Programowanie komponentowe Spring

Ćw6 - ORM

Wstęp

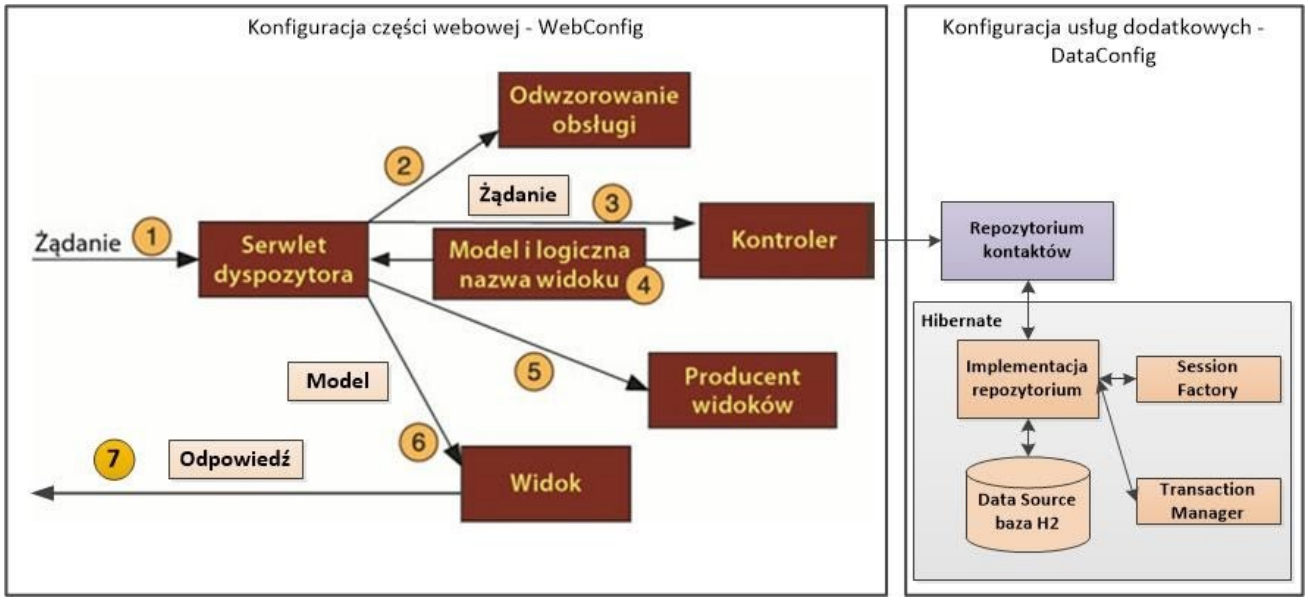
Laboratorium nr 6 składa się z zadań ułatwiających nabycie praktycznych umiejętności w zakresie utrwalania danych przy pomocy następujących technik mapowania obiektowo-relacyjnego (ORM):

- utrwalanie przy pomocy Hibernate,
- utrwalanie przy pomocy Java Persistence API - Hibernate,
- utrwalanie przy pomocy Java Persistence API - Hibernate i automatycznych repozytoriów SpringData.

Do zadań wykorzystano aplikację Contacts7 zbudowaną na poprzednich zajęciach, pozbawioną implementacji przycisku Usuń, którą trzeba uzupełnić.
Zadania polegają na budowaniu warstwy utrwalania danych wykorzystującej wbudowaną bazę H2 umieszczoną w pamięci RAM oraz różnych technologii ORM.

Zadanie 8

Celem zadania 8 jest skonfigurowanie repozytorium wykonanego w technologii Hibernate. Architektura aplikacji MVC z repozytorium Hibernate została pokazana na rysunku 6.1. Technologia Hibernate może być użyta w lekkim serwerze serwletów, więc można jej używać w aplikacjach działających na serwerze Tomcat. Jest to duża zaleta technologii Hibernate. Aby uruchomić aplikację korzystającą z repozytorium Hibernate należy skonfigurować Hibernate i użyć adnotacji do oznaczenia klasy utrwalanych obiektów i jego właściwości, co pozwoli na automatyczne mapowanie ORM. W konsekwencji można użyć sesji Hibernate do realizacji metod repozytorium.



