Plag

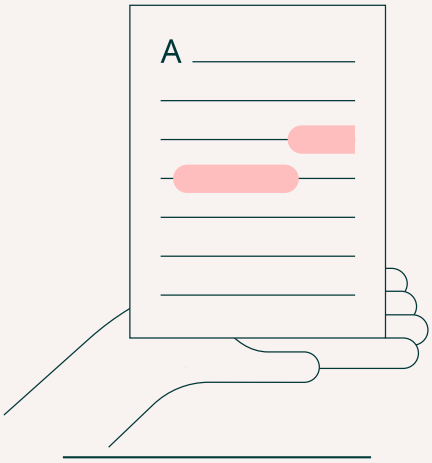
Звіт про оригінальність

● Оцінка схожості 3%

● Ризик плагіату НИЗЬКИЙ

Ігор Кагало 2025-06-14 10:25

Посилання на звіт: 10biE / Посилання користувача: qAHy



Ось вона –
Ваша звіт про оригінальність!

Ми раді повідомити, що перевірка вашого документа завершена, і результати вже готові! Наші алгоритми старанно працювали, щоб знайти збіги в наших базах даних.

На наступних сторінках ви знайдете результати перевірки:

Бали
Збіги
Посилання

Ваш документ було перевірено за такими джерелами:

- ✓ База даних інтернет-джерел
- ✓ База даних наукових статей
- ○ Глибока перевірка (наш вдосконалений алгоритм)

Бали

Збіги тексту	3%
Перефразування	0%
Цитований текст	0%
Неправильне цитування	0%
Збігів не знайдено	97%

Ризик плагіату НИЗЬКИЙ

Ризик плагіату вказує, як збіги тексту розподілені по документу. Вищий ризик виникає, коли збіги з'являються близько один до одного, наприклад, у тому самому абзаці або розділі.

Оцінка схожості 3%

Оцінка схожості показує, скільки слів або символів у вашому документі збігаються з текстами інших документів, включаючи перефразовані тексти або неправильні цитати.

Збіги

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Поняття систем зворотного зв’язку та моніторингу помилок

Системи зворотного зв'язку та моніторингу помилок є наріжними каменями у процесі розробки та підтримки сучасного програмного забезпечення. Їхня синергія забезпечує не лише технічну стабільність, але й сприяє постійному вдосконаленню продукту відповідно до реальних потреб користувачів. Ці інструменти призначені для всебічної підтримки якості програмного забезпечення через виявлення, реєстрацію, аналіз помилок та прийом цінної інформації від користувачів. Їхня кінцева мета — максимально швидке виявлення та усунення несправностей, а також оптимізація та поглиблення взаємодії між користувачем і командою розробки.

Моніторинг помилок можна назвати "очима" системи. У динамічному світі вебзастосунків, особливо тих, що побудовані на гнучких фреймворках, таких як Flask, помилки є неминучою частиною життєвого циклу. Вони можуть виникати на найрізноманітніших рівнях, що потребує комплексного підходу до їхнього відстеження. Це можуть бути помилки на рівні бекенду, що виникають при обробці складних HTTP-запитів, некоректній взаємодії з базами даних, або ж внаслідок логічних помилок у серверному коді. Не менш важливими є помилки на фронтенді, наприклад, JavaScript-винятки в браузері, які можуть призвести до некоректного відображення інтерфейсу або непрацездатності окремих елементів.

Система моніторингу помилок функціонує як постійно пильні "очі" застосунку, автоматично фіксуючи та логуючи будь-які винятки, що виникають у процесі його роботи. Що особливо цінно, вона надає розробникам не просто факт помилки, а й глибокий контекст виконання. Цей контекст включає трасування стека викликів, стан змінних на момент збою, інформацію про операційну систему, браузер користувача, а також дані вхідних запитів. Такий обсяг інформації є критично важливим для швидкої та точної діагностики. Ефективний моніторинг помилок дозволяє розробникам оперативно реагувати на несправності, часто ще до того, як вони стануть широко поширеними або суттєво вплинуть на користувацький досвід. Це дозволяє проактивно підтримувати стабільність системи та мінімізувати простої.

Якщо моніторинг помилок є "очима" системи, то зворотний зв'язок є її "голосом" — прямим каналом комунікації з кінцевим користувачем. Він виходить за рамки автоматичного виявлення технічних збоїв, 12 надаючи користувачам можливість ділитися своїми враженнями, повідомляти про проблеми, пропонувати поліпшення або висловлювати свої думки щодо наявного функціоналу.

Цей механізм є незамінним для виявлення "тихих" проблем — тих, які не призводять до технічних винятків, але можуть суттєво погіршувати користувацький досвід (наприклад, незрозумілий інтерфейс, неочевидні робочі процеси, або функціонал, який не відповідає очікуванням). Зворотний зв'язок також є невичерпним джерелом ідей для подальшого розвитку продукту. Користувачі, що активно взаємодіють із системою, часто можуть запропонувати неочікувані, але дуже цінні рішення або нові функції. Надаючи користувачам можливість легко та ефективно повідомити про проблему або запропонувати поліпшення, розробники не лише покращують продукт, але й роблять систему більш орієнтованою на користувача. Такий підхід сприяє прийняттю обґрунтованих рішень щодо пріоритетів у розробці та створює відчуття спільної участі у створенні продукту, що в свою чергу підвищує лояльність користувачів та їхнє задоволення від використання системи.

