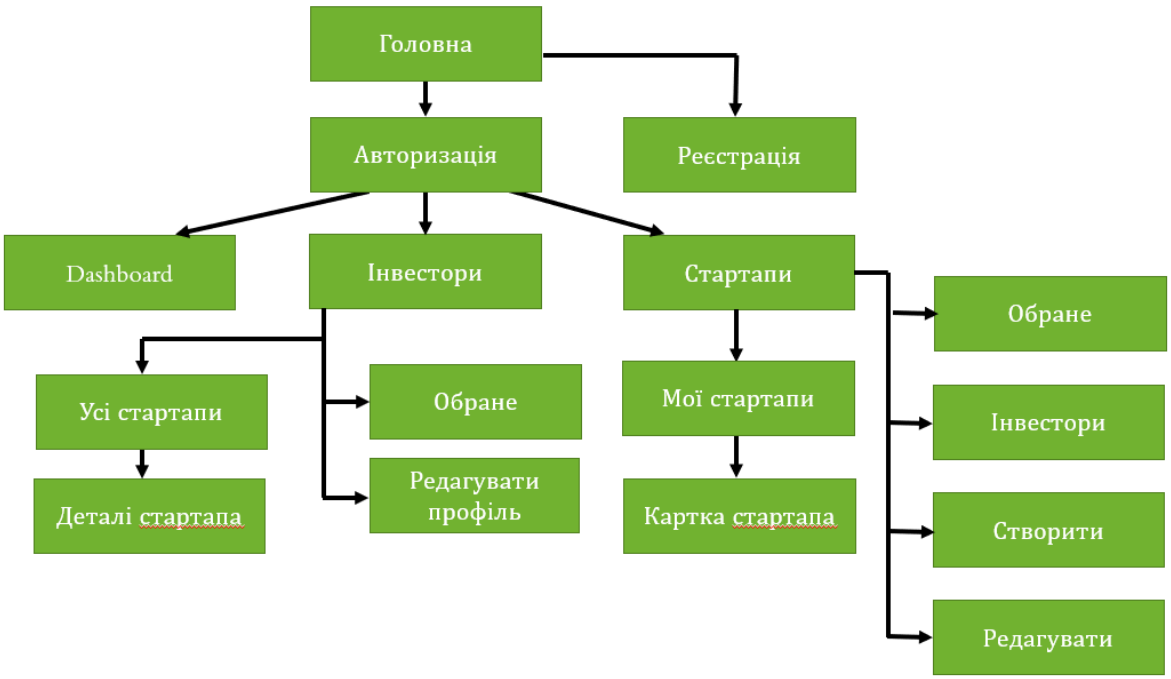

Рисунок 2.6 – Структура веб-додатка

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розроблення інтерфейсу

Кожен сайт відрізняється своїм власним інтерфейсом.

Інтерфейс користувача – це візуальне відображення, що забезпечує передачу інформації між користувачем та сервером (веб-сайтом або додатком). Тобто це перше, що бачить і з чим взаємодіє користувач при потраплянні на сайт: елементи керування, кнопки, зображення тощо.

При виконанні магістерської дипломної роботи інтерфейс веб-додатка було створено за допомогою мови розмітки HTML, каскадних таблиць стилів CSS, мов програмування JavaScript та TypeScript, фреймворків Angular та Bootstrap.

Для підвищення доступності та якості до актуальної для користувачів інформації було розроблено три інтерфейси. Перший – як головна сторінка для всіх користувачів веб-додатка, другий – сторінка для інвесторів, котрі знаходяться в пошуку проєкта для вкладання коштів на взаємовигідних умовах, третій – сторінка для стартаперів, котрі будуть мати можливість розміщувати на сайті оголошення з метою пошуку інвестора, який буде спонсорувати в молоду компанію та їх проєкт чи унікальну ідею.

Усі сторінки веб-додатка розбиті на такі основні частини: верхню («шапка»), основну (контент) та нижню (footer).

У верхній частині сайта розташована «Шапка», яка є важливою складовою макета і є наскрізним елементом (загальний для всіх сторінок сайта).

«Шапка» веб-додатка містить такі пункти меню, як логотип веб-додатка, при натисканні на яку можна повернутися на головну сторінку з будь-якої сторінки веб-сервісу, та кнопка з іконкою людини, натиснувши на яку з’являється кнопка «Login», що дає можливість авторизації користувачеві.

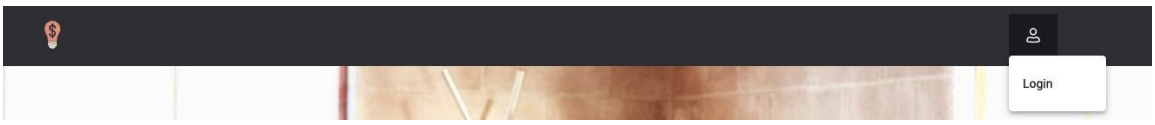

Рисунок 3.1 – «Шапка» веб-додатка

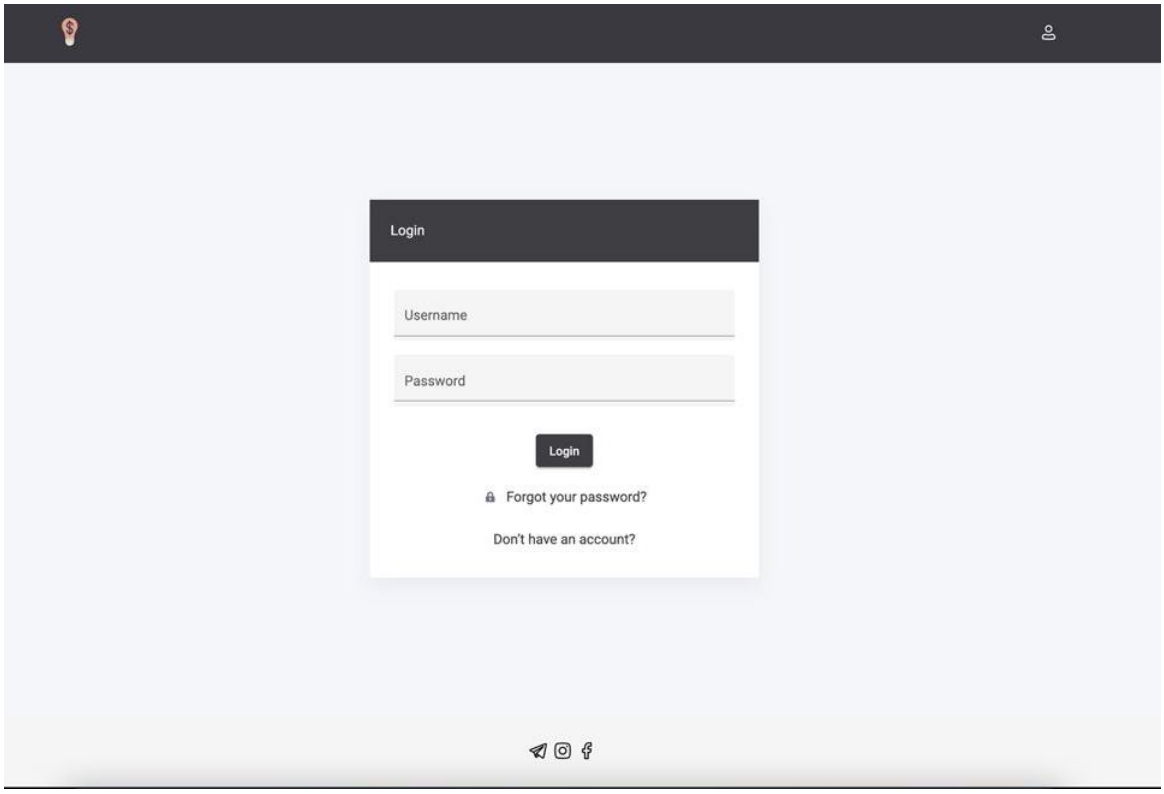

Рисунок 3.2 – Панель авторизації

Процес реєстрації складається с трьох кроків: заповнення особистих даних та вибір профілю (Startup чи Investor), введення електронної пошти і останній крок – задавання паролю (рис. 3.3–3.5).

