

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра «Системний аналіз та обчислювальна математика»
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

– бакалавра
(ступінь вищої освіти)

на тему:

«Розробка бази даних сайту громадської організації»
(назва теми)

Виконав: студент 4 курсу, групи КНТ-811СП

Спеціальності 124 – «Системний аналіз»
(код і назва напрямку підготовки, спеціальності)

Освітня програма (спеціалізація)
«Інтелектуальні технології та прийняття рішень
в складних системах» _____

ГРИНЕВИЧ В.С.
(ПРІЗВИЩЕ та ініціали)

Керівник САВРАНСЬКА А.В.
(ПРІЗВИЩЕ та ініціали)

Рецензент ЛОЗОВСЬКА М.О.
(ПРІЗВИЩЕ та ініціали)

м. Запоріжжя

2024 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет «Факультет комп'ютерних наук і технологій»

Кафедра «Системний аналіз та обчислювальна математика»

Ступінь вищої освіти Бакалавр

Спеціальність 124 – «Системний аналіз»

(код і найменування)

Освітня програма (спеціалізація) «Інтелектуальні технології та прийняття рішень в складних системах»

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

**В.о. завідувача кафедри «Системний аналіз
та обчислювальна математика»**

канд. фіз.-мат. наук, _____

доц. Терещенко Еліна Валентинівна

« _____ » _____ 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГРИНЕВИЧА Валерія Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) «Розробка бази даних сайту громадської організації»
керівник проєкту (роботи) к.ф.-м.н., доцент, Савранська Алла

Володимирівна

затверджені наказом закладу вищої освіти від «__» __ 20__ року №__

2. Строк подання студентом проєкту (роботи) 2 червня 2024 року

3. Вихідні дані до проєкту (роботи) PHPStorm(IDE), PHP, Laravel, MySQL, JWT, GitHub, OOP, MVC, LaravelRoute.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дипломна робота присвячена створенню бази даних для сайту громадської організації. У вступі обґрунтовано актуальність теми, мету, об'єкт, предмет досліджень і завдання. У першому розділі розповідається про зустріч з замовником та отримання технічного завдання та планування архітектури проєкту. У другому розділі описано готова архітектура back-end частини та баз даних. У третьому розділі розглянуто підтримку та розвиток сайту, та контроль версій у GitHub.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) _____

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Теоретична частина	САВРАНСЬКА А.В.	11 січня 2024	
Практична частина	ЄСІНА О. А. Керівник Відокремленого підрозділу "Запорізька асоціація територіальних громад ГО "Солідарна Справа Громад"	19 січня 2024	
Нормоконтроль	ШИРОКОРАД Д.В.		

7. Дата видачі завдання «19» січня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	19 січня 2024	
2	Вивчення референсів	27 січня 2024	
3	Обрання технологій	10 лютого 2024	
4	Розробка архітектури back-end частини	24 березня 2024	
5	Розробка архітектури бази даних	17 квітня 2024	
6	Написання основних кінцевих точок	20 квітня 2024	
7	Оптимізація роботи застосунку	31 квітня 2024	
8	Написання unit-тестів	9 травня 2024	
9	Інтеграція с front-end	17 травня 2024	
10	Демонстрація замовнику	20 травня 2024	
11	Корегування функціоналу	27 травня 2024	
12	Подання готової роботи	2 червня 2024	

Студент(ка) _____ Валерій ГРИНЕВИЧ

Керівник проєкту (роботи) _____ Алла САВРАНСЬКА

РЕФЕРАТ

Пояснювальна записка: 48 с., 6 рис., 2 додатки, 41 джерел.

Об'єктом дослідження в роботі є процес розробки бази даних для веб-сайту громадської організації.

Предметом дослідження є технології створення бази даних, розробка архітектури проєкту, методи інтеграції компонентів та підтримка бази даних веб-сайту громадської організації.

Мета досліджень: обґрунтування теоретичних і практичних аспектів створення бази даних для веб-сайту громадських організацій та розробка практичних рекомендацій щодо її вдосконалення і підтримки.

У теоретичній частині проведено огляд останніх досліджень та публікацій з питань розробки баз даних, архітектури проєктів та інтеграції компонентів. Наведено сутність технологій баз даних, описано різні підходи до створення зручних та ефективних баз даних, їх аналіз та ефективність.

У *практичній частині* розроблено архітектуру проєкту та бази даних для веб-сайту громадської організації. Запропоновано структуру бази даних, реалізовано ключові функції та інтеграцію з клієнтською частиною, створено надійний та зручний для користувачів інтерфейс взаємодії. В даному розділі розроблено модель підтримки та розвитку бази даних, а також план впровадження нових функцій для підвищення ефективності роботи організації.

Наукова новизна полягає у розробці бази даних веб-сайту громадської організації з використанням сучасних технологій та підходів до архітектури проєктів.

Практична цінність отриманих результатів дипломної роботи полягає у створенні бази даних, яка дозволяє покращити комунікацію з громадськістю, залучити нових волонтерів та підвищити рівень довіри з боку суспільства.

Ключові слова: БАЗА ДАНИХ, ГРОМАДСЬКА ОРГАНІЗАЦІЯ,
LARAVEL, ВЕБ-ТЕХНОЛОГІЇ, ПІДТРИМКА ВЕБ-САЙТУ, ІНТЕГРАЦІЯ,
РОЗВИТОК ВЕБ-САЙТ

ABSTRACT

Explanatory note: 48 p., 6 figures, 2 appendices, 41 sources.

The object of research in this paper is the process of developing a database for a public organization's website.

The subject of research is the technologies for creating a database, developing a project architecture, methods for integrating components, and maintaining a database for a public organization's website.

The purpose of the research: to substantiate the theoretical and practical aspects of creating a database for a public organization's website and to develop practical recommendations for its improvement and support.

In the theoretical part, a review of recent research and publications on database development, project architecture, and component integration is conducted. The essence of database technologies is presented, and various approaches to creating convenient and efficient databases, their analysis, and effectiveness are described. In the practical part, the project architecture and database for a public organization's website are developed. The database structure is proposed, key functions and integration with the client side are implemented, and a reliable and user-friendly interaction interface is created. This section develops a model for maintaining and developing the database, as well as a plan for implementing new functions to improve the organization's efficiency.

The scientific novelty is the development of a database for a public organization's website using modern technologies and approaches to project architecture.

The practical value of the results of the thesis is the creation of a database that allows to improve communication with the public, attract new volunteers, and increase the level of public trust.

Keywords: DATABASE, PUBLIC ORGANIZATION, LARAVEL, WEB TECHNOLOGIES, WEBSITE SUPPORT, INTEGRATION, WEBSITE DEVELOPMENT.

ЗМІСТ

Завдання.....	2
РЕФЕРАТ.....	4
ABSTRACT.....	6
ВСТУП.....	8
РОЗДІЛ 1 ПЛАНУВАННЯ ТА ПРОЄКТУВАННЯ.....	10
1.1 Технічне завдання (ТЗ).....	10
1.2 Визначення використовуваних технологій.....	11
1.3 Планування роботи над проєктом.....	13
РОЗДІЛ 2 РОЗРОБКА АРХІТЕКТУРИ ПРОЄКТУ ТА БАЗИ ДАНИХ.....	15
2.1 Архітектура проєкту.....	15
2.2 Архітектура бази даних.....	18
2.3 Створення API.....	21
РОЗДІЛ 3 ТЕСТУВАННЯ ТА ПІДТРИМКА.....	28
3.1 Тестування.....	24
3.2 Контроль версій.....	26
3.3 Підтримка та обслуговування.....	30
ВИСНОВКИ.....	33
ПЕРЕЛІК ПОСИЛАНЬ.....	36
ДОДАТОК А Приклад коду Laravel моделі	40
ДОДАТОК Б Приклад коду Laravel Controller.....	42

ВСТУП

В сучасному світі інформаційні технології відіграють ключову роль у функціонуванні та розвитку громадських організацій. З огляду на значне зростання впливу Інтернету, веб-сайти стали невід'ємною частиною комунікації, залучення волонтерів, збору пожертвувань та поширення інформації про діяльність таких організацій. Створення ефективного та функціонального веб-сайту є важливим інструментом для досягнення цілей громадської організації, зокрема у забезпеченні прозорості, доступності та інтерактивності для всіх зацікавлених сторін.

