Кафедра\_\_\_\_\_\_СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ\_\_\_\_\_\_\_\_

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

|  |
| --- |
| до бакалаврської кваліфікаційної роботи на тему |
| РОЗРОБКА ПЛАТФОРМИ ДЛЯ БЛАГОДІЙНИХ ПОЖЕРТВУВАНЬ ТА ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ |
|  |
|  |

|  |
| --- |
| КН-409 Погуляєв Владислав Володимирович |

Студент групи

(шифр, прізвище та ініціали)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Керівник проекту |  |  |  | **(Станекевич Олена )** | |
| Консультанти |  |  |  | **( )** | |
|  |  |  |  | **( )** | |
|  |  |  |  | **( )** | |
|  |  |  |  | **( )** | |
|  |  |  |  | **( )** | |
| Завідувач кафедри |  | Михайло ЛОБУР |  |
|  |  | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 р. | | |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА

Інститут\_\_\_\_\_\_Інститут комп’ютерних наук та інформаційних технологій\_\_\_\_\_\_\_

Кафедра\_\_\_\_\_\_Систем автоматизованого проектування\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Напрям\_\_\_\_\_\_6.122 Комп’ютерні науки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

„ЗАТВЕРДЖУЮ”

Завідувач

кафедри\_\_\_\_\_САП\_\_\_\_\_\_\_

\_\_\_\_Михайло ЛОБУР\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 р.

**ЗАВДАННЯ**

**на бакалаврську кваліфікаційну роботу студента групи КН-409 ОКР Бакалавр**

|  |
| --- |
| Погуляєв Владислав Володимирович |

(прізвище, ім’я, по батькові)

1. Тема роботи (проекту) \_\_\_\_\_\_\_\_\_ Розробка платформи для благодійних пожертвувань та волонтерської діяльності

затверджена наказом по університету від \_\_\_\_12.05.2023 р. № 1718-4-08\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Тернім здачі студентом закінченого роботи (проекту) 05.06.2023\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані для роботи (проекту) \_Благодійні оголошення; інформація про витрати на благодійні оголошення; інформація про волонтерів; інформація про збори на оголошення; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які треба розробити) «Розробка платформи для благодійних пожертвувань та волонтерської діяльності»; Використані засоби розробки; Розробка системи;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перелік графічного матеріалу \_Структура бази даних; процес «Надслиання документу на підпис»; процес\_\_\_\_\_\_\_ «Забракування задачі»; процес «Підпис документу»; процес «Відмова від підпису»; процес «Завантаження документу».\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) \_\_Visual studio, C#, .NET, MS SQL Server, HTML, CSS, JS, React.JS, Redux.JS, Git.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

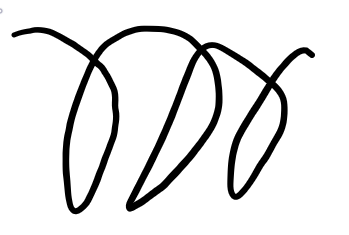
7. Консультування роботи (проекту), із зазначенням розділів роботи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Розділ | Консультант | Завдання видав | | Завдання прийняв | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

8. Дата, коли видано завдання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Станкевич Олена**\_\_\_\_\_\_\_\_\_\_\_\_\_

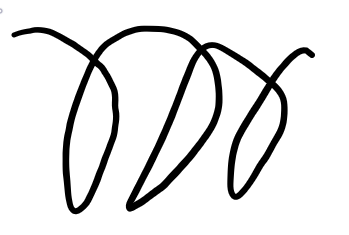
(підпис)

Завдання прийняв до виконання \_\_\_\_  \_\_\_\_\_\_\_\_**Погуляєв Владислав**\_\_\_\_\_

(підпис)

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назв і етапів роботи (проекту) | Термін виконання етапів роботи (проекту) | Примітка |
| 1 | Початок оформлення звіту | 12.05.2023 – 14.05.2023 | Виконано |
| 2 | Написання вступу до БКР | 14.05.2023 – 16.05.2023 | Виконано |
| 3 | Проектування програмного продукту | 16.05.2023 – 19.05.2023 | Виконано |
| 4 | Розробка базового функціоналу користувачів | 19.05.2023 – 24.05.2023 | Виконано |
| 5 | Розробка функціоналу надсилачів | 24.05.2023 – 26.05.2023 | Виконано |
| 6 | Розробка функціоналу підписантів | 26.05.2023 – 31.05.2023 | Виконано |
| 7 | Розробка функціоналу адміністраторів | 31.05.2023 – 04.06.2023 | Виконано |
| 8 | Внесення правок | 04.06.2023 – 05.06.2023 | Виконано |
| 9 | Підготовка до захисту БКР | 05.06.2023 – 13.06.2023 | Виконано |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

**АНОТАЦІЯ**

Погуляєв Владислав, Станкевич Олена (керівник). Розробка платформи для благодійних пожертвувань та волонтерської діяльності. Бакалаврська квалфікаційна робота. – Національний університет “Львівська політехніка”, Львів, 2023.

Платформа для благодійних пожертвувань та волонтерської діяльності є актуальним проектом у сучасному світі. За останні кілька років, зацікавленість людей до благодійності та волонтерства зросла. За допомогою такої платформи, люди можуть з легкістю знайти організації, яким вони можуть допомогти, а також знайти волонтерські можливості для своєї участі у різних проектах.

*Об’єктом* дослідження у бакалаврській роботі є інформаійна система для благодійних пожертвувань.

*Метою* бакалаврської кваліфікаційної роботи є розроблення платформи для благодійних пожертвувань та волонтерської діяльності, яка працює з даними на локальній або віддаленій базі даних.

Для досягнення поставленої мети *необхідно розв’язати такі* *задачі*:

* здійснити аналіз існуючих платформ для благодійних пожертвувань та волонтерської діяльності, виокремити їх переваги та недоліки;
* для реалізації проєкту вибрати сучасні технологіі розробки веб-додатків;
* спроектувати блок-схему програмного продукту;
* розробити структуру бази даних;
* визначити функціональні вимоги до платформи, включаючи можливості для збору пожертвувань, пошуку волонтерських активностей, створення профілів користувачів тощо;
* розробити користувацький інтерфейс платформи, забезпечуючи зручну навігацію та зрозумілість для користувачів;
* створити систему авторизації та безпеки, щоб забезпечити захист особистої інформації користувачів та недопущення несанкціонованого доступу до даних;
* провести тестування та налагодження платформи, забезпечивши її безперебійну роботу та високу продуктивність.

У бакалаврській роботі використали такі *методи досліджень*: аналіз та синтез, проектування, розроблення алгоритмів.

У якості засобів розробки були використанj середовищі [Visual Studio](https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022) [6].

*Наукова новизна* роботи полягає у тому, що після успішного підключення до бази даних користувач може переглядати та сортувати оголошення, зареєструвати обліковий запис та отримати статус волонтера. Крім того, користувачам надається можливість приєднатися до організацій як волонтери або створювати власні оголошення. Створена платформа працює з локальними та віддаленими базами даних.

*Практична цінність* спроектованого продукту у тому, що платформа забезпечує ефективний взаємозв’язок між організаціями, волонтерами та потенційними клієнтами. Тестування системи підтвердило необхідність цього продукту та необхідність вкладення ресурсів у його розробку.

**ABSTRACT**

Pohuliaiev Vladyslav, Stankevych Olena (supervisor). Development of a platform for charitable donations and volunteering. Bachelor's qualification work. - Lviv Polytechnic National University, Lviv, 2023.

The platform for charitable donations and volunteering is a relevant project in the modern world. Over the past few years, people's interest in charity and volunteering has grown. With the help of such a platform, individuals can easily find organizations they can contribute to and discover volunteer opportunities to participate in various projects.

The research object of the bachelor's work is an information system for charitable donations.

The aim of the bachelor's qualification work is to develop a platform for charitable donations and volunteering that works with data on a local or remote database.

To achieve the set goal, the following tasks need to be addressed:

* Conduct an analysis of existing platforms for charitable donations and volunteering, identifying their advantages and disadvantages.
* Select modern web application development technologies for project implementation.
* Design the block diagram of the software product.
* Develop the database structure.
* Define functional requirements for the platform, including capabilities for donation collection, volunteer activity search, user profile creation, etc.
* Design a user interface for the platform, ensuring convenient navigation and user-friendliness.
* Create an authentication and security system to protect users' personal information and prevent unauthorized access to data.
* Conduct testing and debugging of the platform, ensuring its smooth operation and high performance.

The research methods used in the bachelor's work include analysis and synthesis, design, and algorithm development.

Environments were used as development tools [Visual Studio](https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022) [6].

The scientific novelty of the work lies in the fact that after successful database connection, users can view and sort announcements, register an account, and obtain a volunteer status. Additionally, users have the ability to join organizations as volunteers or create their own announcements. The developed platform works with local and remote databases.

The practical value of the designed product lies in providing an efficient interaction between organizations, volunteers, and potential clients. Testing of the system has confirmed the necessity of this product and the need for resource investment in its development.

**СПИСОК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ**

API – Application Programming Interface (Прикладний програмний інтерфейс)

DB – Data base (База даних)

ER – Entity-Relationship (Сутність – Зв’язок)

ID – Identity (Ідентифікатор)

JSON – JavaScript Object Notation (об'єктний запис JavaScript)

UI – User interface (Інтерфейс користувача)

UML – Unified Modeling Language (Уніфікована мова моделювання)

VS – Visual Studio

XML – Extensible Markup Language (Розширювана моова розмітки)

БД – База даних

ОС – Операційна система

ПЗ – Програмне забезпечення

СКБД – Система керування базами даних

**ЗМІСТ**

[ВСТУП 10](#_Toc136945294)

[1.1. Платформа для благодійних пожертвувань та волонтерської діяльності. Поняття та види 18](#_Toc136945295)

[1.2. Приклади існуючих платформ для благодійних пожертвувань та волонтерської діяльності. 18](#_Toc136945296)

[1.2.1. GoFundMe 18](#_Toc136945297)

[1.2.2 DonorsChoose 20](#_Toc136945298)

[1.2.3 VolunteerMatch 21](#_Toc136945300)

[1.2.4 GlobalGiving 23](#_Toc136945301)

[1.2.5 Patreon 24](#_Toc136945302)

[1.3 Збереження даних 26](#_Toc136945303)

[1.3.1 JSON 28](#_Toc136945304)

[1.3.2 База даних 28](#_Toc136945306)

[Висновок 30](#_Toc136945307)

[РОЗДІЛ 2. ВИКОРИСТАНІ ЗАСОБИ РОЗРОБКИ 31](#_Toc136945308)

[2.1. ASP.NET 31](#_Toc136945309)

[2.2. Entity Framework 34](#_Toc136945310)

[2.3. React.JS 36](#_Toc136945311)

[2.4. Redux.JS 38](#_Toc136945312)

[2.5. Visual Studio 40](#_Toc136945313)

[2.7. SQL мова 43](#_Toc136945314)

[Висновок 45](#_Toc136945315)

[РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ 46](#_Toc136945316)

[3.1. Розробка архітектури системи 46](#_Toc136945317)

[3.2. Use-case діаграма 47](#_Toc136945318)

[3.3. Розробка дизайну програми 49](#_Toc136945320)

[3.4. Розробка системи 56](#_Toc136945321)

[3.5. Основний функціонал системи 74](#_Toc136945322)

[Висновок 78](#_Toc136945323)

[ВИСНОВКИ 79](#_Toc136945324)

[Використана література. 80](#_Toc136945325)

ВСТУП

Платформа для благодійних пожертвувань та волонтерської діяльності є актуальним проектом у сучасному світі. За останні кілька років, зацікавленість людей до благодійності та волонтерства зросла. За допомогою такої платформи, люди можуть з легкістю знайти організації, яким вони можуть допомогти, а також знайти волонтерські можливості для своєї участі у різних проектах.

Метою розробки цієї платформи є полегшення процесу благодійного пожертвування та волонтерської діяльності для користувачів. Крім того, платформа надає можливість швидко знайти необхідну інформацію про благодійні організації та проекти, що можуть зацікавити користувачів.

Науково-технічною новизною цього проекту є те, що він дає можливість пожертвувати кошти на різні проекти з усього світу, а також створювати власні оголошення про благодійну допомогу. Основним розв'язком є створення платформи, яка працює з локальними та віддаленими базами даних.

Завдяки цьому проекту спрощується пошук інформації про благодійні організації та волонтерські можливості. Використовуючи розроблену платформу, користувачі можуть швидко знайти необхідну інформацію, зареєструватися як волонтери, а також приєднатися до організацій для допомоги в їхніх проектах.

У сучасному світі, де глобальні проблеми і соціальні виклики потребують невідкладних рішень, платформи для благодійних пожертвувань і волонтерської діяльності виявляються надзвичайно важливими інструментами. Ці платформи надають людям можливість допомагати тим, хто потребує допомоги, та сприяють розвитку громадської свідомості та волонтерства.

Сучасний стан платформ для благодійних пожертвувань і волонтерської діяльності є насиченим і різноманітним. Інтернет та соціальні мережі відіграють вирішальну роль у поширенні інформації про благодійні організації, акції та проекти. Завдяки широкому доступу до Інтернету, люди можуть легко знайти потрібну інформацію, ознайомитися з роботою благодійних організацій та здійснити пожертвування онлайн.

Одним з важливих аспектів розвитку цих платформ є забезпечення безпеки та довіри. Люди хочуть бути впевненими, що їхні пожертвування йдуть на потреби, які заявлені благодійною організацією. Тому платформи розвиваються в напрямку впровадження механізмів перевірки та аудиту, щоб забезпечити прозорість та відкритість діяльності благодійних організацій.

Актуальність платформ для благодійних пожертвувань і волонтерської діяльності є безперечною у сучасному світі. Існує безліч причин, які обгрунтовують їх важливість і необхідність.

По-перше, платформи для благодійних пожертвувань і волонтерської діяльності сприяють вирішенню нагальних соціальних проблем і допомагають тим, хто знаходиться у складних життєвих обставинах. Через такі платформи можна пожертвувати кошти на лікування хворих дітей, допомогу безпритульним тваринам, підтримку постраждалих від природних катастроф і багато іншого. Волонтерська діяльність також відіграє важливу роль у підтримці громад та сприяє їхньому розвитку.

По-друге, платформи для благодійних пожертвувань і волонтерської діяльності допомагають залучати широке коло людей до активної участі в благодійних справах. Вони створюють можливості для кожного бажаючого зробити свій внесок у доброчинність, незалежно від фінансових можливостей або часових обмежень. Це робить благодійність доступною та зручною для широкого загалу людей, що сприяє збільшенню кількості пожертвувань та волонтерських ініціатив.

По-третє, платформи для благодійних пожертвувань і волонтерської діяльності забезпечують прозорість і відкритість в управлінні коштами та розподілі ресурсів. Благодійні організації, які працюють через такі платформи, зобов'язані публікування або соціальної підтримки. Це забезпечує довіру серед донорів і волонтерів, а також забезпечує ефективне використання ресурсів.

Крім того, платформи для благодійних пожертвувань і волонтерської діяльності стимулюють розвиток нових інноваційних рішень і технологій. Вони надають можливість використовувати сучасні цифрові інструменти, такі як мобільні додатки, онлайн-платформи, соціальні медіа та краудфандинг, для збору пожертв і залучення волонтерів. Це сприяє покращенню комунікації, зручності та швидкості взаємодії між благодійними організаціями, донорами та волонтерами.

Метою роботи платформ для благодійних пожертвувань і волонтерської діяльності є не лише забезпечення фінансової та практичної підтримки потребуючим, але й побудова світу, де солідарність, емпатія та допомога є ключовими цінностями. Ці платформи пропагують громадську свідомість та відповідальність, надихаючи людей на активну участь у розв'язанні соціальних проблем.

У майбутньому перспективи розвитку платформ для благодійних пожертвувань і волонтерської діяльності є значними. За допомогою штучного інтелекту, аналітики даних та розширеної реальності можна досягти ще більшої ефективності в зборі коштів, розподілі ресурсів та плануванні благодійних проектів. Також очікується подальше розширення міжнародної спі впрацю та співпраці між благодійними організаціями, які діють на різних континентах. Це дозволить здійснювати глобальні проекти і більш ефективно реагувати на надзвичайні ситуації, такі як природні катастрофи чи гуманітарні кризи.

Також очікується зростання впливу соціальних мереж і медіа на розвиток благодійності. Платформи для благодійних пожертвувань і волонтерської діяльності використовуватимуть їх як засоби для поширення інформації, залучення громадської уваги та мобілізації громадської підтримки. Соціальні мережі стануть платформами для створення спільноти волонтерів та донорів, обміну досвідом і ідеями, а також залучення нових учасників до благодійної діяльності.

Окрім того, платформи для благодійних пожертвувань і волонтерської діяльності будуть активно розвиватися в напрямку забезпечення більш гнучких та індивідуалізованих можливостей для донорів і волонтерів. Вони будуть пропонувати різноманітні форми пожертвування, включаючи регулярні внески, мікропожертви, краудфандингові кампанії та спеціальні проекти. Також будуть розроблятися зручні та інтерактивні інструменти для моніторингу використання коштів та оцінки соціального впливу благодійних проектів.

Отже, загальні відомості щодо сучасного стану і перспектив розвитку платформ для благодійних пожертвувань і волонтерської діяльності показують, що ці платформи відіграють ключову роль у побудові справедливого та гуманітарного суспільства. Вони допомагають збереженню та зміцненню соціального капіталу, сприяють розвитку громадської свідомості та взаємодії між громадами та організаціями. Платформи створюють мостик між тими, хто хоче допомагати, і тими, хто потребує допомоги, сприяючи виникненню справжньої солідарності та емпатії.

Загалом, платформи для благодійних пожертвувань і волонтерської діяльності є невід'ємною частиною сучасного суспільства, яке прогресує і розвивається. Вони мають великий потенціал для зміни світу до кращого, залучаючи більше людей до активної участі у благодійних справах та волонтерстві. Розвиток цих платформ у напрямку інноваційних технологій, глобальної співпраці та прозорості допоможе зробити їх ще більш ефективними та впливовими в боротьбі з соціальними проблемами і підтримці тих, хто потребує допомоги.

Науково-технічна новизна та значення основних положень у сфері платформ для благодійних пожертвувань і волонтерської діяльності не можуть бути недооцінені. Новітні технології та інноваційні підходи грають важливу роль у покращенні ефективності, прозорості та впливу таких платформ. Основні положення теми засвідчують наступні науково-технічні розв’язки:

1. Використання цифрових технологій: Сучасні платформи для благодійних пожертвувань і волонтерської діяльності використовують розрізнені технології, такі як мобільні додатки, онлайн-платформи, соціальні медіа та електронні системи платежів. Це забезпечує зручність, доступність та масштабованість благодійних і волонтерських ініціатив. Крім того, використання штучного інтелекту та аналітики даних дозволяє здійснювати більш ефективне спрямування ресурсів і планування проектів.
2. Краудфандинг і мікропожертви: З'явлення платформ краудфандингу (збору коштів від широкої громадськості) та мікропожертв (невеликих сум пожертв) відкриває нові можливості для залучення фінансової підтримки. Люди можуть стати активними учасниками благодійних ініціатив навіть з невеликими фінансовими можливостями, що робить благодійність більш доступною та демократичною.
3. Соціальні мережі та вплив: Використання соціальних мереж як засобу поширення інформації та мобілізац ації громадської підтримки має велике науково-технічне значення. Соціальні мережі дозволяють швидко поширювати інформацію про благодійні кампанії, викликати емоційну реакцію у користувачів та залучати їх до участі у благодійних заходах. Крім того, аналіз даних з соціальних мереж дозволяє виявляти тенденції та патерни в поведінці донорів та волонтерів, що допомагає удосконалювати стратегії залучення та утримання учасників.
4. Розширена реальність (AR) та віртуальна реальність (VR): Використання AR та VR у благодійних платформах відкриває нові можливості для взаємодії та залучення донорів та волонтерів. Ці технології дозволяють створювати іммерсивні віртуальні середовища, де люди можуть дізнатися більше про проблему або проект, побачити його вплив та пережити емоції, пов'язані з благодійною діяльністю. Це сприяє більш глибокому зв'язку між учасниками та збільшує шанси на залучення пожертв та волонтерів.
5. Блокчейн технології: Використання блокчейн технологій в платформах для благодійних пожертвувань забезпечує високий рівень безпеки, прозорості та достовірності фінансових операцій. Блокчейн дозволяє забезпечити слідкування за кожною транзакцією та гарантує, що кошти дійсно направляються на потреби благодійних проектів. Це збільшує довіру учасників та сприя є більшій прозорості та відкритості в управлінні благодійними коштами.
6. Глобальна співпраця та взаємодія: Одним із значних науково-технічних розв'язків є можливість побудови платформ, які об'єднують благодійні організації, волонтерські групи та донорів з усього світу. Це створює можливість для глобальної співпраці, обміну досвідом та ресурсами, а також координації дій у вирішенні глобальних проблем. Така співпраця дозволяє швидше та ефективніше реагувати на кризові ситуації та гуманітарні катастрофи, а також сприяє обміну найкращими практиками та інноваціями.
7. Застосування аналітики даних: Науково-технічні розв'язки також полягають у використанні аналітики даних для покращення стратегій залучення, утримання та залучення ресурсів. Аналіз даних дозволяє зрозуміти потреби та уподобання донорів, прогнозувати тенденції та ефективно планувати благодійні проекти. Використання машинного навчання та інтелектуальних алгоритмів допомагає автоматизувати процеси та забезпечити персоналізований підхід до донорів та волонтерів.
8. Відкриті дані та стандартизація: Важливим аспектом є науково-технічна стандартизація платформ для благодійних пожертвувань, що сприяє впровадженню відкритих стандартів обміну даними, взаємодії з іншими системами та спільній розробці рішень.

