

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБСИСТЕМИ ДЛЯ НАДАННЯ
ГУМАНІТАРНОЇ ДОПОМОГИ
DEVELOPMENT OF A WEB SYSTEM FOR
PROVIDING HUMANITARIAN AID

Виконав(ла): студент(ка) 2 курсу, групи КНТ-222м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ЖИГУЛІН Д.І.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЖИГУЛІНА Дмитра Ігоровича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебсистеми для надання гуманітарної допомоги.
Development of a Web System for Providing Humanitarian Aid.

керівник проєкту (роботи) к.т.н., доцент, ГОЛУБ Тетяна Василівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «14» листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз технічного завдання.
2. Планування проєкту та структуру вебсистеми.
3. Проєктування бази даних та структури вебсистеми.
4. Програмна реалізація вебсистеми.
5. Дослідження властивостей вебсистеми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Діаграма прецедентів для актора «Системний адміністратор»
Структура вебдодатку для «Адміністратора»
Алгоритм роботи робітника на видачі
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ГОЛУБ Т.В., доцент		
Нормоконтролер	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “ 08 ” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз технічного завдання.	2-3 тижні	Розділ 1
3	Планування проєкту та структуру вебсистеми.	4-5 тижні	Розділ 2
4	Проектування бази даних та структури вебсистеми.	6 тиждень	Розділ 3
5	Програмна реалізація вебсистеми.	7-8 тижні	Розділ 4
6	Дослідження властивостей вебсистеми	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Дмитро ЖИГУЛІН
(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ГОЛУБ
(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
121 с., 14 табл., 39 рис., 6 дод., 16 джерел.

ВЕБСИСТЕМА, ВПО, ГУМАНІТАРНА ДОПОМОГА, ТЕСТУВАННЯ,
СЖО, ТЕЛЕГРАМ-БОТ, CYPRESS, DJANGO, MYSQL, REACT.JS.

Об'єкт дослідження – гуманітарна допомога для ВПО та СЖО в Україні.

Предмет дослідження – вебсистема для надання гуманітарної допомоги.

Мета роботи – розробка вебсистеми, яка організовує та прискорює роботу обласної громади для видачі гуманітарної допомоги та надає людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мови програмування Python та Javascript, месенджер Telegram, мова моделювання UML, інтегроване середовище розробки програм Visual Studio Code, backend фреймворк Django та frontend бібліотека React.JS, система управління базами даних MySQL, утиліти для створення автоматизованих тестів Cypress та створення навантажувальних тестів Locust, інструмент управління проєктами Jira.

Результати. Створено вебсистему, яка містить вебдодаток на базі React.JS та Django в інтеграції з телеграм-ботом.

Висновки. Розроблено вебсистему, яка спрямована на оптимізацію та прискорення діяльності обласної громади у сфері гуманітарної допомоги, створює ефективний механізм для координації та видачі індивідуалізованої підтримки, забезпечуючи можливість громадян отримувати гуманітарну допомогу яка враховує їхні конкретні потреби.

Галузь використання – обласна громада або гуманітарний штаб.

Економічна ефективність. Оптимізація ресурсів, зменшення бюрократії та часу, а також підвищення прозорості та контролю над гуманітарною допомогою.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 121 pages, 14 tables, 39 figures, 6 appendices, 16 sources.

WEB SYSTEM, IDP, HUMANITARIAN AID, TESTING, DLC, TELEGRAM BOT, CYPRESS, DJANGO, REACT.JS.

Object of study is humanitarian aid for IDPs and DLCs in Ukraine.

Subject of the study is a developed web system for providing humanitarian aid.

The purpose of the work is to develop a web system that organizes and accelerates the work of the regional community for the provision of humanitarian aid and provides people with the opportunity to receive humanitarian aid that will take into account their individual needs.

Materials, methods and tools: object-oriented programming, programming languages Python and Javascript, Telegram messenger, modeling language UML, integrated program development environment Visual Studio Code, backend framework Django and frontend library React.JS, database management system MySQL, utilities for creating automated tests Cypress and creating load tests Locust, project management tool Jira.

Results. A web system is created which contains a web application based on React.JS and Django in integration with the Telegram bot.

Conclusions. The developed web system, aimed at optimizing and accelerating the activities of the regional community in the field of humanitarian aid, creates an effective mechanism for coordinating and issuing individualized support, providing the opportunity for citizens to receive humanitarian aid that takes into account their specific needs.

Field of use – regional community or humanitarian headquarters.

Economic efficiency. Optimization of resources, reduction of bureaucracy and time, and improvement of transparency and control over humanitarian aid.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	9
Вступ.....	10
1 Аналіз технічного завдання.....	11
1.1 Дослідження та аналіз цільової аудиторії.....	11
1.2 Дослідження існуючих аналогів	16
1.2.1 Платформа «Є-Допомога».....	16
1.2.2 Платформа «Поряд».....	17
1.2.3 Висновки	18
1.3 Дослідження та формулювання вимог до вебсистеми	20
1.3.1 Аналіз технологій та програмних засобів для розробки системи.....	22
1.3.2 Дослідження пристроїв та браузерів цільової аудиторії.....	28
2 Планування проєкту та моделювання вебсистеми	31
2.1 Менеджмент проєкту в Jira.....	31
2.1.1 Створення проєкту	32
2.1.2 Ведення проєкту.....	32
2.2 UML моделювання вебсистеми	34
2.2.1 Діаграма прецедентів (ролей)	35
2.2.2 Діаграма послідовності.....	38
3 Проєктування бази даних та структури вебсистеми	44
3.1 Проєктування структури та відносин бази даних	44
3.2 Проєктування структури та алгоритму роботи вебсистеми	51
4 Програмна реалізація вебсистеми	54
4.1 Розробка архітектури клієнтської частини	54

4.2 Розробка архітектури серверної частини	56
4.3 Архітектура телеграм-бота.....	59
4.4 Програмна реалізація системи	60
4.4.1 Авторизація.....	60
4.4.2 Локації	61
4.4.3 Обробка запиту на складі	62
4.4.4 Видача допомоги	63
4.4.5 Реалізація серверної частини	64
4.4.6 Реалізація телеграм-бота	65
5 Дослідження властивостей вебсистеми	68
5.1 Створення тест-плану	68
5.1.1 Scope of testing	68
5.1.2 Out of scope	68
5.1.3 Типи тестування	69
5.1.4 Техніки тест-дизайну	70
5.1.5 Критерії входу	71
5.1.6 Критерії призупинення	71
5.1.7 Критерії виходу	72
5.1.8 Системний ресурс	73
5.1.9 Визначення пріоритету помилок	73
5.2 Ручне тестування	74
5.3 Автоматизоване тестування	77
5.4 Дослідження стійкості вебдодатку	78
5.4.1 Опис тестового сценарію	79
5.4.2 Параметри тестування	79

5.4.3 Запуск та результати моніторингу	79
5.4.4 Аналіз результатів та висновки	80
Висновки	81
Перелік джерел посилань	83
Додаток А Технічне завдання	85
Додаток Б Діаграма прецедентів для актора «Системний адміністратор»	91
Додаток В Структура вебдодатку для «Адміністратора»	93
Додаток Г Алгоритм роботи робітника на видачі	95
Додаток Д Текст програми	97
Додаток Е Слайди презентації	113

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– програмний інтерфейс додатку;
CRUD	– модель створення/читання/редагування/видалення;
MTV	– модель-шаблон-представлення;
SRS	– специфікація вимог програмного забезпечення;
UI	– інтерфейс користувача;
UX	– досвід користувача;
БД	– база даних;
ВПО	– внутрішньо переміщена особа;
ПЗ	– програмне забезпечення;
ОС	– операційна система;
СЖО	– складні життєві обставини;
СУБД	– система управління базами даних.

ВСТУП

На фоні світових конфліктів і військових подій, гуманітарна допомога стає важливою складовою глобальної відповіді на гострі потреби населення, що опинилося у зоні конфлікту. Ця проблема набуває особливої актуальності в умовах надзвичайної складності, коли основні засоби інфраструктури та соціально-економічний лад країни руйнуються під впливом військових дій. У роботі з гуманітарною допомогою в обласних центрах відіграє ключову роль організація та координація допомоги для людей, які постраждали від конфліктів, природних катастроф або інших надзвичайних ситуацій. Робота з гуманітарною допомогою в обласних громадах вимагає дбайливого планування, співпраці та взаємодії для забезпечення ефективності та максимального впливу на полегшення страждань населення. Створення вебсистеми є одним з найбільш ефективних способів покращення взаємодії та комунікації між робітниками обласних громад та постраждалих людей (ВПО та СЖО). Вебсистема здатна організувати роботи обласної громади та полегшити процес видачі гуманітарної допомоги людям.

Створення вебсистеми розпочинається із розробки технічного завдання, аналізу цільової аудиторії та створення детально продуманого проєкту. Важливо враховувати вимоги та завдання вебсистеми при розробці структури, інтерфейсу, дизайну, контенту, функціоналу, стилю та особливостей навігації.

Технічне завдання представляє собою документ, що повністю описує мету, завдання та обмеження проєкту, функціональні та нефункціональні вимоги, екрани майбутньої системи та інтеграцію з іншими сервісами. Такий підхід гарантує високу якість та повноту розробленого документа. Приймання готової вебсистеми здійснюється на основі затвердженого та узгодженого технічного завдання.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Дослідження та аналіз цільової аудиторії

Гуманітарна допомога на час війни набуває важливого значення для забезпечення життєвих потреб людей, які опинилися у складних ситуаціях. Вона охоплює широкий спектр заходів, спрямованих на забезпечення безпеки, харчування, медичної допомоги та освітніх можливостей для постраждалих від воєнних подій. Здійснення гуманітарної допомоги вимагає спільних зусиль між міжнародними організаціями, громадськими об'єднаннями та країнами, які об'єднують свої ресурси для полегшення страждань та відновлення зруйнованого суспільства.

У цьому контексті важливо визнати, що гуманітарна допомога не тільки надає негайну допомогу для виживання людей, але й сприяє відновленню соціально-економічної стабільності та покращенню життєвих умов на довгострокову перспективу. Наша спільна здатність мобілізувати ресурси та підтримувати тих, хто опинився у складній ситуації, свідчить про людську солідарність та готовність допомагати тим, хто найбільше цього потребує у важкі часи війни. Гуманітарна допомога під час війни вимагає високого рівня координації між різними гуманітарними організаціями та урядовими структурами, а також великої фінансової підтримки для забезпечення невідкладного та ефективного реагування на гострі гуманітарні потреби.

Обласна громада грає важливу роль у видачі гуманітарної допомоги, особливо в умовах кризових ситуацій, конфліктів або природних катастроф.

Обласна громада виступає важливим центром координації допомоги. Вона взаємодіє з різними гуманітарними організаціями, урядовими структурами, та іншими стейкхолдерами для організації та спрямування допомоги туди, де вона найбільше потрібна. Також громада має глибоке розуміння потреб місцевого населення. Здійснення аналізу потреб допомагає визначити основні проблеми та визначити, які види гуманітарної допомоги є належними та необхідними.

Обласна громада відповідає за ефективний розподіл отриманих гуманітарних ресурсів. Це включає в себе логістичне планування, складське управління та транспортування, щоб забезпечити швидку та точну доставку допомоги до місць потреби.

Важливим елементом діяльності обласної громади є підтримка звітності та транспарентності в розподілі та використанні гуманітарної допомоги. Це сприяє підвищенню довіри громади та донорів.

Донори гуманітарної допомоги є важливим ланцюгом у системі підтримки та надання допомоги людям, які опинилися у складних ситуаціях, таких як конфлікти, природні катастрофи чи інші надзвичайні події. Донори можуть бути різними - від державних урядів і міжнародних організацій до приватних осіб та комерційних компаній.

Багато країн регулярно виділяють кошти на гуманітарну допомогу через свої державні бюджети або спеціальні фонди. Також, міжнародні організації, такі як ООН, Міжнародний Червоний Хрест, Червоний Півмісяць, виконують ключову роль у координації та наданні гуманітарної допомоги на світовому рівні. Фізичні особи, благодійні фонди, та приватні компанії можуть виступати як значущі донори гуманітарної допомоги. Їх внесок може бути в грошовій формі, наданні ресурсів чи волонтерською діяльністю.

Роль донорів полягає не лише в наданні фінансової підтримки, але і в активному участі в розвитку стратегій гуманітарної допомоги, впливі на політику та сприянні створенню сталої системи допомоги для тих, хто потребує підтримки. Також донори можуть фінансувати проєкти, спрямовані на забезпечення тривалої стабільності та підвищення життєвого рівня.

До складу робітників які відповідальні за організацію та видачу гуманітарної допомоги входять адміністратор, робітник складу та робітник на видачі. Тобто організаційна модель громади з видачі гуманітарної допомоги складається (рис. 1.1) з адміністратора який керує та організовує роботу робітників складу та видачі.



Рисунок 1.1 – Організаційна модель громади з видачі гуманітарної допомоги

Функції адміністратора полягають в загальному керівництві робітниками з гуманітарної допомоги та нагляд за виконанням завдань цих робітників.

Адміністратор вирішує наступні задачі:

- розробка та виконання логістичних планів для доставки гуманітарної допомоги до потребуючих регіонів;
- організація транспортування та розподілу ресурсів для максимізації ефективності;
- керування бюджетом проєкту та відстеження витрат;
- ведення фінансової звітності та взаємодія з фінансовими партнерами;
- управління робочими процесами та забезпечення ефективної роботи команди;
- забезпечення дотримання адміністративних процедур та нормативів;
- підготовка звітів для фінансових партнерів та інших зацікавлених сторін.

Робітник складу вирішує наступні задачі:

- приймання, реєстрація та розміщення гуманітарних вантажів на складі;
- систематичний облік кількості та видів отриманих ресурсів;

- складання гуманітарних вантажів у партії для подальшого відправлення;
- упаковка матеріалів з дотриманням стандартів безпеки та зручності транспортування;
- ефективне розміщення товарів на складі для легкого доступу та зручності інвентаризації;
- мітки та системи означення для швидкого знаходження товарів;
- проведення періодичних інвентаризацій для підтримки актуальності обліку товарів на складі.

Робітник на видачі вирішує наступні задачі:

- видача гуманітарних вантажів, їх реєстрація та ведення актуальних списків видачі;
- перевірка документів або інших форм ідентифікації отримувачів для підтвердження їх права на отримання допомоги;
- встановлення системи чергування та розподілу гуманітарних товарів серед отримувачів;
- забезпечення справедливого та рівного доступу до ресурсів;
- надання інформації отримувачам щодо виду та кількості отриманої допомоги, процедур та умов видачі;
- точне ведення документів про видачу та складання звітів для подальшої звітності;
- організація та координація роботи волонтерів, які можуть допомагати у видачі допомоги.

Вимушено переміщені особи (ВПО) — це термін, який використовується для опису людей, які були вимушені залишити свої домівки через конфлікти, природні катастрофи, насильство, порушення прав людини або інші форми небезпеки. Вимушені переміщені особи часто стикаються з великими труднощами, такими як втрата житла, недостатній доступ до основних послуг, загроза безпеки та відсутність стабільності. Гуманітарні організації та уряди зазвичай намагаються надавати допомогу та підтримку цій вразливій групі людей.

Люди у складних життєвих обставинах (СЖО) — вказує на ситуації, коли люди опиняються в складних життєвих обставинах. Це може включати економічні труднощі, проблеми здоров'я, сімейні кризи, втрату роботи, соціальне виключення чи інші виклики, які можуть суттєво ускладнити їхнє життя.

В процесі аналізу визначено профіль користувачів (отримувачів) цільової аудиторії (табл. 1.1). Саме для таких людей (ВПО та СЖО) обласні громади надають допомогу у вигляді гуманітарної допомоги.

Таблиця 1.1 Профіль користувача (отримувача) цільової аудиторії

Тип	Опис
Вік людини	Від 18 до 99 років (Особа) / Сім'я
Стать	Чоловіча/Жіноча
Географія	Україна
Статус	ВПО/СЖО

Протягом повномасштабного вторгнення в Україну кількість внутрішньо переміщених осіб (ВПО) зросла до 7 млн осіб. Зараз майже половина внутрішньо переміщених осіб працездатного віку (44%) не отримують доходу. Лише третина респондентів вказала заробітну плату як основне джерело доходу після переміщення, в той час як 24% покладаються на державну підтримку. Внаслідок цього більшість домогосподарств, які були переміщені, вживають заходи для скорочення витрат (70%), включаючи витрати на продукти, і третина змушена брати позики.

З наближенням осінніх та зимових місяців умови проживання для внутрішньо переміщених осіб викликають зростаючу тривогу. Згідно із звітом Міжнародної організації з міграції (МОМ), 22% опитаних вказали, що їхнє житло не відповідає зимовим умовам в Україні. Понад третина не може собі дозволити переїхати в житло, яке відповідає стандартам, в основному через фінансові

обмеження. 30% тих, хто знаходить притулок у селах (загалом близько 2 млн осіб), не мають доступу до медичних послуг.

1.2 Дослідження існуючих аналогів

Для того щоб розробити вебсистему яку задовольнятиме потреби користувачів без використання іншого програмного забезпечення проведено порівняльний аналіз двох аналогічних платформ: «Є-Допомога» та «Поряд».

1.2.1 Платформа «Є-Допомога»

Проект «Є-Допомога» (рис. 1.2) — це соціальна платформа, на якій українці, які постраждали від війни, можуть отримати допомогу. Це може бути фінансова підтримка від держави чи міжнародної організації, допомога ліками, продуктами, засобами гігієни тощо від волонтерів. Головна мета проєкту — об'єднати можливості країни, міжнародних неурядових організацій та волонтерів для допомоги постраждалим і внутрішньо переміщеним особам під час війни [1].

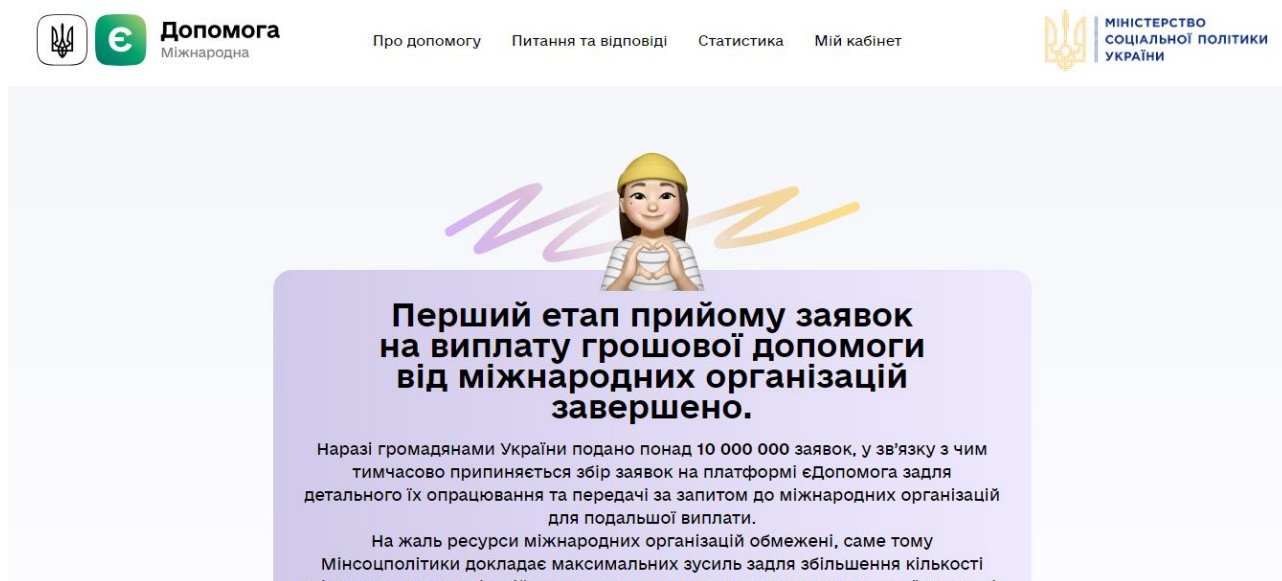


Рисунок 1.2 – Головна сторінка платформи «Є-Допомога» [1]

Доступні три варіанти допомоги:

- грошова допомога від держави;
- виплати від міжнародних організацій;
- отримання та надання волонтерської допомоги.

Якщо людині потрібна допомога, він подає заявку на платформі і показує, що саме йому потрібно. Платформа відображає всі потреби людей для волонтерів у вигляді списку. Якщо волонтер може забезпечити людину необхідними потребами, то волонтер надає подальші інструкції для надання допомоги.

Перевагою цієї платформи:

- грошова допомога;
- розміщення запитів.

Недоліком цієї системи є нерегулярність надання допомоги за попередніми запитами, тому що волонтерів не забезпечують усім необхідним.

1.2.2 Платформа «Поряд»

В Україні запрацював портал «Поряд» (рис. 1.3) для розміщення запитів на допомогу та їхнього швидкого пошуку тими, хто має можливість її надати. Сервіс дозволяє без зайвих зусиль подати запит на отримання допомоги. В свою чергу, люди, які мають змогу допомогти, зможуть швидко дізнатися про потреби та зв'язатись з тими, хто залишив запити [2].

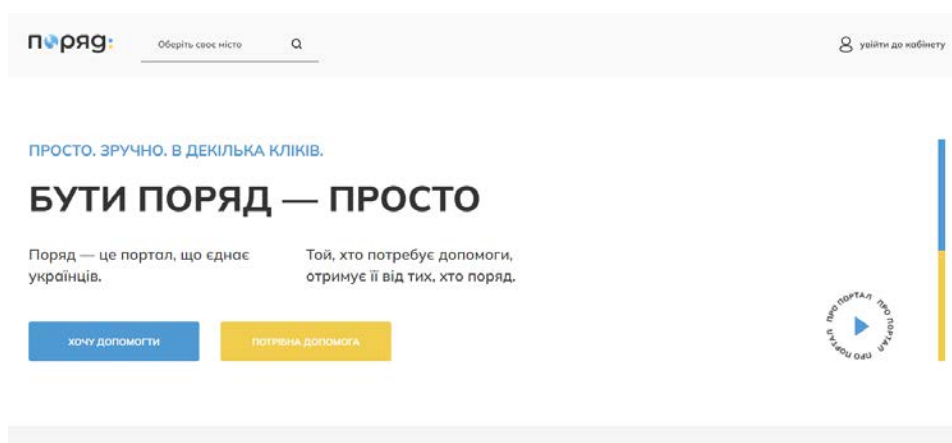


Рисунок 1.3 – Головна сторінка платформи «Поряд» [2]

Тобто платформа має дві основні ролі: волонтер та людина яка потребує допомоги.

Сценарій для волонтера:

- авторизуватися на платформі;
- вибрати регіон для надання допомоги;
- вибрати конкретні запити для допомоги;
- відправити спеціальну форму на допомогу та надати допомогу людям.

