Roteiro para aula prática 4 – Persistência de Dados

Tempo estimado para esta prática: 30 min

IDE utilizada: Eclipse JavaEE

Banco de Dados: Mysql Workbench

Nessa aula iremos fazer a persistência de dados na nossa aplicação para gerenciar tarefas. Os arquivos necessário para essa aula estão disponíveis na rota de aprendizagem.

1. Colocar starter do jpa

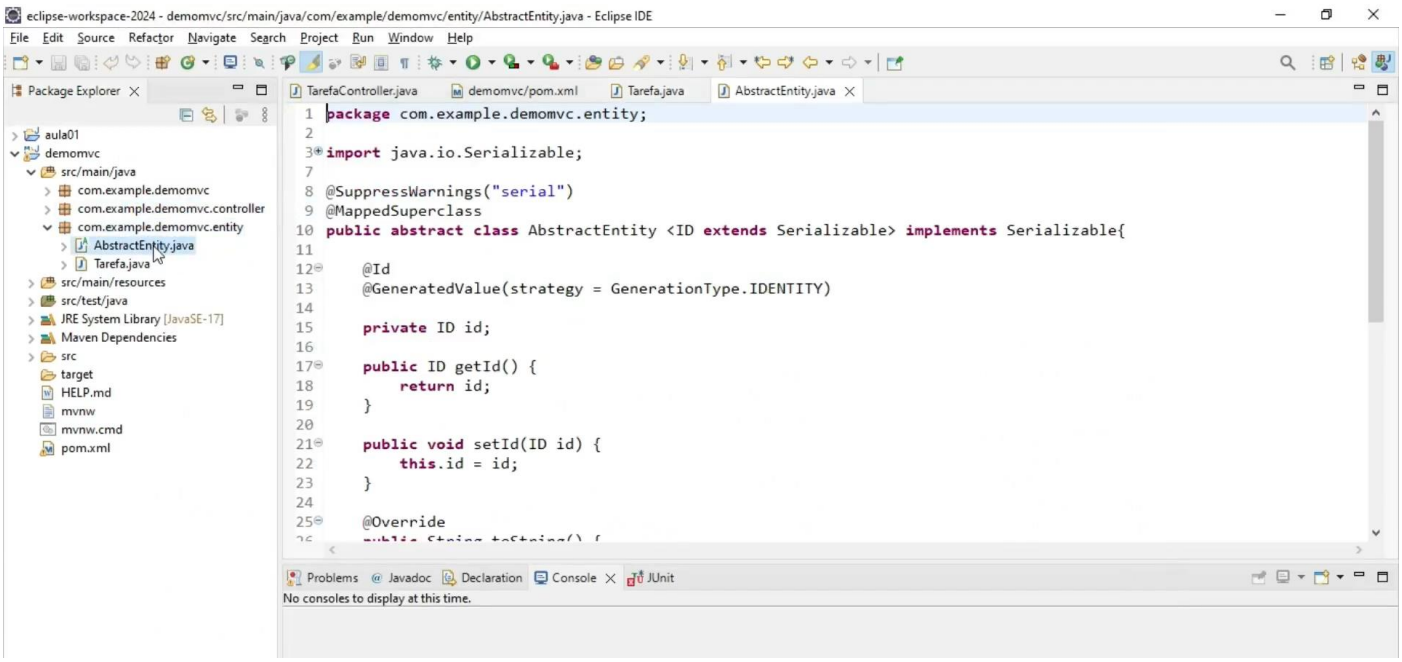
Onde procuraro starter do jpa no repositório do maven:

https://mvnrepository.com/

Colocar a dependência abaixo no pom.xml:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>3.2.4</version>
</dependency>
```

2. Criar uma classe abstrata na pasta Entity com o nome de AbstractEntity. Essa classe já está criada e pode ser copiada para a pasta Entity.



Nessa classe foram sobrescritos os métodos: equals, toString e Hashcode.

Observe que foi MODIFICADO APENAS UMA LINHA NO MÉTODO EQUALS

Observe ainda que estamos utilizando a classe Serializable. A simples implementação dessa interface é suficiente para sinalizar a capacidade de serialização da classe
o código final da classe Abstractentity ficará assim:

```
package com.example.demomvc.entity;
import java.io.Serializable;
import java.util.Objects;
import jakarta.persistence.*;
@SuppressWarnings("serial")
@MappedSuperclass
public abstract class AbstractEntity <ID extends Serializable> implements Serializable{

