МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення, канд. техн. наук, доцент,

_____Є. О. Давиденко

«____»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІДСТЕЖЕННЯ ПОМИЛОК НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 − 408.21810809

Студент _____А. І. Гінжул

підпис

«___» _____ 2022 р.

Керівник ст. викладач кафедри інженерії програмного забезпечення

_____С. Ю. Боровльова

підпис

«___» _____ 2022 р.

Консультант канд. техн. наук, доцент кафедри екології

_____А. О. Алексєєва

підпис

«___» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БД – база даних
МН – машинне навчання
НФ – нормальна форма
СКБД – система керування базами даних
ООП – об’єктно-орієнтоване програмування
ПЗ – програмне забезпечення

CRUD – Create, Read, Update, Delete
CSS – Cascading Style Sheets
CSV – Comma-Separated Values
GUI − Graphical User Interface
JSON – JavaScript Object Notation
MVC – Model-View-Controller
ORM – Object-Relational Mapping
URL – Uniform Resource Locator

ВСТУП

Системи відстеження помилок широко використовуються при розробці програмного забезпечення. Сутність даних застосунків полягає у фіксації, обліку та систематизації знайдених завдяки тестуванню помилок. Крім того, даний тип програмного забезпечення допомагає управляти проєктами.

Існує ще одне важливе застосування для системи відстеження помилок. Вона може бути використана як джерело даних про процес розробки програмного забезпечення. На основі інформації про виконані проєкти можливо формувати корисні статистичні звіти та персоналізовані рекомендації. Детальний опис помилок та обставин їх виникнення допоможе виявити найбільш вразливі елементи програмного забезпечення та провести об’єктивний аналіз роботи команди розробників.

Основні особливості системи, яка буде створена в ході виконання бакалаврської кваліфікаційної роботи:

а) аналітична обробка даних про процес розробки програмного забезпечення та про помилки, що виникли в ході розробки;
б) формування рекомендацій на базі інформації про попередні проєкти;
в) візуалізація статистичних даних та автоматизована генерація звітів.

Тема роботи є **актуальною**, оскільки пандемія COVID-19 та російсько-українська війна змусили представників багатьох професій, в тому числі і програмістів, працювати віддалено. В інших країнах популярність дистанційної роботи також зростає. Отже, інструменти для організації командної роботи та управління проєктами онлайн набувають надзвичайної важливості та популярності в таких обставинах.

Об’єкт кваліфікаційної роботи – процес відстеження помилок при розробці програмного забезпечення.

Предмет кваліфікаційної роботи – використання машинного навчання та аналітичної обробки даних для оптимізації процесу відстеження помилок при розробці програмного забезпечення.

Мета кваліфікаційної роботи – вдосконалення процесу відстеження помилок при розробці програмного забезпечення.

Для досягнення поставленої мети необхідно розв’язати наступні **завдання**:

а) аналіз предметної області та аналогічного програмного забезпечення;
б) специфікація вимог на основі здійсненого аналізу;
в) аналіз алгоритмів машинного навчання та аналітичної обробки даних;
г) проєктування та моделювання системи;
д) реалізація, тестування та відлагодження вебзастосунку.

Для реалізації системи відстеження помилок на основі машинного навчання буде використана мова програмування *Python*. Розроблена система буде представляти собою вебзастосунок.

Отже, поєднання потужної інтелектуальної та математичної складової з перевагами вебзастосунку (сучасний та естетичний графічний інтерфейс, існування у вигляді сервісу в мережі Інтернет) робить систему, що розробляється, якісною та ефективною.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Для з’ясування специфіки системи відстеження помилок на основі машинного навчання необхідно проаналізувати наступні поняття: методологія розробки програмного забезпечення, тестування та машинне навчання.

1.1.1 Методологія розробки програмного забезпечення

Система відстеження помилок має базуватися на певній методології розробки ПЗ. Найбільш актуальними та розповсюдженими є ті методології, що відносяться до сімейства Agile.

Agile представляє собою набір *методів* та *методологій,* що допомагають більш ефективно мислити, працювати та приймати рішення у команді [1]. Вказані методи та методології складаються з *практик*, які оптимізовані для максимально простого та швидкого впровадження. Також Agile можна сприймати як своєрідну філософію (*mindset*) роботи, що базується на наступних принципах [2]:

а) люди та співпраця важливіші за процеси та інструменти;

б) справний продукт важливіший за вичерпну документацію;

в) співпраця з замовником важливіша за умови контракту;

г) готовність до змін важливіша за дотримання плану.

У системі буде використано **Kanban**. Даний метод відноситься до сімейства Agile та допомагає командам вдосконалювати свій підхід до розробки програмного забезпечення і командної роботи [1].

Таким чином, у системі має бути віртуальна дошка, на якій завдання та звіти про помилки (*work items*) будуть розсортовані за категоріями в залежності від їхнього статусу. Необхідно створити наступні категорії:

а) backlog (початкова категорія для всіх елементів);

б) to do (треба зробити найближчім часом);

в) doing (над чим зараз працює команда);

г) done (вже виконано).

Отже, інформацію про помилки буде легше систематизувати, візуалізувати, відстежувати та оперативно доносити до команди завдяки впровадженню Kanban у систему.

1.1.2 Тестування

Основною ідеєю **тестування** програмного забезпечення є випробування застосунка з метою виявлення помилок та наступного підвищення якості. **Помилка** представляє собою розходження між очікуваною та фактичною поведінкою системи [3].

В результаті тестування складаються звіти. При побудові вказаних звітів використовуються наступні атрибути:

а) *ідентифікатор* – унікальне значення, що дозволяє однозначно відрізнити одну помилку від іншої;

б) *короткий опис* – стислий виклад суті помилки;

в) *детальний опис* – детальний виклад суті помилки, в тому числі кроки для відтворення, коди помилок і т.д;

г) *відтворюваність* – регулярність появи помилки при здійсненні всіх кроків для її відтворення:

1) always (завжди з’являється);
2) sometimes (з’являється час від часу).

д) *важливість* – ступінь шкідливості для програмного забезпечення:

1) critical (критична);
2) major (велика);
3) medium (середня);
4) minor (незначна).

е) *терміновість* – опис того, як швидко необхідно усунути дефект:

1) ASAP (as soon as possible, дуже висока);

2) high (висока);
3) normal (середня);
4) low (низька).

ж) *симптом* – класифікація помилки за її типовими ознаками:

1) cosmetic flaw (візуальний недолік, що не впливає на функціональність);
2) data loss (втрата даних);
3) documentation issue (проблема з документацією, а не застосунком);
4) incorrect operation (некоректне виконання операції);
5) installation problem (помилка на стадії встановлення та конфігурування);
6) localization issue (помилка локалізації);
7) missing feature (деяка функціональність не виконується);
8) scalability (при збільшенні доступних застосунку ресурсів не відбувається очікуваного зростання продуктивності);
9) low performance (витрата занадто великої кількості часу на виконання деяких операцій);
10) system crash (аварійне закінчення роботи застосунку);
11) unexpected behavior (нетипова поведінка системи);
12) unfriendly behavior (система створює незручності користувачу);
13) variance from specs (невідповідність вимогам);
14) enhancement (пропозиція щодо покращення).

з) *можливість обійти* – чи можливо досягти мету способом, який не містить помилок;

и) *прикріплені файли* – як правило, скріншоти екрану і т.д.

Отже, модель даних, що відноситься до помилки, має обов’язково містити наведені вище атрибути.

1.1.3 Машинне навчання

Машинне навчання представляє собою науку програмування комп'ютерів таким чином, щоб вони могли вчитися на основі даних. Більш формальне визначення звучить так: говорять, що комп'ютерна програма навчається на основі досвіду ***E*** по відношенню до деякої задачі ***T*** та деякої оцінки продуктивності ***P***, якщо її продуктивність на ***T***, виміряна за допомогою ***P***, покращується з досвідом ***E*** [4]. Машинне навчання застосовується:

а) для задач, рішення яких вимагає тривалого налаштування вручну або довгого списку правил у коді застосунку;
б) для змінних середовищ;
в) для роботи з Big Data;
г) для складних задач, рішення яких так і не було знайдено за допомогою традиційного підходу.

Можливо виділити типи систем машинного навчання, згруповані на основі наступних ознак:

а) наявність людського контролю (навчання з вчителем, навчання без вчителя, напівавтоматичне навчання та навчання з підкріпленням);
б) здатність навчатися на льоту (динамічне або пакетне навчання);
в) здатність виявляти патерни в навчальних даних (навчання на основі зразків або на основі моделей).

В даній системі машинне навчання буде застосоване для прогнозування часу на виконання завдання або виправлення помилки. Дана задача за своїм типом є **регресією**. Регресія представляє собою прогнозування цільового значення на основі набору прогнозаторів (характеристик) [4]. Це одна з найбільш типових задач для **навчання з вчителем**. Система має отримати багато наборів прогнозаторів та відповідних їм міток. Кількість навчальних даних впливає на точність результату.

Існують такі алгоритми навчання з вчителем [4]:

а) лінійна регресія;

б) логістична регресія;

в) метод опорних векторів;

г) дерева прийняття рішень;

д) нейронні мережі;

е) k найближчих сусідів.

У випадку даної системи *прогнозаторами* є особа розробника та складність завдання (помилки), *міткою* – час, витрачений на завершення завдання. Система буде вчитися на основі виконаних завдань. В якості алгоритму МН будуть застосовані ***дерева прийняття рішень***.

Отже, маючи такі підказки, менеджер проєкту зможе визначити реалістичні терміни завершення окремих завдань та проєкту взагалі. Впровадження машинного навчання допоможе скоротити випадки перевантаження команди надмірним об’ємом роботи та зриву обіцяних замовнику термінів завершення проєкту.

1.2 Аналіз аналогічних систем

1.2.1 JIRA

Розробником JIRA є австралійська компанія Atlassian. Мовою реалізації даного програмного забезпечення є Java. Оскільки в JIRA є базова частина та велика кількість додаткових плагінів, то найбільш логічною в даному випадку виглядає мікроядерна архітектура.

Нижче представлено перелік ***основних функцій*** даного програмного забезпечення:

а) можливість планувати робочий процес за методологією Agile;

б) Kanban-дошки для кожного проєкту;

в) розстановка пріоритетів та відмірювання часу на кожне з завдань;

г) статистика та звіти про продуктивність роботи;

д) можливість призначати виконавця завдання.

Переваги JIRA:

а) сумісність з методологією Agile;

б) синхронізація з системами контролю версій та інтегрованими середовищами розробки;

в) гнучкість (завдяки різноманітним плагінам);

г) висока якість та зручність комунікації в режимі реального часу;

д) декомпозиція завдань;

е) управління правами доступу користувачів;

ж) наочне відображення прогресу у виконанні завдань;

з) велика спільнота.

Недоліки JIRA:

а) багато плагінів працюють некоректно;

б) GUI, зображений на рисунку 1.1, не є інтуїтивно зрозумілим;

в) управління правами доступу є досить заплутаним;

г) є проблеми з адаптивністю інтерфейсу для мобільних пристроїв;

д) немає можливості імпортувати набір завдань;

е) формат Kanban-дошки є заздалегідь визначеним;

ж) завдання та баги призначаються користувачу імперативно.

Отже, JIRA представляє собою найбільш сучасну та популярну версію системи відстеження помилок. Розробники постійно збільшують кількість функцій та плагінів, але деякі з них, на жаль, недостатньо протестовані та працюють некоректно. Початківцям буває важко вивчати та налаштовувати таку масштабну та різноманітну систему.

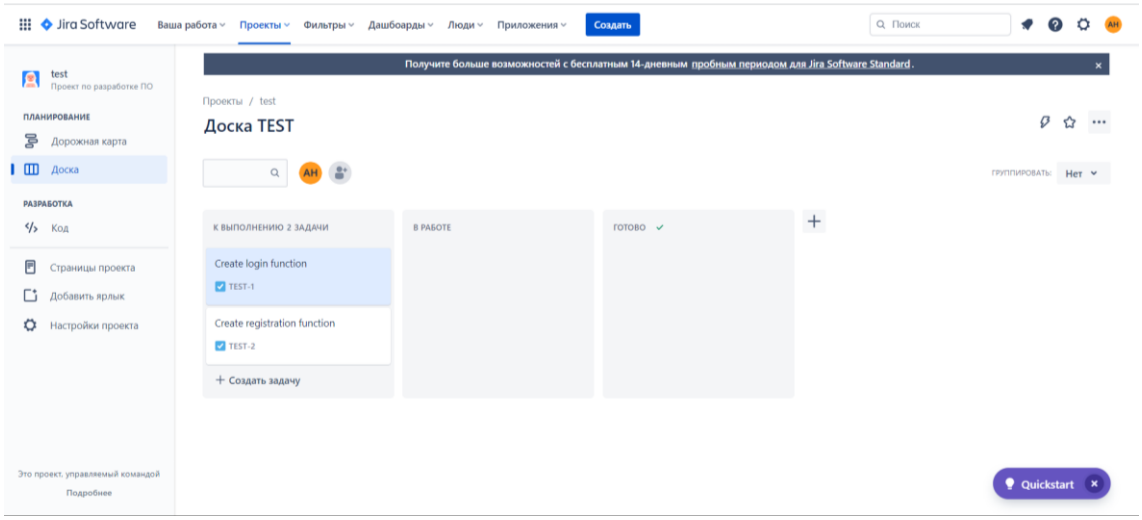

Рисунок 1.1 – Інтерфейс JIRA

Зображений на рисунку 1.2 інтерфейс створення задачі є досить вдалим, тому його варто взяти в якості прикладу.

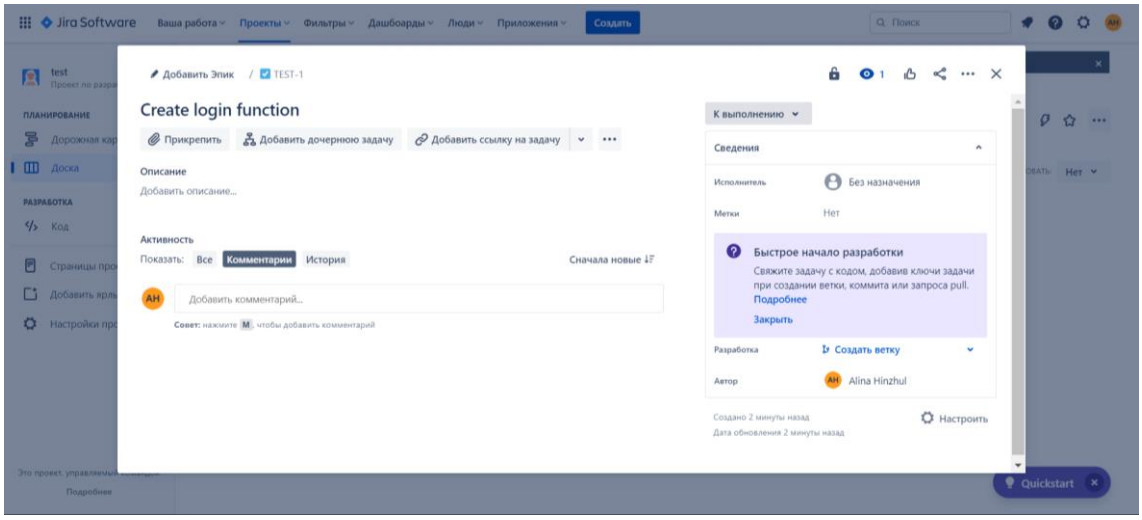

Рисунок 1.2 – Інтерфейс опису задачі в JIRA

1.2.2 Bugzilla

Bugzilla є еталонним прикладом системи відстеження помилок. Розроблена в 1998 організацією Mozilla Foundation, система досі залишається працездатно. Мовою реалізації даного програмного забезпечення є Perl. Застосунок має багаторівневу архітектуру.

