**Міністерство освіти і науки України**

**черкаський державний технологічний університет**

Факультет інформаційних технологій і систем

Кафедра програмного забезпечення автоматизованих систем

Пояснювальна записка

до кваліфікаційної випускної бакалаврської роботи

з напрямку підготовки 121 “Інженерія програмного забезпечення”

(освітньо-кваліфікаційний рівень)

на тему: «СИСТЕМА ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ ПРО РІЗНІ СИТУАЦІЇ В МІСТІ»

Виконав: студент 4 курсу, групи ПЗС-1944

Напряму підготовки

121 “Інженерія програмного забезпечення”

(шифр і назва напряму підготовки)

Студент Білозор Д.О. .

(прізвище та ініціали)

Керівник Метелап В.В. .

(прізвище та ініціали)

Рецензент Образчиков М.А. .

(прізвище та ініціали)

Черкаси, 2022

Перелік скорочень та перелік умовних позначень

|  |  |
| --- | --- |
| Назва | Пояснення |
| ТЗ | Технічне завдання |
| БД | База даних |
| Front end | Клієнтська частина проекту, яка відображається у браузері |
| Back end | Серверна частина проекту, яка взаємодіє із даним БД та чат ботом |
| Паттерн | Набір правил та ресурсів для структуризації розробки |
| Ментор | Авторизований користувач |
| UML | Unified Modeling Language |
| HTTP | Hyper Text Transfer Protocol |
| iOS | iPhone operating system |
| API | Application Programming Interface |

**Мета:** Навчитись складати аналіз вимог до додатку, розробляти технічне завдання, проектувати з використанням діаграм UML, аналізувати архітектуру додатку, проектувати інтерфейс додатку, аналізувати структуру додатку, аналізувати та оцінювати додаток.

**Теоретичні відомості**

**1.1 Збирання та аналіз вимог**

Аналіз вимог полягає в визначенні потреб та умов які висуваються щодо нового, чи зміненого продукту, враховуючи можливо конфліктні вимоги різних замовників.

Аналіз вимог є критичним для успішної розробки проекту. Вимоги мають бути задокументованими, вимірними, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

***1.1.1 Класифікація вимог***

*Вимоги споживача* представляють собою вирази фактів та припущень які описують очікування до системи в термінах цілей, середовища, обмежень, та міри ефективності й придатності.

*Архітектурні вимоги* пояснюють що має бути зроблено ідентифікацією необхідної системної архітектури.

*Структурні вимоги* пояснюють що має бути зроблено ідентифікацією необхідної структури системи.

*Поведінкові вимоги* пояснюють що має бути зроблено ідентифікацією необхідної поведінки системи.

*Функціональні вимоги* пояснюють що має бути зроблено ідентифікацією необхідної задачі, дії, чи діяльності які мають виконуватись. Аналіз функціональних вимог буде використаний в функціях верхніх рівнів для функціонального аналізу.

*Нефункціональні вимоги* задають критерій для оцінки операцій системи, замість її поведінки.

*Вимоги продуктивності* пояснюють, до якої міри місії чи функції повинні бути виконані; зазвичай вимірюється в термінах кількості, якості, охопленні, своєчасності чи готовності. Протягом аналізу вимог, вимоги продуктивності будуть інтерактивно розроблятись вздовж всіх виявлених функції що базуються на факторах життєвого циклу системи, і характеризуються в термінах ступеня визначеності в їх оцінках, ступеня критичності успіху системи, і їх відношення до інших вимог.

*Вимоги дизайну:*

* *успадковані вимоги,* тобто вимоги які маються на увазі вимогами вищого рівня, чи перетворені з них;
* *розподілені вимоги* визначені поділом чи іншим перерозміщенням високорівневих вимог в кілька низькорівневих вимог.

**1.2 Розроблення технічного завдання**

Технічне завдання (ТЗ) – вихідний документ для розробки автоматизованої системи або створення програмного продукту, відповідно до якого проводиться виготовлення, приймання при введенні в дію та експлуатація відповідного об'єкта. ТЗ є основним документом, що визначає вимоги і порядок створення (розвитку або модернізації) інформаційної системи. ***1.2.1 Етапи складання списку вимог ТЗ***

Робота над ТЗ включає виконання низки етапів, а невизначеність, властива цій роботі, викликає проходження їх по кілька разів, ітераційно, від більш загальної постановки завдання до детального опрацювання (проектування носить ітераційний характер і те, що не враховано на початку, може бути враховано на наступних етапах).

Основні етапи розробки ТЗ:

– аналіз завдання заказника;

– конкретизація цілей проектування;

– обробка зібраної інформації:

1) узагальнення та абстрагування;

2) перевірка на суперечливість;

3) розмежування вимог на умови, обмеження та показники якості;

4) параметризація;

5) зменшення списку вимог;

6) зведення вимог та затвердження замовником.

**1.3 Проектування з використанням UML**

Робота над ТЗ включає виконання низки етапів, а невизначеність, властива цій роботі, викликає проходження їх по кілька разів, ітераційно, від більш загальної постановки завдання до детального опрацювання (проектування носить ітераційний характер і те, що не враховано на початку, може бути враховано на наступних етапах).

***1.3.1 Діаграма класів***

Діаграму класів (class diagram) використовують для подання статистичної структури моделі системи в термінології класів об’єктноорієнтованого програмування.

***1.3.2 Діаграма пакетів***

Діаграма пакетів є різновидом діаграми класів, на якій відображаються тільки пакети і залежності. Між двома пакетами існує деяка залежність, якщо існує яканебудь залежність між будь-якими двома класами в пакетах. Хоча пакети не дають відповіді на питання, як зменшити кількість залежностей у вашій системі, проте вони допомагають виділити ці залежності. Як тільки вони опиняться на виду, вам залишиться лише попрацювати над їх скороченням.

***1.3.3 Діаграма компонентів***

Діаграма компонентів описує особливості фізичної уяви системи. Діаграма компонентів відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись, тобто вона дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами.

***1.3.4 Діаграма прецедентів***

Діаграма прецедентів – діаграма, на якій зображено відношення між акторами та прецедентами в системі.

Проектована система представляється у вигляді множини сутностей чи акторів, взаємодіючих із системою за допомогою варіантів використання (use case), що служать для опису сервісів, що система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій системи при діалозі з актором.

***1.3.5 Діаграма станів***

Діаграми станів зображають всі можливі стани, в яких може знаходитися конкретний об'єкт, а також зміни стану об'єкту, які відбуваються в результаті впливу деяких подій на цей об'єкт. У більшості об'єктно-орієнтованих методів діаграми станів будуються для єдиного класу, щоб показати динаміку поведінки єдиного об'єкту.

***1.3.6 Діаграма діяльності***

Діаграма діяльності показує переходи між видами діяльності. Модель видів діяльності (activity model) може подавати в графічній формі потік подій для прецеденту. Кожен прецедент можна моделювати за допомогою одного або декілька графів видів діяльності. Подія, джерелом якої служить суб'єкт прецедент, що ініціює, це та ж сама подія, що запускає виконання графа видів діяльності. Процес виконання послідовно переходить від одного стану виду діяльності до іншого.

***1.3.7 Діаграма кооперації***

Діаграма кооперації призначена для специфікації структурних аспектів взаємодії. Головна особливість діаграми кооперації полягає в можливості графічно уявити не тільки послідовність взаємодії, але і всі структурні відносини між об'єктами, що беруть участь в цій взаємодії.

***1.3.8 Діаграма послідовності***

Діаграма послідовності показує учасників взаємодій і послідовність повідомлень, якими вони обмінюються. На діаграмі послідовності зображуються виключно ті об'єкти, що безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі.

**1.4 Аналіз архітектури веб-додатків з використанням CMS**

CMS (Content Management System – система управління вмістом) – програмний комплекс, що надає функції створення, редагування, контролю та організації структури веб-додатку (сайту), а також текстової та графічної інформації (вмісту).

*Переваги* використання CMS при розробці:

– протестований і перевірений багатьма розробниками код;

– універсальність;

– простота використання.