Сценарій для людини яка потребує допомоги:

- авторизуватися на платформі;
- вибрати пункт «Потрібна допомога»;
- заповнити та відправити форму з переліком необхідної допомоги;
- після отримання допомоги вибрати пункт «Отримав допомогу».

Перевагою цієї платформи:

- розміщення запитів;
- донати від людей.

Недоліком цієї платформи є також нерегулярність надання допомоги.

1.2.3 Висновки

На основі опису двох аналогічних платформ створено порівняльну таблицю (табл. 1.2) яка відображає головний функціонал цих платформ. Також в таблиці (табл. 1.2) розглянуто можливий варіант реалізації вебсистеми.

Таблиця 1.2 – Порівняльна таблиця платформ аналогів

Характеристика	«Є-Допомога»	«Поряд»	Вебсистема
Авторизація та реєстрація	Вебсайт	Вебсайт	Вебдодаток та телеграм-бот
Надання допомоги	Так	Так	Так

Продовження таблиці 1.2

Характеристика	«Є-Допомога»	«Поряд»	Вебсистема
Допомога за розкладом	Ні, тільки за можливістю волонтерів	Ні, тільки за можливістю волонтерів	Регулярно за розкладом
Грошова допомога	Так	Ні	Ні
Чітко визначене місце отримання допомоги	Ні	Ні	Локація визначається заздалегідь, та є регулярним місцем
Створення нових отримувачів	Тільки самостійно через вебсайт	Тільки самостійно через вебсайт	На локації, Робітник на видачі створить акаунт отримувача
Спосіб взаємодії для отримувача	Вебсайт	Вебсайт	Телеграм-бот
Чітко визначені правила видачі	Ні, але грошова допомога регулюється законами України	Ні, тільки волонтерські можливості	ВПО отримують допомогу раз на тиждень, СЖО – раз на місяць
Запити за передзамовленням	Так	Так	Так

Обидві платформи мають функціонал розміщення запитів за попередніми запитами, які обробляються волонтерами та міжнародними організаціями, але ця допомога не є регулярної та не за існуючим розкладом надання. Також платформи мають лише один спосіб створення запиту – через вебдодаток за допомогою багатокрокових форм.

Виходячи з цього, зроблено висновок що ці дві платформи не підходять для вирішення мети дипломної роботи, а саме забезпечення людей регулярною допомогою.

1.3 Дослідження та формулювання вимог до вебсистеми

Формулювання вимог - це процес визначення потреб і очікувань щодо нового продукту, що включає в себе взаємодію з зацікавленими сторонами та кінцевими користувачами. Це може включати в себе ідентифікацію функціональних вимог, нефункціональних вимог та технічних обмежень. Формулювання вимог є ключовим етапом у процесі розробки програмного забезпечення, і воно визначає основний напрямок та параметри подальшої роботи. Основна мета полягає у визначенні очікувань, врегулюванні конфліктів і документуванні всіх ключових вимог.

Однією з основних труднощів для будь-якої організації є спільне розуміння кінцевого продукту зі стороною клієнтів. Таким чином, аналіз бізнес-вимог включає колективні зусилля всіх ключових зацікавлених сторін, розробників програмного забезпечення, кінцевих користувачів і менеджерів клієнтів, з метою досягнення єдиної точки зору щодо того, яким має бути продукт. Цей процес завжди виконується на ранніх етапах будь-якого проєкту, забезпечуючи, що кінцевий продукт відповідає всім вимогам.

Вимоги користувача до системи:

- забезпечення легкості навігації та користування системою для різних користувачів;
- можливість швидко та безпечно реєструвати нових отримувачів гуманітарної допомоги та ідентифікація вже зареєстрованих осіб;
- механізм автоматизованої обробки заявок на отримання гуманітарної допомоги;
- гарантія безпеки та конфіденційності особистих даних отримувачів та інших важливих інформаційних деталей;
- наявність різних варіантів вибору гуманітарних ресурсів, щоб користувачі мали можливість адаптувати допомогу під свої потреби;
- сумісність з мобільними пристроями для зручного доступу до системи з будь-якого місця.

Система має забезпечувати:

- авторизацію та реєстрацію користувачів;
- автоматизовану обробку заявок;
- безпеку та конфіденційність;
- створення профілів отримувачів;
- фільтрацію та пошук локацій та отримувачів;
- обробку запитів отримувачів;
- обробку гуманітарної допомоги робітником складу;
- обробку гуманітарної допомоги робітником на видачі;
- ведення графіку видачі для кожного локації та отримувачів.

Для отримувачів потрібно спроектувати та розробити Telegram Bot, для адміністратора, робітника складу та робітника на видачі - вебдодаток. Для обслуговування та підтримки вебдодатку та Telegram Bot системний адміністратор.

Вимоги до системного адміністратора:

- а) адміністрування вебдодатку та Telegram Bot:
 - 1) виправлення помилок;
 - 2) забезпечення функціонування сервісу логування помилок;
 - 3) обслуговування користувачів;
- б) управління БД:
 - 1) забезпечення неперервної функціональності БД;
 - 2) захист персональних даних користувачів;
 - 3) збереження цілісності даних;
 - 4) оптимізація роботи БД та резервне копіювання;
- в) управління сервером:
 - 1) збір статистики щодо обігу запитів до сервера;
 - 2) обслуговування сервера;
 - 3) забезпечення захисту.

Функції вебсистеми та характеристики користувачів більш детально зображено в додатку А.

1.3.1 Аналіз технологій та програмних засобів для розробки системи

1.3.1.1 Аналіз frontend та backend технологій

Виходячи з аналізу технічного завдання треба обрати frontend та backend бібліотеки/вебфреймворки та інструментальні засоби для реалізації вебдодатку. Статистика використання вебфреймворків для frontend розробки з авторитетного джерела “Imaginary Cloud” за 2022 рік (рис. 1.4) [3] вказує на те, що коефіцієнт використання React.JS дорівнює 82%, Angular - 49% та Vue.JS - 46%. Тобто React.JS, Angular та Vue.JS є найпопулярнішими бібліотеками/вебфреймворками для frontend розробки.

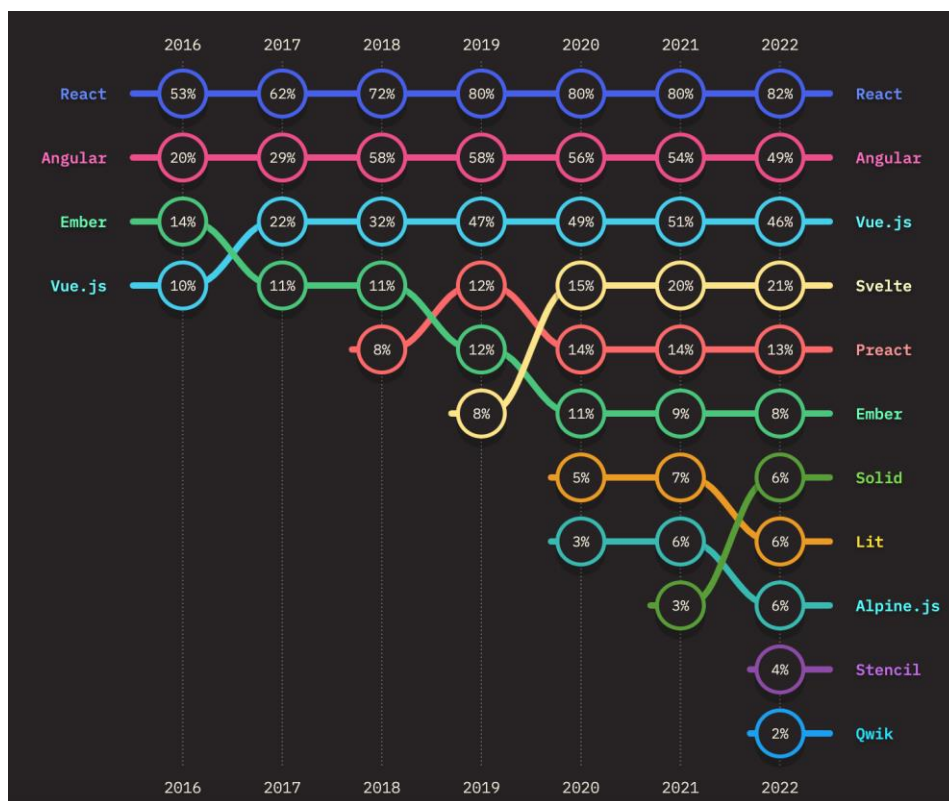


Рисунок 1.4 – Статистика використання бібліотек/вебфреймворків для frontend розробки за 2022 рік [3]

На основі даних про використання бібліотек/вебфреймворків для frontend розробки в 2022 році створено таблицю (табл. 1.3), яка порівнює характеристики цих бібліотек/вебфреймворків.

Таблиця 1.3 – Порівняльна таблиця засобів для frontend розробки

Характеристика	React.JS	Angular	Vue.JS
Тип	Бібліотека	Фреймворк	Бібліотека
Розробник	Facebook	Google	Evan You
Мова	JavaScript, JSX	TypeScript, JavaScript	JavaScript
Архітектура	Компонентна	Модульна, Компонентна	Компонентна
Віртуальний DOM	Так	Ні	Так
Інтеграція з іншими технологіями	Широкомасштабна	Інтегрована в екосистему Google	Широкомасштабна
Швидкість	Висока	Середня	Висока
Продуктивність	Висока	Висока	Висока

Переваги React.JS:

- висока продуктивність та швидкість;
- реюзабельні компоненти;
- широка спільнота та інструменти;
- відмінна документація.

Недоліком React.JS є висока складність для новачків та використання зовнішнього стеку інструментів.

Переваги Angular:

- повноцінний фреймворк;
- типізація та модульність.

Недоліки Angular:

- розмір бандлів;
- високий поріг входження.

Переваги Vue.JS:

- легкість вивчення;
- гнучкість та модульність;
- декларативний підхід.

Недоліки Vue.JS:

- менше готових рішень порівняно з React.JS та Angular;
- обмежена інтеграція з іншими технологіями;
- невелика кількість розроблених інструментів для тестування;
- менше корпоративного використання.

Виходячи з отриманої статистики та аналізу порівнянь бібліотек/вебфреймворків для frontend розробки обрано React.JS, через високу швидкість та продуктивність, коефіцієнт використання, широку спільноту та розгалужену екосистему.

Для реалізації backend частини вебдодатку обрано “Django” вебфреймворк через такі переваги:

- надає структуру та конвенції для розробки вебдодатків, що робить його зручним. У нього є вбудовані інструменти для обробки URL-адрес, роботи з базою даних, формами та іншими складовими, які можуть бути корисними при створенні вебдодатку в інтеграції з телеграм-ботом;
- в останніх версіях Django додана підтримка асинхронності, що може бути корисною при взаємодії з телеграм-ботами, оскільки вони використовують асинхронний код для обробки багатьох одночасних подій;
- взаємодія з зовнішніми API, включаючи той, який використовується для взаємодії з Telegram Bot API. Django дозволяє зручно робити HTTP-запити, обробляти відповіді та працювати з даними, отриманими з API.

1.3.1.2 Аналіз систем управління базами даних

Реляційні бази даних забезпечують структурований підхід до організації даних за допомогою таблиць, які містять рядки та стовпці. Це дозволяє легко визначати відносини між різними сутностями та забезпечує чітку структуру для зберігання та отримання інформації. Також реляційні бази даних підтримують високий рівень цілісності даних за допомогою забезпечення унікальності ключів та використання обмежень. Найголовніше що, реляційні БД можуть легко масштабуватися, дозволяючи розширювати ресурси для забезпечення обробки зростаючого обсягу даних та запитів. Виходячи з цього для зберігання даних обрано реляційну БД.

На основі цього висновку обрано дві СУБД: MySQL та PostgreSQL. Також створено порівняльну таблицю (табл. 1.4) яка відображає характеристики цих СУБД.

Таблиця 1.4 – Порівняльна таблиця реляційних СУБД

Характеристика	MySQL	PostgreSQL
Open Source	Відкритий вихідний код, належить Oracle	Відкрите джерело
Тип	Реляційна	Реляційна
Підтримка NoSQL/JSON	Підтримка деяких функцій	Тільки JSON
Декларативне секціонування	Так	Так
Логічна реплікація	Так	Так
Вкладені селекції	Так	Так

Продовження таблиці 1.4

Характеристика	MySQL	PostgreSQL
Швидкодія	Високий рівень	Середній рівень
Транзакції	Так	Так
Можливість зберігання лише у пам'яті	Так	Ні

Виходячи з таблиці обидві СУБД мають схожі характеристики, але MySQL є безпечнішою та більш швидкою в обробці запитів. А PostgreSQL підходить до підтримки складних та масивних БД

1.3.1.3 Висновки

Отже, базуючись на огляді статистичних даних щодо використання технологій та аналізі технічного завдання, виділяємо такі технології та інструменти:

- СУБД MYSQL;
- вебсервер Apache;
- мова програмування Javascript;
- мова програмування Python;
- бібліотека React.JS;
- вебфреймворк Django;
- месенджер Telegram.

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування (за стандартом ISO/IEC 2382:2015). В загальному випадку база даних містить схеми, таблиці, подання,

збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім самих даних, містить їх опис та може містити засоби для їх обробки [4].

Система управління базами даних (СУБД) – це програмне забезпечення, яке дозволяє керувати базами даних. СУБД відповідають за створення, зміну та вставку елементів у базу даних. Вони також дозволяють користувачам легко та швидко отримувати доступ до даних. Крім того, вони відповідають за забезпечення цілісності збережених даних [5].

MySQL - це система керування базами даних. У реляційній базі даних дані зберігаються не всі скопом, а в окремих таблицях, завдяки чому досягається вигравш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою відносин, завдяки чому забезпечується можливість об'єднувати при виконанні запиту дані з декількох таблиць. SQL як частина системи MySQL можна охарактеризувати як мову структурованих запитів плюс найбільш поширений стандартний мова, яка використовується для доступу до баз даних. MySQL - це ПЗ з відкритим кодом. Застосовувати його і модифікувати може будь-хто. Таке ПЗ можна одержувати по Internet і використовувати безкоштовно. При цьому кожен користувач може вивчити вихідний код і змінити його відповідно до своїх потреб [6].

Вебсервер Apache – це широко використовуване програмне забезпечення вебсервера з відкритим кодом, яке розробляється та підтримується Apache Software Foundation. Це кросплатформне програмне забезпечення, яке сумісне з сучасними операційними системами, такими як UNIX і Windows. Сервер Apache відомий своєю безпекою, ефективністю та розширюваністю, що робить його популярним вибором для розміщення вебсайтів і вебдодатків. Сервер Apache використовує модульну архітектуру, що означає, що він складається з основного набору модулів, який за потреби можна розширити додатковими модулями. Ці модулі можна використовувати для додавання таких функцій, як кешування, стиснення та автентифікація [7].

Javascript – мова програмування, що дозволяє реалізувати ряд складних рішень в розробці вебдодатків. Вона допомагає зробити сторінки сайту більш

інтерактивними, обробляє дії користувачів сайту. Це об'єктно-орієнтована клієнтська мова, яка підтримується додатками, що працюють з дизайном сайту [8].

Python – високорівнева мова програмування, яку називають другою за популярністю в світі. Її використовують для розробки вебдодатків, ПЗ та машинного навчання. Python застосовують для вирішення робочих завдань у компаніях Google, Instagram, Facebook, IBM, NASA, Dropbox, Netflix та інших. Розробники цінують цю мову програмування за простоту у вивченні, ефективність та мультиплатформність [9].

React.JS – це відкритий JavaScript-фреймворк, а точніше, бібліотекою JavaScript, яка використовується для розробки інтерфейсів користувача. Він був створений компанією Facebook і швидко набув популярності серед розробників з усього світу. React.JS дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у React.JS є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за рендеринг певної частини користувацького інтерфейсу [10].

Django - безкоштовний і відкритий фреймворк для створення вебдодатків, написаний мовою програмування Python. Вебфреймворк - набір компонентів, які допомагають розробляти вебсайти швидше і простіше [11].

Telegram Bot – це окремий обліковий запис у Telegram, який самостійно відповідає на повідомлення користувачів. Бізнесу цей інструмент дає можливість оптимізувати комунікацію з клієнтами: чат-бот завжди онлайн та відповідає співрозмовнику миттєво у будь-який час, а також не потребує заробітної плати [12].

1.3.2 Дослідження пристроїв та браузерів цільової аудиторії

Цільова аудиторія вебдодатку це працівники громади або гуманітарного штабу. Цільову аудиторію телеграм-бота (Отримувачів) не розглядаємо тому що вони користуються виключно Telegram. Вебдодаток розроблено для веб використання на комп'ютері та планшеті. Виходячи з цього проведено дослідження використання браузерів на комп'ютерах та планшетах в Україні. Міжнародний

авторитетний ресурс статистики “Statcounter” [13] надає інформацію щодо статистики використання браузерів на комп’ютерах в Україні (рис. 1.5) та статистики використання браузерів на планшетах в Україні (рис. 1.6) за 2023 рік.



Рисунок 1.5 – Статистика використання браузерів на комп’ютерах в Україні за 2023 рік [13]

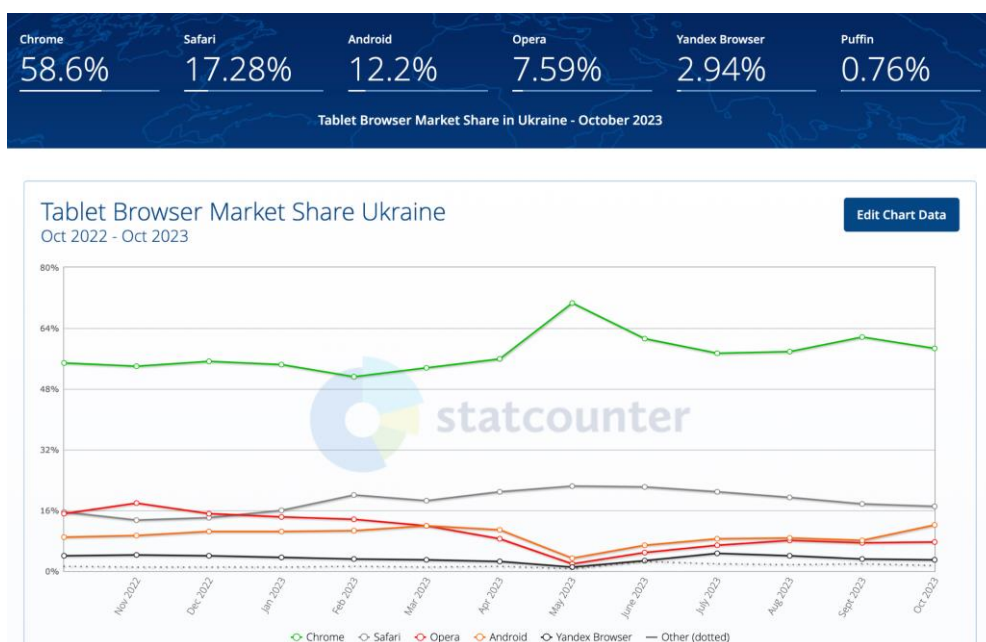


Рисунок 1.6 – Статистика використання браузерів на планшетах в Україні за 2023 рік [13]

Враховуючи статистичні дані з використання браузерів в Україні за останній рік, для тестування вебдодатку рекомендовано зосередитися на трьох основних браузерах. Перший у списку - Google Chrome (Desktop), займаючи більше двох третин ринку, визначається своєю швидкістю та інтеграцією з Google сервісами. Opera (Desktop), з 10.7% використання, видається привабливим варіантом за наявність вбудованого VPN та енергозберігаючого режиму. Також, враховуючи підтримку відкритого коду та фокус на приватності, важливо включити у тестування Firefox (Desktop) з 9.28%. У сегменті планшетів, важливо охопити Google Chrome (Tablet) та Safari (Tablet), які, займаючи велику частку ринку, представляють різні операційні системи та характеризуються різними особливостями користування. Такий підхід до вибору браузерів для тестування дозволить максимально відобразити реальні умови використання вебдодатку в Україні.

2 ПЛАНУВАННЯ ПРОЄКТУ ТА МОДЕЛЮВАННЯ ВЕБСИСТЕМИ

2.1 Менеджмент проєкту в Jira

Основні функції Jira включають у себе можливість створення та відстеження завдань, керування проєктами, спільну роботу команди через коментування та обговорення завдань. Додатково, система має модулі та компоненти, такі як задачі та спринти для роботи за методологією Scrum, дашборди та звіти для відстеження прогресу проєкту, а також керування версіями для контролю за версіями продукту.

Jira також підтримує різні інтеграції, такі як з системами керування версіями Bitbucket та GitHub, а також з системою документації Confluence для полегшення обміну інформацією.

Однією з переваг Jira є її гнучкість та адаптивність до різних методологій розробки. Також варто відзначити високий рівень інтеграції, який полегшує комунікацію та обмін даними в команді. Застосування Jira у дипломному проєкті може значно полегшити управління завданнями, організацію роботи та аналіз прогресу проєкту через дашборди та звіти.

Отже визначимо основні задачі для планування та ведення проєкту в Jira:

- визначення мети, обсягів та термінів;
- визначення основних завдань;
- встановлення основних етапів розробки: frontend, backend, тестування;
- визначення тривалості спринтів/ітерацій;
- деталізація завдань на підзавдання для кращого виконання та відстеження;
- розподіл завдань між командами та окремими учасниками проєкту;
- створення дашбордів для відстеження прогресу та стану завдань;
- використання коментарів для обговорення завдань та вирішення поточних питань;
- планування етапів тестування та збору відгуків від користувачів.

2.1.1 Створення проєкту

Мета проєкту – розробка вебсистеми, яка організовує та прискорює роботу обласної громади для видачі гуманітарної допомоги та надає людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби.

Обсяг проєкту – включає розробку телеграм-бота та вебдодатку з реєстрацію/авторизацією користувачів, менеджменту команди та відгуків, обробки, видачі та отримання гуманітарної допомоги.

Терміни виконання – 5 місяців на повний цикл розробки.

Frontend – розробка інтерфейсу користувача та інтеграція з backend частиною.

Backend – розробка серверної частини, БД, налаштування серверу, розробка телеграм-бота та інтеграція з frontend частиною.

Тестування (QA) – оцінка якості frontend та backend частин розробки, написання тест-плану та тест-кейсів, оформлення баг-репортів та проходження тест-кейсів у тестовому середовищі.

2.1.2 Ведення проєкту

Визначимо робочий процес на Jira дошці (рис. 2.1):

- колонка Backlog є місцем для зберігання всіх ідей, завдань та вимог, які ще не визначені для розробки. Перегляд та призначення пріоритетів для завдань, які готові до взяття в розробку;
- колонка To Do включає завдання, які готові до взяття в розробку. Завдання, які пройшли визначення пріоритетів і готові для взяття в розробку;
- колонка Rework включає завдання, які вимагають доробок чи внесення змін. Завдання, які були перевірені, але потребують додаткових змін чи виправлень;

- колонка In Progress відображає завдання, які в даний момент перебувають у стадії активної розробки. Завдання, які знаходяться у процесі активного виконання розробником;
- колонка Ready for QA містить завдання, які вважаються завершеними розробкою і очікують на проходження тестування якості. Завдання, які пройшли розробку та рецензію і готові для тестування;
- колонка Done включає завдання, які успішно завершено та пройшло всі етапи розробки та тестування. Завдання, які успішно виконані та готові для впровадження в продуктивне середовище;
- колонка Discarded містить завдання, які були скасовані або відкинуті з розгляду. Завдання, які були визнані непотрібними чи неактуальними та скасовані.

