

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
КЕРУВАННЯ ВІДСЛІДКУВАННЯМ ПОМИЛОК
RESEARCH AND SOFTWARE IMPLEMENTATION OF ERROR TRACKING
METHODS

Виконав: студент(ка) 2 курсу, групи КНТ-210м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

Тюріков О.В.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тюрікова Олега Валерійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів
керування відслідковуванням помилок
Research and Software Implementation of Error Tracking Methods

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

затверджені наказом закладу вищої освіти від “5” листопада 2021 року №435

2. Строк подання студентом проєкту (роботи) грудень 2021 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Розробка архітектури програми. 4. Основні рішення
щодо реалізації компонентів системи. 5. Експлуатація, тестування та
експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Федорончак Т.В., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 10 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз та дослідження предметної області	1-2 тиждень	
2	Розробка та затвердження технічного завдання	3 тиждень	Технічне завдання
3	Дослідження методів розв’язання проблеми	4 тиждень	
4	Проектування програмного забезпечення	5 тиждень	
5	Розроблення структури програми та модулів	6-7 тиждень	Структурна схема
6	Розроблення програмного коду	8 тиждень	Текст програми
7	Тестування та відлагодження програмного забезпечення	9 тиждень	Програма
8	Розробка розділів пояснювальної записки	10-11 тиждень	Пояснювальна записка
9	Розробка графічного матеріалу	12 тиждень	Графічний матеріал
10	Захист магістерської роботи	13 тиждень	

Студент(ка)

(підпис)

Тюріков О.В.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Федорончак Т.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
101 с., 27 рис., 3 табл., 3 дод., 21 джерело.

ВІДСЛІДКОВУВАННЯ ПОМИЛОК, СТАН ПОМИЛКИ,
КРИТИЧНІСТЬ ПОМИЛКИ, КЛАСИФІКАЦІЯ ДАНИХ, КОНВОЛЮЦІЙНА
НЕЙРОННА МЕРЕЖА, WORD2VEC.

Об'єкт дослідження – процес відслідковування помилок.

Предмет дослідження – методи керування відслідковуванням помилок.

Мета роботи – розроблення програмного забезпечення керування відслідковуванням помилок для підвищення обґрунтованості прийнятих менеджером рішень.

Матеріали, методи та технічні засоби: мови програмування C# і Python, система Microsoft SQL Server, згорткові нейронні мережі, word2vec.

Результати. Розроблена програма дозволяє відслідковувати помилки різних проєктів компанії з розробки програмного забезпечення.

Науковою новизною роботи є метод класифікації помилок, який визначає, чи належить виявлена помилка до тих, які будуть потребувати повторного усунення, застосуванням моделі на основі конволюційних нейронних мереж і методу word2vec для оброблення текстових описів помилок, що дозволяє на основі отриманих даних менеджера проєктів приймати обґрунтовані рішення про виділення потрібних ресурсів.

Висновки. Виявлення помилок, які можуть потребувати повторної роботи за початкового виділення недостатньої кількості ресурсів на усунення помилки, є важливою практичною особливістю розробленої програми.

Галузь використання – компанії з розробки програмного забезпечення в частині роботи над усуненням виявлених помилок програмних проєктів.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 101 pages, 27 figures, 3 tables, 3 appendixes, 21 sources.

ERROR TRACKING, ERROR STATUS, ERROR SEVERITY, DATA CLASSIFICATION, CONVOLUTIONAL NEURAL NETWORK, WORD2VEC.

The object of research is the process of error tracking.

The subject of research is methods of error tracking management.

The purpose of the work is to develop software for error tracking management to increase the validity of decisions made by project manager.

Materials, methods and technical means: C# and Python programming languages, Microsoft SQL Server system, convolutional neural networks, word2vec.

Results. The developed program enables to track errors of various projects of software development company.

The scientific novelty of the work is the method of error classification. This method determines whether the detected error belongs to the errors which will need to be re-eliminated. It uses model based on convolutional neural networks and word2vec method to process text descriptions of errors. The method enables decision making on the allocation of the necessary resources.

Conclusions. Detecting errors which may be repeated with the initial allocation of insufficient resources to eliminate the error is an important practical feature of the developed software.

Field of use is software development companies in terms of work to eliminate identified errors in software projects.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Проблема відслідковування помилок	10
1.2 Огляд програмних засобів відслідковування помилок	12
1.3 Висновки за розділом 1	14
2 Матеріали і методи	16
2.1 Методи прогнозування довготривалих помилок	16
2.2 Конволюційні нейронні мережі	18
2.3 Метод word2vec	20
2.4 Метод класифікації помилок	21
2.5 Висновки за розділом 2	23
3 Розробка архітектури програми	25
3.1 Визначення функціональних вимог до програми	25
3.2 Вибір засобів розробки	29
3.3 Визначення структури даних	31
3.4 Структура програмного забезпечення	39
3.5 Висновки за розділом 3	40
4 Основні рішення щодо реалізації компонентів системи	41
4.1 Функціонування програми	41
4.2 Опис модулів реалізованої програмної системи	44
4.3 Висновки за розділом 4	52
5 Експлуатація, тестування та експериментальне дослідження програми	53
5.1 Призначення програми	53
5.2 Умови виконання програми	53
5.3 Виконання програми	53
5.4 Експериментальне дослідження класифікації помилок	57
5.5 Висновки за розділом 5	58

Висновки.....	59
Перелік джерел посилання	61
Додаток А Технічне завдання.....	64
Додаток Б Текст програми.....	68
Додаток В Слайди презентації	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

КНМ – конволюційна нейронна мережа;

ПЗ – програмне забезпечення.

ВСТУП

Робота над помилками під час життєвого циклу програмного забезпечення (ПЗ) є важливим завданням, адже без реалізації цього процесу обійтися неможливо. Помилки можуть виявлятися в процесі створення програми, після її впровадження. І кожний такий випадок потрібно зафіксувати, щоб не загубити, а потім обробити належним чином. Існуючі на даний момент програмні рішення дозволяють вирішувати завдання в частині фіксації, оброблення помилок, але не мають інтелектуальних засобів підтримки процесів керування помилками. Такі засоби важливі зокрема для менеджера, адже він в процесі своєї роботи визначає критичність помилки, а найголовніше – встановлює строк, до якого помилка має бути усунена. Нерідко зокрема через дуже обмежені строки помилка усувається не повністю, а тому через певний час виявляється знову в іншому ракурсі. Тоді помилку потрібно усувати повторно і цей процес може повторюватися і декілька разів. Зрозуміти, на скільки конкретна помилка піддається ризику повторного виявлення, тобто є складною для розробників, важливо. Тоді менеджер буде приймати більш обґрунтовані рішення. Для цього і має бути розроблене відповідне ПЗ.

Тому завданнями дипломної кваліфікаційної роботи магістра було визначено:

- дослідження і розробка власного методу керування відслідковуванням помилок;
- проєктування програмного забезпечення;
- програмна реалізація системи керування відслідковуванням помилок.

Мета роботи – розроблення програмного забезпечення керування відслідковуванням помилок для підвищення обґрунтованості прийнятих менеджером рішень.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Проблема відслідковування помилок

За відсутності інструменту відстеження помилок команди використовують електронні таблиці для звітування, відстеження та перенесення помилок, але хоча це може бути хорошим тимчасовим рішенням для невеликих команд і проєктів, це не є стійким методом і відзначається наступними недоліками:

- аркуші Excel з вкладеними знімками екрана іноді займають більше декількох Мб, такі файли часто доводиться надсилати електронною поштою, прикріплюючи до листів електронні таблиці з прикріпленими електронними листами, в результаті купа таких файлів знаходиться в поштовій скриньці: якщо ж такі дані розташовувати в хмарі, то зберігати в них повідомлення за кожною помилкою стане незручно;

- відсутність видимості в реальному часі щодо виявлення помилок та прогресу/стану: невідомо також, чи проблема була повторно перевірена чи повернута тощо в режимі реального часу, оскільки автоматичної системи оповіщення немає, дефекти не привертають до них уваги, якщо тільки хтось навмисно не переглядає їх;

- збір та консолідація статистики дефектів вручну для розуміння якості продукту [1].

Для програмного забезпечення для відстеження помилок важливо мати наступні, абсолютно необхідні, функції:

- функція звітування – з полями, які дозволять надати інформацію про помилку, модуль, серйозність, знімки екрана тощо;

- призначення відповідальних – від програми не буде ніякої користі, коли все, що вона дозволяє зробити, – це знайти помилку та внести дані про неї;

- історія станів, коментарі;

– зберігання та пошук – кожна сутність у процесі тестування має бути однозначно ідентифікована, те ж правило стосується і помилок: інструмент відстеження помилок повинен забезпечувати спосіб мати ідентифікатор, який можна використовувати для зберігання, отримання (пошуку) та упорядкування інформації про помилку [1].

Характеристики та техніки, які використовуються для створення ефективних звітів про помилки:

– наявність чітко визначеного номера помилки: завжди потрібно призначати унікальний номер кожному звіту про помилку: якщо використовується будь-який автоматизований інструмент звітування про помилки, цей унікальний номер буде створюватися автоматично щоразу, коли повідомляється про помилку;

– відтворюваність: якщо помилку неможливо відтворити, її ніколи не буде виправлено, необхідно чітко вказати кроки для відтворення помилки, не можна пропускати жодних кроків відтворення, помилку, описану крок за кроком, легко відтворити та виправити;

– необхідно бути конкретним: не потрібно писати все про проблему, потрібно коротко описати проблему мінімальними словами, але ефективним способом, не можна поєднувати декілька проблем, навіть якщо вони здаються схожими, потрібно написати різні звіти для кожної проблеми;

– хороший звіт про помилки має бути ясним і стислим без жодних пропущених ключових моментів: будь-яка нечіткість призводить до непорозуміння і також уповільнює процес розробки;

– дубльована помилка є тягарем у циклі тестування: потрібно переглянути весь перелік відомих помилок, іноді розробники можуть знати про проблему та ігнорувати її для майбутніх випусків;

– мета написання звіту про помилку – дати розробнику можливість уявити проблему: розробник повинен чітко розуміти дефект зі звіту про помилку;

– про кожну помилку потрібно повідомити окремо: у випадку декількох проблем в одному звіті про помилки, не можна закрити його, доки не буде вирішено всі проблеми, тому найкраще розділити проблеми на окремі помилки, це гарантує, що кожну помилку можна обробити окремо [2].

1.2 Огляд програмних засобів відслідковування помилок

У цьому підрозділі виконано огляд програмних засобів для відслідковування помилок. Оскільки для зручності роботи у великій компанії більш універсальним рішенням є використання онлайн-засобу, тобто створення вебзастосунку, то саме такі рішення і були проаналізовані.

Backlog (рис. 1.1) – інтуїтивно зрозумілий програмний засіб з мінімальною кривою навчання, що робить його зручним варіантом для більшості компаній і розробників [3],[4].

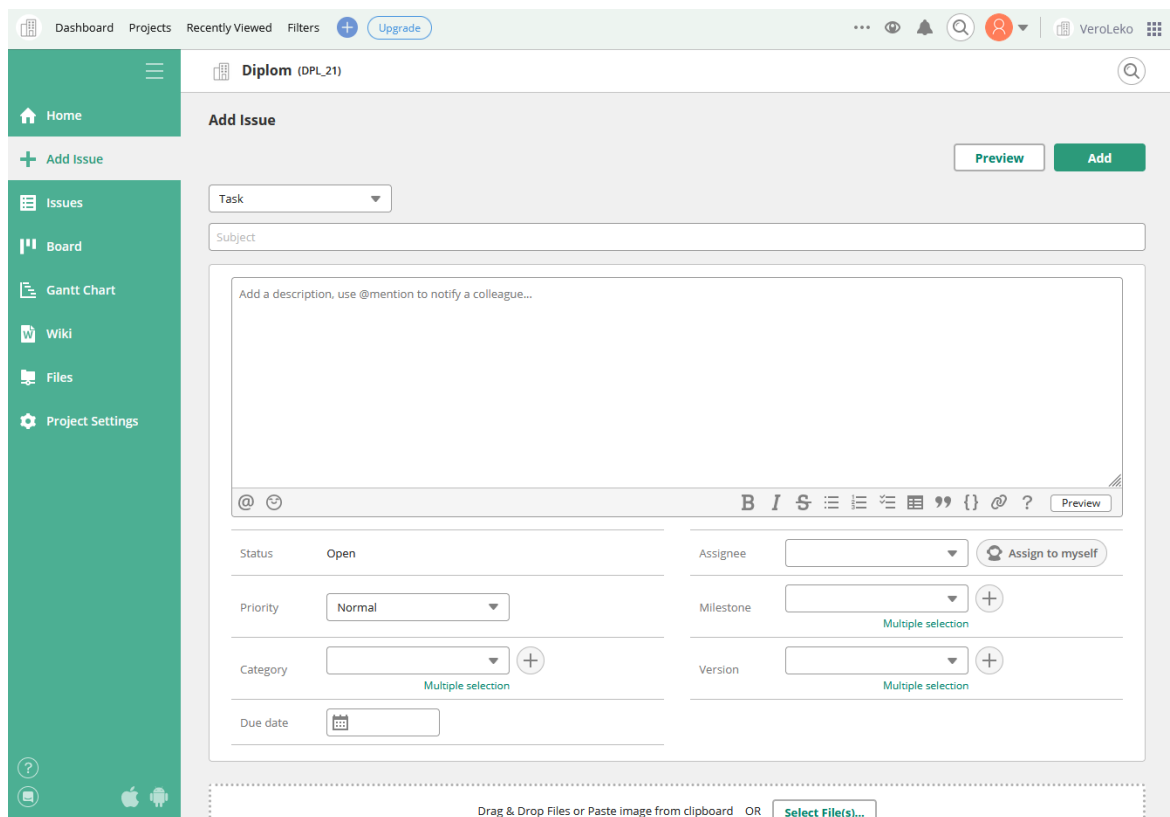


Рисунок 1.1 – Внесення звіту про помилку в програмі Backlog
Backlog відзначається наступним:

- майже будь-хто може розпочати проєкт, додавати завдання та призначати завдання своїй команді;
- завдання можна перетягувати з однієї дошки на іншу: “Відкриті”, “У виконанні”, “Вирішені” та “Закриті”, вказуючи пріоритет для їх організації;
- кожна позиція включає сповіщення та можливість коментування, щоб учасники команди могли детально спілкуватися один з одним;
- програма зручна інтеграцією з популярними інструментами, такими як Slack для спілкування та сповіщень та iCal Sync, щоб стежити за термінами;
- програма має безкоштовний тариф, щоб можна було спробувати її з одним проєктом, проте платні плани, які знадобляться для регулярного використання в бізнесі, починаються від 35 доларів на місяць [4].

Bugzilla – один з найвідоміших програмних засобів відслідковування помилок [5], зручний варіант для команд, які віддають перевагу гнучкості програмного забезпечення з відкритим кодом: спочатку програма була розроблена для проєктів Mozilla із відстеженням помилок, функції включають :

- призначення відповідального користувача;
- можливості коментування та створення вкладень;
- розширений пошук помилок відстежуваних помилок [4].

Важливо відзначити, що це не найбільш зручний у використанні інструмент для менеджерів проєктів та інших нерозробників.

Ще однією альтернативою ПЗ відслідковування помилок є BugHerd. Головною особливістю даного ПЗ є те, що воно інтегрується з вебсайтом і дозволяє для нього відслідковувати помилки. Тобто це достатньо особливий інструмент, зручний для роботи з клієнтами, відслідковування помилок щодо конкретного вебсайту.

BugHerd – це простий інструмент відстеження помилок і ПЗ для візуального зворотного зв’язку [5]: це дозволяє легко розміщати вебсайти та зберігати всі помилки і відгуки в одному центральному місці для легкого

керування, що дозволяє клієнтам прикріплювати відгуки до елементів на вебсторінці для точного визначення місцезнаходження проблем і включати знімки екрана або відео, особливості програми:

- можна робити знімки екрана та коментувати їх або додавати коротке відео;
- дошка завдань у стилі канбан від BugHerd дозволяє відстежувати та керувати помилками і відгуками;
- вбудована інтеграція зі Slack, GitHub і Wordpress або інтеграція з JIRA, Trello, Asana та іншими інструментами управління проєктами;
- відгуки можна закріплювати на елементах вебсайту;
- коментарі відтворюються в режимі реального часу;
- інтегрується з GitHub, GitLab або BitBucket;
- плани BugHerd починаються від 39 доларів за 5 членів команди [7]-[8].

Для використання для відслідковування помилок під час життєвого циклу ПЗ дане ПЗ не направлене.

Результати аналізу програм будуть використані при визначенні функціональних вимог до програми, що розробляється. Але основною особливістю порівняно з цими програмами розроблюваної програми буде наявність засобів прогнозування повторного повернення до помилки, чого у вигляді інтелектуального засобу в розглянутих програмах немає.

1.3 Висновки за розділом 1

Розглянуті переваги і особливості організації систем відслідковування помилок, особливості побудови ефективних звітів дозволили визначити основні можливості, які обов'язково повинні надаватися системами подібного типу.

Виконаний огляд програм, які використовуються в даній галузі (Backlog, Bugzilla, BugHerd), дозволив визначити, що створення

інтелектуальних засобів визначення можливості повторної роботи над помилкою в перспективі є важливим практичним напрямком.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Методи прогнозування довготривалих помилок

У галузі відслідковування помилок на даний момент існує два поширені підходи до прогнозування показників помилок:

- прогнозування тривалості усунення помилок – у такому випадку в результаті має бути виконано прогноз часу, який загалом було витрачено на усунення помилки, починаючи від першого повідомлення про її виявлення, завершуючи остаточним вирішенням;

- класифікація помилок на такі, які відносяться до довготривалих, і такі, які відносяться до короткотривалих: у такому випадку має бути визначено порогове значення часу тривалості усунення помилки, за якого помилка відноситься до довготривалих.

Відповідно до кожного з цих підходів існує ряд робіт, які розглядали питання вирішення відповідних проблем.

Спочатку було проаналізовано роботи, в яких увага приділялась прогнозуванню тривалості усунення помилок.

Результати перших досліджень за даною проблемою були зафіксовані в роботі [9], де виконувалось дослідження прогнозування часу роботи над усуненням помилки. Прогнозування реалізовувалось на основі регресійної моделі. Головним результатом цієї роботи було те, що в ній зафіксовано, що актуальні загалом на той момент моделі дозволяють отримати точність прогнозування між 30 % та 49 %, що звичайно потребувало певного вдосконалення.

У роботі [10] дослідження даної проблеми було розширено за рахунок наївного байєса. При цьому в результаті модель було застосовано для класифікації помилок на помилки з різним часом усунення. Для цього досліджувалось використання різних квантилів. При цьому помилки за часом усунення класифікувались з різними порогами. Дане дослідження важливе щодо вироблення варіантів класифікації помилок у даній галузі.