*Недоліки* використання CMS при розробці:

– обмеження предметної області тими об'єктами які передбачені в CMS;

– визначення основної логіки роботи програми розробниками CMS. Внесення змін в логіку проблематично;

– зайва універсальність і надлишковий функціонал.

**1.5 Проектування інтерфейсу веб-додатків**

Інтерфейс – сукупність засобів і методів взаємодії між елементами системи. Залежно від контексту, поняття застосовне як до окремого елементу (інтерфейс елементу), так і до зв'язків елементів (інтерфейс сполучення елементів).

Інтерфейс користувача – сукупність засобів, за допомогою яких користувач спілкується з різними пристроями:

- інтерфейс командного рядка – інструкції програмою або пристроєм здійснюється шляхом введення з клавіатури текстових рядків;

- графічний інтерфейс – управління програмними функціями реалізовано графічними елементами екрану.

**1.6 Аналіз структури веб-додатків з використанням CMS**

Базові структурні компоненти CMS:

1. База даних. CMS, як правило, забезпечують зберігання даних в єдиному репозиторії (сховищі).
2. Панель управління – окрема частина програми, що має розширений доступ і дозволяє спростити обслуговування ресурсу.
3. Ядро системи – основна частина CMS, яка управляє показом сторінок користувачеві, всієї навігацією, відповідно реалізує всі додаткові функції, покладені на нього.
4. Шаблонізатор – частина системи, що дозволяє розробляти і впроваджувати інтерфейс не торкаючись ядра системи.
5. Додаткові модулі – незалежні частини веб-додатку, що дозволяють розширити його функціональність на базі ядра системи.

**1.7 Розгортання веб-додатків з використанням CMS**

Веб-сервер – це сервер, який приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, і видає їм HTTP-відповіді разом із затребуваними даними: HTML-сторінкою, зображенням, файлом, медіапотоком і т.д.

На прикладі вільного веб-серверу Apache розглянемо більш детально. Ядро Apache включає в себе основні функціональні можливості, такі як обробка конфігураційних файлів, протокол HTTP і система завантаження модулів. Apache HTTP Server підтримує модульність.

У модулях реалізуються такі можливості, як:

– підтримка мов програмування;

– додавання функціоналу;

– виправлення помилок або модифікація основних функцій;

– посилення безпеки.

**LAMP** – акронім, що позначає набір (комплекс) серверного програмного забезпечення, який широко використовують в Інтернеті. LAMP названий за першими літерами продуктів, що входять до його складу:

– Linux – операційна система Linux;

– Apache – веб-сервер;

– MySQL – СУБД;

– PHP – мова програмування, що використовується для створення веб-додатків (крім PHP можуть матися на увазі інші мови, такі як Perl і Python).

Аналогічно йому існує **WAMP**:

– Windows – операційна система від компанії Microsoft;

– Apache;

– MySQL;

– PHP.

Хоча спочатку ці програмні продукти не розроблялися спеціально для роботи один з одним, однак така зв'язка стала вельми популярною через свою гнучкості, продуктивності та низької вартості.

Розгортання CMS в більшості випадків складається з таких етапів:

– створення бази даних в доступній СУБД (MySQL найчастіше);

– завантаження інсталяційного пакета на сервер;

– попередня настройка через систему веб-доступу;

– підключення до бази даних;

– встановлення.

**1.8 Розширення функціональності веб-додатків за допомогою модулів**

Більшість CMS зроблені за принципом модульності: є базова основа, до якої можливе підключення модулів. Модулі можуть вже входити до складу комплексу, можуть бути встановлені окремо, а також багато систем допускають самостійну розробку та включення додаткових модулів. Таким чином, загальний функціонал сайту залежить від функцій включених в нього модулів.

Фактично зміст тієї чи іншої сторінки залежить від модуля який реалізує функціональний рівень (бізнес-логіку).

Переваги модульних CMS наведено нижче.

*Простота сприйняття.* Чітко позначені кордони, зв'язність модулів робить їх легкими

для розуміння. Вивчення системи невеликими частинами, в кінцевому підсумку призводить до більш глибокого її розуміння.

*Змінність.* Якщо кожен модуль системи відомий тільки через інтерфейс, це дозволяє легко замінити один модуль на інший, з таким само інтерфейсом.

*Паралельна розробка.* Завдяки своїй незалежності, модулі можуть розроблятися паралельно. Для команди розробників це дає можливість розподілу завдань по межах модулів.

*Багаторазовість і гнучкість використання.* Кожен модуль, взятий з однієї системи, може бути використаний в іншій. Тут все залежить від його функціоналу і області застосування.

*Покращення тестування.* Поряд з інтеграційним і модульним тестуванням існує можливість тестування кожного модуля окремо, як єдиного цілого.

**1.9 Аналіз якості та оцінка веб-додатків**

*Тестування програмного забезпечення* (Software Testing) – це процес технічного дослідження, який виконується на вимогу замовників, і призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись.

*Якість програмного забезпечення* – характеристика програмного забезпечення, ступінь відповідності ПЗ до вимог. Частіше за все, використовують визначення ISO 9001, згідно з яким якість – це «ступінь відповідності наявних характеристик вимогам».

Фактори якості – це нефункціональні вимоги до ПЗ, що відносяться до, наприклад, надійності та продуктивності програм. Деякі з факторів якості:

* *Зрозумілість.* Призначення ПЗ повинно бути зрозумілим з самої програми та документації.
* *Повнота.* Всі необхідні частини програми повинні бути представлені та реалізовані.
* *Стислість.* Відсутність надлишкової інформації та такою, що дублюється. Реалізація принципів DRY.
* *Можливість портування.* Легкість в адаптації програми до інших умов: архітектури, платформі, операційній системі тощо.
* *Узгодженість.* Вся документація та код повинні виконуватися за єдиними угодами, використовувати єдині формати та позначення.
* *Покриття тестуванням.*
* *Зручність використання.*
* *Надійність.*
* *Безпечність.*

Модульне тестування тестує мінімальний компонент програми, або модуля. Кожний модуль тестується для перевірки правильності його реалізації.

**Інтеграційне тестування** виявляє дефекти в інтерфейсах та у взаємодії між компонентами (модулями).

**Системне тестування** тестує інтегровану систему для перевірки відповідності всім вимогам.

**Системне інтеграційне тестування** перевіряє, чи система інтегрується в будь-яку зовнішню систему (або системи) відповідно до системних вимог.

**Приймальне тестування** може проводитись кінцевим користувачем, замовником, або клієнтом для перевірки, чи може продукт бути прийнятий до використання:

– альфа-тестування – це симульоване або реальне операційне тестування потенційними користувачами/замовником або командою тестувальників на боці розробника.

– бета-тестування йде після альфа-тестування. Версії програмного забезпечення, відомі як бета-версії, надаються у користування обмеженій кількості людей поза компанією для того, щоб упевнитись, що програма не містить великої кількості помилок.

**Хід роботи**

**2.1.** Обрана тема «Система збору та обробки інформації про різні ситуації в місті».

**2.2. Структурно логічна схема**

Структурно логічна схема має такий вигляд:

1. Створення заявки на основі інформації користувача.
2. Збір даних про проблемне місце в Черкасах (опис проблеми, фото місця та геолокація).
3. Реалізація мікросервісної архітектури (розділ телеграм боту або клієнта, та API яка приймає заявки та зберігає в базі даних Postgres).
4. Реалізація геокодингу (парсинг координат в читаєму адресу).
5. Імплементація адмін панелі, де відповідальна людина працює з заявками.
6. Робота з статусами заявки (New, In process, Done, Archived).
7. Реалізація гугл-мапи, де буде видно всі оброблені заявки з маркерами.

**2.3.** **Опис проекту:**

Ідея створення проекту виникла через незручність комунікації з представниками мерії для обговорення тієї чи іншої проблеми. Зараз, щоб на людину звернули увагу, треба шукати кому написати про виникнення якогось випадку, викладати історії в Instagram чи Facebook, або навіть йти кудись.

