ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра програмної інженерії

Пояснювальна записка

До бакалаврської роботи

На ступінь вищої освіти бакалавр

на тему: **«Автоматизована система оптимізації розподілу та аналізу задач проєкту в управлінні процесом розробки програмних продуктів»**

Виконав: студент 5 курсу, групи ППЗ–51
спеціальності
123 Програмна інженерія
(шифр і назва спеціальності)
Гнатюк А.П
(прізвище та ініціали)
Керівник _Трінтіна С.А._____
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2021

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Автоматизована система оптимізації розподілу та аналізу задач проєкту в управлінні процесом розробки програмних продуктів»: 80 сторінок, 55 рисунків, 4 таблиць, 12 використаних джерел, 1 додаток.

Об'єкт дослідження - система управління процесом розробки програмного забезпечення.

Мета дипломної роботи - зменшення навантаження на менеджера проєкту та підвищення продуктивності розробки програмного забезпечення шляхом автоматизації процесу розподілу проєктних робіт.

Метод дослідження – Розробка автоматизованої системи для оптимізації розподілу та аналізу проєктних завдань в управлінні розробкою програмного забезпечення.

У процесі роботи було проведено ретельний аналіз процесу управління розробкою програмного забезпечення.

Результати роботи можуть бути використані в процесі управління розробкою програмного забезпечення в компаніях у галузі інформаційних технологій, що використовує гнучкі методи розробки програмного забезпечення.

Розробка та дослідження проводилися під управлінням Mac OS Catalina. Програма була розроблена у середовищі Visual studio code, мовою програмування Javascript.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення

ІС – інформаційна система

СУК – система управління контентом

JS – мова програмування «JavaScript»

MVC – методологія проєктування вебдодатків

ОС – операційна система

ООП – об’єктно-орієнтоване програмування

СУБД - Система управління базами даних;

БД - База даних;

АРМ - Автоматизоване робоче місце;

ЕОМ – Електронна обчислювальна машина.

ІІР – Ітеративна та інкрементальна розробка.

ВСТУП

У сучасному суспільстві інформація та знання стали джерелом цінності, а здатність швидко передавати інформацію є дуже важливою умовою успіху як для комерційних або державних структур, так і для людини.

На цей час сфера інформаційних технологій є одним із головних драйверів світової економіки, ставши каталізатором для тектонічних змін і трансформацій у багатьох інших індустріях. За підсумками 2021 року обсяги загальносвітових витрат на продукти та послуги у сфері інформаційних технологій складуть 3.7 трлн. доларів та продемонструють зростання на 6.2% у порівнянні з минулим роком, що навіть вище за темпи росту світового ВВП. Разом із загальним ростом ринку, збільшується і частка IT-аутсорсингу (послуги з розробки програмного забезпечення, створення інфраструктурних рішень для замовників). Подальший розвиток сегменту стимулюється стабільно високим попитом, який і надалі буде зростати завдяки «цифровій трансформації», що все активніше проникає як в приватний, так і у державний сектори.

Як й інші традиційні інженерні дисципліни, розробка програмного забезпечення має справу з проблемами якості, вартості та надійності. Складність програмного забезпечення порівнюється зі складністю найбільш складних з сучасних машин, таких як літаки.

Процес розробки програмного забезпечення потребує кваліфікованого і ефективного управління. Чим більший проєкт, тим складніше ним керувати.

Дипломний проєкт присвячений розробці комплексу задач, які стоять перед менеджером програмних продуктів та створення програмного забезпечення для оптимізації розподілу та аналізу задач проєкту в управлінні процесом розробки програмних продуктів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА

1.1 Опис предметного середовища

Розробка сучасних інформаційних систем - це багатоетапний процес зі специфічними технічними та організаційними заходами. Ускладнюється технологія виробництва, збільшуються обсяги та урізноманітнюються інформаційні потоки, що циркулюють в суспільстві, вимоги до безпеки функціонування систем, їх якості і т.д. Своєрідною відповіддю на це є твердження концепції проєкту в області розробки систем різного призначення.

У практичній діяльності та літературі під проєктом розуміється сукупність дій (заходів), спрямованих на досягнення мети створення складної системи з заданими характеристиками якості в обмежені терміни і на обмеженій множині ресурсів.

Процесом досягнення певної конкретної мети необхідно управляти, тому що результат і, відповідно, успіх не може прийти мимовільно. Управління проєктами - це особливий вид діяльності, що включає планування, моніторинг та коригування з використанням сучасних методів управління. Зрозуміло, що основою ефективного управління проєктом є план, в якому:

- дії (заходи, роботи) впорядковані по результатам і термінам їх досягнення;
- Існує чіткий зв’язок між характеристиками системи та ресурсами.
- Процес розробки складається з багатьох підпроцесів або дисциплін, деякі з яких наведені нижче.
- Аналіз вимог
- Проєктування
- Розробка
- Тестування
- Системна інтеграція
- Впровадження
- Супровід

1.2. Моделі розробки програмного забезпечення

Водоспадна модель життєвого циклу (англ. Waterfall model) описана Уінстоном Ройсом в статті "Managing the Development of Large Software Systems" у 1970 році. Вона передбачає поетапне виконання усіх фаз проєкту конкретним чином. Перехід на наступну фазу означає остаточне завершення попередньої стадії.

Вимоги, визначені на етапі їх формування , суворо документуються у виді технічного завдання і встановлюються на весь час розробки проєкту. Кожна стадія закінчується випуском повного набору документації, достатнього для того, щоб інша команда могла продовжити розробку.

Плюсом є узгоджена та повна документація на всіх етапах і легке визначення термінів і витрат на проєкт.

Недоліком даної моделі є перехід від однієї стадії проєкту до іншої, що забезпечує повну коректність результатів (виходу) попередньої стадії. Однак неточності певної вимоги або невірна його інтерпретація в результаті призводить до того, що необхідно повертатись до ранньої фази проєкту, і необхідний процес обробки не лише заважає проєктній команді вчасно виконати план, але також призводить до підвищення витрат та може призвести до завершення проєкту в оригінальній формі.

На думку сучасних експертів, основним непорозумінням автора моделі «водоспад» є таке припущення: проєкт повинен пройти весь процес одноразово, спроєктована архітектура добре структурована і проста у використанні, а помилки впровадження легко усунути за допомогою тестування. Модель передбачає, що всі помилки будуть зосереджені в реалізації, тому процес усунення цих помилок відбуватиметься рівномірно під час тестування компонентів і системи. Таким чином, модель «водоспад» для значних проєктів нереальна і може бути ефективно використана лише для створення невеликих систем.

Модель ітеративної та інкрементальної розробки є альтернативою послідовної моделі.

Ця модель передбачає ділення проєкту на низку ітерацій, кожна ітерація подібна до «невеликого проєкту», порівняно з усім проєктом, включаючи всі

процеси розробки, що використовуються для створення менших функціональних блоків. Метою усіх ітерацій є отримання робочої версії програмної системи, яка включає всі функції, визначені інтегрованим змістом попередньої та поточної ітерацій. Результат остаточної ітерації містить усі необхідні функції продукту. Отже, із завершенням кожної ітерації продукт буде рости (збільшуватися) від нього до своїх можливостей і, отже, поступово розвиватися. У цьому випадку ітерація, інкремент та еволюція використовують різні слова для вираження одного і того ж змісту з дещо іншої точки зору.

За висловом Т. Гілбі, «еволюція - прийом, призначений для створення видимості стабільності. Шанси успішного створення складної системи будуть максимальними, якщо вона реалізується в серії невеликих кроків і якщо кожен крок містить в собі чітко визначений успіх, а також можливість «відкату» до попереднього успішному етапу в разі невдачі. Перед тим, як пустити в справу всі ресурси, призначені для створення системи, розробник має можливість отримувати з реального світу сигнали зворотного зв'язку і виправляти можливі помилки в проєкті».

Метод IIR також має свої недоліки, що насправді є іншою стороною переваг. По-перше, протягом тривалого часу бракувало всебічного розуміння можливостей та обмежень проєкту. По-друге, в ітераційному процесі необхідно відкинути частину раніше зробленої роботи. По-третє, чесність фахівців на роботі все ще знижується, що психологічно очевидно, оскільки вони постійно відчувають, що "все ще можна переробити та вдосконалити в майбутньому".

Спіральну модель розробив Баррі Бем у середині 1980-х. Він заснований на класичному циклі PDCA Демінга. При її використанні програмне забезпечення буде створено за допомогою декількох ітерацій (спіралей) прототипування.

Усі ітерації відповідають за створення програмного фрагмента або версії, яка визначає цілі та характеристики проєкту, оцінює якість результатів та планує наступну ітерацію.

У кожній ітерації буде оцінено:

- ризик перевищення термінів та витрат проєкту;

- потреба у виконанні ще однієї ітерації;
- ступінь повноти і точності розуміння вимог до системи;
- доцільність припинення проєкту.

Відмінною властивістю спіральної моделі є приділення особливої уваги ризикам, що впливає на життєвий цикл та організацію контрольної точки. Найбільш поширеними ризиками є:

- Не вистачає фахівців.
- Нереалістичні терміни та бюджет.
- Впровадження невідповідних функцій.
- Розробка незручного користувальницького інтерфейсу.
- Перфекціонізм, лишня оптимізація і вдосконалення деталей.
- Постійні зміни.
- Відсутність інформації для визначення зовнішніх компонентів, що беруть участь у системному середовищі або інтеграції.
- Недоліки роботи, що виконується зовнішніми (щодо проєкту) ресурсами.
- Одержувана система є недостатньо продуктивною.
- Розрив в кваліфікації фахівців різних областей.

1.3 Методологія гнучкої розробки програмного забезпечення Agile

Гнучка методологія розробки програмного забезпечення стає найпоширенішою технікою управління проєктами: організації шукають шляхи підвищення гнучкості, і 71% організацій повідомили, що використовують ці методи для своїх проєктів. Найпоширенішими спритними методами є Scrum і Kanban.

Методологія гнучкої розробки (Agile software development) – це маніфест, що має основні цінності та принципи, на яких базуються методи управління проєктами, таким чином вирішуючи традиційні проблеми управління проєктами. Agile ідеально підходить для інноваційних проєктів. Він не підходить для інновацій. Ці методи передбачають ітеративну розробку, при якій вимоги

споживачів регулярно оновлюються та реалізуються за допомогою самоорганізуючої команди експертів з різними профілями.

Створення спритних методів суперечить традиційному лінійному методу "водоспаду", що означає, що ітеративна розробка програмного забезпечення може мінімізувати ризик. Справа в тому, що робота з гнучким підходом складається із серії коротких циклів (ітерацій), що тривають 1-2 тижні. Кожна ітерація включає етапи планування, аналізу вимог, проєктування, розробки, випробування та документації. В кінці кожної ітерації команда надаватиме клієнтам "відчутні" результати, такі як початкова версія продукту або функції, які можна переглянути, оцінити, протестувати, а потім змінити або відкоригувати. На основі проведеної роботи команда підсумувала та зібрала нові вимоги та скоригувала план розробки продукту на цій основі.

Декларація про гнучку методологію визначає чотири основні цінності і на цій основі визначає 12 принципів методології.

Цінності:

- Люди та взаємодія важливіші за процеси та інструменти.
- Ефективний продукт важливіший за вичерпну документацію.
- Співпраця з клієнтами важливіша, ніж досягнення умов контракту.
- Порівняно з попереднім планом, важливіше бути готовим до змін.

Основоположні принципи маніфесту Agile:

1. Головним пріоритетом є задоволення потреб клієнта.

2. На будь-якому етапі розвитку вітаються зміни вимог. Ці зміни забезпечують клієнтів конкурентною перевагою.

3. Робочий продукт повинен бути випущеним якомога частіше.

4. У всьому проєкті розробники та замовники повинні працювати разом щодня.

5. У проєкті повинні брати участь мотивовані експерти. Для цього потрібно створити умови, надати підтримку та довіру.

6. Для ефективного спілкування з командою та всередині команди доречно безпосереднє спілкування.

7. Основним показником прогресу є результати роботи.

8. Постійний і стійкий процес розвитку.

9. Орієнтація на технічну досконалість та якість дизайну збільшує гнучкість проєкту.

10. Скорочення непотрібної роботи.

11. Тільки самоорганізована команда може надати найкращу структуру системи та технічні рішення.

12. Команда повинна систематично аналізувати можливі шляхи підвищення ефективності та відповідно коригувати методи роботи.

Термін "гнучка методологія розвитку" слід розуміти як метод, заснований на цій декларації або структурі. Існує багато фреймворкових методів, заснованих на Agile, таких як: Scrum, Extreme Programming, FDD, DSDM.

Scrum - це методологія, яка допомагає командам вести спільну роботу. Як спортивна команда готується до вирішальної гри (до слова, scrum - англ. «Сутичка», елемент гри в регбі), так і команда співробітників компанії повинна зробити висновки з отриманого досвіду, освоювати принципи самоорганізації працюючи над вирішенням проблеми, аналізувати свої успіхи і провали, щоб постійно вдосконалюватися. Scrum сприяє цьому.

Методологію Scrum найчастіше застосовують команди розробників додатків, але принципи та досвід її використання можна застосувати до командної роботи будь-якого роду. Це одна з причин такої популярності. Scrum часто представляють як платформу для управління проєктами за методологією agile. Учасники команди Scrum проводять збори, використовують спеціальні інструменти і приймають на себе особливі ролі, щоб організувати роботу і керувати нею.

Kanban — це метод управління розробкою програмного забезпечення з наголосом з доставкою якраз вчасно та униканні перевантаження членів команди. При цьому підході процес від опису задачі до доставки результатів її виконання користувачу, наочно показується учасникам процесу, і члени команди можуть витягувати роботу з черги.

Kanban це підхід до інкрементної, еволюційної зміни процесів і систем в організації. Він використовує обмеження роботи, що знаходиться в процесі виконання, як ключовий механізм для виявлення проблем в роботі системи та стимулює співпрацю для постійного її вдосконалення. Корені знаходяться в чотирьох базових принципах:

- Почніть з того, що ви маєте зараз. Метод Канбан не описує конкретний набір ролей чи кроків процесу. Він стартує з ролями і процесами, що є у вас зараз, і стимулює постійні інкрементні та еволюційні зміни в системі.
- Погодьтесь домагатись інкрементних, еволюційних змін. Організація (або команда) повинна погодитися з тим, що постійні зміни - це спосіб вдосконалення системи та її впровадження. Глобальні зміни, здається, ефективніші, але через опір і страх організації ризик невдачі стає більшим. Канбан заохочує постійні незначні зміни до поточної системи.
- Поважайте поточні процеси, ролі, обов'язки та посади. Можливо, в організації є деякі задовільні елементи роботи, які слід підтримувати. Канбан намагається отримати більшу підтримку, погоджуючись поважати поточні ролі, обов'язки та посади, тим самим уникаючи проблем.
- Лідерство на всіх рівнях. Схвалено всі рівні керівницьких дій від окремих працівників до вищих керівників.

Через великий обсяг планованих дій, складності методів розподілу та оптимізації плану проводити якісне планування, здійснювати ефективний контроль і управління розробкою проєкту без використання спеціальних програмних засобів неможливо.

1.4. Огляд існуючих систем для управління процесом розробки програмних продуктів

У нинішній час, при створенні та веденні проєкту не обійтися без систем підтримки проєкту і відслідковування помилок, саме ці системи дозволяють оперативно приймати вірні рішення щодо розвитку проєкту і відстеження дії програмістів.

Ці системи дозволяють мінімізувати «паперову роботу» і спростити взаємодію між усіма членами команди, де б вони фізично не знаходилися.

Існує безліч різних систем які реалізують шуканий функціонал найпопулярнішими системами є:

- JIRA,
- Redmine,
- Bugzilla.

Jira - комерційна система для управління проєктами і відслідковування помилок, призначена для організації взаємодії з користувачами. Розроблено компанією Atlassian, є одним з двох її основних продуктів (поряд з вікі-системою Confluence). має велику кількість параметрів конфігурації: ви можете визначити окремий тип завдання для кожної програми, включаючи власний робочий процес, набір статусів, одним або декілька видами уявлення (англ. Screens). Крім того, за допомогою «схем» можна визначити для кожного індивідуального проєкту власні видимість полів, права доступу, поведінку та багато іншого.

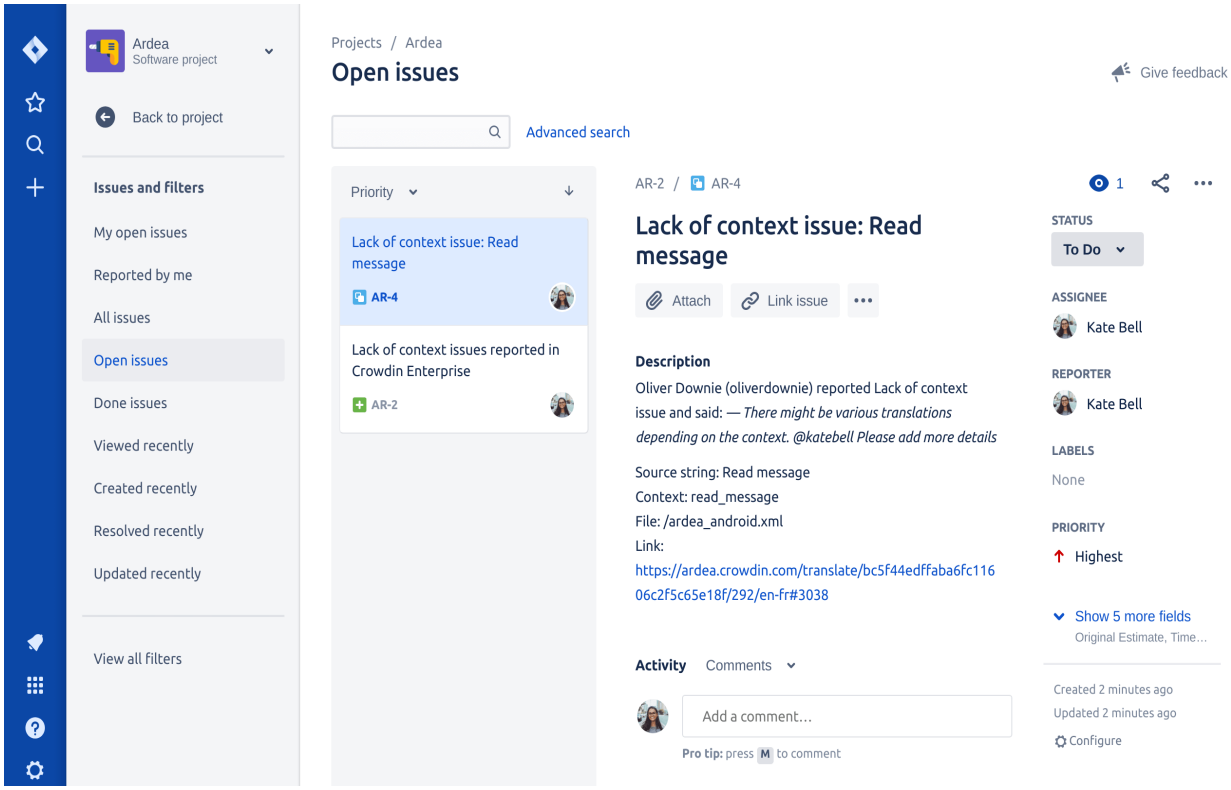


Рисунок 1.1. Приклад інтерфейсу системи Jira для списку відкритих проблем проєкту