Проблематика, пов'язана з платформами для благодійних пожертвувань і волонтерської діяльності, включає такі аспекти:

1. Низька прозорість та недостатня відкритість: Багато благодійних організацій стикаються з проблемою недостатньої прозорості та відкритості у використанні благодійних коштів. Це може спричиняти недовіру у донорів та обмежувати їх бажання здійснювати пожертвування. Однак, розвиток платформ дозволяє впроваджувати технології блокчейну та використовувати прозорі фінансові звіти, що сприяє більшій довірі і забезпечує відкритість у використанні коштів.
2. Обмежені ресурси та неефективне управління: Багато благодійних організацій та волонтерських груп стикаються з проблемою обмежених ресурсів та неефективного управління проектами. Проте, платформи надають можливості для залучення більшої кількості донорів та волонтерів, а також допомагають в управлінні проектами за допомогою інструментів планування, відстеження та звітності.
3. Брак комунікації та взаємодії: Відсутність ефективної комунікації та взаємодії між благодійними організаціями, волонтерами та донорами може ускладнювати розвиток проектів та обмежувати їх потенціал. Але завдяки платформам, таким як соціальні медіа та онлайн-форуми, забезпечується зручна комунікація та спільна робота між усіма учасниками, що сприяє обміну ідеями, ресурсами та кращими практиками. Крім того, використання новітніх технологій, таких як відеозв'язок і чат-боти, полегшує комунікацію та спілкування між учасниками платформи, забезпечуючи швидкий доступ до інформації та підтримку.
4. Безпека та захист даних: Однією з головних проблем є забезпечення безпеки та захисту даних у платформах для благодійних пожертвувань. Зростання кількості трансакцій та обміну конфіденційною інформацією вимагає високого рівня кібербезпеки. Тому розробники платформ активно працюють над впровадженням шифрування даних, двофакторної аутентифікації та інших заходів для захисту особистої інформації користувачів.
5. Залучення нових аудиторій: Однією з важливих задач є залучення нових аудиторій до благодійних платформ. Це може включати молодь, малозабезпечені групи, людей з різних соціокультурних середовищ тощо. Для досягнення цієї мети, розробники платформ розробляють інтерфейси, які є зрозумілими, доступними та привабливими для різних груп користувачів. Також активно впроваджуються механізми для залучення та стимулювання нових донорів та волонтерів.

Загалом, науково-технічні розв'язки у цій області спрямовані на поліпшення ефективності, прозорості, комунікації та безпеки в платформах для благодійних пожертвувань і волонтерської діяльності. Вони допомагають побудувати сильні стійкий екосистема благодійництва, яка сприяє розвитку соціальних змін, підтримці уразливих груп населення, реалізації гуманітарних проектів та вирішенню нагальних глобальних проблем. Ці науково-технічні розв'язки пропонують нові можливості для залучення широкого кола учасників, покращення ефективності ресурсного використання, забезпечення прозорості та довіри, а також сприяють взаємодії, обміну інформацією та співпраці між всіма сторонами, що долучаються до благодійних ініціатив. Завдяки цим розв'язкам, благодійність стає більш доступною, інноваційною та впливовою, допомагаючи побудувати краще майбутнє для всіх.

**РОЗДІЛ 1. ПЛАТФОРМА ДЛЯ БЛАГОДІЙНИХ ПОЖЕРТВУВАНЬ ТА ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ**

# 1.1. Платформа для благодійних пожертвувань та волонтерської діяльності. Поняття та види

Платформа для благодійних пожертвувань та волонтерської діяльності – це програмне забезпечення, яке дає змогу створювати волонтерські оголошення та переглядати їх , використовуючи базу даних (БД).

База даних визначають як середовище зберігання даних, які організовані за визначеними правилами. Існують різні критерії класифікації баз даних. За моделлю організації даних можна виділити такі типи:

* Ієрархічна база даних, яка складається з об’єктів різних рівнів та має зв’язки “предок-нащадок”.
* Мережна база даних, яка має схожу структуру з ієрархічною, але об’єкт може мати декілька предків.
* Реляційна база даних, що зберігає дані у вигляді таблиць та є найпоширенішою моделлю серед систем керування базами даних (СКБД).
* Об’єктно-орієнтована база даних, де дані відображають у вигляді об’єктів.

За розміщенням даних, бази даних поділяють на локальні (централізовані) та розподілені, які можуть бути виконані на різних платформах, таких як веб, стаціонарні системи, мобільні додатки та системи для влаштованих пристроїв.

# 1.2. Приклади існуючих платформ для благодійних пожертвувань та волонтерської діяльності.

# 1.2.1. GoFundMe

GoFundMe – це онлайн-платформа збору коштів для різних цілей, від медичних потреб до благодійності та технологій. Заснована в 2010 році, платформа допомагає людям зібрати кошти на проекти та цілі, що мають значення для них та їхніх близьких.



Рис. 1.1. Логотип GoFundMe.

За допомогою GoFundMe ([Рис. 1.1](#Рис_1_1)) користувачі можуть створювати особисті сторінки та розповідати про свої історії, мету та потребу в коштах. Люди можуть додавати фотографії та відео, щоб краще розповісти про своє запитання на допомогу. Користувачі також можуть ділитися своїми сторінками з друзями та родичами, які можуть пожертвувати на їхню підтримку.

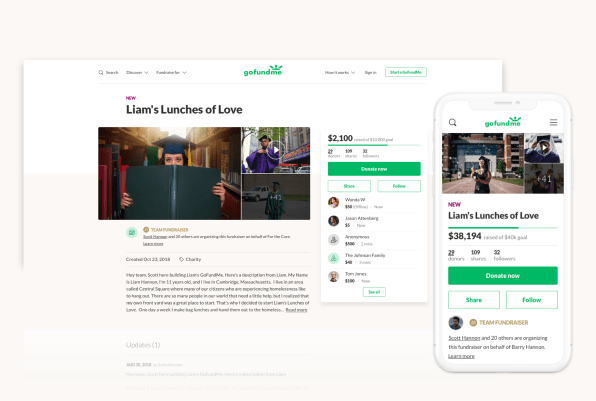


Рис. 1.2. Інтерфейс GoFundMe.

Одна з найбільш важливих функцій GoFundMe ([Рис. 1.2](#Рис_1_2)) полягає в тому, що користувачі можуть збирати гроші швидко та ефективно, не залежно від того, де вони живуть. Таким чином, люди можуть отримати допомогу та підтримку з усього світу.

GoFundMe також є корисною для благодійних організацій та некомерційних організацій, які можуть створювати свої власні сторінки та збирати кошти на благодійні проекти та інші потреби. Це може допомогти підвищити увагу до їхніх проблем та підтримати їхню місію.

# 1.2.2 DonorsChoose

# DonorsChoose – це онлайн-платформа для збору коштів на проекти, які пов'язані з освітою. Заснована в 2000 році, платформа допомагає вчителям з США створювати проекти та збирати кошти на придбання матеріалів та обладнання для своїх класів.

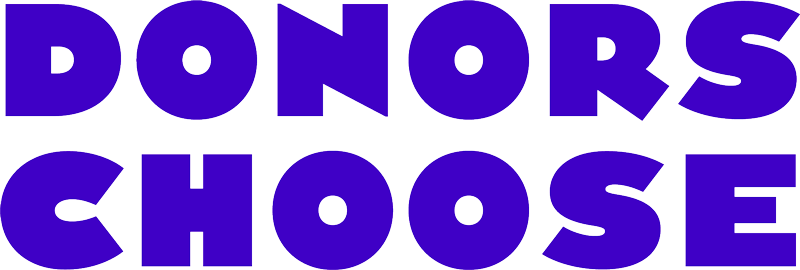


Рис. 1.3. Логотип DonorsChoose.

За допомогою DonorsChoose ([Рис. 1.3](#Рис_1_3)) вчителі можуть створювати особисті сторінки та описувати свої проекти та потреби. Люди можуть знайти проекти, які їм цікаві та пожертвувати на їхню підтримку. Користувачі також можуть ділитися своїми сторінками з друзями та родичами, щоб залучити ще більше підтримки.

Одна з головних переваг DonorsChoose полягає в тому, що це дає змогу вчителям отримати необхідні матеріали та обладнання для своїх класів, які можуть допомогти покращити якість освіти та навчання учнів. Крім того, це допомагає підвищити зацікавленість учнів та збільшити їхню мотивацію до навчання.

DonorsChoose також є корисною для тих, хто хоче пожертвувати на освітні проекти та допомогти вчителям та учням отримати необхідні ресурси. Це може допомогти зробити позитивний внесок у майбутнє дітей та підтримати освітню місію вчителів.

Узагалі, DonorsChoose є потужним інструментом для збору коштів та підтримки освітніх проектів. Це дає можливість вчителям отримати необхідні матеріали та обладнання, а учням - якісну освіту та навчання.

На [рис. 1.4](#Рис_1_4) можна побачити сторінку DonorsChoose.

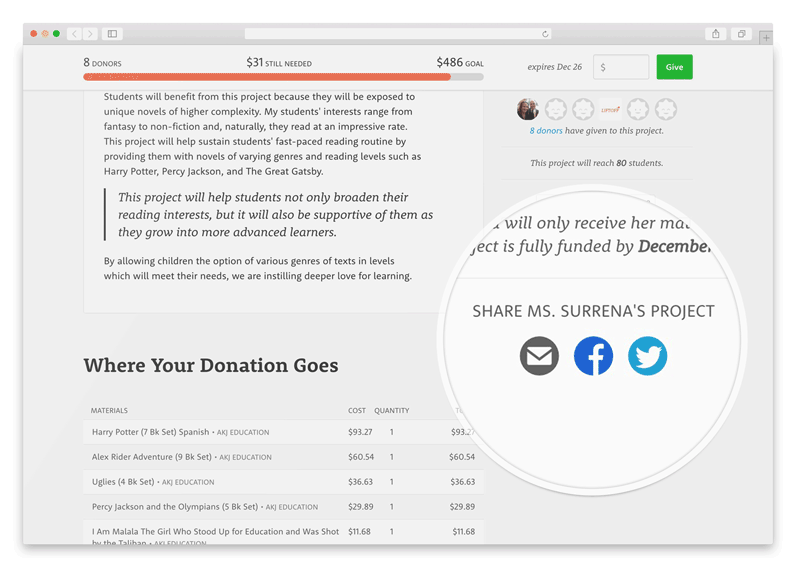


Рис. 1.4. Інтерфейс DonorsChoose.

# 1.2.3 VolunteerMatch

VolunteerMatch – це онлайн-платформа, яка об’єднує волонтерів та неприбуткові організації. Заснована в 1998 році, платформа допомагає знаходити волонтерські можливості у багатьох країнах світу, допомагаючи організаціям та волонтерам знаходити один одного.



Рис. 1.5. Логотип VolunteerMatch.

За допомогою VolunteerMatch ([Рис. 1.5](#Рис_1_5)) неприбуткові організації можуть розміщувати вакансії для волонтерів та знаходити людей, які готові надати свою безоплатну допомогу. Волонтери також можуть шукати відповідні вакансії на платформі та зв'язуватися з організаціями. На [рис. 1.6](#Рис_1_6) ми можемо бачити приклад сторінки VolunteerMatch.

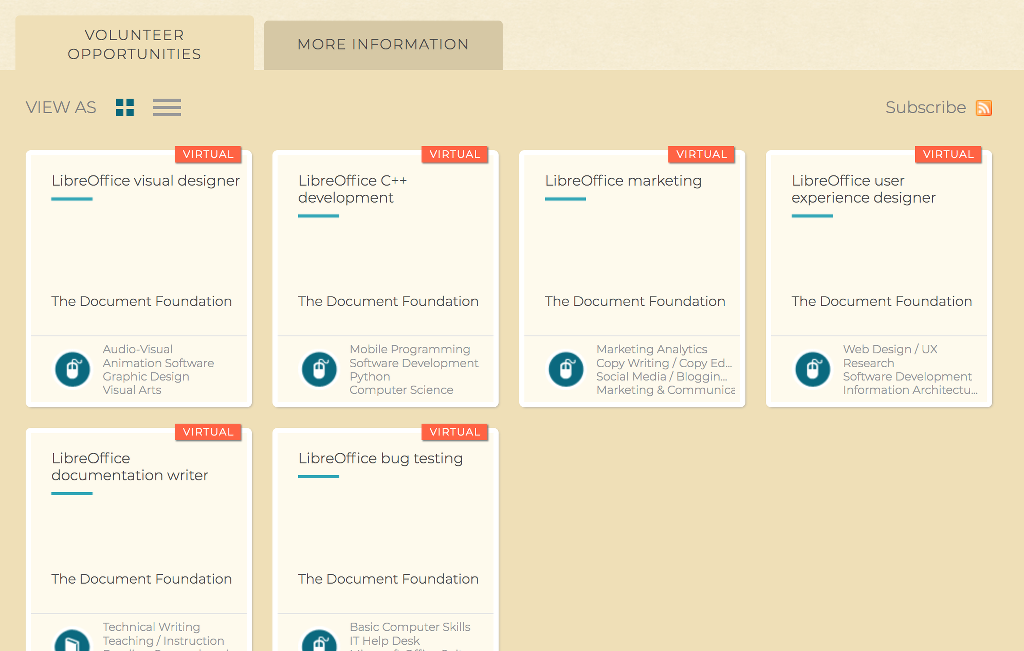


Рис. 1.6. Інтерфейс VolunteerMatch.

Одна з головних переваг VolunteerMatch полягає в тому, що волонтери можуть знайти відповідну волонтерську роботу, яка відповідає їхнім інтересам та здібностям. Також це допомагає неприбутковим організаціям знаходити відповідних волонтерів, які можуть допомогти їм виконувати їхні місії.

VolunteerMatch також є корисною для тих, хто хоче зайнятися волонтерською діяльністю та зробити позитивний внесок у свій спільноту та світ. Це може допомогти відчути себе корисним, знайти нових друзів та розвиватися в процесі надання допомоги.

Загалом, VolunteerMatch є потужним інструментом для пошуку волонтерських можливостей та збільшення відчуття корисності та зацікавленості в наданні допомоги. Це дає змогу організаціям та волонтерам знаходити один одного та виконувати важливу місію для спільної користі.

# 1.2.4 GlobalGiving

GlobalGiving ([Рис. 1.7](#Рис_1_7)) - це онлайн-платформа для збору коштів, яка забезпечує зв'язок між неприбутковими організаціями та людьми, які бажають надати допомогу. Заснована в 2002 році, GlobalGiving дозволяє організаціям розміщувати свої проекти на платформі та збирати кошти від людей, які хочуть пожертвувати на підтримку цих проектів.



Рис. 1.7. Логотип GlobalGiving.

Організації можуть зареєструватися на платформі та створювати сторінки для своїх проектів, де вони можуть ділитися інформацією про свою місію, цілі та плани. Люди з усього світу можуть переглядати проекти та пожертвувати на ті, які вони хочуть підтримати.

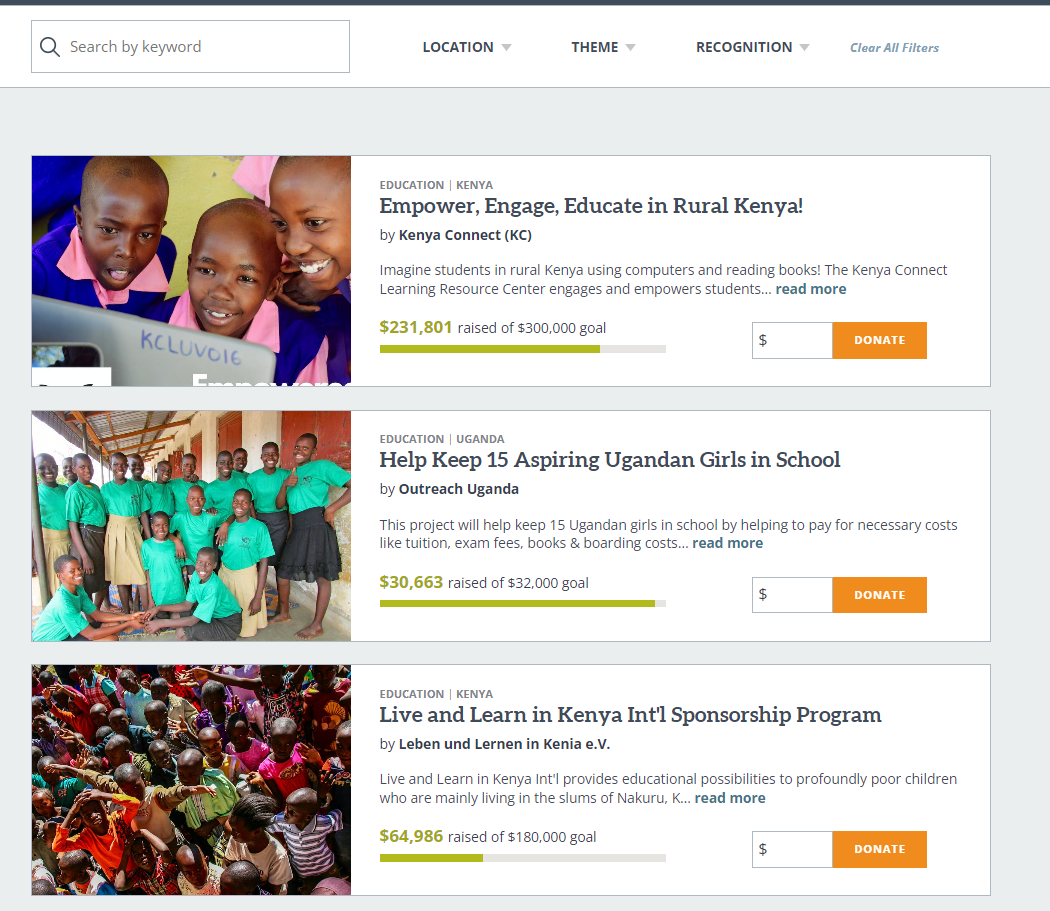


Рис. 1.8. Інтерфейс GlobalGiving.

Одна з переваг GlobalGiving полягає в тому, що платформа пропонує можливості збору коштів для проектів у більш ніж 170 країнах світу. Крім того, організації можуть отримувати фінансову підтримку не тільки від індивідуальних спонсорів, але й від компаній та інших організацій, які бажають підтримати благодійні проекти.

Ще одна перевага GlobalGiving полягає в тому, що платформа дозволяє неприбутковим організаціям отримувати доступ до різних ресурсів та інструментів для розвитку та підвищення ефективності їхньої діяльності. Наприклад, організації можуть отримувати курси з фінансового менеджменту, маркетингу та фондової роботи.

На [рис. 1.8](#Рис_1_8) ми можемо бачити сторінку з GlobalGiving.

# 1.2.5 Patreon

Patreon ([Рис. 1.9](#Рис_1_9)) – це онлайн-платформа, яка допомагає творчим людям отримувати гроші від своїх шанувальників, щоб вони могли продовжувати свою роботу. На платформі є багато творців від музикантів, художників та блогерів до письменників та журналістів.



Рис. 1.9. Логотип Patreon.

Платформа дозволяє творцям створювати різні рівні підписки, на які можуть підписатися їхні шанувальники. Кожен рівень підписки має свої вигоди та винагороди, які творець може пропонувати своїм підписникам, наприклад, ексклюзивний контент, ранній доступ до нових робіт або особистий доступ до творця.

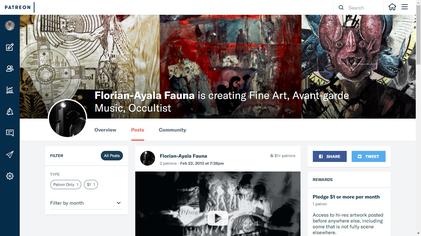


Рис. 1.10. Інтерфейс Patreon.