На нашу думку це досить архаїчно за наявності потенціалу, який надають сучасні технології. Тому ми вирішили трохи його розкрити, мінімізувавши дії, які необхідно зробити для комунікації з комунальними службами.При постановці цілей проекту потрібно найти відповіді на наступні питання:

1. Як конкретно повинен виглядати результат проекту?

Кінцевий результат повинен бути такий: має бути телеграм бот, в який може написати будь хто бажаючий. Бот спершу привітає людину, запитає про опис проблеми в текстовому форматі, фото та геолокацію місця де потрібно приділити увагу.

2. Які умови повинні враховуватись в процесі реалізації проекту (вимоги та обмеження)?

* Обов’язковою умовою є використання бібліотеки для написання боту в телеграмі
* Бот повинен запитувати дані в строгому порядку (текст-фото-локація)
* При отриманні всіх даних, бот повинен сгрупувати заявку і відправити ії в відповідь користувачу для подальшої валідації. У випадку, якщо все добре – заявка відправляється на API за допомогою POST запиту.
* API повинна приймати запити, проводити валідацію даних, та в результаті зберігати всю інформацію в базі даних.
* API має мати функціонал для адмін панелі (модерування всіх заявок, зміна статусів і так далі)

3. Хто це буде робити? Коли це буде зроблено?

Наша команда складається з двох осіб – Дмитро Білозор та Олександр Кретов. Правду кажучи коли команда тільки зібралась, ми не думали, що нашу ідею хтось підтримає. Ми були приємно здивовані коли до нас на сцені приєднались небайдужі люди. Орієнтовний термін розробки 3 місяці.

**2.4. Системні сервіси:**

Межі системи:

Функціональні вимоги

Сумісність:

Актуальна версія додатку Telegram

Безпека:

Ніякої небезпеки в користуванні телеграм ботом немає, тому наш додаток безпечний.

Відповідність:

Додаток відповідає державним стандартам розробки програмного забезпечення.

Всі завдання виконувалися на ноутбуці MacBook Pro 2021 з такими характеристиками:

* Процесор: Apple M1 Silicon
* ОЗУ: 16 Гб;
* ОС: MacOS v12.2 Sierra;
* ГПУ: M1;
* ПЗУ: SSD 512 Gb.

**2.5. Системні обмеження:**

Вимоги до інтерфейсу: бот має мати максимально не навантажений в плані User Interface. При першому повідомленні

На другому мінімузованому вікні після завантаження файлу повинно повідомяти чи успішно завантажився файл.

Вимоги до продуктивності:

* Час відповіді системи на запити даних має бути 1 секунда.
* Очікуваний рівень активності користувачів(кількість музичного програвача в годину 1 раз, в день 3 рази, тиждень 21 раз, місяць 63 рази).
* Очікуване максимальне навантаження, при якому система має працюватиме нормально

(ЦП - 0.7%, Пам’ять – 900Мб, Диск – 0.1 Мбіт).

Експлуатаційні вимоги:

Правильність: програма функціонує у відповідності з технічним завданням. Програма була протестована велику кількість раз і ніяких ніяких помилок не знайдено.

Універсальність: програма забезпечена правильністю роботи при довільних допустимих даних і має захист від неправильних даних.

Надійність – програма протестована велику кількість раз і забезпечує повну повторюваність результатів, тобто забезпечує їх правильності при наявності різного роду збою.

Перевірюваність – можливість перевірки результатів. Для цього документально зафіксовано початкові дані, вибраний метод та іншу інформацію, яка впливає на результати.

Захищеність – програма забезпечує конфіденційність інформації.

Програмна сумісність – програма не має ніяких проблем із сумісним функціонуванням з іншим програмним забезпеченням.

Ефективність – програма використовує мінімально можливу кількість ресурсів технічних засобів (часу мікропроцесора, об’єму оперативної пам’яті, об’єму зовнішньої пам’яті, кількості зовнішніх пристроїв).

Повторне входження – програма має можливість повторного виконання без перезавантаження з диску. Дана вимога зазвичай пред’являється до програмного забезпечення, резидентно завантаженому в оперативну пам’ять (наприклад, драйвери);

Рентерабельність – програма має можливість «паралельного» використання декількома процесами. Для цього процесу потрібно відкрити повторно програму в іншому вікні.

**2.6. Проектні питання:**

План-графік виконання проектних завдань

|  |  |  |
| --- | --- | --- |
| № | Назва етапів виконання проектного завдання | Термін виконання |
| 1 | Вивчення та аналіз задачі | 18.02.2022 |
| 2 | Розробка архітектури та загальної структури додатку | 21.02.2022 – 26.02.2022 |
| 3 | Підготовка матеріалів | 01.03.2022 – 05.03.2022 |
| 4 | Програмна реалізація додатку | 07.03.2022 – 18.03.2022 |
| 5 | Тестування програмного додатку | 25.03.2022 – 05.04.2022 |

Попередній бюджет: 50$. + 300$ безкоштовного кредиту від Google

Успіх проекту насамперед залежить від керівника, замовника проекту. Також успіх залежить від того наскільки правильно виконавець зрозумів завдання. Наступним етапом є правильне розподілення процесу виконання проєкту.

**2.7. Додатки до документу опису вимог.**

Глосарій

|  |  |
| --- | --- |
| ПЗ | Програмне забезпечення |
| ОС | Операційна система |
| ПК | Персональний комп’ютер |
| ТЗ | Технічне завдання |
| ІС | Інформаційна система |
| МІС | Моніторингова інформаційна система |

**Список літератури:**

1. Документація Node.js. [Електронний ресурс] Режим доступу: <https://nodejs.org/dist/latest-v16.x/docs/api/>
2. Документація бібіотеки Telegraf (чат-бот), [Електронний ресурс] Режим доступу: <https://telegraf.js.org/>
3. Туторіал по Docker та docker-compose, DigitalOcean, [Електронний ресурс] Режим доступу: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04>
4. Docker and PostgreSQL YouTube, [Електронний ресурс] Режим доступу: <https://www.youtube.com/watch?v=aHbE3pTyG-Q>
5. Документація по Docker. [Електронний ресурс] Режим доступу: <https://docs.docker.com/get-started/>
6. Документація по knex.js. [Електронний ресурс] Режим доступу: <http://knexjs.org/>
7. Документація по Javascript, MDN, [Електронний ресурс] Режим доступу: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript>

**2.8. Характеристика області застосування програмного забезпечення, підстави розроблення системи та призначення розробки.**

Основною метою створення додатку стала проблема взаємодії громадян і комунальних служб та адміністрації міста Черкаси. Раніше для того, аби вирішити якусь проблему, треба було шукати відповідні сторінці в Facebook, чи записувати історії в Instagram. Зараз треба лише відправити заявку, яка одразу опиниться на сайті і адміністратор буде передавати її відповідним службам.

**2.9. Основні вимоги до програмного забезпечення та до програмної документації. Оформити специфікацію вимог.**

* Функціональний характер — вимоги до поведінки системи
* Бізнес-вимоги
* Вимоги користувача
* Функціональні вимоги
* Нефункціональний характер — вимоги до характеру поведінки системи
* Бізнес-правила — визначають обмеження, що витікають з предметної області.
* Системні вимоги — вимоги до програмних інтерфейсів, надійності, обладнанню.
* Атрибути якості
* Зовнішні системи та інтерфейси
* Обмеження

Основні компоненти документації:

* специфікація - перелік і призначення всіх файлів програмного виробу, включаючи файли документації;
* відомість власників оригіналів - список підприємств, які зберігають оригінали програмних документів, складається тільки для складних програмних виробів;
* текст програми - запис кодів програми та коментарі до них;
* опис програми - інформація про логічну структуру та функціонування програми;
* програма і методика випробувань - перелік і опис вимог, які повинні бути перевірені в ході випробування програми, методи контролю;
* технічне завдання - документ, в якому викладаються призначення і область застосування програми, вимоги до програмного виробу, стадії і терміни розробки, види випробувань;
* пояснювальна записка - обґрунтування прийнятих і застосованих технічних і техніко-економічних рішень, схеми та опис алгоритмів, загальний опис роботи програмного виробу;

**2.10. Порядок контролю та приймання.**

Поточні результати роботи над дипломним проектом регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за дипломний проект відповідно до критеріїв оцінювання.