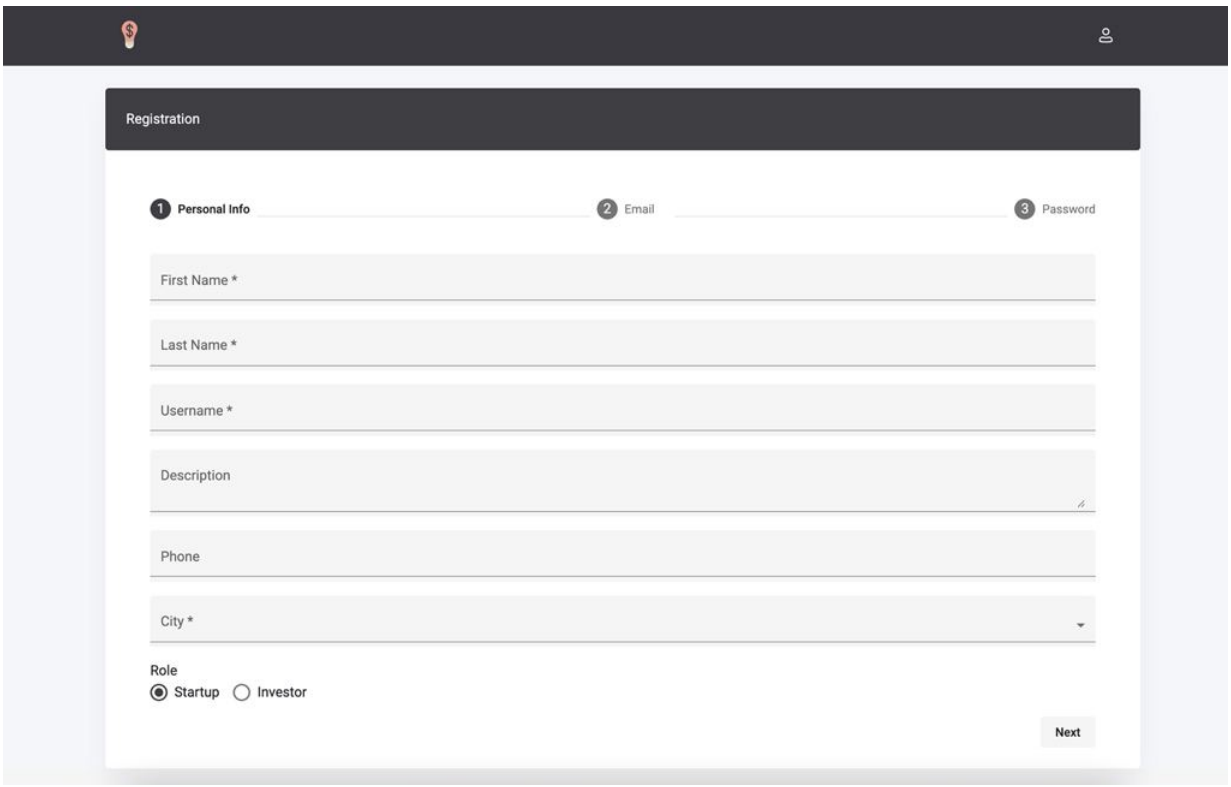

Рисунок 3.3 – Перший крок реєстрації на сайті

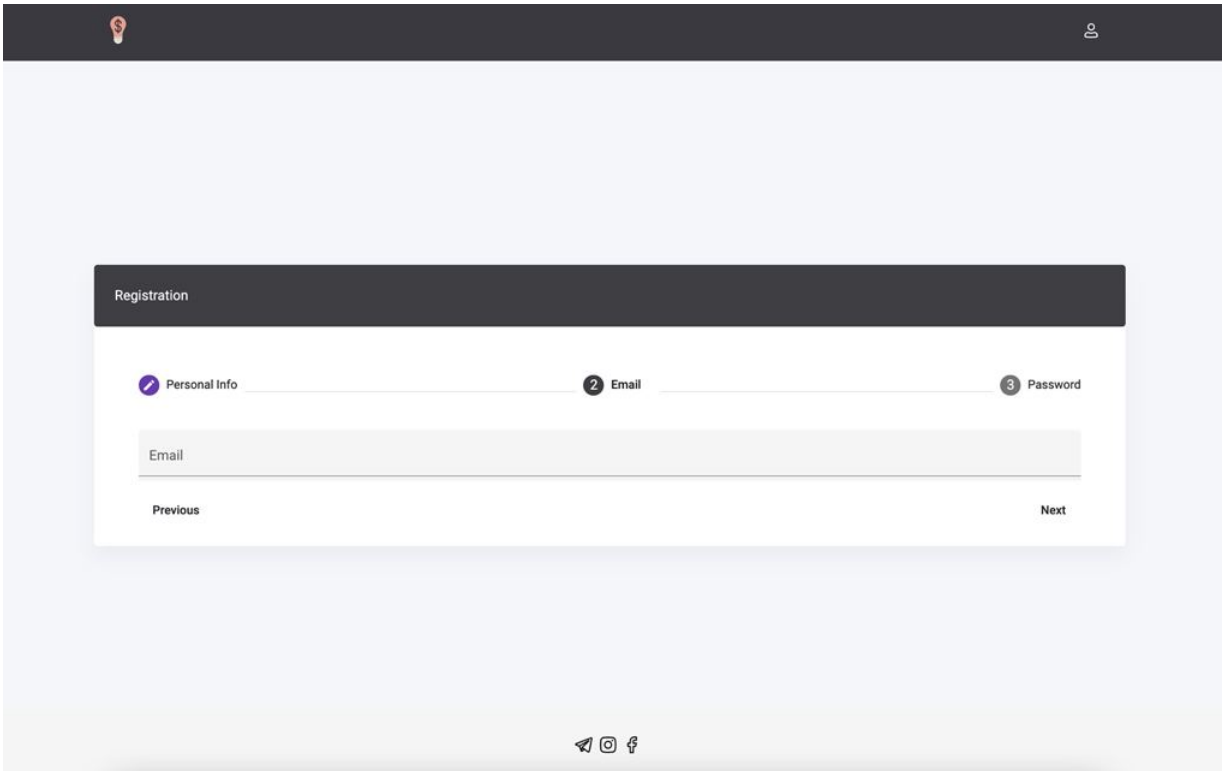

Рисунок 3.4 – Другий крок реєстрації на сайті

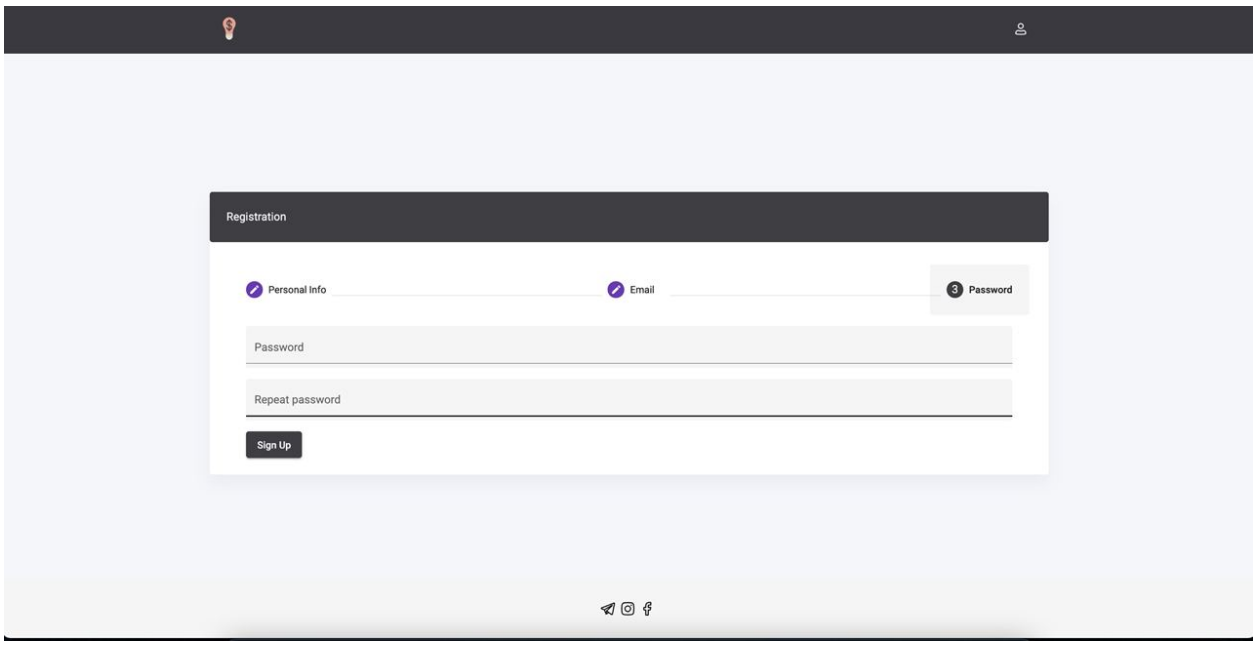

Рисунок 3.5 – Третій крок реєстрації на сайті

При авторизації у верхній частині веб-додатка функціонал дещо зміниться та з’явиться можливість перейти за посиланням “Panel” (панель управління профілем) та посиланням “Logout” (вихід зі свого профілю) (рис. 3.6).

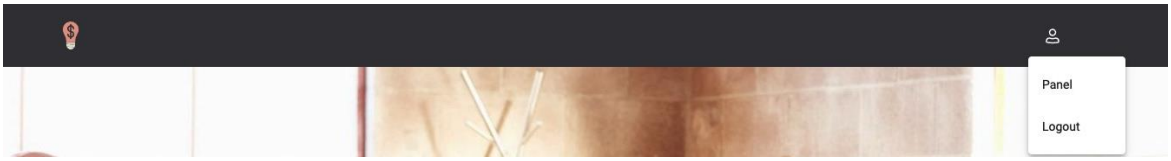

Рисунок 3.6 – «Шапка» веб-додатка після авторизації

Центральна частина сайта є найбільшим блоком на сайті, де розміщений необхідний контент: фонове зображення та три блока з ознайомлювальною інформацією з сайтом (рис. 3.7).

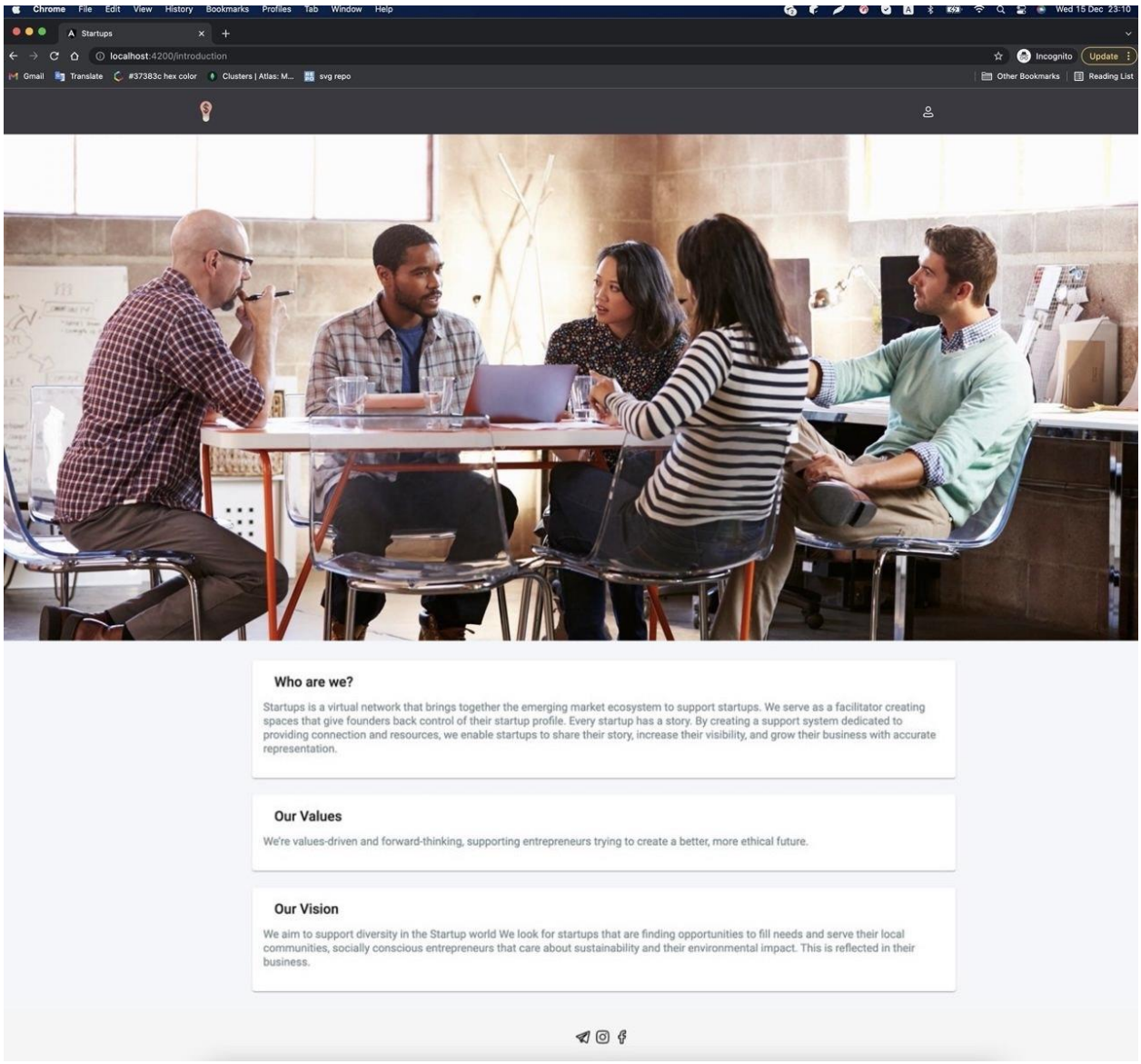

Рисунок 3.7 – Контентна частина веб-додатка

Також після авторизації з’являється навігація у лівій стороні веб-додатка з можливістю перейти за такими посиланнями як:

1. Main (головна сторінка) (рис. 3.7);

2. Profile (редагування профілю включаючи редагування пароля) (рис. 3.8);

3. Startup:

1) з профілю інвестора: перелік стартапів (рис. 3.9);