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private ID id;
	public ID getId() {
		return id;
	}
	public void setId(ID id) {
		this.id = id;
	}
	@Override
	public String toString() {
		return "AbstractEntity [id=" + id + "]";
	}
	@Override
	public int hashCode() {
		return Objects.hash(id);
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		AbstractEntity <?> other = (AbstractEntity <?>) obj;
		return Objects.equals(id, other.id);
	}
}
```

3. Na classe de entidade Tarefa (já está criada)

- Retirar o campo id, getter e setter do id, tostring, equals e hashcode.
- A classe Tarefa agora deve extender a classe AbstractEntity:

public class Tarefa extends AbstractEntity<Long> {}

classe Tarefa.java:

```
package com.example.demomvc.entity;
import java.time.LocalDate;
import org.springframework.format.annotation.DateTimeFormat;
@SuppressWarnings("serial")
public class Tarefa extends AbstractEntity<Long>{
```

```
	private String nome;

	@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
	private LocalDate dataEntrega;
	private String responsavel;


	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public LocalDate getDataEntrega() {
		return dataEntrega;
	}
	public void setDataEntrega(LocalDate dataEntrega) {
		this.dataEntrega = dataEntrega;
	}
	public String getResponsavel() {
		return responsavel;
	}
	public void setResponsavel(String responsavel) {
		this.responsavel = responsavel;
	}

}
```

4. Fazer o Mapeamento objeto relacional (ORM)
Informar ao framework que está relacionando a sua classe a uma entidade do banco de dados.

mapeamento da classe de entidade:

`@Entity:` definir uma entidade para que o JPA tenha conhecimento dela
`@Table(name="TAREFA"):` especificar o nome da tabela usando a anotação @Table. Nesse caso nem precisaria dessa anotação, pois o nome da tabela é o mesmo da classe Tarefa.
`@Column(name="nome", nullable=false, unique=true, length=60)`: nomear e detalhar uma coluna da tabela
`@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)`: especifica como uma data deve ser formatada ou interpretada durante a conversão entre uma string e um objeto de data. Indica que a data deve estar no formato padrão ISO 8601, que é yyyy-MM-dd.

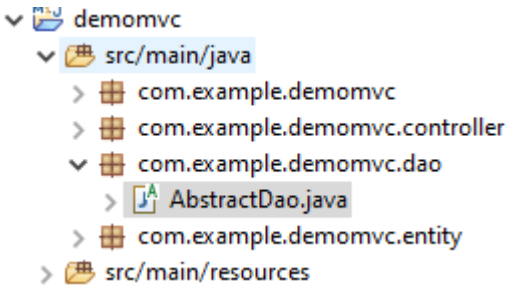
Abaixo segue código com o mapeamento(em amarelo).

```
package com.example.demomvc.entity;
import java.time.LocalDate;
import java.util.Date;
import java.util.Objects;
import org.springframework.format.annotation.DateTimeFormat;
import jakarta.persistence.*;

@Entity
@Table(name="TAREFA")
public class Tarefa extends AbstractEntity<Long>{

	@Column(name="nome", nullable=false, unique=true, length=60)
	private String nome;

	@Column( nullable=false, name="data_entrega", columnDefinition="DATE")
```

```
	@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
	private LocalDate dataEntrega;

	@Column(name="responsavel", nullable=false, length=60)
	private String responsavel;



	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}

	public LocalDate getDataEntrega() {
		return dataEntrega;
	}
	public void setDataEntrega(LocalDate dataEntrega) {
		this.dataEntrega = dataEntrega;
	}
	public String getResponsavel() {
		return responsavel;
	}
	public void setResponsavel(String responsavel) {
		this.responsavel = responsavel;
	}