2.11.

Технічне завдання

1. Мета: Метою дипломної роботи є створення сервісу для благоустрою у місті Черкаси, який дозволить зручно і оперативно повідомити комунальним службам та керівництву про наявність тієї чи іншої проблеми.
2. Дата початку роботи: «\_\_»\_\_\_\_\_\_\_\_ 2022 р.
3. Дата закінчення роботи: «\_\_»\_\_\_\_\_\_\_\_ 2022 р.
4. Вимоги до програмного забезпечення:

1. Функціональні вимоги:

* Створити мікросервіс для збору та обробки інформації про проблемні місця м. Черкаси
* Побудувати архітектуру проекту та визначити оптимальну структуру коду всіх сервісів
* Обернути сервіс в контейнер за допомогою Docker
* Задеплоїти всі сервіси в платформу Google.

2. Нефункціональні вимоги:

* Створити інтуїтивно зрозумілий, фукціональний і гнучкий інтерфейс для користувача
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 95 - Розробка технічної документації.

5. Стадії та етапи розробки:

1) Вивчення та аналіз задачі(до «\_\_»\_\_\_\_\_\_2022р.)

2) Об'єктно-орієнтоване проектування архітектури програмної системи (до «\_\_»\_\_\_\_\_\_2022р.)

3) Розробка програмного забезпечення (до «\_\_»\_\_\_\_\_\_2022р.)

4) Тестування розробленої програми (до «\_\_»\_\_\_\_\_2022р.)

5) Розробка пояснювальної записки (до «\_\_»\_\_\_\_\_\_2022 р.).

6) Захист дипломної роботи (до «\_\_»\_\_\_\_\_\_2022 р.).

6. Порядок контролю та приймання. Поточні результати роботи над дипломним проектом регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за дипломний проект відповідно до критеріїв оцінювання.

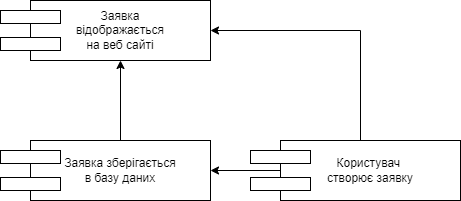
**2.12 Діаграма класів**



**2.13. Діаграма пакетів.**



**2.14. Діаграми компонентів.**



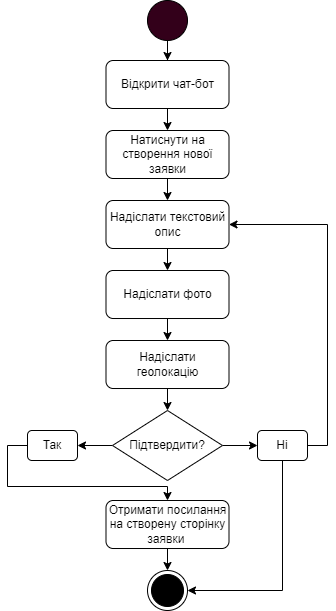
**2.15. Діаграми прецедентів.**





**2.16. Діаграми діяльності.**

Користувач:



Адміністратор:



**2.17. Діаграми станів.**

Користувач:

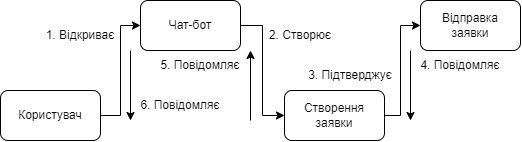


Адміністратор:

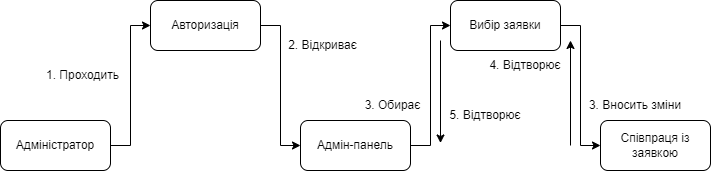


**2.18. Діаграми кооперації.**

Користувач:

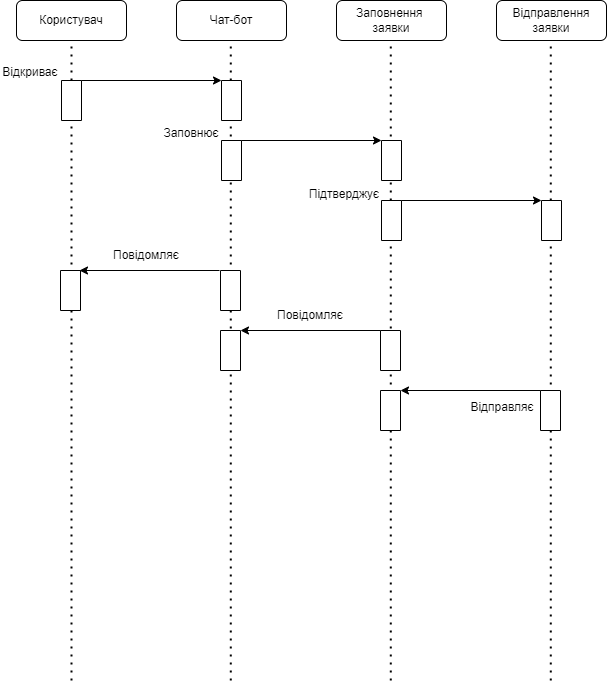


Адміністратор:

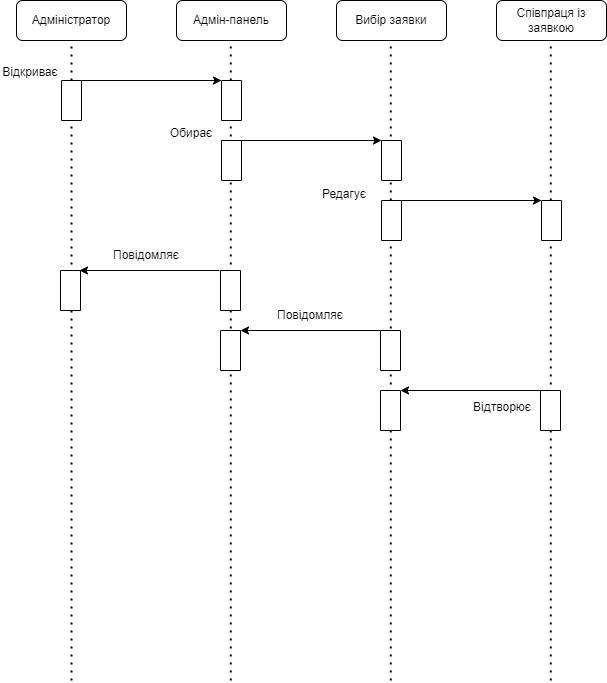


**2.19. Діаграми послідовності.**

Користувач:



Адміністратор:



**2.20. Вибір способу побудови архітектури веб-додатку.**

Даний проект створений на основі мови програмування JavaScript. Незважаючи на те що JS задумувався як розширення функціоналу веб-сторінок в браузері, зараз він набув широкої популярності і його застосування стало набагато більшим.

