Curso

Aplicações JAVA com SPRING BOOT

Prof. Msc. Antonio B. C. Sampaio Jr
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos
@amazoncodebr

www.abctreinamentos.com.br
www.amazoncode.com.br

AULA
DE
HOJE

- UNIDADE 1 – INTRODUÇÃO
- UNIDADE 2 – FUNDAMENTOS DO SPRING BOOT
- UNIDADE 3 – PERSISTÊNCIA DE DADOS NO SPRING BOOT
- UNIDADE 4 – PROJETO WEB NO SPRING BOOT
- UNIDADE 5 – PROJETO REST API NO SPRING BOOT
- UNIDADE 6 – PROJETO REST API NO SPRING BOOT COM REACTJS

PROJETOS
DE
HOJE

- 1º Projeto Spring Boot – Impressão de Mensagens
- 2° Projeto Spring Boot – Impressão de Mensagens na WEB
- 3° Projeto Spring Boot – Aplicação Servidor Público
- 4° Projeto Spring Boot – Aplicação Servidor Público na WEB
- 5° Projeto Spring Boot – Aplicação Servidor Público no SGBD MYSQL
- 6° Projeto Spring Boot – Aplicação Servidor Público no MONGO DB

UNIDADE 3

PERSISTÊNCIA DE DADOS NO SPRING BOOT

PERSISTÊNCIA DE DADOS NO JAVA

SPRING DATA JPA

COMPONENTES SPRING DATA JPA

PROPRIEDADES DO APPLICATION.PROPERTIES

PROJETOS PRÁTICOS

5° Projeto Spring Boot – Aplicação Servidor Público no MySQL

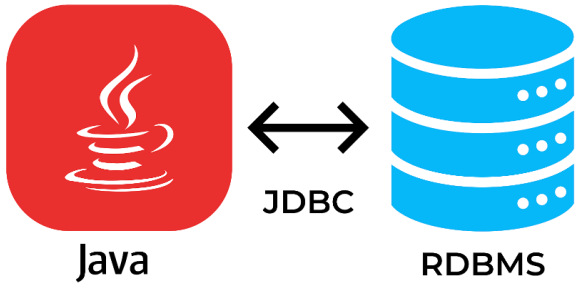
6° Projeto Spring Boot – Aplicação Servidor Público no MongoDB

PERSISTÊNCIA DE DADOS NO JAVA

PERSISTÊNCIA DE DADOS NO JAVA

- **API JDBC**
  - A API JDBC (*Java Database Connectivity*) fornece um conjunto de métodos para uma aplicação Java acessar e manipular bancos de dados relacionais.

  - Essa API oferece um conjunto de classes e interfaces para acessar bancos de dados e executar operações como consultas, inserções, atualizações e exclusões.



PERSISTÊNCIA DE DADOS NO JAVA

- **Código Java para Acessar BD com JDBC**

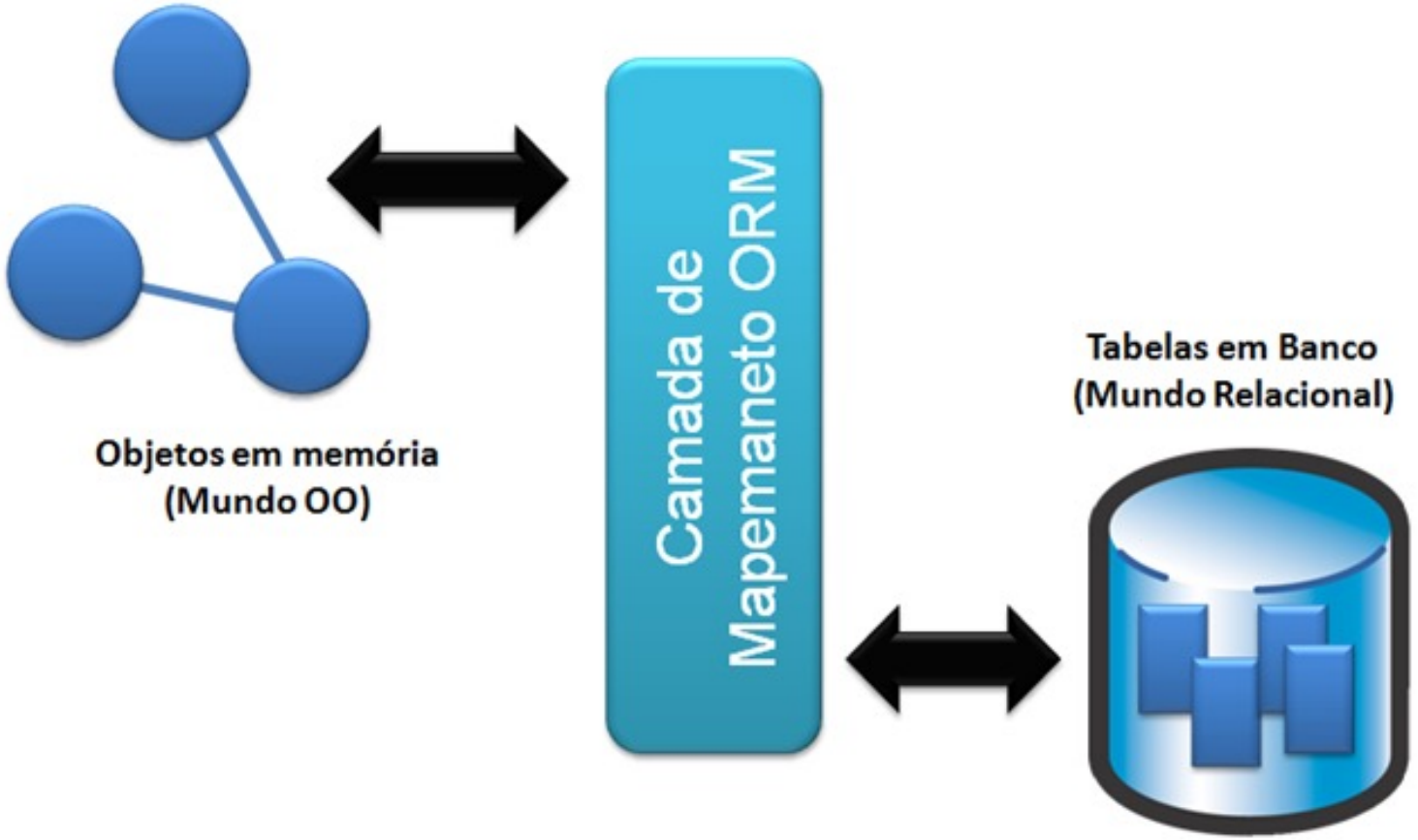
```
// Propriedades do BD
String url = "jdbc:oracle:thin:@localhost:1521:XE";
String usuario = "curso_java"; String senha = "schema";

//Conexão com o BD
Connection conexao = DriverManager.getConnection(url,usuario,senha);
Statement statement = conexao.createStatement();
//Consulta SQL
String consulta = "SELECT * FROM Cliente";
//Objeto de resultado da Consulta
ResultSet rs = statement.executeQuery(consulta);

//Resultado
while(rs.next()) {
     JOptionPane.showMessageDialog(null, "cpf:"+rs.getInt(1)+
       " nome:"+ rs.getString(2)+ " email"+ rs.getString(3));
}
//Fechar Conexão
conexao.close();
```