6.1 Aplikacja webowa wykorzystująca repozytorium kontaktów zaimplementowane w technologii Hibernate

Konfiguracja Hibernate

W celu konfiguracji Hibernate należy utworzyć klasę konfiguracyjną **DataConfig** w pakiecie **contacts.config**, która zostanie automatyczne odszukana przez klasę konfiguracyjną **RootConfig**. Klasa **DataConfig** powina być oznaczona adnotacjami @**Configuration**, @**EnableTransactionManagement** i @**ComponentScan.**
Zawiera definicję trzech komponentów:

- komponent - źródło danych jest identyczne, jak w dostępie przez JDBC

```
@Bean
public DataSource dataSource() {
 return new EmbeddedDatabaseBuilder()
      .setType(EmbeddedDatabaseType.H2)
      .addScript("schema.sql") // import schematu bazy danych
      .addScript("data.sql")  // import danych testowych
      .build();
}
```

- komponent - fabryka sesji Hibernate

```
@Bean
public SessionFactory sessionFactoryBean() throws IOException {
LocalSessionFactoryBean lsfb = new LocalSessionFactoryBean();
lsfb.setDataSource(dataSource());
      // nazwa pakietu do wyszukania klas utrwalanych obiektów
lsfb.setPackagesToScan("contacts");
Properties props = new Properties();
props.setProperty("dialect", "org.hibernate.dialect.H2Dialect");
lsfb.setHibernateProperties(props);
lsfb.afterPropertiesSet();
SessionFactory object = lsfb.getObject();
return object;
}
```

- komponent - menedżer transakcji Hibernate

```
@Bean
public PlatformTransactionManager annotationDrivenTransactionManager() throws
IOException {
HibernateTransactionManager transactionManager = new HibernateTransactionManager();
transactionManager.setSessionFactory(sessionFactoryBean());
return transactionManager;
}
```

Oznaczenie klas utrwalanych obiektów - encji

Klasa utwalanego obiektu - encji powinna być oznaczona za pomocą adnotacji @**Entity**. Klasa **Contacts** posiada już adnotacje pozwalające na walidację wartości wprowadzonych w formularzu

aplikacji. Obydwa rodzaje adnotacji właściwości mogą być naraz użyte i nie prowadzi to do konfliktów. Właściwości klasy powinny być oznaczone adnotacją **@Column(name="nazwa_kolumny")**, co pozwala odnaleźć wartości właściwości w kolumnach mapowanej tabeli. Identyfikator powinien być oznaczony adnotacjami **@Id** i **@GeneratedValue(strategy = GenerationType.IDENTITY)**.

Implementacja repozytorium kontaktów

Klasa implementująca repozytorium kontaktów powinna być oznaczona adnotacjami **@Repository** i **@Transactional.** Implementacja metod repozytorium odbywa się przy pomocy sesji Hibernate, która jest tworzona przy pomocy fabryki sesji, komponentu utworzonego w klasie konfiguracyjnej **DataConfig,** wstrzykniętego do repozytorium przy pomocy adnotacji **@Inject**. Fabryka sesji jest wykorzystywana do pobrania referencji do bieżącej sesji Hibernate.

```
@Repository
@Transactional
public class HibernateContactRepository implements ContactRepository {
private SessionFactory sessionFactory;
@Inject
public HibernateContactRepository(SessionFactory sessionFactory) {
this.sessionFactory = sessionFactory;
}
private Session currentSession() {
return sessionFactory.getCurrentSession();
}
```

