**ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № з/п | Формат | Позначення | Найменування | Кількість листів | Примітка |
| 1 | А4 | КПІ.ІП-5203. ХХХХХХ.02.81 | WEB-застосування проведення тематичних дискусій та on-line опитувань. Технічне завдання |  |  |
| 2 | А4 | КПІ.ІП-5203. ХХХХХХ.02.81 | WEB-застосування проведення тематичних дискусій та on-line опитувань.Пояснювальна записка |  |  |
| 3 | А4 | КПІ.ІП-5203. ХХХХХХ.03.81 | Програма та методика тестування. |  |  |
| 4 | А4 | КПІ.ІП-5203. ХХХХХХ.04.51 | Керівництво користувача. |  |  |
| 5 | А4 | КПІ.ІП-5203. ХХХХХХ.04.51 | Керівництво адміністратора. |  |  |
| 6 | А3 | КПІ.ІП-5203. ХХХХХХ.04.51 | Діаграма сценаріїв використання. |  |  |
| 7 | А3 | КПІ.ІП-5203. ХХХХХХ.04.51 | Діаграма структури бази даних |  |  |
| 8 | А3 | КПІ.ІП-5203. ХХХХХХ.04.51 | Діаграма бізнес-процесу керування даними…. |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з трьох розділів, містить 3 таблиці, Х додатків та Х джерел – загалом ХХХ сторінок.

**Об`єкт дослідження:** веб системи моделі клієнт-сервер з реляційними базами даних, що використовуються для роботи сайту для комунікації користувачів.

**Мета дипломного проекту:**  створити веб сервіс, що може бути використаний для проведення дискусійних обговорень та опитувань онлайн.

У першому розділі було проаналізовано предметну модель та розроблено архітектуру для застосування. Побудовано структурну схему класів та діаграму послідовності.

У другому розділі проведено тестування веб застосунку за розробленим планом тестування. Описано процес тестування.

У третьому розділі описано розгортання та впровадження веб застосунку, а також наведено схему структурну розгортання.

У додатках наведено: опис програми, схема структурна класів програмного забезпечення, схема структурна послідовності виконання.

КЛЮЧОВІ СЛОВА: ОНЛАЙН ДИСКУСІЇ, ОНЛАЙН ОПИТУВАННЯ, КОМУНІКАЦІЙНИЙ СЕРВІС

ABSTRACT

Explanatory note of the diploma project consists of 3 sections, X annexes, 3 tables and 5 sources – total XX pages…..

**The object of study**: server application systems, which can be used for continuous integration and delivery with Github and Kubernetes integration.

**The aim of the diploma project**: create easy scalable web application, capable of fast, rapid execution of integration and delivery process to Kubernetes cluster.

In the first section, the architecture of microservice web applications was described and developed. A structural diagram of classes and a sequence diagram are constructed.

In the second secion, resulting web application was tested according to the developed test plan. The process of testing is described.

The third section describes the deployment and implementation of microservice web applications. The diagram of structural deployment is provided.

Annexes contain the description of the program, the diagram of the structural classes of the software, the scheme of the structural sequence of execution.

KEYWORDS: CONTINUOUS INTEGRATION, CONTINUOUS DELIVERY, MICROSERVICES

ВМЕСТО эТОГО ЛИСТА ВСТАВИТЬ ЛИСТ ТИТУЛА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

ЗМІСТ

[Перелік умовних позначень, символів, одиниць, скорочень і термінів 11](#_Toc9372808)

[Вступ 12](#_Toc9372809)

[1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 14](#_Toc9372810)

[1.1 Загальні положення 14](#_Toc9372811)

[1.2 Змістовний опис і аналіз предметної області 14](#_Toc9372812)

[1.3 Аналіз успішних IT-проектів 15](#_Toc9372813)

[1.4 Висновки по розділу 16](#_Toc9372814)

[2 Моделювання та конструювання програмного забезпечення 17](#_Toc9372815)

[2.1 Моделювання та аналіз програмного забезпечення 17](#_Toc9372816)

[2.2 Аналіз безпеки даних 44](#_Toc9372817)

[2.3 Висновки по розділу 45](#_Toc9372818)

[3 АНАЛІЗ ЯКОСТІ ТА ТеСТУВАННЯ програмного забезпечення 46](#_Toc9372819)

[3.1 Аналіз якості ПЗ 46](#_Toc9372820)

[3.2 Підходи до тестування 48](#_Toc9372821)

[3.2.1 Компонентне тестування 48](#_Toc9372822)

[3.2.2 Інтеграційне тестування 49](#_Toc9372823)

[3.2.3 Тестування продуктивності 49](#_Toc9372824)

[3.3 Критерії проходження тестування 49](#_Toc9372825)

[3.3.1 Компонентне тестування 49](#_Toc9372826)

[3.3.2 Інтеграційне тестування 49](#_Toc9372827)

[3.3.3 Тестування швидкодії 50](#_Toc9372828)

[3.4 Процес тестування 50](#_Toc9372829)

[3.4.1 Дані до тестів 50](#_Toc9372830)

[3.4.2 Задачі тесту 51](#_Toc9372831)

[3.4.3 План виконання 51](#_Toc9372832)

[3.5 Вимоги до середовища 51](#_Toc9372833)

[3.5.1 Апаратна частина 51](#_Toc9372834)

[3.5.2 Програмна частина 51](#_Toc9372835)

[3.5.3 Вимоги до безпеки 51](#_Toc9372836)

[3.5.4 Інструменти 51](#_Toc9372837)

[3.6 Опис контрольного прикладу 52](#_Toc9372838)

[4 впровадження та супровід програмного забезпечення 54](#_Toc9372839)

[4.1 Розгортання програмного забезпечення 54](#_Toc9372840)

[4.1.1 Створення даних авторизації для кластеру 54](#_Toc9372841)

[4.1.2 Створення даних авторизації для реестру Google Container Registry 54](#_Toc9372842)

[4.1.3 Встановлення мікросервісу неперервної доставки 56](#_Toc9372843)

[4.1.4 Встановлення мікросервісу неперерної інтеграції 56](#_Toc9372844)

[4.1.5 Встановлення мікросервісу API 56](#_Toc9372845)

[4.2 Інструкція користувача 57](#_Toc9372846)

[4.2.1 Регістрація 57](#_Toc9372847)

[4.2.2 Логін 58](#_Toc9372848)

[4.2.3 Додання токену доступа до GitHub 58](#_Toc9372849)

[4.2.4 Підключення репозиторію 59](#_Toc9372850)

[4.3 Інструкція адміністратора 62](#_Toc9372851)

[4.3.1 Регістрація 62](#_Toc9372852)

[4.3.2 Логін 62](#_Toc9372853)

[4.3.3 Додання нових адміністраторів 62](#_Toc9372854)

[4.3.4 Створення деплою 63](#_Toc9372855)

[4.3.5 Налаштування та деталі деплою 63](#_Toc9372856)

[Висновки 65](#_Toc9372857)

[ПЕРЕЛІК ПОСИЛАНЬ 66](#_Toc9372858)

[Додаток А Технічне завдання 67](#_Toc9372859)

[Додаток б Опис програми 68](#_Toc9372860)

[додаток В ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ 69](#_Toc9372861)

[Додаток г керівництво системного програміста Додаток д керівництво програміста Додаток є керівництво користувача 70](#_Toc9372862)

[Додаток е Графічний матеріал 73](#_Toc9372863)

[Лист 1. Схема структурна варіантів використань 74](#_Toc9372864)

[Лист 2. Схема структурна станів системи 75](#_Toc9372865)

[Лист 3. Схема бази даних 76](#_Toc9372866)

[Лист 4. Схема структурна класів програмного забезпечення 77](#_Toc9372867)

[Лист 5. Креслення вигляду екранних форм 78](#_Toc9372868)

Перелік умовних позначень, символів, одиниць, скорочень і термінів

Дискусія –

Пост –

Фреймворк -

Драйвер –

СУБД-

Роути -

….

Вступ

В всесвітній мережі існує безліч сайтів та сервісів, що мають різні функції та цілі, однією з таких цілей є комунікація між людьми. Для того, щоб кожен міг в зручному форматі обмінятись думками з іншими створена велика кількість форумів, чатів, систем ведення блогу. Кожна з цих систем має свою модель яка зручна для якогось із аспектів комунікації. Також сайти можуть мати свою тематику, та додаткові можливості. Були створені різні моделі обміну інформацією, деякі сервіси комбінують їх, це, наприклад, соціальні мережі, інші залишаються в рамках своєї.

…………..

Завданням роботи є створення ВЕБ-застосунку для дискусії між користувачами, а також проведенням опитувань.

Мета розробки – систематизування теоретичних знань отриманих впродовж учбового процесу для створення сервісу з можливістю комунікації on-line. Призначення розробки - використання проекту для надання можливості проведення тематичних обговорень on-line в текстовому вигляді та зручному відображенню повідомлень згідно їх рейтингу.

Задачі:

* авторизація користувачів в системі;
* створення поділу на спільноти, обмежених деякою темою обговорень або ідеологією;
* створення та перегляд постів;
* створення та перегляд опитувань;
* коментування постів та перегляд релевантних коментарів до них;

Цілі:

* надати користувачеві можливість здійснити авторизацію, ввести необхідні дані про себе;
* надати можливість приєднатись користувачеві до цікавих йому спільнот;
* можливість користувача створювати пости до спільноти;
* зробити можливість коментувати та переглядати коментарі інших в зручному вигляді ;
* зробити можливість голосувати в опитуванні та переглядати голоси інших;

# АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Загальні положення

Програмний продукт, що задовольняє вимогам повинен бути написаний на мові та платформі, що дає можливості для створення ВЕБ-серверу, підключення до бази данних в якості мови серверу. На Javascript, або мові, що може транспілюватись в нього, наприклад Typescript, для створення браузерної частини застосунку, та реляційної бази даних з одним із діалектів SQL, до якої є драйвер від серверної платформи. Система повинна відповідати сучасним стандартам створення ВЕБ-застосунків, тому має бути використана актуальна платформа зі всіма необхідними властивостями, в ролі такої можна взяти ASP.NET Core, вона є гарним вибором також тому, що дає можливість запускати сервер як на ОС Windows, так і на Linux. Також програмне забезпечення повинне мати можливість зберігати дані про виконання, та проводити зневаження. Вищезазначені технології це дозволяють.

## Змістовний опис і аналіз предметної області

………

Комплекси для інтеграції та розгортання програмних продуктів мають підтримувати хоча б одну з популярних хмарних систем контролю версій як GitHub, BitBucket чи GitLab. Вони повинні мати можливість запускати процес одразу після оновлення кодової бази, з поверненням результату побудови у систему контролю версій. Такі комплекси мають також інтегруватися з хмарними рішеннями для зберігання образів контейнерів Docker з результатами побудов для подальшої доставки їх у деплої кластеру Kubernetes. Так як ці процеси можуть виконуватися дуже часто та одночасно, комплекс повинен бути оптимізований під паралельне виконання великої кількості задач та масштабованість.

## Аналіз успішних IT-проектів

Серед сервісів зі схожими властивостями можна виділити такі сервіси як Reddit, Habrahabr, Dou.ua – їх досить багато, оскільки такий формат є зручним для комунікації, коли необхідно дати можливість спілкуватись та мати корисний контент, що зручний читачам. Проте між ними є досить великі відмінності, вони можуть мати додаткові специфічні можливості, користі для загальної тематики сайту, різний користувацький інтерфейс.

Reddit – відомий сайт, що містить загальну тематику, в ньому є розділення на сабреддіти (тематики), в яких можливо створювати пости, коментувати їх, оцінювати як пости так і коментарі. Відображує найкращі відповіді в пріоритеті. Має досить старий традиційний дизайн. В порівнянні з даним проектом Reddit має старий дизайн, не має функціонал швидкого опитування.

Habrahabr – сайт, що має загальну професійну тематику – IT. Містить багато цікавих та професійних статей, головною особливістю є те, що має строгу систему, завдяки якій на ньому коментує та створює пости відносно невелика кількість користувачів, але при цьому дані що публікуються загалом мають більшу інформаційну цінність. Має також рейтингову систему коментарів та постів, хаби (тематики), блоги компаній. На відміну від даного проекту Habrahabr має обмежену тематику та рейтинг, що розповсюджується на весь сайт, замість одної спільноти.

Dou.ua – сайт-форум для розробників з України, має менш строгу систему, серед тематики як ІТ-технології, так і працевлаштування в ІТ, має додаткові розділи пов’язані з цим а також містить рейтинг компаній, виділяє найбільш оцінені коментарі. В порівнянні з Dou.ua, даний проект не має спеціалізації на темі та не має специфічного функціоналу – весь контент створюється в рамках моделі спільнот.

## Висновки по розділу

……

У цьому розділі було описано та проаналізовано предметну область розробки. Було виділено успішні IT- проекти у даній області та виконано порівняння даного комплексу задач с готовими продуктами.

