УДК 004.4
DOI https://doi.org/10.32782/2663-5941/2025.2.2/09

Горбань Г. В.
Чорноморський національний університет імені Петра Могили

Фісун М. Т.
Чорноморський національний університет імені Петра Могили

Раленко В. С.
Чорноморський національний університет імені Петра Могили

Стоєв Є. Д.
Чорноморський національний університет імені Петра Могили

АВТОМАТИЗАЦІЯ ПЕРЕВІРКИ ПРОГРАМНОГО КОДУ З ВИКОРИСТАННЯМ СУЧАСНИХ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ

У статті розглянуто проблему підвищення ефективності та якості процесу рев'ю коду шляхом розробки платформи для автоматизованої перевірки програмного коду з використанням технологій штучного інтелекту. Проаналізовано сучасні інструменти автоматизованого аналізу коду та виявлено їх обмеження: стандартні статичні аналізатори коду (наприклад, SonarQube, Codacy, ESLint) здатні знайти синтаксичні помилки чи порушення стилю, але не враховують контекст програмної логіки та можуть генерувати хибно-позитивні зауваження. Запропоновано архітектуру інформаційної системи, що інтегрується з платформою GitHub та використовує модель GPT-4 для «розумної» перевірки коду. Платформу побудовано за мікросервісним принципом: виділено модуль інтеграції з GitHub (обробник подій pull request), модуль аналізу коду на базі ШІ та модуль керування даними pull request. Наведено UML-діаграми архітектури системи та послідовності взаємодії її компонентів. Розроблено веб-застосунок (Angular) для відображення результатів аналізу та реалізовано автоматичне коментування виявлених проблем безпосередньо у pull request на GitHub. Прототип платформи протестовано на прикладі реального репозиторію: система успішно виявила логічні помилки та надала рекомендації щодо покращення коду (зокрема, в тестовому pull request було знайдено 6 помилок і запропоновано 11 покращень), що підтверджує працездатність та ефективність підходу. Зроблено висновки про переваги використання штучного інтелекту в перевірці коду (зменшення навантаження на команду, більш глибокий аналіз програмної логіки) та окреслено напрямки подальшого розвитку системи, такі як підтримка більшої кількості мов програмування, оптимізація витрат на виклики до API мовних моделей та розширення функціональності платформи.

Ключові слова: *автоматизована перевірка коду, рев'ю коду, штучний інтелект, pull request, GitHub,. NET, Angular.*

Постановка проблеми. Рев'ю коду є невід'ємною частиною сучасного процесу розробки програмного забезпечення. Воно дозволяє виявити дефекти та покращити якість програмного коду на ранніх етапах, попереджаючи потенційні помилки у продуктивному середовищі. Традиційно перевірка коду здійснюється розробниками вручну: один або кілька рецензентів аналізують зміни в коді (наприклад, у форматі pull request) та залишають коментарі з зауваженнями чи пропозиціями. Такий підхід, хоча і ефективний у забезпеченні високої якості, має суттєві недоліки – значні витрати часу та людських ресурсів, суб'єктивність оцінок, можливість пропуску дефектів через людський фактор. Автоматизація цього процесу є актуальним завданням, що може підвищити продуктивність команди розробки та стандартизувати перевірку коду. Останнім часом, зі стрімким розвитком технологій штучного інтелекту (ШІ), з'явилися нові можливості для автоматизації аналізу програмного коду. Сучасні мовні моделі, зокрема архітектури на основі трансформерів (GPT-3, GPT-4), демонструють здатність розуміти контекст та семантику коду і генерувати осмислені тексти-відповіді [1, 2]. Це відкриває перспективи створення інтелектуальних

систем автоматизованого рев'ю коду, які можуть функціонувати подібно до досвідченого рецензента: виявляти не лише синтаксичні помилки, але й логічні дефекти, порушення архітектурних принципів, недотримання кращих практик програмування тощо.

Аналіз останніх досліджень і публікацій. Питання автоматизації перевірки програмного коду привертало увагу дослідників та індустрії протягом останніх десятиліть. Традиційно для виявлення дефектів у коді застосовуються статичні аналізатори коду – інструменти, що перевіряють вихідний код на наявність помилок без виконання програми. Прикладами є SonarQube, Codacy, ESLint та інші. SonarQube – потужний інструмент статичного аналізу, який підтримує понад 25 мов програмування і здійснює комплексну перевірку коду на відповідність стандартам, вразливості та продуктивність [3]. Він інтегрується з системами контролю версій та CI/CD, що дозволяє автоматично перевіряти код після кожного коміту або створення pull request. Codacy має подібний функціонал і забезпечує швидке хмарне розгортання, пропонуючи зручну інтеграцію з Git-платформами (GitHub, GitLab тощо) для коментування результатів аналізу прямо у pull request. Такі системи значно полегшують підтримку стандартів кодування в команді. Проте, вони зазвичай базуються на наперед визначених правилах і шаблонах, через що обмежені у розумінні контексту. Наприклад, інструмент Danger дозволяє автоматизувати деякі аспекти рев'ю коду, виконуючи задані розробниками правила на етапі CI. Однак, його потрібно вручну налаштовувати під кожен проєкт, додаючи власні правила перевірки [4]. Це ускладнює використання Danger нефахівцями та може призвести до пропуску нетривіальних логічних помилок, які не були явно прописані в правилах. Окрім статичних аналізаторів, існують лінтери та інші спеціалізовані інструменти для окремих мов. Зокрема, ESLint – популярний статичний аналізатор для JavaScript/TypeScript, що допомагає виявляти синтаксичні помилки та порушення стилю [5]. Проте він обмежений лише цими мовами і набором правил, визначених спільнотою, і не виконує глибшого семантичного аналізу.