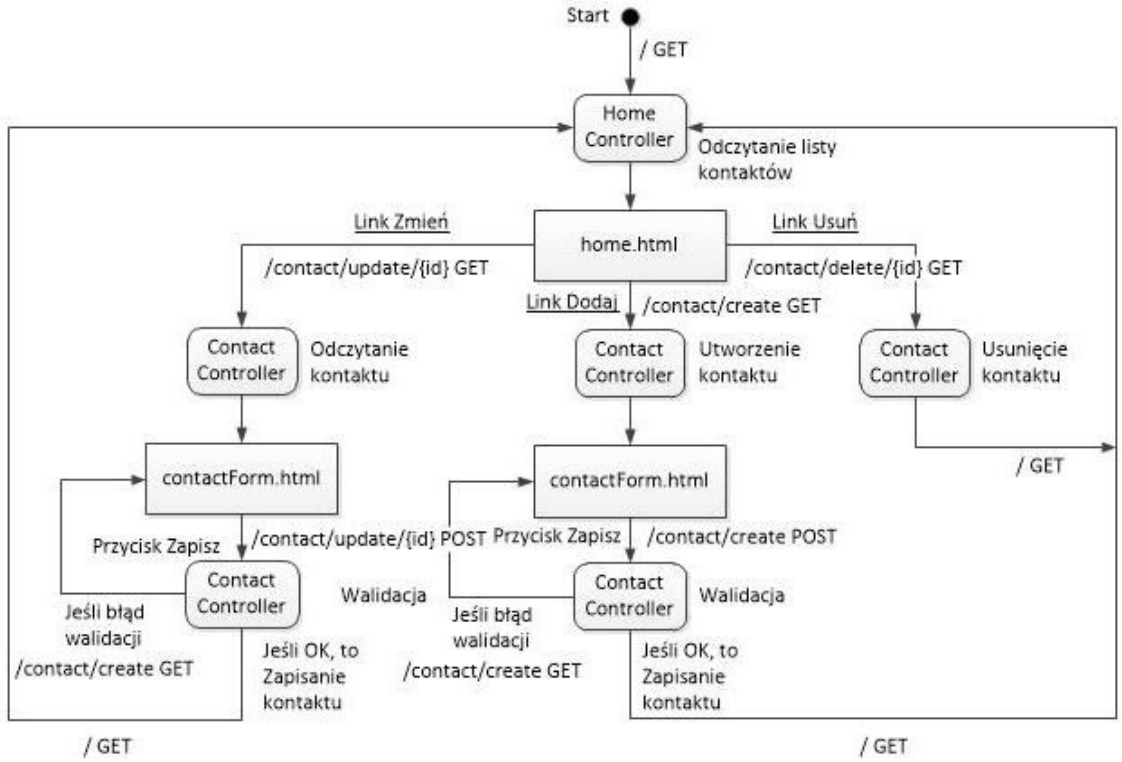
Metody wykonujące operacje na danych używają do tego referencji do bieżącej sesji:

```
@Override
public List<Contact> findContacts(long max, int count) {
return (List<Contact>) currentSession().createCriteria(Contact.class).list();
}
@Override
public Contact findOne(long id) {
return (Contact) currentSession().get(Contact.class, id);
}
@Override
public void insert(Contact contact) {
currentSession().save(contact);
}
@Override
public void update(Contact contact) {
currentSession().update(contact);
}
```

Treść zadania 8

Proszę zaimportować projekt contacts8. Jest to aplikacja Contacts7 z poprzednich zajęć używająca repozytorium JDBC. Do wykonania zadania proszę użyć STS 3.9, ponieważ tam mamy zdefiniowany serwer Apache Tomcat 9.1. Implementację repozytorium wykonaną w technologii JDBC należy usunąć, rónież niektóre elementy konfiguracji w klasie **DataConfig**. Następnie proszę skonfigurować Hibernate, oznaczyć adnotacjami klasę Contacts i utworzyć implementację interfejsu repozytorium dla Hibernate.

Aplikację proszę uruchamiać na serwerze Tomcat i uzupełnić o obsługę funkcji Usuń zgodnie z Rys. 6.2.