Перелік функцій, представлений на офіційному сайті Bugzilla, виглядає наступним чином:

а) гнучка система пошуку з наступним збереженням запиту;

б) сповіщення про внесені зміни за допомогою електронних листів;

в) відображення списку помилок у різних форматах (Atom, iCal і т.д.);

г) автоматичне виявлення повторюваних записів;

д) звітність та діаграми;

е) робота з даними про помилки за допомогою електронних листів;

ж) відмірювання часу на виправлення помилки;

з) призначення певного завдання іншому користувачу. Користувач має право прийняти або відхилити дане призначення;

и) автоматичне перенесення списку помилок з одного екземпляру програмного забезпечення в інший;

к) приватні прикріплені файли та коментарі.

Переваги Bugzilla:

а) система проста для розуміння та не містить нічого зайвого;

б) система ретельно протестована;

в) користувач може погодитися або відмовитися від роботи над певним завданням;

г) опис помилки є максимально інформативним.

Недоліки Bugzilla:

а) відсутність сумісності з Agile;

б) Не відображається загальний стан роботи над проєктом;

в) Обмежена кількість плагінів;

г) Обмежена інформативність звітів;

д) Не вистачає можливостей для кастомізації;

е) застарілий GUI;

ж) в якості коментаря не можна додати посилання, зображення і т.д.

Отже, Bugzilla є якісним, добре протестованим та відлагодженим інструментом, що не містить в собі нічого зайвого. Дане програмне забезпечення є простим для опанування та використання. Розробники користуються цим інструментом у таких компаніях, як NATO, NASA, Nokia та Wikipedia. Але неможливість пристосувати Bugzilla до методології Agile та зосередженість лише на помилках, а не на проєкті взагалі, робить дану систему застарілою. Сторінка для опису помилки надає можливість дуже детально описати помилку. Зображений на рисунку 1.3 інтерфейс варто використати як зразок для форми створення звіту про помилку.

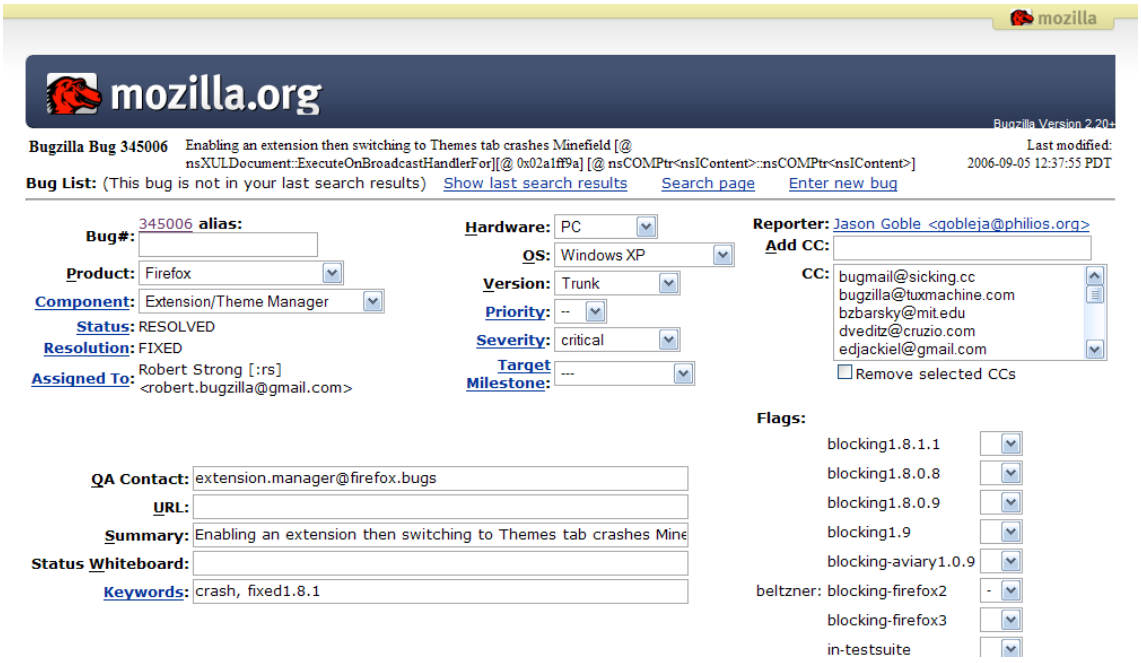

Рисунок 1.3 – Приклад опису помилки в Bugzilla

1.2.3 Mantis Bug Tracker

Розробником Mantis Bug Tracker є Victor Boctor. Мовою реалізації є PHP . Архітектура системи є багаторівневою. Система має наступний набір функцій:

а) сповіщення про внесення зміни за допомогою електронних листів;

б) інтеграція з системами контролю версій;

в) підключення плагінів;

г) контроль версій запитів;

д) повнотекстовий пошук;

е) підсумок змін;

ж) візуалізація зв’язків між запитами;

з) інтеграція з Wiki-системами.

Переваги Mantis Bug Tracker:

а) система проста для вивчення та використання;

б) система є досить гнучкою;

в) управління правами доступу.

Недоліки Mantis Bug Tracker:

а) відсутність звітності;

б) примітивний інтерфейс (зразок GUI зображено на рисунку 1.4);

в) неможливо прикріпити одразу декілька зображень до опису помилки.

Отже, Mantis Bug Tracker представляє собою простий та зручний застосунок для відстеження помилок та управління проєктами.

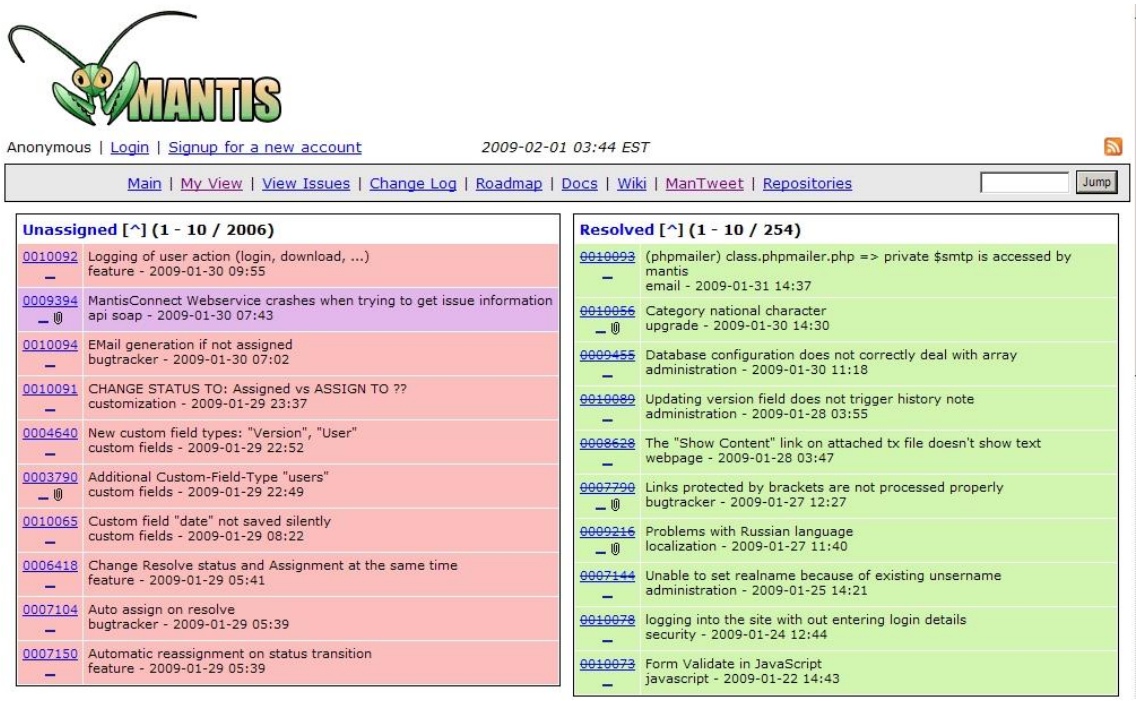

Рисунок 1.4 – Інтерфейс Mantis Bug Tracker

1.3 Специфікація вимог

1.3.1 Опис основних функцій системи

Програмне забезпечення відстеження помилок на основі машинного навчання, яке розробляється в ході бакалаврської дипломної роботи, має наступний набір функціональних можливостей:

а) реєстрація та авторизація *користувачів* в системі з наступним розмежуванням прав доступу в залежності від ролі;
б) здійснення CRUD-операцій над даними про *проєкти*;
в) здійснення CRUD-операцій над даними про *завдання*, які необхідно виконати для завершення проєкту;
г) призначення користувача, відповідального за виконання завдання;
д) здійснення CRUD-операцій над *звітами про помилки*, що мають включати в себе всі стандартні поля, наведені в розділі 1.1.2;
е) прикріплення зображень до звітів про помилки (максимум 3 зображення до одного звіту);
ж) призначення користувача, відповідального за виправлення помилки;
з) здійснення CRUD-операцій над всіма даними системи з використанням адмін-панелі;
и) відображення віртуальної Kanban-дошки з можливістю змінювати статус розміщених на ній елементів;
к) відображення статистики про завдання та помилки;
л) формування прогнозів щодо часу, який піде на виконання завдання або виправлення помилки, з використанням машинного навчання;
м) експорт завдань та звітів про помилки у формат CSV з наступним завантаженням відповідного файлу.

1.3.2 Опис ролей користувачів в системі

У даній системі відстежування помилок, що розробляється, користувачі можуть мати наступні ролі:

а) *адміністратор* – має найбільше прав в межах системи;

б) *гість* (неавторизований користувач) – не має доступу до системи;

в) *менеджер проєкту* – має найбільше прав в межах проєкту та здійснює керування процесом розробки;

г) *розробник* – працює з завданнями та помилками;

д) *тестувальник* – працює з помилками.

Висновки до розділу 1

В ході створення першого розділу була детально розглянута предмета область. Описано методологію розробки програмного забезпечення, яка буде використана в системі. Обрання методології допомогло визначити специфічні функціональні вимоги. Проаналізовано сутність тестування та атрибути звітів про помилки. Дані атрибути будуть використані при розробці моделі даних. Детально проаналізовано визначення, застосування та різновиди машинного навчання. На базі цього аналізу обрано алгоритм та структуру наборів даних для МН в системі.

Також проаналізовано наявні на ринку аналоги. Кожен аналог має свої переваги та недоліки. Визначено які саме недоліки аналогічного програмного забезпечення будуть виправлені та які переваги будуть взяті за зразок і впроваджені в систему. На інтерфейси аналогів можна спиратися при розробці макетів GUI.

В заключній частині розділу були описані функціональні вимоги до системи. На основі першого розділу можливо приступати до опису алгоритмів, які будуть застосовані при розробці системи.

2 АЛГОРИТМИ АНАЛІТИЧНОЇ ОБРОБКИ ДАНИХ

В даному розділі будуть описані алгоритми аналітичної обробки даних, що використовуються у системі відстеження помилок на основі машинного навчання. Для полегшення подальшого моделювання та розробки необхідно дослідити такі поняття: аналіз даних та дерева прийняття рішень.

2.1 Аналіз даних

2.1.1 Визначення аналізу даних

Мета **аналізу даних** – видобуток інформації. Після проведення аналізу легше зрозуміти роботу системи, що продукує ці дані, та здійснити припущення щодо її подальшої поведінки.

Всі методології аналізу даних спрямовані на побудову моделей. **Модель** представляє собою математичний опис системи. Основною її характеристикою є **передбачувальна сила** (*predictive power*). Дана характеристика залежить від точності моделі та набору даних, який був використаний для аналізу. Таким чином, процеси **пошуку** та **підготовки** даних мають велике значення.

Потужним інструментом є **візуалізація**: за допомогою графічного представлення можна зрозуміти склад та роль наборів даних, а також інтерпретувати результати проведеного аналізу [5].

2.1.2 Аналіз даних за допомогою pandas

Мета аналітичної обробки даних у системі відстеження помилок – отримання інформації про продуктивність роботи команди. Для досягнення поставленої мети буде проведено аналіз кількості завдань на кожному з етапів життєвого циклу як для команди взагалі, так і для кожного окремого розробника.

На рисунку 2.1 зображено етапи життєвого циклу елементів (як завдань, так і звітів про помилки).

Рисунок 2.1 – Життєвий цикл елементу

Для впровадження алгоритмів аналітичної обробки даних у систему буде використана бібліотека **pandas**, що містить відповідні структури даних та функції. Завдяки цій бібліотеці аналітична складова застосунка може бути розроблена швидко та ефективно. Є такі варіанти структур даних у pandas [6]:

а) *Series* – об'єкт, що складається з послідовності даних та асоційованого масиву з мітками (подібний до одновимірного масиву);

б) *DataFrame* – об'єкт, що складається з проіндексованих рядків та колонок (подібний до двовимірного масиву або таблиці).

Що стосується функцій, то для досягнення поставленої мети потрібно згрупувати дані за певними характеристиками. Для групування використовується функція **groupby**. Алгоритм її роботи простіше всього описати як послідовність *split-apply-combine*. Рисунок 2.2 наочно демонструє даний алгоритм.

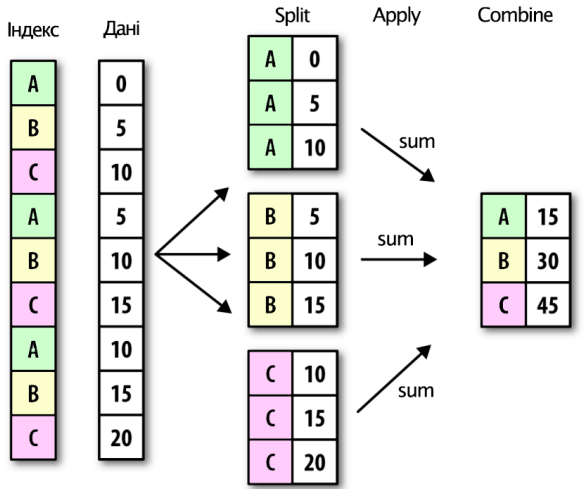

Рисунок 2.2 – Алгоритм роботи функції groupby

Отже, для досягнення поставленої мети групування у застосунку буде здійснено на основі особи розробника та статусу завдань. В результаті буде розрахована кількість завдань зі статусом *X* для розробника *Y*. Для групування доречно використати **DataFrame** в якості структури даних. Функція **groupby** буде застосована у поєднанні з **count**. Візуалізацію отриманої інформації краще всього зробити за допомогою **кругових діаграм**.

2.2 Дерева прийняття рішень

2.2.1 Визначення дерев прийняття рішень

Дерева прийняття рішень є універсальним алгоритмом машинного навчання, що застосовується для задач класифікації та регресії. Вказаний алгоритм можна використовувати з наборами даних будь-якої складності. Дерева прийняття рішень входять до випадкових лісів. Випадкові ліси, в свою чергу, є одним з найбільш потужних алгоритмів машинного навчання [4].

Нейронні мережі, наприклад, вважаються моделлю «чорного ящика», бо здогадатися про точну послідовність кроків цього алгоритму майже неможливо. Дерева прийняття рішень, навпаки, вважаються моделлю **«білого ящика»**: при їх застосуванні можливо вручну відтворити здійснені для формування прогнозу кроки [4]. Таким чином, обраний алгоритм є потужним та простим.

2.2.2 Дерева прийняття рішень в Scikit-learn

Машинне навчання буде впроваджене у застосунок за допомогою бібліотеки Scikit-learn, яка вже містить готову реалізацію обраного алгоритму.

Для пояснення принципу роботи дерев прийняття рішень у Scikit-learn варто скористатися прикладом [7]. На рисунку 2.3 маємо візуалізоване за допомогою **export_graphviz** дерево, що генерує прогноз музичних вподобань в залежності від статі та віку.

