Hibernate ir Spring

Spring projekte importu

Panaudodami Maven importuokime hibernate bibliotekas:

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.12</version>
</dependency>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-agroal</artifactId>
<version>5.3.5.Final</version>
<type>pom</type>
</dependency>
```

Bibliotekų importavimas

Panaudodami Maven importuokime papildomas bibliotekas (Java 9+ versijai):

```
<dependency>
<groupId>javax.xml.bind</groupId>
<artifactId>jaxb-api</artifactId>
<version>2.2.11</version>
</dependency>
<dependency>
<groupId>com.sun.xml.bind</groupId>
<artifactId>jaxb-core</artifactId>
<version>2.2.11</version>
</dependency>
<dependency>
<groupId>com.sun.xml.bind</groupId>
<artifactId>jaxb-impl</artifactId>
<version>2.2.11</version>
</dependency>
<dependency>
<groupId>javax.activation</groupId>
<artifactId>activation</artifactId>
<version>1.1.1</version>
</dependency>
```

Servleto konfigūracija

Servleto konfigūracijoje (standartinis konfigūracinis failas: servlet-context.xml)

Sukonfigūruokime:

1. Datasource, prisijungimą prie DB ir prisijungimų pool‘ą
2. Hibernate sesiją
3. Hibernate tranzakcijas
4. Tranzakcijų valdymą anotacijų pagalba

Datasource ir prisijungimų pool‘o konfigūracija

```
 <!-- Datasource ir Pool nustatymas -->
<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
          destroy-method="close">
        <property name="driverClass" value="com.mysql.cj.jdbc.Driver" />
        <property name="jdbcUrl"
value="jdbc:mysql://localhost:3306/studentai?useSSL=false&amp;serverTimezone=UTC"
/>
        <property name="user" value="gediminas" />
        <property name="password" value="labas" />

        <!- Prisijungimų Pool‘as -->
        <property name="minPoolSize" value="5" />
        <property name="maxPoolSize" value="20" />
        <property name="maxIdleTime" value="30000" />
</bean>
```

Hibernate sesijų konfigūracija

```
<!-- Hibernate sesijos nustatymai -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
<property name="dataSource" ref="myDataSource" />
<property name="packagesToScan" value="lt.poligonas.students.entity" />
<property name="hibernateProperties">
   <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
   </props>
</property>
</bean>
```

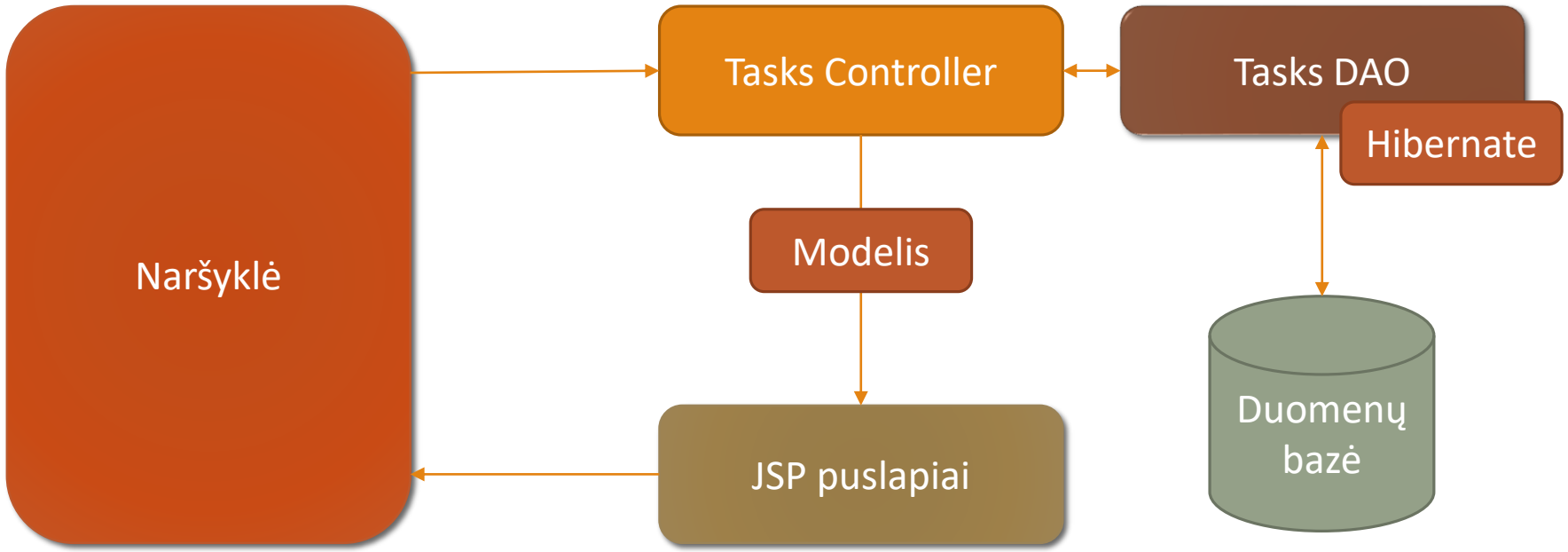
Hibernate transakcijų konfigūracija

