Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи
бакалавра

на тему: «Розробка системи відстеження помилок (баг-трекінг)»
за освітньою програмою «**12 Інженерія програмного забезпечення**»
зі спеціальності: «**121 Інженерія програмного забезпечення**»

Виконав: студент групи ПЗ1911: _____ /Владислав ЄРМАКОВ
(підпис)

Керівник: _____ /Ірина ШАПОВАЛ
(підпис)

Нормоконтролер: _____ /Світлана ВОЛКОВА
(підпис)

Засвідчую, що у цій роботі немає
запозичень з праць інших авторів
без відповідних посилань
Студент: _____
(підпис)

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of a bug tracking system (bug tracking)»
Software engineering»
in the Speciality: «**121 Software engineering**»

(підпис)

(підпис)

(підпис)

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри _____КІТ
_____ Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

На кваліфікаційну роботу ____Бакалавр_____

студенту Єрмаков Владислав Віталійович_____

ема роботи: «Розробка системи відстеження помилок (баг-трекінг)»
Керівник роботи: Шаповал Ірина Вікторівна, старший викладач
затверджені наказом № 1209 ст від 07.12.2022
рок подання студентом роботи: 22.06.2023
ихідні дані до роботи: _____.
міст пояснювальної записки (перелік питань до розробки): реферат, вступ, аналіз сучасного стану предметної області, проектування, тестування та відлагодження, опис програмного продукту, висновки, бібліографічний список
ерелік демонстраційного матеріалу:
оповідь;
резентація;
емонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
	Вступ	12.09.22 – 26.10.22	
	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	27.10.22 – 04.03.23	
	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	05.03.23 – 31.04.23	
	Постановка задачі, технічне завдання	01.05.23 – 07.05.23	
	Техніко-економічні показники	08.05.23 – 15.05.23	
	Розробка інструментальних засобів дослідження	16.05.23 – 21.05.23	
	Виконання досліджень	22.05.23 – 28.05.23	
	Оформлення тез доповідей	.05.23 – 01.06.23	
	Оформлення статті у фаховий журнал	02.06.23 – 06.06.23	
	Оформлення пояснювальної записки	07.06.23 – 11.06.23	
	Розробка демонстраційних матеріалів	12.06.23 – 18.06.23	
	Подання кваліфікаційної роботи до кафедри		
	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії		

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра 187 с., 15 рис., 2 табл., 3 додатки, 11 джерел.

Метою роботи було реалізація баг-трекера, що надаватиме можливість розробникам у командах вести обік усіх проблем та пропозицій в одному місці. Програма надає самі ключові функції як менеджмент команди, проектів та тікетів. Як допоміжні функції програма надає можливість робити нотатки, коментарі к тікетам та відсилати повідомлення. За результатами роботи було створено саме такий додаток.

Пояснювальна записка складається зі вступу, трьох розділів, висновків, бібліографічного списку та чотирьох додатків.

Вступ описує суть, мету та актуальність роботи (3 сторінки).

Перший розділ: аналіз сучасного стану предметної області (18 сторінок).

Другий розділ: зовнішнє та внутрішнє проектування (19 сторінок).

Третій розділ: тестування та відлагодження (10 сторінок).

Додатки: технічне завдання, специфікація, текст програми.

Ключові слова: баг-трекер, тікет, бек-енд, фронт-енд, веб-додаток.

ЗМІСТ

ВСТУП

У сучасному цифровому світі, що швидко змінюється, програмне забезпечення асимілювалося з нашим повсякденним життям. Розробка програмного забезпечення стає дедалі складнішою і тепер включає в себе все – від мобільних додатків до онлайн-платформ. Це вимагає ефективних процедур забезпечення якості. Зі зростанням складності програмного забезпечення зростає ризик виникнення помилок або недоліків, які можуть заважати користувачам і зашкодити загальній роботі програми. Щоб вирішити цю проблему, програмне забезпечення для відстеження помилок стало важливим інструментом для команд розробників програмного забезпечення для ефективного управління та виправлення помилок у програмному забезпеченні.

Метою цієї тези є дослідження світу веб-додатків для відстеження помилок, їхнього значення у розробці програмного забезпечення та їхнього внеску в покращення процесу забезпечення якості. У цій роботі ретельно розглядаються цілі, переваги, підходи до реалізації та вплив веб-інструментів для відстеження помилок на життєвий цикл розробки програмного забезпечення.

Команди розробників програмного забезпечення можуть ефективно реєструвати, відстежувати та керувати повідомленнями про помилки або проблеми з програмним забезпеченням за допомогою веб-програми відстеження помилок. Ці веб-рішення спрощують весь процес, пропонуючи системний підхід до управління помилками, що дозволяє командам швидко та ефективно виявляти, визначати пріоритети, призначати та виправляти дефекти. Веб-інструмент для відстеження помилок також полегшує членам команди спілкування та спільну роботу, гарантуючи, що кожен буде поінформований про статус помилок, оновлення та виправлення.

У цьому дослідженні будуть виділені ключові елементи, які роблять інструменти для відстеження помилок такими корисними в ініціативах з розробки програмного забезпечення. Такі функції, як відстеження проблем, адаптивні робочі процеси та сповіщення в реальному часі, дозволяють

командам розробників дотримуватися відповідальності та прозорості впродовж усього процесу усунення помилок.

У дипломі також обговорюються переваги інтеграції веб-інструменту для відстеження помилок у компанії, що займається розробкою програмного забезпечення. Краще управління помилками призводить до кращої якості програмного забезпечення, скорочення часу розробки та задоволеності клієнтів. Зацікавлені сторони можуть краще контролювати розробку рішень проблем, ефективно розподіляти ресурси та приймати мудрі рішення, якщо вони мають чітке уявлення про процес усунення помилок. Зрештою, веб-інструмент відстеження помилок допомагає проекту з розробки програмного забезпечення бути успішним в цілому, гарантуючи запуск стабільних, надійних і зручних для користувача програм.

Цей диплом також розглядає реальні приклади ефективних рішень, щоб краще оцінити практичні переваги відстеження помилок в онлайн-додатках. Ці приклади ілюструють, як компанії з різних секторів використовують відстеження помилок у веб-додатках, щоб прискорити розробку програмного забезпечення, позбутися серйозних дефектів і підвищити загальний рівень своїх продуктів.

Основна мета цієї тези – висвітлити значення відстеження помилок у веб-додатках для розробки програмного забезпечення. Ці веб-інструменти полегшують процес вирішення проблем, покращують комунікацію та співпрацю і, зрештою, призводять до розробки високоякісних програмних додатків завдяки застосуванню методичного та ефективного підходу до управління помилками. Це дослідження має на меті надати командам розробників програмного забезпечення знання та розуміння, необхідні для використання веб-інструментів для відстеження помилок у реальному світі, шляхом вивчення їхніх переваг, особливостей та корисних застосувань.

Веб-інструменти для відстеження помилок забезпечують централізовану платформу для ефективного управління проблемами, їх відстеження та вирішення. Ці веб-додатки прискорюють процедуру усунення помилок,

сприяють комунікації та командній роботі між членами команди, а також покращують загальну якість програмного забезпечення, пропонуючи такі функції, як відстеження проблем, настроювані робочі процеси, сповіщення в реальному часі та ретельні можливості звітування.

1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Огляд програмних аналогів

На ринку існує низка програмних аналогів веб-додатків для відстеження помилок, кожен з яких має свої особливості та функції, що відповідають вимогам команд розробників програмного забезпечення. У цьому розділі будуть описані та оглянуті різні відомі онлайн-додатки для відстеження помилок, які набули популярності серед експертів у цій галузі.

1

Jira, один з найпопулярніших онлайн-додатків для відстеження помилок, .
був створений компанією Atlassian. Вона надає повний спектр інструментів для управління проектами, гнучкої розробки програмного забезпечення та відстеження проблем. Організації різного розміру використовують Jira завдяки її адаптивним робочим процесам, складним функціям звітності та легкому зв'язку з іншими інструментами розробки.

Переваги Jira:

Надійне відстеження проблем: Jira пропонує надійне, масштабоване рішення для відстеження проблем. Протягом усього життєвого циклу розробки воно дозволяє командам генерувати, визначати пріоритети, розподіляти і відстежувати проблеми, дефекти і завдання. Завдяки гнучкості в налаштуванні процесів і користувацьких полів воно підходить для різних підходів до управління проектами.

Розширені налаштування: Jira має широкий спектр опцій кастомізації, що дозволяє командам адаптувати додаток до власних вимог. Користувачі можуть налаштувати Jira так, щоб вона відповідала вимогам і процедурам їхніх проектів, створюючи нові дані, робочі процеси, типи завдань і відображення.

Jira здатна інтегруватися з широким спектром додаткових інструментів для співпраці та розробки, включаючи чат-програми, такі

як Slack, інструменти безперервної інтеграції, такі як Jenkins, та системи контролю версій, такі як Git. Завдяки цій інтеграційній екосистемі команди можуть оптимізувати свою роботу, синхронізувати дані та покращити співпрацю між різними технологіями.

Підтримка Agile та Scrum: Jira пропонує першокласну підтримку Agile та Scrum підходів. Завдяки спеціальним дошкам, бэклогам та можливостям планування спринтів, Agile-команди отримують можливість ефективно організовувати та контролювати свою роботу. Agile-дошки Jira спрощують візуалізацію статусу виконуваної роботи, що спрощує управління гнучкими проектами.

Надійна звітність та аналітика: Jira має потужні функції звітності та аналітики. Команди можуть створювати різноманітні звіти, включаючи статистику випусків, діаграми швидкості та діаграми вигорання. Ці аналізи допомагають командам приймати рішення на основі даних, пропонуючи глибоку інформацію про розвиток проекту, продуктивність команди та розподіл ресурсів.

Недоліки Jira

Широкий набір функцій та можливостей налаштування Jira може відлякати користувачів-початківців. Складність і крива навчання: Jira має круту криву навчання. Jira може потребувати тривалого навчання та адміністративної роботи, щоб налаштувати і налаштувати її відповідно до вимог конкретного проекту.

Інтерфейс користувача (UI) Jira може бути щільним і складним, особливо для менш досвідчених користувачів. Може знадобитися деякий час, щоб звикнути до численних панелей, опцій та меню. В останніх оновленнях Atlassian намагався зробити інтерфейс кращим.

Вартість Jira може бути проблемою для невеликих команд або нових компаній, оскільки вона може бути дорожчою, ніж альтернативні платформи для управління проектами. Крім того, деякі складні функції та доповнення можуть мати додаткову плату, що може збільшити загальну вартість.

Продуктивність та масштабованість: Користувачі іноді скаржаться на продуктивність Jira, особливо при роботі з великими, складними проектами або коли програмне забезпечення розміщене на самокерованих серверах. Для забезпечення ідеальної продуктивності та масштабованості може знадобитися розширити ресурси та внести корективи.

Залежність від сторонніх плагінів: Jira надає велику кількість власних функцій, однак для деяких специфічних функцій можуть знадобитися сторонні плагіни або доповнення. Використання великої кількості сторонніх плагінів може ускладнити роботу, спричинити проблеми сумісності та створити загрози безпеці.

Вже багато років використовується онлайн-додаток для відстеження помилок з відкритим вихідним кодом під назвою Bugzilla. Він пропонує простий і зрозумілий інтерфейс для звітування, відстеження та виправлення помилок. Bugzilla є надійним вибором для команд розробників програмного забезпечення, які шукають економічно ефективне рішення, оскільки він включає такі функції, як сповіщення на електронну пошту, поля, що налаштовуються, та потужний пошук.

Переваги Bugzilla:

Відкритий вихідний код та економічність: Bugzilla – це програма з відкритим вихідним кодом, яка доступна для використання та модифікації безкоштовно. Це робить її доступним варіантом для компаній з обмеженим бюджетом або приватних осіб, яким подобається програмне забезпечення з відкритим кодом.

Комплексне відстеження помилок: Bugzilla пропонує надійне рішення для відстеження помилок. Команди можуть використовувати його для відстеження, визначення пріоритетів та розподілу помилок і проблем, що виникають під час процесу розробки. Він надає поля, що

налаштовуються, гнучкі процеси та моніторинг статусу, щоб задовольнити різні потреби проекту.

Bugzilla має широкий спектр налаштувань, що дозволяє командам адаптувати інструмент до власних потреб. Користувачі можуть налаштувати Bugzilla відповідно до своїх конкретних методологій і термінології розробки, створюючи нові поля, статуси проблем і робочі процеси.Bugzilla має надійну систему запитів, яка дозволяє користувачам шукати конкретні проблеми на основі різноманітних параметрів. Крім того, вона має функції звітності, які дозволяють командам створювати метрики та звіти про дані про помилки, тенденції та продуктивність проекту.Активна спільнота та підтримка: Спільноти користувачів і розробників Bugzilla процвітають і є активними. Це свідчить про те, що існує безліч інформації, форумів та інструментів, доступних для вирішення проблем, модифікації та покращення роботи Bugzilla.

Недоліки Bugzilla:

Встановлення та налаштування Bugzilla може бути складним завданням, особливо для тих, хто не має достатніх знань з адміністрування серверів та встановлення програмного забезпечення. Для того, щоб забезпечити безперебійне встановлення та інтеграцію з іншими інструментами, можуть знадобитися технічні знання та зусилля.

Складність інтерфейсу користувача (UI): У порівнянні з деякими іншими системами відстеження помилок, Bugzilla відома тим, що має менш зручний і візуально привабливий користувальницький інтерфейс. Користувачам може знадобитися деякий час, щоб звикнути до інтерфейсу і навчитися користуватися всіма його меню і налаштуваннями.Обмежена підтримка Agile: Bugzilla не була створена з урахуванням практик Agile. Хоча його можна адаптувати до Agile-методів, йому може не вистачати деяких специфічних функцій і дощок, які частіше зустрічаються в програмах для відстеження помилок, орієнтованих на Agile.

Відсутність вбудованої інтеграції: У порівнянні з іншими платформами для співпраці та розробки, Bugzilla не пропонує багато вбудованих інтеграцій. Для інтеграції Bugzilla з іншими системами може знадобитися використання сторонніх плагінів або індивідуальне програмування.

Як програма з відкритим вихідним кодом, Bugzilla значною мірою покладається на підтримку та допомогу спільноти. Існує активна спільнота, однак у порівнянні з комерційними програмами відстеження помилок зі спеціальними командами підтримки, обсяг допомоги та доступність оновлень може відрізнятися.

1
. – це передовий онлайн-додаток для відстеження помилок, створений JetBrains,
1
який надає ряд можливостей для гнучкого управління проектами та відстеження проблем. Завдяки надзвичайно адаптивному та зручному дизайну, команди можуть налаштовувати інструмент відповідно до власних процесів. YouTrack забезпечує автоматизацію завдяки налаштованим робочим процесам і сповіщенням на основі правил, а також взаємодію з відомими системами контролю версій.

Переваги YouTrack

Гнучкість і кастомізація: YouTrack надає широкий спектр можливостей кастомізації, що дозволяє командам адаптувати продукт до власних робочих процесів і процедур. Гнучкі дошки, кастомні поля, робочі процеси, типи проблем та інші функції можуть бути визначені користувачами, що створює дуже гнучке середовище для управління проектами.

YouTrack добре підходить для гнучких підходів до управління проектами, таких як Scrum та Kanban. Команди можуть ефективно планувати, відстежувати та керувати своїми гнучкими проектами за допомогою спеціалізованих гнучких дощок, беклогів, спринтів та діаграм, що надаються.

Потужні можливості пошуку та запитів: YouTrack містить потужну мову пошуку та запитів, яка дозволяє користувачам створювати складні та індивідуалізовані пошукові запити. Команди можуть швидко знаходити і сортувати важливі питання за допомогою численних фільтрів цієї функції, що підвищує продуктивність і економить час.

YouTrack забезпечує зв'язок з різноманітними інструментами для співпраці та розробки, включаючи комунікаційні платформи, інструменти безперервної інтеграції та системи контролю версій. Інтеграція YouTrack з іншими інструментами сприяє співпраці в екосистемі розробки та забезпечує безперебійний потік інформації.

Кілька варіантів звітності та аналітики: YouTrack пропонує кілька варіантів звітності та аналітики. Користувачі можуть створювати різноманітні звіти, включаючи діаграми, діаграми швидкості та дані про випуски, щоб зрозуміти, наскільки добре працюють їхні команди та де вони можуть покращити роботу.

Недоліки YouTrack

YouTrack має криву навчання, особливо для тих, хто тільки починає працювати з ним. Може знадобитися певний час і зусилля, щоб повністю зрозуміти і успішно використовувати всі його функції та можливості налаштування.

Складність користувацького інтерфейсу: користувацький інтерфейс YouTrack може бути заплутаним, особливо для менш досвідчених користувачів. Може знадобитися деякий час, щоб звикнути до навігації по численних сторінках і опціях, а інтерфейс може здатися менш зручним, ніж у деяких інших додатках для управління проектами.

Споживання ресурсів: Для найкращої роботи YouTrack може знадобитися велика кількість серверних ресурсів, залежно від обсягу та складності проекту. Щоб забезпечити приємний досвід, організації з невеликими командами або обмеженою інфраструктурою повинні ретельно оцінити вимоги до апаратного забезпечення та мережі.

YouTrack пропонує безкоштовну версію, а також преміум-версію з більшою кількістю функцій і підтримки. Комерційна версія може включати плату, яку компанії з обмеженим бюджетом, можливо, захочуть взяти до уваги, навіть якщо безкоштовна версія підходить для невеликих команд.

Обмежена спільнота користувачів та документація: Незважаючи на зростаючу базу користувачів, документація та допоміжні матеріали YouTrack можуть бути не такими вичерпними, як у деяких інших додатків для управління проектами. Незважаючи на те, що допомога пропонується, споживачі можуть мати проблеми з отриманням ретельної та своєчасної підтримки.

1
. Онлайн-додаток з відкритим вихідним кодом під назвою Redmine пропонує інструменти для відстеження помилок, управління проектами та командної роботи. Він надає широкий спектр функцій, таких як вікі-сторінки, діаграми Ганта, відстеження проблем та моніторинг часу. Завдяки своїй гнучкості та розширюваності за допомогою плагінів, Redmine підходить для кастомізації для виконання унікальних вимог проекту.

Переваги Redmine

– це програма з відкритим вихідним кодом, яка вільно використовується і може бути адаптована до вимог певних проектів. Це робить його доступним варіантом для компаній з обмеженим бюджетом або окремих осіб, які люблять програмне забезпечення з відкритим кодом.

Redmine має великий ступінь кастомізації, що дозволяє командам модифікувати інструмент відповідно до власних процесів і цілей. Щоб зробити Redmine сумісним з їхніми процедурами управління проектами, користувачі можуть створювати власні поля, робочі процеси, типи проблем і структури проектів.

Управління проектами та відстеження проблем: Redmine має ретельні функції відстеження проблем, які дозволяють командам генерувати, розподіляти і контролювати проблеми, дефекти і завдання. Крім того, він надає інструменти управління проектами, такі як діаграми Ганта, календарі та управління документами, що дають повну картину того, як просувається проект.

Redmine має потужну систему плагінів, яка дозволяє користувачам розширювати його можливості та поєднувати його з іншими програмами. Функціональність Redmine може бути покращена за допомогою різноманітних плагінів, які надають нові можливості та інтеграції з іншими

Підтримка спільноти: Redmine має активну і корисну базу користувачів, яка пропонує настанови, документацію та плагіни. – це платформа, керована спільнотою, тому користувачі можуть отримувати доступ до інформації, вирішувати проблеми та отримувати поради від більш досвідчених користувачів.

Користувацький інтерфейс (UI) Redmine часто критикують за те, що він застарілий і менш зручний, ніж деякі більш сучасні рішення для управління проектами. Користувачам може знадобитися деякий час, щоб звикнути до інтерфейсу, і він може не пропонувати той самий рівень користувацького досвіду, що й більш естетичні продукти.