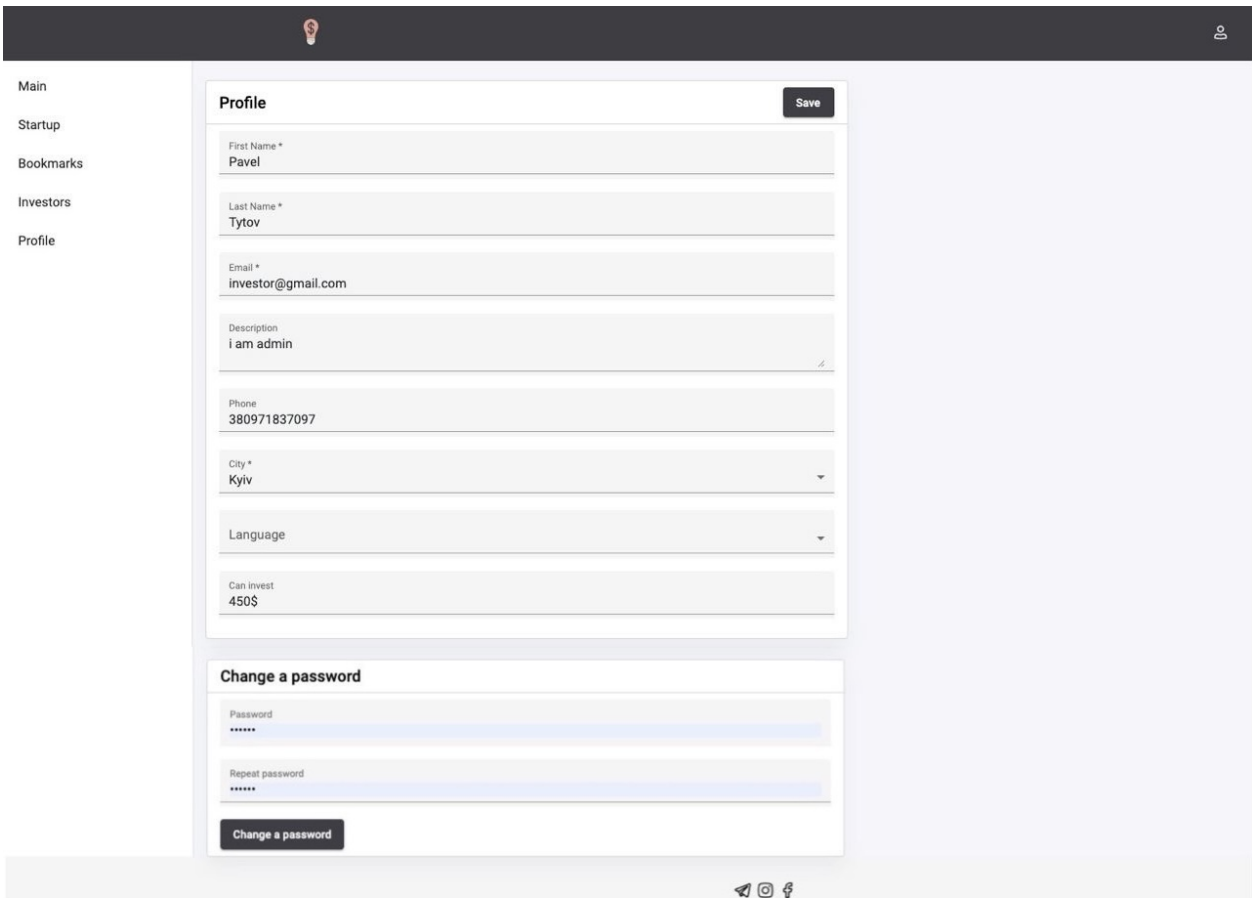

Рисунок 3.8 – Форма Profile

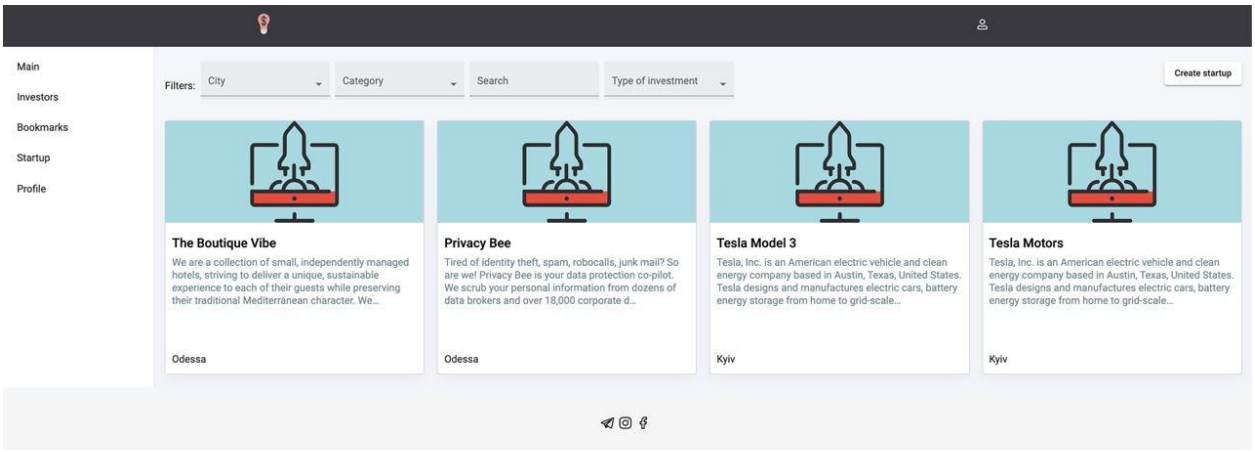

Рисунок 3.9 – Перелік стартапів

2) з профілю стартапера – перелік стартапів та можливість створити нову картку для проєкту за допомогою кнопки “Created” у правому верхньому кутку (рис. 3.10).

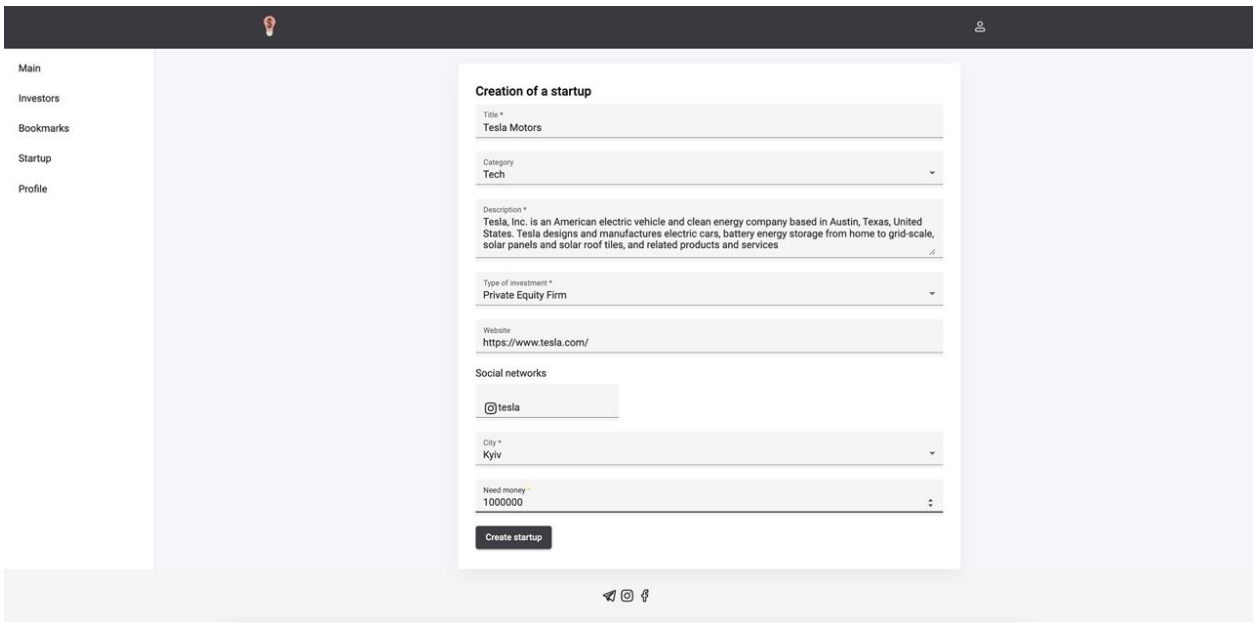

Рисунок 3.10 – Створення картки для стартапу

Картка інвестора містить у собі два блоки: перший (зліва) – особиста інформація інвестора (місто, контакти, сума яку планує інвестувати), другий (справа) – перелік стартапів, котрі інвестор додав собі в «обрані» (рис. 3.11).

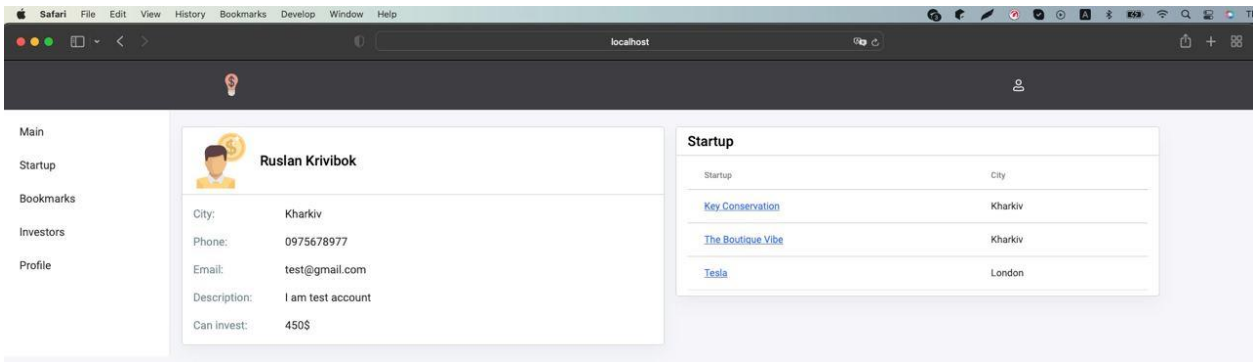

Рисунок 3.11 – Картка інвестора

Bookmarks: якщо вхід здійснено на профіль стартапера – буде відображено перелік обраних інвесторів, якщо з профілю інвестора – буде перелік обраних стартапів.

Investors: відображення переліку усіх інвесторів (рис. 3.12).

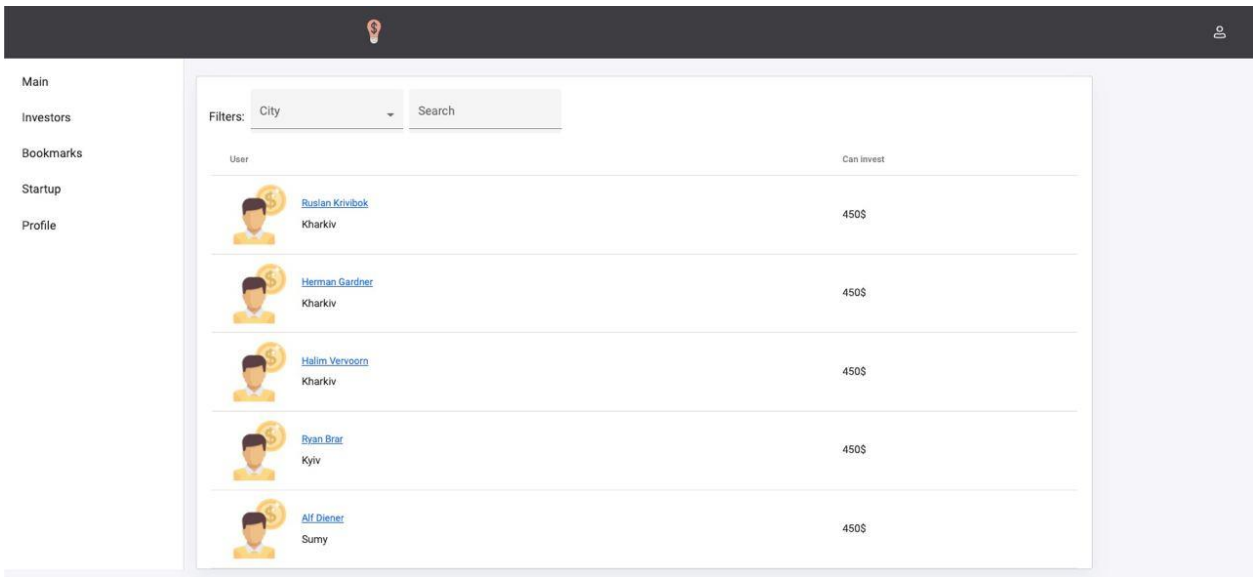

Рисунок 3.12 – Перелік зареєстрованих інвесторів

З профілю стартапера є можливість редагувати інформацію про проєкт (рис. 3.13). З профілю інвестора дана картка буде виглядати так само, тільки без можливості на її редагування (рис. 3.14).