Мова JavaScript використовується для:

* написання сценаріїв вебсторінок для надання їм інтерактивності
* створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js)
* програмування на боці сервера (Node.js (Express.js ))
* стаціонарних застосунків мобільних застосунків
* сценаріїв в прикладних програмах

**2.22. Особливості роботи кожної групи користувачів з кожним елементом взаємодії користувача з веб-додатком.**

В кінцевому результаті існує дві групи користувачів:

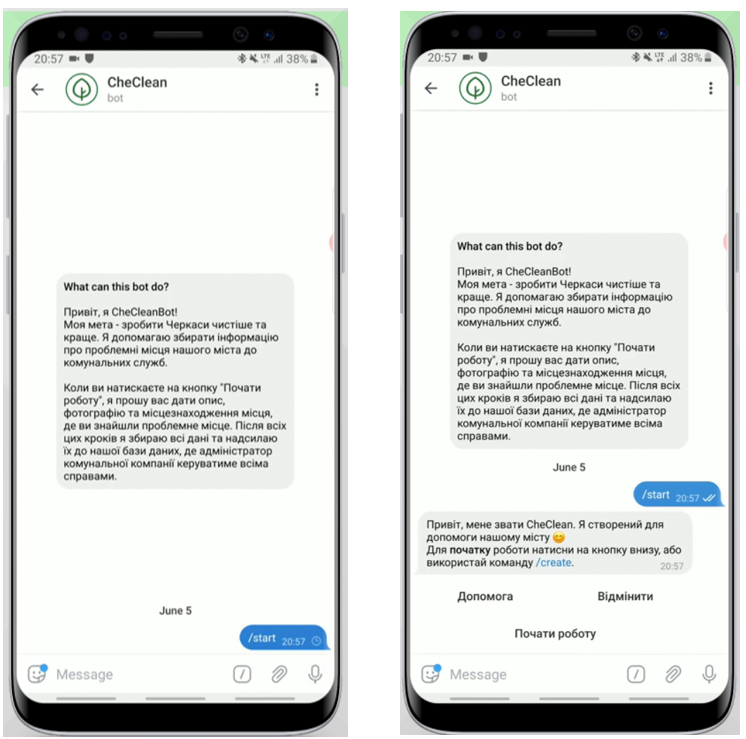
* користувачі чат-боту;
* користувачі сайту для управління заявками.

Основне завдання користувачів чат-боту це створення заявок, які містять текстовий опис проблеми, фотографію та геолокацію. Після створення заявки користувачем, бот запитує чи всі дані введено правильно, і якщо користувач підтверджує, заявка зберігається в базу даних, якщо щось не вірно, є можливість редагувати заявку. Далі заявка з’являється на сайті, де можна побачити всі дані, що були введені, посилання на сторінку заявки клієнт отримує після того як підтвердив її створення.

Основне завдання адміністратора це правити створені користувачами заявки. Заявка з’являється на сайті, адміністратор заходить, ознайомлюється з характером проблеми, передає її у відповідний відділ і після вирішення проблеми міняє статус на виконаний, і лишає коментар. Якщо заявка не валідна – видаляє.

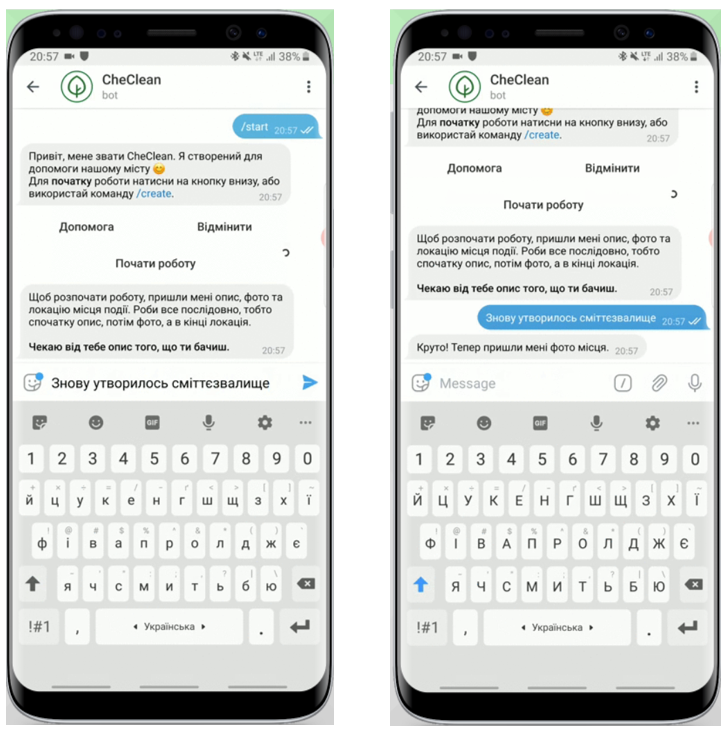
**2.23. Інтерфейс**

Огляд взаємодії користувача і боту



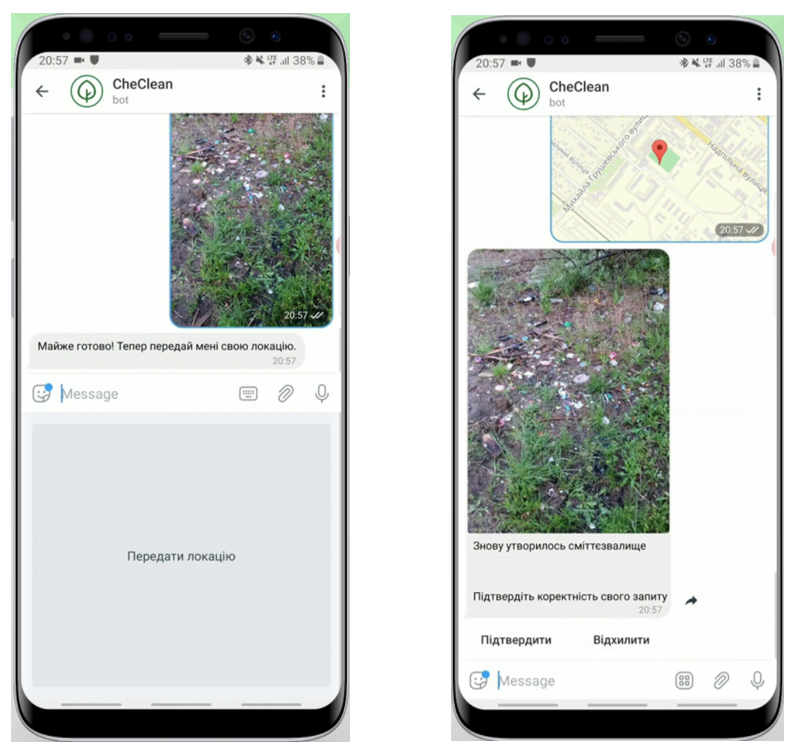
*Рисунок 1. Початок роботи.*

На початку бот зустрічає користувача повідомленням з коротким поясненням для чого він створений. Для того щоб почати взаємодіяти, треба запустити бот командою /start і в наступному повідомленні натиснути «Почати роботу».



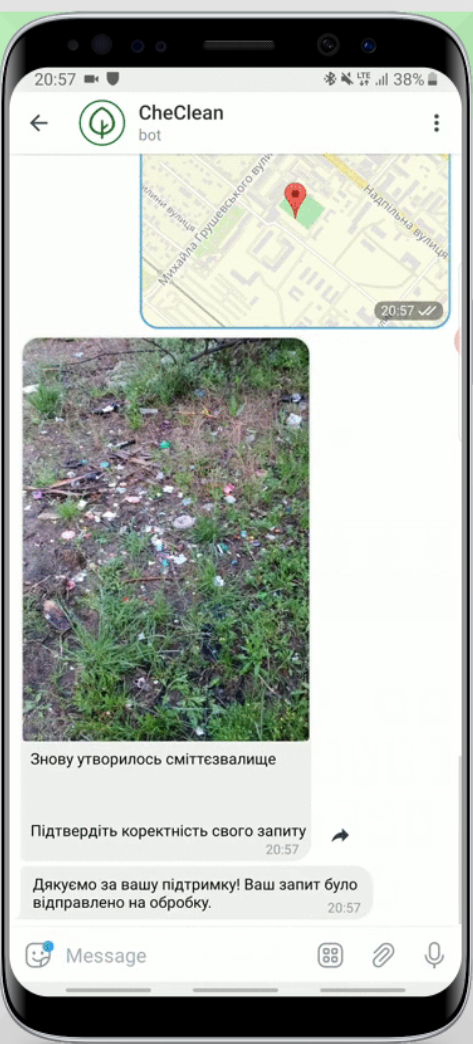
*Рисунок 2. Заповнення опису.*

Далі користувач відправляє текстовий опис проблеми, після чого бот просить відправити фотографію місця.