# Моделювання та конструювання програмного забезпечення

## Моделювання та аналіз програмного забезпечення

Розроблена система виконується під операційними системами на базі ядра Linux. Використання під операційними системами Windows та macOS можливе, але не було протестовано.

Використовується мова програмування F# на платформі .NET Core та фреймворку Giraffe в якості серверної частини. Typescript разом з бібліотекою React для браузерної частини. PostgreSQL в якості технології реляційної БД. ВЕБ-сервер працює за принципами SPA: сервер отримує надходження запиту, якщо це запит на сторінку повертає компільований Frontend-додаток, якщо запит на API – виконує необхідну обробку даних та повертає результат у вигляді JSON. Серверна частина складається з моделі, модулю роботи з базою даних, модулем логіки за стосунку та модулем будування pipeline-ів обробки запиту, засоби програмування на F# дає можливість лаконічно робити Dependency injection, побудову rout-ів, робити обробку даних.

Клієнтська частина складається з точки входу – яка в залежності від шляху запиту відображає потрібний компонент, компонентів - основних інтерфейсних блоків програми та міні-компонентів, що дозволяють повторне використання та використовуються в різних блоках, вони з себе представляють окремі елементи інтерфейсу.

Таблиця 2.1 – Опис інтерфейсів системи

|  |  |
| --- | --- |
| **Інтерфейс** | **Опис** |
| DatabaseClient | Інтерфейс, який використовується для роботи з базою даних |
| Queue | Інтерфейс, який використовується для роботи з чергою повідомлень. |
| Logger | Інтерфейс, що використовується для логів. |
| DistLock | Інтерфейс, що використовується для роботи з розподіленими блокуваннями |

Таблиця 2.2 – Опис класів (структур) системи

|  |  |
| --- | --- |
| **Клас (структура)** | **Опис** |
| AdminDashComponent | Клас, який містить бізнес-логіку головної сторінки адміністратора. |
| UsersListComponent | Клас, який містить бізнес-логіку списку користувачів. |
| BranchConfigsListComponent | Клас, який містить бізнес-логіку компоненту списку налаштувань гілок. |
| BuildComponent | Клас, який містить бізнес-логіку компоненту сторінки керування побудовами. |
| BuildsListComponent | Клас, який містить бізнес-логіку компоненту списку побудов. |
| CreateBranchComponent | Клас, який містить бізнес-логіку компоненту для створення нової конфігурації для гілки. |

Продовження таблиці 2.2

|  |  |
| --- | --- |
| **Клас (структура)** | **Опис** |
| CreateDeploymentComponent | Клас, який містить бізнес-логіку компоненту для створення нового деплою. |
| CreateRepoComponent | Клас, який містить бізнес-логіку компоненту для додання нового репозиторію. |
| DeploymentComponent | Клас, який містить бізнес-логіку компоненту сторінки деплою. |
| DeploymentsListComponent | Клас, який містить бізнес-логіку компоненту списку деплоїв. |
| LoginComponent | Клас, який містить бізнес-логіку компоненту сторінки авторизації. |
| RepoComponent | Клас, який містить бізнес-логіку компоненту сторінки репозиторію. |
| ReposListComponent | Клас, який містить бізнес-логіку компоненту списку репозиторіїв. |
| RevisionComponent | Клас, який містить бізнес-логіку компоненту сторінки ревізії. |
| RevisionsListComponent | Клас, який містить бізнес-логіку компоненту списку ревізій. |
| ClipperService | Клас, який надає методи для взаємодії з API серверної частини. |
| StorageService | Клас, який надає методи для взаємодії з локальним сховищем браузеру. |
| SettingsComponent | Клас, який містить бізнес-логіку компоненту сторінки налаштувань. |

Продовження таблиці 2.2

|  |  |
| --- | --- |
| **Клас (структура)** | **Опис** |
| UserDashComponent | Клас, який містить бізнес-логіку компоненту головної сторінки користувача. |
| CDClient | Структура, яка надає методи для взаємодії з мікросервісом неперервної доставки. |
| CIClient | Структура, яка надає методи для взаємодії з мікросервісом неперервної інтеграції. |
| PostgresClient | Структура, яка реалізує інтерфейс DatabaseClient, надаючи методи для взаємодії з базою даних PostgreSQL. |
| StdoutLogger | Структура, яка реалізує інтерфейс Logger, надаючи методи для виводу логів у стандартний потік виводу. |
| RMQQueue | Структура, яка реалізує інтерфейс Queue, надаючи методи для взаємодії з чергою повідомлень RabbitMQ. |
| Server | Структура, що містить бізнес-логіку API мікросервісу серверної частини. |
| RedisLock | Структура, яка реалізує інтерфейс DistLock, надаючи методи для роботи з розподіленими блокуваннями у сховищі Redis. |

Продовження таблиці 2.2

|  |  |
| --- | --- |
| **Клас (структура)** | **Опис** |
| Kubectl | Структура, що надає методи для роботи з утилітою командного рядка kubectl. |
| Worker | Структура, що містить бізнес-логіку мікросервісу неперервної інтеграції. Мікросервіс неперервної доставки містить аналогічну структуру. |

Таблиця 2.3 – Опис методів класів та інтерфейсів системи

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient | Close() | Закриває підключення до бази даних. |
| DatabaseClient | CreateSchema() | Створює таблиці у базі даних. |
| DatabaseClient (API мікросервіс) | CreateUser(login, pass string, isAdmin bool) | Створює нового користувача у базі даних. |
| DatabaseClient (API мікросервіс) | SaveUser(user \*types.User) | Зберігає користувача у базі даних. |
| DatabaseClient (API мікросервіс) | FindUser(login string) | Знаходить користувача у базі даних по логіну. |
| DatabaseClient (API мікросервіс) | FindUserByID(userID int64) | Знаходить користувача у базі даних по ідентифікатору. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (API мікросервіс) | FindAllUsers(q url.Values) | Знаходіть усіх користувачів у базі даних. |
| DatabaseClient (API мікросервіс) | FindAllUsersCount() | Знаходіть кількість усіх користувачів у базі даних. |
| DatabaseClient (API мікросервіс) | ChangeUserAdminStatus(userID int64, isAdmin bool) | Змінює статус користувача на розробника або адміністратора. |
| DatabaseClient (API мікросервіс) | CreateRepo(repo \* type.GithubRepo) | Створює новий репозиторій у базі даних. |
| DatabaseClient (API мікросервіс) | SaveRepo(repo \*types.GithubRepo) | Зберігає репозиторій у базі даних. |
| DatabaseClient (API мікросервіс) | FindRepoByName(fullName string) | Знаходить репозиторій у базі даних по назві. |
| DatabaseClient (API мікросервіс) | FindRepoByID(repoID int64) | Знаходить репозиторій у базі даних по ідентифікатору. |
| DatabaseClient (API мікросервіс) | DeleteRepoByID(repoID int64) | Видяляє репозиторій з бази даних. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (API мікросервіс) | FindAllUserRepos(userID int64, q url.Values) | Знаходить усі репозиторії що належать користувачеві. |
| DatabaseClient (API мікросервіс) | FindAllUserReposCount(userID int64) | Знаходить кількість репозиторіїв що належать користувачеві. |
| DatabaseClient (API мікросервіс) | FindAllRepos(q url.Values) | Знаходить усі репозиторії у базі даних. |
| DatabaseClient (API мікросервіс) | CreateBranchConfig(c \*types.BranchConfig) | Створює нове налаштування гілки репозиторію у базі даних |
| DatabaseClient (API мікросервіс) | FindBranchConfig(repoID int64, branch string) | Знаходить налаштування гілки репозиторію у базі даних |
| DatabaseClient (API мікросервіс) | DeleteBranchConfig(repoID int64, branch string) | Видяляє налаштування гілки репозиторію з бази даних |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (API мікросервіс) | DeleteBranchConfigByID(configID int64) | Видяляє налаштування гілки репозиторію з бази даних за ідентифікатором |
| DatabaseClient (API мікросервіс) | FindAllBranchConfigs(repoID int64, q url.Values) | Знаходить налаштування гілок для вказаного репозиторію у базі даних |
| DatabaseClient (API мікросервіс) | FindAllBranchConfigsCount(repoID int64) | Знаходить кількість налаштувань гілок для вказаного репозиторію у базі даних |
| DatabaseClient (мікросервіс неперервної доставки) | CreateDeployment(kd \*types.Deployment) | Створює новий деплой у базі даних. |
| DatabaseClient (мікросервіс неперервної доставки) | DeleteDeployment(kd \*types.Deployment) | Видяляє деплой з бази даних. |
| DatabaseClient (мікросервіс неперервної доставки) | FindAllDeployments(page, limit int64) | Знаходить усі деплої у базі даних. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (мікросервіс неперервної доставки) | FindDeployment(deploymentID int64) | Знаходить деплой у базі даних по ідентифікатору. |
| DatabaseClient (мікросервіс неперервної доставки) | FindDeploymentCount() | Знаходить кількість усіх деплоїв у базі даних. |
| DatabaseClient (мікросервіс неперервної доставки) | FindRevision(revisionID int64) | Знаходить ревізію у базі даних за ідентифікатором. |
| DatabaseClient (мікросервіс неперервної доставки) | FindRevisions(depoymentID, page, limit int64) | Знаходить усі ревізії деплою у базі даних. |
| DatabaseClient (мікросервіс неперервної доставки) | FindRevisionsCount(deploymentID int64) | Знаходить кількість усіх ревізій деплою у базі даних. |
| DatabaseClient (мікросервіс неперервної доставки) | SaveDeployment(kd \*types.Deployment) | Зберігає деплой у базі даних. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (мікросервіс неперервної інтеграції) | CreateBuild(b \*types.Build) | Створює побудову у базв даних. |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindAllBuilds(repoID int64, branch string, page, limit int64) | Знаходить усі побудови репозиторію у базі даних. |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindBuildsCount(repoID int64, branch string) | Знаходить кількість усіх побудов репозиторію у базі даних. |
| DatabaseClient (мікросервіс неперервної інтеграції) | CreateBuildArtifact(b \*types.BuildArtifact) | Створює артефакт побудови у базі даних. |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindBuildArtifact(buildID int64) | Знаходить артефакт побудови у базі даних. |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindBuildArtifactByID(ID int64) | Знаходить артефакт побудови у базі даних за ідентифікатором. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindBuildByID(buildID int64) | Знаходить побудову у базі даних за ідентифікатором. |
| DatabaseClient (мікросервіс неперервної інтеграції) | FindBuildArtifactsCount(repoID int64, branch string) | Знаходить кількість усіх побудов у базі даних. |
| Queue | Close() | Закрити з’єднання з чергою повідомлень. |
| Queue (мікросервіс API) | PublishCIJob(jobMsg \*commonTypes.CIJob) | Надіслати повідомлення для запуску процесу інтеграції. |
| Queue (мікросервіс неперервної доставки) | MakeCDMsgChan() | Створити канал для отримання повідомлень для запуску процесу доставки. |
| Queue (мікросервіс неперервної інтеграції) | PublishCDJob(jobMsg \*commonTypes.CDJob) | Надіслати повідомлення для запуску процесу доставки. |
| Queue (мікросервіс неперервної інтеграції) | MakeCIMsgChan() | Створити канал для отримання повідомлень для запуску процесу інтеграції. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Logger | Info(I string) | Записати інформаційне повідомлення у лог. |
| Logger | Error(msg string, err error) | Записати повідомлення про помилку у лог разом з її деталями. |
| Logger | Fatal(msg string, err error) | Записати повідомлення про помилку у лог разом з її деталями, потім завершити виконання програми. |
| DistLock | Lock(resName string) | Отримати блокування ресурсу. |
| DistLock | Unlock(resName string) | Звільнити блокування ресурсу. |
| BranchConfigListComponent | getPage(page: number) | Завантажити сторінку списку налаштувань гілок. |
| BranchConfigListComponent | deleteConfig(branch: string) | Видалити налаштування. |
| BuildComponent | loadBuild() | Завантажити побудову. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| UsersListComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| UsersListComponent | getPage(page: number) | Завантажити сторінку списку користувачів. |
| UsersListComponent | changeAdminStatus(userID: number, admin: boolean) | Змінити статус користувача на адмінстратора чи розробника. |
| BuildsListComponent | getPage(page: number) | Завантажити сторінку списку побудов. |
| CreateBranchComponent | onBranchAdd() | Створити налаштування гілки репозиторію. |
| CreateDeploymentComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| CreateDeploymentComponent | loadRepos() | Завантажити репозиторії. |
| CreateDeploymentComponent | loadRepoArtifacts(i: string) | Завантажити артефакти побудов репозиторію. |
| CreateDeploymentComponent | selectArtifact(i: string) | Обрати артефакт побудови. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| CreateDeploymentComponent | onDeploymentCreate() | Створити новий деплой. |
| CreateRepoComponent | onRepoAdd() | Додати новий репозиторій. |
| DeploymentComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| DeploymentComponent | loadDeployment() | Завантажити деплой. |
| DeploymentComponent | loadRepo() | Завантажити репозиторій. |
| DeploymentComponent | selectArtifact(artifactID: number) | Обрати артефакт побудови. |
| DeploymentComponent | loadRepoArtifacts() | Завантажити артефакти побудови. |
| DeploymentComponent | onScale() | Масштабувати деплой. |
| DeploymentComponent | onChangeImage() | Змінити образ контейнеру деплою. |
| DeploymentComponent | onChangeManifest() | Змінити маніфест деплою. |
| DeploymentComponent | onDeploymentDelete() | Видалити деплой. |
| DeploymentsListComponent | ngOnInit() | Викликається при завантаженні компоненту. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| DeploymentsListComponent | getPage(page: number) | Завантажити сторінку списку деплоїв. |
| LoginComponent | onRegister() | Регістрація нового користувача. |
| LoginComponent | onLogin() | Авторизація користувача. |
| RepoComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| RepoComponent | loadRepo() | Завантажити інформацію про репозиторій. |
| RepoComponent | onRepoDelete() | Видалити репозиторій. |
| ReposListComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| ReposListComponent | getPage(page: number) | Завантажити сторінку списку репозиторіїв. |
| RevisionComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| RevisionComponent | loadRevision() | Завантажити інформацію про ревізію. |