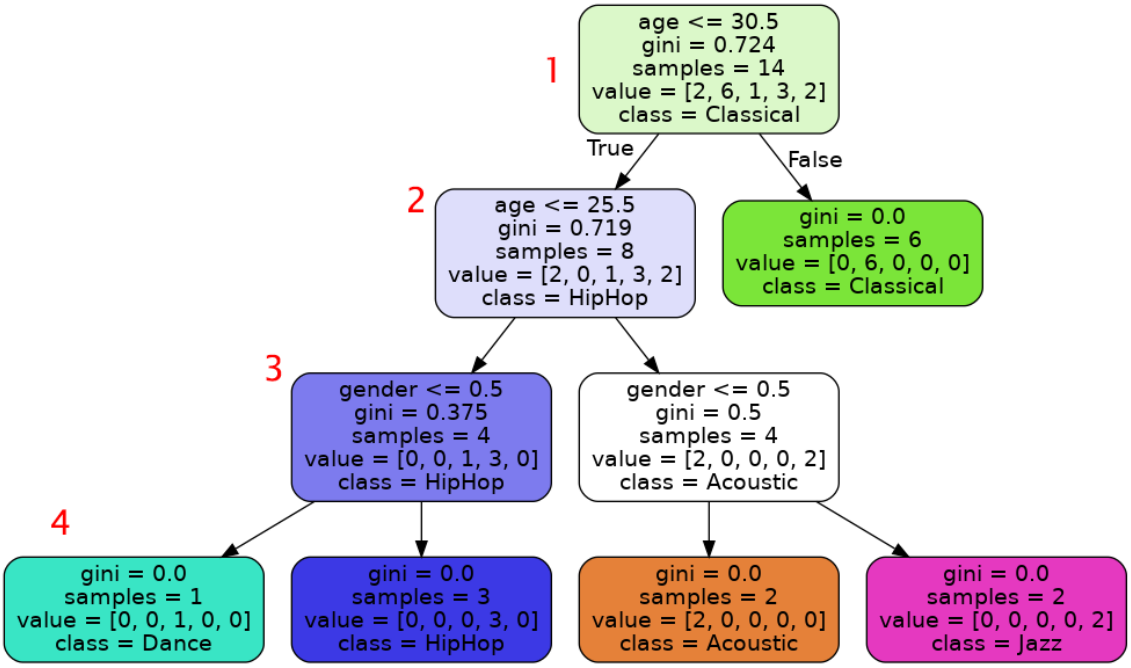

Рисунок 2.3 – Приклад дерева прийняття рішень

Отже, якщо вік становить 21 рік та стать є жіночою (жіноча стать позначена за допомогою 0, а чоловіча за допомогою 1), то проходимо наступні кроки:

а) крок №1: перевірка того, чи є вік меншим або рівним 30.5 років (True);

б) крок №2: перевірка того, чи є вік меншим або рівним 25.5 рокам (True);

в) крок №3: перевірка того, чи є стать жіночою (True);

г) крок №4: здійснення прогнозу щодо улюбленого жанру музики (в даному випадку це жанр «Dance»).

Побудоване дерево є **бінарним деревом**. Вузол з класом «Dance» є **листовим** (з нього не можна потрапити в інші вузли). Атрибут **samples** показує до скількох навчальних зразків застосовано даний вузол. Атрибут **gini** (показник Джині) показує ступінь забрудненості: вузол вважається чистим (значення gini становить 0) коли всі навчальні зразки, до яких він застосовується, належать до того самого класу [4].

У Scikit-learn використовується алгоритм **класифікацій та регресії (Classification And Regression Tree – CART)** для навчання дерева прийняття рішень. Даний алгоритм працює наступним чином: він ділить навчальний набір на дві частини на основі характеристики k та її граничного значення t_k. В наведеному вище прикладі (на першому кроці) характеристикою є вік, а граничним значенням є 30.5 років.

Алгоритм шукає пари (k, t_k), які продукують найбільш чисті набори, зважені за розміром. Він намагається мінімізувати функцію, яка описана формулою 2.1.

$$J(k, t_k) = \frac{m_{\text{лівий}}}{m} * G_{\text{лівий}} + \frac{m_{\text{правий}}}{m} * G_{\text{правий}} \quad (2.1)$$

де $m_{\text{лівий}}$ − кількість зразків у лівому наборі;

$m_{\text{правий}}$ − кількість зразків у правому наборі;

m – загальна кількість зразків;

$G_{\text{лівий}}$ – забрудненість лівого набору;

$G_{\text{правий}}$ – забрудненість правого набору;

Таким чином, алгоритм продовжує розділення до моменту досягнення максимальної глибини. Також він зупиниться тоді, коли більше не зможе скорочувати забрудненість наборів.

Варто зауважити, що вказаний алгоритм є **«жадібним»** − він шукає варіант оптимального розділення на верхньому рівні та повторює цей процес на кожному з нижчих рівнів. Дерева прийняття рішень продукують «достатньо хороший», але не найбільш оптимальний прогноз. Враховуючи специфіку системи, така точність є задовільно.

Навіть при роботі з великими наборами даних даний алгоритм є швидким, оскільки загальна складність прогнозу становить $O(\log_2 m)$ [4].

Що стосується загального плану впровадження машинного навчання в систему, то він виглядає наступним чином [7]:

а) завантаження набору даних;

б) підготовка (очищення) даних;

в) розділення набору даних на навчальний та тренувальний набір (найбільш вдале співвідношення: 70 − 80% набору використати для навчання та 30 − 20% для випробування);

г) створення моделі;

д) навчання моделі;

е) здійснення прогнозу;

ж) вимірювання точності та вдосконалення.

Отже, дерева прийняття рішень прості для інтерпретації та застосування, універсальні та потужні. Їх реалізація з бібліотеки Scikit-learn буде впроваджена в систему, що розробляється.

Висновки до розділу 2

В ході створення другого розділу проаналізовано алгоритми аналітичної обробки даних, що були обрані в попередньому розділі для вирішення задач системи.

Дано визначення та описано загальні принципи алгоритмів аналізу даних та машинного навчання. Проведене дослідження дозволило зрозуміти шляхи впровадження вказаних аспектів у систему. Також визначено функції та структури даних з бібліотек pandas та Scikit-learn, що будуть використані в системі.

Саме завдяки аналізу дані перетворюються на інформацію. Дерева прийняття рішень є простим та потужним алгоритмом, що використовується для розв’язку задач класифікації та регресії.

Отже, використовуючи матеріали першого та другого розділів можливо приступати до проєктування та детального моделювання системи відстеження помилок на основі машинного навчання.

3 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ СИСТЕМИ

Метою третього розділу є визначення архітектурних підходів та функціональної моделі системи відстеження помилок на основі машинного навчання. Також в даному розділі буде розроблена мета-модель даних системи.

3.1 Архітектура системи

Архітектура ПЗ – це спосіб проєктування та структурування обчислювальної системи. Патерни проєктування представляють собою перевірені досвідом багатьох розробників способи усунення розповсюджених недоліків у архітектурі програмного забезпечення [8].

Патерни проєктування надають спільну термінологію, яка підвищує ефективність співпраці у команді та полегшує читання і супровід коду. Знаючи, що система базується, наприклад, на MVC, легко розібратися в її структурі та зробити припущення щодо логіки взаємодії різних її частин.

Всі патерни проєктування намагаються наблизити архітектуру ПЗ до відповідності хоча би деяким з наступних принципів [8]:

а) інкапсуляція змін;
б) надання переваги композиції перед наслідуванням;
в) програмування на рівні інтерфейсів;
г) слабка зв’язаність взаємодіючих об’єктів;
д) відкритість до розширення та закритість до змін;
е) код має залежати від абстракцій, а не від конкретних класів;
ж) клас може мати лише одну причину для змін;
з) мінімальна інформованість (обмежити взаємодію між об’єктами до кількох близьких «друзів»);
и) голлівудський принцип («не викликайте нас – ми вас самі викличемо», тобто не можна допустити заплутаність архітектури через велику кількість складних залежностей).

Отже, використання патернів сприяє досягненню головної мети архітектури програмного забезпечення – зменшення трудовитрат на створення та супровід системи [9]. Таким чином, патерни проєктування є своєрідними алгоритмами розробки ПЗ.

Model-View-Controller (MVC) – це патерн проєктування, що передбачає організацію коду таким способом, щоб дані та логіка застосунка були розподілені між 3 компонентами: моделлю, контролером та представленнями. Це *складений* патерн, тобто він представляє собою поєднання декількох інших патернів в єдину систему. У таблиці 3.1 наведено опис складових MVC.

Таблиця 3.1 – Складові MVC

Складова	**Патерн**
Модель (дані)	*Спостерігач*. Сповіщає про зміну стану без формування сильної зв’язаності.
Представлення (візуальне відображення)	*Компонувальник*. Використовується для реалізації користувацького інтерфейсу, який містить ієрархію компонентів.
Контролер (посередник між моделлю та представленням)	*Стратегія*. Завдяки цьому патерну представлення може обирати різні варіанти реалізації для забезпечення гнучкості поведінки.

На рисунку 3.1 проілюстровано принцип роботи патерну MVC.

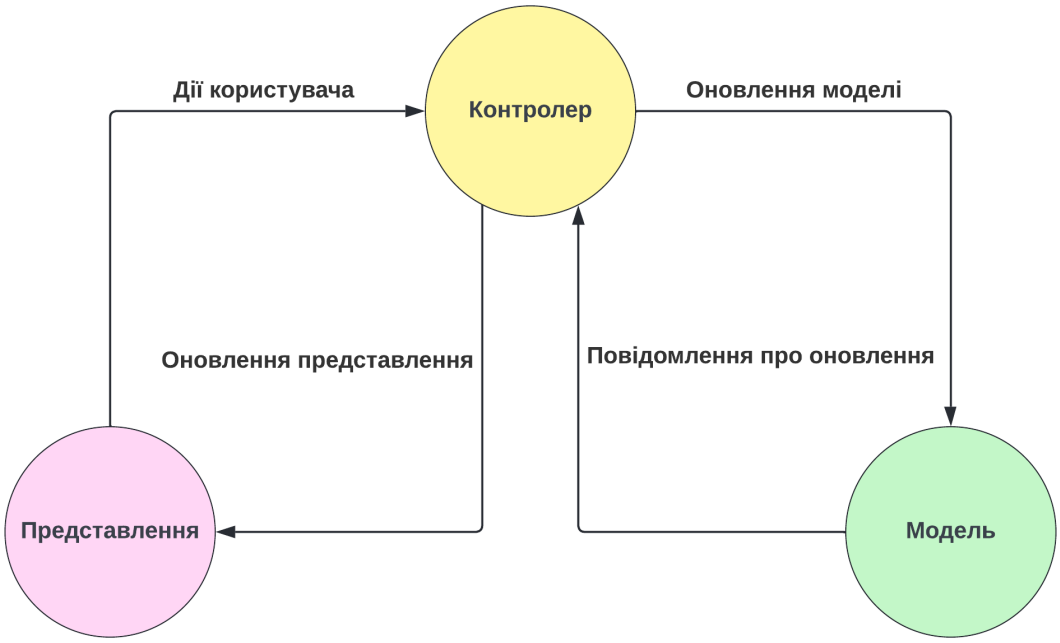

Рисунок 3.1 – Принцип роботи патерну MVC

Завдяки використанню MVC в системі забезпечена слабка зв’язаність компонентів та гнучкість архітектури.

3.2 Функціональна модель системи

На базі здійсненої в першому розділі специфікації вимог можливо розробити сценарії використання системи. Сценарій використання описує відповідь системи на запити різних категорій користувачів (акторів).

У таблицях 3.2 – 3.12 здійснено детальний опис сценаріїв використання системи відстеження помилок на основі машинного навчання.

Таблиця 3.2 – Сценарій використання UC-01

UC-01: Реєстрація	
Діючі особи	Гість та система.
Мета	Створити обліковий запис.
Передумова	Гість неавторизований.
Успішний сценарій: 1. Гість переходить до сторінки реєстрації. 2. Гість заповнює форму. 3. Система здійснює валідацію даних. 4. Після збереження облікового запису відбувається автоматизована авторизація користувача в системі. 5. Авторизований користувач перенаправляється на головну сторінку.	
Результат	Збережено новий обліковий запис.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може створити новий обліковий запис.
1a	Гість ввів у форму реєстрації недопустимі дані. **Результат:** система виводить повідомлення про помилку та надає повторну можливість створити обліковий запис (кількість спроб необмежена).

Таблиця 3.3 – Сценарій використання UC-02

UC-02: Авторизація	
Діючі особи	Гість та система.
Мета	Авторизація в системі.
Передумова	Обліковий запис існує в системі. Авторизація не була здійснена раніше.
Успішний сценарій: 1. Гість переходить до сторінки авторизації. 2. Гість заповнює форму. 3. Система здійснює валідацію даних. 4. Авторизований користувач перенаправляється на головну сторінку.	
Результат	Гість авторизувався в системі.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може авторизуватися в системі.
1a	Гість ввів в форму авторизації недопустимі дані. **Результат:** система виводить повідомлення про помилку авторизації та надає можливість повторно спробувати увійти в обліковий запис (кількість спроб необмежена).

Таблиця 3.4 – Сценарій використання UC-03

UC-03: Отримання прогнозів щодо розробки	
Діючі особи	Користувач та система.
Мета	Отримання прогнозу щодо часу, який піде на виконання завдання або виправлення помилки.
Передумова	У системі достатньо даних для навчання моделі.
Успішний сценарій: 1. Користувач переходить до сторінки з завданням (помилкою). 2. Користувач натискає кнопку «Predict time». 3. Система здійснює прогнозування. 4. Система відображає прогноз на сторінці.	
Результат	Користувач отримав прогноз щодо розробки.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може отримати прогноз щодо розробки.
1a	Система не має зразків виконання завдань та виправлення помилок. **Результат:** система виводить попередження про те, що для прогнозування недостатньо даних.

Таблиця 3.5 – Сценарій використання UC-04

UC-04: Перегляд статистики	
Діючі особи	Користувач та система.
Мета	Отримання статистики про процес розробки.
Передумова	Користувач переглянув статистику.
Успішний сценарій: 1. Користувач обирає один з проєктів, до яких має доступ. 2. Користувач переходить до сторінки зі списком завдань або списком звітів про помилки. 3. Користувач переходить до сторінки зі статистикою. 4. Система здійснює аналіз даних. 5. Система відображає статистику у вигляді діаграм.	
Результат	Користувач здійснив перегляд статистики.
Розширення	
***а**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може переглянути статистику.

Таблиця 3.6 – Сценарій використання UC-05

UC-05: Експорт звітів про помилки у формат CSV	
Діючі особи	Користувач та система.
Мета	Збереження файлу .csv, що містить всі звіти про помилки з обраного проєкту.
Передумова	Користувач авторизований в системі.
Успішний сценарій: 1. Користувач обирає один з проєктів, до яких має доступ. 2. Користувач переходить до сторінки зі списком звітів про помилки. 3. Користувач натискає кнопку «Load CSV» 4. Система генерує файл .csv зі списком звітів про помилки. 5. Користувач завантажує файл.	
Результат	Здійснено експорт звітів про помилки у формат CSV.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може завантажити файл .csv зі списком звітів про помилки.

Таблиця 3.7 – Сценарій використання UC-06

UC-06: Експорт завдань у формат CSV	
Діючі особи	Користувач та система.
Мета	Збереження файлу .csv, що містить всі завдання з обраного проєкту.
Передумова	Користувач авторизований в системі.
Успішний сценарій: 1. Користувач обирає один з проєктів, до яких має доступ. 2. Користувач переходить до сторінки зі списком завдань. 3. Користувач натискає кнопку «Load CSV». 4. Система генерує файл .csv зі списком завдань. 5. Користувач завантажує файл.	
Результат	Здійснено експорт завдань у формат CSV.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** користувач не може завантажити файл .csv зі списком завдань.

Таблиця 3.8 – Сценарій використання UC-07

UC-07: Управління статусом завдань та помилок	
Діючі особи	Розробник та система.
Мета	Зміна статусу завдань та помилок.
Передумова	Розробник авторизований в системі.
Успішний сценарій: 1. Розробник обирає один з проєктів, до яких має доступ. 2. Розробник переходить до сторінки з віртуальною Kanban-дошкою. 3. Розробник обирає завдання або помилку. 4. Розробник перетягує завдання або помилку з однієї колонки в іншу. 5. Система змінює статус завдання або помилки. 6. Система відображає зміни на Kanban-дошці.	
Результат	Здійснено управління статусом завдань та помилок.
Розширення	
***а**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** розробник не може управляти статусом завдань та помилок.
б	Розробник перетягнув елемент (завдання або помилку) не в колонку віртуальної Kanban-дошки, а в будь-яку іншу область сторінки. **Результат:** у елемента не змінюється статус.