Мета цієї дипломної роботи полягає у розробці бек-енд частини веб-сайту для громадської організації, яка дозволить автоматизувати та спростити управління інформацією, що розміщується на сайті, а також забезпечити високий рівень безпеки та продуктивності. Бек-енд буде критичною складовою веб-сайту, оскільки саме він відповідатиме за обробку даних, інтеграцію з базами даних, обробку запитів користувачів та забезпечення коректної роботи фронт-енд частини.

У рамках цієї роботи буде проведено детальний аналіз потреб громадської організації, визначено функціональні та нефункціональні вимоги до веб-сайту. На основі цього буде сплановано архітектуру бек-енд частини, з урахуванням масштабованості, надійності та безпеки. Аналіз вимог включатиме дослідження поточних потреб організації, їхніх пріоритетів, а також аналіз аналогічних рішень на ринку, що дозволить визначити оптимальний підхід до розробки. На підставі цього аналізу буде створено технічне завдання та визначено ключові етапи реалізації проєкту.

Буде детально описано процес розробки, включаючи створення RESTful API[1] для взаємодії з фронт-ендом, реалізацію системи автентифікації та авторизації користувачів, інтеграцію з базою даних та реалізацію логіки обробки запитів. Особлива увага приділятиметься забезпеченню безпеки даних

користувачів, включаючи захист від потенційних загроз. В процесі розробки будуть застосовані принципи модульності та розширюваності, що дозволить легко адаптувати систему під нові вимоги та функціональні можливості.

Буде описано методи та інструменти, які будуть використані для тестування розробленої системи, зокрема юніт-тести, інтеграційні тести та тестування безпеки. Буде забезпечено високий рівень надійності та відмовостійкості системи шляхом впровадження автоматизованих тестів та безперервної інтеграції. Це дозволить оперативно виявляти та виправляти помилки, а також мінімізувати ризики збоїв у роботі системи.

Буде розглянуто процеси розгортання розробленої бек-енд частини на сервері, налаштування середовища виконання, а також подальша підтримка та оновлення системи. Буде описано використання сучасних підходів для забезпечення стабільності та портативності розробленого рішення, а також застосування інструментів моніторингу та логування для відстеження стану системи в реальному часі.

Ця дипломна робота спрямована на створення ефективного та надійного бек-енд рішення, яке допоможе громадській організації досягти своїх цілей, забезпечити високу продуктивність та зручність у використанні веб-сайту. Основна увага буде приділена питанням безпеки, масштабованості та підтримки в умовах змінного навантаження. Таким чином, результатом роботи стане сучасний та функціональний веб-сайт, здатний задовольнити потреби громадської організації та її аудиторії. Успішна реалізація цього проєкту сприятиме підвищенню ефективності діяльності організації, покращенню комунікації з громадськістю та залученню нових ресурсів для реалізації соціально значущих ініціатив.

РОЗДІЛ 1 Планування та проєктування

1.1 Технічне завдання (ТЗ)

Сайт надаватиме можливість користувачам зареєструватися та увійти до системи, робити донати, ділитися інформацією про волонтерську діяльність, читати новини та переглядати звіти. Основна аудиторія цього сайту – користувачі, які бажають підтримувати волонтерські проєкти та стежити за їх діяльністю.

Наш веб-сайт створено для того, щоб об'єднати людей, які хочуть допомагати іншим, робити благодійні внески та активно брати участь у волонтерських проєктах. На нашому сайті ви зможете зареєструватися та створити свій обліковий запис, що дозволить вам увійти до системи і скористатися всіма доступними функціями. Реєстрація проста та швидка, а логін надасть вам доступ до персоналізованих налаштувань та історії вашої діяльності на платформі.

Однією з ключових функцій сайту є можливість робити донати. Ми надаємо безпечний і зручний спосіб підтримати волонтерські проєкти та благодійні ініціативи. Ваші внески допоможуть нам робити світ кращим, підтримуючи тих, хто цього потребує.

Користувачі також зможуть ділитися інформацією про свою волонтерську діяльність. Це дозволить створити активну спільноту, де кожен може показати свій внесок і надихати інших на добрі справи. Ви зможете розповідати про свої проєкти, ділитися фото та відео, а також обмінюватися досвідом з іншими волонтерами.

Наш розділ новин триматиме вас в курсі останніх подій у світі волонтерства та благодійності. Тут ви знайдете актуальні новини, статті та огляди, що стосуються волонтерської діяльності. Ми прагнемо, щоб ви завжди були проінформовані про те, що відбувається у нашій спільноті та за її межами.

Окрім цього, користувачі зможуть переглядати детальні звіти про роботу волонтерських проєктів та використання донатів. Прозорість і відкритість – наші пріоритети, тому ми надаємо вичерпну інформацію про те, як використовуються ваші внески і які результати досягнуто завдяки вашій підтримці.

Ми впевнені, що наш сайт стане цінним ресурсом для всіх, хто хоче зробити світ кращим місцем. Приєднуйтеся до нашої спільноти, підтримуйте волонтерські проєкти і разом ми зможемо досягти значних змін.

1.2 Визначення використовуваних технологій

Для реалізації даного проєкту обрано такі технології:

Мова програмування: PHP

PHP (Hypertext Preprocessor) - це серверна мова програмування, спеціально створена для веб-розробки, яка вбудовується в HTML і використовується для створення динамічних веб-сторінок та серверних додатків.

PHP є популярною мовою програмування для веб-розробки, яка дозволяє створювати динамічні веб-сторінки та серверні додатки. Її популярність обумовлена простотою використання, великою спільнотою розробників та широкою підтримкою різних серверів і баз даних.

Фреймворк: Laravel

Фреймворк – це платформа для розробки програмного забезпечення, що забезпечує структуру та набір інструментів для швидкого створення додатків. Вони допомагають стандартизувати процес розробки, полегшують підтримку коду та дозволяють швидко впроваджувати нові функції.

Laravel[2] - це фреймворк на мові програмування PHP, створений для розробки веб-додатків з метою спрощення повторюваних задач і забезпечення більш чистого та зрозумілого коду.

Laravel забезпечує зручний синтаксис, модульність і потужний набір інструментів для розробки веб-додатків. Він має вбудовану підтримку для роботи з базами даних, маршрутизації, автентифікації та інших важливих аспектів веб-розробки.

База даних: MySQL

MySQL[3] - це система керування реляційними базами даних з відкритим кодом, що використовується для зберігання і управління даними з високою продуктивністю та надійністю.

MySQL є однією з найпопулярніших систем керування реляційними базами даних (РСКБД), яка забезпечує високу продуктивність і надійність. Вона добре інтегрується з Laravel і забезпечує швидку обробку запитів до бази даних.

Система керування версіями: Git

Git - це розподілена система керування версіями, яка використовується для відстеження змін у вихідному коді під час розробки програмного забезпечення. Вона дозволяє кільком розробникам працювати над одним проєктом одночасно, забезпечуючи зручний механізм злиття та розгалуження змін.

Git є розподіленою системою керування версіями, яка дозволяє розробникам відстежувати зміни у коді, співпрацювати над проєктами і зберігати історію змін. Вона забезпечує надійний спосіб зберігання різних версій проєкту та повернення до попередніх станів у разі потреби.

Безпека: JWT(JSON Web Token)

JWT (JSON Web Token)[4] - це компактний, URL-безпечний засіб для передачі об'єктів JSON між сторонами з використанням цифрового підпису. Він часто використовується для автентифікації та авторизації в веб-додатках, забезпечуючи безпечний і ефективний спосіб обміну інформацією.

JWT (JSON Web Token) є стандартом для створення токенів, що використовуються для автентифікації і передачі інформації між сторонами у вигляді JSON об'єктів. Ці токени містять закодовану інформацію і підписані для забезпечення цілісності та аутентичності даних.