Значний прогрес у галузі штучного інтелекту для аналізу коду спостерігається з появою інструментів, що використовують машинне навчання та великі мовні моделі. Компанія DeepCode (нині технологія інтегрована в продукт Snyk Code) застосувала ML-моделі, навчені на мільйонах репозиторіїв, для виявлення уразливостей і багів, які не завжди фіксуються традиційними лінтерами. Такі AI-інструменти можуть пропонувати автоматичні виправлення коду та інтегруються в середовища розробки. Ще одним прикладом є сервіс GitHub Copilot, який на базі моделі GPT-3 (Codex) допомагає розробникам писати код, а також може підказувати виправлення. Хоч Copilot більше орієнтований на генерацію коду, деякі дослідження зазначають його потенціал і для автоматизації рев'ю – зокрема, Copilot здатен попереджати про прості помилки чи субоптимальні конструкції під час написання коду [6].

В наукових публікаціях останніх років описано перші спроби інтеграції великих мовних моделей (LLM) безпосередньо у процес перевірки коду. Наприклад, у роботі [7] представлено результати впровадження інструмента на базі GPT-3.5 (Qodo PR Agent) для автоматичного рев'ю pull request у промисловому середовищі. Згідно з цим дослідженням, автоматизовані коментарі, згенеровані мовною моделлю, були корисними і розробники погодилися приблизно з 73,8 % таких зауважень, що свідчить про їх практичну цінність [7]. Водночас виявлено і недолік – середній час закриття pull request дещо збільшився через необхідність опрацювання згенерованих AI-коментарів [7]. Це підкреслює, що хоча ШІ-асистенти здатні підвищити якість рев'ю коду, повністю замінити експерта-людину вони поки що не можуть, а їх використання вимагає додаткового контролю.

Постановка завдання. Проблема, яка вирішується в рамках даного дослідження, полягає у створенні платформи, що здатна автоматично аналізувати зміни у вихідному коді програмного проєкту та надавати змістовні рекомендації щодо покращення коду в режимі, інтегрованому в робочий процес розробників. Аналіз існуючих рішень показав, що, незважаючи на наявність потужних статичних аналізаторів, розробники все ще витрачають значний час на ручне рев'ю коду, особливо перевіряючи логіку та архітектурні аспекти змін. Людський фактор може призводити до непослідовності в оцінках та пропуску помилок. З іншого боку, застосування надто загальних автоматичних інструментів (лінтерів) часто генерує занадто багато непріоритетних зауважень, що знижує довіру команди до таких інструментів.

Тому згідно до зазначеного вище сформульовано такі основні завдання:

– розробити архітектуру програмної системи, яка інтегрується з популярною платформою спільної розробки (GitHub) та реагує на події відкриття/оновлення pull request;

– забезпечити аналіз коду із врахуванням контексту – для цього обрати і інтегрувати сучасну модель штучного інтелекту, здатну розуміти програмний код (зокрема, мовну модель GPT-4 від OpenAI);

– реалізувати модуль видачі результатів у зручній формі – як безпосередні коментарі в системі контролю версій та/або структуровані звіти у власному веб-застосунку;

– перевірити працездатність розробленої платформи на реальних сценаріях, порівняти результати з ручним рев'ю та оцінити, чи дійсно платформа допомагає виявити важливі проблеми.

Таким чином, поставлена задача охоплює повний цикл створення інформаційної системи: від проектування архітектури до імплементації та тестування прототипу в наближених до реальних умовах.

Виклад основного матеріалу. Запропонована платформа автоматизованої перевірки коду спроєктована з використанням мікросервісної архітектури. Це означає, що різні функціональні підсистеми реалізовані у вигляді незалежних сервісів, які взаємодіють через чітко визначені інтерфейси (HTTP API). Такий підхід полегшує підтримку і масштабування системи, дозволяючи модифікувати або замінювати окремі компоненти без впливу на інші. На рис. 1 зображено загальну архітектуру платформи у вигляді структурної схеми з основними модулями та їх зв'язками.

Як видно з архітектурної діаграми, система складається з таких ключових компонентів:

– *Інтеграційний модуль з GitHub (Event Handler API)*. Це мікросервіс, що відповідає за взаємодію з інфраструктурою контролю версій, зокрема з подіями платформи GitHub. Він отримує повідомлення (Webhooks) про створення або оновлення pull request у відстежуваному репозиторії. При надходженні такого повідомлення Event Handler ініціює процес автоматичної перевірки коду – тобто надсилає відповідний запит до сервісу аналізу коду. Крім того, інтеграційний модуль