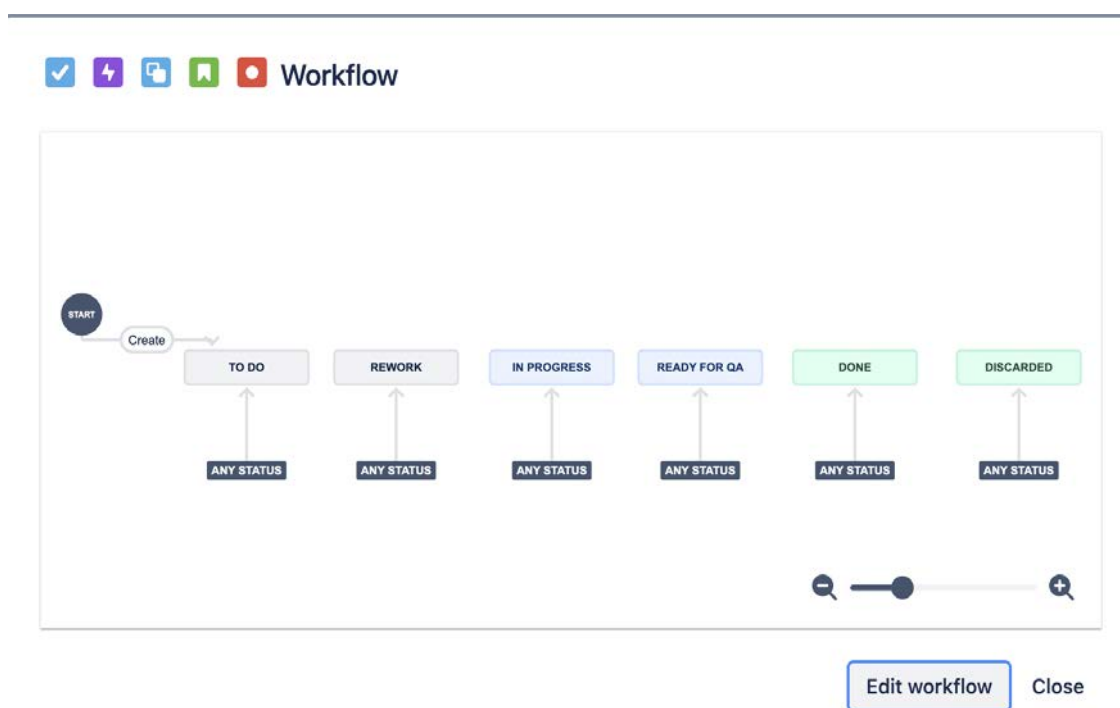


Рисунок 2.1 – Робочий процес згідно Jira

Після визначення робочого процесу можна створювати завдання на дошці згідно сформульованих вимог (рис. 2.2).

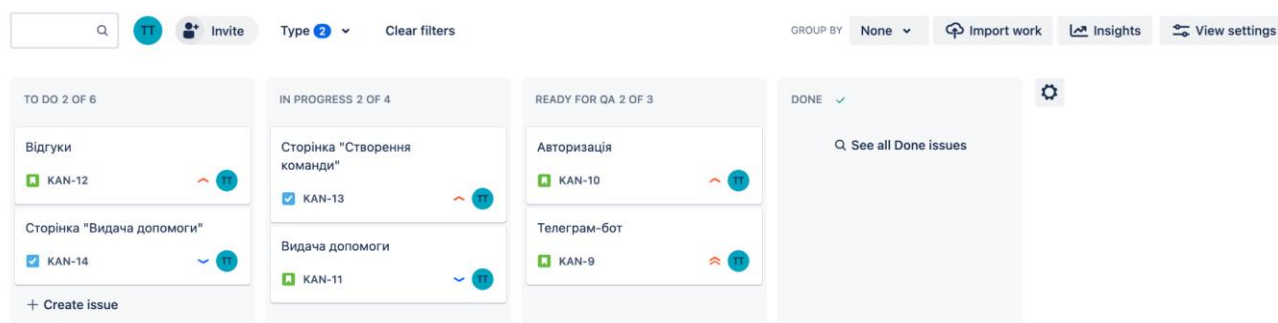


Рисунок 2.2 – Jira дошка

2.2 UML моделювання вебсистеми

Моделювання програмного забезпечення - це процес створення абстрактних моделей або представлень системи програмного забезпечення для аналізу, проєктування, тестування та управління її розробкою та експлуатацією. Цей підхід дозволяє розробникам і інженерам використовувати концептуальні та графічні представлення для вивчення властивостей програмного забезпечення та визначення його поведінки без прямого створення фактичного коду.

UML (Unified Modeling Language) – це мова позначень або побудови діаграм, призначена для визначення, візуалізації і документування моделей зорієнтованих на об'єкти систем програмного забезпечення. UML не є методом розробки, іншими словами, у конструкціях цієї мови не повідомляється про те, що робити першим, а що останнім, і не надається інструкцій щодо побудови вашої системи, але ця мова допомагає вам наочно переглядати компонування системи і полегшує співпрацю з іншими її розробниками [14].

Моделі дозволяють розробникам аналізувати потреби та вимоги користувачів, а також проєктувати структуру та функціональність програми перед початком розробки.

Моделювання дозволяє проводити тестування програмного забезпечення без його фактичного виконання, що може значно зекономити час та ресурси.

Моделі допомагають управляти різними аспектами розробки, визначаючи завдання, ресурси та терміни.

Моделі дозволяють розробникам, клієнтам та іншим зацікавленим сторонам краще розуміти функції та властивості програмного забезпечення, сприяючи ефективній комунікації.

Моделювання може бути виконане на різних етапах розробки програмного забезпечення, від концептуальної фази до тестування та експлуатації. Основні типи моделей включають структурні, поведінкові, архітектурні, та інші, які використовуються для вирішення конкретних завдань.

2.2.1 Діаграма прецедентів (ролей)

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених межею системи, асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система під час діалогу з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів із системою [15].

Актори представляють зовнішніх учасників чи сутності, які взаємодіють з системою. Це можуть бути користувачі, інші системи або навіть зовнішні процеси.

Прецеденти представляють конкретні функціональні можливості або дії, які може виконати система. Кожен прецедент може бути розглянутий як окремий сценарій взаємодії між актором і системою.

У деяких випадках можуть бути визначені умови або розширення, які деталізують додаткові аспекти взаємодії між акторами та прецедентами.

Виходячи характеристики користувачів із специфікації вимог описані такі актори:

- робітник на видачі (рис. 2.3);
- робітник складу (рис. 2.4);
- адміністратор (рис. 2.5);
- отримувач (рис. 2.6);
- системний адміністратор – зображено в додатку Б.

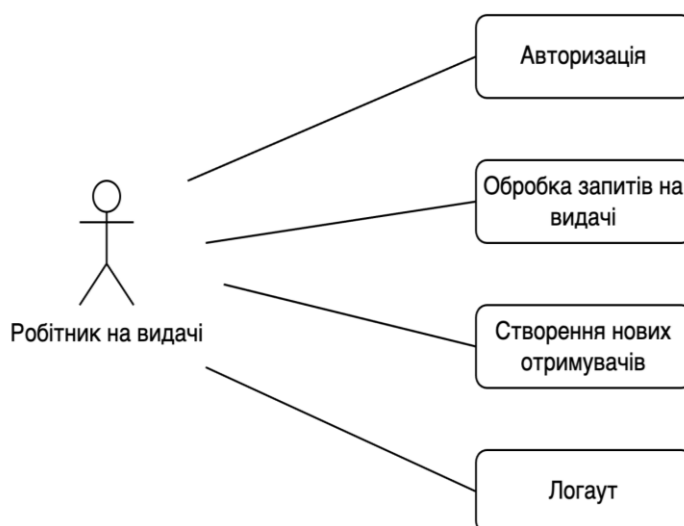


Рисунок 2.3 – Актор “Робітник на видачі” (діаграма прецедентів)

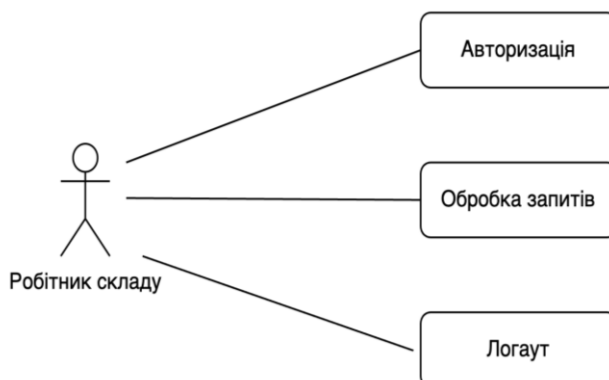


Рисунок 2.4 – Актор “Робітник складу” (діаграма прецедентів)

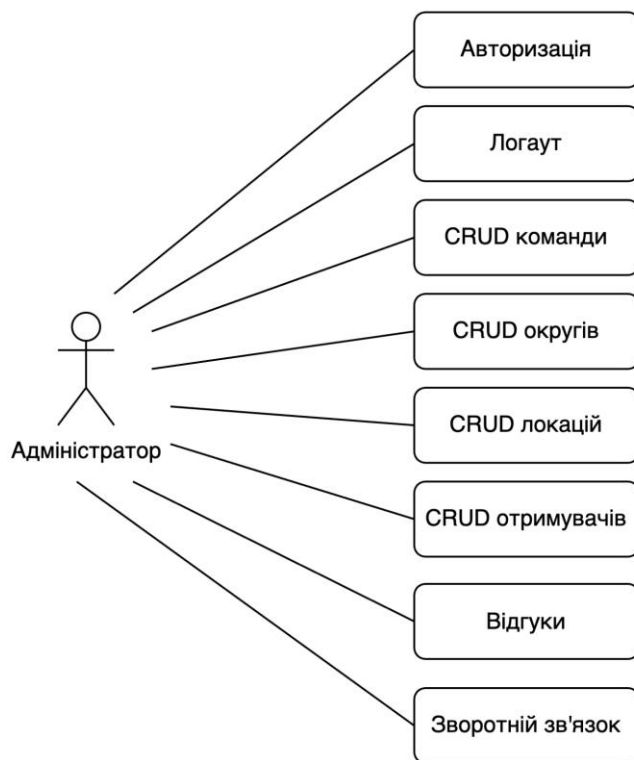


Рисунок 2.5 – Актор “Адміністратор” (діаграма прецедентів)

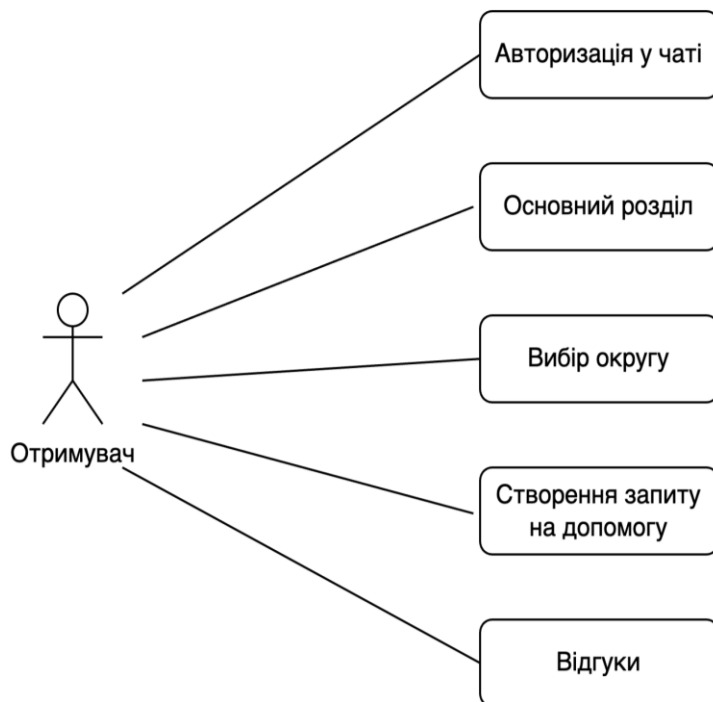


Рисунок 2.6 – Актор “Отримувач” (діаграма прецедентів)

Використання "extend" дозволяє визначити, що певний прецедент (розширюючий) може бути викликаний в рамках іншого прецеденту (базового), якщо виконуються певні умови.

2.2.2 Діаграма послідовності

Діаграма послідовностей відображає взаємодію об'єктів в динаміці. Головними елементами діаграм послідовності є об'єкти, які є логічними сутностями, що представляють окремі елементи системи та повідомлення, якими вони обмінюються. В якості об'єктів можуть виступати також актори. Повідомлення можуть бути не тільки абстрактними діями, що виконуються, але і методи класів, створених на діаграмі класів. Повідомлення на діаграмі послідовності пронумеровані, тобто мають чітку послідовність. Діаграма послідовностей відноситься до діаграм взаємодії UML, що описує поведінкові аспекти системи, але розглядає взаємодію об'єктів в часі. Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об'єктами [16].

Основні компоненти діаграми послідовності включають:

- об'єкти;
- лінії життєвого циклу;
- повідомлення;
- фрагменти;
- загальні повідомлення.

2.2.2.1 Діаграма послідовності для прецеденту «Обробка запитів на складі»

Діаграма послідовності для прецеденту «Обробка запитів на складі» відображає об'єкти, між якими відбувається взаємодія у прецеденті (рис. 2.8).

Об'єктами є: Робітник складу, Вебдодаток, База даних та Сервер. Описано процеси роботи всіх об'єктів під час обробки запитів.

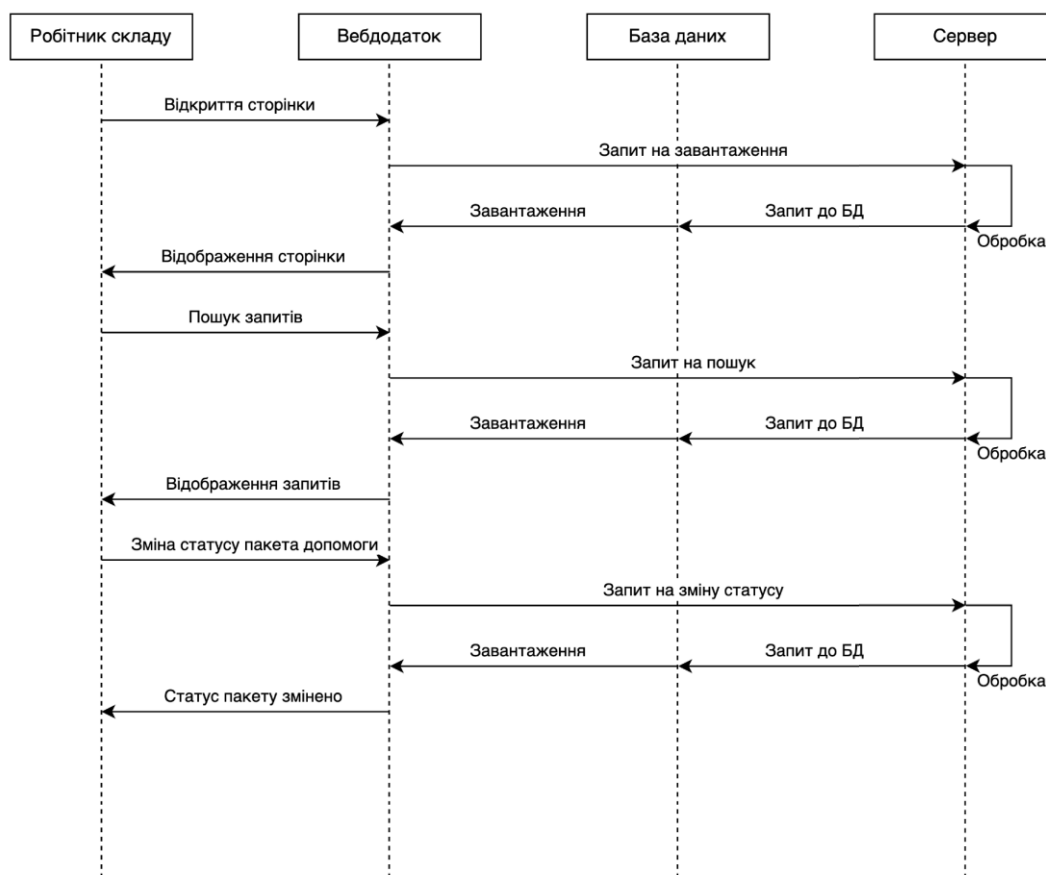


Рисунок 2.8 – Діаграма послідовності для прецеденту «Обробка запитів на складі»

Сценарій:

- робітник складу відкриває вебдодаток (авторизований заздалегідь);
- вебдодаток робить запит до серверу;
- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку «Обробка запитів на складі» у вебдодатку;
- вебдодаток відображає сторінку «Обробка запитів на складі»

Робітнику складу;

- робітник складу шукає запити;
- вебдодаток робить запит до серверу;

- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку з результатами пошуку;
- вебдодаток відображає сторінку з результатами пошуку Робітнику складу;
- робітник складу змінює статус пакету допомоги на «Упакований»;
- вебдодаток робить запит до серверу;
- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку «Обробка запитів на складі» у вебдодатку зі зміною статусу;
- вебдодаток відображає сторінку зі зміною статусу Робітнику складу.

2.2.2.2 Діаграма послідовності для прецеденту «Обробка запитів на видачі»

Діаграма послідовності для прецеденту «Обробка запитів на видачі» відображає об'єкти, між якими відбувається взаємодія у прецеденті (рис. 2.9). Об'єктами є: Робітник на видачі, Вебдодаток, База даних та Сервер. Описано процеси роботи всіх об'єктів під час обробки запитів.

Сценарій:

- робітник на видачі відкриває вебдодаток (авторизований заздалегідь);
- вебдодаток робить запит до серверу;
- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку «Видача допомоги» у вебдодатку;
- вебдодаток відображає сторінку «Видача допомоги» Робітнику на видачі;
- робітник на видачі вводить ПІБ отримувача;
- вебдодаток робить запит до серверу;

- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку з результатами пошуку;
- вебдодаток відображає сторінку з результатами пошуку Робітнику на видачі;
- робітник на видачі натискає кнопку «Видати допомогу»;
- вебдодаток робить запит до серверу;
- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку з повідомленням про успішну видачу;
- вебдодаток відображає сторінку з повідомленням про успішну видачу Робітнику на видачі.

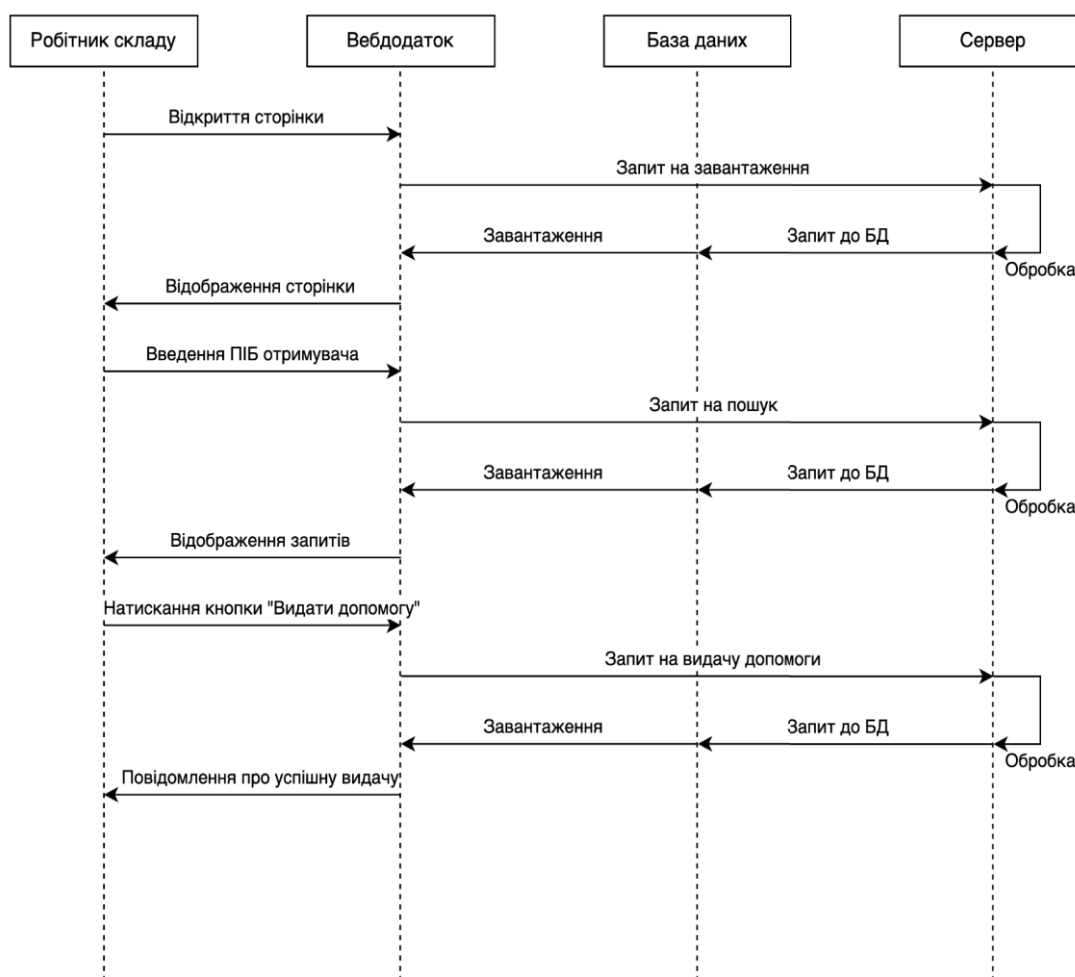


Рисунок 2.9 – Діаграма послідовності для прецеденту «Обробка запитів на видачі»

2.2.2.3 Діаграма послідовності для прецеденту «Створення команди»

Діаграма послідовності для прецеденту «Створення команди» відображає об'єкти, між якими відбувається взаємодія у прецеденті (рис. 2.10). Об'єктами є: Адміністратор, Вебдодаток, База даних, Сервер та Email-сервіс. Описано процеси роботи всіх об'єктів під час обробки запитів.

