

USER NETWORK

Projekt zaliczeniowy z przedmiotu *Projektowanie Zaawansowanych Systemów Informatycznych*.

Link do projektu github:

<https://github.com/IvanLyadov/usernetwork>

Autorzy:

Album: 48948, Sebastian Bartczak

Album: 45926, Ivan Lyadov

SPIS TREŚCI

[1. Wstęp i przegląd](#)

[Opis aplikacji](#)

[Zakres dokumentacji](#)

[Struktura katalogów](#)

[2. Wymagania techniczne i kroki instalacyjne](#)

[Docker Compose](#)

[Docker Desktop](#)

[Java Development Kit \(JDK\) 11](#)

[HashiCorp Vault](#)

[Maven](#)

[Git](#)

[ClickHouse](#)

[3. Instrukcja użytkownika](#)

[Podstawowe funkcje](#)

[Zarządzanie Użytkownikami](#)

[Zarządzanie Przedmiotami](#)

[Zarządzanie Politykami](#)

[Zarządzanie Poświadczeniami Vault](#)

[Szczegółowa Realizacja Funkcji](#)

[Tworzenie Nowego Użytkownika](#)

[Pobieranie Listy Użytkowników](#)

[Aktualizacja Danych Użytkownika](#)

[Usuwanie Użytkownika](#)

[Najczęstsze problemy i rozwiązania:](#)

[Problemy z instalacją Vault i jak im zapobiec:](#)

[Struktura ClickHouse:](#)

[4. Dokumentacja techniczna](#)

[Architektura aplikacji](#)

[Warstwa prezentacji \(kontrolery\)](#)

[Warstwa usług \(serwisy\)](#)

[Warstwa dostępu do danych \(repozytoria\)](#)

[Warstwa modelu \(encje\)](#)

[Warstwa konfiguracji](#)

[Pliki konfiguracyjne](#)

[Testy jednostkowe i integracyjne](#)

[API Aplikacji](#)

[API Użytkowników \(/users\)](#)

[API Przedmiotów \(/items\)](#)

[API Policies \(/policies\)](#)

[API Poświadczeń Vault \(/vault/credentials\)](#)

[5. Przewodnik](#)

[Struktura projektu](#)

[Pliki konfiguracyjne i skrypty](#)

[Katalog .mvn/wrapper](#)

[Katalog src/main/java/com/example/userapi](#)

[Katalog src/main/resources](#)

[Katalog src/test/java/com/example/userapi](#)

[Opis funkcji głównych komponentów](#)

[Konwencje kodowania](#)

[Testowanie](#)

[Historia Repozytorium Github](#)

1. Wstęp i przegląd

Opis aplikacji

Aplikacja `usernetwork-main` jest zaprojektowana do zarządzania użytkownikami, przedmiotami, politykami oraz poświadczeniami Vault. Jest to typowa aplikacja CRUD (Create, Read, Update, Delete), która umożliwia pełne zarządzanie danymi za pośrednictwem interfejsów API.

Aplikacja wykorzystuje nowoczesne technologie i narzędzia, które zapewniają skalowalność, łatwość w utrzymaniu oraz bezpieczeństwo. Technologie takie jak Java, Spring Boot, Docker, HashiCorp Vault oraz ClickHouse integrują się, aby dostarczyć pełne rozwiązanie do zarządzania danymi użytkowników, przedmiotów, polityk i poświadczeń.

Główne funkcje aplikacji obejmują:

- **Zarządzanie użytkownikami:** Tworzenie, aktualizowanie, pobieranie i usuwanie użytkowników.
- **Zarządzanie przedmiotami:** Tworzenie, aktualizowanie, pobieranie i usuwanie przedmiotów.
- **Zarządzanie politykami:** Tworzenie, aktualizowanie, pobieranie i usuwanie polityk.
- **Zarządzanie poświadczeniami Vault:** Pobieranie i aktualizowanie poświadczeń Vault.

Technologie i Narzędzia

Aplikacja `usernetwork-main` wykorzystuje szereg technologii i narzędzi, które wspomagają jej rozwój, testowanie, wdrażanie i działanie:

Język Programowania

- **Java:** Główny język programowania używany do tworzenia aplikacji.

Frameworki

- **Spring Boot:** Framework do budowania aplikacji opartych na Spring, który ułatwia konfigurację i wdrażanie aplikacji.

- **Spring Data JPA:** Abstrakcja nad JPA (Java Persistence API) ułatwiająca operacje na bazie danych.
- **Spring MVC:** Komponent Spring Framework używany do tworzenia aplikacji webowych i API.

Narzędzia do Zarządzania Projektem

- **Maven:** Narzędzie do zarządzania projektem i automatyzacji procesu budowania. Umożliwia zarządzanie zależnościami, kompilację, testowanie i wdrażanie aplikacji.

Bazy Danych

- **ClickHouse:** Kolumnowa baza danych, używana do przechowywania i analizy dużych ilości danych.
- **Spring Data JPA:** Ułatwia integrację z ClickHouse i zarządzanie operacjami na bazie danych.

Narzędzia do Konteneryzacji

- **Docker:** Platforma do tworzenia, wdrażania i uruchamiania aplikacji w kontenerach.
 - **Docker Compose:** Narzędzie do definiowania i uruchamiania aplikacji wielokontenerowych za pomocą pliku YAML.
 - **Dockerfile:** Skrypt definiujący, jak zbudować obraz Docker dla aplikacji.
 - **docker-compose.yml:** Plik konfiguracyjny definiujący usługi uruchamiane w środowisku Docker.
 - **docker-compose.vault.yml:** Plik konfiguracyjny dla Docker Compose, definiujący usługi związane z Vault.
 - **Docker Desktop**

Zarządzanie Secrets

- **HashiCorp Vault:** Narzędzie do zarządzania tajemnicami i chronionym dostępem do danych. Używane do bezpiecznego przechowywania i zarządzania poświadczeniami.

Zakres dokumentacji

Dokumentacja aplikacji "User Network" ma na celu dostarczenie kompleksowego i szczegółowego przewodnika po aplikacji, umożliwiającego różnym grupom

odbiorców (deweloperom, użytkownikom końcowym, testerom i menedżerom) pełne zrozumienie jej funkcjonalności, instalacji oraz konfiguracji.

Struktura katalogów

```
usernetwork-main/
├── .mvn/
│   └── wrapper/
│       └── maven-wrapper.properties
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/
│   │   │   │   ├── example/
│   │   │   │   │   ├── userapi/
│   │   │   │   │   │   ├── config/
│   │   │   │   │   │   │   ├── ClickhouseConfig.java
│   │   │   │   │   │   │   ├── DataSourceConfig.java
│   │   │   │   │   │   │   └── VaultConfig.java
│   │   │   │   │   │   ├── controller/
│   │   │   │   │   │   │   ├── ItemController.java
│   │   │   │   │   │   │   ├── PolicyController.java
│   │   │   │   │   │   │   ├── UserController.java
│   │   │   │   │   │   │   └── VaultCredentialsController.java
│   │   │   │   │   │   ├── model/
│   │   │   │   │   │   │   ├── Item.java
│   │   │   │   │   │   │   ├── Policy.java
│   │   │   │   │   │   │   └── User.java
│   │   │   │   │   │   ├── repository/
│   │   │   │   │   │   │   ├── ItemRepository.java
│   │   │   │   │   │   │   ├── PolicyRepository.java
│   │   │   │   │   │   │   └── UserRepository.java
│   │   │   │   │   │   └── service/
│   │   │   │   │   │       ├── ItemService.java
│   │   │   │   │   │       ├── PolicyService.java
│   │   │   │   │   │       ├── UserService.java
│   │   │   │   │   │       └── VaultService.java
│   │   └── resources/
│   │       ├── application.properties
│   │       ├── logback.xml
│   │       └── templates/
│   │           └── vault-credentials.html
│   └── test/
```

```
├── java/
│   ├── com/
│   │   ├── example/
│   │   │   ├── userapi/
│   │   │   │   ├── config/
│   │   │   │   │   ├── ClickhouseConfigTest.java
│   │   │   │   │   ├── DataSourceConfigTest.java
│   │   │   │   ├── controller/
│   │   │   │   │   ├── ItemControllerTest.java
│   │   │   │   │   ├── PolicyControllerTest.java
│   │   │   │   │   ├── UserControllerTest.java
│   │   │   │   │   └── VaultCredentialsControllerTest.java
│   │   │   │   ├── model/
│   │   │   │   │   ├── ItemTest.java
│   │   │   │   │   ├── PolicyTest.java
│   │   │   │   │   └── UserTest.java
│   │   │   │   └── service/
│   │   │   │       ├── PolicyServiceTest.java
│   │   │   │       ├── UserServiceTest.java
│   │   │   │       └── VaultServiceTest.java
│   └── resources/
├── .gitignore
├── Dockerfile
├── README.md
├── docker-compose.vault.yml
├── docker-compose.yml
├── mvnw
├── mvnw.cmd
├── pom.xml
└── vault-entypoint.sh
```

2. Wymagania techniczne i kroki instalacyjne

Docker Compose

Opis: Docker Compose to narzędzie umożliwiające definiowanie i uruchamianie wielokontenerowych aplikacji Docker.

- Umożliwia uruchomienie wszystkich komponentów aplikacji "User Network" (np. serwera aplikacji, bazy danych, Vault) w jednym poleceniu.
- Upraszcza konfigurację i zarządzanie usługami zależnymi.
- Definiowanie i uruchamianie usług aplikacji w kontenerach.
- Konfiguracja sieci i wolumenów dla kontenerów.

Instalacja z oficjalnej strony: <https://docs.docker.com/compose/install/>

The screenshot shows the Docker Docs website for the 'Overview of installing Docker Compose' page. The page is divided into three main sections: 'Overview', 'Installation scenarios', and 'Table of contents'. The 'Overview' section contains a summary of the available options for installing Docker Compose. The 'Installation scenarios' section lists three scenarios: 'Scenario one: Install Docker Desktop', 'Scenario two: Install the Compose plugin', and 'Scenario three: Install the Compose standalone'. The 'Table of contents' section provides a quick overview of the page structure. The page also includes a sidebar with navigation links for various Docker topics, a search bar, and a 'Warning' box at the bottom.

Overview of installing Docker Compose

This page contains summary information about the available options for installing Docker Compose.

Installation scenarios

Scenario one: Install Docker Desktop

The easiest and recommended way to get Docker Compose is to install Docker Desktop. Docker Desktop includes Docker Compose along with Docker Engine and Docker CLI which are Compose prerequisites.

Docker Desktop is available on:

- [Linux](#)
- [Mac](#)
- [Windows](#)

If you have already installed Docker Desktop, you can check which version of Compose you have by selecting **About Docker Desktop** from the Docker menu.

Scenario two: Install the Compose plugin

If you already have Docker Engine and Docker CLI installed, you can install the Compose plugin from the command line, by either:

- [Using Docker's repository](#)
- [Downloading and installing manually](#)

Important

This is only available on Linux

Scenario three: Install the Compose standalone

You can [install the Compose standalone](#) on Linux or on Windows Server.