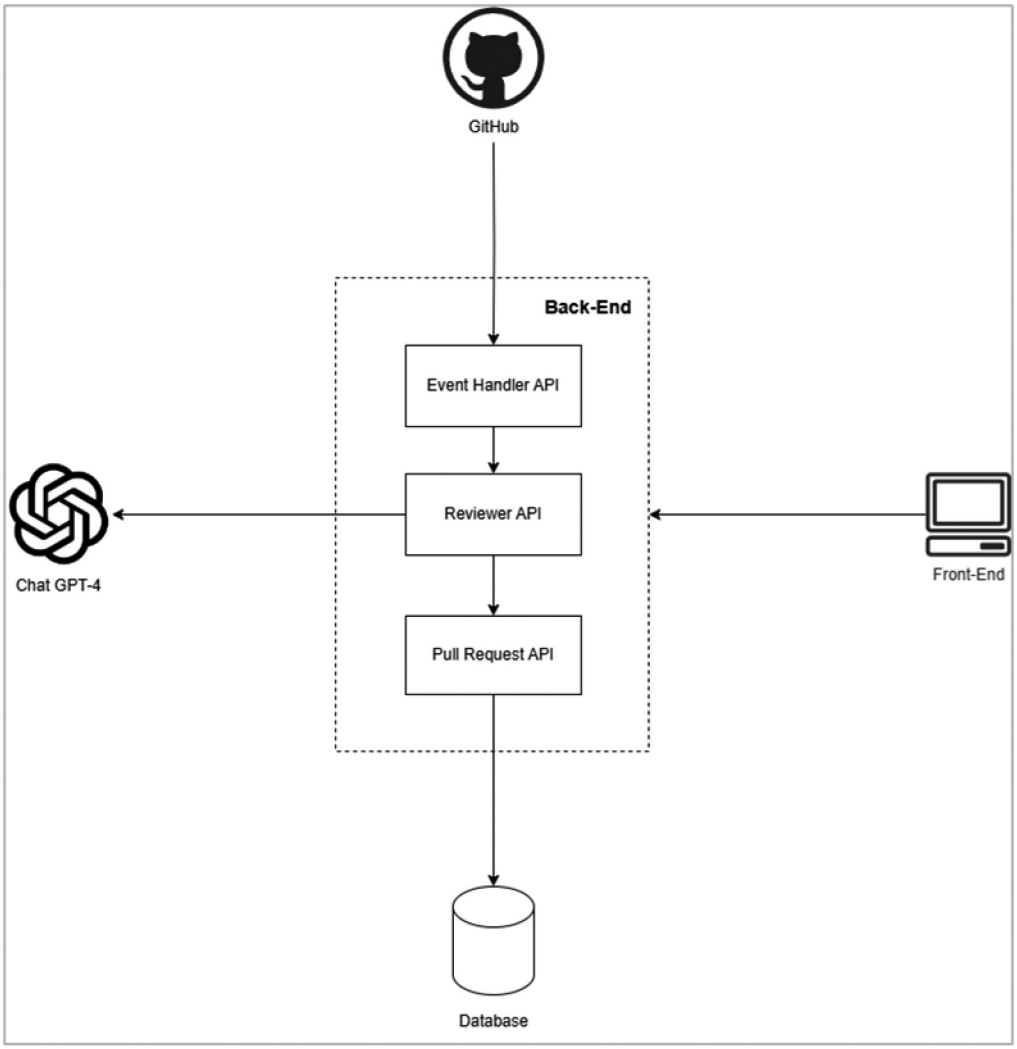

Рис. 1. Загальна архітектура платформи автоматизованого рев'ю коду (структурні компоненти та інтеграції)

може через GitHub API розміщувати коментарі з результатами аналізу у тому ж pull request (від імені бота або користувача, що авторизував систему).

– *Сервіс аналізу коду (Reviewer API).* Даний мікросервіс реалізує безпосередньо логіку перевірки коду із використанням штучного інтелекту. Отримавши від Event Handler запит на аналіз конкретного pull request, цей сервіс звертається до зовнішнього API мовної моделі (OpenAI API) з підготовленим prompt-запитом, що містить потрібну інформацію: фрагменти коду або diffs змін, контекст (назва проєкту, опис змін) та допоміжні вказівки (які саме проблеми слід шукати). Для взаємодії з моделлю GPT-4 у компоненті Reviewer API використано офіційний SDK OpenAI для. NET, що спрощує надсилання запитів і отримання відповідей від моделі. Отримавши відповідь від моделі (список знайдених потенційних проблем та рекомендацій), Reviewer API структурує ці дані (наприклад, у форматі списку коментарів із прив'язкою до конкретних рядків коду) і передає результати назад до Event Handler або безпосередньо у базу даних.

– *Сервіс керування pull request (Pull Request API)*. Це окремий мікросервіс, що виконує допоміжні функції: зберігає інформацію про репозиторії та pull request, керує авторизацією користувачів платформи, надає інтерфейс для отримання списку pull request і деталей аналізу через вебдодаток. По суті, Pull Request API виступає як бекенд для фронтенду системи, забезпечуючи доступ до даних, збережених у базі (результати аналізу, історія перевірок тощо). Через цей же API користувач може налаштувати інтеграцію системи зі своїм GitHub-репозиторієм (наприклад, додати Webhook або прив'язати OAuth-токен для доступу).

– *База даних*. Для зберігання інформації про користувачів, підключені репозиторії, pull request та результати їх перевірки використовується реляційна база даних. У прототипі реалізовано схему БД, що містить таблиці для сутностей "Repository", "PullRequest", "Comment" тощо. База необхідна, щоб мати історію всіх аналізів – це дозволяє у будь-який момент переглянути звіт по старих pull request, а також уникнути повторного аналізу одних і тих самих змін. Збереження даних відбувається переважно на етапі аналізу: Reviewer API після обробки відповіді від ШІ-сервісу створює запис про результати (список знайдених проблем) у БД, а також Event Handler фіксує факт перевірки конкретного pull request.

– *Фронтенд (клієнтський застосунок)*. Для зручності користувачів створено веб-інтерфейс, що дозволяє переглядати підключені репозиторії та їх pull request, запускати аналіз вручну (за потреби) та ознайомлюватися з детальними результатами перевірки. Фронтенд розроблено на основі сучасного фреймворку Angular; він взаємодіє з бекендом через HTTPS-запити до Pull Request API. Головний екран фронтенду відображає перелік останніх pull request зі статусом їх перевірки, а екран деталей аналізу – список виявлених помилок і рекомендацій з поясненнями. Для покращення користувацького досвіду реалізовано механізм push-нотифікацій: після завершення аналізу коду (що може тривати декілька десятків секунд при зверненні до моделі GPT-4) фронтенд отримує сигнал (через WebSocket) і автоматично оновлює інтерфейс з результатами, щоб розробник бачив зауваження майже в реальному часі.

Важливо зазначити, що вибір мікросервісної архітектури зумовлений вимогами до масштабованості та гнучкості системи. Зокрема, модуль Reviewer API (аналіз коду) є потенційно ресурсоємним, оскільки виконує запити до мовної моделі, що можуть мати затримки та обмеження по частоті. Тому цей модуль можна розгорнути окремо і масштабувати горизонтально (запустити кілька екземплярів), щоб обробляти кілька запитів паралельно. Інші компоненти (Event Handler, Pull Request API) здатні обробляти значно більшу кількість запитів з мінімальними ресурсами, тому їх можна масштабувати незалежно. Для взаємодії між сервісами використовуються легковагові REST-запити HTTP, що спрощує відлагодження та дозволяє при необхідності замінити один з сервісів (наприклад, перейти на іншу AI-модель у Reviewer API) без зміни коду інших.