Крива навчання Redmine може бути більш складною для менш досвідчених користувачів або тих, хто не звик до рішень для управління проектами. Може знадобитися деякий час і навчання, щоб розібратися в численних варіантах налаштувань, параметрах і процедурах.

Недостатня підтримка Agile: У порівнянні з інструментами, спеціально розробленими для управління гнучкими проектами, можна сказати, що підтримка Redmine для гнучких підходів, таких як Scrum або Kanban, є недостатньою. Хоча вона може бути дещо скоригована, деякі Agile-орієнтовані функції та дошки можуть бути відсутніми.

Продуктивність і масштабованість: Redmine може зіткнутися з проблемами продуктивності та масштабованості у великих проектах з високим рівнем використання. Для забезпечення оптимальної продуктивності може знадобитися ретельна конфігурація сервера та регулярне обслуговування.

Redmine має доступну документацію, однак вона може бути не такою повною або актуальною, як деякі інші комерційні системи управління проектами. Для вирішення певних вимог або проблем користувачам може знадобитися покладатися на ресурси спільноти або сторонню документацію.

1.2 Огляд літератури

1.2.1 Сучасна архітектура веб-додатків

Архітектура веб-додатків

Простіше кажучи, архітектура веб-додатків – це «скелет» або план, який показує, як користувацькі інтерфейси, бази даних, системи проміжного програмного забезпечення та компоненти додатків взаємодіють один з одним. Численні програми можуть працювати в тандемі завдяки такому типу зв'язку (рис. 1.1).

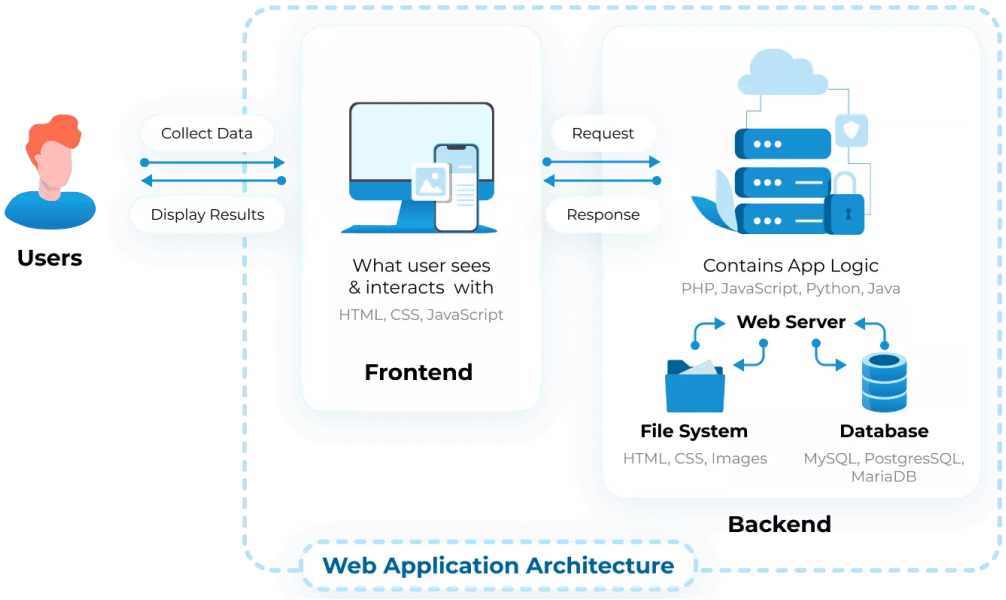

Рисунок 1.1 – Архітектура веб-додатків

Коли користувач отримує доступ до веб-сторінки, сервер відповідає на запит користувача, надсилаючи певні дані браузеру. Точніше кажучи, веб-клієнт (або агент користувача) може запитувати у веб-сервера веб-ресурси або більш поширені онлайн-документи (наприклад, HTML, JSON, PDF і так далі). Після цих простих операцій бажана інформація матеріалізується. Після цього користувач і веб-сайт починають взаємодіяти [1].

Основи архітектури веб-додатків можна пояснити за допомогою схеми процесу "користувач-сервер" (рис 1.2):

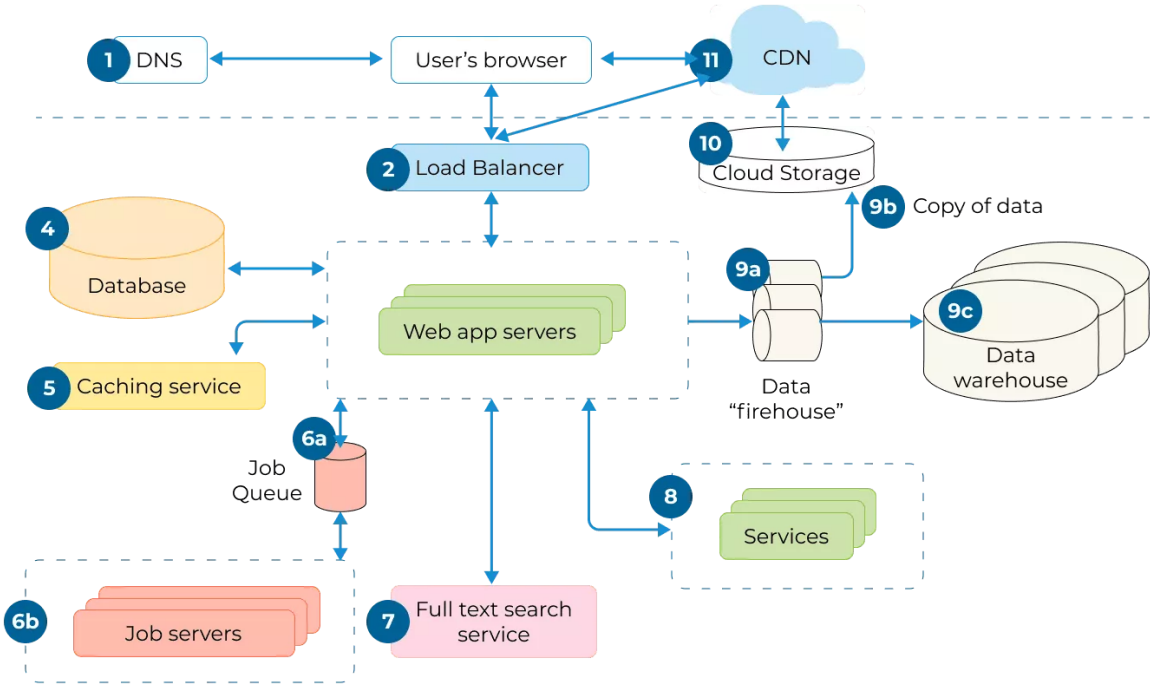

Рисунок 1.2 – Схеми процесу «користувач-сервер»

1.2.1.2 Компоненти архітектури веб-додатків

Більш детально розглянемо основні компоненти архітектури веб-додатків:

Користувач може надіслати запит на певний сервер через DNS, або систему доменних імен, яка є фундаментальним механізмом, що допомагає у пошуку доменного імені та IP-адреси. DNS можно порівняти з телефонною книгою для веб-сайтів в Інтернеті.

Балансувальник навантаження

Основна функція балансувальника навантаження – горизонтальна масштабованість. Балансувальник навантаження відповідає користувачеві, перенаправляючи вхідні запити на один з декількох серверів. Сервери веб-додатків зазвичай існують у вигляді декількох версій, які є дзеркальними відображеннями один одного. В результаті всі сервери обробляють запити однаково, а балансувальник навантаження розподіляє роботу між ними, щоб запобігти перевантаженню.

Сервери для веб-додатків:

У відповідь на запит користувача цей компонент обробляє і повертає документи (наприклад, JSON, XMK тощо) браузеру. Зазвичай він звертається до внутрішніх інфраструктур, таких як бази даних, кеш-сервери, робочі черги та інші, щоб виконати цю роботу. Крім того, щонайменше два сервери, які пов'язані з балансувальником навантаження, можуть обробляти запити користувачів.

Бази даних:

Назва цього компонента веб-додатку говорить сама за себе. База даних надає інструменти для групування інформації, додавання елементів, їх пошуку, редагування або видалення, а також виконання обчислень. Здебільшого сервери завдань і сервери веб-додатків мають прямий зв'язок.

Кеш сервіс:

Зберігання даних забезпечується службою кешування, що дозволяє зберігати дані та здійснювати їх пошук. Результати цієї дії зберігаються в кеші кожного разу, коли користувач запитує інформацію з сервера. Таким чином, відповіді на наступні запити надаються швидше. Простіше кажучи, кешування дозволяє швидко виконувати обчислення, використовуючи попередній результат як еталон. Як наслідок, кешування найкраще працює, коли: обчислення повільні, обчислення, ймовірно, повторюються і коли відповіді на певний запит однакові.

Черга завдань (необов'язково):

Сама черга завдань і сервери складають чергу завдань. Ці сервери обробляють завдання, що стоять у черзі. Більшість веб-серверів повинні керувати кількома завданнями, які не мають найвищого пріоритету. В результаті, коли робота повинна бути завершена, вона додається до черги завдань і виконується відповідно до розкладу.

Сервіс повнотекстового пошуку (опціонально):

Численні веб-програми включають можливість текстового пошуку, або так званого запиту, і згодом надають користувачеві найбільш релевантні результати. Повнотекстовий пошук – так називають цю технологію. Він шукає необхідну інформацію за ключовими словами серед величезної колекції документів.

Сервіси:

Сервіси створюються як окремі додатки, коли веб-додаток досягає певного рівня. Веб-додаток та інші сервіси взаємодіють з ними, навіть якщо вони не особливо помітні серед інших компонентів веб-додатку.

Сховище даних:

Майже всі сучасні додатки передбачають роботу з даними, включаючи їх збір, зберігання та аналіз. Для цих процесів необхідні три кроки:

Дані передаються до "пожежного шлангу" даних, який пропонує потоковий інтерфейс для обробки та поглинання даних.

Дані доставляються до хмарного сховища, включаючи необроблені, оброблені та додаткові дані.

Крім того, оброблені та додаткові дані надсилаються до сховища даних.

Це особливий тип онлайн-системи обміну та зберігання даних, що використовує Інтернет. У сховищі даних можна зберігати відео, фотографії та інші файли різних типів.

CDN або система доставки контенту займається відправкою HTML, CSS, JavaScript і графічних файлів. Вона розподіляє вміст кінцевого сервера по всьому світу, щоб користувачі могли завантажувати його з різних джерел.

1.2.1.3 Типи архітектур веб-додатків

Сучасна архітектура складається з Frontend (Фронтенд) та Backend (Бекенд). У цьому розділі поговоримо про їх різновидності архітектури

SPA, або односторінкові додатки, призначені для спрощення програмування. Користувачеві не потрібно завантажувати абсолютно нові сторінки щоразу, коли він хоче виконати якусь дію на веб-сайті. Замість цього користувачі можуть взаємодіяти з ним і отримувати свіжий матеріал для поточної сторінки.

Така архітектура у веб-дизайні вимагає найнеобхідніших матеріалів і даних. Для того, щоб покращити інтуїтивно зрозумілий та цікавий користувацький досвід, SPA усувають переривання користувацької активності. До речі, найпопулярнішою мовою програмування в такому дизайні є JavaScript.

Веб-сторінка на клієнтській стороні фреймворку Javascript рендериться за допомогою SSR в HTML і CSS на сервері. Цей інструмент дозволяє швидко надавати найважливіші елементи, прискорюючи здатність браузера генерувати сторінку і скорочуючи час рендерингу для користувача.

Щоразу, коли робиться запит до програми SSR, сервер збирає всі дані і повертає свіжий HTML-відображення. Крім того, коли браузер має CSS, він може малювати користувацький інтерфейс, не чекаючи на завантаження JavaScript. Це прискорює завантаження сторінки.

Microservices (мікросервіси):

Одним із різновидів SOA (service-oriented architecture / сервіс-орієнтованої архітектури) є мікросервіси. Мікросервіси часто мають справу з крихітними, легкими сервісами, які виконують певну операцію. Використовувати такий тип веб-архітектури ефективно і раціонально. З його допомогою розробники можуть заощадити значну кількість часу.

Компоненти мікросервісів не пов'язані між собою, оскільки не всі вони створені за допомогою однієї мови програмування. Це означає, що програмісти можуть вільно обирати технологію на свій розсуд, що прискорює і спрощує розробку архітектури мікросервісів.

Serverless Architecture (Безсерверна архітектура):

З назви можна зробити висновок, що безсерверна архітектура – це архітектура без серверів. Але це не так. Насправді це означає, що управління та налаштування серверів, на яких працює програмне забезпечення, більше не потрібне. Але всю інфраструктуру підтримують сторонні постачальники послуг. Управління інфраструктурою та аутсорсинг серверів – це внесок третьої сторони.

Для аутсорсингу серверів та адміністрування інфраструктури в такому дизайні веб-додатків розробник додатків звертається до стороннього постачальника послуг хмарної інфраструктури.

Безсерверна архітектура – це знахідка для користувача, якщо він не хочете займатися обслуговуванням і підтримкою серверів та обладнання. Ця стратегія є чудовою, оскільки дозволяє запускати логіку коду, зберігаючи інфраструктуру на місці.

1.3 Основні поняття

Bug:

Помилка, недолік або дефект програмного забезпечення називається вадою, коли воно поводиться несподівано або неправильно. Помилки кодування, логічні помилки, проблеми з конфігурацією та сумісністю – все це може призвести до багів. Баги можуть негативно впливати на зручність використання, продуктивність та досвід користувача програмного забезпечення.

Bug Tracking:

Практика реєстрації, управління та вирішення дефектів або проблем, виявлених під час розробки та тестування програмного забезпечення, відома як відстеження помилок. Воно передбачає методичну реєстрацію проблем,

розподіл їх між відповідними членами команди, моніторинг їхнього прогресу та забезпечення їхнього виправлення у встановлені терміни. У командах розробників програмного забезпечення відстеження помилок сприяє ефективній комунікації, співпраці та відповідальності.

Bug Report:

Документ або запис у блозі, який містить детальну інформацію про певну ваду або проблему, називається звітом про ваду. Зазвичай він містить опис проблеми, інструкції для її відтворення, інформацію про середовище, скріншоти або повідомлення про помилки, а також будь-яку іншу необхідну інформацію. Звіти про помилки використовуються для повідомлення та фіксації дефектів з метою прискорення їх виправлення.

Issue Tracking System:

Програмний інструмент або онлайн-додаток, відомий як система відстеження проблем, також відомий як система відстеження помилок або система відстеження дефектів, полегшує роботу з проблемами і питаннями протягом усього життєвого циклу розробки програмного забезпечення. Він пропонує єдине місце для документування, моніторингу та виправлення дефектів, що дозволяє членам команди працювати разом, делегувати обов'язки та стежити за розробкою.

Workflow:

Порядок етапів або дій, пов'язаних з відстеженням і виправленням помилок, представлений робочим процесом. Він окреслює кроки, які проходить проблема з моменту її першого виявлення до остаточного виправлення. Генерація, сортування, призначення, розслідування, вирішення та перевірка помилок є загальними етапами робочого процесу. Залежно від унікальних вимог і процедур команд розробників програмного забезпечення, робочі процеси можуть бути модифіковані.

Reporting and Analytics:

Практика класифікації дефектів відповідно до їхньої серйозності, впливу на програмну програму та загальних пріоритетів проекту відома як

пріоритизація помилок. Пріоритизація допомагає ефективно використовувати обмежені ресурси, гарантуючи, що термінові проблеми виправляються негайно, а повільніші дефекти виправляються, коли дозволить час.

Ticket:

Тікет – це спеціальний запис або елемент, який вказує на дефект або проблему з програмним забезпеченням у баг-трекерах. Він діє як консолідоване і організоване зображення проблеми, збираючи всі відповідні дані про помилку.

Коли користувач, тестувальник або розробник знаходить проблему, вони часто створюють новий тікет у баг-трекері, щоб зафіксувати її. Тікет містить ряд полів і властивостей, які надають вичерпну інформацію про проблему, що полегшує її обслуговування і вирішення.

1.4 Постановка проблеми

Незважаючи на те, що програмні додатки стали більш складними і важливими для багатьох компаній, виникнення помилок і недоліків продовжує залишатися головною перешкодою в розробці програмного забезпечення. Відсутність надійної системи відстеження помилок може призвести до низки проблем, таких як зниження якості програмного забезпечення, неефективне усунення помилок, затягування термінів виконання проекту та незадоволеність кінцевих користувачів. Для того, щоб покращити загальні зусилля із забезпечення якості в проектах з розробки програмного забезпечення та оптимізувати процес управління проблемами, потужний веб-додаток для відстеження помилок є вкрай необхідним.

Зазвичай головними проблемами баг-трекерів є відсутність або неналежна імплементація речей, які мають вирішувати ці труднощи:

Відсутність централізованого управління вадами: Команди розробників програмного забезпечення часто мають справу з фрагментарними звітами про помилки, що призводить до неефективної комунікації, дублювання записів про помилки, а також до проблем з відстеженням та вирішенням проблем. Відсутність централізованої структури ускладнює координацію та ефективну

спільну роботу членів команди, що збільшує час, необхідний для виправлення помилок, і знижує якість програми.

Неефективний розподіл багів та визначення пріоритетів: Ручні методи призначення та визначення пріоритетів помилок іноді є довільними та трудомісткими. Без методичного підходу серйозні проблеми можуть залишатися непоміченими або вирішуватися надто повільно, що впливає на загальну якість програми та користувацький досвід. Проблема ще більше ускладнюється відсутністю автоматизованих методів розподілу багів між відповідними членами команди, що призводить до плутанини та неефективності.

Недостатні можливості звітування та аналізу: Надійні навички звітування та аналізу мають важливе значення для ефективного відстеження помилок. Однак відсутність такої функціональності в системах відстеження помилок, що використовуються зараз, ускладнює збір глибокої інформації про тенденції виникнення проблем, типові патерни та основні причини. Відсутність аналітичних навичок обмежує здатність приймати рішення на основі даних, розумно розподіляти ресурси та визначати можливості для покращення процесів.

Для вирішення цих проблем потрібен комплексний і зручний веб-додаток для відстеження помилок, який пропонує централізоване управління проблемами, автоматичне визначення пріоритетів і призначення завдань, потужні можливості звітності та аналізу, а також безшовний інтерфейс з іншими інструментами розробки. Таке рішення покращить весь процес забезпечення якості, пришвидшить процес виправлення помилок, підвищить якість програмного забезпечення та гарантуватиме задоволеність користувачів. Вирішивши ці проблеми, команди розробників програмного забезпечення зможуть покращити свої процедури управління помилками, оптимізувати свої робочі процеси та створювати високоякісні програмні продукти швидко і ефективно.

1.5 Постановка завдання

Розробка, створення та впровадження веб-додатку для відстеження помилок, який вирішує труднощі, з якими стикаються команди розробників програмного забезпечення при ефективному управлінні та вирішенні проблем з програмним забезпеченням. Основною метою є розробка повної та зручної платформи, яка впорядковує всі зусилля по забезпеченню якості в проектах розробки програмного забезпечення, покращує процедуру відстеження помилок та підвищує якість програми. Сама програма повинна виконувати наступні функції та обов'язки:

Розробка функцій:

Реалізувати основну функціональність веб-додатку для відстеження помилок, таку як введення помилки, призначення, відстеження, видалення, оновлення статусу, визначення пріоритетів, функціонал пошуку та сортування. Додавання складних функцій, такі як сповіщення, звітність, інструменти аналітики та коментування. Також, функціонал менеджменту проектів та персоналу організації.

Дизайн інтерфейсу користувача та зручність використання:

Створіть зручний та інтуїтивно зрозумілий користувацький інтерфейс для веб-додатку для відстеження помилок, проектів та команди розробників. Переконатись, що користувальницький інтерфейс забезпечує просте введення даних про баги, навігацію та пошук. Для покращення юзабіліті використати найкращі практики в дизайні користувацького досвіду.