Творці можуть встановлювати місячну плату за кожен рівень підписки або встановлювати мінімальний поріг для платежів від своїх шанувальників. Також є можливість зробити одноразовий внесок на розвиток творчості певного автора.

Patreon забезпечує творцям можливість заробляти на своїй роботі, зокрема на зйомках відео, виготовленні музики, написанні книг чи роботі над іншими проектами. Платформа дає можливість творцям зосередитися на творенні контенту, не витрачаючи час на пошук спонсорів та збір коштів.

Підтримка платформи Patreon надає творцям можливість мати стабільний прибуток та продовжувати свою творчість, що допомагає забезпечити стабільність та розвиток культурної та творчої сфери.

На [рис.1.10](#Рис_1_10) ми можемо бачити сторінку з Patreon.

# 1.3 Збереження даних

Під час роботи з великими обсягами даних потрібно забезпечити їх структуризоване зберігання. Зберігання може бути локальним або віддаленим, на сервері, і для цього використовують різні системи, такі як Block storage, File storage, Object storage та Key-value storage. Щоб вибрати оптимальну систему зберігання, потрібно провести тести та перевірити певні параметри в різних умовах.

Основні характеристики, які потрібно врахувати під час вибору системи зберігання:

• Надійність у роботі та відмовостійкість.

• Швидкість відновлення даних за виникнення проблем.

• Продуктивність, яка відповідає вашим запитам та потребам.

• Консистентність даних для забезпечення їх коректності та цілісності.

**1.3.1 JSON**

[JSON](https://www.json.org/json-en.html) [13] є текстовим форматом обміну даними між комп'ютерами, який може бути зрозумілим людиною і дозволяє описувати об'єкти та інші структури даних. Основне призначення JSON полягає в передачі структурованої інформації через мережу, завдяки процесу серіалізації. Формат був розроблений і популяризований Дугласом Крокфордом і головним чином використовується для написання веб-програм, зокрема технології AJAX. У порівнянні з XML, JSON займає менше місця, дозволяє складні структури в атрибутах та прямо інтерпретується за допомогою [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript) [11] в об'єкти. Це робить його більш придатним для серіалізації складних структур. JSON також підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші. У контексті веб-застосунків JSON доречний для обміну даними між браузером і сервером (AJAX) або між серверами (програмні HTTP-інтерфейси).

# 1.3.2 База даних

БД – це набір даних, які організовані відповідно до певної концепції, що описує їх характеристики та зв’язки між ними, і містить щонайменше одну область застосування (відповідно до стандарту ISO/IEC 2382:2015). Загалом, база даних включає схеми, таблиці, представлення, збережені процедури та інші об’єкти, і її дані організовані згідно з моделлю організації даних. Сучасні бази даних також можуть містити засоби для обробки даних та їх опис.

БД – це структурована набір даних, що відображає стан об'єктів певної предметної області та зв'язки між ними. СКБД – це програма, яка забезпечує можливість створення БД та виконання операцій з даними, які в ній зберігаються. СКБД не є окремим видом програмного забезпечення, а лише документи, з якими оперують спеціальні прикладні програми.

Загалом будь-який впорядкований набір даних можна вважати базою даних, наприклад, паперову картотеку про працівників підприємства в відділі кадрів. Застосунки для роботи з базами даних є дуже поширеними прикладними програмами, і сучасні інформаційні системи використовують СКБД для забезпечення роботи з базами даних. СКБД – це система, яка ґрунтується на програмних та технічних засобах, забезпечує визначення, створення, маніпулювання, контроль, керування та збереженням баз даних.

# ВИСНОВОКИ ДО РОЗДІЛУ 1

Для розробки платформи для благодійних пожертвувань та волонтерської діяльності насамперед необхідно провести аналіз існуючих додатків на ринку. У розділі розглянуто найпопулярніші системи, які можна знайти, використовуючи пошуковик Google. Усі системи мають свої переваги та неділоки, деякі платформи мають доволі застарілий інтерфейс, що робить не дуже зручним їх використання, але на томість платформа Patreon має дуже хороший пророблений інтерфейс та хорошу вітальну сторінку. Усі перелічені платформи створені в форматі веб-додатків, що є логічно, оскільки це досить зручно як для людини, яка вирішить зробити пожертвування, так і для самих авторів оголошень.

Для реалізації проекту прийнято рішення використовувати СКБД [MS SQL](https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16) [8]. Для початку роботи з базою даних її необхідно створити з новими таблицями та налаштувати зв’язки між ними.

# РОЗДІЛ 2. ВИКОРИСТАНІ ЗАСОБИ РОЗРОБКИ

Комп’ютери є універсальними інструментами, які здатні здійснювати складні обчислення, полегшувати повсякденні завдання, автоматизувати нудні процеси та дають змогу легко обмінюватись інформацією, незалежно від місцязнаходження. Проте, комп’ютери не мають “інтелекту”, як люди, що створює мовний бар’єр між користувачем та машинами. Для того, щоб комп’ютер виконав певну дію, необхідно подати набір інструкцій, створених людиною, оскільки сам комп’ютер не може нічого зробити самостійно.

# 2.1. ASP.NET

[ASP.NET](https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0.) [1] – це один з найпопулярніших фреймворків для створення веб-додатків. Він розробляється Microsoft і використовується для створення веб-сайтів, веб-додатків та веб-сервісів. ASP.NET ([рис. 2.1](#Рис_2_1)) базується на платформі [.NET](https://learn.microsoft.com/en-us/dotnet/) [2] і використовує мови програмування C# або Visual Basic .NET для розробки веб-додатків.



Рис. 2.1. Логотип ASP.NET.

ASP.NET має багато переваг, серед яких варто виокремити:

1) Швидкість та продуктивність – ASP.NET забезпечує високу продуктивність завдяки використанню принципу компіляції на етапі встановлення та підтримки кешування на сервері.

2) Безпека – ASP.NET забезпечує безпеку веб-додатків завдяки вбудованим механізмам автентифікації, авторизації та оброблення помилок. Крім того, він має вбудований механізм захисту від атак Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

3) Масштабованість – ASP.NET забезпечує масштабованість веб-додатків завдяки використанню розподіленої архітектури та можливості використання кластерів серверів.

4) Легкість використання – ASP.NET має широкий вибір інструментів для розролення веб-додатків, включаючи інтегровану розробку середовища (IDE) Visual Studio, що дає змогу швидко створювати веб-додатки.

5) Підтримка – Microsoft надає широку підтримку ASP.NET, включаючи документацію, навчальні матеріали та форуми спільноти, що дозволяє розробникам швидко вирішувати проблеми.

ASP.NET дає моєливість розробникам створювати веб-додатки різних типів, включаючи веб-сайти, веб-додатки та веб-сервіси. Для цього ASP.NET пропонує різні типи проектів, такі як ASP.NET Web Forms, ASP.NET MVC та ASP.NET Web API. Кожен із цих типів проектів має свої особливості та спрощує розробку веб-додатків певного типу.

ASP.NET Web Forms – це тип проекту, який базується на концепції сторінок, тобто кожна сторінка має своє представлення та код, що відповідає за її функціональність. ASP.NET Web Forms дає змогу швидко створювати веб-сайти та веб-додатки з використанням компонентів, таких як текстові поля, кнопки та інші.

ASP.NET MVC – це тип проекту, який базується на концепції розділення сторінки на моделі, представлення та контролери. Кожен із цих компонентів має свою функцію під час оброблення запитів та відображення відповіді. ASP.NET MVC дає змогу швидко створювати веб-додатки з більш гнучким управлінням, представленням та логікою додатку.

ASP.NET Web API - це тип проекту, який дозволяє створювати веб-сервіси з використанням технологій RESTful. Це дає можливість розробникам створювати веб-сервіси з використанням HTTP-протоколу та стандартних форматів даних, таких як JSON та XML.

Для розробки веб-додатків з використанням ASP.NET розробникам потрібно мати знання мови програмування C# або Visual Basic .NET та веб-технологій, таких як HTML, CSS та JavaScript. Крім того, розробникам слід мати знання про архітектуру веб-додатків та підходи до розроблення з використанням ASP.NET.

ASP.NET – це потужний та популярний фреймворк для розробки веб-додатків, який забезпечує швидкість, безпечні та масштабовані веб-додатки. ASP.NET має вбудовану підтримку для багатьох інших технологій, таких як Entity Framework, SignalR, Razor та інших. Це дає можливість розробникам швидко створювати веб-додатки з використанням сучасних технологій та підходів.

Одна з головних переваг ASP.NET полягає у тому, що він забезпечує високу продуктивність та ефективність роботи з базами даних. ASP.NET має вбудовану підтримку для баз даних, таких як Microsoft SQL Server, та може використовувати інші бази даних, такі як MySQL та Oracle. Крім того, ASP.NET забезпечує високу безпеку веб-додатків, шляхом використання різних методів, таких як автентифікація та авторизація, валідація введення користувача та захист від атак типу Cross-Site Scripting (XSS) та SQL Injection.

Загалом, ASP.NET – це потужний та високопродуктивний фреймворк для розробки веб-додатків, який забезпечує розробникам широкі можливості для створення різноманітних веб-додатків та веб-сервісів. Будучи частиною платформи .NET, ASP.NET дає змогу розробникам використовувати різноманітні засоби, бібліотеки та сервіси для створення високоякісних веб-додатків.

# 2.2. Entity Framework

Entity Framework ([рис. 2.2](#Рис_2_2)) – це технологія об’єктно-орієнтованого доступу до даних (ORM), яка дозволяє розробникам працювати з базами даних, як з об’єктами у програмі. Це забезпечує високу продуктивність та ефективність роботи з базами даних та зменшує кількість коду, необхідного для створення та роботи з базами даних.



Рис. 2.2. Логотип EF Core.

[Entity Framework](https://learn.microsoft.com/en-us/ef/core/) [3] має вбудовану підтримку різних баз даних, включаючи Microsoft SQL Server, MySQL та PostgreSQL. Він дає можливість розробникам працювати з даними за допомогою LINQ (Language Integrated Query) – розширення мови C#, яке дає змогу виконувати запити до бази даних з використанням звичайних мовних конструкцій.

Entity Framework має кілька різновидів:

1) EF Core – це платформонезалежний, легкий та швидкий виконавчий код, який працює на Windows, Linux та Mac.

2) EF6 – це більш старіший варіант EF, який працює тільки на платформі .NET Framework та має менше можливостей, ніж EF Core.

Entity Framework забезпечує можливість розробникам створювати різні типи запитів до баз даних, такі як вибірка, вставка, оновлення та видалення, використовувати транзакції для збереження даних у БД та забезпечує підтримку роботи з даними багатьох користувачів.

Entity Framework дає змогу використовувати підхід “Code First”, коли розробники можуть створювати моделі даних на основі коду класів, а потім за допомогою міграцій змінювати структуру бази даних. Крім того, EF дає можливість використовувати підхід “Database First”, коли розробники можуть створювати моделі даних на основі структури бази даних.

Підсумовуючи, Entity Framework – є потужним та універсальним інструментом для роботи з базами даних, який дає можливість розробникам швидко та ефективно працювати з даними у своїх програмах. Він дає змогу створювати складні запити до баз даних та використовувати міграції для зміни структури баз даних, що робить роботу з базами даних зручнішою та простішою.

Entity Framework також підтримує підключення до баз даних з допомогою різних провайдерів, що дозволяє розробникам використовувати той провайдер, який найбільш підходить для їх потреб.

Однією з великих переваг Entity Framework є можливість використовувати його у складі ASP.NET, що дає змогу розробникам створювати високопродуктивні веб-додатки зі зручним доступом до баз даних. Крім того, Entity Framework має декілька інструментів для покращення продуктивності, таких як кешування та пакетні операції. Вони дають можливість збільшити швидкість роботи з базами даних та зменшити кількість запитів до них, що позитивно впливає на продуктивність та ефективність додатка в цілому.

Загалом, Entity Framework є потужним та корисним інструментом для роботи з базами даних, який дає можливість розробникам швидко та легко створювати високопродуктивні та надійні програми. Він має багато можливостей та підтримує різні провайдери баз даних, що дає змогу розробникам використовувати його в різних проектах та у різних середовищах.

# 2.3. React.JS

[React.js](https://legacy.reactjs.org/docs/getting-started.html) [4] – це бібліотека для створення інтерфейсів користувача, що дозволяє розробникам створювати динамічні веб-сторінки та веб-додатки. Вона розроблена Facebook в 2011 році і відтоді стала дуже популярною у розробці фронтенду.

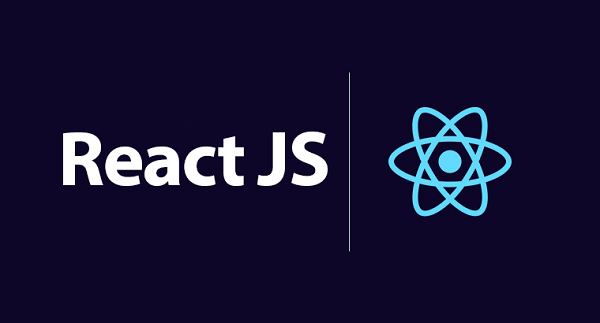


Рис. 2.3. Логотип React.JS.

React.js ([Рис. 2.3](#Рис_2_3)) працює з компонентами, які є самодостатніми частинами веб-сторінки. Кожен компонент містить [HTML-код](https://developer.mozilla.org/en-US/docs/Web/HTML) [9], [CSS-стилі](https://developer.mozilla.org/en-US/docs/Web/CSS) [10] та JavaScript-логіку. За допомогою компонентів, розробники можуть створювати складні інтерфейси, розділяючи їх на простіші блоки.

Один з головних принципів React.js – це “одне джерело правди” (single source of truth). Це означає, що стан компонента повинен бути збережений в одному місці, а не розподілений по всій програмі. Це забезпечує простіше управління станом та запобігає появі непередбачуваної поведінки.

Ще одним важливим аспектом React.js є використання віртуального DOM (Document Object Model). Він є копією реального DOM, але не містить реальних елементів веб-сторінки. Віртуальний DOM зберігає стан всіх компонентів та забезпечує ефективнішуу роботу з реальним DOM. Коли стан компонента змінюється, React.js змінює тільки необхідні елементи реального DOM, а не всю сторінку.

Ще одним важливим аспектом React.js є можливість використовувати JSX-синтаксис, що дає змогу вбудовувати JavaScript-код в HTML-код. Це забезпечує простішу та зрозумілішу розробку інтерфейсу.

React.js також підтримує широкий діапазон інструментів для розробки, включаючи багато корисних бібліотек, таких як [Redux](https://redux.js.org/) [5], React Router та Material-UI. Redux дає змогу зберігати стан додатка в одному місці, що спрощує його управління. React Router – це бібліотека для роутингу веб-додатків, що дає можливість змінювати вміст сторінки без перезавантаження сторінки, а Material-UI – це бібліотека для розроблення стильних та сучасних інтерфейсів.

React.js також працює дуже добре з іншими технологіями, такими як Node.js та Express.js. Node.js дає змогу розробникам створювати серверну частину веб-додатків, а Express.js – це фреймворк для розроблення серверних додатків на Node.js. За допомогою React.js та Node.js можна створювати повноцінні веб-додатки з різними можливостями, такими як авторизація користувачів, відправка електронної пошти, робота з базами даних та багато іншого.

React.js має також велику спільноту розробників, яка допомагає розвивати та підтримувати цю бібліотеку. У спільноті розробників React.js є багато корисних матеріалів для вивчення та розробки, таких як документація, курси, відеоуроки, блоги та інші ресурси.

У підсумку, React.js – це потужна технологія для розробки інтерфейсів користувача, яка дає моєливість створювати динамічні та привабливі веб-додатки. Вона підтримує широкий діапазон інструментів та має велику спільноту розробників, що допомагає вивчати та розвивати цю технологію. Якщо ви шукаєте потужну технологію для розробки фронтенду, React.js може бути ідеальним вибором для нас.

# 2.4. Redux.JS

Redux.js ([Рис. 2.4](#Рис_2_4)) - це бібліотека для керування станом додатка у React-додатках. Вона була розроблена, щоб спростити управління станом додатка та забезпечити однорідність в управлінні станом для різних компонентів React-додатків.



Рис. 2.4. Логотип Redux.JS.

Redux заснований на патерні Flux, який розроблений у Facebook. Патерн Flux включає односторонню потік даних, де дії (actions) виконуються з компонентів React та пересилаються в диспетчер (dispatcher), який відповідає за розподіл цих дій до магазину (store). Магазин містить стан додатка та логіку, яка відповідає за зміну стану за допомогою редукторів (reducers), які виконуються на кожній дії та змінюють стан додатка відповідно до логіки, яка описана в них. Компоненти React можуть підписуватись на зміни стану магазину та оновлювати свій стан відповідно до змін.

Redux дає змогу зберігати стан додатка в одному місці, що спрощує його управління та підтримку. Також він дає можливість зберігати історію змін стану, що дає змогу повернутись до будь-якої попередньої версії стану додатка. Це можна використати для налагодження та відлагодження додатка.

Redux також дає можливість розбити стан додатка на окремі частини та зберігати їх в окремих редукторах. Це сприяє спрощенню логіки додатка та підтримки коду. Крім того, Redux дає змогу розробникам використовувати бібліотеки middleware для додавання додаткової логіки до процесу зміни стану додатка.

Redux має велику спільноту розробників, що допомага є вирішувати проблеми та надавати підтримку для розробників, що робить його ще більш привабливим для використання.

Redux можна використати для будь-якого типу додатка, від простих до складних, забезпечуючи однорідний підхід до управління станом додатка. Зокрема, Redux дає змогу писати тестований та декларативний код, що забезпечує більшу надійність та стабільність додатку.

Іншою корисною можливістю Redux є можливість розширення за допомогою плагінів та middleware. Middleware є функціями, які дають можливість додавати додаткову логіку до процесу зміни стану додатка, наприклад, логування або зберігання історії змін. Це дає змогу розробникам зберігати чистоту та простоту коду додатка, знаючи, що додаткова логіка може бути додана у будь-який час.

Redux також інтегрується добре з React-додатками, що дає змогу розробникам зберігати стан додатка та логіку в Redux, а компоненти React можуть підписуватись на зміни стану та оновлювати свій стан відповідно до змін. Це дає моживість розробникам використовувати привабливі функції React, такі як контекст та власники, та використовувати Redux для управління станом.

Redux є потужною бібліотекою для управління станом додатка в React-додатках. Він забезпечує простий та однорідний підхід до управління станом, дозволяє зберігати стан додатка в одному місці та підтримує розширення за допомогою плагінів та middleware. Redux інтегрується добре з React-додатках та дає можливість розробникам використовувати функціональність React, щоб створювати привабливі та потужні додатки. Крім того, Redux забезпечує можливість писати тестований та декларативний код, що допомагає зберегти надійність та стабільність додатку.

Однак, на вибір Redux впливає складність додатка та його потреби в управлінні станом. Якщо додаток є простим та має небагато стану, можливо, не потрібно використовувати Redux. Однак, якщо додаток має складні взаємодії між компонентами та багато стану, то Redux може бути потужним інструментом для управління станом.

У будь-якому випадку, якщо ви використовуєте Redux, важливо мати на увазі, що він не є срібною кулею. Іноді розробники можуть піти занадто далеко з використанням Redux, створивши занадто багато додаткової логіки та middleware. Тому важливо зберігати баланс між простотою та потужністю, використовуючи Redux у доречних випадках та зберігаючи код додатку простим та легким для розуміння.

Загалом, Redux є потужним інструментом для управління станом додатка, який дозволяє зберігати стан в одному місці та забезпечує декларативний та тестований код. Він інтегрується добре з React-додатками та дозволяє розробникам створювати привабливі та потужні додатки. Однак, використовуйте Redux розумно, зберігаючи баланс між простотою та потужністю, щоб забезпечити стабільність та надійність додатку.

# 2.5. Visual Studio

Набір інструментів Microsoft Visual Studio призначений для створення програмного забезпечення, включаючи планування, розробку інтерфейсу користувача, написання коду, тестування, налагодження, аналіз якості коду та продуктивності, розгортання у клієнтських середовищах та збір телеметрії щодо використання. Всі ці інструменти можна знайти в одному інтегрованому середовищі розробки (IDE) – Visual Studio. Вони були створені з метою максимально ефективної спільної роботи.



Рис. 2.5. Логотип Visual Studio.

Visual Studio є універсальним інструментарієм для розробдення різних типів додатків, від простих мобільних додатків і ігор до складних підприємств та центрів обробки даних. Наприклад, ви можете створювати додатки та ігри для Windows, Android та iOS, веб-сайти та веб-служби на основі платформ ASP.NET, JQuery та AngularJS, додатки для різних платформ та пристроїв, таких як Office, Sharepoint, Hololens, Kinect та “Інтернет речей”, а також ігри та графічні додатки для Windows пристроїв, включаючи Xbox, з підтримкою DirectX.