MODELO ORM

- **Definição**
  - ORM (*Object-Relational Mapping*) é um modelo de programação que permite aos desenvolvedores de software trabalharem com bancos de dados relacionais usando uma abordagem orientada a objetos. Com o ORM, o mapeamento entre as classes de objetos e as tabelas do banco de dados é feito automaticamente, sem a necessidade de escrever código SQL manualmente.

  - Ao invés de escrever SQL para recuperar e manipular dados, os desenvolvedores podem trabalhar com objetos em um nível mais alto de abstração, usando consultas e comandos de uma linguagem de programação orientada a objetos, como Java.

MODELO ORM

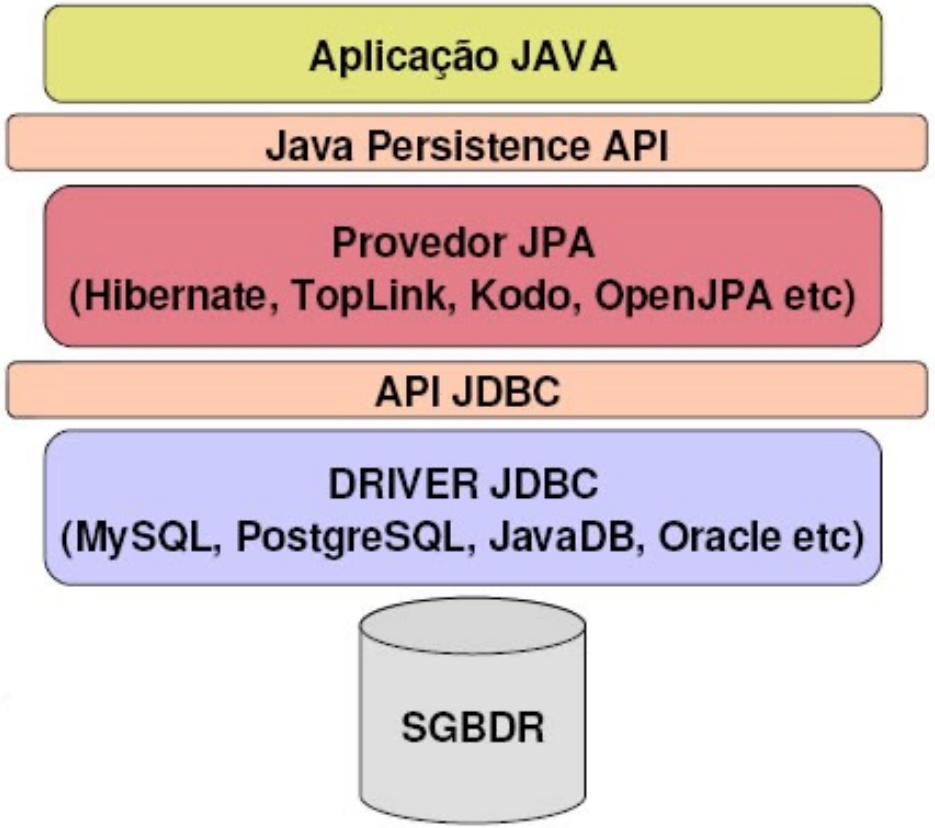

Objetos em memória
(Mundo OO)
Camada de
Mapemaneto ORM
Tabelas em Banco
(Mundo Relacional)

JAVA PERSISTENCE API (JPA)

- **Definição**
  - A JPA é uma especificação Java que descreve uma interface comum para frameworks de mapeamento objeto-relacional (ORM). Esta API define como as entidades Java são mapeadas para tabelas de um banco de dados relacional e fornece uma maneira fácil de realizar operações CRUD.

  - A JPA é uma abstração de nível superior em relação à JDBC API, permitindo aos desenvolvedores lidar com as operações de banco de dados de maneira mais orientada a objetos e menos dependente de detalhes de implementação específicos de um banco de dados.

JAVA PERSISTENCE API (JPA)