}
```

5. Criar classes DAOs

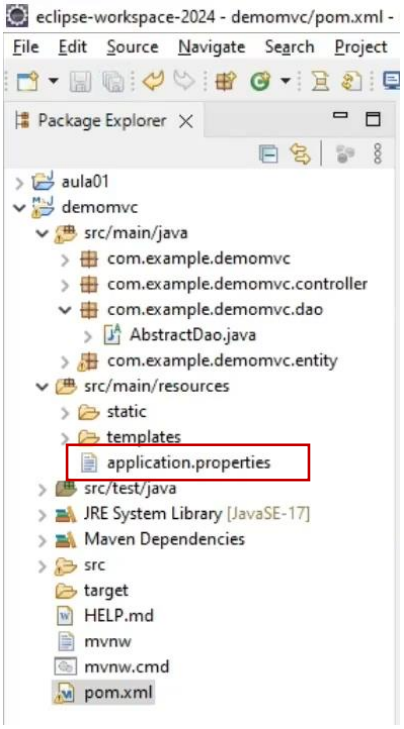
- criar package dao
- copiar e colar o arquivo AbstractDao.java no pacote dao criado



Obs: Arquivo disponibilizado na rota da aula.

6. Configuração para Banco de Dados

- Copiar a configuração no arquivo application.properties localizado na pasta resources.



Configuração do application,properties

```
spring.application.name=demomvc

#DATASOURCE
spring.datasource.url=
jdbc:mysql://localhost:3306/bd_mvc?createDatabaseIfNotExist=true
spring.datasource.username= root
spring.datasource.password= root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#JPA
spring.jpa.hibernate.ddl-auto= update
spring.jpa.show-sql= true
spring.jpa.open-in-view= true
```

A seguir sucinta explicação sobre as configurações:

`spring.datasource.url=`
`jdbc:mysql://localhost:3306/bd_mvc?createDatabaseIfNotExist=true`

Define a URL de conexão. Cria/roda um banco com nome bd_mvc.

`spring.datasource.username= root`
`spring.datasource.password= root`
Especifica o nome do usuário e senha respectivamente. Devem ser os mesmos que colocou no workbench.

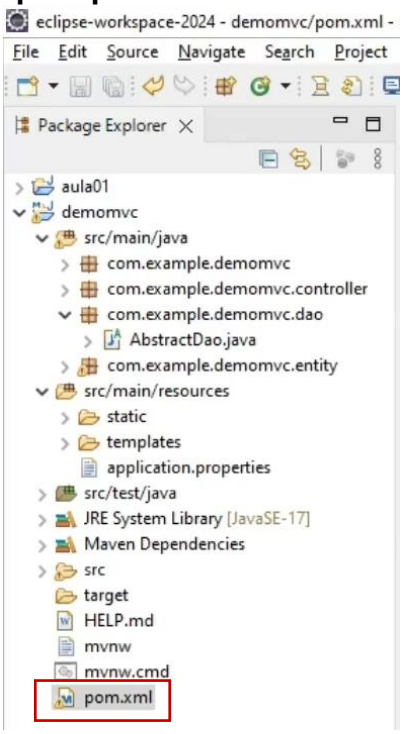
`spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver`
Especifica a classe do driver JDBC que o Spring Boot deve usar para se conectar ao banco de dados.

`spring.jpa.hibernate.ddl-auto= update`
O valor update indica que o Hibernate deve atualizar o esquema do banco de dados automaticamente ao iniciar a aplicação, sem perder os dados existentes.

`spring.jpa.show-sql= true`
Quando configurado como true, esta opção faz com que o Hibernate mostre as instruções SQL

`spring.jpa.open-in-view= true`
Mantém a sessão do Hibernate aberta durante toda a duração de uma requisição web, até que a resposta seja completamente gerada.

7. colocar a dependência do mysql no pom.xml

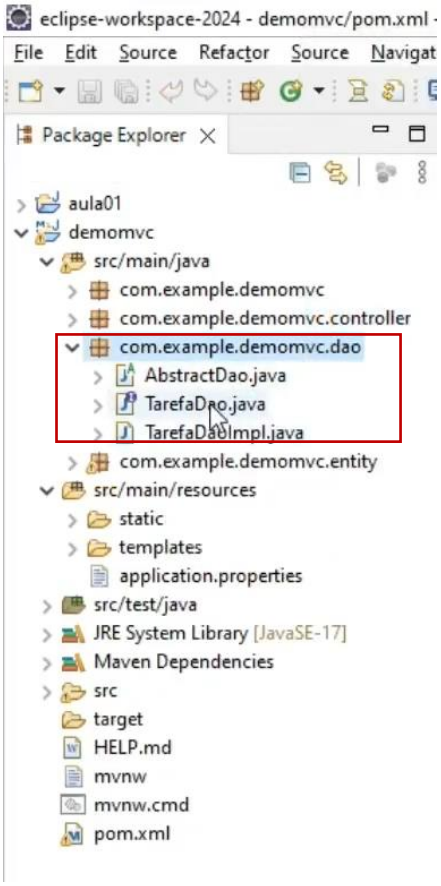



```
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
</dependency>
```

8. Criar a classe de persistência no pacote dao

Copiar e colar as classes abaixo no pacote dao. Elas estão disponíveis na rota.

- Classe PalavraDaoImpl.java
- Interface PalavraDao.java



Interface PalavraDao

Abaixo está o código da interface com a assinatura dos métodos para salvar, atualizar, deletar, pesquisar e listar.