1.3 Планування роботи над проєктом

Планування роботи над проєктом є важливим етапом, який визначає загальну структуру, послідовність і терміни виконання завдань. Це забезпечує організовану та ефективну роботу команди, а також допомагає уникнути затримок і перевитрат. Планування включає кілька ключових аспектів:

Розбиття проєкту на етапи

Першим кроком у плануванні є розбиття проєкту на чітко визначені етапи. Кожен етап має конкретні цілі та завдання, які необхідно виконати:

- **Збір вимог:** На цьому етапі відбувається детальний збір вимог від замовника та кінцевих користувачів. Важливо зрозуміти потреби та очікування всіх зацікавлених сторін, що дозволяє створити точне технічне завдання.
- **Проектування:** На етапі проектування розробляється архітектура системи, вибираються технології та створюються UML-діаграми. Це забезпечує чітке бачення структури додатка та його компонентів.
- **Реалізація:** Етап реалізації включає безпосереднє написання коду згідно з визначеною архітектурою та вимогами. Цей етап може бути розділений на підетапи, такі як розробка окремих модулів, інтеграція з зовнішніми сервісами тощо.
- **Тестування:** Після реалізації слідує етап тестування, де перевіряється коректність роботи всіх компонентів системи. Включає юніт-тести, інтеграційні та функціональні тести.
- **Розгортання:** Завершальний етап включає розгортання системи на продуктивному середовищі, конфігурацію серверів, налаштування бази даних та інших компонентів.
- **Підтримка:** Після розгортання проєкт переходить у стадію підтримки, де здійснюється моніторинг, виправлення помилок та оновлення.

Складання графіка робіт

Для ефективного управління проєктом необхідно скласти детальний графік робіт, який включає терміни виконання кожного етапу та ключових завдань:

- Оцінка тривалості етапів: Визначення реалістичних термінів виконання кожного етапу, враховуючи складність завдань та ресурси команди.
- Контрольні точки: Встановлення контрольних точок (milestones), які допомагають оцінювати прогрес проєкту та своєчасно виявляти відхилення від плану. Наприклад, завершення збору вимог, завершення проєктування, завершення розробки основних модулів тощо.
- Гнучкість графіка: Врахування можливих ризиків та непередбачуваних обставин, які можуть вплинути на терміни виконання. Це дозволяє оперативно коригувати графік у разі необхідності.

Використання інструментів управління проєктом

Для організації роботи та контролю за виконанням завдань використовуються спеціалізовані інструменти управління проєктами:

- Системи управління завданнями: Інструменти, такі як Jira, Trello або Asana, дозволяють створювати завдання, призначати їх відповідальним, відстежувати статус виконання та пріоритезувати роботу.
- Системи контролю версій: Git використовується для відстеження змін у коді, забезпечення співпраці між розробниками та управління версіями проєкту.
- Інструменти для комунікації: Використання таких інструментів, як Slack або Microsoft Teams, для забезпечення ефективної внутрішньої комунікації та швидкого обміну інформацією.

Ці аспекти планування роботи над проєктом дозволяють забезпечити його успішне виконання в межах визначених термінів та бюджету, а також забезпечують високу якість та надійність кінцевого продукту.

РОЗДІЛ 2 Розробка архітектури проєкту та бази даних

2.1 Архітектура проєкту

У проєкті було використано класове програмування, яке є основою для організації коду. Ми активно застосовували концепції та принципи об'єктно-орієнтованого програмування[5] (ООП), що забезпечило модульність, повторне використання коду та легкість його підтримки.

Основні принципи ООП, які використовувались при розробці проєкту, включають:

- Інкапсуляція[6] – об'єднання даних і методів, що працюють з цими даними, в межах одного класу. Це дозволило нам захистити внутрішній стан об'єкта і забезпечити доступ до нього тільки через визначені методи.
- Наслідування[7] – створення нових класів на основі існуючих, що сприяло повторному використанню коду і полегшило розширення функціональності. Завдяки наслідуванню ми могли створювати більш спеціалізовані класи, які успадковують властивості і поведінку загальних класів.
- Поліморфізм[8] – використання однакових методів для різних об'єктів, що дозволило нам використовувати об'єкти різних класів через єдиний інтерфейс. Це значно спростило структуру коду і підвищило його гнучкість.

Крім того, у проєкті були використані певні паттерни проєктування, які допомагають вирішувати типові задачі у розробці програмного забезпечення та сприяють створенню більш стійкої та масштабованої архітектури:

- Singleton – забезпечення того, щоб клас мав лише один екземпляр, і надання глобальної точки доступу до цього екземпляра. Це було корисно для управління єдиним ресурсом, таким як конфігурація додатку.
- Factory – створення об'єктів без вказання конкретного класу об'єкта, який буде створено. Цей паттерн допоміг нам створювати об'єкти більш гнучко і уникати жорсткої прив'язки коду до конкретних класів.

У архітектурі також активно використовували паттерни проєктування, притаманні Laravel, що забезпечують зручність і ефективність розробки:

- **Service Container (Inversion of Control)** – механізм управління залежностями і впровадження залежностей у класи. Це дозволило нам легше керувати залежностями між класами і спростити тестування коду.
- **Service Providers** – класи, що реєструють сервіси і виконують початкову конфігурацію додатку. Вони допомогли нам організувати процес завантаження і налаштування компонентів додатку.
- **Middleware** – механізм для обробки HTTP-запитів на шляху до контролерів, що дозволяє реалізувати різноманітні фільтри і перевірки. Використання Middleware дало нам можливість зручно реалізувати перевірки авторизації, логування та інші аспекти обробки запитів.

Ще однією важливою частиною архітектури нашого проєкту є використання шаблону проєктування MVC (Model-View-Controller)[9] у фреймворку Laravel. Цей шаблон розділяє додаток на три основні компоненти:

- **Model (Модель)** – представляє дані додатку і бізнес-логіку. Усі запити до бази даних та маніпуляції з даними здійснюються через моделі, що забезпечує централізацію доступу до даних. Як приклад реалізації добавлено код у **Додатку А**

- **View (Представлення)** – відповідає за відображення даних користувачу. Представлення отримують дані з контролерів і формують HTML-сторінки, які бачить користувач.

- **Controller (Контролер)** – обробляє запити користувача, взаємодіє з моделлю для отримання даних і передає ці дані у представлення. Контролери виконують роль посередника між моделлю і представленням, забезпечуючи логіку додатку. Як приклад реалізації добавлено код у **Додатку В**

Завдяки використанню шаблону MVC, наш проєкт став більш структурованим і легким для розуміння та підтримки. Це дозволило чітко розділити відповідальності між різними компонентами додатку, полегшуючи роботу над проєктом та забезпечуючи його гнучкість і розширюваність.

Завдяки використанню цих концепцій, принципів та паттернів, проєкт є стійким, гнучким і легким для розширення та підтримки. Це дозволяє нам ефективно розвивати проєкт, додаючи нові функціональні можливості та підтримуючи високу якість коду.