Висновки до пункту 1

У першому розділі розглянуто тему веб-додатків для відстеження помилок та їхню важливість у проектах з розробки програмного забезпечення. Розглянуто труднощі, з якими стикаються команди розробників програмного забезпечення при спробі управління та виправлення програмних проблем, а також вимоги до всеосяжного веб-додатку для відстеження помилок.

Були ознайомлені з аналогами та їх головними особливостями, плюсами та недоліками. Завдяки постановці завдання вивчили вимоги до програми, та тепер є орієнтація на те, які функції має виконувати програма

Численні аспекти веб-додатків для відстеження помилок були прояснені завдяки вивченій літературі, включаючи порівняння різних систем відстеження помилок, методів локалізації помилок, процедур сортування помилок і методів визначення пріоритетності звітів про помилки. Ці дослідження пролили важливе світло на переваги, недоліки та використання онлайн-додатків для відстеження помилок.

Крім того, були визначені основні проблеми, які вимагають створення успішного веб-додатку для відстеження помилок. Відсутність централізованого управління помилками, неефективні процедури визначення пріоритетів і призначення завдань, погані інструменти звітності та аналізу, а також відсутність інтерфейсу з іншими інструментами розробки – ось деякі з цих труднощів.

Поточна мета полягає у розробці, створенні та впровадженні веб-додатку для відстеження помилок, який враховує ці проблеми та покращує процедуру відстеження помилок. Ціль це підвищити якість програмного забезпечення, пришвидшити роботу з помилками та максимізувати загальні зусилля із забезпечення якості в проектах з розробки програмного забезпечення шляхом створення комплексної та зручної платформи.

Перша частина загалом заклала основу для дослідження відстеження помилок в онлайн-додатках і підкреслила важливість подолання труднощів в управлінні помилками.

2 ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

2.1 Зовнішнє проектування

2.1.1 Функціональне призначення

Функціональна мета веб-застосунку для відстеження помилок – запропонувати ретельний та ефективний метод моніторингу та управління дефектами та проблемами програмного забезпечення протягом усього життєвого циклу розробки програмного забезпечення. Програмне забезпечення діє як основний центр для збору, категоризації, визначення пріоритетів та виправлення проблем, виявлених користувачами, тестувальниками або членами команди.

Інша мета програми – запропонувати стабільну та зручну платформу, яка спрощує моніторинг та усунення помилок. Посилення співпраці, спрощення комунікації та забезпечення ефективного управління помилками – все це сприяє підвищенню якості програмного забезпечення та задоволеності клієнтів.

2.1.2 Експлуатаційне призначення

Операційна мета веб-застосунку – запропонувати корисне і практичне рішення, яке допоможе команді розробників щоденно керувати дефектами і проблемами програмного забезпечення та виправляти їх. Основна увага приділяється тому, як успішно та ефективно використовувати додаток з операційної точки зору. Нижче наведені деякі з основних операційних цілей додатка:

– Відстеження та управління помилками: Протягом усього життєвого циклу розробки програмного забезпечення додаток діє як центральне сховище для відстеження та управління помилками. Щоб переконатися, що жодна проблема не залишиться поза увагою, розробники мають відстежувати, впорядковувати та визначати пріоритетність помилок, про які повідомляються інші їх колеги через повідомленн;

– Ефективне виправлення помилок: Пропонуючи методичний підхід і процедуру для роботи з дефектами, програмне забезпечення сприяє

ефективному усуненню помилок. Це дозволяє членам команди призначати дефекти певним людям, встановлювати пріоритети та відстежувати їхній прогрес, гарантуючи швидке виправлення помилок;

– Співпраця та комунікація: Додаток заохочує членів команди, які беруть участь у виправленні помилок, ефективно працювати разом і спілкуватися один з одним. Він пропонує такі функції, як сповіщення та коментарі до звітів про помилки, що дозволяє легко співпрацювати та обмінюватися інформацією для ефективного вирішення проблем;

– Оптимізація процесів: Визначаючи чіткі етапи та переходи для вирішення проблем, програмне забезпечення спрощує процеси. Від початкового повідомлення до вирішення і закриття проблеми, це гарантує, що дефекти будуть вирішуватися відповідно до визначеного підходу, зменшуючи непорозуміння і гарантуючи єдині практики для всієї команди;

– Ефективне виправлення помилок: Пропонуючи методичний підхід і процедуру для роботи з дефектами, програмне забезпечення сприяє ефективному усуненню помилок. Це дозволяє членам команди призначати дефекти певним людям, встановлювати пріоритети та відстежувати їхній прогрес, гарантуючи швидке виправлення помилок;

– Співпраця та комунікація: Додаток заохочує членів команди, які беруть участь у виправленні помилок, ефективно працювати разом і спілкуватися один з одним. Він пропонує такі функції, як бесіди, сповіщення та коментарі до звітів про помилки, що дозволяє легко співпрацювати та обмінюватися інформацією для ефективного вирішення проблем;

– Зручний інтерфейс: Програмне забезпечення прагне запропонувати зручний, інтуїтивно зрозумілий і простий у використанні інтерфейс. Крива навчання скорочується, а ефективність роботи підвищується, оскільки це гарантує, що члени команди можуть негайно зрозуміти і використовувати додаток;

– Фільтрація: Додаток має функції пошуку, сортування за кожним полем, вибір елементів для їх подальшого видалення / зміни. Та це не тільки

стосується багів і проектів, а і списку членів команди. Також є різниця в рівнях доступу. Чим більше рівень доступу, тим більше функцій йому відкривається.

2.1.3 Функціональні вимоги

Програма повинна надавати можливість:

- функціонал облікового запису;
- реєструватися;
- заходити в свій обліковий запис (Логін) ;
- змінювати свої дані;
- просматрювати усю інформацію, яка закріплена за організацією, у котрій складається сам користувач, а саме інформацію о колегах, проектах и усіх своїх тикетов;
- фільтрувати інформацію;
    - через систему пошуку;
    - сортувати за кожним полем даних;
    - виводи лише потрібну кількість елементів на екран;
- дивитися статистику тікетов (які баги ще відкриті, який в них статус, пріоритетність, тощо) через графіки;
- змінювати інформацію про тікети до котрих призначений користувач;
- створювати нові тікети, проекти (якщо роль користувача «Submitter») та нові облікові записи користувачів для своєї організації (якщо роль користувача «Admin») ;
- писати персональні нотатки про проект чи про кожен тікет індивідуально;
- залишати коментарі для кожного тікета окремо;
- прикріпити файл до тікета.

2.1.4 Вхідні та вихідні дані

2.1.4.1 Вхідні дані

Вхідними даними є:

ser (користувач)

– name – ім’я;
– surname – прізвище;
– username – імейл;
– phone – номер телефону;
– position – посада у організації;
– organization – організація;
– password – пароль.

uthority

– level – рівень повноважень.

roject (проект)

– title – назва;
– description – опис;
– note – примітка.

icket (тікет):

– title – назва;
– description – опис;
– note – примітка;
– прикріплений файл;
– опис прикріпленого файла;
– тип тікету (баг, проблема, пропозиція, тощо);
– статус (виконаний, у процесі, не зайнятий, тощо);
– срочність (низька, середня, висока, дуже висока).

omment (коментар):

– text – текст.

otification (сповіщання):

– text – текст.

2.1.4.2 Вихідні дані

ser (користувач):

– name – ім’я;
– surname – прізвище;

– username – імейл;
– phone – номер телефону;
– position – посада у організації;
– organization – організація;
– password – пароль;
– authority level – рівень повноважень.

roject (проект):

– title – назва;
– description – опис;
– примітка;
– createdDate – дата створення об’єкту;
– lastUpdateDate – дата останньої зміни.

icket (тікет):

– title – назва;
– description – опис;
– note – примітка;
– attachment – прикріплений файл;
– attachment description – опис прикріпленого файла;
– type – тип тікету (баг, проблема, пропозиція, тощо);
– status – статус (виконаний, у процесі, не зайнятий, тощо);
– priority – срочність (низька, середня, висока, дуже висока);
– createdDate – дата створення об’єкту;
– lastUpdateDate – дата останньої зміни.

omment (коментар):

– текст;
– createdDate – дата створення об’єкту.

otification (сповіщання):

– текст;
– createdDate – дата створення об’єкту.

icket History (історія тікета):

- oldValue – старе значення, яке було змінене;
- newValue – нове значення, на яке було змінене;
- createdDate – дата створення об’єкту.

2.2 Внутрішнє проектування

2.2.1 Проектування бази даних

Моделювання бази даних на концептуальному та логічному рівні

Проект має велику кількість інформації, яку необхідно зберігати на сервері. Було визначено основні сутності та зв'язки бази даних, щоб структурувати її наступним чином:

- Користувач: може створювати проекти, тікети, інших користувачів, може писати коментарі та отримувати повідомлення;
- Повноваження: повноваження, які має користувач;
- Проект: в проекті знаходяться дані користувачів, котрі призначені до проекту та тікети;
- Користувач_Проект: пов'язує Користувача та Проект;
- Тікет: головний елемент програми, який має усю інформацію про тікет;
- Історія тікета: історія змін конкретного тікету;
- Коментар: коментарі, які можна залишати к кожному тікету;
- Повідомлення: повідомлення, котрі приходять до користувача, коли його призначили до якогось проекту чи тікету.

На основі визначених сутностей та зв’язків було побудовано ER-діаграму логічної схеми бази даних (рис. 2.1).

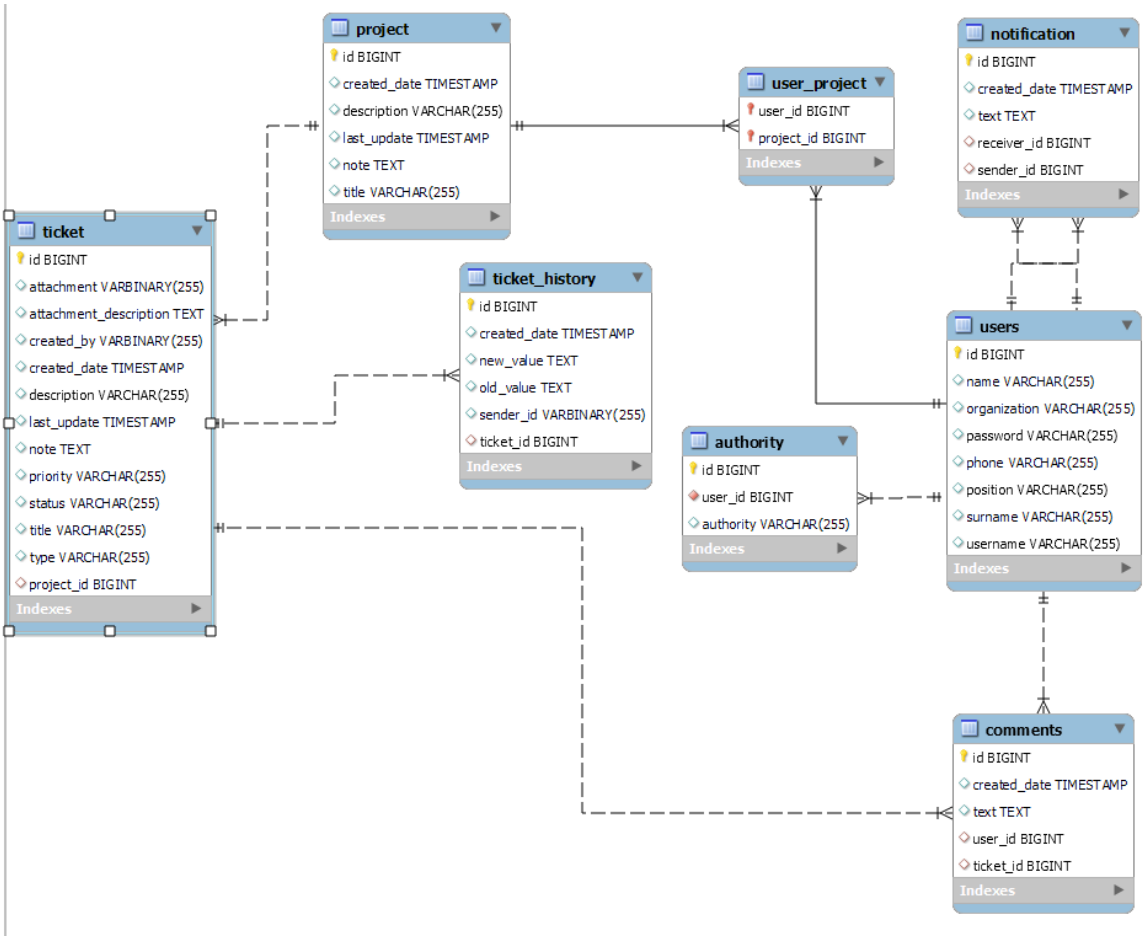

Рисунок 2.1 – Логічна схема бази даних

ормалізація відносин бази даних

База даних знаходиться у першій нормальній формі, тому що відповідає наступним критеріям [3]:

– Кожна комірка таблиці повинна містити одне значення;

– Кожен запис повинен бути унікальним.

База даних знаходиться у другій нормальній формі, тому що відповідає наступним критеріям:

– Вже знаходиться у 1НФ;

– Первинний ключ з одним стовпчиком, який функціонально не залежить від будь-якої підмножини ключового відношення-кандидата.

База даних знаходиться у третій нормальній формі, тому що відповідає наступним критеріям:

– Вже знаходиться у 2НФ;

– Не має перехідних функціональних залежностей.

Проектування архітектури системи

Моделювання словника системи

Під час аналізу зовнішніх специфікацій системи було виділено такі сутності та ідентифіковані такі обов’язки:

– BugTrackerApplication – запуск програми;
– ServletInitializer – ініціалізація сервлета;
– SecurityConfig – конфігурація безпеки;
– Authority – POJO клас рівня повноважень користувача;
– Comment – POJO клас коментар;
– Notification – POJO клас повідомлення;
– Project – POJO клас проект;
– Ticket – POJO клас тікет;
– TicketHistory – POJO клас історія тікету;
– User – POJO клас користувач;
– AuthCredentialRequest – запит на авторизацію;
– ProjectDto – об'єкт передачі даних для класу проект;
– AuthorityEnum – перечень усіх рівней доступу;
– TicketPriorityEnum – перечень усіх пріоритетів для тікету;
– TicketStatusEnum – перечень усіх статусів для тікету;
– TicketTypeEnum – перечень усіх типів для тікету;
– перевіряє JSON Web Token;
– робить настройку CORS;
– репозиторій коментаря;
– NotificationRepository – репозиторій повідомлення;
– ProjectRepository – репозиторій проекта;
– TicketHistoryRepository – репозиторій історії тікету;
– TicketRepository – репозиторій тікету;
– UserRepository – репозиторій користувача;
– сервіс проекту;

– реалізація «UserDetails» сервісу;

– сервіс користувача;

– допоміжний клас для рівней доступу;

– кодувальник паролю;

– допоміжний клас для JSON Web Token;

– контролер для уровнів доступу;

– контролер для коментаря;

– контролер повідомлення;

– контролер проекту;

– контролер тікету;

– контролер користувача.

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу можна буде побачити на діаграмі класів (див. рис. 2.1).

2.2.2.2 Моделювання залежностей

Результат визначення залежностей серед класів наведено у таблиці 2.1.

Таблиця 2.1 – Моделювання залежностей

Клас, який зв'язується	Клас, з яким зв’язуються	Тип зв’язку
BugTrackerApplication	-	-
ServletInitializer	-	-
SecurityConfig		Композиція
		Композиція
Authority	User	Композиція
Comment	Ticket	Композиція
	User	Композиція
Notification	User	Композиція
Project	User	Агрегація

Продовження таблиці 2.1

Ticket	User	Агрегація

	Project	Композиція
TicketHistory	User	Агрегація
	Ticket	Композиція
User		Агрегація
	Project	Асоціація
AuthCredentialRequest	-	-
ProjectDto	User	Агрегація
	Project	Композиція
AuthorityEnum	Authority	Композиція
TicketPriorityEnum	Ticket	Композиція
TicketStatusEnum	Ticket	Композиція
TicketTypeEnum	Ticket	Композиція
JwtFilter		Композиція
		Композиція
WebSecurityCorsFilter	-	-
CommentRepository	Comment	Композиція
NotificationRepository	Norification	Композиція
ProjectRepository	Project	Композиція
TicketHistoryRepository	Ticket	Композиція
	TicketHistory	Композиція
TicketRepository	Ticket	Композиція
UserRepository	User	Композиція
	Project	Асоціація
ProjectService	Project	Композиція
	User	Агрегація
	ProjectRepository	Композиція
UserDetailsServiceImpl	User	Композиція

Продовження таблиці 2.1

	UserRepository	Композиція

	CustomPasswordEncoder	Композиція
UserService	Project	Асоціація
	User	Композиція
	ProjectRepository	Асоціація
	UserRepository	Композиція
AuthorityUtil	User	Композиція
CustomPasswordEncoder	-	-
JwtUtil	-	-
AuthContoller	User	Композиція
		Композиція
	JwtUtil	Композиція
CommentContoller	Comment	Композиція
	User	Композиція
	CommentDto	Композиція
	CommentService	Композиція
NotificationContoller	Notification	Композиція
	User	Композиція
	NotificationService	Композиція
ProjectContoller	Project	Композиція
	User	Агрегація
	ProjectService	Композиція
TicketContoller	Ticket	Композиція
	TicketService	Композиція
UserContoller	Project	Асоціація
	User	Композиція
	UserService	Композиція

Побудова об’єктної моделі (діаграми класів)

Заключним результатом моделювання представимо у вигляді діаграми класів (рис. 2.2).

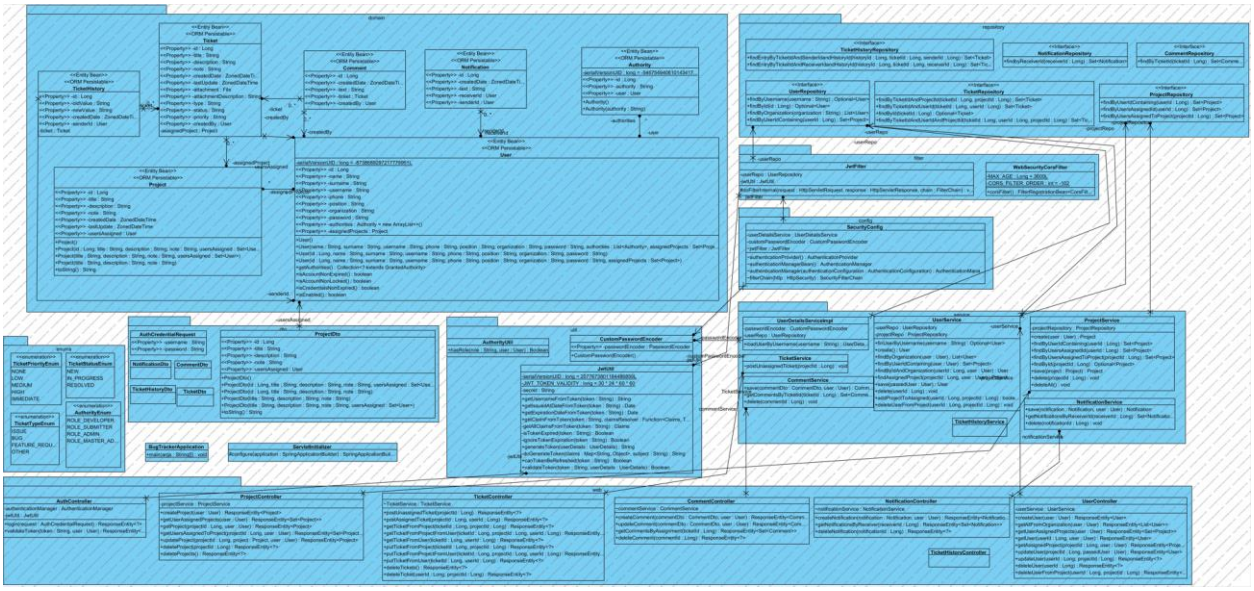

Рисунок 2.2 – Діаграма класів

. Проектування інтерфейсу користувача

Проектування форм та компонентів

На основі функціональних вимог [див. 2.1.3], можна описати які сторінки має надавати інтерфейс програми, та які функції будуть у цих сторінок. Щоб переходити по сторінкам, уверху програми буде створений компонент з посиланнями на всі сторінки (рис. 2.4). Посилання будуть на такі компоненти:

- Логін та Реєстрація (рис. 2. 3);
  - Логін;
    - Полі для ведення інформації (рис. 2.3);
    - Кнопка «Зайти у систему» ;
  - Реєстрація;
    - Полі для ведення інформації (див. рис. 2.3);
    - Кнопка «Зареєструватися»;
- Інформаційна панель (рис. 2.4);
  - Таблиця з проектами;
    - Рядок пошуку;
    - Кнопка «Создати»;
    - Кнопка «Змінити»;
    - Кнопка «Видалити»;
    - Рядок з заголовками елементів таблиці;
    - Рядки з елементами таблиці;
  - Компонент для зміни даних проекта;

- Тікети (рис. 2.4);
  - Таблиця з тікетами користувача;
    - Рядок пошуку;
    - Кнопка «Создати»;
    - Кнопка «Змінити»;
    - Кнопка «Видалити»;
    - Рядок з заголовками елементів таблиці;
    - Рядки з елементами таблиці;
  - Компонент для зміни даних тікету;
- Організація (рис. 2.4);
  - Таблиця користувачів з організації користувача;
    - Рядок пошуку;
    - Кнопка «Создати»;
    - Кнопка «Змінити»;
    - Кнопка «Видалити»;
    - Рядок з заголовками елементів таблиці;
    - Рядки з елементами таблиці;
  - Компонент для зміни даних сотрудника;
- Повідомлення (рис. 2.4);
  - Таблиця з повідомленнями;
    - Рядок пошуку;
    - Кнопка «Создати»;
    - Кнопка «Змінити»;
    - Кнопка «Видалити»;
    - Рядок з заголовками елементів таблиці;
    - Рядки з елементами таблиці;
- Змінити дані облікового запису (рис. 2.5);
  - Pop-up вікно для зміни даних;
  - Кнопка закрити;
  - Кнопка зберегти;
- Про додаток (рис. 2.6);
  - Pop-up вікно;
  - Кнопка закрити;
- Вийти з системи (рис. 2.4).