Warning

Wersja dockera w cmd:


```
Wiersz polecenia X Windows PowerShell X + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Ren> docker --version
Docker version 25.0.3, build 4debf41
PS C:\Users\Ren> |
```

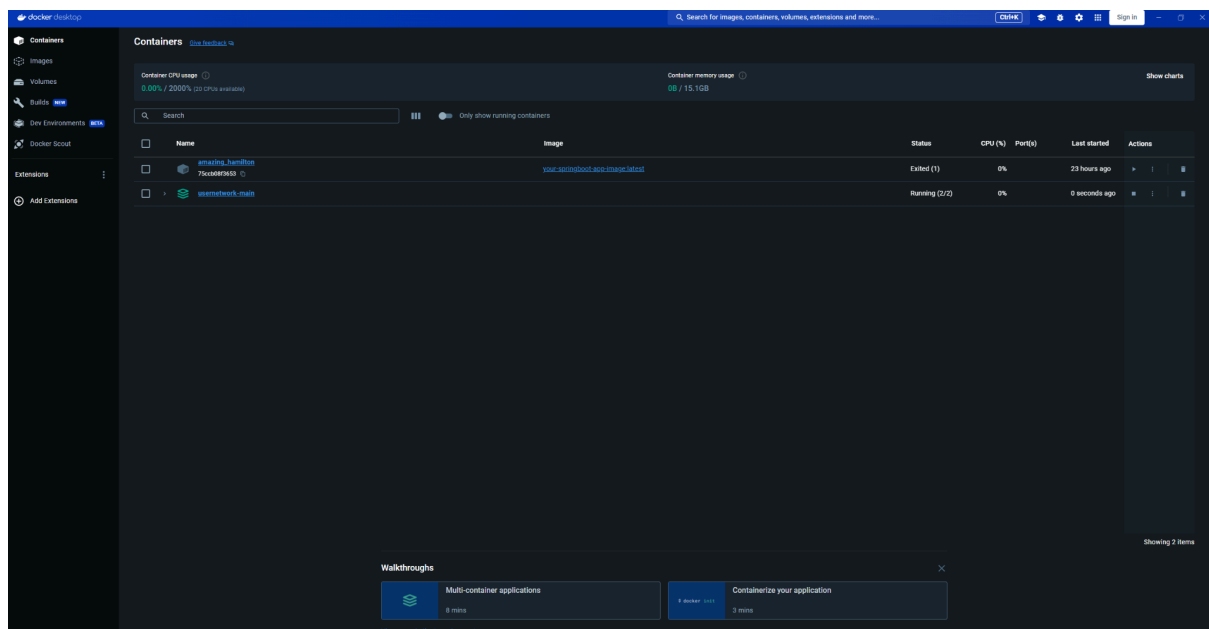
Plik docker-compose.yml:

```
1  version: '3.6'
2
3  services:
4    springboot-app:
5      image: your-springboot-app-image
6      container_name: springboot-app
7      restart: always
8      ports:
9        - "8080:8080"
10     environment:
11       VAULT_ADDR: 'http://vault:8200'
12       SPRING_PROFILES_ACTIVE: 'docker'
13     networks:
14       - sk_cloud
15
16   networks:
17     sk_cloud:
18       external: true
```

Docker Desktop

Opis: Docker Desktop to narzędzie do uruchamiania i zarządzania kontenerami Docker na komputerach z systemami Windows i macOS. Integruje się z Docker i Docker Compose, ułatwiając zarządzanie kontenerami.

- Docker Desktop zapewnia łatwe w użyciu GUI do zarządzania kontenerami, obrazami i wolumenami.
- Umożliwia uruchamianie kontenerów Docker na systemach Windows i macOS bez konieczności korzystania z linii poleceń.
- Zarządzanie kontenerami Docker w środowisku graficznym.
- Ułatwienie konfiguracji i uruchamiania aplikacji w kontenerach.



Java Development Kit (JDK) 11

Opis: JDK to zestaw narzędzi programistycznych, które umożliwiają tworzenie i uruchamianie aplikacji Java.

- Aplikacja "User Network" jest napisana w języku Java, więc JDK jest niezbędne do jej kompilacji i uruchamiania.
- JDK 11 zawiera kompilator, środowisko uruchomieniowe Java (JRE) oraz narzędzia niezbędne do zarządzania kodem Java.
- Kompilacja kodu źródłowego aplikacji.

- Uruchamianie serwera aplikacji.

Java instaluje się wraz z *docker build*. Podczas procesu budowania obrazu Docker, w pliku Dockerfile można określić, że obraz bazowy ma zawierać JDK, co pozwala na uruchomienie aplikacji Java wewnątrz kontenera.

Dzięki temu Java jest zainstalowana w kontenerze i nie ma potrzeby instalowania JDK na maszynie hosta. Wszystkie niezbędne zależności i środowisko uruchomieniowe są zdefiniowane i zarządzane w ramach obrazu Docker.

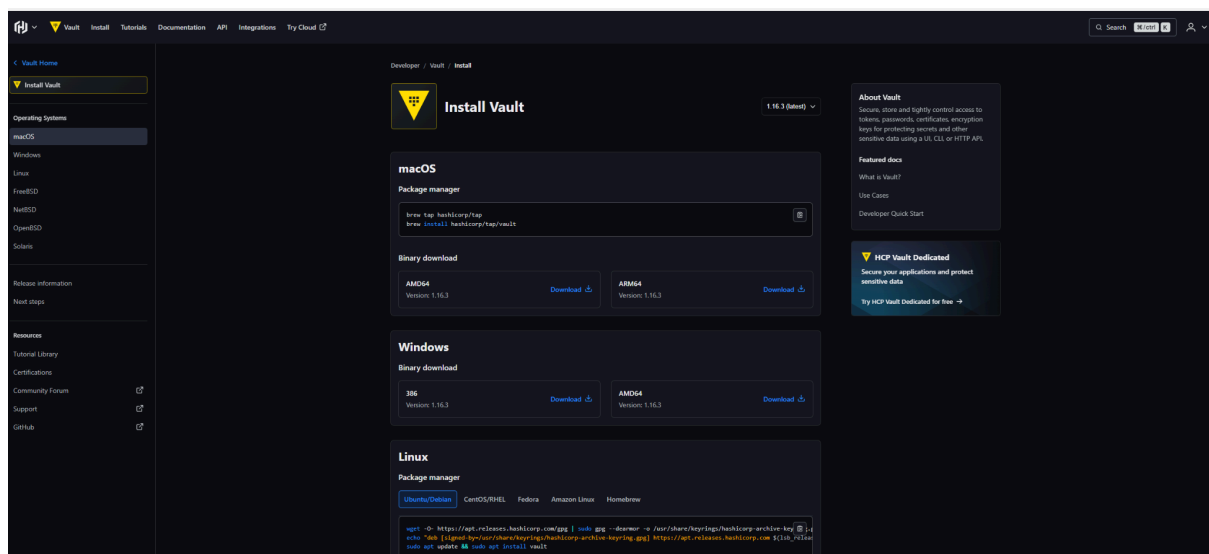
HashiCorp Vault

Opis: HashiCorp Vault to narzędzie do zarządzania tajemnicami i kontrolą dostępu do danych wrażliwych.

- Vault zapewnia bezpieczne przechowywanie i dostęp do tajemnic, takich jak klucze API, hasła, certyfikaty.
- Umożliwia centralne zarządzanie tajemnicami w aplikacji.
- Bezpieczne przechowywanie danych wrażliwych.
- Zarządzanie dostępem do tajemnic i polityk bezpieczeństwa.

Vault oficjalna strona: <https://www.vaultproject.io/>

Instalacja Vault: https://developer.hashicorp.com/vault/install?product_intent=vault



Instalacja i konfiguracja Vault przy użyciu Docker

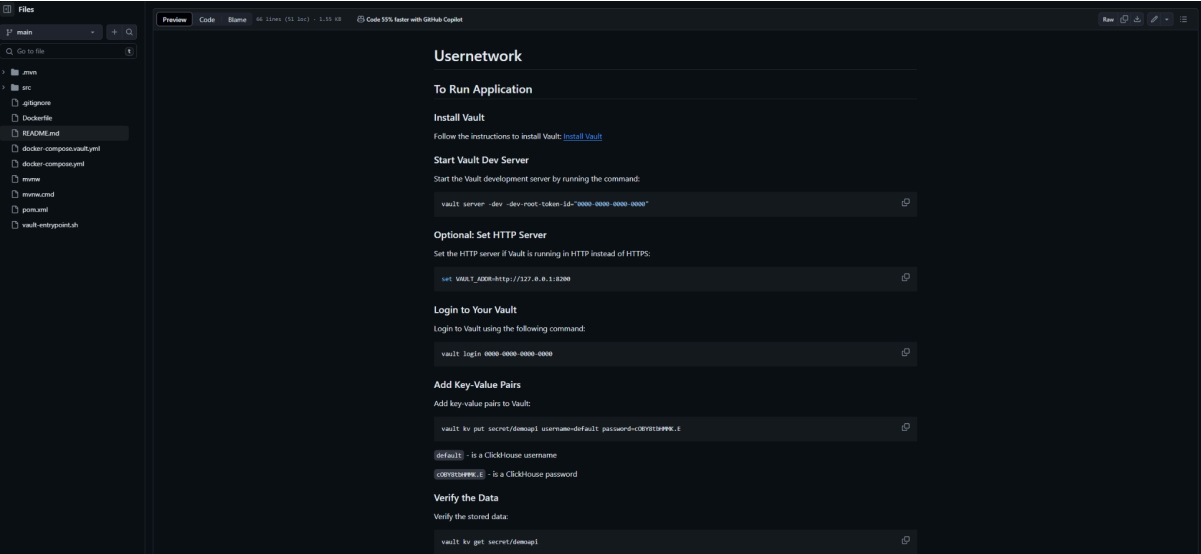
Vault instaluje się i konfiguruje wraz z Dockerem. W projekcie "User Network" do instalacji i konfiguracji Vault używamy plików Docker Compose, które definiują wszystkie usługi niezbędne do działania aplikacji, w tym HashiCorp Vault.

