



Análise e Desenvolvimento de Sistemas

Prof. MSc Marcelo Tomio Hama

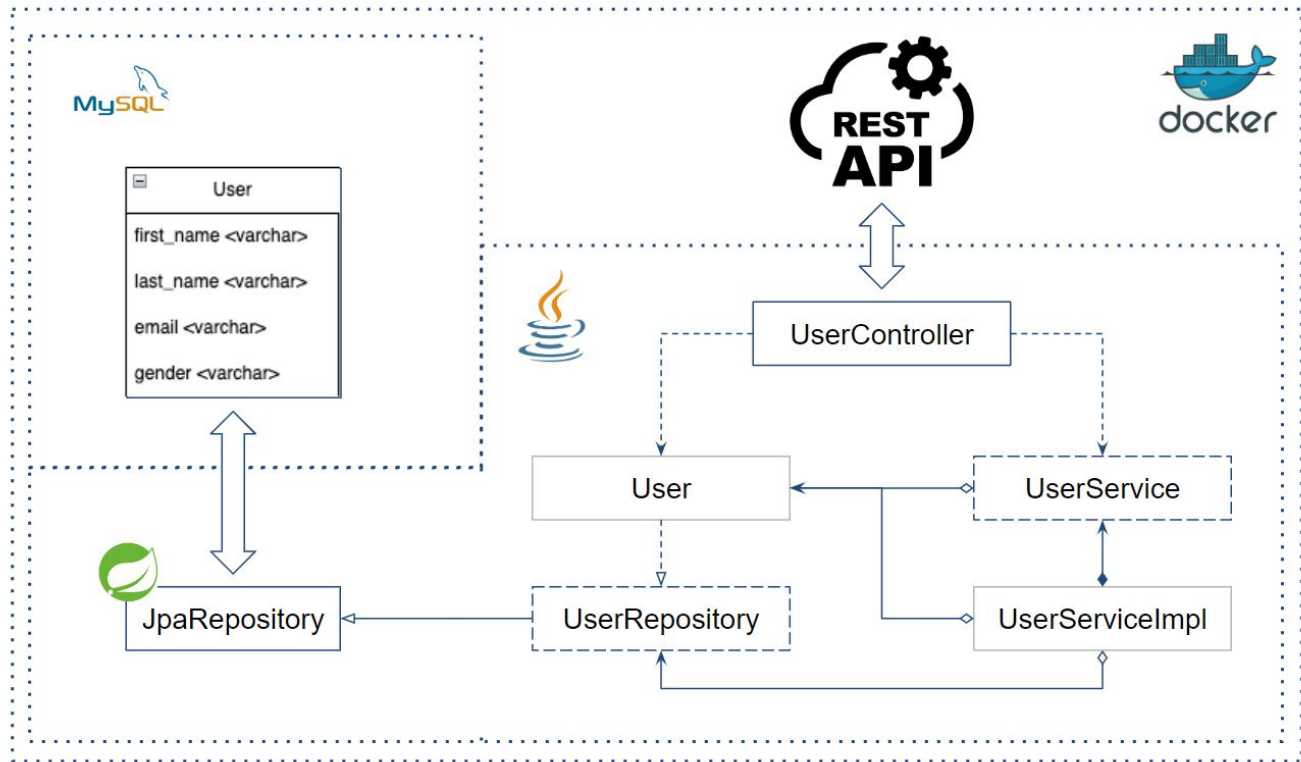
AULA 3

API Restful - Controller, Anotações. Padrões de Projetos (DTO), Java Record. Introdução aos Frameworks de ORM.