PERSISTÊNCIA DE DADOS NO JAVA

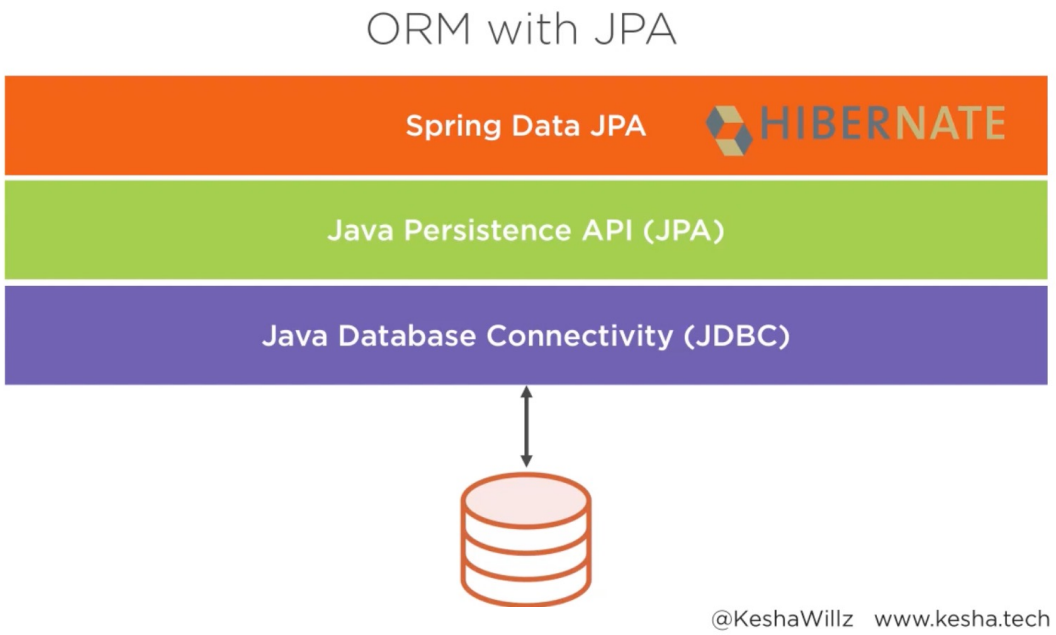
- **Código Java para Acessar BD com JPA**

```
public static void main(String[] args) {
     EntityManagerFactory emf =
         Persistence.createEntityManagerFactory("JPAApp");
     EntityManager em = emf.createEntityManager();
     EntityTransaction tx = em.getTransaction();
     tx.begin();
     //OPERAÇÕES DE CRUD
     tx.commit();
     em.close();
     emf.close();
}
```

SPRING DATA JPA

SPRING DATA JPA

- O Spring Data JPA é um projeto do Spring Framework que visa facilitar o desenvolvimento de aplicações que utilizam o JPA para acesso a banco de dados. Ele fornece uma camada de abstração sobre o JPA, permitindo que os desenvolvedores escrevam menos código repetitivo e mais focado nas regras de negócio da aplicação.



SPRING DATA JPA

- O Spring Data JPA permite que os desenvolvedores utilizem os recursos do JPA sem precisar escrever todo o código para gerenciamento de transações, criação de EntityManagers, etc. Além disso, o Spring Data JPA é facilmente integrado com outros projetos do Spring, como o Spring MVC e o Spring Boot.

- O Spring Data JPA oferece recursos como:
  - Criação automática de queries através do uso de convenções de nomenclatura
  - Suporte a consultas dinâmicas através do uso de *specifications*
  - Paginação de resultados
  - Suporte a consultas nativas do SQL
  - Eventos para auditoria, como @CreatedBy e @CreatedDate
  - Cache de resultados

SPRING DATA JPA

- Para utilização do Spring Data JPA em um projeto Spring Boot, faz-se necessário a utilização do **spring-boot-starter-data-jpa.**

- Com Spring Data JPA configurado em um projeto Spring Boot, é possível criar aplicativos que persistem dados de forma fácil e eficiente, sem precisar escrever muito código repetitivo.

SPRING DATA JPA

- **Dependências**
  - Além da inclusão da dependência do Spring Data, faz-se necessário a inclusão do banco de dados que será utilizado.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Spring Data

Hibernate, Spring Data JPA, Spring ORM

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

MySql

MongoDB

COMPONENTES SPRING DATA JPA

COMPONENTES SPRING DATA JPA

- Os principais componentes do Spring Data JPA são:
  - Entity: é a classe Java que representa uma tabela no banco de dados.
  - Repository: é uma interface que fornece métodos para acessar e manipular dados do banco de dados.
  - EntityManager: é uma classe responsável por gerenciar as entidades do JPA, como persistir, atualizar, recuperar e excluir.
  - Query: é uma interface que permite a criação de consultas personalizadas ao banco de dados.

COMPONENTES SPRING DATA JPA

- Os principais componentes do Spring Data JPA são:
  - Criteria API: é uma API do JPA que permite a construção de consultas de forma programática, sem a necessidade de escrever consultas em SQL.
  - Spring Data Commons: é uma biblioteca que fornece recursos comuns para todos os projetos do Spring Data, como a definição de interfaces de repositório, classes utilitárias, etc.

ENTITY

- As Entidades são classes Java que representam uma tabela em um SGBD relacional.

- Cada objeto dessa classe representa um registro da tabela.

```
@Entity
@Table(name="servidorpublico")
public class ServidorPublico
{
     @Id
     private Long matricula;
     private String nome;
     private String foto;
     private String orgao;
     ...
     //getters/setters
}
```

ENTITY

- A anotação @Entity é uma das anotações mais importantes do Spring Data JPA. Ela é usada para marcar uma classe como uma entidade de banco de dados, ou seja, uma classe que será mapeada para uma tabela em um banco de dados relacional.

- Quando uma classe é anotada com @Entity, ela deve ter um identificador que é marcado com a anotação @Id.

- A anotação @Id é uma anotação obrigatória e determina qual campo da entidade representa a chave primária da tabela no banco de dados.

- Caso a tabela possua um nome diferente, podemos estabelecer esse mapeamento com a anotação @Table, a qual será explorada em outra documentação.

ENTITY

```
@Entity
@Table(name="servidorpublico")
public class ServidorPublico
{
     @Id @GeneratedValue(strategy = GenerationType.AUTO)
     private Long matricula;
          ...
     //getters/setters
}
```

- A anotação @GeneratedValue é usada para especificar a estratégia de geração de valores para o identificador único. Neste caso, a estratégia é GenerationType.AUTO, o que significa que o provedor de persistência escolherá a melhor estratégia de geração de valores com base no banco de dados e no driver JDBC.

REPOSITORY

- Um repositório é uma interface que define um conjunto de operações que podem ser realizadas em uma entidade específica. Essas operações incluem a busca de entidades por identificador, a busca de entidades por propriedades, a contagem de entidades, a exclusão de entidades e a atualização de entidades.

- No Spring Data JPA, um Repository representa a Camada de Dados de uma Aplicação (similar ao padrão DAO) e oferece uma interface para a realização de todas as operações CRUD em tabelas/coleções de um Banco de Dados.