1.2 Вимоги до систем управління зворотним зв’язком

Для побудови ефективної системи зворотного зв’язку та моніторингу помилок у вебдодатку, яка дійсно буде корисною як для розробників, так і для користувачів, необхідно врахувати низку ключових вимог. Ці вимоги поділяються на функціональні, що описують, що система повинна робити, та нефункціональні, що визначають, як вона це повинна робити.

Функціональні вимоги це те, що система повинна робити. Функціональні вимоги окреслюють основні можливості, які система повинна надавати для успішного збору та обробки інформації. Сюди входить: збір даних про помилки, можливість залишити зворотний зв’язок, автоматичне сповіщення розробника, зручний інтерфейс для аналізу помилок тощо.

Детальне логування винятків. Система повинна мати здатність фіксувати різноманітні типи винятків, що виникають на сервері, включаючи HTTP-винятки (наприклад, помилки 4xx і 5xx), 500-помилки (внутрішні помилки сервера), а також SQL-винятки (помилки, пов'язані з базою даних). Цей процес має бути автоматизованим, щоб не вимагати ручного втручання.

Збереження контексту помилки. Недостатньо просто зафіксувати факт помилки. Система повинна зберігати максимально повну інформацію про контекст її виникнення. Це включає дані про користувача, який зіткнувся з проблемою

(анонімізовані або ідентифіковані, залежно від політики безпеки), дії, що призвели до помилки (послідовність кроків або запитів), IP-адресу користувача та його user-agent (інформація про браузер та операційну систему). Це дозволяє розробникам відтворити сценарій та швидко знайти корінь проблеми.

Зручні форми комунікації. Система має надавати користувачам прості та інтуїтивно зрозумілі способи для залишення зворотного зв'язку. Це може бути інтегрована контактна форма, спеціальне модальне вікно відгуку, доступне з будь-якої сторінки, або навіть віджет.

Підтримка мультимедіа. Для кращого розуміння проблеми користувачі повинні мати можливість завантажувати скриншоти або надавати детальний опис помилки вручну. Візуальні матеріали значно спрощують діагностику та дозволяють уникнути непорозумінь.

Миттєва доставка повідомлень. Коли виникає критична помилка або надходить важливий зворотний зв'язок, система повинна автоматично сповіщати команду розробників. Це може бути реалізовано через email, інтеграцію зі службами обміну повідомленнями, такими як Slack або Telegram, або через інші API.

Інтерфейс для аналізу помилок являє собою централізовану адміністративну панель. Розробникам необхідна спеціалізована адміністративна панель для зручного перегляду та аналізу історії помилок та повідомлень зворотного зв'язку.

Ефективні інструменти фільтрації та пошуку. Ця панель повинна надавати розширені можливості для фільтрації даних за різними критеріями. часом виникнення, типом помилки, статусом вирішення (наприклад, "нова", "в процесі", "вирішена"), а також за іншими параметрами, такими як користувач або URL. Це дозволяє швидко знаходити та групувати пов'язані проблеми.

Нефункціональні вимоги це те, 8 як система повинна працювати. Нефункціональні вимоги визначають атрибути якості системи, такі як продуктивність, безпека та надійність.

Масштабованість. Система зворотного зв'язку та моніторингу 3 повинна бути спроектована таким чином, щоб ефективно обробляти значне збільшення кількості запитів і повідомлень без втрати продуктивності. Зростання аудиторії або збільшення кількості помилок не повинно призводити до збоїв або уповільнення роботи самої системи моніторингу.

Безпека. Збір даних про користувачів та їхні дії вимагає особливої уваги до безпеки. Особисті дані користувача мають бути належним чином анонімізовані або захищені

згідно з відповідними нормативними вимогами, такими як GDPR (Загальний регламент про захист 3 даних). Це включає шифрування даних, обмеження доступу та регулярний аудит безпеки.

Продуктивність. Система моніторингу та зворотного зв'язку не повинна створювати додаткового навантаження на основний вебдодаток. Її робота не повинна впливати на швидкість завантаження сторінок або обробку запитів від кінцевих користувачів. Це означає, що процеси логування та відправки повідомлень мають бути максимально асинхронними та ефективними.

Мінімальне втручання. Процес відправки помилки або зворотного зв'язку має бути максимально непомітним і легким для користувача. Це означає, що відправка інформації не повинна викликати додаткове навантаження на клієнтську сторону (наприклад, зависання браузера) або вимагати складних дій від користувача. В ідеалі, це має відбуватися автоматично або за один-два кліки, мінімізуючи будь-які перешкоди для користувацького досвіду.

Врахування всіх цих вимог дозволяє створити надійну та ефективну систему, яка стане цінним активом для будь-якого вебдодатка, сприяючи його стабільності та постійному розвитку.

1.3 Огляд існуючих вебсистем для моніторингу помилок та зворотного зв’язку

На сучасному етапі розвитку веброзробки існує широкий спектр спеціалізованих вебсистем, призначених для ефективного моніторингу помилок та управління зворотним зв'язком. Ці платформи допомагають розробникам не лише 7 виявляти та виправляти збої, але й розуміти, як користувачі взаємодіють з їхніми продуктами, що є критично важливим для постійного вдосконалення. Нижче представлено огляд деяких з найпопулярніших та функціональних рішень.