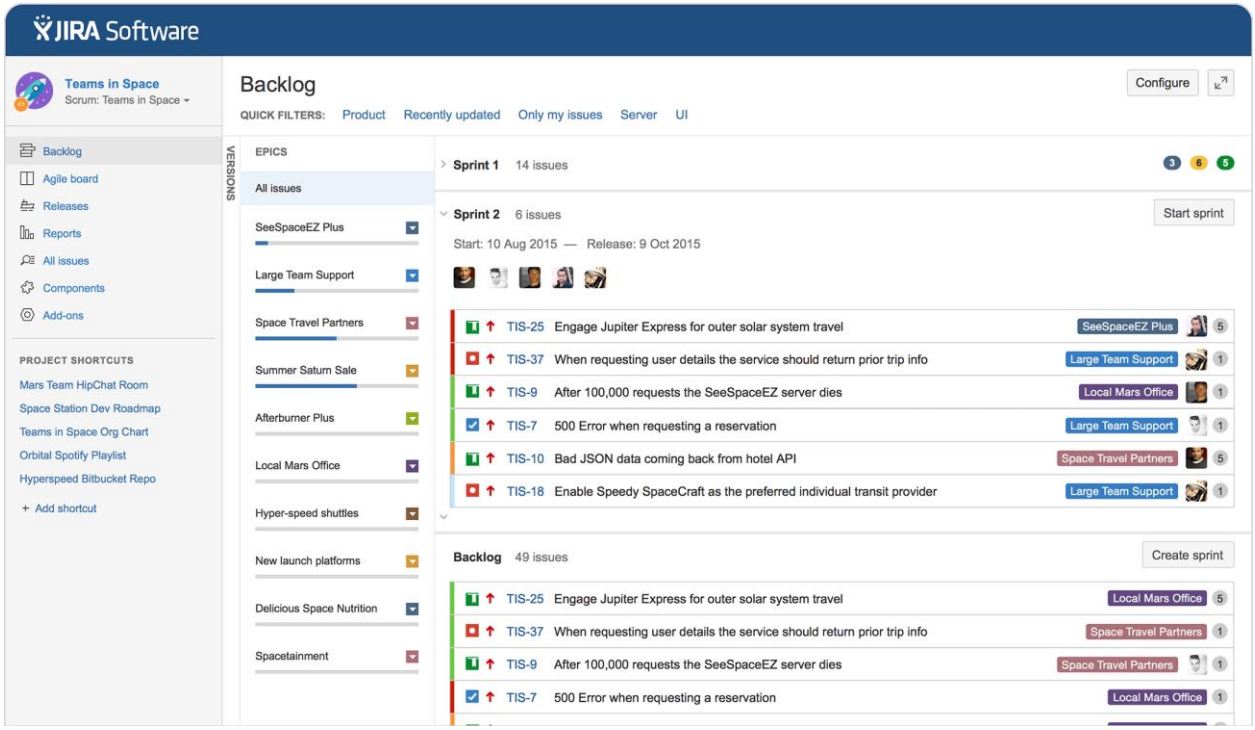


Рисунок 1.2. Приклад інтерфейсу системи Jira для списку задач проєкту

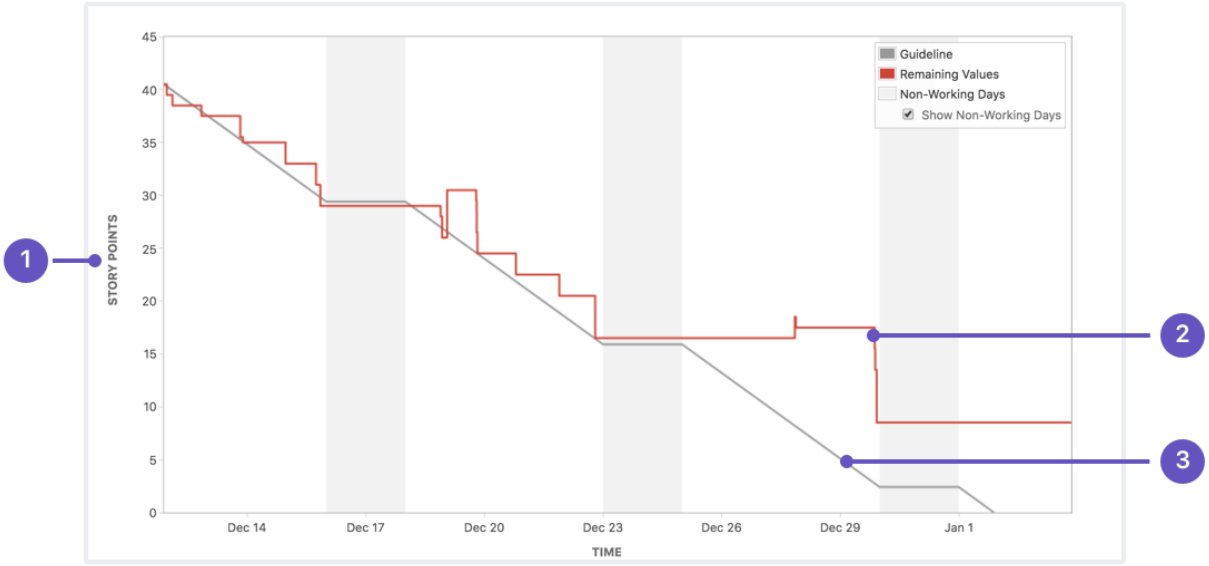
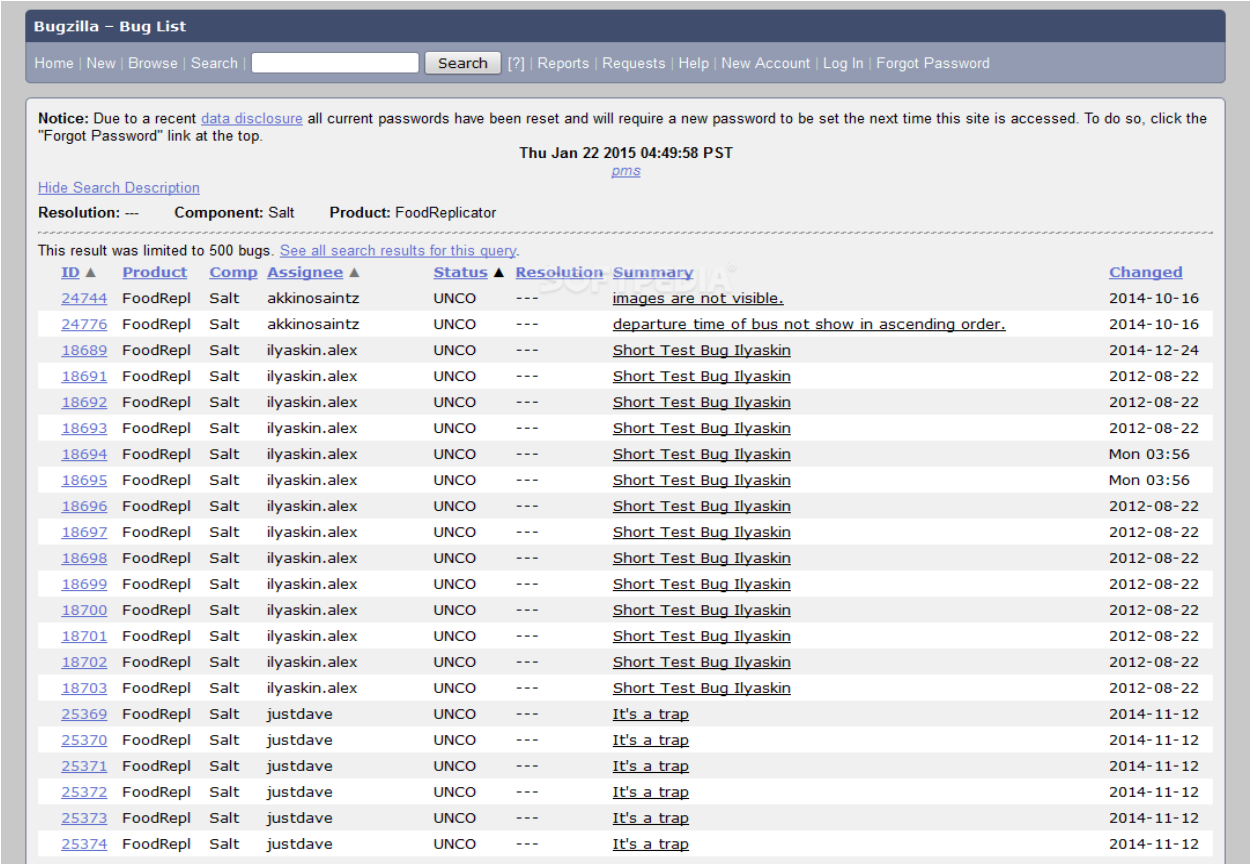


Рисунок 1.3. Приклад системи Jira для відображення графіку згорання проєкту

Bugzilla - це вільна система відслідковування помилок. Для управління надається вебінтерфейс. Система дозволяє:

- відслідковувати помилки і зміни коду
- спілкуватися з членами команди
- розміщувати і описувати патчі
- здійснювати контроль якості продуктів



ID ▲	Product	Comp	Assignee ▲	Status ▲	Resolution	Summary	Changed
24744	FoodRepl	Salt	akkinosaintz	UNCO	---	images are not visible.	2014-10-16
24776	FoodRepl	Salt	akkinosaintz	UNCO	---	departure time of bus not show in ascending order.	2014-10-16
18689	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2014-12-24
18691	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18692	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18693	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18694	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	Mon 03:56
18695	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	Mon 03:56
18696	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18697	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18698	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18699	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18700	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18701	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18702	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
18703	FoodRepl	Salt	ilyaskin.alex	UNCO	---	Short Test Bug Ilyaskin	2012-08-22
25369	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12
25370	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12
25371	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12
25372	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12
25373	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12
25374	FoodRepl	Salt	justdave	UNCO	---	It's a trap	2014-11-12

Рисунок 1.4. Приклад інтерфейсу системи Bugzilla для списку відкритих проблем проєкту

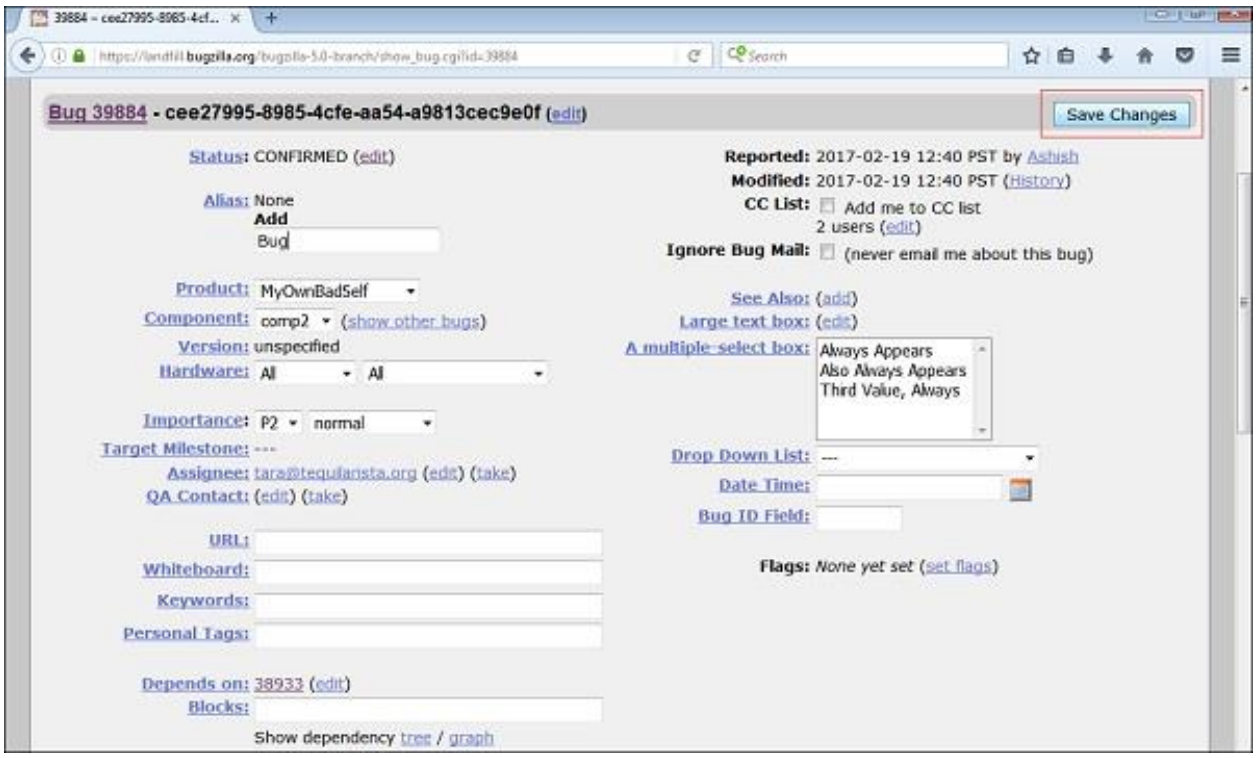


Рисунок 1.5. Приклад інтерфейсу системи Bugzilla при створенні нової проблеми проєкту

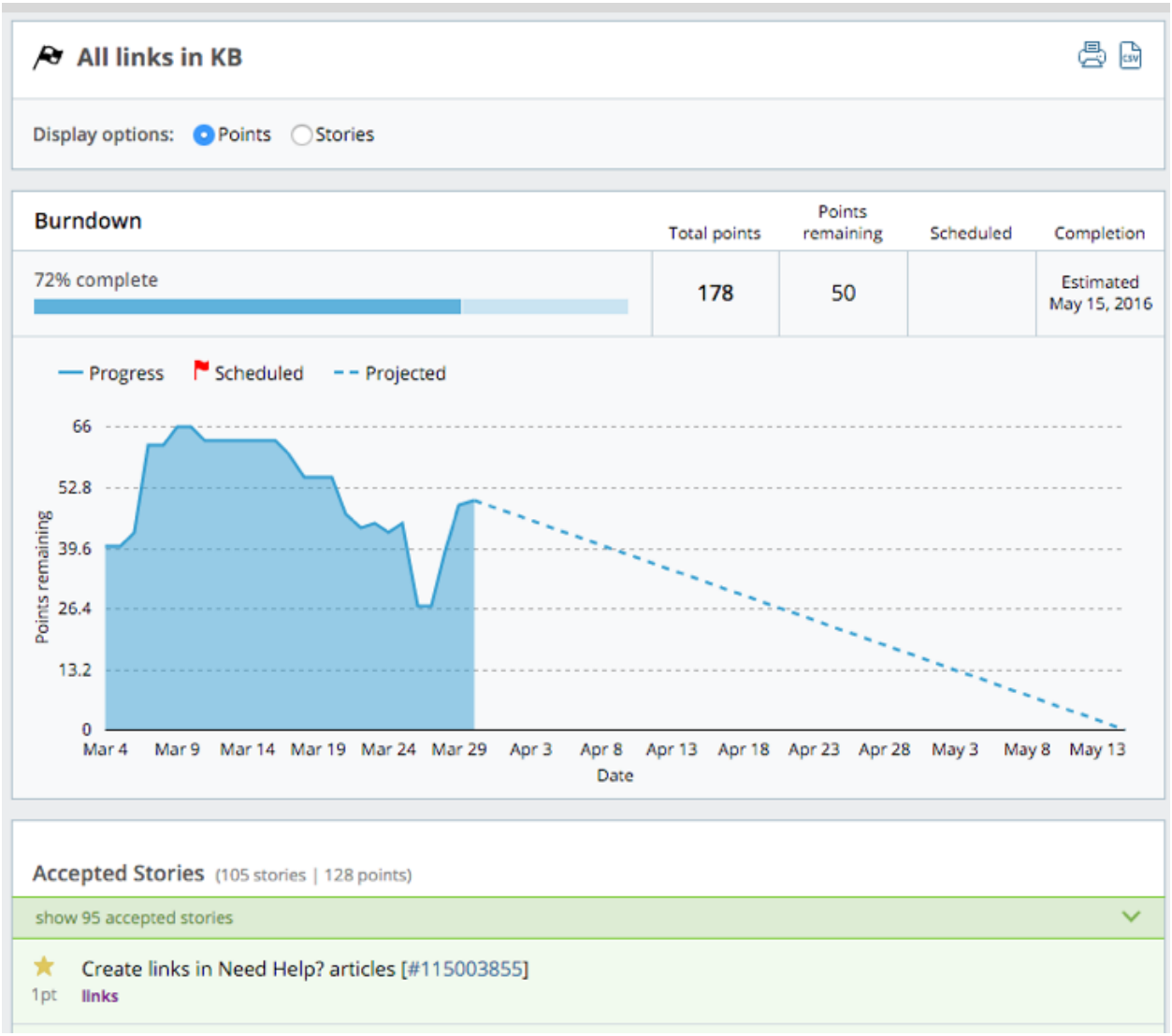


Рисунок 1.6. Приклад системи Bugzilla для відображення графіку згорання проєкту

Redmine - відкритий вебдодаток для управління проєктами та завданнями (включаючи відстеження помилок).

- ведення декількох проєктів;
- гнучка система доступу но основі ролей;
- система моніторингу дефектів;
- календар і діаграма Ганта;
- управління файлами, ведення новин проєкту та документів;
- сповіщення про зміни за допомогою електронної пошти та RSS-потоків;
- інформаційні сторінки для усіх проєктів;

- форуми для усіх проєктів;

-підрахунок тимчасових витрат;

- налаштування довільних полей для тимчасових витрат, інцидентів проєктів і користувачів;

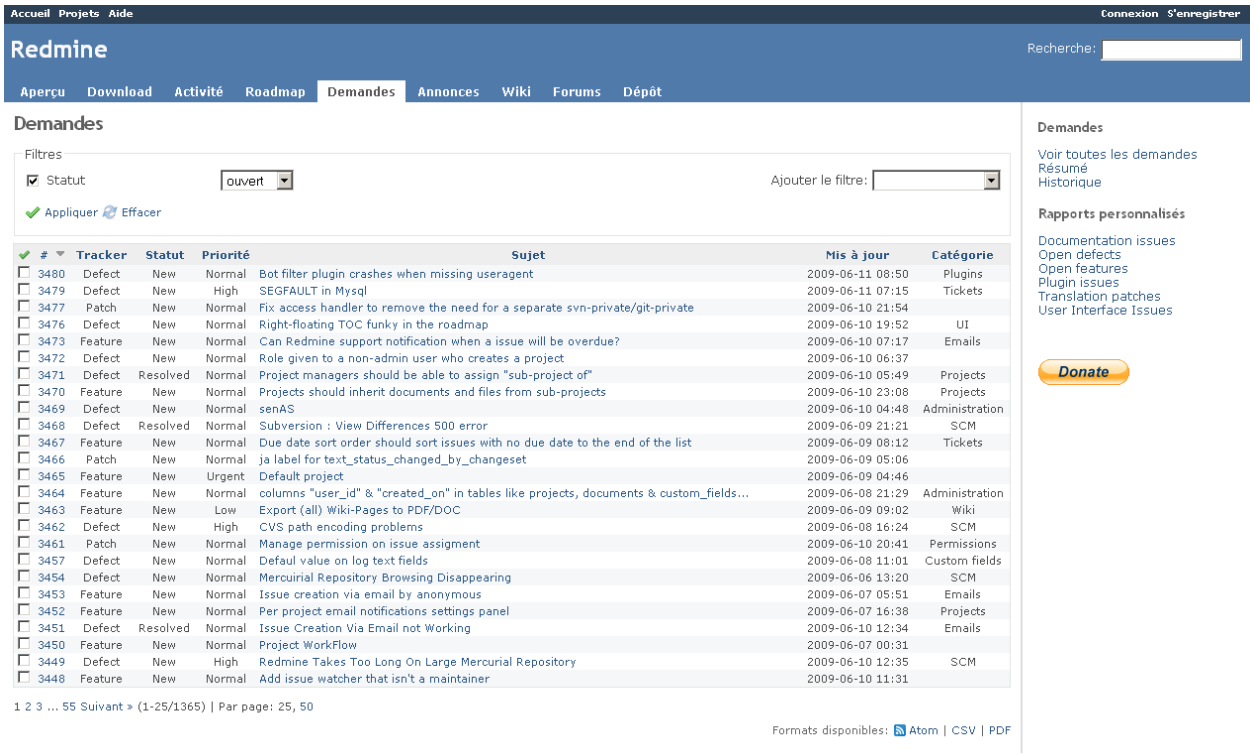


Рисунок 1.7. Приклад інтерфейсу системи Redmine для відображення списку існуючих відкритих проблем проєкту

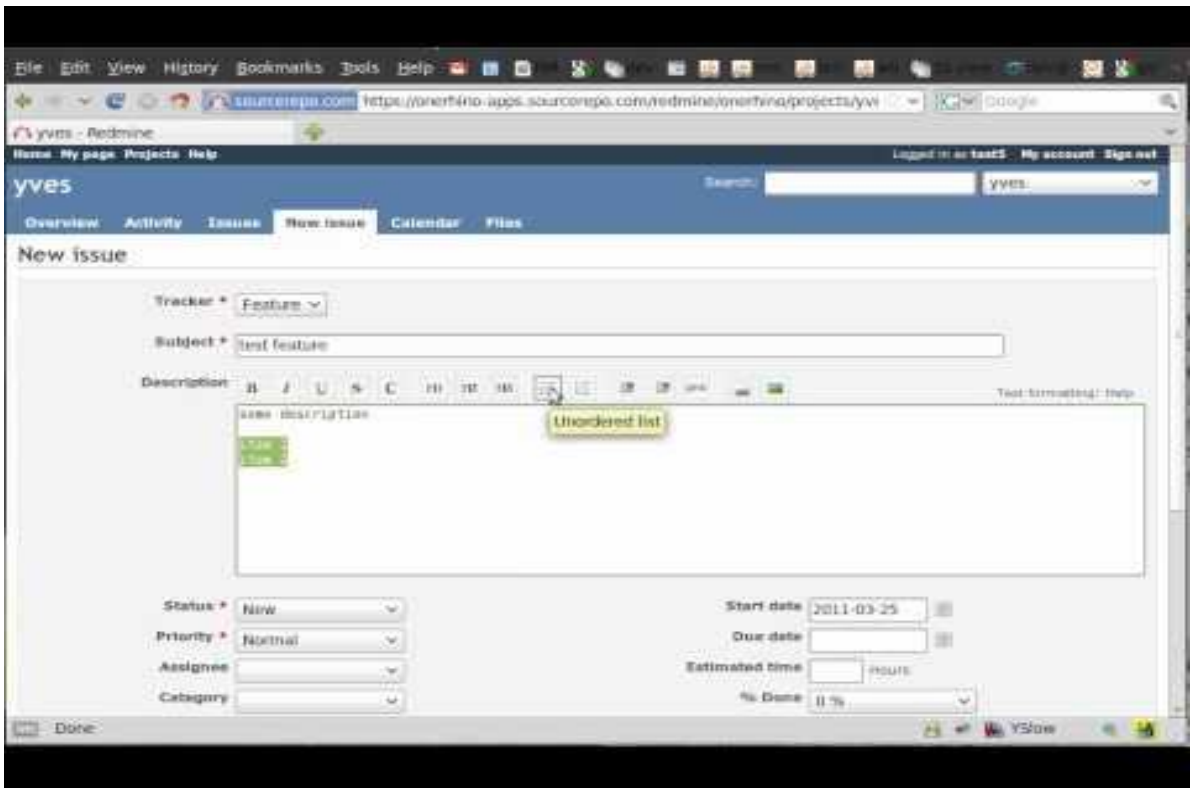


Рисунок 1.8. Приклад інтерфейсу системи Redmine для створення нової задачі

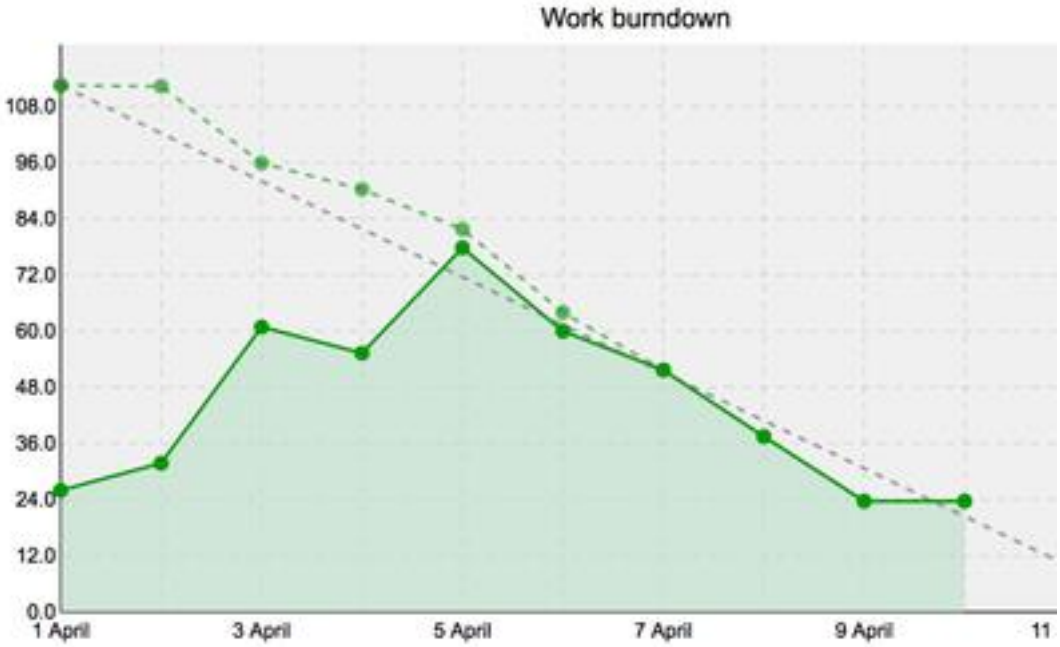


Рисунок 1.9 Приклад системи Redmine для відображення графіку згорання

Таблиця 1.1.

Порівняння можливостей існуючих систем

Можливості	**Jira**	**Bugzilla**	**Redmine**
Agile-планування та звітність	+	-	+
Дошки Scrum і Kanban	+	-	-
Розстановка пріоритетів завдань шляхом перетягування	+	+	-
Дашбоарди з налаштованим гаджетами	+	-	+
Налаштовуваний дизайнер процесів	+	+	+
Інтерфейси REST API	-	-	+
Проведення зустрічей і ретроспектив спринтів	+	-	-
Проведення мозкових штурмів з пошуку ідей для продуктів і подальшої розробки	+	-	-
Інтеграція з Bitbucket або GitHub	+	+	+
Простота настройки і управління	-	+	-
Agile-інструментарій для управління портфелем проєктів	+	+	+
Потужний пошук	+	-	-
Комплексна agile-звітність	+	-	+
Розстановка пріоритетів завдань шляхом перетягування	+	+	-

Можливість роботи в хмарі і на сервері	+	-	+

1.5. Постановка проблеми

Одним з напрямків управління процесом розробки великих частин продукту, який переважно використовується при розробці частин великих продуктів - є організація розробки в “потоках”.

Потік - певна, велика група працівників яка відповідає за розробку частини продукту. Він очолюється однією людиною - проєктним менеджером, в задачі якого входить:

- Планування та організація робочого процесу
- Декомпозиція великої задачі на менші
- Формування із потоку тимчасових команд під наявні задачі
- Присвоювання задач командам

Тобто, процес розробки має наступний вигляд. Потік отримує певну задачу на виконання, зазвичай з великим часовим терміном, після композиції задачі, проєктному менеджеру необхідно сформувати команди з наявних в потоці працівників під задачі на певний (короткий період). Створені команди працюють використовуючи одну з реалізацій Agile методології, таку як - Scrum. Кожна команда після закінчення роботи над своєю задачею, розформовується, таким чином звільняючи ресурси потоку для нових завдань. В подальшому процес повторюється знову до моменту закінчення розробки продукту.

Перевагами даного підходу є:

- Обмін знаннями про продукт між працівниками
- Чіткий аналіз прогресу в розробці
- Стабільна психологічна ситуація між працівниками в силу частих змін команд.

Недоліком даного підходу є високе навантаження на проєктного менеджера та особлива складність формування тимчасових команд.

Саме для вирішення цієї проблеми необхідно створення нової системи для управління процесом розробки проєкту, яка буде оптимізувати процес створення команд під задачі розробки продукту.

1.6. Цілі та задачі розробки

Дана система створюється для проєктних менеджерів які управляють розробкою великих продуктів використовуючи систему потоків та гнучкі методології розробки. Ціллю є: автоматизація процесу формування тимчасових команд та аналізу розподілення задач між ними, що в свою чергу зменшить навантаження на проєктного менеджера та підвищить продуктивність процесу розробки продукту.

Висновки

В розділі наданий опис процесу розробки програмного забезпечення. Був наданий опис існуючих моделей процесів розробки:

- Водоспадна
- Ітераційна
- Спіральна

Був наданий опис гнучкої методології розробки Agile, як найпоширенішої техніки управління проєктами та її основними реалізаціями - Scrum та Kanban. Розглянуті основні існуючі системи для управління процесом розробки: Jira, Bugzilla, Redmine, висвітлені їх переваги та недоліки і сформована таблиця з ключовими функціональними відмінностями.