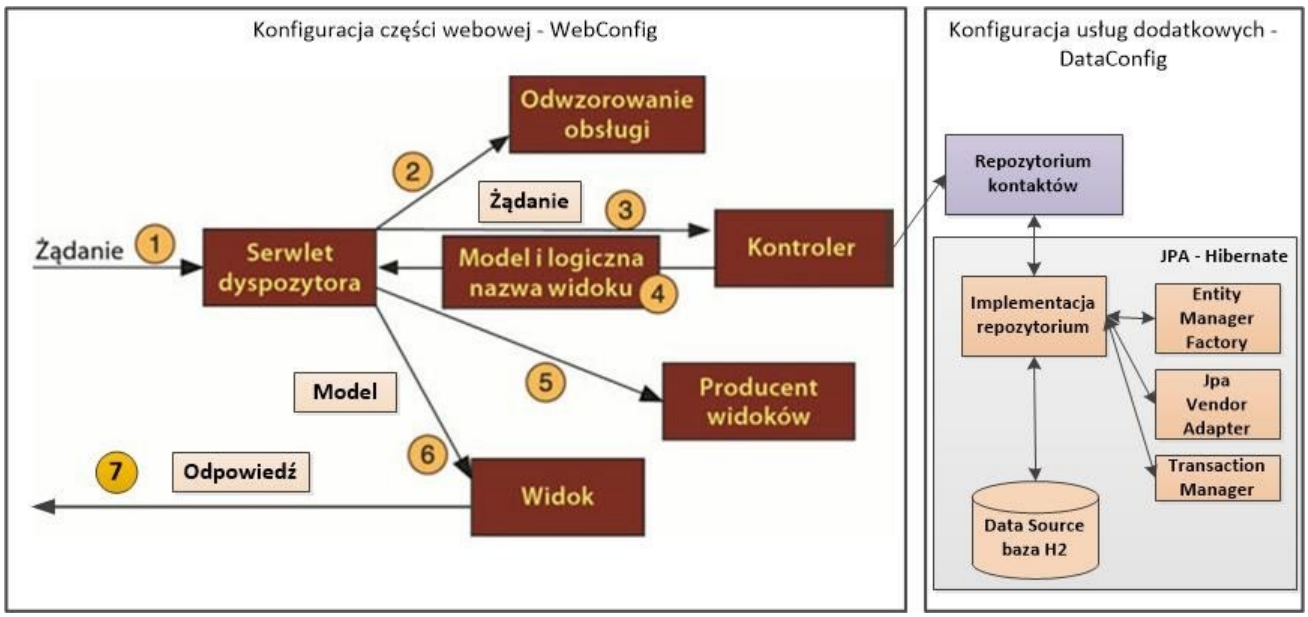


Rys 6.2 Przepływ żądań i widoków przy dodawaniu, zmianie kontaktu z walidacją danychi usuwaniu

Zadanie 9

Celem zadania 9 jest skonfigurowanie repozytorium wykonanego w technologii Java Persistence API w implementacji dostarczonej przez Hibernate.

Punktem początkowym prac może być kopia projektu z ukończonego zadania 8. Aby uruchomić aplikację z warstwą persystencji wykonaną w technologii JPA należy skonfigurować JPA w klasie DataConfig a następnie użyć obiektu EntityManager do zaimplementowania repozytorium w technologii JPA. Adnotacje klasy Contacts określające encje są w JPA identyczne, jak w Hibernate.



6.4 Aplikacja webowa wykorzystująca repozytorium kontaktów zaimplementowane w technologii Java Persistence API - Hibernate

Konfiguracja JPA

W celu konfiguracji JPA należy utworzyć klasę konfiguracyjną **DataConfig** w pakiecie **contacts.config**, która zostanie automatyczne odszukana przez klasę konfiguracyjną **RootConfig**. Klasa **DataConfig** jest oznaczona adnotacjami **@Configuration**, **@EnableTransactionManagement** i **@EnableJpaRepositories** Zawiera ona definicję czterech komponentów:

- komponent - źródło danych jest identyczne, jak w dostępie przez JDBC i Hibernate

```
@Bean
public DataSource dataSource() {
  return new EmbeddedDatabaseBuilder()
        .setType(EmbeddedDatabaseType.H2)
        .addScript("schema.sql") // import schematu bazy danych
        .addScript("data.sql")  // import danych testowych
        .build();
  }
```