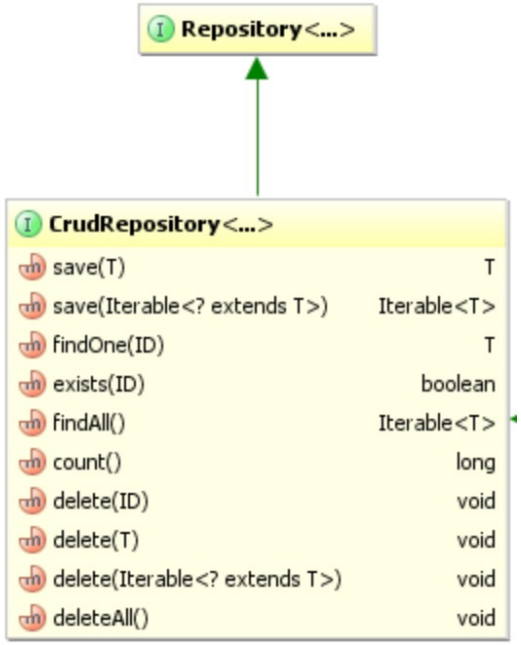
- Os repositórios são anotados com a anotação @Repository e estendem a interface JpaRepository ou outra interface de repositório fornecida pelo Spring Data JPA, dependendo da funcionalidade que você precisa.

REPOSITORY

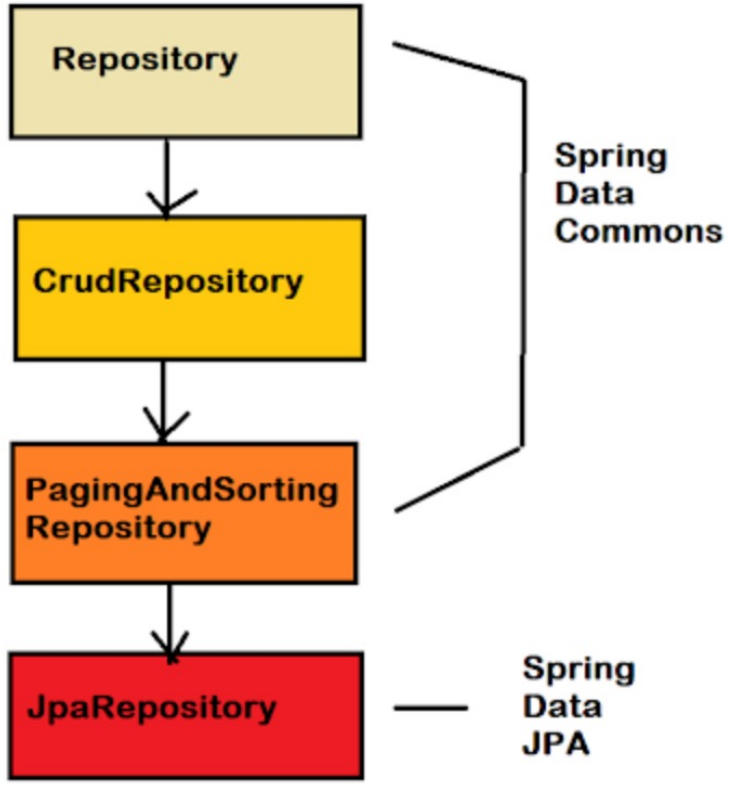
```
@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long>
{ }
```

- No Spring Data JPA, existem 03 Interfaces que podem ser utilizadas como Repository: CrudRepository, JpaRepository e PagingAndSortingRepository.
  - CrudRepository é aquela interface que oferece as operações básicas de CRUD.
  - JpaRepository é uma subinterface de CrudRepository que oferece operações adicionais além das básicas de CRUD.
  - PagingAndSortingRepository é outra subinterface de CrudRepository que oferece métodos para paginação e ordenação.

REPOSITORY



https://www.devmedia.com.br/persistencia-com-spring-data-jpa/24390



https://javarevisited.blogspot.com/2021/10/what-is-spring-data-repository.html#axzz81DKPDfj2

REPOSITORY

```
@Repository
public interface ServidorPublicoRepository
           extends CrudRepository<ServidorPublico, Long>
{
  ...
}
```

MySql

```
@Repository
public interface ServidorPublicoRepository
      extends MongoRepository<ServidorPublico, ObjectId>
{
    @Query("{ 'matricula' : ?0 }")
    Optional<ServidorPublico> findById(Long matricula);
}
```

MongoDB

PROPRIEDADES DO APPLICATIONS.PROPERTIES

APPLICATION.PROPERTIES

```
# Configuração do datasource
spring.datasource.url=jdbc:mysql://localhost:3306/meu_banco_de_dados

# Configurações do usuário e senha
spring.datasource.username=meu_usuario
spring.datasource.password=minha_senha

# Configurações do driver MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect

# Configuração da estratégia de criação e atualização do banco de dados
spring.jpa.hibernate.ddl-auto=update

# Configuração do pool de conexões
spring.datasource.hikari.maximum-pool-size=5
spring.datasource.hikari.idle-timeout=30000
```

APPLICATION.PROPERTIES

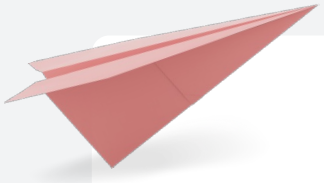
```
# Configuração do pool de conexões
spring.datasource.hikari.maximum-pool-size=5
spring.datasource.hikari.idle-timeout=30000
```

- Essas 02 propriedades definem as configurações de pool de conexões usando o HikariCP como padrão do Spring Boot:

  - spring.datasource.hikari.maximum-pool-size define o número máximo de conexões que devem estar no pool.

  - spring.datasource.hikari.idle-timeout define o tempo em milissegundos que uma conexão deve permanecer inativa antes de ser removida do pool.