Також був описаний напрям розробки в потоках, та процес створення тимчасових команд для роботи над задачами. В ході аналізу даного підходу були виявлені недоліки, для вирішення яких була поставлена ціль - зменшити навантаження на проєктних менеджерів та підвищити продуктивність розробки продукту шляхом створення автоматизованої інформаційної системи розподілу та аналізу задач проєкту.

РОЗДІЛ 2.
ВИМОГИ ДО ПРОГРАМНОГО ЗАСОБУ

2.1. Збір вимог

Збір вимог - це процес, необхідний для затвердження та створення документа, що містить специфікацію системних вимог [18].

Великі і складні проєкти зазвичай налічують тисячі вимог. Бізнес-аналіз якраз і дозволяє виявляти проблеми і визначати, що потрібно для їх подолання. У великих проєктах, таких як розробка програмного забезпечення, збір вимог - є одним з найважливіших етапів життєвого циклу проєкту, який може зайняти кілька тижнів / місяців.

Існують наступні категорії вимог:

1. Функціональні вимоги - це вимоги до системи.
2. Бізнес-вимоги - бізнес цілі проєкту.
3. Користувацькі вимоги - вимоги кінцевих користувачів які отримуються методом інтерв'ювання.
4. Призначені для користувача вимоги формуються в термінах предметної області, а функціональні вимоги - в термінах системи.

2.1.1 Формування вимог

Вимоги формуються за допомогою наступних етапів:

- Аналіз предметної області. Аналітик вивчає предметну область, в якій функціонуватиме система.
- Збір вимог. Це кооперація з людьми, які формують вимоги. Предметна область продовжить аналізуватись під час цього процесу.
- Класифікація вимог. На цьому етапі неформатована колекція вимог перетворюється на логічно пов’язані групи.
- Вирішення конфліктів. Без сумнівів, що потреби багатьох людей, які беруть участь у процесі встановлення вимог, суперечливі. На цій фазі ці протиріччя визначаються та вирішуються.

- Призначення пріоритетів. У будь-якому наборі вимог одні будуть важливішими за інші. На цьому етапі визначаються найбільш важливі вимоги разом з людьми, що формують ці вимоги.
- Перевірка вимог. На цьому етапі визначається послідовність, узгодженість і повнота.

2.2. Вимоги до інформаційної системи

2.2.1. Функціональні вимоги

Функціональні вимоги - це перелік сервісів, які система повинна виконувати, як система реагує на вхідні дані та як вона веде себе у різних ситуаціях тощо. У деяких випадках призначена система не повинна виконуватися.

Функціональні вимоги визначають функції програмного забезпечення, які розробники повинні будувати, щоб користувачі могли виконувати свої завдання в рамках бізнес-вимог. Іноді їх називають поведінковими вимогами, вони містять традиційні положення "повинен" або "повинна": "Система повинна надіслати підтвердження замовлення користувачеві електронною поштою".

Стандартні форми для визначення функціональних вимог:

- Опис функцій або об'єкта.
- Опис даних входу та їх джерела.
- Опис даних виходу із пунктом їх призначення.
- Допис, що необхідно для виконання функції.
- Специфікація функції, та патенти, опис попередніх умов (передумов).
- Опис побічних ефектів.

Дана система повинна бути побудована як два незалежних продукту - система управління проєктами та модуль оптимізації розподілення і аналізу задач.

Система управління проєктом має реалізувати наступні функції:

- Інструменти для аналізу продуктивності роботи над проєктом
- Інтеграція з google сервісами
- Можливість імпорту даних зі схожих систем

- Можливість коментування задач
- Створення зв’язаних сторінок з детальним описом задачі
- Підтримка можливості роботи багатьох команд всередині одного проєкту
- Можливість створення та редагування задач проєкту
- Можливість встановлення пріоритетів задачам
- Можливість побудови графіків для аналізу результатів роботи над проєктом
- Можливість створення дошок Scrum і Kanban
- Зміна пріоритетів задач
- Дашбоарди з налаштованим гаджетами
- Налаштовуваний дизайнер процесів
- Інтеграція з модулем оптимізації розподілу та аналізу задач
- Інтерфейси REST API
- Проведення зустрічей і ретроспектив спринтів
- Інтеграція з GitHub
- Agile-інструментарій для управління портфелем проєктів
- Потужний пошук по задачам
- Фільтрація задач по багатьом критеріям
- Можливість вивантаження звітів в .pdf форматі
- Розстановка пріоритетів завдань шляхом перетягування
- Можливість роботи в хмарі і на сервері
- Можливість додавання працівників в систему
- Можливість створення користувацьких фільтрів
- Можливість створення користувацьких інформативних сторінок

Вимоги до модулю оптимізації розподілу та аналізу задач:

- Модуль повинен реалізувати систему критеріїв для задач
- Модуль повинен реалізувати систему вмінь працівників для порівняння з критеріями задач
- Модуль повинен вміти порівнювати вміння працівників з критеріями задач
- Кожен критерій задачі повинен мати присвоєний коефіцієнт важливості

- Кожен навик працівника повинен мати коефіцієнт який описує рівень володіння
- Модуль повинен вміти автоматично формувати команди зі списку наявних працівників під вибрану задачу
- У випадку коли менеджер самостійно формує команду під задачу, модуль повинен надавати аналітичну інформацію щодо коректності даного вибору

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги описують характеристики системи та її середовища, а не поведінку системи. Це також перелік обмежень, накладених на дії та функції, що виконуються системою.

Сюди входять обмеження часу, обмеження процесу розвитку системи та стандарти.

Нефункціональні вимоги не пов’язані безпосередньо з функціями, що виконуються системою, вони пов"язані з інтеграційними характеристиками системи (такими як розмір системи , час відгуку або надійність). Крім того, нефункціональні вимоги можуть визначати обмеження для системи, такі як пропускна здатність пристрою вводу-виводу або формат даних, що використовується в системному інтерфейсі.

Нефункціональні вимоги засновані на бюджетних обмеженнях та враховують організаційні можливості розробника, здатність взаємодіяти з іншим програмним забезпеченням та комп’ютерними системами та зовнішні фактори, такі як захист інтелектуальної власності або правила безпеки.

Нефункціональні вимоги описують цілі та ознаки якості. Атрибути якості - це додаткові описи функцій продукту, що висловлюються через опис характеристик товару, важливих для користувачів або розробників.

Нефункціональними вимогами даної системи є:

- легкість і простота використання;
- легкість переміщення;
- цілісність;

- ефективність і стійкість до збоїв.

2.2.3 Інтеграційні вимоги

Вимоги до інтеграції описують низькорівневий інтерфейс взаємодії нової системи з декількома іншими системами компанії. Мета даного документу обгрунтувати і формалізувати вибір методу інтеграції. Документ містить в собі опис методів і способів інтеграції з зовнішніми системами, сервісами.

Інтеграція додатків - це технологічні процеси, які використовуються для організації обміну даними між різними інформаційними системами за допомогою засобів інтеграції, наданими додатками. До засобів інтеграції, наданими додатками відносяться API функції, пакети обробки та експорту / імпорту даних.

Оскільки дана система складається з двох частин - системи управління проєкту та модулю для розподілу та аналізу задач проєкту. Система повинна бути незалежним продуктом, модуль має бути заздалегідь влаштований в неї. В свою чергу сам модуль аналізу повинен буди зовнішнім продуктом, який повинен мати можливість працювати як в рамках даної системи управління проєктом, так і іншими подібними, вже відкритими на ринку продуктами.

Модуль повинен працювати з будь-якими реляційними базами даних через систему адаптерів, вивантажувати дані з таблиць в наступні файлові формати:

- .excel
- .json
- .db

Також повинна бути реалізована можливість для поширення схеми таблиць та їх взаємовідносин для імпортування в інші модулі. Дані повинні проходити компресію перед вивантаженням.

Модуль повинен бути розробленим як серверний додаток та реалізовувати архітектурний підхід REST.

REST - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST це узгоджений набір обмежень, що враховуються при проєктуванні розподіленої системи. У певних випадках (інтернет-магазини, пошукові системи,

інші системи, засновані на даних) це призводить до підвищення продуктивності і спрощення архітектури. У широкому сенсі компоненти в REST взаємодіють на зразок взаємодії клієнтів і серверів у всесвітній павутині.

2.2.4 Вимоги до розподілення ролей

В даній системі планується три групи користувачів: головний адміністратор, проєктний менеджер та працівник. До їх обов'язків входять різні функції, відповідно кінцевий програмний продукт повинен містити різні рівні доступу. Функції головного адміністратора, проєктного менеджера та працівника зображені на рисунках 2.1, 2.2, 2.3 відповідно.

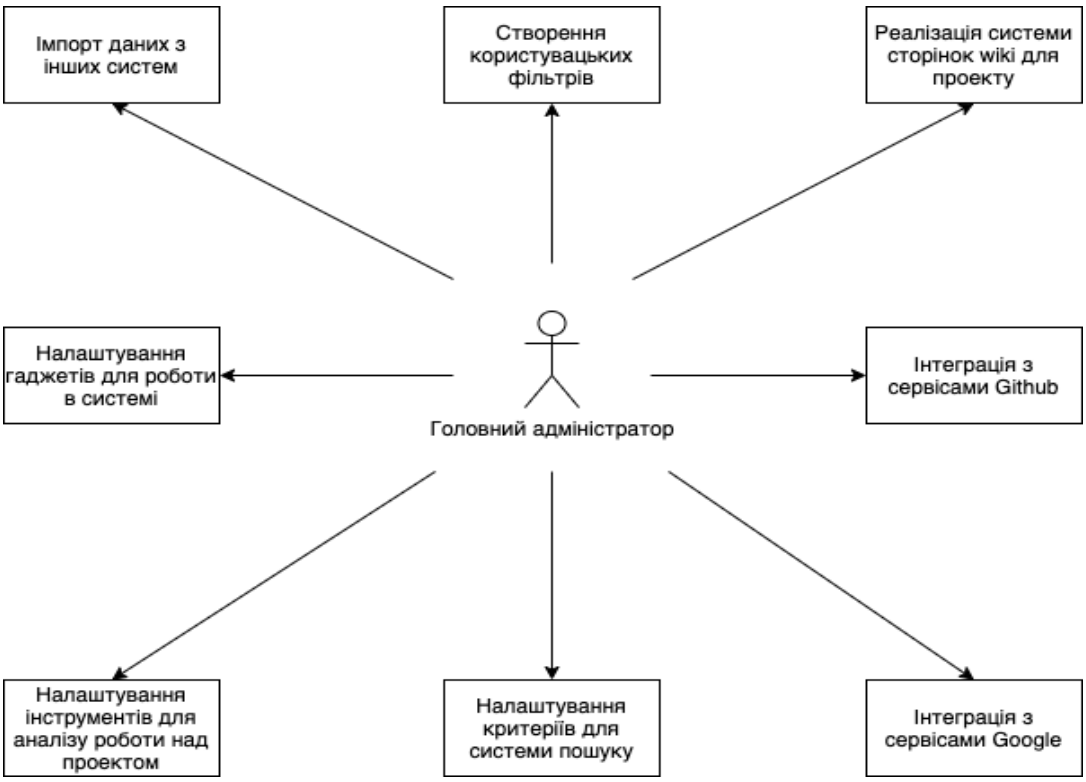


Рисунок 2.1. Основні функції головного адміністратора

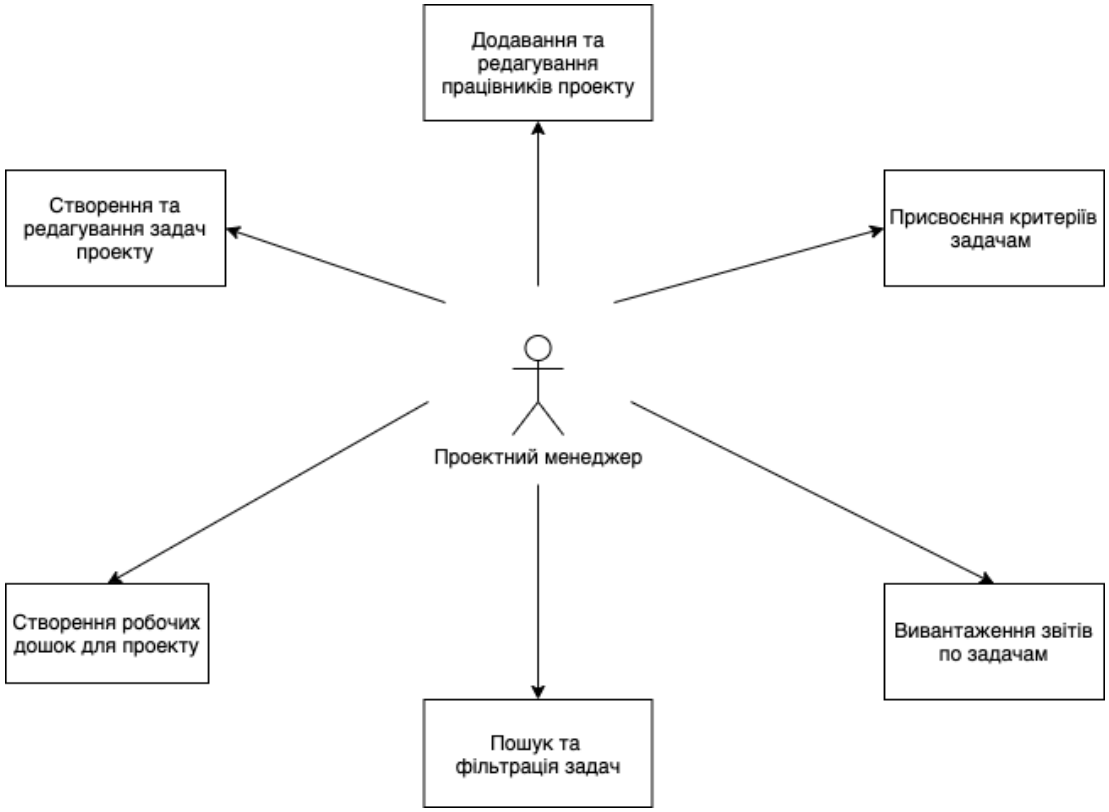


Рисунок 2.2. Основні функції проєктного менеджера

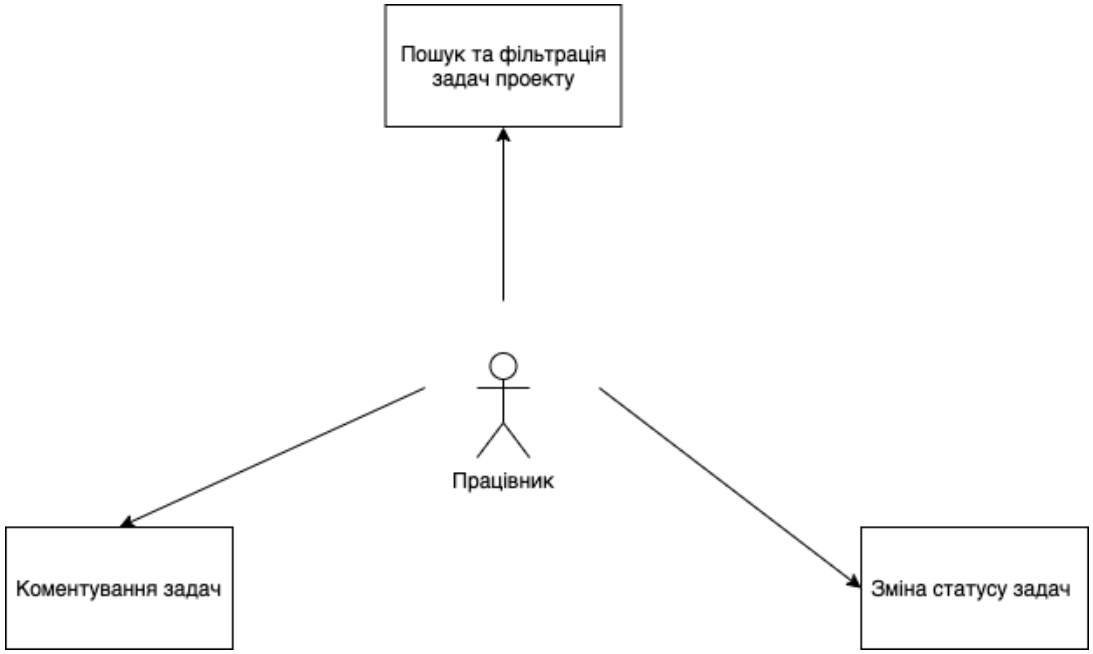


Рисунок 2.3. Основні функції працівника

2.2.5 Вимоги до архітектури системи.

Архітектура ІС повинна відповідати на загальноприйняті технології, стандарти, рекомендації, специфікації, засоби розробки та мови програмування. Для даної системи вибрана модель клієнт - серверної архітектури.

Технологія «Клієнт - сервер» - це архітектура програмного комплексу, в якій відбувається розподіл прикладної програми за двома логічно різними компонентами (клієнт і сервер), які взаємодіють за схемою «запит-відповідь» і вирішують свої певні завдання, загальна схема зображена на рисунку 2.4.

Архітектура серверу повинна бути реалізована через мікросервіси. Це повинно дозволити розробляти серверну частину використовуючи різні технології та полегшити горизонтальне масштабування проєкту.

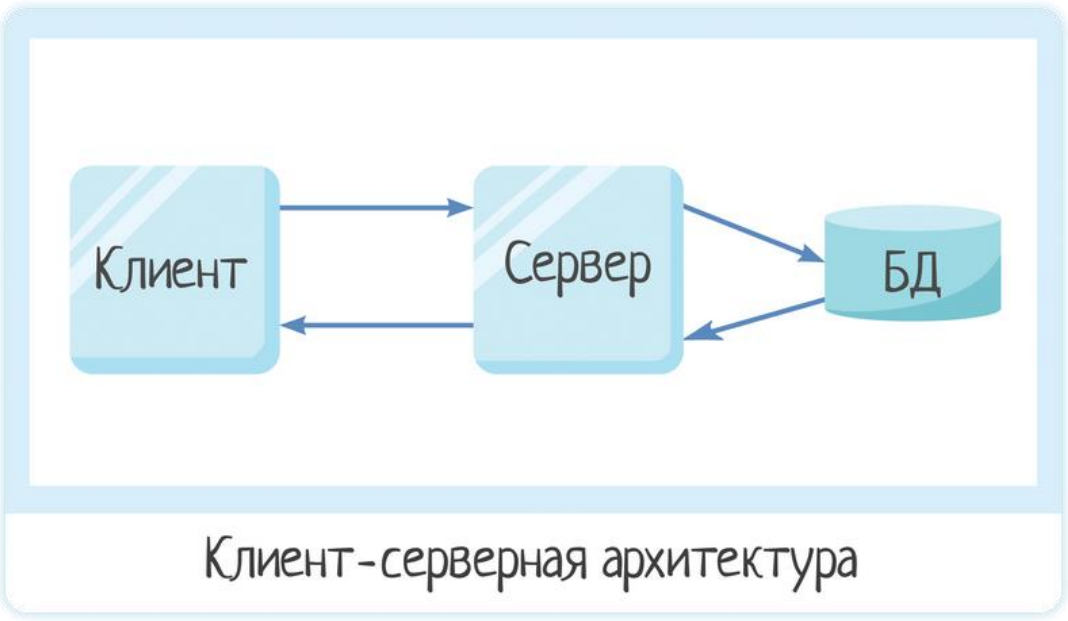


Рисунок 2.4. Загальна схема клієнт-серверної архітектури

Мікросервіс - це архітектурний стиль, при якому окрема програма будується як набір невеликих служб, кожна невелика служба працює у своєму власному процесі та використовує простий і швидкий протокол передачі даних (як правило, HTTP) для зв"язку з іншими службами зв"язку. Ці послуги побудовані з урахуванням бізнес-вимог і розгортаються незалежно, використовуючи типове повністю автоматизоване середовище. Централізоване управління цими службами є абсолютно найнижчим. Зі свого боку, вони можуть бути написані з використанням різних мов програмування та технологій зберігання.

Порівняно із сервісно-орієнтованою архітектурою, архітектура мікросервісу полегшує реалізацію безперервного процесу доставки програмних продуктів.Мікросервіс має на меті створити єдиний додаток, тоді як сервісно-орієнтована система - це група програм, що взаємодіють між собою.

Загальний вид архітектури мікросервісної архітектури представлений на рисунку 2.5.

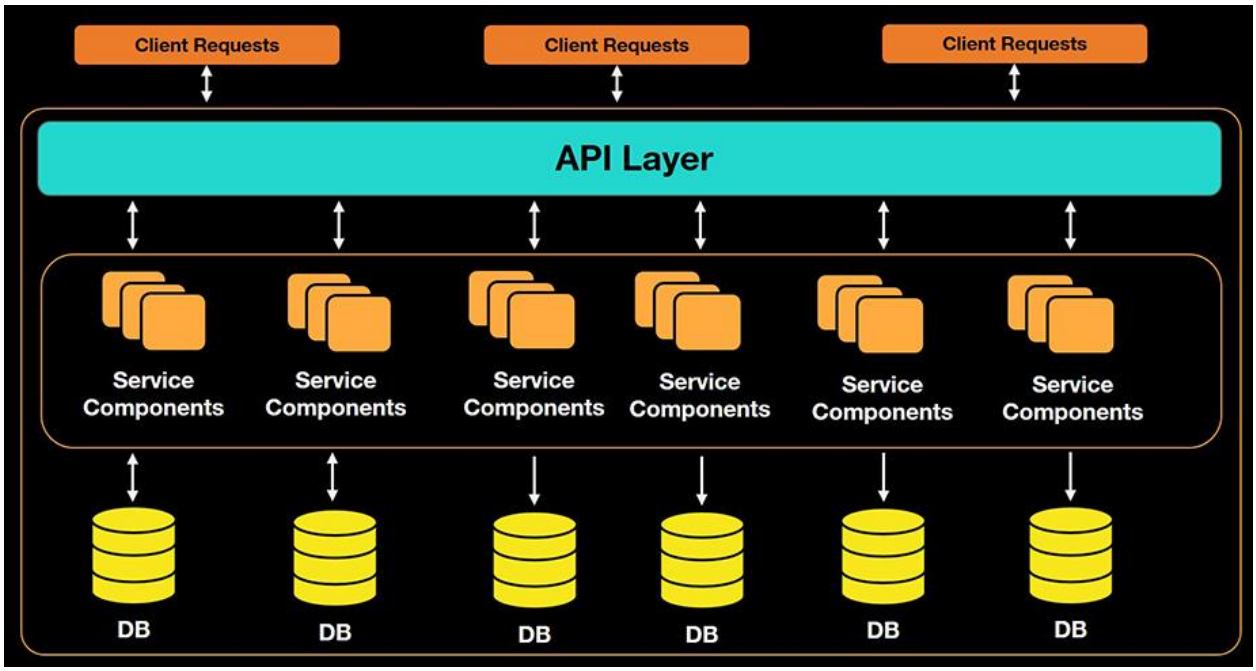


Рисунок 2.5 Загальна схема мікросервісної архітектури

Клієнтська частина повинна бути реалізована використовуючи підхід мікрофронтенд архітектури. Цей підхід є імплементацією серверної мікросервісної архітектури для великих клієнтських додатків.

Клієнтський мікросервіс має наступні характеристики:

- повністю ізольована частина користувацького інтерфейсу, жодним чином не залежить від інших; радикальна ізольованість; буквально розробляється як окремий додаток
- кожен клієнтський мікросервіс відповідає за певний набір бізнес-функцій від початку до кінця, тобто є повнофункціональним сам по собі
- може бути написаний на будь-яких технологіях.

Даний архітектурний підхід до розробки клієнт-серверної архітектури ефективно підходить для розробки великих довготривалих проєктів, над якими працює велика кількість команд.

2.2.6 Вимоги до надійності інформаційної системи

Програмні засоби інформаційної системи повинні забезпечувати:

− Контроль коректності даних, що вводяться, а також контроль несуперечності вхідних даних.

− Оповіщення користувача про помилки вхідних даних і суперечливості даних.

− Сервери інформаційної системи повинні мати також механізми резервного копіювання та відновлення даних, як після програмних збоїв, так і при відмові апаратних засобів або припиненні подачі електроенергії.

2.2.7 Вимоги до серверного обладнання

Використовуване серверне обладнання повинно відповідати наступним вимогам:

− Безперервний режим роботи обладнання. Серверне обладнання не вимагає втручання обслуговуючого персоналу для підтримки роботи. Втручання обслуговуючого персоналу здійснюється при зміні конфігурації або усунення аварійних збоїв в роботі.