Plik `docker-compose.vault.yml`

Poniżej znajduje się plik `docker-compose.vault.yml`, który pokazuje, jak zainstalować i skonfigurować Vault przy użyciu Docker Compose:

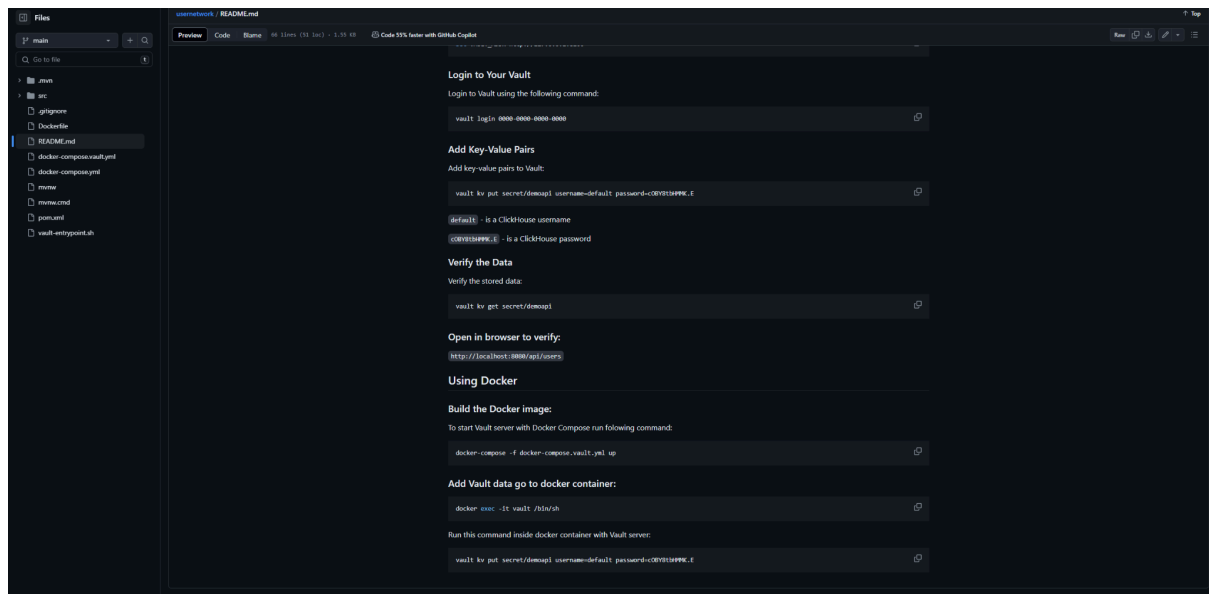
```
1  version: '3.6'
2
3  services:
4    vault:
5      image: vault:1.10.3
6      container_name: vault
7      restart: on-failure:10
8      ports:
9        - "8200:8200"
10     environment:
11       VAULT_ADDR: 'http://0.0.0.0:8200'
12       VAULT_LOCAL_CONFIG: '{"listener": [{"tcp": {"address": "0.0.0.0:8201", "tls_disable": "1"}}, {"default_lease_ttl": "168h", "max_lease_ttl": "720h", "ui": true}]}'
13       VAULT_DEV_ROOT_TOKEN_ID: '0000-0000-0000-0000'
14       VAULT_TOKEN: '0000-0000-0000-0000'
15     cap_add:
16       - IPC_LOCK
17     volumes:
18       - vault-volume:/data
19     healthcheck:
20       test: ["CMD-SHELL", "curl --silent --fail http://localhost:8200/v1/sys/health || exit 1"]
21       interval: 30s
22       timeout: 10s
23       retries: 10
24     command: server -dev -dev-root-token-id="0000-0000-0000-0000"
25     networks:
26       - sk_cloud
27
28  volumes:
29    vault-volume:
30
31  networks:
32    sk_cloud:
33      external: true
```

Instalacja Vault - plik README.md



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like `main`, `src`, `github`, `dockerfile`, `README.md`, `docker-compose.yml`, `docker-compose.yml`, `main`, `main.yml`, `main.yml`, and `vault-entrypoint.sh`. The code editor shows the content of `README.md`, which is titled "Usernetwork". The content includes sections for "To Run Application", "Install Vault", "Start Vault Dev Server", "Optional: Set HTTP Server", "Login to Your Vault", "Add Key-Value Pairs", and "Verify the Data".

```
1  To Run Application
2
3  Install Vault
4  Follow the instructions to install Vault: Install Vault
5
6  Start Vault Dev Server
7  Start the Vault development server by running the command:
8
9  vault server -dev -dev-root-token-id="0000-0000-0000-0000"
10
11 Optional: Set HTTP Server
12 Set the HTTP server if Vault is running in HTTP instead of HTTPS:
13
14 set VAULT_ADDR=http://127.0.0.1:8200
15
16 Login to Your Vault
17 Login to Vault using the following command:
18
19 vault login 0000-0000-0000-0000
20
21 Add Key-Value Pairs
22 Add key-value pairs to Vault:
23
24 vault kv put secret/demo1 username=default password=00000000-0
25
26 default - is a ClickHouse username
27 00000000-0 - is a ClickHouse password
28
29 Verify the Data
30 Verify the stored data:
31
32 vault kv get secret/demo1
```



Maven

Opis: Maven to narzędzie do zarządzania projektem i automatyzacji budowania, które jest powszechnie używane w projektach Java.

- Maven zarządza zależnościami projektu, kompilacją kodu, uruchamianiem testów i pakowaniem aplikacji.
- Ułatwia zarządzanie cyklem życia projektu.
- Kompilacja kodu źródłowego.
- Zarządzanie zależnościami i konfiguracją projektu.
- Automatyzacja procesu budowania i testowania aplikacji.

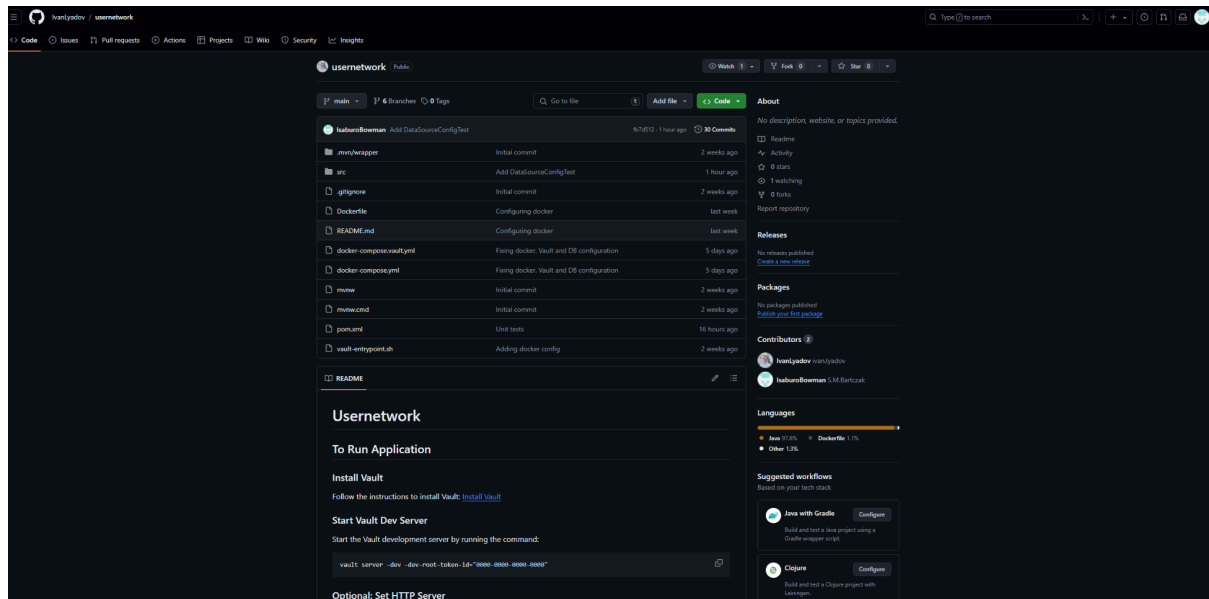
Git

Opis: Git to rozproszony system kontroli wersji, który umożliwia śledzenie zmian w kodzie źródłowym.

- Git pozwala na zarządzanie wersjami kodu źródłowego, współpracę zespołową i śledzenie historii zmian.
- Umożliwia pobieranie i aktualizowanie repozytorium kodu aplikacji.
- Śledzenie zmian w kodzie źródłowym.
- Współpraca zespołowa nad projektem.

- Zarządzanie wersjami i rozgałęzieniami kodu.

Repository GitHub:



ClickHouse

ClickHouse to system zarządzania bazą danych typu kolumnowego, zaprojektowany do przeprowadzania analitycznych zapytań SQL w czasie rzeczywistym. Jest zoptymalizowany do analizy dużych ilości danych, oferując wysoką wydajność i skalowalność. ClickHouse jest szczególnie efektywny w scenariuszach, gdzie dane są często zapisywane, a rzadko modyfikowane, i gdzie wykonywane są złożone zapytania analityczne.

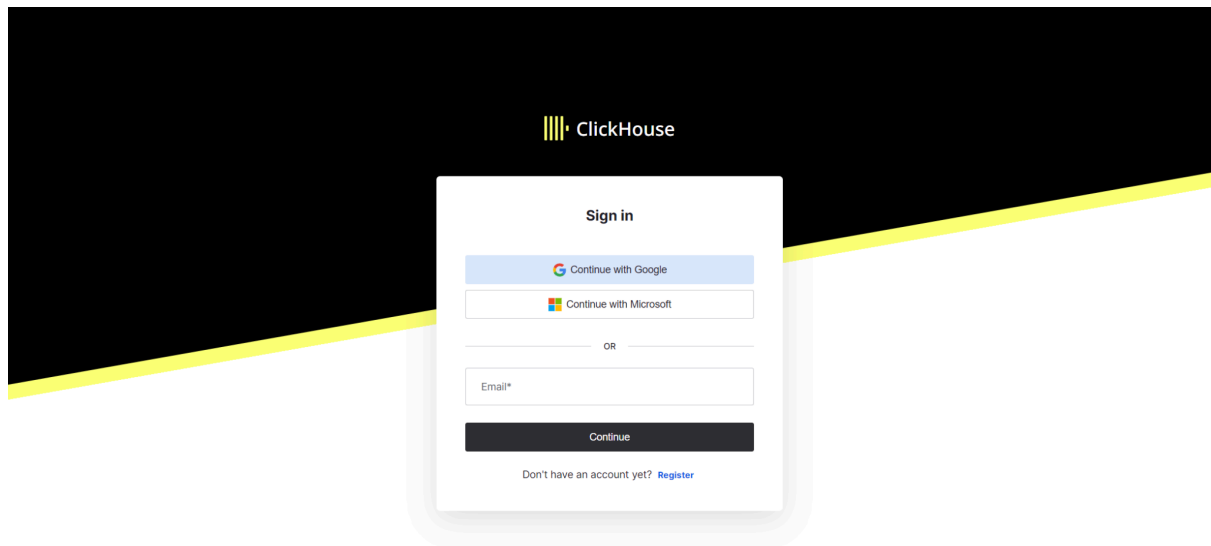
W aplikacji "User Network", ClickHouse może być używany do przechowywania i analizowania dużych ilości danych związanych z użytkownikami, ich działaniami i innymi metadanymi. Jego architektura kolumnowa pozwala na szybkie przetwarzanie zapytań analitycznych, co jest kluczowe w przypadku raportowania i analizy danych w czasie rzeczywistym.

Działanie ClickHouse

ClickHouse przechowuje dane w formacie kolumnowym, co oznacza, że dane z każdego pola są przechowywane oddzielnie. To podejście jest bardziej efektywne niż tradycyjna architektura wierszowa dla zapytań, które przetwarzają duże ilości danych w kilku kolumnach.

ClickHouse jest zbudowany w sposób umożliwiający łatwe dodawanie węzłów do klastra, co pozwala na poziome skalowanie. Dzięki temu można obsługiwać bardzo duże wolumeny danych i wysokie obciążenia zapytań.