Продовження таблиці 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** | |
| RevisionsListComponent | ngOnInit() | Викликається при завантаженні компоненту. | |
| RevisionsListComponent | getPage(page: number) | Завантажити сторінку списку ревізій. | |
| ClipperService | login(username: string, password: string) | Запит до API на авторизацію. | |
| ClipperService | register(username: string, password: string) | Запит до API на регістрацію користувача. | |
| ClipperService | addRepo(fullName: string) | | Запит до API на додання репозиторію. |
| ClipperService | getUsers(page:number, limit: number) | | Запит до API на отримання інформації про користувачів. |
| ClipperService | changeUserAdminStatus(userID: number, admin: boolean) | | Запит до API на зміну ролі користувача. |
| ClipperService | getRepos(page: number, limit: number) | | Запит до API на отримання інформації про репозиторії. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| ClipperService | getRepo(repoID: number) | Запит до API на отримання інформації про репозиторій. |
| ClipperService | deleteRepo(repoID: number) | Запит до API на видалення репозиторію. |
| ClipperService | getBuild(buildID: number) | Запит до API на отримання інформації про побудову. |
| ClipperService | getBuilds(repoID: number, branch: string, page: number, limit: number) | Запит до API на отримання інформації про побудови. |
| ClipperService | addBranchConfig(branch: string, repoID: number) | Запит до API на додання налаштування гілки. |
| ClipperService | deleteBranchConfig(repoID: number, branch: string) | Запит до API на видалення налаштування гілки. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| ClipperService | getBranchConfigs(repoID: number, page: number, limit: number) | Запит до API на отримання інформації про налаштування гілок репозиторію. |
| ClipperService | getDeployment(depID: number) | Запит до API на отримання інформації про деплой. |
| ClipperService | getArtifacts(repoID: number, branch: string, page: number, limit: number) | Запит до API на отримання інформації про артефакти побудов. |
| ClipperService | addDeployment(dep: Clipper.PostDeploymentRequest) | Запит до API на створення деплою. |
| ClipperService | deleteDeployment(depID: number) | Запит до API на видалення деплою. |
| ClipperService | getRevisions(depID: number, page: number, limit: number) | Запит до API на отримання інформації про ревізії деплоїв. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| ClipperService | scaleDeployment(depID: number, replicas: number) | Запит до API на масштабування деплою. |
| ClipperService | changeDeploymentImage(depID: number, imageID: number) | Запит до API на зміну образу контейнеру деплою. |
| ClipperService | changeDeploymentManifest(depID: number, manifest: string) | Запит до API на зміну маніфесту деплою. |
| ClipperService | getRevision(revID: number) | Запит до API на отримання інформації про ревізію деплою. |
| StorageService | setToken(token: string) | Запам`ятати токен JWT. |
| StorageService | parseToken(token: string) | Десеріалізувати дані з токену JWT. |
| StorageService | isTokenExpired() | Перевірити чи актуальний токен. |
| StorageService | isUserAdmin() | Перевірити чи є поточний користувач адміністратором. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| StorageService | getUserName() | Отримати ім’я поточного користувача. |
| SettingsComponent | ngOnInit() | Викликається при завантаженні компоненту. |
| SettingsComponent | onWebhookSave() | Зберегти секрет для вебхуку. |
| SettingsComponent | onAccessTokenSave() | Зберегти токен доступу до Github status API. |
| CDClient | CreateDeployment(d \*types.DeploymentMessage) | RPC виклик для створення деплою. |
| CDClient | GetDeployment(deploymentID int64) | RPC виклик для отримання інформації про деплой. |
| CDClient | GetAllDeployments(params types.PaginationQueryParams) | RPC виклик для отримання інформації про всі деплої. |
| CDClient | DeleteDeployment(deploymentID int64) | RPC виклик для видалення деплою. |
| CDClient | UpdateImage(d \*types.DeploymentMessage) | RPC виклик для оновлення образу контейнера деплою. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| CDClient | ScaleDeployment(d \*types.DeploymentMessage) | RPC виклик для масштабування деплою. |
| CDClient | UpdateManifest(d \*types.DeploymentMessage) | RPC виклик для оновлення маніфесту деплою. |
| CDClient | GetRevision(revisionID int64) | RPC виклик для отримання інформації про ревізію деплою. |
| CDClient | GetRevisions(deploymentID int64, params types.PaginationQueryParams) | RPC виклик для отримання інформації про всі ревізії деплою. |
| CIClient | GetBuild(buildID int64) | RPC виклик для отримання інформації про побудову репозиторію. |
| CIClient | GetBuildArtifact(buildID int64) | RPC виклик для отримання інформації про артефакт побудови. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| CIClient | GetAllBuilds(repoID int64, params types.BuildsQueryParams) | RPC виклик для отримання інформації про всі побудови репозиторію. |
| CIClient | GetAllArtifacts(repoID int64, params types.BuildsQueryParams) | RPC виклик для отримання інформації про всі артефакти побудов репозиторію. |
| Server | Run() | Запуск API серверу. |
| Server | Routes() | Налаштування обробників шляхів API. |
| Server | getBuildArtifactHandler(c \*gin.Context) | Обробник запитів на отримання артефакту побудов. |
| Server | getAllArtifactsHandler(c \*gin.Context) | Обробник запитів на отримання усіх артефактів побудов. |
| Server | postBranchConfigHandler(c \*gin.Context) | Обробник запитів на створення налаштування гілки репозиторію. |
| Server | getAllBranchConfigsHandler(c \*gin.Context) | Обробник запитів на отримання усіх налаштування гілки репозиторію. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Server | deleteBranchConfigHandler(c \*gin.Context) | Обробник запитів на видалення налаштування гілки репозиторію. |
| Server | getBuildHandler(c \*gin.Context) | Обробник запитів на отримання побудови репозиторію. |
| Server | postDeploymentHandler(c \*gin.Context) | Обробник запитів на створення деплою. |
| Server | getDeploymentHandler(c \*gin.Context) | Обробник запитів на отримання деплою. |
| Server | getAllDeploymentsHandler(c \*gin.Context) | Обробник запитів на отримання усіх деплоїв. |
| Server | deleteDeploymentHandler(c \*gin.Context) | Обробник запитів на видалення деплою. |
| Server | changeDeploymentImageHandler(c \*gin.Context) | Обробник запитів на зміну образу контейнера деплою. |
| Server | scaleDeploymentHandler(c \*gin.Context) | Обробник запитів на масштабування деплою. |
| Server | updateManifestHandler(c \*gin.Context) | Обробник запитів на зміну маніфесту деплою. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Server | postRepoHandler(c \*gin.Context) | Обробник запитів на додання репозиторію. |
| Server | getRepoHandler(c \*gin.Context) | Обробник запитів на отримання інформації про репозиторій. |
| Server | getAllReposHandler(c \*gin.Context) | Обробник запитів на отримання інформації про всі репозиторії. |
| Server | getAllUserRepos(userID int64, c \*gin.Context) | Обробник запитів на отримання усіх репозиторіїв користувача. |
| Server | deleteRepoHandler(c \*gin.Context) | Обробник запитів на видалення репозиторію. |
| Server | getRevisionsHandler(c \*gin.Context) | Обробник запитів на отримання усіх ревізій деплою. |
| Server | getRevisionHandler(c \*gin.Context) | Обробник запитів на отримання ревізії деплою. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Server | loginHandler(c \*gin.Context) | Обробник запитів на авторизацію користувача. |
| Server | registerHandler(c \*gin.Context) | Обробник запитів на регістрацію користувача. |
| Server | setSecretHandler(c \*gin.Context) | Обробник запитів на встановлення секрету вебхуку користувача. |
| Server | setAccessTokenHandler(c \*gin.Context) | Обробник запитів на встановлення токену доступу до Github status API користувача. |
| Server | webhookHandler(c \*gin.Context) | Обробник вебхуків від сервісу Github. |
| Server | jwtMiddleware(secret []byte) | Проміжний обробник, що не дозволяє виконати обробку запиту без авторизації. |
| Server | userIsAdminMiddleware(c \*gin.Context) | Проміжний обробник, що не дозволяє виконати обробку запиту якщо користувач не є адміністратором. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Server | getRevisionsHandler(c \*gin.Context) | Обробник запитів на отримання усіх ревізій деплою. |
| Server | getRevisionHandler(c \*gin.Context) | Обробник запитів на отримання ревізії деплою. |
| Kubectl | ScaleDeployment(deployment string, replicas int64) | Масштабувати деплой. |
| Worker | Run() | Запустити мікросервіс. |
| Worker (мікросервіс неперервної доставки) | updateDeploymentImage(dep types.Deployment, artifactID int64) | Змінити образ контейнеру деплою. |
| Worker (мікросервіс неперервної доставки) | scaleDeployment(dep types.Deployment, replicas int64) | Масштабувати деплой. |
| Worker (мікросервіс неперервної доставки) | executeCDJob(CDJob commonTypes.CDJob) | Виконати процес доставки. |
| Worker (мікросервіс неперервної доставки) | updateImageFromProto(d types.Deployment) | Обробник RPC виклику зміни образу контейнеру деплою. |
| Worker (мікросервіс неперервної доставки) | scaleFromProto(d types.Deployment) | Обробник RPC виклику масштабування деплою. |
| Worker (мікросервіс неперервної доставки) | deleteDeployment(d types.Deployment) | Видалення деплою. |

Продовження таблиці 2.3

|  |  |  |
| --- | --- | --- |
| **Клас/Інтерфейс** | **Метод** | **Опис** |
| Worker (мікросервіс неперервної доставки) | initDeployment(d types.Deployment) | Ініціалізація деплою. |
| Worker (мікросервіс неперервної доставки) | reInitDeployment(d types.Deployment, manifest string) | Повторна ініціалізація (сброс) деплою. |
| Worker (мікросервіс неперервної доставки) | updateManifestFromProto(d types.Deployment) | Обробник RPC виклику зміни маніфесту деплою. |
| Worker (мікросервіс неперервної доставки) | deleteFromProto(d types.Deployment) | Обробник RPC виклику видалення деплою. |
| Worker | startConsuming() | Запуск головного циклу обробки мікросервісу. |
| Worker (мікросервіс неперервної інтеграції) | executeBuilder(payload types.BuilderPayload) | Запуск побудови образу контейнера. |

## Аналіз безпеки даних

…

Паролі користувачів у базі зберігаються у вигляді результату обробки функцією bcrypt, тому праобраз паролю майже неможливо знайти без повного перебору, який буде сповільнено через особливість роботи bcrypt що змушує витрачати багато часу на визначення хешу паролю.

Інформація що поступає у вебхуці перевіряється за допомогою секретного ключа, що використовується для HMAC, завдяки чому перевіряється що запит був саме від GitHub.

Взаємодія з Google Container Registry та kubernetes відбувається за допомогою сервісних акаунтів яким можна обмежувати права та які самі по собі не дають повного доступу що не можна відкликати.

## Висновки по розділу

У даному розділі було створено архітектуру веб сервісу, обрано технології його розробки, зокрема: серверну мову програмування, що є зручною для такого класу задач, клієнтську мову програмування та фреймворк для створення інтерфейсу, також обрано реляційну СУБД, що є зручною, та має драйвер в серверній мові. Також було представлено структуру класів та функцій.

# АНАЛІЗ ЯКОСТІ ТА ТеСТУВАННЯ програмного забезпечення

## Аналіз якості ПЗ

Якість програмного забезпечення – важлива характеристика продукту, оскільки вона є складовою актуальності для кінцевого користувача, зокрема є важливою для клієнт-серверних багатокористувацьких сервісів.

Для забезпечення якості одним із важливих процесів в розробці програмного забезпечення є його тестування. Тому воно є одним із головних етапів в при створенні програмних продуктів. Тестування дозволяє не допустити неякісний продукт до наступних етапів розгортання програмного продукту, завчасно знайти помилки в життєвому циклі розробки.