− Сервери сертифіковані, відповідають усім санітарно-епідеміологічним вимогам і сертифіковані для застосовуваних серверних операційних систем.

Загальносистемне програмне забезпечення серверного обладнання повинно забезпечувати:

− роботу служби автоматичних оновлень, що дозволяє з мінімальними витратами підтримувати працездатність програмного забезпечення;

− простоту експлуатації і наявність сертифікованих фахівців з підтримки встановлених операційних систем.

2.2.8 Вимоги щодо безпеки

Система повинна забезпечувати:

− запобігання несанкціонованому доступу до інформації та (або) передачі її особам, які не мають права на доступ до інформації;

− застосування механізмів виявлення спроб вторгнення на сайт і отримання несанкціонованого доступу;

− розпізнавання типів всіх відомих атак по їх сигнатурам та зберігання їх в окремій базі на сервері;

− визначення ступеня важливості сигнатури атаки і налаштування сповіщень або блокування активності в залежності від даного показника;

− оповіщення адміністратора сайту про спроби вторгнення на сайт;

− надання можливості для адміністратора сайту заблокувати несанкціоновану призначену для користувача активність на сайті.

2.2.9 Документація вимог

Початкова документація - повинні бути створені діаграми високорівневої архітектури, на яких буде відображено основні компоненти системи та їх взаємодія. Зі сторони функціональних вимог повинні бути описані основні характеристики продукту, оскільки проєкт буди розроблятися за ітераційної моделлю за допомогою гнучкої методології розробки.

Під час роботи над проєктом повинні бути задокументовані:

− Нові вимоги, або вимоги що зазнали змін

− Технологічні зміни

− Тестувальники повинні зберігати результати тестів для подальшого аналізу

Під час розробки програмісти повинні використовувати загальноприйняті передові практики, які є частиною методології Agile, такі, як TDD (Test-Driven Development - розробка через тестування) і BDD (Behavior-Driven Development - розробка через реалізацію поведінки), а також писати добре структурований код, який зможуть читати і не технічні фахівці.

2.3 Опис використовуваних інструментів та технологій

На сьогодні незаперечним є той факт, що сучасний рівень web-технологій дозволяє вважати їх найбільш перспективними для створення розподілених інформаційних систем. Це можуть бути не тільки відкриті Internet-системи, але і «закриті» корпоративні автоматизовані системи управління, розподілені на великі території і відстані. Суттєвою особливістю таких систем є здійснення віддаленого доступу до сховищ інформації - баз даних.

Об'єднання Internet-технологій і технологій СУБД як способу організації доступу до даних має ряд безумовних переваг і вимагає не тільки знання цих технологій, а й вміння аналізувати і вибирати оптимальні архітектури таких інформаційних систем.

В проєкті використовуються наступні технології та інструменти:

1. SASS - (Syntactically Awesome Stylesheets) - це метамова на основі CSS, призначений для збільшення рівня абстракції CSS коду та спрощення файлів каскадних таблиць стилів [11].

Альтернативами SASS є подібні технології LESS та POSTCSS, але SASS має над ними наступні переваги:

− Вихідний код коротше, тому ви можете писати CSS швидше.

− Він сумісний з усіма версіями CSS. Таким чином, ви можете використовувати будь-які доступні бібліотеки CSS.

− Вона забезпечує систему наслідування, тому ви можете використовувати вкладені селектори і корисні функції, такі як маніпуляція кольором, математичні функції та інші значення.

2. JavaScript – мультипарадигмальна мова програмування. Підтримує об'єктноорієнтований, імперативний і функціональний стилі. Є реалізацією мови ECMAScript (стандарт ECMA-262).

JavaScript, як правило, використовується як вбудована мова для програмного забезпечення, що використовується для доступу до об"єктів додатків, і він широко використовується як мова сценаріїв у браузерах, щоб зробити вебсторінки інтерактивними.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.

Переваги:

− Основною перевагою є повна інтеграція з браузером.

− Популярність JavaScript відкриває перед програмістом чималу кількість готових бібліотек, які дозволяють значно спростити написання коду і нівелювати недосконалості синтаксису.

− JavaScript надає велику кількість можливостей для вирішення найрізноманітніших завдань. Гнучкість мови дозволяє використовувати безліч шаблонів програмування стосовно до конкретних умов.

− Дуже висока швидкість роботи Javascript.

− Можливість використовувати мову для серверної розробки.

3. React (іноді React.js або ReactJS) - JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів [7].

React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій.

React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість. В якості бібліотеки для розробки користувацьких інтерфейсів React часто використовується з іншими бібліотеками, такими як Redux.

Альтернативами React.js є фреймворки Angular.js (розроблено компанією Google) та View.js, кожен з них має свої плюси та мінуси, але в нашому випадку перевага надана React по наступним причинам:

− Наявність технології віртуального DOM (об'єктна модель документа), що дозволяє організовувати документи в форматах HTML, XHTML або XML в дерево, які краще сприймаються веббраузерами при розборі різних елементів вебпрограми.

− Висока швидкість, оскільки дані, що виконуються на стороні користувача, в той самий час можуть бути представлені на стороні сервера.

− У поєднанні з ES6 / 7 ReactJS може легко працювати з високим навантаженням.

− Бібліотека зі 100% відкритим вихідним кодом, яка отримує багато щоденних оновлень і вдосконалень відповідно до внесків розробників по всьому світу.

4. Redux - відкрита JavaScript бібліотека призначена для управління станом програми. Найчастіше використовується разом з React або Angular для побудови інтерфейсів користувача.

5. Node.js - програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду.

В проєкті Node.js вибрана в якості мови для серверної частини системи [10]. Вона має ряд переваг над своїми конкурентами, такими як PHP, яка використовувалася в попередній версії інформаційної системи.

Переваги Node.js:

− В Node.js використовується не блокуюча I/O модель, що позитивно позначається на швидкості обробки запитів.

− Оскільки Node.js використовує Javascript, то ми отримуємо єдиний синтаксис для клієнтської та серверної частини вебсайту. Це покращує можливість повторного використання коду і полегшує роботу full-stack розробника [9].

− Модуль потоку полегшує роботу з великими файлами, тобто ми можемо завантажувати файл частинами та використовувати його до повноцінного завантаження.

− Більша швидкість роботи відносно PHP.

6. Express - Express.js, або просто Express, фреймворк вебдодатків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він призначений для створення серверної частини веб додатку та API. Де-факто є стандартним каркасом для Node.js [8].

Єдиним конкурентом для Express.js є Hapi.js, різниця між ними не значна, але завдяки внутрішній реалізації Express, він не значимо, але все ж, швидше в процесі обробки запитів.

7. PostgreSQL - це вільно поширювана об'єктно-реляційна система управління базами даних (ORDBMS), найбільш розвинена з відкритих СУБД в світі і є реальною альтернативою комерційних баз даних.

Для початку слід коротко розповісти про різницю між реляційною та документоорієнтованою базами даних.

Реляційні бази даних використовують структуровану мову запитів (Structured Query Language, SQL) для визначення і обробки даних. З одного боку, це відкриває великі можливості для розробки: SQL одна з найбільш гнучких і поширених мов запитів, так що його вибір дозволяє мінімізувати ряд ризиків, і буде особливо до речі, якщо має бути робота з комплексними запитами. З іншого боку, в SQL є ряд обмежень. Побудова запитів на цій мові зобов'язує визначати структуру даних, бо подальша зміна структури даних може бути згубною для всієї системи.

Нереляційні бази даних, в свою чергу, пропонують динамічну структуру даних, які можуть зберігатися кількома способами: орієнтовано по колонках, документо-орієнтовано, в вигляді графів або на основі пар «ключ-значення». Така гнучкість означає наступне:

− Ви можете створювати документи, не ставлячи їх структуру заздалегідь.

− Кожен документ може мати власну структурою.

− У кожної бази даних може бути власний синтаксис.

− Ви можете додавати поля прямо під час роботи з даними.

В нашому випадку, оскільки цілісність бази даних є дійсно важливою, реляційна база даних – оптимальний варіант.

Серед реляційних СУБД, PostgreSql, не має реальних конкурентів, вона використовується такими компаніями як, Google, Facebook, eBay, та багатьма іншими.

Висновки

В цьому розділі був проведений збір вимог до програмного забезпечення. Для визначення вимог був проведений процес аналізу предметної області.

В результаті аналізу були визначені наступні вимоги:

− Функціональні вимоги.

− Інтеграційні вимоги.

− Нефункціональні вимоги.

− Вимоги до розподілення ролей.

− Експлуатаційні вимоги.

− Вимоги до архітектури.

− Вимоги до надійності.

− Вимоги до серверного обладнання.

− Вимоги щодо безпеки.

Також, детально були описані використані технології розробки програмного забезпечення для системи управління проєктом. В результаті аналізу, були поставлені основні вимоги до технічного забезпечення необхідного для стабільної та коректної роботи розробленого програмного забезпечення.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції, і мати багато фаз роботи, кожна з яких може мати окрему архітектуру. Архітектура повинна будуватись щоб найкраще відповідати вимогам до системи що створюється.

Проєктування системи будемо здійснювати на основі об'єкто-орієнтованого моделювання методом UML, який дозволяє враховувати аспекти, властиві діючим особам (акторам) системи та створювати сценарії виконання системи.

Доцільна побудова діаграм UML фази уточнення вказаних в таблиці 3.1.

Таблиця 3.1.

Перелік створених UML діаграм

Тип діаграми	*Роль у процесі проєктування*
Діаграма послідовності	Визначає набір класів та їхню послідовність взаємодії для вирішення певної задачі.
Діаграма компонентів	Визначає взаємозв’язок між компонентами системи
Діаграма розгортання	Містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними.

3.1 Діаграми послідовності

Для опису особливостей передачі та прийому даних між об’єктами ми покажемо діаграми послідовності для наступних сценаріїв:

- На рисунку 3.1 представлений процес створення нової задачі проєкту та присвоєння її працівнику.

- На рисунку 3.2 представлений процес автоматичного підбору виконавців під певну задачу.

На діаграмах задіяні наступні класи:

Таблиця 3.2

Класи інформаційної системи

Клас	**Відповідальність**
Клієнт (Проєктний менджер)	Введення логіну, паролю, отримання списку задач, створення нової, присвоєння користувачу.
Система	Виконує авторизацію користувача та встановлює зв’язок з базою даних.
Сервер БД	Сервер БД надає клієнтам доступ до об’єктів в базі даних.

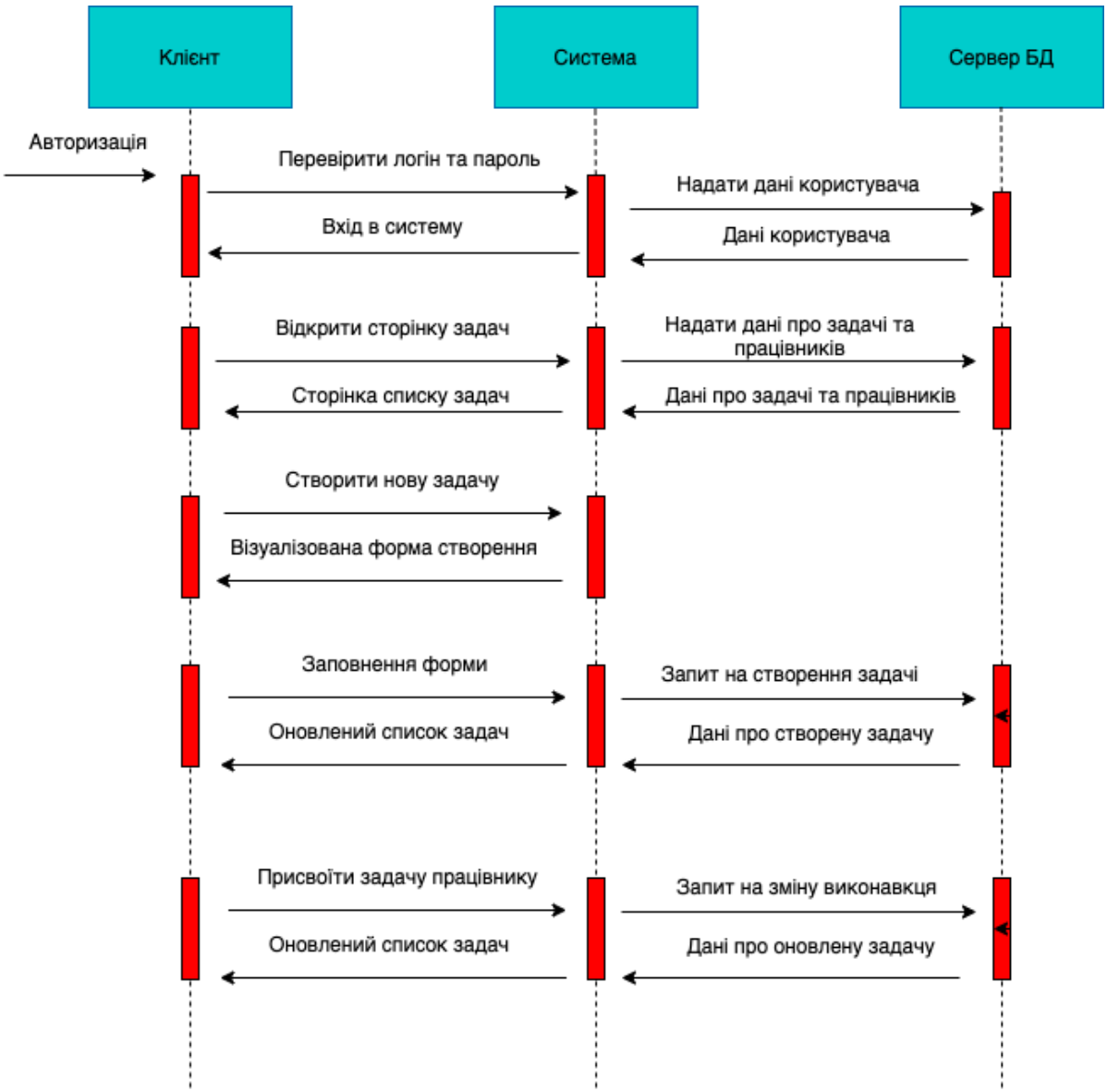


Рисунок 3.1 Діаграма послідовності для процесу створення та присвоєння працівнику задачі проєкту

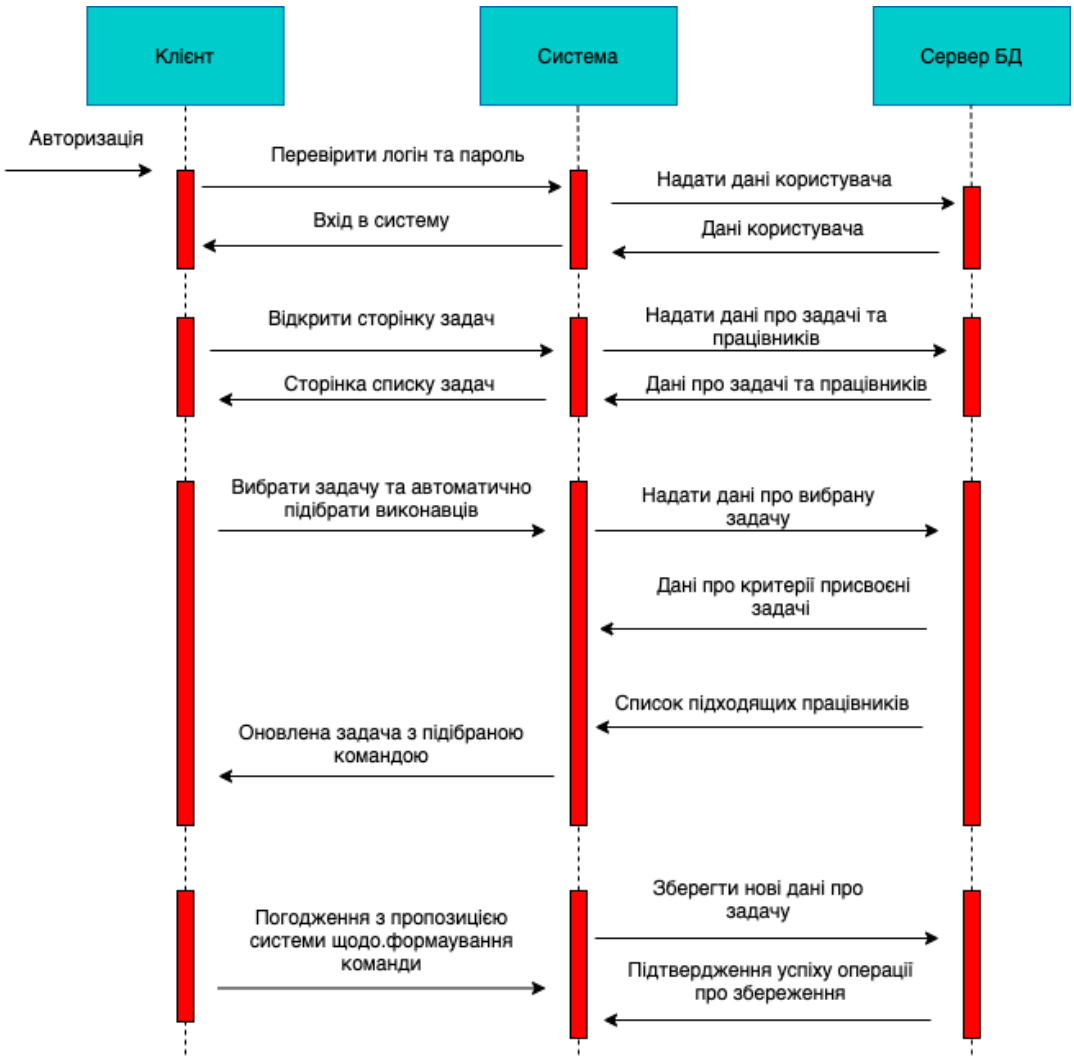


Рисунок 3.2 Діаграма послідовності для процесу автоматичного формування команди під задачу проєкту

3.2 Діаграма розгортання

Оскільки корпоративні додатки часто вимагають для своєї роботи деякої IT-інфраструктури, зберігають інформацію в базах даних, розташованих десь на серверах компанії, викликають веб-сервіси, використовують загальні ресурси і т. д. У таких випадках корисно мати графічне представлення інфраструктури, на яку буде розгорнуто додаток.

Інфраструктура даної системи зображена на рисунку 3.3.

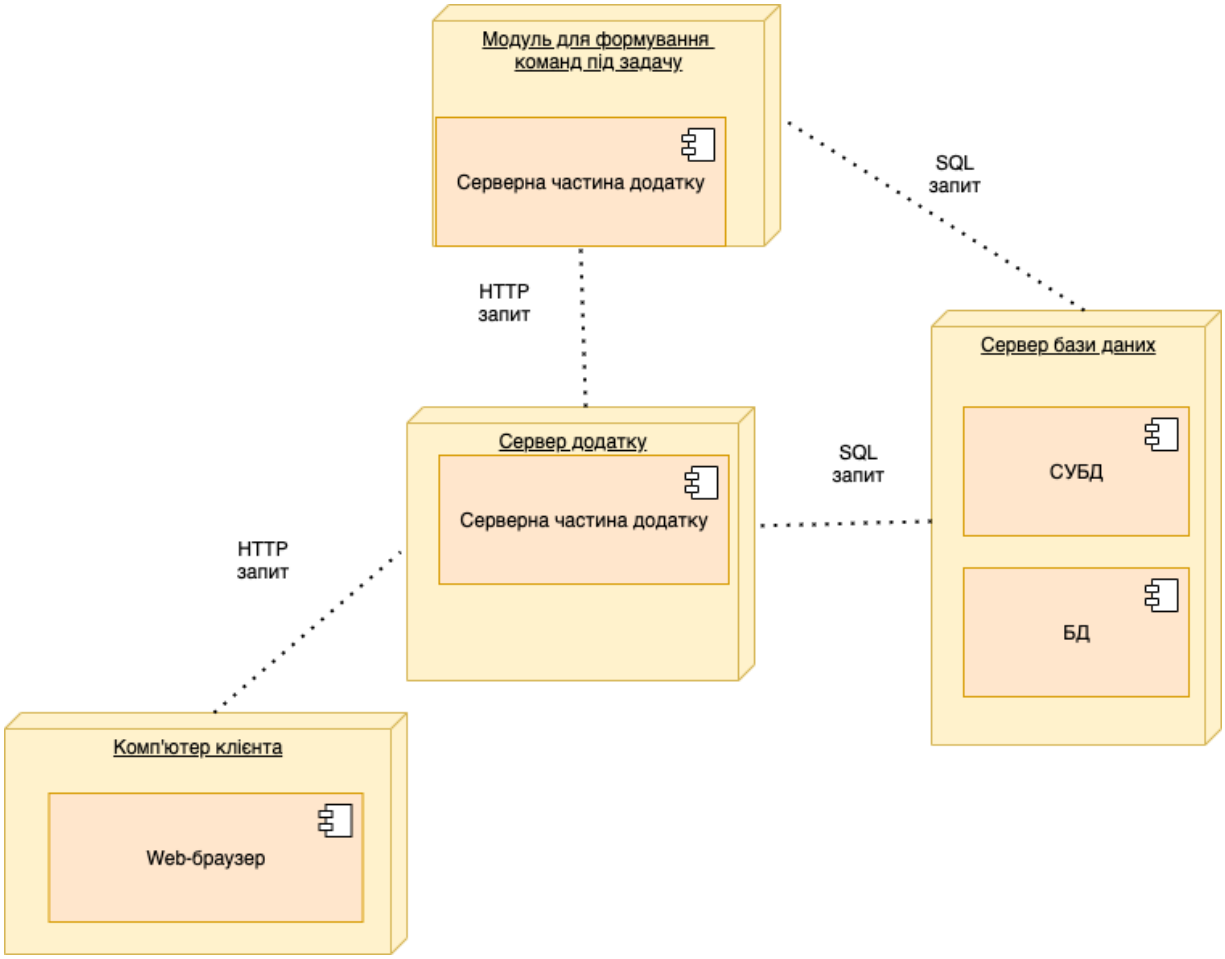


Рисунок 3.3 Інфраструктура інформаційної системи

3.3 Діаграма компонентів системи

Для відображення залежності між компонентами програмного забезпечення було розроблено структурну схему компонентів клієнтської частини системи (рисунок 3.4).