Таблиця 3.9 – Сценарій використання UC-08

UC-08: Управління звітами про помилки	
Діючі особи	Тестувальник та система.
Мета	Здійснення CRUD-операцій над звітами про помилки.
Передумова	Тестувальник авторизований в системі..
Успішний сценарій: 1. Тестувальник обирає один з проєктів, до яких має доступ. 2. Тестувальник переходить до сторінки зі звітами про помилки. 3. Тестувальник вносить зміни в звіти про помилки. 4. Система здійснює валідацію внесених даних. 5. Система зберігає зміни. 6. Система відображує оновлену сторінку зі звітами про помилки.	
Результат	Здійснено управління звітами про помилки.
Розширення	
***а**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** розробник не може управляти статусом завдань та помилок.
1а	Тестувальник намагається зберегти недопустимі дані. **Результат:** система виводить повідомлення про відповідну помилку та дає можливість внести виправлення та повторно зберегти дані (кількість спроб не обмежена).

Таблиця 3.10 – Сценарій використання UC-09

UC-09: Управління даними про завдання	
Діючі особи	Менеджер та система.
Мета	Здійснення CRUD-операцій над даними про помилки.
Передумова	Менеджер авторизований в системі.
Успішний сценарій: 1. Менеджер обирає один з проєктів, до яких має доступ. 2. Менеджер переходить до сторінки зі списком завдань. 3. Менеджер вносить зміни в дані про завдання. 4. Система здійснює валідацію внесених даних. 5. Система зберігає зміни. 6. Система відображає оновлену сторінку зі списком завдань.	
Результат	Здійснено управління даними про завдання.
Розширення	
***а**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** менеджер не може управляти даними про завдання.
1а	Менеджер намагається зберегти недопустимі дані **Результат:** система виводить повідомлення про відповідну помилку та дає можливість внести виправлення та повторно зберегти дані (кількість спроб не обмежена).

Таблиця 3.11 – Сценарій використання UC-10

UC-10: Управління даними про проєкти	
Діючі особи	Менеджер та система.
Мета	Здійснення CRUD-операцій над даними про проєкти.
Передумова	Менеджер авторизований в системі.
Успішний сценарій: 1. Менеджер обирає один з проєктів, до яких має доступ. 2. Менеджер обирає CRUD-операцію. 3. Менеджер змінює (створює, редагує або видаляє) дані про проєкт. 4. Система здійснює валідацію внесених даних. 5. Система зберігає зміни. 6. Система відображає оновлену сторінку зі списком доступних проєктів.	
Результат	Здійснено управління даними про проєкти.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** менеджер не може управляти даними про проєкти.
1a	Менеджер намагається зберегти недопустимі дані **Результат:** система виводить повідомлення про відповідну помилку та дає можливість внести виправлення та повторно зберегти дані (кількість спроб не обмежена).

Таблиця 3.12 – Сценарій використання UC-11

UC-11: Управління даними системи	
Діючі особи	Адміністратор та система.
Мета	Здійснення CRUD-операцій над усіма даними системи.
Передумова	Адміністратор авторизований в системі.
Успішний сценарій: 1. Адміністратор переходи за адресою <URL застосунка>/**admin**. 2. Адміністратор обирає дані, які необхідно змінити. 3. Адміністратор змінює (створює, редагує або видаляє) дані. 4. Система здійснює валідацію внесених даних. 5. Система зберігає зміни. 6. Система відображує оновлені дані.	
Результат	Здійснено управління даними системи.
Розширення	
***a**	Немає доступу до БД. Система відображає відповідне повідомлення. **Результат:** адміністратор не може управляти даними системи.
1a	Адміністратор намагається зберегти недопустимі дані. **Результат:** система виводить повідомлення про відповідну помилку та дає можливість внести виправлення та повторно зберегти дані (кількість спроб не обмежена).

На рисунку 3.2 зображена діаграма варіантів використання системи відстеження помилок на основі машинного навчання.

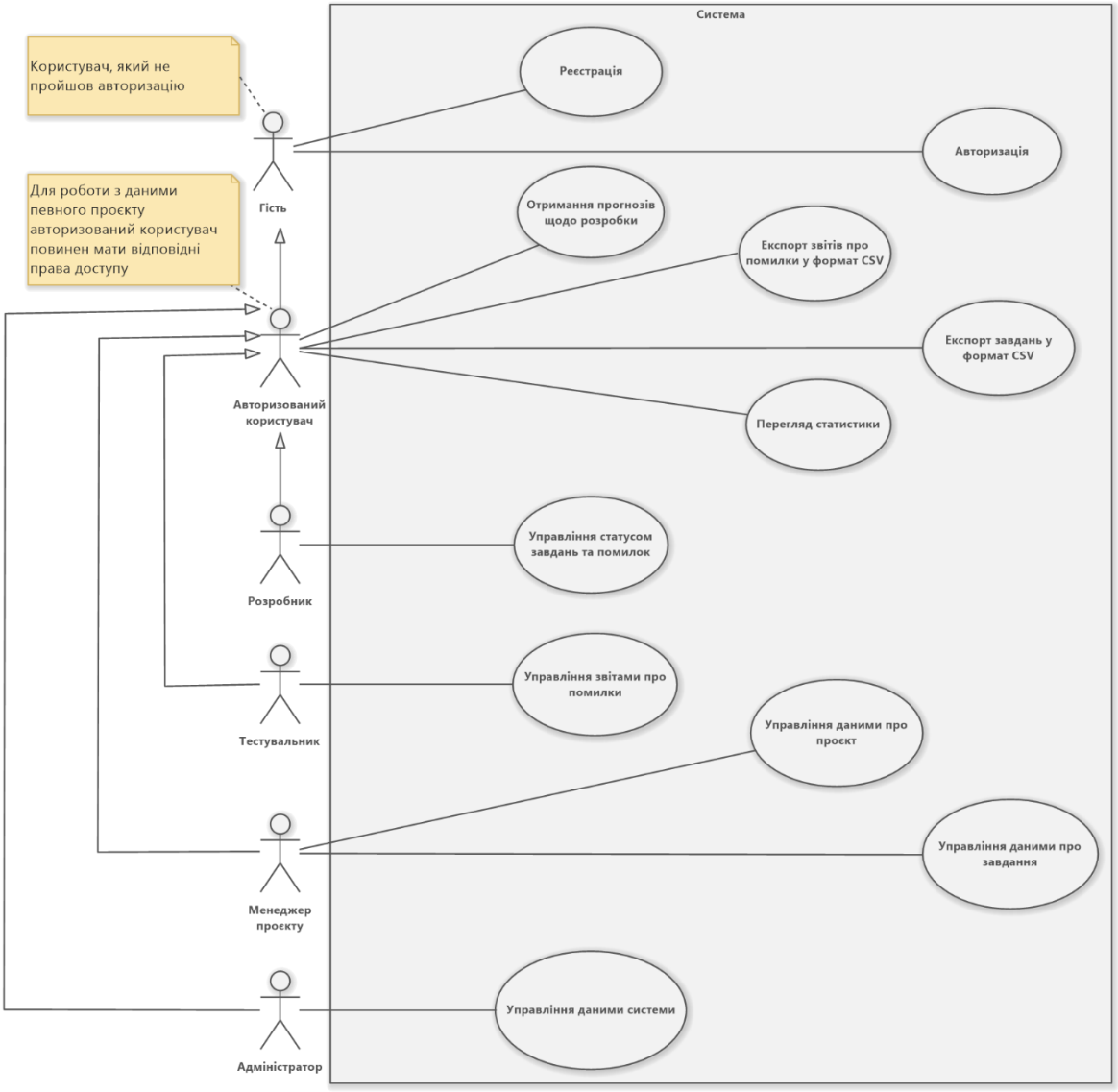

Рисунок 3.2 – Діаграма варіантів використання системи

На основі діаграми варіантів використання легко розподіляти права доступу та впорядковувати функціональні вимоги.

3.3 Модель даних системи

Діаграма «сутність-зв’язок» (Entity Relationship Diagram − **ERD**) представляє собою мета-модель даних. На рисунку 3.3 зображена ER-діаграма системи.

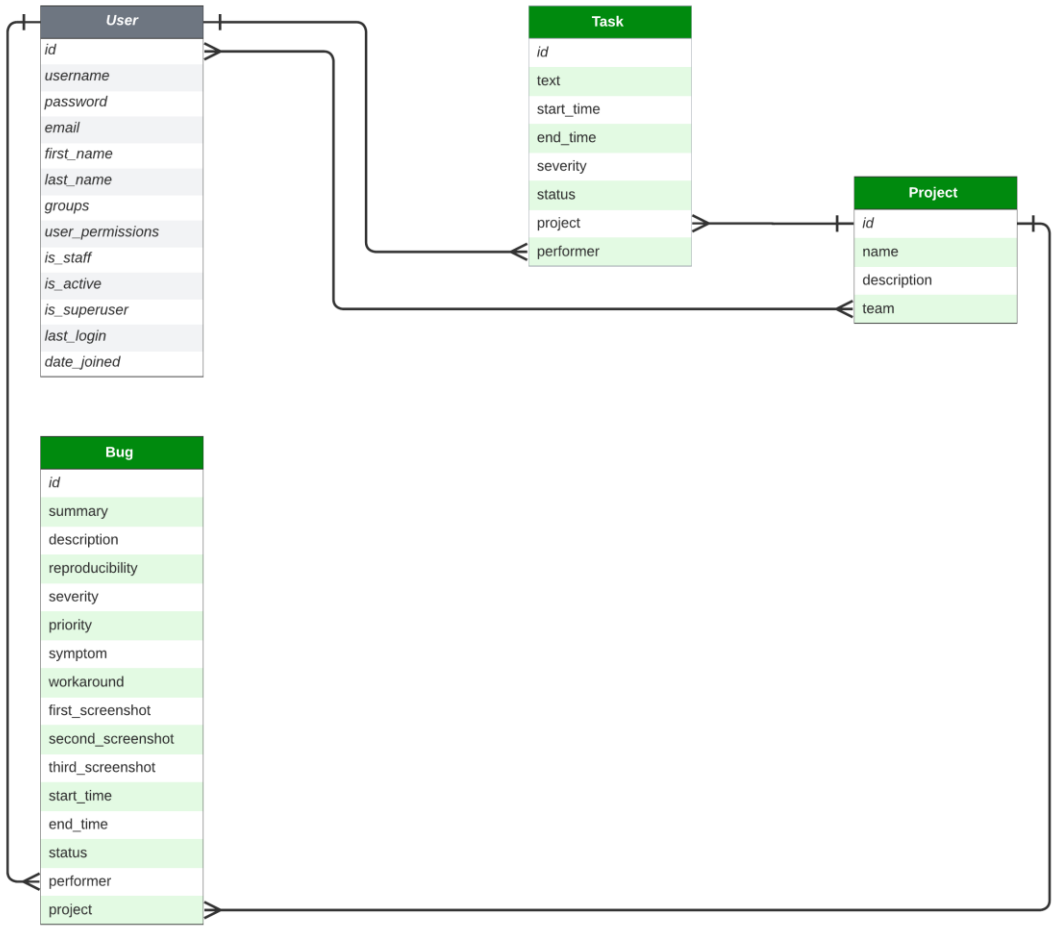

Рисунок 3.3 – ER діаграма системи

Отже, маємо наступні сутності:

а) *User*. Зберігає дані про користувача. Має стандартний набір полів, визначений фреймворком Django.

б) *Task.* Зберігає дані про завдання.

в) *Bug.* Зберігає звіт про помилку. Побудовано на основі інформації про тестування ПЗ з розділу 1.

г) *Project.* Зберігає дані про проєкти. Зв’язує всі інші сутності між собою.

На основі полів *start_time* та *end_time* можна визначити час, який пішов на виконання завдання або виправлення помилки. Поле *id* генерується в Django автоматично, його не потрібно прописувати явно в класі моделі.

Отже, ER-діаграма буде використана для реалізації класів моделей, на основі яких (з використанням вбудованої у Django ORM) можливо автоматично згенерувати схему БД та отримувати доступ до даних.

База даних у системі є нормалізованою. Нормалізація представляє собою техніку організації даних у вигляді декількох взаємопов'язаних таблиць у такий спосіб, щоб кількість повторів була мінімальною. Повторення даних призводить до того, що БД займає більший об'єм пам'яті. Крім того, повторювані дані важко підтримувати в актуальному стані – виникають проблеми при додаванні, видаленні та редагуванні. Отже, розроблена база даних перебуває у 3НФ. У таблиці 3.13 наведено правила нормалізації.

Таблиця 3.13 – Нормалізація БД

Нормальна форма (НФ)	**Правила**
1НФ	Кожна колонка таблиці має містити атомарні значення одного типу та мати унікальний заголовок. Порядок, у якому зберігаються дані, не має значення.
2НФ	У таблиці не має бути *partial dependency* (залежності атрибутів від частини складеного первинного ключа).
3НФ	У таблиці не має бути *transitive dependency* (залежність атрибутів від іншого атрибута, що не є ключем).

Висновки до розділу 3

Поставлена мета була досягнута, оскільки описано архітектуру, модель даних та сценарії використання системи відстеження помилок на основі машинного навчання. Також виконано нормалізацію бази даних та проаналізовано патерн проєктування MVC. Отже, на основі третього розділу можливо приступати до реалізації вебзастосунку.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метою четвертого розділу є обґрунтування вибору технологій, опис основних кроків реалізації (налаштування, структура проєкту та код) і створення керівництва користувача.

4.1 Обґрунтування вибору технологій для реалізації системи

Для розробки серверної частини вебзастосунку буде використано **Python**. Вказана мова програмування представляє собою високорівневу та платформно-незалежну мову загального призначення. Python було обрано через те, що він дозволяє ефективно впровадити у вебзастосунок машинне навчання та аналіз даних. Таким чином, у системі поєднуються переваги використання потужної математичної та аналітичної складової з загальними перевагами вебзастосунку (зручний GUI та існування ПЗ у вигляді сервісу в мережі Інтернет). Логотип Python зображений на рисунку 4.1.

Рисунок 4.1 – Логотип Python

Клієнтська частина вебзастосунку реалізована за допомогою **HTML**, **CSS**, **Sass** та **JavaScript**. Препроцесор Sass використовується для перевизначення змінних фреймворку Bootstrap та ефективного написання користувацьких стилів. Шрифт взято з бібліотеки **Google Fonts**. Піктограми для кнопок та посилань взято з бібліотеки **Font Awesome**.

В якості СКБД обрано **SQLite**. Оскільки таблиці створюються за допомогою міграцій та керування даними відбувається з використанням ORM, то тип СКБД не впливає на код.

У системі будуть застосовані наступні фреймворки та бібліотеки:

а) **Django** (клієнтська та серверна частина вебзастосунку, інтеграція з машинним навчанням);

б) **Bootstrap** (стилізація сторінок);

в) **Chart.js** (відображення діаграм);

г) **pandas** (аналіз даних);

д) **Scikit-learn** (машинне навчання).

Django представляє собою *full stack* фреймворк, тобто він надає ефективну та просту функціональність для поєднання клієнтської (*frontend*) та серверної (*backend*) частин вебзастосунку. Для інтеграції з машинним навчанням та аналізом даних необхідно імпортувати відповідні бібліотеки. Фреймворк Django базується на наступних принципах [10]:

а) низька зв’язаність і високе зачеплення;

б) мінімальна кількість коду;

в) швидка розробка;

г) DRY (don`t repeat yourself);

д) явне краще, ніж приховане;

е) послідовність та логічність.

Використання бібліотек та фреймворків дозволяє скоротити код та зробити його більш читабельним. Немає сенсу самостійно писати функції, які вже мають ефективну, відлагоджену та протестовану реалізацію. Крім того, наведені фреймворки та бібліотеки супроводжуються вичерпною документацією. Завдяки такому підходу іншим розробникам буде легко читати, розширювати та повторно використовувати код системи, що розробляється.

4.2 Опис програмної реалізації

4.2.1 Загальна структура проєкту

Реалізацією системи, моделювання якої було здійснено в попередньому розділі, є проєкт на базі фреймворку **Django**. На рисунку 4.2 представлено загальну структуру створеного проєкту.