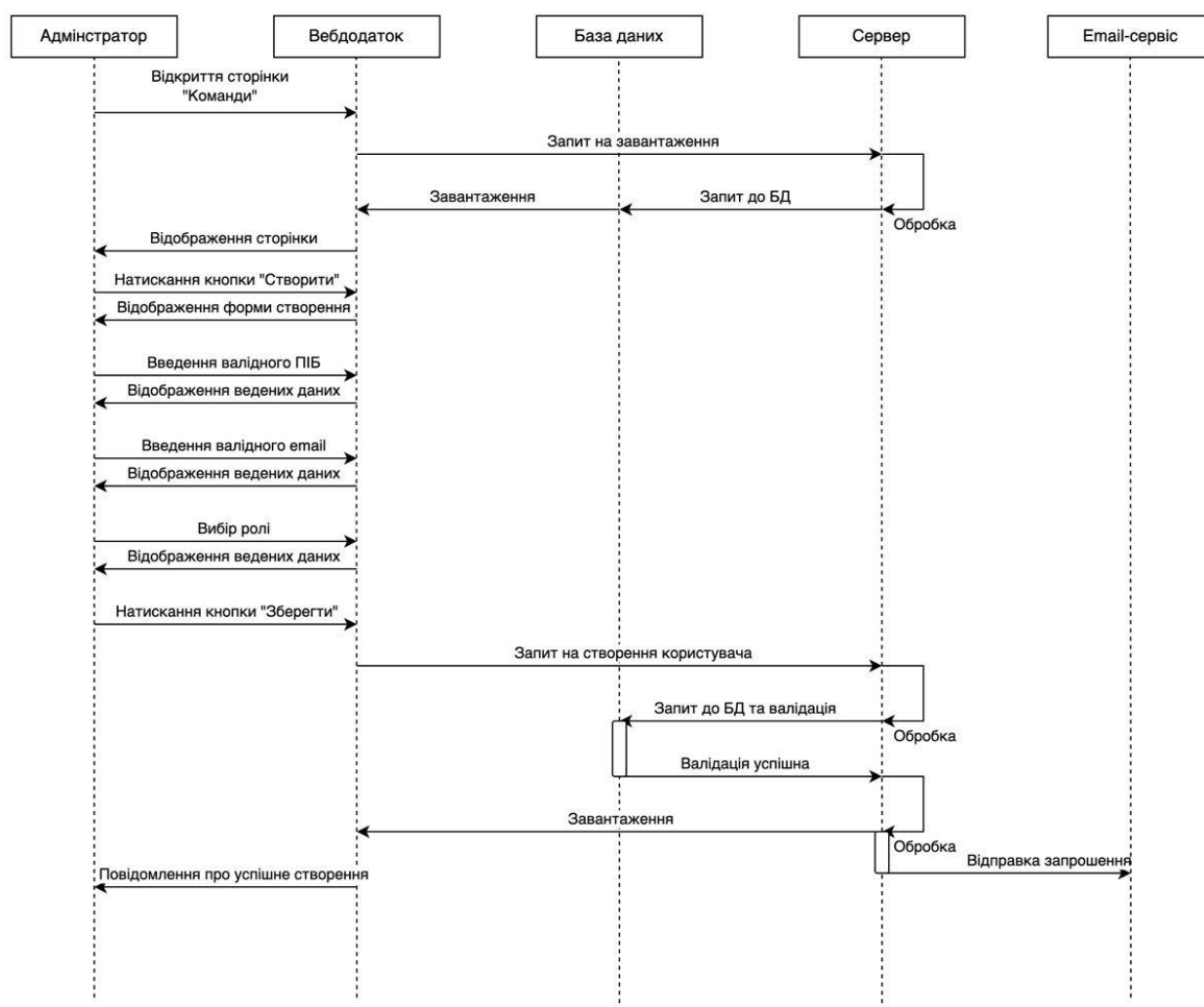


Рисунок 2.10 – Діаграма послідовності для прецеденту «Створення команди»

Сценарій:

- адміністратор відкриває сторінку «Команда» (авторизований заздалегідь);
- вебдодаток робить запит до серверу;

- сервер обробляє звернення;
- сервер робить запит до БД;
- сервер завантажує сторінку «Команда» у вебдодатку;
- вебдодаток відображає сторінку «Команда» Адміністратору на видачі;
- адміністратор вводить натискає кнопку «Створити»;
- вебдодаток відображає форму;
- адміністратор вводить валідні ПІБ користувача;
- вебдодаток відображає дані у формі;
- адміністратор вводить валідний email;
- вебдодаток відображає дані у формі;
- адміністратор вибирає роль користувача;
- вебдодаток відображає дані у формі;
- адміністратор натискає кнопку «Зберегти»;
- вебдодаток робить запит до серверу;
- сервер обробляє звернення;
- сервер робить запит до БД;
- база даних валідує створення нового об'єкта;
- база даних повідомляє серверу про успішну валідацію;
- сервер обробляє звернення;
- сервер завантажує сторінку з повідомленням про успішну видачу;
- сервер відправляє запрошення до Email-сервісу;
- сервер завантажує сторінку з результатами пошуку;
- вебдодаток відображає сторінку з повідомленням про успішне створення нового члена команди.

3 ПРОЄКТУВАННЯ БАЗИ ДАНИХ ТА СТРУКТУРИ ВЕБСИСТЕМИ

3.1 Проєктування структури та відносин бази даних

У контексті даного розділу структура бази даних визначається як ключовий елемент системи, що впливає на її продуктивність та зручність використання. Вибір оптимальної структури бази даних має критичне значення для забезпечення ефективного функціонування інформаційної системи в цілому. Метою є розробка структури бази даних, що найкращим чином відповідає потребам системи. Завдання включають визначення основних сутностей та атрибутів, встановлення відносин між ними, нормалізацію структури для уникнення аномалій та реалізацію цієї структури в конкретній базі даних.

Виходячи з вимог виявлено такі таблиці БД:

- таблиця «Користувачі» (табл. 3.1);
- таблиця «Ролі» (табл. 3.2);
- таблиця «Запити» (табл. 3.3);
- таблиця «Локації» (табл. 3.4);
- таблиця «Адреси» (табл. 3.5).

Таблиця 3.1 – Таблиця «Користувачі»

Table	Fields	Type	Length	Nullable
Користувачі	id	int	20	-
	first name	varchar	64	-
	last name	varchar	64	-
	role id	int	20	-
	email	varchar	64	-
	password	varchar	64	-

Таблиця 3.2 – Таблиця «Ролі»

Table	Fields	Type	Length	Nullable
Ролі	id	int	20	-
	title	varchar	64	-

Таблиця 3.3 – Таблиця «Запити»

Table	Fields	Type	Length	Nullable
Запити	id	int	20	-
	type	boolean	-	-
	recipient_comment	text	255	+
	worker_comment	text	255	+
	status	varchar	64	-
	location_id	int	20	-
	recipient_id	int	20	-

Таблиця 3.4 – Таблиця «Локації»

Table	Fields	Type	Length	Nullable
Локації	id	int	20	-
	title	varchar	64	-
	description	text	255	-
	schedule	datetime	-	+
	address_id	int	20	+
	district_id	int	20	-

Таблиця 3.5 – Таблиця «Адреси»

Table	Fields	Type	Length	Nullable
Адреси	id	int	20	-
	address	varchar	150	-
	description	text	255	-
	city_id	int	20	-

Також виявлено такі таблиці БД:

- таблиця «Райони» (табл. 3.6);
- таблиця «Міста» (табл. 3.7);
- таблиця «Отримувачі» (табл. 3.8);
- таблиця «Типи отримувачів» (табл. 3.9);
- таблиця «Відгуки» (табл. 3.10).

Таблиця 3.6 – Таблиця «Райони»

Table	Fields	Type	Length	Nullable
Райони	id	int	20	-
	district	varchar	64	-
	description	text	255	-

Таблиця 3.7 – Таблиця «Міста»

Table	Fields	Type	Length	Nullable
Міста	id	int	20	-
	city	varchar	64	-

Таблиця 3.8 – Таблиця «Отримувачі»

Table	Fields	Type	Length	Nullable
Отримувачі	id	int	20	-
	first_name	varchar	64	-
	last_name	varchar	64	-
	phone	varchar	64	-
	date_of_birth	datetime	-	-
	city	varchar	32	-
	gender	boolean	-	-
	type_id	int	20	-

Таблиця 3.9 – Таблиця «Типи отримувачів»

Table	Fields	Type	Length	Nullable
Типи отримувачів	id	int	20	-
	title	varchar	64	-

Таблиця 3.10 – Таблиця «Відгуки»

Table	Fields	Type	Length	Nullable
Відгуки	id	int	20	-
	review	text	255	-
	recipient_id	int	20	-
	is_checked	boolean	-	-

Відношення «Користувачі» містить список користувачів вебдодатку.

Відношення «Ролі» містить ролі користувачів вебдодатку.

Відношення «Запити» містить у собі всі запити отримувачів.

Відношення «Локації» описує всі локації для видачі гуманітарної допомоги..

Відношення «Адреси» містить у собі список адрес локацій.

Відношення «Райони» містить у собі список округів.

Відношення «Міста» містить список міст та населених пунктів для локацій.

Відношення «Отримувачі» містить у собі список отримувачів гуманітарної допомоги.

Відношення «Типи отримувачів» містить у собі список категорій отримувачів: ВПО та СЖО.

Відношення «Відгуки» описує всі відгуки створені отримувачами.

На підставі виявлених відношень та детальної інформації про поля які містяться у кожній з таблиць виділено сутності, які зберігається в БД:

- сутність «Користувачі»;
- сутність «Ролі»;
- сутність «Запити»;
- сутність «Локації»;
- сутність «Адреси»;
- сутність «Райони»;
- сутність «Міста»;
- сутність «Отримувачі»;
- сутність «Типи отримувачів»;
- сутність «Відгуки».

В БД, зв'язки між сутностями визначають способи взаємодії між ними. Зв'язки вказують на те, як одна сутність пов'язана з іншою та як вони взаємодіють в межах бази даних. Зазвичай зв'язки визначаються відносинами між таблицями в базі даних, і вони реалізуються за допомогою ключів.

Зв'язок М:1 (рис. 3.1) описується між сутностями «Користувачі» і «Ролі». Один роль може мати кілька користувачів.



Рисунок 3.1 – Зв'язок «Користувачі» - «Ролі»

Виходячи з отриманих відношень система має зв'язок M:1 (рис. 3.2) між сутностями «Отримувачі» і «Типи отримувачів». Один тип може міститись у декількох отримувачів.

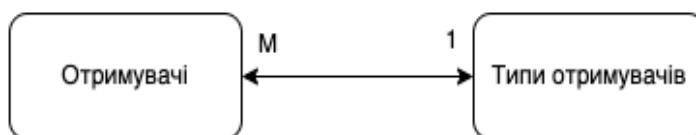


Рисунок 3.2 – Зв'язок «Отримувачі» - «Типи отримувачів»

Зв'язок 1:M (рис. 3.3) з'являється між сутностями «Запити» і «Отримувачі». Один отримувач може мати кілька запитів.



Рисунок 3.3 – Зв'язок «Запити» - «Отримувачі»

Також маємо зв'язок M:1 (рис. 3.4) який описується між сутностями «Запити» і «Локації». Один локація може містити кілька локацій.



Рисунок 3.4 – Зв'язок «Запити» - «Локації»

Виходячи з формулювання вимог та технічного завдання БД має зв'язок 1:M (рис. 3.5) між сутностями «Отримувачі» та «Відгуки». Один отримувач може розмістити кілька відгуків.

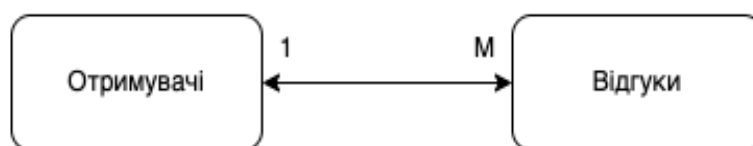


Рисунок 3.5 – Зв'язок «Отримувачі» - «Відгуки»

Зв'язок 1:1 (рис. 3.6) який є між сутностями «Локації» і «Адреси». Один локація може тільки одну адресу і навпаки.



Рисунок 3.6 – Зв'язок «Локації» - «Адреси»

Також зв'язок М:1 (рис. 3.7) міститься між сутностями «Локації» і «Райони». Декілька локацій може знаходитись в одному районі.



Рисунок 3.7 – Зв'язок «Локації» - «Райони»

Також маємо зв'язок М:1 (рис. 3.8) яка описується між сутностями «Адреси» і «Міста». Один місто або населений пункт може містити кілька адрес.



Рисунок 3.8 – Зв'язок «Адреси» – «Міста»

ER-діаграма є графічним інструментом для моделювання взаємозв'язків між різними сутностями в інформаційній системі. Цей вид діаграми є основним елементом при проєктуванні баз даних і допомагає відображати та аналізувати структуру даних.

Виходячи з результатів виявлення сутностей та відносин між ними створено ER-діаграма БД (рис. 3.9).

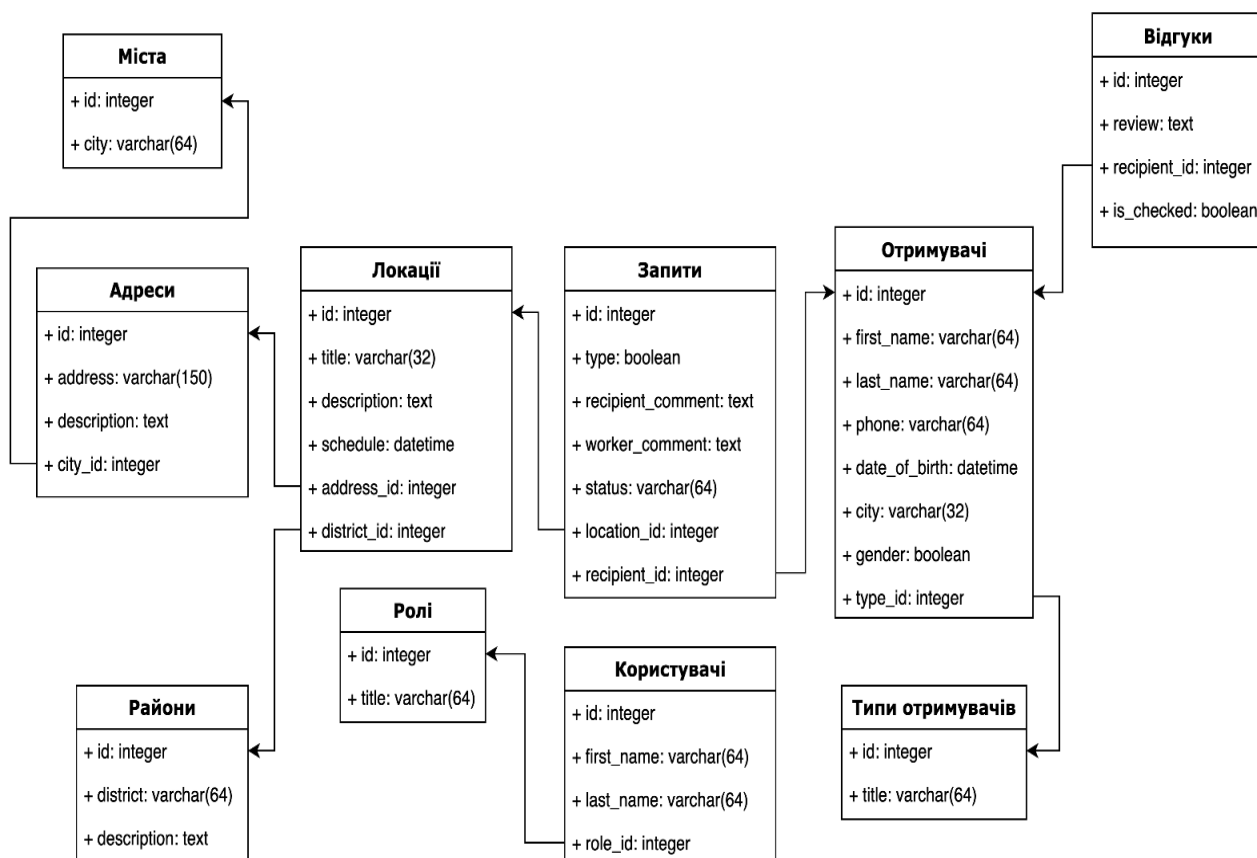


Рисунок 3.9 – ER-діаграма БД

3.2 Проектування структури та алгоритму роботи вебсистеми

Структура вебдодатку - це організаційна схема або логічна конструкція додатку, яка визначає, як його компоненти взаємодіють між собою та як користувачі взаємодіють з інтерфейсом. Коректна структура вебдодатку є важливою для забезпечення ефективності, легкості розробки та розширення, а також для зручності використання користувачами.

Основні принципи побудови структури вебдодатку:

- забезпечення інтуїтивно зрозумілої логіки взаємодії з вебдодатком, використовуючи природні та очікувані елементи інтерфейсу;
- адаптація до змін, забезпечуючи гнучкість та легкість оновлення;
- розміщення ключових функцій так, щоб користувачі легко знаходили та переходили між ними;
- використання звичайних елементів управління;

- вкладеність сторінок каталогу повинна бути логічною;
- забезпечення можливості фільтрації та пошуку для швидкого доступу до конкретної інформації.

Виходячи з SRS специфікації та основних принципів побудови структури вебсистема адаптована для кожного з користувачів: «Отримувача» (рис. 3.10), «Робітника складу» (рис. 3.11) та «Робітника на видачі» (рис. 3.12). Структура для «Адміністратора» зображена в додатку В.



Рисунок 3.10 – Структура вебдодатку для «Отримувача»

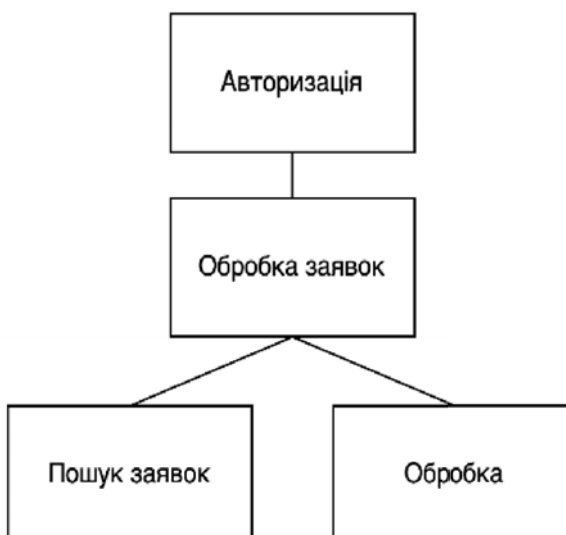


Рисунок 3.11 – Структура вебдодатку для «Робітника складу»



Рисунок 3.12 – Структура вебдодатку для «Робітника на видачі»

Розглянемо основний алгоритм роботи робітника на видачі:

- робітник на видачі заповнює форму пошуку локації валідними даними;
- робітник на видачі натискає кнопку «Пошук»;
- робітник на видачі натискає кнопку «Перейти до видачі» на знайдений локації;
- робітник на видачі шукає потрібного отримувача (якщо не знаходить: створює його);
- робітник на видачі натискає кнопку «Видати допомогу» та натискає «Видати» кнопку у формі;
- робітник на видачі видає допомогу отримувачу;
- створюється новий запит для отримувача на майбутню доступну дату на локації (Якщо немає доступною дати у майбутньому - відображається помилка).

Креслення алгоритму роботи робітника на видачі зображено в додатку Г.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ

4.1 Розробка архітектури клієнтської частини

React.js надає ефективні засоби для розробки інтерфейсів на основі компонентів. Архітектура React (рис. 4.1), у поєднанні з популярними бібліотеками управління станом, такими як Redux, забезпечує зручність та структурованість розробки. Структуруємо проєкт, розділяючи його на компоненти, контейнери та використовувати Redux для керування станом. Кожен компонент відповідає за свою сферу відповідальності, що спрощує підтримку та розуміння коду.

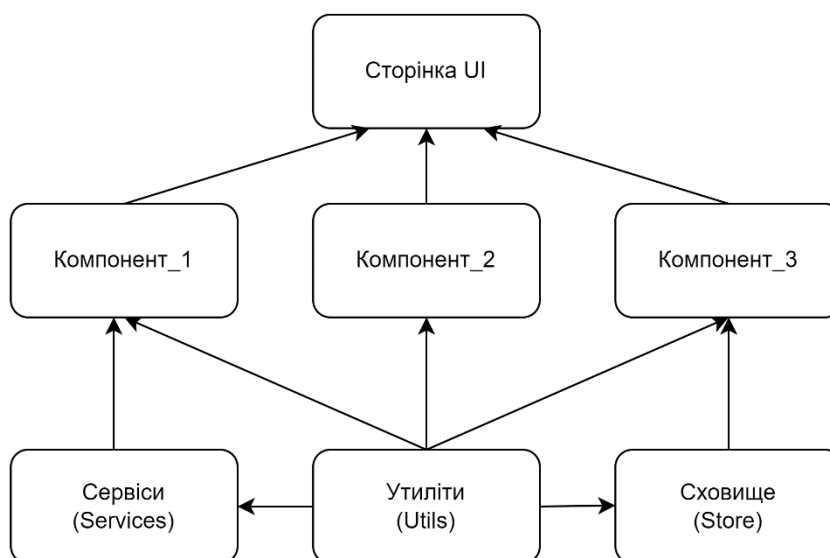


Рисунок 4.1 – Архітектура React.JS

Отже, Services в React може бути будь-якою окремою частиною коду, яка відповідає за виконання конкретних завдань, таких як робота з сервером (наприклад, отримання або відправлення даних), автентифікація, робота з локальним сховищем, аналітика тощо. Наприклад:

- сервіс ApiService – надає інтерфейс для взаємодії з сервером чи зовнішніми API. Може містити методи для отримання та відправлення даних;

- сервіс `AuthService` – відповідає за управління автентифікацією та авторизацією користувача. Може містити функції для входу, виходу та перевірки прав доступу.

`Utils` – це модуль чи клас, який містить різноманітні допоміжні функції та методи для роботи з різними операціями, не пов'язаними прямо з логікою відображення і взаємодії з користувачем. Наприклад:

- утиліта `DateUtils` – забезпечує функції для роботи з датами, такі як форматування чи обчислення різниці між датами;
- утиліта `StringUtils` – містить функції для обробки рядків та текстової інформації, такі як перетворення регістрів чи обрізка пробілів.

“`Redux`” - це бібліотека для керування станом застосунків в `JavaScript`, особливо в контексті інтерфейсів користувача та односторінкових додатків. `Redux` реалізує парадигму архітектури `Flux` і відокремлює стан застосунку від компонентів.

`Store` у контексті `Redux` відповідає за централізоване сховище стану вашого додатка. `Redux` надає паттерн управління станом, де весь стан додатка зберігається в одному місці - у `store`. Цей `store` використовується для забезпечення прогнозованої та однозначної зміни стану додатка. Основні частини `Redux`:

- `actions` – об'єкти, які містять інформацію про події, що відбуваються у додатку, наприклад, `ADD_TODO` чи `FETCH_DATA_SUCCESS`;
- `actionCreators` – функції, які створюють `actions`. Вони допомагають уникнути дублювання коду при створенні `actions`;
- `reducers` – функції, які змінюють стан додатка в залежності від `actions`. Вони повинні бути чистими функціями, які не залежать від зовнішніх факторів і завжди повертають новий стан;
- `store` – об'єкт, який утримує весь стан додатка. Створюється за допомогою функції `createStore` з бібліотеки `Redux`.