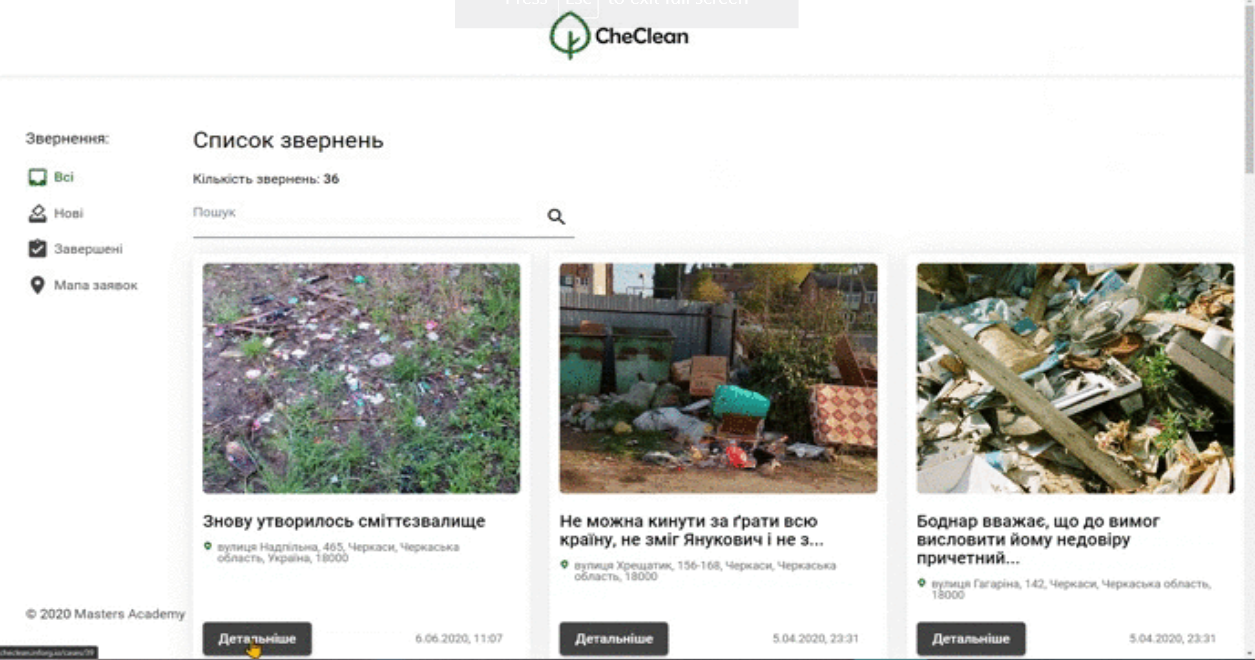
*Рисунок 3. Відправлення геолокації.*

Після того як користувач відправив фотографію, останнє що потребується надіслати – це локація.



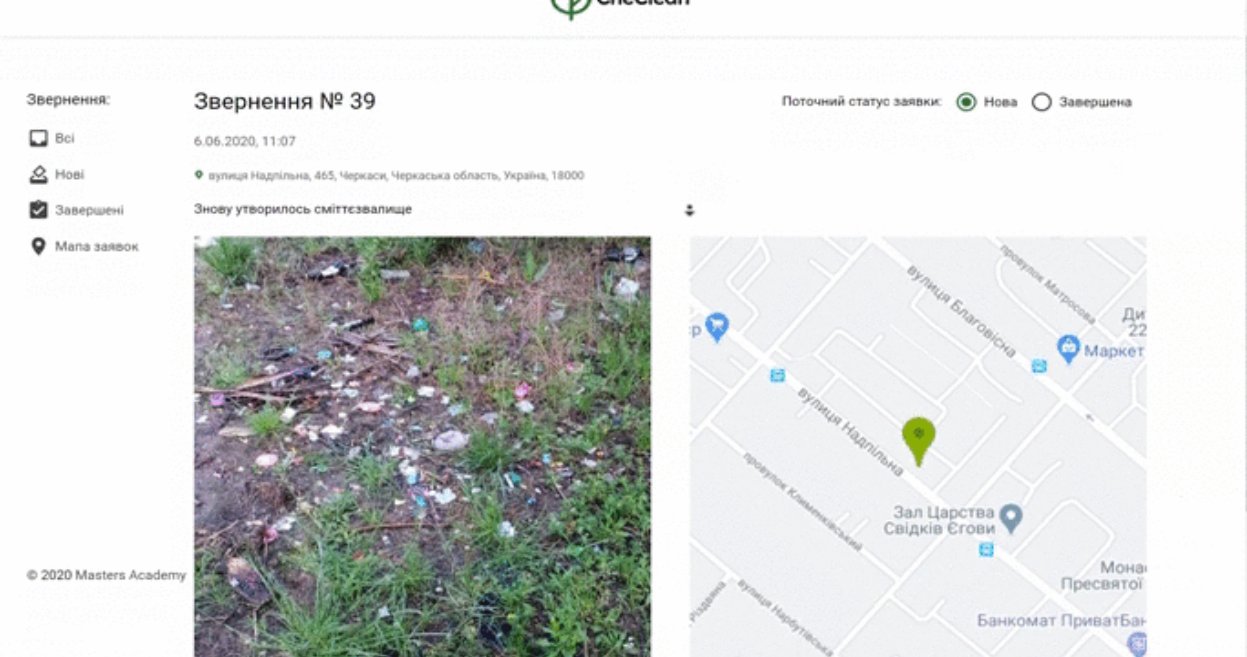
*Рисунок 4. Підтвердження заявки.*

Після того як всі дані зібрано, бот відправляє попередній перегляд заявки, після чого користувач може редагувати її, або підтвердити, і вона збережеться в базу даних та буде доступною на сайті.



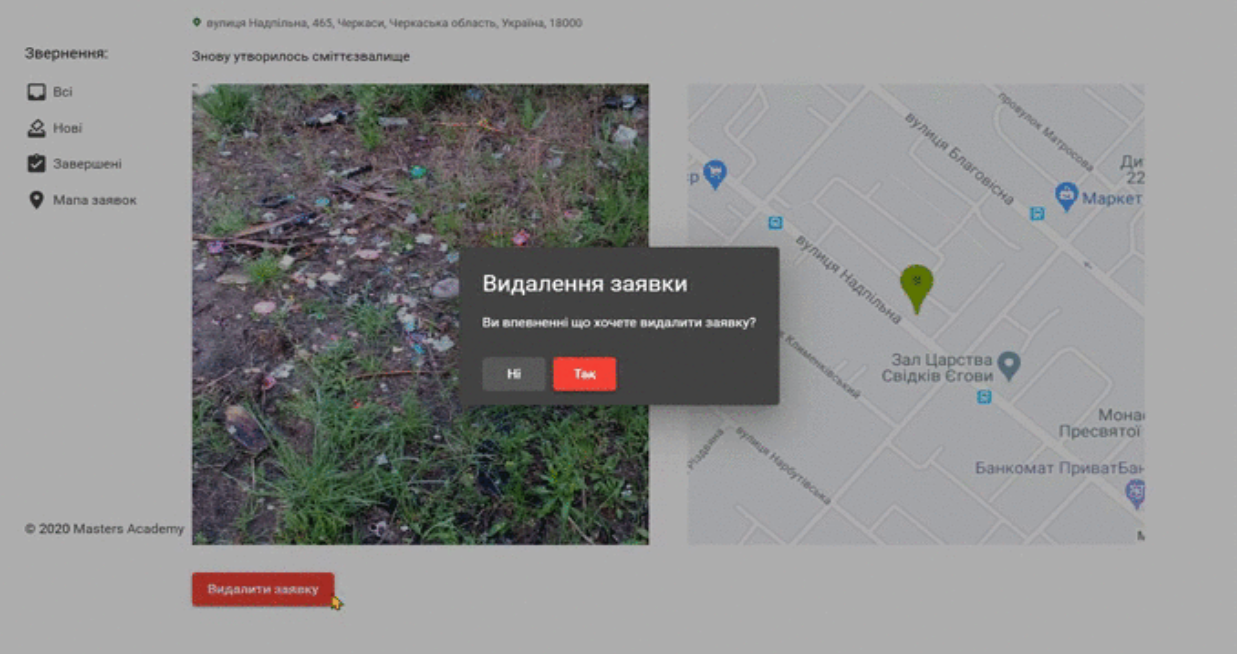
*Рисунок 5. Список заявок на сайті.*

Головна сторінка представляє собою список із всіх звернень з превью фотографії, локації і опису (див. рис.5). Зліва можна відфільтрувати заявки на нові та завершені, також, можна перейти на мапу заявок (див. рис.8).