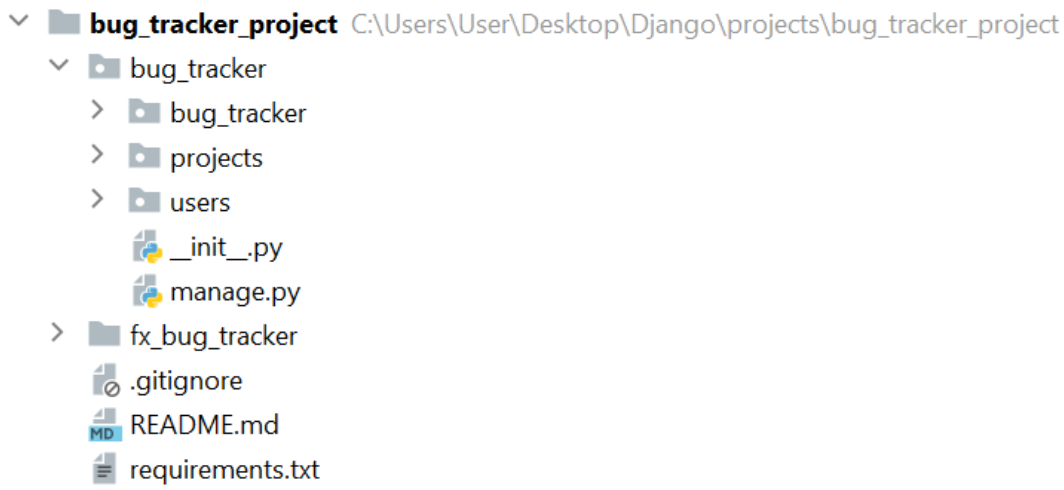

Рисунок 4.2 – Загальна структура проєкту

Призначення складових:

а) застосунок *«bug_tracker»* містить модулі та налаштування, що стосуються всього проєкту та описують його конфігурацію;

б) застосунок *«users»* відповідає за авторизацію та реєстрацію;

в) застосунок *«projects»* відповідає за роботу з проєктами, завданнями та звітами про помилки;

г) директорія *«fx_bug_tracker»* містить встановлені за допомогою webpack препроцесор Sass та фреймворк Bootstrap [11];

д) у файлі *«README.md»* наведена інструкція для встановлення застосунка (на випадок командної роботи над кодом системи);

е) у файлі *«.gitignore»* перераховано файли та директорії, для яких не передбачено відстежування системою контролю версій Git;

ж) у файлі *«requirements.txt»* міститься список всіх пакетів Python, що необхідні для коректної роботи розробленого вебзастосунку;

з) файл *«manage.py»* містить утиліту, що дозволяє виконувати різні дії над проєктом (наприклад, створення міграцій і запуск сервера).

4.2.2 Застосунок «bug_tracker»

Застосунок *«bug_tracker»* містить налаштування, статичні файли і шаблони, загальні для всього проєкту. Загальними шаблонами в даному випадку є шаблон *«404.html»* (використовується у випадку помилки з кодом 404) та шаблон *«base.html»*, який наслідують всі інші сторінки системи. Загальна структура застосунку *«bug_tracker»* зображена на рисунку 4.3.

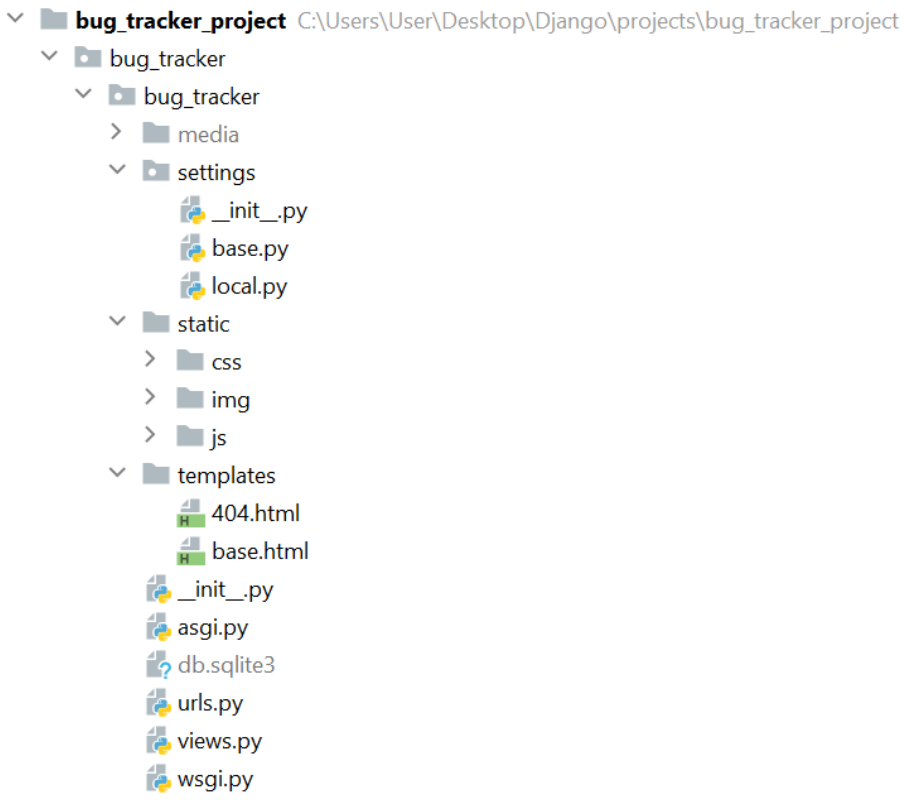

Рисунок 4.3 – Структура застосунку *«bug_tracker»*

Базові та користувацькі налаштування містяться у файлі *«base.py»*. Для запуску системи на локальній машині необхідно використати файл *«local.py»*, який наслідує та розширює загальні налаштування з *«base.py»*. Заради безпеки у налаштуваннях встановлено наступні значення:

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = getenv('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False
```

Будь-який проєкт на Django є поєднанням декількох застосунків. Застосунки можливо самостійно розробити або завантажити за допомогою системи керування пакунками **pip**. У налаштуваннях до розробленої системи відстеження помилок включено наступні застосунки:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    "users.apps.UsersConfig",
    "projects.apps.ProjectsConfig",
    "crispy_forms",
    "django_cleanup.apps.CleanupConfig",
]
```

Призначення включених застосунків:

а) «admin» – відповідає за адміністрування (вбудований);

б) «auth» – відповідає за авторизацію (вбудований);

в) «contenttypes» – відповідає за роботу з моделями (вбудований);

г) «sessions» – відповідає за сесії (вбудований);

д) «messages» – відповідає за відправку повідомлень (вбудований);

е) «staticfiles» – відповідає за взаємодію зі статичними файлами (вбудований);

ж) «users» – розширює та змінює вбудовану авторизацію, реєстрацію та CRUD-операції над моделлю User (самостійно розроблений);

з) «projects» – відповідає за машинне навчання, аналіз даних та CRUD-операції над даними про проєкти, помилки і завдання (самостійно розроблений);

и) *«crispy_forms»* – використовується для стилізації згенерованих на основі моделей форм (додатково завантажений);

к) *«cleanup»* – дозволяє автоматизувати процес видалення та оновлення файлів і зображень, пов'язаних з моделлю (додатково завантажений).

4.2.3 Застосунок «projects»

Структура застосунку *«projects»* зображена на рисунку 4.4.

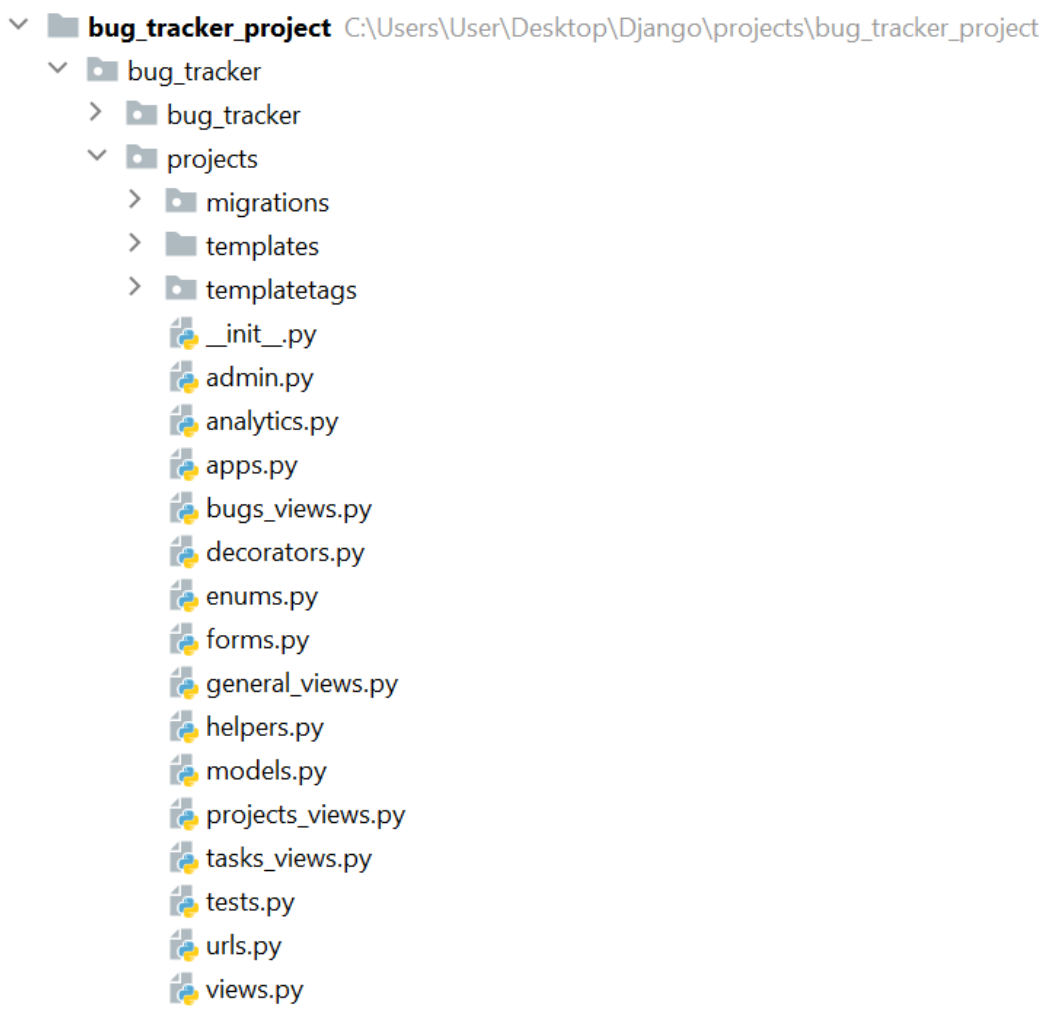

Рисунок 4.4 – Структура застосунку *«projects»*

Структура, яку фреймворк Django створює за замовчуванням, була розширена наступними користувацькими файлами:

а) файл *«analytics.py»* містить код, що відповідає за машинне навчання та аналіз даних;

б) файл *«bugs_views.py»* представляє собою контролер для моделі звіту про помилку (Bug);

в) файл *«decorators.py»* містить реалізацію декораторів, за допомогою яких обмежується доступ користувачів до певної функціональності системи в залежності від ролі;

г) файл *«enums.py»* містить значення, які використовуються в якості варіантів вибору у моделях;

д) файл *«general_views.py»* представляє собою контролер для статистики та віртуальної Kanban-дошки;

е) файл *«helpers.py»* містить функції, що допомагають працювати з об’єктами з файлу *«enums.py»*;

ж) файл *«projects_views.py»* представляє собою контролер для моделі проєкту (Project);

з) файл *«tasks_views.py»* представляє собою контролер для моделі завдання (Task).

Згідно до концепції патерну MVC *«models.py»* містить моделі, *«templates»* містить представлення, а *«views.py»* є контролером.

Для призначення ролі використовується клас Group, який за замовчуванням вбудований у фреймворк Django. Отже, зареєстрований користувача додається в групу «manager», «developer» або «tester» і на основі своєї групи отримує права доступу.

Декоратори з файлу *«decorators.py»* представляють собою функції, що приймають в якості параметру інші функції та додають їм нові можливості і властивості [12]. В даному випадку – додають перевірку ролі (групи) авторизованого користувача.

Код декоратора, що перевіряє права доступу користувачів, виглядає наступним чином:

```
from django.shortcuts import redirect


def allowed_users(allowed_roles=[]):
    def decorator(view_func):
        def wrapper_func(request, *args, **kwargs):
            group = None
            if request.user.groups.exists():
                group = request.user.groups.all()[0].name
            if group in allowed_roles:
                return view_func(request, *args, **kwargs)
            else:
                return redirect("/restricted")
        return wrapper_func
    return decorator
```

Приклад використання створеного декоратора з функцією для видалення завдання (лише менеджер може видаляти завдання):

```
@login_required(login_url="/users/login")
@allowed_users(allowed_roles=["manager"])
def delete_task(request, project_id, task_id):
    Task.objects.filter(id=task_id).delete()
    return redirect("/tasks/" + str(project_id))
```

У додатку А наведено вміст файлу *«analytics.py»*. Наведені у даному файлі функції відповідають за обробку даних, статистику та машинне навчання. Враховуючи незначний об’єм даних (завершені завдання та виправлені помилки у певному проєкті), доцільніше щоразу створювати нову модель разом з викликом функції, а не зберігати її. Вказані функції написані мовою **Python**.

У додатку Б міститься код, що відповідає за перетягування елементів по віртуальній Kanban-дошці з відповідною зміною статусу. Даний код написаний мовою **JavaScript**.

У додатку В міститься код сторінки, що відображає статистику (як для завдань, так і для помилок). Відповідний скрипт написаний мовою **JavaScript**.

4.2.4 Застосунок «users»

На рисунку 4.5 зображено структуру застосунку *«users»*.

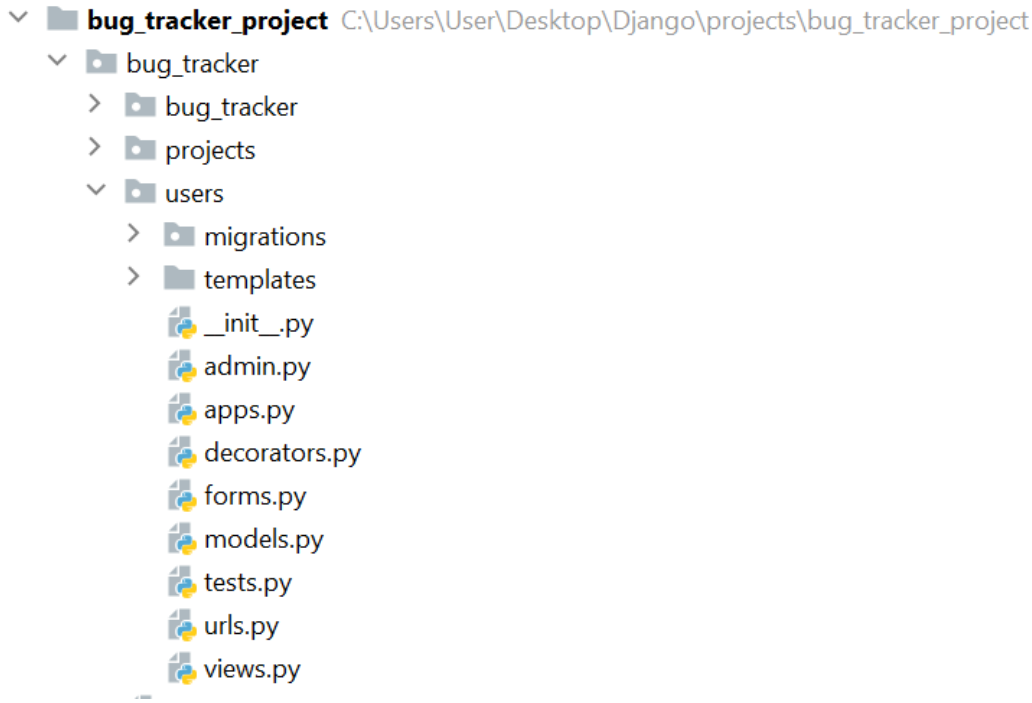

Рисунок 4.5 – Структура застосунку *«users»*

Весь код, необхідний для авторизації та реєстрації, генерується за замовчуванням. У даній системі необхідно створити користувацькі шаблони для сторінок авторизації, реєстрації та виходу з облікового запису.

