



Análise e Desenvolvimento de Sistemas

Prof. MSc Marcelo Tomio Hama

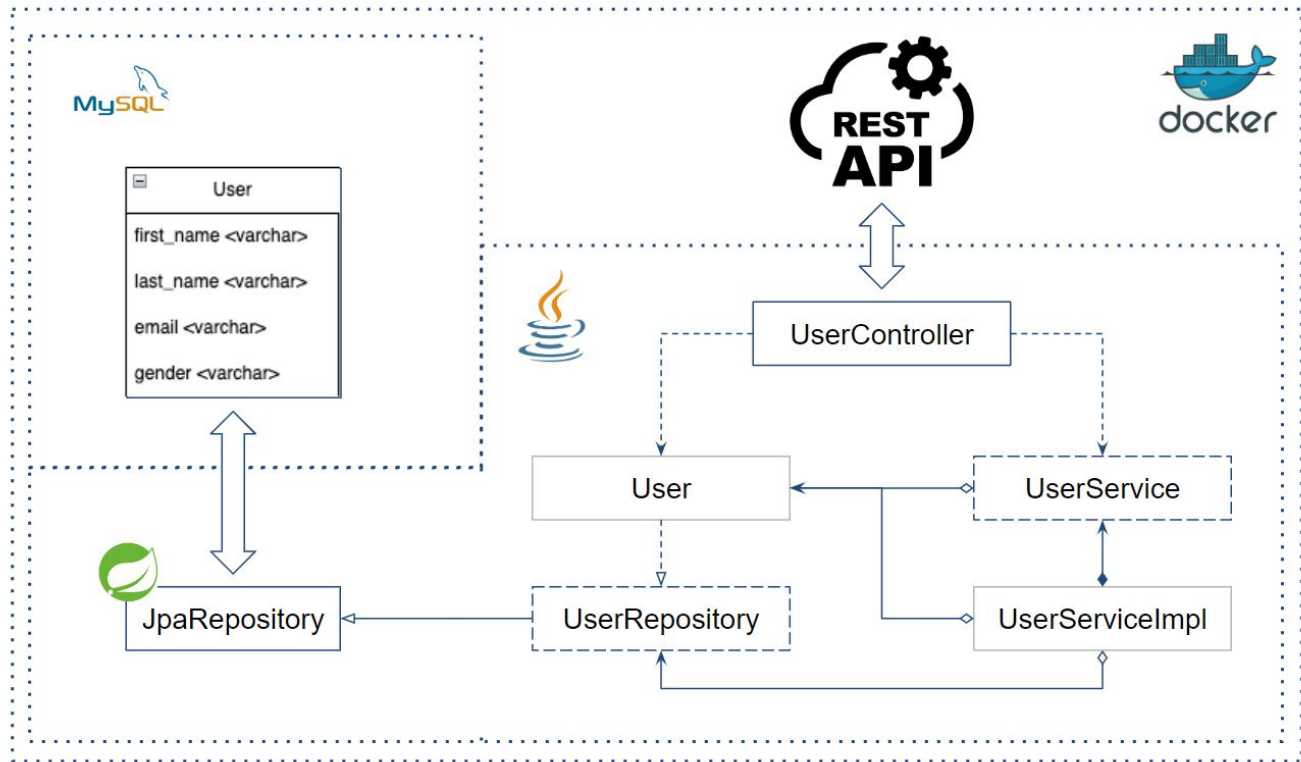
## AULA 4

# Entidades (Anotações), Repository (JpaRepository), Injeção de Dependência.

### OBJETIVOS

1. Revisar e aplicar os conhecimentos obtidos da aula anterior
2. Checar/corriger/comentar os exercícios da aula anterior;
3. Lorem ipsum

# Arquitetura do Nosso Case



Trata-se de uma pequena aplicação dockerizada que executa sob um container docker instalado e publicado localmente.