Для якісного тестування слід дотримуватись правил та стандартів, що були складені на основі досвіду в великій кількості програмних продуктів. Тому їх дотримання є необхідною умовою в процесах розробки. Зокрема існує такий стандарт тестування як IEEE 829—1998 Standard for Software Test Documentation.

До плану тестування ввійде обсяг, підхід, ресурси та план усіх методів тестування. План описує програмні об’єкти що будуть протестовані, тип тестів, ресурси та план необхідний для виконання тестування.

У рамках цього плану буде виконано тестування частини продукту, що відповідає за ….

У даному плані будуть протестовані наступні функції:

* реєстрація користувачів;
* авторизація користувачів;
* оновлення даних користувачів;
* створення спільнот;
* оновлення даних спільнот;
* вхід до спільнот;
* створення посту;
* коментування посту
* оцінка посту та коментарю
* створення опитування
* голосування та перегляд результатів опитування.

Налаштовані наступні тестові модулі:

…

* реєстрація користувачів;
* авторизація користувачів;
* оновлення даних користувачів;
* створення спільнот;
* оновлення даних спільнот;
* вхід до спільнот;
* створення посту;
* коментування посту
* оцінка посту та коментарю
* створення опитування
* голосування та перегляд результатів опитування.

## Підходи до тестування

В рамках даного плану будуть використані наступні методи тестування:

* компонентне;
* інтеграційне;
* продуктивності.

### Компонентне тестування

Методом компонентного тестування будуть перевірені логічно окремі частини веб сервісу, такі як:

* роути сервісу;
* методи доступу до бази даних;
* окремі допоміжні функції;
* методи авторизації

### Інтеграційне тестування

Методом інтеграційного тестування будуть перевірені взаємодії між модулями системи, такі як:

* взаємодія веб сервера з базою даних;
* взаємодія веб сервера з клієнтом;

### Тестування продуктивності

Методом тестування продуктивності буде перевірена швидкодія наступних елементів системи:

* запити до бази даних;
* обробка сервером запитів без бази даних;
* обробка сервером запитів з базою даних.

## Критерії проходження тестування

### Компонентне тестування

Для компонентного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один пункт не був успішно виконаний – тестування вважається не пройденим.

### Інтеграційне тестування

Для інтеграційного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один пункт не був успішно виконаний – тестування вважається не пройденим.

### Тестування швидкодії

Для тестування швидкодії критерієм проходження є успішне виконання тесту з кожним доступним набором параметрів (кількість даних у одному запиті, кількість запитів, конкурентність) не довше ніж максимально допустимий час. У разі якщо хоча б один варіант тесту не був успішним або виконувався довше максимально допустимого часу – тестування вважається не пройденим.

## Процес тестування

### Дані до тестів

Вхідними даними для компонентного тестування є набори параметрів на яких очікується певний результат, що є вихідними даними даного тесту.

Вхідними даними для інтеграційного тестування є набори повідомлень що будуть передані від одного компоненту системи до іншого відповідно до конкретного тесту. Вихідними даними для даного виду тестування є результат роботи останнього компоненту у ланцюзі (наприклад, запис у базі даних у випадку тестування взаємодії API серверу та бази даних).

Вхідними даними до тестування швидкодії є набори даних, що покривають усі варіанти роботи системи у конкретному випадку. Вихідними даними є швидкість обробки запитів, кількість оброблених запитів, кількість неправильних реакцій на набір даних, дані по навантаженню на апаратну платформу (завантаженість процесору, вільна оперативна пам’ять, завантаженість мережі тощо).

### Задачі тесту

Кожен тест повинен перевірити як правильність програми у відповідності до умов виконання тесту (test-driven development), так і виявити можливі помилки у роботі.

### План виконання

Компонентне тестування повинне виконуватися до інтеграційного, яке, у свою чергу, виконується до тестування швидкодії.

## Вимоги до середовища

### Апаратна частина

Вимоги до апаратної частини співпадають з вимогами з технічного завдання.

### Програмна частина

Для виконання тестування платформа повинна мати операційну систему на базі Linux.

### Вимоги до безпеки

Для тестування бажано створити відповідний тестовий акаунт.

### Інструменти

Для виконання тестування використовувати наступні програмні інструменти:

* Insomnia;
* Vegeta (https://github.com/tsenart/vegeta) – для тестування швидкодії.

## Опис контрольного прикладу

У якості прикладу можна навести тестування логіну користувача за допомогою утиліти Insomnia.

Вхідні дані:

* пара логін та пароль *notexistant notexistant* (не наявні у комплексі);
* пара логін та пароль *admin wrongpassword* (неправильний пароль, існуючий користувач);
* пара логін та пароль *admin admin* (існуючий користувач).

Вихідні дані:

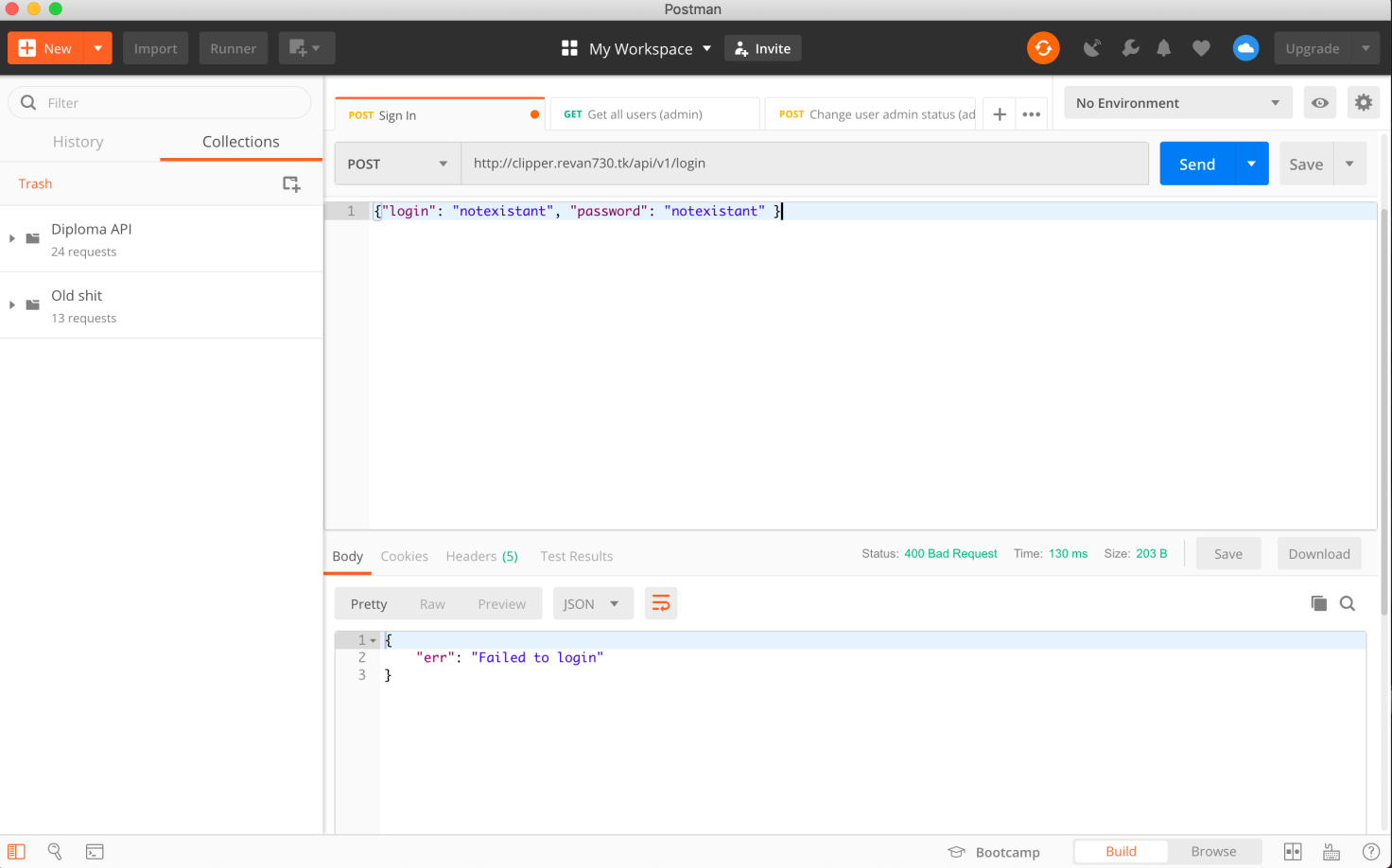
* повідомлення про помилку у вигляді JSON стрічки  
  *{“err”: “Failed to login”}* ;
* повідомлення з токеном доступа при успішному логіні у вигляді JSON стрічки *{ “token”: “TOKEN”}* де TOKEN – токен доступу до API.

У даному тестовому прикладі усього 3 ситуації, перші 2 відповідають першим двом парам вхідних даних та першому набору вихідних, відповідно 3 ситуація – останньому набору вхідних та вихідних даних.

Для виконання тесту потрібно в Postman створити POST запит за адресою <http://API_URL/api/v1/login> де API\_URL - адреса API мікросервісу.

У якості тіла запиту використати JSON об’єкт наступного формату:

*{“login”: “LOGIN”, “password”: “PASSWORD”}* де LOGIN та PASSWORD – логін та пароль з однієї з пар вхідних даних. Після введення даних натиснути кнопку “Send” та перевірити відповідність вихідних даних до наведених вище (Рисунок 3.1).

Рисунок 3.1 – Приклад виконання тесту

# впровадження та супровід програмного забезпечення

## Розгортання програмного забезпечення

Для повного розгортання даного комплексу задач потрібно виконати наступні етапи:

* створити дані авторізації для кластеру kubernetes;
* створити дані авторизації для реестру Google Container Registry;
* встановити мікросервіс неперервної доставки;
* встановити мікросервіс неперервної доставки;
* встановити мікросервіс API.

### Створення даних авторизації для кластеру

Використовуючи утиліту gcloud під’єднатися до кластеру [6], після чого виконати команду на отримання даних авторизації у кластері gcloud container clusters get-credentials CLUSTER\_NAME, де CLUSTER\_NAME - назва кластеру. Після цього необхідні для роботи з кластером дані будуть записані у файл kubeconfig.

### Створення даних авторизації для реестру Google Container Registry

Для створення даних авторизації потрібно перейти на головну сторінку керування Google Cloud Platform (Рисунок 4.1), після чого перейти на сторінку IAM – Сервісні аккаунти [7].