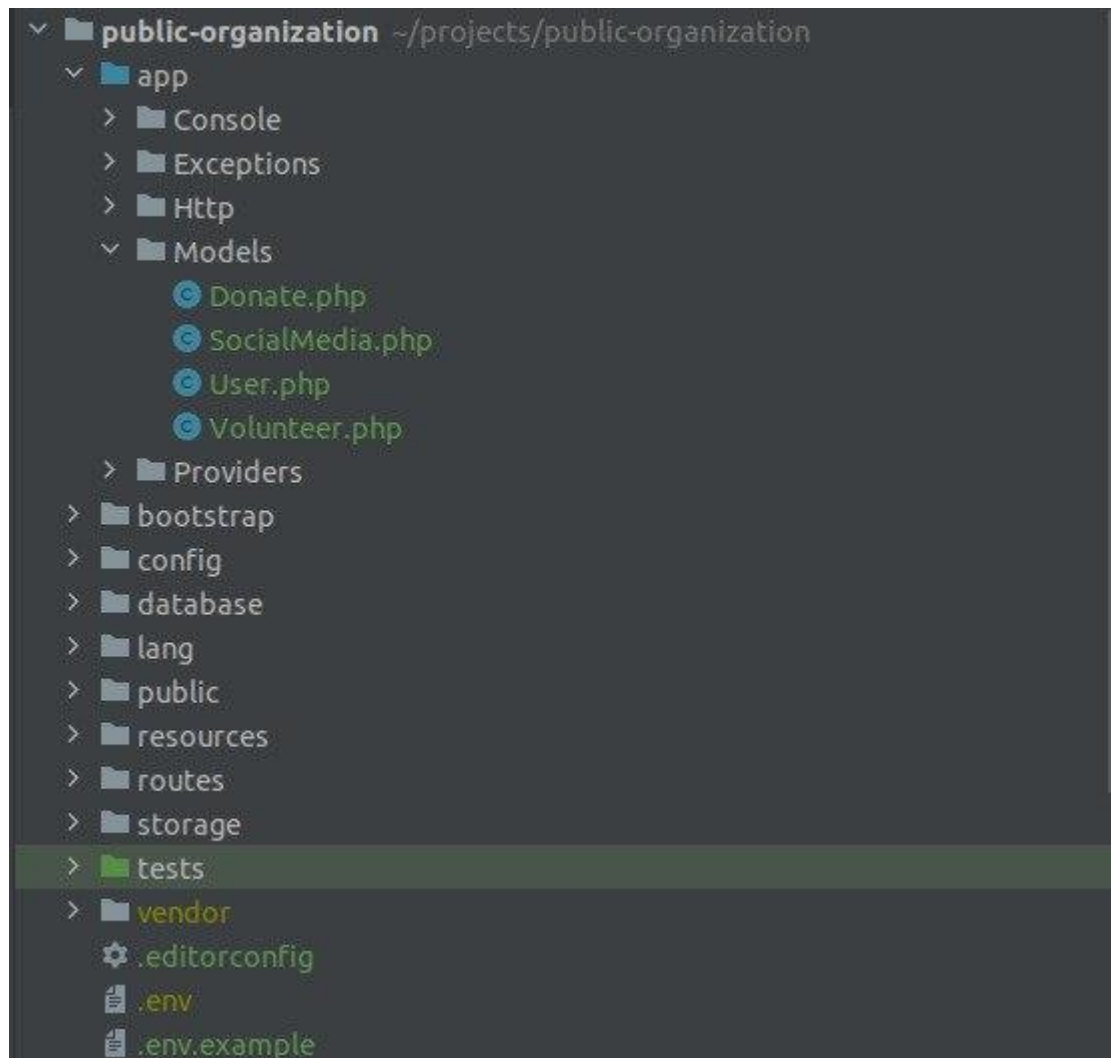


Рисунок 2.1 – Архітектура проєкта

2.2 Архітектура бази даних

Для проєкту ми обрали MySQL як основну систему керування базою даних (СКБД). MySQL є однією з найпопулярніших і надійних реляційних СКБД, що забезпечує високу продуктивність і масштабованість. Вона ідеально підходить для додатків, які вимагають швидкої обробки запитів і надійного зберігання даних.

Використання MySQL:

MySQL дозволяє нам ефективно зберігати, організовувати і отримувати дані. Це особливо важливо для додатків з великою кількістю користувачів і значним обсягом даних. Система підтримує складні SQL-запити, транзакції та індекси, що сприяє швидкій обробці даних і оптимізації запитів.

Для взаємодії з базою даних у проєкті ми використовували Eloquent ORM[5], який є стандартним ORM-фреймворком у Laravel. Eloquent забезпечує простий і інтуїтивно зрозумілий синтаксис для роботи з базою даних, дозволяючи зосередитися на бізнес-логіці додатку, а не на написанні складних SQL-запитів.

Основні особливості Eloquent:

- **Активні записи (Active Record):** Eloquent реалізує паттерн "Active Record", що дозволяє кожному об'єкту моделі відповідати одному запису в базі даних. Це значно спрощує процес створення, читання, оновлення та видалення (CRUD)[7] записів.
- **Зв'язки між моделями:** Eloquent дозволяє легко визначати зв'язки між різними моделями, такими як "один до одного", "один до багатьох" та "багато до багатьох". Це забезпечує зручну роботу з пов'язаними даними і дозволяє здійснювати складні вибірки з мінімальними зусиллями.
- **Пошук і фільтрація:** Eloquent надає потужні методи для фільтрації даних, що дозволяє легко знаходити потрібні записи за різними критеріями.

Завдяки цьому ми можемо швидко і ефективно отримувати потрібну інформацію з бази даних.

- **Міграції:** Міграції в Laravel дозволяють легко керувати схемою бази даних, створювати та змінювати таблиці, індекси і зв'язки між ними. Це забезпечує зручність і контроль версій бази даних, що особливо важливо для командної розробки.

Структура бази даних:

- **Таблиці:** У нашій базі даних визначені таблиці, які відповідають основним об'єктам додатку, таким як користувачі, проєкти, донати та новини. Кожна таблиця має відповідні стовпці для зберігання атрибутів цих об'єктів.

- **Зв'язки:** Між таблицями встановлені зв'язки, що відображають відношення між об'єктами. Наприклад, таблиця "користувачі" має зв'язок "один до багатьох" з таблицею "донати", що дозволяє кожному користувачеві мати кілька донатів.

- **Індекси:** Для оптимізації запитів і підвищення продуктивності були створені індекси на ключових стовпцях, таких як первинні ключі та зовнішні ключі.

Використання MySQL разом з Eloquent ORM дозволило нам створити надійну, ефективну та масштабовану архітектуру бази даних. Це забезпечило швидку обробку запитів, зручність управління даними і легкість інтеграції з іншими компонентами додатку. Завдяки цьому ми змогли створити потужну основу для нашого проєкту, яка підтримує всі необхідні функціональні можливості і забезпечує високий рівень продуктивності та надійності.

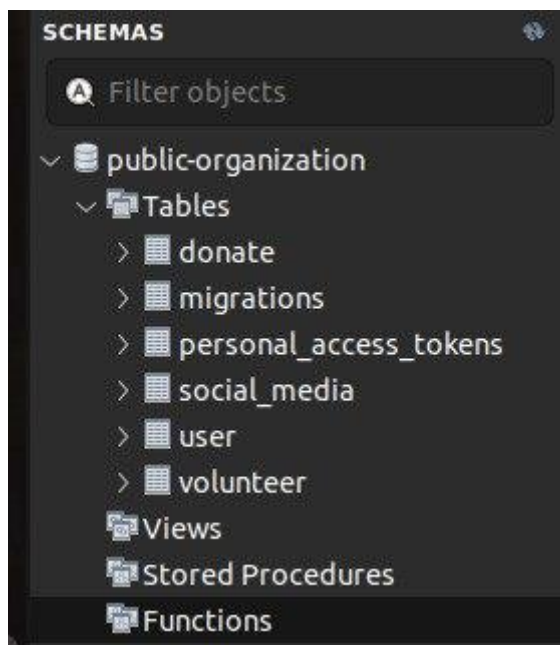


Рисунок 2.2 – Архітектура баз даних

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
↑ id	BIGINT	☒	☒	☐	☐	☒	☐	☒	☐	
• username	VARCHAR(255)	☐	☒	☒	☐	☐	☐	☐	☐	
• email	VARCHAR(255)	☐	☒	☒	☐	☐	☐	☐	☐	
• password	VARCHAR(255)	☐	☒	☐	☐	☐	☐	☐	☐	
• remember_token	VARCHAR(100)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
• created_at	TIMESTAMP	☐	☐	☐	☐	☐	☐	☐	☐	NULL
• updated_at	TIMESTAMP	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Рисунок 2.3 – Структура таблиці "user"

#	id	name	link	created_at	updated_at
1	1	Facebook	https://www.facebook.com/spravag...	2024-04-13 14:44:08	2024-04-13 14:44:08
2	2	Instagram	https://www.instagram.com/spgr_zp	2024-04-13 14:44:08	2024-04-13 14:44:08
3	3	Telegram	https://t.me/spgr_zp	2024-04-13 14:44:08	2024-04-13 14:44:08
4	4	YouTubeMusic	https://music.youtube.com/watch?v...	2024-04-13 14:44:08	2024-04-13 14:44:08
*	NULL	NULL	NULL	NULL	NULL

Рисунок 2.4 – Дані таблиці "social_media"

2.3 Створення API

Що таке API:

API (Application Programming Interface) - це інтерфейс програмування додатків, який дозволяє різним програмним компонентам взаємодіяти один з одним. API надає набір правил і протоколів, що визначають, як один програмний компонент може запитувати послуги або дані від іншого. У веб-розробці API часто використовуються для обміну даними між сервером і клієнтом.