Sentry — це потужна, з відкритим кодом (хоча доступна і як SaaS-рішення) платформа, яка фокусується на моніторингу помилок у реальному часі. Вона 7 дозволяє розробникам швидко виявляти, відстежувати та усувати збої в програмному забезпеченні на різних етапах його життєвого циклу. Її архітектура розроблена для мінімізації впливу на продуктивність додатка, одночасно надаючи максимальний обсяг діагностичної інформації.

Автоматичне виявлення виключень. Sentry автоматично перехоплює та агрегує винятки, що виникають як на серверній, так і на клієнтській стороні, надаючи миттєві сповіщення.

Глибока інтеграція з Flask. Завдяки офіційному sentry-sdk, інтеграція з Flask є надзвичайно простою. Це дозволяє легко налаштовувати перехоплення помилок, що

виникають у вашому вебдодатку, забезпечуючи повний контроль над процесом моніторингу.

Збір розширеного контексту. Крім самої помилки, Sentry автоматично збирає цінну контекстну інформацію. змінні оточення, дані про користувача (які можна анонімізувати), версії програмного забезпечення, трасування стека викликів, HTTP-запити та інше. Це дає розробнику повну картину того, що призвело до збою.

Широкий спектр інтеграцій. Sentry легко інтегрується з популярними інструментами розробки та комунікації, такими як GitHub (для прив'язки помилок до конкретних комітів), Slack (для миттєвих сповіщень команді), та багатьма іншими, що оптимізує робочі процеси.

LogRocket — це унікальний сервіс, який виходить за межі традиційного моніторингу помилок, пропонуючи можливість запису сесій користувачів. Це дозволяє розробникам буквально "відтворити" те, що бачив і робив користувач безпосередньо перед виникненням помилки, надаючи безпрецедентний рівень деталізації для діагностики проблем.

Запис дій користувача на клієнтській стороні. LogRocket записує всі взаємодії користувача з інтерфейсом. рухи миші, кліки, введення тексту, прокручування сторінки. Це дозволяє точно зрозуміти сценарій, який призвів до проблеми.

Комплексний аналіз інтерфейсу та UX-проблем. Завдяки візуалізації сесій, розробники можуть не лише виявляти технічні помилки, але й ідентифікувати проблеми з UX (User Experience) та юзабіліті, що значно покращує загальний досвід користувача.

Відображення консольних повідомлень, мережевих запитів та кліків. Разом із візуальним записом, LogRocket показує логи консолі браузера, всі мережеві запити, що відбувалися під час сесії, та навіть деталі кожного кліка. Це забезпечує глибокий інсайт у поведінку фронтенду.

Bugsnag — це ще одна потужна платформа для виявлення та відстеження збоїв, яка підтримує широкий спектр платформ. мобільні (iOS, Android), веб- (JavaScript, Ruby, Python тощо) та десктоп-додатки. Вона допомагає командам швидко виявляти критичні проблеми та підтримувати стабільність своїх продуктів.

Інтелектуальне групування помилок за типом. Bugsnag використовує алгоритми для автоматичного групування подібних помилок, що значно зменшує "шум" і дозволяє розробникам зосередитися на найбільш поширених або критичних проблемах.

Відслідковування стабільності релізів. Платформа надає інструменти для моніторингу

впливу нових релізів на стабільність додатка, дозволяючи швидко виявляти та відкочувати проблемні зміни.

Гнучкі інтеграції з CI/CD. Bugsnag легко інтегрується з системами 6 безперервної інтеграції та доставки (CI/CD), що дозволяє автоматизувати процес моніторингу та включити його в загальний пайплайн розробки.

Jira — це багатофункціональна система управління проектами та задачами, яка широко застосовується в IT-індустрії. Хоча Jira не є безпосередньо системою моніторингу помилок або збору зворотного зв'язку, вона є ключовим інструментом для управління процесом вирішення інцидентів, які виявляють інші системи.

Створення задач вручну або автоматично. Задачі (наприклад, баги, покращення, нові функції) можуть бути створені в Jira вручну командою або автоматично через інтеграцію з такими системами, як Sentry (коли помилка фіксується, відповідна задача створюється в Jira).

Призначення відповідальних та дедлайнів. Jira дозволяє ефективно розподіляти завдання між членами команди, призначати відповідальних осіб та встановлювати дедлайни, що забезпечує структурований підхід до вирішення проблем.

Відстеження прогресу вирішення інцидентів. За допомогою Jira команди можуть відстежувати життєвий цикл кожної задачі. від моменту її створення до повного вирішення. Це включає зміну статусів, коментарі, прикріплення файлів та інші деталі, що забезпечує прозорість процесу та дозволяє оцінювати ефективність команди.