До сторінок з реєстрацією та авторизацією може мати доступ лише неавторизований користувач. Код відповідного декоратора:

```
def unauthenticated_user(view_func):
    def wrapper_func(request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect("/")
        else:
            return view_func(request, *args, **kwargs)
    return wrapper_func
```

Створений декоратор застосовується наступним чином (для сторінки реєстрації):

```
@unauthenticated_user
def register(request):
    if request.POST:
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            group = Group.objects.get(name=request.POST["role"])
            user.groups.add(group)
            login(request, user)
            return redirect("/")
    else:
        form = RegisterForm()

    return render(
        request,
        "users/register.html",
        {
            "form": form
        }
    )
```

Крім того, у системі використовується вбудований декоратор «login_required», що забороняє доступ для неавторизованого користувача та перенаправляє на сторінку авторизації в разі потреби.

Якщо користувач спробує отримати доступ до певної сторінки через URL, то лише декоратор може обробити цю ситуацію. Просте приховування заборонених посилань не є ефективним підходом до вирішення проблеми з розмежуванням доступу.

Для адміністрування використовується вбудована у Django адмін-панель, до якої можливо перейти за відносним посиланням **/admin**.

Отже, в ході програмної реалізації системи було створено проєкт, що відповідає розробленим у попередніх розділах вимогам та сценаріям використання. Написаний код є ефективним та читабельним. Інтеграція всіх бібліотек та фреймворків була успішною. Вебзастосунок пройшов вдале тестування.

4.3 Керівництво користувача

4.3.1 Авторизація та реєстрація

Авторизація вимагається для переходу до будь-якої сторінки вебзастосунку. На рисунку 4.6 зображено форму авторизації.

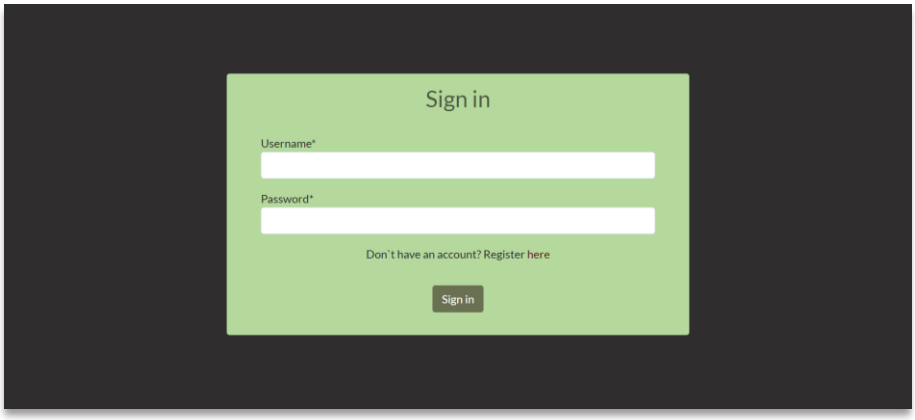

Рисунок 4.6 – Форма авторизації

На даній формі знаходяться поля «Username» та «Password», які необхідно заповнити. Для авторизації потрібно натиснути кнопку з написом «Sign in». Якщо у користувача ще немає облікового запису, то можна перейти до форми реєстрації за відповідним посиланням («Register here»).

На рисунку 4.7 зображено форму авторизації у випадку помилки. Якщо введено неправильний логін або пароль, то виводиться повідомлення «There was a problem with your login».

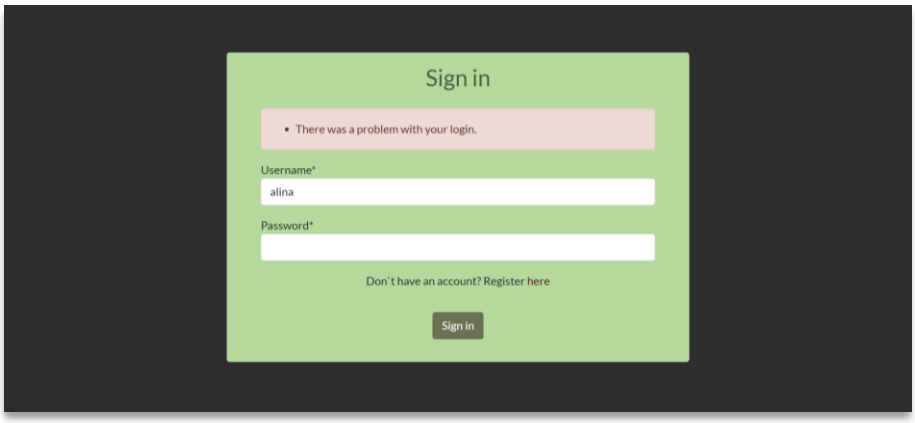

Рисунок 4.7 – Відображення помилок авторизації

На рисунку 4.8 зображена форма реєстрації користувача.

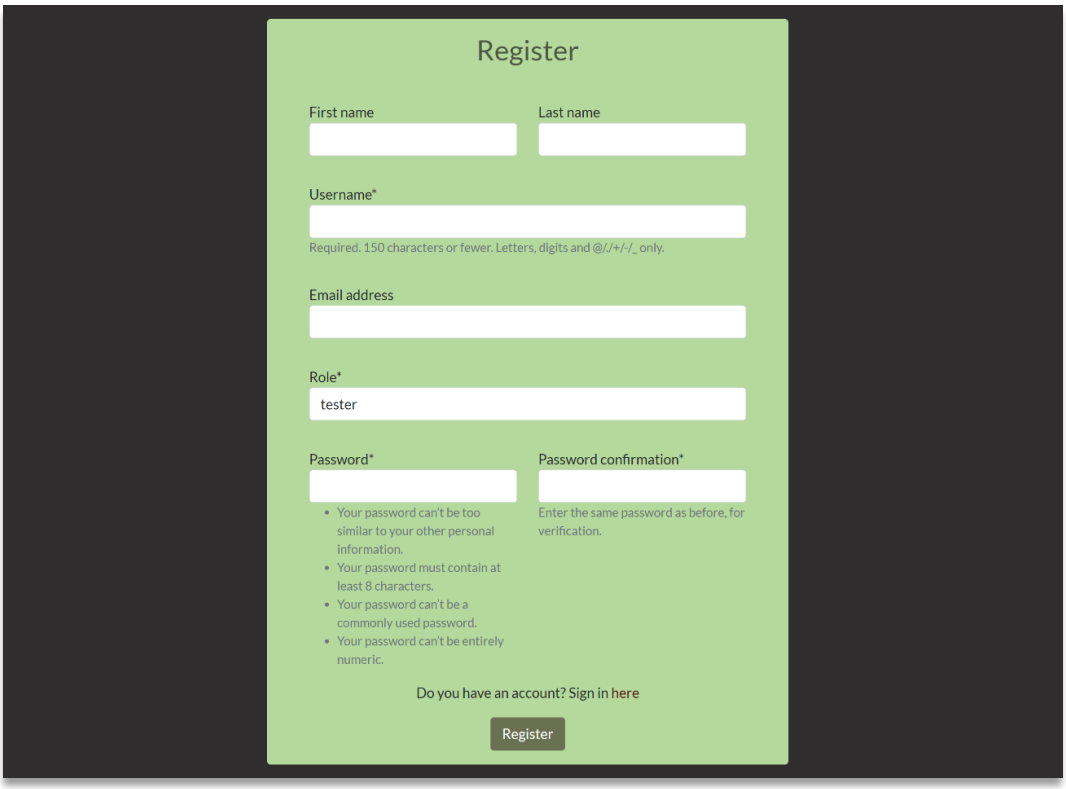

Рисунок 4.8 – Форма реєстрації

Для того, щоб зареєструватися, необхідно обов’язково заповнити наступні поля:

а) First name (ім’я користувача);

б) Last name (прізвище користувача);

в) Username (логін);

г) Email address (адреса електронної пошти);

д) Role (обрати роль – «tester», «manager» або «developer»);

е) Password (створити пароль);

ж) Password confirmation (підтвердити створений пароль).

Для реєстрації облікового запису необхідно натиснути кнопку з написом «Register». Крім того, є можливість повернутися до форми авторизації за відповідним посиланням («Sign in here»). У випадку введення недопустимих даних виводяться відповідні повідомлення.

4.3.2 Робота з даними про проєкти

Після авторизації користувач потрапляє на головну сторінку, де виводиться список проєктів, до яких він має доступ. На рисунку 4.9 зображено приклад головної сторінки для менеджера.

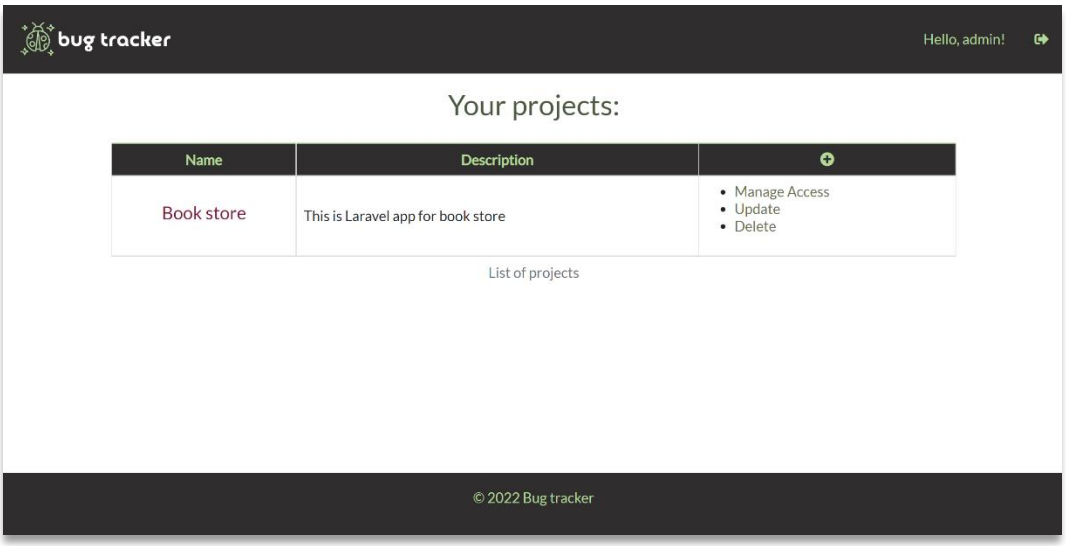

Рисунок 4.9 – Вигляд головної сторінки для менеджера

Крім перегляду колонок з назвою та описом, менеджер має доступ до посилань для створення, видалення та зміни проєкту. Також лише менеджер може управляти доступом користувачів до проєкту.

Якщо у користувача немає жодного проєкту, то замість таблиці виводиться повідомлення «No projects yet…». На рисунку 4.10 зображено фрагмент головної сторінки без проєктів.

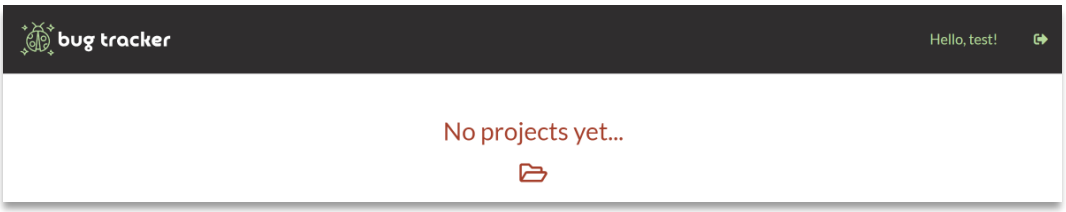

Рисунок 4.10 – Вигляд головної сторінки без проєктів

Управління доступом до проєкту здійснюється за допомогою спеціальної форми, зображеної на рисунку 4.11. У полі для вибору «Add member» можливо обрати тих користувачів, кому буде надано доступ до проєкту. Зберегти зміни можливо натиснувши кнопку з написом «Save». Для того, щоб забрати доступ, необхідно натиснути піктограму сміттєвої корзини напроти відповідного користувача.

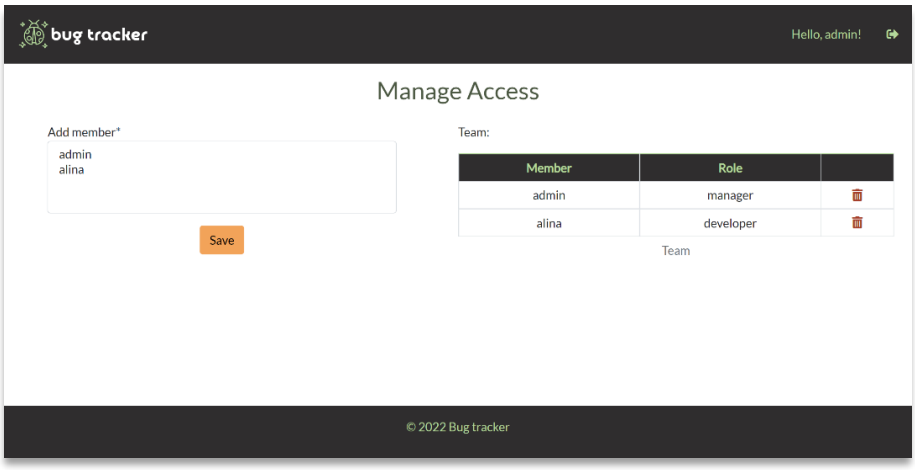

Рисунок 4.11 – Вигляд сторінки для управління доступом до проєкту

Для створення та редагування проєкту використовується форма, що містить поля для назви та опису. Приклад сторінки з вказаною формою зображено на рисунку 4.12. Для збереження даних варто натиснути кнопку «Save».

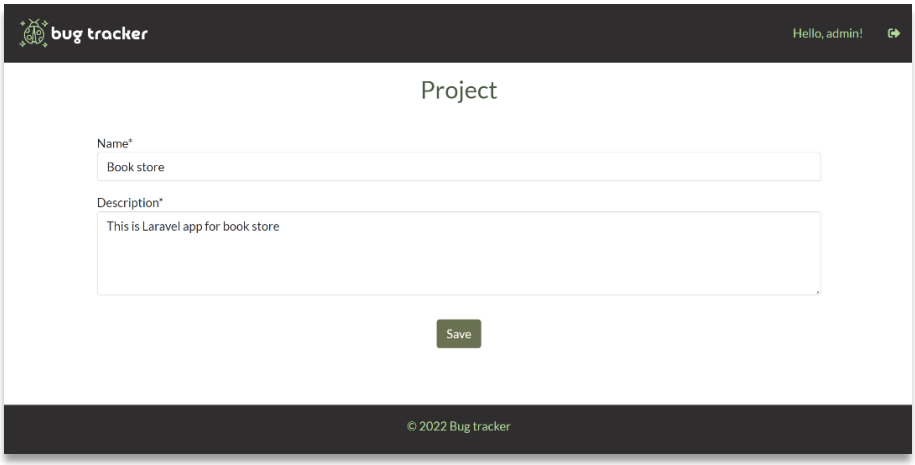

Рисунок 4.12 – Вигляд форми для створення та редагування проєкту

4.3.3 Робота з віртуальною Kanban-дошкою

При натисненні на назву проєкту відбувається перехід до сторінки з віртуальною Kanban-дошкою. На рисунку 4.13 зображено приклад Kanban-дошки.

Рисунок 4.13 – Віртуальна Kanban-дошка

На вказаній сторінці можна перетягувати картки з завданнями та звітами про помилки, змінюючи їх статус. Крім того, натиснувши на посилання «Tasks backlog» або «Bugs backlog» можливо перейти до списку всіх завдань та помилок відповідно. У даних списках є всі елементи, а не лише ті, що прикріплені до Kanban-дошки.

