

CUSTOMER RELATIONSHIP MANAGEMENT – system zarządzania relacjami z klientami

Serhii Humennyi

Nr albumu: 88455

Informatyka 3 rok

Prowadzący : Stanisław Ambroszkiewicz

Streszczenie

Projekt CRM ma na celu stworzenie nowoczesnego systemu zarządzania relacjami z klientami, opartego na technologii Spring Boot, z wykorzystaniem baz danych H2 oraz MySQL, a także narzędzia do wersjonowania schematu bazy danych – Liquibase. W środowisku deweloperskim zastosowano lekką bazę H2 w trybie plikowym, umożliwiającą szybkie uruchamianie aplikacji oraz zachowanie danych pomiędzy restartami. Środowisko produkcyjne korzysta z w pełni skonfigurowanej bazy MySQL 8, zapewniającej stabilne i bezpieczne przechowywanie danych.

W projekcie wdrożono jednolitą strukturę migracji Liquibase, gwarantującą pełną kontrolę nad wersjonowaniem schematu oraz jego spójność we wszystkich środowiskach. Przygotowano również pliki seed data dla środowiska dev, co ułatwia szybkie testowanie funkcjonalności.

Równolegle z konfiguracją warstwy persystencji zaimplementowano system kontroli wersji Git oraz integrację z GitHub, wraz z przyjętą konwencją commitów, modelem pracy na gałęziach oraz podstawowym workflow Pull Request. Dodatkowo przygotowano fundament pod automatyzację w CI/CD, w tym mechanizmy odczytu wersji aplikacji z tagów.

Efektem wykonanych prac jest solidna i dobrze udokumentowana podstawa techniczna pod dalszy rozwój aplikacji CRM, obejmująca konfigurację środowisk, migracji, kontroli wersji oraz procesów deweloperskich. Umożliwia to sprawną, uporządkowaną i skalowalną rozbudowę systemu w kolejnych etapach projektu.

1. WPROWADZENIE

Projekt CRM korzysta z technologii Spring Boot, k8, Docker, Liquibase, Git oraz GitHub. Środowisko dev wykorzystuje bazę H2 w trybie file, zaś produkcyjne środowisko opiera się na MySQL 8. W projekcie zaimplementowano wersjonowanie schematu bazy danych, konfigurację środowisk, integrację gita oraz system pracy z gałęziami.

2. KONFIGURACJA BAZY DANYCH H2 (DEV)

Cel: zapewnienie lekkiego środowiska deweloperskiego bez instalacji zewnętrznych baz danych oraz zachowanie danych po restarcie.

Plik application-dev.properties:

```
spring.datasource.url=jdbc:h2:file:./data/crmdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
```

```
spring.datasource.username=sa spring.datasource.password= spring.datasource.driver-class-name=org.h2.Driver spring.liquibase.contexts=dev spring.jpa.show-sql=true spring.jpa.properties.hibernate.format_sql=true
```

Powod zastosowania DB_CLOSE_DELAY=-1: zapobiega utracie danych.

3. KONFIGURACJA BAZY DANYCH MYSQL (PROD)

Baza docelowa przechowuje dane w wersji produkcyjnej.

Instrukcje tworzenia bazy:

```
CREATE DATABASE crmdb CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

```
CREATE USER 'crmuser'@'localhost' IDENTIFIED BY 'secret';
```

```
GRANT ALL PRIVILEGES ON crmdb.* TO 'crmuser'@'localhost';
```

```
FLUSH PRIVILEGES;
```

Konfiguracja application-prod.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/crmdb?useSSL=false&serverTimezone=UTC
spring.datasource.username=crmuser spring.datasource.password=secret
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.jpa.hibernate.ddl-auto=validate
```

```
spring.liquibase.contexts=prod
```

4. LIQUIBASE

Celem jest utrzymanie jednej historii migracji.

Struktura katalogow:

```
src/main/resources/db/changelog/customers
```

Pierwszy changeset:

- utworzenie tabeli customers z polami:

id (BIGINT, PK, auto increment)

name VARCHAR(255) email VARCHAR(255) unikalny phone VARCHAR(255) status VARCHAR(50)

Rollback usuwa tabele customers.

Format YAML zapewnia wieksza czytelnosc.

5. SEED DATA (DEV)

Plik data.sql:

```
INSERT INTO customers (name, email, phone, status) VALUES
('Alice Johnson', 'alice.johnson@example.com', '+15551234567', 'LEAD'),
('Bob Smith', 'bob.smith@example.com', '+15557654321', 'ACTIVE'),
('Carol Williams', 'carol.williams@example.com', '+15559876543', 'INACTIVE'),
('David Brown', 'david.brown@example.com', '+15553456789', 'CLOSED'),
('Eve Davis', 'eve.davis@example.com', '+15552345678', 'LEAD');
```

6. INTEGRACJA GIT

Projekt zostal przekształcony w repozytorium Git poprzez IntelliJ IDEA.

Dodano plik .gitignore ignorujacy:

- katalog .idea/
 - pliki .iml
 - katalog target/
 - logi
-

7. KONWENCJA COMMITOW (CONVENTIONAL COMMITS)

feat - nowa funkcjonalnosc

fix - poprawa bledu

docs - zmiany dokumentacji

style - zmiany formatowania bez logiki

refactor - poprawa jakosci kodu

test - dodanie testow

ci - zmiany CI/CD

build - zmiany w Mavenie

chore - czynnosci techniczne

perf - optymalizacja

8. POLACZENIE Z GITHUB

Utworzono repozytorium GitHub i polaczono je poleceniem: `git remote add origin`

Nastepnie wyslano pierwsze zmiany: `git push -u origin main`

9. STRUKTURA GALEZI W PROJEKCIE

main - stabilna galaz produkcyjna

dev - galaz integracyjna

feature/* - prace nad funkcjonalnosciami

hotfix/* - szybkie poprawki produkcyjne

10. WERSJONOWANIE (SEMVER)

MAJOR - zmiany niekompatybilne

MINOR - nowe funkcje

PATCH - poprawki

Przyklad: `git tag -a v1.0.0 -m "Release 1.0.0" git push origin v1.0.0`

11. SZABLON PULL REQUEST

Dodano plik .github/PULL_REQUEST_TEMPLATE.md z sekcjami:

WHAT?

WHY?

HOW TO TEST?

12. CI/CD

Przygotowano pipeline odczytujący wersje z taga:

```
${{ github.ref_name }} oraz przekazujący ją do Mavena: mvn clean package -Dapp.version=${{ env.APP_VERSION }}
```

13. PODSUMOWANIE

- Konfiguracja środowiska dev z bazą H2.
 - Usunięcie konfliktów AUTO_SERVER.
 - Dodanie Liquibase oraz pierwszego changesetu.
 - Utworzenie seed data dla dev.
 - Wprowadzenie konfiguracji produkcyjnej z MySQL 8.
 - Przygotowanie SQL do tworzenia usera i bazy.
 - Konfiguracja application-prod.properties.
 - Integracja Git + GitHub.
 - Ustalenie konwencji commitów oraz workflow gałęzi.
 - Przygotowanie szablonu PR.
-