Варто зазначити, що в роботах [9]-[10] в якості вхідних ознак використовувались числові ознаки, тобто текстова інформація про помилки не розглядалась.

У роботі [11] досліджується питання визначення тривалості роботи над помилками на основі їх текстового опису. Задачу було зведено до класифікації помилок на так звані швидкі та повільні. Тобто фактично це є задача класифікації помилок в описаному вище вигляді. Класифікація виконується на основі методу опорних векторів і в результаті експериментальних досліджень вирішення проблеми на основі даного методу вважається достатньо ефективним.

У роботі [12] окрім прогнозування тривалості усунення помилки виконувалась ще класифікація помилок, результатом якої було визначення критичності помилки. Для цього в роботі було використано набір методів класифікації та регресії, включаючи метод опорних векторів, наївний байєс, метод k-найближчих сусідів, порядкова регресія, нечітка лінійна регресія, нечітка багатолінійна регресія, множинна лінійна регресія, регресія опорних векторів та регресія методом k-найближчих сусідів для побудови моделей, в результаті метод регресії опорних векторів дозволив отримати кращі результати порівняно з множинною лінійною регресією і методом k-найближчих сусідів для прогнозування часу усунення помилок, при цьому отримані експериментальні результати дозволили стверджувати, що методи добре працюють для визначення критичності помилок і прогнозування часу усунення [12].

Велике дослідження довготривалих помилок виконано в роботі [13]. Дану роботу присвячено вирішенню саме проблеми класифікації. У ній помилки розподіляються на два класи: довготривалі, до яких відносять помилки, час усунення яких відповідає більше ніж 1 року, та короткотривалі – усі інші. Для вирішення цієї проблеми класифікації використовуються метод k-найближчих сусідів, наївний байєс, нейронна мережа, випадкові ліси

та метод опорних векторів. Найкращі результати в даному дослідженні дозволили отримати саме нейронні мережі.

Результати порівняння методів зведено до табл. 2.1.

Таблиця 2.1 – Порівняння методів прогнозування довготривалих помилок

Метод	Клас задачі	Результуючий показник	Основна модель
[P. Bhattacharya, I. Neamtiu]	Задача прогнозування	Час усунення помилки	Регресійна модель
[W. Abdelmoez, M. Kholief, F. M. Elsalmy]	Задача класифікації	Клас за часом усунення	Наївний байєс
[P. Ardimento, A. Dinapoli]	Задача класифікації	Клас за часом усунення	Метод опорних векторів
[M. Sharma, M. Kumari, V. B. Singh]	Задача прогнозування	Час усунення помилки	Метод регресії опорних векторів
[Luiz Gomes, Ricardo Torres, Mario Cortes]	Задача класифікації	Клас за часом усунення	Багатошарова нейронна мережа
Представлений метод	Задача класифікації	Клас щодо можливого повторного усунення помилки	Конволюційна нейронна мережа

2.2 Конволюційні нейронні мережі

Конволюційна нейронна мережа (КНМ) або згорткова нейронна мережа – це категорія нейронних мереж, які в основному використовуються для класифікації та розпізнавання зображень, при цьому було доведено, що

КНМ успішно ідентифікують об'єкти, знаки і навіть обличчя, ці алгоритми глибокого навчання беруть зображення як вхідні дані, щоб виявити та призначити значення різним ознакам зображення, щоб відрізнити одне зображення від іншого, хоча прості нейронні мережі мають певний успіх у класифікації основних бінарних зображень, вони не можуть обробляти складні зображення з піксельними залежностями, вони також не мають обчислювальної потужності, яка необхідна для обробки зображень з великими пікселями, і саме тут на допомогу приходять КНМ, допомагаючи класифікувати навіть найскладніші зображення з високою точністю, алгоритми КНМ також можуть застосовувати відповідні фільтри для визначення просторових, а також тимчасових залежностей у зображеннях [14]-[16].

Однак з часом було виявлено, що КНМ може вирішувати не тільки задачі класифікації зображень, але і класифікації тексту за відповідного кодування такого тексту.

КНМ має декілька прихованих шарів, які допомагають витягувати інформацію з зображення:

- шар згортки – це перший крок у процесі виділення цінних елементів із зображення, шар згортки має декілька фільтрів, які виконують операцію згортки, при цьому кожне зображення розглядається як матриця значень пікселів: для реалізації даного шару вводиться фільтр заданого розміру (наприклад, 3×3), він проходить по зображенню та обчислюється крапковий добуток, в результаті чого створюється згорнута матриця ознак;

- шар ReLU: ReLU означає випрямлений лінійний блок, після вилучення карт об'єктів наступним кроком є їх переміщення на рівень ReLU, ReLU виконує поелементну операцію і встановлює всі пікселі з негативним значенням в 0, це вносить в мережу нелінійність, а згенерований вихід є виправленою картою ознак;

- об'єднуючий шар: об'єднання – це операція зменшення вибірки, яка зменшує розмірність карти об'єктів, виправлена карта об'єктів проходить

через шар об'єднання для створення об'єднаної карти об'єктів: реалізовуватися може, наприклад, проходженням фільтра по зображенню та знаходженням в межах даного фільтра максимального значення, яке і поміщається в результат обчислення;

– повністю пов'язаний шар: перед реалізацією останнього шару отримані значення переводяться в одномірний вигляд, а тоді подаються на повністю пов'язаний шар, який як звичайна нейронна мережа виконує класифікацію об'єкта [14]-[16].

2.3 Метод word2vec

Word2vec – це техніка вбудовування слів для вирішення складних проблем оброблення природної мови, що може повторювати великий корпус тексту, щоб дізнатися асоціації або залежності між словами: word2Vec знаходить схожість між словами за допомогою косинусної метрики подібності, якщо метрика дорівнює 1, це означає, що слова перекриваються, а якщо кут дорівнює 90, це означає, що слова незалежні або не мають контекстної подібності, в результаті призначаються подібні векторні представлення подібним словам [17].

В основі word2Vec використовується два підхода:

– перша архітектура (continuous bag-of-words): усі слова проєктуються в однакову позицію (їхні вектори усереднюються), вважаючи при цьому, що порядок слів в історії не впливає на проєкцію, крім того також використовуються слова з наступних позицій, в результаті створюється класифікатор із декількома наступними та декількома попередніми словами на вході, де критерієм навчання є правильна класифікація поточного (середнього) слова;

– друга архітектура (skip-gram) схожа на першу, але замість того, щоб передбачати поточне слово на основі контексту, вона намагається максимізувати класифікацію слова на основі іншого слова в тому ж реченні,

використовується кожне поточне слово як вхід до логарифмічного лінійного класифікатора з безперервним проєкційним шаром і прогнозуються слова в певному діапазоні до і після поточного слова, при цьому виконуючи такі дослідження було виявлено, що збільшення діапазону покращує якість результуючих векторів слів, але також підвищує складність обчислень, оскільки більш віддалені слова зазвичай менш пов'язані з поточним словом, ніж ближчі до нього, відповідно визначається менша вага віддаленим словом, відбираючи менше з цих слів у наших навчальних прикладах, таким чином skip-gram приймає одне слово в якості вхідних даних і прогнозує його тісно пов'язані контекстні слова [18].

2.4 Метод класифікації помилок

Для керування роботою над помилками потрібно отримати всю наявну і прогнозовану інформацію, що має підвищити обґрунтування прийнятих в результаті рішень. Тільки маючи всю необхідну інформацію, можна прийняти адекватне рішення. Такою інформацією є інформація про те, чи буде дана помилка розглядатися повторно після її усунення. На отримання даної інформації і направлений представлений сукупністю наступних етапів метод класифікації помилок.

Етап 1. Створити вибірку даних про виявлені помилки.

Кожна помилка повинна представлятися наступними атрибутами:

- опис помилки, включаючи повну інформацію про помилку;
- кількість кроків для відтворення помилки;
- кількість вкладень, доданих до опису даної помилки;
- кількість рядків у описі;
- результуючий клас, який визначає, чи розглядалася помилка повторно (значення 1), чи ні (значення 0), – вихідний атрибут.

Етап 2. Виконати попереднє оброблення текстових даних і забезпечити їх кодування на основі методу word2vec.

Етап 3. Створити модель для класифікації помилок.

Модель побудована на основі КНМ, що складається з двох окремих частин, які в подальшому з'єднуються в єдину модель на рівні повністю пов'язаного шару.

Модель складається з наступних шарів (рис. 2.1):

- вхідний шар, який отримує інформацію про помилку (екземпляр помилки з вибірки даних);
- шар вбудовування, на якому формується результат застосування методу word2vec до опису помилки в першій частині шару і об'єднуються значення кількості кроків, кількості вкладень, кількості рядків у описі в другій частині шару;
- шар згортки;
- об'єднуючий шар;
- два повністю пов'язані шари, в яких з'єднуються значення з чарунок двох частин об'єднуючого шару;
- вихідний шар, де визначається клас, до якого належить помилка (значення вихідного атрибута).

Етап 4. Навчання моделі на основі вибірки даних, визначеної на першому кроці.

Етап 5. Визначення нової помилки, формування її у вигляді екземпляра, описаного для етапу 1, та визначення класу цієї помилки на основі навченої на етапі 4 моделі.

Після того, як для помилки визначено її клас, менеджер проєктів може приймати рішення щодо виділення ресурсів для роботи з нею. Таким рішенням може бути:

- виділення більшої кількості часу на роботу, тобто визначення більш пізньої дати, до якої має бути усунено помилку;
- створення більшої команди і/або команди з більшим професійним рівнем учасників;
- виконання розширеного тестування проєкту або модуля щодо

виявленої, а в подальшому і усуненої помилки (замість повторення тестів, на яких було виявлено помилку).

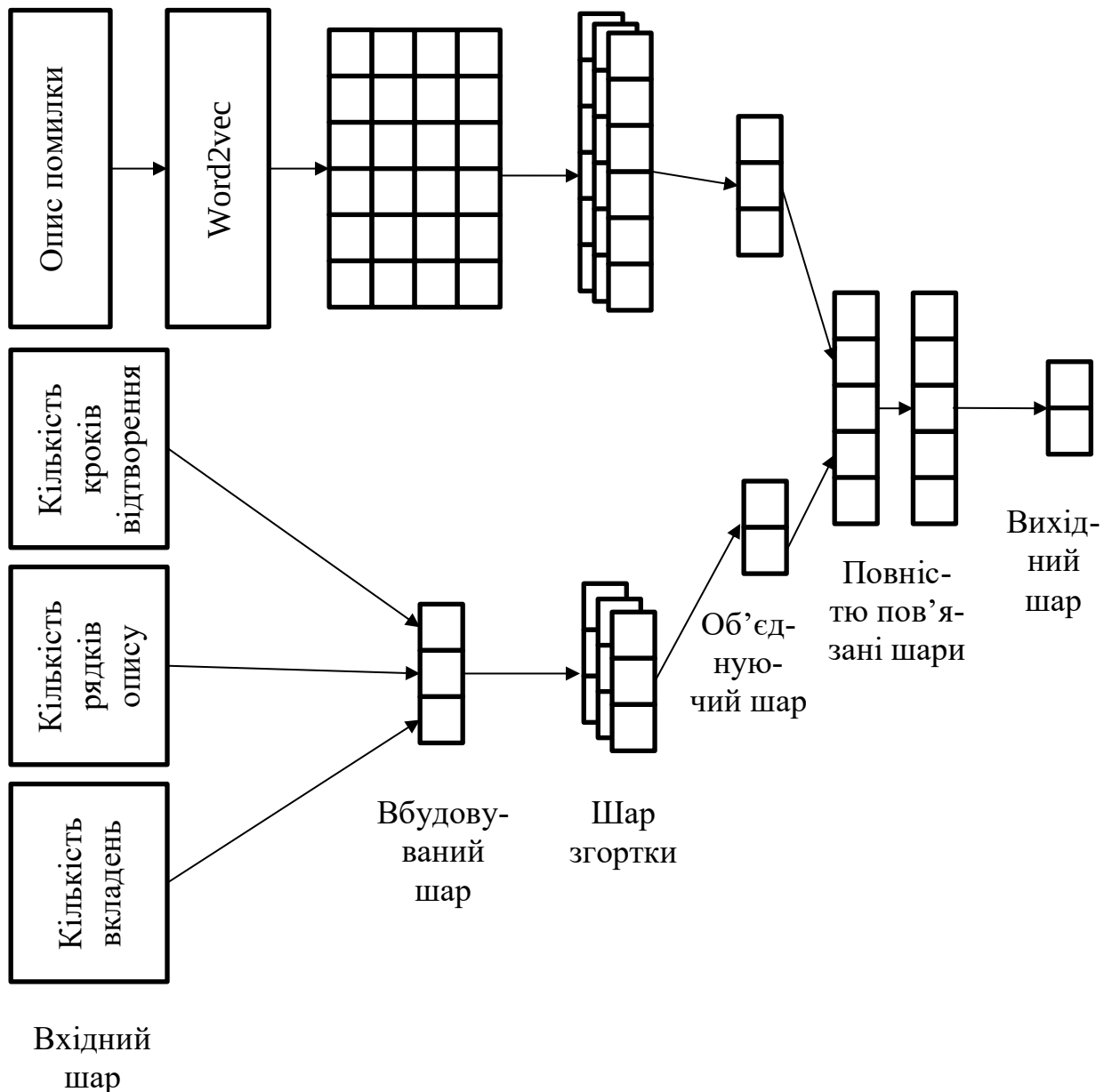


Рисунок 2.1 – Модель класифікації помилок

2.5 Висновки за розділом 2

Проаналізовано існуючі методи, які на даний момент використовуються для прогнозування довготривалих помилок в формі розв'язання задач прогнозування і класифікації.

Запропоновано метод класифікації помилок, який визначає, чи належить виявлена помилка до тих, які будуть потребувати повторного усунення, застосуванням моделі на основі конволюційних нейронних мереж і методу word2vec для оброблення текстових описів помилок, що дозволяє на основі отриманих даних менеджера проєктів приймати обґрунтовані рішення про виділення потрібних ресурсів.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Визначення функціональних вимог до програми

Для змоги практичного застосування ПЗ, що розробляється, потрібно, щоб воно, з одного боку, реалізовувало актуальні інструменти в цій практичній сфері, а з іншого боку, надавало особливості у вигляді передбачення можливості повторного розв'язання помилки. Тому до набору функціональних вимог до ПЗ було виділено:

- створення запису про нову помилку;
- редагування даних помилки;
- перегляд повних даних помилки;
- пошук даних помилки;
- перегляд помилок, внесених тестувальником;
- перегляд помилок, за які відповідальний розробник;
- перегляд помилок за проектом;
- перегляд помилок, над якими працює група розробників;
- перегляд групи помилок;
- визначення можливості повторного виникнення помилки;
- перегляд помилок, які можуть бути розв'язані не з першої спроби;
- перегляд помилок, які розв'язуються не вперше;
- перегляд ситуацій повідомлення про одну й ту саму помилку різними тестувальниками;
- визначення виявлених помилок як таких, що стосуються однієї помилки;
- з'єднання помилок в єдину групу;
- визначення команди розробників;
- перегляд помилок, що стосуються заданого тегу;
- перегляд помилок за модулем;
- перегляд помилок за заданою проблемою;
- перегляд помилок з заданими інтервалами часу роботи;

- перегляд повідомлень користувачів за помилкою;
- авторизація користувача;
- створення нового проєкту;
- перегляд проєктів;
- додавання модулів до проєкту;
- редагування даних проєкту та окремих модулів;
- публікація повідомлення про стан роботи над помилкою;
- створення нових тегів;
- створення вкладання за помилкою;
- перегляд вкладень за помилкою;
- визначення відповідального розробника за усунення помилки;
- визначення початку роботи розробника над помилкою;
- визначення завершення роботи розробника над помилкою;
- визначення успішного тестування помилки;
- визначення неуспішного тестування помилки.

Після визначення функціональних вимог до програми було реалізовано моделювання роботи користувача з програмою. Для цього було спочатку визначено потоки роботи користувачів та сформульовано передумови, основним з яких є авторизація облікового запису. Якщо користувач не авторизований, то він не може отримати доступ до інформації про будь-які помилки в проєктах компанії.

Функціональність, яка доступна тестувальнику (рис. 3.1), доступна і іншим працівникам компанії фактично. Тобто і розробник, і менеджер можуть також вносити дані про помилки. Розробнику ця функціональність доступна, оскільки він як учасник команди може знаходити помилки і таким чином виносити їх на загальну більш широку аудиторію, яка долучилася до створення цього проєкту загалом. Менеджеру ця функціональність потрібна, оскільки він може вносити інформацію, наприклад, про помилки, виявлені клієнтами.

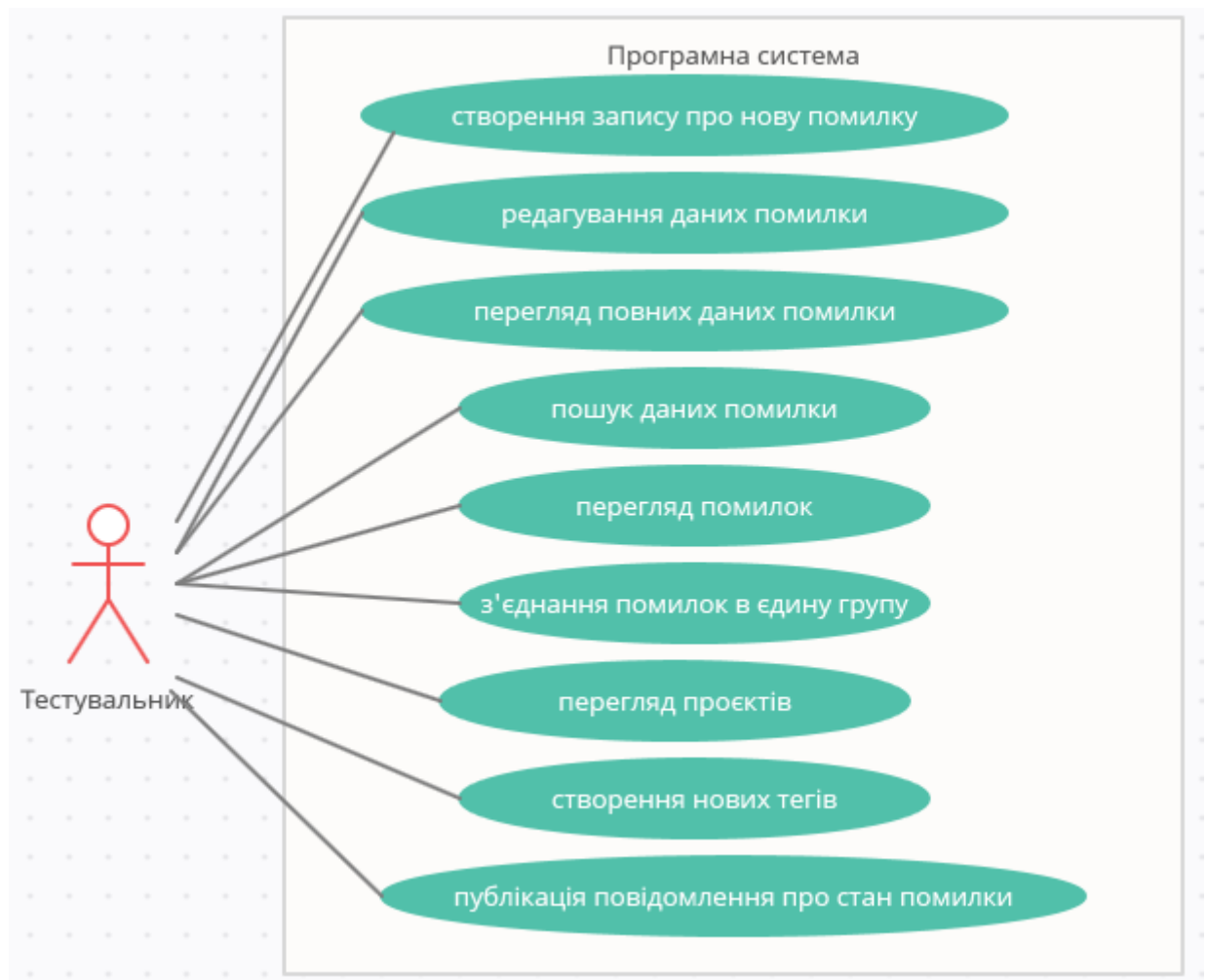


Рисунок 3.1 – Діаграма прецедентів тестувальника

Окрім того і тестувальнику, і розробнику, і менеджеру надаються загальні засоби перегляду помилок. Тобто загалом виявлені помилки можуть переглядати всі працівники компанії. Це необхідно для того, щоб можна було отримувати загальну інформацію і таким чином розуміти, які помилки повторюються, які пов'язані з іншими помилками.