4.3.4 Робота з даними про завдання

Вигляд сторінки зі списком завдань для менеджера зображений на рисунку 4.14. Для повернення на сторінку з віртуальною Kanban-дошкою необхідно натиснути посилання «Dashboard». Для перегляду статистики про завдання необхідно натиснути посилання «Get tasks statistic». Прикріплення завдання до віртуальної дошки відбувається за допомогою натиснення на посилання «On dashboard». Користувач також може завантажити список завдань у форматі CSV, натиснувши на кнопку «Load CSV».

Крім звичайного перегляду та прикріплення завдань до віртуальної Kanban-дошки, лише менеджер може додавати (піктограма з плюсом), редагувати (піктограма з олівцем) та видаляти (піктограма з сміттєвою корзиною) завдання. Назва завдання є посиланням на сторінку з детальною інформацією про це завдання.

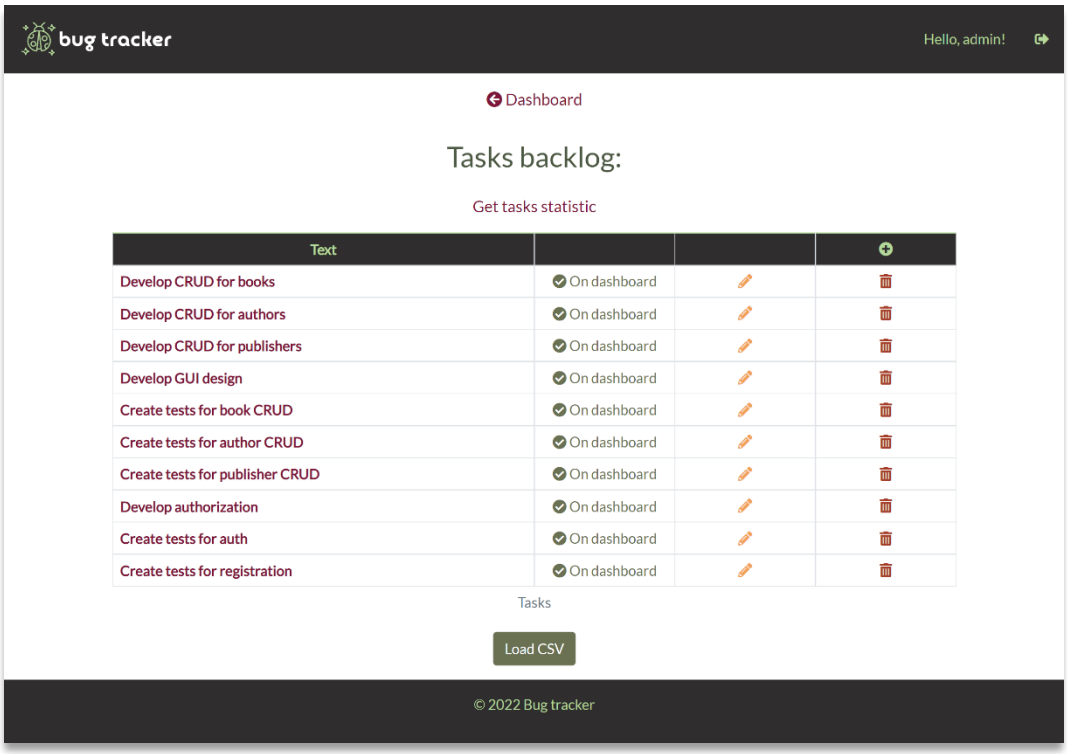

Рисунок 4.14 – Вигляд сторінки зі списком завдань для менеджера

На рисунку 4.15 зображено форму для створення та редагування завдання. Необхідно заповнити поля з назвою, складністю та виконавцем і натиснути кнопку з написом «Save» для збереження даних.

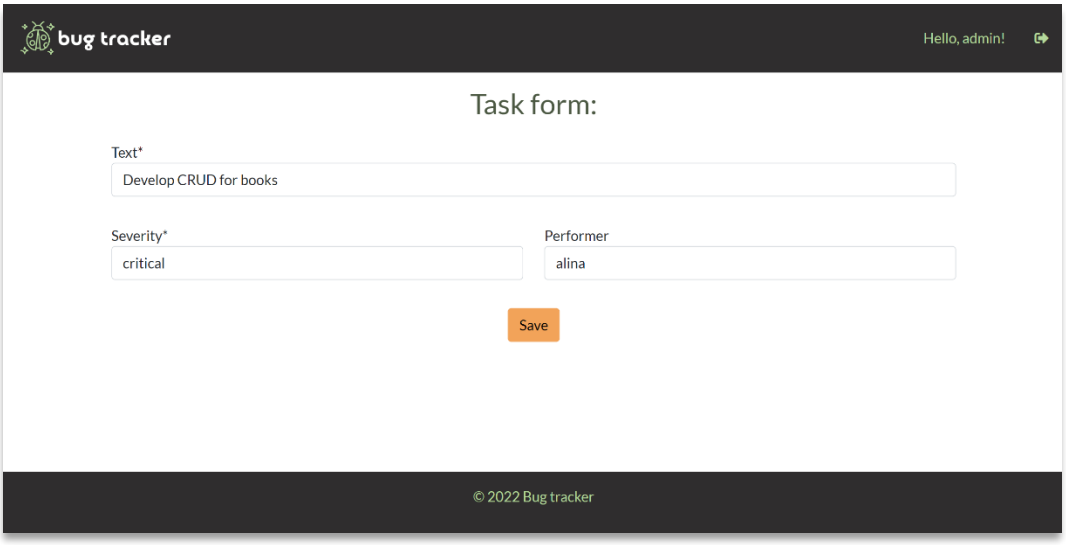

Рисунок 4.15 – Вигляд форми для редагування та додавання завдань

На рисунку 4.16 зображено зовнішній вигляд сторінки з детальними даними про завдання. За допомогою посилання «All tasks» можна повернутися до сторінки зі списком завдань. За допомогою кнопки «Predict time» можна отримати прогноз щодо часу, який піде на виконання даного завдання.

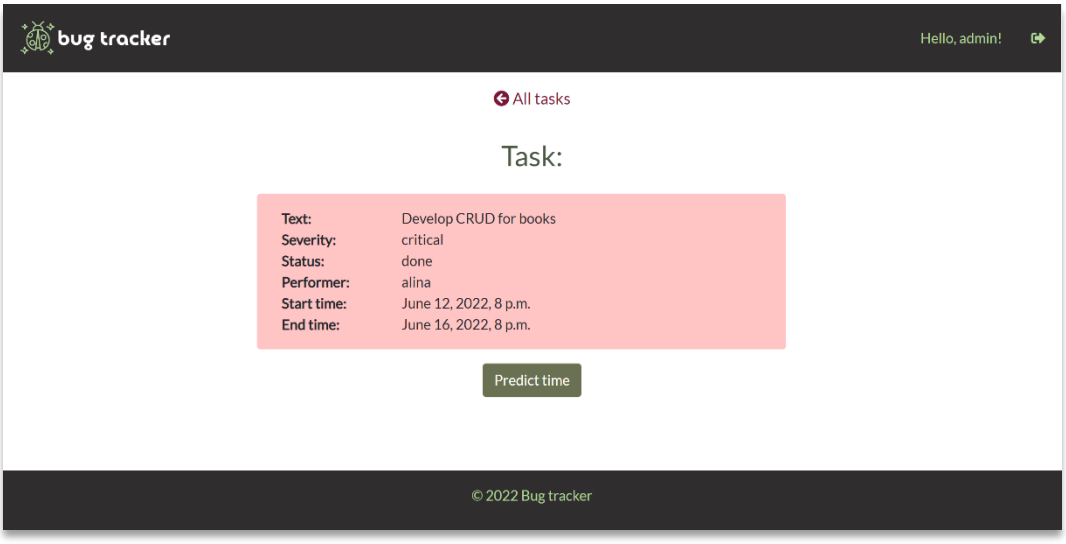

Рисунок 4.16 – Вигляд сторінки з даними про обране завдання

4.3.5 Робота з даними про помилки

Сторінку зі списком помилок зображено на рисунку 4.17. Для повернення на сторінку з віртуальною Kanban-дошкою необхідно натиснути посилання «Dashboard». Для перегляду статистики про помилки необхідно перейти за посиланням «Get tasks statistic». Прикріплення звіту про помилку до віртуальної Kanban-дошки відбувається за допомогою натиснення на посилання «On dashboard». Користувач також може завантажити список звітів про помилки у форматі CSV, натиснувши на кнопку «Load CSV».

Менеджер та тестувальник можуть додавати (піктограма з плюсом), редагувати (піктограма з олівцем) та видаляти (піктограма з сміттєвою корзиною) звіти про помилки. Назва звіту про помилку є посиланням на сторінку з детальною інформацією про обраний звіт.

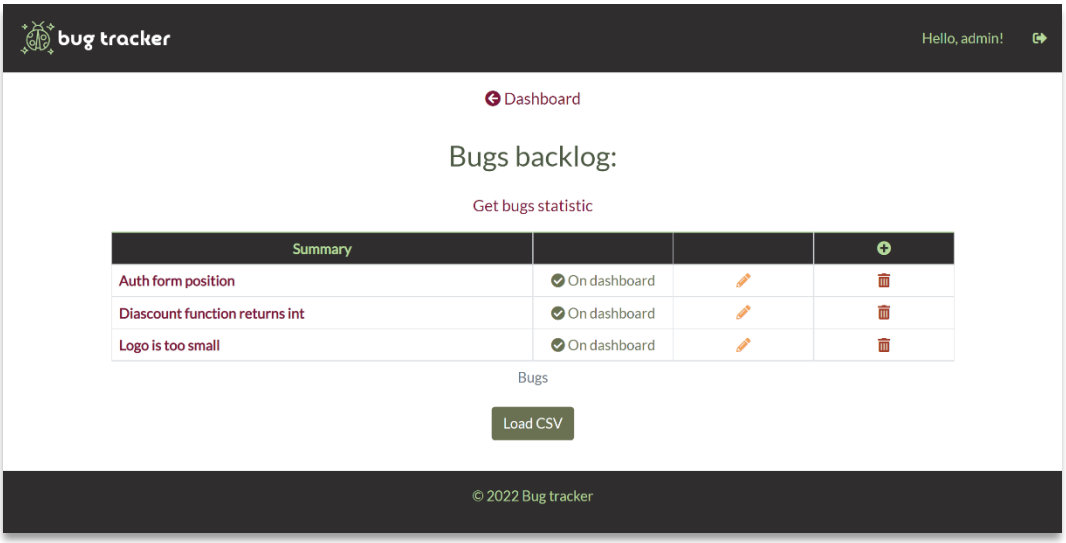

Рисунок 4.17 – Вигляд сторінки зі списком звітів про помилки

На рисунку 4.18 зображено форму для створення та редагування звіту про помилку. Необхідно заповнити наступні поля:

а) «Summary» − короткий опис помилки;

б) «Severity» − складність;

в) «Priority» − пріоритет;

г) «Description» − детальний опис помилки;

д) «Symptom» − прояв;

е) «Reproducibility» − відтворюваність;

ж) «Workaround» − можливість обійти.

Поля для прикріплення зображень («Screenshot #1», «Screenshot #2», «Screenshot #3») не є обов’язковими. Виконавця завдання («Performer») також можна обрати згодом. Для збереження звіту про помилку необхідно натиснути кнопку з написом «Save».

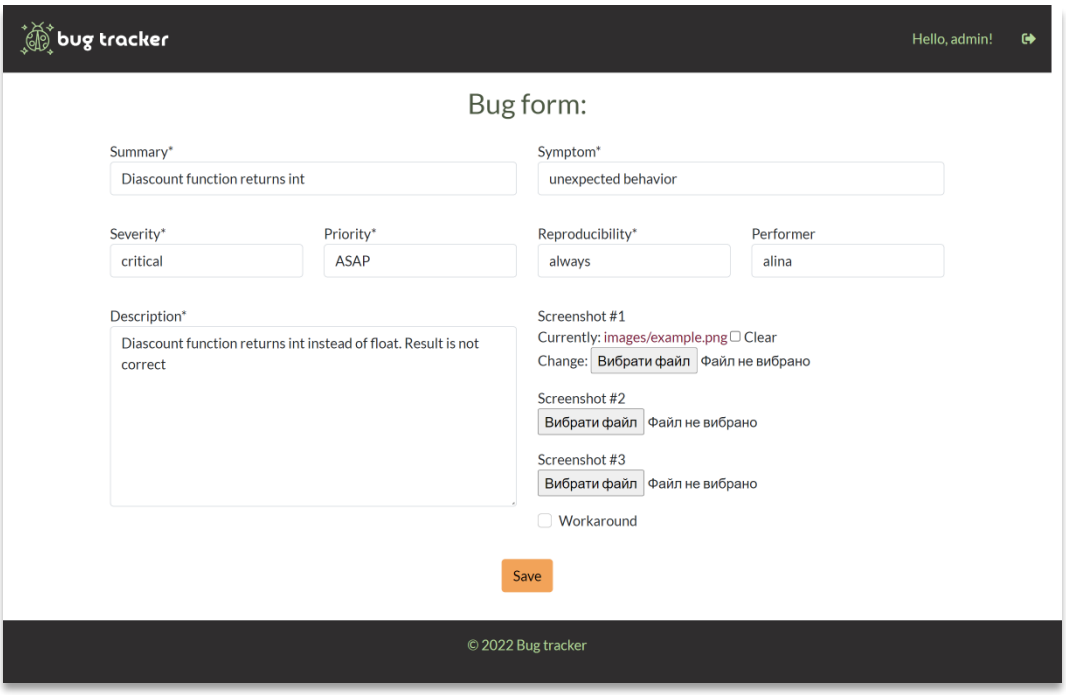

Рисунок 4.18 – Вигляд форми для редагування та додавання звітів про помилки

На рисунку 4.19 зображено зовнішній вигляд сторінки з детальними даними про помилку. За допомогою посилання «All bugs» можна повернутися до сторінки зі списком звітів про помилки. За допомогою кнопки «Predict time» можна отримати прогноз щодо часу, який піде на виправлення помилки.

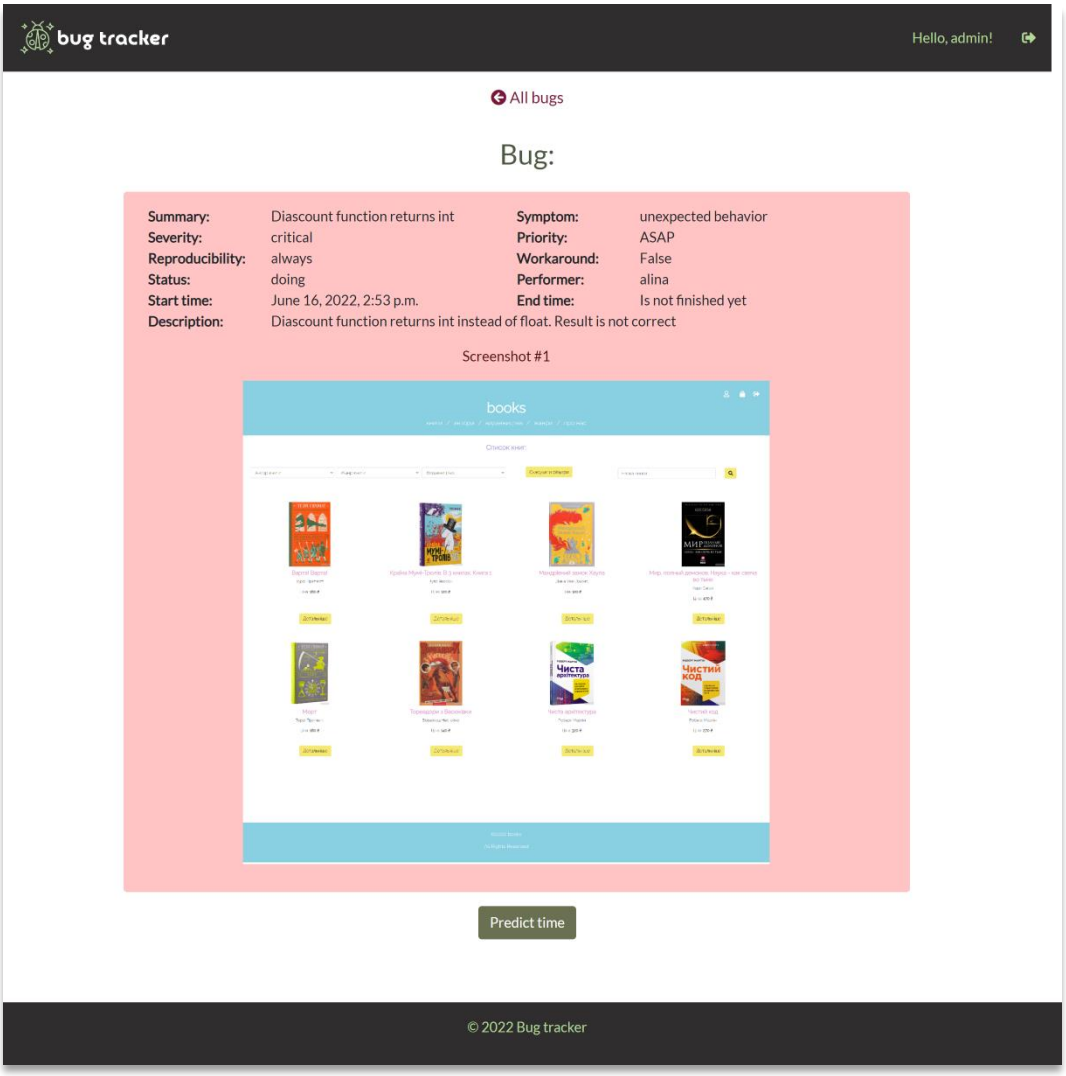

Рисунок 4.19 – Вигляд сторінки з даними про обрану помилку

4.3.6 Аналіз даних та машинне навчання

Діаграма, що відображає статистику про завдання, зображена на рисунку 4.20. З вказаної сторінки можливо повернутися до віртуальної Kanban-дошки за допомогою посилання «Dashboard». Сторінка зі статистикою про помилки виглядає аналогічно.