Do ClickHouse rejestrujemy się na oficjalnej stronie <https://auth.clickhouse.cloud/>



3. Instrukcja użytkownika

Podstawowe funkcje

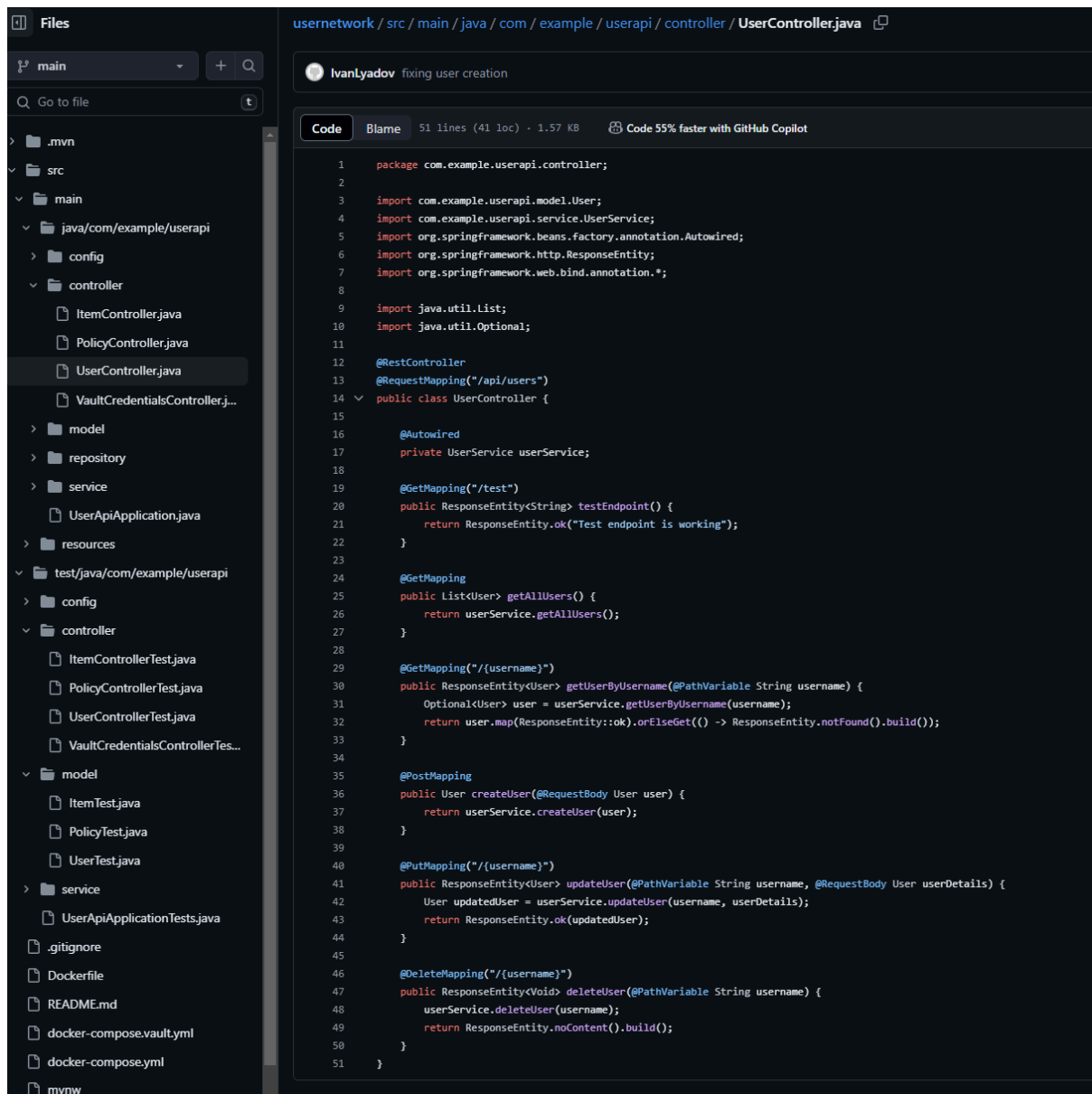
Podstawowe funkcje aplikacji usernetwork-main obejmują zarządzanie użytkownikami, przedmiotami, politykami oraz poświadczeniami Vault. Realizacja tych funkcji jest dobrze zorganizowana, z podziałem na kontrolery obsługujące żądania HTTP, serwisy zawierające logikę biznesową oraz repozytoria zapewniające operacje na bazie danych. Modele danych reprezentują strukturę przechowywanych informacji, co pozwala na efektywne zarządzanie i manipulację danymi w aplikacji.

Zarządzanie Użytkownikami

Aplikacja umożliwia pełne zarządzanie danymi użytkowników, w tym tworzenie nowych użytkowników, aktualizowanie danych istniejących użytkowników, pobieranie szczegółowych informacji oraz usuwanie użytkowników.

- **Funkcje:**
 - Tworzenie nowego użytkownika.
 - Pobieranie listy wszystkich użytkowników.

- Pobieranie szczegółowych informacji o użytkowniku na podstawie jego ID.
- Aktualizacja danych użytkownika.
- Usuwanie użytkownika.
- **Realizacja:**
 - Klasa `UserController` odpowiada za obsługę żądań HTTP związanych z użytkownikami.



```
1 package com.example.userapi.controller;
2
3 import com.example.userapi.model.User;
4 import com.example.userapi.service.UserService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.*;
8
9 import java.util.List;
10 import java.util.Optional;
11
12 @RestController
13 @RequestMapping("/api/users")
14 public class UserController {
15
16     @Autowired
17     private UserService userService;
18
19     @GetMapping("/test")
20     public ResponseEntity<String> testEndpoint() {
21         return ResponseEntity.ok("Test endpoint is working");
22     }
23
24     @GetMapping
25     public List<User> getAllUsers() {
26         return userService.getAllUsers();
27     }
28
29     @GetMapping("/{username}")
30     public ResponseEntity<User> getUserByUsername(@PathVariable String username) {
31         Optional<User> user = userService.getUserByUsername(username);
32         return user.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
33     }
34
35     @PostMapping
36     public User createUser(@RequestBody User user) {
37         return userService.createUser(user);
38     }
39
40     @PutMapping("/{username}")
41     public ResponseEntity<User> updateUser(@PathVariable String username, @RequestBody User userDetails) {
42         User updatedUser = userService.updateUser(username, userDetails);
43         return ResponseEntity.ok(updatedUser);
44     }
45
46     @DeleteMapping("/{username}")
47     public ResponseEntity<Void> deleteUser(@PathVariable String username) {
48         userService.deleteUser(username);
49         return ResponseEntity.noContent().build();
50     }
51 }
```

- Klasa `UserService` zawiera logikę biznesową związaną z operacjami na użytkownikach.

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'main' package with 'controller', 'model', 'repository', and 'service' sub-packages. The 'UserService.java' file is selected in the 'service' package. The code in the editor is as follows:

```
1 package com.example.userapi.service;
2
3 import com.example.userapi.model.User;
4 import com.example.userapi.repository.UserRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import org.springframework.vault.core.VaultKeyValueOperationsSupport;
8 import org.springframework.vault.core.VaultTemplate;
9
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13 import java.util.Optional;
14 import java.util.UUID;
15
16 @Service
17 public class UserService {
18
19     @Autowired
20     private UserRepository userRepository;
21
22     @Autowired
23     private VaultTemplate vaultTemplate;
24
25     public List<User> getAllUsers() {
26         return userRepository.findAll();
27     }
28
29     public Optional<User> getUserByUsername(String username) {
30         return userRepository.findById(username);
31     }
32
33     public User createUser(User user) {
34         // Generate a unique ID for the user
35         String userId = UUID.randomUUID().toString();
36         user.setid(userId);
37
38         // Save user in Clickhouse
39         User savedUser = userRepository.save(user);
40
41         // Generate random token
42         String token = UUID.randomUUID().toString();
43
44         // Store token in Vault
45         storeTokenInVault(savedUser.getUsername(), token);
46
47         return savedUser;
48     }
49
50     public User updateUser(String username, User userDetails) {
51         User user = userRepository.findById(username).orElseThrow(() -> new RuntimeException("User not found"));
52         user.setEmail(userDetails.getEmail());
53         return userRepository.save(user);
54     }
55
56     public void deleteUser(String username) {
57         userRepository.deleteById(username);
58         deleteTokenFromVault(username);
59     }
60 }
```

- Interfejs `UserRepository` zapewnia operacje CRUD na użytkownikach w bazie danych.

```
Code Blame 9 lines (7 loc) · 277 Bytes Code 55% faster with GitHub Copilot

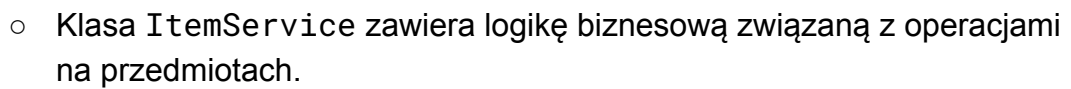
1 package com.example.userapi.repository;
2
3 import com.example.userapi.model.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface UserRepository extends JpaRepository<User, String> {
9 }
```

- Klasa User jest modelem danych reprezentującym użytkownika.

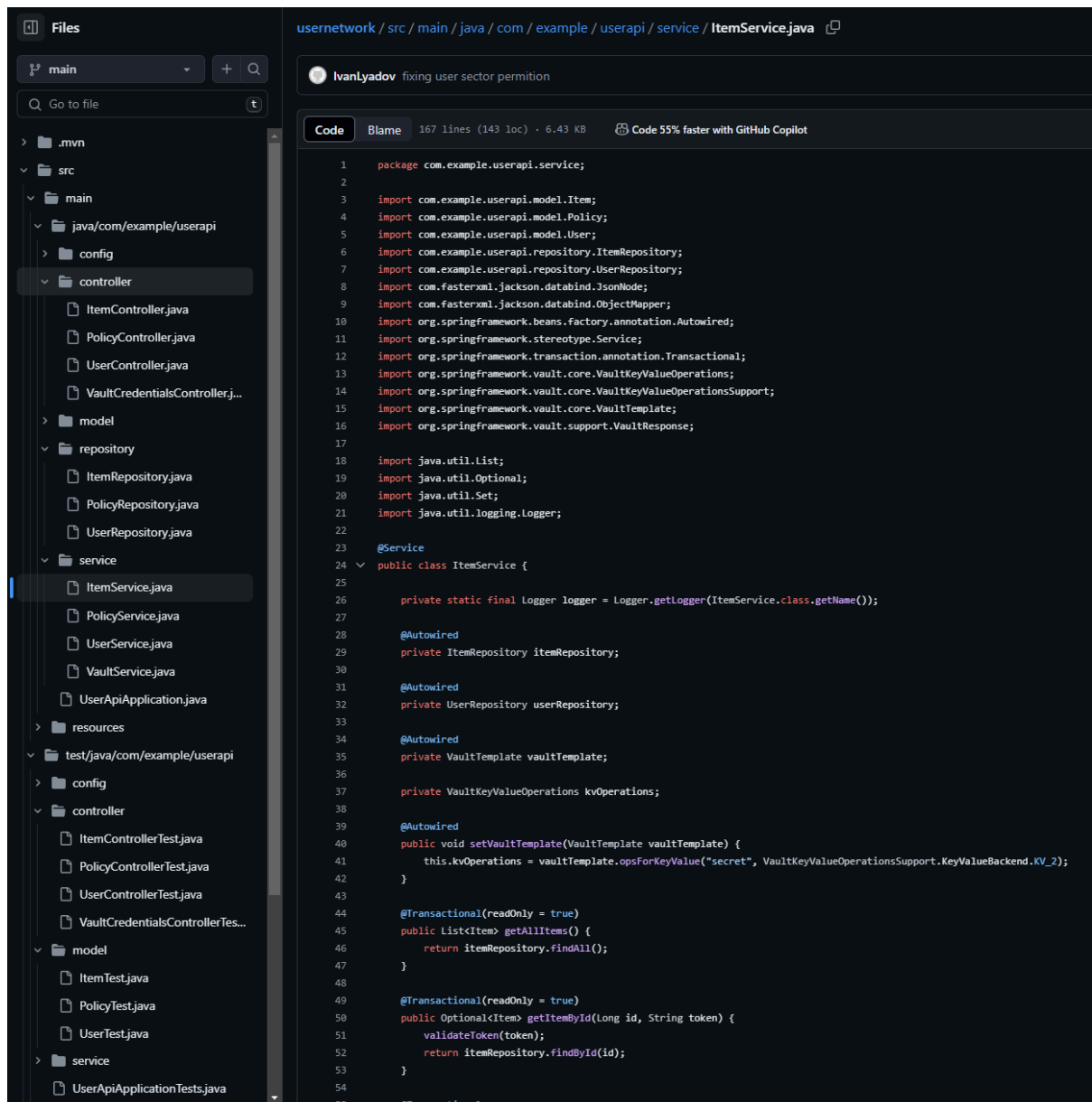
Zarządzanie Przedmiotami

Aplikacja umożliwia zarządzanie przedmiotami, w tym tworzenie nowych przedmiotów, aktualizowanie danych istniejących przedmiotów, pobieranie szczegółowych informacji oraz usuwanie przedmiotów.