11 Узагальнений аналіз результатів проведеного дослідження наведено у Таблиці 1.1

Таблиця 1.1 - Аналіз переваг та недоліків існуючих рішень

Система

Переваги

Недоліки

Sentry

Глибока інтеграція з Python/Flask, багатий API, контекст помилки

У безкоштовній версії обмежено кількість подій/місяць

LogRocket

Детальний запис поведінки користувача, UX-аналітика

Не фіксує бекенд-винятки, потребує великої пропускної здатності

Bugsnag

Зручна панель для аналітики стабільності додатку

Платна модель, менша гнучкість в кастомізації для Python

Jira

Широкі можливості для управління життєвим циклом задач

Складна конфігурація, потребує адаптації до потреб невеликого проєкту

Вибір конкретної системи або комбінації систем залежить від масштабу проекту, його специфічних потреб, бюджету та вимог до інтеграції з існуючими інструментами розробки. Результати проведеного аналізу будуть враховані при розробці вебсистеми управління зворотнім зв’язком та моніторингом помилок у програмних проектах в межах даної дипломної роботи

2 ПРОЄКТУВАННЯ ВЕБСАЙТУ

2.1 Визначення основних функціональних вимог

Для створення ефективної системи зворотного зв'язку та моніторингу помилок у вебдодатку, необхідно чітко визначити її функціональні можливості. Ці вимоги поділяються на ті, що стосуються користувачів, адміністраторів та самої системи.

Вимоги до вебсистеми, які стосуються користувача:

Система має бути зручною для користувачів, дозволяючи їм легко надсилати відгуки.

Користувачі повинні мати простий вебінтерфейс для надсилання пропозицій, скарг чи запитань.

Форма звернення повинна включати поля для імені, email, теми та тексту повідомлення.

Після надсилання форми користувач має отримати підтвердження про її успішну доставку.

Вимоги до вебсистеми, які стосуються адміністратора:

Адміністративна панель повинна забезпечувати ефективне управління та аналіз отриманої інформації.

Доступ до адмін-панелі має бути захищеним за допомогою авторизації.

Усі отримані повідомлення мають відображатися у зручній табличній формі.

Можливість сортувати та фільтрувати повідомлення за статусом (наприклад, "нове", "опрацьоване").

Адміністратор повинен мати можливість позначати повідомлення як опрацьовані або видаляти їх.

Загальні вимоги до вебсистеми:

Технічна частина системи має бути надійною та автоматизованою.

Автоматичний запис усіх помилок, що виникають у вебдодатку.

Збереження детальних винятків у журнал подій з контекстною інформацією.

Автоматичне сповіщення розробників про критичні помилки (наприклад, через email або інтеграцію з Sentry/Jira).

Описаний набір вимог до вебсистеми управління зворотнім зв’язком та моніторингом помилок у програмних проектах дозволить отримати зручний, користниу та ефективний результат.

2.2 Архітектура вебсистеми

Архітектура побудована за принципом класичної клієнт-серверної моделі. Основними її компонентами є:

1. Клієнтська частина (Frontend).

Представляє вебінтерфейс, через який користувачі взаємодіють із системою;

Включає форми введення зворотного зв’язку, таблиці для перегляду повідомлень (у адмін-панелі), кнопки дій.

2. Серверна частина (Backend).

Побудована на основі Flask (Python);

Приймає HTTP-запити, обробляє логіку, зберігає та видає дані;

Обробляє винятки, логуючи їх у БД або файл.

3. База даних.

Відповідає за зберігання інформації про зворотній зв’язок, користувачів (адмінів), логи;

Взаємодія відбувається через ORM SQLAlchemy.

4. Комунікація між компонентами.

Здійснюється через HTTP-запити. форми передають POST-запити на Flask-сервер, який обробляє та відповідає JSON або перенаправленням;

Для обробки помилок використовується try-except із логуванням.

Схема взаємодії компонентів зображена на рис. 2.1.

Рис 2.1 - Схема взаємодії компонентів

2.3 1 Опис структури бази даних

1 У 2 проєкті використовується база даних, модельована за допомогою SQLAlchemy ORM (Object-Relational Mapper). Це дозволяє взаємодіяти 1 з базою даних, використовуючи об'єкти Python, що спрощує роботу з даними та забезпечує гнучкість. Основні структури бази даних подано у таблицях 2.1-2.3, що описують сутності для логів, зворотного зв'язку та адміністраторів.

Таблиця 2.1- logs, таблиця помилок

Поле

Тип

Опис

id

Integer, PK

Унікальний ідентифікатор

error_type

String

Тип помилки

message

Text

Текст помилки / traceback

created_at

DateTime

Дата й час помилки

request_data

Text

Інформація про запит (user-agent, IP тощо)

Таблиця 2.2 – feedback, таблиця зворотного зв’язку

Поле

Тип

Опис

id

Integer, PK

Унікальний ідентифікатор

name

String

Ім’я користувача

email

String

Email користувача

subject

String

Тема повідомлення

message

Text

Текст повідомлення

status

String

Статус. new / reviewed

created_at

DateTime

Дата створення

Таблиця 2.3- admin, таблиця адміністраторів

Поле

Тип

Опис

id

Integer, PK

Унікальний ідентифікатор

username

String

Логін

password

String (hash)

Пароль (у вигляді хешу)

2.4 Вибір технологій для реалізації

Для реалізації системи було обрано стек технологій, який поєднує простоту, гнучкість і достатню потужність для вебзастосунку середнього масштабу (Таблиця 2.4).