Visual Studio за замовчуванням підтримує C#, C, C++, JavaScript, F# та Visual Basic, і легко інтегрується зі сторонніми додатками, такими як Unity та Apache Cordova, за допомогою розширень, таких як Набір засобів Visual Studio для Unity та Інструменти Visual Studio для Apache Cordova відповідно. Ви можете самостійно розширити Visual Studio, створивши власні інструменти для виконання спеціалізованих завдань.

**2.6. Git**

[Git](https://git-scm.com/doc) [7] – це система контролю версій, яка дає змогу розробникам зберігати та керувати змінами в програмному коді та інших файлах. Вона стала незамінною частиною сучасного програмного розроблення та дає можливість командам розробників ефективно співпрацювати, зберігати історію змін, відслідковувати помилки та швидко відновлювати стан додатків.



Рис. 2.6. Логотип Git.

Git ([Рис. 2.6](#Рис_2_6)) дає змогу зберігати код у віддаленому репозиторії, який можна синхронізувати з локальними копіями. Кожен розробник може створювати свою власну гілку (branch), що дозволяє працювати зі своїми власними змінами, не впливаючи на головний код. Після завершення роботи, зміни можна об’єднати (merge) з головною гілкою, забезпечуючи збереження коду в одному місці та уникнення конфліктів.

Однією з найважливіших функцій Git є можливість повернення до попередніх версій коду. Якщо під час розробки виникають проблеми, Git дозволяє відновити попередню версію коду та продовжити роботу з цієї точки. Крім того, Git дає змогу співпрацювати з іншими розробниками, які можуть вносити зміни до коду та вирішувати проблеми шляхом взаємодії та обговорення.

Git також має багато інших корисних функцій, таких як відслідковування змін та дозвіл на доступ до репозиторію. Він інтегрується з багатьма редакторами коду та іншими інструментами, що дає можливість розробникам легко і ефективно працювати з Git.

Хоча Git може здатися складним на початку, він має дуже добре документовану та розгалужену систему допомоги, що дозволяє вирішувати будь-які проблеми, які можуть виникнути. Багато розробників вважають Git незамінною частиною своїх робочих процесів та рекомендують його для будь-яких проектів.

Однією з найбільших переваг Git є те, що він є відкритою системою, що означає, що ви можете безкоштовно використовувати його та вносити свої власні зміни в систему. Багато відомих компаній, таких як Google, Facebook, Microsoft та інші, використовують Git для своїх проектів та активно співпрацюють зі спільнотою розробників, що дозволяє їм розвивати цю систему контролю версій та покращувати її функціональні можливості.

Загалом, Git є дуже корисним інструментом для будь-якої команди розробників, яка працює над програмними проектами. Він дозволяє розробникам зберігати та керувати змінами в коді, співпрацювати з іншими розробниками, відслідковувати помилки та швидко відновлювати стан додатків. Хоча Git може здатися складним на початку, він має дуже потужну та детально документовану систему, яка дозволяє вирішувати будь-які проблеми, які можуть виникнути. Тому використання Git є необхідним для будь-якого проекту програмного розроблення та допомагає зберігати якість коду та робочий процес команди на високому рівні.

# 2.7. SQL мова

SQL ([Рис. 2.7](#Рис_2_7)) – це мова, яка дає змогу створювати та працювати з реляційними базами даних, які є наборами пов'язаної інформації, збереженої у таблицях.

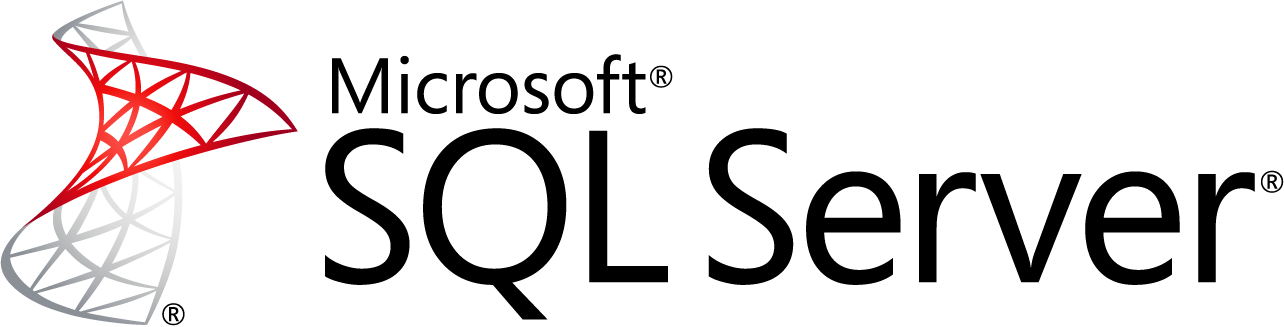


Рис. 2.7. Microsoft SQL Server.

Завдяки постійному зростанню інформаційного простору, з’явилась необхідність у створенні стандартної мови, яка могла б бути використана в різних типах комп’ютерних середовищ. Стандартна мова дає можливість користувачам, які знають один набір команд, використовувати їх для створення, пошуку, зміни та передачі інформації, незалежно від типу комп'ютера.

У нашому все більш і більш взаємопов’язаному комп’ютерному світі, користувач, який володіє цією мовою, має значну перевагу у використанні та узагальненні інформації з різних джерел.

SQL є основною стандартною мовою технології реляційних баз даних завдяки своїй елегантності та незалежності від специфіки комп’ютерних технологій. Стандарт SQL визначається Американським Національним Інститутом Стандартів (ANSI) і Міжнародною Організацією по Стандартизації (ISO).

Більшість комерційних програм баз даних розширюють SQL, додаючи різні особливості в цю мову, які вони вважають корисними. Іноді ці особливості порушують стандарт мови, але деякі з них стають стандартами “ринку” через свою корисність.

Microsoft SQL Server є високопродуктивною платформою для обробки та аналізу даних будь-якого розміру, яка відповідає всім сучасним вимогам роботи з даними будь якої складності. SQL Server має широкий спектр функцій та інструментів, що дозволяють ефективно управляти даними, виконувати запити, забезпечувати безпеку та доступність даних.

За допомогою SQL Server можна створювати реляційні бази даних, виконувати транзакції, встановлювати зв’язки між таблицями, індексувати дані для швидкого пошуку та виконувати різноманітні запити. Також платформа підтримує роботу з геоданими, що дає змогу створювати геопросторові додатки та аналізувати дані за географічними параметрами.

SQL Server надає зручні інструменти для резервного копіювання та відновлення даних, а також має вбудовану систему безпеки, яка дозволяє керувати доступом до даних та забезпечувати їх захист від несанкціонованого доступу. Платформа також має вбудовану підтримку реплікації, що дозволяє забезпечити доступність даних в режимі реального часу.

Окрім того, SQL Server підтримує інтеграцію з різноманітними додатками, включаючи різні сервіси та інші бази даних. SQL Server можна використовувати для розробки та реалізації багатошарових додатків, які працюють з великим обсягом даних та мають складну структуру. Це робить SQL Server потужним інструментом для розробки підприємницьких додатків та систем управління даними будь-якого розміру та складності.

## ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі докладно розглянуто всі основні технології розробки, наведено їх призначення та переваги. Кожну з цих технологій обрано з урахуванням масштабів проекту і вимог, які він повинен виконувати. Зазначені технології виявляються дуже ефективними в своїх сферах застосування, крім того, вони взаємодіють добре з іншими інструментами і утворюють єдину гармонійну систему, що забезпечує комфортну, продуктивну та високоякісну розробку дипломного проекту.

# РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ

# 3.1. Розробка архітектури системи

Структуру програми організовано у вигляді сукупності незалежних блоків, які називають модулями. Кожен модуль має свою структуру та поведінку, яка відповідає встановленим правилам, що сприяє спрощенню тестування та виявленню помилок. До того ж, завдання, які можуть виконуватись не лише в веб-додатку, можна чітко відокремити від інших завдань, що забезпечує більшу універсальність серверного ПЗ згідно з правилами [REST API](https://uk.wikipedia.org/wiki/REST) [17]. Кожен модуль виконує свою ізольовану роль у застосунку, та має власне місце в проекті. Архітектура програми включає такі модулі, як модуль для управління БД, модуль API, модуль сервісів та модуль моделей.

На [рис. 3.1](#Рис_3_1) подано взаємозв’язок модулів, де клієнт взаємодіє з сервісами які обробляють дані клієнт через передачу за допомогою API модуля, який приймає вхідні запити на сервер. Водночас кожен серверний модуль пов’язаний з модулем моделей, який необхідний для того аби визначити всі сутності які будуть використовуватись в усіх модулях.

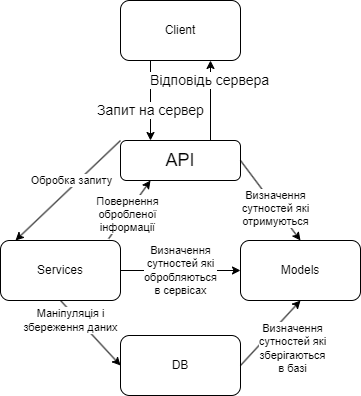


Рис. 3.1. Діаграма взаємозв’язоку модулів системи.

# 3.2. Use-case діаграма

# Use case діаграма є одним із видів діаграм, які використовують в галузі розробки програмного забезпечення для моделювання функціональної взаємодії між системою та її користувачами. Вона дає змогу описати поведінку системи на основі різних сценаріїв взаємодії між користувачами та системою, тобто вказати, які дії можуть виконувати користувачі, які відповіді може повертати система та як вона може взаємодіяти з іншими системами або сервісами. Use case діаграма допомагає зрозуміти функціональні вимоги до системи та забезпечує основу для подальшого розроблення архітектури системи.

Зокрема use-case діаграма вимагає наведення так званих use-кейсів, тобто дій, які зможуть здійснювати представлені в застосунку сутності. Нижче перераховано use-кейси для всіх сутностей.  
 Користувач:

-Створення облікового запису  
 -Вхід в обліковий запис

-Редагувати обліковий запис

-Здійснити пожертвування

-Поповнити рахунок

-Подати заявку на волонтерський статус

Волонтер:

-Створити оголошення

-Редагувати оголошення

-Закрити оголошення

-Видалити оголошення

-Заповнювати і редагувати волонтерську сторінку

Адміністратор:  
 -Видалити користувача

-Скасувати для волонтера його статус

-Перевірити запит

На [рис. 3.2](#Рис_3_2) подано use-case діарама для нашого застосунку.



Рис. 3.2. Use-case діаграма.

Створивши use-case діаграму та перерахувавши use-cases, ми досягли кількох цілей. По-перше, ми з'ясували, які конкретно функції та можливості повинна мати наша система для забезпечення задоволення потреб користувачів. По-друге, отримали зрозуміння взаємодії між користувачами та системою, що дасть нам змогу розробляти кращі інтерфейси та взаємодію зі системою. Нарешті, створили основу для подальшого розроблення архітектури системи, яка буде відповідати вимогам користувачів та забезпечувати ефективну та безперебійну роботу системи.

# 3.3. Розробка дизайну програми

Розробка дизайну програми є важливою складовою створення програмного продукту. Ефективний та привабливий дизайн відіграє вирішальну роль у взаємодії користувача з програмою, забезпечуючи зручність, зрозумілість та задоволення від використання.

Першим кроком у розробці дизайну програми є збір вимог та розуміння цільової аудиторії. Необхідно ретельно дослідити потреби та очікування користувачів, їхні пріоритети та побажання щодо вигляду та функціональності програми. Це допоможе визначити основні принципи дизайну, колірну палітру, шрифти та загальну структуру програми.

Далі необхідно розробити інформаційну архітектуру програми, визначити логіку навігації та розміщення елементів на екрані. Основна мета полягає у створенні інтуїтивно зрозумілого інтерфейсу, де користувачі зможуть легко знаходити потрібну інформацію та виконувати необхідні дії.

Важливим етапом є розробка візуального дизайну програми. Дизайнер створює графічні елементи, включаючи іконки, кнопки, фонові зображення, а також визначає стиль та композицію екранів. Колірна гама, типографіка та використання простору мають велике значення в створенні приємного візуального сприйняття програми.

Після розробки візуального дизайну важливо провести його тестування на різних пристроях та платформах, щоб переконатися в його адаптивності та належному відображенні на різних розмірах екранів. Виявлені недоліки та виправлення вносяться на цьому етапі, забезпечуючи оптимальну роботу програми на різних пристроях.

Окрім візуального дизайну, необхідно також розробити інтерактивний дизайн програми. Це охоплює визначення взаємодії користувача з програмою, анімацію, переходи між екранами та інші елементи, що забезпечують динаміку та комфорт використання.

Розробку дизайну програми необхідно провести з урахуванням усіх функціональних вимог та особливостей програмного продукту. Важливо забезпечити зручний доступ до всіх необхідних функцій та інформації, враховуючи логіку використання програми користувачами.

Останнім етапом розробки дизайну програми є його документування та передача команді розробників. Документація повинна містити детальний опис всіх дизайнерських рішень, включаючи специфікації кольорів, шрифтів, розміщення елементів та іншу необхідну інформацію.

Узгоджений та професійно розроблений дизайн програми грає важливу роль у створенні успішного та конкурентоспроможного програмного продукту. Він забезпечує не лише зручність використання, але й створює позитивне враження у користувачів, підвищуючи їхню задоволеність та залученість до програми. Тому, розробка дизайну програми є невід'ємною складовою процесу створення високоякісного програмного продукту.

Для цього проекту, розроблено декілька шаблонів інтерфейсу, який необхідно релізувати.

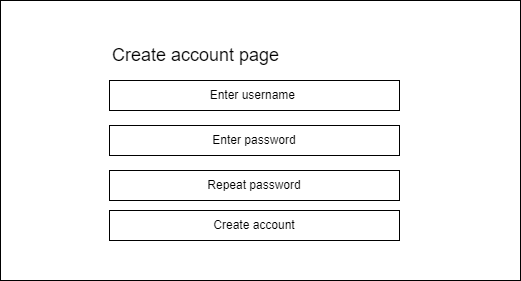


Рис. 3.3. Шаблон сторінки створення облікового запису.

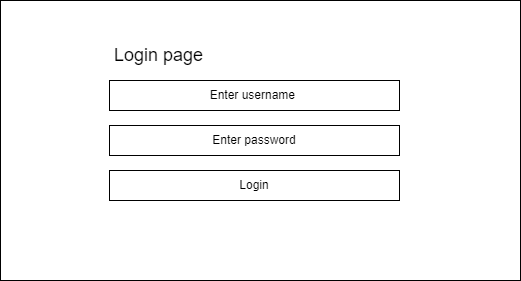


Рис. 3.4. Шаблон стрінки для авторизації.

На [рис.3.3](#Рис_3_3) та [рис.3.4](#Рис_3_4) подано шаблон для автентифікації та авторизації, обидві у вигляді форми для введення користувачем своїх облікових даних.

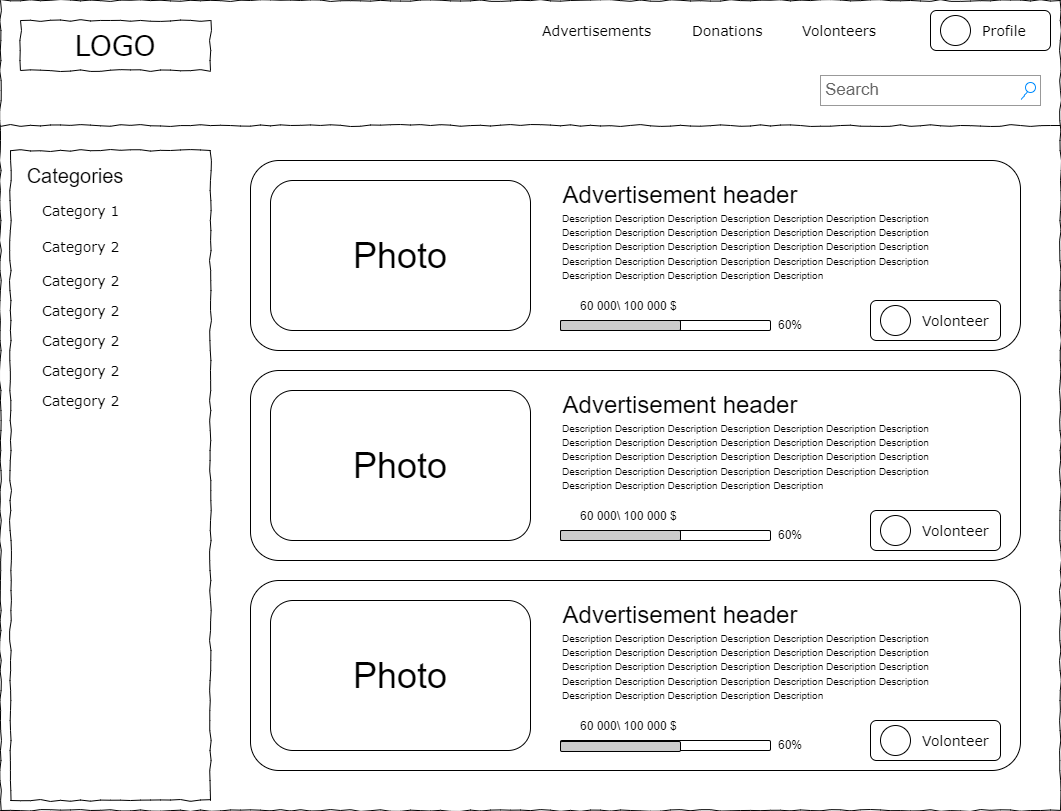


Рис. 3.5. Шаблон головної сторінки.

На [рис.3.5](#Рис_3_5) представлено головну сторінку. На ній зверху є навігаційна панель, яку повинен бачити кожен користувач який увійшов у систему для зручності використання. На самій панелі є місце для логотипу, який переноситиме користувача на головну сторінку, вкладку Advertisements, яка теж буде переносити на головну сторінку, тому що головна сторінка – це оголошення. Далі вкладки Donations та Volonteers будуть переносити користувача на сторінки з пожертвуванями та волонтерами. Також на панелі створено вкладку з профілем, яка буде переносити користувача на сторінку з його профілем, на ній можна буде побачити ім’я самого користувача.  
 На самій сторінці відображено панель з категоріями, у ній ми можемо вмикати певні категорії і тим самим відокремлювати оголошення, які нас цікавлять.  
 По середині самої сторінки міститься перелік оголошень, оголошення на сторінці будуть представлені у стиснутій формі, у ній будне видно перше зображення в оголошені, співвідношення затребуваної суми та вже зібраної суми, назву оголошення, посилання на профіль волонтера, і неповний опис оголошення. Повний опис і всі інші зображення прикріплені до оголошення можна буде побачити відкривши його на цій сторінці.

Також на сторінці є панель пошуку, за допомогою якої можна здійснювати пошук оголошення за назвою.

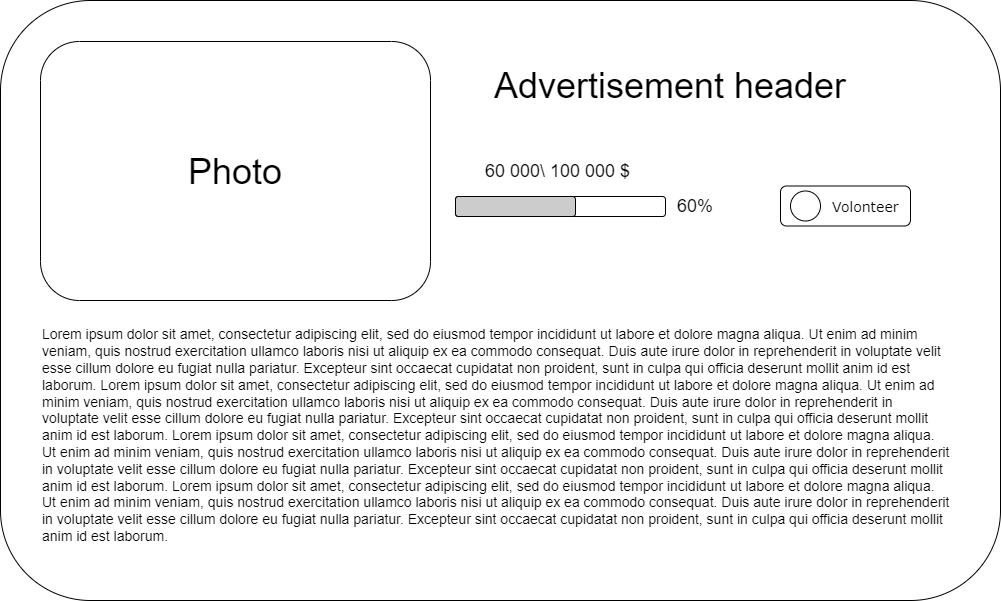


Рис. 3.6. Повне представлення оголошення.