`Components` у контексті `React` - це візуальні частини інтерфейсу користувача, які реагують на зміни стану та взаємодіють з користувачем. Компоненти можуть бути як простими, так і складними, та вони можуть включати

в себе інші компоненти. Розподіл інтерфейсу на компоненти допомагає полегшити управління кодом та станом. Наприклад:

- компонент `CommonComponents` – включає загальні компоненти, які можуть бути використані у всьому додатку. Це забезпечує повторне використання та стандартизацію;
- компонент `FeatureComponents` – містить компоненти, які використовуються в конкретних функціональних частинах додатка. Кожен компонент має свою визначену відповідальність.

4.2 Розробка архітектури серверної частини

Створено клієнт-серверну архітектуру вебсистеми (рис. 4.2), яка взаємодія між користувачами та сервером відбувається через клієнтські програми, а дані зберігаються в БД.

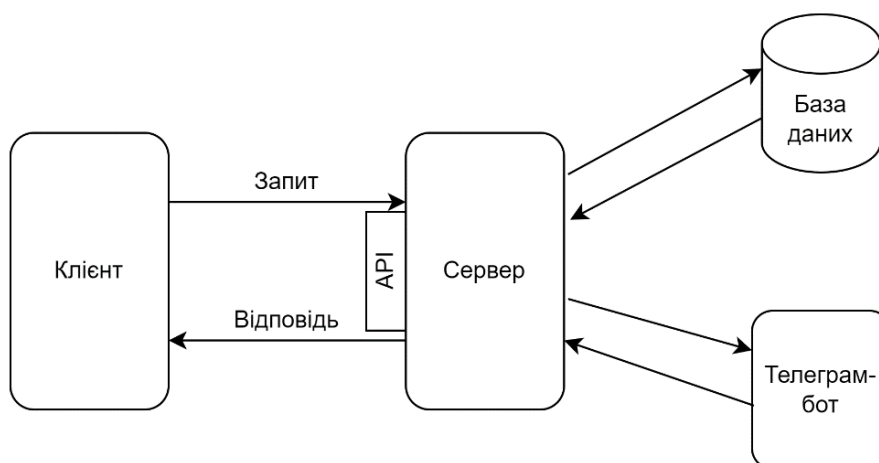


Рисунок 4.2 – Архітектура вебсистеми

Клієнт – програма, за допомогою якої користувачі взаємодіють із системою. Клієнти надсилають запити серверу та отримують відповіді.

Сервер – центральний вузол системи, яка обробляє запити від клієнтів. Він відповідає за бізнес-логіку, обробку даних та взаємодію з базою даних та телеграм-ботом.

БД – зберігає дані, необхідних функціонування системи. Запити до бази даних можуть виконуватися для отримання, оновлення або видалення інформації.

Телеграм-бот – інтегрований бот в Telegram, що надає користувачеві інтерфейс для взаємодії із системою. Бот може приймати команди від користувачів, надсилати повідомлення та отримувати інформацію з сервера.

Визначимо процес взаємодії цих компонентів:

- користувач надсилає запит через клієнтську програму або телеграм-бот;
- запит надходить на сервер;
- сервер обробляє запит, виконує необхідні дії та, при необхідності, взаємодіє з базою даних для отримання або оновлення даних;
- сервер формує відповідь;
- відповідь може містити запитані дані або підтвердження дії;
- відповідь відправляється назад користувачеві через клієнтську програму або Телеграм-бот;
- дані зберігаються або оновлюються у базі даних;
- якщо запит користувача включає зміну даних, сервер оновлює інформацію в базі даних для подальшого використання.

Django – це високорівневий вебфреймворк на Python, який полегшує створення вебдодатків. Використовуємо архітектуру заснована на моделі MTV (рис. 4.3), де модель відповідає за роботу з даними, шаблон – за представлення, а представлення – за бізнес-логіку.

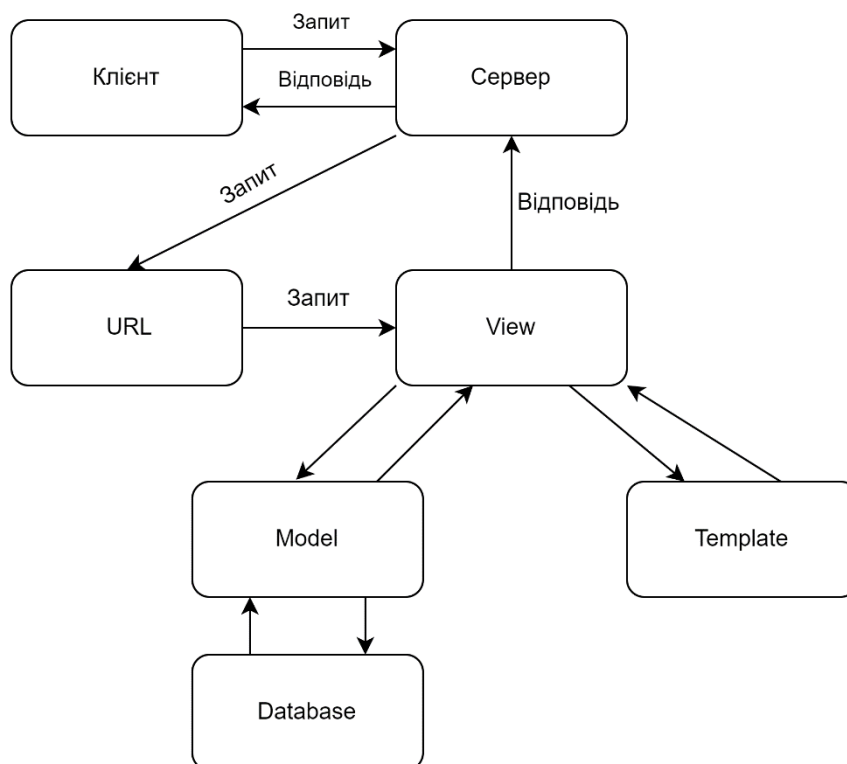


Рисунок 4.3 – MTV архітектура в Django для вебдодатку

Отже, модель відповідає за представлення даних та бізнес-логіку програми. Django моделі описуються класами Python, які визначають структуру бази даних. Модель не тільки описує структуру БД, але також служить інтерфейсом для взаємодії з БД через ORM. Django ORM дозволяє працювати з БД, не використовуючи мову SQL безпосередньо. Застосування моделі Django наведено у додатку Д.1.

В шаблонах використовуємо мову розмітки, подібну до HTML, зі вставками спеціальних тегів і змінних Django. Вони надають дані користувачеві. Застосування цього шаблону наведено у додатку Д.2.

Представлення обробляють запити від користувача, взаємодіють з моделями для отримання даних та передають їх у шаблони. Застосування цього шаблону наведено у додатку Д.3.

Архітектура Django також включає ORM для зручної роботи з БД, вбудований адміністративний інтерфейс для управління даними, а також безліч вбудованих інструментів для обробки URL, форм, сесій та безпеки.

QuerySet - це абстракція, що надається Django ORM, що дозволяє будувати запити до БД для отримання, фільтрації та маніпуляції даними. Застосування цього шаблону наведено у додатку Д.4.

За допомогою менеджерів Django створюємо інтерфейс для роботи з моделями. За замовчуванням кожна модель Django має власний менеджер. Застосування цього шаблону наведено у додатку Д.5.

4.3 Архітектура телеграм-бота

Архітектура телеграм-бота (рис. 4.4) визначає структуру та взаємодію компонентів, що дозволяє боту ефективно виконувати свої функції у віртуальному просторі месенджера Telegram. Здатність взаємодіяти з користувачами, обробляти їх запитання та команди, а також виконувати завдання в реальному часі визначається не лише функціоналом, але і архітектурою бота. В цьому контексті, розгляд архітектури телеграм-бота охоплює важливі аспекти, такі як взаємодія з Telegram API, обробка повідомлень, бізнес-логіка, інтеграція з базою даних та інші компоненти, які сприяють оптимізованій та ефективній роботі бота у цифровому середовищі месенджера.

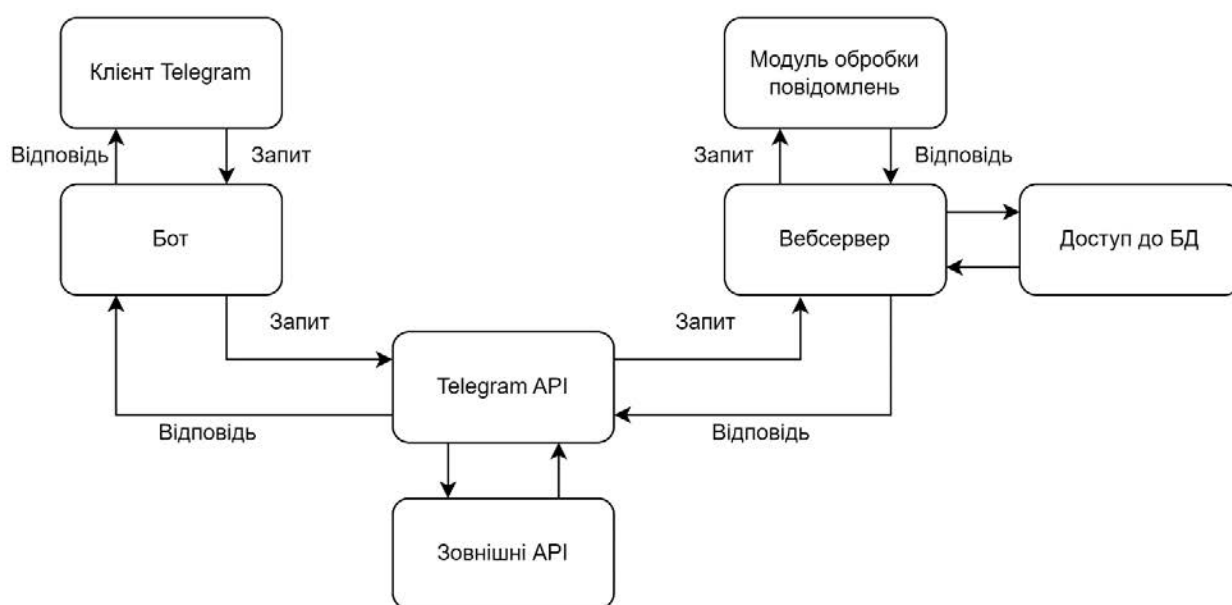


Рисунок 4.4 – Архітектура телеграм-бота

Клієнт Telegram – це програма, яку використовують користувачі для взаємодії з ботом. Бот може відправляти повідомлення, зображення, відео тощо через Telegram API.

Вебсервер або хмарна платформа – деякі боти можуть використовувати вебсервер для обробки запитів, інші можуть бути розгорнуті на хмарних платформах, таких як Heroku або AWS. Вони отримують вхідні повідомлення від Telegram API і обробляють їх.

Модуль обробки повідомлень – це місце, де виконується обробка всіх вхідних повідомлення від користувачів. Код реагує на команди, запитання, або будь-яку іншу взаємодію.

Доступ до БД (опціонально) – якщо бот повинен зберігати або витягувати дані, вам може знадобитися БД. Це може бути SQLite, MySQL, або будь-яка інша БД.

Зовнішні API (опціонально) – якщо бот потребує додаткового функціоналу, який можна отримати через зовнішні сервіси, ви можете інтегрувати його, використовуючи API.

4.4 Програмна реалізація системи

4.4.1 Авторизація

На основі вимог технічного завдання створено сторінку авторизації для користувачів, які ще не увійшли в систему (рис. 4.5). У фрагменті коду, представленому у додатку Д.6, використано локальний стан для відстеження значень електронної пошти та пароля з форми, для цього використовуємо метод `useState`. Функція `handleLogin` викликається при відправці форми. У функції виконується асинхронний POST-запит до `/api/auth/login` з даними авторизації.

Якщо запит успішний, у консоль виводиться повідомлення та зберігається отриманий токен в локальному сховищі (`localStorage`). Колбек `onLogin` призначений для оновлення стану авторизації в батьківському компоненті.

Рисунок 4.5 – Сторінка «Авторизація»

Кожне поле у формі має обробник подій onChange, який викликає setCredentials, оновлюючи локальний стан.

4.4.2 Локації

Також виходячи з технічного завдання розроблено сторінку «Створення локації», яка містить форму адреси (рис. 4.6) та форму розкладу (рис. 4.7) для користувача з роллю Адміністратор. Він може створити адресу та розклад на локації.

Рисунок 4.6 – Форма адреси на сторінці «Створення локації»

Рисунок 4.7 – Форма розкладу на сторінці «Створення локації»

У компоненті `LocationForm` також використано локальний стан для відстеження значень форми та асинхронний POST-запит для створення нової локації на сервері. Це застосування наведено у додатку Д.7. Використано `useState` для створення локального стану компоненту для відстеження значень полів форми. `handleSubmit` викликається при відправці форми. Вона відправляє POST-запит на сервер з даними локації для створення нової локації. викликається асинхронний POST-запит до `/api/locations` з даними локації (`locationData`). У випадку успішної відповіді, виводиться повідомлення у консоль та очищається форма для можливості введення нових даних.

Також як в авторизації використовуємо обробник подій `onChange`, який викликає `setLocationData`, оновлюючи локальний стан.

4.4.3 Обробка запиту на складі

Для того щоб Робітник складу обробляв запити від отримувачів розроблено сторінку «Обробка запитів» (рис. 4.8). Робітник складу має шукати потрібну дату на локації: обробляти запити та залишати коментарі.

Локація: test

Статус: Очікує обробки x v | Дата видачі: 21.11.2023 x v | Примітка: | Знайти

Оброблено	Статус	Примітка	Залишити коментар
☐	Очікує обробки	Хліб, макарони	

Оновити заявки

Усього: 1

ⓘ Зміни не будуть автоматично збережені, якщо Ви зробите пошук або перейдете на наступну сторінку. Не забудьте натиснути кнопку "Оновити заявки"

Рисунок 4.8 – Сторінка «Обробка запитів»

В компоненті `PackageFilterForm`, який наведено у додатку Д.8, використано локальний стан для відстеження значень форми та асинхронний GET-запиту для фільтрації пакетів на сервері. Метод `useState` відстежує значення для дати та статусу пакетів. При поданні форми викликається функція `handleSubmit` та при цьому викликається GET-запит на `/api/packages` з параметрами фільтрації, які знаходяться в `filterData`. Коли результати отримані, викликається переданий колбек `onFilter`, який призначений для оновлення стану компонента, який відображає результати фільтрації.

В компоненті `PackageList`, який наведено у додатку Д.9 відображається список гуманітарних пакетів та надається можливість оновлювати їх статус за допомогою кнопок.

4.4.4 Видача допомоги

Виходячи з формулювання вимог розроблено сторінку «Видача допомоги» (рис. 4.9), на якій Робітник на видачі відкриває потрібну локацію та шукає отримувача через форму пошуку для видачі допомоги. В компоненті `RecipientsListPage` який наведено у додатку Д.10, представлено сторінку зі списком отримувачів допомоги. Змінна `recipients` зберігає весь список отримувачів, `searchTerm` зберігає значення для пошуку за іменем або містом отримувача та `filteredRecipients` зберігає відфільтрований список отримувачів відповідно до

введеного пошукового терміну. Функція `useEffect` використана для отримання списку отримувачів при завантаженні сторінки. Вона викликається при першому рендері компонента та оновлює стани `recipients` та `filteredRecipients` із отриманими даними.

Функція `handleSearch` створена для фільтрації отримувачів за введеним терміном пошуку. Викликається при натисканні кнопки пошуку. Використовує `filter` для відфільтрування отримувачів за іменем або містом.

Функція `handleCreateRequest` використовується для створення запиту на отримання допомоги. Викликається при натисканні кнопки "Створити запит".

Локація: test 📅

Дмитро Знайти Створити отримувача

ПІБ	Особистий номер телефону	Дії	
Дмитро Дмитро Дмитро	+380 98 665 2386	Видати допомогу	Створити запит
Дмитро Дмитро Дмитро	+380 98 665 2387	Видати допомогу	Створити запит
Дмитро Дмитро Дмитро	+380 98 665 2380	Видати допомогу	Створити запит

Рисунок 4.9 – Сторінка «Видача допомоги»

4.4.5 Реалізація серверної частини

Виходячи з проєктування БД та визначення сутностей, створено моделі Django. Додаток Д.11 описує фрагмент створення сутностей:

- `location` – місце, де відбувається видача гуманітарної допомоги. Має поля, такі як: `title`, `description`, `schedule`.
- `recipient` – відображає отримувача гуманітарної допомоги. Має поля, такі як: `first_name`, `last_name`, `phone`, `date_of_birth`, `city`, `gender`.
- `recipient_type` – типи отримувачів. Має поле `title`.

Файл представлення, який наведено у додаток Д.12 описує створення представлень: `LocationViewSet`, `RecipientViewSet` та `RecipientType` з використанням CRUD операцій

Серіалізатори використано для перетворення об'єктів Django у формат JSON та навпаки. Код серіалізаторів наведено у додаток Д.13.

Файл `urls.py` описує налаштування URL-шляхів для `views` Django. Код файлу наведено додаток Д.14. Метод `router` використано для автоматичного визначення URL-шляхів на основі визначених представлень.

Створено додаткові методи до представлень для обробки видачі та створення запиту. Код наведено у додатку Д.15. Метод `provide_help` викликається методом POST на URL, який включає ідентифікатор конкретного пакета. За допомогою метода `get_object` отримуємо конкретний об'єкт пакета. Змінюємо статус пакета на 'shipped' (відправлено). Зберігаємо змінену модель пакета. Повертаємо відповідь, яка містить інформацію про те, що допомога була надана. Метод `create_request` також викликається методом POST на URL, який включає ідентифікатор конкретного пакета. У цьому випадку, логіка створення запиту на отримання допомоги ще не визначена і повертає простий об'єкт відповіді, що підтверджує створення запиту.

У файлі `admin.py`, який наведено додатку Д.16 зареєстровано моделі в адміністративному інтерфейсі Django, для управління ними через Django Admin.

4.4.6 Реалізація телеграм-бота

Виходячи з розроблених вимог та аналізу цільової аудиторії, створено телеграм-бот (рис. 4.10) який задовольняє потреби Отримувачів гуманітарної допомоги. Функція `start` викликається при команді `/start`. Вона відправляє повідомлення користувачеві, яке містить Inline-клавіатуру з кнопкою "Відправити номер телефону". Після виклику цієї команди бот очікує від користувача відправлення номера телефону. Функція повертає `PHONE`, новий стан конверсації.

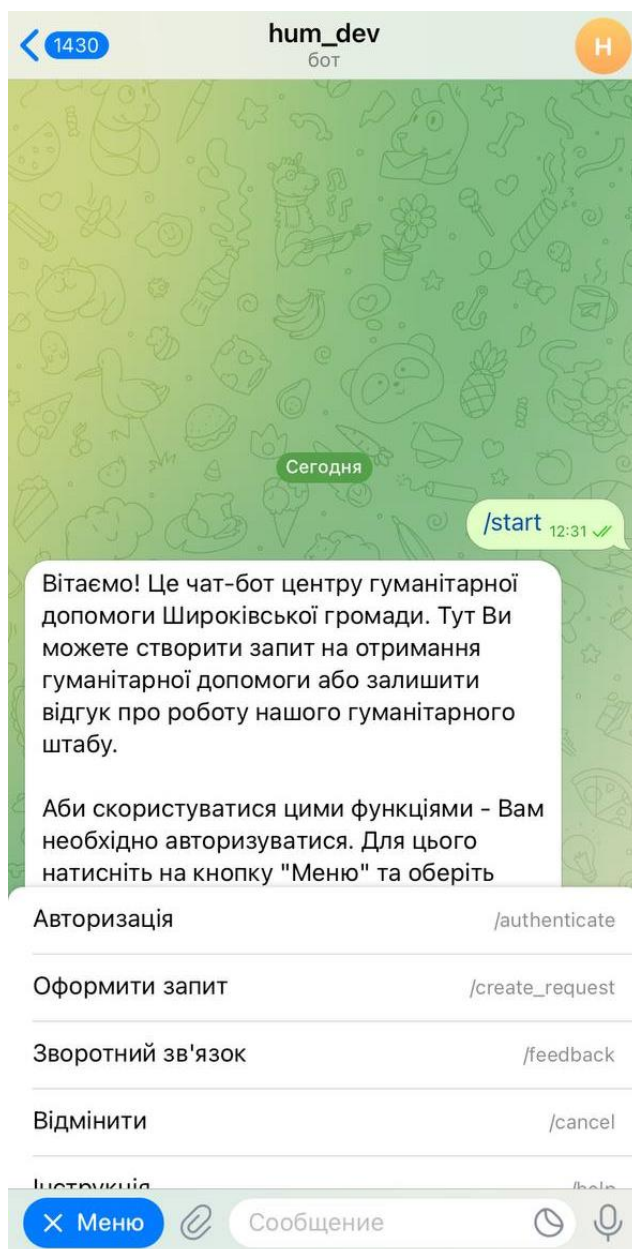


Рисунок 4.10 – Розроблений телеграм-бот

Функція `get_phone` викликається при отриманні номера телефону від користувача. Вона відправляє номер телефону до Django API для подальшої обробки. Після успішного відправлення номера телефону бот відправляє повідомлення користувачеві про успішну реєстрацію номера телефону та завершує конверсацію, повертаючи `ConversationHandler.END`.

Функція `humanitarian_request` викликається при отриманні текстового повідомлення від користувача з командою "Запит на гуманітарну допомогу". Вона отримує ID користувача та опис гуманітарного запиту і відправляє ці дані до Django

API за допомогою HTTP-запиту. Функція `main` є основною точкою входу програми. Вона створює об'єкт `Updater` для з'єднання з Telegram API та об'єкт `Dispatcher` для обробки подій та повідомлень. Далі вона додає обробники команд та повідомлень, включаючи обробник конверсації, та запускає цикл обробки оновлень бота.

Файл `bot.py` наведено у додатку Д.17.

5 ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ВЕБСИСТЕМИ

5.1 Створення тест-плану

Тест-план є документом, який визначає стратегію, обсяг, підхід та ресурси, які будуть використовуватися для виконання тестування програмного продукту. Цей документ визначає, як будуть проводитися тести, які тести будуть виконані, хто виконуватиме тести, та інші важливі аспекти тестування.

5.1.1 Scope of testing

Всі функції визначені у вимогах до ПЗ повинні пройти тестування:

- авторизація користувачів;
- керування командою (робітники складу та видачі);
- менеджмент округів та локацій;
- обробка запитів на допомогу;
- отримання та видача допомоги та оформлення наступної допомоги;
- створення нових отримувачів;
- менеджмент відгуків.