Веб-сторінки (форми) додатка:

- Логін та Реєстрація;
- Інформаційна панель;
- Проекти;
- Тікети;

– Організація;
– Повідомлення.

2

. На основі моделюванні форм та компонентів [див. 2.2.3.1], можна зробити ескіз інтерфейсу програми:

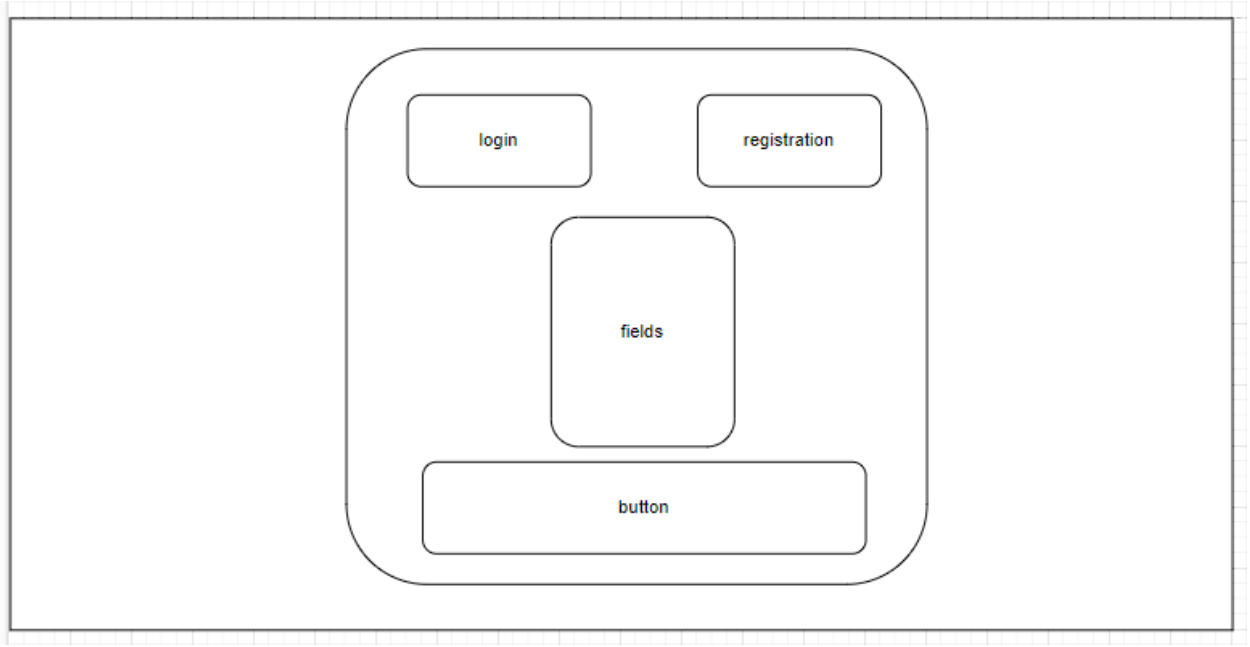

Рисунок 2.3 – Ескіз веб-сторінки (форми) «Логін та Реєстрація»

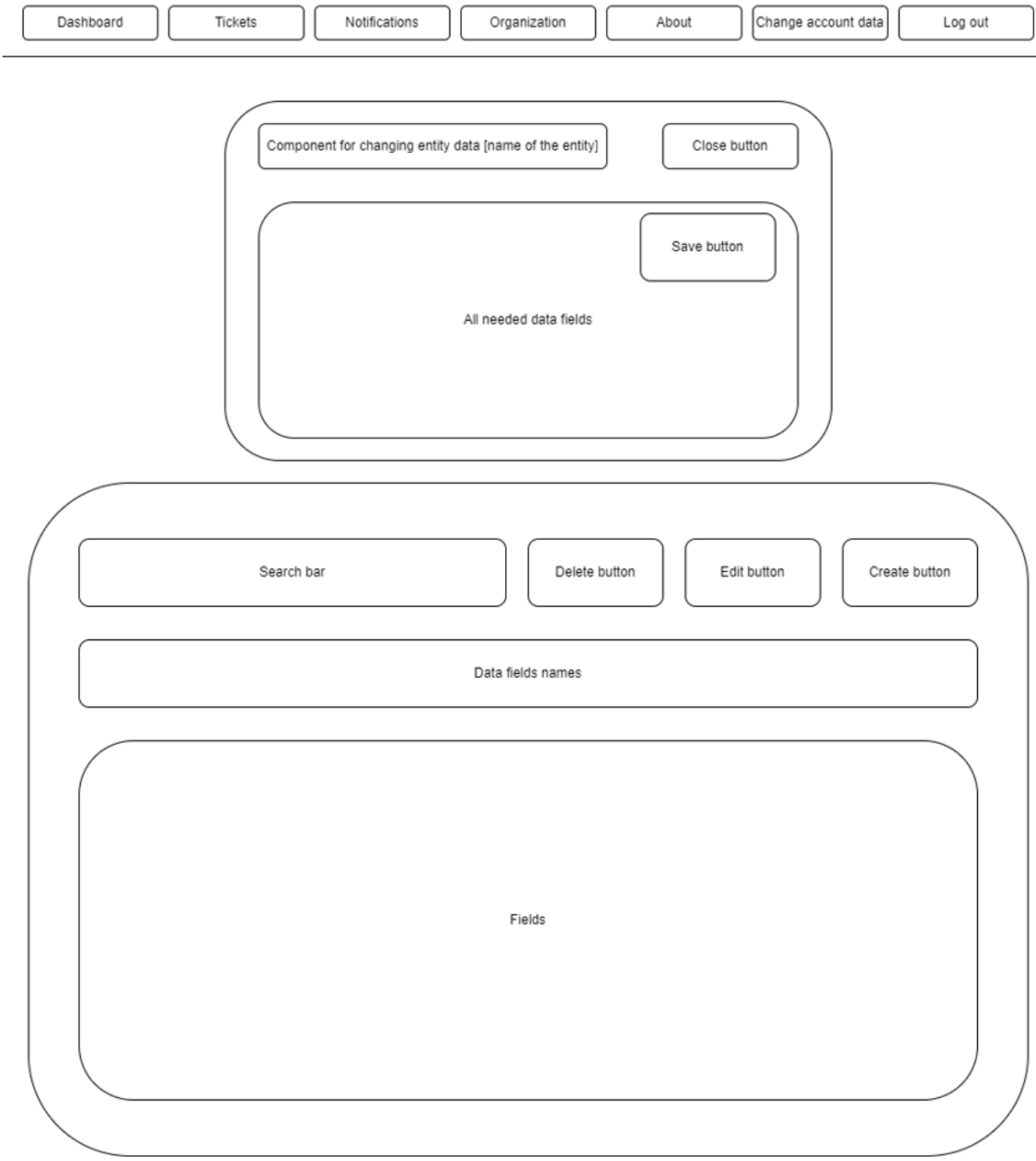

Рисунок 2.4 – Ескіз веб-сторінки (форми) «Інформаційна панель» / «Тікети» /

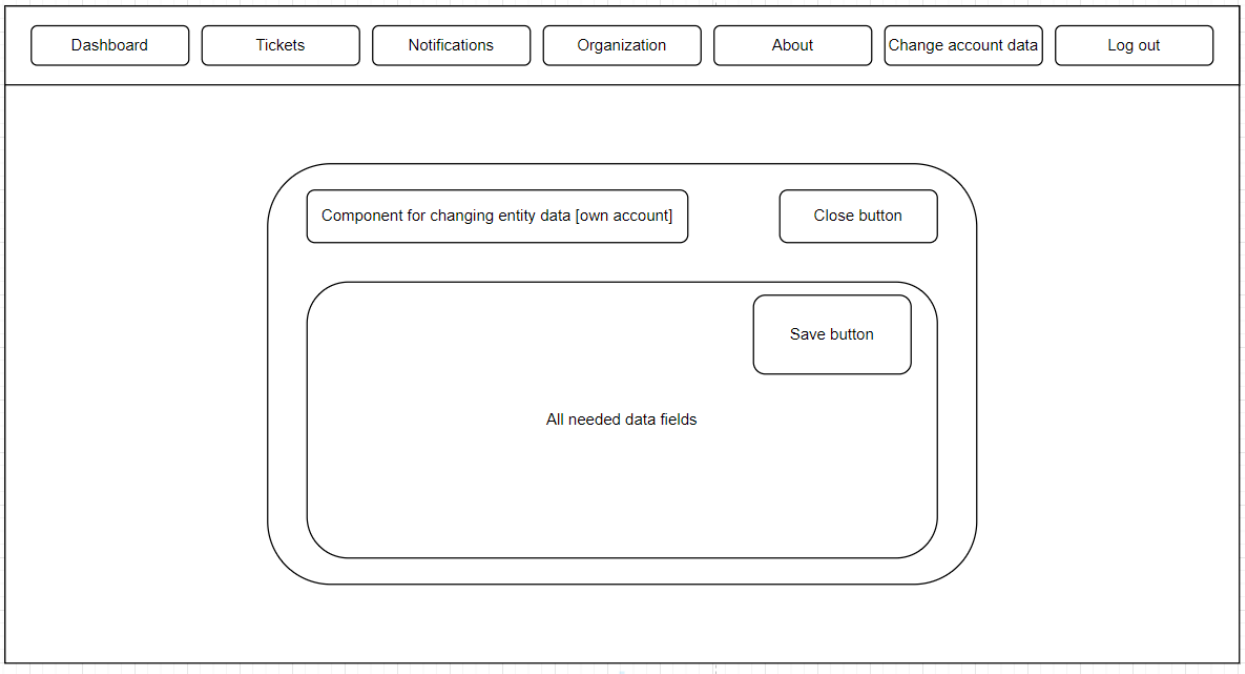

Рисунок 2.5 – Ескіз веб-сторінки (форми) «Змінити дані облікового запису»

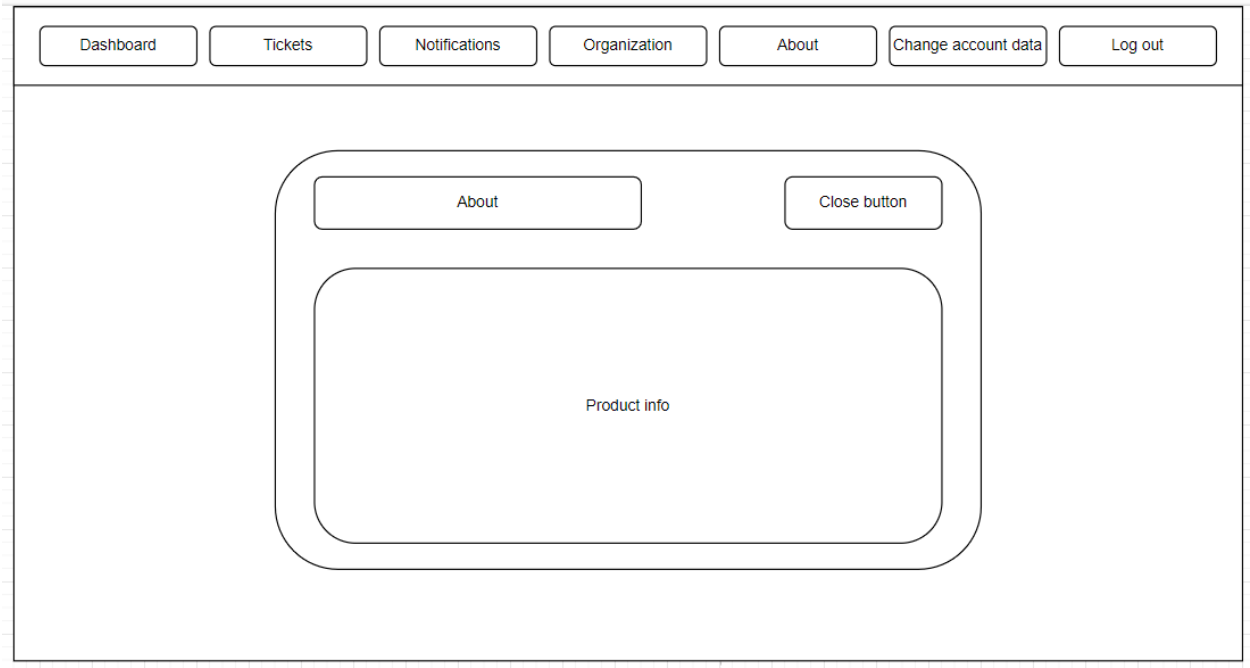

Рисунок 2.6 – Ескіз веб-сторінки (форми) «Про додаток»

2

.

Створення діаграми класів

На основі створених ескізів форм [див. 2.2.3.2] та діаграми класів бек-енду [див. 2.2.2.3], можна зробити діаграму з усіма утилітарними функціями, компонентами та сторінками, по яким можна буде побудувати фронт-енд додатка (рис. 2.7):

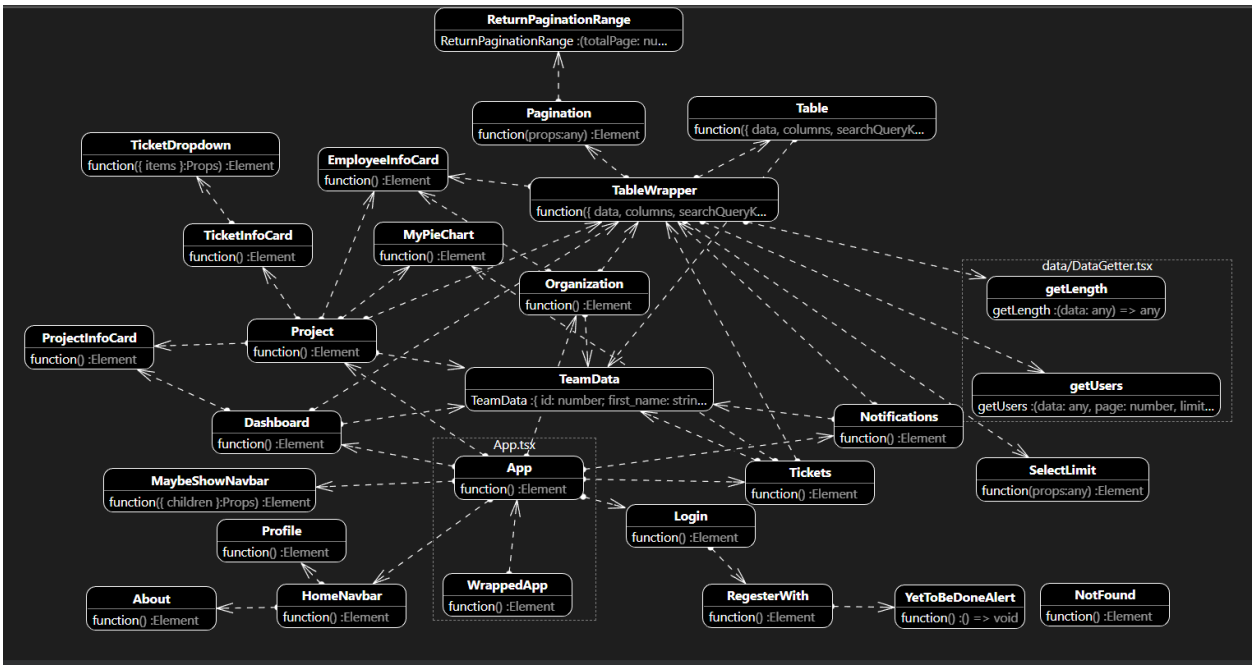

Рисунок 2.7 – Діаграма фрагментів,

Проектування динаміки системи

В якості об’єктів виступають, котрий ініціює взаємодію, та класи, що володіють поведінкою в системі.

Посилаючись на функціональні вимоги [пункт 2.1.3], побудуємо діаграму послідовності, яка продемонструю:

– заходження в систему;

– створення нового тікету;

– зміну інформації тікета.

Діаграма послідовності представлена на рис. 2.8.