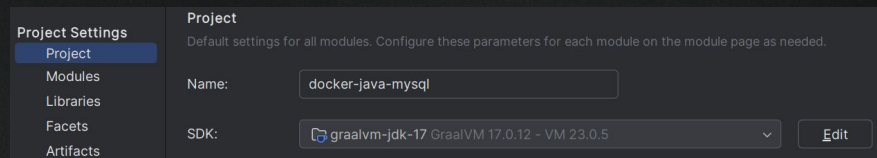
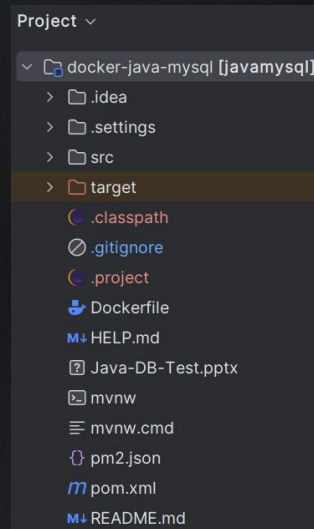
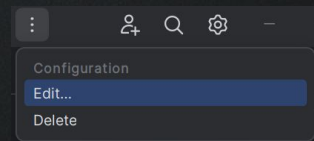
A aplicação faz uso das funções mínimas do JPA, do MySQL (executado em outro container), e que publica APIs para consumo local.

O JPA (Java Persistence API) facilita o desenvolvimento de tecnologias de acesso a dados.

# Hands-On Laboratorial | Ambiente

## Passo-a-Passo Geral (Windows 10)

1. Faça a instalação do IntelliJ IDEA
  - a. Download: <https://www.jetbrains.com/pt-br/idea/download/?section=windows>
  - b. Configure a licença gratuita: <https://www.jetbrains.com/shop/eform/students>
2. Faça a instalação do Docker Desktop;
  - a. Download: <https://www.docker.com/products/docker-desktop/>
  - b. Crie sua conta em <https://app.docker.com/> e faça o login no Docker Desktop;
3. Faça o clone do repositório:  
<https://github.com/marcelohama/docker-java-mysql>
4. Crie o projeto no IntelliJ IDEA
  - a. File > New -> Project from existing sources, selecione o diretório raiz do repositório clonado “docker-java-mysql”;
  - b. No ícone “engrenagem” vá em Project Structure e depois em Project, e configure o SDK para usar o graalvm-jdk-17;
  - c. Nos “3 pontinhos”, clique em edit para criar uma config de build, e use a seguinte linha para run: “*-Dmaven.test.skip=true package*”





# Hands-On Laboratorial | MySQL

## Subindo um Container MySQL e Criando Tabela

1. Abra o terminal do IntelliJ IDEA, local ao diretório raiz do projeto
2. Com o dashboard do Docker Desktop aberto, execute a linha para criar uma rede para ponte entre containers/aplicações:

```
$ docker network create --driver bridge javamysql-network
```