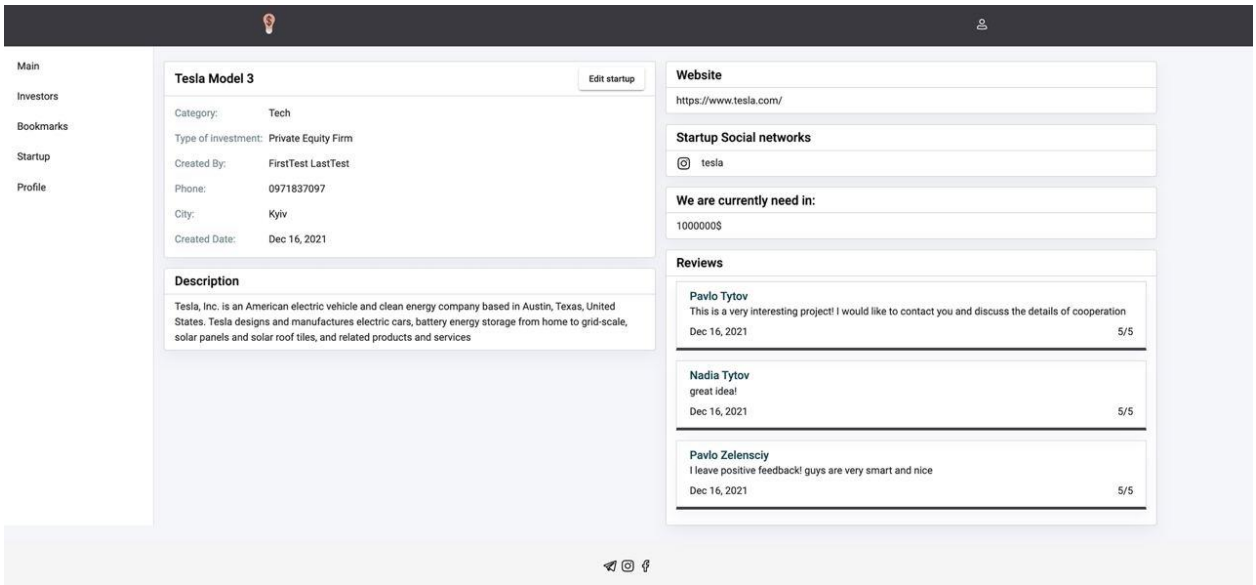

Рисунок 3.13 – Картка стартапу (очима стартапера)

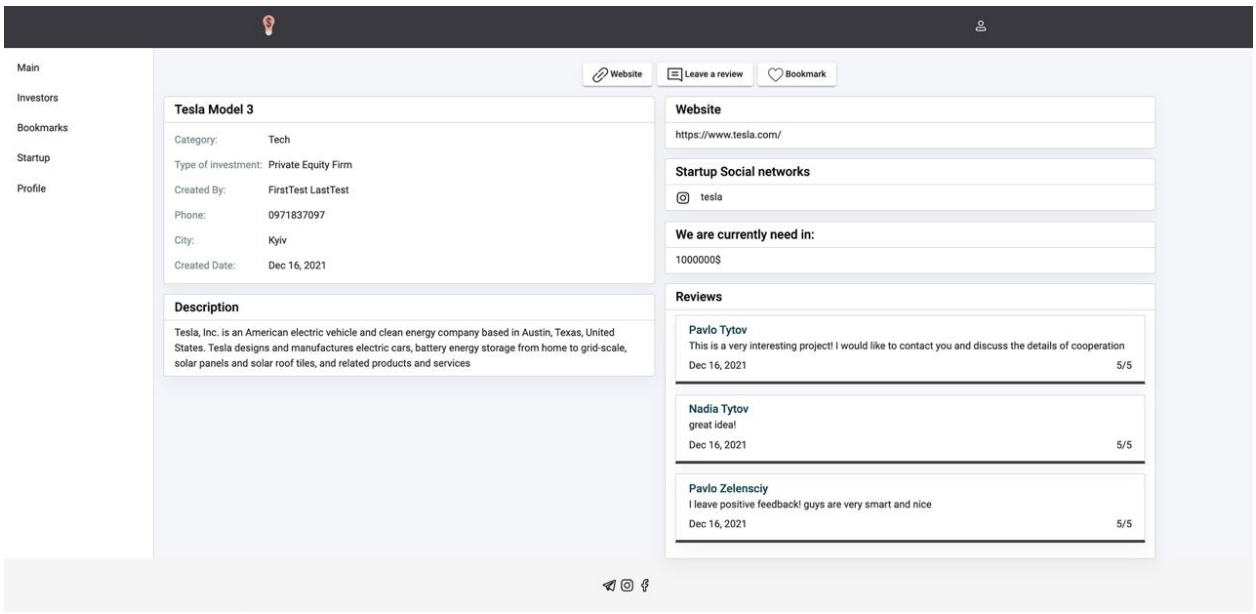

Рисунок 3.14 – Картка стартапу (очима інвестора)

Нижня частина сайта (footer) зазвичай вміщує в собі деяку службову інформацію, в даному веб-додатку він містить посилання на соціальні мережі веб-додатка (рис. 3.15).

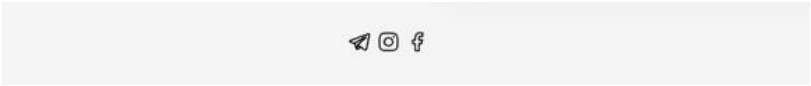

Рисунок 3.15 – Footer веб-додатка

3.2 Розроблення бази даних

У веб-додатку використовується база даних документів MongoDB із масштабованістю та гнучкістю, із необхідними запитами та індексуванням. Дана БД задовольняє найвимогливіші запити без обмежень на масштабованість.

“myFirstDatabase.startups” є основною базою даних веб-додатка, яка включає в себе дві колекції даних: startups та users (рис. 3.16–3.18).

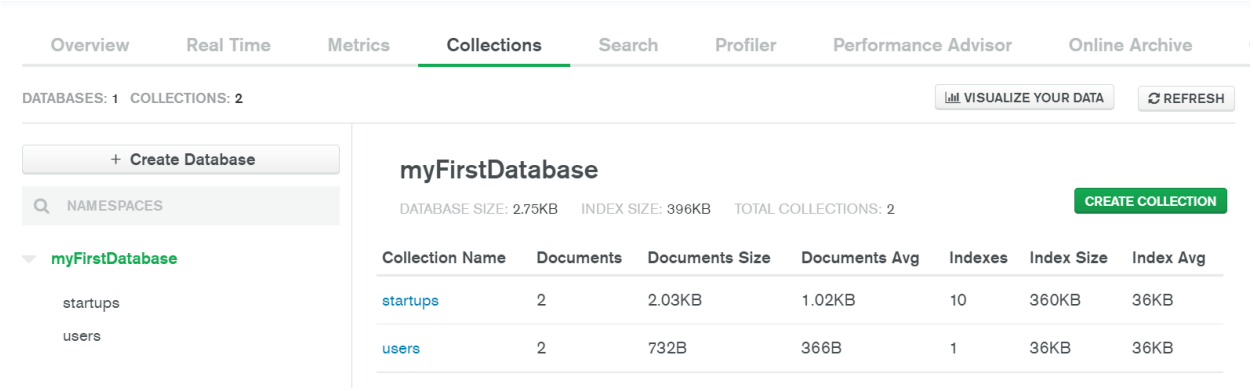

Рисунок 3.16 – Адміністративна панель бази даних “myFirstDatabase.startups”

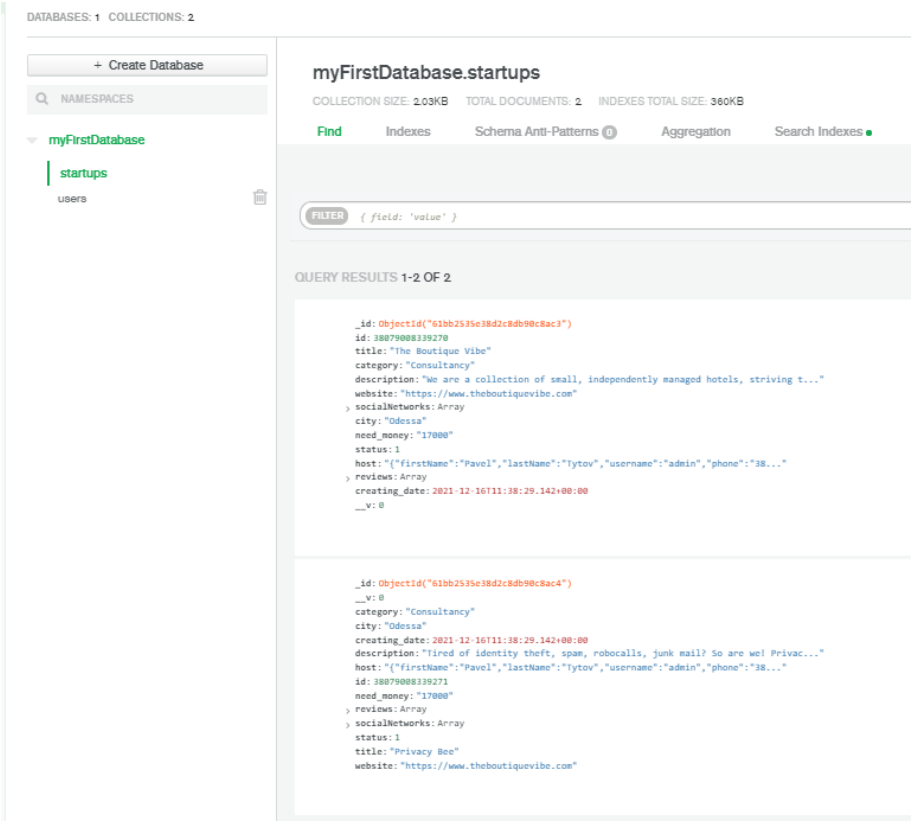

Рисунок 3.17 – Приклад даних в колекції “startups”

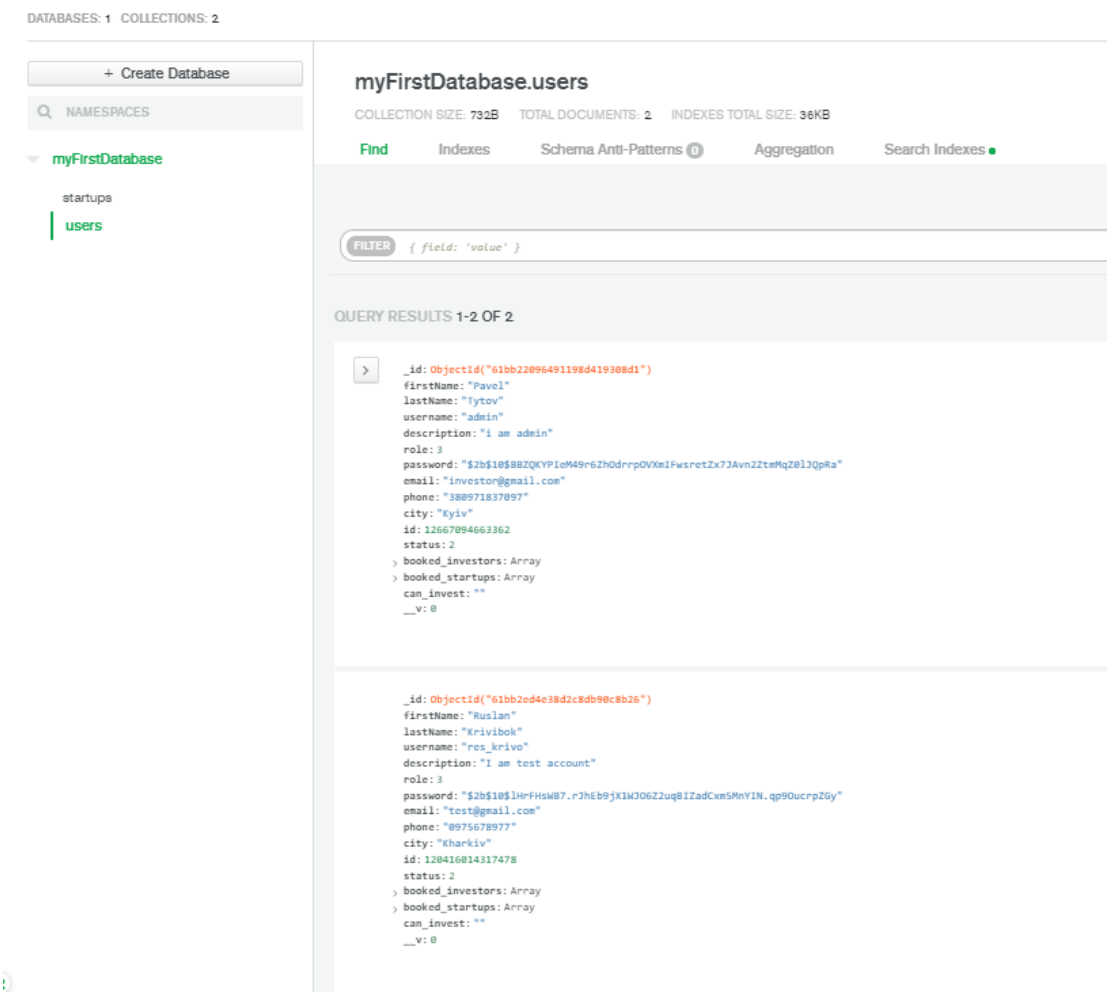

Рисунок 3.18 – Приклад даних в колекції “users”