Що таке CRUD

CRUD - це акронім, що означає основні операції, які можна виконувати з даними у базі даних:

- **Create (створення)** - операція створення нових записів у базі даних.
- **Read (читання)** - операція читання або отримання даних з бази даних.
- **Update (оновлення)** - операція оновлення існуючих записів у базі даних.
- **Delete (видалення)** - операція видалення записів з бази даних.

Ці операції є фундаментальними для роботи з будь-якою базою даних і часто реалізуються у веб-додатках через API.

Для створення API у проєкті було використано REST (Representational State Transfer) архітектуру. REST є стилем архітектури, який використовує стандартизовані HTTP методи для взаємодії з ресурсами. Основні методи HTTP, які використовуються у REST API, включають:

- **GET** - отримання ресурсів.
- **POST** - створення нових ресурсів.
- **PUT/PATCH** - оновлення існуючих ресурсів.
- **DELETE** - видалення ресурсів.

REST API має кілька важливих переваг, які роблять його популярним вибором для створення веб-сервісів. Простота REST API полягає у використанні простих HTTP запитів, що робить його легким для розуміння і використання. Масштабованість досягається завдяки архітектурі, яка дозволяє легко масштабувати і підтримувати додаток. REST API також відзначається гнучкістю, оскільки може працювати з різними форматами даних, такими як JSON, XML та інші. Крім того, можливість кешування відповідей підвищує продуктивність і зменшує навантаження на сервер.

Хоча GraphQL[9] є сучасною альтернативою REST для побудови API, REST має кілька ключових переваг. Простота і зрозумілість REST API виражаються в тому, що він базується на добре відомих HTTP методах і принципах, що робить його легшим для розуміння і використання. REST API легко інтегрується з існуючими механізмами кешування HTTP, що може значно підвищити продуктивність. Також, REST API підтримується всіма веб-браузерами і багатьма бібліотеками, що забезпечує його сумісність з більшістю існуючих систем. Стандартні HTTP методи і статуси REST API полегшують логування і моніторинг запитів і відповідей, що є важливим аспектом для підтримки і управління додатком.

Використання REST API у проєкті забезпечує простий і ефективний спосіб взаємодії з додатком. Це дозволяє легко виконувати операції CRUD з ресурсами, забезпечуючи при цьому високу продуктивність і масштабованість системи.

```

use Illuminate\Support\Facades\Route;

Route::group(['namespace' => 'App\Http\Controllers', 'middleware' => ['api'], 'prefix' => 'auth'], function ($router) {
    Route::post( url: 'login', action: 'AuthController@login');
    Route::post( url: 'register', action: 'AuthController@register');
});

Route::group(['namespace' => 'App\Http\Controllers', 'middleware' => ['api']], function ($router) {
    Route::get( url: '/get-donate-links', action: 'DonateController@getDonateLinks')->name( name: 'code.verify');
    Route::get( url: '/get-social-links', action: 'SocialMediaController@getSocialMediaLinks')->name( name: 'code.verify');
    Route::get( url: '/get-volunteers', action: 'VolunteerController@getVolunteer')->name( name: 'code.verify');
});

Route::group(['namespace' => 'App\Http\Controllers', 'middleware' => ['api', 'jwt']], function ($router) {
    Route::post( url: '/add-donate', action: 'DonateController@addDonateLink')->name( name: 'code.verify');
    Route::put( url: '/update-donate', action: 'DonateController@updateDonateLink')->name( name: 'code.verify');
    Route::delete( url: '/delete-donate', action: 'DonateController@deleteDonateLink')->name( name: 'code.verify');

    Route::post( url: '/add-social-link', action: 'SocialMediaController@addSocialMedia')->name( name: 'code.verify');
    Route::put( url: '/update-social-link', action: 'SocialMediaController@updateSocialMedia')->name( name: 'code.verify');
    Route::delete( url: '/delete-social-link', action: 'SocialMediaController@deleteSocialMedia')->name( name: 'code.verify');

    Route::post( url: '/add-volunteer', action: 'VolunteerController@addVolunteer')->name( name: 'code.verify');
    Route::put( url: '/update-volunteer', action: 'VolunteerController@updateVolunteer')->name( name: 'code.verify');
    Route::delete( url: '/delete-volunteer', action: 'VolunteerController@deleteVolunteer')->name( name: 'code.verify');
});

```

Рисунок 2.5 – Список ендпоінтів

РОЗДІЛ 3 Тестування та підтримка

3.1 Тестування

Unit-тести[11] - це фундаментальна частина розробки програмного забезпечення, яка полягає у випробуванні найменших можливих частин коду - "одиниць" - для підтвердження їх коректної роботи. У PHP unit-тести часто використовують фреймворки, такі як PHPUnit, для автоматизації процесу тестування. Використання unit-тестів допомагає забезпечити стабільність і надійність коду, спрощуючи виявлення і виправлення помилок на ранніх етапах розробки.

Unit-тести охоплюють різні аспекти функціональності коду, включаючи перевірку логіки, коректності обробки даних, валідацію введення та багато іншого. Важливо розуміти, що unit-тести зосереджені на тестуванні окремих функцій або методів у відриві від інших частин системи. Це дозволяє ізолювати та виявляти помилки конкретно в тій частині коду, яка тестується.

Тестування авторизації є важливою складовою розробки веб-додатків, оскільки авторизація визначає, хто має доступ до різних частин системи. Unit-тести допомагають переконатися, що механізми авторизації працюють належним чином і захищають ваш додаток від несанкціонованого доступу.

Авторизація включає кілька аспектів, таких як аутентифікація користувача (перевірка його особи) та розподіл прав доступу (визначення, що конкретний користувач може робити в системі). Важливо переконатися, що ці механізми працюють відповідно до специфікацій та забезпечують належний рівень безпеки.

У Laravel, наприклад, можна створити unit-тести, щоб перевірити, чи користувач з певними ролями має доступ до певних ресурсів. Ці тести можуть включати перевірку HTTP-статусів для різних запитів, щоб впевнитися, що доступ надається або відмовляється згідно з налаштуваннями авторизації.

Тестування авторизації також включає перевірку коректної роботи middleware, яке може бути відповідальним за перевірку прав доступу перед

виконанням певних дій. Це гарантує, що користувачі з належними правами можуть виконувати необхідні операції, а інші отримують відповідні повідомлення про помилку або перенаправляються на сторінки логіна чи помилки.

Тестування важливо для забезпечення якості програмного забезпечення та виявлення помилок до їх релізу. Unit-тести допомагають виявити і виправити помилки в коді на ранніх етапах розробки, що робить його більш надійним і менш вразливим до непередбачених ситуацій. Також тести дають можливість розробникам бути впевненими, що зміни в коді не порушують наявний функціонал, та сприяють рефакторингу коду без страху впровадження нових багів.

Однією з головних переваг unit-тестів є їх здатність швидко надавати зворотний зв'язок розробникам. Коли новий код додається до проєкту, тести можуть бути виконані автоматично, що дозволяє негайно виявити будь-які порушення функціональності. Це значно знижує ризик інтеграції нового коду та сприяє більш стабільному і надійному розвитку програмного забезпечення.