Основною зоною відповідальності менеджера (рис. 3.2) є розподіл відповідальних за помилки, тому йому доступний прецедент перегляду окремо непризначених помилок, а також прецедент визначення відповідального за помилку. Окрім того йому доступні засоби формування команд розробників, внесення даних за проєктами і їх модулями.

Основною зоною відповідальності розробника (рис. 3.3) є робота з помилками, за які він відповідальний. Для цього йому доступний прецедент

перегляду призначений помилок, прецедент усунення помилки і прецедент приєднання до роботи над помилкою.



Рисунок 3.2 – Діаграма прецедентів менеджера



Рисунок 3.3 – Діаграма прецедентів розробника

3.2 Вибір засобів розробки

Основна функціональність, розглянута в підрозділі 3.1, передбачає доступ через вебзастосунок та потребу у великій і потужній кількості засобів роботи над відображеннями даних у потрібних формах. Тому розглянуто було можливість вибору мови С# як однієї з потужних, але близьких до стандартної системи С-подібних мов, де був наявний певний досвід, і мови Ruby, яка достатньо відрізняється, а тому є логічним вибором в якості альтернативи в галузі веброзробки (табл. 3.1).

Таблиця 3.1 – Обґрунтування вибору мови програмування для розробки вебзастосунку

Показник порівняння	С#	Ruby
Об'єктно-орієтована парадигма	Так	Так
Тип мови	Компільована	Інтерпретована
Продуктивність	Висока	Середня
Зручність створення великих застосунків	Висока	Середня
Простота створення інтерфейсу користувача	Висока	Висока
Підтримка розробників	Краща	Гірша

При порівнянні мов програмування слід виділити наступне:

– відносно створення інтерфейсів Ruby on Rails і ASP.NET Core працюють дуже схожим чином – використовують механізми шаблонів, який за своєю сутністю є сумішшю HTML і певної мови програмування: коли браузер надсилає запит, контролери подають шаблон із даними, після цього сервер на основі отриманої інформації відтворює сторінку і показує її користувачеві, у певному сенсі це дуже зручно, але, з іншого боку, це може

призвести до широкого змішування макета та логіки, що може бути проблемою в подальшому розвитку програми;

- продуктивність: за даним критерієм C# є безперечним переможцем, робота базових можливостей у C# відбувається набагато швидше, ніж у Ruby on Rails, на C# код компілюється в проміжний код, який потім запускається Common Language Runtime, окрім того за допомогою C# можна писати високомасштабований асинхронний код, з іншого боку, Ruby on Rails має досить повільний час обробки процесором в порівнянні з багатьма іншими мовами, що працює Ruby на шкоду, більше того, Ruby не компілюється і повністю інтерпретується під час виконання;

- зручність створення великих застосунків: творці Ruby on Rails намагаються керувати ним за допомогою конфігурації або парадигми неповторення, які заохочують повторне використання коду для зменшення прихованих залежностей і довжини коду, при цьому C# за допомогою .NET Core може значно зменшити кількість коду, необхідного для створення великих програм;

- підтримка розробників: мова C# має досить велику спільноту, окрім того компанія-розробник мови створює велику кількість матеріалів, тож говорити про нестачу матеріалів у даному випадку неможливо, у той же час знаходити матеріал стосовно роботи з мовою програмування Ruby складніше, що може не бути значною проблемою за наявності досвіду, але є значною проблемою за відсутності досвіду роботи з мовою [19].

Реалізацію вебзастосунку має бути виконано мовою програмування C#. Безпосередньо реалізацію моделі на основі КНМ та відповідного представленого методу з використанням wordvec має бути виконано за допомогою мови програмування Python, оскільки для даних задач на даний момент це інструмент, який найбільш адекватно відповідає задачам.

3.3 Визначення структури даних

Для збереження і структуризації даних було виконано проєктування бази даних. Для бази даних було визначено всі необхідні сутності, відношення між ними, зведено структуру бази даних до таблиці і відношень між ними.

Сутність Error призначена для збереження даних помилок, які було виявлено під час роботи або тестування ПЗ, представляючи у складі таблиці наступні поля:

- поле ідентифікатора ErrorId, що визначає первинний ключ;
- поле назви помилки Title (скорочене представлення для характеристики помилки);
- поле деталізованого опису помилки Detailing;
- поле номера проєкту ProjectId, до якого належить помилка;
- поле номера облікового запису ReporterId, за яким користувач повідомив про помилку;
- поле поточного стану помилки Status;
- поле номера облікового запису користувача AssigneeId, що призначений відповідальним за розв’язання помилки;
- поле критичності помилки Severity;
- поле модуля ModuleId, до якого належить помилка (значення може бути пропущеним, оскільки помилка може стосуватися проєкту загалом, а не конкретного модуля);
- поле ідентифікатора команди TeamId, що відповідальна за усунення помилки;
- поле класу проблеми ProblemclassId, до якого належить помилка;
- поле дати і часу виявлення помилки StartDate (або повторного виявлення помилки);
- поле дати і часу завершення роботи з помилкою EndDate;

- поле дати і часу, до якого має бути завершено роботу над помилкою, DueDate;
- поле RepeatPrediction, що зберігає результат передбачення можливої повторної роботи над помилкою;
- поле Repeat, що визначає, чи помилка розглядається вперше, чи це вже повторне повернення до помилки;
- поле ідентифікатора оригінальної помилки SrcErrorId, яке визначає дублювання інформації про помилку і посилається на помилку, робота над якою вже ведеться;
- поле Steps опису кроків, необхідних для відтворення помилки.

Таблиця Project зв'язана з таблицею Error зв'язком типу один-до-багатьох: один проєкт може мати від 0 до багатьох помилок, але кожна помилка може відноситись тільки до одного проєкту. Для цього таблиця Error містить поле зовнішнього ключа ProjectId.

Таблиця User зв'язана з таблицею Error зв'язком типу один-до-багатьох, при чому двічі, тобто через два поля: один користувач може повідомити про помилки у кількості від 0 до багатьох, але кожна помилка повинна бути визначена тільки одним користувачем. Якщо декілька користувачів виявили помилку, то вони створюють окремі повідомлення про помилки, а вони далі менеджером зв'язуються між собою, але одне повідомлення має тільки одного автора. Для цього таблиця Error містить поле зовнішнього ключа ReporterId. Окрім того у помилки може бути тільки один відповідальний користувач, але у кожного користувача може бути від 0 до багатьох помилок, за які він відповідає. Даний зв'язок реалізовано зовнішнім полем AssigneeId у складі таблиці Error.

Таблиця Module зв'язана з таблицею Error зв'язком типу один-до-багатьох: один модуль може мати від 0 до багатьох помилок, але кожна помилка може відноситись тільки до одного модуля або до жодного з них. Для цього таблиця Error містить поле зовнішнього ключа ModuleId.

Таблиця Team зв'язана з таблицею Error зв'язком типу один-до-багатьох: одна команда може працювати над усуненням від 0 до багатьох помилок, але кожна помилка може усуватися тільки однією командою. Для цього таблиця Error містить поле зовнішнього ключа TeamId.

Таблиця Problemclass зв'язана з таблицею Error зв'язком типу один-до-багатьох: один клас проблем може мати від 0 до багатьох прикладів конкретних проблем, але кожна помилка може стосуватися тільки одного класу проблем. Для цього таблиця Error містить поле зовнішнього ключа ProblemclassId.

Таблиця Error окрім того зв'язана з самою собою зв'язком типу один-до-багатьох: одна помилка може мати від 0 до багатьох дублювань, але кожна помилка може посилатися в цій якості тільки на 1 оригінальну помилку. Для цього таблиця Error містить поле зовнішнього ключа ErrorId.

Сутність State необхідна для збереження всіх виявлених станів помилки. Це дозволяє мати інформацію про попереднє виявлення помилки, про будь-яку зміну стану помилки. Таблиця має поля:

- поле ідентифікатору виявленого стану помилки StateId;
- поле ідентифікатора помилки ErrorId, якій належить інформація про даний стан;
- поле конкретного стану помилки Status;
- поле дати і часу початку роботи над даним станом помилки StartDate;
- поле дати і часу завершення роботи над даним станом помилки EndDate;
- поле дати і часу DueDate, до якого мало бути завершено роботу над помилкою;
- поле ідентифікатора облікового запису автора зміни стану помилки UserId;
- поле опису стану помилки Detailing;
- поле поточного рівня критичності помилки Severity.

Таблиця Error зв'язана з таблицею State зв'язком типу один-до-багатьох: одна помилка може мати від 0 до багатьох записів про свій стан, але кожен такий запис належить тільки одній помилці. Для цього таблиця State містить поле зовнішнього ключа ErrorId.

Таблиця User зв'язана з таблицею State зв'язком типу один-до-багатьох: один користувач може відповідати за різні помилки на різних етапах роботи над ними у кількості від 0 до багатьох, але за кожного стану тільки один користувач може відповідати за помилку. Для цього таблиця State містить поле зовнішнього ключа UserId.

Сутність Message необхідна для збереження спілкування розробників, тестувальників, менеджерів щодо виявленої помилки. Таким чином можна дізнаватися про поточну роботу. У даній сутності представлено повідомлення такого спілкування. Кожне спілкування відбувається в межах однієї конкретної помилки. Таблиця Message має поля:

- поле ідентифікатора повідомлення MessageId;
- поле тексту повідомлення Record;
- поле ідентифікатора користувача UserId, який створив дане повідомлення;
- поле дати і часу створення помилки CreateDate;
- поле ідентифікатора помилки ErrorId, до якого належить повідомлення.

Таблиця Error зв'язана з таблицею Message зв'язком типу один-до-багатьох: одна помилка може мати від 0 до багатьох повідомлень про себе, але кожне повідомлення належить тільки одній помилці. Для цього таблиця Message містить поле зовнішнього ключа ErrorId.

Таблиця User зв'язана з таблицею Message зв'язком типу один-до-багатьох: один користувач може створити від 0 до багатьох повідомлень про роботу над помилками, але кожне повідомлення може бути створене тільки одним користувачем. Для цього таблиця State містить поле зовнішнього ключа UserId.

Сутність Project визначає проєкт, над яким працює компанія, і містить поля:

- поле ідентифікатора проєкту ProjectId;
- поле назви проєкту Title;
- поле дати і часу початку над проєктом StartDate;
- поле ідентифікатора облікового запису UserId, який створив проєкт і є відповідальним за нього;
- поле дати і часу завершення роботи над проєктом EndDate;
- поле дати і часу DueDate, до якої має бути завершено роботу над проєктом;
- поле детального опису проєкту Detailing;
- поле стану роботи над проєктом Status.

Таблиця User зв'язана з таблицею Project зв'язком типу один-до-багатьох: один користувач може відповідати за різні проєкти у кількості від 0 до багатьох, але кожен проєкт повинен мати тільки одного відповідального. Для цього таблиця Project містить поле зовнішнього ключа UserId.

Сутність User визначає обліковий запис користувача і містить наступні поля:

- поле ідентифікатора облікового запису UserId;
- поле адреси електронної пошти Email (використовується замість логіна);
- поле паролю облікового запису Password;
- поле ідентифікатора департаменту DepartmentId, в якому працює користувач;
- поле посади користувача Position.

Таблиця Department зв'язана з таблицею User зв'язком типу один-до-багатьох: один користувач може працювати тільки в одному департаменті компанії, але кожен департамент може включати від 1 до багатьох працівників (тимчасово може бути 0 учасників). Для цього таблиця User містить поле зовнішнього ключа DepartmentId.

Сутність Department визначає департаменти компанії і містить наступні поля:

- поле ідентифікатора департаменту DepartmentId;
- поле назви департаменту Title.

Сутність Attachment визначає вкладення для помилки, що необхідні для роботи над цією помилкою. Таким чином можуть додаватися різні необхідні файли, на які потім можна посилатися в повідомленнях. Відповідна таблиця має поля:

- поле ідентифікатора вкладення AttachmentId;
- поле адреси, за якою розташовано файл вкладення, Source;
- поле ідентифікатора помилки ErrorId, до якої належить вкладення;
- поле ідентифікатора облікового запису UserId, який зберіг дане вкладення.

Таблиця Error зв'язана з таблицею Attachment зв'язком типу один-до-багатьох: одна помилка може мати від 1 до багатьох вкладень, але кожне вкладення може стосуватися лише однієї помилки. Для цього таблиця Attachment містить поле зовнішнього ключа ErrorId.

Таблиця User зв'язана з таблицею Attachment зв'язком типу один-до-багатьох: один користувач може додати від 1 до багатьох вкладень, але кожне вкладення може мати тільки одного автора. Для цього таблиця Attachment містить поле зовнішнього ключа UserId.

Сутність Module визначає модулі проєкту і містить поля:

- поле ідентифікатора модуля ModuleId;
- поле назви модуля Title;
- поле шляху до файлів модуля Source;
- поле ідентифікатора проєкту ProjectId, до якого належить модуль.

Таблиця Project зв'язана з таблицею Module зв'язком типу один-до-багатьох: один проєкт може мати від 0 (якщо деталізацію не виконано в застосунку) до багатьох модулів, але кожен модуль може належати тільки

одному проєкту. Для цього таблиця `Module` містить поле зовнішнього ключа `ProjectId`.

Сутність `Team` визначає команди, в межах яких усуваються виявлені помилки, визначаючи для цього у складі таблиці поля:

- поле ідентифікатора команди `TeamId`;
- поле назви команди `Title`;
- поле більш детального опису команди `Detailing`.

Для виділення загальної логіки помилки було виділено сутність класу проблеми `Problemclass`, структура таблиці якої містить поля:

- поле ідентифікатора класу `ProblemclassId`;
- поле назви класу проблеми `Title`.

Сутність `Tag` визначає тег, що співвідноситься з помилками. Теги необхідні зокрема для того, щоб зручно було виконувати пошук помилок шляхом з'єднання їх за певною міткою. Структура таблиці `Tag`:

- поле ідентифікатора тегу `TagId`;
- поле назви тегу `Title`.

Сутність `Photo` визначає фотографію, яка характеризує помилку. Відповідна таблиця має поля:

- поле ідентифікатора вкладки `PhotoId`;
- поле адреси, за якою розташовано файл фотографії, `Source`;
- поле ідентифікатора помилки `ErrorId`, до якої належить фотографія;
- поле коментаря до фотографії `Detailing`.

Таблиця `Error` зв'язана з таблицею `Photo` зв'язком типу один-до-багатьох: одна помилка може мати від 0 до багатьох зображень для опису виявленої проблеми, але кожне зображення може характеризувати тільки одну помилку. Для цього таблиця `Photo` містить поле зовнішнього ключа `ErrorId`.

Структура створеної бази даних на основі використання системи керування базами даних Microsoft SQL Server представлена на рис. 3.4.

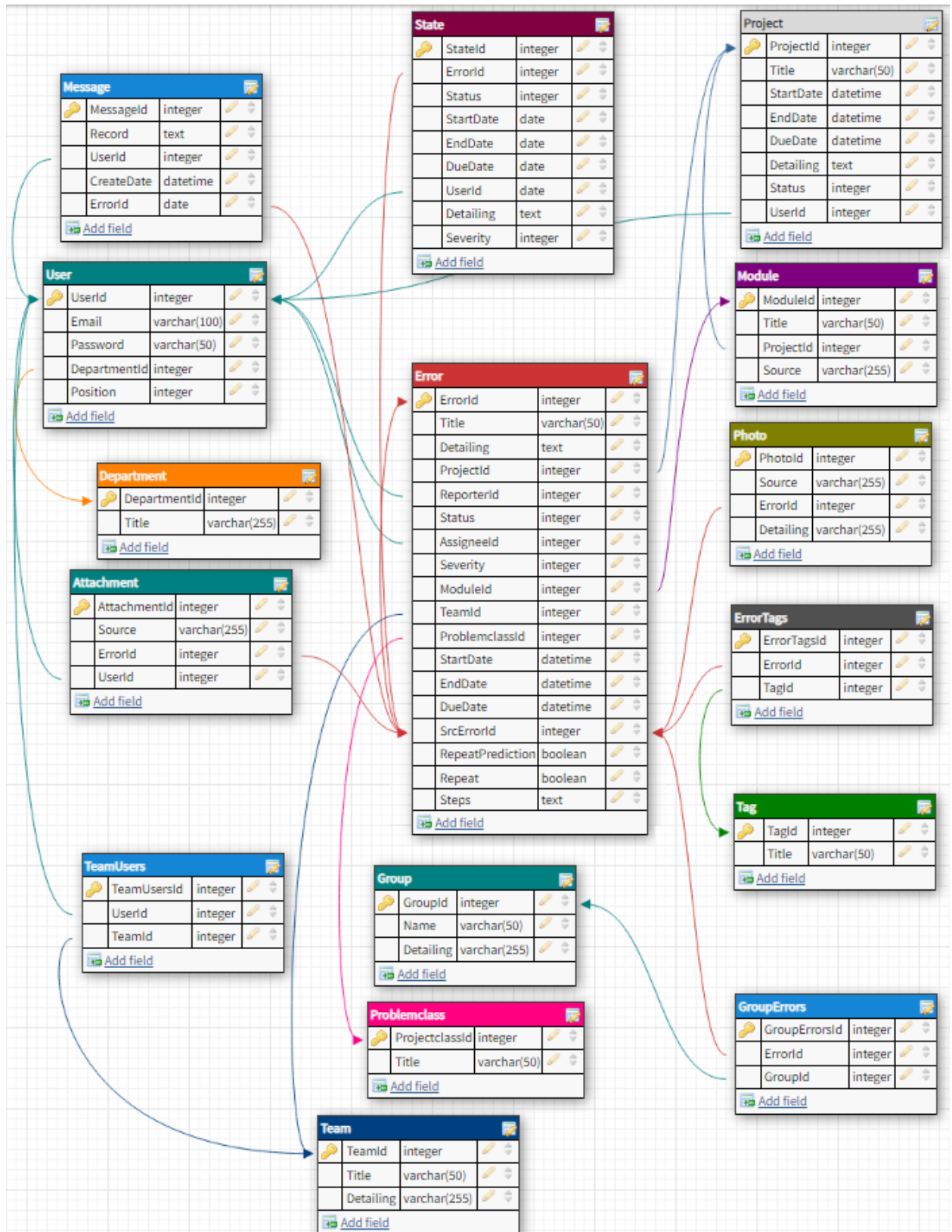


Рисунок 3.4 – Структура бази даних

Сутність Group визначає групи, в які об'єднуються подібні або пов'язані між собою помилки, визначаючи для цього у складі таблиці поля:

- поле ідентифікатора групи `GroupId`;
- поле назви групи `Title`;
- поле більш детального опису групи `Detailing`.