OBJETIVOS

1. Revisar e aplicar os conhecimentos obtidos da aula anterior
2. Checar/corriger/comentar os exercícios da aula anterior;
3. Entender e aplicar conceitos de APIs Restful, através da customização de um controller;
4. Entender e aplicar conhecimentos sobre padrões Data Transfer Object (DTO) e Java Record;
5. Introdução no Object Relational Mapping (ORM)

Arquitetura do Nosso Case



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

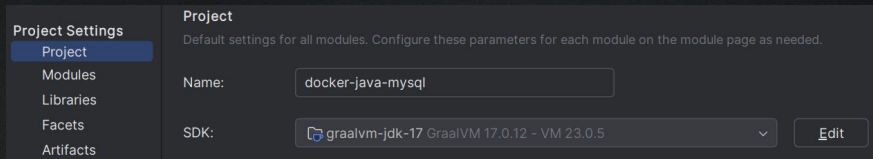
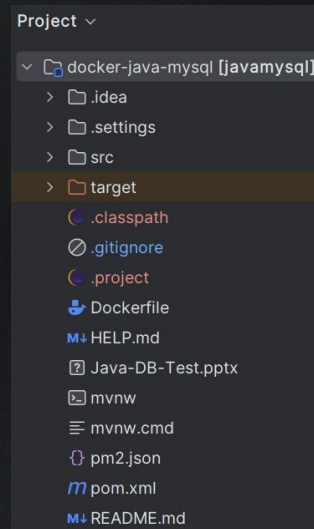
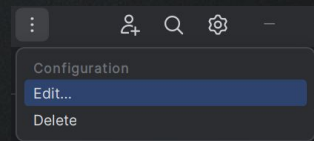
A aplicação faz uso das funções mínimas do JPA, do MySQL (executado em outro container), e que publica APIs para consumo local.

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:

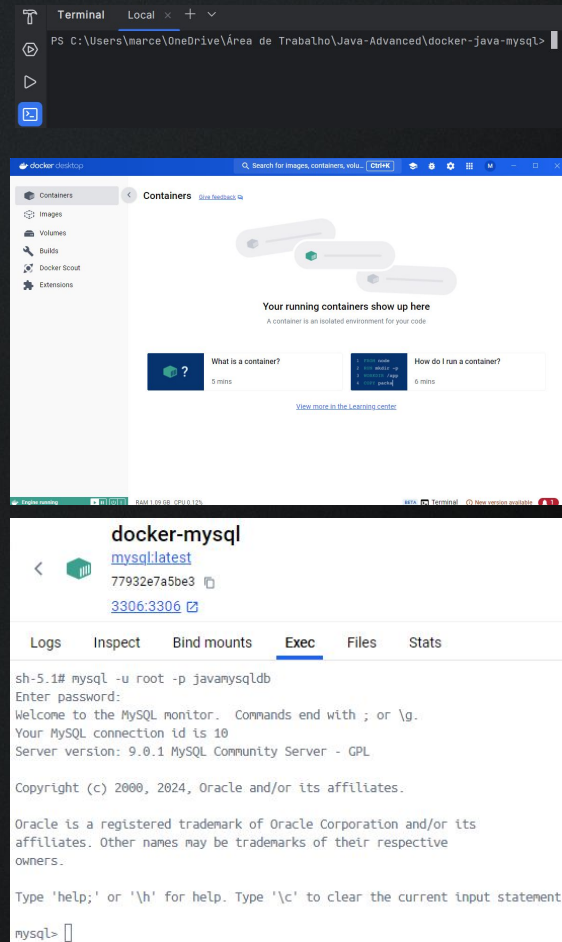
\$ docker container run -p 3306:3306 --name docker-mysql --network javamysql-network -e MYSQL_ROOT_PASSWORD=1q2w3e4r5t -e MYSQL_DATABASE=javamysqldb -d mysql:latest

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');
```

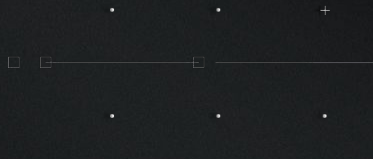
2. Faça um teste de leitura dos registros criados:

```
$ 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:

[illegible]

```
2024-08-11T18:42:42.726Z INFO 1 --- [main] br.com.javamysql.JavaMySQL:
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		
 docker-1 77932e7a	mysql:lates	Running	3306:3306 	0.72%	37 minutes			
 java-bac 7c9886dc	javamysql-l	Running	8080:8080 	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`

