Spring Data JPA

Głównym celem tego projektu jest zredukowanie dużej części standardowego kodu, który zwykle musimy napisać w przypadku podstawowych operacji dostępu do danych.

Spring Data JPA

Konieczne jest dodanie dependecji do projektu starter-data-jpa oraz bazy danych.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Testy

Napiszmy 2 testy dla istniejącego rozwiązania przechowującego dane w mapie. Po zmianie na bazę danych testy te dalej powinny przechodzić.

```java
@Test
public void whenSavingNewProject_thenSuccess() {
    Project newProject = new Project( id: 1L, randomAlphabetic( count: 6), LocalDate.now());
    assertNotNull(projectRepository.save(newProject));
}

@Test
public void givenProject_whenFindById_thenSuccess() {
    Project newProject = new Project( id: 1L, randomAlphabetic( count: 6), LocalDate.now());
    projectRepository.save(newProject);
    Optional<Project> retrievedProject = projectRepository.findById(newProject.getId());
    assertThat(retrievedProject.get().getId(), is(equalTo(newProject.getId())));
}
```

CrudRepository

Dzięki Spring Data możemy tworzyć repozytoria, rozszerzając domyślne interfejsy repozytorium takie jak np. CrudRepository.

CrudRepository zawiera proste metody dostępu do danych, takie jak save, saveAll, findAll, findById, itp.

Nie musimy ich implementować ponieważ framework zrobi to za nas automatycznie.

@Entity

Aby Spring Data mógł zapisywać i odczytywać dane muszą one zostać zamodelowane w postaci encji.

W tym celu w klasie Project dodamy adnotację @Entity, @Id oraz @GeneratedValue.

```
@Entity
public class Project {

    @Id
    @GeneratedValue
    private Long id;
```

DataAccessException

Spring tłumaczy wyjątki specyficzne dla technologii, takie jak SQLException, na własną hierarchię klas wyjątków np. DataAccessException.

Ta hierarchia wyjątków ma na celu pomóc deweloperom znaleźć i obsłużyć błąd bez znajomości szczegółów konkretnego interfejsu API dostępu do danych.

DataAccessException

NonTransientDataAccessException — ponowna próba operacji zakończy się niepowodzeniem, chyba że przyczyna wyjątku zostanie usunięta.

TransientDataAccessException — ponowna próba może się powieść bez żadnych działań ze strony aplikacji.

RecoverableDataAccessException — ponowna próba może się powieść, jeśli aplikacja wykona pewne kroki.

ScriptException - wyjątek związany z przetwarzaniem skryptów SQL.

Schemat bazy danych i inicjalizacja danych

Czasami chcemy mieć pełną kontrolę nad procesem tworzenia i inicjalizacji bazy danych. Wówczas możemy skorzystać z plików data.sql i schema.sql. Wystarczy je umieścić w ścieżce src/main/resources.

Schemat bazy danych i inicjalizacja danych

```
//schema.sql
CREATE TABLE project (
  id   INTEGER      NOT NULL AUTO_INCREMENT,
  name VARCHAR(128) NOT NULL,
  date_created DATE NOT NULL,
  PRIMARY KEY (id)
);
```

Ważne jest, aby wyłączyć automatyczne tworzenie schematu podczas korzystania z pliku schema.sql, aby uniknąć konfliktów. Możemy to zrobić, ustawiając właściwość:

spring.jpa.hibernate.ddl-auto=none

Schemat bazy danych i inicjalizacja danych

//data.sql

```
INSERT INTO project (name, date_created)
VALUES ('Spring Boot', '2021-08-06');

INSERT INTO project (name, date_created)
VALUES ('Spring Security', '2021-04-25');
```

Rozszerzanie domyślnego repozytorium

```java
public interface ProjectRepository extends CrudRepository<Project, Long> {
    Optional<Project> findByName(String name);
}
```

Nie musimy pisać implementacji!