3. Crie um container mysql com imagem da comunidade:

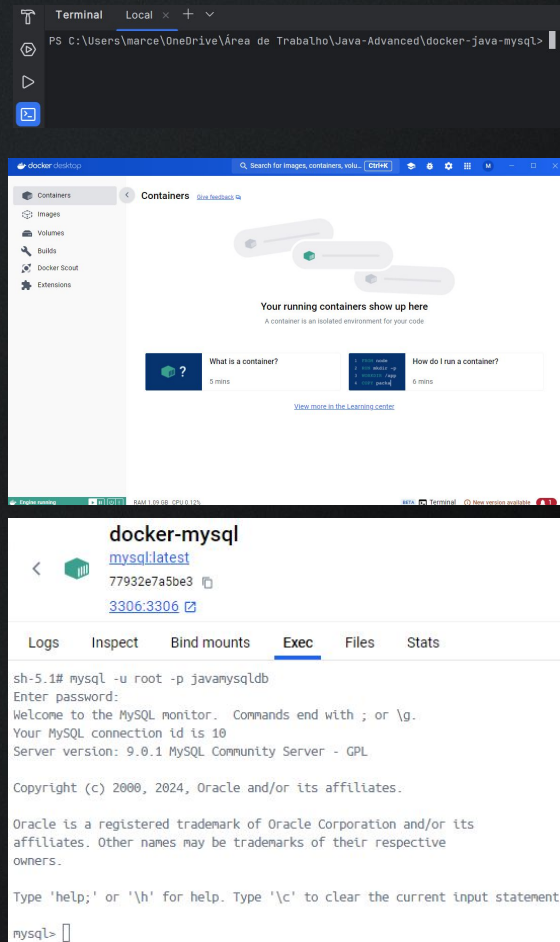
```
$ docker container run -p 3306:3306 --name docker-mysql --network  
javamysql-network -e MYSQL_ROOT_PASSWORD=1q2w3e4r5t -e  
MYSQL_DATABASE=javamysqldb -d mysql:latest
```

4. Entre no bash do servidor MySQL, clicando no container MySQL visível no Docker Desktop e depois em exec, e entre no CLI do servidor MySQL com o root no schema criado:

```
$ mysql -u root -p javamysqldb
```

5. Dentro do CLI, execute a query para criar uma tabela USERS:

```
$ CREATE TABLE users ( id int NOT NULL,  
first_name varchar(255) NOT NULL,  
last_name varchar(255),  
email varchar(255),  
gender varchar(255)  
);
```



# Hands-On Laboratorial | MySQL

## Criando Registros no Banco

1. Insira os registros no banco de dados:

```
$ INSERT INTO users (id, first_name, last_name, gender, email) VALUES (1, 'donatello', 'Ninja Turtle', 'male', 'donatello@fiap.com.br');
```

```
$ INSERT INTO users (id, first_name, last_name, gender, email) VALUES (2, 'leonardo', 'Ninja Turtle', 'male', 'leonardo@fiap.com.br');
```

```
$ INSERT INTO users (id, first_name, last_name, gender, email) VALUES (3, 'michelangelo', 'Ninja Turtle', 'male', 'michelangelo@fiap.com.br');
```

```
$ INSERT INTO users (id, first_name, last_name, gender, email) VALUES (4, 'rafael', 'Ninja Turtle', 'male', 'rafael@fiap.com.br');
```

2. Faça um teste de leitura dos registros criados:

```
$ SELECT * FROM users;
```

```
mysql> SELECT * FROM users;
+----+-----+-----+-----+-----+
| id | first_name | last_name | email | gender |
+----+-----+-----+-----+-----+
| 1 | donatello | Ninja Turtle | donatello@fiap.com.br | male |
| 2 | leonardo | Ninja Turtle | leonardo@fiap.com.br | male |
| 3 | michelangelo | Ninja Turtle | michelangelo@fiap.com.br | male |
| 4 | rafael | Ninja Turtle | rafael@fiap.com.br | male |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```













1. Execute o build clicando no botão verde “play”;
2. Faça a construção do container com o pacote `jar` spring boot criado:

### 3. Execute o container criado:

```
.      _-----_      _-----_
/\ \ /___'__-____-(_)___-___-\ \ \ \
( ( )__\ | ' | ' | ' | ' \_ ' | \ \ \ \
\ \ / ___| | | | | | | | | | | | | | | |
'   | ____| ._| | | | | | | | | | | | | |
=====|_|=====|_|_/=//_/_/_/