5.1.2 Out of scope

Цей функціонал не є предметом тестування, оскільки відсутній у специфікаціях вимог до ПЗ:

- third party;
- стрес тестування;
- багатомовна підтримка;
- використання мобільних пристроїв окрім планшетів.

5.1.3 Типи тестування

Планується провести 10 типів тестування:

- модульне тестування – перевірка правильності роботи окремих модулів чи компонентів програми;
- інтеграційне тестування – перевірка взаємодії між різними модулями чи компонентами системи;
- системне тестування – перевірка відповідності системи визначеним вимогам та очікуванням користувачів;
- end-to-end тестування - перевірка функціональності та взаємодії всіх компонентів системи з точки зору кінцевого користувача;
- регресійне тестування – перевірка того, чи нові зміни в коді не впливають на наявний функціонал системи;
- UI/UX тестування – оцінка користувацького інтерфейсу (UI) та користувацького досвіду (UX) для переконання, що вони відповідають дизайнерським та функціональним вимогам та очікуванням користувачів;
- функціональне тестування – перевірка функціональності програмного забезпечення для забезпечення відповідності вимогам та очікуванням;
- API тестування – оцінка правильності та надійності інтерфейсу програмного забезпечення для взаємодії з іншими програмами чи компонентами;
- тестування безпеки – перевірка системи на виявлення вразливостей та забезпечення заходів безпеки;
- регресійне тестування – тестування для підтвердження того, що нещодавня зміна програми або коду не вплинула на наявні функції.

Загальна концепція тестування збудована на класичній піраміді тестування (рис. 5.1), яка включає в себе модульне, інтеграційне, системне та end-to-end тестування. Згідно концепції тестування відбувається «знизу-догори».

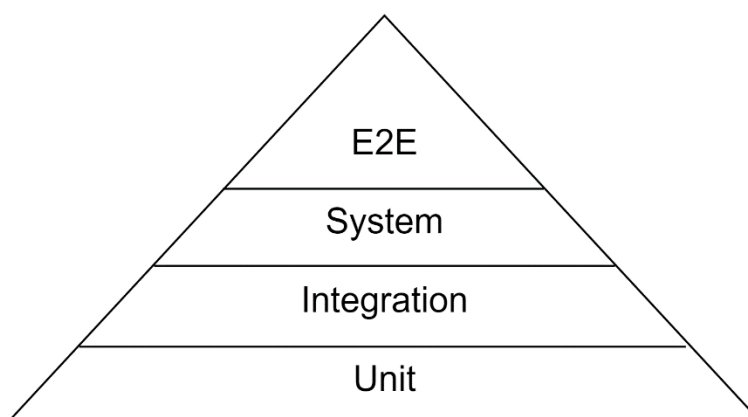


Рисунок 5.1 – Класична піраміда тестування

5.1.4 Техніки тест-дизайну

Планується використовувати 6 типів тестування:

- еквівалентність значень – ця техніка передбачає вибір представників класів еквівалентності вхідних даних для тестування. Якщо вхідні дані можуть поділитися на групи, які поведуть себе однаково, вибір лише одного представника з кожної групи може бути достатнім для тестування;
- граничні значення – тут акцент робиться на виборі тестових випадків, які перевіряють поведінку системи на границях діапазону вхідних даних. Це важливо, оскільки саме на границях можуть виникати непередбачені ситуації;
- парний аналіз – техніка включає в себе розгляд всіх можливих комбінацій значень вхідних характеристик. Це дозволяє ефективно виявляти можливі взаємодії між параметрами системи;
- стратегія вибіркового тестування – тут важливо вибрати тестові випадки на основі аналізу функціональності та ідентифікації потенційних ризиків. Спрямоване тестування на найважливіших аспектах може бути більш ефективним;
- тестування структур – ця техніка базується на аналізі внутрішньої структури програми. Включає тестування окремих функцій, процедур або алгоритмів;

– таблиця рішень – це інструмент для систематизації та відстеження різних варіантів вирішення певних проблем чи сценаріїв. Така таблиця часто використовується для нагляду за перевагами та недоліками різних альтернатив.

5.1.5 Критерії входу

Визначено 6 критеріїв входу в тестування:

- документ з вимогами готовий – усі функціональні та нефункціональні вимоги до програмного забезпечення були задокументовані, переглянуті та затверджені. Будь-які зміни вимог були задокументовані та повідомлені;
- тест-план готовий – план тестування та формати документації створені та доступні для використання;
- тестове середовище готове – тестове середовище, включаючи апаратне забезпечення, програмне забезпечення та мережеві конфігурації, налаштовано та доступне для тестування. Максимально відповідає виробничому середовищу;
- тестові інструменти та ресурси готові – необхідні інструменти та ресурси тестування, такі як програмне забезпечення для керування тестуванням і тестові дані, наявні та доступні;
- розклад випробувань визначено – розклад проєкту, включаючи основні етапи та терміни тестування, визначено та узгоджено. Це включає розуміння дат початку та завершення етапів тестування;
- процес управління дефектами визначено – встановлено процес звітування, відстеження та керування дефектами, а також налаштовано інструменти для відстеження дефектів.

5.1.6 Критерії призупинення

Визначено 5 критеріїв призупинення в тестуванні:

- нестабільність середовища – якщо тестове середовище стає нестабільним або не відповідає заздалегідь визначеним вимогам (наприклад,

недоступність ключових ресурсів), тестування може бути призупинено до відновлення стабільності;

- ідентифікація критичних дефектів – якщо під час тестування буде виявлено критичний дефект (Blocker or Critical), який серйозно впливає на функціональність продукту або загрожує його безпеці, тестування може бути призупинено до вирішення проблеми;

- зміна вимог – якщо під час виконання проєкту відбуваються значні зміни у вимогах або функціональності продукту, які не відповідають поточним тестам, тестування може вимагати перегляду та зміни тестових випадків;

- відхилення від графіка – якщо тестування сповільнюється і існує ризик затримки проєкту, можливо, потрібно буде переглянути графік або додати додаткові ресурси;

- недоступність ключових ресурсів – якщо ключові ресурси тестування, такі як QA, тестові дані або тестове середовище, стають недоступними або обмеженими, тестування може знадобитися призупинити.

5.1.7 Критерії виходу

Визначено 8 критеріїв виходу з тестування:

- виконання тестів – виконано 100% запланованих тестів;
- успішність проходження тестів – принаймні 90-95% тестів пройшли успішно без критичних дефектів;

- виправлені критичні дефекти – усі критичні дефекти, виявлені під час тестування, були виправлені, а повторне тестування показує, що вони були усунені. Залишилося нуль критичних дефектів;

- дефекти з високим і середнім пріоритетом – жодні високопріоритетні дефекти не повинні бути видатними. Не більше 5% дефектів середнього пріоритету мають бути невирішеними;

- усунення дефектів – усі некритичні дефекти реєструються, відстежуються та мають план усунення;

- очищення тестових даних – тестові дані, створені з метою тестування, було повністю видалено або повернуто до початкового стану;
- тестова документація завершена – усі артефакти тестування, включаючи плани тестування, тестові випадки, тестові дані та звіти про тестування, повністю задокументовані та оновлені;
- архівація тестових результатів – усі тестові артефакти та результати, включаючи тестові сценарії, документацію та звіти, архівуються та зберігаються для подальшого використання та аудиту.

5.1.8 Системний ресурс

Інструментальні засоби:

- Postman – API тестування;
- OWASP ZAP – тестування безпеки;
- Cypress – автоматизоване тестування;
- Sentry – моніторинг та логування помилок.

5.1.9 Визначення пріоритету помилок

Визначено 6 пріоритетів помилок:

- blocker – це найсерйозніші та критичні проблеми, які взагалі перешкоджають функціонуванню програмного забезпечення або спричиняють повний збій його основних функцій. Ці помилки необхідно негайно усунути перед подальшим тестуванням або розробкою. Вони можуть зупинити весь проєкт або зробити його непридатним для використання;
- highest – це серйозні проблеми, які суттєво впливають на функціональність програмного забезпечення, але не обов'язково можуть спричинити повний збій. Ці помилки слід усунути терміново, як правило, до випуску програмного забезпечення для виробництва. Вони можуть включати

критичні вразливості безпеки, проблеми з втратою даних або серйозні проблеми з функціональністю;

- high – це важливі проблеми, які впливають на функціональність і взаємодію з програмним забезпеченням. Хоча ці проблеми не такі критичні, як помилки блокувальника або найвищого пріоритету, все ж вимагають негайної уваги. Вони можуть включати значні проблеми зручності використання, вузькі місця продуктивності або критичні функціональні помилки;

- medium – це помірні проблеми, які мають помітний, але не критичний вплив на функціональність програмного забезпечення або взаємодію з користувачем. Ці помилки часто можна усунути під час наступного циклу розробки або обслуговування. Зазвичай вони включають некритичні проблеми з функціональністю, незначні проблеми з зручністю використання або покращення продуктивності;

- low – це незначні проблеми, які суттєво не впливають на основні функції програмного забезпечення чи взаємодію з користувачем. Зазвичай ці помилки не є терміновими, і їх можна виправити в наступних випусках або оновленнях. Вони включають косметичні проблеми, незначні покращення зручності використання або запити на некритичні функції;

- lowest – це найменш серйозні проблеми, які мають мінімальний вплив на функціональність програмного забезпечення та взаємодію з користувачем. Вони можуть включати незначні налаштування інтерфейсу користувача, орфографічні чи граматичні помилки або дуже специфічні та рідкісні крайні випадки.

5.2 Ручне тестування

Ручне тестування – це процес перевірки та валідації функціоналу, ефективності та безпеки програмного продукту вручну, без використання автоматизованих засобів. Учасники тестування виконують тестові сценарії, спостерігають за реакцією системи та реєструють результати.

Користувач працює з вебдодатком за допомогою комп'ютера або планшета (мобильні пристрої з телеграм-ботом Отримувача не берем у залік) на базі iOS чи Android.

Desktop тестування проводилося на таких браузерях:

- Google Chrome (Latest);
- Opera (Latest);
- Firefox (Latest).

Tablet тестування проводилося на таких браузерях:

- Google Chrome (Latest);
- Safari (Latest).

На всіх браузерах що брали участь у тестах UI вебдодатку відображається згідно вимог.

Функціональне тестування вебдодатку проведено за наступною структурою:

а) авторизація:

- 1) авторизація;
- 2) відновлення пароля;
- 3) коректна обробка помилок;
- 4) вихід із системи;

б) сторінка «Команда»:

- 1) створення робітника складу/видачі;
- 2) редагування робітника складу/видачі;
- 3) видалення робітника складу/видачі;
- 4) коректна обробка помилок;

в) сторінка «Видалення отримувачів»:

- 1) видалення отримувачів;
- 2) коректна обробка помилок;

г) сторінка «Округи та локації»:

- 1) перегляд створених локацій та округів
- 2) створення локацій та округів;

- 3) редагування локацій та округів;
- 4) налаштування розкладу;
- 5) видалення локацій та округів;
- 6) коректна обробка помилок;
- 7) пошук локацій та округів;
- д) сторінка «Відгуки»:
 - 1) перегляд відгуків;
 - 2) видалення відгуків;
- е) сторінка «Обробка заявок на складі»:
 - 1) перегляд заявок;
 - 2) зміна статусів;
 - 3) пошук заявок за локаціями та датою;
 - 4) коректна обробка помилок;
- ж) сторінка «Видача допомоги»:
 - 1) перегляд локацій за видачею;
 - 2) пошук отримувачів;
 - 3) коректна обробка помилок;
 - 4) створення отримувачів;
 - 5) редагування отримувачів;
 - 6) видача допомоги;
 - 7) створення заявки на наступну видачу згідно розкладу.

Функціональне тестування телеграм-бота проведено за наступною структурою:

- а) авторизація;
- б) головне меню:
 - 1) зворотній зв'язок;
 - 2) запит на допомогу;
 - 3) інструкція.

Всі виявлені на етапі тестування помилки були усунені.

5.3 Автоматизоване тестування

Автоматизоване тестування – це процес використання спеціальних програмних засобів для виконання тестових сценаріїв та перевірки функціоналу програмного забезпечення. Замість того, щоб мануального виконання сценаріїв тестування, використовуються автоматизовані скрипти або інструменти для автоматичного виконання цих завдань.

Основні цілі UI автоматизації:

- забезпечення високої якості графічного інтерфейсу для задоволення очікувань користувачів;
- виявлення і усунення помилок та аномалій, пов'язаних з UI, для забезпечення стабільності функціоналу програмного продукту;
- використання автоматизованих тестів для перевірки того, чи не порушено раніше визначену функціональність під час внесення змін або додавання нового коду;
- тестувальники матимуть менше рутинних та монотонних завдань завдяки автоматизованій перевірці UI.

Обрано Cypress у якості інструмента для автоматизованого тестування з використанням PageObject шаблону. Cypress має простий та легкий у використанні синтаксис, що полегшує написання та розуміння тестів. Використовує JavaScript для написання тестів, що дозволяє використовувати існуючий код та бібліотеки. Має можливість тестування як вебінтерфейсу, так і REST API. Інтегрується з системами Continuous Integration та Continuous Deployment для автоматизованого запуску тестів.

Для прикладу використання Cypress представлено написання тестів на функціонал авторизації з використанням Page Object Pattern у Cypress. Спочатку потрібно створити Page Object для сторінки авторизації, а потім використовувати його у тестах.

Файл LoginPage.js, код якого наведено у додатку Д.18, описує процес створення Page Object шаблону для сторінки авторизації. Метод visit робить

перехід на сторінку авторизації. Метод `fillEmail` знаходить елемент з ідентифікатором `email` і вводить в нього надане `email` користувача. Метод `fillPassword` знаходить елемент з ідентифікатором `'password'` і вводить наданий пароль. Метод `clickLoginButton` знаходить елемент з ідентифікатором `loginBtn` і виконує на ньому клік. Рядок `“export default new LoginPage()”` експортує екземпляр класу `LoginPage`. Це дозволяє імпортувати та використовувати цей екземпляр в інших файлах без необхідності створювати новий екземпляр щоразу.

Файл `login.spec.js`, код якого наведено у додатку Д.19 описує відтворення двох тестів: позитивний та негативний. У першому тесті використовуємо коректні дані користувача для авторизації, у другому - некоректні, та перевіряємо обробку помилки. Використано хук `beforeEach`, щоб відвідувати сторінку входу перед кожним тестом. Метод `LoginPage.visit()` відповідає за перехід на сторінку авторизації. Методи `LoginPage.fillEmail()`, та `LoginPage.fillPassword()` відповідають за введення даних в поля `Email` та `Password`. Метод `LoginPage.clickLoginButton()` відповідає за клік елемента з ідентифікатором `loginBtn`. Тест у позитивному сценарії перевіряє що поточна URL-адреса сторінки містить підрядок `“/dashboard”`, що може вказувати на успішний вхід. Тест у негативному сценарії перевіряє що на сторінці є елемент з класом `“error-message”`, і цей елемент відображено.

Всі виявлені на етапі тестування помилки були усунені.

5.4 Дослідження стійкості вебдодатку

Навантажувальне тестування є важливою частиною процесу розробки для забезпечення ефективності та стійкості програмного продукту. Розглянемо проведення навантажувального тестування в рамках нашого проєкту та вибір інструменту, який найкраще відповідає нашим потребам.

Для виконання навантажувального тестування обрано інструмент `Locust` з наступних причин:

- відзначається інтуїтивно зрозумілим синтаксисом та можливістю швидко розробляти тестові сценарії;

- дозволяє легко розширювати функціональність за допомогою Python, що є дуже зручним для адаптації під наші конкретні потреби та випробувань;
- надає можливість гнучко конфігурувати параметри тестування, такі як кількість користувачів, швидкість навантаження та тривалість тесту. Це надає можливість точно налаштовувати тести під реальні умови використання.

5.4.1 Опис тестового сценарію

Для навантажувального тестування розроблено тестовий сценарій, який відображає типові дії користувачів нашого продукту. Сценарій включає в себе реалістичні дії та взаємодії з системою для максимально точного моделювання реального використання

5.4.2 Параметри тестування

Для нашого навантажувального тесту обрано параметри:

- кількість користувачів – 1000;
- розподіл користувачів – 10 нових користувачів кожні 10 секунд;
- тривалість тесту – 5 хвилин.

Ці параметри обрані на основі очікуваного обсягу використання та допомагають визначити, як система веде себе під значущим навантаженням.

5.4.3 Запуск та результати моніторингу

Під час тестування використано інструмент Locust, запустивши тести з використанням командної строки з явно вказаними параметрами `headless` та `run-time`, які вказують на кількість користувачів, розподіл користувачів (за кількістю та часом) та тривалість тестування. Результати тестування (рис. 5.2) представлені через вебінтерфейс Locust.



Рисунок 5.2 – Графік “Response Times”

5.4.4 Аналіз результатів та висновки

Після завершення навантажувального тестування при 1000 користувачах та аналізу отриманих результатів, досягнуто ключових висновків:

- результати тестування підтвердили стійкість системи під значущим навантаженням. В системі не виявлено критичних помилок, зависань чи інших проблем, які можуть призвести до втрати продуктивності;
- час відгуку системи підтвердив очікувану швидкість реакції на запити користувачів при максимальному навантаженні. Система продовжує демонструвати задовільну відзивчивість, що є ключовим аспектом для забезпечення позитивного користувацького досвіду;
- система продемонструвала готовність до роботи під великими навантаженнями, що є важливим показником її ефективності;
- враховуючи отримані результати, можна визначити області, де можливі подальші оптимізації для поліпшення ефективності та масштабельності.

ВИСНОВКИ

В даному дипломному проєкті спроектовано та розроблено вебсистему, яка включає в себе вебдодаток та телеграм-бот для організації та прискорення роботи обласної громади для видачі гуманітарної допомоги та надання людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби.

Вебсистему розроблено для того, щоб поліпшити процес видачі гуманітарної допомоги. Крім того, вебсистема ставить на меті пришвидшити проходження черг на пунктах видачі, а також дати людям можливість формувати запити на отримання допомоги в режимі онлайн, які організовують та прискорюють роботу обласної громади для видачі гуманітарної допомоги.

Виходячи з цього виділимо основні аспекти, які вносять інновації:

- розробка алгоритмів автоматичної обробки та класифікації заявок на гуманітарну допомогу для швидкого реагування на індивідуальні потреби людей;
- реалізація інтерактивного інтерфейсу для користувачів через телеграм-бота для зручності подання заявок;
- зменшення паперової роботи для робітників громади та гуманітарних штабів;
- пришвидшення черг на локаціях видачі допомоги;
- електронний облік отримувачів в системі.

Під час виконання роботи виконано: аналіз цільової аудиторії, формулювання вимог до ПЗ, специфікацію вимог до ПЗ (SRS), статистичний аналіз використання браузерів в Україні, планування проєкту в Jira, UML моделювання, проєктування структури БД та структури вебсистеми, розробку та програмну реалізацію, створення тестового плану, ручне та автоматизоване тестування вебсистеми.

Дослідження вибірки браузерів серед користувачів в Україні, проведене з використанням статистичного аналізу, розкриває цікаві та важливі тенденції у використанні інтернет-переглядачів в цій країні. Згідно із зібраними даними, найпопулярнішим браузером в Україні залишається Google Chrome, який займає

значну частку ринку. Його високу популярність можна пояснити швидкістю, надійністю та рядом функцій, які задовольняють потреби різних категорій користувачів. Додатковий інтерес викликає зростання популярності мобільних браузерів, зокрема Safari та Chrome Mobile. Це свідчить про тенденцію збільшення кількості користувачів, які переглядають інтернет-контент на мобільних пристроях. Такі результати мають важливе значення для веброзробників, оскільки вони вказують на потребу адаптації вебсайтів під мобільні пристрої.

Вебдодаток розроблено за допомогою frontend бібліотеки React.JS з використанням Redux та backend фреймворка Django з використанням MTV архітектури. Після формулюванні вимог до вебсистеми розроблено тестовий план, який докладно описує всі тестові активності та визначає деталі створення тестових артефактів. Це забезпечує систематичний та вичерпний підхід до тестування, забезпечуючи високий рівень якості вебдодатку.

Після завершення розробки було проведено як ручне, так і автоматичне тестування. Важливим аспектом автоматизації тестування стало використання інструменту Cypress, який забезпечив високу надійність та ефективність тестового покриття. Це дозволило здійснити швидке виявлення та усунення помилок, підвищуючи загальну якість вебдодатку.

Наступні дослідження, проведені з використанням технології Locust, визначили можливість вебдодатку тримати навантаження у 1000 користувачів одночасно. У рамках цього дослідження було залучено 1000 користувачів, що дозволило отримати вичерпні дані про ефективність системи під великим навантаженням. Результати вказують на те, що програмний продукт продемонстрував високу стабільність та швидкодію навіть при екстремальних умовах використання. Отже ці результати будуть використані у подальшому розвитку вебсистеми.

Отже, розроблена вебсистема задовільно пройшла приймальне тестування та готова до релізу на продуктивне середовище. У подальших планах проєктування та розробка наступної версії вебдодатку, яка охопить ще більше цифровізації у роботи громад на гуманітарних штабів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Що таке «Є-Допомога»? [Електрон. ресурс]. – Режим доступу: <https://bf.diia.gov.ua/articles/shcho-take-proiekt-iedopomoha>.
2. Що таке «Поряд»? [Електрон. ресурс]. – Режим доступу: <https://www.rv.gov.ua/v-ukrayini-zapustili-portal-dopomogi-poryad-1>.
3. Top 10 best front end frameworks in 2024 [Electronic resource]. – Access mode: <https://www.imaginarycloud.com/blog/best-frontend-frameworks/>.
4. Визначення бази даних [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/База_даних.
5. Система управління базами даних [Електрон. ресурс]. – Режим доступу: <https://portalcripto.com.br/uk/dicionario/o-que-e-um-sgbd/>.
6. MySQL [Електрон. ресурс]. – Режим доступу: <https://promoter.net.ua/articles/baza-danix-mysql.html>.
7. Apache [Electronic resource]. – Access mode: <https://www.websiterating.com/uk/web-hosting/glossary/what-is-apache-server/>.
8. Javascript [Electronic resource]. – Access mode: <https://astwellsoft.com/uk/blog/tehnology/javascript.html>.
9. Що таке мова програмування Python? [Електрон. ресурс]. – Режим доступу: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovaniya-python/>.
10. Що таке ReactJS? [Електрон. ресурс]. – Режим доступу: <https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer>.
11. Django [Електрон. ресурс]. – Режим доступу: <https://tutorial.djangogirls.org/uk/django/>.
12. Telegram bot [Електрон. ресурс]. – Режим доступу: <https://marchenko.marketing/scho-take-chat-bot-ta-yak-stvoriti-bota-v-telegram/>.
13. Ресурс з надання статистичних даних – Statcounter [Electronic resource]. – Access mode: <https://gs.statcounter.com/>.