Зв'язок багато-до-багатьох між таблицями `Error` і `Tag` (кожна помилка може характеризуватися набором тегів від 0 до багатьох, а кожен тег може характеризувати від 0 до багатьох різних помилок) реалізований створенням проміжної таблиці `ErrorTag`, що містить поля:

- поле ідентифікатора `ErrorTagsId`;
- поле ідентифікатора помилки `ErrorId`;
- поле ідентифікатора тега `TagId`.

Зв'язок багато-до-багатьох між таблицями `Team` і `User` (кожна команда може включати від 1 до багатьох учасників (тимчасово у команді може бути 0 учасників), кожен користувач може входити до різних команд) реалізований створенням проміжної таблиці `TeamUser`, що містить поля:

- поле ідентифікатора `TeamUsersId`;
- поле ідентифікатора облікового запису користувача `UserId`;
- поле ідентифікатора команди `TeamId`.

Зв'язок багато-до-багатьох між таблицями `Group` і `Error` (кожна помилка може входити до складу від 0 до багатьох груп помилок, в кожна група може містити від 1 до багатьох помилок, але тимчасово група може не містити помилок) реалізований створенням проміжної таблиці `GroupError`, що містить поля:

- поле ідентифікатора `GroupErrorsId`;
- поле ідентифікатора групи помилок `GroupId`;
- поле ідентифікатора помилки `ErrorId`.

3.4 Структура програмного забезпечення

Структуру ПЗ було зпроектовано, використовуючи шаблон проєктування `Model-View-Controller` (рис. 3.5). У результаті основними

структурними складовими створеного ПЗ є моделі, контролери та представлення застосунку. Окрім того до складу застосунку входять його ресурси, тобто файли таблиць стилів, зображень, які використовуються при створенні сторінок, засоби бібліотеки Bootstrap.

Окремо у складі структури ПЗ виділено модуль класифікації помилок, в якому мовою Python має бути реалізовано представлений метод.

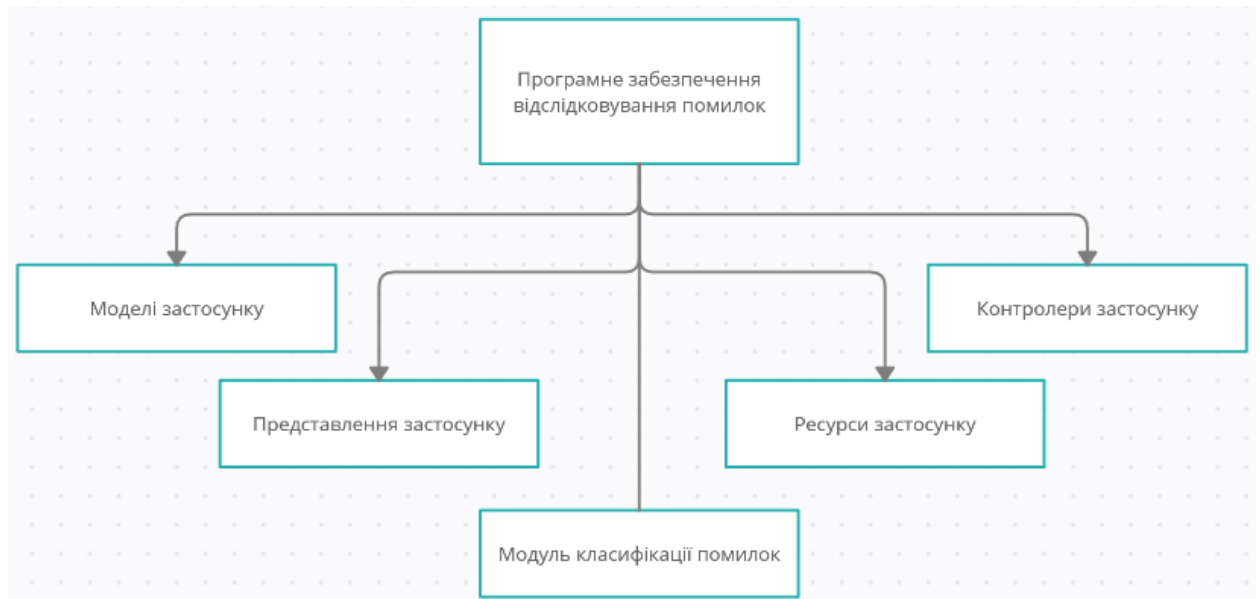


Рисунок 3.5 – Схема структури програми

3.5 Висновки за розділом 3

Визначені функціональні вимоги використано при моделюванні сценаріїв роботи користувачів з програмою. Обґрунтовано вибір мови програмування C# для розробки вебзастосунку, а програмну реалізацію представленого методу має бути виконано мовою Python. Детально описано визначено структуру бази даних та представлено структуру застосунку загалом.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Функціонування програми

Принцип організації функціонування програми зведено до забезпечення загальної роботи з програмою користувачів зі всіма наявними в компанії посадами, але з наданням або ненаданням доступу до певних функціональних можливостей в залежності від займаної посади (рис. 4.1).

Для початку роботи з програмою користувач має авторизуватися, без авторизації подальша робота неможлива. Вже на наступному кроці відображається інформація про виявлені помилки компанією в своїх програмних проєктах, а тому доступ до цієї інформації зовнішніми користувачами неможливий. Після успішної авторизації відразу відображається перелік виявлених помилок. Далі користувач може або обрати конкретну помилку і працювати над нею, або скористатися загальним меню.

З загального меню користувачу доступна можливість створення запису про нову помилку, пошуку помилок, перегляду тегів, перегляду проєктів і редагування, створення проєктів (останні два у випадку, якщо користувач – менеджер), окрім того менеджеру доступна можливість керування командами розробників.

Також користувачам доступна можливість перегляду переліків помилок. Ці переліки формуються за певними критеріями. Але не кожному користувачу доступні всі такі критерії, тобто всі можливі переліки помилок. Менеджеру потрібно працювати з переліками непризначених помилок, тому тільки йому доступний такий пункт меню. Розробнику важливо працювати з власними призначеними помилками. Тому тільки йому доступний такий перелік.

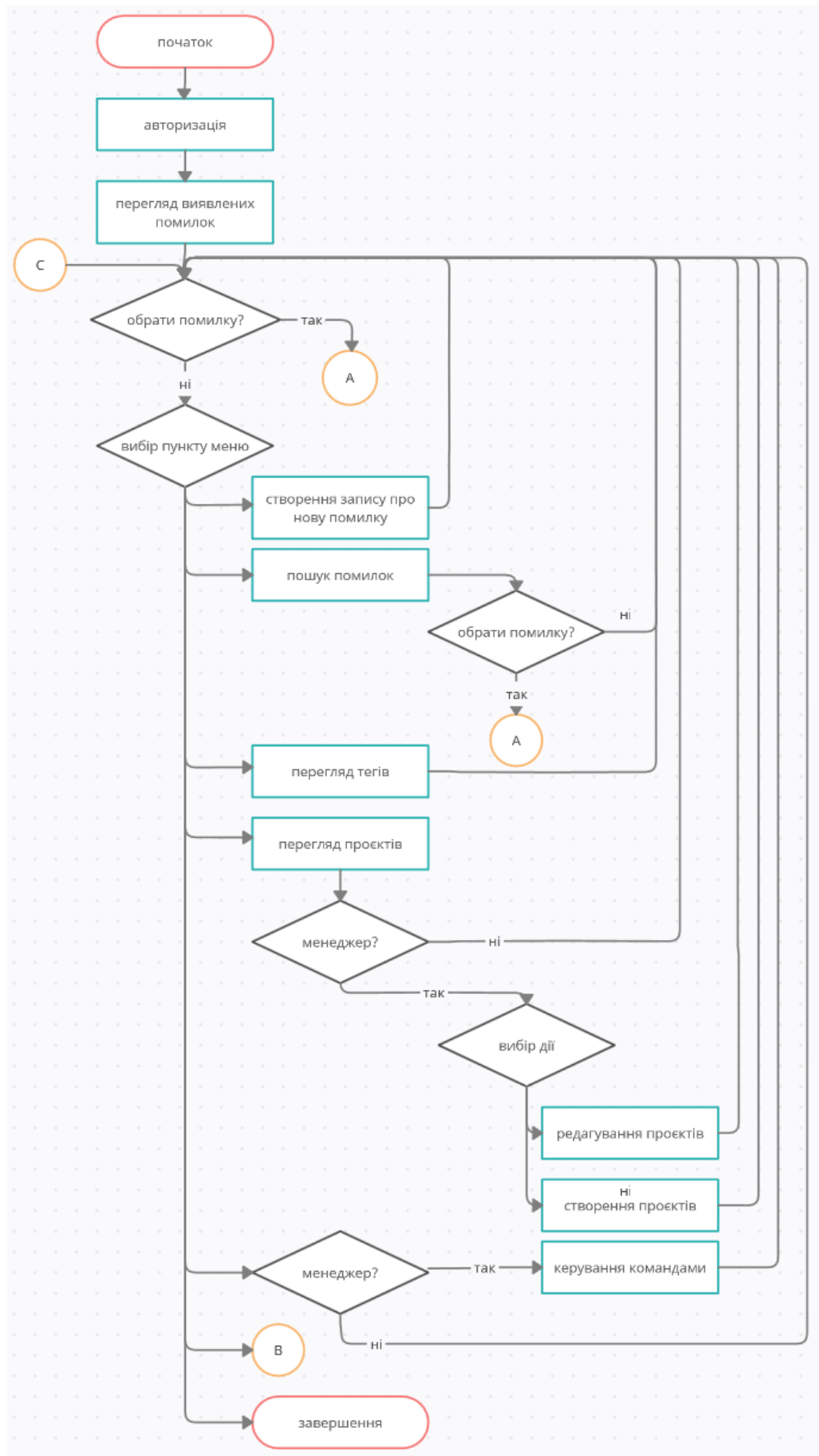


Рисунок 4.1 – Загальна схема функціонування програми

На рис. 4.2 представлено схему функціонування програми при роботі з окремими помилками.

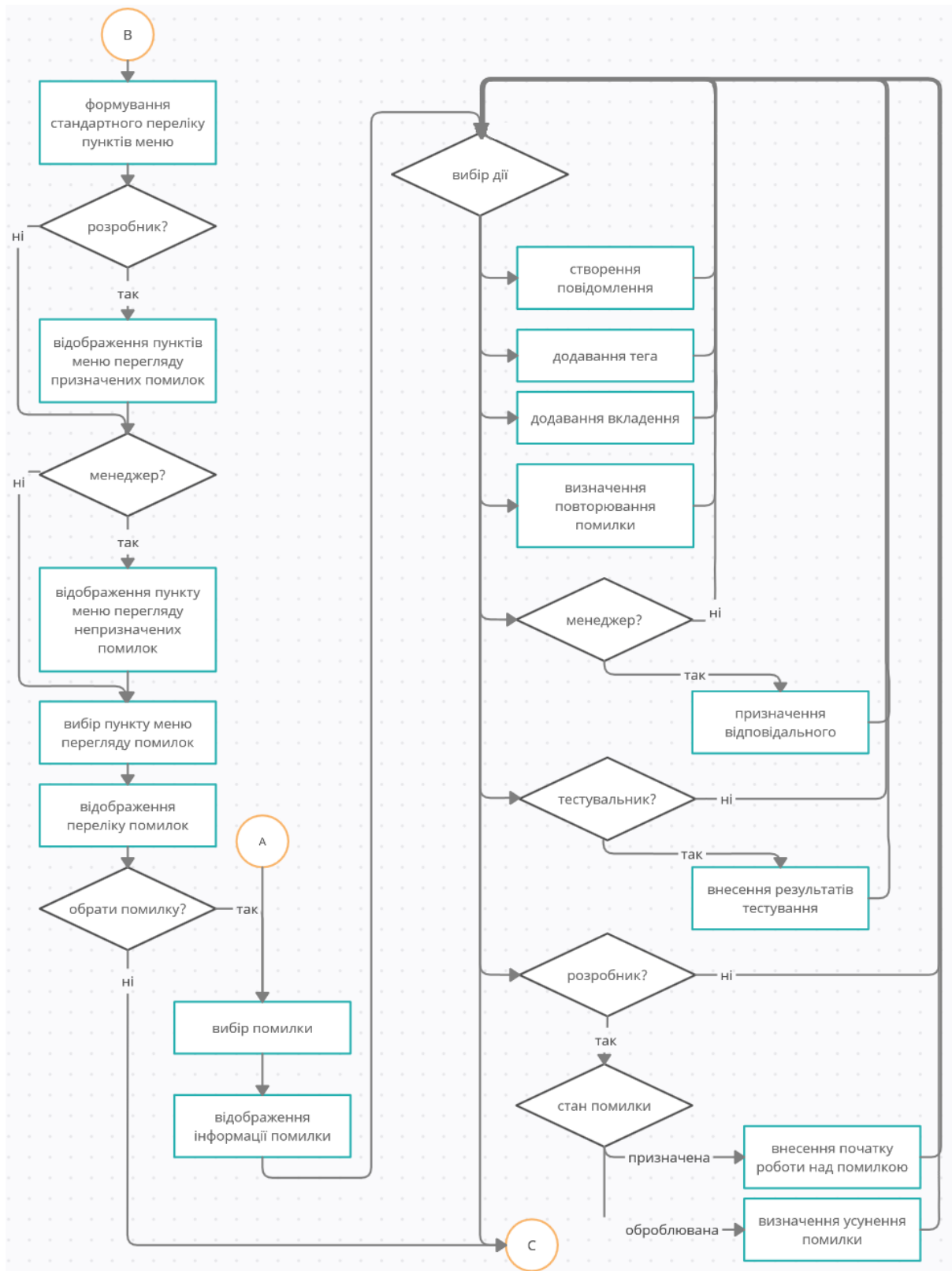


Рисунок 4.2 – Схема функціонування програми при роботі з окремими помилками

4.2 Опис модулів реалізованої програмної системи

Реалізацію програмної системи виконано у повній відповідності зі структурною схемою, представленою в розділі 3. Реалізацію роботи з даними виконано на основі підходу code-first, тобто спочатку створюється код, а на його основі визначається структура бази даних. Такий підхід дозволяє скоротити кількість коду, яку необхідно написати, і не реалізовувати кожен раз необхідний запит до бази даних вручну.

Основною сутністю, навколо якої відбувається робота програмної системи, є звичайно помилка. Саме навколо засобів роботи з нею на різних рівнях програмної системи і буде побудовано опис прийнятих і реалізованих рішень.

Модель Error для цього передбачає наступну структуру власних атрибутів:

- int ErrorId – атрибут ідентифікатора, визначений в коді за допомогою атрибута [Key], що встановлює його в якості ключа і позбавляє необхідності контролю за номером;
- string Title – атрибут назви помилки, який відзначений в коді атрибутами [Required], що вказує на необхідність обов’язкового визначення значення даного атрибута, та [MaxLength(50)], що визначає, що максимальна довжина назви може бути 50 символів;
- string Detailing – атрибут деталізованого опису помилки, який також вказаний як обов’язковий;
- string Steps – атрибут кроків, необхідних для відтворення помилки;
- int ProjectId – атрибут номера проєкту, до якого належить помилка, що організований як зовнішній ключ з таблиці Project, що розпізнається автоматично (ім’я даного атрибута співпадає з іменем ключа відповідної таблиці), є обов’язковим значенням;
- int ReporterId – атрибут номера облікового запису, за яким користувач повідомив про помилку: є зовнішнім ключем, але оскільки

таблиця Error зв'язана з таблицею User двічі, то ім'я даного атрибута не співпадає з іменем ключа відповідної таблиці, тому зв'язок встановлено, визначаючи за допомогою атрибута [ForeignKey("User")], є обов'язковим значенням;

- byte Status – атрибут стану помилки;
- int AssigneeId – атрибут номера облікового запису користувача, що призначений відповідальним за розв'язання помилки: визначений аналогічно атрибуту int ReporterId, тобто є зовнішнім ключем;
- byte Severity – атрибут критичності помилки;
- int ModuleId – атрибут номера модуля, до якого належить помилка, , є зовнішнім ключем;
- int TeamId – атрибут команди, що відповідальна за усунення помилки, є зовнішнім ключем;
- int ProblemclassId – атрибут класу проблеми, до якого належить помилка;
- DateTime StartDate – атрибут дати і часу виявлення помилки;
- DateTime EndDate – атрибут дати і часу завершення роботи з помилкою;
- DateTime DueDate – атрибут дати і часу, до якого має бути завершено роботу над помилкою;
- bool RepeatPrediction – атрибут, що зберігає результат передбачення можливої повторної роботи над помилкою;
- bool Repeat – атрибут, що визначає, чи помилка розглядається вперше, чи це вже повторне повернення до помилки;
- int SrcErrorId – атрибут ідентифікатора оригінальної помилки, яке визначає дублювання інформації про помилку і посилається на помилку, робота над якою вже ведеться: є зовнішнім ключем з таблиці Error, визначеним атрибутом [ForeignKey("Error")];

- Project Project – віртуальний атрибут, який зберігає повні дані проєкту, не відповідає стовпцю таблиці, тому позначений атрибутом [NotMapped];
- User Reporter – віртуальний атрибут, який зберігає повні дані користувача, який повідомив про помилку;
- User Assignee – віртуальний атрибут, який зберігає повні дані користувача, який є відповідальним за помилку на поточний момент;
- Module Module – віртуальний атрибут, який зберігає повні дані модуля, якому відповідає помилка;
- Team Team – віртуальний атрибут, який зберігає повні дані команди, яка працює над помилкою;
- Problemclass Problemclass – віртуальний атрибут, який зберігає повні дані класу проблеми;
- Error SrcError – віртуальний атрибут, який зберігає повні дані помилки, яка є першим встановленим фактом помилки;
- ICollection<Message> Messages – віртуальний атрибут, який зберігає повний перелік повідомлень щодо цієї помилки;
- ICollection<State> States – віртуальний атрибут, який зберігає повний перелік історії станів помилки;
- ICollection<Photo> Photos – віртуальний атрибут, який зберігає повний перелік фотографій;
- ICollection<Tag> Tags – віртуальний атрибут, який зберігає повний перелік тегів;
- ICollection<Attachment> Attachments – віртуальний атрибут, який зберігає повний перелік вкладень помилки;
- string StatusStr – атрибут, який визначає текстове представлення стану помилки;
- string SeverityStr – атрибут, який визначає текстове представлення критичності помилки;

– string DueDateStr – атрибут, який визначає текстове представлення дати, до якої має бути завершено роботу над помилкою, у форматі ДД.ММ.РРРР;

– string EndDateStr – атрибут, який визначає текстове представлення дати, коли було завершено роботу над помилкою, у форматі ДД.ММ.РРРР;

– string StartDateStr – атрибут, який визначає текстове представлення дати початку роботи над помилкою у форматі ДД.ММ.РРРР.