```
import java.util.List;
import com.example.demomvc.entity.Tarefa;
public interface TarefaDao {
	void save(Tarefa tarefa);
	void update(Tarefa tarefa);
	void delete(Long id);
	Tarefa findById(Long id);
	List<Tarefa> findAll();
}
```

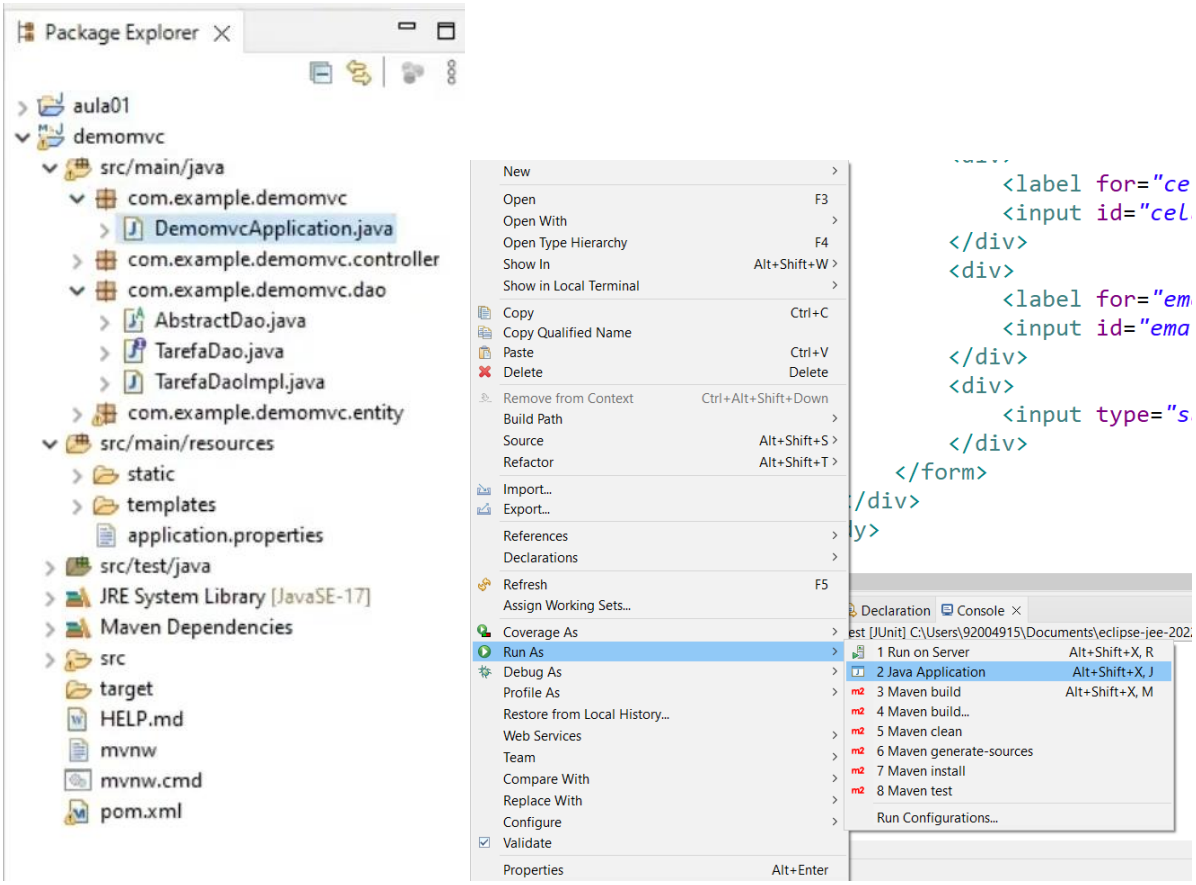
Classe PalavraDaoImpl

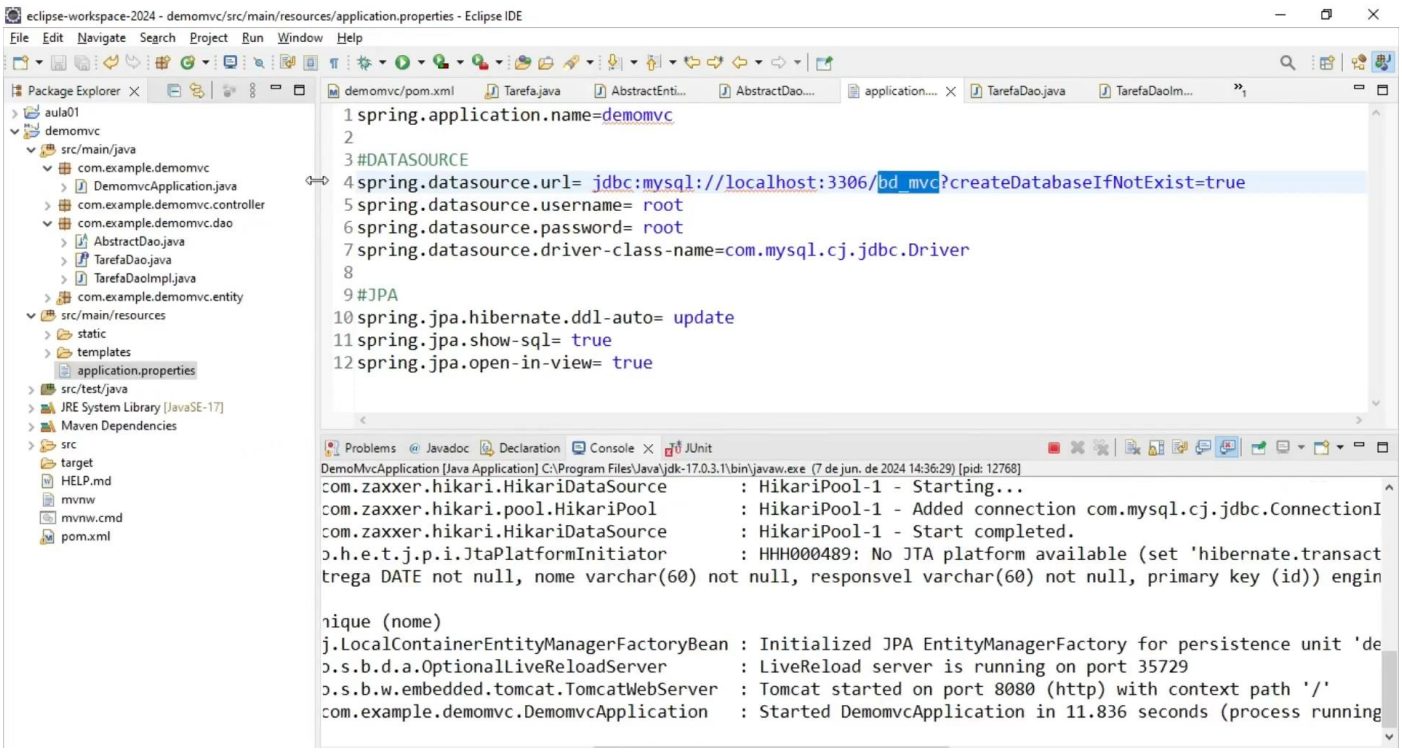
A anotação @Repository indica que a classe vai interagir com o BD.