Для звернення від клієнтської частини додатку до серверної використовуються контролери, які визначають, які функції будуть викликані на серверній частині веб-додатка (рис. 3.19, 3.20).

```
@Controller( prefix: 'startups')
export class StartupsController {
  constructor(private readonly service: StartupsService) {}

  @UseGuards(JwtGuard)
  @Get()
  @HttpCode(HttpStatus.OK)
  getAll(): Observable<StartupViewModel[]> {
    return this.service.getAll();
  }

  @Get( path: ':id')
  @HttpCode(HttpStatus.OK)
  @UseGuards(JwtGuard)
  getByUsername(@Param( property: 'id') id: number): Promise<StartupViewModel> {
    return this.service.getById(id);
  }

  @UseGuards(JwtGuard)
  @Post()
  @HttpCode(HttpStatus.CREATED)
  async create(
    @Body() createMeet: StartupCreateDto,
    @Req() req,
  ): Promise<StartupViewModel> {
    return await this.service
      .create(createMeet, req.user)
      .catch((error: ErrorMessageEnum) => {
        throw new HttpException(error, HttpStatus.CONFLICT);
      });
  }

  @UseGuards(JwtGuard)
  @Delete( path: ':id')
  @HttpCode(HttpStatus.OK)
  async delete(@Param( property: 'id') id: number): Promise<StartupViewModel> {
    return this.service.delete(id);
  }

  @Put( path: ':id')
  @HttpCode(HttpStatus.OK)
  @UseGuards(JwtGuard)
```

Рисунок 3.19 – Зміст контролера для стартапів

```
@Controller( prefix: 'users')
export class UsersController {
  constructor(private readonly userService: UsersService) {}

  @UseGuards(JwtGuard)
  @Get()
  getAll(@Req() req): Observable<Users[]> {
    const currentUsername = req.user.username;
    return this.userService.getAll(currentUsername);
  }

  @Get( path: ':username')
  getByUsername(@Param( property: 'username') username: string): Promise<Users> {
    return this.userService.getByUserName(username);
  }

  @UseGuards(JwtGuard)
  @Post()
  @HttpCode(HttpStatus.CREATED)
  async create(@Body() createUser: CreateUserDto): Promise<Users> {
    return await this.userService
      .create(createUser)
      .catch((error: ErrorMessageEnum) => {
        throw new HttpException(error, HttpStatus.CONFLICT);
      });
  }

  @UseGuards(JwtGuard)
  @Delete( path: ':id')
  async delete(@Param( property: 'id') id: number): Promise<Users> {
    return this.userService.delete(id);
  }

  @Put( path: ':username')
  @HttpCode(HttpStatus.OK)
  @Roles(RoleEnum.Admin)
  @UseGuards(JwtGuard, RolesGuard)
  async update(
    @Body() createUser: UpdateUserDto,
    @Param( property: 'username') username: string,
  ): Promise<Users> {
    return this.userService.update(createUser, username);
  }
```

Рисунок 3.20 – Зміст контролера для інвесторів

3.3 Тестування веб-додатка

Просто розробити веб-сайт недостатньо. Веб-сайт повинен бути інформативним, доступним і зручним для користувачів. Щоб зберегти всі ці якості, веб-сайт повинен бути добре перевірений, і цей процес тестування веб-сайта становить необхідну складову процесу розроблення.

Веб-тестування – це практика тестування програмного забезпечення для перевірки веб-сайтів або веб-додатків на наявність потенційних помилок. Це –

повне тестування веб-додатка перед запуском, перш ніж він стане доступним для кінцевих користувачів.

Тестування створеної системи показало її адекватність. Була перевірена відповідність вимогам, які висувались на етапах проектування і розроблення, практичність системи, сумісність з операційними системами.

Для того, щоб переконатися, що веб-система функціонує належним чином, в процесі тестування були здійснені такі перевірки:

1) Валідація поля на порожнечу.Якщо поле не заповнено – видає помилку після спроби перейти до наступного кроку реєстрації (рис. 3.21);

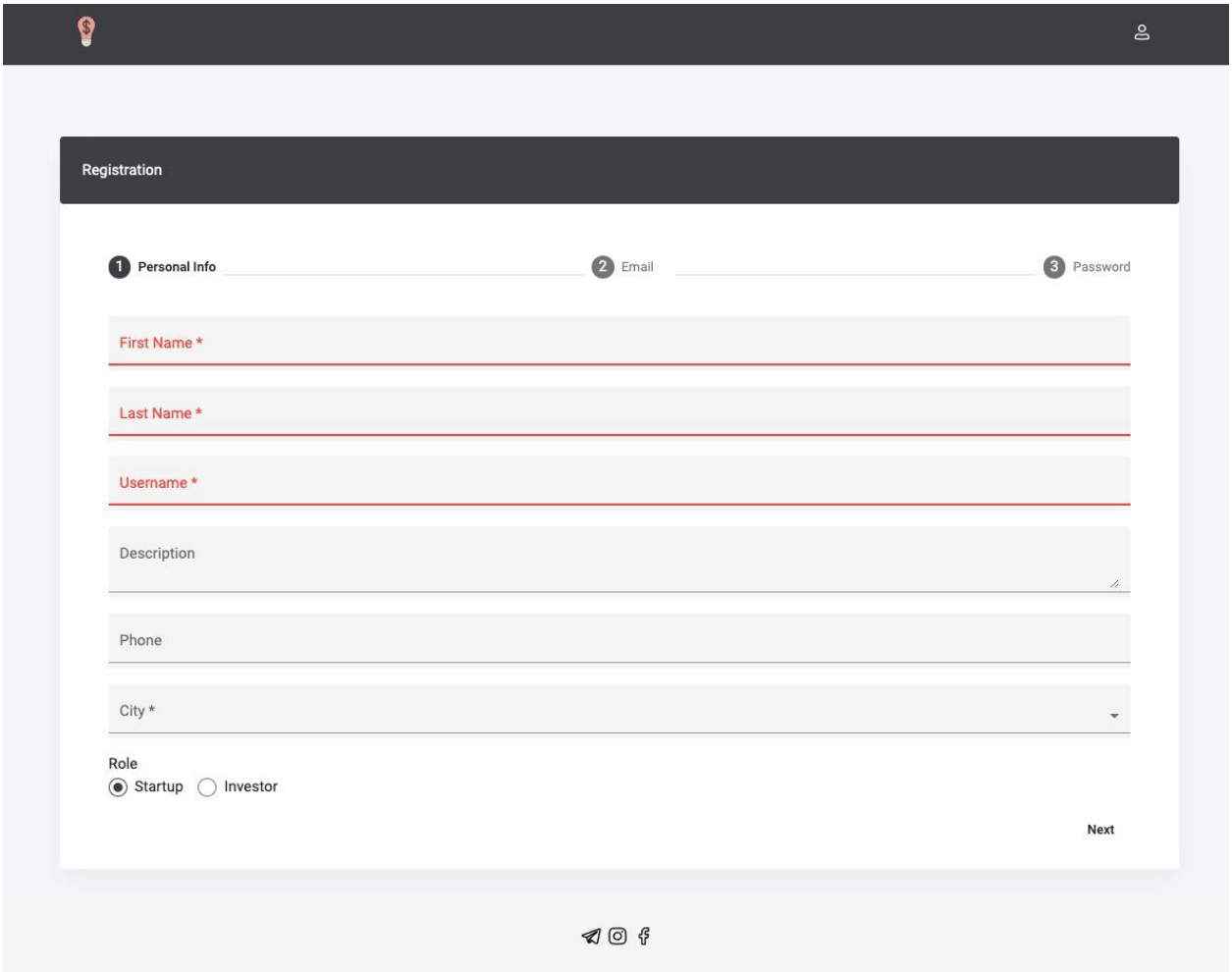

Рисунок 3.21 – Валідація поля на порожнечу

2) Перевірка кросбраузерності веб-додатка. Було перевірено у браузерах: Chrome, Firefox, Safari (рис. 3.22–3.24).