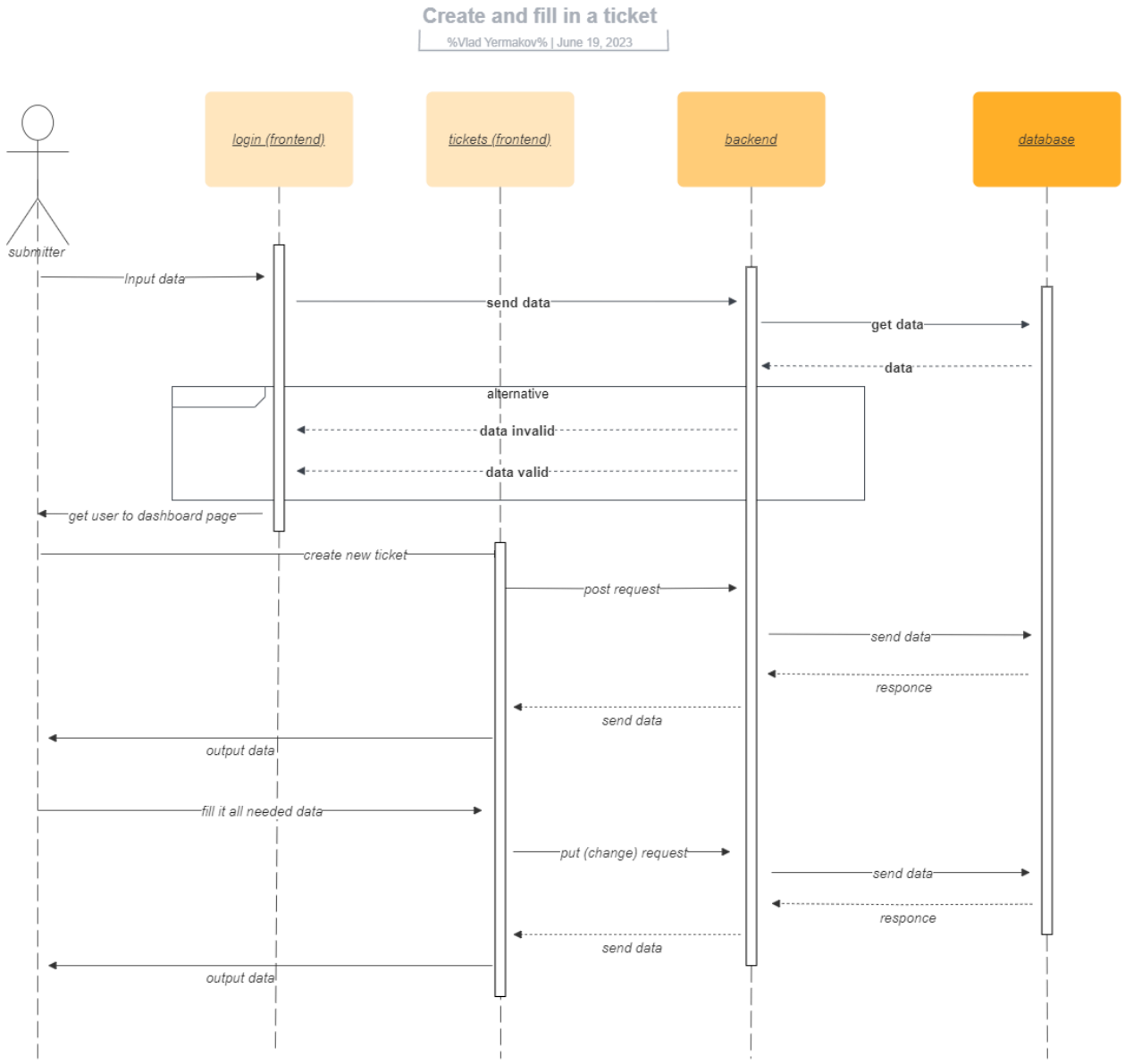

Рисунок 2.8 – Діаграма послідовності для варіанту використання «створення та заповнення тікету користувачем»

Вибір мови та середовища розробки

Для створення бек-енду ПЗ була використана мова Java. Для того, щоб мати якомога менше залежностей при реалізації, Java є високорівневою, об'єктно-орієнтованою мовою програмування, заснованою на класах [4]. Це універсальна мова програмування, розроблена для того, щоб дозволити програмістам написати код один раз і запустити його будь-де (WORA) [5], що означає, що скомпільований код Java може запускатися без необхідності перекомпіляції на будь-яких системах, які підтримують Java [6]. Eclipse був обраний як середовище розробки для бек-енду, тому що в Eclipse є дуже добра

підтримка к інструменталу для розробки бек-енд програм на джаві. Основними фреймворками для розробки бек-енду були обрані Spring Web, для веб-розробки, Spring Security, для зручного логіна та реєстрації, Spring Jpa, для з’єднання бек-енду з базою даних.

Для розробки бази даних було обрано MySQL, так як він має всі необхідні функції для збереження спроектованого обсягу інформації та є однією з найпопулярніших версій SQL.

Для фронт-енду були обрані такі мови як JavaScript, TypeScript, HTML, – це високорівнева мова, яка часто компілюється "just-in-time" і відповідає стандарту ECMAScript[7]. TypeScript – це вільна мова програмування високого рівня з відкритим вихідним кодом, розроблена Microsoft, яка додає до JavaScript статичну типізацію з необов'язковими анотаціями типів[8]. Вона призначена для розробки великих додатків і транспілюється в JavaScript[9]. HTML – це стандартна мова розмітки документів, призначених для відображення у веб-браузері [10]. CSS – це мова таблиць стилів, яка використовується для опису представлення документа, написаного мовою розмітки, наприклад, HTML або XML (включаючи діалекти XML, такі як SVG, MathML або XHTML) [11]. Як основний фреймворк для зручної розробки фронт-енду був обраний React, а основний фреймворк для дизайну – Bootstrap. Обидва ці фреймворка працюють з JavaScript та TypeScript.

Висновки до пункту 2

На етапі проектування системи були створені макети інтерфейсу користувача, фундаментальні діаграми для передбачуваного програмного інструменту та мова програмування, такі як діаграма бази даних, класів та послідовностей. На основі цього розділу був написаний код проекту, якому далі залишилось пройти лише тестування.

3 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

3.1 Тестування методом «білої скриньки»

Тестування "чорного ящика" - це метод, при якому внутрішня організація, структура та використання продукту не беруться до уваги. Іншими словами, тестувальник не знає, як він працює всередині. Чорний ящик оцінює лише зовнішню поведінку системи. Тестуванню піддаються як входи системи, так і її виходи, або реакції. Тестуватися будуть функції фронт-енду, які пов’язані з класом «Користувач» у бек-енді.

Функція 1: Login.

Відповідає за ведення даних для авторизації та відсилання цих даних до бек-енду.

Заголовок function Login() : void

Вхідні дані у неявному вигляді:

–Username (email) – імейл користувача;

–Password – пароль користувача.

Вихідні дані у неявному вигляді:

–Username (email) – імейл користувача;

–Password – пароль користувача.

Код функції:

Тест 1: Ввести некоректні дані.

Програма зазначить де ви помилились або вказали щось не так. На рис 3.1 видно, що користувач не ввів «@» та щось після цього знака у поле email, тому поле світиться красним. Також, програма видає повідомлення «Invalid login attempt».

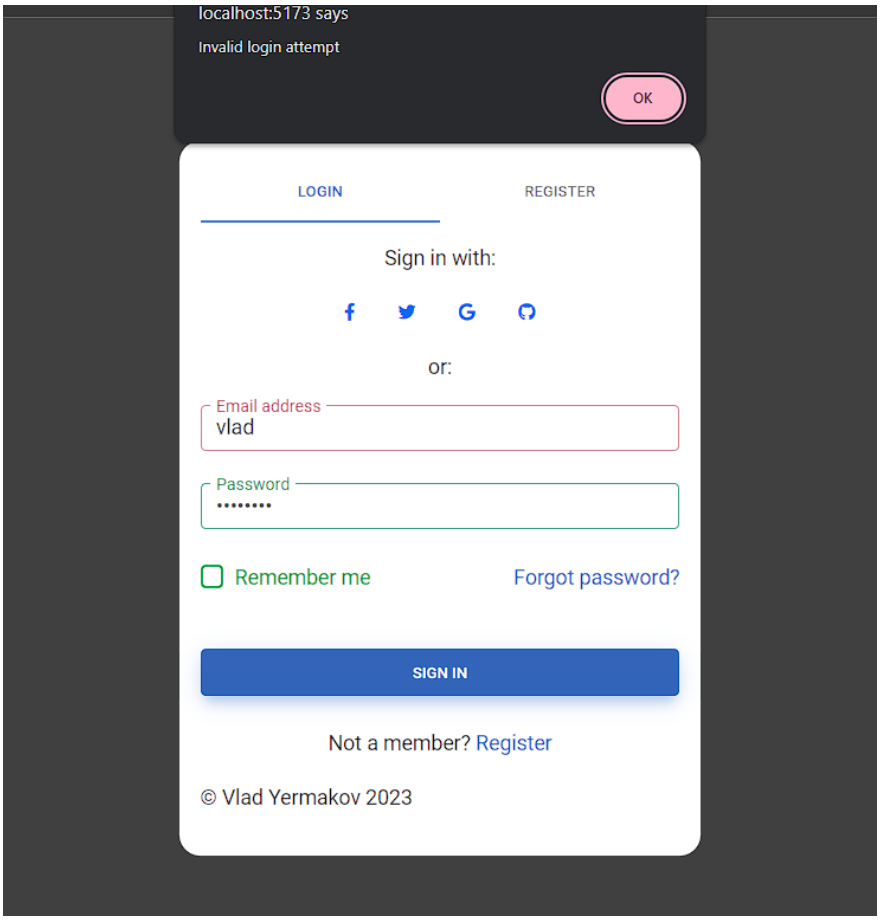

Рисунок 3.1 – Невірне введення інформації у фронт-енду

Тест 2: Ввести коректні дані лише за сенсом (але такого акаунту немає).

Якщо ввести все коректно с точку зору синтаксису та семантики (тобто, наприклад, той ж самий імейл буде ведений вірно), але в загалом такого юзера не має в БД, то програма видасть таке ж повідомлення як на рис. 3.1.

Тест 3: Ввести коректні дані.

Якщо ввести все коректно, то програма перекине користувача на його головну сторінку.

3.2 Тестування методом «чорної скриньки»

Тестування «чорної скриньки» – це метод тестування програмного забезпечення, при якому тестувальник не знає про внутрішню структуру, дизайн або реалізацію того, що тестується. Тестується лише зовнішня структура та дизайн. Для тестування був обраний клас «AuthController», та дії які можна з цим класом робити, а саме приймати дані з фронт-енду та відсилати у базу даних запит за цими даними.

Функція 1: Login.

Відповідає за приймання даних з фронт-енду, відсилання у базу даних запит за цими даними та повернення відповіді до фронт-енду.

Заголовок @PostMapping("login") @CrossOrigin(origins = "*") public ResponseEntity<?> login(@RequestBody AuthCredentialRequest request).

Вхідні дані:

– AuthCredentialRequest request – запит з фронт-енду.

Вихідні дані:

– ResponseEntity<?> – відповідь від бази даних.

Код функції:

Тест 1: Ввести некоректні дані.

Якщо на вхід для бек-енду поступає пустий об’єкт, то отримуємо помилку з кодом 401 – «Unauthorized» (рис. 3.2), що відповідає виключній ситуації наступного змісту (рис. 3.3).

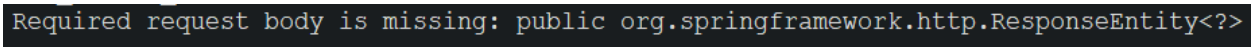

Рисунок 3.2 – Повідомлення бек-енду о помилці з кодом 401

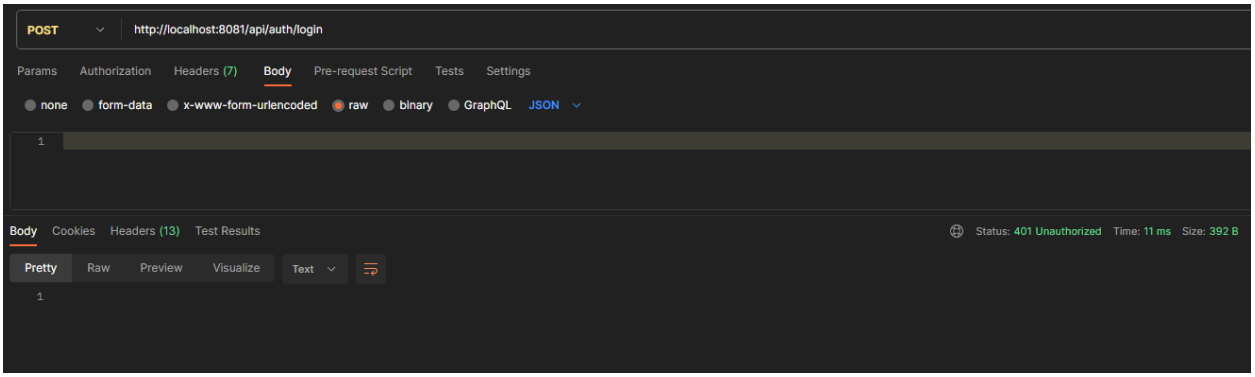

Рисунок 3.3 – Пустий Post Request з помилкою 401

Тест 2: Ввести коректні дані лише за сенсом (але такого акаунту немає).

Якщо ввести все коректно с точку зору синтаксису та семантики (тобто, наприклад, той ж самий імейл буде ведений вірно), але в загалом такого користувача не має в БД, то бек-енд видасть повідомлення як на рис. 3.4, що облікові дані недійсні.

```
org.springframework.security.core.userdetails.UsernameNotFoundException: Invalid credentials
```

Рисунок 3.4 – Повідомлення бек-енду

Тест 3: Ввести коректні дані.

Якщо ввести все коректно, то програма отримає конкретного користувача з БД і відішле об’єкт до фронт-енду (рис. 3.5).

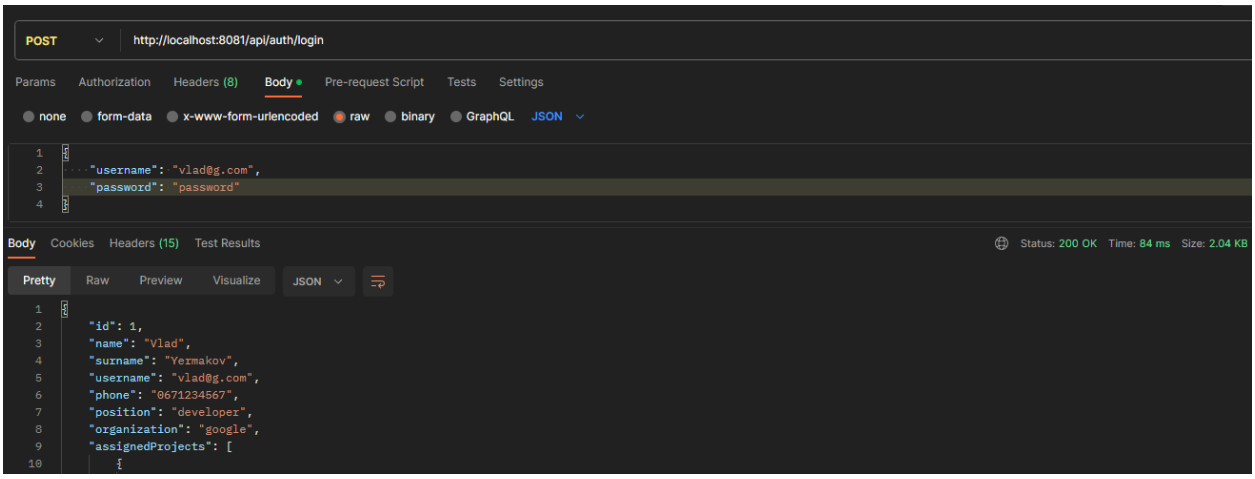

Рисунок 3.5 – Вірний Post Request

Висновки до пункту 3

При тестуванні, помилок виявлено не було. Через обмеження часу проведено тестування лише обмеженого частину проекту, тестування інших складових може проводитись за такою ж схемою.

ВИСНОВОК

Цей дипломний проект був зосереджений на створенні веб-додатку для відстеження помилок. Однією з цілей проекту була ретельна та ефективна система моніторингу та управління проблемами та питаннями програмного забезпечення протягом усього життєвого циклу розробки програмного забезпечення.

Проект розпочався з представлення ідей, що стоять за відстеженням помилок, підкреслюючи цінність ефективного управління проблемами в ініціативах з розробки програмного забезпечення. А вивчена література дозволило отримати інформацію про веб-розробку в цілому.

Веб-додаток для відстеження помилок було розроблено на основі формулювання проблеми та постановки завдання. До програми було додано такі важливі функції, як повідомлення про помилки, відстеження проблем, управління робочим процесом, співпраця та звітність. Ці функції були створені для покращення командної роботи, пришвидшення процесу виправлення помилок та підвищення стандартів програмного забезпечення в цілому.

У процесі розробки використовувалися численні фреймворки, технології, інструменти та мови розробки, такі як HTML, CSS, JavaScript, TypeScript, React, Bootstrap. Фреймворки, технології та інструменти бекенду та мови розробки, такі як Java, Spring Web, Spring Security, Spring Jpa. А також MySql як мова для роботи з базою даних.

Проект пройшов всі необхідні етапи розробки, такі як збір вимог до ПЗ, зовнішнє та внутрішнє проектування, саме проектування тести та повну документацію с усіма потрібними діаграмами.

Отже, онлайн-додаток, створений для відстеження помилок в цьому дипломному проекті, надає корисну послугу для команд розробників програмного забезпечення. Він пропонує зручну та адаптивну платформу для ефективного відстеження проблем з програмним забезпеченням, сприяння співпраці та підвищення ефективності всього проекту. Можливості та функції додатку відповідають найкращим практикам та галузевим стандартам, що

допомагає успішно вирішувати проблеми та створювати високоякісні програмні продукти.

ЛІТЕРАТУРА

emuk I. Web Application Architecture: A Guide Through the Intricate Process of Building an App | LITSLINK Blog [Electronic resource] / Iryna Deremuk // L

i[Electronic resource] / Saul Greenberg // www.researchgate.net/. – Mode of t

alization in DBMS (SQL)? 1NF, 2NF, 3NF Example [Electronic resource] / Richard Piterson // Guru99. – Mode of i

4. Contributors to Wikimedia projects. Java (programming language) - Wikipedia [Electronic resource] / Contributors to Wikimedia projects // Wikipedia, the free encyclopedia. – Mode of

– resource] / Contributors to Wikimedia projects // Wikipedia, the free encyclopedia (date of access: 20.06.2023). – Title from screen. Mode of

6. The Java Language Environment [Electronic resource] // Oracle | Cloud Applications and Cloud Platform. – Mode of

7. ECMAScript® 2024 Language Specification [Electronic resource] // TC39 - S

8. Contributors to Wikimedia projects. TypeScript - Wikipedia [Electronic resource] / Contributors to Wikimedia projects // Wikipedia, the free e

9. Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem? [Electronic resource] // Ars Technica. – Mode of

ДОДАТОК А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

18.02.23

Веб-додаток «БАГ-ТРЕКЕР»

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

-01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

18.02.23

Керівник розробки

Ірина ШАПОВАЛ

18.02.23

Виконавець

Вадим АЛЄКСЄЄНКО

18.02.23

Норм-контролер

Світлана ВОЛКОВА

18.02.22

2023

ЗАТВЕРДЖЕНО

1116130.01290-01-ЛЗ

ВЕБ-ДОДАТОК «БАГ-ТРЕКЕР»

Технічне завдання

Листів 11

2022

ЗМІСТ

ВЕДЕННЯ

Веб-додаток «Баг-трекер», що розробляється, має виконувати функції менеджменту команд, проектів та тікетів.

Основна мета програми – висвітлити значення відстеження помилок у веб-додатках для розробки програмного забезпечення. Ці веб-інструменти полегшують процес вирішення проблем, покращують комунікацію та співпрацю і, зрештою, призводять до розробки високоякісних програмних додатків завдяки застосуванню методичного та ефективного підходу до управління помилками. Це дослідження має на меті надати командам розробників програмного забезпечення знання та розуміння, необхідні для використання веб-інструментів для відстеження помилок у реальному світі, шляхом вивчення їхніх переваг, особливостей та корисних застосувань.

ІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.22 №1209ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи - “Баг-трекер”. Керівник - ст. викладач Шаповал І. В.

РИЗНАЧЕННЯ РОЗРОБКИ

Функціональна мета веб-застосунку для відстеження помилок – запропонувати ретельний та ефективний метод моніторингу та управління дефектами та проблемами програмного забезпечення протягом усього життєвого циклу розробки програмного забезпечення. Програмне забезпечення діє як основний центр для збору, категоризації, визначення пріоритетів та виправлення проблем, виявлених користувачами, тестувальниками або членами команди.

Інша мета програми – запропонувати стабільну та зручну платформу, яка спрощує моніторинг та усунення помилок. Посилення співпраці, спрощення комунікації та забезпечення ефективного управління помилками – все це сприяє підвищенню якості програмного забезпечення та задоволеності клієнтів.

ИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

имоги до функціональних характеристик

Програма повинна надавати можливість:

– функціонал облікового запису;

– реєструватися;

– заходити в свій обліковий запис (Логін) ;

– змінювати свої дані;

– просматрювати усю інформацію, яка закріплена за організацією, у котрій складається сам користувач, а саме інформацію о колегах, проектах и усіх своїх тикетов;

– фільтрувати інформацію;

– через систему пошуку;

– сортувати за кожним полем даних;

– виводи лише потрібну кількість елементів на екран;

– дивитися статистику тікетов (які баги ще відкриті, який в них статус, пріоритетність, тощо) через графіки;

– змінювати інформацію про тікети до котрих призначений користувач;

– створювати нові тікети, проекти (якщо роль користувача Submitter») та нові облікові записи користувачів для своєї організації (якщо роль користувача «Admin») ;

– писати персональні нотатки про проект чи про кожен тікет індивідуально;

– залишати коментарі для кожного тікета окремо;

– прикріпити файл до тікета.