```
package com.example.demomvc.dao;
import org.springframework.stereotype.Repository;
import com.example.demomvc.entity.Tarefa;
@Repository
public class TarefaDaoImpl extends AbstractDao<Tarefa, Long> implements TarefaDao  {
}
```

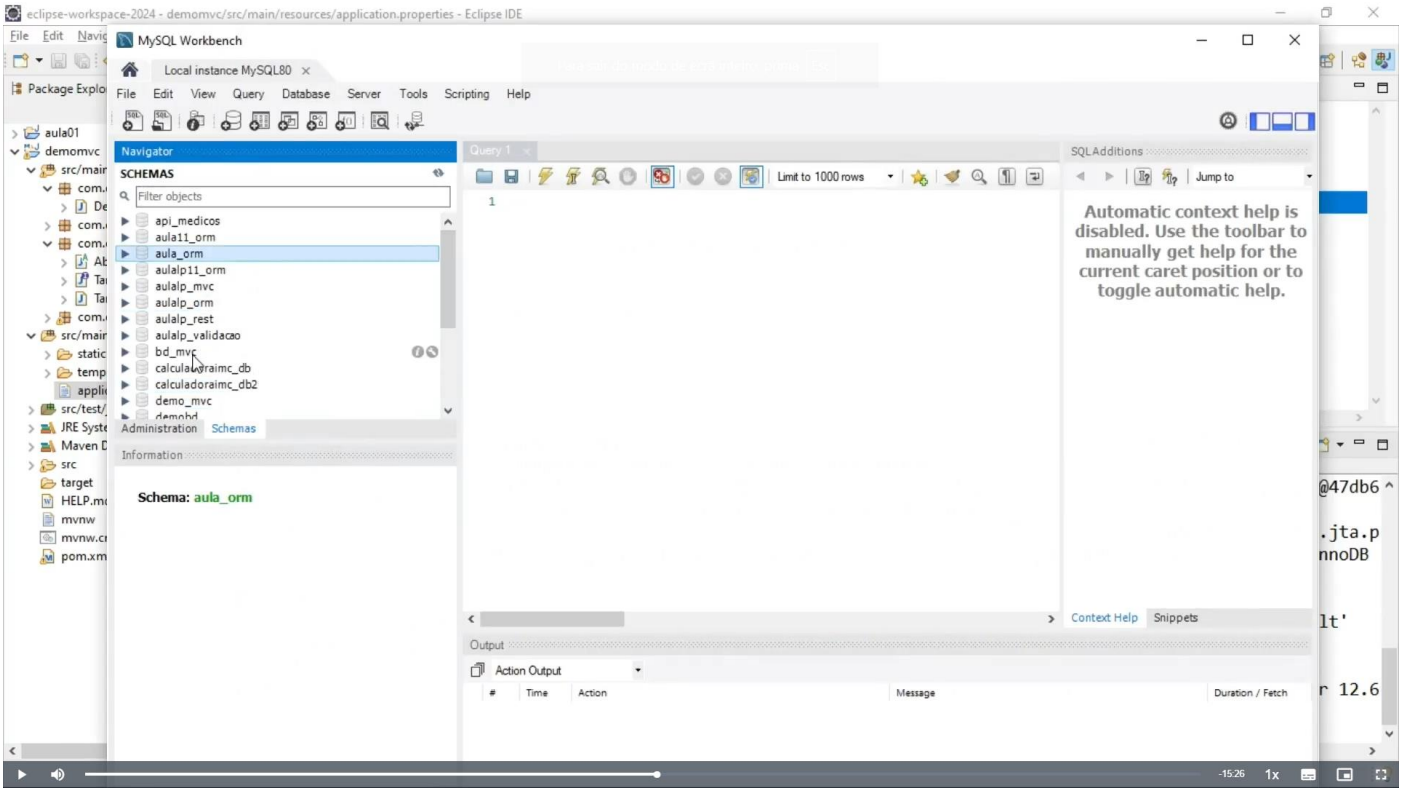
Testar e verificar se cria o banco de dados no mysql workbench.

Para Compilar a aplicação vá na classe DemomcApplication.java, clique com o botão direito do mouse : Run As > Java Application





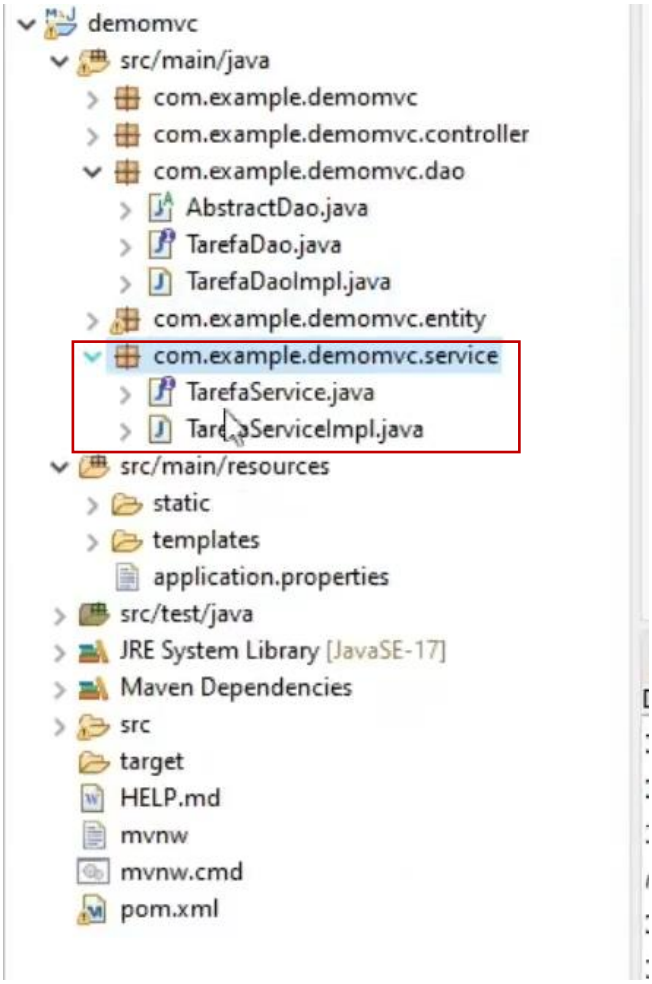
Verifique no workbench que seu banco já foi criado



9. Criar as Classes de Servico

Implementaremos a classe de serviço.

- Criar pacote service
- Colocar as duas classes java que estão disponibilizadas na pasta da rota
    - TarefaService
    - TarefaServiceImpl



Abaixo está o código da interface com a assinatura dos métodos para salvar, editar, excluir, buscarPorId e buscaTodos.