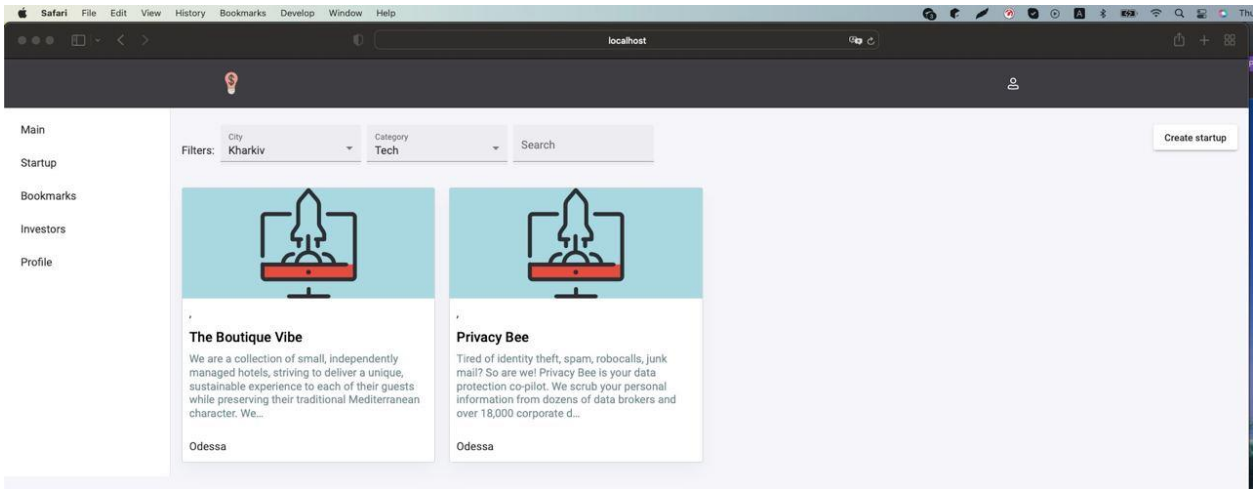

Рисунок 3.22 – Відображення сайта в браузері Safari

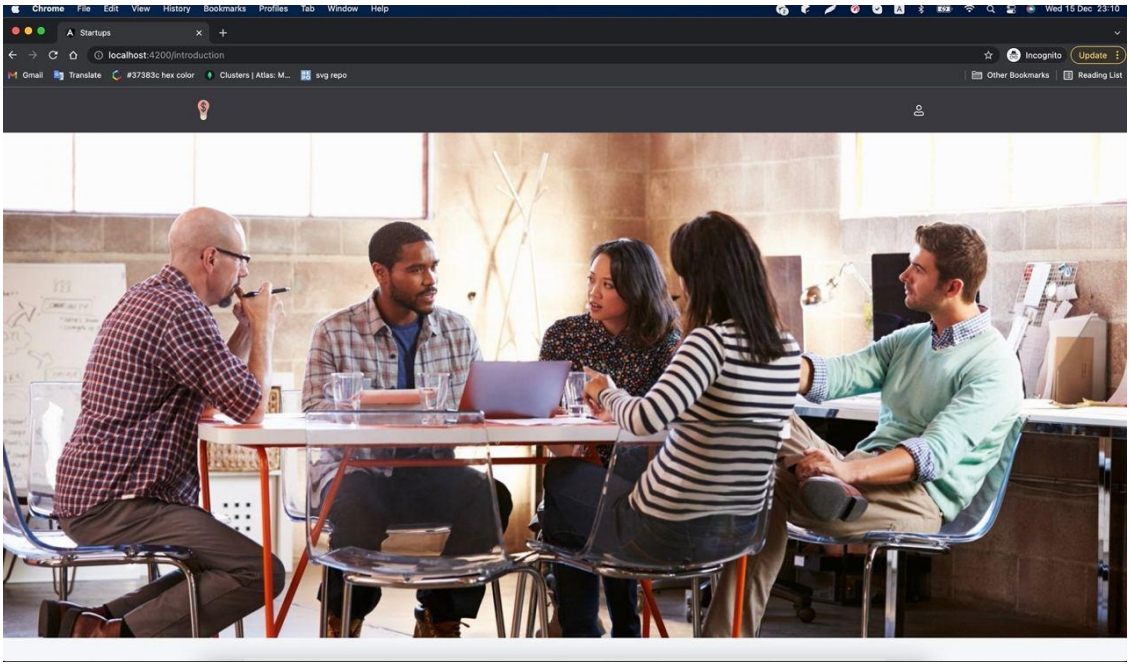

Рисунок 3.23 – Відображення сайта в браузері Chrome

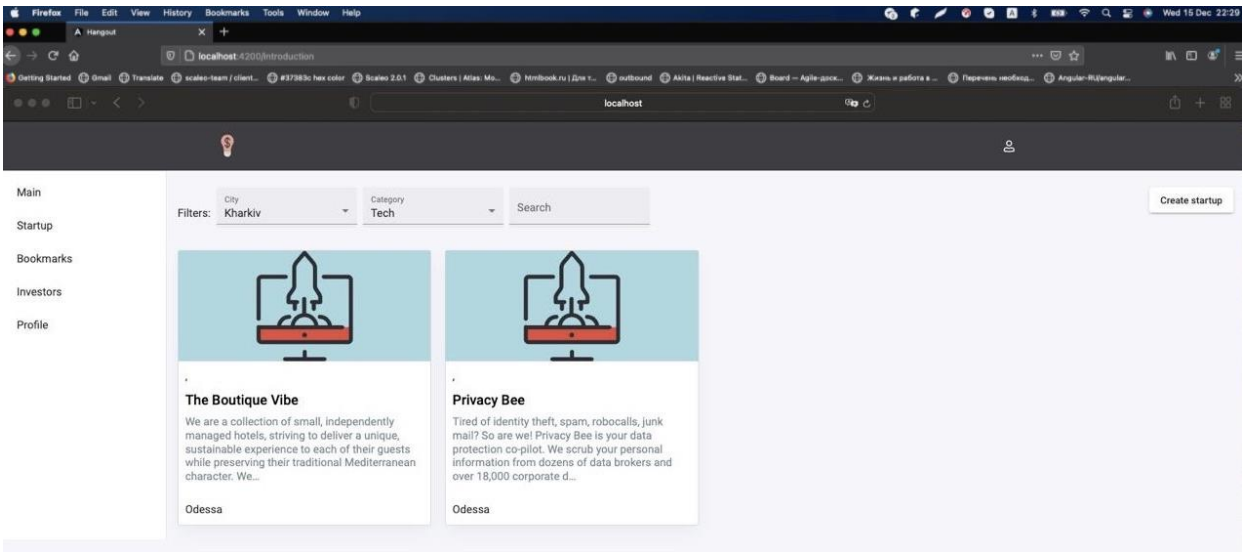

Рисунок 3.24 – Відображення сайта в браузері FireFox

3) Перевірка на коректність заповнення даних. Якщо паролі не співпадають, або друге поле порожнє – викликається помилка “Repeat password” (рис. 25).

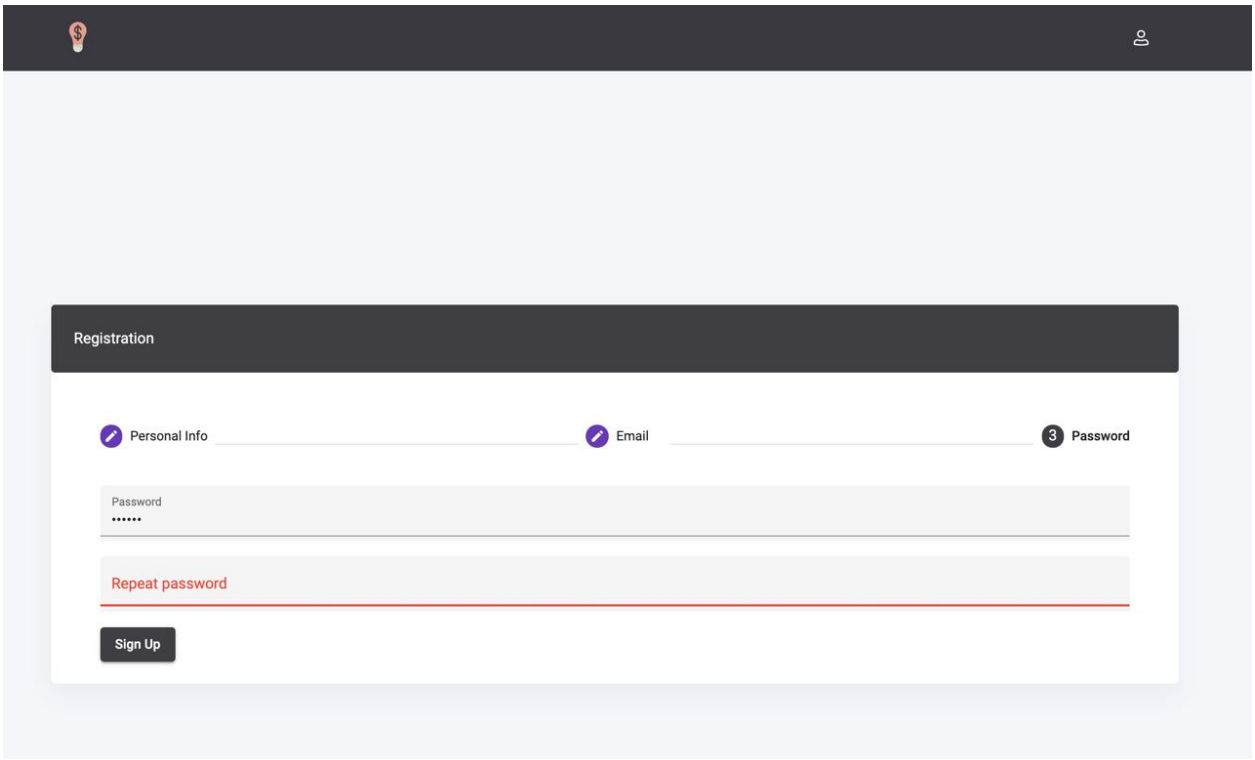

Рисунок 3.25 – Перевірка на коректність заповнення даних

4) Перевірка на захищеність засобів авторизації:

- при введені неіснуючого user-name – викликається помилка “User not found” (рис. 3.26);
- при введені некоректного паролю – викликається помилка “Incorrect password” (рис. 3.27).

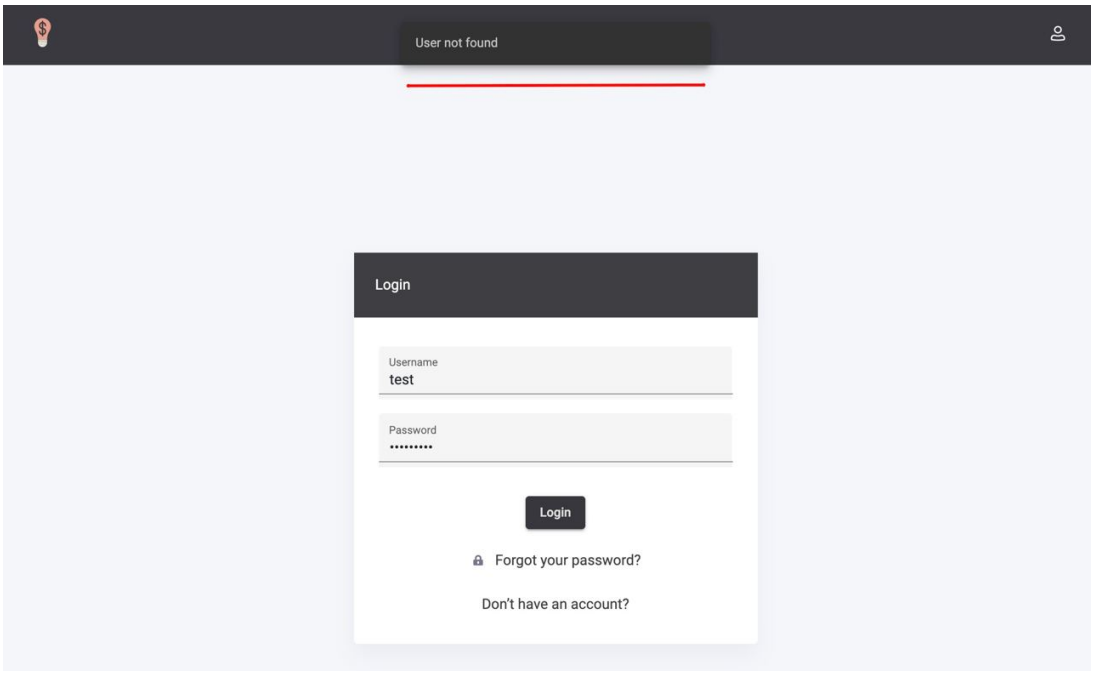

Рисунок 3.26 – Виклик помилки «User not found»

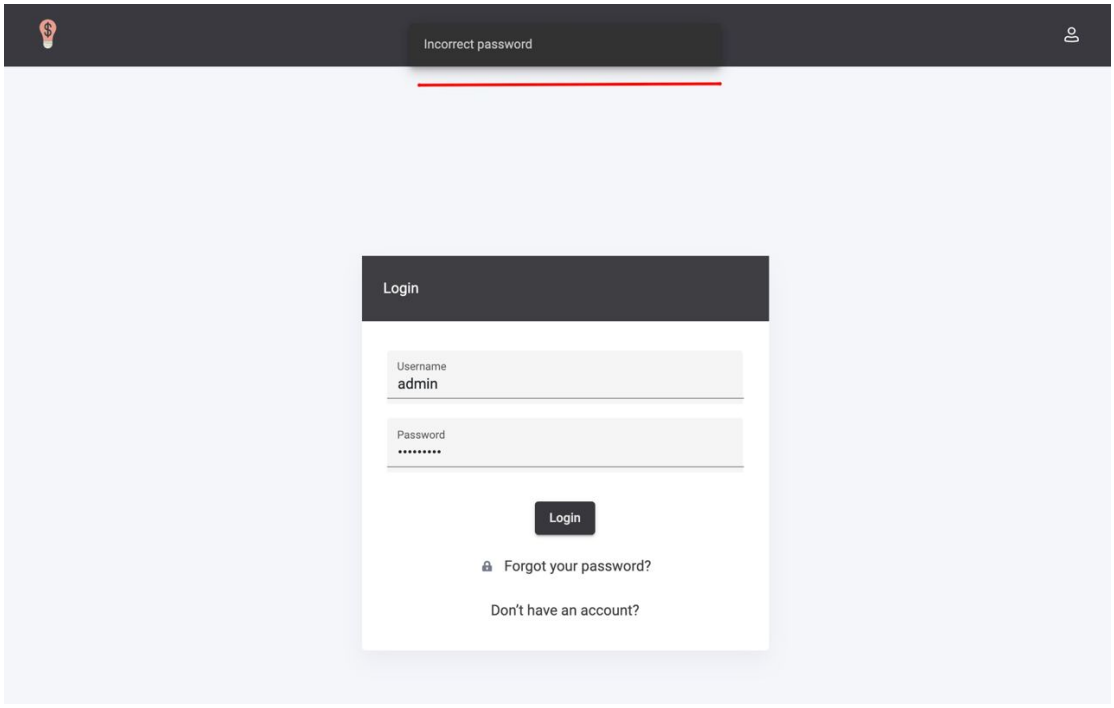

Рисунок 3.27 – Виклик помилки «Incorrect password»

5) Була виконана перевірка на коректність запису даних в базу при реєстрації (рис. 3.28–3.29).