Критичність помилки на основі цілих чисел перетворюється у текстове представлення наступним чином:

- значення “1” відповідає критичності “низька”;
- значення “2” відповідає критичності “середня”;
- значення “3” відповідає критичності “висока”;
- значення “4” відповідає критичності “топ”;
- значення “0” та всі інші значення відповідають невстановленій критичності.

Стан помилки на основі цілих чисел перетворюється у текстове представлення наступним чином:

- значення “0” відповідає стану “відкритий”: повідомлення про помилку внесено, але обробка його ще не виконана;
- значення “1” відповідає стану “призначений”: визначено відповідального за усунення помилки;
- значення “2” відповідає стану “оброблюваний”: розробник, який усуває помилку, почав роботу над помилкою;
- значення “3” відповідає стану “вирішений”: розробник завершив роботу над помилкою і успішно усунув її, тобто до роботи може переходити тестувальник (який створив повідомлення про помилку);
- значення “4” відповідає стану “закритий”: тестування було успішно завершено;
- усі інші значення відповідають невідомому стану.

Оскільки представлення відображає дані на вебсторінці, то необхідно підготувати дані в потрібній формі. Для цього використовується контролер помилки, який організований класом `ErrorController`.

Метод `ErrorController` (`TuErrorTrackerContext` `tetc`) створює конструктор, який визначає контекст даних, за допомогою яких виконується завантаження даних з бази даних.

Метод `Task<IActionResult> ReporterErrors(int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, про які було повідомлено авторизованим у даний момент користувачем (тестувальником), відображаючи помилки, що відносяться до стану `mode`, виконуючи сортування за номером або за критичністю (в залежності від значення параметра `sortmode`), у другому випадку встановлюючи потрібний рівень критичності. Аналогічно організовано наступні методи. Усі вони ґрунтуються на методі `SortAndSeverity` класу `Processor`, де знаходяться базові засоби організації допоміжної роботи. Цей метод організовано наступним чином.

Якщо в якості значення параметра задано значення 5, то беруться всі помилки, незалежно від стану. За менших значень виділяються тільки ті помилки, які відповідають вказаному значенню стану:

```
if (mode < 5)
{
    errors = errors.Where(e => e.Status == mode);
}
```

Якщо параметр `sortmode` має істинне значення, а параметр `severity` має нульове значення, то виконується сортування за ступенем критичності, а у випадку рівного рівня критичності – за зменшенням ідентифікатора помилки, якщо ж при істинному значенні `sortmode`, а параметр `severity` має ненульове

значення, то вибираються тільки помилки з заданим рівнем критичності severity:

```

        if (sortmode && severity == 0)
        {
            errors = errors.OrderByDescending (e =>
e.Severity).ThenByDescending (e => e.Id);
        }
        else if (sortmode && severity != 0)
        {
            errors = errors.Where(e => e.Severity == severity);
            errors = errors.OrderByDescending (e => e.Id);
        }
        else
        {
            errors = errors.OrderByDescending(e => e.Id);
        }

```

Метод `Task<IActionResult> AssigneeErrors(int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, за які відповідальний авторизований користувач, використовуючи сортування помилок описаним вище способом.

Метод `Task<IActionResult> ProjectErrors(int? project, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок проєкту з номером `project`, використовуючи сортування помилок описаним вище способом.

Метод `Task<IActionResult> TeamErrors(int? team, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, за які відповідальна команда з номером `team`, відсортованим описаним вище способом.

Метод `Task<IActionResult> GroupErrors(int? group, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, що належать групі `group`, використовуючи сортування помилок описаним вище способом.

Метод `Task<IActionResult> PredictedErrors(bool? pred, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, щодо яких було визначено можливість повторного відкриття на основі методу, який було описано в розділі 2, якщо значення параметра `pred` істинне, і відповідно переліком помилок, які оцінюються як такі, що будуть розв'язані за один раз, якщо значення параметра `pred` хибне. При цьому виконується сортування помилок описаним вище способом.

Метод `Task<IActionResult> RepeatedErrors(int? rep, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, які фактично розв'язуються вдруге щонайменше (якщо значення параметра `rep` істинне), або переліком помилок, які розв'язуються вперше (якщо значення параметра `rep` хибне), відсортованим описаним вище способом.

Метод `Task<IActionResult> DoubleErrors(int? err, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, які дублюють помилку з номером `err`, відсортованим описаним вище способом.

Метод `Task<IActionResult> ModuleErrors(int? module, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, які стосуються заданого модуля `module`, відсортованим описаним вище способом.

Метод `Task<IActionResult> ProblemclassErrors(int? pc, int? mode, bool? sortmode, int? severity)` повертає представлення, заповнене переліком помилок, які належать до класу проблем `pc`, відсортованим описаним вище способом.

Метод `Task<IActionResult> SearchDateErrors(bool sdatemode, string sdate1, string sdate2, bool ddatemode, string ddate1, string ddate2, bool edatemode, string edate1, string edate2)` виконує пошук помилок з обмеженням на дати. Параметри `sdatemode`, `ddatemode`, `edatemode` своїм істинним або хибним значенням визначаються, чи буде виконуватися пошук за точно вказаною датою або за інтервалом. Параметри відповідно визначають спосіб визначення для дати початку, дедлайну або дати завершення роботи над помилками.

Метод `Task<IActionResult> TagError(string tagtitle)` повертає представлення, заповнене переліком помилок, які характеризуються тегом `tagtitle`.

Метод `Task<IActionResult> SearchError(string errtitle, int? mode, bool? sortmode, int? severity)` виконує пошук помилки з назвою `errtitle`, повертаючи представлення, заповнене відповідним переліком помилок, відсортованим описаним вище способом.

Метод `Task<IActionResult> Details(int? errorid)` повертає представлення, заповнене повною інформацією про помилку з номером `errorid`.

Метод `IActionResult Create()` виконує початкове заповнення представлення створення повідомлення про помилку, надаючи зокрема перелік проєктів, модулів, класів проблем, що використовуватимуться для вибору відповідних значень за помилкою при збереженні її.

Метод `Task<IActionResult> Create([Bind("Id, Title, Detailing, ProjectId, ModuleId, ProblemclassId, Tags, Steps")] Error error)` виконує збереження нової помилки з заданими назвою `Title`, описом `Detailing`, номером проєкту `ProjectId`, номером модуля `ModuleId`, класом проблеми `ProblemclassId` та переліком тегів `Tags`.

Метод `Task<IActionResult> Edit(int? errorid)` виконує початкове заповнення представлення редагування помилки з номером `errorid`.

Метод `Task<IActionResult> Edit(int errorid, [Bind("Id, Title, Detailing, ProjectId, StatusId, AssignnessId, ModuleId, ProblemclassId, TeamId, SrcErrorId, Severity, EndDate, DueDate, Repeat, Tags")] Error error)` виконує збереження відредагованої помилки з заданими назвою `Title`, описом `Detailing`, номером проєкту `ProjectId`, станом `StatusId`, відповідальним за вирішення помилки `AssignnessId`, номером модуля `ModuleId`, класом проблеми `ProblemclassId`, відповідальною командою `TeamId`, оригінальним повідомленням про помилку `SrcErrorId`, критичністю `Severity`, датою і часом завершення роботи над помилкою `EndDate`, дедлайну помилки `DueDate`, визначенням повторюваності помилки `Repeat` та переліком тегів `Tags`.

Для створення представлень, у яких відображається вміст даних, застосовано вільний шаблон [20].

4.3 Висновки за розділом 4

У результаті розроблено класи моделей і контролерів застосунку, реалізовано представлення застосунку. Основні реалізовані засоби, рішення, прийняті під час цієї реалізації, а також особливості реалізації на основі підходу “code first” представлено в даному розділі. Окремо описано порядок функціонування програми за допомогою схеми функціонування програми.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програми

ПЗ призначене для збереження будь-яких виявлених у програмних проєктах компанії помилок, фіксації подальших етапів роботи над усуненням помилок, керування роботою над помилками у складі компанії.

5.2 Умови виконання програми

Для виконання програми потрібно забезпечити на сервері наявність наступних апаратних засобів:

- процесор: щонайменше два ядра, мінімальна частота – 2 ГГц;
- оперативний запам'ятовуючий пристрій – не менше 4 Гб;
- жорсткий диск: вільний простір не менше 1 Гб.

Окрім того має бути інстальовано систему Microsoft SQL Server, засоби платформи .NET.

5.3 Виконання програми

Після звернення до вебсервера користувачу відображається форма авторизації облікового запису. Виконавши успішно авторизацію, користувачу відкривається сторінка, вигляд якої залежить від того, на якій саме посаді працює користувач.

Розробник працює з вебзастосунком, використовуючи в якості одного з базових засобів сторінку перегляду помилок (рис. 5.1). На даній сторінці відображається велика кількість способів роботи з підмножиною помилок.

Розробник може за допомогою кнопки “Власні” переглянути перелік помилок, які були ним внесені в систему. Дана кнопка доступна і іншим

користувачам. Звичайно, що даний пункт є базовим для роботи тестувальника.

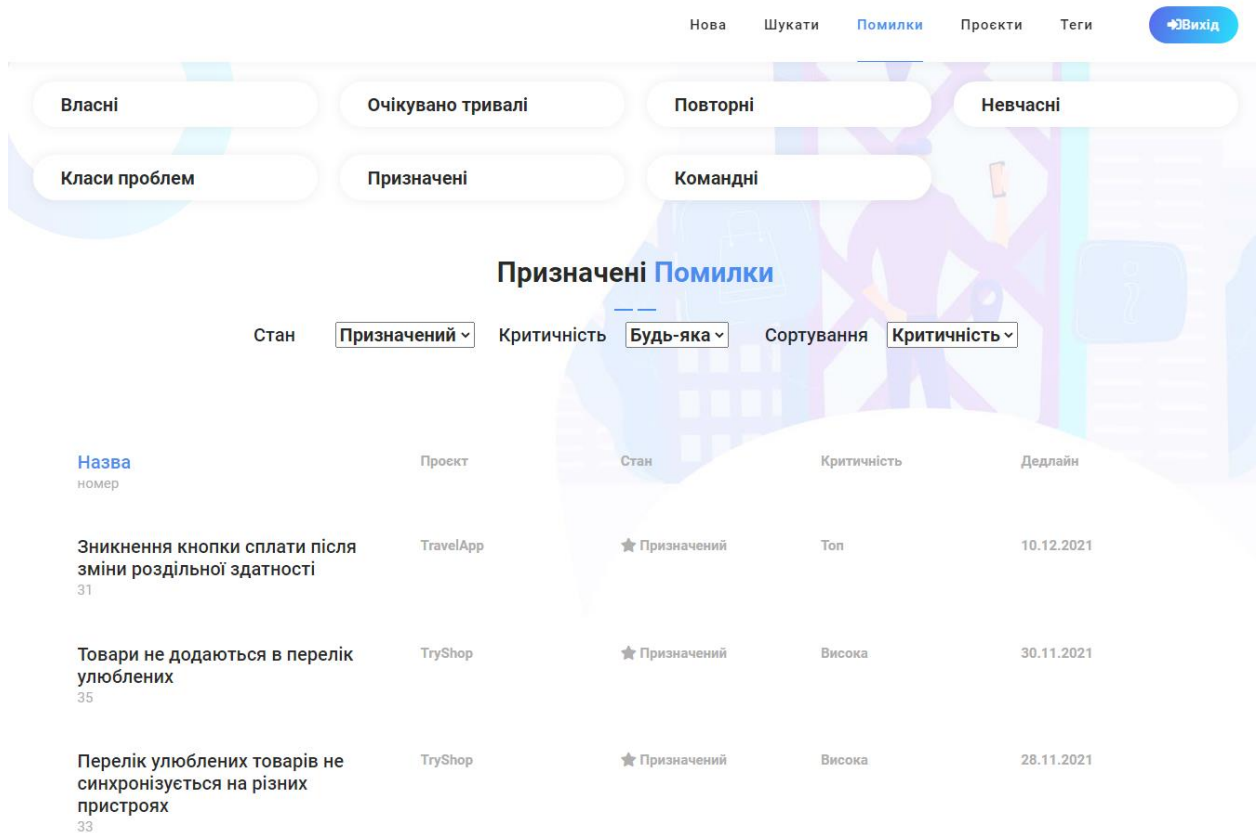


Рисунок 5.1 – Відтворення призначених розробнику помилок

Розробник може за допомогою кнопки “Очікувано тривалі” переглянути перелік помилок, які були за результатами класифікації на основі представленого методу класифікації помилок віднесені до помилок, які можуть призвести до необхідності повторної роботи з ними через певний час. Дана кнопка доступна і іншим користувачам.

Розробник може за допомогою кнопки “Повторні” переглянути перелік помилок, які вже на даний момент вже розглядалися повторно. Дана кнопка доступна і іншим користувачам.

Розробник може за допомогою кнопки “Невчасні” переглянути перелік помилок, які не вдалося усунути до встановленої дати, до якої

помилка повинна бути завершена. Дана кнопка доступна і іншим користувачам.

Розробник може за допомогою кнопки “Клас проблеми” переглянути перелік помилок за заданим класом. Для цього розробник обирає потрібний клас з доступних з переліку класів. Дана кнопка доступна і іншим користувачам.

При натисканні на відповідну кнопку відображається таблиця з відповідними помилками. За сформованим переліком можна задати потрібний стан помилок, обмежуючи таким чином сформований перелік. Для цього призначений відповідний випадаючий перелік. Можна також обрати потрібний рівень критичності за допомогою відповідного випадаючого переліку. Можна також не обмежувати рівень критичності. Окрім того можна обрати спосіб сортування помилок.

Приклад відтворення очікувано тривалих помилок на рис. 5.2 представлено для менеджера.

Назва номер	Проект	Стан	Критичність	Дати	Дедлайн
Відсутній доступ до пункту інсталяції плагінів 40	TravelApp	Відкритий	немає	26.11.2021 - не визначено	не визначено
Вміст кошика періодично губиться 37	TryShop	Відкритий	немає	26.11.2021 - не визначено	не визначено

Рисунок 5.2 – Відтворення очікувано тривалих помилок за прогнозом

У таблиці стан помилок відображається зірочками. В залежності від значення стану відображається відповідна кількість зірочок.

З переліку помилок можна натиснути на назву потрібної помилки, тоді відкриється сторінка роботи з обраною помилкою (рис. 5.3). Звідси доступні відповідні засоби керування помилкою, а також вся інформація про помилку.

The screenshot shows a web interface for error management. At the top, there is a navigation bar with links: Нова, Шукати, Помилки, Проекти, Теги, and a blue button labeled 'Вихід'. The main content area has a blue background with a white title 'Помилка № 50: Втрата Поточного Замовлення При Втраті Інтернет-З'єднання'. Below the title is a descriptive text: 'Помилка виникає в мобільному застосунку під Android, якщо є сформоване поточне замовлення, але в цей момент зникає інтернет-з'єднання. Тоді при його відновленні замовлення разом зі складом корзини зникає.' Below this text are three buttons: 'Відтестовано', 'Групувати', and 'Повторення'. A table follows with four columns: Стан, Дані проекту, Вирішення, and Теги. The table contains one row of data. Below the table is a section titled 'Контакти' with email addresses. At the bottom, there is a list of steps for reproduction and a row of buttons: 'Повідомлення', 'Групи', 'Однакові', 'Вкладення', and 'Фото'.

Стан	Дані проекту	Вирішення	Теги
Вирішений	Проект: TryShop Модуль: TryShopAndroidApp Висока критичність Клас: втрата даних	Початок: 26.11.2021 Завершення: не визначено Дедлайн: 30.11.2021 Вперше розглядається: Так Прогноз повтору: Ні	#basket #order

Контакти
Заявник: zhuk@pomylok.net
Відповідальний: bestdeveloper@pomylok.net

Кроки для відтворення:
1. Відкрити застосунок.
2. Додати товари в корзину.
3. Відключити інтернет-з'єднання.
4. Обрати одну з функцій в меню.
5. Підключити інтернет-з'єднання.
6. Відкрити корзину.

Рисунок 5.3 – Відтворення інформації про помилку

За допомогою кнопок “Повідомлення”, “Групи”, “Однакові”, “Вкладення”, “Фото” можна внизу сторінки відтворити відповідні дані.

Перегляд помилок доступний з меню за допомогою пункту “Помилки”. Пункт меню “Нова” дозволяє створити запис про нову помилку. Пункт меню “Шукати” дозволяє виконувати пошук помилки за її назвою.

Пункт меню “Проекти” дозволяє переглянути перелік проєктів компанії. Звідси можна перейти до переліку модулів проєкту, відобразити перелік помилок за проєктом загалом або за обраним модулем. Менеджер може також створювати нові проєкти звідси за допомогою відповідної кнопки.

Пункт меню “Теги” дозволяє переглянути перелік тегів та перейти до переліку помилок за обраним тегом.

Додатково до цих пунктів менеджеру доступний пункт “Команди”, який дозволяє створювати нові команди розробників, а також редагувати існуючі.