Unit-тести також сприяють підтримці високої якості коду. Вони заохочують розробників до написання більш чистого, модульного та легкого для тестування коду. Код, який складно тестувати, часто є ознакою поганого дизайну, і процес написання тестів може допомогти виявити і виправити ці проблеми на ранніх етапах.

Крім того, тести є цінним інструментом для документації коду. Вони надають конкретні приклади використання функцій і можуть допомогти новим розробникам швидше зрозуміти логіку і структуру проєкту. Це особливо корисно у великих командах або проєктах з довгою історією, де зрозуміти всі деталі і залежності може бути складно.

Таким чином, unit-тести не тільки підвищують якість коду, але й сприяють кращій організації роботи команди розробників, забезпечуючи стабільність і прогнозованість розробки програмного забезпечення.

```
use PHPUnit\Framework\TestCase;

class AuthenticationTest extends TestCase {
    public function testAuthenticateUser() {
        $result = authenticateUser('user@example.com', 'password123');
        $this->assertTrue($result);
    }
}
```

Рисунок 3.1 – Приклад тестування методу авторизації

3.2 Контроль версій

Використання Git на проєкті

На проєкті ми використовуємо Git як основний інструмент для контролю версій. Git є одним із найбільш популярних і потужних систем контролю версій, що широко використовується в індустрії розробки програмного забезпечення. Завдяки своїм багатим можливостям, Git дозволяє ефективно керувати змінами в коді, співпрацювати з командою та забезпечувати високу якість розробки.

Чому Git краще, ніж Bitbucket або GitLab

Хоча Bitbucket і GitLab є чудовими інструментами, вони фактично побудовані навколо Git. Основна різниця полягає в тому, що Bitbucket і GitLab є хостинг-сервісами для Git-репозиторіїв, тоді як Git - це сама система контролю версій. Обидва ці сервіси надають додаткові можливості, такі як інтеграція з системами CI/CD, управління проєктами, візуалізація змін тощо, але саме Git залишається основою, на якій вони базуються.

Переваги використання Git:

Розподілена система контролю версій: На відміну від централізованих систем контролю версій, Git є розподіленою системою. Це означає, що кожен розробник має повну копію репозиторію, включаючи всю історію змін. Це

забезпечує більшу гнучкість та надійність, оскільки дозволяє працювати автономно і зменшує залежність від центрального сервера.

Швидкість і продуктивність: Git оптимізований для швидкої роботи. Більшість операцій, таких як коміти, злиття гілок (мержі) та порівняння версій, виконуються локально і надзвичайно швидко. Це підвищує продуктивність розробників і дозволяє їм зосередитися на написанні коду, а не на чеканні завершення операцій з репозиторієм.

Гнучка система гілок: Git пропонує одну з найбільш гнучких і потужних систем управління гілками. Це дозволяє легко створювати, змінювати та об'єднувати гілки, що сприяє ефективному впровадженню нових функцій та виправленню помилок без порушення основної версії коду.

Інтеграція з різними сервісами: Git підтримується багатьма платформами для хостингу репозиторіїв, такими як GitHub[32], GitLab, Bitbucket та інші. Це забезпечує легку інтеграцію з іншими інструментами розробки, такими як системи безперервної інтеграції (CI/CD), системи управління завданнями, інструменти для аналізу коду тощо.

Для чого потрібен Git:

Git виконує кілька важливих функцій у процесі розробки програмного забезпечення:

Відстеження змін у коді: Git дозволяє відстежувати всі зміни, внесені до коду, зберігаючи повну історію комітів. Це допомагає розробникам розуміти, хто, коли і чому вніс ті чи інші зміни.

Співпраця в команді: Завдяки розподіленій природі Git, розробники можуть працювати незалежно один від одного, об'єднуючи свої зміни через коміти та пул-реквести. Це сприяє ефективній командній роботі та запобігає конфліктам при внесенні змін.

Управління версіями: Git дозволяє легко створювати і перемикатися між різними версіями коду, що є надзвичайно корисним під час роботи над новими функціями або виправленням помилок. Це дозволяє мати стабільну версію

продукту, яка готова до випуску, в той час як розробка нових функцій продовжується в окремих гілках.

Відкат змін: Завдяки можливості зберігати історію змін, Git дозволяє легко відкотити зміни до попередньої версії, якщо нові зміни виявилися проблемними. Це знижує ризики при впровадженні нових функцій і покращує стабільність продукту.

Користь Git при розробці, особливо в команді

Покращення командної роботи

Git значно спрощує співпрацю в команді. Кожен розробник може працювати над своєю гілкою, не турбуючись про конфлікти з іншими членами команди. Коли функція або виправлення готові, зміни можна об'єднати з основною гілкою через пул-реквест. Це дозволяє проводити код-рев'ю та обговорення змін перед їх інтеграцією в основну гілку.

Забезпечення стабільності коду

Завдяки можливості створювати гілки для різних етапів розробки (наприклад, розробка, тестування, продакшн), Git забезпечує стабільність коду. Основна гілка може залишатися стабільною і готовою до випуску, в той час як нові функції розробляються та тестуються в окремих гілках. Це дозволяє уникати збоїв у роботі продакшн-версії продукту.

Спрощення інтеграції змін

Git робить процес інтеграції змін простішим і безпечнішим. Інструменти для злиття гілок допомагають автоматично вирішувати конфлікти, а візуалізація історії змін дозволяє легко зрозуміти, як і коли зміни були внесені. Це знижує ризик помилок під час інтеграції і дозволяє швидко реагувати на будь-які проблеми.

Поліпшення якості коду

Процес код-рев'ю, який полегшується за допомогою пул-реквестів у Git, сприяє поліпшенню якості коду. Інші члени команди можуть переглядати зміни, надавати зворотний зв'язок і виявляти потенційні проблеми до того, як зміни

будуть інтегровані в основну гілку. Це забезпечує вищий рівень коду та знижує кількість багів.

Легкість в управлінні релізами

Git дозволяє ефективно керувати релізами за допомогою тегів та гілок. Кожен реліз може бути позначений тегом, що полегшує відстеження історії релізів та забезпечує легкий доступ до конкретних версій коду. Це особливо корисно для великих проєктів, де необхідно підтримувати кілька версій продукту одночасно.

Додаткові переваги використання Git

Безпека

Git забезпечує високу безпеку збереження даних. Оскільки кожен розробник має повну копію репозиторію, ризик втрати даних мінімізується. Навіть якщо сервер з репозиторієм виходить з ладу, дані можна відновити з локальних копій розробників.

Підтримка різних робочих процесів

Git підтримує різні робочі процеси, що дозволяє адаптувати його під потреби конкретного проєкту. Це можуть бути централізовані робочі процеси, такі як Git-flow, або більш гнучкі підходи, такі як GitHub-flow. Це забезпечує максимальну гнучкість і дозволяє команді вибрати той робочий процес, який найкраще відповідає їх вимогам.

Велика спільнота та підтримка

Git має велику та активну спільноту користувачів і розробників. Це означає, що ви завжди можете знайти відповіді на свої питання, отримати підтримку або знайти готові рішення для типових проблем. Крім того, існує багато навчальних матеріалів, курсів і документації, що полегшує освоєння Git навіть для новачків.

Інтеграція з іншими інструментами

Git легко інтегрується з різними інструментами для розробки програмного забезпечення, такими як системи безперервної інтеграції (CI/CD), трекери завдань, системи моніторингу та багато іншого. Це дозволяє створювати

ефективні та автоматизовані процеси розробки, що підвищує продуктивність команди та якість продукту.

Використання Git як системи контролю версій на проєкті надає численні переваги, особливо в контексті командної роботи. Завдяки своїй розподіленій природі, високій швидкості, гнучкій системі гілок та широким можливостям інтеграції, Git забезпечує стабільність, надійність та ефективність процесу розробки. Використання Git сприяє покращенню якості коду, полегшує співпрацю в команді та дозволяє ефективно керувати змінами та релізами продукту. Ці фактори роблять Git незамінним інструментом для сучасної розробки програмного забезпечення.