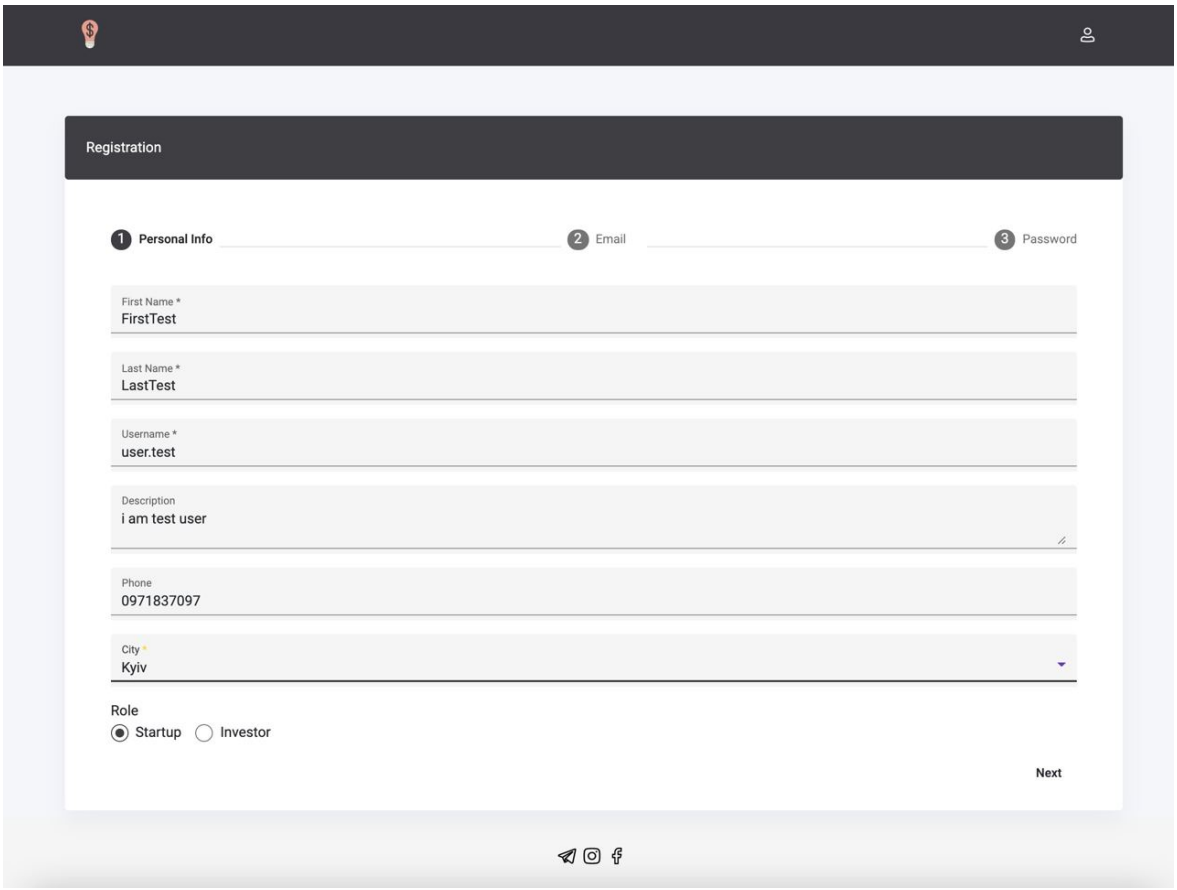

Рисунок 3.28 – Виконання реєстрації, заповнення даних

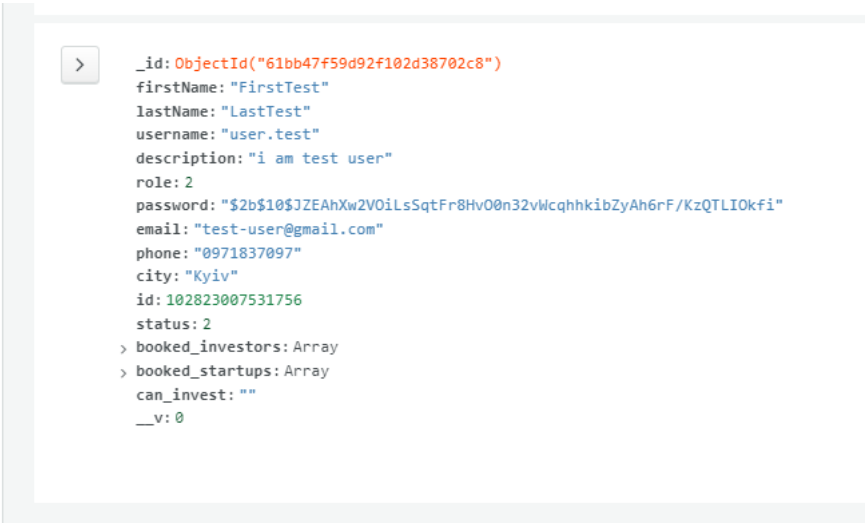

Рисунок 3.29 – Коректне відображення даних в БД після виконання реєстрації користувачем

6) Працездатність сайта перевірена на найпопулярніших операційних системах Windows та MacOs. Результати тестування підтверджують стабільну і безпомилкову роботу інформаційної системи.

3.4 Інструкція з використання веб-додатка

На стартовій сторінці веб-додатка на «шапці» розміщені такі пункти меню як логотип, котрий дає змогу з будь-якого місця повернутися на головну сторінку, іконка з зображенням людини, що дає змогу авторизуватися (рис. 3.1). Також на стартовій сторінці на контентній частині розміщена деяка ознайомлююча інформація про веб-сайт (рис. 3.7).

У випадку, якщо ви ще не зареєстровані - на панелі авторизації слід натиснути на клікабельний текст “Don’t have an account”. Якщо забули пароль, тоді натисніть на “Forgot your password” (рис. 3.2).

Для того щоб зареєструвати свій акаунт необхідно виконати три кроки (рис. 3.3–3.5):

– заповнення особистих даних та вибір профілю (Startup чи Investor);

– введення електронної пошти;

– задавання паролю.

На сторінці картки інвестора можна переглянути його особисту інформацію (місто, контакти, сума яку планує інвестувати) та перелік стартапів, котрі інвестор додав собі в «обрані» (рис. 3.11).

На сторінці картки стартапу розміщена така інформація як категорія, хто творець та його контактний номер, місто та дата створення картки. Також на даній сторінці є опис проєкту, соціальні мережі, коментарі від інвесторів та сума, котру потребує молода команда.

З профілю стартапера є можливість редагувати інформацію про проєкт (рис. 3.13). З профілю стартапа дана картка буде виглядати так само, тільки без можливості на її редагування (рис. 3.14).

Щоб перейти на панель управління профілем, потрібно перейти за посиланням “Panel”, що розміщене у верхній частині веб-додатка. Щоб вийти зі свого профілю – скористайтеся посиланням “Logout”, що розміщене поряд з посиланням “Panel” (рис. 3.6).

Скористайтеся навігацією, що розміщена у лівій стороні веб-додатка, для того щоб перейти за такими посиланнями як:

- Main (головна сторінка) (рис. 3.7);
- Profile (редагування профілю включаючи редагування пароля) (рис. 3.8);
- Startup:
    - з профілю інвестора: перелік стартапів (рис. 3.9);
    - з профілю стартапера – перелік стартапів та можливість створити нову картку для проєкту за допомогою кнопки “Created” у правому верхньому кутку (рис. 3.10).
- Bookmarks: якщо вхід здійснено на профіль стартапера – буде відображено перелік обраних інвесторів, якщо з профілю інвестора – буде перелік обраних стартапів;
- Investors: відображення переліку усіх інвесторів (рис. 3.12).

На футері веб-додатку розміщені посилання на соціальні мережі веб-додатку (Instagram, FaceBook, Telegram), щоб перейти на будь-яку з них – потрібно лише натиснути на іконку обраної вами соціальної мережі (рис. 3.15).

ВИСНОВКИ

Близько 90 відсотків стартапів зазнають невдачі, і лише 10 відсотків виживають. Існує безліч факторів, які можуть перетворити ідею стартапу на успішний бізнес, але окрім унікальної ідеї, потрібні знання про те, як залучити капітал та знайти інвесторів.

У кваліфікаційній магістерській роботі були проведені такі дослідження:

– Обґрунтована необхідність сприяння діяльності стартапів для забезпечення їхньої конкурентоспроможності на вітчизняному ринку та показаний вплив новітніх технологій на успішний розвиток бізнесу.

– Проаналізовані основні методи залучення інвесторів для просування стартапів.

– Проведений аналіз предметної області застосування, описана актуальність задачі, визначені наявні проблеми, моделі фінансування стартапів та сформульована мета.

– Створена інформаційна технологія проєктування системи залучення інвесторів для просування стартапів молоді з доступним і зручним у використанні інтерфейсом, який містить необхідну інформацію про інвесторів та бізнес, поєднано її з базою даних, щоб користувачі могли легко відфільтрувати та переглянути інформацію, яка є найбільш актуальною для їх бізнесу.

– Після реалізації проєкту було проведено тестування роботи веб-системи.

Веб-система створена згідно з усіма заданими нормами і вимогами. Усі поставлені в роботі завдання виконані у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванченко Н. О., Подскребко О. С., Сідлецька А. О. Основні проблеми та перспективи розвитку ринку стартапів в Україні. Бізнес Інформ. 2020. №4. С. 303–311. https://doi.org/10.32983/2222-4459-2020-4-303-311

2. 5 of the most common types of startups (and how they scale) [Електронний ресурс]. – Режим доступу : https://www.ringcentral.com/us/en/blog/types-of-startups/

3. Investopedia [Електронний ресурс]. – Режим доступу : https://www.investopedia.com/

4. Start-Ups That Last [Електронний ресурс]. – Режим доступу : https://hbr.org/2016/03/start-ups-that-last

5. Баб’ячок Р.І, Кульчицький І.І. Основні тенденції розвитку стартапів в Україні – проблеми, перешкоди і можливості. [Електронний ресурс]. – Режим доступу : https://www.civic-synergy.org.ua/wp-content/uploads/2018/04/Osnovni-tendentsiyi-rozvytku-startapiv-v-Ukrayini-1-1.pdf

6. Startup Financing [Електронний ресурс]. – Режим доступу : https://www.thehartford.com/business-insurance/strategy/startup/money

7. Моделі фінансування стартапів [Електронний ресурс]. – Режим доступу : https://www.businesslaw.org.ua/modeli-finansuvannya-startapiv/

8. How To Find The Right Investors To Fund Your Startup [Електронний ресурс]. – Режим доступу : https://about.crunchbase.com/blog/investors-for-startup/

9. World Wide Web [Електронний ресурс]. – Режим доступу : https://www.britannica.com/topic/World-Wide-Web

10. HTML [Електронний ресурс]. – Режим доступу : https://www.computerhope.com/jargon/h/html.htm

11. What is CSS? [Електронний ресурс]. – Режим доступу : https://www.tutorialspoint.com/css/what_is_css.htm

12. Tech 101: What Is JavaScript? [Електронний ресурс]. – Режим доступу : https://skillcrush.com/blog/javascript/

13. Flanagan D. JavaScript. The Definitive Guide. Master the World's Most-Used Programming Language, 7th Edition. O'Reilly Media, 2020. 706 p.

14. TypeScript - Overview [Електронний ресурс]. – Режим доступу : https://www.tutorialspoint.com/typescript/typescript_overview.htm

15. Murray N., Coury F., Lerner A., Taborda C. ng-book: The Complete Guide to Angular, 5th edition. Technical Editor: Frode Fikke, 2018. 626 p.

16. What is AngularJS? Introduction, Architecture & Features [Електронний ресурс]. – Режим доступу : https://www.guru99.com/angularjs-introduction.html

17. What is AngularJS? [Електронний ресурс]. – Режим доступу : https://www.tutorialsteacher.com/angularjs/what-is-angularjs

18. What is Bootstrap? A Short Bootstrap Tutorial on the What, Why, and How [Електронний ресурс]. – Режим доступу : https://www.toptal.com/front-end/what-is-bootstrap-a-short-tutorial-on-the-what-why-and-how

19. Why Choose NestJS As Your Backend Framework? [Електронний ресурс]. – Режим доступу : https://selleo.com/blog/why-choose-nest-js-as-your-backend-framework

20. Creating a Secure REST API in Node.js [Електронний ресурс]. – Режим доступу : https://www.toptal.com/nodejs/secure-rest-api-in-nodejs

21. REST API [Електронний ресурс]. – Режим доступу : https://nodejsdev.ru/doc/rest-api/

22. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide, 3rd Edition. O'Reilly Media, 2019. 514 p.