имоги до надійності

Вимоги до надійності наступні:

Вимоги до надійності наступні: забезпечення стійкого функціонування програми; контроль вхідної і вихідної інформації; наявність архівної копії тексту програми на зовнішньому носії

имоги експлуатації і вимоги до кліматичних умов:

Температура – 21-25 С, відносна вологість 40-60%. Обслуговування не потрібне. Для роботи із ПЗ достатньо однієї людини, що має досвід роботи із ПК, і бажає проводити дослідження у сфері навчання і тестування згорткових нейронних мереж.

имоги до складу та параметрів технічних засобів

Склад технічних засобів: процесор з тактовою частотою 2 ГГЦ або вище; 6 Мб. місця на накопичувачі; 2 Гб. оперативної пам’яті.

имоги до інформаційної і програмної сумісност

Програма має функціонувати під управлінням ОС Windows 10\11. Програма написана на мові програмування Java. Перевагою цієї мови програмування є те, що вона об’єктно-орієнтована, тобто код можна поділити на логічні частини. Також ця мова має великий рівень безпеки і створення для написання додатків на операційну систему Windows та інші платформи. На системі має бути встановлений Spring Boot 3.0 або вище.

имоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

имоги до транспортування і зберігання

Транспортування повинно проводитись в упаковці. Умови зберігання повинні забезпечувати безпеку продукту.

ИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача.

Вся документація програмного додатку повинна задовольняти вимоги до програмної документації.

ТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиці 1. – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	
Робочий проект	Програмування та відлагодження програми.	
	Тестування програми	
	Розробка, узгодження і затвердження програмної документації.	

ОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

Прийом здійснюється комісією у складі:

- Горячкін В. М. (керівник підрозділу);
- Шаповал І. В. (керівник розробки).

ІБЛІОГРАФІЧНИЙ СПИСОК

Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

ДОДАТОК Б

Специфікація

ЗАТВЕРДЖЕНО

1290-01-ЛЗ

БАГ-ТРЕКЕР

Специфікація

Листів 2

Специфікації

Таблиця 1.1 – Специфікації

Позначення	Найменування	Примітка
-01-ЛЗ	Документація Лист затвердження	
-01-ЛЗ	Технічне завдання Лист затвердження	
-01 12 01-ЛЗ	Специфікація Лист затвердження Текст програми	

ДОДАТОК В

Текст програми

ЗАТВЕРДЖЕНО

1116130.01315-01 12 01-ЛЗ

БАГ-ТРЕКЕР

Текст програми

Листів 110

BugTrackerApplication.java

```
package com.yermakov.bugtracker;

import
org.springframework.boot.SpringAppli
cation;
import
org.springframework.boot.autoconfigur
e.SpringBootApplication;

@SpringBootApplication
public class BugTrackerApplication {

	public static void main(String[]
args) {

	SpringApplication.run(BugTrack
erApplication.class, args);
	}

}
```

ServletInitializer.java

```
package com.yermakov.bugtracker;


import
org.springframework.boot.builder.Spri
ngApplicationBuilder;
import
org.springframework.boot.web.servlet.
support.SpringBootServletInitializer;


public class ServletInitializer extends
SpringBootServletInitializer {


	@Override
	protected
SpringApplicationBuilder
configure(SpringApplicationBuilder
application) {
		return
application.sources(BugTrackerApplic
ation.class);
	}


}
```

SecurityConfig.java

```
package
com.yermakov.bugtracker.config;


import java.util.Arrays;


import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.context.annotatio
n.Bean;
import
org.springframework.context.annotatio
n.Configuration;
import
org.springframework.security.authentic
ation.AuthenticationManager;
import
org.springframework.security.authentic
ation.AuthenticationProvider;
```

```
import
org.springframework.security.authentic
ation.ProviderManager;
import
org.springframework.security.authentic
ation.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.an
notation.authentication.configuration.A
uthenticationConfiguration;
import
org.springframework.security.config.an
notation.web.builders.HttpSecurity;
import
org.springframework.security.config.an
notation.web.configuration.EnableWeb
Security;
import
org.springframework.security.config.an
notation.web.configuration.WebSecurit
yConfiguration;
import
org.springframework.security.config.ht
tp.SessionCreationPolicy;
import
org.springframework.security.core.user
details.UserDetailsService;
import
org.springframework.security.web.Sec
urityFilterChain;
import
org.springframework.security.web.auth
entication.UsernamePasswordAuthenti
cationFilter;

import
com.yermakov.bugtracker.filter.JwtFilt
er;
import
com.yermakov.bugtracker.util.Custom
PasswordEncoder;

import
jakarta.servlet.http.HttpServletRespons
e;

@EnableWebSecurity
@Configuration
public class SecurityConfig {

        @Autowired
        private UserDetailsService
userDetailsService;
        @Autowired
        private CustomPasswordEncoder
customPasswordEncoder;
        @Autowired
        JwtFilter jwtFilter;

    @Bean
```

```
    AuthenticationProvider
authenticationProvider() {

        DaoAuthenticationProvider
provider = new
DaoAuthenticationProvider();

        provider.setUserDetailsService(u
serDetailsService);

        provider.setPasswordEncoder(cu
stomPasswordEncoder.getPasswordEn
coder());
                return provider;
        }

    @Bean
    AuthenticationManager
authenticationManagerBean() throws
Exception {
                return new
ProviderManager(Arrays.asList(authen
ticationProvider()));
        }

    @Bean
    AuthenticationManager
authenticationManager(Authentication
Configuration
authenticationConfiguration)
            throws Exception {
                return
authenticationConfiguration.getAuthen
ticationManager();
        }

    @Bean
    SecurityFilterChain
filterChain(HttpSecurity http) throws
Exception {
                // working basic code
                // http.csrf(csrf ->
                //
csrf.disable()).authorizeHttpRequests().
anyRequest().authenticated().and()
                //
.httpBasic(withDefaults());

                // Enable CORS and
disable CSRF
                // Enable CORS and
disable CSRF
            http.csrf(csrf ->
csrf.disable());
            http.cors(cors ->
cors.disable());

            // Set session management to
stateless
```

```
http.sessionManagement(management -
> management

.sessionCreationPolicy(SessionCreatio
nPolicy.STATELESS));

        // Set unauthorized requests
exception handler
        http.exceptionHandling(handling -
>
handling.authenticationEntryPoint((req
uest, response, exception) -> {

response.sendError(HttpServletRespon
se.SC_UNAUTHORIZED,
exception.getMessage());
        }));



          http.authorizeHttpRequests(
                                  auth ->
auth.requestMatchers("/api/auth/**").p
ermitAll().anyRequest().authenticated()
);

                    // we say that when
UsernamePasswordAuthenticationFilte
r happens, use our jwtFilter first

          http.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilte
r.class);

                    return http.build();
          }
}
```

Authority.java

```
package
com.yermakov.bugtracker.domain;

import
org.springframework.security.core.Gra
ntedAuthority;

import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import
jakarta.persistence.ManyToOne;

@Entity
public class Authority implements
GrantedAuthority {
```

```
	private static final long
serialVersionUID = -
5467549406101434175L;
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	private Long id;
	private String authority;
	@ManyToOne(optional = false)
	private User user;

	public Authority() {

	}

	public Authority(String
authority) {
		this.authority = authority;
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	@Override
	public String getAuthority() {
		return authority;
	}

	public void setAuthority(String
authority) {
		this.authority = authority;
	}

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}
}
```

Comment.java

```
package
com.yermakov.bugtracker.domain;

import java.time.ZonedDateTime;

import
com.fasterxml.jackson.annotation.JsonI
gnore;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import
jakarta.persistence.ManyToOne;
```

```
import jakarta.persistence.Table;

//'@Entity' would create a table in DB
@Entity
@Table(name = "comments")
public class Comment {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY) // going
to leverage MySQL auto-increment
functionality
	private Long id;
	// use @JsonIgnore so we don't
send this data object (assignment) back
to the
	// front-end when we send
Comment object
	@JsonIgnore
	@ManyToOne
	private Ticket ticket;
	@ManyToOne
	@JoinColumn(name =
"user_id")
	private User createdBy;
	@Column(columnDefinition =
"TIMESTAMP")
	private ZonedDateTime
createdDate;
	@Column(columnDefinition =
"TEXT")
	private String text;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public ZonedDateTime
getCreatedDate() {
		return createdDate;
	}

	public void
setCreatedDate(ZonedDateTime
createdDate) {
		this.createdDate =
createdDate;
	}

	public User getCreatedBy() {
		return createdBy;
	}

	public void setCreatedBy(User
createdBy) {
		this.createdBy =
createdBy;
	}

	public String getText() {
		return text;
	}

	public void setText(String text) {
		this.text = text;
	}

	public Ticket getTicket() {
		return ticket;
	}

	public void setTicket(Ticket
ticket) {
		this.ticket = ticket;
	}
}
```

Notification.java

```
package
com.yermakov.bugtracker.domain;

import java.time.ZonedDateTime;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
```

```
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import
jakarta.persistence.ManyToMany;
import
jakarta.persistence.ManyToOne;

@Entity
public class Notification {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY) // going
to leverage MySQL auto-increment
functionality
	private Long id;
	@ManyToOne
	@JoinColumn(name =
"receiver_id")
	private User receiverId;
	@ManyToOne
	@JoinColumn(name =
"sender_id")
	private User senderId;
	@Column(columnDefinition =
"TIMESTAMP")
	private ZonedDateTime
createdDate;
	@Column(columnDefinition =
"TEXT")
	private String text;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public User getReceiverId() {
		return receiverId;
	}

	public void setReceiverId(User
receiverId) {
		this.receiverId =
receiverId;
	}

	public User getSenderId() {
		return senderId;
	}

	public void setSenderId(User
senderId) {
		this.senderId = senderId;
	}

	public ZonedDateTime
getCreatedDate() {
		return createdDate;
	}

	public void
setCreatedDate(ZonedDateTime
createdDate) {
		this.createdDate =
createdDate;
	}

	public String getText() {
		return text;
	}

	public void setText(String text) {
		this.text = text;
	}
}
```

Project.java

```
package
com.yermakov.bugtracker.domain;

import java.time.ZonedDateTime;
import java.util.Set;
```

```
import
com.fasterxml.jackson.annotation.Json
BackReference;
import
com.fasterxml.jackson.annotation.JsonI
dentityInfo;
import
com.fasterxml.jackson.annotation.JsonI
gnore;
import
com.fasterxml.jackson.annotation.JsonI
gnoreProperties;
import
com.fasterxml.jackson.annotation.Obje
ctIdGenerators;

import jakarta.annotation.Nonnull;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import
jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;

@JsonIgnoreProperties({"hibernateLaz
yInitializer"})
@JsonIdentityInfo(generator =
ObjectIdGenerators.PropertyGenerator.
class, property = "id")
@Entity
@Table(name = "project")
public class Project {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	@Column(name="id")
	private Long id;
	@Nonnull
	@Column(name="title")
	private String title;
	@Column(name="description")
	private String description;
	@Column(name="note",
columnDefinition = "TEXT")
	private String note;
	@Column(name="created_date",
columnDefinition = "TIMESTAMP")
	private ZonedDateTime
createdDate;
	@Column(name="last_update",
columnDefinition = "TIMESTAMP")
	private ZonedDateTime
lastUpdate;

	@ManyToMany(mappedBy =
"assignedProjects")
	private Set<User>
usersAssigned;

	public Project() {
	}

	public Project(Long id, String
title, String description, String note,
Set<User> usersAssigned) {
		this.id = id;
		this.title = title;
		this.description =
description;
		this.note = note;
		this.lastUpdate =
ZonedDateTime.now();
		this.usersAssigned =
usersAssigned;
	}

	public Project(String title, String
description, String note, Set<User>
usersAssigned) {
		this.title = title;
		this.description =
description;
		this.note = note;
```

```
            this.createdDate =
ZonedDateTime.now();
            this.lastUpdate =
ZonedDateTime.now();
            this.usersAssigned =
usersAssigned;
        }

        public Project(String title, String
description, String note) {
            this.title = title;
            this.description =
description;
            this.note = note;
            this.createdDate =
ZonedDateTime.now();
            this.lastUpdate =
ZonedDateTime.now();
        }

        public Set<User>
getUsersAssigned() {
            return usersAssigned;
        }

        public void
setUsersAssigned(Set<User>
usersAssigned) {
            this.usersAssigned =
usersAssigned;
        }

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public String getTitle() {
            return title;
        }

        public void setTitle(String title)
{
            this.title = title;
        }

        public String getDescription() {
            return description;
        }

        public void setDescription(String
description) {
            this.description =
description;
        }

        public String getNote() {
            return note;
        }

        public void setNote(String note)
{
            this.note = note;
        }

        public ZonedDateTime
getCreatedDate() {
            return createdDate;
        }

        public void
setCreatedDate(ZonedDateTime
createdDate) {
            this.createdDate =
createdDate;
        }

        public ZonedDateTime
getLastUpdate() {
            return lastUpdate;
        }

        public void
setLastUpdate(ZonedDateTime
lastUpdate) {
```

```
                this.lastUpdate =
lastUpdate;
        }

        @Override
        public String toString() {
                return "Project [id=" + id
+ ", title=" + title + ", description=" +
description + ", note=" + note
                                + ",
createdDate=" + createdDate + ",
lastUpdate=" + lastUpdate + ",
usersAssigned=" + usersAssigned
                                + "]";
        }
}
```

Ticket.java

```
package
com.yermakov.bugtracker.domain;

import java.io.File;
import java.time.ZonedDateTime;
import java.util.Set;

import jakarta.annotation.Nonnull;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import
jakarta.persistence.ManyToMany;
import
jakarta.persistence.ManyToOne;

@Entity
public class Ticket {
        @Id
        @GeneratedValue(strategy =
GenerationType.IDENTITY)
        private Long id;
        @Nonnull
        private String title;
        private String description;
        @Column(columnDefinition =
"TEXT")
        private String note;
        @JoinColumn(name =
"user_id")
        private User createdBy;
        @ManyToOne
        @JoinColumn(name =
"project_id")
        private Project assignedProject;
        @Column(columnDefinition =
"TIMESTAMP")
        private ZonedDateTime
createdDate;
        @Column(columnDefinition =
"TIMESTAMP")
        private ZonedDateTime
lastUpdate;
        private File attachment;
        @Column(columnDefinition =
"TEXT")
        private String
attachmentDescription;
        private String type;
        private String status;
        private String priority;

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public String getTitle() {
                return title;
        }

        public void setTitle(String title)
{
                this.title = title;
```

```
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String
description) {
		this.description =
description;
	}

	public String getNote() {
		return note;
	}

	public void setNote(String note)
{
		this.note = note;
	}

	public User getCreatedBy() {
		return createdBy;
	}

	public void setCreatedBy(User
createdBy) {
		this.createdBy =
createdBy;
	}

	public ZonedDateTime
getCreatedDate() {
		return createdDate;
	}

	public void
setCreatedDate(ZonedDateTime
createdDate) {
		this.createdDate =
createdDate;
	}

	public ZonedDateTime
getLastUpdate() {
		return lastUpdate;
	}

	public void
setLastUpdate(ZonedDateTime
lastUpdate) {
		this.lastUpdate =
lastUpdate;
	}

	public File getAttachment() {
		return attachment;
	}

	public void setAttachment(File
attachment) {
		this.attachment =
attachment;
	}

	public String
getAttachmentDescription() {
		return
attachmentDescription;
	}

	public void
setAttachmentDescription(String
attachmentDescription) {
		this.attachmentDescription
= attachmentDescription;
	}

	public String getType() {
		return type;
	}

	public void setType(String type)
{
		this.type = type;
	}

	public String getStatus() {
		return status;
	}
```

```
	public void setStatus(String
status) {
		this.status = status;
	}

	public String getPriority() {
		return priority;
	}

	public void setPriority(String
priority) {
		this.priority = priority;
	}
}
```

TicketHistory.java

```
package
com.yermakov.bugtracker.domain;

import java.time.ZonedDateTime;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import
jakarta.persistence.ManyToMany;
import
jakarta.persistence.ManyToOne;

@Entity
public class TicketHistory {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY) // going
to leverage MySQL auto-increment
functionality
	private Long id;
	@JoinColumn(name =
"user_id")
	private User senderId;
	@Column(columnDefinition =
"TEXT")
	private String oldValue;
	@Column(columnDefinition =
"TEXT")
	private String newValue;
	@Column(columnDefinition =
"TIMESTAMP")
	private ZonedDateTime
createdDate;
	@ManyToOne
	private Ticket ticket;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public User getSenderId() {
		return senderId;
	}

	public void setSenderId(User
senderId) {
		this.senderId = senderId;
	}

	public String getOldValue() {
		return oldValue;
	}

	public void setOldValue(String
oldValue) {
		this.oldValue = oldValue;
	}

	public String getNewValue() {
		return newValue;
	}
```

```
        public void setNewValue(String
newValue) {
                this.newValue =
newValue;
        }

        public ZonedDateTime
getCreatedDate() {
                return createdDate;
        }

        public void
setCreatedDate(ZonedDateTime
createdDate) {
                this.createdDate =
createdDate;
        }
}
```

User.java

```
package
com.yermakov.bugtracker.domain;


import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Set;


import
org.springframework.security.core.Gra
ntedAuthority;
import
org.springframework.security.core.user
details.UserDetails;
import
com.fasterxml.jackson.annotation.JsonI
dentityInfo;
import
com.fasterxml.jackson.annotation.JsonI
gnore;
import
com.fasterxml.jackson.annotation.JsonI
gnoreProperties;
import
com.fasterxml.jackson.annotation.Json
ManagedReference;
import
com.fasterxml.jackson.annotation.Obje
ctIdGenerators;


import jakarta.annotation.Nonnull;
import
jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import
jakarta.persistence.GeneratedValue;
import
jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
```

```
import
jakarta.persistence.ManyToMany;
import
jakarta.persistence.OneToMany;
import jakarta.persistence.Table;

@JsonIgnoreProperties({"hibernateLaz
yInitializer"})
@JsonIdentityInfo(generator =
ObjectIdGenerators.PropertyGenerator.
class, property = "id")
@Entity
@Table(name = "users")
public class User implements
UserDetails {
	private static final long
serialVersionUID = -
8738689297217779951L;
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	@Column(name="id")
	private Long id;
	@Nonnull
	@Column(name="name")
	private String name;
	@Nonnull
	@Column(name="surname")
	private String surname;
	@Nonnull
	@Column(name="username",
unique = true)
	private String username;
	@Nonnull
	@Column(name="phone")
	private String phone;
	@Nonnull
	@Column(name="position")
	private String position;
	@Nonnull
	@Column(name="organization")
	private String organization;
	@JsonIgnore
	@Nonnull
	@Column(name="password")
	private String password;
	// spring demands prefix
"ROLE_" to work properly to all
authorities
	@OneToMany(fetch =
FetchType.EAGER, mappedBy =
"user")
	@JsonIgnore
	@Nonnull
	private List<Authority>
authorities = new ArrayList<>();

	@ManyToMany(fetch =
FetchType.EAGER, cascade =
CascadeType.ALL)
```

```
	@JoinTable(name =
"user_project", joinColumns = {

	@JoinColumn(name =
"user_id", referencedColumnName =
"id") }, inverseJoinColumns = {

	@JoinColumn(name =
"project_id", referencedColumnName
= "id") })
	private Set<Project>
assignedProjects;

	public User() {
	}

	public User(String name, String
surname, String username, String
phone, String position,
			String organization,
String password, List<Authority>
authorities, Set<Project>
assignedProjects) {
		this.name = name;
		this.surname = surname;
		this.username = username;
		this.phone = phone;
		this.position = position;
		this.organization =
organization;
		this.password = password;
		this.assignedProjects =
assignedProjects;
	}

	public User(Long id, String
name, String surname, String
username, String phone, String
position,
			String organization,
String password) {
		this.id = id;
		this.name = name;
		this.surname = surname;
		this.username = username;
		this.phone = phone;
		this.position = position;
		this.organization =
organization;
		this.password = password;
	}

	public User(Long id, String
name, String surname, String
username, String phone, String
position,
			String organization,
String password, Set<Project>
assignedProjects) {
		this.id = id;
```

```
            this.name = name;
            this.surname = surname;
            this.username = username;
            this.phone = phone;
            this.position = position;
            this.organization =
organization;
            this.password = password;
            this.assignedProjects =
assignedProjects;
        }

        public Set<Project>
getAssignedProjects() {
            return assignedProjects;
        }

        public void
setAssignedProjects(Set<Project>
assignedProjects) {
            this.assignedProjects =
assignedProjects;
        }

        @Override
        public Collection<? extends
GrantedAuthority> getAuthorities() {
            // spring demands prefix
"ROLE_" to work properly to all
authorities
            return authorities;
        }

        public void
setAuthorities(List<Authority>
authorities) {
            this.authorities =
authorities;
        }

        @Override
        public boolean
isAccountNonExpired() {
            return true;
        }

        @Override
        public boolean
isAccountNonLocked() {
            return true;
        }

        @Override
        public boolean
isCredentialsNonExpired() {
            return true;
        }

        @Override
        public boolean isEnabled() {
```

```
        return true;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String
name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String
surname) {
        this.surname = surname;
    }

    @Override
    public String getUsername() {
        return username;
    }

    public void setUsername(String
username) {
        this.username = username;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String
phone) {
        this.phone = phone;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String
position) {
        this.position = position;
    }

    public String getOrganization() {
        return organization;
    }
```

```
	public void
setOrganization(String organization) {
		this.organization =
organization;
	}

	@Override
	public String getPassword() {
		return password;
	}

	public void setPassword(String
password) {
		this.password = password;
	}
}
```