Таблиця 2.4 – Використані технології

Технологія

Призначення

Обґрунтування вибору

HTML5

Розмітка сторінок, форм зворотного зв’язку, кнопок, модалок

Сучасний стандарт для структурованої веб-розмітки, що забезпечує семантичність, доступність та сумісність з усіма сучасними браузерами

CSS3

Стилізація елементів інтерфейсу, включаючи теми та анімації

Остання версія каскадних таблиць стилів, що надає широкі можливості для дизайну, адаптивності та створення динамічних візуальних ефектів. Це дозволило реалізувати естетично привабливий дизайн, включаючи світлу та темну теми, та забезпечити адаптивність інтерфейсу під різні розміри екранів

JavaScript

Динамічна поведінка форм, підтвердження, інтерактивність

Мова програмування для фронтенду, що дозволяє реалізувати інтерактивні елементи інтерфейсу, валідацію форм на стороні клієнта, анімації та інші динамічні функції без перезавантаження сторінки

Python

Основна мова реалізації логіки застосунку, обробка запитів

Високорівнева, інтерпретована мова програмування, відома своєю читабельністю, простотою та великою кількістю бібліотек

Flask

Легкий вебфреймворк на Python для побудови REST-серверу

Мінімалістичний, але потужний мікрофреймворк для Python. Flask забезпечує гнучку структуру для маршрутизації, обробки запитів та керування сесіями, що ідеально

підходить для застосунку середнього масштабу

Обраний 9 стек технологій, що включає HTML5, CSS3, JavaScript, Python та Flask, є оптимальним рішенням для розробки вебзастосунку середнього масштабу. HTML5, CSS3 та JavaScript забезпечують створення сучасного, візуально привабливого та інтерактивного фронтенду, здатного адаптуватися до різних пристроїв. Для бекенду, Python разом з Flask дозволяють швидко розробляти надійну логіку застосунку завдяки своїй простоті та гнучкості. Такий вибір гарантує створення функціонального, естетичного та легко підтримуваного рішення.

3 РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ

3.1 Розробка серверної частини на Flask

На етапі розробки серверної частини системи дійсно було обрано мікрофреймворк Flask, що відомий своєю гнучкістю та мінімалістичністю, дозволяючи розробнику самостійно обирати компоненти та структуру проєкту. Основна логіка серверної частини зосереджена у файлі app.py. Тут відбувається ініціалізація екземпляра Flask-додатку (app = Flask(__name__)), завантаження конфігурацій, таких як шлях до бази даних (app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite.///tasks.db') та секретний ключ (app.config['SECRET_KEY'] = os.environ.get('FLASK_SECRET_KEY', '...')), а також налаштування папки для завантажених файлів (app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER). У цьому ж файлі ініціалізуються розширення Flask, зокрема db = SQLAlchemy() для 1 роботи з базою даних та migrate = Migrate(app, db) для управління міграціями схеми бази даних за допомогою Alembic.

Архітектура серверної частини, справді нагадує принципи MVC, хоча у Flask це частіше називають Model-View-Template (MVT) або просто розглядають функції-обробники як контролери:

Моделі (Model) представлені класами у файлі models.py, що успадковуються від db.Model SQLAlchemy. Ці класи (User, Task, Comment, Rating) описують структуру таблиць бази даних, їхні поля та взаємозв'язки через db.Column, db.ForeignKey та db.relationship. Це дозволяє абстрагуватися від написання прямих SQL-запитів.

Вигляд/Шаблони (View/Template) реалізовані через HTML-шаблони, які рендеряться функціями-контролерами (детальніше у п. 3.2).

Контролери (Controller) представлені функціями-обробниками маршрутів у файлі app.py (наприклад, login(), register(), create_task(), admin_dashboard()).

Маршрути, відповідальні за обробку HTTP-запитів, таких як GET та POST, реалізовані безпосередньо у головному файлі застосунку, app.py. 2 Це означає, що логіка

визначення URL-шляхів та їх прив'язки до відповідних функцій-контролерів зосереджена в одному місці. Ця прив'язка здійснюється за допомогою декораторів @app.route(...), які розміщуються безпосередньо над функціями, що виконують певні дії у відповідь на запити.

Щодо конфігураційних параметрів, таких як SECRET_KEY та SQLALCHEMY_DATABASE_URI , вони визначені у файлі diplom/config.py. Ці параметри інкапсульовані в класі Config у цьому файлі. У поточній імплементації, застосунок app.py безпосередньо завантажує ці параметри, присвоюючи їх значення конфігураційному об'єкту app.config через синтаксис app.config[...] = ...

Скрипт create_db.py виконує функцію ініціалізації бази даних, викликаючи db.create_all() в контексті додатку, що створює таблиці на основі визначених моделей, як і зазначено. Це корисно для першого запуску або скидання бази даних у середовищі розробки.

Загалом, серверна частина забезпечує основний функціонал. обробку HTTP-запитів, взаємодію з базою даних для збереження та отримання даних користувачів і завдань, управління сесіями користувачів, а також завантаження файлів.

3.2. Реалізація клієнтської частини

Клієнтська частина системи, як і описано, реалізована з використанням стандартних веб-технологій. HTML для структурування контенту, CSS для його стилізації, та JavaScript для забезпечення інтерактивності. Усі HTML-шаблони розміщено у директорії templates, що є стандартною практикою для Flask-проєктів.