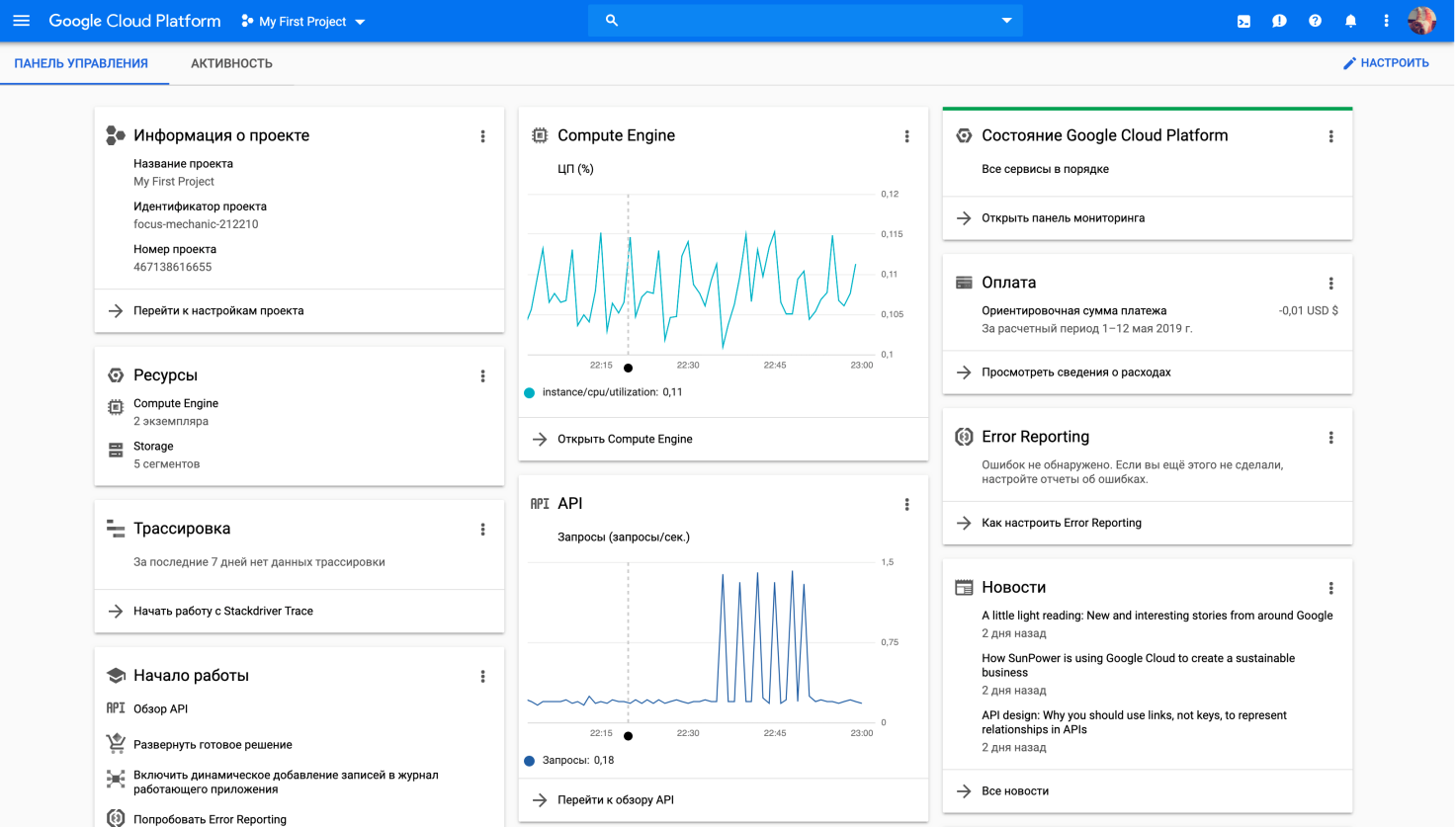


Рисунок 4.1 – Головна сторінка керування Google Cloud Platform

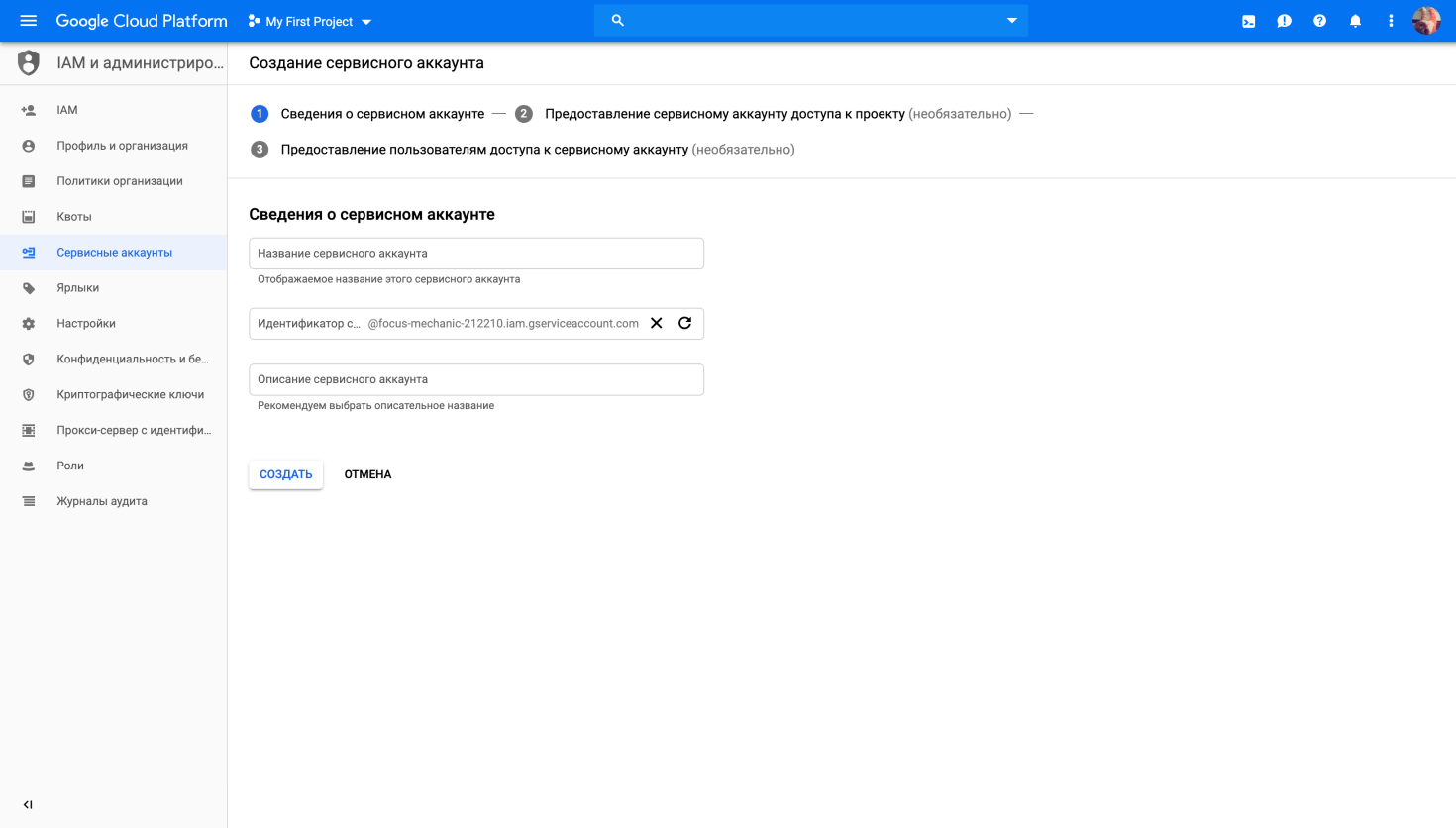
Далі потрібно натиснути кнопку “Створити сервісний аккаунт” та дотримуватися вказівок на екрані (Рисунок 4.2). Обрати тип файлу ключа для завантаження – JSON. Завантажити файл.

Рисунок 4.2 – Сторінка створення сервісного аккаунту для реестру

### Встановлення мікросервісу неперервної доставки

Для встановлення мікросервісу неперервної доставки бажано використати метод запуску за допомогою контейнеру Docker. Для цього потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-cd-worker* [8]. Після цього можна запустити мікросервіс виконавши команду *docker run clipper-cd-worker –v /PATH\_TO\_KUBECTL:/config/.kube --rabbitmq=RABBITMQ\_ADDR –redis=REDIS\_ADDR* де PATH\_TO\_KUBECTL – шлях до файлу авторизації у кластері отриманому у пункті 4.1.1, RABBITMQ\_ADDR – адреса з’єднання з чергою повідомлень rabbitmq, REDIS\_ADDR – адреса для з’єднання з redis.

### Встановлення мікросервісу неперерної інтеграції

Встановлення мікросервісу неперервної інтеграції аналогічне до попереднього пункту. Спочатку потрібно створити образ контейнеру зі скриптом-будівником, виконавши у директорії сирцевого коду під назвою ci-builder команду *docker build . –t clipper-ci-builder*. Після цього потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-ci-worker*. Після цього можна запустити мікросервіс виконавши команду *docker run clipper-ci-worker –v /PATH\_TO\_JSON:/opt/clipper-ci-secrets/ --rabbitmq=RABBITMQ\_ADDR –gcr=GCR\_URL –builder=BUILDER\_IMAGE –redis=REDIS\_ADDR* де PATH\_TO\_JSON – шлях до файлу авторизації у Google Container Registry отриманому у пункті 4.1.2, RABBITMQ\_ADDR – адреса з’єднання з чергою повідомлень rabbitmq, REDIS\_ADDR – адреса для з’єднання з redis, GCR\_URL – адреса регіону Google Container Registry для використання, BUILDER\_IMAGE – назва або повна адреса образу контейнеру з будівником.

### Встановлення мікросервісу API

Встановлення мікросервісу API виконується аналогічно. Потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-api*. Після цього можна запустити мікросервіс виконавши команду *docker run clipper-api --rabbitmq=RABBITMQ\_ADDR –ci=CI\_ADDR –cd=CD\_ADDR* де RABBITMQ\_ADDR – адреса з’єднання з чергою повідомлень rabbitmq, CI\_ADDR – адреса для з’єднання з мікросервісом неперервної інтеграції, CD\_ADDR – адреса для з’єднання з мікросервісом неперервної доставки.

## Інструкція користувача

### Регістрація

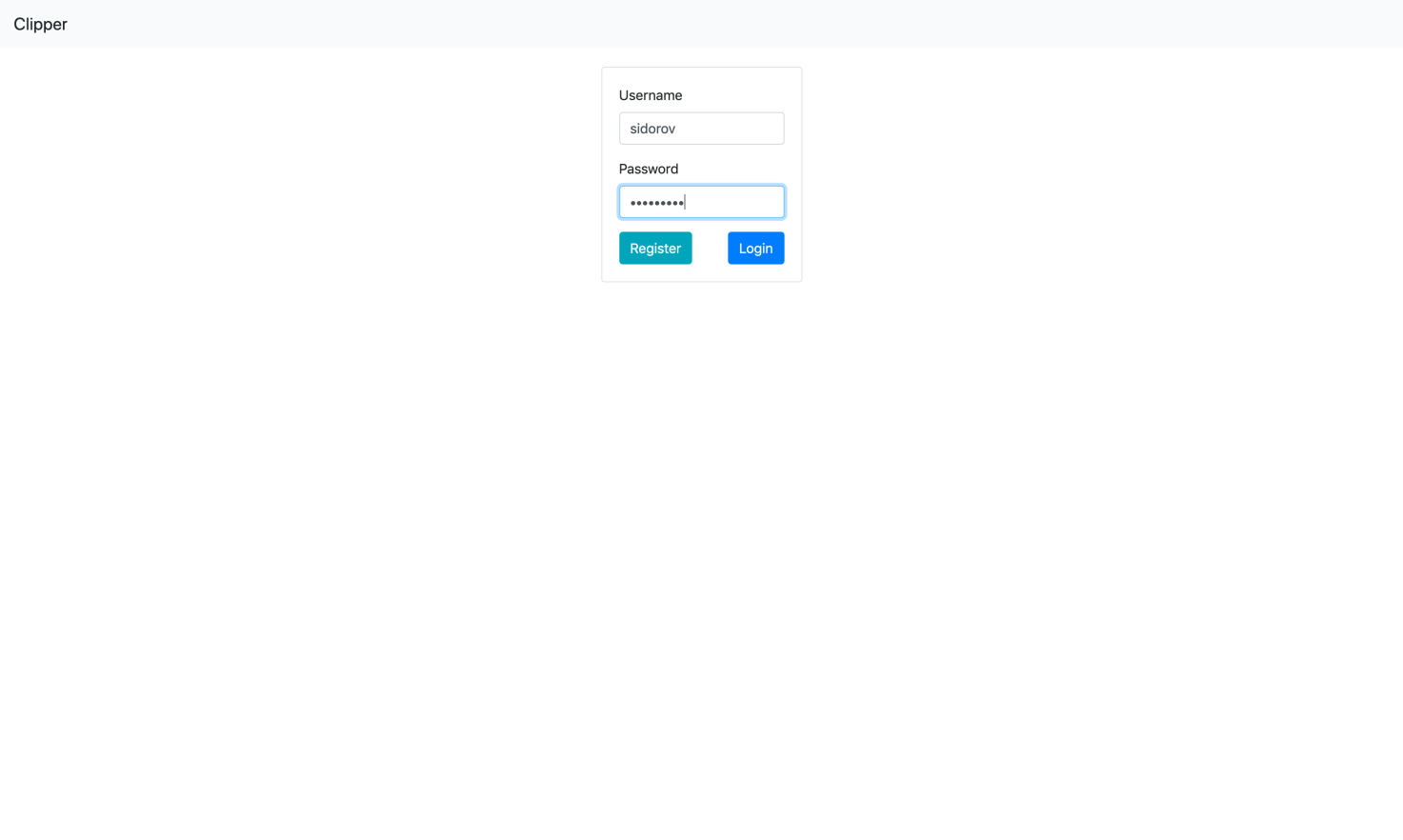
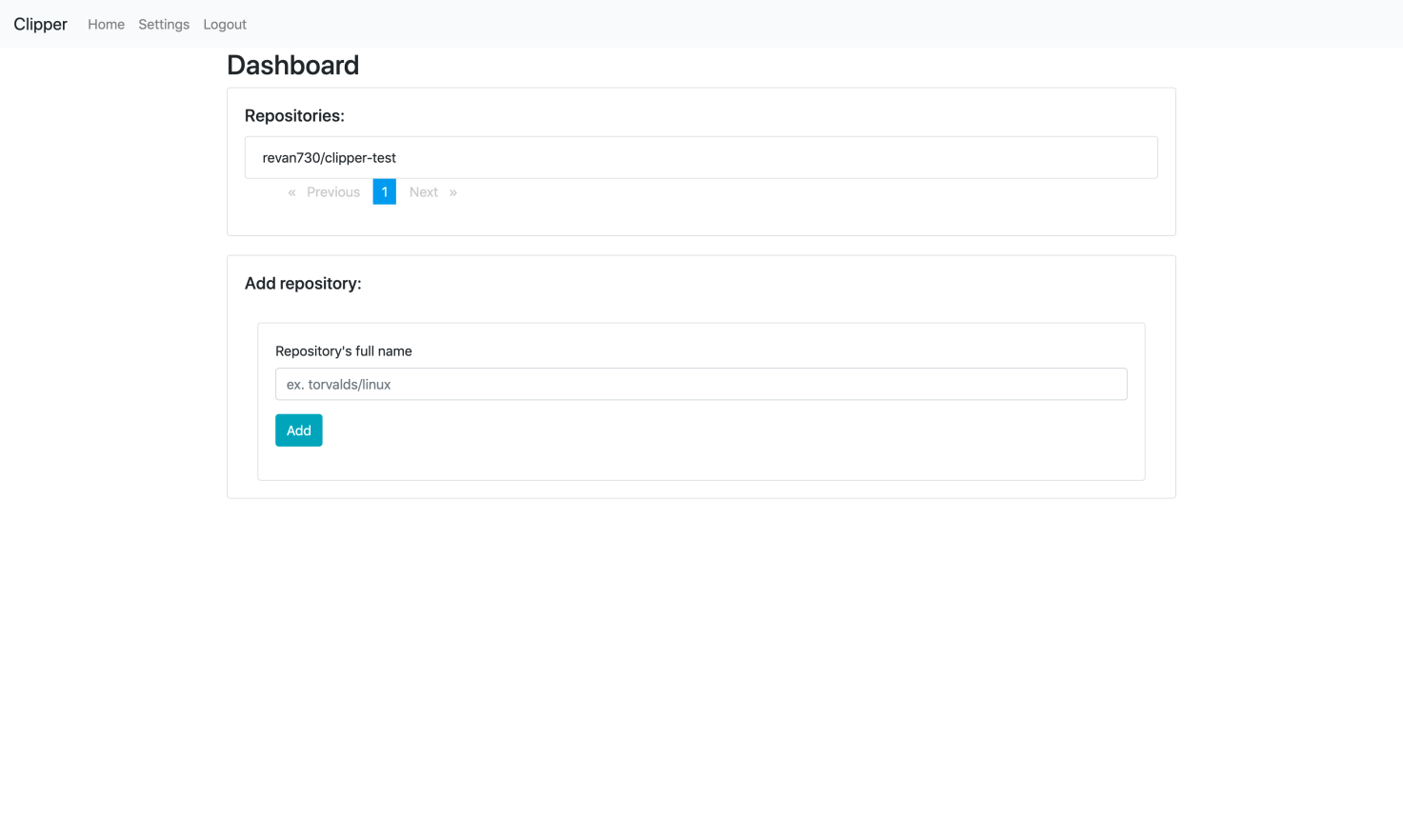
Перед початком роботи з комплексом задач користувач повинен пройти регістрацію, для чого потрібно ввести бажаний логін та пароль на сторінці авторизації та натиснути кнопку "Register" (Рисунок 4.3).