- komponent określajacy dostawcę implementacji JPA - Hibernate

```
@Bean
public HibernateJpaVendorAdapter jpaVendorAdapter() {
HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
adapter.setDatabase(Database.H2);
adapter.setShowSql(false);
adapter.setGenerateDdl(true);
return adapter;
}
```

- komponent - fabryka menedżerów encji

```
@Bean
public EntityManagerFactory entityManagerFactory() {
LocalContainerEntityManagerFactoryBean emf = new
LocalContainerEntityManagerFactoryBean();
emf.setDataSource(dataSource());
emf.setJpaVendorAdapter(jpaVendorAdapter());
emf.setPersistenceUnitName("contactsPU");
emf.setPackagesToScan("contacts");
emf.afterPropertiesSet();
return emf.getObject();
}
```

- komponent - menedżer transakcji Hibernate

```
@Bean
public PlatformTransactionManager transactionManager() {
JpaTransactionManager txManager = new JpaTransactionManager();
txManager.setEntityManagerFactory(entityManagerFactory());
return txManager;
}
```

Implementacja repozytorium kontaktów

Klasa implementująca repozytorium kontaktów powinna być oznaczona adnotacjami **@Repository** i **@Transactional.** Implementacja metod repozytorium odbywa się przy pomocy menadżera encji, która jest tworzony przy pomocy fabryki sesji, komponentu utworzonego w klasie konfiguracyjnej **DataConfig,** wstrzykniętego do repozytorium przy pomocy adnotacji **@PersistenceContext**.

```
@Repository
@Transactional
public class JPAContactRepository implements ContactRepository {
@PersistenceContext
private EntityManager em;
```

Metody wykonujące operacje na danych używają do tego menadżera encji:

```
@Override
public List<Contact> findContacts(long max, int count) {
return (List<Contact>) em.createQuery("select c from Contact c where id > ? ORDER by id
asc").setParameter(1, max).getResultList();
}
@Override
public Contact findOne(long id) {
return em.find(Contact.class, id);
}
```

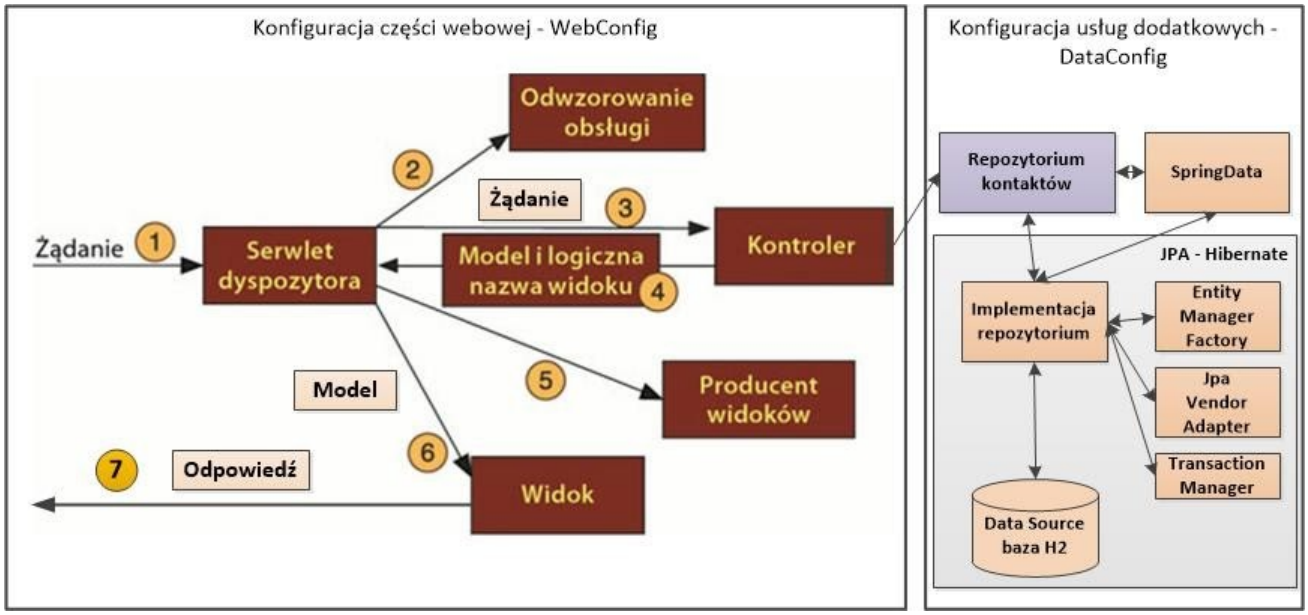
```
@Override
public void insert(Contact contact) {
em.persist(contact);
}
@Override
public void update(Contact contact) {
em.merge(contact);
}
}
```