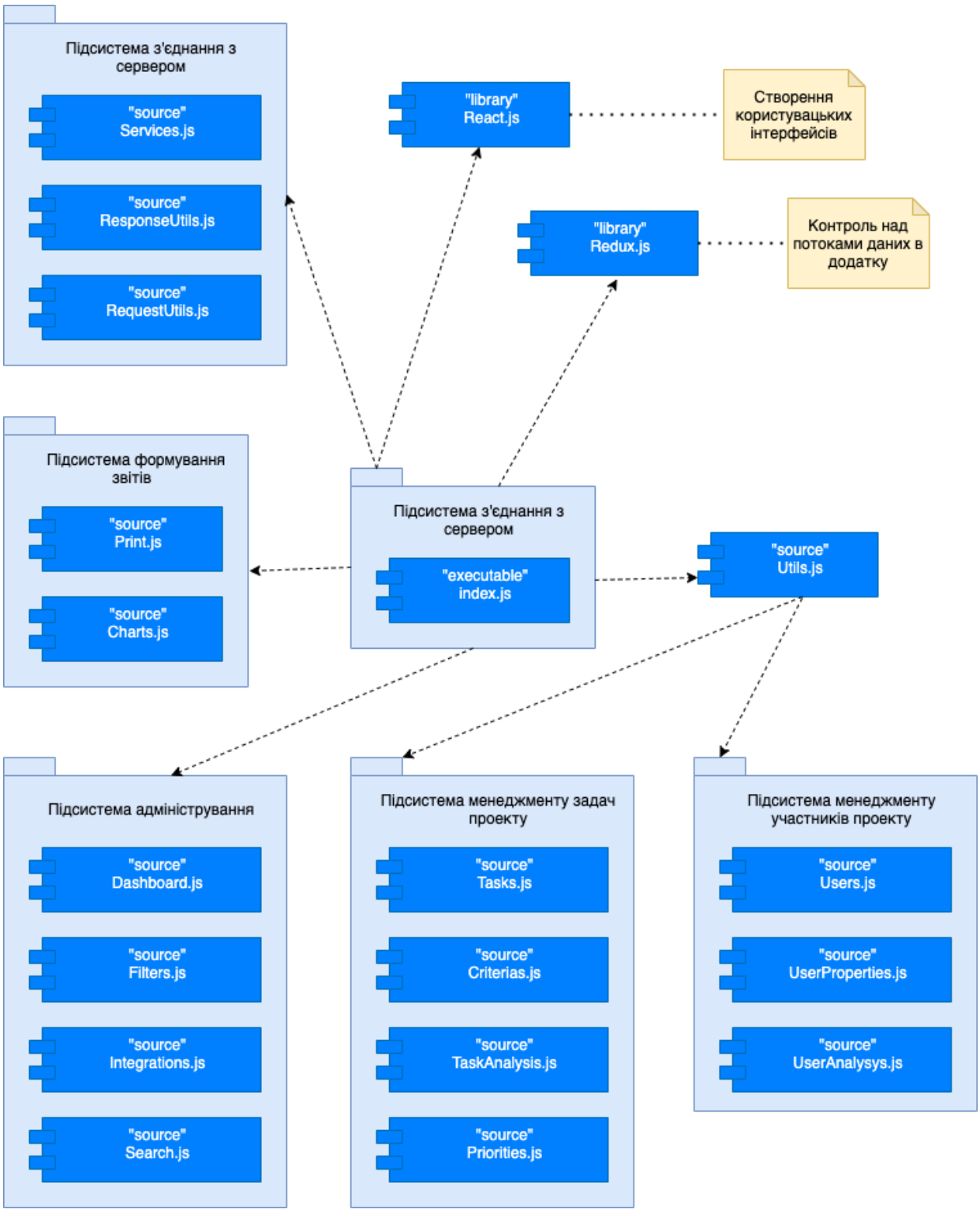


Рис. 3.4. Структура схема компонентів клієнтської частини системи

3.4 Опис мікросервісної архітектури серверної сторони додатку.

Мікросервіси - це архітектурний стиль, в якому окрема програма будується як набір невеликих служб, кожна з яких працює у своєму власному процесі та використовує простий і швидкий протокол передачі даних (як правило, HTTP) для зв'язку з іншими службами. Ці послуги побудовані з урахуванням бізнес-вимог і розгортаються незалежно, використовуючи типове повністю автоматизоване середовище. Централізоване управління цими службами є абсолютно найнижчим. Зі свого боку, вони можуть бути написані з використанням різних технологій зберігання і мов програмування.

Загальна структура мікросервісів даної системи представлена на рисунку 3.5.

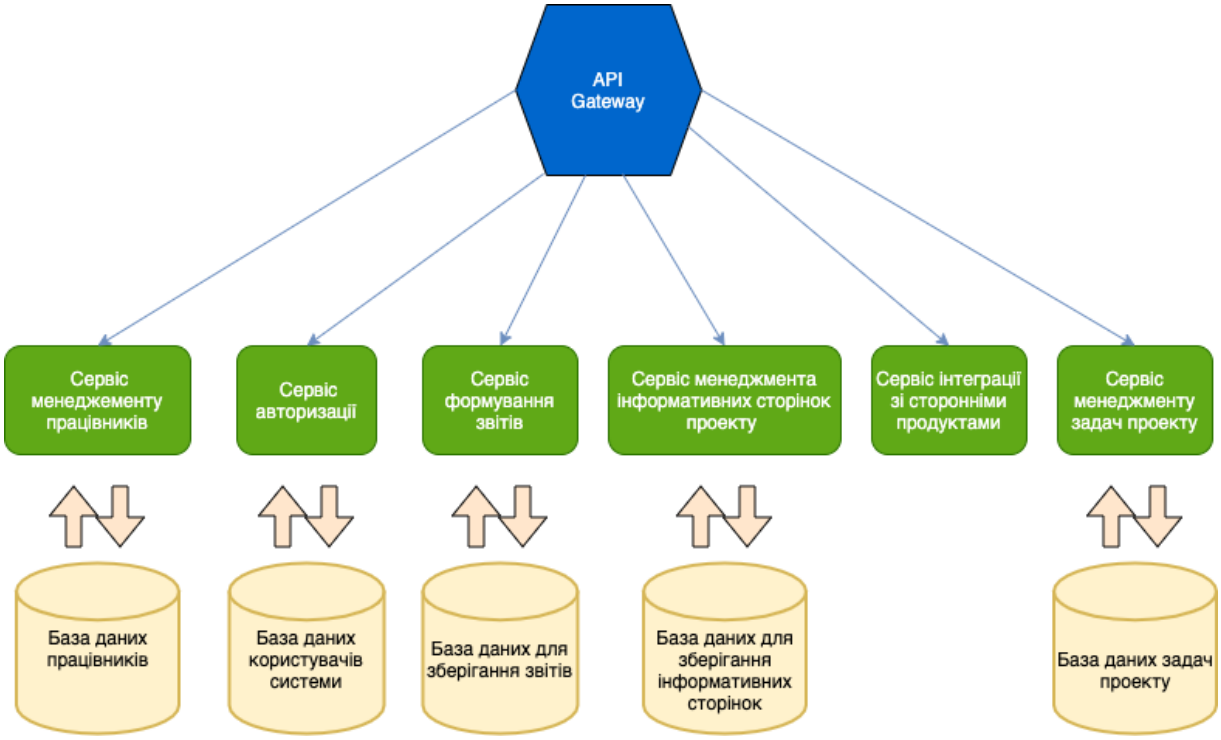


Рисунок 3.5. Загальна структура мікросервісів системи

В даному додатку мікросервісна архітектура реалізована наступним чином: кожен запит який приходе на сервер, попадає на API шлюз.

API шлюз - центральна точка входу для всіх запитів які приходять на сервер. Він тримає в собі інформацію про всі зареєстровані сервіси і в свою чергу не виконує ніяких обчислень або взаємодії з базою даних. Його робота полягає в тому щоб перенаправляти запити на відповідні сервіси. В подальшому сервіси отримують запит і відправляють певний результат шлюзу. Він в свою чергу відправляє результат клієнту.

Наявність даного прошарку є необхідною, адже саме через нього проходить перевірка авторизація запиту, кешування відповідей, обмеження кількості запитів.

Кожен із сервісів під'єднаних до системи мікросервісів, інкапсулює в собі певну частину логіки продукту та є повноцінною незалежною частиною проєкту зі своєю базою даних.

В таблиці 3.3. представлений короткий опис кожного з мікросервісів проєкту.

Таблиця 3.3.

Короткий опис мікросервісів системи

Назва сервісу	**Опис**
Сервіс менеджменту працівників	Основними функціями даного сервісу є: - Створення нових працівників проєкту - Редагування працівників проєкту - Видалення працівників проєкту - Присвоєння рівня навичок для працівників проєкту - Присвоєння задач працівниками
Сервіс авторизації	Основними функціями даного сервісу є: - Реєстрація користувачів - Авторизація та автентифікація користувачів - Розділення ролей для користувачів
Сервіс формування звітів	Основними функціями даного сервісу є: - Формування звітів по задачам проєкту

	- Формування звітів по продуктивності працівників проєкту - Формування графіків для аналізу прогресу роботи над проєктом - Вивантаження звітів в необхідних форматах
Сервіс менеджменту інформативних сторінок проєкту	Основними функціями даного сервісу є: - Створення інформаційних сторінок для проєкту - Зв’язування інформаційних сторінок з задачами - Редагування інформаційних сторінок проєкту - Видалення інформаційних сторінок проєкту - Підтримка мови Markdown для стилізації тексту
Сервіс інтеграції зі сторонніми продуктами	Основними функціями даного сервісу є: - Взаємодія системи з сервісом контролю версій проєкту Github. - Взаємодія системи з сервісами Google (пошта: авторизація) - Взаємодія системи з корпоративним месенджером Slack.
Сервіс менеджменту задач проєкту	Основними функціями даного сервісу є: - Створення нових задач проєкту - Редагування задач проєкту - Видалення задач проєкту - Переведення статусу задач - Присвоєння задачам критеріїв для подальшого підбору працівників - Встановлення пріоритетів задачам - Зміна статусів задач - Представлення списку дійсних задач

3.5 Опис клієнтської архітектури

Архітектура клієнтської частини реалізована як SPA (Single page application).

Односторінковий застосунок (англ. single-page application, SPA), також відомий як односторінковий інтерфейс (англ. single-page interface, SPI) - це веб-застосунок чи вебсайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою.

У односторінковій програмі зазвичай увесь необхідний код (HTML, JavaScript та CSS) завантажується на сторінку або динамічно завантажується за необхідності, як правило, у відповідь на операції користувача. Ця сторінка не оновлює та не перенаправляє користувача на іншу сторінку під час використання. Взаємодія з односторінковою програмою, як правило, передбачає динамічне спілкування з веб-сервером.

Більшість сучасних веб додатків реалізовано за монолітною архітектурою.

Монолітна архітектура - це архітектурний підхід, в якому вся основна логіка додатка зібрана в одному місці. Монолітний додаток складається з одношарового об'єднання різних компонентів в одне ціле. Тобто всі команди працюють над одним репозиторієм проєкту. Зміна будь якої частини проєкту зазвичай має вплив на інші.

Загальна структура монолітного проєкту представлена на рисунку 3.6.

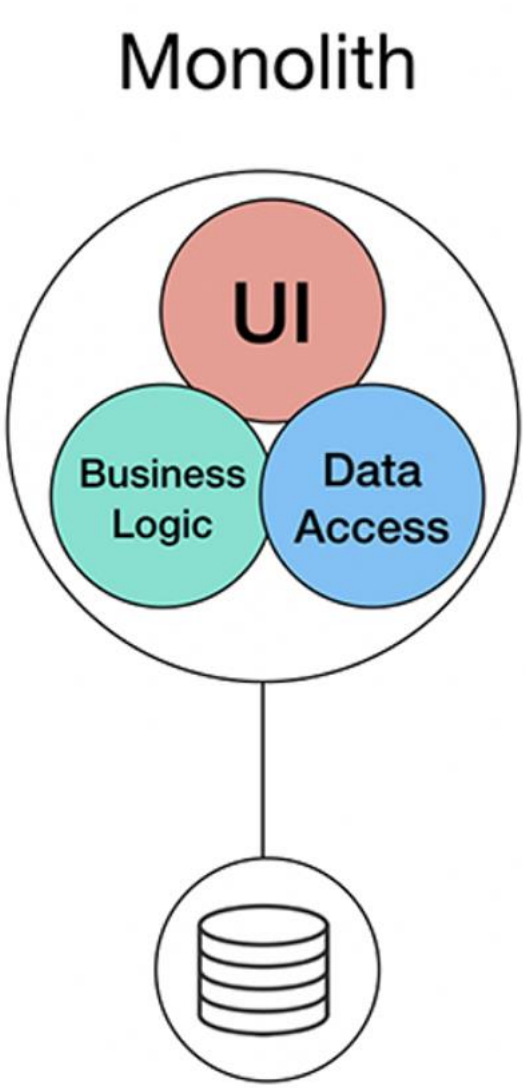


Рисунок 3.6 Загальна структура монолітного проєкту

Оскільки монолітна система погано підходить для великих проєктів в силу сильної прив’язаності її компонентів, для даної системи використана архітектура мікрофронтендів.

Мікрофронтенд (micro frontend) - архітектурний підхід, в якому незалежні додатки зібрані в один великий додаток. Він дає можливість об'єднати в одному додатку різні віджети або сторінки, написані різними командами з використанням різних технологій.

Головні переваги мікрофронтенд-підходу:

- модульна архітектура. Окремі віджети або сторінки - це повністю незалежні додатки;
- швидкість тестування. Зміни в одному віджеті або сторінці можна

протестувати ізольовано і тільки в цьому додатку, не витрачаючи часу на тестування всього іншого функціоналу;

- паралельне розгортання. Окремі віджети або сторінки можуть і повинні розгортатися незалежно.

Загальна структура мікрофронтед архітектури представлена на рисунку 3.7.

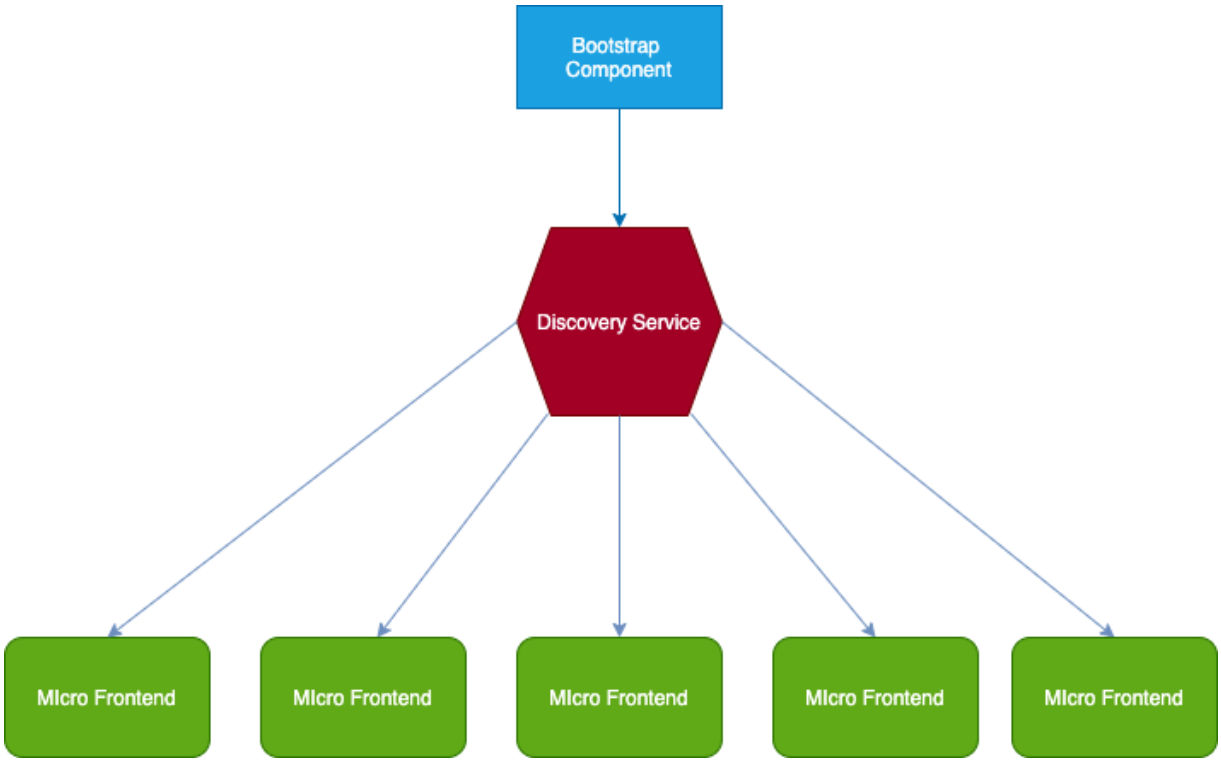


Рисунок 3.7. Загальна структура архітектури мікрофронтенду

Принцип роботи додатку з мікрофронтед архітектурою слідуючий. При старті додатку першим запускається Bootsrap Component, його завданням є визначити який сервіс необхідно завантажити, потім зробити запит на Discovery Service та отримати дані які необхідно для початку процесу запуску необхідного сервісу. Discovery Service - це допоміжний додаток, його роль полягає в тому щоб зареєструвати кожен з сервісів і тримати в собі інформацію необхідну для завантаження кожної частини додатку.

3.6 Специфікація функцій

Функції класів програмного забезпечення досить багато. В таблиці 3.3 наведені основні функції класів програмного забезпечення.

Таблиця 3.3.

Функції класів програмного забезпечення

Назва	**Примітка**
auth(user)	Виконує авторизацію в програмі
loadInitialData()	Виконує завантаження наявних даних із БД у програму
regUser()	Виконує реєстрацію нового користувача
validateSearch()	Виконує валідацію даних при пошуку об’єктів
handleObjectButtonClicked()	Застосовується для редагування даних в таблиці задач проєкту
handleAddUserClicked()	Застосовується для додавання даних до таблиці користувачів
handleEditUserClicked()	Застосовується для редагування даних в таблиці користувачів
handleDeleteUserClicked()	Застосовується для видалення даних з таблиці користувачів
getTaskCriteria()	Отримати список крітерій задачі
dbConnect(url)	Виконує підключення до бази даних
userLogout()	Виконує вихід користувача із системи
printObjectDetails(obj)	Застосовується для друку даних про задачу
sendEmail()	Застосовується для відправлення листів працівникам
addArticle	Застосовується для додавання нової інформативної сторінкі

3.7 Структура бази даних

Цілі створення бази даних:

– Давати повний та точний опис всіх ресурсів.

– Скоротити до мінімуму час працівників підприємства.

– Створити базу даних, яка буде актуальною та буде зберігати інформацію про персонал, клієнтів та об’єкти нерухомості.

– Повинна бути можливість вносити зміни в дані і поповнювати новими даними.

Для того, щоб досягти цих цілей треба:

– Мати підтверджену та своєчасну інформацію про ресурси.

– Підтримувати актуальність інформації в базі.

– Оперативно отримувати будь-яку інформацію з бази даних.

Інформація, яка має бути представлена в структурі даних:

1. Сутність «Користувач».
   - Унікальний ідентифікатор
   - Дата реєстрації
   - Електронний адрес
   - Пароль

2. Сутність «Задача проєкту».
   - Унікальний ідентифікатор
   - Статус задачі
   - Опис задачі
   - Заголовок
   - Коментарі
   - Унікальний ідентифікатор інформаційної сторінки
   - Унікальний ідентифікатор співробітника
   - Зв’язані файли
   - Пріоритет

- Тип
- Дата створення
- Дата останнього оновлення
- Час на виконання
- Унікальний ідентифікатор зв’язаних задач
- Критерії задачі
- Компоненти
- Зв’язані сервіси
- Лейбли

3. Сутність «Дошка проєкту».
    - Унікальний ідентифікатор
    - Назва дошки
    - Тип дошки
    - Дата створення
    - Опис
    - Дата останнього оновлення
    - Унікальні ідентифікатори критеріїв
    - Тип фільтрів

4. Сутність «Працівник».
    - Унікальний ідентифікатор
    - Електронна адреса
    - Пароль
    - Ім’я
    - Роль
    - Прізвище
    - Навички
    - Ідентифікатор групи користувачів
    - Зображення

- Рівень доступів
- Групи

5. Сутність «Коментар задачі».
   - Унікальний ідентифікатор
   - Текст
   - Код задачі
   - Код автора
   - Дата

6. Сутність «Інформаційна сторінка».
   - Унікальний ідентифікатор
   - Заголовок
   - Контент

7. Сутність «Спринт».
   - Назва
   - Дата початку
   - Дата закінчення
   - Опис цілі спринту

8. Сутність «Критерій»
   - Назва
   - Опис
   - Аватар

Взаємовідносини між сутностями зображені на рисунку 3.4:

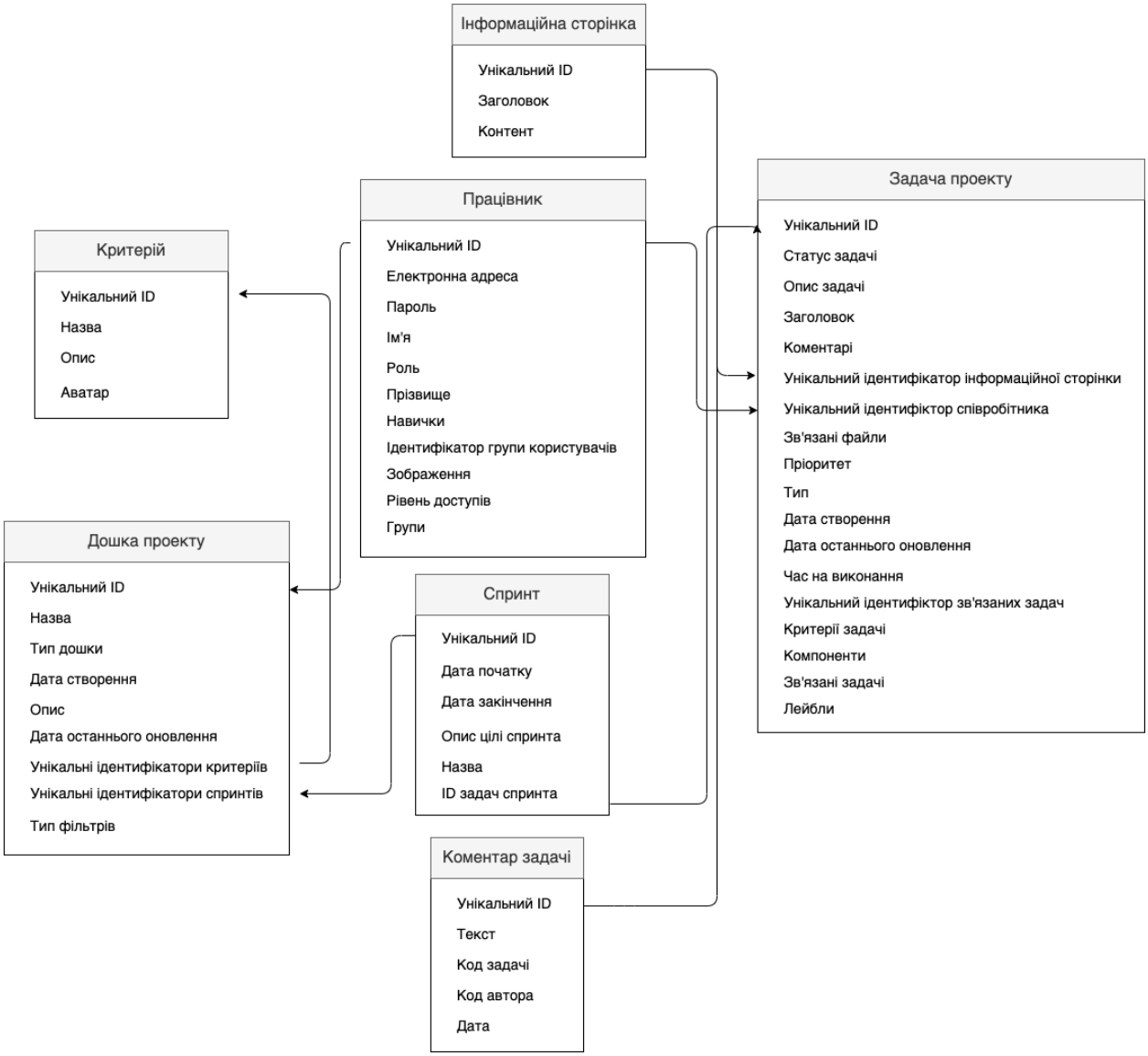


Рисунок. 3.8. Взаємозв’язок та структура сутностей бази даних

Висновки

В даному розділі було виконано проєктування автоматизованої системи аналізу та розподілу задач проєкту. На даному етапі були побудовані моделі логічного та фізичного представлення системи. Для опису архітектури були побудовані та відображені наступні діаграми та схеми:

– Структурна схема компонентів;

– Діаграма класів;

– Діаграма сутність-зв’язок;

– Діаграма послідовності;

– Діаграма розгортання;

Завдяки наявності діаграми класів була наведена специфікація основних функцій системи.

Також була спроєктована та описана структура бази даних та її внутрішньої ієрархії.