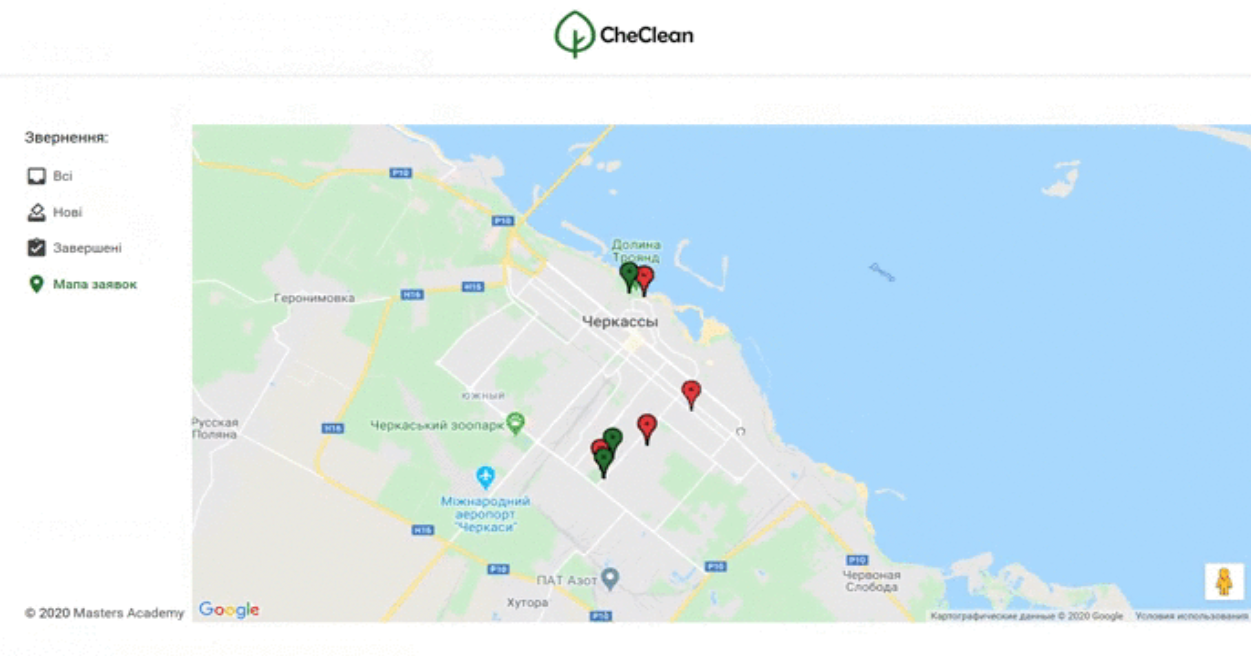
*Рисунок 6. Інтерфейс заявки.*

Якщо натиснути на заявку, то ми перейдем на її розгорнуту сторінку, де можна змінити її статус, або видалити.

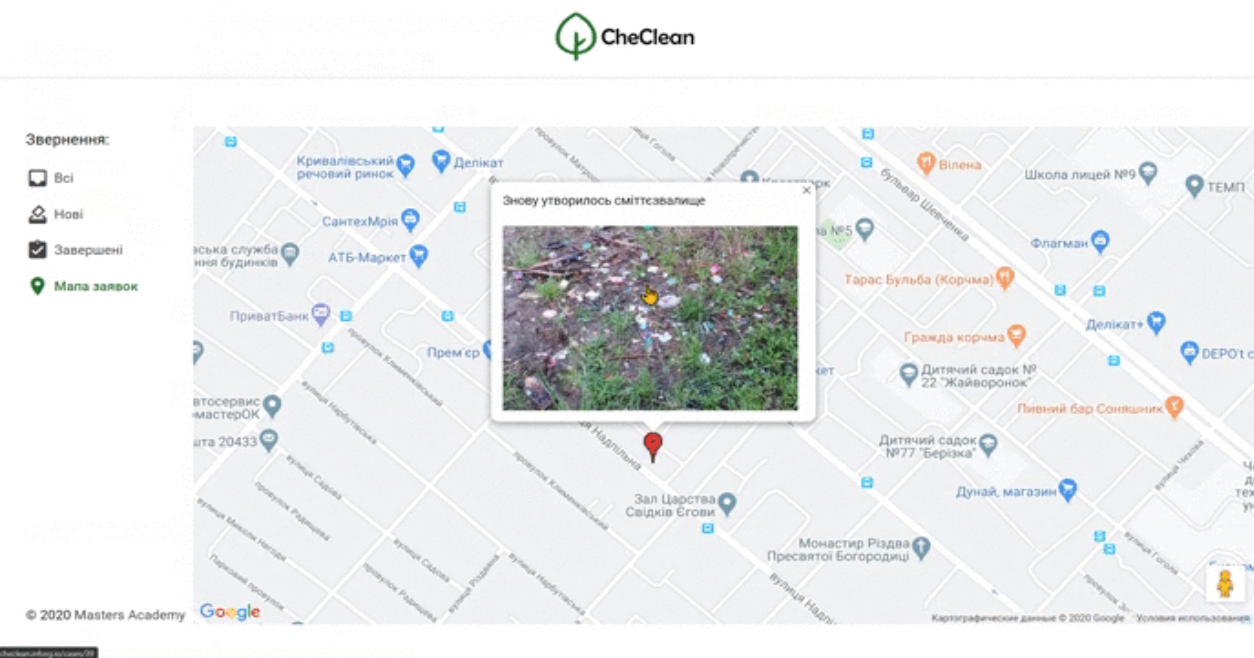


*Рисунок 7. Видалення заявки.*

На мапі заявок ми можем побачити локацію кожної заявки, та інтуїтивно, за допомогою кольору точки, зрозуміти її статус. Точки інтерактивні і якщо на них натиснути, можна також побачити опис заявки і перейти на її сторінку (див. рис.9).



*Рисунок 8. Мапа заявок.*



*Рисунок 9. Обрана заявка на мапі.*

**2.24 Основними властивостями, яким повинен задовольняти інтерфейс, є такі:**

**Адаптованість** означає, що інтерфейс повинен бути:

– сумісним з потребами та можливостями користувача;

– забезпечувати простоту переходу від виконання однієї функції до іншої;

– забезпечувати користувача на високому рівні вказівками стосовно його можливих дій, а також генерувати належний зворотний зв’язок на його запити;

– надавати користувачу можливість відчувати себе повноправним керівником ситуації при розв’язанні всіх типів задач, тобто, забезпечувати його всією необхідною інформацією; користувач повинен бути впевненим, що він сам розв’язує поставлену задачу;

– забезпечувати користувача різними, взаємно доповнюючими формами представлення результатів в залежності від типу запиту або від характеру отриманого рішення;

– враховувати особливості користувачів різних рівнів; наприклад, для керівника підприємства зручнішим є узагальнене графічне ставлення результатів роботи СППР у вигляді діаграм та графіків, а інженеру-економісту потрібні конкретні цифри у їх часовій послідовності.

**Достатність інтерфейсу означає таке:**

– допустимі запити користувача повинні бути чіткими і однозначними для користувачів всіх рівнів, а також для прикладних задач всіх типів;

– реакція системи на всі типи запитів також повинна бути однозначною і зрозумілою і, по можливості, простою.

**Дружність інтерфейсу**

Це максимальна простота його використання і готовність в повній мірі задовольнити запити користувача при розв’язанні визначеного класу задач.