Система містить розгалужену структуру шаблонів для різних аспектів взаємодії:

Автентифікація та реєстрація користувачів. login.html, register.html.

Перегляд та редагування профілю. profile.html, edit_profile.html.

Створення та перегляд завдань. create_task.html, view_task.html.

Панелі для різних ролей користувачів. user_dashboard.html для звичайних користувачів, та admin_dashboard.html і admin_users.html для адміністратора.

Також наявна головна сторінка index.html та службова сторінка success.html. Файл diplom/html.html є окремим файлом з HTML-розміткою, що не використовується в основній системі шаблонів Flask, а скоріше є чернеткою або прикладом.

HTML-шаблони активно використовують можливості шаблонізатора Jinja2.

Вставка змінних. {{ task.title }}, {{ user.username }}.

Умовні блоки. {% if session.role == 'admin' %} ... {% else %} ... {% endif %} для відображення різних елементів залежно від ролі користувача.

Цикли. {% for task in tasks %} для ітерації по списку завдань.

Виклик функцій Flask (url_for). {{ url_for('login') }} для генерації URL-адрес.

Обробка flash-повідомлень. {% with messages = get_flashed_messages(with_categories=true) %}.

CSS-стилізація шаблонів (admin_dashboard.html, user_dashboard.html, profile.html, create_task.html та інших) реалізована через вбудовані теги <style>. Шаблони login.html та register.html додатково використовують CSS-фреймворк Bootstrap для швидкого формування візуального вигляду. Винесення кастомних CSS-стилів в окремі файли значно покращує організацію коду, його перевикористання та полегшило б підтримку адаптивного дизайну.

JavaScript використовується для покращення користувацького досвіду.

Реалізовано функціонал перемикання теми (світла/темна) зі збереженням вибору у localStorage у більшості інтерактивних сторінок.

Динамічне відображення flash-повідомлень, що приходять з сервера.

Обробка завантаження файлів у create_task.html, де відображається ім'я обраного файлу.

У проєкті реалізовано систему реєстрації з підтвердженням електронної пошти, що підвищує безпеку та валідацію користувачів. Коли новий користувач реєструється, його дані тимчасово зберігаються в таблиці PendingUser бази даних, а на вказану електронну адресу надсилається унікальний 6-значний код підтвердження. Цей процес ініціюється функцією send_confirmation_email з модуля email_service.py, яка використовує Flask-Mail для відправлення HTML-листа з кодом та посиланням для підтвердження (Рис 3.1). Код підтвердження має обмежений термін дії, встановлений на 30 хвилин. Після отримання та введення користувачем коректного коду, його обліковий запис переноситься з PendingUser до основної таблиці User, що завершує реєстрацію та дозволяє увійти до системи. У випадку невірного коду або закінчення терміну дії, тимчасовий запис може бути видалений, а користувачу пропонується зареєструватися повторно.

Рисунок 3.1 – Форма завершення реєстрації

У проекті також реалізовано інтерактивність зірок рейтингу на сторінці view_task.html. JavaScript застосовується для динамічного керування елементами інтерфейсу, зокрема для функціоналу перемикання тем оформлення та обробки сповіщень. Основна увага JS-коду зосереджена саме на цих аспектах, варто зазначити, що базова валідація форм, як-от перевірка на заповнення обов'язкових полів, може бути ефективно реалізована за допомогою вбудованих атрибутів HTML5 (наприклад, required).

Таким чином, клієнтська частина забезпечує необхідний інтерфейс (рис. 3.2) для взаємодії користувачів з функціоналом системи, хоча й має потенціал для покращення в частині організації CSS та розширення JavaScript-інтерактивності.

Рисунок 3.2 – Інтерфейс користувацької панель

3.3. Інтерфейс адміністратора для перегляду зворотного зв’язку та моніторингу помилок

У рамках розробки було реалізовано окремий інтерфейс для адміністратора, що представлений шаблонами admin_dashboard.html та admin_users.html.

Шаблон admin_dashboard.html слугує головною панеллю для адміністратора, де відображається таблиця всіх завдань у системі. Для кожного завдання показано його номер, заголовок, автора (task.creator.username), відповідального адміністратора (task.assignee.username, якщо призначено), та поточний статус (Виконано або В процесі). Адміністратор має можливість перейти до перегляду деталей завдання, видалити завдання (з підтвердженням), а також призначити завдання собі, якщо воно ще не має відповідального. Шаблон admin_users.html надає адміністратору доступ до списку всіх зареєстрованих користувачів. У таблиці відображається номер користувача, ім'я, email та роль (Адмін або Користувач). Адміністратор може видалити будь-якого користувача, окрім іншого адміністратора або себе самого, через цю панель (ця логіка реалізована на бекенді у функції delete_user в app.py). Також реалізована пагінація списку користувачів.

Таким чином реалізована панель адміністратора, яку можна переглянути на рисунку 3.3

Рисунок 3.3 – Панель адміністратора

Щодо перегляду зворотного зв'язку, адміністратор може отримати доступ до коментарів та оцінок (рейтингів) конкретного завдання, перейшовши на сторінку його детального перегляду (view_task.html).