```
 <!-- Hibernate tranzakcijų nustatymas -->

<bean id="myTransactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">

        <property name="sessionFactory" ref="sessionFactory"/>

</bean>
```

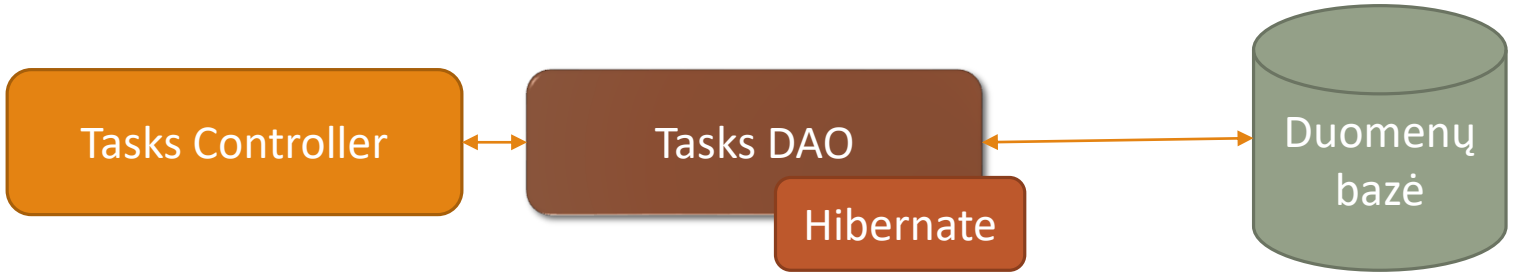
Transakcijų valdymo konfigūracija