5.4 Експериментальне дослідження класифікації помилок

Експериментальне дослідження було реалізовано для розв’язання проблеми класифікації помилок щодо вирогідності їх повторного виникнення на основі вибірки даних, головним чином сформованої з вибірки [21]. Дана вибірка включає помилки, які були виявлені у наступних вільних проєктах: Eclipse, Freedesktop, GCC, Gnome, Mozilla та WinHQ. Загалом базова вибірка складалась з більше ніж 50 тисяч помилок.

З початкової вибірки було створено вибірку, екземпляри якої визначені на основі структури, сформульованої в розділі 2. За кожною помилкою було виявлено, чи з’являлось повідомлення про неї знову. Таким чином визначався цільовий клас кожного екземпляру.

Для класифікації було застосовано наступні методи і моделі (табл. 5.1):

- наївний байєсів класифікатор;
- метод k-середніх;
- метод опорних векторів;
- багат шарова нейронна мережа;
- представлений в роботі метод.

Таблиця 5.1 – Результати експериментального дослідження

Метод або модель класифікації	Точність класифікації
Наївний байєсів класифікатор	0,48
Метод k-середніх	0,65
Метод опорних векторів	0,67
Багатошарова нейронна мережа	0,72
Представлений метод	0,91

Найкращі результати класифікації були отримані за допомогою представленого методу, модель в основі якого дозволила отримати достатньо високу точність класифікації, яка окрім того значно перевищила результати інших методів.

5.5 Висновки за розділом 5

Розглянуто виконання основних функцій, потрібних користувачам в програмі. Проведено експериментальне дослідження проблеми класифікації помилок. Представлений метод дозволив отримати найкращі результати.

ВИСНОВКИ

У роботі виконано розроблення ПЗ керування відслідковуванням помилок для підвищення обґрунтованості прийнятих менеджером рішень. Для цього було виконано усі завдання роботи і досягнуто наступні результати:

- було виконано аналіз проблеми і програмних засобів, які використовуються для відслідковування помилок, у результаті чого виявлено, що вони не надають можливість інтелектуальної класифікації помилок на такі, які будуть потребувати повторного усунення, виконано дослідження існуючих методів, які використовуються для прогнозування часу усунення помилок і класифікації помилок за часом усунення, і розроблено власний метод керування відслідковуванням помилок;

- було виконано проєктування ПЗ, для чого було виконано моделювання роботи тестувальників, менеджерів проєктів і розробників з програмою, виконано вибір мови програмування C# на основі технології ASP.NET для реалізації вебзастосунку та мови програмування Python для реалізації запропонованого методу, визначено структуру бази даних, структуру програми загалом;

- було виконано програмну реалізацію системи керування відслідковуванням помилок на основі підходу code first, що дозволило оптимізувати обсяг коду щодо роботи з даними.

Розроблений програмний застосунок може використовуватися для внесення даних про виявлені помилки, керування процесом роботи над їх усуненням, надаючи для цього автоматизований засіб визначення потенційної можливості повторного повернення до усунення помилки, групування помилок між собою, визначення дублювань помилок.

Науковою новизною роботи є запропонований метод класифікації помилок, який визначає, чи належить виявлена помилка до тих, які будуть потребувати повторного усунення, застосуванням моделі на основі

конволюційних нейронних мереж і методу word2vec для оброблення текстових описів помилок, що дозволяє на основі отриманих даних менеджера проєктів приймати обґрунтовані рішення про виділення потрібних ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 17 Best Bug Tracking Tools: Defect Tracking Tools Of 2021 [Electronic resource]. – Access mode : <https://www.softwaretestinghelp.com/popular-bug-tracking-software/>.
2. How To Write A Good Bug Report? Tips And Tricks [Electronic resource]. – Access mode : <https://www.softwaretestinghelp.com/how-to-write-good-bug-report/>.
3. Online Project Management Software for Developers Backlog [Electronic resource]. – Access mode : <https://backlog.com/>.
4. Best Bug Tracking Software [Electronic resource]. – Access mode : <https://www.crazyegg.com/blog/best-bug-tracking-software/>.
5. Home Bugzilla [Electronic resource]. – Access mode : <https://www.bugzilla.org/>.
6. Best Bug Tracking Tools & Visual Feedback Software | BugHerd [Electronic resource]. – Access mode : <https://bugherd.com/>.
7. 26 Best Bug Tracking Tools (Free and Paid) for 2021 [Electronic resource]. – Access mode : <https://www.softwaretestingmaterial.com/popular-defect-tracking-tools/>.
8. Krotov, S. Top 11 Issue Tracking Software: Top Bug/Issue Tracking Tools [Electronic resource] / S. Krotov. – Access mode : <https://cillax.com/top-11-issue-tracking-software-top-bug-issue-tracking-tools.html>.
9. Bhattacharya, P. Bug-fix Time Prediction Models [Text] : Can We Do Better? / Pamela Bhattacharya, Iulian Neamtiu // MSR '11 : Proceedings of the 8th Working Conference on Mining Software Repositories, May 2011. – New York : Association for Computing Machinery, 2011. – Pp. 207-210.
10. Abdelmoez, W. Bug Fix-Time Prediction Model Using Naïve Bayes Classifier [Text] / W. Abdelmoez, M. Kholief, F. M. Elsalmy // ICCTA 2012, 13-15 October 2012, Alexandria, Egypt. – Piscataway : IEEE, 2012. – Pp. 167-172.
11. Ardimento, P. Knowledge extraction from on-line open source bug

tracking systems to predict bug-fixing time [Text] / Pasquale Ardimento, Andrea Dinapoli // WIMS '17: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, June 2017. – Article 7. – Pp. 1-9.

12. Sharma, M. Multi-attribute dependent bug severity and fix time prediction modeling [Text] / M. Sharma, M. Kumari, V. B. Singh // International Journal of System Assurance Engineering and Management. – 2019. – Vol. 10 (5). – Pp. 1328-1352.

13. Gomes, L. A. On the prediction of long-lived bugs: An analysis and comparative study using FLOSS projects [Text] / Luiz Gomes, Ricardo Torres, Mario Cortes // Information and Software Technology. – 2021. – Issue 132. – Article 106508. – Pp. 1-19.

14. Convolutional Neural Network Tutorial [Electronic resource]. – Access mode : <https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network>.

15. Goodfellow, I. Deep Learning [Text] / Ian Goodfellow, Yoshua Bengio, Aaron Courville. – Cambridge : MIT Press, 2016. – 326 p.

16. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation [Text] / M. V. Valueva, N. N. Nagornov, P. A. Lyakhov, G. V. Valuev, N. I. Chervyakov // Mathematics and Computers in Simulation. – 2020. – Vol. 177. – Pp. 232–243.

17. Agarwal, N. The Ultimate Guide To Different Word Embedding Techniques In NLP [Electronic resource] / N. Agarwal. – Access mode : <https://www.kdnuggets.com/2021/11/guide-word-embedding-techniques-nlp.html>.

18. Efficient Estimation of Word Representations in Vector Space [Electronic resource] / T. Mikolov, K. Chen, G. Corrado, J. Dean. – Access mode : <https://arxiv.org/abs/1301.3781>.

19. Karczewski, D. Ruby on Rails vs C# – Which Technology Should You Choose? [Electronic resource] / D. Karczewski. – Access mode : <https://www.ideamotive.co/blog/ruby-on-rails-vs-c>.

20. Chain App Dev Template [Electronic resource]. – Access mode :

<https://templatemo.com/tm-570-chain-app-dev>.

21. Gomes, L. A Dataset for Long-lived Bug Prediction in FLOSS [Electronic resource] / Luiz Gomes, Ricardo Torres, Mario Cortes. – Access mode : <https://data.mendeley.com/datasets/v446tfssgj/2>.

ДОДАТОК А
Технічне завдання

Вступ

Робота над помилками під час життєвого циклу ПЗ є важливим завданням, адже без реалізації цього процесу обійтися неможливо. Помилки можуть виявлятися в процесі створення програми, після її впровадження. І кожний такий випадок потрібно зафіксувати, щоб не загубити, обробити належним чином. Для цього і має бути розроблене відповідне ПЗ.

A.1 Підстави для розробки

Дипломну кваліфікаційну роботу магістра має бути виконано за темою “Дослідження та програмна реалізація методів керування відслідковуванням помилок”, яку було затверджено наказом № 435 від 5 листопада 2021 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

ПЗ призначене для збереження будь-яких виявлених у програмних проєктах компанії помилок, фіксації подальших етапів роботи над усуненням помилок, керування роботою над помилками у складі компанії.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

ПЗ повинно реалізовувати функції:

- створення запису про нову помилку;
- редагування даних помилки;
- перегляд повних даних помилки;
- пошук даних помилки;
- перегляд помилок, внесених тестувальником;

- перегляд помилок, за які відповідальний розробник;
- перегляд помилок за проєктом;
- перегляд помилок, над якими працює група розробників;
- перегляд групи помилок;
- визначення можливості повторного виникнення помилки;
- перегляд помилок, які можуть бути розв’язані не з першої спроби;
- перегляд помилок, які розв’язуються не вперше;
- перегляд ситуацій повідомлення про одну й ту саму помилку різними тестувальниками;
- визначення виявлених помилок як таких, що стосуються однієї помилки;
- з’єднання помилок в єдину групу;
- визначення команди розробників;
- перегляд помилок, що стосуються заданого тегу;
- перегляд помилок за модулем;
- перегляд помилок за заданою проблемою;
- перегляд помилок з заданими інтервалами часу роботи;
- перегляд повідомлень користувачів за помилкою;
- авторизація користувача;
- створення нового проєкту;
- додавання модулів до проєкту;
- редагування даних проєкту та окремих модулів;
- публікація повідомлення про стан роботи над помилкою;
- створення нових тегів;
- перегляд проєктів;
- створення вкладання за помилкою;
- перегляд вкладень за помилкою;
- визначення відповідального розробника за усунення помилки;
- визначення початку роботи розробника над помилкою;
- визначення завершення роботи розробника над помилкою;

- визначення успішного тестування помилки;
- визначення неуспішного тестування помилки.

A.3.2 Вимоги до надійності

Для забезпечення надійного функціонування програмного забезпечення необхідна постійна підтримка віддалених запитів користувачів як через локальну мережу, так і через мережу Інтернет. Для цього програма має розташовуватися на виділеному для цього сервері, який повинен функціонувати постійно і обробляти запити до вебсервера та сервера бази даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби, за допомогою яких організовується підтримка роботи серверної частини програмної системи, повинні відповідати наступним вимогам:

- процесор: щонайменше два ядра, мінімальна частота – 2 ГГц;
- оперативний запам'ятовуючий пристрій – не менше 4 Гб;
- жорсткий диск: вільний простір не менше 1 Гб.

A.4 Порядок контролю та приймання

Контроль виконання роботи здійснюється за календарним планом роботи. Під час приймання роботи здійснюється перевірка відповідності визначеним функціональним характеристикам програми.

ДОДАТОК Б
Текст програми

Б.1 Текст файла Error.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace TuErrorTracker.Models
{
    public class Error
    {
        [Key]
        public int ErrorId
        {get; set;}
        [Required]
        [MaxLength(50)]
        public string Title
        {get; set;}
        [Required]
        public string Detailing
        {get; set;}
        [Required]
        public string Steps
        {get; set;}
        [Required]
        public byte Status
        {get; set;}
        [Required]
        public int ProjectId
        {get; set;}
        [Required]
        [ForeignKey("User")]
        public int ReporterId
        {get; set;}
        [Required]
        [ForeignKey("User")]
        public int AssigneeId
        {get; set;}
        public int ModuleId
        {get; set;}
        public int TeamId
    }
}

```

```

{get; set;}
public int ProblemclassId
{get; set;}
[ForeignKey("Error")]
public int SrcErrorId
{get; set;}
public byte Severity
{get; set;}
[Required]
public DateTime StartDate
{get; set;}
public DateTime EndDate
{get; set;}
public DateTime DueDate
{get; set;}
public bool RepeatPrediction
{get; set;}
public bool Repeat
{get; set;}
[NotMapped]
public string StartDateStr
{
    get
    {
        return this.StartDate == null ? "не визначено" :
((DateTime)this.StartDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string EndDateStr
{
    get
    {
        return this.EndDate == null ? "не визначено" :
((DateTime)this.EndDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string DueDateStr
{
    get
    {

```

```

        return this.DueDate == null ? "не визначено" :
((DateTime)this.DueDate).ToString("dd.MM.yyyy");
    }
}
public virtual Project Project
{get; set;}
public virtual User Reporter
{get; set;}
public virtual User Assignee
{get; set;}
public virtual Module Module
{get; set;}
public virtual Team Team
{get; set;}
public virtual Problemclass Problemclass
{get; set;}
public virtual Error SrcError
{get; set;}
public virtual ICollection<Message> Messages
{get; set;}
public virtual ICollection<State> States
{get; set;}
public virtual ICollection<Photo> Photos
{get; set;}
public virtual ICollection<Tag> Tags
{get; set;}
public virtual ICollection<Attachment> Attachments
{get; set;}
[NotMapped]
public string StatusStr
{
    get
    {
        switch (this.Status)
        {
            case 0:
                return "відкрита";
            case 1:
                return "призначена";
            case 2:
                return "оброблювана";
            case 3:
                return "вирішена";
        }
    }
}

```

```

        case 4:
            return "закрита";
        default:
            return "невідома";
    }
}
set
{
    switch (value)
    {
        case "відкрита":
            this.Status = 0;
            break;
        case "призначена":
            this.Status = 1;
            break;
        case "оброблювана":
            this.Status = 2;
            break;
        case "вирішена":
            this.Status = 3;
            break;
        case "закрита":
            this.Status = 4;
            break;
        default:
            this.Status = 5;
    }
}
}
[NotMapped]
public string SeverityStr
{
    get
    {
        switch (this.Severity)
        {
            case 0:
                return "немає";
            case 1:
                return "низька";
            case 2:
                return "середня";

```

```

        case 3:
            return "висока";
        case 4:
            return "топ";
        default:
            return "немає";
    }
}
set
{
    switch (value)
    {
        case "немає":
            this.Severity = 0;
            break;
        case "низька":
            this.Severity = 1;
            break;
        case "середня":
            this.Severity = 2;
            break;
        case "висока":
            this.Severity = 3;
            break;
        case "топ":
            this.Severity = 4;
            break;
        default:
            this.Severity = 0;
    }
}
}
}
}
}
}

```

Б.2 Текст файла State.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;

```

```

using System.Threading.Tasks;

namespace TuErrorTracker.Models
{
    public class State
    {
        [Key]
        public int StateId
        {get; set;}
        [Required]
        public int ErrorId
        {get; set;}
        public virtual Error Error
        {get; set;}
        [Required]
        public byte Status
        {get; set;}
        [Required]
        public DateTime StartDate
        {get; set;}
        public DateTime EndDate
        {get; set;}
        public DateTime DueDate
        {get; set;}
        [Required]
        public int UserId
        {get; set;}
        public virtual User User
        {get; set;}
        public string Detailing
        {get; set;}
        public byte Severity
        {get; set;}
        [NotMapped]
        public string StartDateStr
        {
            get
            {
                return this.StartDate == null ? "не визначено" :
                ((DateTime)this.StartDate).ToString("dd.MM.yyyy");
            }
        }
        [NotMapped]
    }
}

```

```

public string EndDateStr
{
    get
    {
        return this.EndDate == null ? "не визначено" :
((DateTime)this.EndDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string DueDateStr
{
    get
    {
        return this.DueDate == null ? "не визначено" :
((DateTime)this.DueDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string StatusStr
{
    get
    {
        switch (this.Status)
        {
            case 0:
                return "відкрита";
            case 1:
                return "призначена";
            case 2:
                return "оброблювана";
            case 3:
                return "вирішена";
            case 4:
                return "закрита";
            default:
                return "невідома";
        }
    }
}
set
{
    switch (value)
    {
        case "відкрита":

```

```

        this.Status = 0;
        break;
    case "призначена":
        this.Status = 1;
        break;
    case "оброблювана":
        this.Status = 2;
        break;
    case "вирішена":
        this.Status = 3;
        break;
    case "закрита":
        this.Status = 4;
        break;
    default:
        this.Status = 5;
    }
}
}
[NotMapped]
public string SeverityStr
{
    get
    {
        switch (this.Severity)
        {
            case 0:
                return "немає";
            case 1:
                return "низька";
            case 2:
                return "середня";
            case 3:
                return "висока";
            case 4:
                return "топ";
            default:
                return "немає";
        }
    }
    set
    {
        switch (value)

```

```

        {
            case "немає":
                this.Severity = 0;
                break;
            case "низька":
                this.Severity = 1;
                break;
            case "середня":
                this.Severity = 2;
                break;
            case "висока":
                this.Severity = 3;
                break;
            case "топ":
                this.Severity = 4;
                break;
            default:
                this.Severity = 0;
        }
    }
}
}
}
}

```

Б.3 Текст файла Project.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace TuErrorTracker.Models
{
    public class Project
    {
        [Key]
        public int ProjectId
        {get; set;}
        [Required]
        [MaxLength(50)]

```

```

public string Title
{get; set;}
public DateTime StartDate
{get; set;}
public DateTime EndDate
{get; set;}
public DateTime DueDate
{get; set;}
[Required]
public string Detailing
{get; set;}
[Required]
public byte Status
{get; set;}
public int UserId
{get; set;}
public virtual User User
{get; set;}
public virtual ICollection<Error> Errors
{get; set;}
[NotMapped]
public string StartDateStr
{
    get
    {
        return this.StartDate == null ? "не визначено" :
((DateTime)this.StartDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string EndDateStr
{
    get
    {
        return this.EndDate == null ? "не визначено" :
((DateTime)this.EndDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string DueDateStr
{
    get
    {

```

```

        return this.DueDate == null ? "не визначено" :
((DateTime)this.DueDate).ToString("dd.MM.yyyy");
    }
}
[NotMapped]
public string StatusStr
{
    get
    {
        switch (this.Status)
        {
            case 0:
                return "Pre-alpha";
            case 1:
                return "Alpha";
            case 2:
                return "Beta";
            case 3:
                return "RC";
            case 4:
                return "RTM";
            case 5:
                return "GA";
            default:
                return "невизначена";
        }
    }
    set
    {
        switch (value)
        {
            case "Pre-alpha":
                this.Status = 0;
                break;
            case "Alpha":
                this.Status = 1;
                break;
            case "Beta":
                this.Status = 2;
                break;
            case "RC":
                this.Status = 3;
                break;
        }
    }
}

```

```

        case "RTM":
            this.Status = 4;
            break;
        case "GA":
            this.Status = 5;
            break;
        default:
            this.Status = 6;
    }
}
}

```

Б.4 Текст файла Message.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace TuErrorTracker.Models
{
    public class Message
    {
        [Key]
        public int MessageId
        {get; set;}
        [Required]
        public int ErrorId
        {get; set;}
        public virtual Error Error
        {get; set;}
        [Required]
        public DateTime CreationDate
        {get; set;}
        [Required]
        public int UserId
        {get; set;}
        public virtual User User
        {get; set;}
        public string Record
    }
}