3.4. Реалізація системи логування та збору помилок

Розроблено та впроваджено систему керування завданнями, що включає комплексну систему логування, яка відіграє ключову роль у моніторингу подій та забезпеченні

стабільності функціонування. Ця система логування реалізована з використанням стандартного модуля Python logging. Вона налаштована таким чином, щоб реєструвати всі значущі операції та події в додатку, що дозволяє отримувати вичерпну інформацію про його роботу. Лог-повідомлення записуються до файлу app.log з автоматичною ротацією. максимальний розмір файлу обмежений 5 МБ, і зберігається до 3 резервних копій, що запобігає надмірному зростанню лог-файлів та забезпечує ефективне використання дискового простору. Додатково, для зручності розробки та оперативного моніторингу, всі лог-повідомлення дублюються у консоль. Мінімальний рівень логування встановлено на INFO, що дозволяє фіксувати основні операційні події та загальну інформацію про роботу системи. Для зменшення обсягу логів від веб-сервера werkzeug, його логер налаштовано на рівень ERROR.

Система фіксує різноманітні події та помилки, використовуючи відповідні функції логера (log.info(), log.warning(), log.error()), що дозволяє детально відстежувати життєвий цикл кожної операції в системі. Це охоплює широкий спектр взаємодій користувачів із системою, а також внутрішні процеси. Зокрема, фіксуються всі дії, пов'язані з аутентифікацією та авторизацією користувачів, включаючи успішний вхід до системи, невдалі спроби входу, а також вихід користувачів із облікових записів. Особлива увага приділяється безпеці. система реєструє та попереджає про будь-які несанкціоновані спроби доступу до адміністративних ресурсів, що 10 дозволяє оперативно реагувати на потенційні загрози.

10 Процес реєстрації нових користувачів також детально логується. Система фіксує створення тимчасових облікових записів, які використовуються для підтвердження електронної пошти, успішне завершення реєстрації після введення коду підтвердження, а також випадки, коли користувачі намагаються зареєструватися з уже існуючою електронною поштою. Будь-які проблеми, пов'язані з надсиланням електронних листів, наприклад, листів підтвердження, реєструються як помилки, що допомагає в діагностиці проблем зі зв'язком.

Управління завданнями є центральною функціональністю системи, і всі пов'язані з нею дії також ретельно відстежуються. Це включає створення нових завдань користувачами, видалення завдань адміністраторами, призначення завдань певним адміністраторам, а також зміни статусу завдань, наприклад, з "в процесі" на "виконано" і навпаки. Логування цих подій дозволяє адміністраторам відстежувати хід виконання завдань та ефективність роботи системи.

Система також фіксує всі дії, пов'язані з коментуванням та оцінюванням завдань. Записується кожен доданий коментар до завдання, а також оцінки, які користувачі виставляють завданням, що є важливим для зворотного зв'язку та оцінки якості підтримки.

Окремий блок логування присвячений роботі з файлами. Система реєструє завантаження файлів, які додаються до завдань або коментарів. Також відстежуються випадки видалення застарілих файлів аватарів користувачів. Важливо, що система попереджає про спроби завантаження файлів у недійсних форматах, що допомагає підтримувати цілісність даних та безпеку системи.

Зміни у профілях користувачів також детально логуються, включаючи оновлення даних профілю, таких як ім'я користувача або біографія, а також зміни пароля. Це забезпечує аудит змін облікових записів та сприяє безпеці.

На етапі запуску додатку система логує інформацію про ініціалізацію середовища, включаючи перевірку наявності папки migrations та ініціалізацію міграцій бази даних, якщо вони відсутні. Це підтверджує, що система підготовлена до 1 роботи з базою даних. Також логується інформація про створення адміністратора за замовчуванням, якщо такий обліковий запис ще не існує. Результат логування подано в Додатку А.

3.5 Безпека системи та захист даних

Безпека системи є критично важливим аспектом у розробці будь-якого програмного забезпечення, особливо для систем, що працюють з користувацькими даними та надають 5 доступ до певних функціональних можливостей. У розробленій системі керування завданнями були впроваджені та застосовані заходи для забезпечення безпеки та захисту даних.

Система забезпечує надійну аутентифікацію та авторизацію користувачів. Паролі користувачів ніколи 5 не зберігаються у відкритому вигляді, а замість цього використовуються криптографічні хеші. Для підтримки стану користувача між запитами використовується механізм сесій Flask, де кожна сесія підписується за допомогою секретного ключа, що запобігає підробці сесій. Система реалізує контроль доступу на основі ролей, де доступ до захищених маршрутів дозволено лише автентифікованим користувачам, а до адміністративних ресурсів – тільки користувачам з відповідною роллю. Спроби несанкціонованого доступу логуються як попередження.

Особлива увага приділяється захисту завантажуваних файлів. При завантаженні файлів, таких як аватари та вкладення до завдань і коментарів, система суворо перевіряє їх розширення на відповідність дозволеним типам. Це запобігає завантаженню потенційно шкідливих виконуваних файлів. Для очищення імен завантажуваних файлів від небезпечних символів використовується функція secure_filename. Завантажені файли зберігаються в спеціально відведеній папці uploads, що знижує ризики прямого виконання файлів.