- **Funkcje:**
 - Tworzenie nowego przedmiotu.
 - Pobieranie listy wszystkich przedmiotów.
 - Pobieranie szczegółowych informacji o przedmiocie na podstawie jego ID.
 - Aktualizacja danych przedmiotu.
 - Usuwanie przedmiotu.
- **Realizacja:**
 - Klasa `ItemController` odpowiada za obsługę żądań HTTP związanych z przedmiotami.



- Klasa `ItemService` zawiera logikę biznesową związaną z operacjami na przedmiotach.



- Interfejs ItemRepository zapewnia operacje CRUD na przedmiotach w bazie danych.

```
Code Blame 12 lines (9 loc) · 359 Bytes Code 55% faster with GitHub Copilot

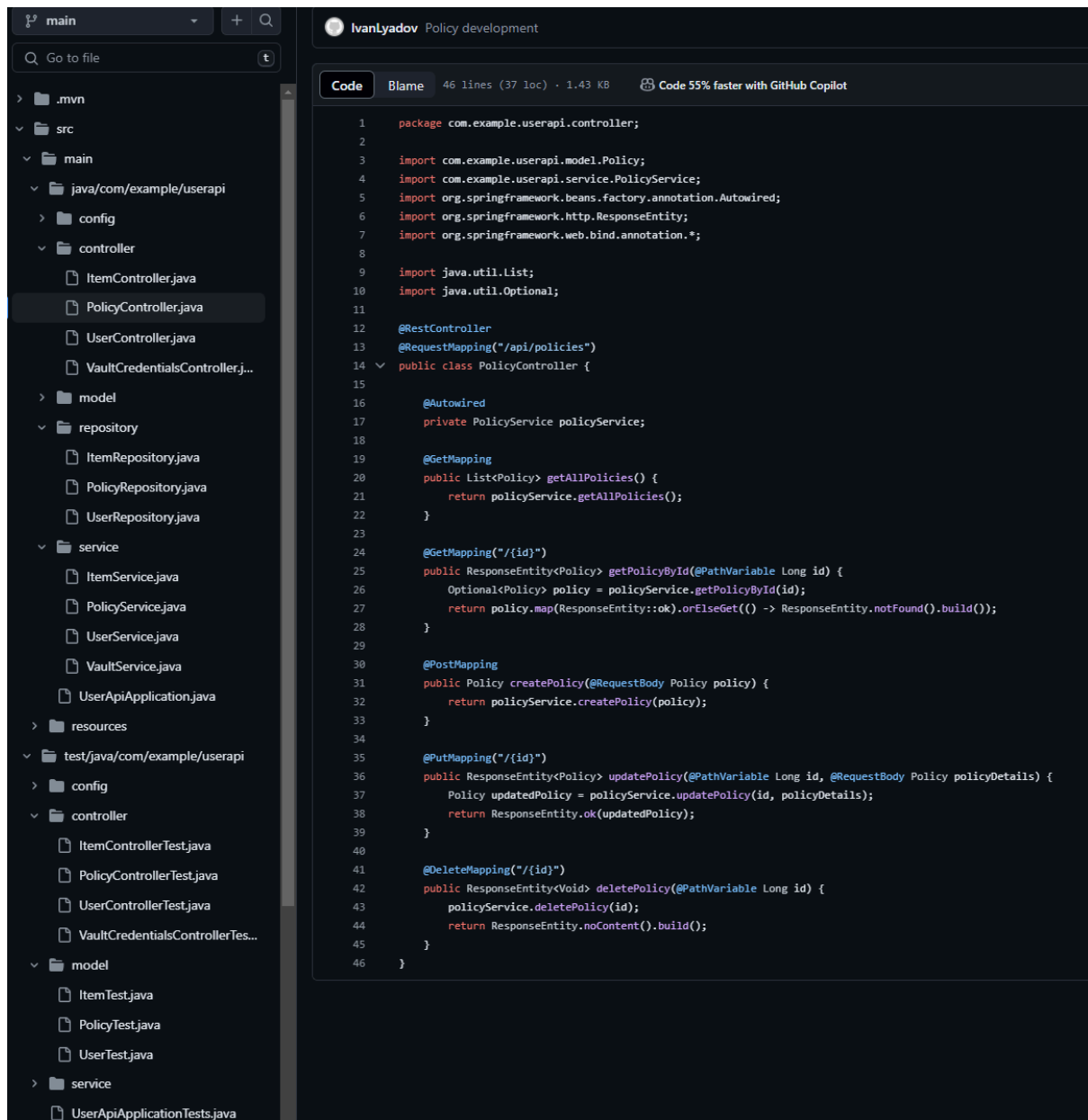
1 package com.example.userapi.repository;
2
3 import com.example.userapi.model.Item;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 import java.util.List;
8
9 @Repository
10 public interface ItemRepository extends JpaRepository<Item, Long> {
11     List<Item> findBySectorId(String sectorId);
12 }
```

- Klasa `Item` jest modelem danych reprezentującym przedmiot.

Zarządzanie Politykami

Aplikacja umożliwia zarządzanie politykami, w tym tworzenie nowych polityk, aktualizowanie danych istniejących polityk, pobieranie szczegółowych informacji oraz usuwanie polityk.

- **Funkcje:**
 - Tworzenie nowej polityki.
 - Pobieranie listy wszystkich polityk.
 - Pobieranie szczegółowych informacji o polityce na podstawie jej ID.
 - Aktualizacja danych polityki.
 - Usuwanie polityki.
- **Realizacja:**
 - Klasa `PolicyController` odpowiada za obsługę żądań HTTP związanych z politykami.



- Klasa `PolicyService` zawiera logikę biznesową związaną z operacjami na politykach.

```
1 package com.example.userapi.service;
2
3 import com.example.userapi.model.Policy;
4 import com.example.userapi.repository.PolicyRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @Service
12 public class PolicyService {
13
14     @Autowired
15     private PolicyRepository policyRepository;
16
17     public List<Policy> getAllPolicies() {
18         return policyRepository.findAll();
19     }
20
21     public Optional<Policy> getPolicyById(Long id) {
22         return policyRepository.findById(id);
23     }
24
25     public Policy createPolicy(Policy policy) {
26         return policyRepository.save(policy);
27     }
28
29     public Policy updatePolicy(Long id, Policy policyDetails) {
30         Policy policy = policyRepository.findById(id).orElseThrow(() -> new RuntimeException("Policy not found"));
31         policy.setName(policyDetails.getName());
32         policy.setDefinition(policyDetails.getDefinition());
33         return policyRepository.save(policy);
34     }
35
36     public void deletePolicy(Long id) {
37         policyRepository.deleteById(id);
38     }
39 }
```

- Interfejs PolicyRepository zapewnia operacje CRUD na politykach w bazie danych.

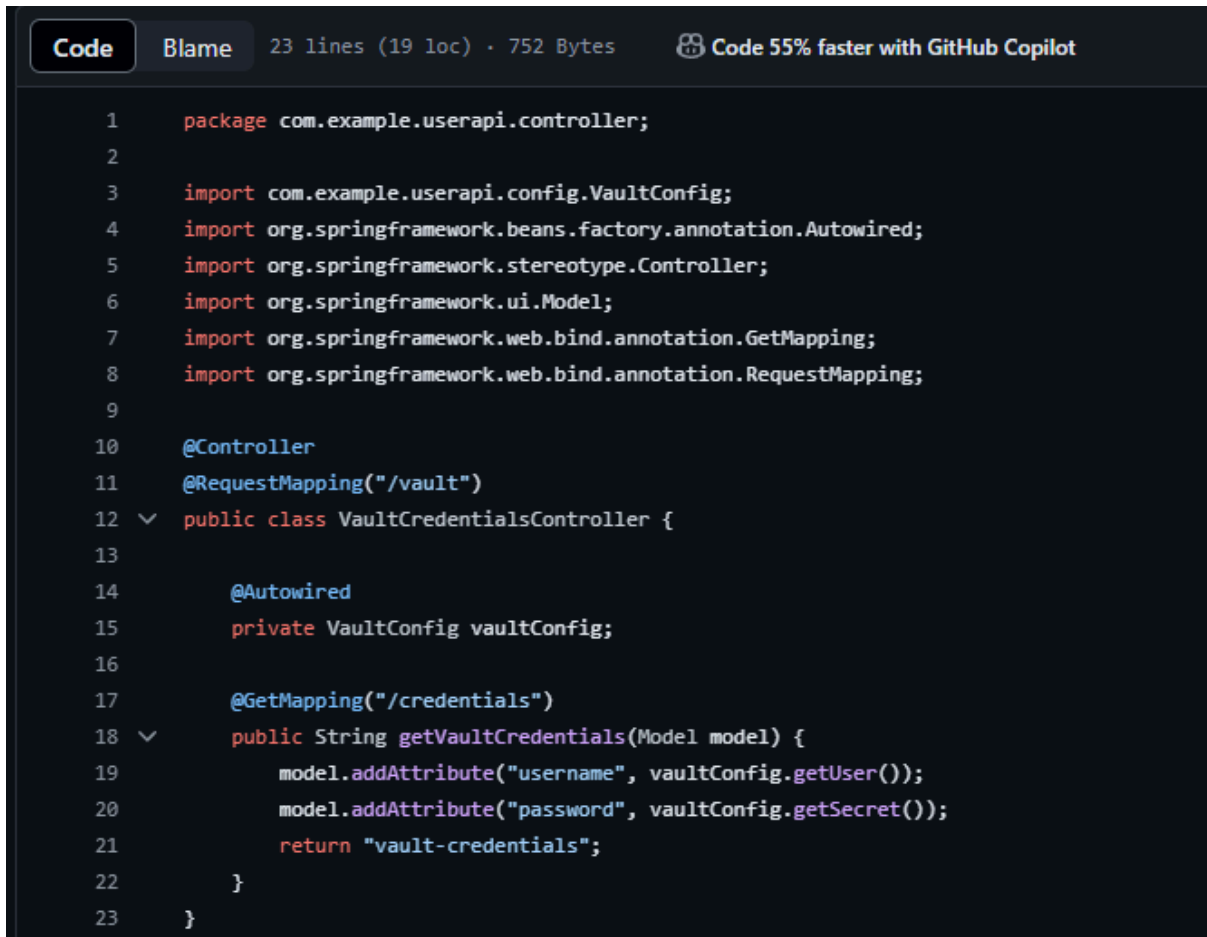
```
1 package com.example.userapi.repository;
2
3 import com.example.userapi.model.Policy;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface PolicyRepository extends JpaRepository<Policy, Long> {
9 }
```

- Klasa Policy jest modelem danych reprezentującym politykę.