РОЗДІЛ 4
ПРОТОТИП АВТОМАТИЗОВАНОЇ СИСТЕМИ

4.1 Керівництво користувача

Для запуску додатку необхідно відкрити браузер, та перейти по адресі розгортання додатку. З'явиться головна сторінка авторизації в системі: відображена на рисунку 4.1:

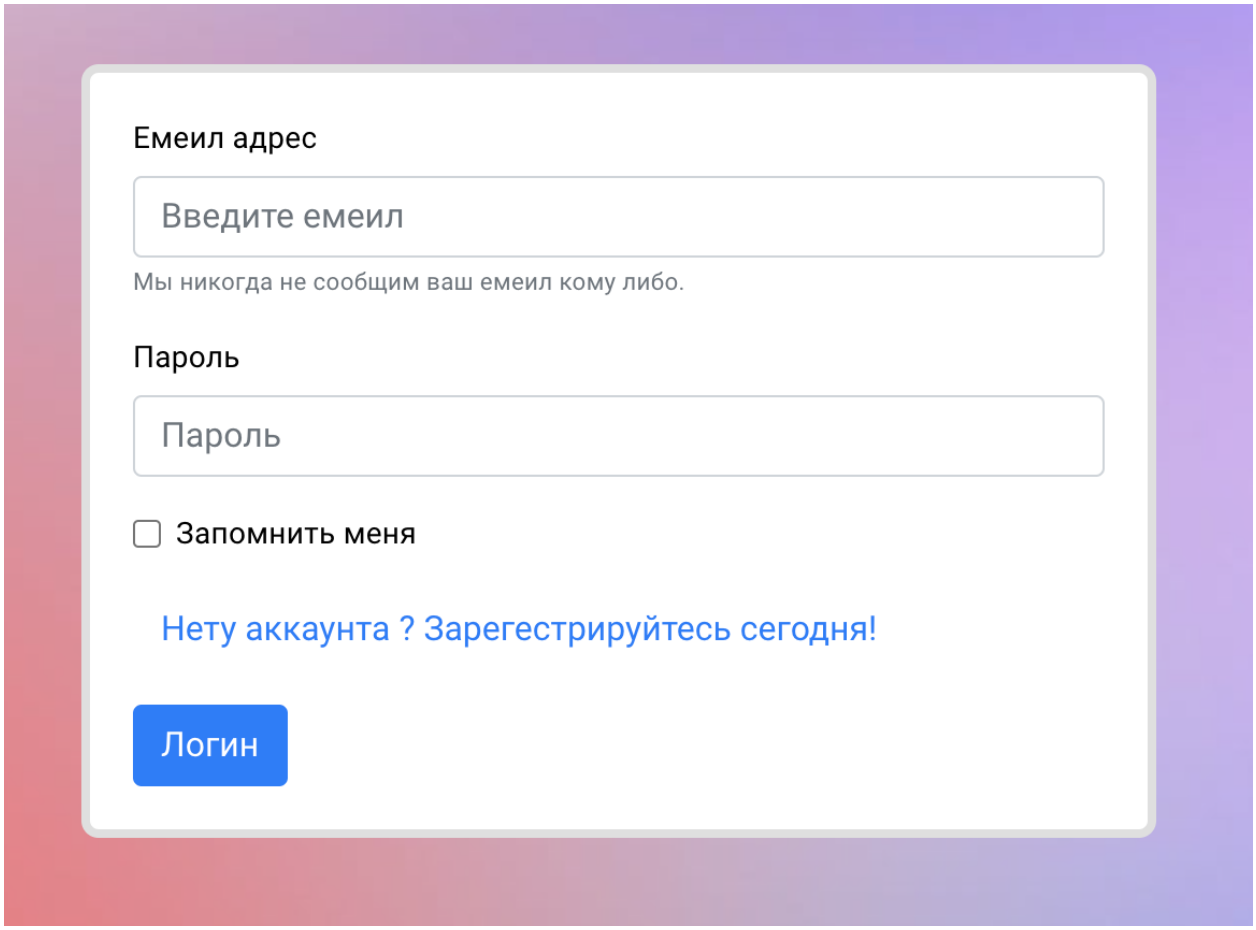


Рисунок. 4.1. Вікно авторизації

Користувачу пропонується ввести електронну адресу та пароль від його облікового запису та можливість зберегти дані для подальших спроб входу.

При введенні даних та їх некоректності, користувач отримає повідомлення про помилку авторизації у вигляді відображеному на рисунку 4.2.

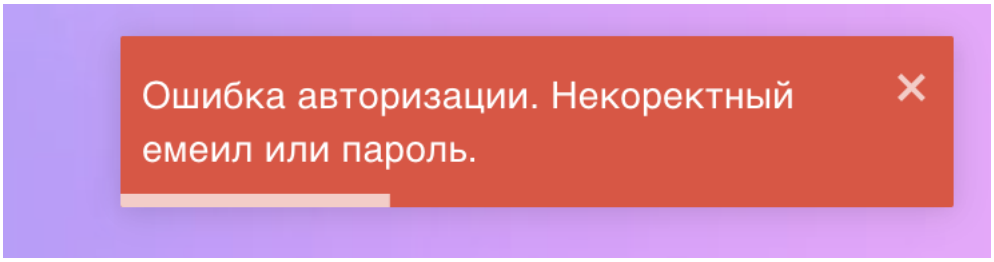


Рисунок. 4.2. Повідомлення про помилку авторизації

У випадку якщо у користувача не створений обліковий запис, він можу перейти на сторінку реєстрації натиснувши на посилання для переходу на сторінку реєстрації (відображену на рисунку 4.3).

Рисунок. 4.3. Посилання на сторінку реєстрації

Сторінка реєстрації для реєстрації користувачів відображена на рисунку 4.5. У користувача є можливість ввести необхідні дані для створення облікового запису або перейти на сторінку авторизації. У випадку успішної реєстрації, користувач отримає повідомлення у вигляді відображеному на рисунку 4.4.

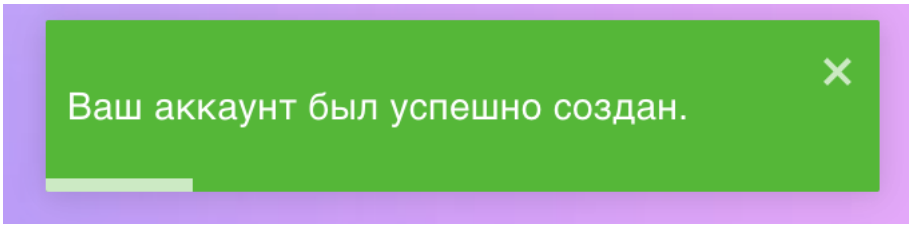


Рисунок. 4.4. Повідомлення про успішне створення облікового запису

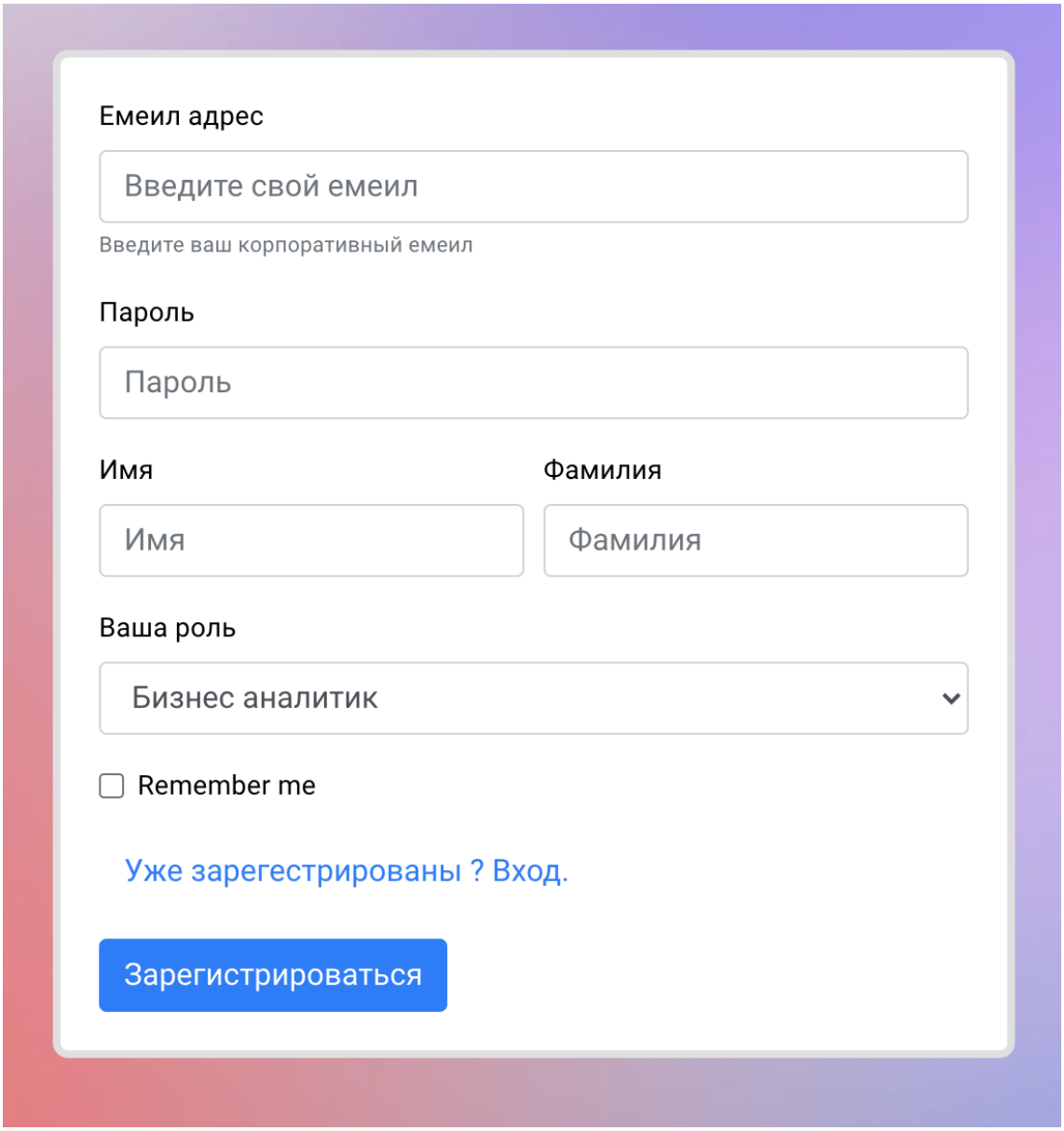


Рисунок. 4.5. Сторінка для реєстрації в системі.

При успішній авторизації в системі користувач буде переведений на головну сторінку системи, відображену на рисунку 4.6.

Головна сторінка складається з панелі навігації відображеної на рисунку 4.7. та поточної активної вкладки, у випадку першого входу в систему це буде сторінка спринтів проєкту.

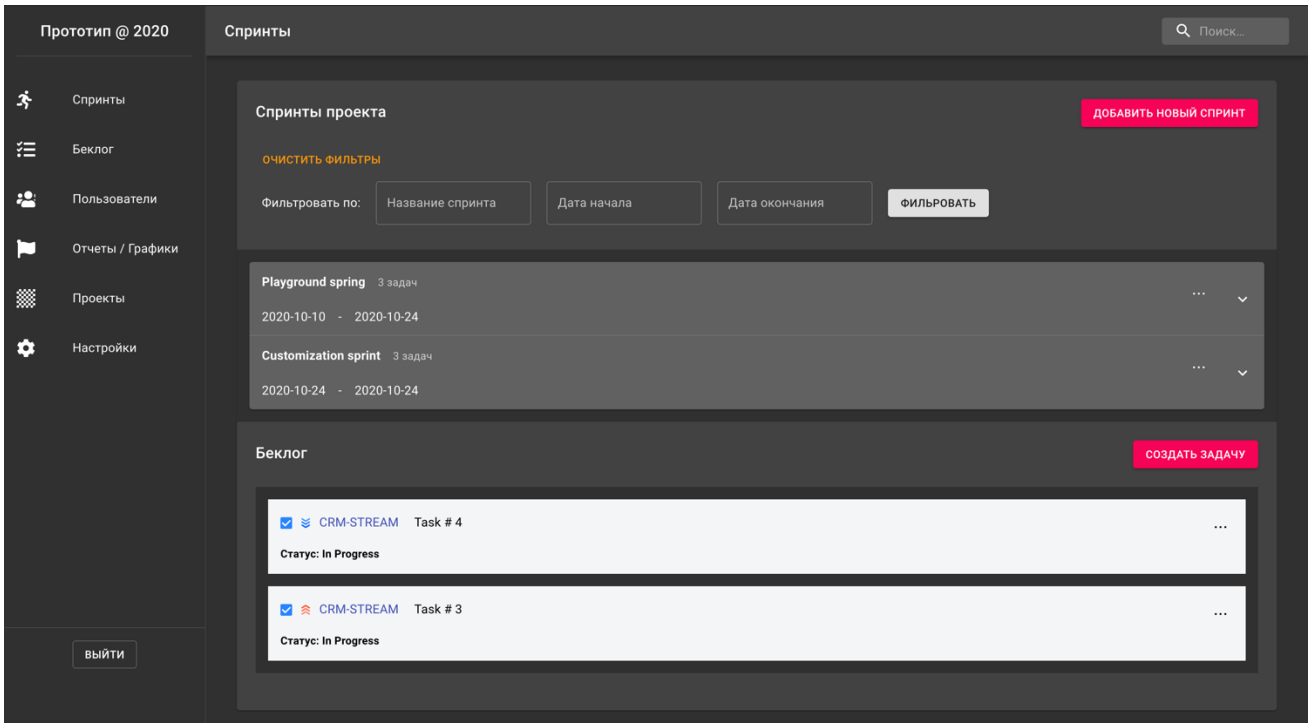


Рисунок. 4.6. Головна сторінка системи.

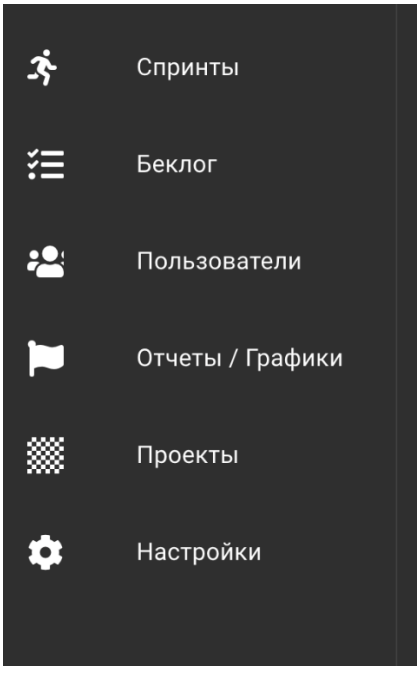


Рисунок. 4.6. Панель навігації.

4.2 Сторінка спринтів проєкту.

Список спринтів проєкту надано в виді списку відображеного на рисунку 4.7. Кожен елемент списку може розкриватися по натисканню та показувати список задач (відображено на рисунку 4.8.) які входять в кожен етап та кнопку для конфігурації цих задач.

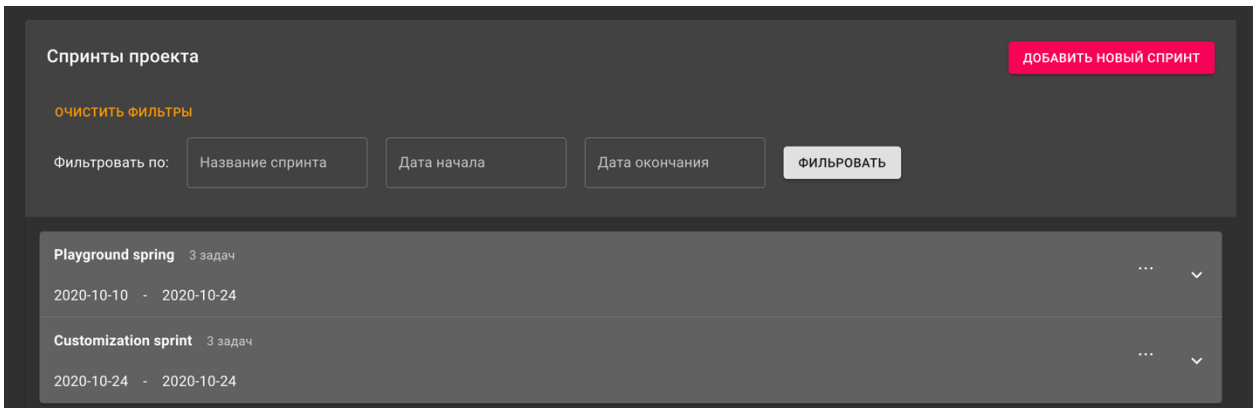


Рисунок. 4.7. Список спринтів проєкту.

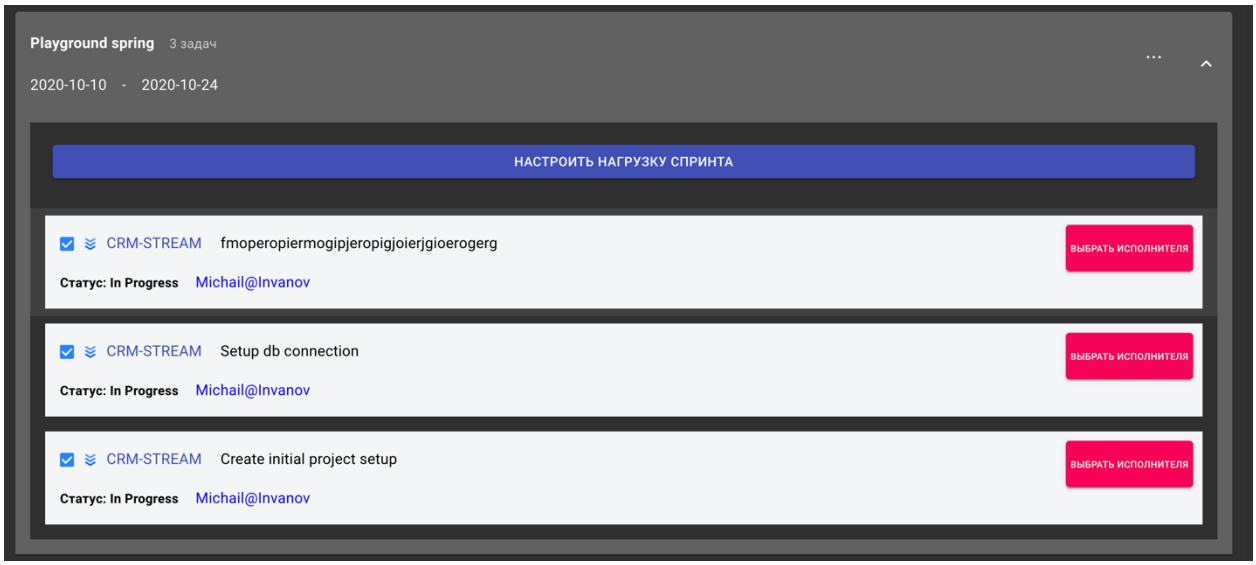


Рисунок. 4.8. Розгорнутий вид елементу списку спринтів.

Нижче списку спринтів відображений беклог проєкту і всі задачі які на даний момент не увійшли до виконання в будь-яких спринтах. Він відображений на рисунку 4.9.

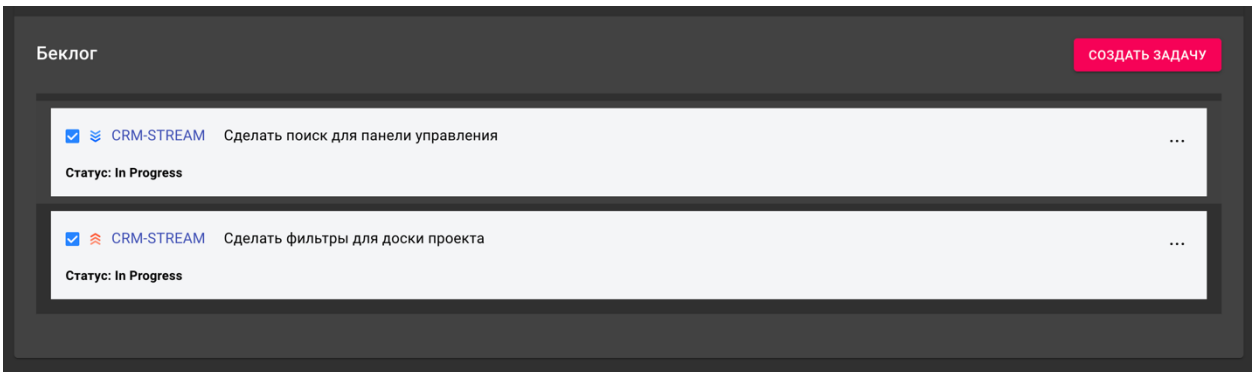


Рисунок. 4.9. Беклог проєкту.

4.3 Сторінка менеджменту користувачів проєкту

По кліку на кнопку «Пользователи» (відображена на рисунку 4.10.) в боковому меню навігації, відкриється вікно для відображення діючих користувачів системи, рисунок 4.11.

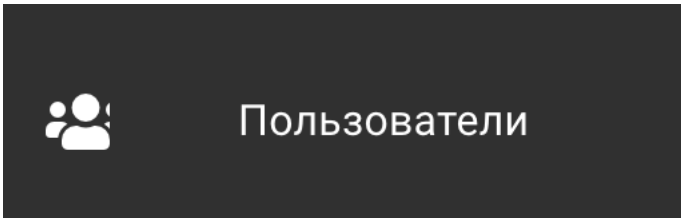


Рисунок. 4.10. Кнопка для переходу на сторінку менеджменту користувачів.

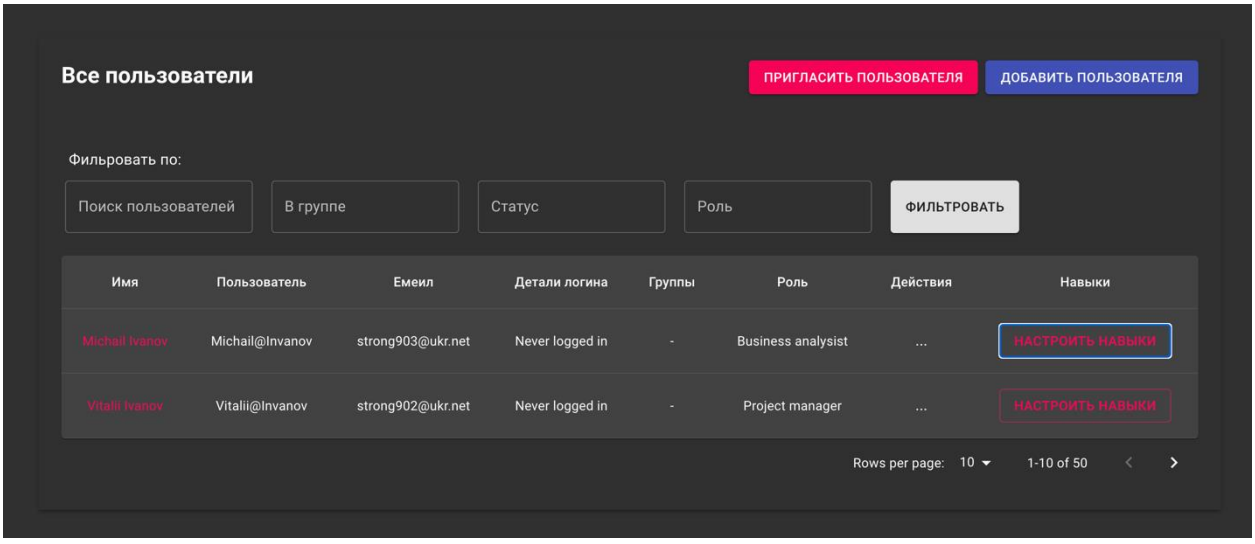


Рисунок. 4.11. Сторінка для менеджменту користувачів.

На сторінці користувачів є можливість створення запрошення, додавання нових користувачів в систему, конфігурації списку їх навичок, пошуку та фільтрації серед них.

4.4. Сторінка менеджменту дошок.

По кліку на кнопку « Проекты» (відображена на рисунку 4.12.) в боковому меню навігації, відкриється вікно для відображення існуючих проєктів системи, рисунок 4.13.

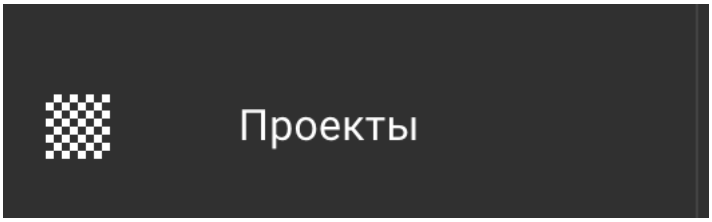


Рисунок. 4.12. Кнопка для переходу на сторінку менеджменту проєктів.