Робота системи побудована навколо подій, що надходять від платформи контролю версій (у нашому випадку – GitHub). Загальний сценарій можна описати наступним чином: коли розробник створює новий pull request або додає зміни в існуючий, GitHub відправляє повідомлення (HTTP POST-запит) на спеціальний URL нашого сервісу Event Handler (цей URL реєструється як Webhook при підключенні репозиторію до платформи). Отримавши таке повідомлення, Event Handler виконує кілька кроків: перевіряє, чи належить pull request підключеному проєкту і чи потрібно його аналізувати, після чого формує запит на аналіз коду і надсилає його до Reviewer API. Reviewer API, у свою чергу, звертається до моделі GPT-4 з відповідним prompt (текстом-запитом) і отри-

мує від неї список знайдених проблем/порад. Цей список повертається в Event Handler, який зберігає результати у базі та додатково може зробити запит до GitHub API для автоматичного розміщення коментарів у pull request з описом кожної проблеми. Після завершення цих кроків розробники отримують сповіщення або можуть оновити сторінку pull request, щоб побачити коментарі від бота-рецензента.

На рис. 2 наведено UML-діаграму послідовності, що ілюструє взаємодію між користувачем, системою контролю версій та компонентами платформи у процесі автоматизованої перевірки коду.

Описана послідовність забезпечує повністю автоматизований цикл рев'ю коду одразу після створення pull request, без участі людини-рецензента на початковому етапі. Це дозволяє відфільтрувати тривіальні та очевидні проблеми (які система здатна знайти самостійно) ще до того, як до перегляду коду підключаться члени команди. Таким чином, економиться час досвідчених розробників: вони можуть зосередитись лише на нетривіальних аспектах змін, тоді як стилістичні питання, дрібні баги та інші речі, що легко перевіряються, вже будуть висвітлені в автоматичних коментарях.

З метою перевірки функціональності сторінки аналізу коду було створено pull request у власному репозиторії GameRecomendations, який на сторінці налаштування репозиторію підключено до платформи автоматизованої перевірки коду (рис. 3). Цей pull request під номером 4 має назву «Changes in FileDataLoader.cs» та включає зміну одного файлу, у якому код навмисно погіршено для зниження його якості та внесення помилок.

Після відкриття pull request він відобразився на сторінці списку (рис. 3). Згідно з результатами автоматизованої перевірки, у цьому pull request було виявлено 6 помилок і запропоновано 11 рекомендацій щодо покращення коду.

Частину рекомендацій також продемонстровано на сторінці аналізу коду (рис. 4).

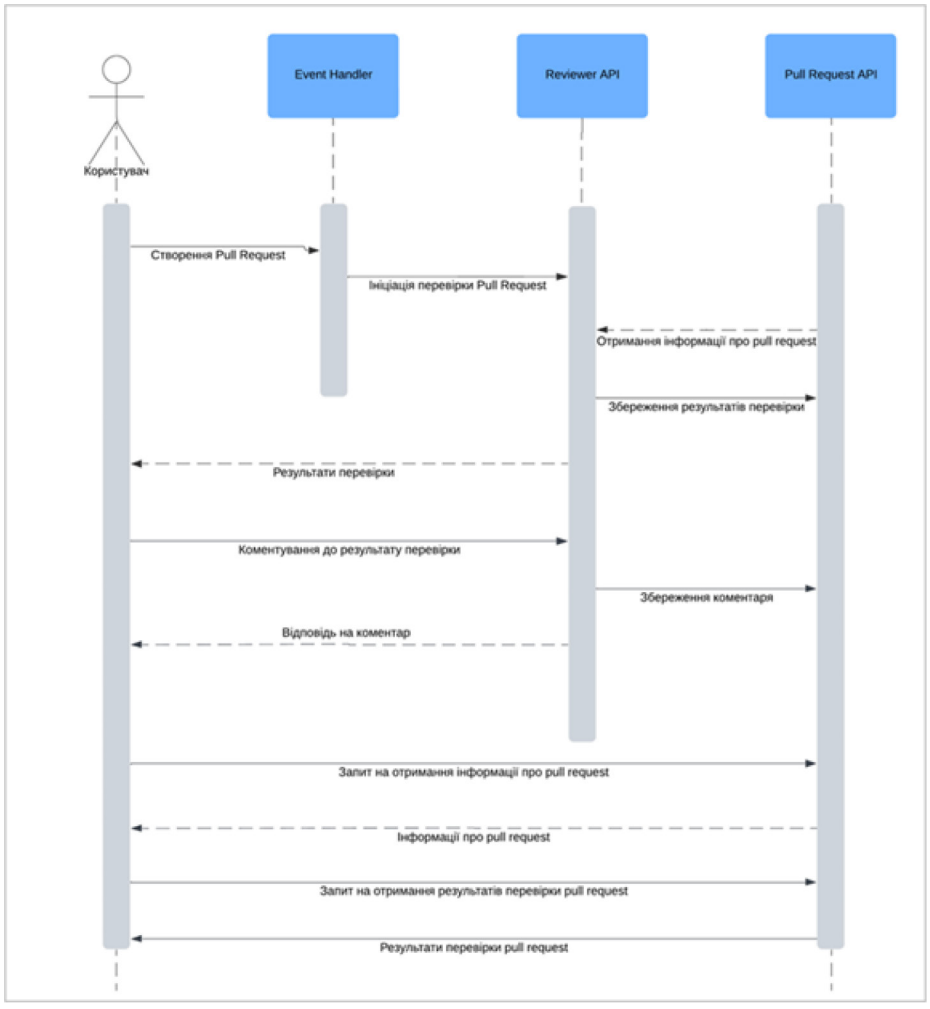

Рис. 2. Діаграма послідовності процесу автоматизованого рев'ю коду (взаємодія користувача, GitHub та мікросервісів платформи)