Рисунок 4.3 – Сторінка авторізації

### Логін

Після успішної регістрації користувач повинен авторизуватися у комплексі задач на тій самій сторінці, але на цей раз натиснувши кнопку "Login", після чого буде відображено головну сторінку користувача (Рисунок 4.4).

Рисунок 4.4 – Головна сторінка користувача

### Додання токену доступа до GitHub

Перед тим як додавати репозиторії системи GitHub потрібно створити та додати до комплексу токени доступу до Github та секретный ключ для підпису запитів вебхуків (не обов'язково).

Для цього потрібно перейти на сторінку налаштувань токенів доступу системи GitHub (Рисунок 4.5), та створити новий токен доступу. Після цього перейти на сторінку налаштувань користувача у комплексі, ввести отриманий токен доступу, ввести секретний ключ для підпису запитів вебхуків по бажанню та натиснути кнопку "Save" (Рисунок 4.6).

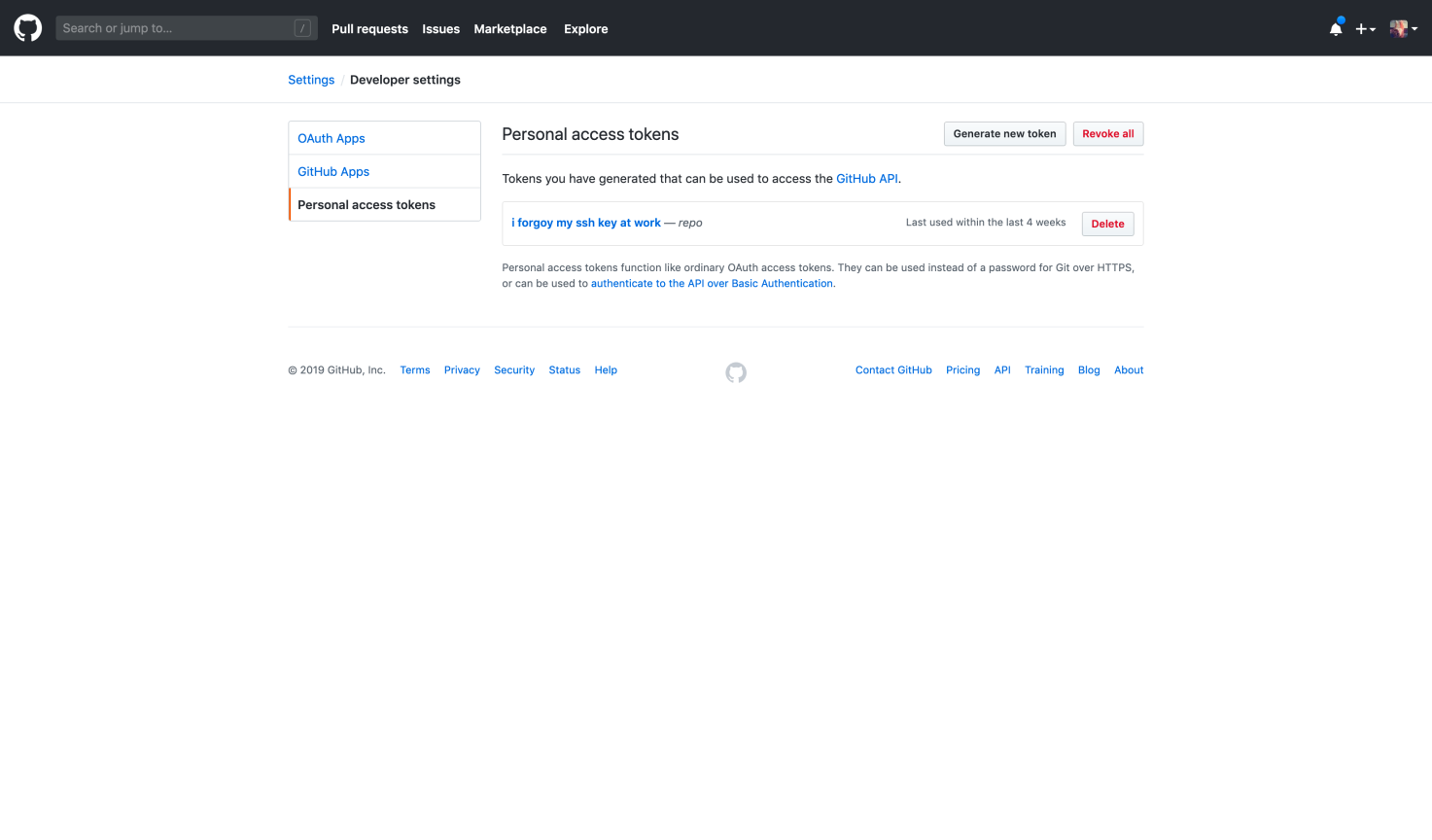
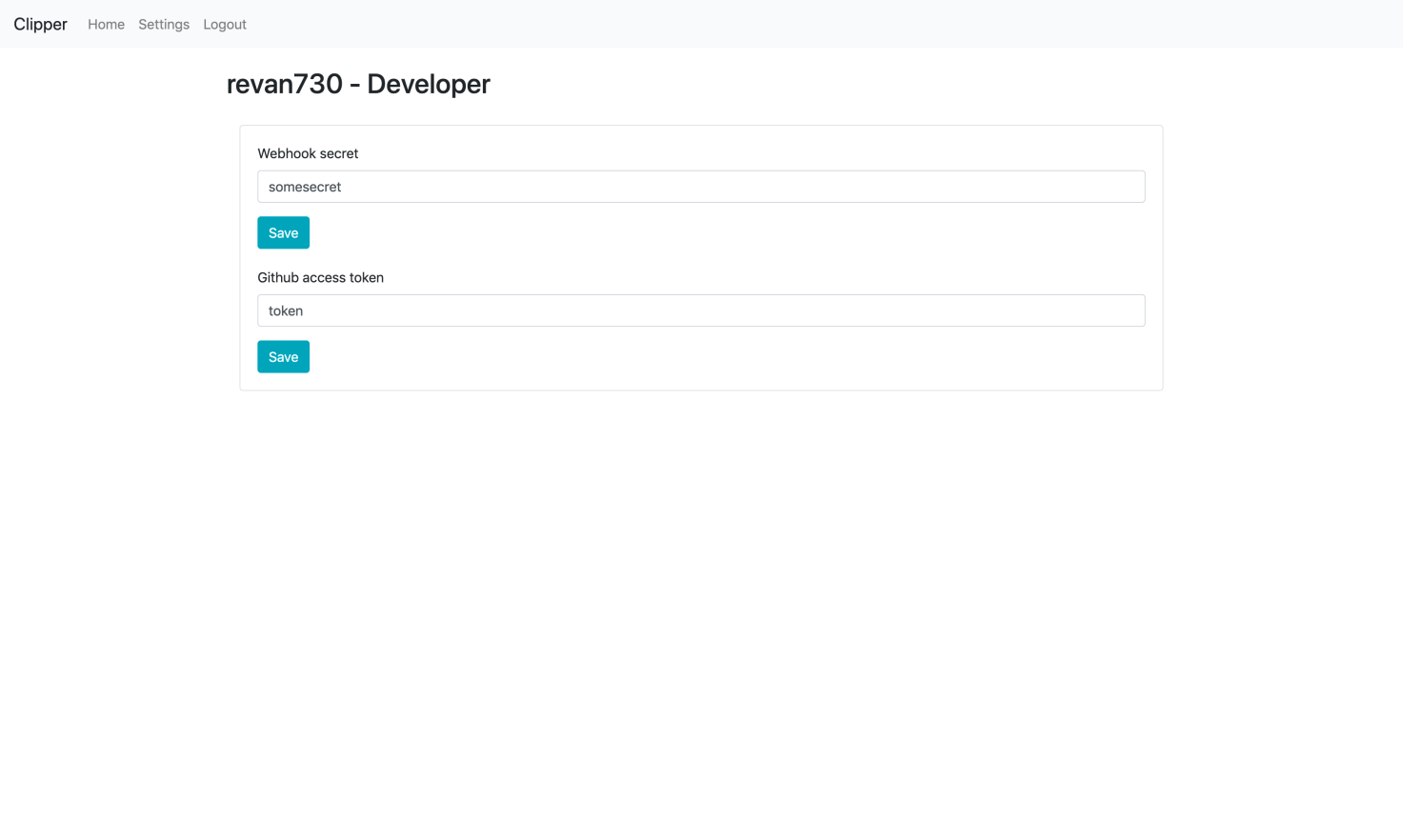
Рисунок 4.5 – Сторінка створення токенів доступу до GitHub