```
package com.example.demomvc.service;
import java.util.List;
import com.example.demomvc.entity.Tarefa;
public interface TarefaService {
	void salvar(Tarefa tarefa);
	void editar(Tarefa tarefa);
	void excluir(Long id);
	Tarefa buscarPorId(Long id);
	List<Tarefa> buscaTodos();
}
```

Classe TarefaServiceImpl é uma classe que vai implementar a interfaceTarefaService

Nessa classe foram incluídas:

- A anotação @Service - marca uma classe como um serviço. indicar que a classe contém a lógica de negócios ou coordena as operações entre várias classes de repositório (repositories).
- A anotação @Transactional – marca a classe que é usada para gerenciar as transações
- A anotação @Autowired - injeta automaticamente as dependências em um bean gerenciado pelo contêiner do Spring.
- Uma variável TarefaDao
- Chamada aos métodos do dao

```
import java.util.List;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.demomvc.dao.TarefaDao;
import com.example.demomvc.entity.Tarefa;
@Service
@Transactional(readOnly=false)
public class TarefaServiceImpl implements TarefaService {

	@Autowired
	private TarefaDao dao;
	@Override
	public void salvar(Tarefa tarefa) {
		// TODO Auto-generated method stub
		dao.save(tarefa);

	}
	@Override
	public void editar(Tarefa tarefa) {
		// TODO Auto-generated method stub
		dao.update(tarefa);
	}
	@Override
	public void excluir(Long id) {
		// TODO Auto-generated method stub
		dao.delete(id);
	}
	@Override
	@Transactional(readOnly=true)
	public Tarefa buscarPorId(Long id) {
		// TODO Auto-generated method stub
		return dao.findById(id);
	}
	@Override
	@Transactional(readOnly=true)
	public List<Tarefa> buscaTodos() {
		// TODO Auto-generated method stub
		return dao.findAll();
	}
}
```

Testar a página e verificar que não cadastrou no banco por que os controllers não estão usando as classes de serviço

10. modificar os controllers porque agora usaremos Banco de Dados

Retirar o que está em vermelho e substituir pelo código em amarelo(disponível na rota de aprendizagem)

classe TarefaController

```java
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
```

```java
import com.example.demomvc.entity.Tarefa;
@Controller
@RequestMapping("/tarefas")
public class TarefaController {
	List<Tarefa> tarefas = new ArrayList<>();
	@GetMapping("/cadastro")
	public String cadastro(Tarefa tarefa) {
		return "/tarefa/cadastro";
	}

	@PostMapping("/salvar")
	public String salvar(Tarefa tarefa) {
	Long id = tarefas.size() + 1L;
	Tarefa t = new Tarefa();
	t.setId(id);
	t.setNome(tarefa.getNome());
	t.setDataEntrega(tarefa.getDataEntrega());
	t.setResponsavel(tarefa.getResponsavel());
	tarefas.add(t);
	System.out.println(tarefas.get(0).getNome());
		return "redirect:/tarefas/lista";
	}