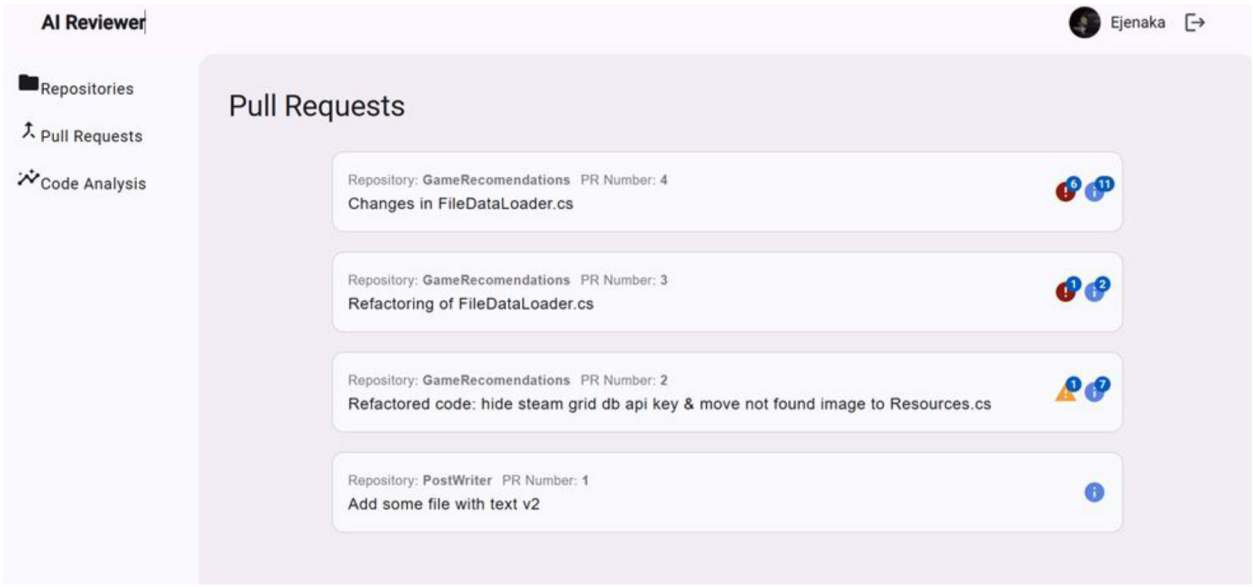

Рис. 3. Сторінка списку Pull Requestів у веб-застосунку платформи

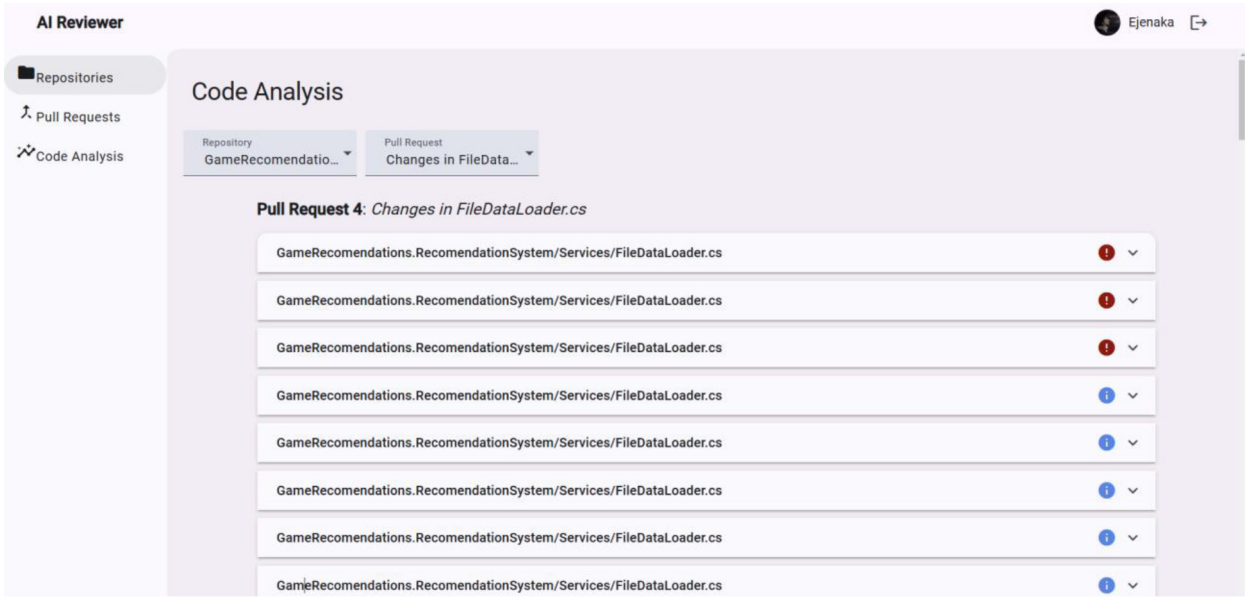

Рис. 4. Вигляд сторінки аналізу коду для створеного pull request

Декілька рекомендацій було переглянуто, і серед них виокремлено такі:

– помилка некоректного використання типів у C# (рис. 5);

– рекомендація щодо належного використання пробілу в коді (рис. 6);

– помилка при додаванні числового та рядкового значень, яка показана на рис. 7;

– рекомендація щодо видалення невикористовуваного методу (рис. 8).

З наведених рисунків видно, що система перевірки коду на платформі дійсно працює: вона аналізує програмний код на наявність помилок і пропонує покращення, які можна в ньому реалізувати.

Для перевірки роботи системи коментарів було залишено коментар з поясненням до рекомендації щодо належного використання пробілу в коді (рис. 6). Даний коментар разом із відповіддю на нього показано на рис. 9.

Система розгорнуто відповіла на коментар користувача та пояснила, які стандарти стосуються використання пробілів між операторами у мові програмування C#.