Test

```java
@Test
public void givenProjectCreated_whenFindByName_thenSuccess() {
    Project newProject = new Project( id: 1L, randomAlphabetic( count: 6), LocalDate.now());
    projectRepository.save(newProject);

    Optional<Project> retrievedProject = projectRepository.findByName(newProject.getName());
    assertEquals(retrievedProject.get().getId(), newProject.getId());
}
```

Bardziej złożone zapytanie

```
List<Project> findByStartDateBetween(LocalDate start, LocalDate end);
```

```
@Test
public void givenProjectCreated_whenFindByDateCreatedBetween_thenSuccess() {
    Project oldProject = new Project(randomAlphabetic( count: 6), LocalDate.now().minusYears(1));
    projectRepository.save(oldProject);

    Project newProject = new Project(randomAlphabetic( count: 6), LocalDate.now());
    projectRepository.save(newProject);

    Project newProject2 = new Project(randomAlphabetic( count: 6), LocalDate.now());
    projectRepository.save(newProject2);

    List<Project> retreivedProjects = projectRepository.findByStartDateBetween(LocalDate.now().minusDays(1), LocalDate.now().plusDays(1));
    assertThat(retreivedProjects, hasItems(newProject, newProject2));
}
```

Nowa encja Task

```java
@Entity
public class Task {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    private String description;

    private LocalDate dateCreated;

    private LocalDate dueDate;

    private TaskStatus status;
```

Relacja pomiędzy Project i Task

```java
@OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinColumn(name = "project_id")
private Set<Task> tasks;
```

Zapytanie JPQL

```
public interface TaskRepository extends CrudRepository<Task, Long> {
    @Query("select t from Task t where t.name like %?1%")
    List<Task> findByNameMatches(String name);
}
```

Test

```java
@Test
public void givenProjectCreated_thenFindByTaskNameMatchesSuccess() {
    Task task1 = new Task( name: "Low Priority Task",  description: "Low Priority Task", LocalDate.now(), LocalDate.now());
    Task task2 = new Task( name: "Low Priority Task",  description: "Low Priority Task", LocalDate.now(), LocalDate.now());
    Task task3 = new Task( name: "High Priority Task",  description: "High Priority Task", LocalDate.now(), LocalDate.now());
    Task task4 = new Task( name: "High Priority Task",  description: "High Priority Task", LocalDate.now(), LocalDate.now());

    taskRepository.save(task1);
    taskRepository.save(task2);
    taskRepository.save(task3);
    taskRepository.save(task4);

    List<Task> retreivedTasks = taskRepository.findByNameMatches("High");
    assertThat(retreivedTasks, contains(task3, task4));
}
```

Stronicowanie danych

```
public interface ProjectRepository extends PagingAndSortingRepository<Project, Long> {
```

Stronicowanie danych

Ładowanie przykładowych danych

```
@Component
public class TestDataLoader implements ApplicationContextAware {

    @Autowired
    private ProjectRepository projectRepository;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        projectRepository.save(new Project(randomAlphabetic( count: 6), LocalDate.now()));
        projectRepository.save(new Project(randomAlphabetic( count: 6), LocalDate.now()));
        projectRepository.save(new Project(randomAlphabetic( count: 6), LocalDate.now()));
        projectRepository.save(new Project(randomAlphabetic( count: 6), LocalDate.now()));
    }
}
```

Stronicowanie danych

Test poprawności pobrania strony danych

```
@Test
public void givenDataCreated_whenFindAllPaginated_thenSuccess() {
    Page<Project> retrievedProjects = projectRepository.findAll(PageRequest.of( page: 0,  size: 2));
    assertThat(retrievedProjects.getContent(), hasSize(2));
}
```

Sortowanie danych

```java
@Test
public void givenDataCreated_whenFindAllSort_thenSuccess() {
    List<Project> retrievedProjects = (List<Project>) projectRepository.findAll(Sort.by(Sort.Order.asc("name")));

    List<Project> sortedProjects = retrievedProjects.stream().collect(Collectors.toList());
    sortedProjects.sort(Comparator.comparing(Project::getName));

    assertEquals(sortedProjects, retrievedProjects);
}
```