:: Spring Boot ::                (v3.0.1)
```

```
2024-08-11T18:42:42.726Z INFO 1 --- [main] br.com.javamysql.Javamys  
g Java 17-ea with PID 1 (/javamysql-0.0.1-SNAPSHOT.jar started by root in /)
```

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions		
 <a href="#">docker-1</a> 77932e7a	<a href="#">mysql:lates</a>	Running	<a href="#">3306:3306</a> 	0.72%	37 minutes			
 <a href="#">java-bac</a> 7c9886dc	<a href="#">javamysql-l</a>	Running	<a href="#">8080:8080</a> 	0.38%	3 minutes a			



# Hands-On Laboratorial | Backend

## Simulando o Backend JAVA na rede

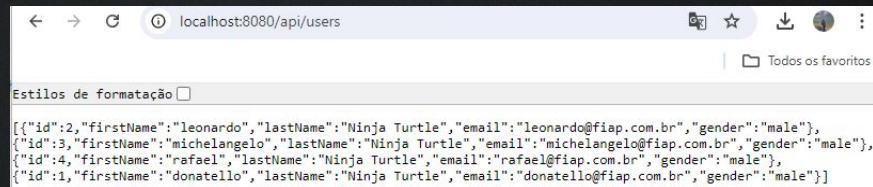
Acesse <http://localhost:8080/api/users>

## Opcional: Teste as chamadas com o Postman

1. Download: <https://www.postman.com/downloads/>
2. GET, POST, PUT, DELETE

## Opcional: Exponha seu Serviço na rede pública

1. Download do ngrok: <https://ngrok.com/download> e crie uma conta em <https://dashboard.ngrok.com>
2. Crie seu token em <https://dashboard.ngrok.com/tunnels/auth/tokens>
3. Abra o ngrok e configure o token:  
`$ ngrok authtoken <NGROK_AUTHTOKEN>`
4. Execute o ngrok no terminal do IntelliJ IDEA:  
`$ ./ngrok http 8080`
5. Em um dispositivo diferente, abra a URL:  
`<forwarding_url>/api/users`

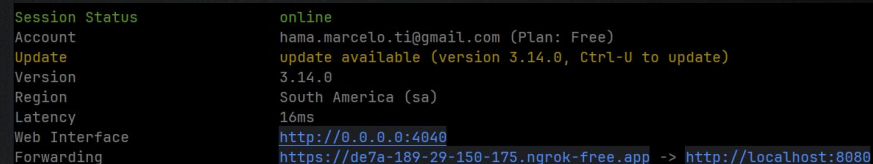


```
localhost:8080/api/users

Estilos de formatação

[{"id":2,"firstName":"leonardo","lastName":"Ninja Turtle","email":"leonardo@fiap.com.br","gender":"male"}, {"id":3,"firstName":"michelangelo","lastName":"Ninja Turtle","email":"michelangelo@fiap.com.br","gender":"male"}, {"id":4,"firstName":"rafael","lastName":"Ninja Turtle","email":"rafael@fiap.com.br","gender":"male"}, {"id":1,"firstName":"donatello","lastName":"Ninja Turtle","email":"donatello@fiap.com.br","gender":"male"}]
```

ID	Description	Owner	Metadata	Created
cr_4lAyJA	credential for 'hama.marcelo.ti@gmail.com'	hama.marcelo.ti@gmail.com	0 bytes	5y ago
cr_wBCJuT	Tunnel Authtoken for 'hama.marcelo.ti@gmail.com'	hama.marcelo.ti@gmail.com	0 bytes	2m ago



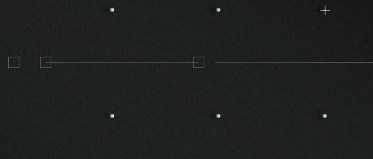
```
Session Status      online
Account            hama.marcelo.ti@gmail.com (Plan: Free)
Update             update available (version 3.14.0, Ctrl-U to update)
Version            3.14.0
Region             South America (sa)
Latency            16ms
Web Interface      http://0.0.0.0:4040
Forwarding          https://de7a-189-29-150-175.ngrok-free.app -> http://localhost:8080
```



# Hands-On Laboratorial

## Exercícios

1. Crie uma nova classe modelo para uma nova entidade “**Address**”;
  - a. Inclua todos os campos/propriedades que achar necessário
  - b. Use o lombok para reduzir a verbosidade;
2. Modele as queries MySQL para:
  - a. Criação da entidade/tabela “**address**”;
  - b. Inserção de registros;
3. Crie um repositório JPA **AddressRepository** para acesso às persistências da entidade
4. Crie uma interface **AddressService** com um protocolo CRUD de manipulação para os registros
5. Crie uma classe **AddressServiceImpl** e implemente os métodos da interface **AddressService**
6. Crie o controller **AddressController**. Deixe-o somente declarado como classe por enquanto.



# Annotations em Entidades

**@Entity** Anota a classe como uma entidade a ser persistida

**@Table** Nomeia a tabela que a classe representa

**@Id** Define o(s) Id(s) único(s) da tabela

**@GeneratedValue** Define a estratégia da geração do Id único

**@Transient** Sinaliza que o campo não deve ser persistido

**@Temporal** Sinaliza que o campo é do tipo DATE

**@Enumerated** Sinaliza que o campo é enumerável