```

```

        {get; set;}
        [NotMapped]
        public string CreationDateStr
        {
            get
            {
                return this.CreationDate == null ? "не визначено" :
                ((DateTime)this.StartDate).ToString("dd.MM.yyyy");
            }
        }
    }
}

```

Б.5 Текст файла ErrorController.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using TuErrorTracker.Data;
using TuErrorTracker.Models;

namespace TuErrorTracker.Controllers
{
    public class ErrorController : Controller
    {
        private readonly TuErrorTrackerContext tetcontext;

        public ErrorController (TuErrorTrackerContext tetc)
        {
            tetcontext = tetc;
        }

        [Authorize]
        public async Task<IActionResult> ReporterErrors(int? mode,
        bool? sortmode, int? severity)
        {

```

```

        var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.Status)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.DueDateStr);
        var email = User.Identity.Name;
        errors = errors.Where(e => e.Reporter.Email == email);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> AssigneeErrors(int? mode,
bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.Status)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.DueDateStr);
        var email = User.Identity.Name;
        errors = errors.Where(e => e.Assignee.Email == email);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> ProjectErrors(int?
project, int? mode, bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.ProjectId)
        .Include(e => e.Status)

```

```

        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)
        .Include(e => e.EndDateStr)
        .Include(e => e.DueDateStr);
        errors = errors.Where(e => e.ProjectId == project);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> TeamErrors(int? team, int?
mode, bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.TeamId)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)
        .Include(e => e.EndDateStr)
        .Include(e => e.DueDateStr);
        errors = errors.Where(e => e.TeamId == team);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> GroupErrors(int? group,
int? mode, bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.Groups)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)

```

```

        .Include(e => e.EndDateStr)
        .Include(e => e.DueDateStr);
        foreach (var error in errors)
        {
            if (!error.Groups.Contains(group))
                errors.Remove(error);
        }
        errors = errors.Where(e => e.GroupId == group);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

```

```

[Authorize]
public async Task<IActionResult> PredictedErrors(bool?
pred, int? mode, bool? sortmode, int? severity)
{
    var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)
        .Include(e => e.EndDateStr)
        .Include(e => e.RepeatPrediction)
        .Include(e => e.DueDateStr);
    if (!pred)
    {
        errors = errors.Where(e => e.RepeatPrediction ==
false);
    }
    else
    {
        errors = errors.Where(e => e.RepeatPrediction ==
true);
    }
    errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
    return View(await errors.ToListAsync());
}

```

```

[Authorize]

```

```

        public async Task<IActionResult> RepeatedErrors(bool? rep,
int? mode, bool? sortmode, int? severity)
        {
            var errors = await tetcontext.Errors
                .Include(e => e.ErrorId)
                .Include(e => e.Title)
                .Include(e => e.Project)
                .Include(e => e.StatusStr)
                .Include(e => e.SeverityStr)
                .Include(e => e.StartDateStr)
                .Include(e => e.EndDateStr)
                .Include(e => e.Repeat)
                .Include(e => e.DueDateStr);
            if (!rep)
            {
                errors = errors.Where(e => e.Repeat == false);
            }
            else
            {
                errors = errors.Where(e => e.Repeat == true);
            }
            errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
            return View(await errors.ToListAsync());
        }

```

```

[Authorize]
public async Task<IActionResult> DoubleErrors(int? err,
int? mode, bool? sortmode, int? severity)
{
    var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.StatusStr)
        .Include(e => e.SrcErrorId)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)
        .Include(e => e.EndDateStr)
        .Include(e => e.DueDateStr);
    errors = errors.Where(e => e.SrcErrorId == err);
    errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
}

```

```

        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> ModuleErrors(int? module,
int? mode, bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
            .Include(e => e.ErrorId)
            .Include(e => e.Title)
            .Include(e => e.Project)
            .Include(e => e.ModuleId)
            .Include(e => e.StatusStr)
            .Include(e => e.SrcErrorId)
            .Include(e => e.SeverityStr)
            .Include(e => e.StartDateStr)
            .Include(e => e.EndDateStr)
            .Include(e => e.DueDateStr);
        errors = errors.Where(e => e.ModuleId == module);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

    [Authorize]
    public async Task<IActionResult> ProblemclassErrors(int?
pc, int? mode, bool? sortmode, int? severity)
    {
        var errors = await tetcontext.Errors
            .Include(e => e.ErrorId)
            .Include(e => e.Title)
            .Include(e => e.Project)
            .Include(e => e.ProblemclassId)
            .Include(e => e.StatusStr)
            .Include(e => e.SrcErrorId)
            .Include(e => e.SeverityStr)
            .Include(e => e.StartDateStr)
            .Include(e => e.EndDateStr)
            .Include(e => e.DueDateStr);
        errors = errors.Where(e => e.ProblemClassId == pc);
        errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
        return View(await errors.ToListAsync());
    }

```

```

    }

    [Authorize]
    public async Task<IActionResult> SearchDateErrors(bool
sdatemode, string sdate1, string sdate2, bool ddatemode, string ddate1,
string ddate2, bool edatemode, string edate1, string edate2)
    {
        DateTime sdt1 = Processor.GetDateTime(sdate1);
        DateTime sdt2 = Processor.GetDateTime(sdate2);
        DateTime ddt1 = Processor.GetDateTime(ddate1);
        DateTime ddt2 = Processor.GetDateTime(ddate1);
        DateTime edt1 = Processor.GetDateTime(edate1);
        DateTime edt2 = Processor.GetDateTime(edate1);
        var errors = await tetcontext.Errors
.Include(e => e.ErrorId)
.Include(e => e.Title)
.Include(e => e.Project)
.Include(e => e.StatusStr)
.Include(e => e.SeverityStr)
.Include(e => e.StartDateStr)
.Include(e => e.EndDateStr)
.Include(e => e.DueDateStr);
        if (sdatemode)
        {
            errors = errors.Where(e => e.StartDateStr >= sdt1
&& e.StartDateStr <= sdt2);
            errors = errors.OrderByDescending(s =>
s.StartDateStr);
        }
        else
        {
            errors = errors.Where(e => e.StartDateStr == sdt1);
        }
        if (ddatemode)
        {
            errors = errors.Where(e => e.DueDateStr >= ddt1 &&
e.StartDateStr <= ddt2);
            errors = errors.OrderByDescending(s =>
s.DueDateStr);
        }
        else
        {
            errors = errors.Where(e => e.DueDateStr == ddt1);

```

```

    }
    if (edatemode)
    {
        errors = errors.Where(e => e.StartDateStr >= edt1
&& e.StartDateStr <= edt2);
        errors = errors.OrderByDescending(s =>
s.EndDateStr);
    }
    else
    {
        errors = errors.Where(e => e.StartDateStr == edt1);
    }
    errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
    return View(await errors.ToListAsync());
}

```

```

[Authorize]
public async Task<IActionResult> TagError(string tagtitle)
{
    var errors = await tetcontext.Errors
.Include(e => e.ErrorId)
.Include(e => e.Title)
.Include(e => e.Project)
.Include(e => e.Status)
.Include(e => e.StatusStr)
.Include(e => e.SeverityStr)
.Include(e => e.DueDateStr);
    var tags = await tetcontext.Tags;
    var otag = tags.FirstOrDefaultAsync(t => t.Title ==
tagtitle);

    foreach (var error in errors)
    {
        if (!error.Tags.Contains(otag))
            errors.Remove(error);
    }
    errors = errors.OrderByDescending(s => s.ErrorId);
    return View(await errors.ToListAsync());
}

```

```

[Authorize]
public async Task<IActionResult> SearchError(string
errrtitle, int? mode, bool? sortmode, int? severity)

```

```

{
    var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Project)
        .Include(e => e.Status)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.DueDateStr);
    errors = errors.Where(e => e.Title.Contains(errtitle));
    errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
    return View(await errors.ToListAsync());
}

[Authorize]
public async Task<IActionResult> Details(int? errorid)
{
    var errors = await tetcontext.Errors
        .Include(e => e.ErrorId)
        .Include(e => e.Title)
        .Include(e => e.Detailing)
        .Include(e => e.Project)
        .Include(e => e.StatusStr)
        .Include(e => e.SeverityStr)
        .Include(e => e.StartDateStr)
        .Include(e => e.EndDateStr)
        .Include(e => e.DueDateStr)
        .Include(e => e.RepeatPrediction)
        .Include(e => e.Repeat)
        .Include(e => e.Reporter)
        .Include(e => e.Assignee)
        .Include(e => e.Module)
        .Include(e => e.Team)
        .Include(e => e.Problemclass)
        .Include(e => e.SrcError)
        .Include(e => e.Messages)
        .Include(e => e.States)
        .Include(e => e.Photos)
        .Include(e => e.Tags)
        .Include(e => e.Attachments)
        .Include(e => e.Steps)
        .FirstOrDefaultAsync(e => e.ErrorId == errorid);

```

```

        if (error == null)
        {
            return NotFound();
        }
        return View(error);
    }

    [Authorize]
    public IActionResult Create()
    {
        ViewData["Projects"] = new
SelectList(tetcontext.Projects, "ProjectId", "Title");
        ViewData["Modules"] = new
SelectList(tetcontext.Modules, "ModuleId", "Title");
        ViewData["Problemclassss"] = new
SelectList(tetcontext.Problemclassss, "ProblemclassId", "Title");
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("Id, Title,
Detailing, ProjectId, ModuleId, ProblemclassId, Tags, Steps")] Error
error)
    {
        if (ModelState.IsValid)
        {
            var users = await tetcontext.User;
            var email = User.Identity.Name;
            var rep = users.FirstOrDefaultAsync(u => u.Email ==
email);

            error.ReporterId = rep.Id;
            error.StartDate = DateTime.Now;
            error.Status = 0;
            error.Severity = 0;
            tetcontext.Add(error);
            await tetcontext.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(error);
    }

    [Authorize]

```

```

public async Task<IActionResult> Edit(int? errorid)
{
    var error = await tetcontext.Errors.FindAsync(errorid);
    if (error == null)
    {
        return NotFound();
    }
    ViewData["Projects"] = new
SelectList(tetcontext.Projects, "ProjectId", "Title");
    ViewData["Users"] = new SelectList(tetcontext.Users,
"UserId", "Email");
    ViewData["Modules"] = new
SelectList(tetcontext.Modules, "ModuleId", "Title");
    ViewData["Problemclassss"] = new
SelectList(tetcontext.Problemclassss, "ProblemclassId", "Title");
    return View(error);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int errorid,
[Bind("Id, Title, Detailing, ProjectId, StatusId, AssignessId, ModuleId,
ProblemclassId, TeamId, SrcErrorId, Severity, EndDate, DueDate, Repeat,
Tags")] Error error)
{
    if (ModelState.IsValid)
    {
        tetcontext.Update(error);
        await tetcontext.SaveChangesAsync();
    }
    ViewData["Projects"] = new
SelectList(tetcontext.Projects, "ProjectId", "Title");
    ViewData["Users"] = new SelectList(tetcontext.Users,
"UserId", "Email");
    ViewData["Modules"] = new
SelectList(tetcontext.Modules, "ModuleId", "Title");
    ViewData["Problemclassss"] = new
SelectList(tetcontext.Problemclassss, "ProblemclassId", "Title");
    return View(error);
}