Stronicowanie i sortowanie danych

```
public void givenDataCreated_whenFindAllPaginatedAndSort_thenSuccess() {
    Page<Project> retrievedProjects = projectRepository.findAll(PageRequest.of( page: 0,  size: 2, Sort.by(Sort.Order.asc("name"))));
    assertThat(retrievedProjects.getContent(), hasSize(2));
}
```

Transakcyjność

Dodajmy logowanie informacji o wykonywanych transakcjach:

logging.level.org.hibernate.engine.transaction.internal=DEBUG

Transakcyjność

Metoda, która zapisuje projekty z zadaniami

```
@Transactional
@Override
public void createProjectWithTasks() throws TaskNotSavedException {

    LOG.info("Start create project with tasks method in service");

    Project project = new Project( name: "Project 1", LocalDate.now());

    Project newProject = save(project);

    Task task1 = new Task( name: "Task 1",  description: "Project 1 Task 1", LocalDate.now(), LocalDate.now()
        .plusDays(7));

    taskService.save(task1);

    Set<Task> tasks = new HashSet<>();
    tasks.add(task1);

    newProject.setTasks(tasks);

    save(newProject);
}
```

Transakcyjność

Uruchomienie metody

```
@PostConstruct
public void postConstruct() {

    try {
        LOG.info("Create project with tasks");
        projectService.createProjectWithTasks();
    } catch (Exception e) {
        LOG.error("Error occurred in creating project with tasks", e);
    }

    LOG.info("Fetching all Projects");
    projectService.findAllProjects()
        .forEach(p -> LOG.info(p.toString()));

    LOG.info("Fetching all tasks");
    taskService.findAll()
        .forEach(t -> LOG.info(t.toString()));
}
```

Transakcyjność

Debug

```
main] pl.fis.spring1.Spring1Application       : Fetching all Projects
main] o.h.e.t.internal.TransactionImpl        : On TransactionImpl creation, JpaCompliance#isJpaTransactionComplianceEnabled == false
main] o.h.e.t.internal.TransactionImpl        : begin
main] o.h.e.t.internal.TransactionImpl        : committing
main] pl.fis.spring1.Spring1Application       : Project{id=1, name='Project 1', startDate=2021-08-13, tasks=[pl.fis.spring1.entity.Task@e8c9522]}
main] pl.fis.spring1.Spring1Application       : Fetching all tasks
main] o.h.e.t.internal.TransactionImpl        : On TransactionImpl creation, JpaCompliance#isJpaTransactionComplianceEnabled == false
main] o.h.e.t.internal.TransactionImpl        : begin
main] o.h.e.t.internal.TransactionImpl        : committing
main] pl.fis.spring1.Spring1Application       : pl.fis.spring1.entity.Task@e8c9522
```

Transakcyjność

Co jeśli wykonanie metody zakończy się wyjątkiem?

```java
@Override
public Task save(Task task) throws TaskNotSavedException {
    throw new TaskNotSavedException("Unable to save task");
}
```

```java
@Transactional(rollbackOn = TaskNotSavedException.class)
@Override
public void createProjectWithTasks() throws TaskNotSavedException {
```

```
main] pl.fis.spring1.Spring1Application        : Create project with tasks
main] o.h.e.t.internal.TransactionImpl         : On TransactionImpl creation, JpaCompliance#isJpaTransactionComplianceEnabled == false
main] o.h.e.t.internal.TransactionImpl         : begin
main] p.f.spring1.service.ProjectServiceImpl1  : Start create project with tasks method in service
main] o.h.e.t.internal.TransactionImpl         : rolling back
main] pl.fis.spring1.Spring1Application        : Error occurred in creating project with tasks
```