Zarządzanie Poświadczeniami Vault

Aplikacja umożliwia zarządzanie poświadczeniami Vault, w tym pobieranie aktualnych poświadczeń oraz ich aktualizowanie.

- **Funkcje:**
 - Pobieranie aktualnych poświadczeń Vault.
 - Aktualizacja poświadczeń Vault.
- **Realizacja:**
 - Klasa VaultCredentialsController odpowiada za obsługę żądań HTTP związanych z poświadczeniami Vault.



The screenshot shows a code editor with a dark theme. At the top, there are tabs for 'Code' and 'Blame', and a status bar indicating '23 lines (19 loc) · 752 Bytes'. A badge on the right says 'Code 55% faster with GitHub Copilot'. The code is in Java and defines the VaultCredentialsController class. It includes imports for VaultConfig, Autowired, Controller, Model, GetMapping, and RequestMapping. The class is annotated with @Controller and @RequestMapping("/vault"). It has a private VaultConfig field and a public getVaultCredentials method that returns a string 'vault-credentials'.

```
1 package com.example.userapi.controller;
2
3 import com.example.userapi.config.VaultConfig;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.ui.Model;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.RequestMapping;
9
10 @Controller
11 @RequestMapping("/vault")
12 public class VaultCredentialsController {
13
14     @Autowired
15     private VaultConfig vaultConfig;
16
17     @GetMapping("/credentials")
18     public String getVaultCredentials(Model model) {
19         model.addAttribute("username", vaultConfig.getUser());
20         model.addAttribute("password", vaultConfig.getSecret());
21         return "vault-credentials";
22     }
23 }
```

- Klasa VaultService zawiera logikę biznesową związaną z operacjami na poświadczeniach Vault.


```
Code Blame 33 lines (26 loc) · 1.13 KB Code 55% faster with GitHub Copilot

1  package com.example.userapi.service;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Service;
5  import org.springframework.vault.core.VaultKeyValueOperationsSupport;
6  import org.springframework.vault.core.VaultTemplate;
7  import org.springframework.vault.support.VaultResponse;
8
9  import java.util.HashMap;
10 import java.util.Map;
11
12 @Service
13 public class VaultService {
14
15     @Autowired
16     private VaultTemplate vaultTemplate;
17
18     public void createOrUpdateSecret(String path, Map<String, String> data) {
19         vaultTemplate.opsForKeyValue("secret", VaultKeyValueOperationsSupport.KeyValueBackend.KV_2)
20             .put(path, data);
21     }
22
23     public Map<String, Object> readSecret(String path) {
24         VaultResponse response = vaultTemplate.opsForKeyValue("secret", VaultKeyValueOperationsSupport.KeyValueBackend.KV_2)
25             .get(path);
26         return response != null ? response.getData() : null;
27     }
28
29     public void deleteSecret(String path) {
30         vaultTemplate.opsForKeyValue("secret", VaultKeyValueOperationsSupport.KeyValueBackend.KV_2)
31             .delete(path);
32     }
33 }
```

Szczegółowa Realizacja Funkcji

Tworzenie Nowego Użytkownika

- **Opis:** Endpoint POST /users pozwala na tworzenie nowego użytkownika. Klient przesyła dane nowego użytkownika w formacie JSON.
- **Proces:**
 - Żądanie trafia do UserController, który wywołuje metodę createUser z klasy UserService.
 - UserService przetwarza dane i zapisuje nowego użytkownika za pomocą UserRepository.

Pobieranie Listy Użytkowników

- **Opis:** Endpoint GET /users zwraca listę wszystkich użytkowników.
- **Proces:**
 - Żądanie trafia do UserController, który wywołuje metodę getAllUsers z klasy UserService.

- UserService pobiera listę użytkowników za pomocą UserRepository i zwraca ją do kontrolera, który zwraca odpowiedź do klienta.

Aktualizacja Danych Użytkownika

- **Opis:** Endpoint PUT /users/{id} pozwala na aktualizację danych istniejącego użytkownika na podstawie jego ID.
- **Proces:**
 - Żądanie trafia do UserController, który wywołuje metodę updateUser z klasy UserService.
 - UserService aktualizuje dane użytkownika za pomocą UserRepository.

Usuwanie Użytkownika

- **Opis:** Endpoint DELETE /users/{id} pozwala na usunięcie użytkownika na podstawie jego ID.
- **Proces:**
 - Żądanie trafia do UserController, który wywołuje metodę deleteUser z klasy UserService.
 - UserService usuwa użytkownika za pomocą UserRepository.

Najczęstsze problemy i rozwiązania:

Problemy z instalacją Vault i jak im zapobiec:

Typowe problemy z instalacją Vault

- A. Niepoprawna konfiguracja sieci
- B. Brak uprawnień do uruchomienia Vault
- C. Niepoprawne zmienne środowiskowe
- D. Problemy z połączeniem do Vault
- E. Niewłaściwe wersje Vault i Docker

Jak zapobiegać problemom

A. Niepoprawna konfiguracja sieci

Opis: Vault może nie uruchomić się poprawnie, jeśli występują problemy z konfiguracją sieci, takie jak konflikty portów lub błędne ustawienia sieciowe.

Zapobieganie:

- Upewnić się, że porty używane przez Vault (domyślnie 8200) nie są zajęte przez inne aplikacje.

W pliku **docker-compose.yml**, upewnić się, że porty są poprawnie zmapowane:

```
services:
```

```
  vault:
```

```
    image: vault:latest
```

```
    container_name: vault
```

```
    ports:
```

```
      - "8200:8200"
```

B. Brak uprawnień do uruchomienia Vault

Opis: Vault wymaga odpowiednich uprawnień do zapisu danych i logów, co może być problemem, jeśli kontener nie ma wymaganych uprawnień do dostępu do systemu plików.

Zapobieganie:

- Upewnić się, że użytkownik uruchamiający kontener Docker ma odpowiednie uprawnienia do zapisu w katalogach używanych przez Vault.

Można skonfigurować odpowiednie wolumeny, aby zapewnić trwałość danych:

```
volumes:
```

```
- vault-data:/vault/data
```

C. Niepoprawne zmienne środowiskowe

Opis: Vault może nie uruchomić się poprawnie, jeśli zmienne środowiskowe są niepoprawnie skonfigurowane.

Zapobieganie:

Upewnić się, że wszystkie wymagane zmienne środowiskowe są poprawnie ustawione. Przykład konfiguracji:

environment:

VAULT_DEV_ROOT_TOKEN_ID: root

VAULT_DEV_LISTEN_ADDRESS: "0.0.0.0:8200"

D. Problemy z połączeniem do Vault

Opis: Aplikacja może mieć problemy z połączeniem do Vault, jeśli adres URL lub token autoryzacyjny są niepoprawne.

Zapobieganie:

Upewnić się, że adres URL i token autoryzacyjny są poprawnie skonfigurowane w pliku `application.properties` lub `application.yml`:

`vault.uri=http://localhost:8200`

`vault.token=root`

E. Niewłaściwe wersje Vault i Docker

Opis: Niekompatybilne wersje Vault i Docker mogą powodować problemy z instalacją i uruchamianiem kontenera.

Zapobieganie:

- Upewnić się, że używasz kompatybilnych wersji Vault i Docker.
Sprawdzić dokumentację Vault i Docker, aby upewnić się, że wersje są zgodne.

Używać stabilnych wersji obrazów Docker. Przykładowa konfiguracja:

`image: vault:latest`

Diagnostyka i rozwiązywanie problemów

Jeśli napotkane będą problemy z uruchamianiem Vault, oto kilka kroków, które mogą pomóc w diagnozie i rozwiązaniu problemu:

Sprawdzić logi kontenera:

```
docker logs vault
```

- Logi mogą dostarczyć informacji na temat przyczyny problemu.

Sprawdzić status kontenera:

```
docker ps -a
```

- Upewnić się, że kontener Vault jest uruchomiony i działa poprawnie.

Testowanie połączenia do Vault: Użyć narzędzi takich jak curl, aby przetestować połączenie do Vault:

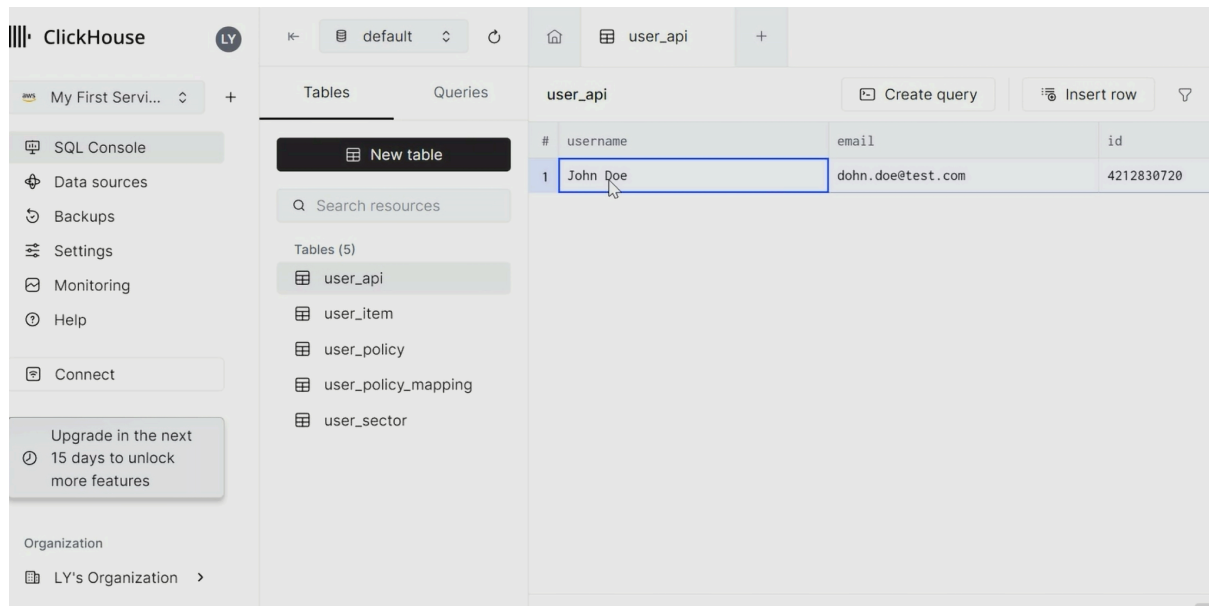
```
curl http://localhost:8200/v1/sys/health
```

Zrestartować usługi: Jeśli problem nadal występuje, spróbować zrestartować kontenery Docker:

```
docker-compose down
```

```
docker-compose up
```

Struktura ClickHouse:



The screenshot shows the ClickHouse web interface. On the left is a sidebar with navigation links: SQL Console, Data sources, Backups, Settings, Monitoring, Help, and Connect. Below these is a notification about upgrading and an organization section. The main area is divided into 'Tables' and 'Queries' tabs. Under 'Tables', a list of tables is shown: user_api, user_item, user_policy, user_policy_mapping, and user_sector. The 'user_api' table is selected, and its structure is displayed in a table format. The table has three columns: #, username, email, and id. A single row of data is visible, with a blue selection box highlighting the 'username' cell containing 'John Doe'.