AuthCredentialRequest.java

ProjectDto.java

```
package com.yermakov.bugtracker.dto;

import java.time.ZonedDateTime;
import java.util.Set;

import
com.fasterxml.jackson.annotation.JsonI
dentityInfo;
import
com.fasterxml.jackson.annotation.Obje
ctIdGenerators;
import
com.yermakov.bugtracker.domain.User
;

// created for handling last update time
@JsonIdentityInfo(generator =
ObjectIdGenerators.PropertyGenerator.
class, property = "project_id")
```

```
public class ProjectDto {
	private Long id;
	private String title;
	private String description;
	private String note;
	private Set<User>
usersAssigned;


	public ProjectDto() {
		super();
	}

	public ProjectDto(Long id,
String title, String description, String
note, Set<User> usersAssigned) {
		super();
		this.id = id;
		this.title = title;
		this.description =
description;
		this.note = note;
		this.usersAssigned =
usersAssigned;
	}

	public ProjectDto(Long id,
String title, String description, String
note) {
		super();
		this.id = id;
		this.title = title;
		this.description =
description;
		this.note = note;
	}

	public ProjectDto(String title,
String description, String note) {
		super();
		this.title = title;
		this.description =
description;
		this.note = note;
	}

	public ProjectDto(String title,
String description, String note,
Set<User> usersAssigned) {
		super();
		this.title = title;
		this.description =
description;
		this.note = note;
		this.usersAssigned =
usersAssigned;
	}

	public Long getId() {
		return id;
```

```
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getTitle() {
		return title;
	}

	public void setTitle(String title)
{
		this.title = title;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String
description) {
		this.description =
description;
	}

	public String getNote() {
		return note;
	}

	public void setNote(String note)
{
		this.note = note;
	}

	public Set<User>
getUsersAssigned() {
		return usersAssigned;
	}

	public void
setUsersAssigned(Set<User>
usersAssigned) {
		this.usersAssigned =
usersAssigned;
	}

	@Override
	public String toString() {
		return "ProjectDto [id=" +
id + ", title=" + title + ", description="
+ description + ", note=" + note
				+ ",
usersAssigned=" + usersAssigned +
"]";
	}
}
```

AuthorityEnum.java

TicketPriorityEnum.java

TicketStatusEnum.java

TicketTypeEnum.java

JwtFilter.java

```
package
com.yermakov.bugtracker.filter;

import java.io.IOException;
import java.util.List;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.http.HttpHeaders;
import
org.springframework.security.authentic
ation.UsernamePasswordAuthenticatio
nToken;
import
org.springframework.security.core.cont
ext.SecurityContextHolder;
import
org.springframework.security.core.user
details.UserDetails;
import
org.springframework.security.web.auth
entication.WebAuthenticationDetailsSo
urce;
```

```
import
org.springframework.stereotype.Comp
onent;
import
org.springframework.util.StringUtils;
import
org.springframework.web.filter.OncePe
rRequestFilter;

import
com.yermakov.bugtracker.repository.U
serRepository;
import
com.yermakov.bugtracker.util.JwtUtil;

import jakarta.servlet.FilterChain;
import
jakarta.servlet.ServletException;
import
jakarta.servlet.http.HttpServletRequest;
import
jakarta.servlet.http.HttpServletRespons
e;

@Component
public class JwtFilter extends
OncePerRequestFilter {
	@Autowired
	private UserRepository
userRepo;
	@Autowired
	private JwtUtil jwtUtil;

	// spring security checks the
chain of filters or basically goes
through them
	@Override
	protected void
doFilterInternal(HttpServletRequest
request, HttpServletResponse response,
FilterChain chain)
			throws
ServletException, IOException {
		// Get authorization header
and validate
		final String header =
request.getHeader(HttpHeaders.AUTH
ORIZATION);

		// is this is not a json token
then just forget it
		if
(!StringUtils.hasText(header) ||
(StringUtils.hasText(header) &&
!header.startsWith("Bearer "))) {

	chain.doFilter(request,
response);
			return;
```

```
            }

            // here we do have a token and want to split it out
            // example of token: Authorization -> [Bearer], [asdfask.sdfsdfv.sdfsdf]
            final String token = header.split(" ")[1].trim();

            // Get user identity and set it on the spring security context
            UserDetails userDetails = userRepo.findByUsername(jwtUtil.getUsernameFromToken(token)).orElse(null);

            // Get jwt token and validate
            if (!jwtUtil.validateToken(token, userDetails)) {

        chain.doFilter(request, response);
                    return;
            }

        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null,
                                userDetails == null ? List.of() : userDetails.getAuthorities());

        authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

        SecurityContextHolder.getContext().setAuthentication(authentication);
                chain.doFilter(request, response);
        }

}
```

WebSecurityCorsFilter.java

```
package com.yermakov.bugtracker.filter;

import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
```

```
import
org.springframework.context.annotatio
n.Configuration;
import
org.springframework.http.HttpHeaders;
import
org.springframework.http.HttpMethod;
import
org.springframework.web.cors.CorsCo
nfiguration;
import
org.springframework.web.cors.CorsCo
nfigurationSource;
import
org.springframework.web.cors.UrlBase
dCorsConfigurationSource;
import
org.springframework.web.filter.CorsFil
ter;
import
org.springframework.web.servlet.confi
g.annotation.EnableWebMvc;

import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.FilterConfig;
import
jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import
jakarta.servlet.http.HttpServletRespons
e;

import java.io.IOException;
import java.util.Arrays;

@Configuration
@EnableWebMvc
public class WebSecurityCorsFilter {
    private static final Long MAX_AGE
= 3600L;
    private static final int
CORS_FILTER_ORDER = -102;

    @Bean
    public
FilterRegistrationBean<CorsFilter>
corsFilter() {

UrlBasedCorsConfigurationSource
source = new
UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new
CorsConfiguration();
        config.setAllowCredentials(true);

config.addAllowedOrigin("http://localh
ost:5173");
```

```
config.setAllowedHeaders(Arrays.asLi
st(

HttpHeaders.AUTHORIZATION,

HttpHeaders.CONTENT_TYPE,
            "x-auth-token",
            HttpHeaders.ACCEPT));

config.setAllowedMethods(Arrays.asLi
st(
            HttpMethod.GET.name(),
            HttpMethod.POST.name(),
            HttpMethod.PUT.name(),

HttpMethod.DELETE.name(),

HttpMethod.OPTIONS.name(),

HttpMethod.PATCH.name()));
      config.setMaxAge(MAX_AGE);

config.setExposedHeaders(Arrays.asLi
st("x-auth-token"));

source.registerCorsConfiguration("/**"
, config);

FilterRegistrationBean<CorsFilter>
bean = new
FilterRegistrationBean<CorsFilter>(ne
w CorsFilter(source));

      // should be set order to -100
because we need to CorsFilter before
SpringSecurityFilter

bean.setOrder(CORS_FILTER_ORDE
R);
      return bean;
   }
}
```

CommentRepository.java

```
package
com.yermakov.bugtracker.repository;

import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;

import
com.yermakov.bugtracker.domain.Com
ment;
```

```
public interface CommentRepository
extends JpaRepository<Comment,
Long>{

        @Query("select c from
Comment c "
                        + " where c.ticket.id
= :ticketId")
        Set<Comment>
findByTicketId(Long ticketId);
}
```

NotificationRepository.java

```
package
com.yermakov.bugtracker.repository;

import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
import
org.springframework.stereotype.Reposi
tory;

import
com.yermakov.bugtracker.domain.Noti
fication;
import
com.yermakov.bugtracker.domain.Tick
et;

@Repository
public interface NotificationRepository
extends JpaRepository<Notification,
Long> {
        @Query("select n from
Notification n "
                        + " where
n.receiver.id = :receiverId")
        Set<Notification>
findbyReceiverId(Long receiverId);
}
```

ProjectRepository.java

```
package
com.yermakov.bugtracker.repository;

import java.util.List;
import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
```

```
import
com.yermakov.bugtracker.domain.Proj
ect;
import
com.yermakov.bugtracker.domain.User
;

public interface ProjectRepository
extends JpaRepository<Project, Long>
{

	@Query(value = "SELECT p.*
FROM project p "
			+ " INNER JOIN
user_project up "
			+ " ON p.id =
up.project_id "
			+ " WHERE
up.user_id = :userId", nativeQuery =
true)
	Set<Project>
findByUserIdContaining(Long userId);

	Set<Project>
findByUsersAssignedId(Long userId);

	@Query(value = "SELECT u.*
FROM users u "
			+ " INNER JOIN
user_project up "
			+ " ON u.id =
up.user_id "
			+ " WHERE
up.project_id = :projectId",
nativeQuery = true)
	Set<Project>
findByUsersAssignedToProject(Long
projectId);
}
TicketHistoryRepository.java
package
com.yermakov.bugtracker.repository;

import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
import
org.springframework.stereotype.Reposi
tory;

import
com.yermakov.bugtracker.domain.Tick
et;
```

```
import
com.yermakov.bugtracker.domain.Tick
etHistory;

@Repository
public interface
TicketHistoryRepository  extends
JpaRepository<TicketHistory, Long> {

        @Query("select th from
TicketHistory th "
                        + " where
th.ticket.id = :ticketId"
                        + " and th.sender.id
= :senderId"
                        + " and th.id =
:historyId")
        Set<Ticket>
findEntryByTicketIdAndSenderIdandH
istoryId(Long historyId, Long ticketId,
Long senderId);

        @Query("select th from
TicketHistory th "
                        + " where
th.ticket.id = :ticketId"
                        + " and
th.receiver.id = :receiverId"
                        + " and th.id =
:historyId")
        Set<Ticket>
findEntryByTicketIdAndReceiverIdand
HistoryId(Long historyId, Long
ticketId, Long receiverId);
}
```

TicketRepository.java

```
package
com.yermakov.bugtracker.repository;

import java.util.Optional;
import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
import
org.springframework.stereotype.Reposi
tory;

import
com.yermakov.bugtracker.domain.Tick
et;

@Repository
public interface TicketRepository
extends JpaRepository<Ticket, Long>
{
```

```
        @Query("select t from Ticket t "
                + " where
t.project.id = :projectId"
                + " and t.id =
:ticketId")
        Set<Ticket>
findByTicketIdAndProjectId(Long
ticketId, Long projectId);

        @Query("select t from Ticket t "
                + " where t.user.id =
:userId"
                + " and t.id =
:ticketId")
        Set<Ticket>
findByTicketIdAndUserId(Long
ticketId, Long userId);

        @Query("select t from Ticket t "
                + " where t.id =
:ticketId")
        Optional<Ticket>
findById(Long ticketId);

        @Query("select t from Ticket t "
                + " where
t.project.id = :projectId"
                + " and t.user.id =
:userId"
                + " and t.id =
:ticketId")
        Set<Ticket>
findByTicketIdAndUserIdAndProjectI
d(Long ticketId, Long userId,Long
projectId);
}
```

UserRepository.java

```
package
com.yermakov.bugtracker.repository;

import java.util.List;
import java.util.Optional;
import java.util.Set;

import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
import
org.springframework.stereotype.Reposi
tory;

import
com.yermakov.bugtracker.domain.Proj
ect;
```

```
import
com.yermakov.bugtracker.domain.User
;

@Repository
public interface UserRepository
extends JpaRepository<User, Long> {

	// JPA will type the finding
query of the username for us
	Optional<User>
findByUsername(String username);

	Optional<User> findById(Long
id);

	List<User>
findByOrganization(String
organization);

	@Query(value = "SELECT p.*
FROM project p "
			+ " INNER JOIN
user_project up "
			+ " ON p.id =
up.project_id "
			+ " WHERE
up.user_id = :userId", nativeQuery =
true)
	Set<Project>
findByUserIdContaining(Long userId);
}
```

ProjectService.java

```
package
com.yermakov.bugtracker.service;

import java.util.HashSet;
import java.util.List;
import java.util.Optional;
import java.util.Set;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.stereotype.Servic
e;

import
com.yermakov.bugtracker.domain.Proj
ect;
import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.dto.ProjectD
to;
```

```
import
com.yermakov.bugtracker.repository.Pr
ojectRepository;

@Service
public class ProjectService {

	@Autowired
	private ProjectRepository
projectRepository;

	public Project create(User user)
{
		Project project = new
Project();
		Set<User> users = new
HashSet<>();
		users.add(user);

	project.setUsersAssigned(users);
		return
projectRepository.save(project);
	}
	public Set<Project>
findByUserIdContaining(Long userId)
{
		return
projectRepository.findByUserIdContai
ning(userId);
	}
	public Set<Project>
findByUsersAssignedId(Long userId) {
		return
projectRepository.findByUsersAssigne
dId(userId);
	}

	public Set<Project>
findByUsersAssignedToProject(Long
projectId) {
		return
projectRepository.findByUsersAssigne
dToProject(projectId);
	}

	public Optional<Project>
findById(Long projectId) {
		return
projectRepository.findById(projectId);
	}

	public Project save(Project
project) {
		return
projectRepository.save(project);
	}

	public void delete(Long
projectId) {
```

```
        projectRepository.deleteById(pr
ojectId);
        }

        public void deleteAll() {

        projectRepository.deleteAll();
        }
}
```

UserDetailsServiceImpl.java

```
package
com.yermakov.bugtracker.service;

import java.util.Optional;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.security.core.user
details.UserDetails;
import
org.springframework.security.core.user
details.UserDetailsService;
import
org.springframework.security.core.user
details.UsernameNotFoundException;
import
org.springframework.stereotype.Servic
e;

import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.repository.U
serRepository;
import
com.yermakov.bugtracker.util.Custom
PasswordEncoder;

@Service
public class UserDetailsServiceImpl
implements UserDetailsService {

        @Autowired
        private CustomPasswordEncoder
passwordEncoder;

        @Autowired
        private UserRepository
userRepo;

        @Override
        public UserDetails
loadUserByUsername(String
username) {
```

```
            try {
                // 'find by' is a special prefix
                // when JPA sees that prefix
                // it will try to find table with the name that goes after the prefix
                Optional<User> userOpt = userRepo.findByUsername(username);
                return userOpt.orElseThrow(() -> new UsernameNotFoundException("Invalid credentials"));
            } catch (UsernameNotFoundException u) {
                u.printStackTrace();
                return null;
            }
        }
}
```

UserService.java

```
package com.yermakov.bugtracker.service;

import java.util.HashSet;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.yermakov.bugtracker.domain.Project;
import com.yermakov.bugtracker.domain.User;
import com.yermakov.bugtracker.repository.ProjectRepository;
import com.yermakov.bugtracker.repository.UserRepository;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepo;

    @Autowired
```

```
	private ProjectRepository
projectRepo;

	public Optional<User>
finUserByUsername(String username)
{
		return
userRepo.findByUsername(username);
	}

	public User create() {
		return userRepo.save(new
User());
	}

	public List<User>
findByOrganization(User user) {
		return
userRepo.findByOrganization(user.get
Organization());
	}

	public Set<Project>
findByUserIdContaining(User user) {
		return
userRepo.findByUserIdContaining(use
r.getId());
	}

	public User
findByIdAndOrganization(Long
userId, User user) throws Exception {
		Optional<User>
userOptional =
userRepo.findById(userId);
		if (userOptional != null) {
			String org1 =
userOptional.get().getOrganization();
			String org2 =
user.getOrganization();
			if
(org1.compareTo(org2) == 0) {
				return
userOptional.get();
			}
			throw new
Exception("This user is from wrong
organization");
		}
		throw new
Exception("There is no user by that
id");
	}

	public Project
findAssignedProject(Long projectId,
User user) {
		Set<Project>
assignedProjects =
```

```
userRepo.getReferenceById(user.getId(
)).getAssignedProjects();
            Project inputtedProject =
projectRepo.getReferenceById(projectI
d);
            if
(assignedProjects.contains(inputtedProj
ect)) {
                for (Project p :
assignedProjects) {
                  if
(p.equals(inputtedProject))
                    return p;
                }
              }
            return null;
        }

        public User save(User
passedUser) {
            return
userRepo.save(passedUser);
        }

        public void delete(Long userId)
{

        userRepo.deleteById(userId);

        }
        public boolean
addProjectToAssigned(Long userId,
Long projectId) {
            return
projectRepo.findById(projectId).get().g
etUsersAssigned().add(userRepo.getRe
ferenceById(userId));
        }
}
```

AuthorityUtil.java

CustomPasswordEncoder.java

```
package com.yermakov.bugtracker.util;

import
org.springframework.security.crypto.bc
rypt.BCryptPasswordEncoder;
```

```
import
org.springframework.security.crypto.pa
ssword.PasswordEncoder;
import
org.springframework.stereotype.Comp
onent;

@Component
public class CustomPasswordEncoder
{
	private PasswordEncoder
passwordEncoder;

	public
CustomPasswordEncoder() {
		this.passwordEncoder =
new BCryptPasswordEncoder();
	}

	public PasswordEncoder
getPasswordEncoder() {
		return passwordEncoder;
	}
}
```