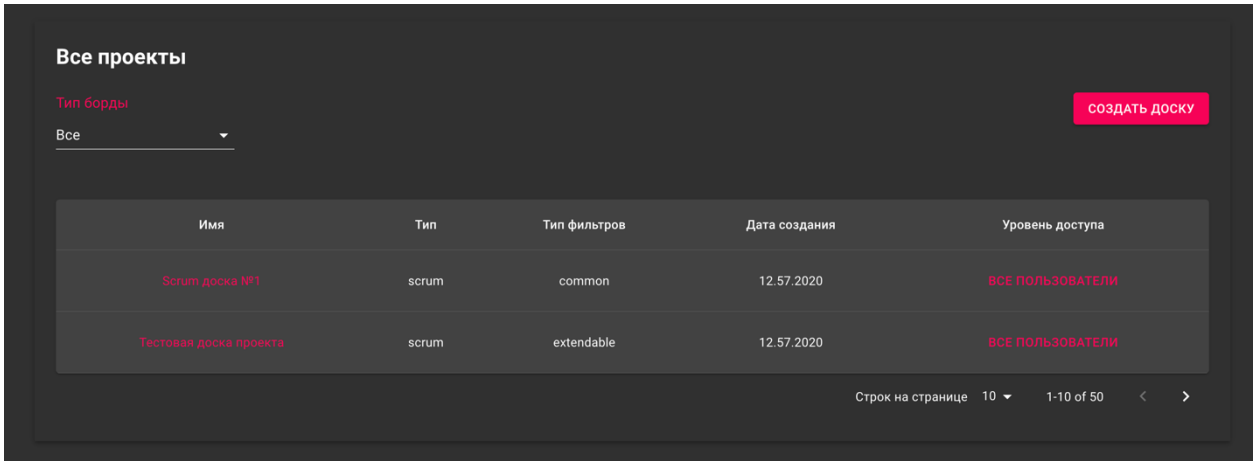


Рисунок. 4.13. Сторінка для менеджменту проєктів.

На сторінці проєктів є можливість створення нових дошок, відображення та фільтрації по типу.

4.5. Сторінка налаштування системи.

По кліку на кнопку «Настройки» (відображена на рисунку 4.14.) в боковому меню навігації, відкриється вікно для відображення налаштувань системи, рисунок 4.15.

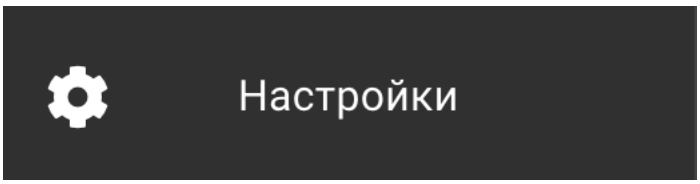


Рисунок. 4.14. Кнопка для переходу на налаштувань системи.

Рисунок. 4.15. Сторінка для налаштувань системи.

Налаштування діляться на «Загальні» та «Налаштування проєктів». Загальні налаштування застосовуються для всієї системи та кожного проєкту. Налаштування проєктів - це налаштування кожного проєкту особисто, всі зміни будуть застосовані лише до нього.

На рисунках 4.16. та 4.17. відображені розгорнуті списки загальних та проєктних налаштувань відповідно.

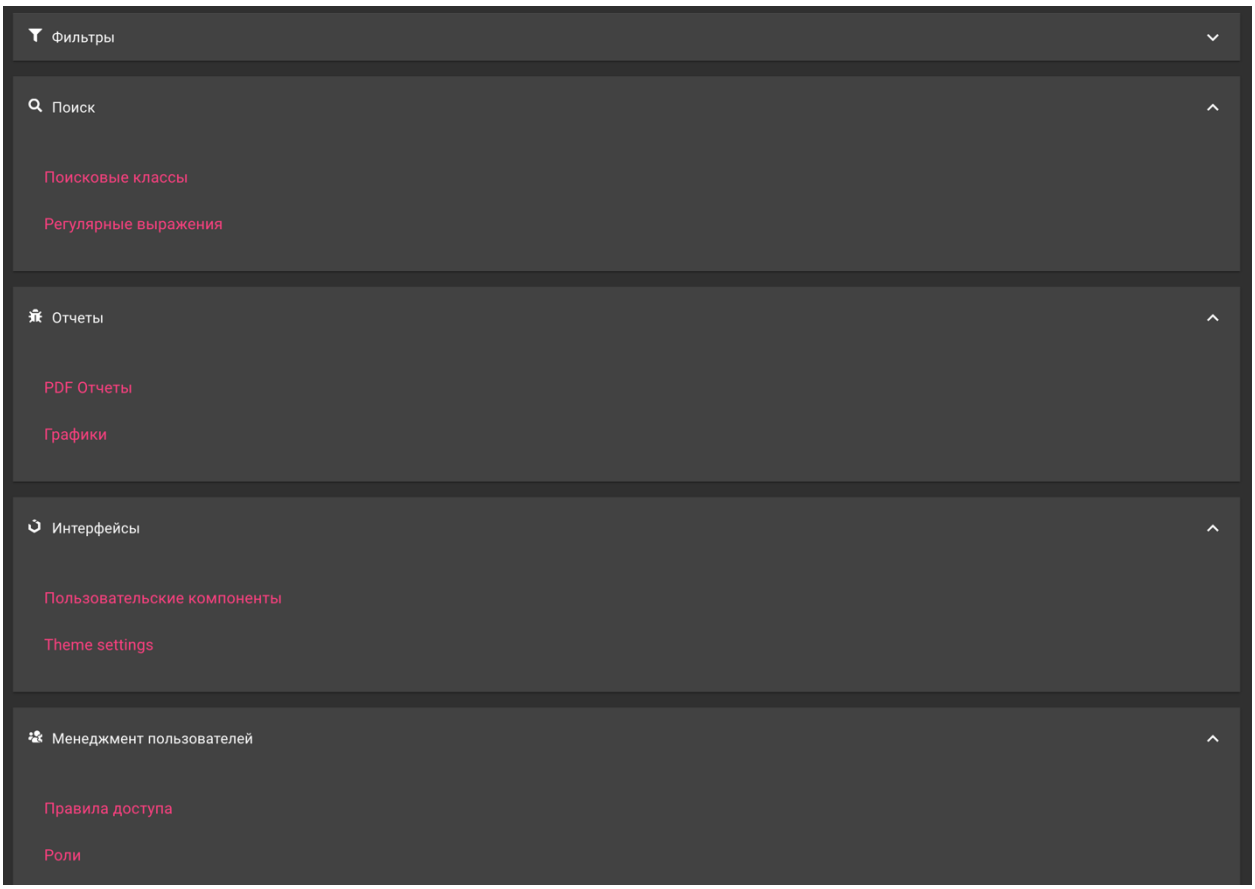


Рисунок. 4.16. Загальні налаштування системи.

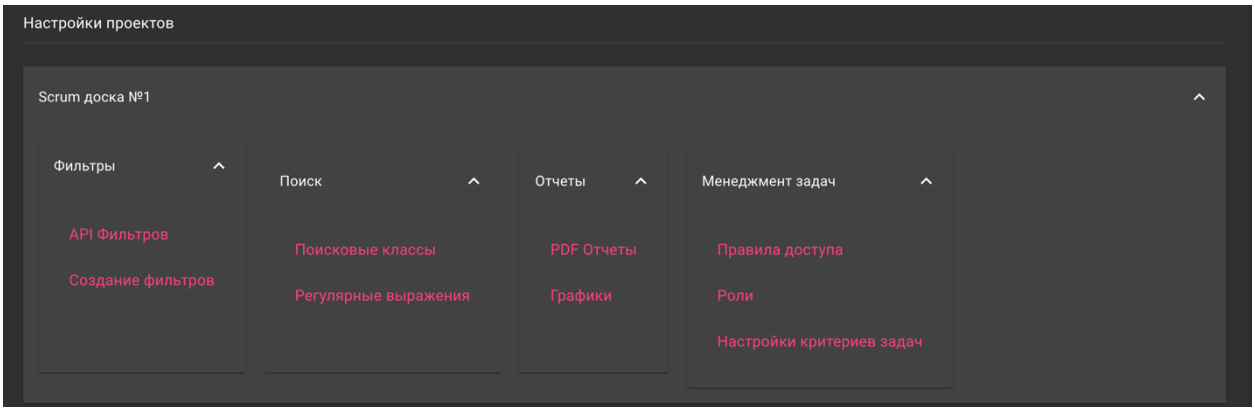


Рисунок. 4.17. Налаштування проєкту.

4.6. Сценарій: автоматичне підбір виконавця для задачі проєкту.

Для авторизації в системі необхідно заповнити поля в формі авторизації як показано на рисунку 4.18. та натиснути кнопку «Вход».

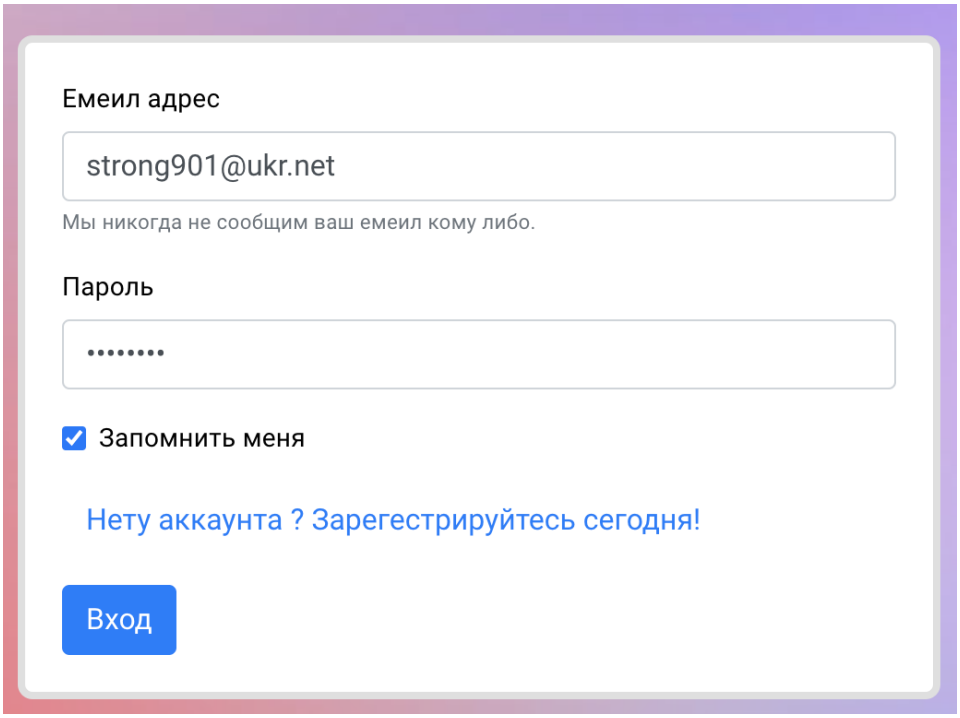


Рисунок. 4.16. Загальні налаштування системи.

Далі потрібно перейти на сторінку проєктів та натиснути кнопку «Создать доску» як показано на рисунку 4.18. В результаті відкриється форма створення нової дошки проєкту відображена на рисунку 4.17.

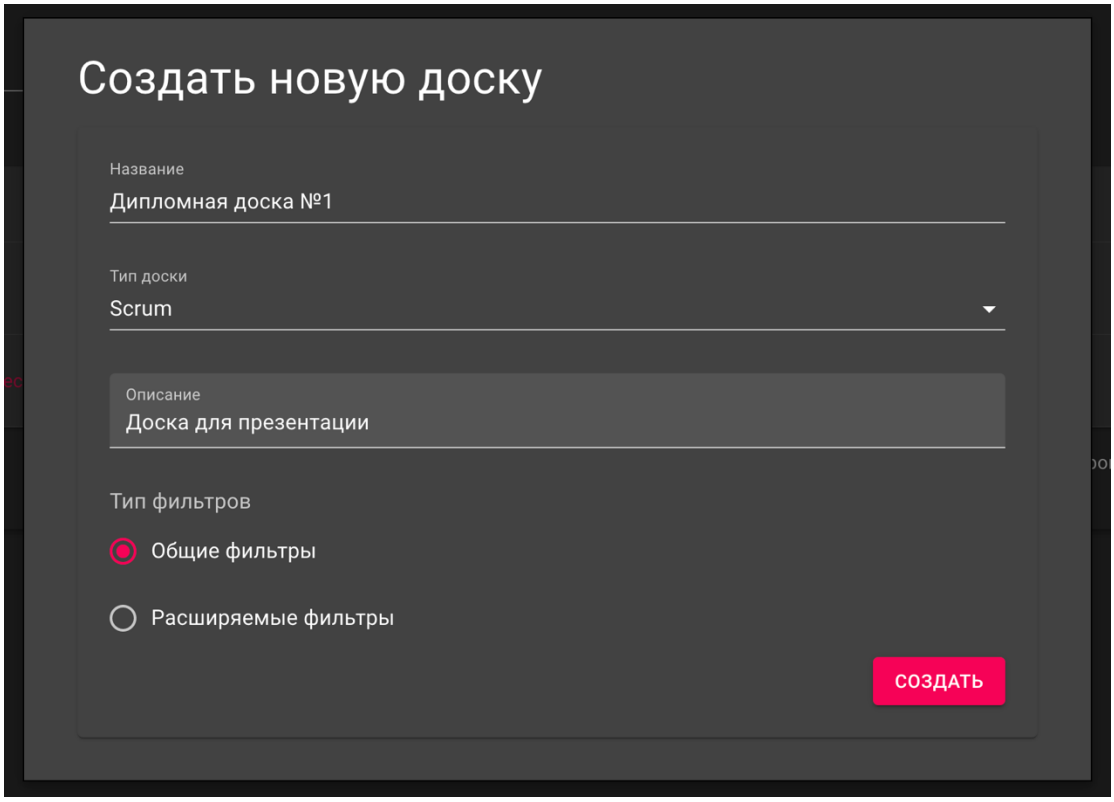


Рисунок. 4.17. Форма створення нової дошки проєкту.

Рисунок. 4.18. Форма створення нової дошки проєкту.

Після заповнення необхідних полей натиснувши на кнопку «Создать» ми отримуємо повідомлення про створення нової дошки яке показано на рисунку 4.19. та побачимо що створення дошка додалась в список існуючих проєктів, рисунок 4.20.

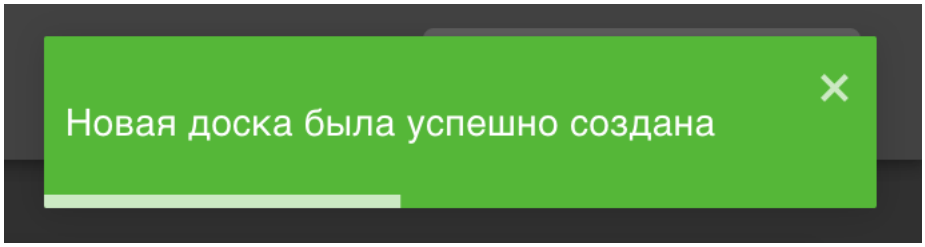


Рисунок. 4.19. Форма створення нової дошки проєкту.

Рисунок. 4.20. Форма створення нової дошки проєкту.

Далі потрібно налаштувати список доступних критеріїв проєкту для можливості подальшого автоматичного підбору виконавців для задач спринтів. Для цього потрібно перейти на сторінку налаштувань критеріїв для даного проєкту і натиснути кнопку «Настройки критериев задач» як показано на рисунку 4.21.

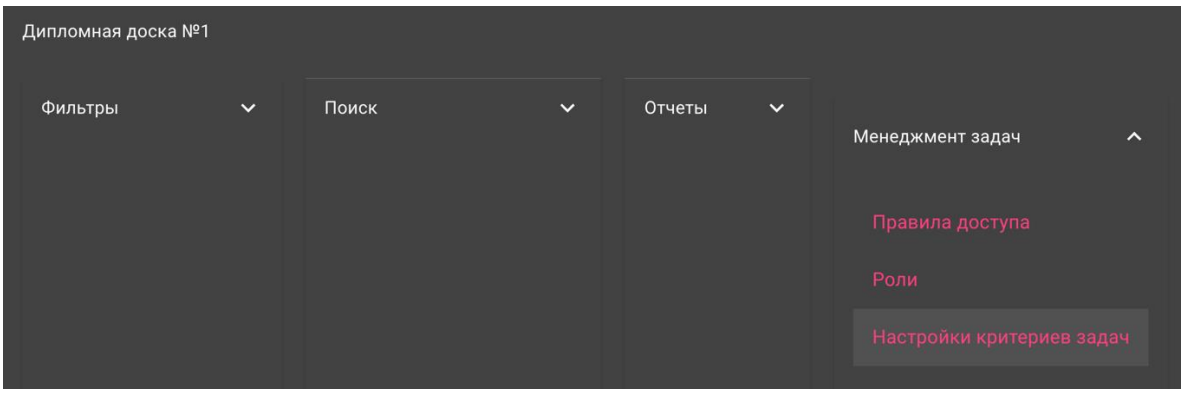


Рисунок. 4.21. Перехід в налаштування задач проєкту.

В результаті ми перейдемо на сторінку налаштувань критеріїв (рисунок 4.22.) на якій буде відображено список всіх існуючих критеріїв задач системи, можливості створення нових, та додавання критеріїв в проєкт.

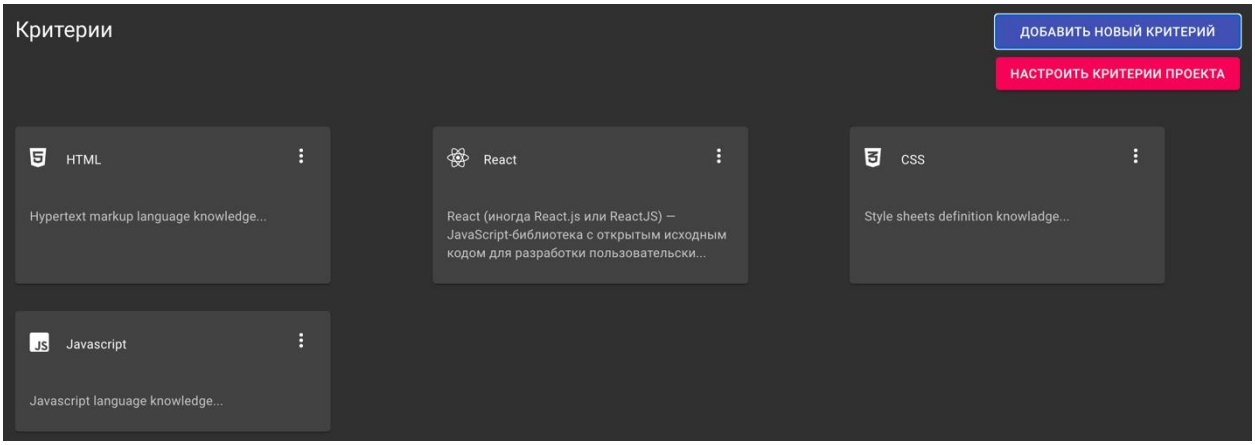


Рисунок. 4.22. Список існуючих критеріїв системи.

Для додавання нового критерію потрібно натиснути «Добавить новый критерий». Відкриється форма створення відображена на рисунку 4.23., заповнивши всі поля та натиснувши «Создать», користувач отримає повідомлення про успішне створення і критерій буде відображений в списку всіх критеріїв системи.

Щоб налаштувати список критеріїв проєкту, потрібно натиснути на кнопку «Настроить критерии проєкта». В результаті відкриється форма налаштування відображена на рисунку 4.24. В лівій колонці відображений список всіх доступних критеріїв системи, а в правій - список критеріїв даного проєкту.

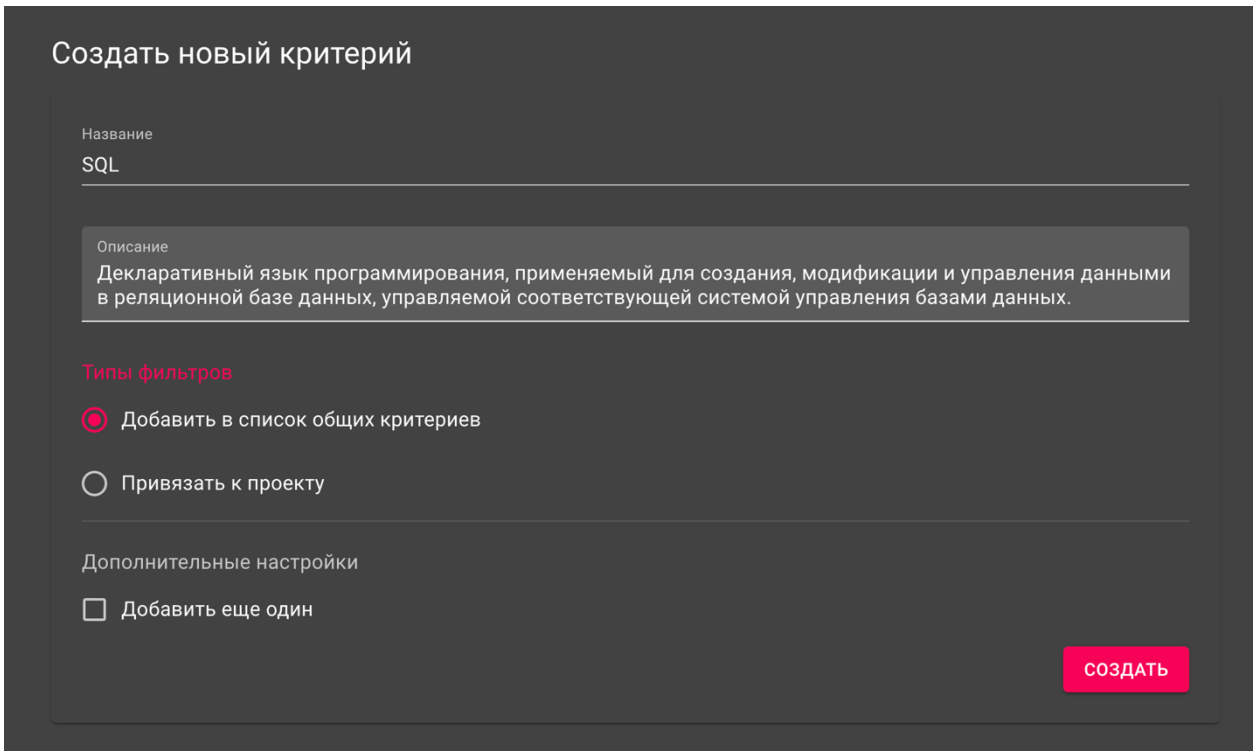


Рисунок. 4.23. Список існуючих критеріїв системи.

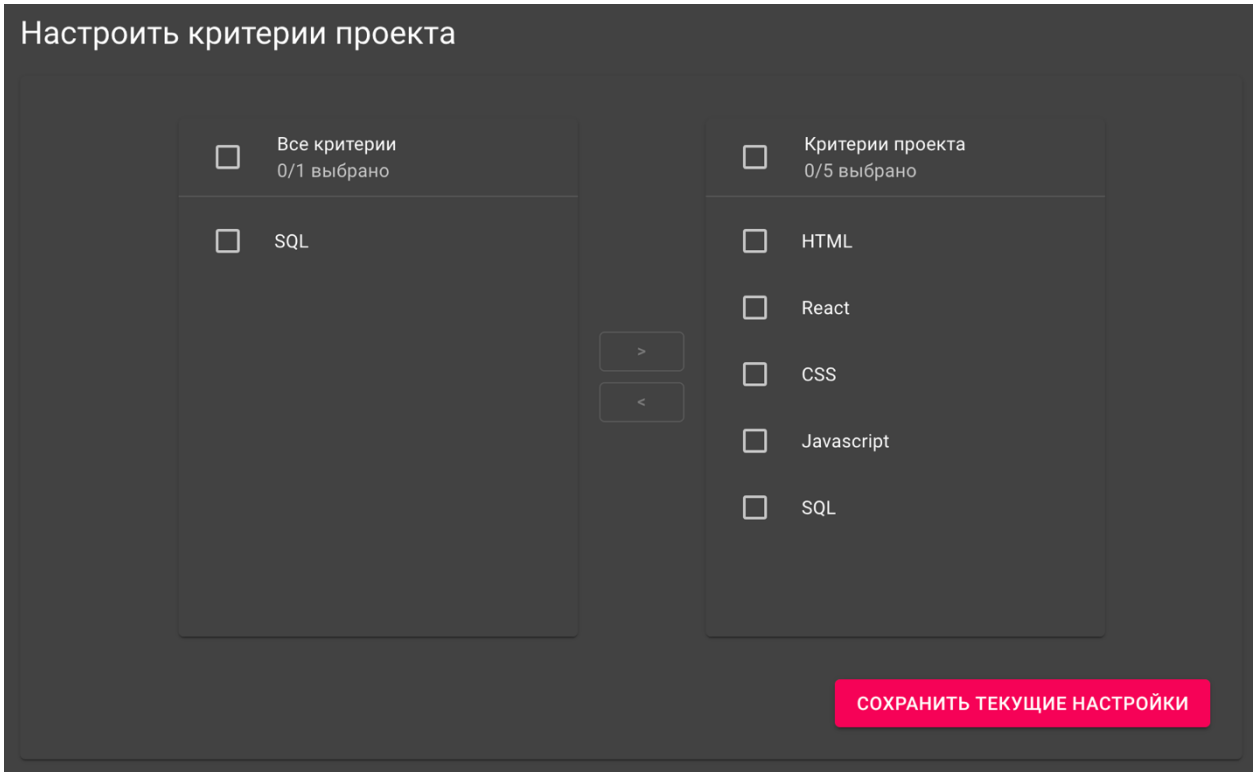


Рисунок. 4.24. Форма налаштування критеріїв проєкту.