На [рис. 3.6](#Рис_3_6) подано представлення повного оголошення, на ньому вже можна буде прогорнути повний опис оголошення та прогортувати всі прикріплені зображення.



Рис. 3.7. Шаблон сторінки з пожертвуваннями.

На [рис. 3.7](#Рис_3_7) подано вигляд сторінки, де користувач зміг би переглянути перелік пожетвувань, які він зробив, у кожному елементі списку буде продемонстровано суму пожертвування, посилання на оголошення та дату і час пожертвування.

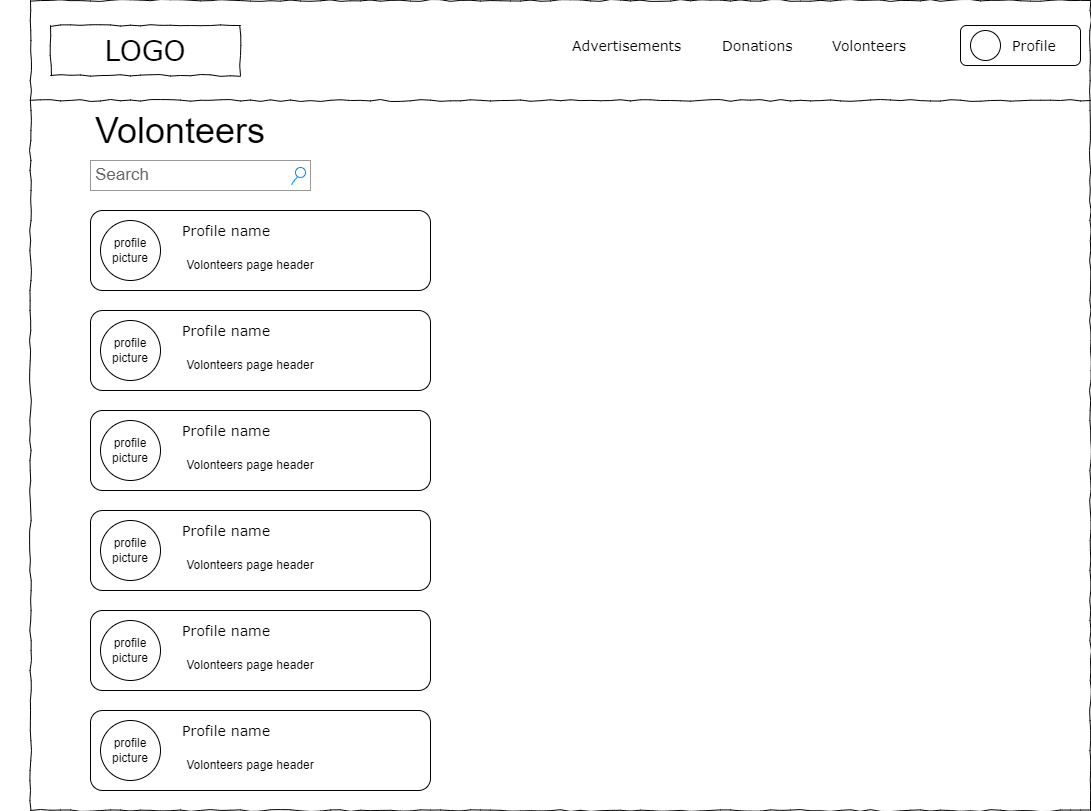


Рис. 3.8. Шаблон сторінки з волонтерами

На [рис.3.8](#Рис_3_8) представлено вигляд сторінки з волонтерами, у списку будуть представлені волонтери, які зареєстровані на платформі. Також на сторінці є поле пошуку, за допомогою якого можна буде знайти певного волонтера.

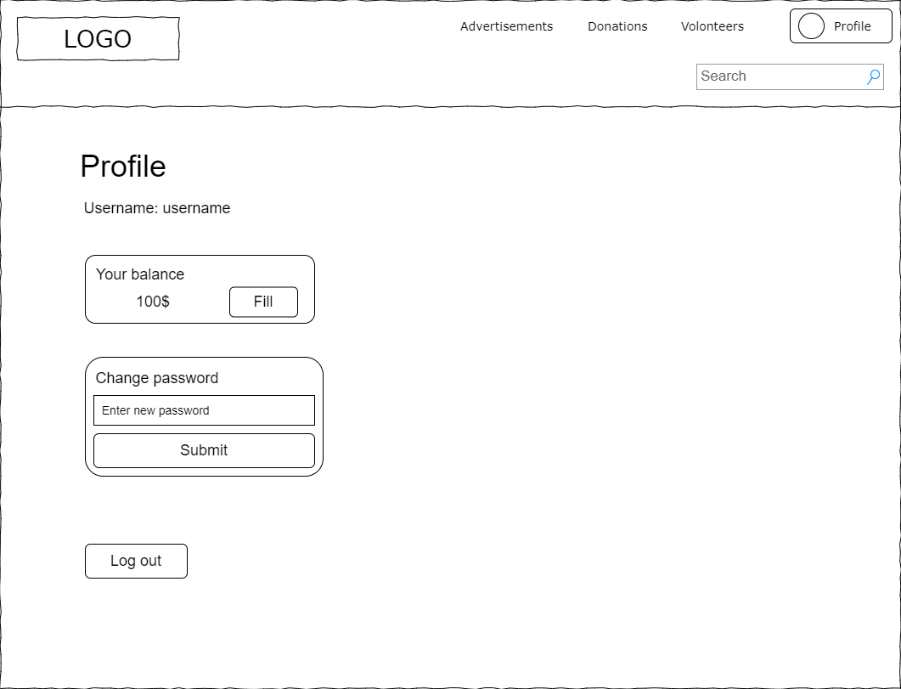


Рис. 3.9. Шаблон сторінки профіля.

На [рис. 3.9](#Рис_3_9) зображено вигляд сторінки з профілем користувача. На ній користувач зможе переглянути та поповнити свій баланс для пожертвувань, змінювати пароль та виходити з облікового запису.

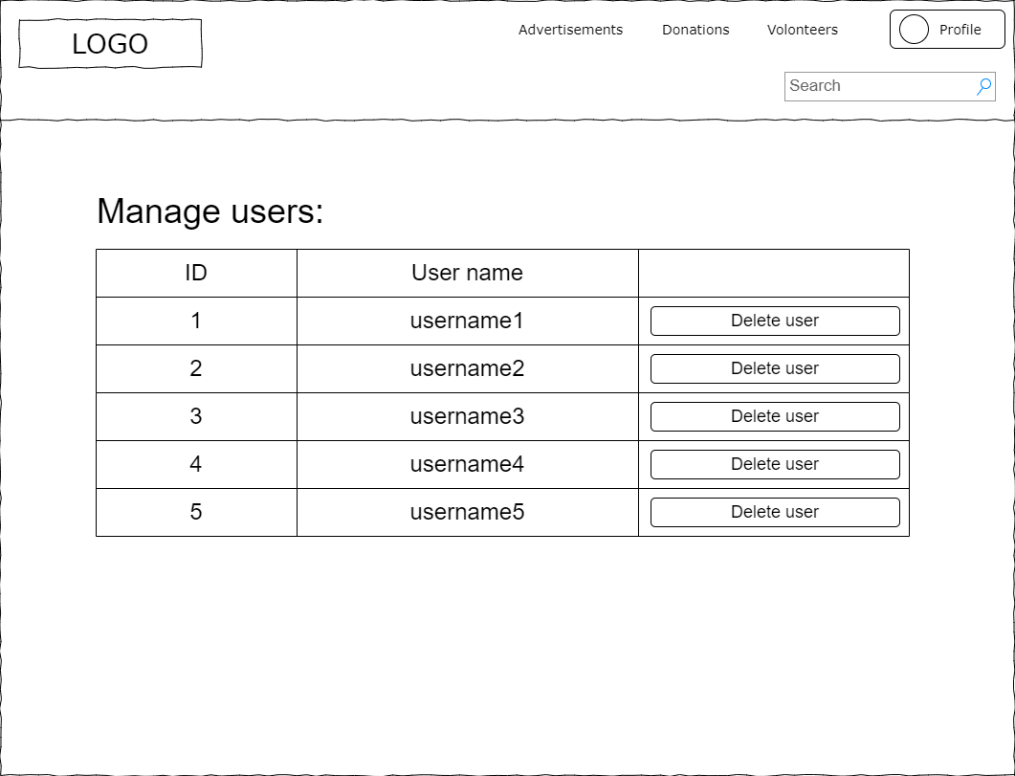


Рис. 3.10. Сторінка керування користувачами.

На [рис.3.10](#Рис_3_10) поадно вигляд сторінки керування користувачами, доступ до цієї сторінки може мати лише адміністратор. На самій сторінці присутня таблиця з користувачами, окрім адміністраторів, за допомогою цієї таблиці адміністратор буде мати змогу видаляти користувачів.

# 3.4. Розробка системи

У розробці програмного забезпечення насамперед потрібно визначитись з архітектурою та шаблонами, які допоможуть спроектувати потрябний проект так, щоб він був прдатний до масштабування, не генерував системні несправності, та не обтяжував процес розробки.  
 Проект розроблений на базі ASP.NET, де найчастіше використовують архітектуру WebAPI або MVC. Застосуємо у цьому проекті WebAPI, такий підхід є найзручнішим для роботи з API (програмованим інтерфейсом програми), яке відкриває серверну частину для спілкування з зовнішнім світом, зокрема, і з клієнтом.

У доповнення до WebAPI також використаємо шаблон «Репозиторій», який відокремлює сервіси, що реалізовують всю функціональність програми від самого веб-проекту. Це доброякістно впливає на масштабованість проекту, а також допомагає не порушувати правила REST та [SOLID](https://uk.wikipedia.org/wiki/SOLID_(%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) [14], які є завжди необхідними для розробки великих додатків.

Отже, визначено, що сервіси будемо відокремлювати від модуля веб-серверу задля реалізації шаблону «Репозиторій», також окрім сервісів необхідно відокремити модуль, який відповідає за роботу з базою даних та визначення моделей проекту. Моделі – це сутності описуємі в проекті, які дають змогу за допомогою Entity Framework створити необхідні таблиці в базі даних, а також реалізовувати роботу з ними в есервісах та в веб-сервері.  
 Роздробивши проект, отримали три модулі: «Веб-сервер», «Сервіси» та «База даних». Такий поділ створює відовідні зв’язки між ними, модулі «Веб-сервер» та «Сервіси» зсилаються на модуль «База даних», тому що сервісам потрібно мати доступ до бази даних та сутностей для обробки даних, а веб-серверу потрібен доступ до моделей для того, щоб працювати зі сервісами. Таким чином, маємо односторонній зв’язок у кожному відношені, що дає зручну та ефективну структуру.

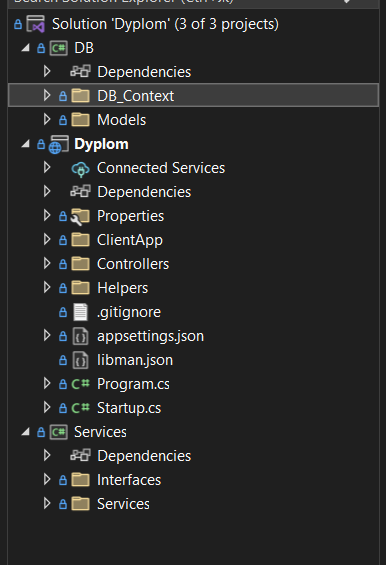


Рис. 3.11. Структура проекту.

На [рис.3.11](#Рис_3_11) подано створену структуру проекту. Тут є модулі DB, Dyplom, який є веб-сервером, та Services. Модулі DB та Services є проектами «класових бібліотек», тобто їхня мета – зберігати сутності, та оброку певних даних.

У модулі DB створено папку DB\_Context, у ній зберігаємо файл «контексту бази даних», тобто файл, в якому використовуємо EntityFramework для створення та налаштування бази даних. Також у модулі є папка Models. Уній зберігаємо моделі, тобто сутності, які зберігаються в базі даних, та визначення яких використовуємо в сервісах та веб-сервері для обробки запитів.

У модулі веб-серверу існує папка Properties, яка створюється за замовчуванням Visual Studio, у ній міститься json файл з налаштуванням веб-серверу. Папка ClienApp містить проект на React і власне весь клієнтський код загалом. Папка Controllers містить код, який здійснює роботу веб-серверу, він приймає запити та використовує сервіси для їх обробки. Папка Helpers містить допоміжний код, який можна застосувати в веб-сервері для того, щоб оптимізувати використання сервісів та уникнути повторювання коду. Файли Program.cs та Startup.cs – це стартові файли проекту ASP.NET, вони містять первинні налаштування для серверу, та визначають сутності модулів.  
 У модулі Services створено папки Interfaces та Services. Папка Interfaces містить інтерфейси, які реалізовуються в конкретних сутностях в Services, вони потрібні для того, щоб сервіси можна було використовувати в веб-сервері як абстракції, та виконувати правило інверсії залежностей з SOLID.

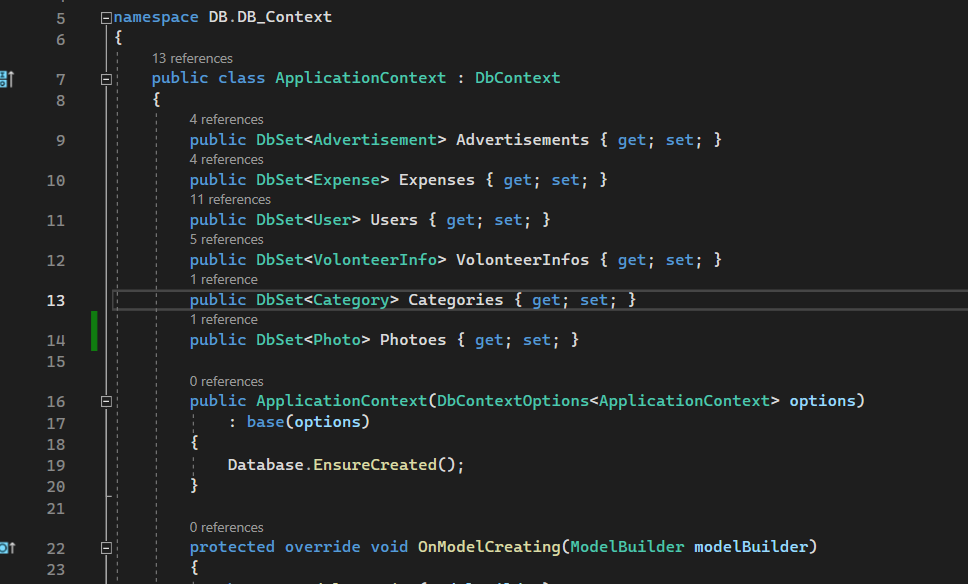


Рис. 3.12. Демонстрація файлу контексту бази даних.

У файлі Context.cs ([рис. 3.12](#Рис_3_12)) подано код який налаштовує базу даних, зоарема перелік полів, на основі яких будуть створенні таблиці, саме тут можна зрозуміти, що Entity Framework використовує поля класів моделей для визначення стовпців таблиць в базі даних.

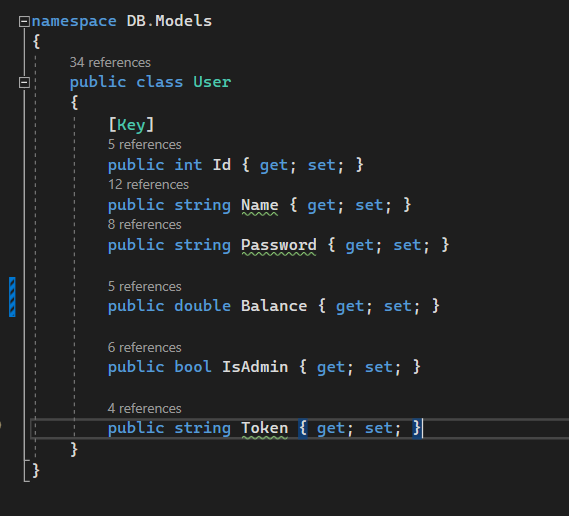


Рис. 3.13. Модель User.

На прикладі моделі з [рис.3.13](#Рис_3_13) наведено приклад визначення моделі. Видно, що поля описані примітивними типами даних такі як числа, текстові рядки, та булеві значення, також поле Id яке відіграє роль зовнішнього ключа позначаємо за допомогою [Key]. Зв’язні ключі в моделях визначаємо як число, оскільки зв’язки між таблицями базуються на зовнішніх ключах.

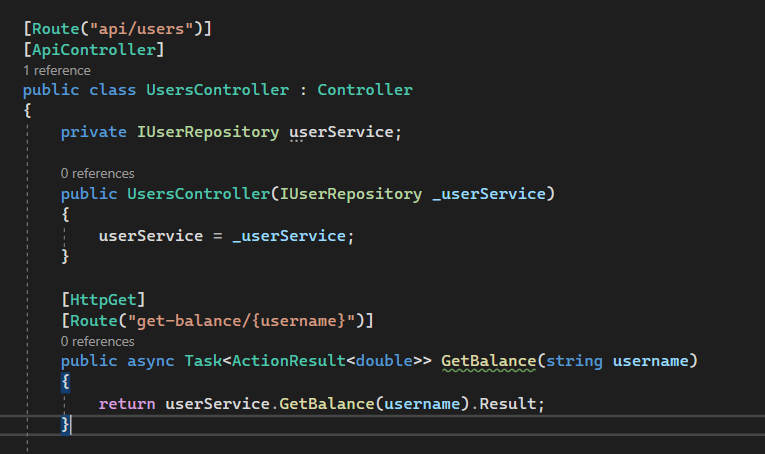


Рис. 3.14. Приклад контролеру для моделі User.

На [рис.3.14](#Рис_3_14) наведено приклад контроллеру. Бачимо, що клас UsersController наслідується від класу Controller та має атрибут [ApiController]. Це допомагає ASP.NET визначити, що даний клас є контроллером та може приймати запити з клієнта.  
 Також контроллер та методи контроллеру мають атрибут [Route(“”)], цей атрибут необхідний для створення архітектури API в проекті, за допомогою Route в контроллері вказуємо, що запити, які починаються з «api/users», будуть оброблятись в UsersController. Далі, якщо програма знаходить метод, який відповідає цьому шляху, то він і обробляє цей запит. Також є методи, які не мають прописаного шляху. Це робиться для того, щоб ожна було робити запити лише по шляху контроллера, і метод визначається типом HTTP запиту, тобто у нашому випадку, якщо зробимо запит «api/users» з методом HTTP GET, тоді цей запит прийме метод з атрибутом HttpGet, але без атрибуту Route.

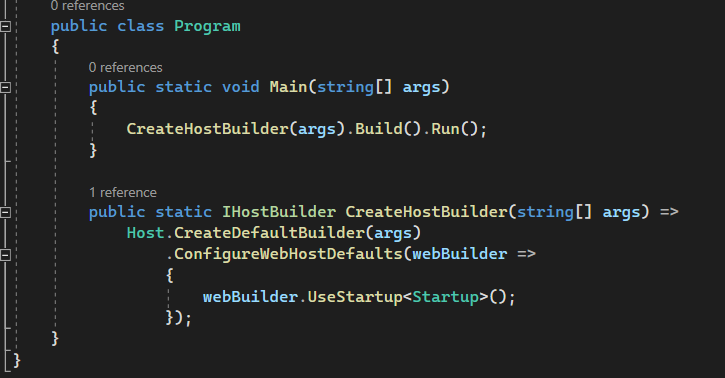


Рис. 3.15. Код Program.cs.

Файл Program.cs ([Рис. 3.15](#Рис_3_15)) є початковим файлом, з якого за замовчуванням починаються усі програми в C#, у ньому створюємо і запускаємо веб-сервер, через сутність Startup, в якій здійснюємо конфігурацію сервера.