**Гнучкість інтерфейсу**

Гнучкість інтерфейсу – це можливість його адаптування до розв’язання конкретної задачі. Якщо розв’язувана задача дуже складна, то інтерфейс повинен полегшувати формулювання запитів і видавати результати у формі, яка легко і швидко сприймається користувачем. Тобто інтерфейс повинен буди максимально простим навіть у випадку, коли розв’язується дуже складна задача. При цьому простота означає таке:

– інтерфейс не повинен бути перевантажений деталями щодо представлення розв’язку поставленої задачі – користувач може не охопити всіх подробиць (і в цьому, як правило, немає потреби) – тобто нічого зайвого, крім того, що необхідно для розуміння результату;

– він не повинен містити зайвих декоративних деталей, які відволікають від головної задачі;

– інтерфейс повинен бути консистентним, тобто, ґрунтуватись на використанні відомих, загальноприйнятих методів і засобів представлення інформації;

– в ідеалі процес взаємодії користувача з системою не повинен представляти ніяких труднощів.

**2.25. Структура веб-додатку**

Після того як користувач надіслав заявку через чат-бот вона надходить до бекенду, через який вона потрапляє на сайт із заявками (адмін-панелі). Адмін-панель складається з 3х частин: загальний список, детальна інформацію про заявку та загальна мапа заявок. В загальному списку заявки можна відсортувати по категоріям, знайти заявку через пошук.

В сторінці "детальна інформація" можна переглянути повний опис проблеми, скачати повнорозмірне фото, переглянути карту проблемного місця та змінити статус заявки

Веб-додаток розділений на дві частини:

* для користувача;
* для адміністратора

У версії для користувача є можливість переглядати список заявок, їх статуси, поточну кількість та мапу заявок.

Версія для адміністратора відрізняється від користувацької тим, що після авторизації, адміністратор може співпрацювати із заявками: міняти статус, лишати коментарі, видаляти.

**2.37.** Вимоги до функціональності, що визначені в оформленій документації на програмне забезпечення.

* Можливість користуватись на будь-якому пристрою;
* Можливість звернутись до розробників, якщо виникають питання;
* Можливість попереднього перегляду заявки перед відправленням;
* Можливість редагувати заявку, якщо виявлені помилки;
* Можливість переглядати створену заявку на сайті;
* Можливість фільтрувати заявки, в залежності від їх статусу;
* Можливість інтуїтивно розуміти статус заявки, виходячи з кольору;
* Можливість змінювати статус заявки (якщо адміністратор);
* Можливість видаляти заявку (якщо адміністратор);
* Можливість лишати коментар у заявці (якщо адміністратор);

**2.28. Необхідні модулі розширення функціональності.**

Стек використаних бібліотек та технологій:

* Angular
* Angular Material
* Google Maps API
* Docker
* Express
* PostgreSQL
* Knex.js
* Node-Geocoder
* Queue-promise

**2.29. Внесені зміни.**

Останніми було зроблено:

* авторизацію;
* загальну мапу кейсів;
* можливість видалення кейсу.

**2.30 Аналіз якості сервісу.**

В якості кінцевого результату було реалізовано:

* Інтуїтивно зрозумілий бот.
* Зручна навігація по боту, можливість редагувати звернення.
* Веб-додаток з інтуїтивно зрозумілим інтерфейсом.
* Зручна навігація по сайту, можливість пошуку та фільтрів.
* Мапа заявок з інтерактивними точками локації, можливість переходу на сторінку звернення при натисканні.

**2.31 Тестування сумісності.**

Програмне забезпечення працює на всіх версіях Windows:

* Windows Vista
* Windows XP
* Windows 7
* Windows 10
* Windows 11

Також працюватиме і на macOS.

Чат-ботом можна скористатись на любому пристрої з будь-якою системою, якщо є підтримка телеграму.

**2.33 Перевірити ергономічність веб-додатку.**

Сервіс має такі показники:

* Легкість вивчення (learnability)
* Ефективність (efficiency)
* Легкість запам'ятовування(memorability)
* Відсутність помилок (errors)
* Суб'єктивне задоволення (satisfaction)

**2.34. Безпека веб-додатку.**

На сайті була створена можливість міняти статуси звернень та видаляти їх, якщо це зможе робити будь-хто, то це буде серйозною загрозою, для обмеження прав на взаємодію зі зверненнями було додано авторизацію. Тільки адміністратор має право редагувати та видаляти звернення, звичайні користувачі можуть лише переглядати сайт.

**2.35 Провести тестування стабільності.**

Після проведення тестування веб-додатку, прийшов до висновку що він працює стабільно, без затримок і помилок. Невід’ємною умовою є якісне підключення до інтернету.

**2.36 Специфікація на програмне забезпечення.**

Дане програмне забезпечення відноситься до сервісу благоустрою. Після проведення дослідження предметної області було зроблено висновок, що подібного сервісу досі не існувало. Саме тому на меті було створити щось нове і корисне.



*Рисунок 10. «Живий» приклад роботи сервісу*

**2.37 Опис програми.**

Цей проект повинен покращити взаємодію городян та комунальних служб. Ідея створення проекту виникла через незручність комунікації з представниками мерії для обговорення тієї чи іншої проблеми. Зараз, щоб на людину звернули увагу, треба шукати кому написати про виникнення якогось випадку, викладати історії в Instagram чи Facebook, або навіть йти кудись. Це досить архаїчно за наявності потенціалу, який надають сучасні технології. Тому було прийнято рішення його розкрити, мінімізувавши дії, які необхідно зробити для комунікації з комунальними службами.

**2.38 Керівництво програміста.**

Для запуску проекту необхідною вимогою є наявність:

* node v12
* postgres v12
* docker

Встановлення:

npm install

Запуск:

npm start

Ви можете використовувати docker desktop для запуску проекту. В перший раз необхідно виконати команду:

npm run docker:init

Після першого запуску, в наступні рази слід користуватись командою:

npm run docker

Для того щоб запустити міграцію слід скористатись командою:

npm run docker:migrate

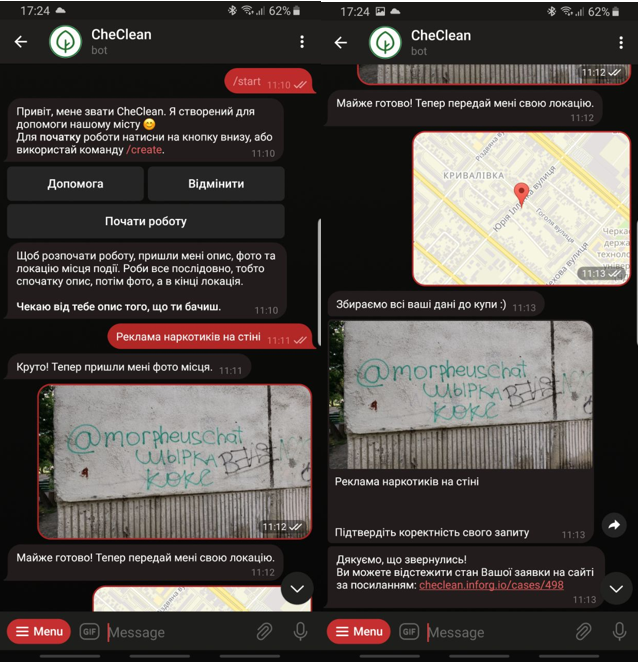
Для того щоб вставити мок-дані в базу необхідно використати команду:

npm run docker:seed

**2.39 Керівництво користувача.**

На початку бачим повідомлення з коротким поясненням для чого створений бот. Для того щоб почати взаємодіяти, треба запускаємо бот командою /start і в наступному повідомленні натиснути «Почати роботу».

Далі відправляєм текстовий опис проблеми, після чого бот просить відправити фотографію місця.



*Рисунок 11. Робота з ботом.*

Після того як відправили фотографію, останнє що потребується надіслати – це локація.

Після того як всі дані зібрано, бот відправляє попередній перегляд заявки, після чого можна редагувати її, або підтвердити, і вона збережеться в базу даних та буде доступною на сайті. Побачити її на сайті можна перейшовши за посиланням в останньому повідомленні (див. рис.11).

**Висновок**

Під час виконання даної лабораторної роботи здобув навички у побудові UML діаграм для програмного забезпечення. Проводити аналізи щодо зручності використання програмно забезпечення, тестування та виправлення помилок. Було здійснено широкий опис та дослідження роботи сервісу CheClean.