[Authorize]

```

```

        public async Task<IActionResult> UnTimedErrors(int? mode,
bool? sortmode, int? severity)
        {
            var today = DateTime.Now;
            var errors = await tetcontext.Errors
                .Include(e => e.ErrorId)
                .Include(e => e.Title)
                .Include(e => e.Project)
                .Include(e => e.Status)
                .Include(e => e.StatusStr)
                .Include(e => e.SeverityStr)
                .Include(e => e.Assignee)
                .Include(e => e.DueDateStr);
            errors = errors.Where(e => e.DueDate < today &&
e.EndDate == null);
            errors = Processor.SortAndSeverity(errors, mode,
sortmode, severity);
            return View(await errors.ToListAsync());
        }
    }
}

```

ДОДАТОК В
Слайди презентації

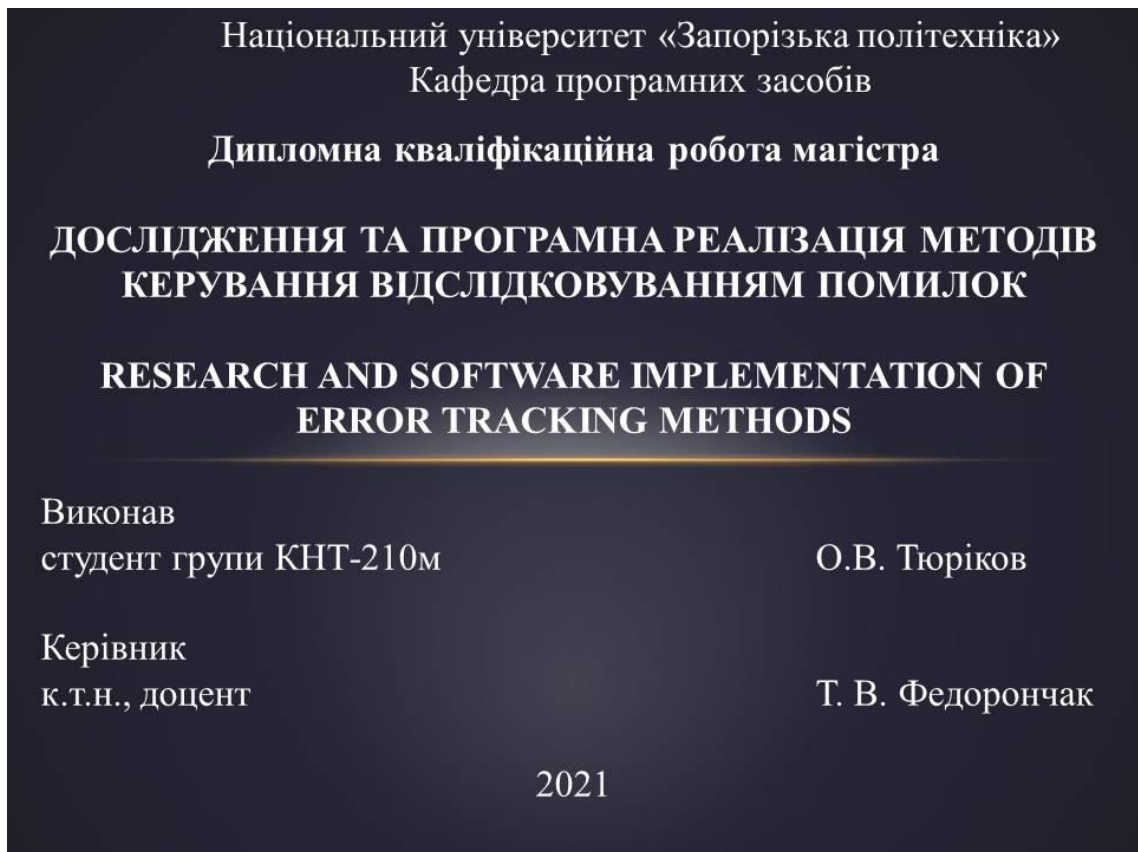


Рисунок В.1 – Слайд 1

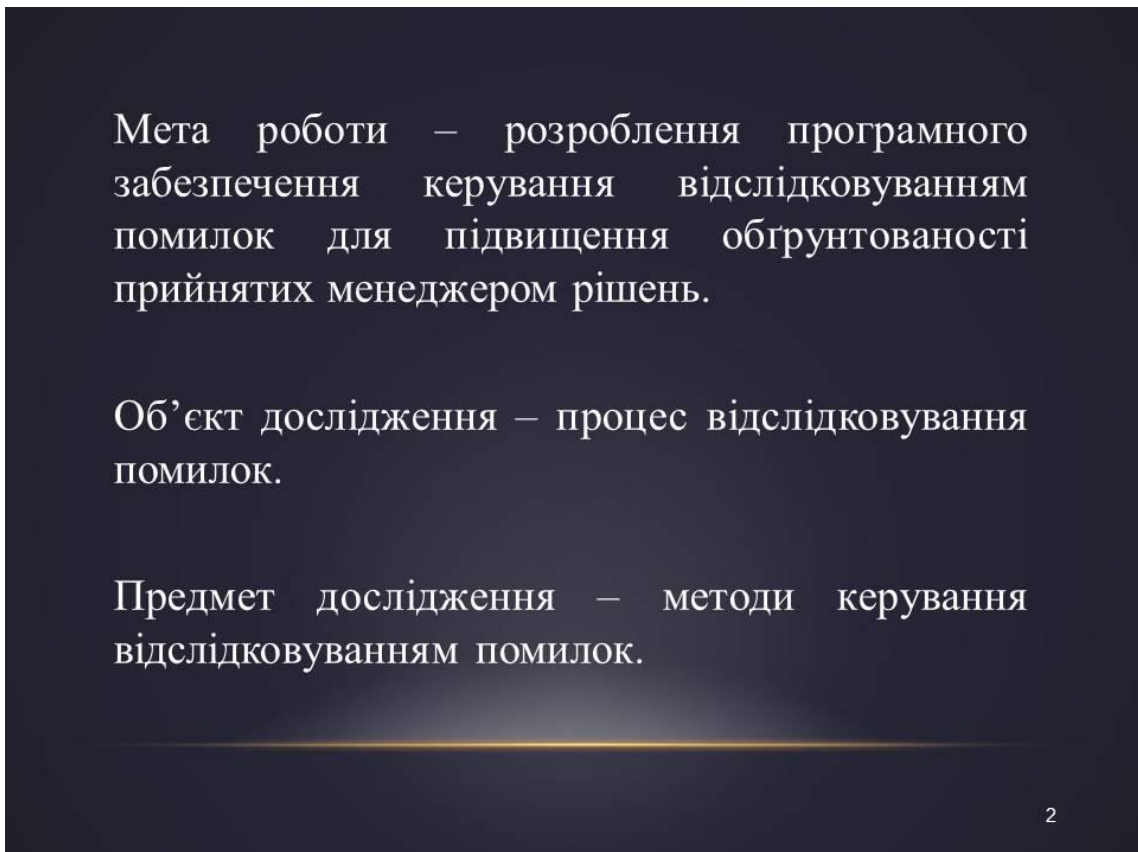


Рисунок В.2 – Слайд 2

Завдання роботи

- ✓ дослідження і розробка власного методу класифікації помилок;
- ✓ проектування програмного забезпечення;
- ✓ програмна реалізація системи керування відслідковуванням помилок.

3

Рисунок В.3 – Слайд 3

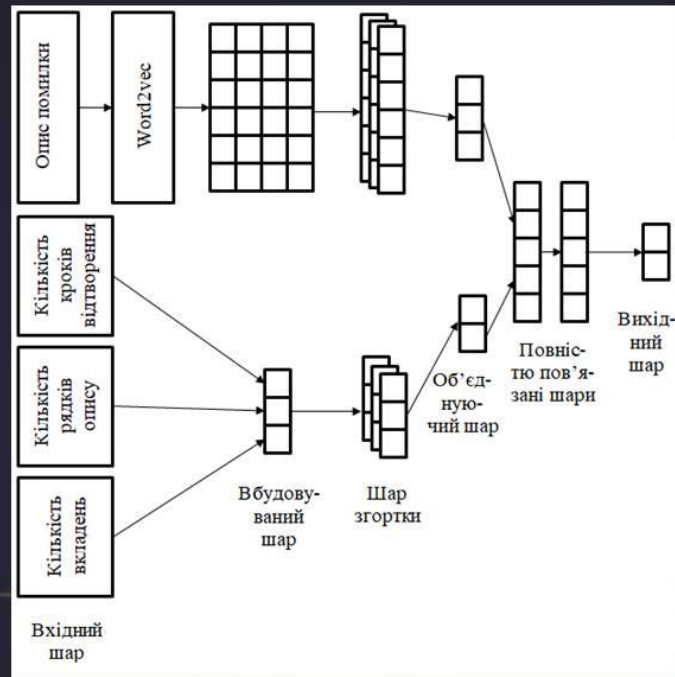
Порівняння методів прогнозування довготривалих помилок

Метод	Клас задачі	Результуючий показник	Основна модель
[P. Bhattacharya, I. Neamtiu]	Задача прогнозування	Час усунення помилки	Регресійна модель
[W. Abdelmoez, M. Kho-lief, F. Elsalmy]	Задача класифікації	Клас за часом усунення	Наївний байєс
[P. Ardimento, A. Dinapoli]	Задача класифікації	Клас за часом усунення	Метод опорних векторів
[M. Sharma, M. Kumari, V. B. Singh]	Задача прогнозування	Час усунення помилки	Метод регресії опорних векторів
[Luiz Gomes, Ricardo Torres, Mario Cortes]	Задача класифікації	Клас за часом усунення	Багатопшарова нейронна мережа
Представлений метод	Задача класифікації	Клас щодо можливого повторного усунення помилки	Конволюційна нейронна мережа

4

Рисунок В.4 – Слайд 4

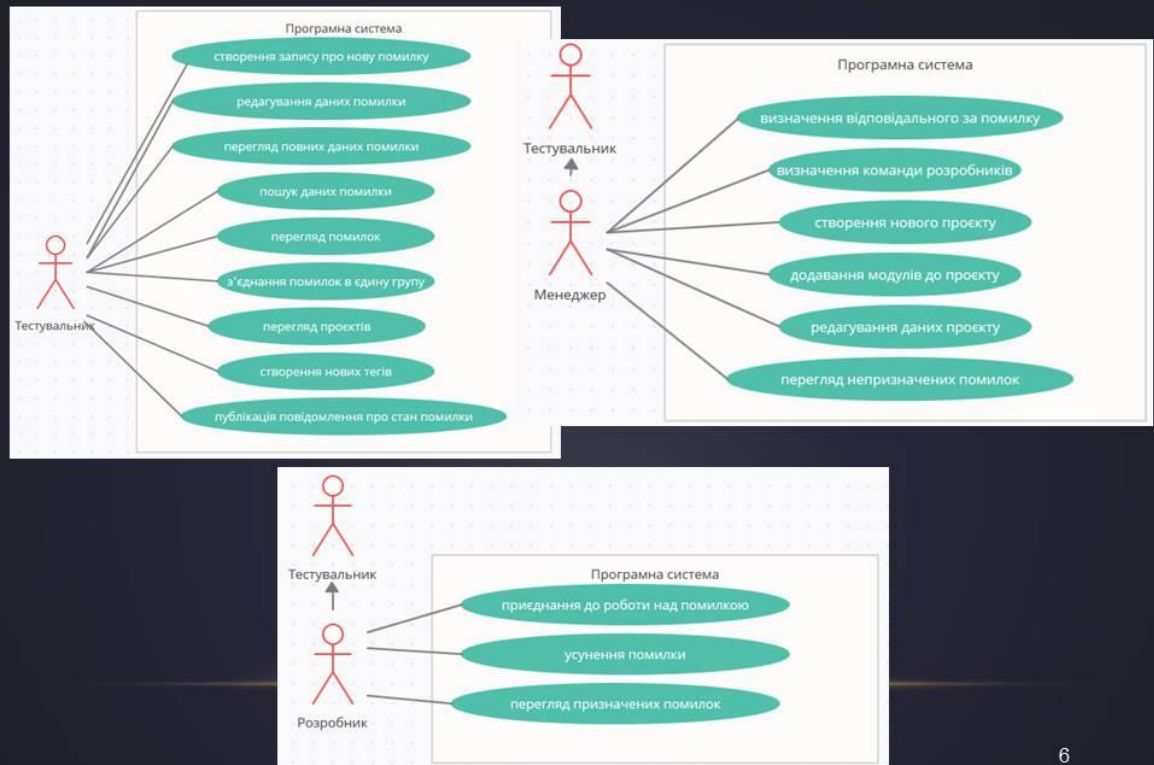
Метод і відповідна модель класифікації помилок



5

Рисунок В.5 – Слайд 5

Сценарії використання програми



6

Рисунок В.6 – Слайд 6

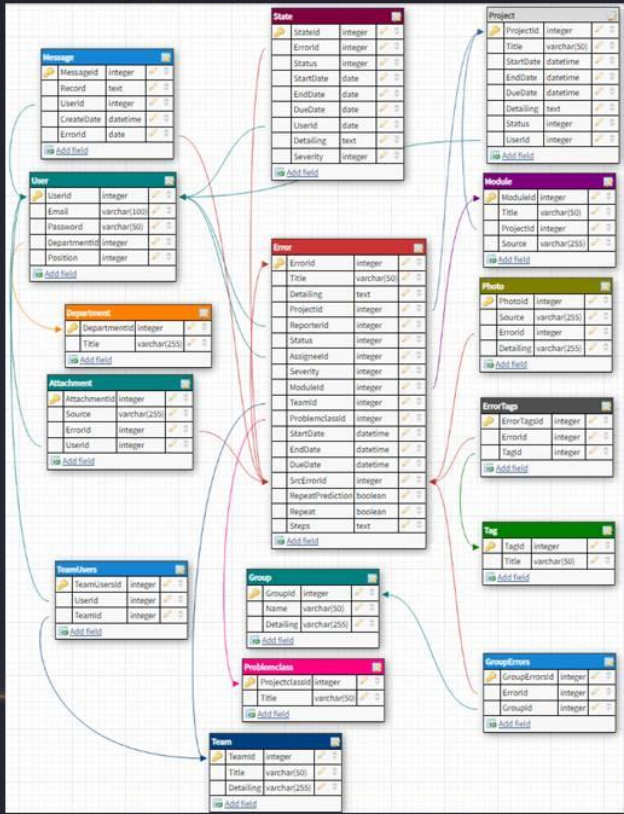
Обґрунтування вибору мови програмування для розробки вебзастосунку

Показник порівняння	C#	Ruby
Об'єктно-орієтована парадигма	Так	Так
Тип мови	Компільована	Інтерпретована
Продуктивність	Висока	Середня
Зручність створення великих застосунків	Висока	Середня
Простота створення інтерфейсу користувача	Висока	Висока
Підтримка розробників	Краща	Гірша

7

Рисунок В.7 – Слайд 7

Структура бази даних



8

Рисунок В.8 – Слайд 8

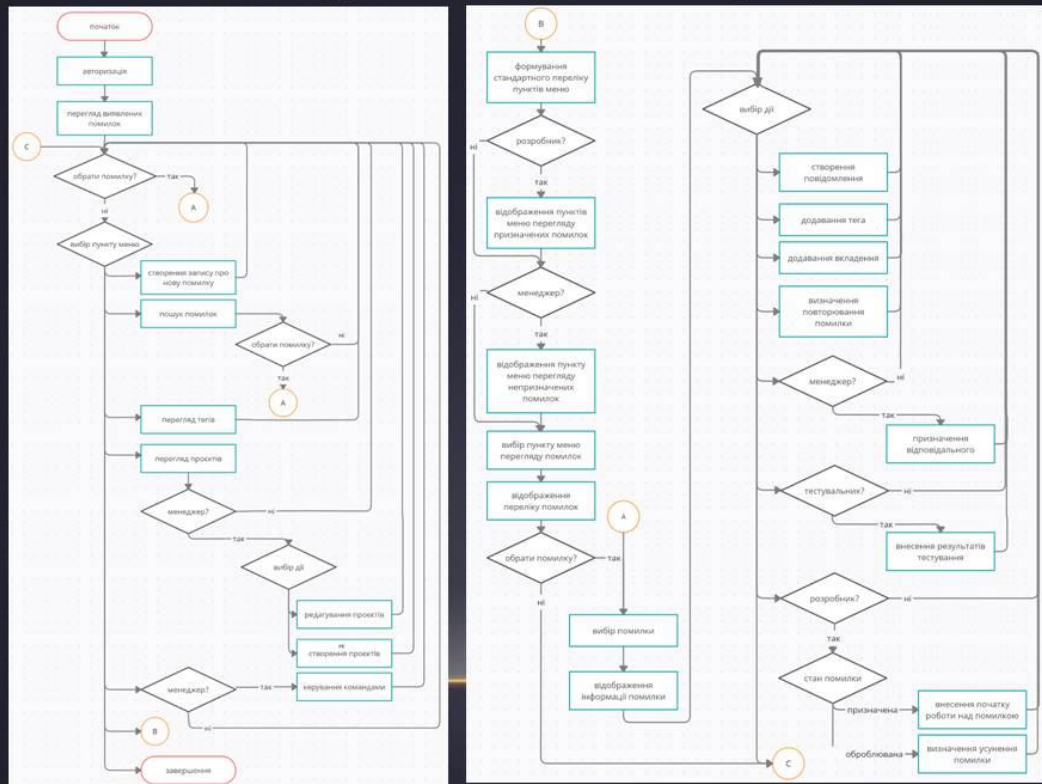
Схема структури програми



9

Рисунок В.9 – Слайд 9

Схема функціонування програми



10

Рисунок В.10 – Слайд 10

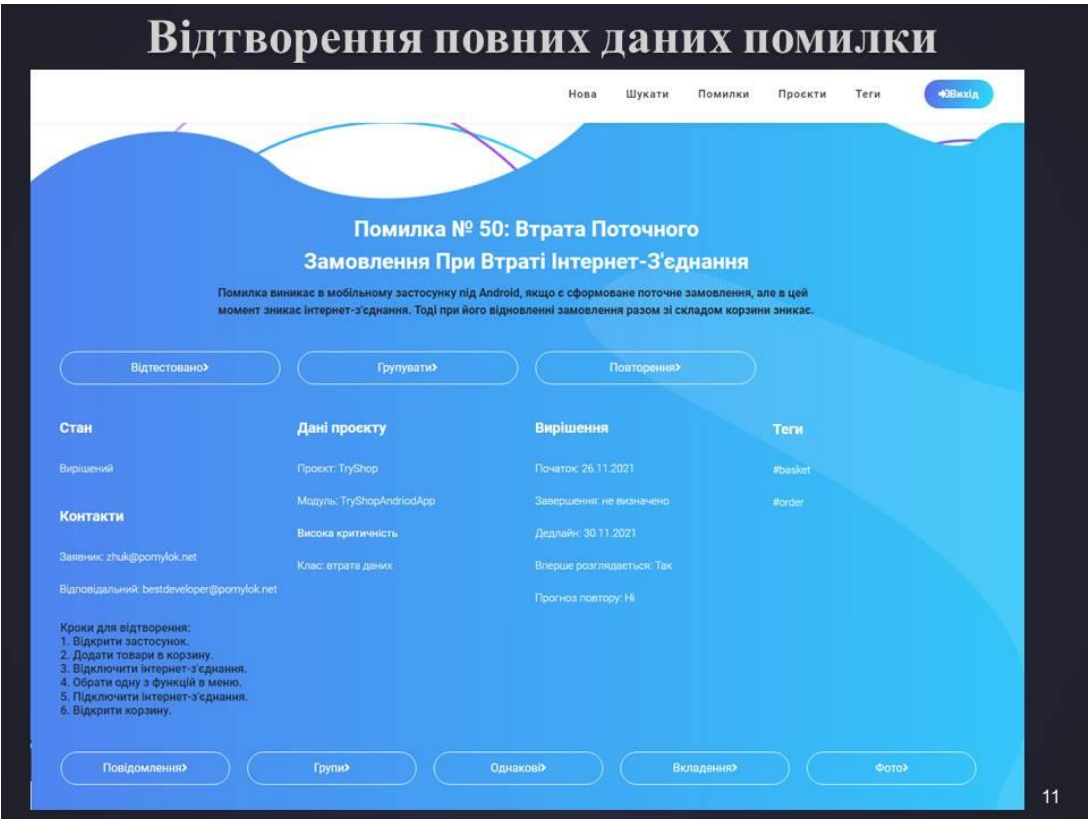


Рисунок В.11 – Слайд 11

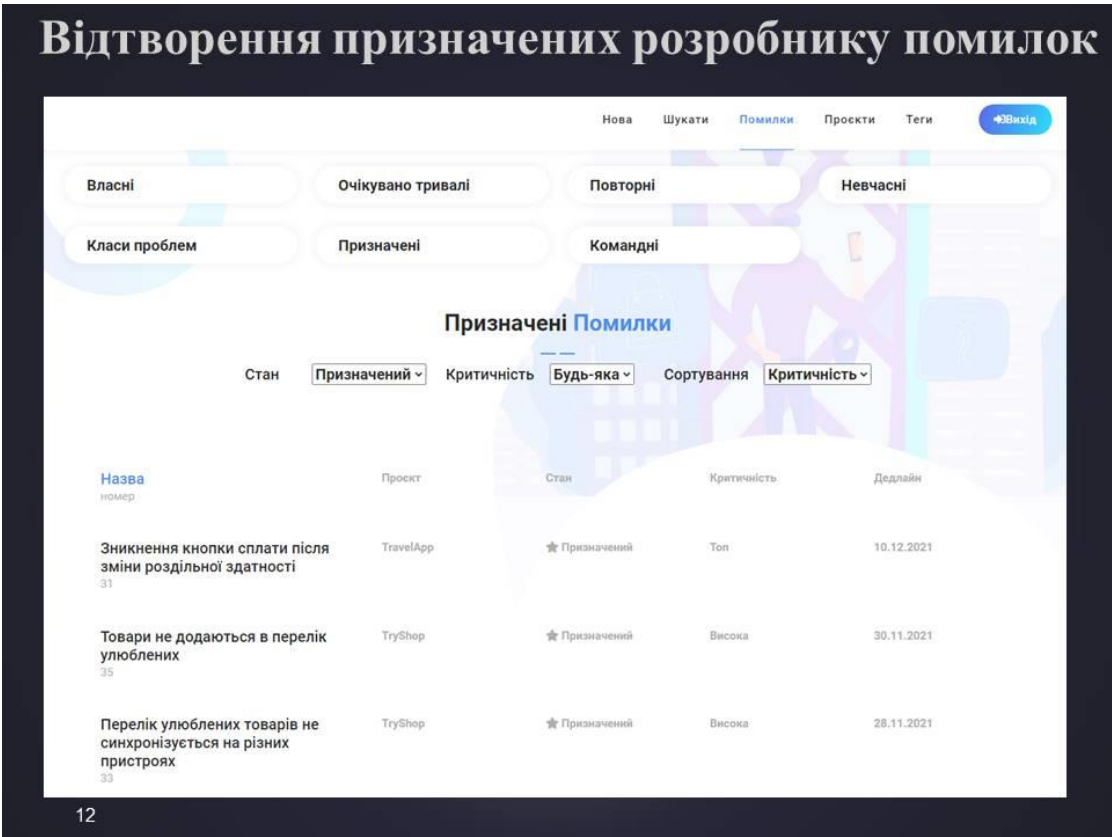


Рисунок В.12 – Слайд 12

Відтворення очікувано тривалих помилок за прогнозом

Нова
Шукати
Помилки
Проекти
Теги
Команди
Вихід

Власні
Непризначені
Незавершені вчасно
Очікувано тривалі

Повторні
Невчасні
Класи проблем

Очікувано Тривалі Помилки

Помилки, які за прогнозом можуть бути вирішені не за раз

Стан
Відкритий ▾
Критичність
Будь-яка ▾
Сортування
Стандартно ▾

Назва номер	Проект	Стан	Критичність	Дати	Дедлайн
Відсутній доступ до пункту інсталяції плагінів 40	TravelApp	Відкритий	немає	26.11.2021 - не визначено	не визначено
Вміст корзини періодично губиться 37	TryShop	Відкритий	немає	26.11.2021 - не визначено	не визначено

13

Рисунок В.13 – Слайд 13

Експериментальне дослідження

Метод або модель класифікації	Точність класифікації
Наївний байєсів класифікатор	0,48
Метод k-середніх	0,65
Метод опорних векторів	0,67
Багатошарова нейронна мережа	0,72
Представлений метод	0,91

14

Рисунок В.14 – Слайд 14

Висновки

У роботі виконано розроблення програмного забезпечення керування відслідковуванням помилок для підвищення обґрунтованості прийнятих менеджером рішень. Для цього було виконано усі завдання роботи і досягнуто наступні результати:

- було виконано аналіз проблеми і програмних засобів, які використовуються для відслідковування помилок, у результаті чого виявлено, що вони не надають можливість інтелектуальної класифікації помилок на такі, які будуть потребувати повторного усунення, виконано дослідження існуючих методів, які використовуються для прогнозування часу усунення помилок і класифікації помилок за часом усунення, і розроблено власний метод керування відслідковуванням помилок;

- було виконано проєктування програмного забезпечення, для чого було виконано моделювання роботи тестувальників, менеджерів проєктів і розробників з програмою, виконано вибір мови програмування C# на основі технології ASP.NET для реалізації вебзастосунку та мови програмування Python для реалізації запропонованого методу, визначено структуру бази даних, структуру програми загалом;

- було виконано програмну реалізацію системи керування відслідковуванням помилок на основі підходу code first, що дозволило оптимізувати обсяг коду щодо роботи з даними.

Науковою новизною роботи є запропонований метод класифікації помилок, який визначає, чи належить виявлена помилка до тих, які будуть потребувати повторного усунення, застосуванням моделі на основі конволюційних нейронних мереж і методу word2vec для оброблення текстових описів помилок, що дозволяє на основі отриманих даних менеджера проєктів приймати обґрунтовані рішення про виділення потрібних ресурсів.

15

Рисунок В.15 – Слайд 15