Рисунок 4.6 – Сторінка налаштувань токену доступу та секрету

### Підключення репозиторію

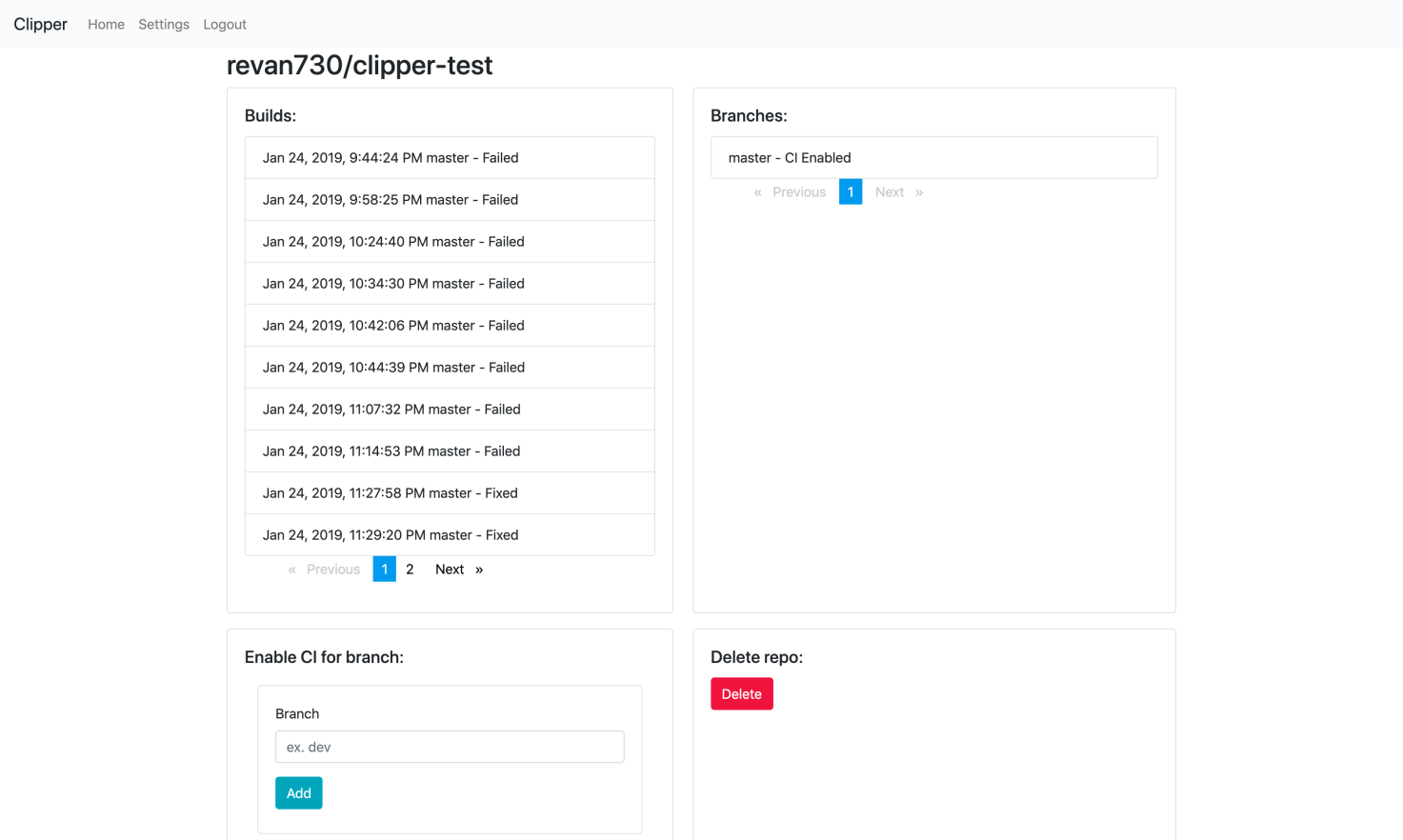
Тепер користувач має можливість підключати репозиторії до комплексу задач. Для цього на головній сторінці потрібно ввести назву репозиторію (таку саму як і у системі GitHub) та натиснути Save. Після цього (по бажанню чи необхідності) натиснувши на репозиторії зі списку можна перейти на сторінку з його детальним описом, де можна додати конфігурацію побудови гілки окрім master. (Рисунок 4.7)

Рисунок 4.7 – Сторінка репозиторію

Для завершення підключення репозиторію до комплексу, потрібно перейти на сторінку налаштувань вебхуків репозиторію у системі GitHub та натиснути кнопку "Add webhook" (Рисунок 4.8).

Ввести адресу вебхуку у вигляді http://API\_URL/api/v1/octohook/USER, де API\_URL - адреса API мікросервісу, USER - логін користувача у системі. Також ввести секретний ключ якщо користувач його обрав та встановив. Натиснути кнопку "Save". (Рисунок 4.9)

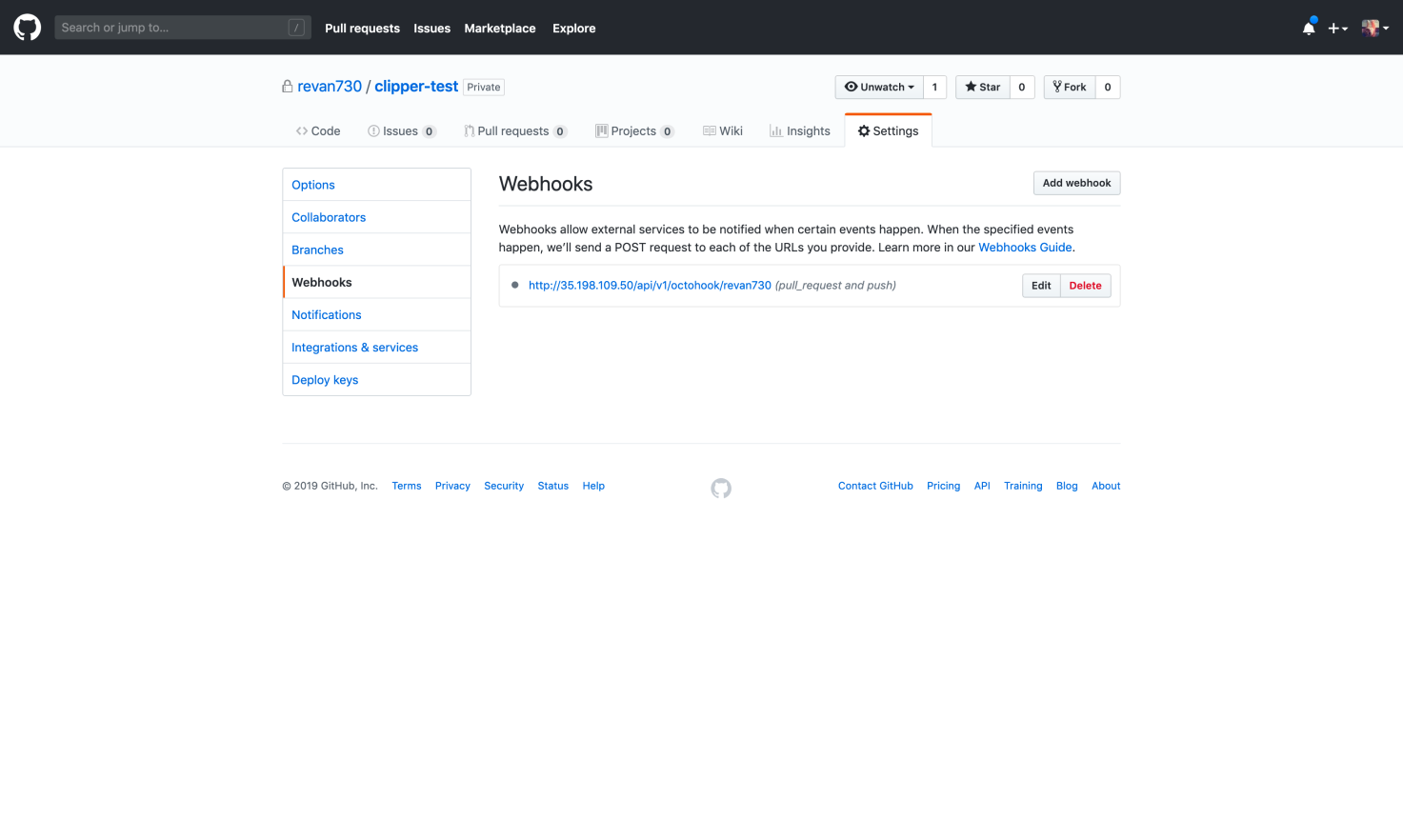
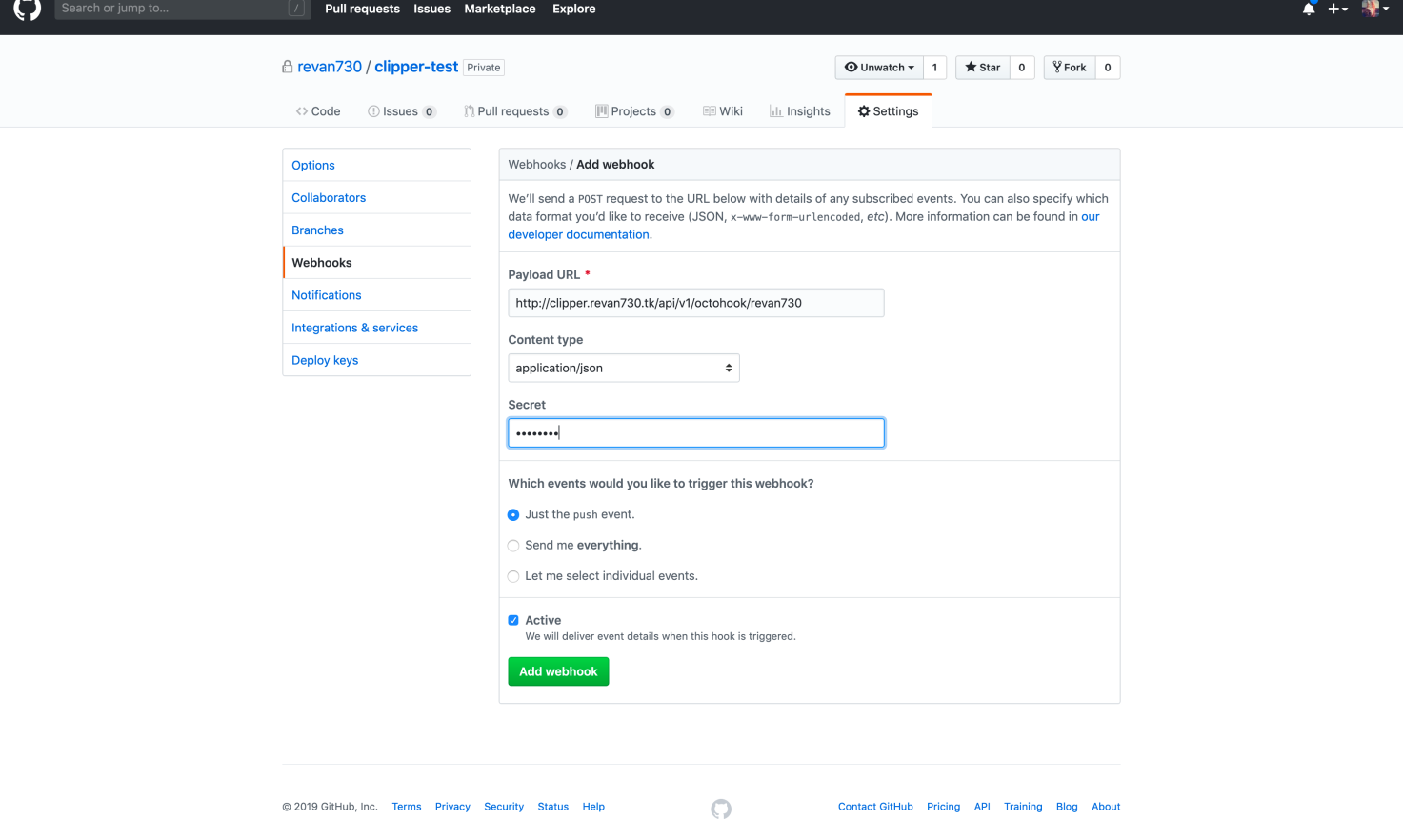
Рисунок 4.8 – Додання вебхуку

Рисунок 4.9 – Налаштування вебхуку

Підключення репозиторію до комплексу завершено. Після побудов їх історію та деталі можна отримати на сторінці репозиторію у системі.

## Інструкція адміністратора

### Регістрація

Регістрація адміністратора аналогічна до регістрації користувача. Перший адміністратор створюється у комплексі автоматично під час встановлення. Новими адміністраторами головний адміністратор може назначати користувачів та навпаки, робити адміністраторів користувачами.