```
<!-- Tranzakcijų valdymas anotacijų pagalba -->
<tx:annotation-driven transaction-manager="myTransactionManager" />
```

Bendra sistemos schema



Tasks DAO metodai



Metodai	
saveTask(...)	Išsaugoti įrašą
getTask(...)	Parsiųsti įrašą
getTasks()	Parsiųsti visus įrašus
updateTask(...)	Atnaujinti įrašą
deleteTask(...)	Ištrinti įrašą

Duomenų paėmimas

Atliksime šiuos žingsnius:

1. Sukursime MySQL DB, lentelę Tasks ir įterpsime keletą įrašų
2. Sukursime entity klasę – **Task.java**
3. Sukursime TaskDAO.java interfeisą
4. Sukursime TaskDAOImpl.java klasę, realizuojančią interfeisą TaskDAO
5. Sukursime kontrolerį TaskController.java
6. Sukursime JSP puslapį: tasks_list.jsp

1. sukurkime MySQL DB ir lentelę

Sukurkime DB lentelę Tasks:

Laukelio pavadinimas	Tipas	Aprašas
id	integer, autoincrement	Užduoties ID
name	varchar, 255	Užduoties pavadinimas
description	text	Užduoties aprašas
status	integer	Užduoties statusas: 0 – nejvykdyta 1 – įvykdyta

2. Sukurkime Task klasę (entity)

Sukurkime Task entity klasę.

Šią klasę turi sudaryti tokie laukai:

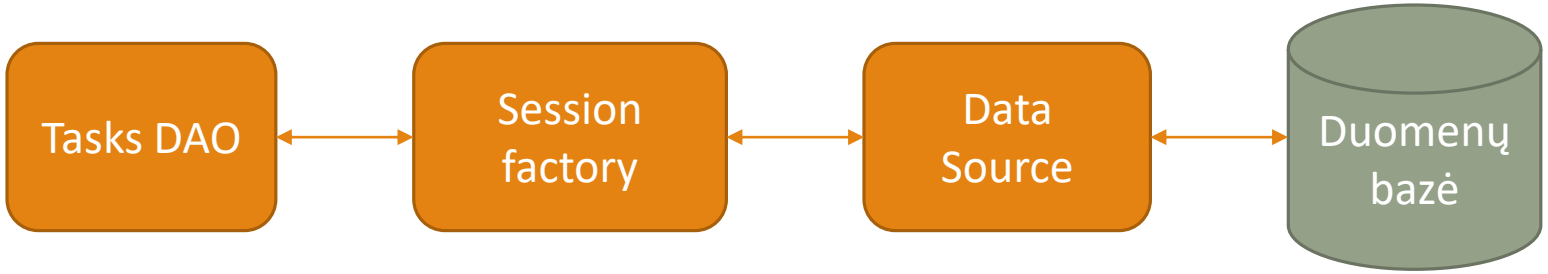
id	- auto increment
name	- string
description	- string
status	- integer

Sukurkime setterius, getterius, konstruktorius

Nepamirškite, kad entity klasė turi būti pakete kurį nurodėme sesijos konfigūracijoje: packagesToScan

Jei turėtumėme daugiau nei vieną paketą, konfigūracijoje juos nurodytume atskirtus kableliais

Tasks DAO



Mūsų kruiamai sistemai klasei paimančiai duomenis iš DB (TasksDAOImpl) bus reikalingas Session factory, tuo tarpu Sessions Factory bus reikalingas DataSource. Šiem įrankiam gauti panaudosime Spring Dependency Injection.

Session factory ir Data Source aprašėme servleto konfigūraciniame faile.

3. Tasks DAO klasės ir interfeiso sukūrimas

Sukurkime naują paketą **.dao**, ten patalpinsime DAO interfeisus ir realizacijas

Sukurkime interfeisą TasksDAO,

- Jis turės turėti metodą:

  public List<Task> getTasks();

TasksDAOImpl sesijos pridėjimas

Sukurkime klasę TasksDAOImpl, kuri realizuos prieš tai sukurtą interfeisą.

TasksDAOImpl mums bus reikalingas Session Factory, todėl norint, kad Spring atliktų Dependency Injection, mes galime panaudoti spring anotaciją ir klasėje nurodyti:

@Autowired

private SessionFactory sessionFactory;

sessionFactory pavadinimas sutampa su mūsų nurodytu vienu iš bean id.

@Transactional anotacija

Spring anotacija @Transactional nurodo jog mūsų kodo pradžioje nereikia pradėti transakcijos, o vėliau jos commitinti, Spring tai atliks už mus. Todėl toks kodas tampa nebereikalingas:

```
session.beginTransaction();
....
session.getTransaction().commit();
```

@Transactional nurodoma priešmetodą kurio pradžioje bus pradėta transakcija, o po kodu bus uždaroma pavyzdžiui:

```
@Transactional
public List<Task> getTasks(){
....
}
```

@Repository anotacija

Spring anotacija @Repository naudojama kartu su DAO objektais.

Jos pagalba:

- Automatiškai registruojamos DAO implementacijos
- Spring padės lengviau valdyti JDBC išimtis

@Repository anotacija nurodoma virš DAO klasės implementacijos:

```
@Repository
public class TasksDAOImpl implements TasksDAO {
...
}
```

Metodo getTasks() įgyvendinimas

```
public List<Task> getTasks(){
 //Sesijos paėmimas
  Session session= sessionFactory.getCurrentSession();

 //Užklausos visiem įrašams paimti generavimas
 Query<Customer> query=session.createQuery("from Task", Task.class);

 //Įrašų paėmimas
  List<Task> tasks=query.getResultList();

//Įrašų gražinimas
  return tasks;
}
```

4. Sukursime kontrolerį TaskController

Sukurkime kontrolerį TasksController.

Kontroleryje mums bus reikalingas TasksDAO objektas. Jį galime gauti pasinaudodami Spring Dependency injection kontroleryje nurodę:

@Autowired

private TasksDAO tasksDAO;

Jis bus prijungtas dėl to, kad TasksDAO interfeiso implementacijoje nurodėme @Repository anotaciją.

Objektas tasksDAO bus klasės tasksDAOImpl.

Duomenų perdavimas į JSP failą

Metodas skirtas duomenų perdavimui galėtų atrodyti taip:

```
public String(Model model){
  //Paimamos visos užduotys
  List<Task> tasks=tasksDAO.getTasks();
  //Įdedame duomenis į modelį
  model.addAttribute("tasks",tasks);
  //Atvaizduojame tasks.jsp failą
  return "tasks";
}
```

Sukurkime JSP failą atvaizdavimui

Atvaizduojant panaudokime JSTL tagus:

```
<c:forEach var="task" items="${tasks}">


</c:forEach>
```

Puslapio atvaizdavimas

Pagal nutylėjimą, užėjus į projektą kraunamas tas failas kuris yra nurodytas web.xml faile (arba turintis bazinį route).

Norėdami iš JSP failo padaryti nukreipimą, galime parašyti:

<% response.sendRedirect("tasks/list"); %>

@GetMapping ir @PostMapping

Mes jau naudojome anotaciją:

@RequestMapping – ji priima ir GET ir POST metodu siunčiamus duomenis ir atvaizduoja juos

Taip pat galime naudoti ir

@GetMapping – apdoros tik užklausas siųstas GET metodu

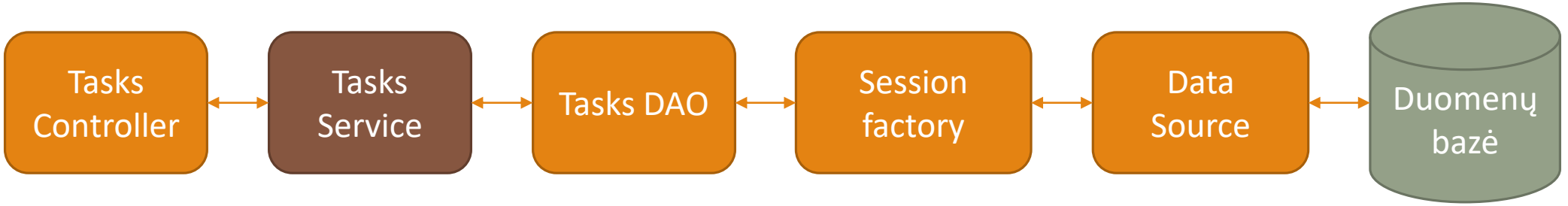
@PostMapping –apdoros tik užklausas siųstas POST metodu

@GetMapping("/processForm") atliktų tą patį ką ir

@RequestMapping(path="/processForm", method=RequestMethod.GET)

Service layer

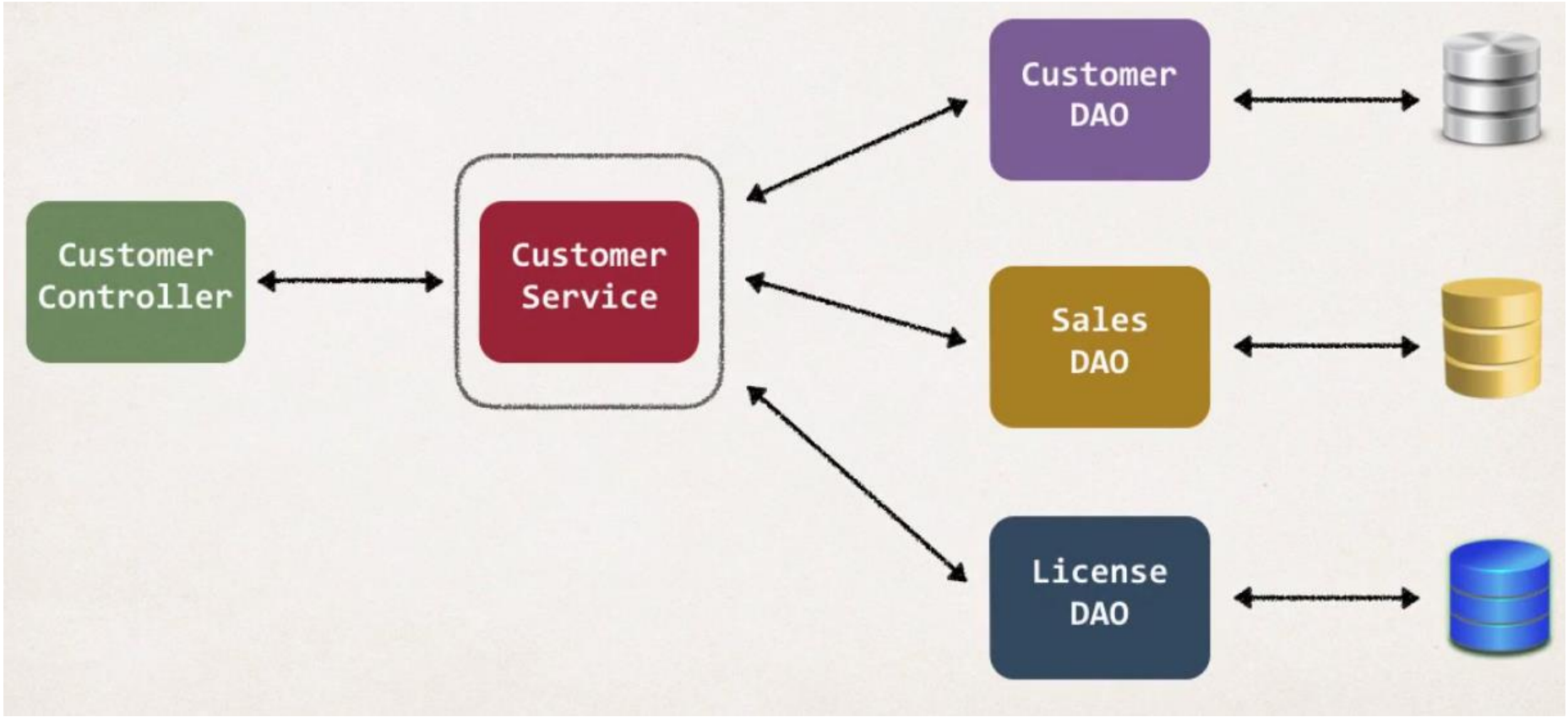
Prie mūsų kuriamos programos pridėkime papildomą sluoksnį: Service Layer



Service Layer reiktų naudoti todėl, kad:

- padės įgyvendinti sistemoje Service Facade projektavimo šabloną,
- Tarpinis sluoksnis kuriame įgyvendinama visa veiklos logika
- Integruoja duomenis iš daug šaltinių (DAO/repositorijų)

Service Layer integruoja daug DAO



@Service anotacija

@Service anotacijų pagalba sukuriami Service‘s

Spring automatiškai priregistruos Service‘s klases, todėl bus galima vykdyti DI

Services įgyendinamos panašiai kaip DAO klasės:

- Sukuriamas Service interfeisas su visais metodais
- Sukuriama Service interfeisą įgyvendinanti klasė
- Atliekamas DI su visais DAO

Sukurkime service interfeisą

Sukurkime naują paketą .services, jame sukurkime interfeisą TasksService

Sukurkime vieną metodą:

```
public interface TasksService {
   public List<Task> getTasks();
}
```

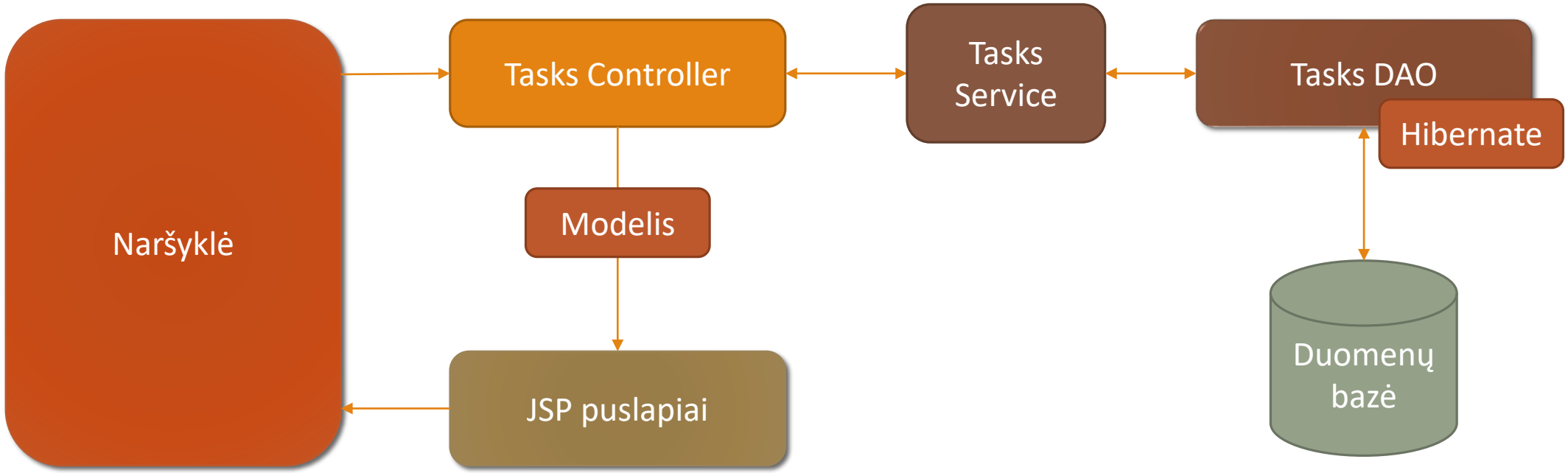
Sukurkime TasksService interfeisą įgyvendinančią klasę

Sukurkime klasę:

```
@Service
public class TasksServiceImpl implements TasksService {
   @Autowired
   private TasksDAO tasksDAO;

  @Transactional
  public List<Task> getTasks(){
  // ….
  }
}
```

Kadangi @Transactional perkėlėme į Service layer, ją turėtumėme pašalinti iš DAO klasės

Bendra sistemos schema



Kontrolerio modifikacija

Modifikuokime kontrolerį taip, kad jame nebenaudotumėme DAO, o naudotumėmės tik Service.

Jį taip pat galime prisidėti tokiu būdu:

@Autowired

Private TasksService tasksService;