14. Unified Modeling Language [Electronic resource]. – Access mode: <https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html>.
15. UML Діаграма прецедентів [Електрон. ресурс]. – Режим доступу: https://www.wikiwand.com/uk/Діаграма_прецедентів.
16. UML Діаграма послідовності [Електрон. ресурс]. – Режим доступу: <http://www.tsatu.edu.ua/kn/wp-content/uploads/sites/16/laboratorna-robota-9-diahrama-poslidovnosti.pdf>.

ДОДАТОК А
Технічне завдання

A.1 Призначення специфікація вимог до програмного забезпечення

Специфікація вимог до програмного забезпечення (SRS) — це документ, що детально описує функціональність та робочі принципи програмного забезпечення. Вона також визначає, як ПЗ повинно функціонувати та виконувати свої завдання. SRS містить важливу інформацію для всіх етапів проєкту, зокрема для розробки, забезпечення якості, експлуатації та технічного обслуговування, а також визначає функціональні вимоги, що необхідні для врахування потреб бізнесу та користувачів.

Специфікація вимог до ПЗ охоплює функціональні та нефункціональні вимоги до вебсистеми. Цей документ призначений для проєктної команди та замовника.

A.2 Область дії

«HumanAidConnect» - це вебсистема, яка включає в себе вебдодаток та Telegram Bot, що мають поліпшити процес видачі гуманітарної допомоги. Головна мета проєкту - надати людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби. Крім того, вебсистема ставить на меті пришвидшити проходження черг на пунктах видачі, а також дати людям можливість формувати запити на отримання допомоги в режимі онлайн, які організовують та прискорюють роботу обласної громади для видачі гуманітарної допомоги.

A.3 Визначення, акроніми та скорочення

Локація — місце з видачі гуманітарної допомоги. Конкретна адреса за якою відбувається видача гуманітарної допомоги Одержувачам.

Округ — територіальна одиниця, що містить в собі локації з видачі гуманітарної допомоги.

Період обробки запитів – період, коли локація припиняє прийом запитів Одержувачів на отримання гуманітарної допомоги. Тривалість періоду визначається типом Одержувача, що намагається створити запит. Для ВПО цей період становить 2 доби, для СЖО - один тиждень.

Стандартний пакет гуманітарної допомоги – одиниця гуманітарної допомоги, яку обов'язково отримують усі типи Одержувачів. Вміст регулюється обласною громадою або штабом.

А.4 Загальний опис інтерфейсу користувача

Вимоги до інтерфейсу “Отримувача” (Telegram Bot):

- можливість зайти у чат з телеграм-ботом за посиланням або відсканувавши QR-код;
- можливість зайти в основний розділ телеграм-бота;
- можливість залишити відгук;
- створення запиту на допомогу;
- можливість обирати округ для видачі допомоги.

Вимоги до інтерфейсу “Адміністратора” (Вебдодаток):

- можливість авторизуватися у вебдодатку;
- можливість вийти з профілю;
- створення/редагування/видалення акаунтів робітників складу та видачі;
- створення/редагування/видалення локацій та округів;
- створення та видалення отримувачів;
- можливість обробки запитів на видачі;
- можливість обробки запитів на складі;
- можливість перегляду сторінки “Зворотнього зв’язку”;
- перегляд відгуків.

Вимоги до інтерфейсу “Робітник складу” (Вебдодаток):

- можливість авторизуватися у вебдодатку;

- можливість вийти з профілю;
- можливість обробки запитів на складі.

Вимоги до інтерфейсу “Робітник на видачі” (Вебдодаток):

- можливість авторизуватися у вебдодатку;
- можливість вийти з профілю;
- можливість обробки запитів на видачі;
- створення нових отримувачів.

Системний адміністратор має доступ до адмін панелі яка містить у собі всі налаштування вебдодатку. За допомогою адмін панелі, системний адміністратор може коригувати дані користувачів, локацій, та округів. Він повинен мати елементарні навички DevOps для користування адмін-панеллю вебдодатку, а також для управління БД та сервером.

A.5 Функції виробу

Основними функціями вебсистеми є:

- авторизація користувачів;
- керування командою (робітники складу та видачі);
- менеджмент округів та локацій;
- обробка запитів на допомогу;
- отримання та видача допомоги та оформлення наступної допомоги;
- створення нових отримувачів;
- менеджмент відгуків.

A.6 Характеристики користувача

В процесі розробки вимог виявлено користувачів вебдодатку (табл. A.1) та телеграм-бота (табл. A.2).

Таблиця А.1 – Користувачі вебдодатку

Користувач	Характеристика
Адміністратор	Має можливість: створювати, редагувати та видаляти акаунти волонтерів; створювати, редагувати, видаляти Округи; створювати, редагувати, видаляти Локації; створювати та редагувати розклад проведення видачі гуманітарної допомоги на Локаціях.
Робітник складу	Має можливість: переглядати списки запитів по обраній локації; переглядати коментарі до запитів; опрацювати запит; додавати свій коментар до запиту
Робітник на видачі	Має можливість: реєструвати у системі нових Отримувачів; забирати опрацьовані Робітником складу запити зі складу; проводити видачу гуманітарної допомоги на Локаціях
Системний адміністратор	Людина, що займається підтримкою вебдодатку

Таблиця А.2 – Користувачі телеграм-бота

Користувач	Характеристика
Отримувач	Має можливість: зареєструватися в телеграм-боті; залишити відгук про роботу системи; створити запит на отримання гуманітарної допомоги

Вебдодаток має інтерфейс, який є легким та зрозумілим для будь-якого користувача, незалежно від віку та рівня освіти. Функціонал додатку відповідає потребам кожного з робітників громади. У разі виникнення труднощів з розумінням роботи вебдодатку користувач може звертатися за допомогою до системного адміністратора. Телеграм-бот також має зрозумілий інтерфейс та покрокову інструкцію.

А.7 Умови експлуатації

Для коректної роботи програми необхідно встановити та налаштувати вебсервер Apache, а також систему управління базами даних MySQL.

A.8 Вимоги до програмної документації

Елементи базової програмної документації охоплюють технічне завдання, текст програмного коду, а також можливі додатки до пояснювальної записки. Технічне завдання визначає вимоги та функціональні можливості програми. Сам текст програми є ключовим компонентом документації, що надає вичерпну інформацію про логіку та реалізацію програмних модулів. Додатки до пояснювальної записки можуть включати діаграми, схеми, або інші матеріали, які допомагають в розумінні та використанні програмного продукту.

ДОДАТОК Б

Діаграма прецедентів для актора «Системний адміністратор»

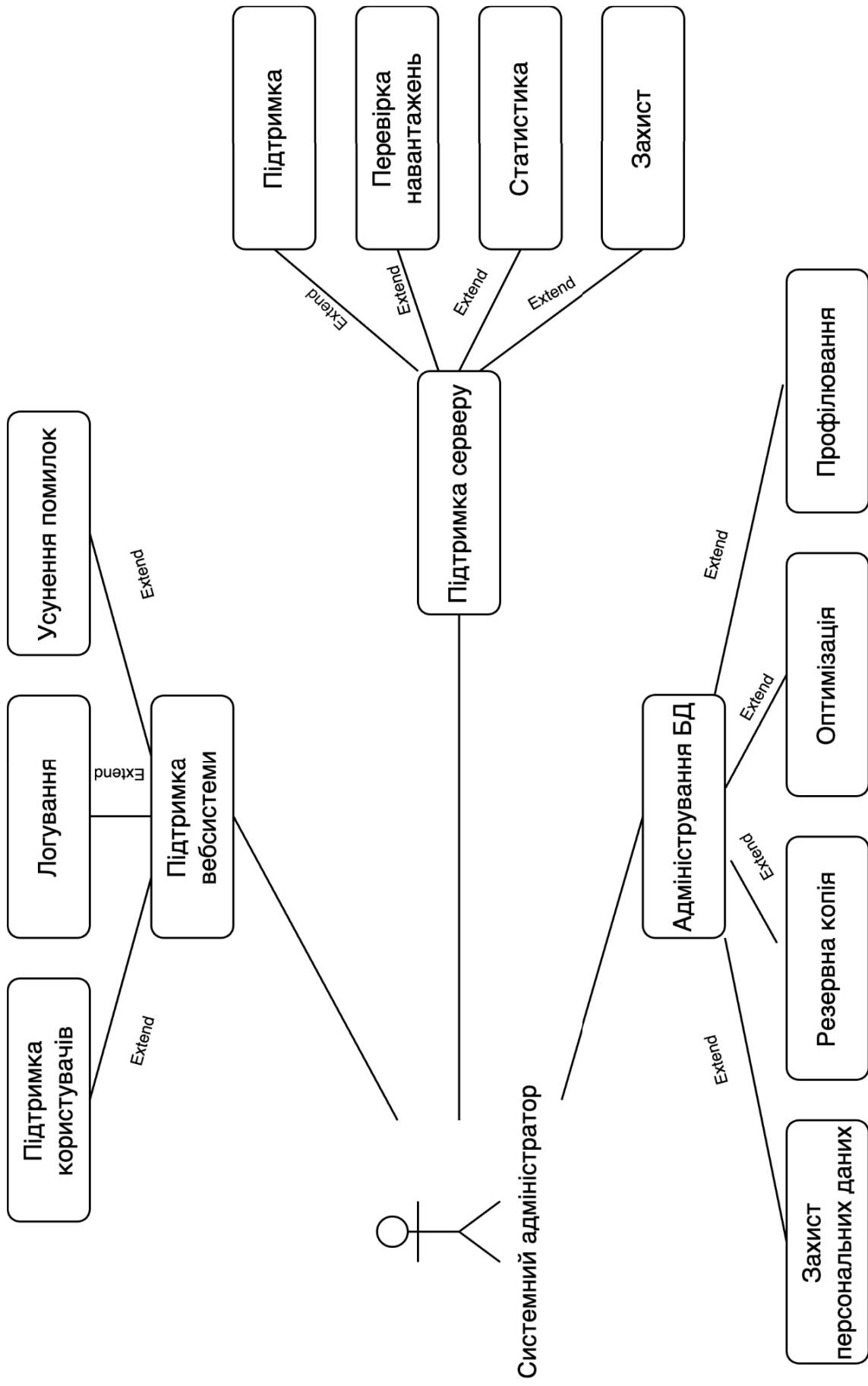


Рисунок Б.1 – Актор “Системний адміністратор” (діаграма прецедентів)

ДОДАТОК В
Структура вебдодатку для «Адміністратора»

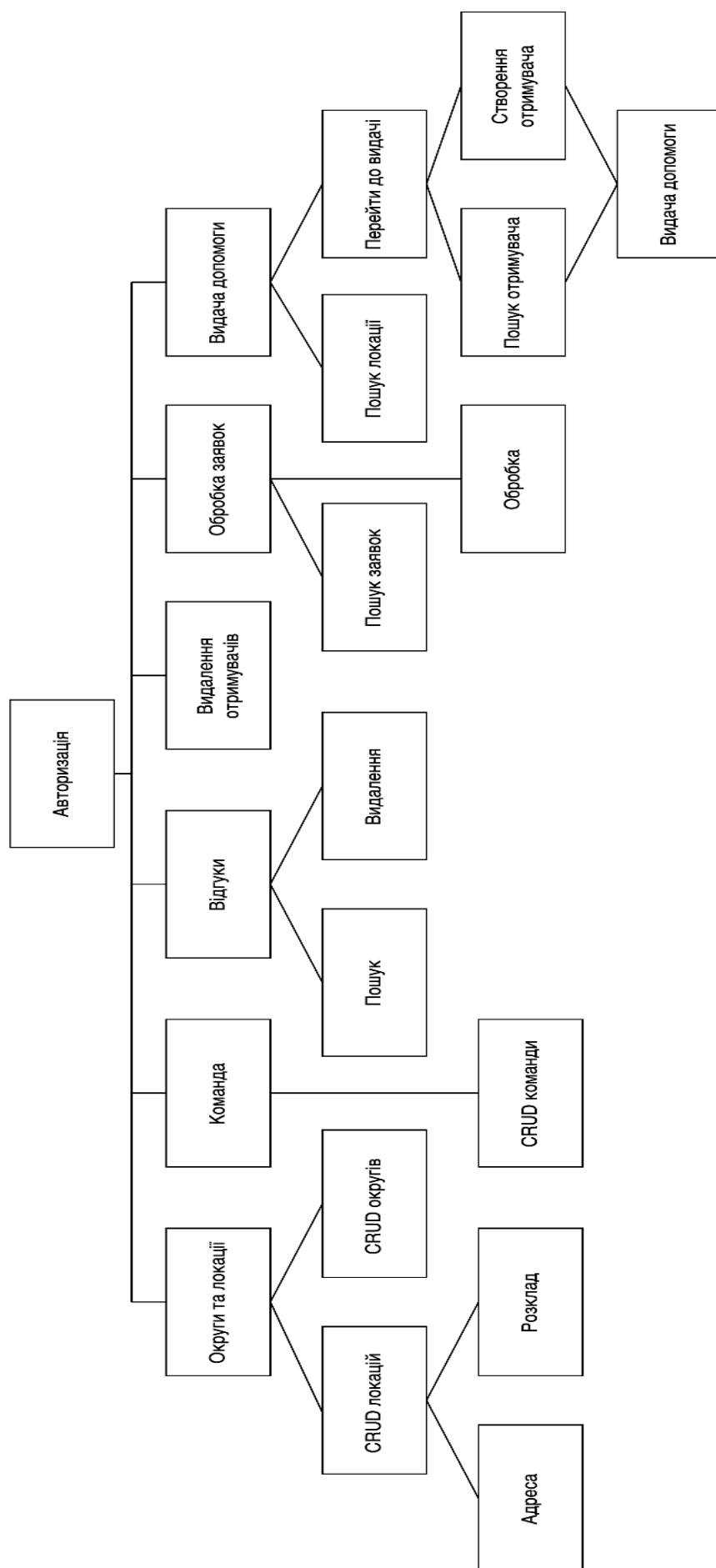


Рисунок В.1 – Структура вебдодатку для «Адміністратора»

ДОДАТОК Г

Алгоритм роботи робітника на видачі

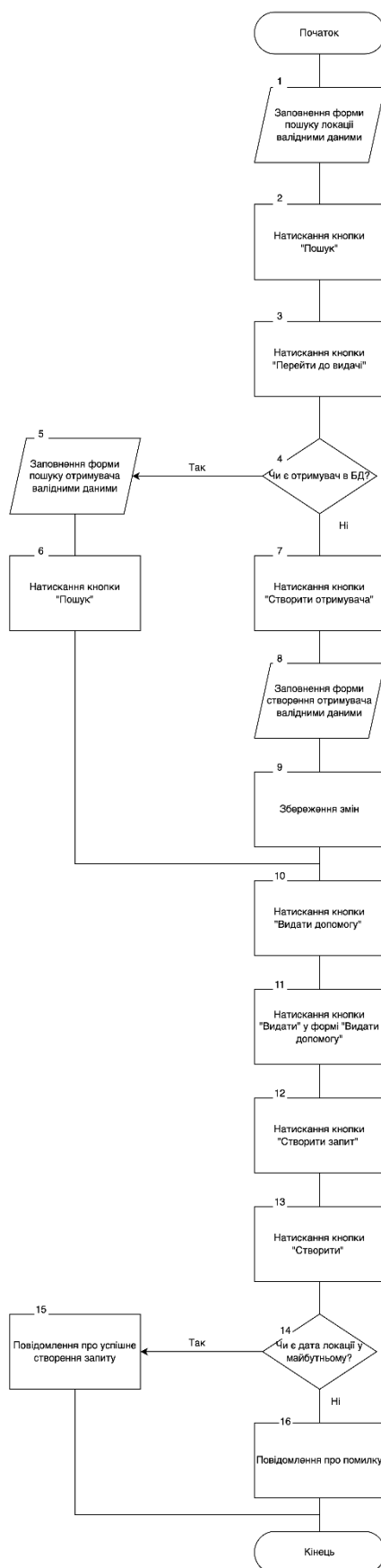


Рисунок Г.1 – Алгоритм роботи робітника на видачі

ДОДАТОК Д
Текст програми

Д.1 Фрагмент коду з файлу models.py

```
from django.db import models
class District(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
```

Д.2 Фрагмент коду з файлу templates.py

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ district.title }}</title>
</head>
<body>
    <h1>{{ district.title }}</h1>
    <p>{{ district.description }}</p>
</body>
</html>
```

Д.3 Фрагмент коду з файлу views.py

```
from django.shortcuts import render
from .models import District

def district_detail(request, district_id):
    district = District.objects.get(pk=district_id)
    return render(request, 'district / district
_detail.html', {'district ': district })
```

Д.4 Фрагмент коду з файлу models.py

```
zp_districts
District.objects.filter(district__gt='zp') =
```

Д.5 Фрагмент коду з файлу models.py

```
class DistrictManager(models.Manager):
    def get_zp_districts(self):
        return self.filter(district__gt='zp')

class Article(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    objects = DistrictManager()
```

Д.6 Фрагмент коду з файлу AuthForm.js

```
import React, { useState } from 'react';
import axios from 'axios';

function AuthForm({ onLogin }) {
    const [credentials, setCredentials] = useState({
        email: '',
        password: '',
    });

    const handleLogin = async (e) => {
        e.preventDefault();
        try {
            const response = await
            axios.post('/api/auth/login', credentials);
            console.log('Успішна авторизація',
            response.data);

            localStorage.setItem('token',
            response.data.token);

            onLogin();
        } catch (error) {
            console.error('Помилка при авторизації', error);
        }
    };

    return (
        <form onSubmit={handleLogin}>
            <label>
```

```

        Електронна пошта:
        <input
          type="email"
          value={credentials.email}
          onChange={(e) => setCredentials({
...credentials, email: e.target.value })}
        />
      </label>
      <label>
        Пароль:
        <input
          type="password"
          value={credentials.password}
          onChange={(e) => setCredentials({
...credentials, password: e.target.value })}
        />
      </label>
      <button type="submit">Увійти</button>
    </form>
  );
}

export default AuthForm;

```

Д.7 Фрагмент коду з файлу LocationForm.js

```

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

function LocationForm() {
  const [locationData, setLocationData] = useState({
    name: '',
    city: '',
    address: '',
    schedule: '',
  });

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await
    axios.post('/api/locations', locationData);

```

```

        console.log('Локація успішно створена',
response.data);
        setLocationData({ name: '', city: '', address:
'', schedule: '' });
    } catch (error) {
        console.error('Помилка при створенні локації',
error);
    }
};

return (
    <form onSubmit={handleSubmit}>
        <label>
            Назва:
            <input
                type="text"
                value={locationData.name}
                onChange={(e) => setLocationData({
...locationData, name: e.target.value })}
            />
        </label>
        <label>
            Місто:
            <input
                type="text"
                value={locationData.city}
                onChange={(e) => setLocationData({
...locationData, city: e.target.value })}
            />
        </label>
        <label>
            Адреса:
            <input
                type="text"
                value={locationData.address}
                onChange={(e) => setLocationData({
...locationData, address: e.target.value })}
            />
        </label>
        <label>
            Розклад:
            <input
                type="text"
                value={locationData.schedule}

```

```

        onChange={(e) => setLocationData({
...locationData, schedule: e.target.value })}
      />
    </label>
    <button type="submit">Створити локацію</button>
  </form>
);
}

export default LocationForm;

```

Д.8 Фрагмент коду з файлу PackageFilterForm.js

```

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

function PackageFilterForm({ onFilter }) {
  const [filterData, setFilterData] = useState({
    date: '',
    status: 'pending',
  });

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await
axios.get('/api/packages', { params: filterData });
      console.log('Пакети, відфільтровані за датою та
статусом', response.data);
      onFilter(response.data);
    } catch (error) {
      console.error('Помилка при фільтрації пакетів',
error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Дата:
        <input
          type="date"
          value={filterData.date}

```

```

        onChange={(e) => setFilterData({
...filterData, date: e.target.value })}
      />
    </label>
    <label>
      Статус:
      <select
        value={filterData.status}
        onChange={(e) => setFilterData({
...filterData, status: e.target.value })}
      >
        <option value="pending">Очікується</option>
        <option value="shipped">Відправлено</option>
        <option
value="delivered">Доставлено</option>
      </select>
    </label>
    <button
                                type="submit">Застосувати
фільтр</button>
  </form>
);
}

export default PackageFilterForm;

```

Д.9 Фрагмент коду з файлу PackageList.js

```

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

function PackageList({ packages }) {
  const [updatedPackages, setUpdatedPackages] =
useState([]);

  const handleStatusUpdate = async (packageId,
newStatus) => {
    try {
      const response = await
axios.patch(`/api/packages/${packageId}`, { status:
newStatus });
      console.log(`Статус пакета ${packageId}
оновлено`, response.data);
    }
  }
}

```

```

        setUpdatedPackages([...updatedPackages,
response.data]);
    } catch (error) {
        console.error(`Помилка при оновленні статусу
пакета ${packageId}`, error);
    }
};
return (
    <div>
        <h2>Гуманітарні пакети</h2>
        <ul>
            {packages.map((package) => (
                <li key={package.id}>
                    <div>{`ID: ${package.id}`}</div>
                    <div>{`Статус: ${package.status}`}</div>
                    <button onClick={() =>
handleStatusUpdate(package.id, 'shipped')}>
                        Відправити
                    </button>
                    <button onClick={() =>
handleStatusUpdate(package.id, 'delivered')}>
                        Доставлено
                    </button>
                </li>
            ))}
        </ul>
        <h3>Змінені статуси</h3>
        <ul>
            {updatedPackages.map((updatedPackage) => (
                <li key={updatedPackage.id}>
                    <div>{`ID: ${updatedPackage.id}`}</div>
                    <div>{`Новий статус:
${updatedPackage.status}`}</div>
                </li>
            ))}
        </ul>
    </div>
);
}
export default PackageList;

```

Д.10 Фрагмент коду з файлу RecipientListPage.js

```

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

function RecipientListPage() {
  const [recipients, setRecipients] = useState([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [filteredRecipients, setFilteredRecipients] =
    useState([]);

  useEffect(() => {
    const fetchRecipients = async () => {
      try {
        const response = await
        axios.get('/api/recipients');
        setRecipients(response.data);
        setFilteredRecipients(response.data);
      } catch (error) {
        console.error('Помилка при отриманні списку
отримувачів', error);
      }
    };

    fetchRecipients();
  }, []);

  const handleSearch = () => {
    const filtered = recipients.filter(
      (recipient) =>
        recipient.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
        recipient.city.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setFilteredRecipients(filtered);
  };

  const handleProvideHelp = (recipientId) => {
    console.log(`Видано допомогу отримувачеві з ID
    ${recipientId}`);
  };

```