Для захисту від поширених веб-атак, таких як XSS (Cross-Site Scripting),

використовується шаблонізатор Jinja2, який автоматично екранує вихідні дані. Це значно знижує ризики XSS-атак, оскільки будь-який користувацький ввід, що містить шкідливий JavaScript-код, буде відображений як звичайний текст, а не виконаний браузером.

Система також забезпечує захист даних та конфіденційність через процес реєстрації, який вимагає підтвердження електронної пошти за допомогою унікального коду. Цей код має обмежений термін дії, і тимчасові записи користувачів, які не підтвердили реєстрацію, автоматично видаляються. Всі критичні події, пов'язані з безпекою, такі як невдалі спроби входу, несанкціонований доступ, видалення користувачів та операції з файлами, ретельно фіксуються в системі логування. Це дозволяє адміністраторам відстежувати потенційні загрози та оперативно реагувати на інциденти безпеки. Завдяки реалізованим заходам, система керування завданнями забезпечує належний рівень безпеки та захисту даних користувачів.

ВИСНОВКИ

У рамках даної дипломної роботи було успішно розроблено вебсистему для ефективного моніторингу помилок та обробки зворотного зв'язку користувачів, що відповідає актуальним потребам сучасного цифрового середовища. Запропонована система є інтегрованою платформою, яка дозволяє оперативно виявляти та усувати технічні несправності, а також збирати цінні відгуки від аудиторії для подальшого вдосконалення функціоналу.

Вимоги до системи були чітко визначені, охоплюючи як функціональні аспекти (збір даних про помилки, збереження їх контексту, можливість залишати зворотний зв'язок, автоматичні сповіщення розробників, інтерфейс для аналізу помилок), так і нефункціональні (масштабованість, безпека, продуктивність, мінімальне втручання).

Архітектура вебсистеми побудована за класичним клієнт-серверним принципом, де клієнтська частина (інтерфейс користувача) взаємодіє з серверною частиною, реалізованою на базі мікрофреймворку Flask (Python). Для збереження даних використовується база даних, взаємодія з якою відбувається через ORM SQLAlchemy. Комунікація між компонентами здійснюється за допомогою HTTP-запитів.

Вибір технологічного стеку був обґрунтованим, поєднуючи перевірені та гнучкі інструменти. HTML5, CSS3, JavaScript для фронтенду; Python та Flask для бекенду; SQLAlchemy як ORM; Jinja2 для шаблонізації. Цей вибір забезпечив повний цикл розробки вебдодатку, від візуального інтерфейсу до логіки обробки даних.

Серверна частина, розроблена на Flask, забезпечує основний функціонал, включаючи обробку HTTP-запитів, взаємодію з базою даних для збереження та отримання даних

користувачів і завдань, управління сесіями користувачів та завантаження файлів. Клієнтська частина реалізована за допомогою HTML, CSS та JavaScript, надаючи користувачам інтуїтивно зрозумілий інтерфейс для взаємодії з системою.

Особливу увагу приділено інтерфейсу адміністратора, який дозволяє керувати завданнями, переглядати та видаляти користувачів. Хоча система веде детальні логи на сервері, інтегрований у графічний інтерфейс інструмент для моніторингу помилок та агрегованого перегляду зворотного зв'язку наразі не реалізований. Це є перспективним напрямком для подальшого вдосконалення системи.

Важливою складовою є комплексна система логування, реалізована на базі стандартного модуля Python logging. Вона реєструє всі значущі операції та події, такі як аутентифікація, авторизація, реєстрація, управління завданнями, коментарями, файлами та профілями, забезпечуючи вичерпну інформацію про роботу додатку та його безпеку.

Безпека системи забезпечена хешуванням паролів, управлінням сесіями за допомогою секретного ключа, контролем доступу на основі ролей та захистом завантажуваних файлів (перевірка розширень, очищення імен, зберігання у відведеній папці). Захист від XSS реалізовано через автоматичне екранування даних шаблонізатором Jinja2. Процес реєстрації вимагає підтвердження електронної пошти, а критичні події, пов'язані з безпекою, ретельно фіксуються в системі логування. Завдяки цим заходам, система керування завданнями забезпечує належний рівень безпеки та захисту даних користувачів.

Розроблена система може бути легко адаптована до різноманітних сфер застосування, від корпоративних ресурсів до навчальних чи соціальних платформ. Подальший розвиток системи передбачає вдосконалення інтерфейсу адміністратора для розширеного моніторингу та, у перспективі, впровадження автоматизованого тестування для забезпечення ще вищого рівня надійності та стабільності.

Посилання

Це джерела виділених збігів у вашому документі. Кожен збіг позначено темно-зеленим числом, яке відповідає вказаному тут джерелу. Джерела впорядковані за схожістю — чим вищий бал, тим сильніше збіг.

#	Джерело	%
1	100balov.com	0.6%
2	repository.ldufk.edu.ua	0.3%
3	elartu.tntu.edu.ua	0.3%
4	openarchive.nure.ua	0.3%
5	ir.nmu.org.ua	0.2%
6	duikt.edu.ua	0.2%
7	ts2.space	0.2%
8	korets.com.ua	0.2%
9	dspace.wunu.edu.ua	0.2%
10	ula.lantec.ua	0.1%
11	enpuir.npu.edu.ua	0.1%
12	morningdough.com	0.1%

Plag

Дякуємо, що перевірили
свій документ за допомогою
Plag!