PROJETOS PRÁTICOS



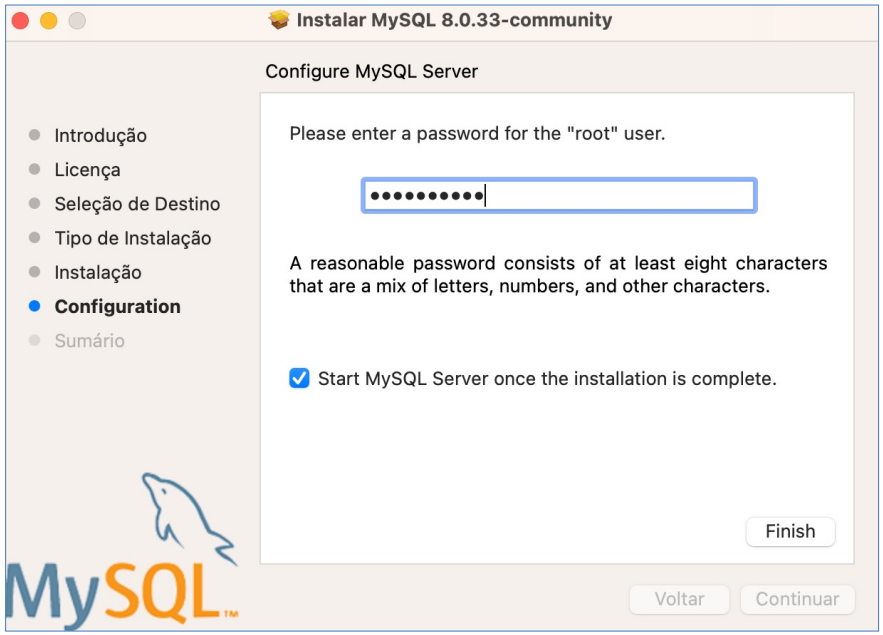
5° Projeto Spring Boot – Aplicação Servidor Público no MySQL

Passos:

- Instalar o SGBD MySQL
- Instalar o MySQL Workbench (Cliente)
- Criar o banco de dados “siscapacit”
- Criar a tabela ServidorPublico
- Criar o projeto com todas as suas dependências

Passos:

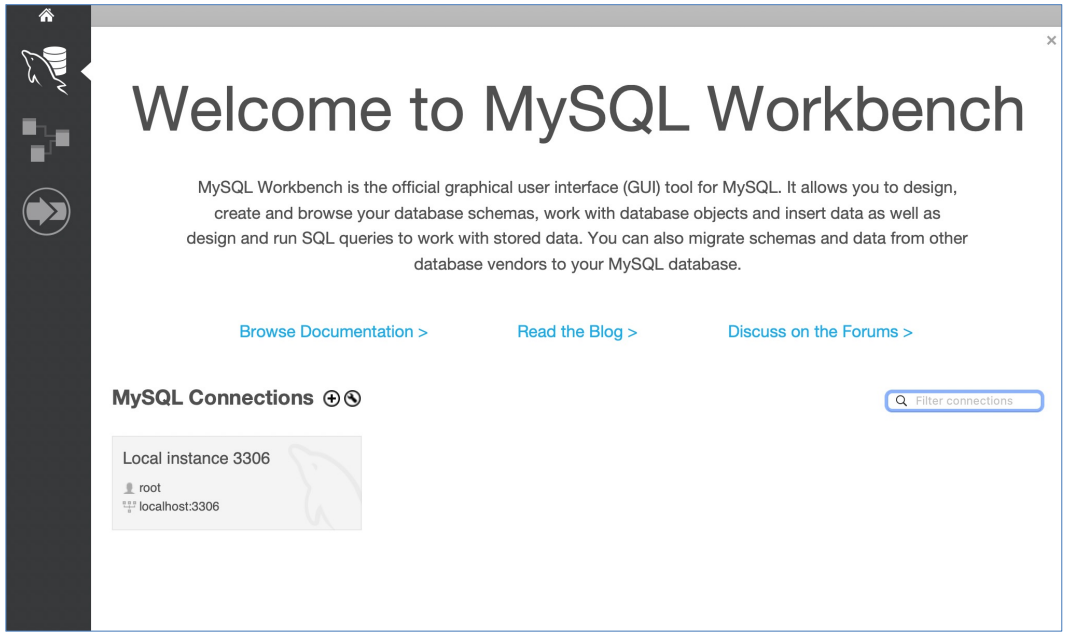
- Instalar o SGBD MySQL



https://dev.mysql.com/downloads/mysql/

Passos:

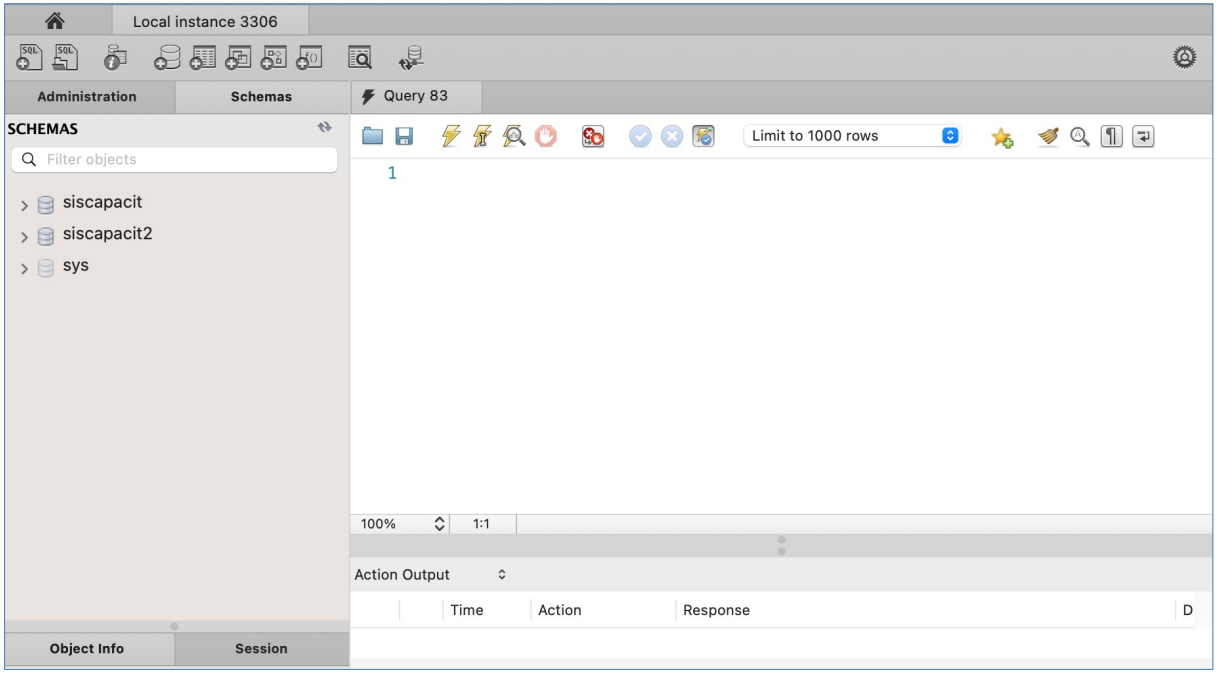
- Instalar o MySQL Workbench (Cliente)