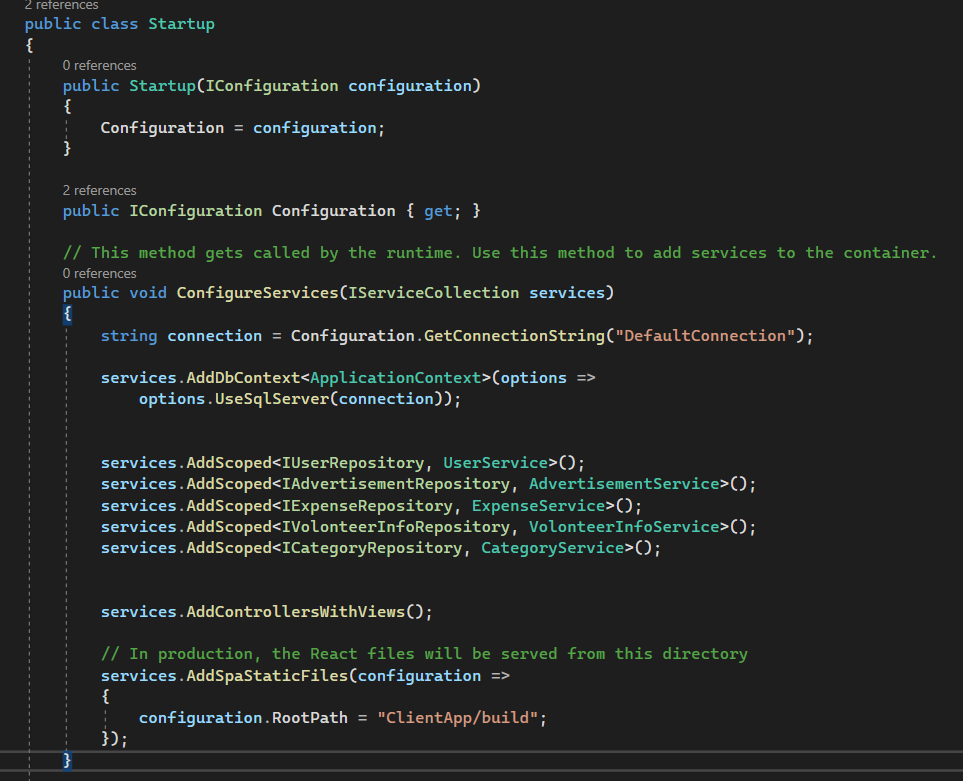


Рис. 3.16. Код визначення Startup та ConfigureServices*.*

На [рис. 3.16](#Рис_3_16) подано початок визначення класу Startup. У його методі ConfigureServices() здійснюємо підключення до бази даних та під’єднання сутності контексту бази даних. Далі під’єднуємо сервіси через .AddScoped(), через нього даємо контроллерам доступ до сервісів через абстракції, як цього вимагає принцип інверсії залежностей з SOLID.

Також в кінці ми вказуємо директорію з клієнтський кодом аби він був підєднаний до сервера.

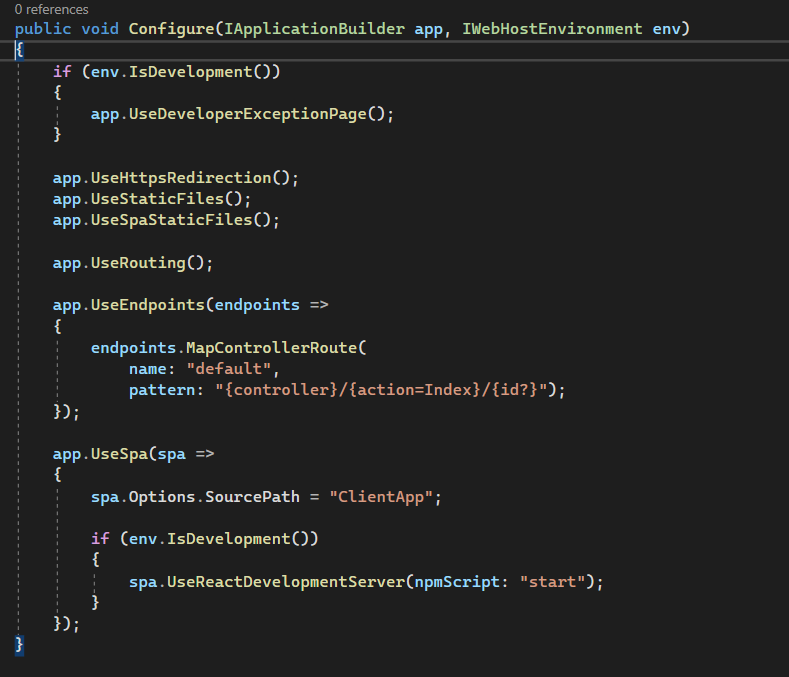
**

Рис. 3.17. Метод Configure з Startup.cs.

У цьому методі на [рис.3.17](#Рис_3_17) налаштовуємо деякі важливі речі, такі як доступ до редирекції HTTP, доступ до використання статичних файлів для роботи з React, а також за допомогою app.UseSpa() вказуємо серверу застосувати npm для запуску проекту на реакт. Це необхідно для одночасного відкриття клієнтського та серверного додатку, для економії часу на розгортання.

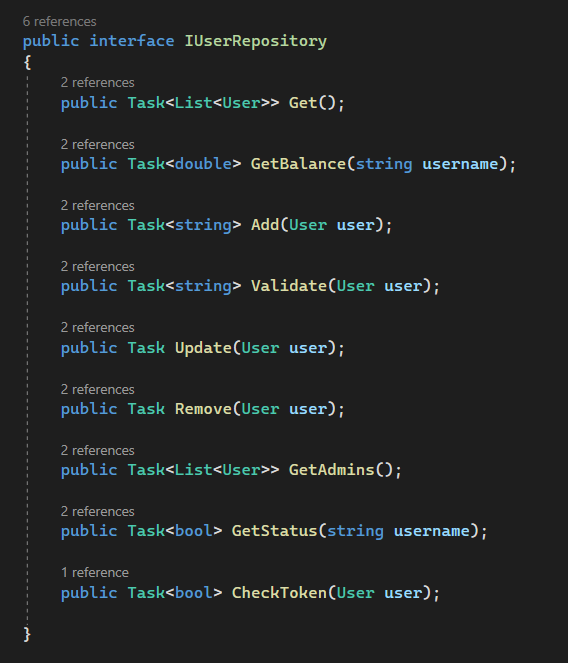


Рис. 3.18. Абстракція сервісу на прикладу IUserRepository*.*

За допомогою абстракції IUserRepository з [рис.3.18](#Рис_3_18) реалізували шаблону [«Репозиторій»](https://uk.wikipedia.org/wiki/Repository) [15] та створили необхідну абстракцію для інверсії залежностей. У коді визначаємо перелік методів, які необхідно реалізувати в сервісі, абстрагуючись від його конкретної реалізації. Тип інтерфейсу вже застосовували в Startup.cs для підключення сервісу, та звертались до сервісу в контроллері через тип цього інтерфейсу.

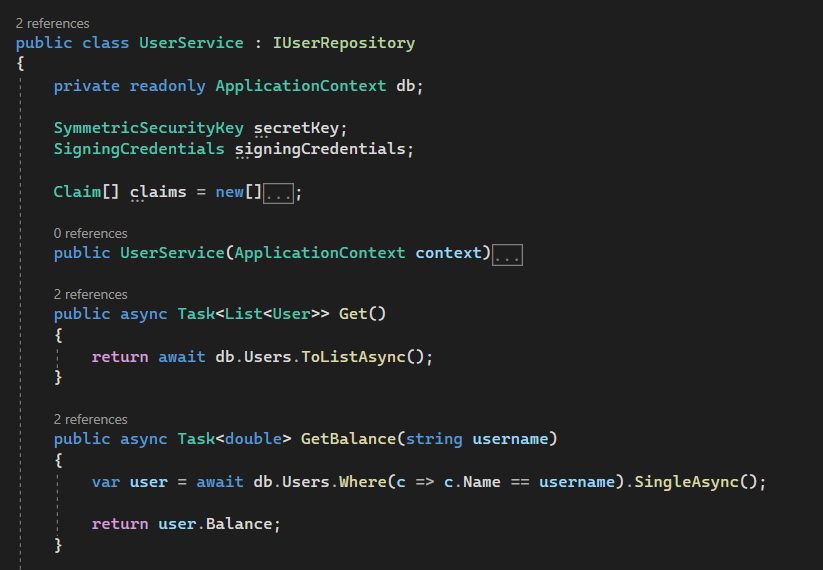


Рис. 3.19. Сервіс на прикладі сервісу користувачів.

На [рис.3.19](#Рис_3_19) подано приклад самого сервісу з реалізаціями методів. Бачимо, що через конструктор класу здійснюємо підключення до бази даних, і використовуємо її в методах для обробки запитів, використання методів сервісів ми вже бачили в контроллері. Також у сервісі реалізовуємо раніше згаланий інтерфейс *IUserRepository*, так як через нього передаємо реалізацію наших методів. Таким чином досягнуто абстрагування логіки серверного додатку від прийому запитів, що дало змогу розбити проект на більш спеціалізовані модулі, не створювати надто нагромаджений код, а також досягти сильної масштабованості, яка вкрай необхідна для розробки повноцінних додатків.

Загалом у серверній стороні досягли зрозумілої та ефективної структури, яка може обробляти запити та робить це ефективно. Також система побудована таким чином, що будь-які проблеми пов’язані з запитами, будуть легко відстежуватись та виправлятись.

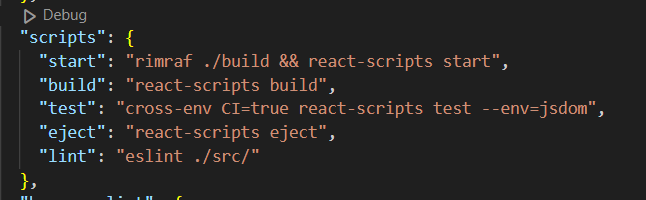


Рис. 3.20. Записані команди в package.json.

Переходячи до клієнтської сторони, варто згадати команди прописані в файлі package.json ([рис. 3.20](#Рис_3_20)), команди, записані у цьому файл,і виконуються, коли серверний додаток використовує поле start, раніше застосовували його в Startup.cs.

Варто також згадати що React.js використовує менеджер пакетів npm, цей менеджер дозволяє не тільки займатися менеджментом зовнішній модулів, а також і здійснювати автоматизоване розгортання проекту на React.

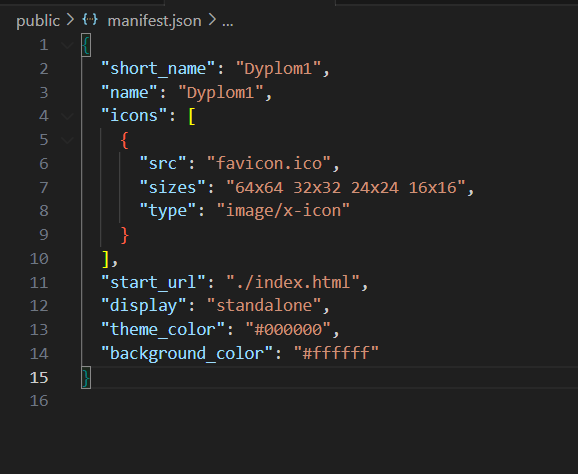


Рис. 3.21. Конфігурації в manifest.json.

Менеджер [npm](https://docs.npmjs.com/) [12] окрім файлу пакетів package.json також використувує файл, зображений на [рис. 3.21](#Рис_3_21). Тут бачимо базові налаштування найменування, іконки в браузері, базового кольору, а також вказується початковий файл index.html, який є логічним початком всі веб-додатків.

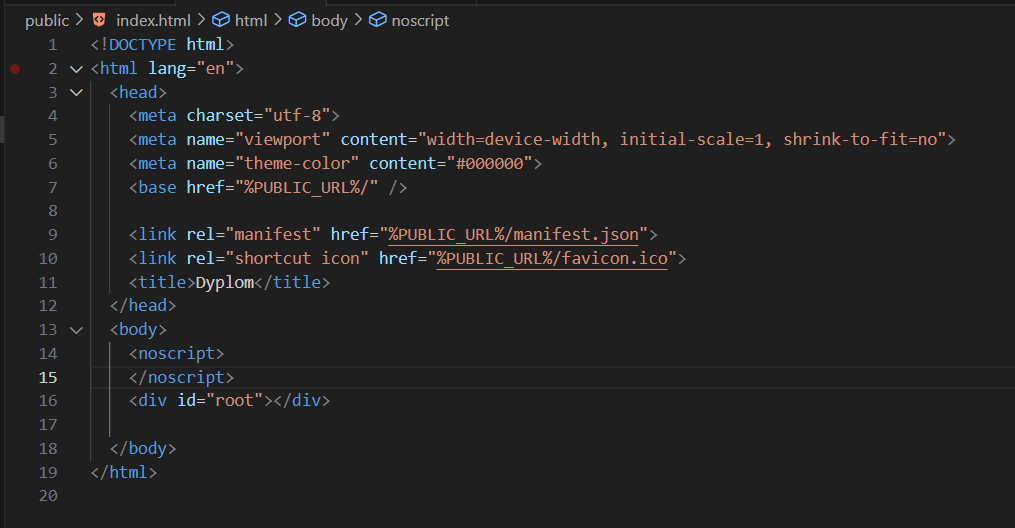


Рис. 3.22. Код в файлі index.html.

На [рис.3.22](#Рис_3_22) подано початковий файл створеного клієнтського проекту. Тут бачимо підключення manifest.json, а також у тілі файлу div з індентифікатором root, це необхідно для того аби наш JS код на React зміг помістити вміст нашого інтерфейсу в index.html. Справа в тому що React.js працює за принципом SPA(Single Page Application) або ж додатку на базі однієї сторінки. Це не означає, що на React можна створювати лише додатки з однією сторінкою, це лише описує сам принцип збирання проєкту на React. Синтаксис фреймворку відрізняється від HTML/CSS/JS і має зовсім інший структурний підхід, тому коли пишемо компоненти на React, npm інтерпретує його код в один єдиний файл, який містить величезне нагромадження HTML/CSS/JS коду, який може зрозуміти рушій браузера, і в результаті маємо один файл index.html який містить всю логіку інтерфейсу, який отримали, інтерпретувавши код React, що є принципом роботи SPA.

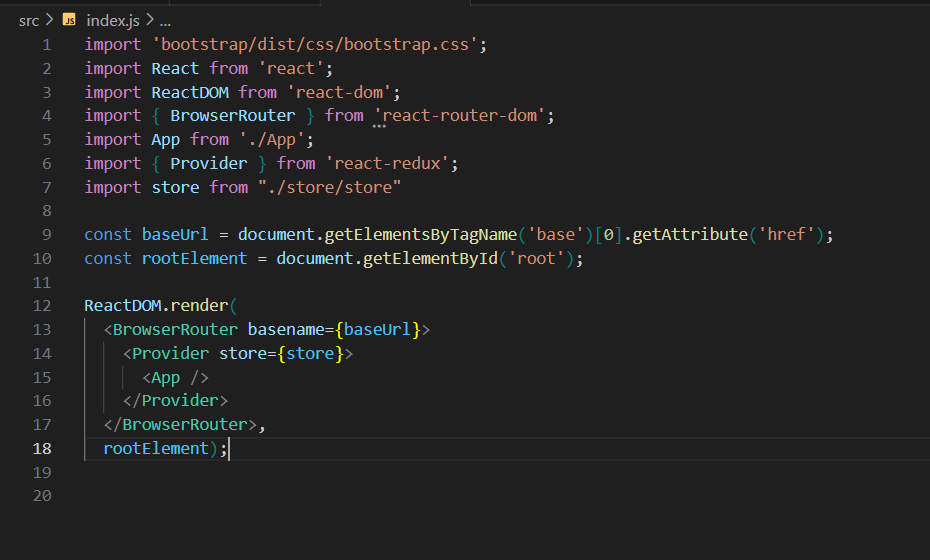


Рис. 3.23. Код файлу index,js.

Додаток починається з файлу index,js, npm використовує його для запуску всієї логіки в клієнті ([рис. 3.23](#Рис_3_23)). Тут видно, що rootElement містить в собі об’єкт згаданого раніше root. У нього через метод ReactDOM.render() розташовуємо інтерпретований код з компонентів, зокрема з кореневого компонету App. Метод ReactDOM.render() здійснює інтерпретацію коду та переміщення його в rootElement для того, щоб отимати готовий код для рушія браузера. У першому аргументі методу, бачимо кілька вкладених компонентів, BrowserRouter потрібний для здійснення навігації в додатку шляхами, Provider для імплементування Redux.js та корінний компонент додатку App.

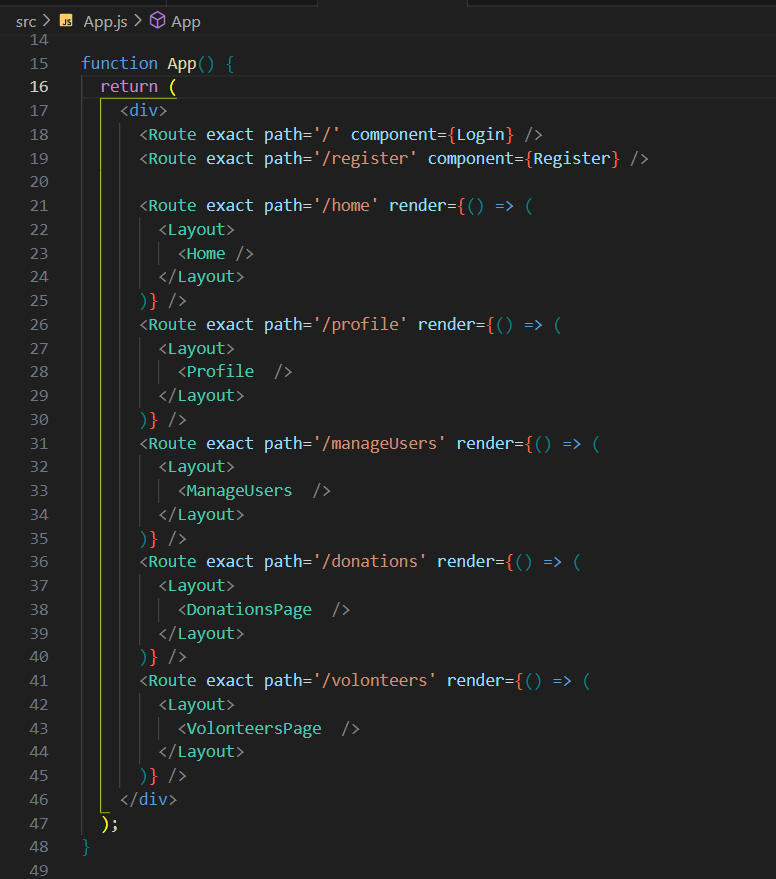


Рис. 3.24. Код компоненту App.

У коді компоненту App ([рис. 3.24](#Рис_3_24)) бачимо перелік компонентів Route, з їх допомогою визначаємо, як буде працювати навігація в нашому додатку, і які компоненти будемо відображати на сторінці, тобто компоненти сторінок. Також компоненти сторінок вкладаємо в компонент Layout, це необхідно для того, щоб можна було давати потрібним нам сторінкам навігацію, яка буде у додатку, це набагато зручніше зробити в одному місці, ніж в кожному файлі.

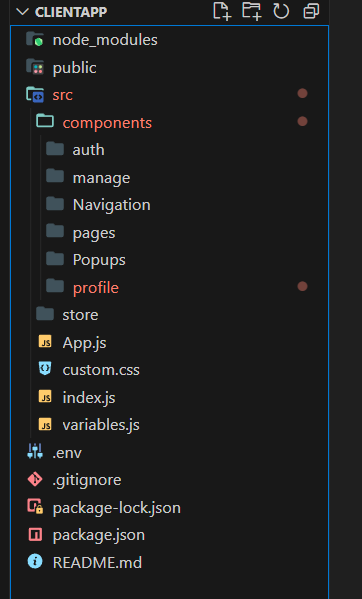


Рис. 3.25. Структура клієнтського проекту.

На [рис. 3.25](#Рис_3_25) поданоструктуру клієнтського проекту. Маємо папку node\_modules, яка зберігає npm пакети, public зберігає згадані вище index.html, manifest.json та іконку. Папка components містить вже компоненти інтерфейсу додатку, у ній знаходитяться компонети розташовані у своїх папках. У папці src зберігаються згадані вище App.js, index.js файл зі статичними змінними variables.js, в якому зокрема зберігається адреса додатку в браузері, у папці store зберігаємо файли, необхідні для Redux. У кореневій папці пристуній .gitignore для роботи з системою Git, а також файли для npm package.json та package-lock.json.

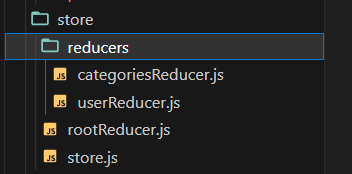


Рис. 3.26. Папка store, в якій зберігаємо дані Redux.js.