### Логін

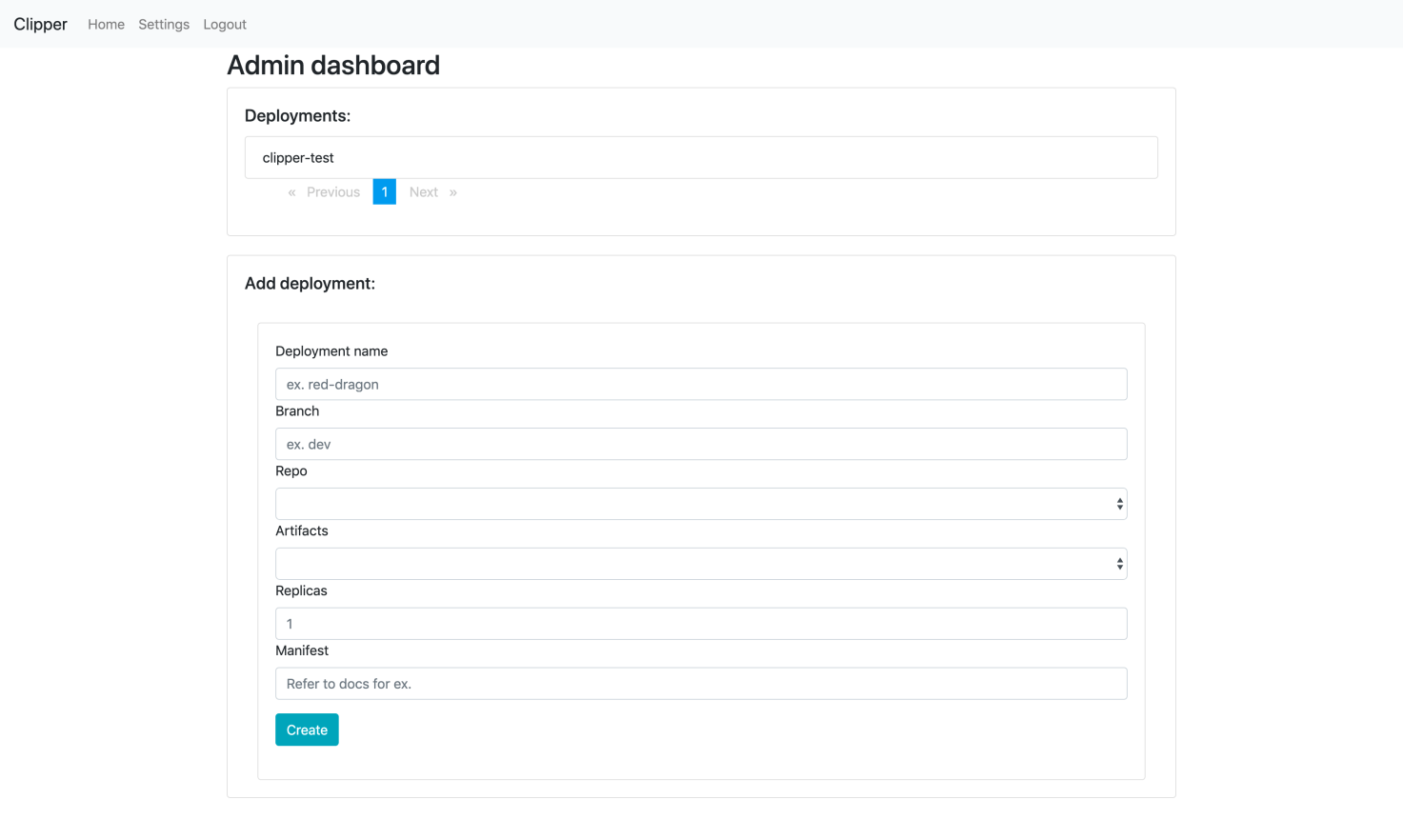
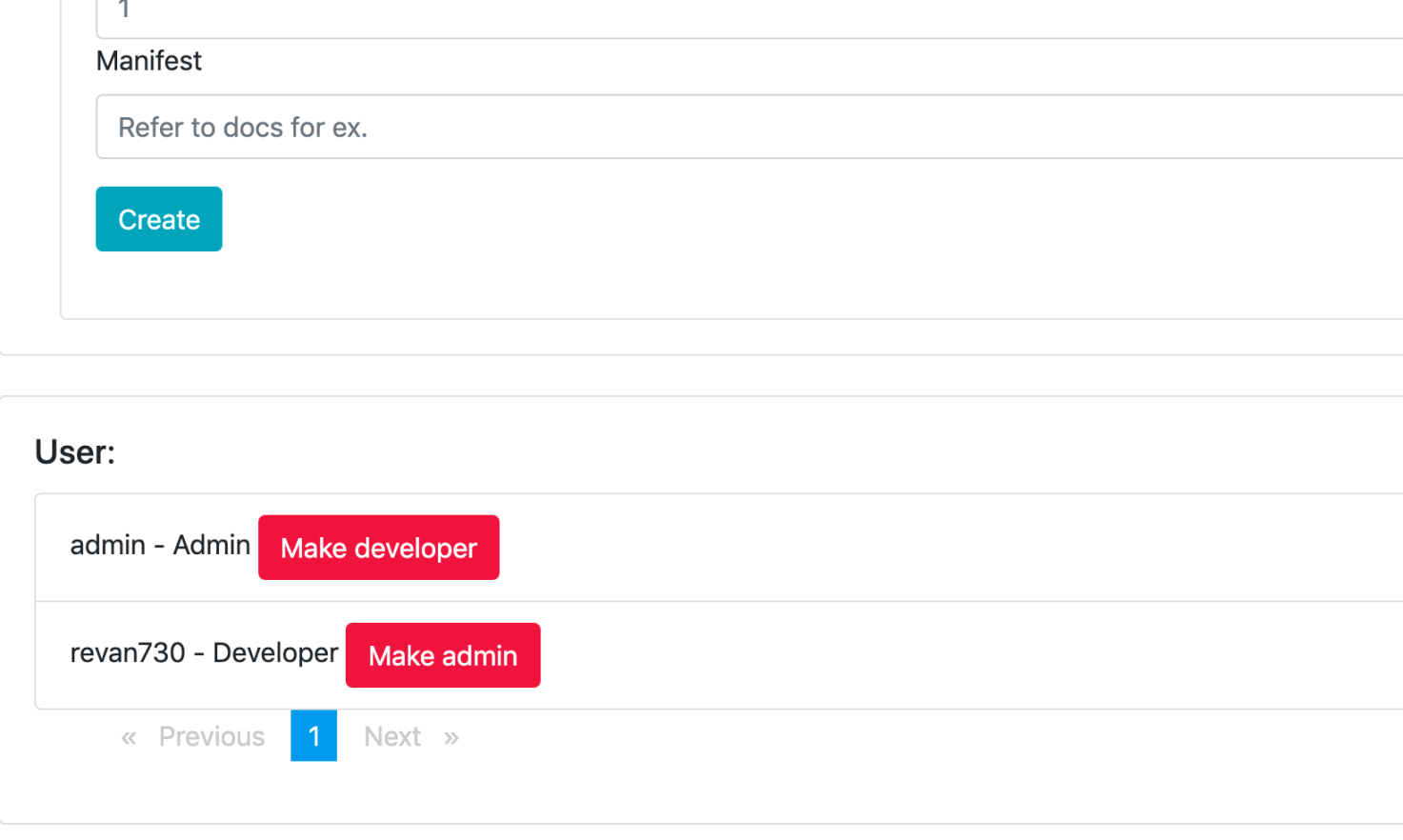
Логін адміністратора аналогічний до логіну користувача. Після логіну адміністратор потрапляє на головну сторінку (Рисунок 4.10).

Рисунок 4.10 – Головна сторінка адміністратора

### Додання нових адміністраторів

Для того щоб зробити користувача адміністратором потрібно на головній сторінці адміністратора знайти користувача у списку та натиснути кнопку “Make admin” (Рисунок 4.11).

Рисунок 4.11 – Зміна ролі користувача

### Створення деплою

Для створення нового деплою на головній сторінці адміністратор повинен у секції “Add deployment” ввести та обрати усі необхідні дані та натиснути “Create”. Після цього новий деплой буде відображено у списку.

### Налаштування та деталі деплою

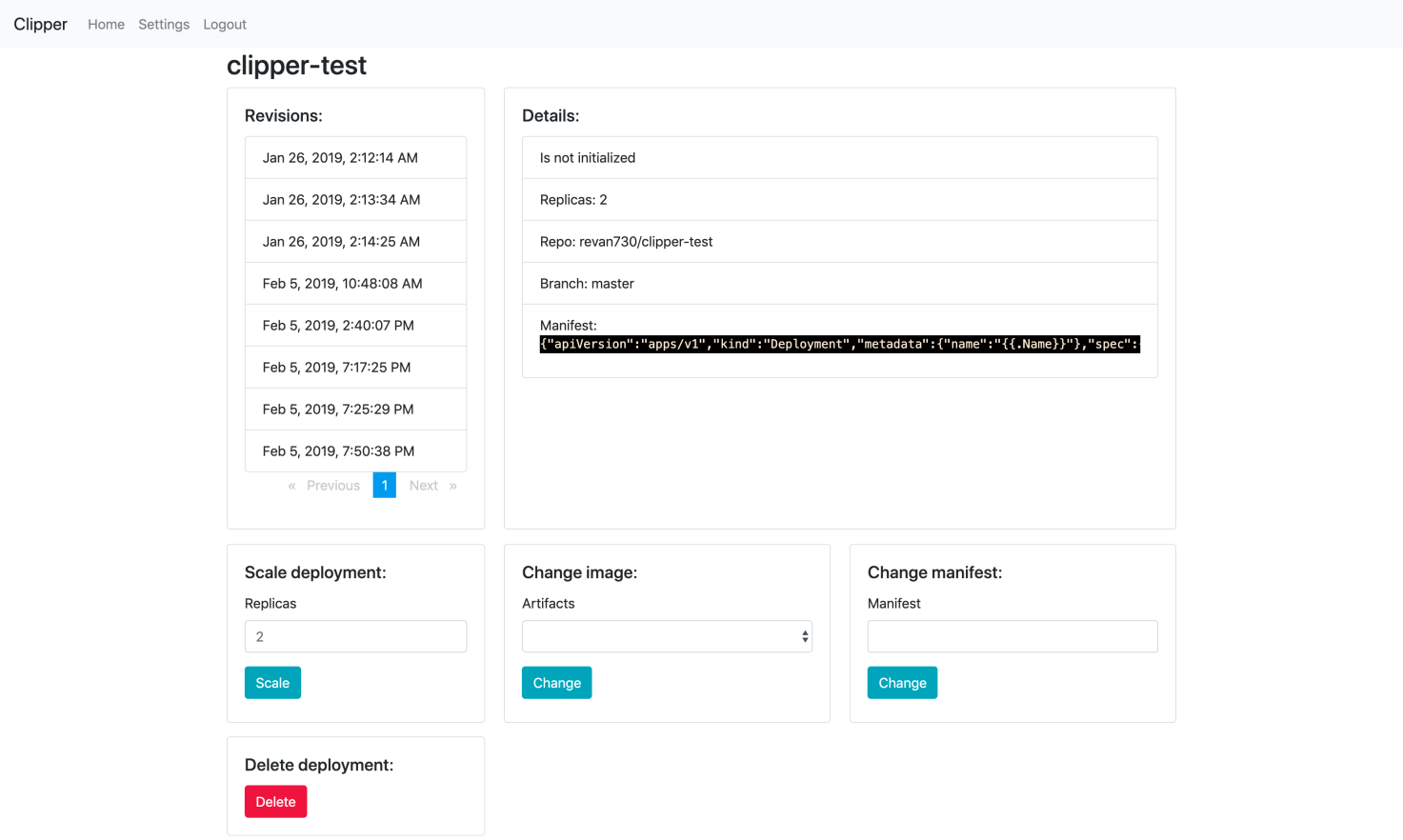
Натиснувши на деплой у списку адміністратор потрапляє на його сторінку (Рисунок 4.12). 

Рисунок 4.12 -  Сторінка керування деплою

На ній можна отримати інформацію про ревізії деплою, його розмір, репозиторій тощо. Також на цій сторінці можна масштабувати деплой, змінити маніфест або образ контейнеру.

Висновки

У розділі “Аналіз вимог до програмного забезпечення” було описано та проаналізовано предметну область розробки.

У розділі “Моделювання та конструювання програмного забезпечення” було розроблено архітектуру мікросервісів комплексу задач.

У розділі “Аналіз якості та тестування програмного забезпечення” було розроблено план тестування комплексу задач та виконано приклад тестування.

У розділі “Впровадження та супровід програмного забезпечення” було описано процес встановлення комплексу задач на апаратну платформу. Розроблено інструкції для користувача-програміста та для адміністратора.

У рамках даного дипломного проєкту було застосовано набуті знання з розробки баз даних, архітектури програмного забезпечення, веб розробки, розробки програм з використанням паралельних обчислень, безпеки даних, об’єктно-орієнтованого програмування, мультипарадигменого програмування.

Розроблений комплекс задач є повноцінним програмним продуктом, готовим для використання, який легко може бути доповнений новим функціоналом та масштабований.

ПЕРЕЛІК ПОСИЛАНЬ

1. CircleCI https://circleci.com - офіційна сторінка у мережі Інтернет.
2. Jenkins <https://jenkins.io> - офіційна сторінка у мережі Інтернет.
3. Helm <https://helm.sh> - офіційна сторінка у мережі Інтернет.
4. deploy-node-app <https://github.com/kubesail/deploy-node-app> - офіційна сторінка у мережі Інтернет.
5. TEST PLAN OUTLINE IEEE829 <https://www.fit.vutbr.cz/study/courses/ITS/public/ieee829.html> -  офіційний опис стандарту
6. Google Cloud Platform https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl – Configuring cluster access for kubectl (Налаштування доступу до кластеру за допомогою утиліти kubectl)
7. Google Cloud Platform <https://cloud.google.com/container-registry/docs/advanced-authentication#json_key_file> - Advanced authentication (Додаткові методи аутентіфікації)
8. Docker Documentation docs.docker.com/engine/reference/commandline/ - Docker CLI reference (Опис утиліти командного рядку Docker)
9. Kubernetes Documentation <https://kubernetes.io/docs/home/> - Документація оркестратору Kubernetes
10. Golang Docs <https://golang.org/doc/> - Документація мови Go
11. Typescript Documentation <https://www.typescriptlang.org/docs/home.html> - Документація мови Typescript

Додаток А Технічне завдання

Все эти листы не печатаются! Они нужны только для того, чтоб сделать правильное содержание

Додаток б Опис програми

додаток В ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

Все эти листы не печатаются! Они нужны только для того, чтоб сделать правильное содержание

Додаток г керівництво системного програміста  
Додаток д керівництво програміста   
Додаток є керівництво користувача

Додаток е Графічний матеріал

Лист 1. Схема структурна варіантів використань

Лист 2. Схема структурна станів системи

Лист 3. Схема бази даних

Лист 4. Схема структурна класів програмного забезпечення

Лист 5. Креслення вигляду екранних форм