https://dev.mysql.com/downloads/workbench/

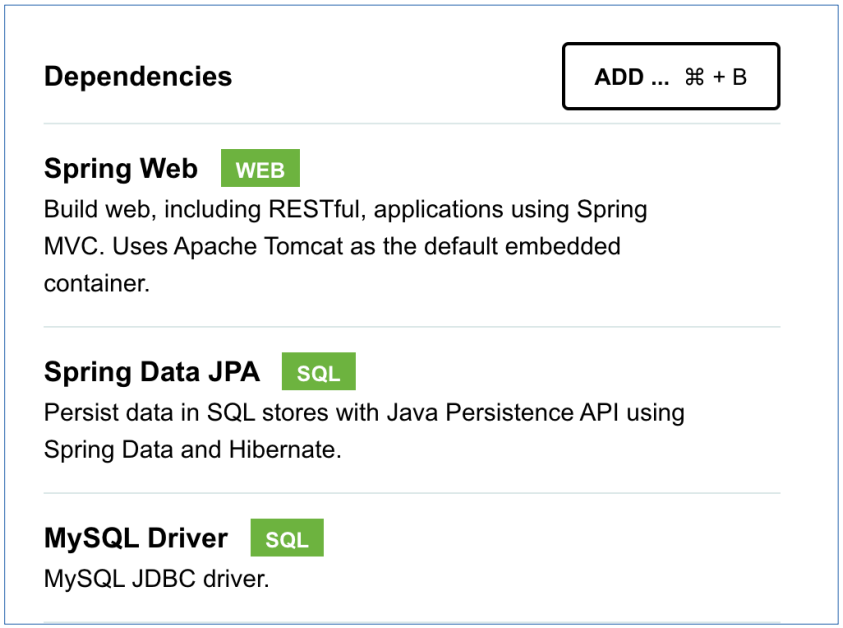
Passos:

- Criar a tabela ServidorPublico

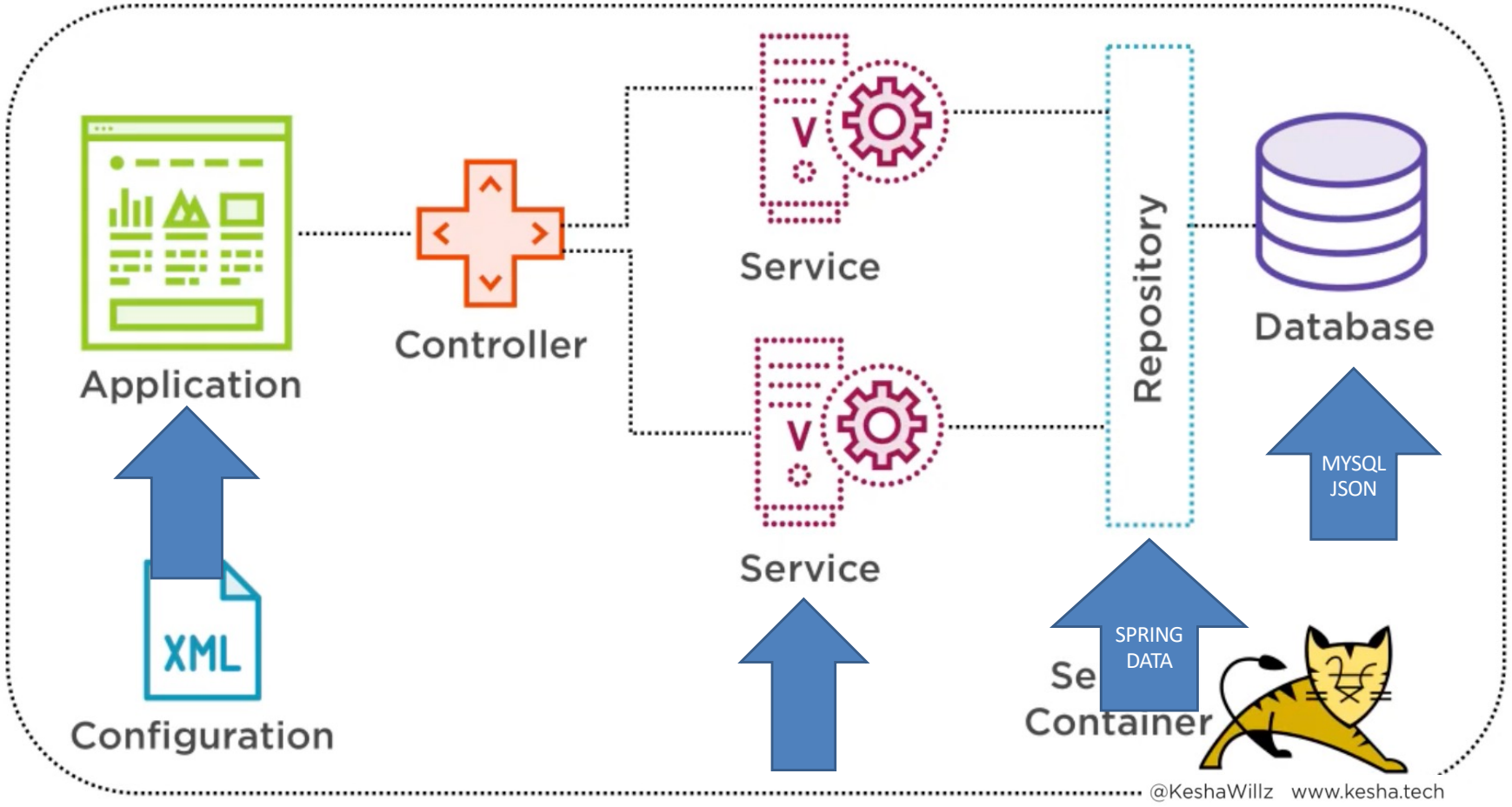


Passos:

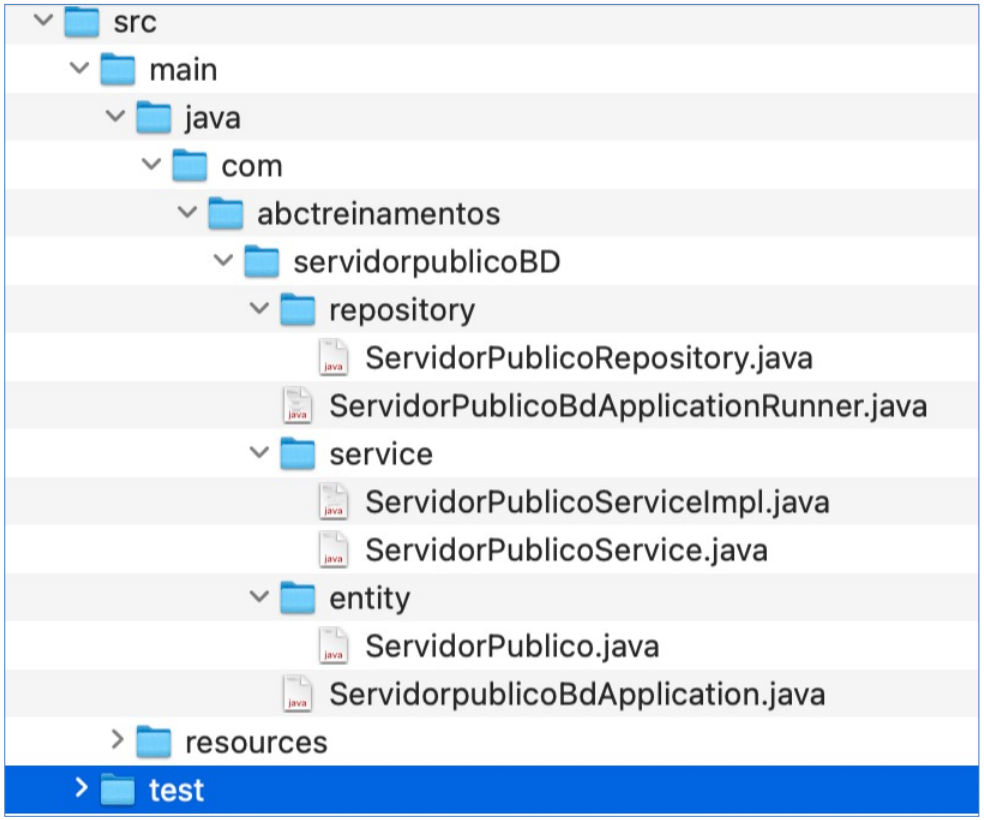
- Criar o projeto com todas as suas dependências

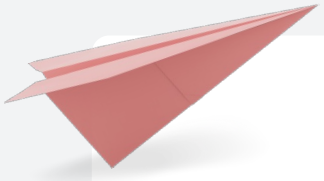


ARQUITETURA SPRING BOOT



ARQUITETURA DO PROJETO



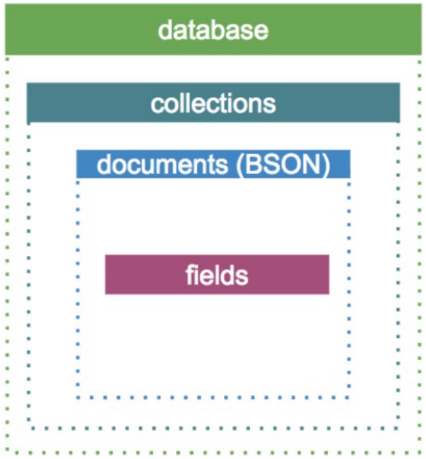


6° Projeto Spring Boot – Aplicação Servidor Público no MongoDB

MongoDB.

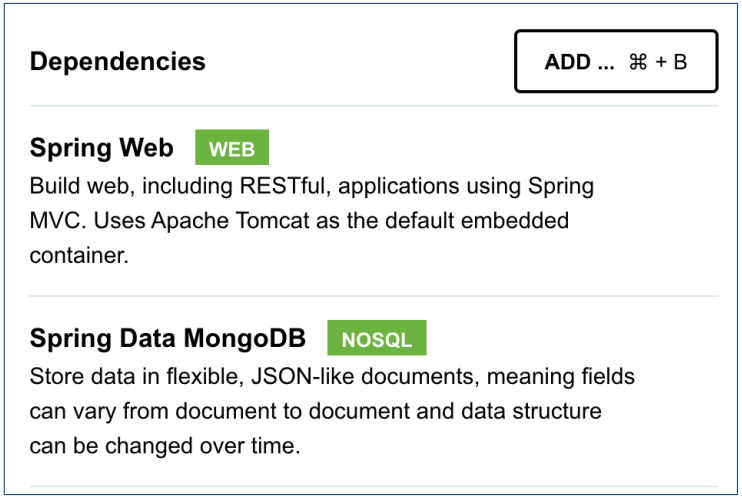
- É um banco de dados não relacional gratuito, *open source*, de alta performance e flexível, sendo considerado o principal banco de dados NoSQL.

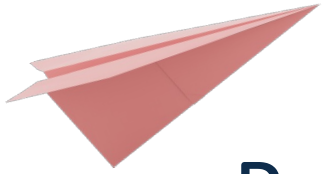
- O MongoDB é orientado a documentos, ou seja, os dados são armazenados como documentos, ao contrário de bancos de dados de modelo relacional, onde trabalhamos com registros em linhas e colunas. Os documentos podem ser descritos como dados no formato de chave-valor, no caso, utilizando o formato JSON.