3.3 Підтримка та обслуговування

Підтримка та обслуговування програмного забезпечення є важливою частиною життєвого циклу будь-якого проєкту, забезпечуючи його стабільну роботу, безпеку та довговічність. Для нашого проєкту, який включає в себе реєстрацію користувачів, донати, інформацію про волонтерську діяльність, новини та звіти, підтримка та обслуговування мають особливе значення. Ось кілька ключових аспектів, які необхідно врахувати для успішної підтримки та обслуговування:

Моніторинг і логування

Моніторинг системи є критичним для забезпечення її безперебійної роботи. Постійне спостереження за станом серверів, бази даних та інших компонентів дозволяє швидко виявляти та усувати проблеми, що можуть виникнути. Впровадження систем логування дозволяє зберігати детальну інформацію про роботу додатку, що є важливим для діагностики та вирішення проблем. Наприклад, система логування може зберігати інформацію про помилки, запити до серверу, транзакції бази даних тощо. Це дозволяє

розробникам швидко знаходити причини збоїв та впроваджувати необхідні виправлення.

Регулярні оновлення

Для забезпечення безпеки та продуктивності програмного забезпечення важливо регулярно оновлювати фреймворк Laravel, бібліотеки PHP та інші залежності. Регулярні оновлення дозволяють отримувати останні виправлення безпеки, нові функції та поліпшення продуктивності, що робить систему більш надійною та ефективною. Планування оновлень слід здійснювати з урахуванням потенційних ризиків, пов'язаних з впровадженням нових версій, тому важливо тестувати оновлення на тестовому середовищі перед впровадженням у продуктивну систему.

Резервне копіювання

Резервне копіювання є необхідною складовою підтримки системи. Регулярне створення резервних копій бази даних та інших важливих даних дозволяє забезпечити їх збереження у випадку збою системи, втрати даних або інших непередбачуваних ситуацій. Важливо встановити регулярний графік резервного копіювання та перевіряти цілісність резервних копій для забезпечення можливості швидкого відновлення системи у разі необхідності.

Документація

Якісна та детальна документація є важливою частиною процесу підтримки та обслуговування. Документація повинна включати інструкції з розгортання, налаштування, використання та обслуговування системи. Вона допомагає новим розробникам швидко ознайомитися з проєктом, а також зменшує ймовірність помилок під час обслуговування. Документація повинна постійно оновлюватися та доповнюватися відповідно до змін у системі.

Автоматизоване тестування

Окрім юніт-тестів, важливо впровадити автоматизоване тестування інтеграції та функціональності. Це дозволяє перевіряти взаємодію між різними компонентами системи та забезпечувати її цілісність після внесення змін. Автоматизоване тестування допомагає виявляти потенційні проблеми на ранніх

стадіях розробки, що зменшує час та зусилля, необхідні для виправлення помилок.

Підтримка користувачів

Важливою складовою підтримки є забезпечення користувачів доступом до допомоги та підтримки. Це може включати створення системи підтримки користувачів, яка включає документацію для кінцевих користувачів, часті питання (FAQ) та можливість зв'язку з командою підтримки через електронну пошту або спеціальний інтерфейс на сайті. Своєчасна та ефективна підтримка користувачів сприяє їх задоволенню та зменшує кількість технічних проблем.

Контроль версій та розгортання

Використання системи контролю версій Git дозволяє легко управляти версіями коду, створювати гілки для нових функцій та виправлення помилок. Це забезпечує надійність та стабільність процесу розгортання нових версій програмного забезпечення. Автоматизовані інструменти розгортання можуть значно полегшити процес оновлення системи, зменшити ризик помилок та прискорити впровадження нових функцій.

Безпека

Забезпечення безпеки є критичним аспектом підтримки та обслуговування веб-додатків. Це включає регулярне оновлення системи безпеки, впровадження надійних механізмів автентифікації та авторизації, захист від атак, таких як SQL-ін'єкції, CSRF та XSS. Постійний моніторинг безпеки та проведення аудиту безпеки допомагають виявляти та усувати вразливості.

Ці заходи підтримки та обслуговування забезпечують надійність, безпеку та стабільність веб-додатка, що дозволяє забезпечити високий рівень задоволеності користувачів та тривалу експлуатацію системи.

ВИСНОВКИ

Протягом роботи над даним проєктом було виконано значний обсяг робіт, що охоплює різні аспекти сучасної веб-розробки. В результаті не лише створено функціональний і зручний веб-додаток, але й поглиблено знання та навички у кількох ключових напрямках.

Розробка з використанням Laravel

Вибір фреймворку Laravel виявився виправданим рішенням. У процесі розробки вдалося ефективно використовувати його потужний набір інструментів та зручний синтаксис для створення сучасних веб-додатків. Laravel забезпечив модульність і можливість легкої інтеграції додаткових компонентів, таких як Eloquent ORM, що значно спростило роботу з базою даних. Використання міграцій у Laravel дозволило легко керувати схемою бази даних, створювати та змінювати таблиці, індекси та зв'язки між ними, що було особливо важливим у контексті командної розробки.

Архітектура бази даних

Для проєкту було обрано MySQL як основну систему керування базою даних, що дозволило забезпечити високу продуктивність і надійність роботи з даними. Використання Eloquent ORM у Laravel значно спростило взаємодію з базою даних, забезпечуючи простий і інтуїтивно зрозумілий синтаксис для операцій CRUD. Створено складну структуру бази даних з таблицями, зв'язками та індексами, що дозволило забезпечити ефективне зберігання і доступ до даних.

Створення REST API

Одним з ключових аспектів проєкту було створення REST API для взаємодії між клієнтом і сервером. Глибше зрозумілося принципи REST архітектури та реалізовано CRUD-операції за допомогою стандартних HTTP методів. Простота, масштабованість, гнучкість і можливість кешування REST API стали важливими перевагами, які дозволили забезпечити ефективну роботу

додатку. Також розглянуто відмінності між REST і GraphQL, що допомогло краще зрозуміти переваги кожного підходу.

Тестування

Особливу увагу приділено тестуванню, зокрема unit-тестам. Використання PHPUnit дозволило автоматизувати процес тестування і забезпечити високу якість коду. Створено тести для перевірки авторизації, а також для всіх CRUD-операцій, що дозволило переконатися в правильності роботи кожної частини додатку. Тестування стало важливим етапом у виявленні та виправленні помилок на ранніх стадіях розробки, що значно підвищило надійність і стабільність системи.

Об'єктно-орієнтоване програмування

У процесі розробки активно застосовувалися принципи об'єктно-орієнтованого програмування (ООП). Використання класового програмування дозволило створювати модульний і легко масштабований код. Впроваджено основні концепції ООП, такі як інкапсуляція, наслідування та поліморфізм, що сприяло більш структурованій і зрозумілій архітектурі додатку. Використання патернів проєктування, зокрема тих, що рекомендовані Laravel, дозволило підвищити ефективність розробки та підтримки коду.

Взаємодія з Git та JWT

Здобуто значний досвід у використанні Git для контролю версій та управління командною роботою. Git дозволив відслідковувати зміни у коді, створювати різні гілки для нових функцій та ефективно вирішувати конфлікти. Використання JWT (JSON Web Token) для авторизації забезпечило безпечний спосіб автентифікації користувачів і захисту даних.

Результати і перспективи

В результаті роботи над проєктом створено функціональний веб-додаток, який дозволяє користувачам реєструватися та логінитися, робити донати, ділитися інформацією про волонтерську діяльність, читати новини та переглядати звіти. Основна аудиторія сайту - це користувачі, які хочуть підтримувати волонтерські проєкти та слідкувати за їхньою діяльністю.

Завдяки цьому проєкту було покращено технічні навички, отримано цінний досвід командної роботи, управління проєктами та вирішення складних технічних задач. Набуті знання і досвід будуть корисними у майбутніх проєктах і дозволять реалізовувати ще більш складні та амбіційні ідеї.