```

[[{"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

```

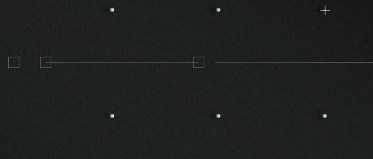
Session Status      online
Account             hama.marcelo.ti@gmail.com (Plan: Free)
Update              update available (version 3.14.0, Ctrl-U to update)
Version             3.14.0
Region              South America (sa)
Latency             16ms
Web Interface       http://0.0.0.0:4040
Forwarding 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

```

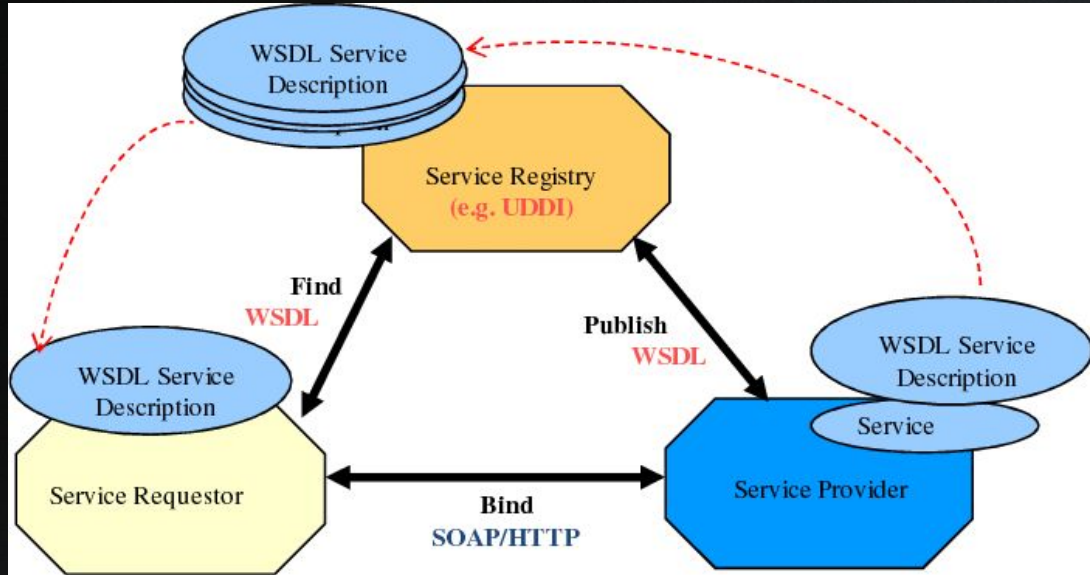

Hands-On Laboratorial

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.



Quando a Internet era Jovem...