Вибравши необхідні критерії та натиснувши «Сохранить текущие настройки». Ми побачимо оновлений список критеріїв відображений на рисунку 4.25.

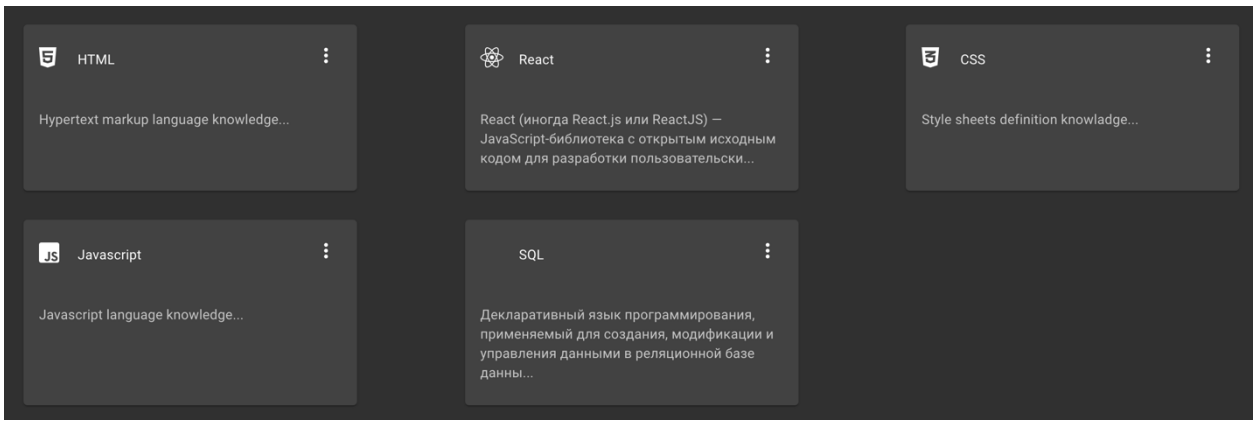


Рисунок. 4.25. Оновлений список доступних критеріїв проєкту.

Для можливості роботи автоматичного підбору виконавців для задач, потрібно налаштувати навички учасників проєкту. Для цього необхідно перейти на сторінку менеджменту користувачів натиснувши на кнопку «Пользователи» із меню навігації. Вибравши цікавого нам учасника, потрібно натиснути на кнопку «Настроить навыки» в таблиці учасників як показано на рисунку 4.26.

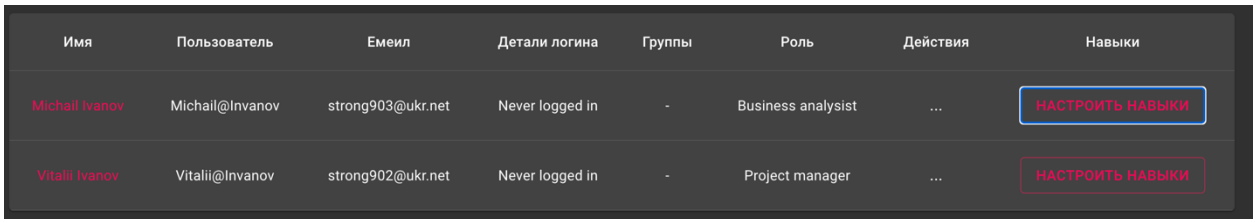


Рисунок. 4.26. Приклад виклику форми налаштування навичок учасника проєкту.

В результаті ми побачимо відкриття форми для налаштування навичок для даного користувача. Форма представляє собою список з повзунків, кожен з них відображає певний критерій проєкту та може приймати значення від 0 до 100 яке відповідає рівню володіння даним умінням учасником.

Налаштувавши всі вміння як показано на рисунку 4.27. та натиснувши кнопку «Сохранить», ми збережемо дану конфігурацію користувача.

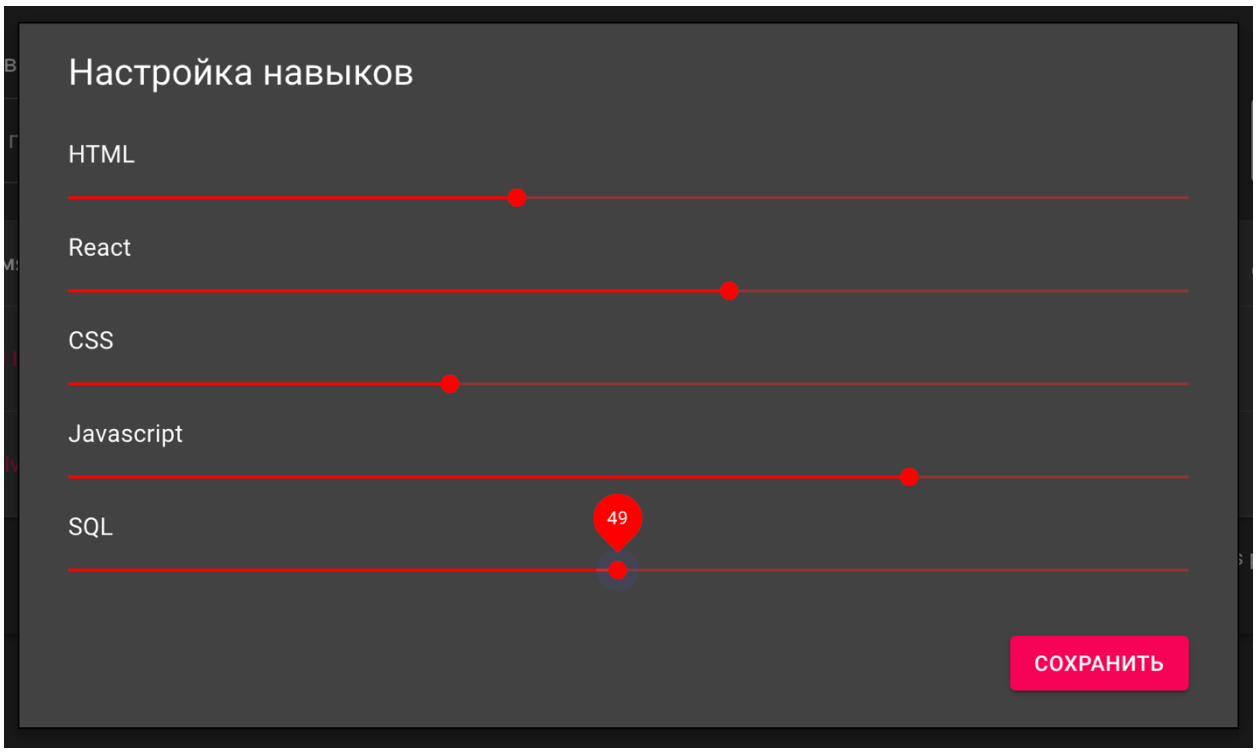


Рисунок. 4.26. Форма налаштування навичок користувача.

На даному етапі з налаштованими навичками учасників проєкту та критеріями потрібно перейти на сторінку спринтів натиснувши кнопку «Спринты» в меню навігації.

Далі необхідно додати задачі в спринт натиснувши кнопку «Настроить нагрузку спринта» як показано на рисунку 4.27. В результаті побачимо форму додання задач в спринт показану на рисунку 4.28, заповнивши яку задачі будуть прибрані з беклогу та попадуть в список задач спринту.

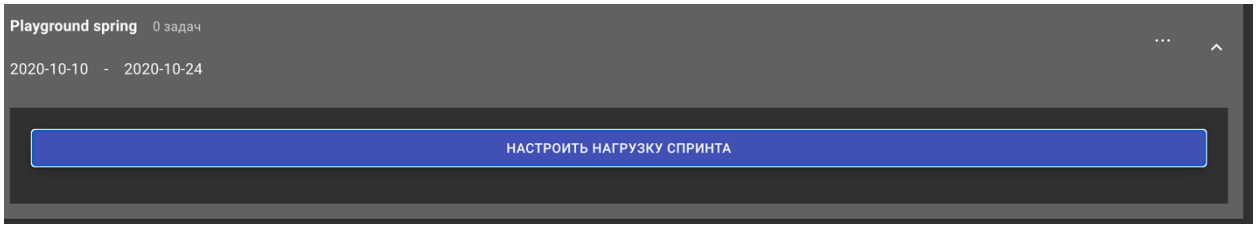


Рисунок. 4.27. Кнопка для відкриття форми налаштування навантаження спринта.

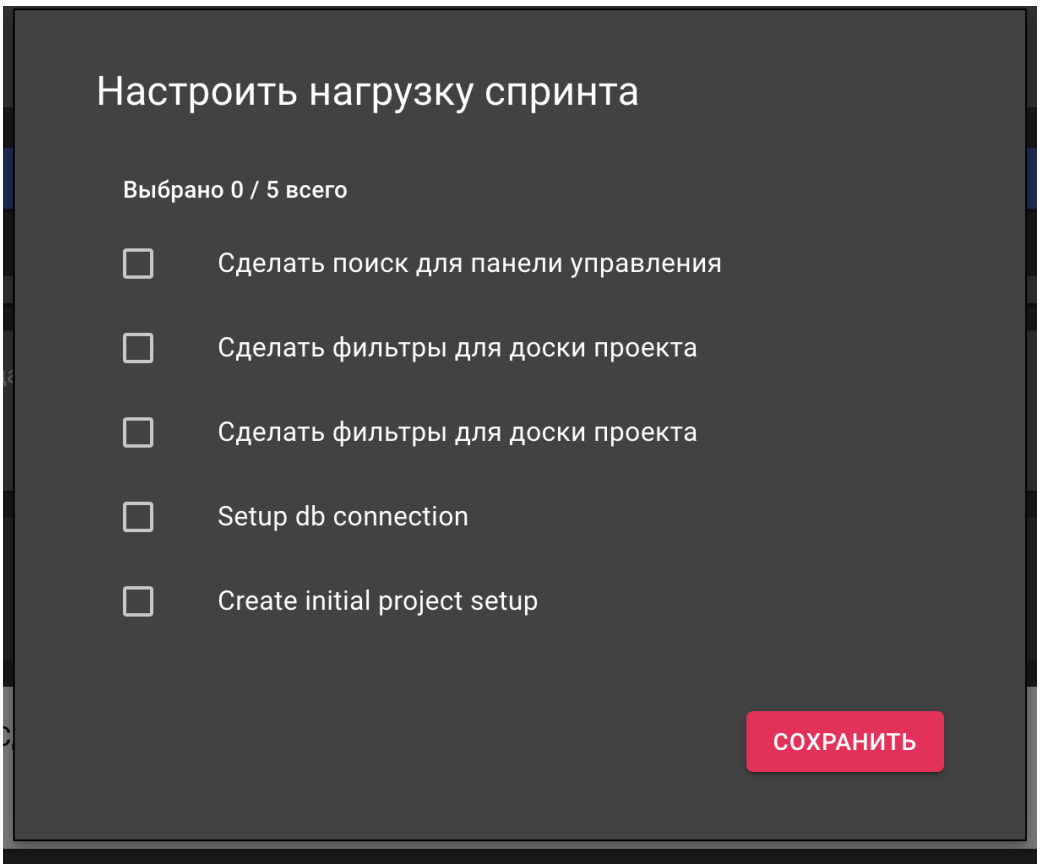


Рисунок. 4.28. Форма налаштування навантаження спринта.

Вибравши задачі та натиснувши кнопку «Сохранить», список задач спринта буде оновлений відповідно до рисунку 4.29.

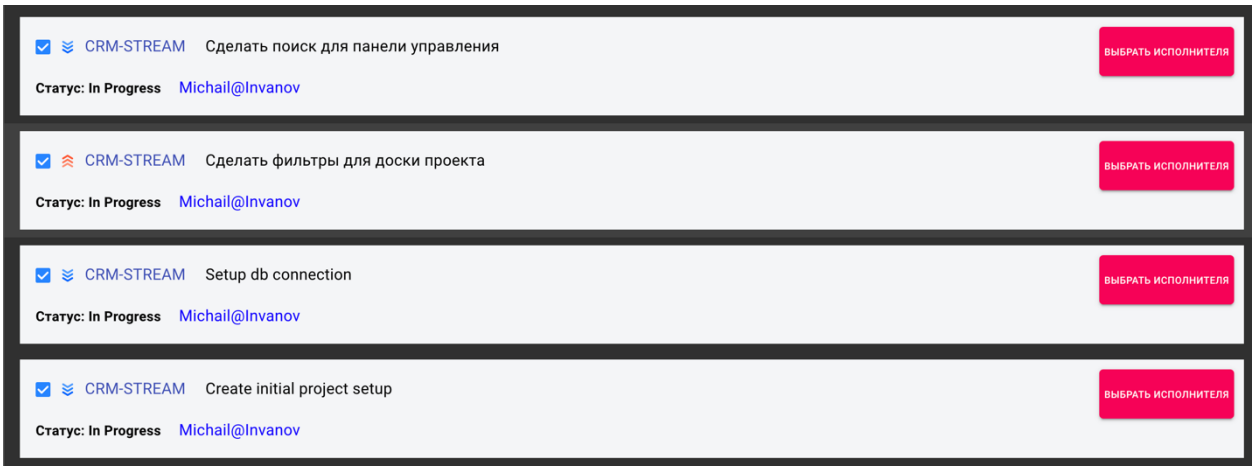


Рисунок. 4.29. Оновлений список задач спринта.

Для автоматичного підбору виконавців задачі потрібно натиснути кнопку «Выбрать исполнителя». В результаті відкриється форма автоматичного підбору користувачів показана на рисунку 4.30.

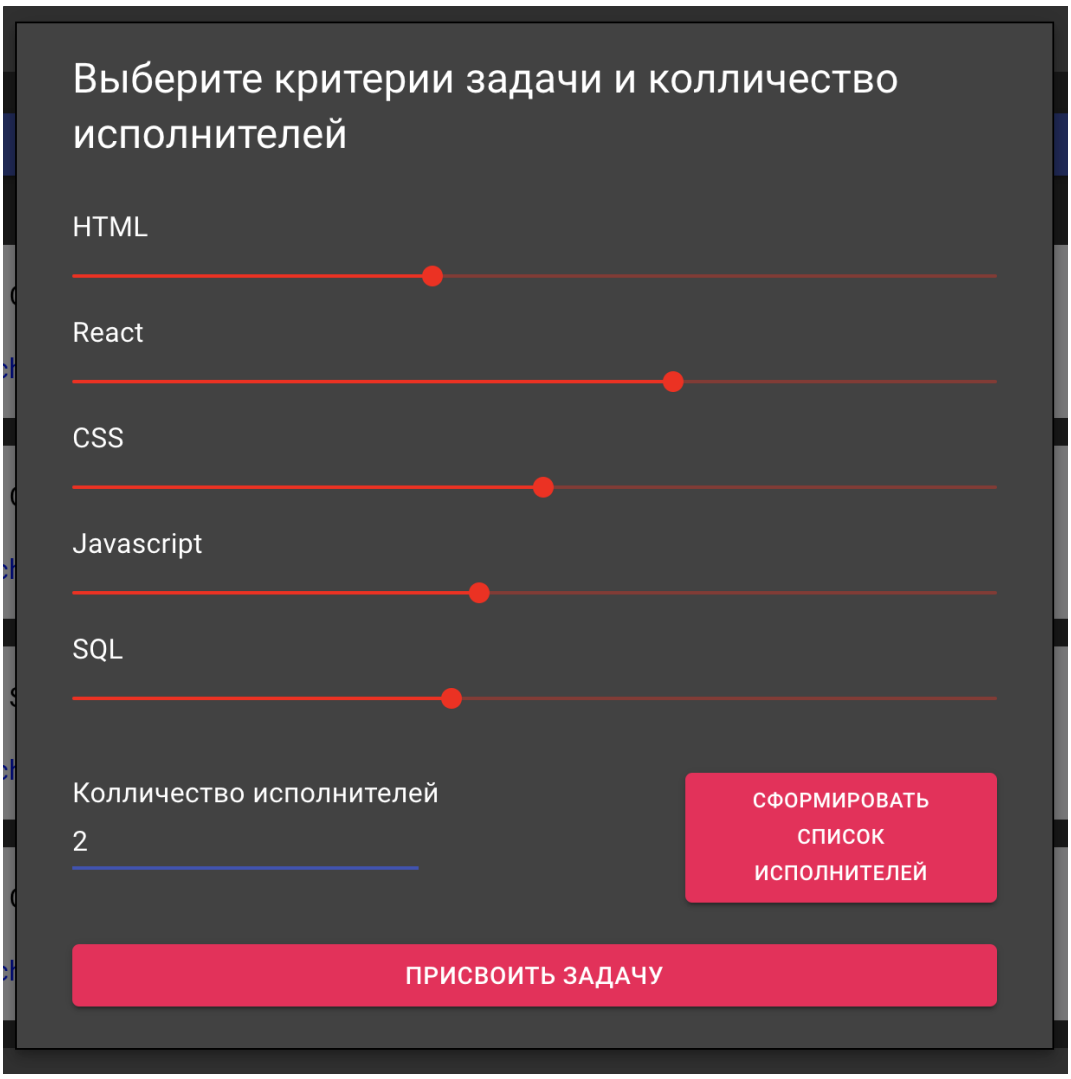


Рисунок. 4.30. Форма автоматичного підбору виконаців задачі.

Вибравши критерії задачі, та кількість виконавців як показано на рисунку 4.30. потрібно натиснути на кнопку «Сформировать список исполнителей». В результаті буде сформований список підходящих виконавців як показано на рисунку 4.31.

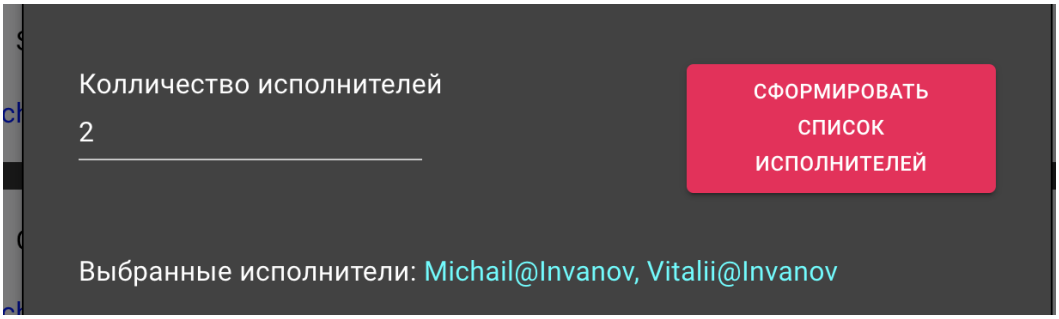


Рисунок. 4.31. Список автоматично підібраних виконавців задачі.

У разі згоди з результатом автоматичного підбору натиснувши на кнопку «Присвоить задачу», дані про задачу будуть оновлені новими даними про виконавців як показано на рисунку 4.32.

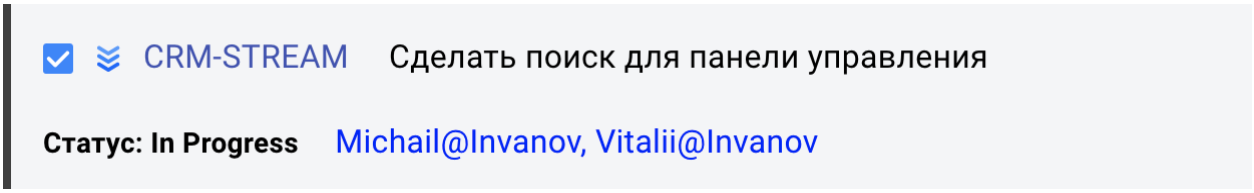


Рисунок. 4.32. Результат присвоєння задачі виконавців.

Висновки

Для опису роботи основних функцій програмного забезпечення було описано керівництво користувача. Були наведені креслення наступних екранних форм:

- Сторінка авторизації та реєстрації.
- Сторінка менеджменту спринтів.
- Сторінка менеджменту проєктів.
- Сторінка менеджменту користувачів системи.

Був детально описаний сценарій створення нового проєкту, налаштування його критеріїв та відповідних навичок користувачів, додання задач в спринт та процес автоматичного підбору виконавців для вибраної задачі.

Детально описаний зміст кожного з креслень екранних форм програмного забезпечення із поясненням місцезнаходження користувача в програмі у поточний момент і можливі дії, які користувач може виконати в даний момент.

ВИСНОВКИ

У ході виконання дипломного проєкту було проведено аналіз функціонування систем управління розробкою проєктів. Зафіксовано всю наявну інформацію, яка має стосунок до об’єкта автоматизації. Були виділені основні ключові етапи, притаманні процесу, та взаємозв’язки між ними. Визначено призначення системи та цілі створення програмного забезпечення.

Був проведений ґрунтовний аналіз процесу розробки програмних продуктів. Ретельно описані моделі розробки програмного забезпечення та методологія гнучкої розробки Agile. Був проведений огляд існуючих систем для управління процесом розробки програмного забезпечення та проведений їх порівняльний аналіз. На основі даних, отриманих в процесі аналізу, була сформульована відповідна задача. Автоматизувати процес підбору виконавців для задач проєкту в процесі його розробки при використанні гнучких методологій. Це дозволить зменшити навантаження на проєктного менеджера та підвищити продуктивність розробки програмного продукту.

Для розробки програмного забезпечення була використана мова Javascript з використанням технологій для створення Single Page Application продуктів.

Розроблена модель бази даних, яка дає змогу ефективно та надійно здійснювати доступ до даних, що надходять та використовуються в процесі функціонування системи управління проєктами. Для управління базою даних обрана СУБД Posgresql.

Наведена детальна інструкція користувача по експлуатації програмного забезпечення. Проведено тестування системи на відповідність вимог заданим у технічному завданні. В результаті було встановлено, що система успішно пройшла всі тести і відповідає заявленим відносно неї вимогам, тому може бути введена в експлуатацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React js – створення реактивних веб сторрінок [Електронний ресурс] // Режим доступу: https://reactjs.org/.

2. Сем Нюмен создание микросервисов. К: Издательский дом Питер, 2018. - 304 с.

3. Крис Ричардсон Microservices Patterns: With Examples. К: Издательский дом Питер, 2018. - 554 с.

4. Express js [Електронний ресурс] // Режим доступу: https://expressjs.com/ru/

5. Серверна розробка з Node js [Електронний ресурс] // Режим доступу: https://nodejs.org/uk/.

10. Вибір технології для розробки серверної сторони веб сайту [Електронний ресурс] // Режим доступу: https://habr.com/ru/company.

11. Sass - Syntactically Awesome Style Sheets [Електронний ресурс] // Режим доступу: https://sass-lang.com/.

12. Система контролю версій - Github [Електронний ресурс] // Режим доступу: https://github.com.

13. Етан Браун. Изучаем JavaScript: руководство по созданию современных вебсайтов. К: Издательский дом O'Reilly, 2018. – 365 с.

14. Створення баз даних. Етапи проєктування [Електронний ресурс] // Языки программирования Life-prog.ru.– Режим доступу: http://life-prog.ru/ukr/1_181_stvorennya-baz-danih-etapi-proektuvannya.html.

15. Бази даних. Структура БД. Основні операції з базами даних. [Електронний ресурс] // Режим доступу: https://wiki.fizmat.tnpu.edu.ua/index.php.

16. Publishing your website [Електронний ресурс] // – Режим доступу: https://developer.mozilla.org/en/docs/Learn.

18. Методи збору вимог [Електронний ресурс] // - Режим доступу: https://habr.com/ru/company/simbirsoft/blog/307844/.