	@GetMapping("/lista")
	public ModelAndView lista() {
	ModelAndView mv = new ModelAndView("tarefa/lista");
	mv.addObject("tarefas", tarefas);
		return mv;
	}


	// editar e excluir
	@PostMapping("/editar")
	public String editar(Tarefa tarefa) {
	Tarefa t = new Tarefa();
	for (int i = 0; i < tarefas.size(); i++) {
	if (tarefas.get(i).getId().equals(tarefa.get
	t = tarefas.get(i);
	}
	}
	tarefas.set(tarefas.indexOf(t), tarefa);
return "redirect:/tarefas/lista";



		}
@GetMapping("/excluir/{id}")
public String excluir(@PathVariable("id") Long id) {
Tarefa tarefa;
for (int i = 0; i < tarefas.size(); i++) {
if (tarefas.get(i).getId().equals(id)) {
tarefa = tarefas.get(i);
tarefas.remove(tarefa);
}
}
return "redirect:/tarefas/lista";
		}
@GetMapping("/editar/{id}")
public ModelAndView preEditar(@PathVariable("id") Long id) {
ModelAndView mv = new ModelAndView();
mv.setViewName("tarefa/cadastro");
```

```java
@Autowired
Private TarefaService service;
```

```java
@PostMapping("/salvar")
public String salvar(Tarefa tarefa) {
	service.salvar(tarefa);

return "redirect:/tarefas/cadastro";
	}
```

```java
@GetMapping("/lista")
public String listar(ModelMap model){
model.addAttribute("tarefas",
service.buscaTodos());
return "tarefa/lista";
}
```

```java
@PostMapping("/editar")
public String editar(Tarefa tarefa) {
service.editar(tarefa);
return "redirect:/tarefas/cadastro";
}
```

```java
@GetMapping("/excluir/{id}")
public String excluir(@PathVariable("id")
Long id, ModelMap model) {
service.excluir(id);
return listar(model);
	}
```

```java
@GetMapping("/editar/{id}")
public String
preEditar(@PathVariable("id") Long id,
```

```
Tarefa tarefa = null;
for (int i = 0; i < tarefas.size(); i++) {
      if (tarefas.get(i).getId().equals(id)) {
      tarefa = tarefas.get(i);
System.out.println("@GetMapping");
System.out.println(tarefa.getId());
      }
      }
mv.addObject("tarefa", tarefa);
      return mv;
            }
}
```

Testar e verificar que insere.

Testar também o editar. Verificar que o editar está errado. Vamos precisar modificar o cadastro.html

11. Modificar o cadastro.html

Colocar o namespace do thymelead no cadastro.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```

Fazer algumas modificações no nosso form (em amarelo).
O comando th:field="*{nome}" vincula um campo de formulários HTML a uma propriedade de um objeto de modelo em uma aplicação Spring MVC usando Thymeleaf.

cadastro.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Cadastro de Tarefas</title>
<!--  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">-->
<!-- Bootstrap core CSS -->
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet" />

<link rel="stylesheet" type="text/css" media="all" href="../../css/style.css"  />
</head>
<body>

<form th:action="${tarefa.id == null} ? @{/tarefas/salvar} : @{/tarefas/editar}"
                        th:object="${tarefa}" method="POST">

<input type="hidden"  th:field="*{id}">


 <div class="mb-3">
   <label for="exampleInputTarefa" class="form-label">Nome da Tarefa</label>
   <input type="text" class="form-control" id="nome" name = "nome" th:field="*{nome}">
 </div>

  <div class="mb-3">
   <label for="exampleInputDataEntrega" class="form-label">Data entrega: </label>
```

```
   <input type="date" class="form-control" id="datacriacao" aria-describedby="datalHelp"
name="dataEntrega" th:field="*{dataEntrega}" >

 </div>
  <div class="mb-3">
   <label for="exampleInputResponsavel" class="form-label">Responsável: </label>
   <input type="text" class="form-control" id="responsavel" aria-describedby="responsavellHelp"
name="responsavel" th:field="*{responsavel}" >

 </div>
 <button type="submit" class="btn btn-primary">Submit</button>
</form>
<div >&copy; 2024 modelo estático</div>
</body>
</html>
```