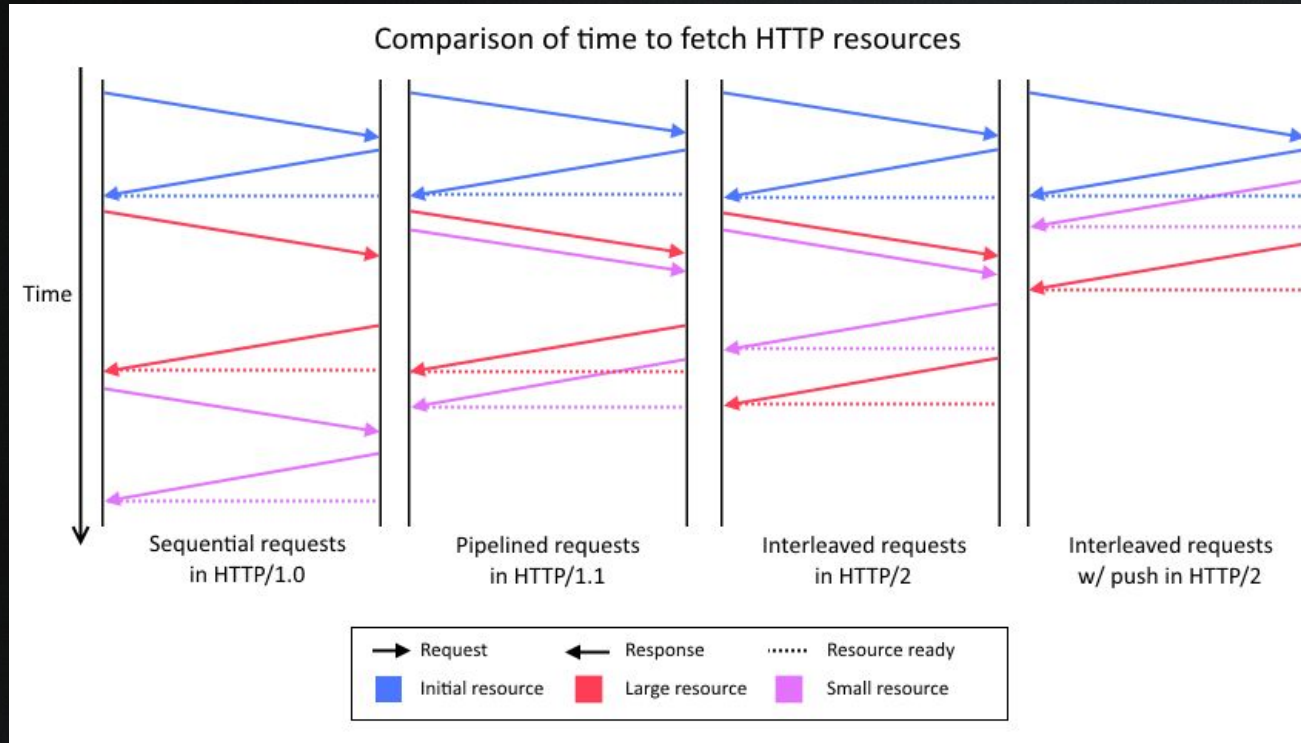


Arquitetura SOAP e exemplo de WSDL.

Fonte: <https://www.researchgate.net/>

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" ...>
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      ...
      <s:element name="GetWordListResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWordListResult" type="tns:ArrayOfString" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetWordListSoapIn">
    <wsdl:part name="parameters" element="tns:GetWordList" />
  </wsdl:message>
  <wsdl:message name="GetWordListSoapOut">
    <wsdl:part name="parameters" element="tns:GetWordListResponse" />
  </wsdl:message>
  <wsdl:portType name="AutoWebServiceSoap">
    <wsdl:operation name="GetWordList">
      <wsdl:input message="tns:GetWordListSoapIn" />
      <wsdl:output message="tns:GetWordListSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="AutoWebServiceSoap" type="tns:AutoWebServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWordList">
      <soap:operation soapAction="http://tempuri.org/GetWordList" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AutoWebService">
    <wsdl:port name="AutoWebServiceSoap" binding="tns:AutoWebServiceSoap">
      <soap:address location="http://www.vietnamofficeexpress.com/AutoWebService.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

API Restful | Evolução do HTTP



Milestones entre HTTP/1.0 e HTTP /2. Fonte:

<https://medium.com/@sandeep4.verma/http-1-to-http-2-to-http-3-647e73df67a8>

API Restful | Conceitos

REST (Representational State Transfer)

É um modelo de arquitetura e não uma linguagem ou tecnologia de programação. Os conceitos do REST foram submetidos à tese de doutorado de Roy Fielding nos anos 2000, onde o princípio fundamental é usar o protocolo HTTP para comunicação de dados.

O REST precisa que um **cliente** faça uma requisição para o **servidor** para enviar ou modificar dados. Uma requisição consiste em:

- Um verbo ou método HTTP, que define que tipo de operação o servidor vai realizar: GET, POST, PUT, DELETE, ou PATCH;
- Um header, com o cabeçalho da requisição que passa informações sobre a requisição;
- Um path (caminho ou rota) para o servidor;
- Informação no corpo da requisição, sendo esta informação opcional.