Фреймворк Redux.js необхідний для роботи в React. Оскільки компоненти в додатках проекту розташовуються у вигляді дерева, то якщо нам знадобиться передати дані з одного розрізненого компонента в інший розрізнений компонент, нам доведеться знаходити компонент який хї об’єднує, і передавати дані нагору, а потім вниз до потрібного компонента, це робить код нечитабульним, а структуру неефективною, а процес розробки вкрай неефективним. Для вирішення таких проблем і використовується Redux.js, який дає змогу створити окреме сховище даних ([рис. 3.26](#Рис_3_26)), відокремлюючи їх від структури компонентів інтерфейсу, та відкриваючи доступ для кожного з них до сховища, що дає можливсть, значно спростити процес передачі даних між компонентами, та зберегти якість коду.

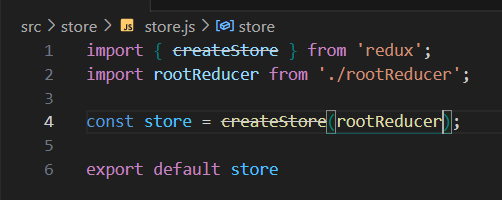


Рис. 3.27. Код файлу store.js.

Об’єкт файлу store ([рис. 3.27](#Рис_3_27)) вже використовували в index.js для підключення Redux через компонет Provider. Тут передаємо в аргумент методу createStore() кореневий редюсер rootReducer, редюсер – це компонент, який дає змогу зберігати в собі певні дані та надає функціонал для їх зміни.

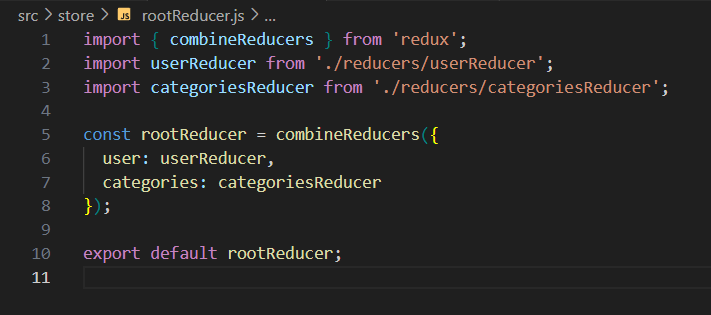


Рис. 3.28. Код файлу rootReducer.js.

У кореневому редюсері [(рис. 3.28)](#Рис_3_28) ми використовуємо метод combineReducers, для об’єднання кількох редюсерів, це потрібно для того аби ми могли не зберігати всі дані в одному місці, і відокремити їх по окремих полях, та покращити структуру їх зберігання.

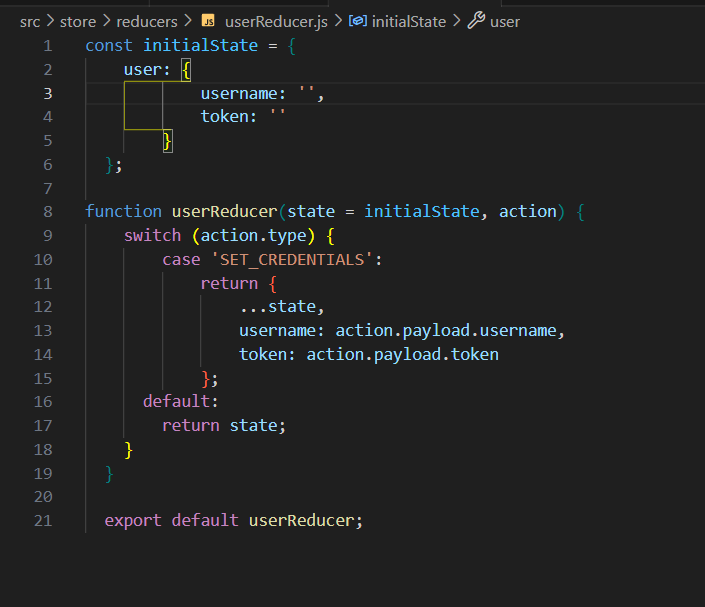


Рис. 3.29. Редюсер на прикладі редюсера для користувача*.*

*У* коді редюсера користувача ([рис. 3.29](#Рис_3_29)) бачимо, як дані зберігаються та як ми можемо їх використовувати. Змінна initialState описує дані, які зберігаємо в редюсері, у вигляді зjs об’єкта. У функції userReducer()робимо розгалуження на основі action.type, з його допомогою визначаємо, яку саме операцію удемо здійснювати з даними редюсера.

# 3.5. Основний функціонал системи

Коли користувач вперше зайде в додаток, він побачить форму для аутентифікації ([рис. 3.30](#Рис_3_30)). Користувач може ввести своє ім’я, якщо він має профіль у додатку, поле з паролем приховує введені символи для безпеки. Якщо коистувач не має профілю, він може створити новий профіль натиснувши відповідне посилання.

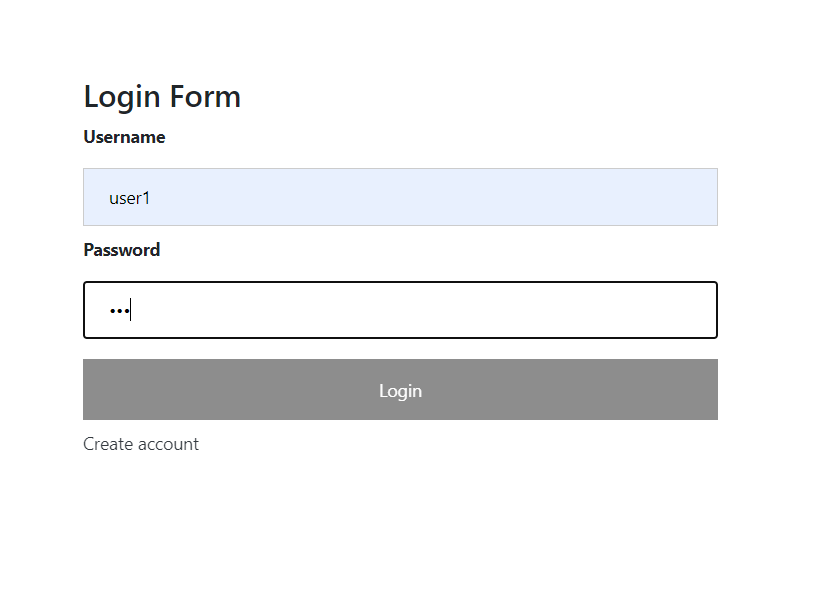


Рис. 3.30. Сторінка форми аутентифікації.

Сторінка реєстрації ([рис. 3.31](#Рис_3_31)) має схожий вигляд, вона має додаткове поле для підтвердження паролю, та за аналогією з попередньою формою, ця форма містить посилання на аутентифікацію.

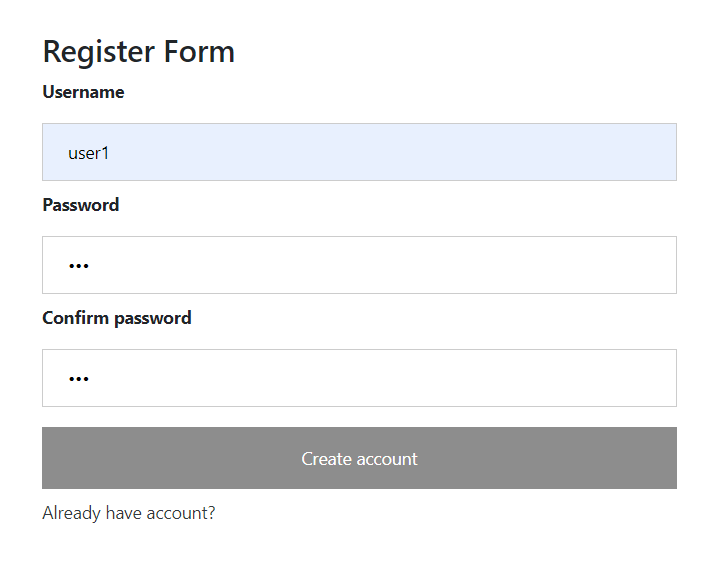


Рис. 3.31. Форма реєстрації.

Якщо користувач увійде в систему, він зустріне головну сторінку ([рис.3.32](#Рис_3_32)), де можна буде побачити перелік існуючих на разі оголошень, оголошення можна відбирати за категоріями, переглянути заповненість збору, переглянути коротку версію оголошення, а також шукати оголошення за заголовком.

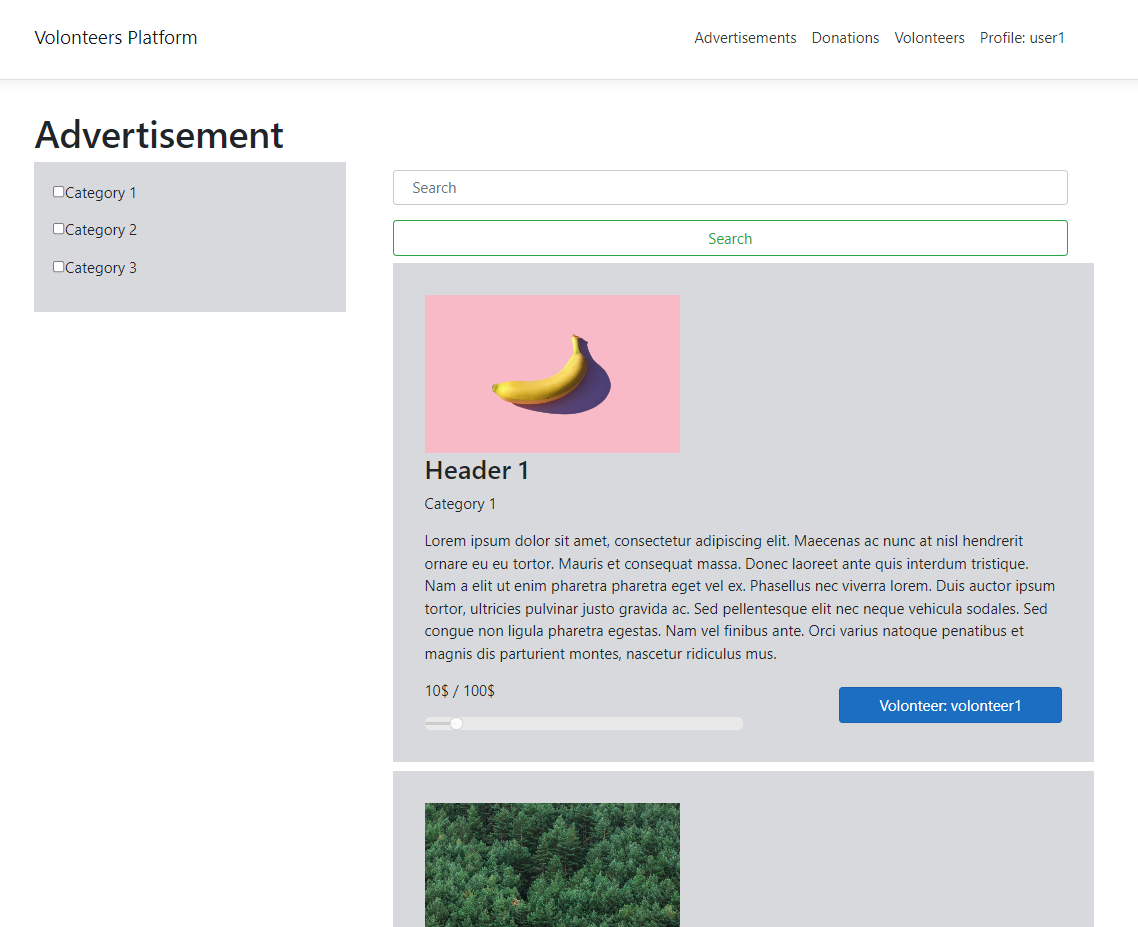
**

Рис. 3.32. Головна сторінка додатку.

У користувача такоє є сторінка профілю ([рис. 3.33](#Рис_3_33)), у ній він може вийти з аккаунту, поповнити свій рахунок для пожертвувань, та змінити у разі необхідності пароль.

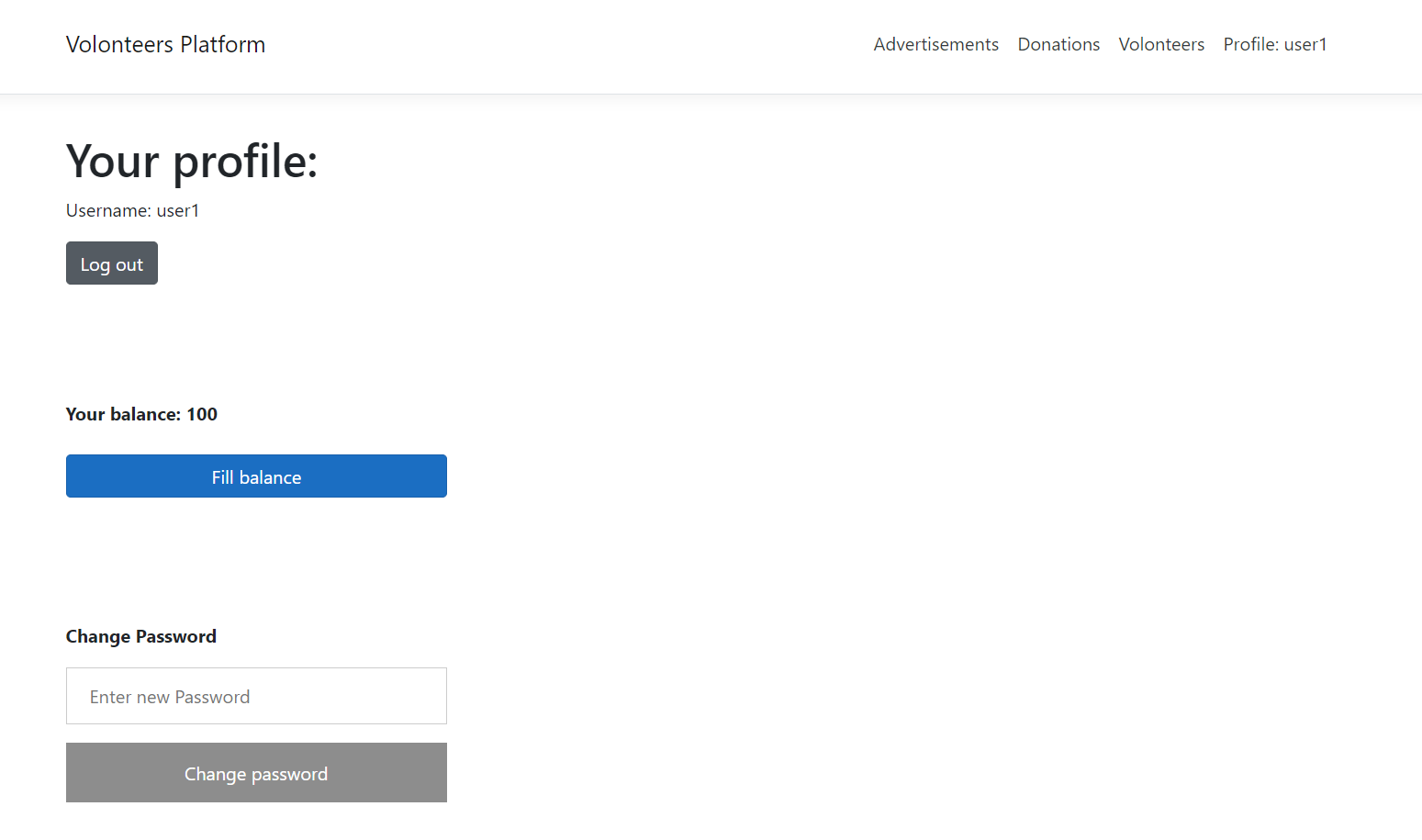
**

Рис. 3.33. Сторінка профілю користувача.

Для того аби поповнити рахунок користувача користувач може натиснути кнопку на сторінці і йому відкриється наступне вікно ([рис. 3.34](#Рис_3_34)), в якому він зможе це зробити.

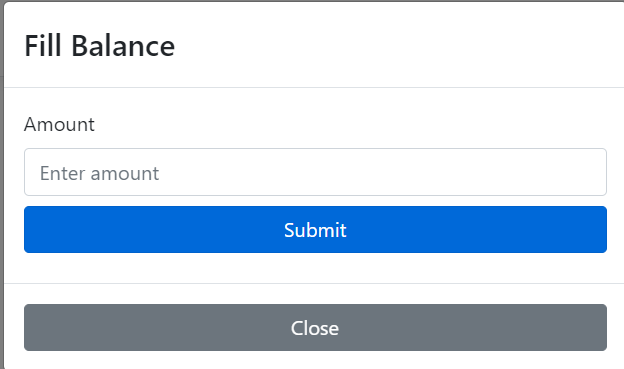


Рис. 3.34. Вікно поповнення рахунку.

Користувач, що має права адміністратора, має доступ до спеціальної сторінки менеджменту користувачів ([рис. 3.35](#Рис_3_35)), в якій він може видаляти користувачів.

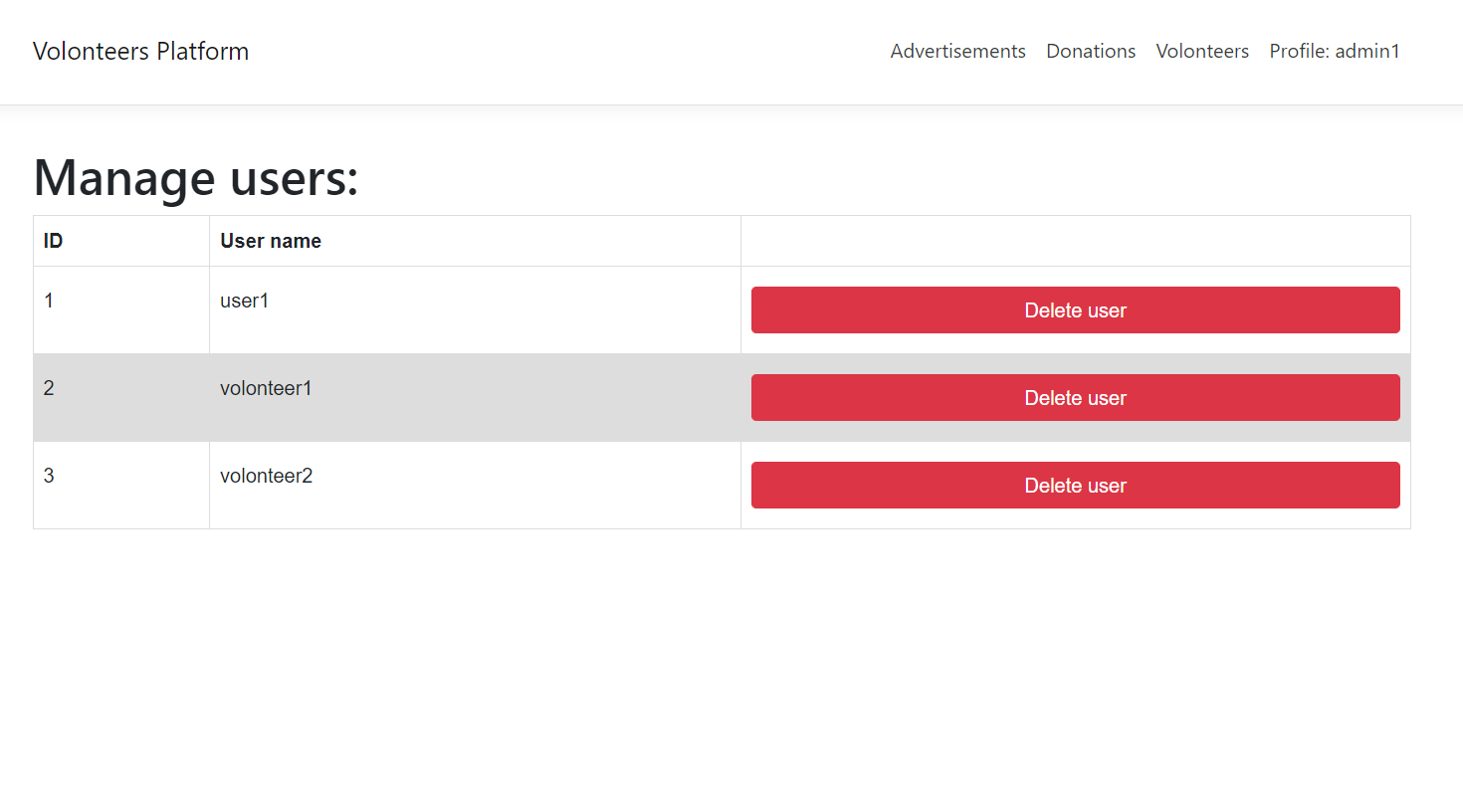


Рис. 3.35. Сторінка керування користувачами.

# 

# ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі описано повний процес розробки програмного забезпечення: проектування структури програми, розробка дизайну, різноманітні діаграми для визначення функціональності, яка мала бути реалізована.