```
CREATE TABLE Student
(
  StudentID INT,
  StudentFirstName VARCHAR(40),
  StudentLastName VARCHAR(40),
  Age INT,
  Course VARCHAR(60)
)
```

## Exercício Prático

Usando o Spring Framework, crie uma classe para servir de entidade para a tabela ilustrada ao lado.



# JPA Repository

## Conceito do JPA

JpaRepository é uma especificação e extensão do Repositório JPA (Java Persistence API). Possui API para operações básicas de CRUD e também API para paginação e classificação.

```
public interface IUserRepository extends JpaRepository<Usuario, Long> {  
    //nome do método deve conter o atributo da classe  
    public Usuario findByLogin(String login);  
  
    //palavra-chave da consulta: Between  
    public List<Usuario> findByIdadeBetween(int startAge, int endAge);  
  
    //consulta por namedQuery  
    public List<Usuario> findByDtCadastro(Date dtCadastro);  
  
    //consulta por named parameters  
    @Query("from Usuario where login = :login and senha = :senha")  
    public Usuario findByLoginAndSenha(@Param("login")String login,  
    @Param("senha") String senha);  
  
    //consulta com ordinal parameters  
    @Query("from Usuario where login = ?3 and senha = ?2 and idade = ?1")  
    public Usuario findByLoginAndSenhaAndIdade  
    (int idade, String senha, String login);  
}
```

Exemplo de uma interface JPA, utilizando queries parametrizadas



# JPA Repository

Palavra chave	Exemplo de Método	Trecho JPQL
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	... where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection age)	... where x.age not in ?1

Palavras chaves do Spring Data JPA no nome dos métodos

# Injeção de Dependência

## Conceito da Injeção de Dependência

A injeção de dependência (DI) é um processo pelo qual os objetos definem suas dependências (ou seja, os outros objetos com os quais trabalham) apenas por meio de argumentos de construtor, argumentos para um método de fábrica ou propriedades que são definidas na instância do objeto após sua construção ou retornado de um método de fábrica.

```
public class SimpleMovieLister {  
  
    // the SimpleMovieLister has a dependency on a MovieFinder  
    private final MovieFinder movieFinder;  
  
    // a constructor so that the Spring container can inject a MovieFinder  
    public SimpleMovieLister(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
  