Таким чином, можна підсумувати, що даний проєкт став важливим кроком у професійному розвитку, відкрив нові горизонти у веб-розробці та підготував до майбутніх викликів у світі інформаційних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. RESTful API Design [Електронний ресурс]. — Режим доступу:
<https://restfulapi.net>
2. Laravel Official Documentation [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs>
3. MySQL Official Documentation [Електронний ресурс]. — Режим доступу:
<https://dev.mysql.com/doc>
4. JSON Web Tokens (JWT) Introduction [Електронний ресурс]. — Режим доступу: <https://jwt.io/introduction>
5. Об'єктно-орієнтоване програмування [Електронний ресурс]. — Режим доступу: https://wikipedia.org/wiki/Object-oriented_programming
6. Інкапсуляція в ООП [Електронний ресурс]. — Режим доступу:
[https://wikipedia.org/wiki/Encapsulation_\(computer_programming\)](https://wikipedia.org/wiki/Encapsulation_(computer_programming))
7. Наслідування в ООП [Електронний ресурс]. — Режим доступу:
[https://wikipedia.org/wiki/Inheritance_\(object-oriented_programming\)](https://wikipedia.org/wiki/Inheritance_(object-oriented_programming))
8. Поліморфізм в ООП [Електронний ресурс]. — Режим доступу:
[https://wikipedia.org/wiki/Polymorphism_\(computer_science\)](https://wikipedia.org/wiki/Polymorphism_(computer_science))
9. MVC Архітектура [Електронний ресурс]. — Режим доступу:
<https://wikipedia.org/wiki/Model-view-controller>
10. Eloquent ORM Documentation [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/eloquent>
11. PHP Official Website [Електронний ресурс]. — Режим доступу:
<https://www.php.net>
12. PHP: The Right Way [Електронний ресурс]. — Режим доступу:
<https://phptherightway.com>
13. CRUD Operations Explained [Електронний ресурс]. — Режим доступу:
<https://restapitutorial.com/lessons/crud.html>

14. Introduction to REST [Електронний ресурс]. — Режим доступу:
<https://restfulapi.net>
15. GraphQL vs REST [Електронний ресурс]. — Режим доступу:
<https://graphql.org/learn>
16. HTTP Caching [Електронний ресурс]. — Режим доступу:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>
17. Unit Testing in PHP with PHPUnit [Електронний ресурс]. — Режим доступу:
<https://phpunit.de>
18. PHPUnit Documentation [Електронний ресурс]. — Режим доступу:
<https://phpunit.readthedocs.io>
19. Патерни проектування [Електронний ресурс]. — Режим доступу:
<https://amazon.com>
20. Laravel Design Patterns [Електронний ресурс]. — Режим доступу:
<https://laravel-news.com>
21. Git Official Documentation [Електронний ресурс]. — Режим доступу:
<https://git-scm.com/doc>
22. JWT Security Best Practices [Електронний ресурс]. — Режим доступу:
<https://auth0.com/blog>
23. HTTP Status Codes [Електронний ресурс]. — Режим доступу:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
24. REST API Best Practices [Електронний ресурс]. — Режим доступу:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/RESTful_web_API/Best_practices
25. Введення до API [Електронний ресурс]. — Режим доступу:
https://wikipedia.org/wiki/Application_programming_interface
26. MVC Архітектура [Електронний ресурс]. — Режим доступу:
<https://wikipedia.org/wiki/Model-view-controller>
27. Laravel MVC Framework [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/structure>

- 28.CRUD Operations in Laravel [Електронний ресурс]. — Режим доступу:
<https://laravel-news.com/your-first-laravel-crud-application>
- 29.Laravel Migration System [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/migrations>
- 30.Laravel Routing [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/routing>
- 31.PHP Standard Recommendations (PSR) [Електронний ресурс]. — Режим доступу: <https://php-fig.org/psr>
- 32.Composer - PHP Dependency Manager [Електронний ресурс]. — Режим доступу: <https://getcomposer.org>
- 33.Документація про GitHub [Електронний ресурс]. — Режим доступу:
<https://docs.github.com>
- 34.Документація про Docker [Електронний ресурс]. — Режим доступу:
<https://docs.docker.com>
- 35.Безпека веб-додатків [Електронний ресурс]. — Режим доступу:
<https://owasp.org>
- 36.PHPUnit та тестування у Laravel [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/testing>
- 37.Тестування REST API [Електронний ресурс]. — Режим доступу: <https://rest-assured.io>
- 38.Використання JWT у Laravel [Електронний ресурс]. — Режим доступу:
<https://laravel-news.com/jwt-authentication>
- 39.Кешування у Laravel [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/cache>
- 40.Laravel Middleware [Електронний ресурс]. — Режим доступу:
<https://laravel.com/docs/middleware>
- 41.Документація PHP-FIG [Електронний ресурс]. — Режим доступу: <https://php-fig.org>
- 42.Git Tutorials and Guides [Електронний ресурс]. — Режим доступу: <https://git-scm.com/doc>

- 43.Основи GraphQL [Електронний ресурс]. — Режим доступу:
<https://graphql.org/learn>
- 44.GraphQL vs REST API [Електронний ресурс]. — Режим доступу:
<https://howtographql.com/basics/2-core-concepts>
- 45.RESTful Web Services [Електронний ресурс]. — Режим доступу:
https://wikipedia.org/wiki/Representational_state_transfer
- 46.Різниця між REST та GraphQL [Електронний ресурс]. — Режим доступу:
<https://medium.com/@localytics/graphql-vs-rest>
- 47.Git Flow [Електронний ресурс]. — Режим доступу:
<https://datasift.github.io/gitflow/IntroducingGitFlow.html>
- 48.Docker для розробників [Електронний ресурс]. — Режим доступу:
<https://docker.com/why-docker>
- 49.Основи Docker [Електронний ресурс]. — Режим доступу: <https://docker-curriculum.com>
- 50.Введення до Docker [Електронний ресурс]. — Режим доступу:
[\(https://wikipedia.org/wiki/Docker_\(software\)\)](https://wikipedia.org/wiki/Docker_(software))(<https://wikipedia.org>

ДОДАТОК А

Фрагменти коду

Приклад коду Laravel моделі

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Tymon\JWTAuth\Contracts\JWTSubject;

class User extends Authenticatable implements JWTSubject
{
    use HasApiTokens, HasFactory, Notifiable;

    protected $table = 'user';

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'username',
        'email',
        'password',
    ];
}
```

```

/**
 * The attributes that should be hidden for serialization.
 *
 * @var array<int, string>
 */
protected $hidden = [
    'password',
    'remember_token',
];

/**
 * The attributes that should be cast.
 *
 * @var array<string, string>
 */
protected $casts = [
    'email_verified_at' => 'datetime',
];

public function getJWTIdentifier()
{
    return $this->getKey();
}

public function getJWTCustomClaims()
{
    return [];
}
}

```

Приклад коду Laravel Controller

```
<?php

namespace App\Http\Controllers;

use App\Http\Services\DonateService;
use Illuminate\Database\Eloquent\Collection;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

class DonateController extends Controller
{
    public function __construct(
        private DonateService $donateService
    ) {
    }

    /**
     * @return Collection
     */
    public function getDonateLinks(): Collection
    {
        return $this->donateService->getDonateLinks();
    }

    /**
     * @param Request $request
     * @return JsonResponse|mixed
     */
}
```

```

public function addDonateLink(Request $request): mixed
{
    $validator = Validator::make($request->all(), [
        'label' => 'required|string',
        'link' => 'required|string',
    ]);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 422);
    }

    return $this->donateService->addDonateLinks($request);
}

/**
 * @param Request $request
 * @return JsonResponse
 */
public function updateDonateLink(Request $request): JsonResponse
{
    $validator = Validator::make($request->all(), [
        'donateId' => 'required|integer',
        'label' => 'string',
        'link' => 'string',
    ]);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 422);
    }
}

```

```

        return $this->donateService->updateDonateLinks($request);
    }

    /**
     * @param Request $request
     * @return JsonResponse|\Exception|string
     */
    public function deleteDonateLink(Request $request):
    JsonResponse|\Exception|string
    {
        $validator = Validator::make($request->all(), [
            'donateId' => 'required|integer'
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        return $this->donateService->deleteDonateLinks($request-
>donateId);
    }
}

```