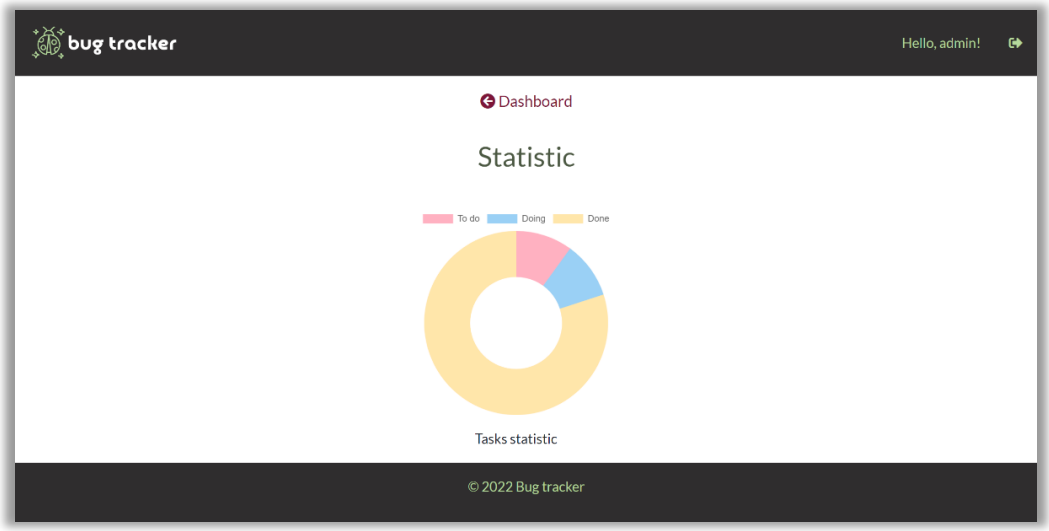

Рисунок 4.20 – Відображення статистики про завдання

Сторінка з прогнозом щодо часу, який піде на виконання завдання, зображена на рисунку 4.21. Прогноз для помилки виглядає аналогічно.

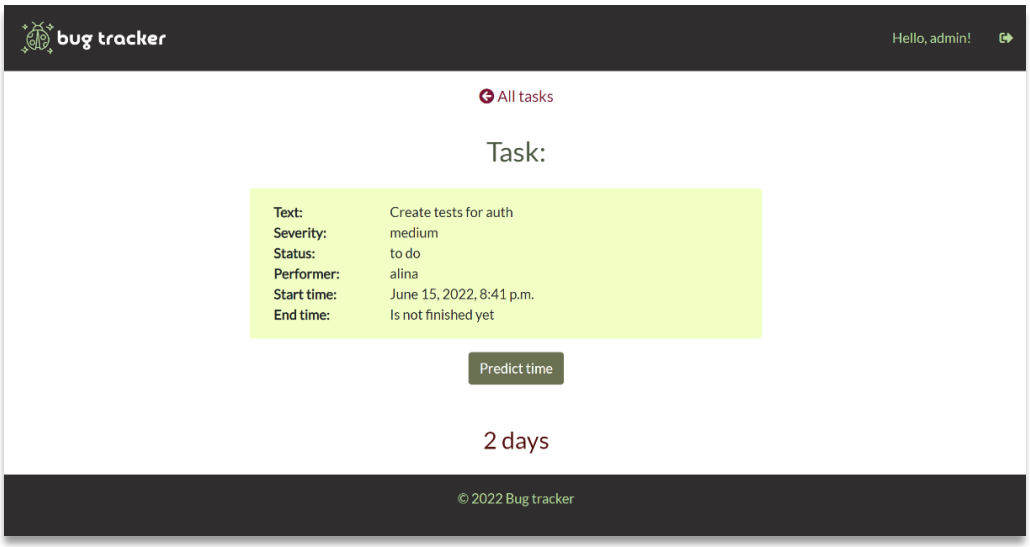

Рисунок 4.21 – Сторінка з прогнозом щодо розробки

4.3.7 Умовні позначення

На рисунку 4.22 зображено фрагмент сторінки з попередженням про помилку. Якщо користувач бачить подібне зображення, то він намагався перейти до неіснуючої сторінки.

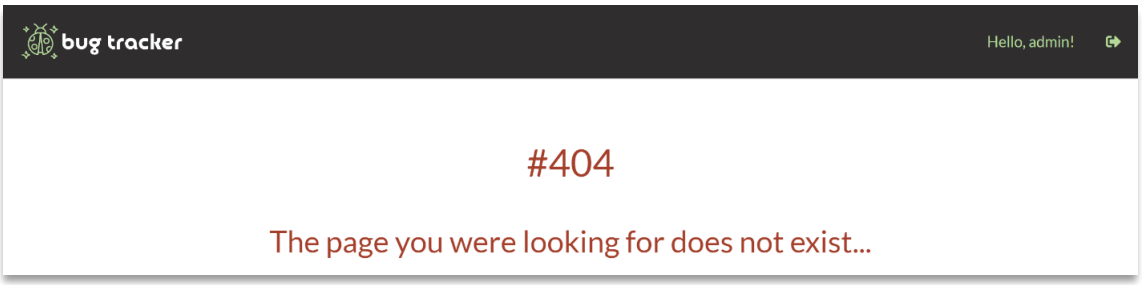

Рисунок 4.22 – Сторінка з попередженням про неіснуючу сторінку

На рисунку 4.23 зображено фрагмент сторінки з попередженням про відсутність доступу. Якщо користувач бачить таке зображення, то він намагався порушити домовленість щодо прав доступу користувачів з різними ролями.

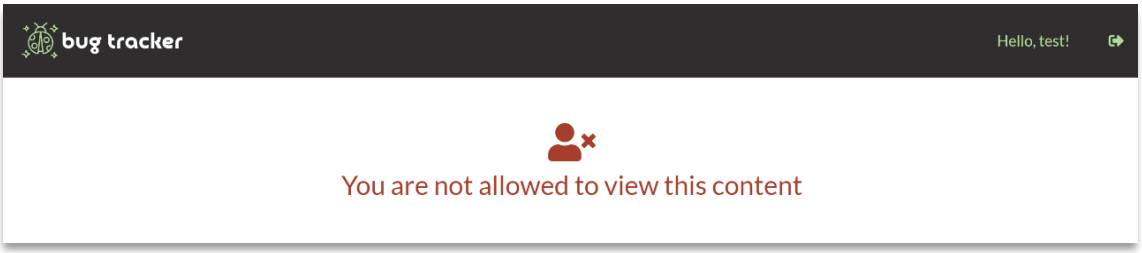

Рисунок 4.23 – Сторінка з попередженням про відсутність доступу

Висновки до розділу 4

В ході створення четвертого розділу було обґрунтовано вибір технологій для реалізації системи, описано код та структуру проєкту. Також створено керівництво користувача. Обрані програмні засоби дозволили розробити вебзастосунок ефективно та швидко. Створений вебзастосунок відповідає вимогам. Отже, поставлена мета була досягнута.

ВИСНОВКИ

В результаті створення кваліфікаційної роботи бакалавра було реалізовано вебзастосунок, що вдосконалює процес відстеження помилок за допомогою аналітичної обробки даних та машинного навчання. Отже, поставлена мета досягнута. Для досягнення мети виконано наступні завдання:

а) проведено аналіз предметної області та аналогічного програмного забезпечення;
б) виконано специфікацію вимог на основі здійсненого аналізу;
в) проаналізовано алгоритми машинного навчання та аналітичної обробки даних;
г) здійснено проєктування та моделювання системи;
д) відповідний вебзастосунок реалізовано, протестовано та відлагоджено.

На основі набутого в ході розробки досвіду можна зробити наступні висновки:

а) чим більш детальним буде процес моделювання та проєктування, тим швидше і якісніше пишеться код;
б) найголовніша цінність архітектури програмного забезпечення – гнучкість та масштабованість;
в) найголовніша цінність коду – читабельність та лаконічність;
г) використання ефективних та протестованих функцій зі спеціалізованих бібліотек для аналізу даних та машинного навчання суттєво скорочує витрати часу та кількість помилок;
д) основна проблема при впровадженні машинного навчання – залежність точності прогнозів від якості даних, на основі яких відбувається навчання.

Отже, розроблена система відповідає всім вимогам та своєму призначенню. За потреби вона може бути вдосконалена та розширена.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stellman A., Greene J. Learning Agile. Sebastopol : O'Reilly Media, Inc., 2014. 418 p.
2. Fowler M., Highsmith J., others. The agile manifesto. *Software development*. Vol. 9, Issue 8. P. 28–35.
3. Tian J. Software Quality Engineering. Hoboken : John Wiley & Sons, Inc., 2005. 440 p.
4. Géron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol : O'Reilly Media, Inc., 2017. 563 p.
5. Nelli F. Python Data Analytics. Rome : Apress, 2018. 575 p.
6. McKinney W. Python for Data Analysis. Sebastopol : O'Reilly Media, Inc., 2018. 540 p.
7. Python Machine Learning Tutorial (Data Science) - YouTube. URL: https://www.youtube.com/watch?v=7eh4d6sabA0 (дата звернення 12.05.2022).
8. Freeman Er., Freeman El., Sierra K. et al. Head First Design Patterns. Sebastopol : O'Reilly Media, Inc., 2004. 680 p.
9. Martin R. C. Clean Architecture. Hoboken : Prentice Hall, 2018. 351 p.
10. Design philosophies | Django documentation | Django. URL: https://docs.djangoproject.com/en/4.0/misc/design-philosophies/ (дата звернення 01.06.2022).
11. Django and Webpack - Static Files Managed Efficiently (Webpack 4, Django, Sass, Bootstrap) - YouTube. URL: https://www.youtube.com/watch?v=k78Oi383DRc (дата звернення 12.06.2022).
12. User Role Based Permissions & Authentication. URL: https://www.youtube.com/watch?v=eBsc65jTKvw (дата звернення 03.06.2022).

ДОДАТОК А

Код для аналітичної обробки даних

```
import pandas, numpy
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier


# Groups items by status
def get_data_for_statistics(dataset):
    data = list_to_dataframe(dataset)
    if data is not None:
        return data.groupby(["status"])["status"].count()


# Makes predictions about item`s progress based on severity and performer
def predict_time_for_item_by_ml(performer, severity, items):
    data_frame = list_to_dataframe(items)

    if data_frame is not None and len(data_frame) > 4:
        # Clean data
        X = data_frame[["severity", "performer_id"]]  # data (input)
        y = data_frame["end_time"] - data_frame["start_time"]  # answers

        # Split data into Training/Test sets
        X_train, X_test, y_train, y_test = train_test_split(
            X,
            y,
            test_size=0.2)

        # Train model
        model = DecisionTreeClassifier()
        model.fit(X_train, y_train)

        # Make prediction
        prediction = model.predict([[severity, performer.id]])[0]

        return numpy.timedelta64(prediction, "D")
    else:
        return "Not enough data for machine learning. Sorry :("


# Converts list to DataFrame
def list_to_dataframe(list_of_models):
    if len(list_of_models) > 0:
        row = list_of_models[0].__dict__
        columns = row.keys()

        return pandas.DataFrame(
            [
                [getattr(row, col) for col in columns]
                for row in list_of_models
            ],
            columns=columns
        )
```

ДОДАТОК Б

Код для перетягування елементів по Kanban-дошці

```
// Draggable elements
const items = document.getElementsByClassName("js-item");

function drag_start(event){
    let id = event.target.id;
    item = document.getElementById(id);

    setTimeout(() => {item.classList.add("hide");}, 0);

    event.dataTransfer.setData("id", id);
    if(item.classList.contains("task")){
        event.dataTransfer.setData("type", "task");
    }else if(item.classList.contains("bug")){
        event.dataTransfer.setData("type", "bug");
    }
}

function drag_end(event){
    window.location.href = "/dashboard/" + {{project_id}};
}

for (var i = 0; i < items.length; ++i) {
    var item = items[i];
    item.ondragstart = drag_start;
    item.ondragend = drag_end;
}

// Drop zones
const zones = document.getElementsByClassName("js-zone");

function allow_drop(event){
    event.preventDefault();
}

function drop(event){
    let zone_id = event.target.id;
    let id = event.dataTransfer.getData("id");
    let type = event.dataTransfer.getData("type");
    let link = "/dashboard/" + {{project_id}};

    if(zone_id === "to_do"){
        link = "/set_" + type + "_to_do/" + {{project_id}} + "/" + id;
    }
    else if(zone_id === "doing"){
        link = "/set_" + type + "_doing/" + {{project_id}} + "/" + id;
    }
    else if(zone_id === "done"){
        link = "/set_" + type + "_done/" + {{project_id}} + "/" + id;
    }

    window.location.href = link;
}

for (var i = 0; i < zones.length; ++i) {
    var zone = zones[i];
    zone.ondragover = allow_drop;
    zone.ondrop = drop;
}
```

ДОДАТОК В

Код сторінки для відображення статистики

```
{% extends "base.html" %}
{% load crispy_forms_tags %}

{% block title %}
    Statistic
{% endblock %}

{% block content %}
<p class="text-center pt-3 link">
    <a href="/dashboard/{{project_id}}">
        <span>
            <i class="fas fa-arrow-circle-left"></i> Dashboard
        </span>
    </a>
</p>

<h2 class="text-center text-secondary p-3">Statistic</h2>

<div class="row align-items-center justify-content-center">
    <div class="col-md-3">
        <canvas id="items" class="p-3"></canvas>
        <p class="text-center">{{description}}</p>
    </div>
</div>

<script>
const items_ctx = document.getElementById('items');
const items = new Chart(items_ctx, {
    type: 'doughnut',
    data: {
        labels: ['To do', 'Doing', 'Done'],
        datasets: [{
            label: '{{label}}',
            data: [
                {% if data.1 is None %}0{% else %}{{data.1}}{% endif%},
                {% if data.2 is None %}0{% else %}{{data.2}}{% endif%},
                {% if data.3 is None %}0{% else %}{{data.3}}{% endif%},
            ],
            backgroundColor: [
                'rgba(255, 99, 132, 0.5)',
                'rgba(54, 162, 235, 0.5)',
                'rgba(255, 206, 86, 0.5)',
            ],
            borderColor: [
                'rgba(255, 99, 132, 1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
            ],
            borderWidth: 0,
        }]
    },
});
</script>
{% endblock %}
```