    // business logic that actually uses the injected MovieFinder is omitted...  
}
```



# Hands-On Laboratorial

## Prepare o “arcabouço” estrutural do seu sistema

- Escreva e/ou complemente as classes que deverão ser usadas em seu futuro projeto prático da disciplina;
- Prepare scripts de banco de dados, para criar tabelas e preenchê-las com registros;
- Use bibliotecas do Lombok para reduzir verbosidade;
- Crie seu próprio repositório no Github e passe a realizar pull/commit/push do seu código nele;

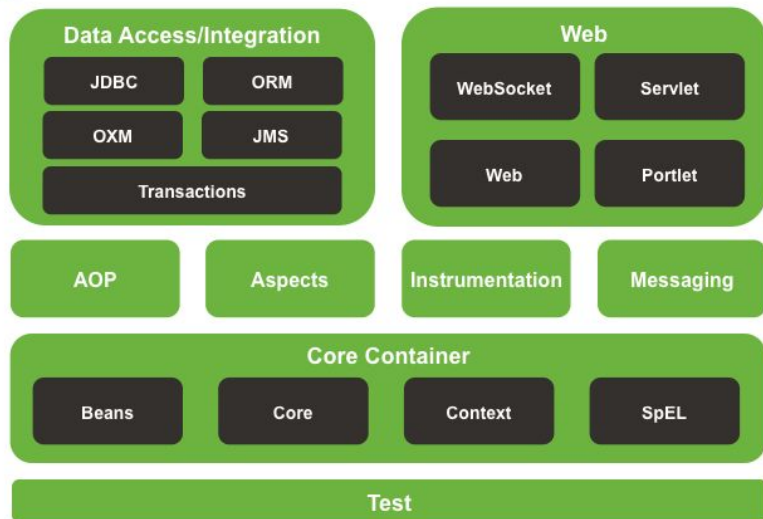
**ITS  
CODE  
TIME**



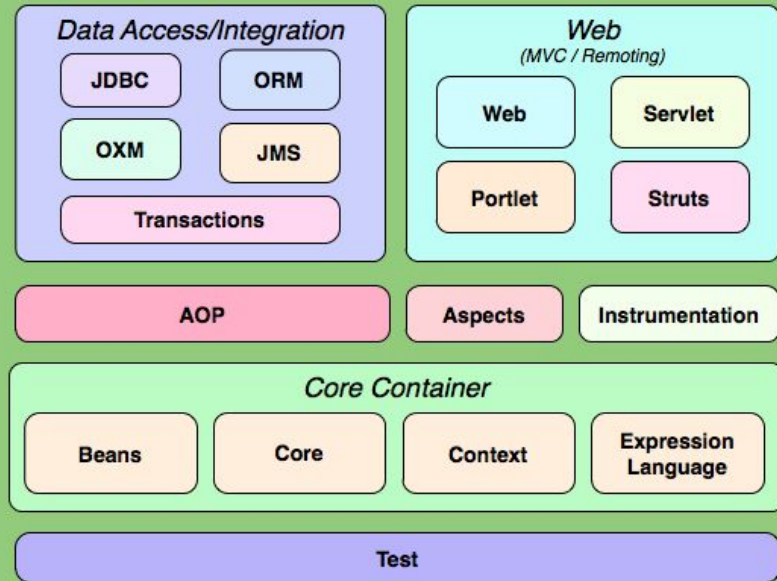
# Framework Spring



## Spring Framework Runtime



## Spring Framework Runtime



Execução do Spring Framework. Fonte:

<https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/spring-introduction.html>

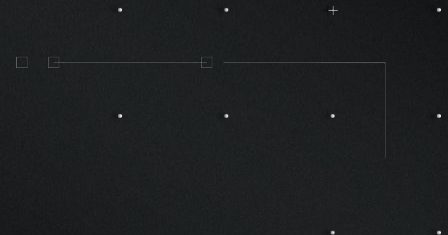
# Referências

## Email do Professor

[profmarcelo.hama@fiap.com.br](mailto:profmarcelo.hama@fiap.com.br)

## Bibliografias/Sites

- <https://www.baeldung.com/jpa-entities>  
<https://www.devmedia.com.br/persistencia-com-spring-data-jpa/24390>  
<https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html>





*"Inteligência é a capacidade de se adaptar a mudanças. A genialidade é antes de tudo a habilidade de aceitar a disciplina."*

Stephen Hawking

**FIM**

---