Для перевірки механізму зміни статусу рекомендації на «вирішено» для декількох рекомендацій було натиснуто кнопку «Resolve». Результат цих дій наведено на рис. 10.

Як видно, п’ять рекомендацій закрито для перегляду, оскільки їхній статус було змінено

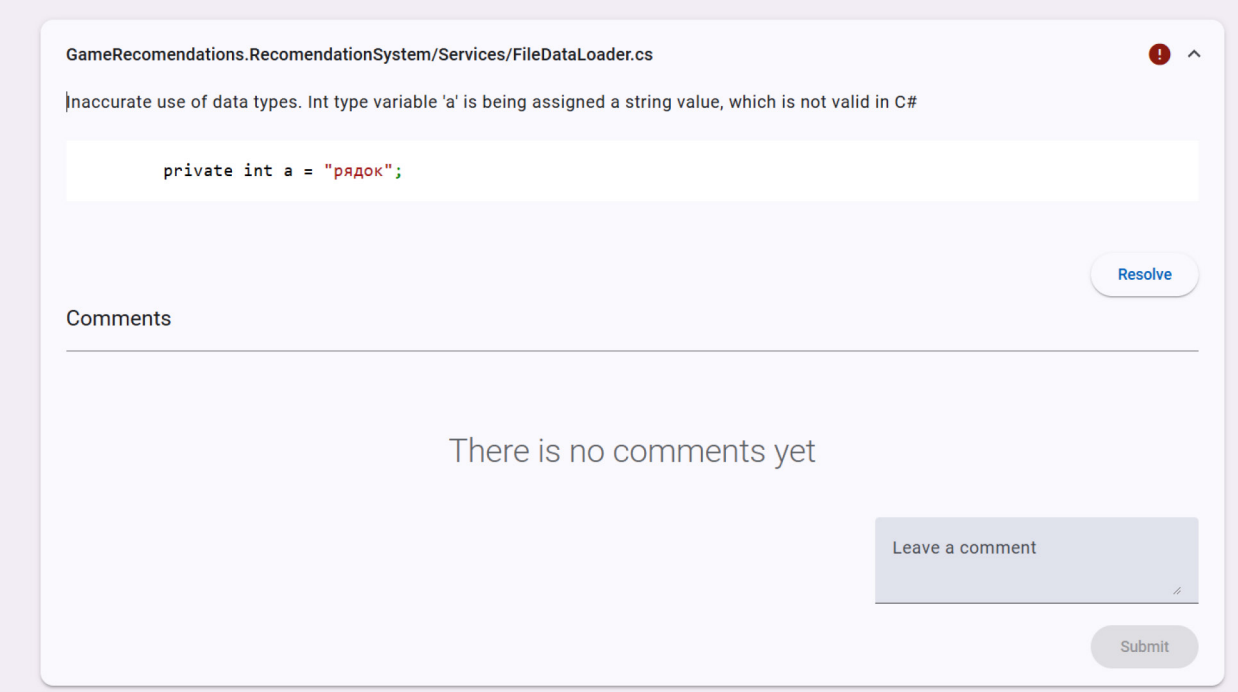

Рис. 5. Помилка некоректного використання типів

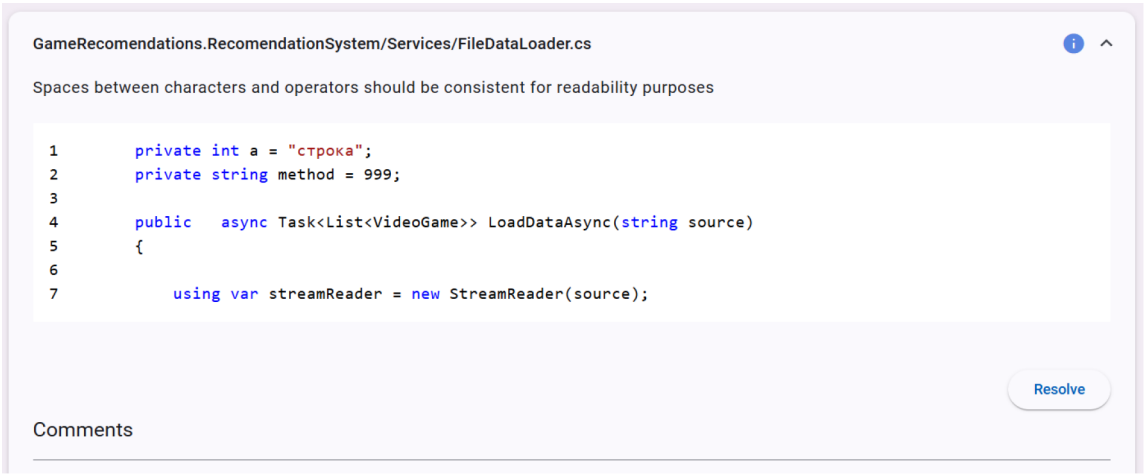

Рис. 6. Рекомендація щодо належного використання пробілу в коді

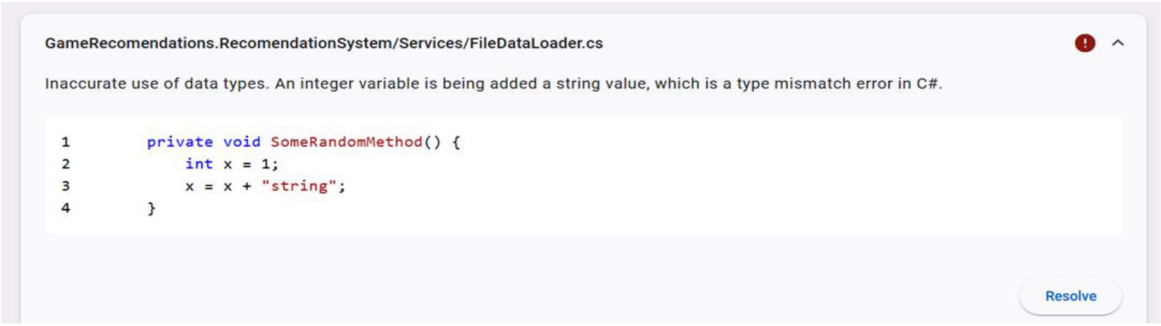

Рис. 7. Помилка при додаванні числового та рядкового значень

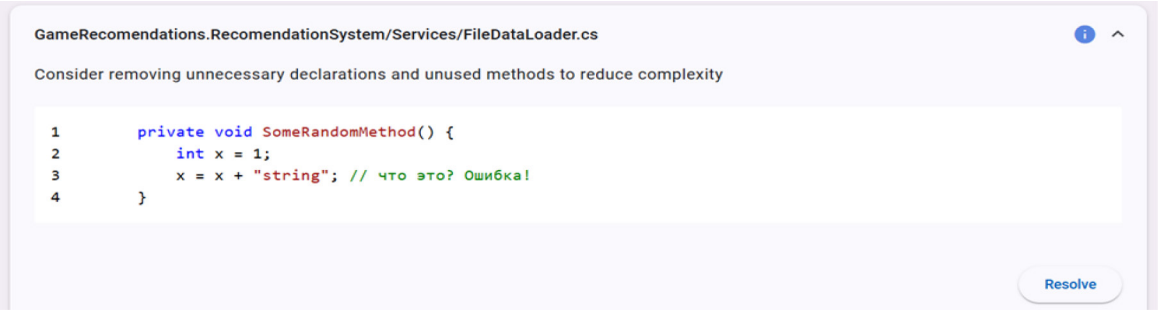

Рис. 8. Рекомендація щодо видалення невикористовуваного методу

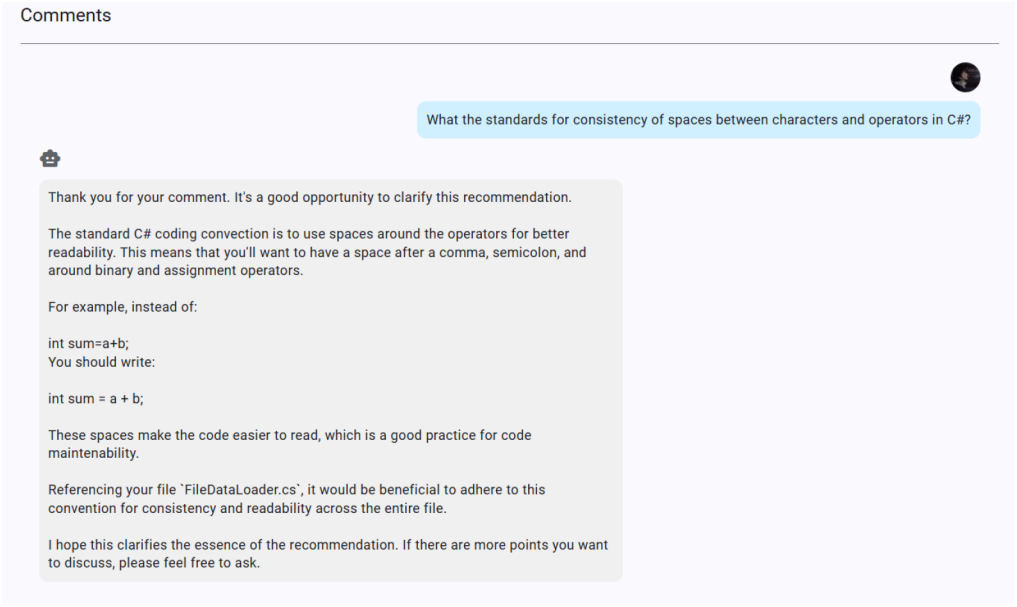

Рис. 9. Коментар і відповідь на нього

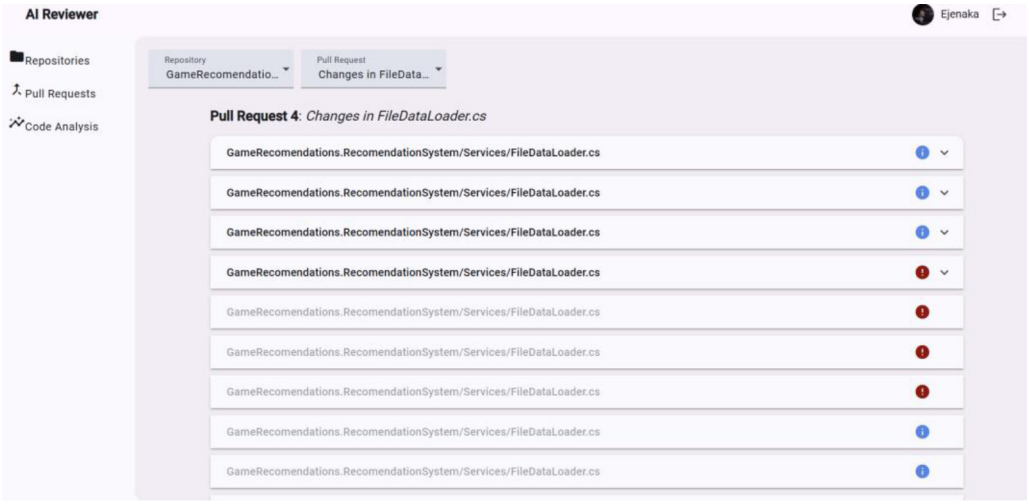

Рис. 10. Рекомендації зі статусом «вирішено»

на «вирішено». Отже, практичне випробування прототипу підтвердило, що запропонована платформа справді здатна полегшити процес рев'ю коду. Вона автоматично виявляє вагому частину тих проблем, на які зазвичай звертають увагу колеги-рецензенти, і робить це відразу після створення pull request. Безумовно, не всі зауваження, згенеровані моделлю, були ідеальними – окремі з них потребували додаткової перевірки. Однак під час тестування жодне важливе виправлення не

пройшло повз увагу системи. Ба більше, модель GPT-4 надала кілька порад з рефакторингу, які на перший погляд були неочевидними (наприклад, порадила винести дубльований код в окремий метод), що підкреслює цінність інтелектуального аналізу.