JwtUtil.java

```
package com.yermakov.bugtracker.util;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

import
org.springframework.beans.factory.ann
otation.Value;
import
org.springframework.security.core.user
details.UserDetails;
import
org.springframework.stereotype.Comp
onent;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import
io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtUtil implements
Serializable {

	private static final long
serialVersionUID =
2077673801184486859L;

	// valid for a month
```

```
	private static final long
JWT_TOKEN_VALIDITY = 30 * 24 *
60 * 60;

	// secret is stored in
application.properties
	@Value("${jwt.secret}")
	private String secret;

	// so, token is unreadable for us,
so these funcs pull some info for us
	public String
getUsernameFromToken(String token)
{
		return
getClaimFromToken(token,
Claims::getSubject);
	}

	public Date
getIssuedAtDateFromToken(String
token) {
		return
getClaimFromToken(token,
Claims::getIssuedAt);
	}

	public Date
getExpirationDateFromToken(String
token) {
		return
getClaimFromToken(token,
Claims::getExpiration);
	}

	// for adding your custom claims
	// or adding some extra own data
into token
	public <T> T
getClaimFromToken(String token,
Function<Claims, T> claimsResolver)
{
		final Claims claims =
getAllClaimsFromToken(token);
		return
claimsResolver.apply(claims);
	}

	private Claims
getAllClaimsFromToken(String token)
{
		return
Jwts.parser().setSigningKey(secret).par
seClaimsJws(token).getBody();
	}

	private Boolean
isTokenExpired(String token) {
		final Date expiration =
getExpirationDateFromToken(token);
```

```
        return
expiration.before(new Date());
    }

    private Boolean
ignoreTokenExpiration(String token) {
        // here you specify tokens,
for that the expiration is ignored
        return false;
    }

    public String
generateToken(UserDetails
userDetails) {
        Map<String, Object>
claims = new HashMap<>();

        claims.put("authorities",

    userDetails.getAuthorities().strea
m().map(auth ->
auth.getAuthority()).collect(Collectors.
toList()));
        // claims.put("authorities",
userDetails.getAuthorities());

        return
doGenerateToken(claims,
userDetails.getUsername());
    }

    private String
doGenerateToken(Map<String,
Object> claims, String subject) {
        return
Jwts.builder().setClaims(claims).setSub
ject(subject).setIssuedAt(new
Date(System.currentTimeMillis()))

    .setExpiration(new
Date(System.currentTimeMillis() +
JWT_TOKEN_VALIDITY * 1000))

    .signWith(SignatureAlgorithm.H
S512, secret).compact();
    }

    public Boolean
canTokenBeRefreshed(String token) {
        return
(!isTokenExpired(token) ||
ignoreTokenExpiration(token));
    }

    public Boolean
validateToken(String token,
UserDetails userDetails) {
        final String username =
getUsernameFromToken(token);
```

```
        return (userDetails != null
&&
username.equals(userDetails.getUserna
me()) && !isTokenExpired(token));
    }
}
```

AuthContoller.java

```
package
com.yermakov.bugtracker.web;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.http.HttpHeaders;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEnt
ity;
import
org.springframework.security.authentic
ation.AuthenticationManager;
import
org.springframework.security.authentic
ation.BadCredentialsException;
import
org.springframework.security.authentic
ation.UsernamePasswordAuthenticatio
nToken;
import
org.springframework.security.core.Aut
hentication;
import
org.springframework.security.core.ann
otation.AuthenticationPrincipal;
import
org.springframework.web.bind.annotati
on.CrossOrigin;
import
org.springframework.web.bind.annotati
on.GetMapping;
import
org.springframework.web.bind.annotati
on.PostMapping;
import
org.springframework.web.bind.annotati
on.RequestBody;
import
org.springframework.web.bind.annotati
on.RequestMapping;
import
org.springframework.web.bind.annotati
on.RequestParam;
import
org.springframework.web.bind.annotati
on.RestController;
```

```
import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.dto.AuthCre
dentialRequest;
import
com.yermakov.bugtracker.util.JwtUtil;

import
io.jsonwebtoken.ExpiredJwtException;

//Controllers listen to the end-points
//so to speak listen to all the posts,
reads, deletes, updates and so on
@RestController
@RequestMapping("/api/auth")
public class AuthController {
	@Autowired
	private AuthenticationManager
authenticationManager;
	@Autowired
	private JwtUtil jwtUtil;

	// @RequestBody for binding
this thing with body in postman
	@PostMapping("login")
	@CrossOrigin(origins = "*")
	public ResponseEntity<?>
login(@RequestBody
AuthCredentialRequest request) {
		try {
			Authentication
authenticate =
authenticationManager.authenticate(
				new
UsernamePasswordAuthenticationToke
n(request.getUsername(),
request.getPassword()));

			User user = (User)
authenticate.getPrincipal();

	user.setPassword(null);
			return
ResponseEntity.ok().header(HttpHeade
rs.AUTHORIZATION,
jwtUtil.generateToken(user)).body(user
);
		} catch
(BadCredentialsException ex) {
			return
ResponseEntity.status(HttpStatus.UNA
UTHORIZED).build();
		}
	}
```

```
        // ResponseEntity returns
response
        //
localhost:8081/api/auth/validate?token
=token_name
        // @RequestParam ==
?token=token_name
        @GetMapping("/validate")
        @CrossOrigin(origins = "*")
        public ResponseEntity<?>
validateToken(@RequestParam String
token, @AuthenticationPrincipal User
user) {
                try {
                        Boolean
isTokenValid =
jwtUtil.validateToken(token, user);
                        return
ResponseEntity.ok(isTokenValid);
                } catch
(ExpiredJwtException e) {
                        return
ResponseEntity.ok(false);
                }
        }
}
```

CommentContoller.java

```
package
com.yermakov.bugtracker.web;

import java.util.Set;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEnt
ity;
import
org.springframework.security.core.ann
otation.AuthenticationPrincipal;
import
org.springframework.web.bind.annotati
on.DeleteMapping;
import
org.springframework.web.bind.annotati
on.GetMapping;
import
org.springframework.web.bind.annotati
on.PathVariable;
import
org.springframework.web.bind.annotati
on.PostMapping;
import
org.springframework.web.bind.annotati
on.PutMapping;
```

```
import
org.springframework.web.bind.annotati
on.RequestBody;
import
org.springframework.web.bind.annotati
on.RequestMapping;
import
org.springframework.web.bind.annotati
on.RequestParam;
import
org.springframework.web.bind.annotati
on.RestController;

import
com.yermakov.bugtracker.domain.Com
ment;
import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.dto.Commen
tDto;
import
com.yermakov.bugtracker.service.Com
mentService;

//1) Controllers should only depend on
services and never on repositories
//2) Controllers listen to the end-points
//so to speak listen to all the posts,
reads, deletes, updates and so on
//3) Controllers don't talk to
Repositories, they talk to Services
//@RestController returns Data and
@Controller returns Views
//@RequestMapping("/api/comments")
is a link that refers to Comments
@RestController
@RequestMapping("/api/comments")
public class CommentController {

    @Autowired CommentService
commentService;

    // DTO – Data Transfer Object
    // created it because front-end
sends us only 3 params
    // when we need a few more to
fill a Comment object so we can put it
to the database
    // @AuthenticationPrincipal
User user – gets User object
    // @RequestBody means smth is
inside the body of the request
    @PostMapping("")
    public
ResponseEntity<Comment>
createComment(@RequestBody
```

```
CommentDto commentDto,
@AuthenticationPrincipal User user) {
                Comment comment =
commentService.save(commentDto,use
r);
                return
ResponseEntity.ok(comment);
        }
        //
@RequestMapping("/api/comments") +
@PutMapping("{commentId}") ==
/comments/{commentId}
        @PutMapping("{commentId}")
        public
ResponseEntity<Comment>
updateComment(@RequestBody
CommentDto commentDto,
@AuthenticationPrincipal User user) {
                Comment comment =
commentService.save(commentDto,use
r);
                return
ResponseEntity.ok(comment);
        }

        // @RequestParam is part of url
itself rather than body
        @GetMapping("")
        public
ResponseEntity<Set<Comment>>
getCommentsByAssignment(@Reques
tParam Long ticketId) {
                Set<Comment> comments
=
commentService.getCommentsByTick
etId(ticketId);
                return
ResponseEntity.ok(comments);
        }

        // deleting (for git)
        @DeleteMapping("{commentId
}")
 public ResponseEntity<?>
deleteComment (@PathVariable Long
commentId) {
    try {

commentService.delete(commentId);
        return
ResponseEntity.ok("Comment
deleted");
    } catch (Exception e) {
        e.printStackTrace();
        return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
    }
 }
}
```

NotificationContoller.java

```
package com.yermakov.bugtracker.web;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.yermakov.bugtracker.domain.Notification;
import com.yermakov.bugtracker.domain.User;
import com.yermakov.bugtracker.service.NotificationService;

@RestController
@RequestMapping("/api/notifications")
public class NotificationController {

    @Autowired
```

```
        NotificationService
notificationService;

        @PostMapping("")
        public
ResponseEntity<Notification>
createNotification(@RequestBody
Notification notification,

        @AuthenticationPrincipal User
user) {
                Notification
notificationLocal =
notificationService.save(notification,
user);
                return
ResponseEntity.ok(notificationLocal);
        }

        // @RequestParam is part of url
itself rather than body
        @GetMapping("")
        public
ResponseEntity<Set<Notification>>
getNotificationsByReceiver(@Request
Param Long receiverId) {
                Set<Notification>
notifications =
notificationService.getNotificationsBy
ReceiverId(receiverId);
                return
ResponseEntity.ok(notifications);
        }

        // deleting (for git)
        @DeleteMapping("{notificationI
d}")
    public ResponseEntity<?>
deleteNotification (@PathVariable
Long notificationId) {
        try {

notificationService.delete(notificationI
d);
            return
ResponseEntity.ok("Notification
deleted");
        } catch (Exception e) {
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
        }
    }
}
ProjectContoller.java
package
com.yermakov.bugtracker.web;

import java.util.List;
```

```
import java.util.Optional;
import java.util.Set;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.data.jpa.repositor
y.JpaRepository;
import
org.springframework.data.jpa.repositor
y.Query;
import
org.springframework.data.repository.qu
ery.Param;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEnt
ity;
import
org.springframework.security.core.ann
otation.AuthenticationPrincipal;
import
org.springframework.web.bind.annotati
on.DeleteMapping;
import
org.springframework.web.bind.annotati
on.GetMapping;
import
org.springframework.web.bind.annotati
on.PathVariable;
import
org.springframework.web.bind.annotati
on.PostMapping;
import
org.springframework.web.bind.annotati
on.PutMapping;
import
org.springframework.web.bind.annotati
on.RequestBody;
import
org.springframework.web.bind.annotati
on.RequestMapping;
import
org.springframework.web.bind.annotati
on.RestController;

import
com.fasterxml.jackson.databind.Object
Mapper;
import
com.fasterxml.jackson.databind.node.O
bjectNode;
import
com.yermakov.bugtracker.domain.Proj
ect;
```

```
import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.dto.ProjectD
to;
import
com.yermakov.bugtracker.service.Proje
ctService;
import
com.yermakov.bugtracker.service.User
Service;

//1) Controllers should only depend on
services and never on repositories
//2) Controllers listen to the end-points
//so to speak listen to all the posts,
reads, deletes, updates and so on
//3) Controllers don't talk to
Repositories, they talk to Services
//and only than Services talk to
Repositories
//@RestController returns Data and
@Controller returns Views
@RestController
@RequestMapping("/api/projects")
public class ProjectController {

    @Autowired
    private ProjectService
projectService;

    @PostMapping("")
    public ResponseEntity<Project>
createProject(@AuthenticationPrincipa
l User user) {
        Project newProject =
projectService.create(user);
        return
ResponseEntity.ok(newProject);
    }

    @GetMapping("")
    public
ResponseEntity<Set<Project>>
getUserAssignedProjects(@Authentica
tionPrincipal User user) {
        Set<Project>
projectsByUser =
projectService.findByUsersAssignedId
(user.getId());
        return
ResponseEntity.ok(projectsByUser);
    }

    // path variable / dynamic
variable
    @GetMapping("/{projectId}")
```

```
    public ResponseEntity<Project>
getProject(@PathVariable Long
projectId, @AuthenticationPrincipal
User user) {
        Optional<Project>
projectOptional =
projectService.findById(projectId);
        return
ResponseEntity.ok(projectOptional.orE
lse(new Project()));
    }

    @GetMapping("/{projectId}/use
rs")
    public
ResponseEntity<Set<Project>>
getUsersAssignedToProject(@PathVar
iable Long projectId,
@AuthenticationPrincipal User user) {
        Set<Project>
usersByProject =
projectService.findByUsersAssignedTo
Project(projectId);
        return
ResponseEntity.ok(usersByProject);
    }

    // @RequestBody is a project
we're getting from the front-end
    @PutMapping("/{projectId}")
    public ResponseEntity<Project>
updateProject(@PathVariable Long
projectId, @RequestBody Project
project,

    @AuthenticationPrincipal User
user) {
        Project updatedProject =
projectService.save(project);
        return
ResponseEntity.ok(updatedProject);
    }

    @DeleteMapping("/{projectId}"
)
    public ResponseEntity<?>
deleteProject(@PathVariable Long
projectId) {
        try {

    projectService.delete(projectId);
            return
ResponseEntity.ok("Project is
deleted");
        } catch (Exception e) {
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
        }
```

```
    }

    @DeleteMapping("")
    public ResponseEntity<?>
deleteProjects() {
        try {

    projectService.deleteAll();
            return
ResponseEntity.ok("All projects are
deleted");
        } catch (Exception e) {
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
        }
    }
}
```

TicketContoller.java

```
package
com.yermakov.bugtracker.web;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEnt
ity;
import
org.springframework.web.bind.annotati
on.DeleteMapping;
import
org.springframework.web.bind.annotati
on.GetMapping;
import
org.springframework.web.bind.annotati
on.PathVariable;
import
org.springframework.web.bind.annotati
on.PostMapping;
import
org.springframework.web.bind.annotati
on.PutMapping;
import
org.springframework.web.bind.annotati
on.RequestMapping;
import
org.springframework.web.bind.annotati
on.RestController;

import
com.yermakov.bugtracker.service.Tick
etService;

@RestController
```

```
@RequestMapping("/api/tickets")
public class TicketController {

	@Autowired
	TicketService TicketService;

	@PostMapping("/projects/{projectId}")
	public ResponseEntity<?> postUnassignedTicket(@PathVariable Long projectId) {
		return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
	}

	@PostMapping("/projects/{projectId}/users/{userId}")
	public ResponseEntity<?> postAssignedTicket(@PathVariable Long projectId, @PathVariable Long userId) {
		return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
	}

	@GetMapping("/{ticketId}/projects/{projectId}")
	public ResponseEntity<?> getTicketFromProject(@PathVariable Long ticketId, @PathVariable Long projectId) {
		return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
	}

	@GetMapping("/{ticketId}/projects/{projectId}/users/{userId}")
	public ResponseEntity<?> getTicketFromProjectFromUser(@PathVariable Long ticketId, @PathVariable Long projectId, @PathVariable Long userId) {
		return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
	}

	@GetMapping("/{ticketId}/users/{userId}")
	public ResponseEntity<?> getTicketFromUser(@PathVariable Long ticketId, @PathVariable Long userId) {
		return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

```
    }

    @PutMapping("/{ticketId}/projects/{projectId}")
    public ResponseEntity<?> putTicketFromProject(@PathVariable Long ticketId, @PathVariable Long projectId) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    @PutMapping("/{ticketId}/projects/{projectId}/users/{userId}")
    public ResponseEntity<?> putTicketFromProjectFromUser(@PathVariable Long ticketId, @PathVariable Long projectId, @PathVariable Long userId) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    @PutMapping("/{ticketId}/users/{userId}")
    public ResponseEntity<?> putTicketFromUser(@PathVariable Long ticketId, @PathVariable Long userId) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    @DeleteMapping("")
    public ResponseEntity<?> deleteTickets() {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    @DeleteMapping("/{ticketId}")
    public ResponseEntity<?> deleteTicket(@PathVariable Long userId, @PathVariable Long projectId) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```

UserContoller.java

```
package com.yermakov.bugtracker.web;

import java.util.List;
```

```
import java.util.Optional;
import java.util.Set;

import
org.springframework.beans.factory.ann
otation.Autowired;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEnt
ity;
import
org.springframework.security.core.ann
otation.AuthenticationPrincipal;
import
org.springframework.web.bind.annotati
on.DeleteMapping;
import
org.springframework.web.bind.annotati
on.GetMapping;
import
org.springframework.web.bind.annotati
on.PathVariable;
import
org.springframework.web.bind.annotati
on.PostMapping;
import
org.springframework.web.bind.annotati
on.PutMapping;
import
org.springframework.web.bind.annotati
on.RequestBody;
import
org.springframework.web.bind.annotati
on.RequestMapping;
import
org.springframework.web.bind.annotati
on.RestController;

import
com.yermakov.bugtracker.domain.Proj
ect;
import
com.yermakov.bugtracker.domain.User
;
import
com.yermakov.bugtracker.service.User
Service;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("")
```

```
    public ResponseEntity<User>
createUser(@AuthenticationPrincipal
User user) {
        User newUser =
userService.create();
        return
ResponseEntity.ok(newUser);
    }

    @GetMapping("")
    // maybe add JsonIgnore to
UsersAssigned in POJO class
    public
ResponseEntity<List<User>>
getAllFromOrganization(@Authenticat
ionPrincipal User user) {
        List<User>
organizationUsers =
userService.findByOrganization(user);
        return
ResponseEntity.ok(organizationUsers);
    }

    @GetMapping("/projects")
    public
ResponseEntity<Set<Project>>
getUserAssignedProjects(@Authentica
tionPrincipal User user) {
        Set<Project>
projectsByAuthUser =
userService.findByUserIdContaining(u
ser);
        return
ResponseEntity.ok(projectsByAuthUse
r);
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User>
getUser(@PathVariable Long userId,
@AuthenticationPrincipal User user) {
        try {
            User localUser =
userService.findByIdAndOrganization(
userId, user);
            return
ResponseEntity.ok(localUser);
        } catch (Exception e) {
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/projects/{proje
ctId}")
    public ResponseEntity<Project>
getAssignedProject(@PathVariable
Long projectId,
```

```
        @AuthenticationPrincipal User
user) {
                Project assignedProject =
userService.findAssignedProject(projec
tId, user);
                return
ResponseEntity.ok(assignedProject);
        }

        // in front-end
        @PutMapping("/{userId}")
        public ResponseEntity<User>
updateUser(@PathVariable Long
projectId, @RequestBody User
passedUser) {
                User updatedUser =
userService.save(passedUser);
                return
ResponseEntity.ok(updatedUser);
        }

        @PutMapping("/{userId}/projec
ts/{projectId}")
        public ResponseEntity<?>
updateUser(@PathVariable Long
userId, @PathVariable Long projectId)
{
                if
(userService.addProjectToAssigned(us
erId, projectId)) {
                        return
ResponseEntity.ok().build();
                }
                return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
        }

        @DeleteMapping("/{userId}")
        public ResponseEntity<?>
deleteUser(@PathVariable Long
userId) {
                try {

        userService.delete(userId);
                        return
ResponseEntity.ok("Project is
deleted");
                } catch (Exception e) {
                        e.printStackTrace();
                        return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
                }
        }
```

```
        @DeleteMapping("/{userId}/pro
jects/{projectId}")
        public ResponseEntity<?>
deleteUserFromProject(@PathVariable
Long userId, @PathVariable Long
projectId) {
                try {

        userService.deleteUserFromProj
ect(userId, projectId);
                        return
ResponseEntity.ok("Project is
deleted");
                } catch (Exception e) {
                        e.printStackTrace();
                        return
ResponseEntity.status(HttpStatus.INTE
RNAL_SERVER_ERROR).build();
                }
        }
}
```

Pom.xml

```
-– lookup parent from repository -->
```

Vite.config.ts

Main.tsx

Index.css

App.tsx

– here it would work, if i just change

EmployeeInfoCard.tsx

ProjectInfoCard.tsx

TicketInfoCard.tsx

MyPieChart.tsx

RegesterWith.tsx

YetToBeDoneAlert.tsx

HomeNavbar.tsx

Pagination.tsx

MaybeToShowNavbar.tsx

– siblings, 1);

– 2;

– rightItemsCount + 1, totalPage + 1);

ReturnPaginationRange.tsx

SelectLimit.tsx

Table.tsx

TableWrapper.tsx

– 1);

TicketDropdown.tsx

DataGetter.tsx

– 1) * limit; i < page * limit &&

Dashboard.tsx

Notifications.tsx

Organization.tsx

Tickets.tsx

Login.tsx

Project.tsx

NotFound.tsx

About.tsx

Profile.tsx

FetchService.jsx

PrivateRoute.jsx

PageNaming.tsx

UseLocalStorage.jsx

UseInterval.jsx

UserProvider.jsx