#	username	email	id
1	John Doe	dohn.doe@test.com	4212830720

4. Dokumentacja techniczna

Architektura aplikacji

Architektura aplikacji usernetwork-main jest oparta na wzorcach projektowych stosowanych w projektach Java z wykorzystaniem Spring Boot. Aplikacja jest modularna, co pozwala na łatwą skalowalność, testowanie oraz utrzymanie. Poniżej znajduje się szczegółowy opis poszczególnych elementów architektury aplikacji.

Warstwa prezentacji (kontrolery)

Warstwa prezentacji jest odpowiedzialna za interakcję z użytkownikami aplikacji. W tej warstwie znajdują się kontrolery, które obsługują żądania HTTP, przetwarzają dane wejściowe, wywołują odpowiednie metody serwisowe oraz zwracają odpowiedzi. Kontrolery znajdują się w pakiecie `com.example.userapi.controller`.

- ItemController: Obsługuje żądania związane z operacjami na przedmiotach.
- PolicyController: Obsługuje żądania związane z politykami.
- UserController: Obsługuje żądania związane z użytkownikami.

- VaultCredentialsController: Obsługuje żądania związane z poświadczeniami Vault.

Warstwa usług (serwisy)

Warstwa usług zawiera logikę biznesową aplikacji. Serwisy przetwarzają dane, korzystają z repozytoriów do komunikacji z bazą danych oraz wykonują inne operacje biznesowe. Serwisy znajdują się w pakiecie `com.example.userapi.service`.

- ItemService: Zarządza logiką biznesową związaną z przedmiotami.
- PolicyService: Zarządza logiką biznesową związaną z politykami.
- UserService: Zarządza logiką biznesową związaną z użytkownikami.
- VaultService: Zarządza operacjami związanymi z poświadczeniami Vault.

Warstwa dostępu do danych (repozytoria)

Warstwa dostępu do danych jest odpowiedzialna za interakcję z bazą danych. Repozytoria zawierają metody do wykonywania operacji CRUD (Create, Read, Update, Delete) na danych. Repozytoria znajdują się w pakiecie `com.example.userapi.repository`.

- ItemRepository: Interfejs do operacji na przedmiotach.
- PolicyRepository: Interfejs do operacji na politykach.
- UserRepository: Interfejs do operacji na użytkownikach.

Warstwa modelu (encje)

Warstwa modelu zawiera klasy reprezentujące strukturę danych w aplikacji i mapujące te dane na tabele w bazie danych. Modele znajdują się w pakiecie `com.example.userapi.model`.

- Item: Klasa reprezentująca przedmiot.
- Policy: Klasa reprezentująca politykę.
- User: Klasa reprezentująca użytkownika.

Warstwa konfiguracji

Warstwa konfiguracji zawiera klasy konfiguracyjne aplikacji, które definiują ustawienia różnych komponentów, takich jak baza danych, Vault itp. Klasy konfiguracyjne znajdują się w pakiecie `com.example.userapi.config`.

- ClickhouseConfig: Konfiguracja połączenia z Clickhouse.
- DataSourceConfig: Konfiguracja źródła danych.
- VaultConfig: Konfiguracja połączenia z HashiCorp Vault.

Pliki konfiguracyjne

- `application.properties`: Główny plik konfiguracyjny aplikacji Spring Boot. Zawiera ustawienia aplikacji, takie jak połączenia do baz danych, konfiguracje serwera, itp.
- `logback.xml`: Plik konfiguracyjny logowania za pomocą Logback. Definiuje zasady i formaty logowania w aplikacji.

Testy jednostkowe i integracyjne

Testy są zorganizowane w strukturze równoległej do głównej aplikacji w katalogu `src/test/java/com/example/userapi`. Testy te zapewniają, że poszczególne komponenty aplikacji działają poprawnie i zgodnie z oczekiwaniami.

- Testy dla kontrolerów, np. `ItemControllerTest`, `PolicyControllerTest`, `UserControllerTest`, `VaultCredentialsControllerTest`.
- Testy dla serwisów, np. `PolicyServiceTest`, `UserServiceTest`, `VaultServiceTest`.
- Testy dla modeli, np. `ItemTest`, `PolicyTest`, `UserTest`.
- Testy dla konfiguracji, np. `ClickhouseConfigTest`, `DataSourceConfigTest`.

API Aplikacji

Poniżej znajduje się szczegółowy opis każdego endpointu API dostępnego w aplikacji `usernetwork-main`. Opisy te zawierają informacje o funkcjonalności, parametrach oraz zwracanych danych, zgodnie z danymi z repozytorium.

API Użytkowników (`/users`)

Kontroler: `UserController`

Endpoints:

- `GET /users`
 - Opis: Pobiera listę wszystkich użytkowników.
 - Parametry: Brak
 - Zwracane dane: Lista użytkowników w formacie JSON.
- `GET /users/{id}`

- Opis: Pobiera szczegółowe informacje o użytkowniku na podstawie jego ID.
- Parametry: `id` - ID użytkownika (ścieżka)
- Zwracane dane: Informacje o użytkowniku w formacie JSON.
- **POST /users**
 - Opis: Tworzy nowego użytkownika.
 - Parametry: Body JSON z danymi użytkownika
 - Zwracane dane: Informacje o utworzonym użytkowniku w formacie JSON.
- **PUT /users/{id}**
 - Opis: Aktualizuje istniejącego użytkownika na podstawie jego ID.
 - Parametry: `id` - ID użytkownika (ścieżka), Body JSON z aktualizowanymi danymi
 - Zwracane dane: Zaktualizowane informacje o użytkowniku w formacie JSON.
- **DELETE /users/{id}**
 - Opis: Usuwa użytkownika na podstawie jego ID.
 - Parametry: `id` - ID użytkownika (ścieżka)
 - Zwracane dane: Brak (HTTP 204 No Content)

API Przedmiotów (/items)

Kontroler: `ItemController`

Endpoints:

- **GET /items**
 - Opis: Pobiera listę wszystkich przedmiotów.
 - Parametry: Brak
 - Zwracane dane: Lista przedmiotów w formacie JSON.
- **GET /items/{id}**
 - Opis: Pobiera szczegółowe informacje o przedmiocie na podstawie jego ID.
 - Parametry: `id` - ID przedmiotu (ścieżka)
 - Zwracane dane: Informacje o przedmiocie w formacie JSON.
- **POST /items**
 - Opis: Tworzy nowy przedmiot.
 - Parametry: Body JSON z danymi przedmiotu
 - Zwracane dane: Informacje o utworzonym przedmiocie w formacie JSON.
- **PUT /items/{id}**
 - Opis: Aktualizuje istniejący przedmiot na podstawie jego ID.

- Parametry: id - ID przedmiotu (ścieżka), Body JSON z aktualizowanymi danymi
- Zwracane dane: Zaktualizowane informacje o przedmiocie w formacie JSON.
- DELETE /items/{id}
 - Opis: Usuwa przedmiot na podstawie jego ID.
 - Parametry: id - ID przedmiotu (ścieżka)
 - Zwracane dane: Brak (HTTP 204 No Content)

API Policies (/policies)

Kontroler: PolicyController

Endpoints:

- GET /policies
 - Opis: Pobiera listę wszystkich polityk.
 - Parametry: Brak
 - Zwracane dane: Lista polityk w formacie JSON.
- GET /policies/{id}
 - Opis: Pobiera szczegółowe informacje o polityce na podstawie jej ID.
 - Parametry: id - ID polityki (ścieżka)
 - Zwracane dane: Informacje o polityce w formacie JSON.
- POST /policies
 - Opis: Tworzy nową politykę.
 - Parametry: Body JSON z danymi polityki
 - Zwracane dane: Informacje o utworzonej polityce w formacie JSON.
- PUT /policies/{id}
 - Opis: Aktualizuje istniejącą politykę na podstawie jej ID.
 - Parametry: id - ID polityki (ścieżka), Body JSON z aktualizowanymi danymi
 - Zwracane dane: Zaktualizowane informacje o polityce w formacie JSON.
- DELETE /policies/{id}
 - Opis: Usuwa politykę na podstawie jej ID.
 - Parametry: id - ID polityki (ścieżka)
 - Zwracane dane: Brak (HTTP 204 No Content)

API Poświadczeń Vault (/vault/credentials)

Kontroler: VaultCredentialsController

Endpoints:

- GET /vault/credentials
 - Opis: Pobiera poświadczenia Vault.
 - Parametry: Brak
 - Zwracane dane: Poświadczenia Vault w formacie JSON.
- POST /vault/credentials
 - Opis: Aktualizuje poświadczenia Vault.
 - Parametry: Body JSON z nowymi poświadczeniami
 - Zwracane dane: Brak (HTTP 200 OK)

Aplikacja `usernetwork-main` dostarcza bogaty zestaw punktów końcowych API umożliwiających zarządzanie użytkownikami, przedmiotami, politykami oraz poświadczeniami Vault. Każdy z tych punktów końcowych jest dobrze zdefiniowany, oferując standardowe operacje CRUD. Dzięki przejrzystej strukturze API, aplikacja jest łatwa w użyciu i integracji z innymi systemami.

5. Przewodnik

Struktura projektu

Struktura projektu zawartego w repozytorium `usernetwork-main` jest zorganizowana zgodnie z najlepszymi praktykami stosowanymi w projektach Java, opartymi na Spring Boot. Poniżej znajduje się dokładny opis poszczególnych elementów i plików w projekcie:

Pliki konfiguracyjne i skrypty

- **.gitignore**: Plik zawierający listę plików i folderów, które mają być ignorowane przez system kontroli wersji Git.
- **Dockerfile**: Skrypt używany przez Docker do budowania obrazu kontenera aplikacji.
- **README.md**: Plik z opisem projektu, instrukcjami uruchomienia, konfiguracją oraz innymi ważnymi informacjami.
- **docker-compose.vault.yml** i **docker-compose.yml**: Pliki konfiguracyjne dla Docker Compose, definiujące usługi, które mają być uruchomione w środowisku Docker.
- **mvnw** i **mvnw.cmd**: Wrapper Maven, umożliwiający uruchomienie Mavena bez konieczności jego instalacji w systemie.