Respostas HTTP

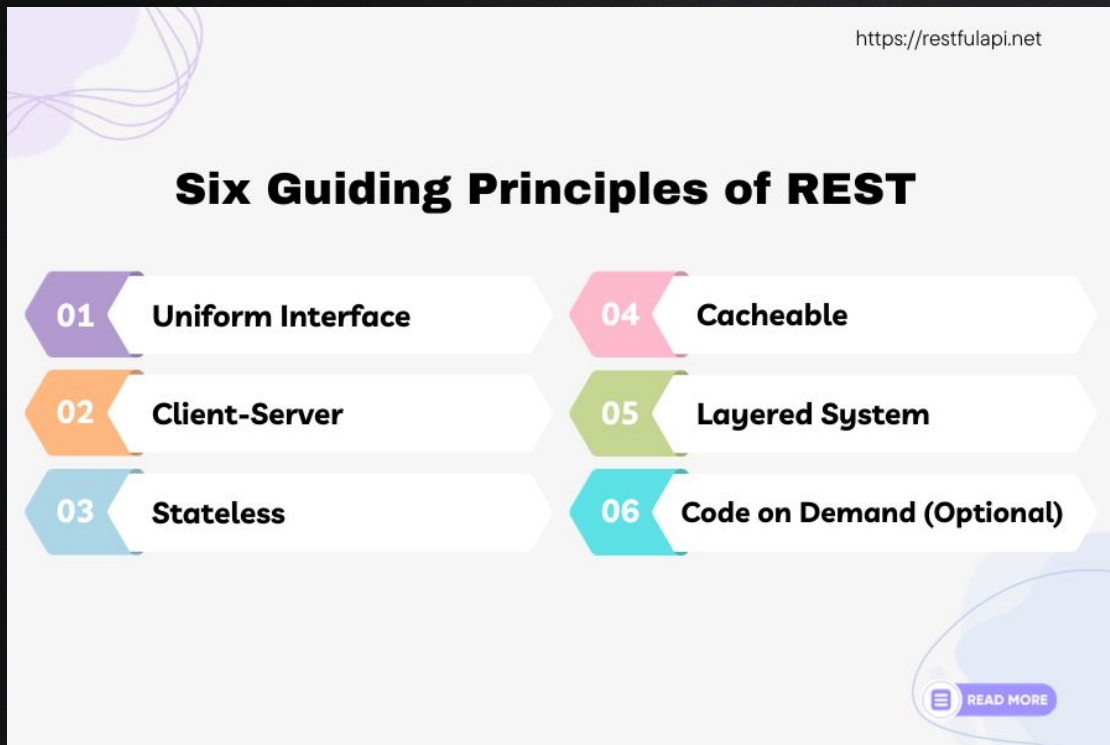
- 2XX: Requisição processada
- 3XX: Requisição com necessidade de correções ou modificações
- 4XX: Erros na requisição do cliente
- 5XX: Erros de servidor



API Restful | Conceitos

HTTP Method	CRUD	Collection Resource (e.g. /users)	Single Resource (e.g. /users/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID	Avoid using POST on a single resource
GET	Read	200 (OK), list of users. Use pagination, sorting, and filtering to navigate big lists	200 (OK), single user. 404 (Not Found), if ID not found or invalid
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution	200 (OK). 404 (Not Found), if ID not found or invalid

API Restful | Princípios



Annotations em Controllers

@RestController Anota a classe como um controller no padrão MVC para o Spring

@RequestMapping Anota o mapeamento para o controller a ser usado na URL da chamada

@GetMapping

Annotation para mapear uma requisição GET

@PostMapping

Annotation para mapear uma requisição POST

@PutMapping

Annotation para mapear uma requisição PUT

@DeleteMapping

Annotation para mapear uma requisição DELETE

@PatchMapping

Annotation para mapear uma requisição PATCH

@ ControllerAdvice	@ PutMapping
@ ControllerMappingReflectiveProcessor	@ RequestAttribute
@ CookieValue	@ RequestBody
@ CrossOrigin	@ RequestHeader
@ DeleteMapping	@ RequestMapping
@ ErrorHandler	@ RequestMethod
@ ErrorHandlerReflectiveProcessor	@ RequestParam
@ GetMapping	@ RequestPart
@ InitBinder	@.ResponseBody
@ Mapping	@ResponseStatus
@ MatrixVariable	@RestController
@ModelAttribute	@RestControllerAdvice
@ package-info	@SessionAttribute
@ PatchMapping	@SessionAttributes
@ PathVariable	@ValueConstants
@ PostMapping	