Treść zadania 9

Proszę skopiować projekt **contacts8**. Następnie proszę skonfigurować JPA w klasie DataConfig aby potem użyć obiektu EntityManager do zaimplementowania repozytorium w technologii JPA.
Metody repozytorium powinny być wykorzystane w kontrolerach aplikacji.

Zadanie 10

Celem zadania 10 jest skonfigurowanie automatycznego repozytorium wykonanego w technologii Java Persistence API przy pomocy SpringData. SpringData automatycznie tworzy implementację repozytorium na podstawie dostarczonego interfejsu. SpringData zapewnia oprócz standardowych 18 operacji na danych także tworzenie implementacji metod interfejsu na podstawie ich nazw oraz możliwość ręcznego dodawania metod do repozytorium.
Punktem początkowym prac może być kopia projektu z ukończonego zadania 9. Aby uruchomić aplikację z warstwą persystencji wykonaną w technologii SpringData JPA należy skonfigurować SpringData i JPA w klasie DataConfig a następnie zdefiniować interrfejsy do automatycznego zaimplementowania repozytorium.



6.5 Aplikacja webowa wykorzystująca repozytorium kontaktów zaimplementowane w technologii SpringData i Java Persistence API - Hibernate

Konfiguracja SpringData

Konfiguracja SpringData jest taka sama, jak konfiguracja JPA Hibernate z zadania 9. SpringData dodatkowo potrzebuje określenia, które pakiety powinny być przeszukane w celu odnalezienia interfejsów repozytoriów do utworzenia automatycznej implementacji. W naszym wypadku dodatkowa adnotacja **DataConfig** powinna wyglądać następująco:
@EnableJpaRepositories(basePackages="contacts.data")

Repozytorium kontaktów

Repozytorium kontaktów wymaga utworzenia interfejsu rozszerzającego standardowy interfejs repozytorium SpringData. W aplikacji Contacts od pierwszej wersji opartej na technologii JDBC istnieje metoda findContacts, która posiada parametry umożliwiające stronicowanie wyświetlanej listy kontaktów. Tak naprawdę ta możliwość w aplikacji nigdy nie była wykorzystywana, jednak użyjemy tej niestandardowej metody jako przykładu do "ręcznego" tworzenia metod repozytorium. Dla tej metody powinien zostać utworzony dodatkowy interfejs:

```
public interface ContactsFind {
List<Contact> findContacts(long max, int count);
}
```

Interfejs automatycznego repozytorium SpringData powinien ten dodatkowy interfejs rozszerzać:

```
public interface ContactRepository extends JpaRepository<Contact, Long>, ContactsFind {
}
```

Oczywiście SpringData nie potrafi utworzyć automatycznie implementacji dodatkowego interfejsu, zatem należy ją dostarczyć w klasie o nazwie interfejsu repozytorium z przyrostkiem Impl:

```
@Repository
@Transactional
public class ContactRepositoryImpl implements ContactsFind {
@PersistenceContext
private EntityManager em;
@Override
public List<Contact> findContacts(long max, int count) {
return (List<Contact>) em.createQuery("select c from Contact c where id > ? ORDER
by id asc").setParameter(1, max).getResultList();
}
}
```

Treść zadania 10

Proszę skopiować projekt **contacts9.** Następnie proszę skonfigurować SpringData w klasie DataConfig, aby potem umieścić niezbędne interfesjy i implementacje w pakiecie przeszukiwanym przez SpringData.