Висновки. Інтеграція сучасної мовної моделі (на прикладі GPT-4) у процес аналізу коду суттєво розширює можливості автоматизованого рев'ю. Система здатна виявляти не лише тривіальні помилки, але й складніші проблеми, пов'язані з логікою та дизайном коду, надаючи при цьому корисні пояснення. Це зменшує навантаження на команду та прискорює процес перевірки, особливо на ранніх етапах. Обраний підхід із використанням мікросервісної архітектури та інтеграції через GitHub Webhooks показав свою дієвість. Платформа органічно вбудовується в стандартний workflow розробників, автоматично реагуючи на події pull request і надаючи результати у звичному вигляді (коментарі у системі контролю версій). В ході тестування підтверджено, що якість зауважень AI-системи є досить високою: більшість згенерованих рекомендацій були корисними і виправданими. Водночас виявлено окремі випадки хибних або зайвих спрацьовувань, що вимагає обережності при інтерпретації результатів. На практиці це означає, що розробники повинні переглядати автоматичні коментарі критично, особливо якщо вони стосуються неоднозначних аспектів.

Перспективи подальших досліджень полягають у вдосконаленні як самої AI-моделі, так і платформи загалом. По-перше, варто розглянути можливість тонкого налаштування мовної моделі на домен програмування. Використання технік навчання з підкріпленням на основі зворотного зв'язку від розробників могло б зменшити кількість хибно-позитивних результатів. По-друге, актуальним є завдання підтримки більш широкого спектру мов програмування та фреймворків. GPT-4 вже зараз справляється з багатьма мовами, але спеціалізовані знання (наприклад, специфічні для низькорівневих мов C/C++ питання управління пам'яттю) можуть вимагати додаткових правил чи підказок. Розширення платформи на інші системи контролю версій (GitLab, Bitbucket) також є практично важливим, аби зробити її універсальнішою. По-третє, потрібно оптимізувати витрати на виконання аналізу: дослідження напряму використання локальних відкритих моделей (наприклад, StarCoder, CodeGen) або гібридного підходу (коли швидкий попередній аналіз виконується меншою моделлю, а тільки складні випадки надсилаються до GPT-4) допоможе зробити платформу економічнішою та автономнішою. І нарешті, цікавим напрямом є доповнення функціоналу системи засобами динамічного аналізу – інтеграція з інструментами тестування, запуск тестів або аналіз виконання програми для виявлення тих проблем, які статичному аналізу не підвладні.

Таким чином, впроваджений підхід продемонстрував життєздатність і доцільність використання ШІ для автоматизації рев'ю коду. Надалі слід зосередитися на підвищенні точності і доречності рекомендацій штучного інтелекту та розширенні можливостей платформи, що сприятиме ще більшому поширенню таких інструментів у індустрії розробки програмного забезпечення. Це, у свою чергу, допоможе командам розробників підтримувати високий рівень якості коду і ефективніше керувати складністю сучасних програмних проєктів.

Список літератури:

1. Bansal G., Chamola V., Hussain A. Transforming Conversations with AI – A Comprehensive Study of ChatGPT. Cognitive Computation, 2024, Vol. 16, No 8, pp. 2487–2510. DOI: 10.1007/s12559-023-10236-2

2. Diksha K. та ін. Natural language processing: state of the art, current trends and challenges. Multimedia Tools and Applications, 2023, Vol. 82, pp. 3713–3744. DOI: 10.1007/s11042-022-13428-4

3. SonarQube. SonarSource: вебсайт. URL: https://www.sonarsource.com/products/sonarqube (дата звернення: 18.09.2024).

4. What is Danger? GitHub: вебсайт. URL: https://github.com/danger/danger (дата звернення: 18.09.2024).

5. ESLint. ESLint: вебсайт. URL: https://eslint.org (дата звернення: 20.09.2024).

6. Finn T., Downie A. AI Code Review. IBM Blog, 15 Oct 2024. URL: https://www.ibm.com/think/insights/ai-code-review (дата звернення: 01.03.2025).

7. Umut C., Haratian V. Automated Code Review In Practice [Електронний ресурс] / arXiv preprint. 2024. № arXiv:2412.18531. 13 p. URL: https://arxiv.org/abs/2412.18531 (дата звернення: 07.03.2025)

Horban H. V., Fisun M. T., Ralenko V. S., Stoiev Ye. D. PLATFORM FOR AUTOMATED CODE VERIFICATION USING ARTIFICIAL INTELLIGENCE TECHNOLOGIES

The paper addresses the problem of improving the efficiency and quality of the code review process by developing a platform for automated code verification using artificial intelligence technologies. Modern automated code analysis tools and their limitations are analyzed: standard static code analyzers (such as SonarQube, Codacy, ESLint) can detect syntax errors or style violations but do not consider the context of program logic and may generate false positives. The architecture of an information system integrated with the GitHub platform and using the ChatGPT-4 model for "intelligent" code review is proposed. The platform is built on a microservice principle: a GitHub integration module (pull request event handler), an AI-based code analysis module, and a pull request data management module are delineated. UML diagrams of the system's architecture and component interaction sequence are provided. A web application (Angular) has been developed to display analysis results, and automatic commenting of identified issues directly in the GitHub pull request is implemented. The platform prototype was tested on a real repository example: the system successfully identified logical errors and provided code improvement recommendations (in a test pull request, 6 errors were found and 11 improvements suggested), confirming the viability and effectiveness of the approach. Conclusions are drawn about the advantages of using artificial intelligence in code verification (reducing the burden on the team, deeper analysis of program logic), and directions for further system development are outlined, such as supporting more programming languages, optimizing the cost of API calls to language models, and expanding the platform's functionality.

Key words: *automated code review, code inspection, artificial intelligence, pull request, GitHub, .NET, Angular.*