23. What is MongoDB? Introduction, Architecture, Features & Example [Електронний ресурс]. – Режим доступу : https://www.guru99.com/what-is-mongodb.html

24. Creating Website Structures that Are Built to Last [Електронний ресурс]. – Режим доступу : https://slickplan.com/blog/creating-website-structures-built-last

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
@Controller( prefix: 'startups')
export class StartupsController {
  constructor(private readonly service: StartupsService) {}

  @UseGuards(JwtGuard)
  @Get()
  @HttpCode(HttpStatus.OK)
  getAll(): Observable<StartupViewModel[]> {
    return this.service.getAll();
  }

  @Get( path: ':id')
  @HttpCode(HttpStatus.OK)
  @UseGuards(JwtGuard)
  getByUsername(@Param( property: 'id') id: number): Promise<StartupViewModel> {
    return this.service.getById(id);
  }

  @UseGuards(JwtGuard)
  @Post()
  @HttpCode(HttpStatus.CREATED)
  async create(
    @Body() createMeet: StartupCreateDto,
    @Req() req,
  ): Promise<StartupViewModel> {
    return await this.service
      .create(createMeet, req.user)
      .catch((error: ErrorMessageEnum) => {
        throw new HttpException(error, HttpStatus.CONFLICT);
      });
  }

  @UseGuards(JwtGuard)
  @Delete( path: ':id')
  @HttpCode(HttpStatus.OK)
  async delete(@Param( property: 'id') id: number): Promise<StartupViewModel> {
    return this.service.delete(id);
  }
```

```
  @Put( path: ':id')
  @HttpCode(HttpStatus.OK)
  @UseGuards(JwtGuard)
  async update(
    @Body() createUser: StartupUpdateDto,
    @Param( property: 'id') id: string,
  ): Promise<StartupViewModel> {
    return this.service.update(createUser, id);
  }
}
```

```
import { Module } from '@nestjs/common';
import { StartupsController } from './startups.controller';
import { StartupsService } from './startups.service';
import { MongooseModule } from '@nestjs/mongoose';
import { AuthModule } from '../auth/auth.module';
import { Startups, StartupsSchema } from './schemas/startups.schema';

@Module( metadata: {
  imports: [
    MongooseModule.forFeature( models: [
      { name: Startups.name, schema: StartupsSchema },
    ]),
    AuthModule
  ],
  controllers: [StartupsController],
  providers: [StartupsService],
})
export class StartupsModule {}
```

```
@Injectable()
export class StartupsService {
  constructor(
    @InjectModel(Startups.name)
    private readonly startupsModel: Model<StartupsDocument>
  ) {}

  public getAll(): Observable<StartupViewModel[]> {
    return from(this.startupsModel.find());
  }

  public async getById(id: number): Promise<StartupViewModel> {
    return this.startupsModel.findOne( filter: { id: id });
  }

  public async delete(id: number): Promise<StartupViewModel> {
    return this.startupsModel.findByIdAndRemove(id);
  }

  public async update(
    body: StartupUpdateDto,
    id: string,
  ): Promise<StartupViewModel> {
    return this.startupsModel.findOneAndUpdate( filter: { id }, body).catch( onrejected: () => {
      throw new HttpException(ErrorMessageEnum.Forbidden, HttpStatus.FORBIDDEN);
    });
  }
```

```
export type StartupsDocument = Startups & Document;

@Schema()
export class Startups {
  @Prop( options: { type: Number, required: true, unique: true })
  id: number;

  @Prop( options: { type: String, required: true, unique: false })
  title: string;

  @Prop( options: { type: String, required: true, unique: false })
  description: string;

  @Prop( options: { type: String, required: true, unique: false })
  website: string;

  @Prop( options: { type: String })
  host: any;

  @Prop( options: { type: Array })
  reviews: StartupReviewModel[] = [];

  @Prop( options: { type: Array })
  socialNetworks: SocialNetworkModel[] = [];

  @Prop( options: { type: String, required: true, unique: false })
  category: string;

  @Prop( options: { type: Date, required: true, unique: false })
  creating_date: Date;

  @Prop( options: { type: Number, required: true, unique: false })
  status: StatusEnum;

  @Prop( options: { type: String, required: true, unique: false })
  city: CityEnum;

  @Prop( options: { type: String, required: true, unique: false })
  need_money: string;
```

```
  @Prop( options: { type: String, required: true, unique: false })
  type_of_investment: TypeOfInvestmentEnum;
}

export const StartupsSchema = SchemaFactory.createForClass(Startups);
```

```
  public async create(
    meet: StartupCreateDto,
    userHost: Users,
  ): Promise<Startups> {
    const { firstName, lastName, username, phone } = userHost;
    const user: ShortUserModel = {
      firstName,
      lastName,
      username,
      phone,
    };
    return new this.startupsModel(
      this.transformCreateMeet(meet, user),
    ).save();
  }

  private transformCreateMeet(
    startup: StartupCreateDto,
    host: ShortUserModel,
  ): Startups {
    return {
      id: UuidUtil.generateNumberId(),
      ...startup,
      status: StatusEnum.Active,
      host: JSON.stringify(host),
      reviews: [],
      creating_date: new Date(),
    };
  }
}
```

```
@Injectable()
export class UsersService {
  constructor(
    @InjectModel(Users.name) private readonly userModel: Model<UsersDocument>,
  ) {}

  public getAll(currentUsername: string): Observable<Users[]> {
    return from(this.userModel.find()).pipe(
      map( project: (users : (EnforceDocument<UsersDocument, {}>)[] ) =>
        users.filter(({ username : string }) => username !== currentUsername),
      ),
    );
  }

  public async getById(id: number): Promise<Users> {
    return this.userModel.findById(id);
  }

  public async create(user: CreateUserDto): Promise<Users> {
    return this.checkUserToDuplicateEmailPhoneUsername(user) Observable<ErrorMessageEnum>
      .pipe(
        switchMap( project: (errorMessage: ErrorMessageEnum) => {
          if (errorMessage) {
            return throwError(errorMessage);
          }
          return from(this.transformUser(user));
        }),
        switchMap( project: (user: Users) => {
          return new this.userModel(user).save();
        }),
      ) Observable<Users & Document<any, any, any>>
      .toPromise();
  }

  public async delete(id: number): Promise<Users> {
    return this.userModel.findByIdAndRemove(id);
  }
```

```
public async delete(id: number): Promise<Users> {
  return this.userModel.findByIdAndRemove(id);
}

public async update(body: UpdateUserDto, username: string): Promise<Users> {
  return this.userModel.findOneAndUpdate( filter: { username }, body).catch( onrejected: () => {
    throw new HttpException(
      ErrorMessageEnum.UserNotFound,
      HttpStatus.FORBIDDEN,
    );
  });
}

public async getByUserName(username: string): Promise<Users> {
  return this.userModel.findOne( filter: { username: username }).exec();
}

getShortInfoByUsername(username: string): Observable<ShortUserModel> {
  return from(this.getByUserName(username)).pipe(
    map( project: (user : Users ) => {
      const { username, lastName, firstName, phone } = user;
      return {
        username,
        lastName,
        firstName,
        phone,
      };
    }),
  );
}
```

```
  private async transformUser(user: CreateUserDto): Promise<Users> {
    const password = await PasswordHashUtil.hashPassword(user.password);
    return {
      ...user,
      id: UuidUtil.generateNumberId(),
      status: StatusEnum.Pending,
      password,
      booked_investors: [],
      booked_startups: [],
      can_invest: '',
    };
  }

  private checkUserToDuplicateEmailPhoneUsername(
    user: CreateUserDto,
  ): Observable<ErrorMessageEnum> {
    const { phone, email, username } = user;
    return from(
      this.userModel.find( filter: { $or: [{ phone }, { email }, { username }] }),
    ).pipe(
      map( project: (users: Users[]) => {
        if (users.some((user : Users ) => user.username === username)) {
          return ErrorMessageEnum.UsernameIsUse;
        }

        if (users.some((user : Users ) => user.email === email)) {
          return ErrorMessageEnum.EmailIsUse;
        }

        if (users.some((user : Users ) => user.phone === phone)) {
          return ErrorMessageEnum.PhoneIsUse;
        }

        return null;
      }),
    );
  }
}
```

```ts
import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { Users, UsersSchema } from './schemas/users.schema';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';

@Module( metadata: {
  imports: [
    MongooseModule.forFeature( models: [{ name: Users.name, schema: UsersSchema }]),
  ],
  controllers: [UsersController],
  providers: [UsersService],
  exports: [UsersService],
})
export class UsersModule {}
```

```ts
import { Injectable } from '@nestjs/common';
import { Observable, of } from 'rxjs';
import { Users } from '../users/schemas/users.schema';
import { ProfileModel } from './models/profile.model';

@Injectable()
export class ProfileService {
  getProfile(user: Users): Observable<ProfileModel> {
    return of(this.transformUserToProfile(user));
  }

  private transformUserToProfile(user: Users): ProfileModel {
    return user;
  }
}
```

```ts
import { Module } from '@nestjs/common';
import { AuthModule } from '../auth/auth.module';
import { UsersModule } from '../users/users.module';
import { ProfileController } from './profile.controller';
import { ProfileService } from './profile.service';

@Module( metadata: {
  imports: [UsersModule, AuthModule],
  controllers: [ProfileController],
  providers: [ProfileService],
})
export class ProfileModule {}
```

```ts
import {
  Controller,
  Get,
  UseGuards,
  Request,
  Req,
  HttpCode,
  HttpStatus,
} from '@nestjs/common';
import { Observable, of } from 'rxjs';
import { JwtGuard } from '../auth/guards/jwt.guard';
import { ProfileModel } from './models/profile.model';
import { ProfileService } from './profile.service';

@Controller( prefix: 'profile')
export class ProfileController {
  constructor(private readonly service: ProfileService) {}

  @Get()
  @UseGuards(JwtGuard)
  @HttpCode(HttpStatus.OK)
  getProfile(@Req() req: Request): Observable<ProfileModel> {
    return this.service.getProfile(req['user']);
  }
}
```

```typescript
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
  ) {}

  async login(username: string): Promise<{ access_token: string }> {
    const payload = { username };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }
```

```typescript
  async validateUser(authData: AuthModel): Promise<string> {
    const { username, password } = authData;
    return await from(this.usersService.getByUserName(username)) Observable<ObservedValueOf<Promise<Users>>>
      .pipe(
        switchMap( project: (user: Users) => {
          if (!user) {
            throw new HttpException(
              ErrorMessageEnum.UserNotFound,
              HttpStatus.FORBIDDEN,
            );
          }
          return forkJoin( sources: [
            from(
              PasswordHashUtil.unHashPasswordIsMatch(user.password, password),
            ),
            of(user),
          ]);
        }),
        switchMap( project: ([unHashPasswordIsMatch, user]: [boolean, Users]) => {
          if (!unHashPasswordIsMatch) {
            throw new HttpException(
              ErrorMessageEnum.IncorrectPassword,
              HttpStatus.FORBIDDEN,
            );
          }

          return of(user.username);
        }),
      ) Observable<string>
      .toPromise();
  }

  verify(token: string): Observable<boolean> {
    return this.jwtService.verify(token);
  }
}
```

```
@Controller( prefix: 'users')
export class UsersController {
  constructor(private readonly userService: UsersService) {}

  @UseGuards(JwtGuard)
  @Get()
  getAll(@Req() req): Observable<Users[]> {
    const currentUsername = req.user.username;
    return this.userService.getAll(currentUsername);
  }

  @Get( path: ':username')
  getByUsername(@Param( property: 'username') username: string): Promise<Users> {
    return this.userService.getByUserName(username);
  }

  @UseGuards(JwtGuard)
  @Post()
  @HttpCode(HttpStatus.CREATED)
  async create(@Body() createUser: CreateUserDto): Promise<Users> {
    return await this.userService
      .create(createUser)
      .catch((error: ErrorMessageEnum) => {
        throw new HttpException(error, HttpStatus.CONFLICT);
      });
  }

  @UseGuards(JwtGuard)
  @Delete( path: ':id')
  async delete(@Param( property: 'id') id: number): Promise<Users> {
    return this.userService.delete(id);
  }
```

```
  @Put( path: ':username')
  @HttpCode(HttpStatus.OK)
  @Roles(RoleEnum.Admin)
  @UseGuards(JwtGuard, RolesGuard)
  async update(
    @Body() createUser: UpdateUserDto,
    @Param( property: 'username') username: string,
  ): Promise<Users> {
    return this.userService.update(createUser, username);
  }
}
```

```
import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { PassportModule } from '@nestjs/passport';
import { UsersModule } from '../users/users.module';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { jwtConstants } from './constants';
import { JwtStrategy } from './strategies/jwt.strategy';

@Module( metadata: {
  imports: [
    UsersModule,
    PassportModule,
    JwtModule.register( options: {
      secret: jwtConstants.secret,
      signOptions: { expiresIn: '30d' },
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [JwtStrategy, PassportModule],
})
export class AuthModule {}
```

```
@Controller( prefix: 'auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly usersService: UsersService,
  ) {}

  @Post( path: 'login')
  @HttpCode(HttpStatus.OK)
  async login(@Body() authData: AuthModel): Promise<{ access_token: string }> {
    return await from(this.authService.validateUser(authData)) Observable<ObservedV
      .pipe(switchMap( project: (username : string ) => this.authService.login(usernam
      .toPromise();
  }

  @Post( path: 'sign-up')
  @HttpCode(HttpStatus.OK)
  signUp(@Body() user: CreateUserDto): Observable<{ access_token: string }> {
    return from(this.usersService.create(user)).pipe(
      switchMap( project: ({ username }: Users) =>
        this.login( authData: { username, password: user.password }),
      ),
    );
  }

  @Post( path: 'logout')
  @HttpCode(HttpStatus.OK)
  logout(@Req() req): void {
    req.logout();
  }
}
```