Padrão de Projeto DTO

Conceito do Data Transfer Object

Um objeto de transferência de dados é um padrão de design que permite a troca de dados entre subsistemas ou camadas de aplicativos de software. O principal objetivo do DTO é reduzir o número de chamadas de métodos entre essas camadas, agregando dados em um único objeto.

```
public class User {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    @Column(nullable = false)  
    private String firstName;  
    @Column(nullable = false)  
    private String lastName;  
    @Column(nullable = false, unique = true)  
    private String email;  
    @Column(nullable = false)  
    private String gender;  
}
```

```
public class UserDTO {  
  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private String email;  
    private String gender;  
}
```

A classe entidade USER e sua representação DTO

Java Record

Conceito do Java Record

Um Record, nada mais é que um tipo de classe que armazena dados. É a mesma ideia de construção similar a um JavaBean, possui construtor, atributos e métodos de acesso. Porém, ao invés de possibilitar qualquer alteração a classe é imutável.

Vantagens:

- Remoção de código padrão
- Redução de bibliotecas (lombok)
- Imutabilidade
- Sem métodos setter

```
public record UserRecord( 2 usages
    String id, no usages
    String firstName, no usages
    String lastName, no usages
    String email, no usages
    String gender no usages
){}

UserRecord user = new UserRecord( no usages
    id: "0",
    firstName: "fulano",
    lastName: "ciclano",
    email: "teste@teste.com",
    gender: "male"
);
```


Frameworks ORM

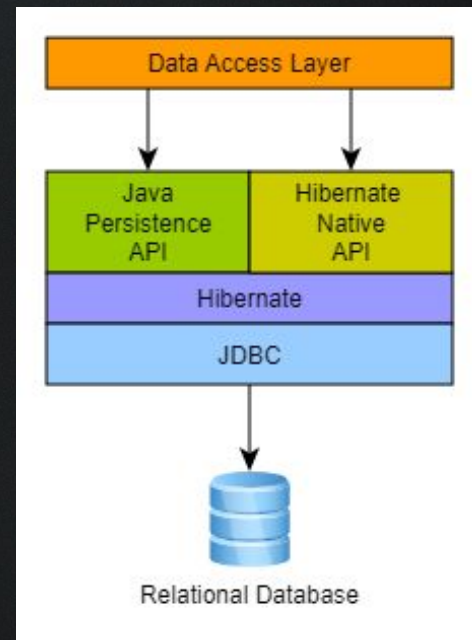
Object Relational Mapping

O Spring Framework suporta integração com Java Persistence API (JPA) e suporta Hibernate nativo para gerenciamento de recursos, implementações de objetos de acesso a dados (DAO) e estratégias de transação.

O ORM pode participar do gerenciamento de recursos e transações do Spring e cumprir as hierarquias genéricas de transações e exceções DAO do Spring. O estilo de integração recomendado é codificar DAOs em APIs simples de Hibernate ou JPA.

Vantagens do Uso do ORM

- Facilidade de testes com o uso de padrões de projeto como SessionFactory e DataSource, que já trazem abstrações para persistência de dados;
- Centralização dos tipos de exceções de dados com o DataAccessException;
- Manipulação de recursos mais fácil, através de classes específicas.



Arquitetura DAO, JPA, Hibernate, JDBC.

Fonte: <https://pw2.rpmhub.dev/topicos/jpa/hibernate.html>

Frameworks ORM

Hibernate

É uma implementação de persistência e manipulação de objetos relacionais. Surgiu antes do JPA, mas depois incorporou/implementou os padrões do JPA.