```

const handleCreateRequest = () => {
  console.log('Створено новий запит на отримання
допомоги');
};

return (
  <div>
    <h2>Отримувачі допомоги</h2>
    <div>
      <label>
        Пошук:
        <input
          type="text"
          value={searchTerm}
          onChange={(e) =>
setSearchTerm(e.target.value)}
        />
      </label>
      <button onClick={handleSearch}>Пошук</button>
    </div>
    <ul>
      {filteredRecipients.map((recipient) => (
        <li key={recipient.id}>
          <div>{`Ім'я: ${recipient.name}`}</div>
          <div>{`Місто: ${recipient.city}`}</div>
          <button onClick={() =>
handleProvideHelp(recipient.id)}>
            Видати допомогу
          </button>
        </li>
      ))}
    </ul>
    <div>
      <button onClick={handleCreateRequest}>Створити
запит</button>
    </div>
  </div>
);
}

export default RecipientListPage;

```

Д.11 Фрагмент коду з файлу models.py

```

from django.db import models

class Location(models.Model):
    title = models.CharField(max_length=32)
    description = models.CharField(max_length=255)
    schedule = models.DateTimeField()

class Recipient(models.Model):
    first_name = models.CharField(max_length=64)
    phone = models.CharField(max_length=255)
    date_of_birth = models.DateTimeField()
    city = models.CharField(max_length=255)
    gender = models.CharField(max_length=255)

class RecipientType(models.Model):
    recipient = models.ForeignKey(Recipient,
on_delete=models.CASCADE)
    location = models.ForeignKey(Location,
on_delete=models.CASCADE)
    title = models.CharField(max_length=64)

```

Д.12 Фрагмент коду з файлу views.py

```

from django.shortcuts import get_object_or_404
from rest_framework import viewsets
from rest_framework.response import Response

from .models import Location, Recipient, RecipientType
from .serializers import LocationSerializer,
RecipientSerializer, RecipientTypeSerializer

class LocationViewSet(viewsets.ModelViewSet):
    queryset = Location.objects.all()
    serializer_class = LocationSerializer

class RecipientViewSet(viewsets.ModelViewSet):
    queryset = Recipient.objects.all()
    serializer_class = RecipientSerializer

```

```
class RecipientType(viewsets.ModelViewSet):
    queryset = HumanitarianPackage.objects.all()
    serializer_class = RecipientTypeSerializer
```

Д.13 Фрагмент коду з файлу serializers.py

```
from rest_framework import serializers
from .models import Location, Recipient, RecipientType
class LocationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Location
        fields = '__all__'
class
RecipientSerializer(serializers.ModelSerializer):
    class Meta:
        model = Recipient
        fields = '__all__'
class
RecipientTypeSerializer(serializers.ModelSerializer):
    class Meta:
        model = RecipientType
        fields = '__all__'
```

Д.14 Фрагмент коду з файлу url.py

```
from django.urls import path, include
from rest_framework import routers
from .views import LocationViewSet, RecipientViewSet,
RecipientTypeViewSet

router = routers.DefaultRouter()
router.register(r'locations', LocationViewSet)
router.register(r'recipients', RecipientViewSet)
router.register(r'types', RecipientTypeViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
]
```

Д.15 Фрагмент коду з файлу views.py

```

from rest_framework.decorators import action

class
HumanitarianPackageViewSet(viewsets.ModelViewSet):

    @action(detail=True, methods=['post'])
    def provide_help(self, request, pk=None):
        package = self.get_object()
        package.status = 'shipped'
        package.save()
        return Response({'status': 'Help provided'})

    @action(detail=True, methods=['post'])
    def create_request(self, request, pk=None):
        return Response({'status': 'Request created'})

```

Д.16 Фрагмент коду з файлу admin.py

```

from django.contrib import admin
from .models import Location, Recipient,
HumanitarianPackage

admin.site.register(Location)
admin.site.register(Recipient)
admin.site.register(RecipientType)
admin.site.register(HumanitarianPackage)

```

Д.17 Фрагмент коду з файлу bot.py

```

from telegram import Update
from telegram.ext import Updater, CommandHandler,
MessageHandler, Filters, CallbackContext,
ConversationHandler
from telegram.ext import CallbackQueryHandler
from telegram import InlineKeyboardMarkup,
InlineKeyboardButton
import requests

TOKEN = ''

```

```

BASE_URL = 'http://localhost:8000/api/'
PHONE, CODE = range(2)

def start(update: Update, context: CallbackContext) ->
int:
    keyboard = [[InlineKeyboardButton("Відправити
номер телефону", callback_data='send_phone')]]
    reply_markup = InlineKeyboardMarkup(keyboard)

    update.message.reply_text("Будь ласка, відправте
свій номер телефону.", reply_markup=reply_markup)
    return PHONE

def get_phone(update: Update, context:
CallbackContext) -> int:
    user = update.message.from_user
    user_id = user.id
    user_phone = update.message.contact.phone_number

    payload = {'user_id': user_id, 'phone': user_phone}
    response = requests.post(BASE_URL + 'auth/phone/',
data=payload)

    update.message.reply_text(f'Дякую,
{user.first_name}! Ваш номер телефону успішно відправлено.')

    return ConversationHandler.END

def main() -> None:
    updater = Updater(TOKEN)
    dp = updater.dispatcher

    conv_handler = ConversationHandler(
        entry_points=[CommandHandler('start', start)],
        states={
            PHONE: [CallbackQueryHandler(get_phone,
pattern='send_phone')],
        },
        fallbacks=[],
    )
    dp.add_handler(conv_handler)
    dp.add_handler(MessageHandler(Filters.text &
~Filters.command, write_review))

```

```

        dp.add_handler(MessageHandler(Filters.command      &
Filters.regex('^(Запит      на      гуманітарну      допомогу)$'),
humanitarian_request))

        updater.start_polling()
        updater.idle()
if __name__ == '__main__':
    main()

```

Д.18 Фрагмент коду з файлу loginPage.js

```

class LoginPage {
  visit() {
    cy.visit('/login');
  }
  fillEmail(email) {
    cy.get('#username').type(email);
  }
  fillPassword(password) {
    cy.get('#password').type(password);
  }
  clickLoginButton() {
    cy.get('#loginBtn').click();
  }
}
export default new LoginPage();

```

Д.19 Фрагмент коду з файлу login.spec.js

```

describe('Login Tests', () => {
  beforeEach(() => {
    LoginPage.visit();
  });

  it('should log in successfully with valid
credentials', () => {
    LoginPage.fillEmail('validEmail');
    LoginPage.fillPassword('validPassword');
    LoginPage.clickLoginButton();
    cy.url().should('include', '/dashboard');
  });
});

```

```
    it('should show an error with invalid credentials',  
    () => {  
        LoginPage.fillEmail ('invalidEmail');  
        LoginPage.fillPassword('invalidPassword');  
        LoginPage.clickLoginButton();  
        cy.get('.error-message').should('be.visible');  
    });  
});
```

ДОДАТОК Е
Слайди презентації



Національний університет «Запорізька політехніка»
кафедра програмних засобів

Дипломна кваліфікаційна робота магістра

Розробка вебсистеми для надання гуманітарної допомоги

Виконав: студент групи КНТ-222м ЖИГУЛІН Дмитро Ігорович

Керівник: к.т.н. ГОЛУБ Тетяна Василівна

Рисунок Е.1 – Слайд №1



Об'єкт дослідження - гуманітарна допомога для ВПО та СЖО в Україні.

Предмет дослідження - вебсистема для надання гуманітарної допомоги.

Мета роботи - розробка вебсистеми, яка організовує та прискорює роботу обласної громади для видачі гуманітарної допомоги та надає людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби.

2/16

Рисунок Е.2 – Слайд №2

Завдання роботи

- проаналізувати предметну область та дослідити цільову аудиторію вебсистеми;
- проаналізувати існуючі аналоги та створити порівняльну таблицю;
- сформулювати вимоги до програмного забезпечення та розробити технічне завдання;
- проаналізувати та дослідити технології для розробки вебсистеми;
- спланувати проєкт та провести моделювання вебсистеми;
- спроектувати базу даних та структуру вебсистеми;
- розробити архітектуру та програмну реалізацію вебсистеми;
- створити тест-план та провести ручне та автоматизоване тестування;
- дослідити стійкість вебдодатку через навантажувальне тестування.

3/16

Рисунок Е.3 – Слайд №3

Дослідження існуючих аналогів

Характеристика	«Є-Допомога»	«Поряд»	Вебсистема
Допомога за розкладом	Ні, тільки за можливістю волонтерів	Ні, тільки за можливістю волонтерів	Регулярно за розкладом
Чітко визначене місце отримання допомоги	Ні	Ні	Локація визначається заздалегідь
Створення нових отримувачів	Тільки самостійно через вебсайт	Тільки самостійно через вебсайт	На локації, Робітником на видачі
Спосіб взаємодії для отримувача	Вебсайт	Вебсайт	Телеграм-бот
Чітко визначені правила видачі	Ні, тільки волонтерські можливості	Ні, тільки волонтерські можливості	ВПО раз на тиждень, СЖО – раз на місяць
Запити за передзамовленням	Так	Так	Так

4/16

Рисунок Е.4 – Слайд №4

Розробка вимог

Вебсистема має забезпечувати:

- авторизацію та реєстрацію користувачів;
- створення профілів отримувачів;
- створення, фільтрацію та пошук локацій та отримувачів;
- авторизацію та взаємодію отримувачів через телеграм-бот;
- обробку запитів отримувачів;
- обробку гуманітарної допомоги робітником складу;
- надання гуманітарної допомоги робітником на видачі;
- ведення розкладу видачі для кожного локації та отримувачів.

Виявлено 5 ролей користувачів у вебсистемі:

- Отримувач (Телеграм-бот);
- Адміністратор (вебдодаток);
- Робітник складу (вебдодаток);
- Робітник на видачі (вебдодаток);
- Системний адміністратор (вебсистема);

5/16

Рисунок Е.5 – Слайд №5

Порівняльна таблиця засобів для frontend розробки

Характеристика	React.JS	Angular	Vue.JS
Тип	Бібліотека	Фреймворк	Бібліотека
Мова	JavaScript, JSX	TypeScript, JavaScript	JavaScript
Архітектура	Компонентна	Модульна, Компонентна	Компонентна
Віртуальний DOM	Так	Ні	Так
Інтеграція з іншими технологіями	Широкомасштабна	Інтегрована в екосистему Google	Широкомасштабна
Швидкість	Висока	Середня	Висока
Продуктивність	Висока	Висока	Висока

6/16

Рисунок Е.6 – Слайд №6

Вибір технологій та програмних засобів

Фінальний вибір:

- СУБД MySQL;
- вебсервер Apache;
- мова програмування Javascript;
- мова програмування Python;
- бібліотека React.JS;
- вебфреймворк Django;
- месенджер Telegram.

Висновок із порівняння СУБД:

MySQL є безпечнішою та більш швидкою в обробці запитів. А PostgreSQL підходить до підтримки складних та масивних БД

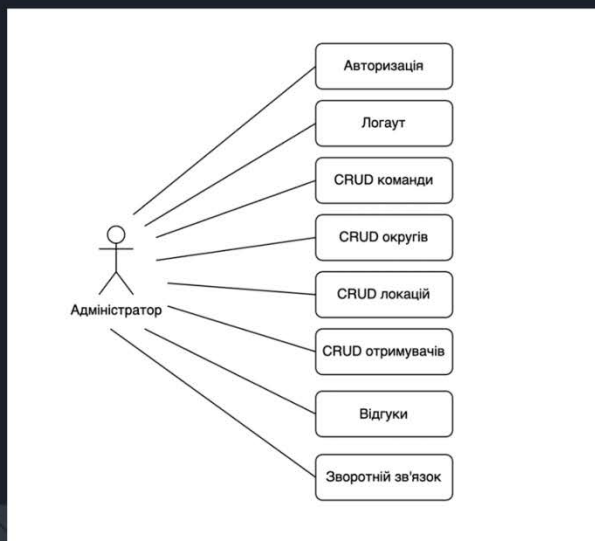
Переваги React.JS:

- висока продуктивність та швидкість;
- реюзабельні компоненти;
- широка спільнота та інструменти;
- відмінна документація.

7/16

Рисунок Е.7 – Слайд №7

UML моделювання



Актор "Адміністратор" (діаграма прецедентів)



Актор "Робітник складу" (діаграма прецедентів)

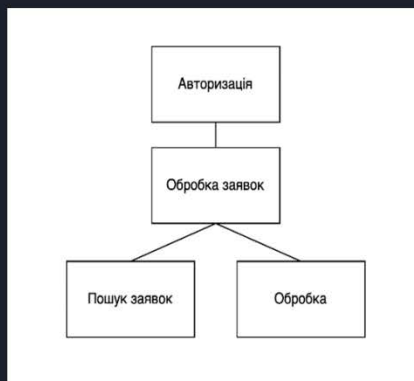
8/16

Рисунок Е.8 – Слайд №8

Структура вебсистеми



Структура для Робітника на видачі



Структура для Отримувача у телеграм-боті



Структура для Робітника складу

9/16

Рисунок Е.9 – Слайд №9

ER-діаграма бази даних

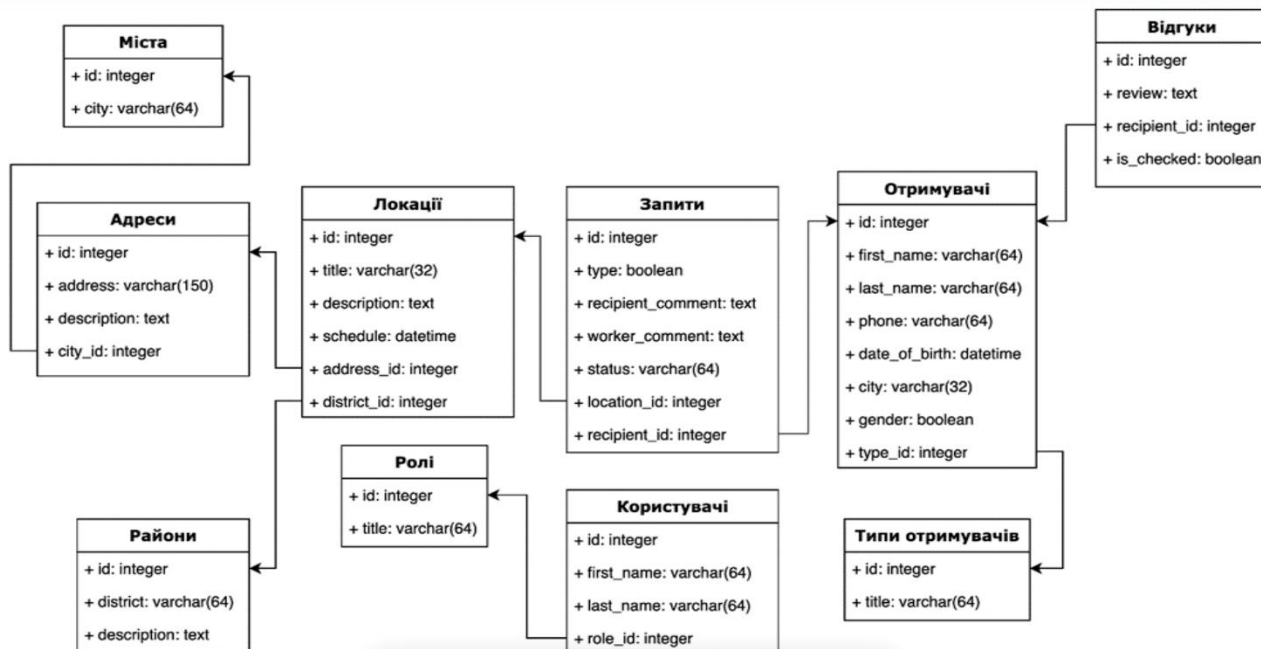
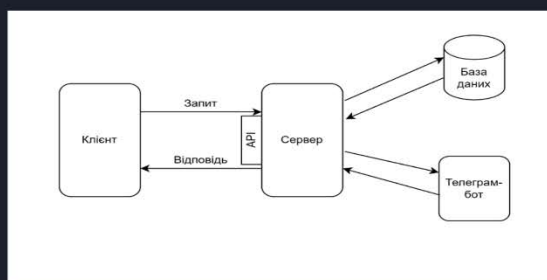
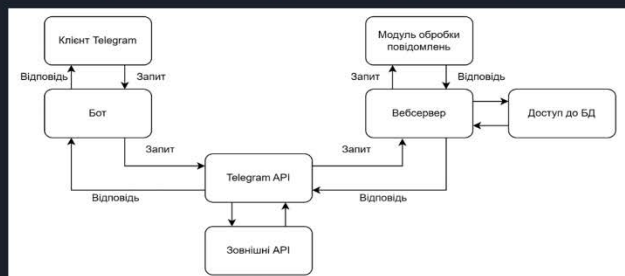


Рисунок Е.10 – Слайд №10

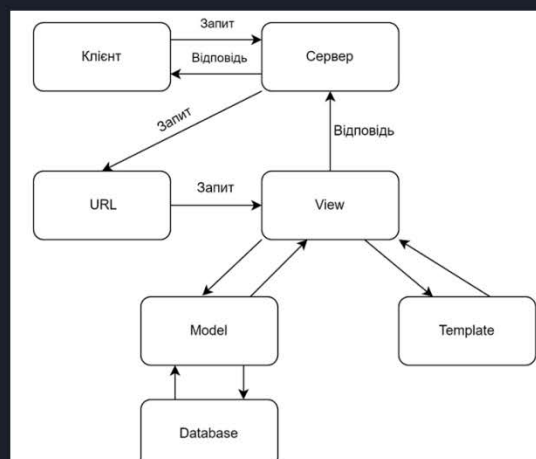
Архітектура вебсистеми



Клієнт-серверна архітектура в інтеграції з телеграм-ботом



Архітектура телеграм-бота



MTV архітектура в Django для вебдодатку

11/16

Рисунок Е.11 – Слайд №11

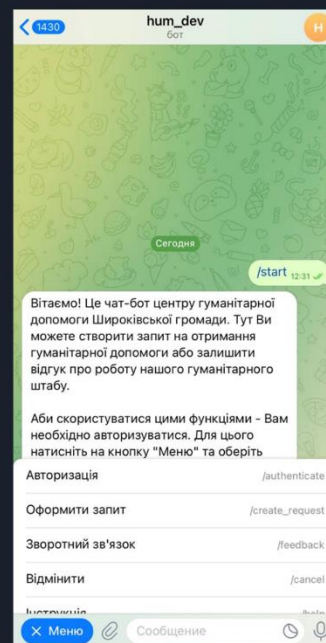
Авторизація у вебдодатку та телеграм-бот

Електронна пошта*

Пароль*

[Забули пароль?](#)

[Увійти](#)



12/16

Рисунок Е.12 – Слайд №12

Створення локації та обробка запитів на складі та на видачі

The figure consists of four screenshots of a web application interface:

- Top Left:** A form titled "Локація: test" for creating a location. It includes fields for "Назва" (Name) with value "test", "Область" (Region) with a dropdown showing "Запорізька обл.", "Населений пункт" (Settlement) with value "с. Широке (Запорізький) (Широківська) Запорізький", and "Адреса" (Address) with value "вул. Даваткова, буд. 2". A green "Зберегти" (Save) button is at the bottom.
- Top Right:** A form titled "Локація: test" for processing requests. It includes fields for "Статус" (Status) with a dropdown showing "Очікує обробки", "Дата видачі" (Issue date) with value "21.11.2023", and "Примітка" (Note). A green "Зберегти" (Save) button is at the bottom right.
- Bottom Left:** A form titled "Розклад локації" (Location schedule). It includes a date field "Початок дії розкладу" (Start of schedule) with value "27.06.2023", a dropdown for "Повторюється кожні" (Repeats every) with value "1" and unit "тиждень", and a table of days (Пн, Вт, Ср, Чт, Пт, Сб, Нд) with checkboxes. A green "Зберегти" (Save) button is at the bottom.
- Bottom Right:** A table titled "Локація: test" showing a list of requests. The table has columns for "ПІБ" (Full name), "Особистий номер телефону" (Personal phone number), and "Ді" (Action). The table contains three rows of data, each with a green "Створити заявку" (Create request) button.

Рисунок Е.13 – Слайд №13

Тестування вебсистеми

Всі функції визначені у вимогах до ПЗ повинні пройти тестування:

- авторизація користувачів;
- керування командой (робітники складу та видачі);
- менеджмент округів та локацій;
- обробка запитів на допомогу;
- отримання та видача допомоги та оформлення наступної допомоги;
- створення нових отримувачів;
- менеджмент відгуків.

Тестування проводилось на:

- Google Chrome (Desktop);
- Opera (Desktop);
- Firefox (Desktop);
- Google Chrome (Tablet);
- Safari (Tablet).

Інструментальні засоби:

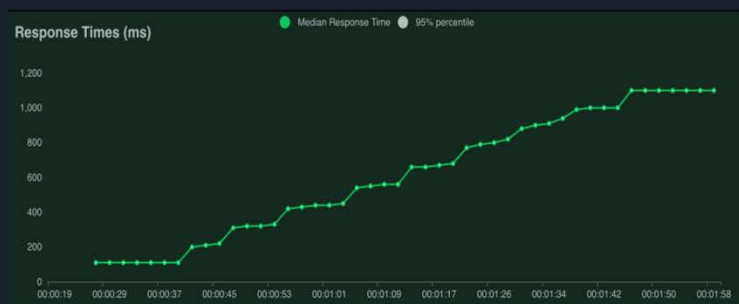
- Postman;
- DevTools;
- OWASP ZAP;
- Cypress;
- Sentry.

Рисунок Е.14 – Слайд №14

Дослідження стійкості вебдодатку

Для навантажувального тесту обрано параметри:

- кількість користувачів – 1000;
- розподіл користувачів – 10 нових користувачів кожні 10 секунд;
- тривалість тесту – 5 хвилин.



Графік "Response Times"

Результати тестування підтвердили стійкість системи під значущим навантаженням. В системі не виявлено критичних помилок, зависань чи інших проблем, які можуть призвести до втрати продуктивності. Час відгуку системи підтвердив очікувану швидкість реакції на запити користувачів при максимальному навантаженні.

15/16

Рисунок Е.15 – Слайд №15

Висновки

Розроблено вебсистему, яка включає в себе вебдодаток та телеграм-бот для організації та прискорення роботи обласної громади для видачі гуманітарної допомоги та надання людям можливість отримувати гуманітарну допомогу, яка буде враховувати їх індивідуальні потреби. Вебсистему розроблено для того, щоб поліпшити процес видачі гуманітарної допомоги. Крім того, вебсистема ставить на меті пришвидшити проходження черг на пунктах видачі, а також дати людям можливість формувати запити на отримання допомоги в режимі онлайн, які організовують та прискорюють роботу обласної громади для видачі гуманітарної допомоги. Виходячи з цього виділимо основні аспекти, які вносять інновації:

- розробка алгоритмів автоматичної обробки та класифікації заявок на гуманітарну допомогу для швидкого реагування на індивідуальні потреби людей;
- реалізація інтерактивного інтерфейсу для користувачів через телеграм-бота для зручності подання заявок;
- зменшення паперової роботи для робітників громади та гуманітарних штабів;
- пришвидшення черг на локаціях видачі допомоги;
- електронний облік отримувачів в системі.

Отже, розроблена вебсистема задовільно пройшла приймальне тестування та готова до релізу на продуктивне середовище.

16/16

Рисунок Е.16 – Слайд №16