Збудовано ефективну, масштабована архітектуру, яка добре себе показала під час розроблення великого додатку, розроблено логіку функціонування додатку, налаштовано сервер та підключено необхідні зовнішні бібліотеки.

Створено ефективний дизайн зовнішнього інтерфейсу, а також ефективно поставлено процес вхідних запитів. Розроблено клієнтський додаток, до якого був підключений фреймворк збереження даних. Підключено всі необхідні для роботи модулі. У результаті створено ефективний та сучасний веб-додаток, який виконує всі поставлені задачі.

**ВИСНОВКИ**

У результаті виконання роботи та розробки платформи для благодійних пожертвувань та волонтерської діяльності здобуто багато практичних навичок у сфері проектування та розробки програмного забезпечення. У роботі застосовано такі сучасні технології як ASP.NET, Entity Framework, Git, React.JS, Redux.JS, JSON.

Система створена для використання через веб-браузер для людей, які будь-як зацікавленні у благодійних ініціативах. Вона дає можливість створювати та переглядати оголошення, робити пожертвування на них, відслідковувати витрати, переглядати волонтерів, які створюють оголошення, та здійснювати модерацію користувачам адміністраторам. Додаток розроблений згідно з сучаними практиками розроблення веб-додатків.

Система розроблена як веб-додаток, що є перевагою в форматі платформи, оскільки користувачам не потрібно нічого завантажувати на свій комп’ютер чи телефон. Це обгрунтовано тим, що благодійні платформи це явно не те, куди пересічний користувач буде заходити щодня.

Проект розроблений згідно з сучасними стандартами розробки програмного забезпечення, дотримана висока якість коду, структура проекту є масштабованою, легкою в розширені, простою у відслідкованні помилок та несправностей, та має зрозумілі процеси обробки даних. У реальному середовищі цей проект зміг би комфортно та успішно проходити свій життєвий цикл та може мати довгий термін підтримки.

# ВИКОРИСТАНА ЛІТЕРАТУРА

1. Документація ASP.NET - [https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0.](%20https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0.)
2. Документація .NET - https://learn.microsoft.com/en-us/dotnet/
3. Документація Entity Framework - <https://learn.microsoft.com/en-us/ef/core/>
4. Документація React.js - <https://legacy.reactjs.org/docs/getting-started.html>
5. Документація Redux.js - <https://redux.js.org/>
6. Документація Visual Studio - <https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022>
7. Документація Git - <https://git-scm.com/doc>
8. Документація MSSQL - <https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16>
9. Документація HTML - <https://developer.mozilla.org/en-US/docs/Web/HTML>
10. Документація CSS - <https://developer.mozilla.org/en-US/docs/Web/CSS>
11. Документація JS - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
12. Документація npm - <https://docs.npmjs.com/>
13. Документація JSON - https://www.json.org/json-en.html
14. SOLID - <https://uk.wikipedia.org/wiki/SOLID_(%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)>
15. Шаблон «Репозиторій» - <https://uk.wikipedia.org/wiki/Repository>
16. Документація WebAPI - <https://learn.microsoft.com/en-us/aspnet/web-api/>
17. REST - https://uk.wikipedia.org/wiki/REST

# ДОДАТОК А

import 'bootstrap/dist/css/bootstrap.css';

import React from 'react';

import ReactDOM from 'react-dom';

import { BrowserRouter } from 'react-router-dom';

import App from './App';

import { Provider } from 'react-redux';

import store from "./store/store"

const baseUrl = document.getElementsByTagName('base')[0].getAttribute('href');

const rootElement = document.getElementById('root');

ReactDOM.render(

<BrowserRouter basename={baseUrl}>

<Provider store={store}>

<App />

</Provider>

</BrowserRouter>,

rootElement);

import React, { useState } from 'react';

import './auth.css';

import { Link, NavLink, useHistory } from "react-router-dom";

import API\_URL from '../../variables';

import { connect } from 'react-redux'

function Login(props) {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const history = useHistory();

const handleUsername = (event) => {

setUsername(event.target.value);

};

const handlePassword = (event) => {

setPassword(event.target.value);

};

const handleSubmit = (event) => {

event.preventDefault();

fetch('api/users/validate', {

method: 'post',

body: JSON.stringify({

Id: 0,

Name: username,

Password: password,

IsVolonteer: false,

VolonteerInfoID: 0,

Token: ''

}),

headers: {

"Content-type": "application/json; charset=UTF-8"

}

})

.then((response) => {

return response.json();

})

.then((data) => {

if (data.result === "") {

alert("Wrong password or username")

setPassword('');

return;

} else {

let user = {

username: username,

token: data.result

}

props.setCredentials(user)

history.push("/home");

}

});

};

return (

<div>

<form onSubmit={handleSubmit}>

<div className="container">

<h3>Login Form</h3>

<label htmlFor="uname"><b>Username</b></label>

<input value={username} onChange={handleUsername} type="text" placeholder="Enter Username" name="uname" required />

<label htmlFor="psw"><b>Password</b></label>

<input value={password} onChange={handlePassword} type="password" placeholder="Enter Password" name="psw" required />

<button type="submit">Login</button>

<NavLink tag={Link} className="text-dark" to="/register">Create account</NavLink>

</div>

</form>

</div>

);

}

const mapStateToProps = (state) => {

return {

user: state.user,

};

};

const mapDispatchToProps = (dispatch) => {

return {

setCredentials: (user) => dispatch({ type: 'SET\_CREDENTIALS', payload: user }),

};

};

export default connect(mapStateToProps, mapDispatchToProps)(Login);

import React, { useEffect, useState } from 'react';

import SmallAd from './SmallAd';

import './AdsList.css';

import { connect } from 'react-redux'

import { Form, FormControl, Button } from 'react-bootstrap';

function AdsList(props) {

const [ads, setAds] = useState([]);

const [searchPreQuery, setSearchPreQuery] = useState('');

const [searchQuery, setSearchQuery] = useState('');

useEffect(() => {

fetch('/api/advertisements', {

headers: {

'Cache-Control': 'no-cache',

},

})

.then((response) => {

if (!response.ok) {

throw new Error('Failed to fetch advertisements');

}

return response.json();

})

.then((data) => {

setAds(data);

})

.catch((error) => {

console.error(error);

});

}, []);

const handleSearch = (event) => {

event.preventDefault();

setSearchQuery(searchPreQuery)

};

const handleInputChange = (event) => {

setSearchPreQuery(event.target.value);

};

return (

<div className='ads'>

<Form style={{

marginTop: "0px",

marginLeft: "40px",

width: "100%"

}} onSubmit={handleSearch}>

<FormControl

type="text"

placeholder="Search"

className=""

value={searchPreQuery}

onChange={handleInputChange}

/>

<Button variant="outline-success" type="submit">Search</Button>

</Form>

<ul>

{ads.map((ad, index) => {

if( (props.categories.categories.includes(ad.category) || props.categories.categories.length === 0) &&

(searchQuery === ad.header || searchQuery === '')){

return (

<li key={index}>

<SmallAd index={index} ad={ad} />

</li>

)

}

else{

return <></>

}

})}

</ul>

</div>

);

}

const mapStateToProps = (state) => {

return {

categories: state.categories

};

};

const mapDispatchToProps = (dispatch) => {

return {

CategoriesToReducer: (categories) => dispatch({ type: 'SET\_CATEGORIES', payload: categories }),

};

};

export default connect(mapStateToProps, mapDispatchToProps)(AdsList);

using System.ComponentModel.DataAnnotations;

using System.Diagnostics.CodeAnalysis;

namespace DB.Models

{

public class User

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Password { get; set; }

public double Balance { get; set; }

public bool IsAdmin { get; set; }

public string Token { get; set; }

}

}

using DB.Models;

using Dyplom.Helpers;

using Microsoft.AspNetCore.Mvc;

using Services.Interfaces;

using Services.Services;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace WebApi.Controllers

{

[ApiController]

[Route("api/advertisements")]

public class AdvertisementController : Controller

{

private IAdvertisementRepository advertisementService;

public AdvertisementController(IAdvertisementRepository \_advertisementService)

{

advertisementService = \_advertisementService;

}

[HttpGet]

public async Task<ActionResult<List<Advertisement>>> Get()

{

return await advertisementService.Get();

}

[HttpGet]

[Route("{id}")]

public async Task<ActionResult<Advertisement>> GetById(int id)

{

return await advertisementService.GetById(id);

}

[HttpGet]

[Route("creator/{username}")]

public async Task<ActionResult<List<Advertisement>>> GetByVolonteerName(string username)

{

return await advertisementService.GetByVolonteerName(username);

}

[HttpGet]

[Route("get-photoes/{id}")]

public async Task<ActionResult<List<Photo>>> GetPhotoes(int id)

{

return await advertisementService.GetPhotoes(id);

}

[HttpPost]

public async Task<ActionResult<Advertisement>> Add(Advertisement advertisement)

{

await advertisementService.Add(advertisement);

return NoContent();

}

[HttpPut]

public async Task<ActionResult<Advertisement>> Update(Advertisement advertisement)

{

await advertisementService.Update(advertisement);

return NoContent();

}

[HttpPut]

[Route("create-transaction")]

public async Task<ActionResult<bool>> CreateTransaction(object AdUserExpense)

{

AdUserExpense typed = TypeHelper.ObjToType<AdUserExpense>(AdUserExpense);

await advertisementService.CreateTransaction(typed.Username, typed.Amount, typed.adId);

return NoContent();

}

[HttpDelete]

public async Task<ActionResult<Advertisement>> Remove(Advertisement advertisement)

{

await advertisementService.Remove(advertisement);

return NoContent();

}

}

}

using DB.DB\_Context;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.SpaServices.ReactDevelopmentServer;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Services.Interfaces;

using Services.Services;

namespace KursachReact

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

string connection = Configuration.GetConnectionString("DefaultConnection");

services.AddDbContext<ApplicationContext>(options =>

options.UseSqlServer(connection));

services.AddScoped<IUserRepository, UserService>();

services.AddScoped<IAdvertisementRepository, AdvertisementService>();

services.AddScoped<IExpenseRepository, ExpenseService>();

services.AddScoped<IVolonteerInfoRepository, VolonteerInfoService>();

services.AddScoped<ICategoryRepository, CategoryService>();

services.AddControllersWithViews();

// In production, the React files will be served from this directory

services.AddSpaStaticFiles(configuration =>

{

configuration.RootPath = "ClientApp/build";

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseSpaStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller}/{action=Index}/{id?}");

});

app.UseSpa(spa =>

{

spa.Options.SourcePath = "ClientApp";

if (env.IsDevelopment())

{

spa.UseReactDevelopmentServer(npmScript: "start");

}

});

}

}

}

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace KursachReact

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

using DB.Models;

namespace Services.Interfaces

{

public interface IAdvertisementRepository

{

public Task<List<Advertisement>> Get();

public Task<Advertisement> GetById(int id);

public Task<List<Advertisement>> GetByVolonteerName(string username);

public Task<List<Photo>> GetPhotoes(int id);

public Task Add(Advertisement advertisement);

public Task Update(Advertisement advertisement);

public Task<bool> CreateTransaction(string Username, double Amount, int adId);

public Task Remove(Advertisement advertisement);

}

}

using System.Collections.Generic;

using System.Linq;

using DB.DB\_Context;

using DB.Models;

using Microsoft.EntityFrameworkCore;

using Services.Interfaces;

namespace Services.Services

{

public class AdvertisementService : IAdvertisementRepository

{

private readonly ApplicationContext db;

public AdvertisementService(ApplicationContext context)

{

db = context;

}

public async Task<List<Advertisement>> Get()

{

return await db.Advertisements.ToListAsync();

}

public async Task<Advertisement> GetById(int id)

{

return await db.Advertisements.Where(c => c.ID == id).SingleAsync();

}

public async Task<List<Advertisement>> GetByVolonteerName(string username)

{

var user = await db.Users.SingleAsync(c => c.Name == username);

var volonteerInfo = await db.VolonteerInfos.SingleAsync(c => c.UserId == user.Id);

return await db.Advertisements.Where(c => c.VolonteerInfoID == volonteerInfo.Id).ToListAsync();

}

public async Task<List<Photo>> GetPhotoes(int id)

{

return await db.Photoes.Where(c => c.AdID == id).ToListAsync();

}

public async Task Add(Advertisement advertisement)

{

await db.Advertisements.AddAsync(advertisement);

db.SaveChanges();

}

public async Task Update(Advertisement advertisement)

{

db.Advertisements.Update(advertisement);

await db.SaveChangesAsync();

}

public async Task<bool> CreateTransaction(string Username, double Amount, int adId)

{

try

{

var ad = await db.Advertisements.Where(c => c.ID == adId).SingleAsync();

ad.CollectedSum += Amount;

var user = await db.Users.Where(c => c.Name == Username).SingleAsync();

if (user.Balance >= Amount)

{

user.Balance -= Amount;

}

else

{

throw new Exception();

}

db.Expenses.Add(new Expense

{

Amount = Amount,

Date = DateTime.Now,

UserId = user.Id,

AdId = ad.ID

});

db.SaveChanges();

return true;

}

catch (Exception ex)

{

return false;

}

}

public async Task Remove(Advertisement advertisement)

{

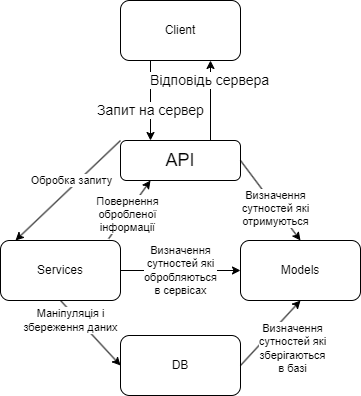
db.Advertisements.Remove(advertisement);

await db.SaveChangesAsync();

}

}

}



Бакалаврська кваліфікаційна робота

Структура веб-додатку

Бакалаврська кваліфікаційна робота

Система

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Погуляєв В.В.

Консул.

Н.Контр.

Зав. каф.

Лобур М.В.

Аркуш 1

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409

Станкевич О.М.



Бакалаврська кваліфікаційна робота

Use-case діаграма

Бакалаврська кваліфікаційна робота

Процес

«Надсилання документу»

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Погуляєв В.В.

Консул.

Н.Контр.

Зав. каф.

Лобур М.В.

Аркуш 2

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409

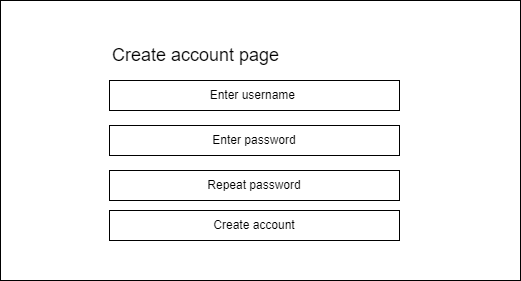
Станкевич О.М.

Станкевич В.В.



Погуляєв В.В.





Бакалаврська кваліфікаційна робота

Шаблон сторінки реєстрації

Бакалаврська кваліфікаційна робота

Система

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Консул.

Н.Контр.

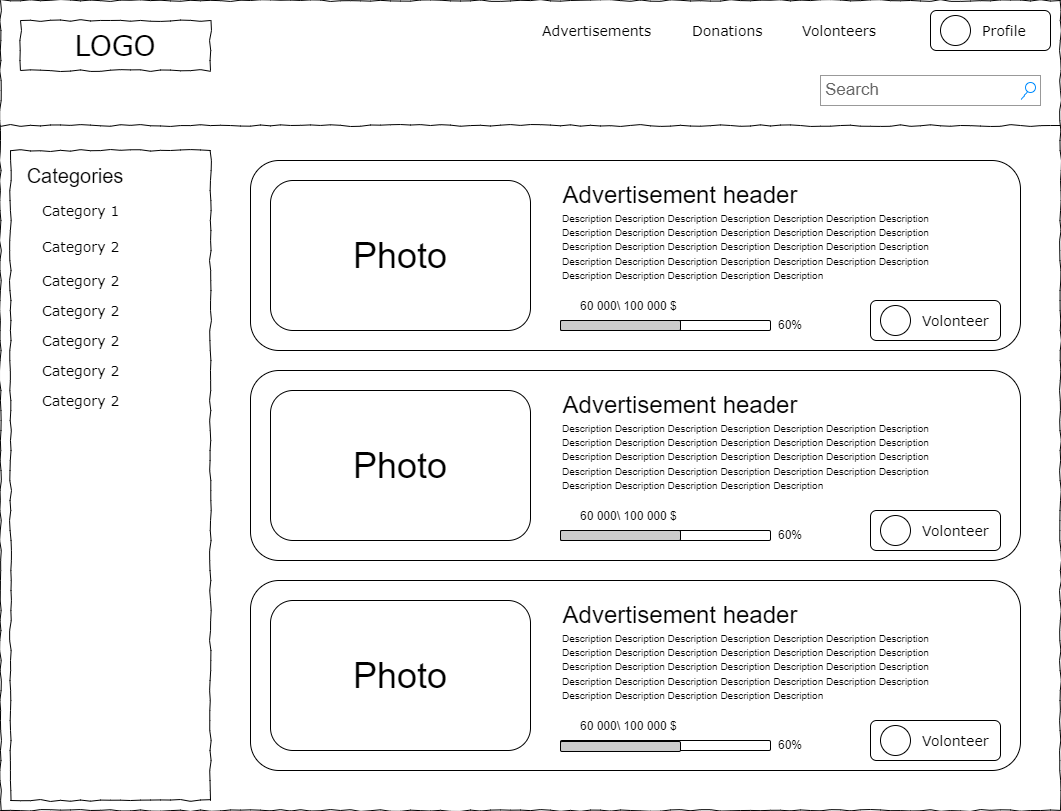
Зав. каф.

Лобур М.В.

Аркуш 3

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409



Бакалаврська кваліфікаційна робота

Шаблон головної

сторінки додатку

Бакалаврська кваліфікаційна робота

Система

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Погуляєв В.В.

Консул.

Н.Контр.

Зав. каф.

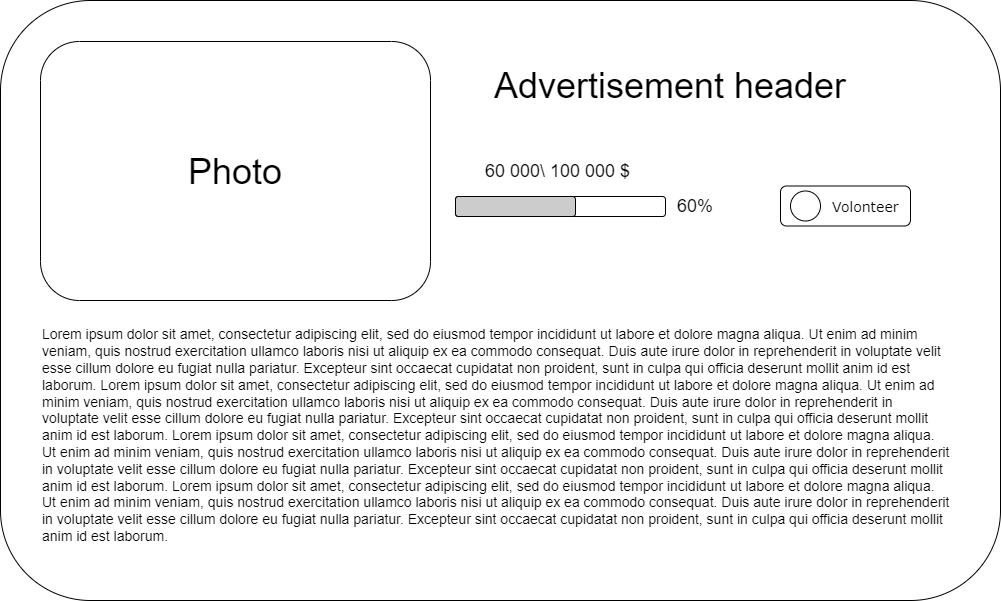
Лобур М.В.

Аркуш 4

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409

Станкевич О.М.



Бакалаврська кваліфікаційна робота

Шаблон оголошення

Бакалаврська кваліфікаційна робота

Система

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Погуляєв В.В.

Консул.

Н.Контр.

Зав. каф.

Лобур М.В.

Аркуш 5

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409

Станкевич О.М.



Бакалаврська кваліфікаційна робота

Шаблон сторінки керування

користувачами

Бакалаврська кваліфікаційна робота

Система

# . Бакалаврська кваліфікаційна робота

Архітектура програмного  
 застосунку

Підпис

Дата

Масштаб

Керівн.

Маса

Літера

Зм

Лист

Розроб.

Погуляєв В.В.

Консул.

Н.Контр.

Зав. каф.

Лобур М.В.

Аркуш 6

Аркушів 6

НУ “ЛП”, ІКНІ, каф. САП  
гр. КН-409

Станкевич О.М.