Java Persistence API (JPA)

É uma especificação para manipulação de objetos relacionais e persistentes. Surgiu depois do Hibernate, para padronizar a persistência de dados.

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class HibernateUserRepository {

    public void save(User user) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction transaction = session.beginTransaction();
        session.save(user);
        transaction.commit();
        session.close();
    }

    public User findById(Long id) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        User user = session.get(User.class, id);
        session.close();
        return user;
    }
}
```

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface SpringDataJpaUserRepository extends JpaRepository<User, Long> {

}
```

Duas classes de repositórios, uma escrita usando Hibernate e outra usando JPA. Fonte <https://medium.com/>

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;

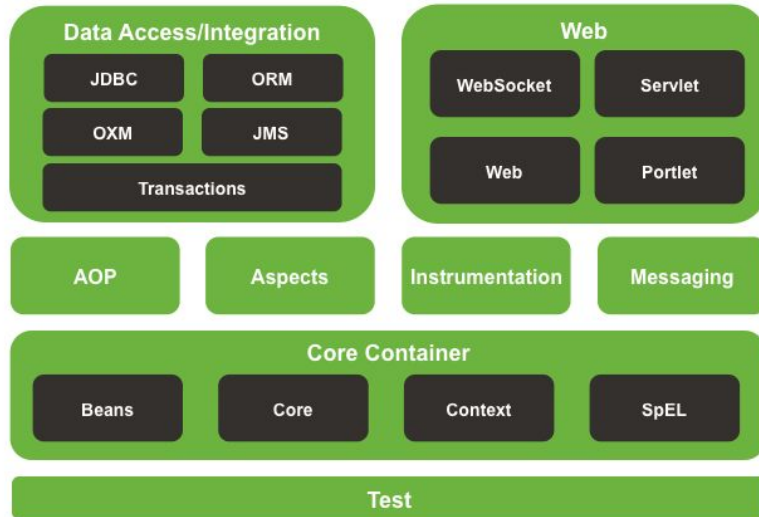


**ITS
CODE
TIME**

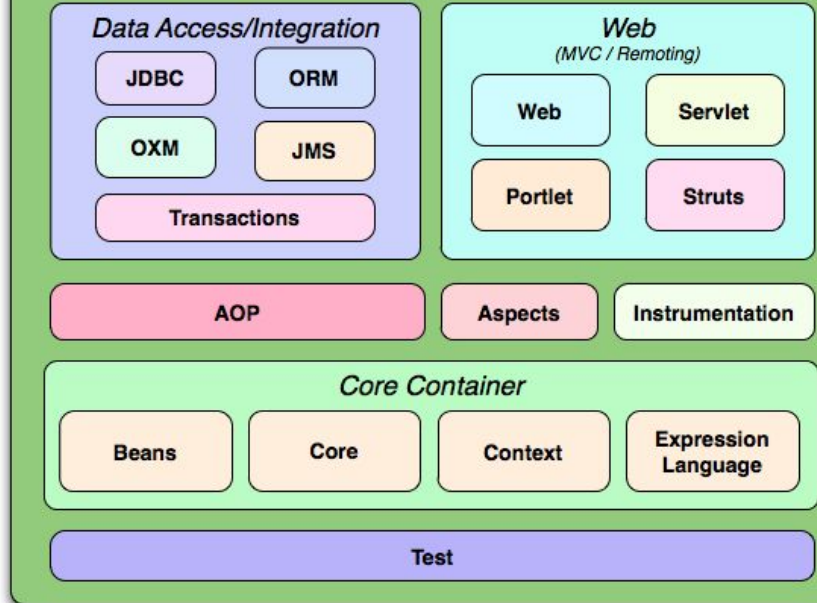
Framework Spring



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>

Tarefa para Casa

Leia o artigo para entender as diferenças entre SOAP e REST

<https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest>



Referências

Email do Professor

profmarcelo.hama@fiap.com.br

Bibliografias/Sites

<https://restfulapi.net>

- <https://docs.spring.io>

<https://hibernate.org/>

<https://docs.spring.io/spring-framework/reference/data-access/orm.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