- **pom.xml**: Plik konfiguracyjny projektu Maven, zawierający zależności, wtyczki i inne ustawienia build'a.
- **vault-entrypoint.sh**: Skrypt uruchamiany podczas startu kontenera Docker, związany z konfiguracją Vault.

Katalog `.mvn/wrapper`

- **maven-wrapper.properties**: Plik konfiguracyjny dla Maven Wrapper.

Katalog `src/main/java/com/example/userapi`

- **UserApiApplication.java**: Główna klasa uruchamiająca aplikację Spring Boot.
- **config**: Pakiet zawierający klasy konfiguracyjne, takie jak `ClickhouseConfig`, `DataSourceConfig`, `VaultConfig`.
- **controller**: Pakiet zawierający klasy kontrolerów, odpowiedzialne za obsługę żądań HTTP, takie jak `ItemController`, `PolicyController`, `UserController`, `VaultCredentialsController`.
- **model**: Pakiet zawierający klasy modelowe (encje), takie jak `Item`, `Policy`, `User`.
- **repository**: Pakiet zawierający interfejsy repozytoriów, takie jak `ItemRepository`, `PolicyRepository`, `UserRepository`.
- **service**: Pakiet zawierający klasy serwisów, odpowiedzialne za logikę biznesową, takie jak `ItemService`, `PolicyService`, `UserService`, `VaultService`.

Katalog `src/main/resources`

- **application.properties**: Główny plik konfiguracyjny aplikacji Spring Boot.
- **logback.xml**: Plik konfiguracyjny dla logowania za pomocą Logback.
- **templates**: Katalog zawierający szablony HTML, takie jak `vault-credentials.html`.

Katalog `src/test/java/com/example/userapi`

- **UserApiApplicationTests.java**: Klasa testowa dla głównej klasy aplikacji.
- **config**: Pakiet zawierający klasy testowe dla konfiguracji, takie jak `ClickhouseConfigTest`, `DataSourceConfigTest`.
- **controller**: Pakiet zawierający klasy testowe dla kontrolerów, takie jak `ItemControllerTest`, `PolicyControllerTest`, `UserControllerTest`, `VaultCredentialsControllerTest`.

- **model:** Pakiet zawierający klasy testowe dla modeli, takie jak ItemTest, PolicyTest, UserTest.
- **service:** Pakiet zawierający klasy testowe dla serwisów, takie jak PolicyServiceTest, UserServiceTest, VaultServiceTest.

Opis funkcji głównych komponentów

- **Kontrolery:** Odpowiadają za przyjmowanie żądań HTTP, przetwarzanie danych wejściowych, wywoływanie odpowiednich metod serwisowych oraz zwracanie odpowiedzi.
- **Serwisy:** Zawierają logikę biznesową aplikacji, przetwarzają dane, korzystają z repozytoriów do komunikacji z bazą danych.
- **Repozytoria:** Interfejsy zapewniające operacje CRUD na bazie danych.
- **Modele:** Klasy reprezentujące strukturę danych w aplikacji i mapujące te dane na tabele w bazie danych.

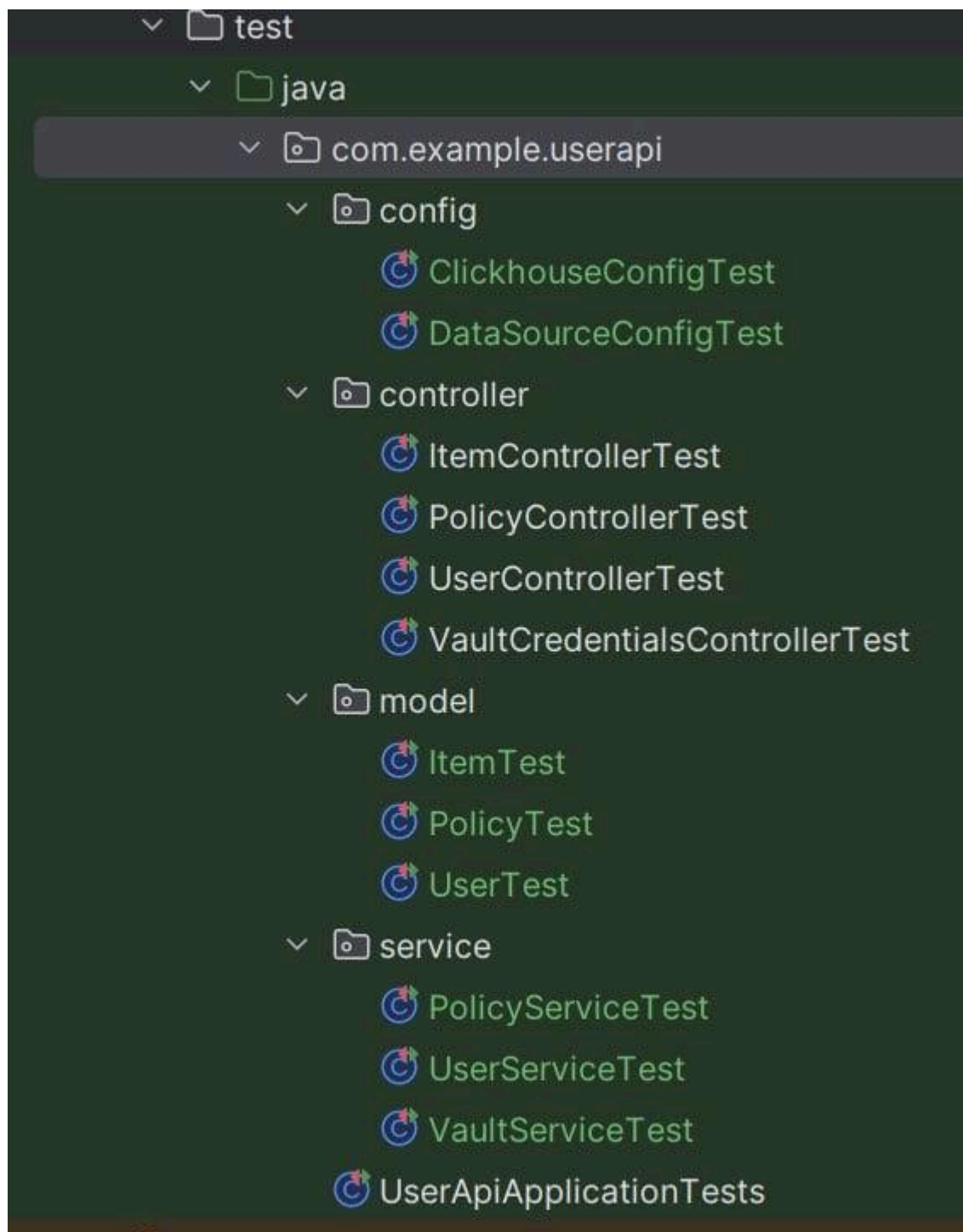
Konwencje kodowania

- Używaj CamelCase do nazewnictwa klas i metod.
- Zawsze dodawaj komentarze do kodu, aby był czytelny dla innych deweloperów.
- Stosuj zasady SOLID.

Testowanie

- **Narzędzia używane do testowania:** Mockito-core unit test
- **Pliki testowe**

Każdy plik testu odpowiada plikowi i strukturze w projekcie.



Historia Repozytorium Github

Activity



All branches ▾



All activity ▾



All users ▾



All time ▾

Add DataSourceConfigTest



IsaburoBowman pushed 1 commit to `main` • 206d8df...fb7d512 • 2 hours ago

Create ClickhouseConfigTest.java



IsaburoBowman pushed 1 commit to `main` • fddce51...206d8df • 2 hours ago

Delete src/test/java/com/example/userapi/ClickhouseConfigTest.java



IsaburoBowman pushed 1 commit to `main` • e7b5c82...fddce51 • 2 hours ago

Create ClickhouseConfigTest.java



IsaburoBowman pushed 1 commit to `main` • ce8f1c0...e7b5c82 • 2 hours ago

Fixes



IvanLyadov pushed 1 commit to `main_policy` • bb35c65...149a6cb • 4 hours ago

Merge pull request #7 from IvanLyadov/main_policy [Pull request merge](#)



IvanLyadov pushed 4 commits to `main` • ae98c01...ce8f1c0 • 17 hours ago

Unit tests



IvanLyadov pushed 2 commits to `main_policy` • 49684b9...bb35c65 • 17 hours ago

fixing user sector permission



IvanLyadov pushed 1 commit to `main_policy` • ecc8fd7...49684b9 • 23 hours ago

Merge pull request #6 from IvanLyadov/main_policy [Pull request merge](#)



IvanLyadov pushed 2 commits to `main` • 6a08751...ae98c01 • yesterday

Policy development



IvanLyadov created `main_policy` • ecc8fd7 • yesterday

Item development



IsaburoBowman pushed 1 commit to `main` • 2e390ef...6a08751 • yesterday

Item development



IsaburoBowman pushed 1 commit to `main` • 44ff6a5...2e390ef • yesterday

Merge pull request #5 from IvanLyadov/main_item_configuration [Pull request merge](#)



IvanLyadov pushed 2 commits to `main` • 009bf9d...44ff6a5 • yesterday

Item object implementation



IvanLyadov created `main_item_configuration` • 2833e34 • yesterday

Add files via upload



IsaburoBowman pushed 1 commit to `main` • ee83846...009bf9d • yesterday


Add files via upload




IsaburoBowman pushed 1 commit to `main` • 4c84718...ee83846 • yesterday

Merge pull request #4 from IvanLyadov/main_configuration [Pull request merge](#)

Fixing docker, Vault and DB configuration

 IvanLyadov created [main_configuration](#) • 85ca93d • 4 days ago


Merge pull request #3 from IvanLyadov/docker [Pull request merge](#)

 IvanLyadov pushed 2 commits to [main](#) • 65274e1...4c3fce0 • 10 days ago


Configuring docker

 IvanLyadov pushed 1 commit to [docker](#) • be27a64...1b324b0 • 10 days ago


Merge pull request #2 from IvanLyadov/docker [Pull request merge](#)

 IvanLyadov pushed 2 commits to [main](#) • 40fb9f1...65274e1 • 11 days ago

Adding docker config

 IvanLyadov created [docker](#) • be27a64 • 11 days ago

Add files via upload

 IsaburoBowman pushed 1 commit to [main](#) • 515f9a7...40fb9f1 • 11 days ago


Merge pull request #1 from IvanLyadov/clickhouse_config [Pull request merge](#)

 IvanLyadov pushed 3 commits to [main](#) • 0951d07...515f9a7 • 11 days ago


Merge branch 'main' into clickhouse_config

 IvanLyadov pushed 3 commits to [clickhouse_config](#) • 7ab6851...b4257a3 • 11 days ago


Adding Clickhouse config

 IvanLyadov created [clickhouse_config](#) • 7ab6851 • 11 days ago

Create VaultIntegration.java

 IsaburoBowman pushed 1 commit to [main](#) • d691413...0951d07 • 13 days ago


Update pom.xml

 IsaburoBowman pushed 1 commit to [main](#) • 1916650...d691413 • 13 days ago

Initial commit

 IvanLyadov pushed 1 commit to [main](#) • 3c1e7cb...1916650 • 14 days ago

Initial commit

 IvanLyadov created [main](#) • 3c1e7cb • 14 days ago

[Share feedback about this page](#)