MongoDB

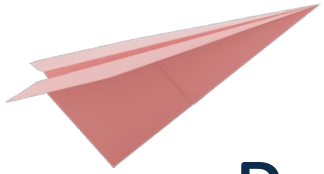
- Para utilização do MongoDB no Spring Boot, faz-se necessário utilizar a dependência spring-boot-starter-data-mongodb para configurar o Spring Data MongoDB e o seu driver MongoDB.





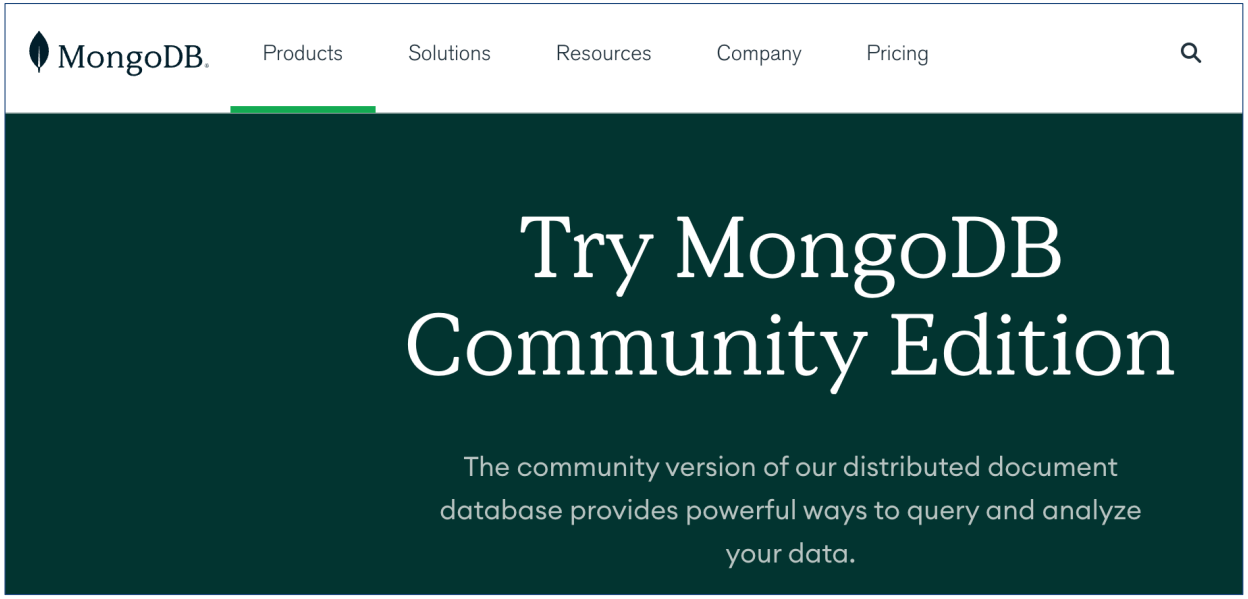
Passos:

- Instalar o SGBD MongoDB
- Instalar o MongoDB Compass (Cliente)
- Utilizar a base de dados “test”
- Criar a coleção ServidorPublico
- Criar o projeto com todas as suas dependências



Passos:

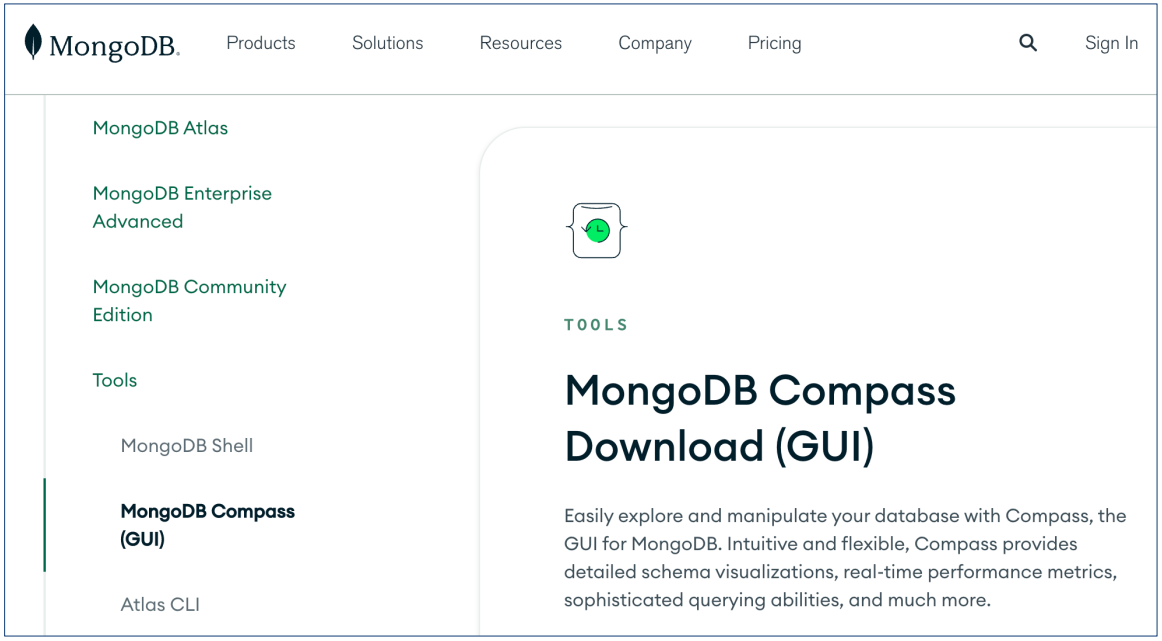
- Instalar o SGBD MongoDB



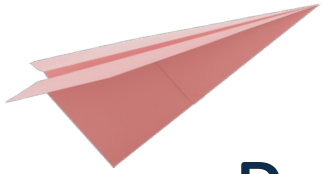
https://www.mongodb.com/try/download/community

Passos:

- Instalar o MongoDB Compass (Cliente)

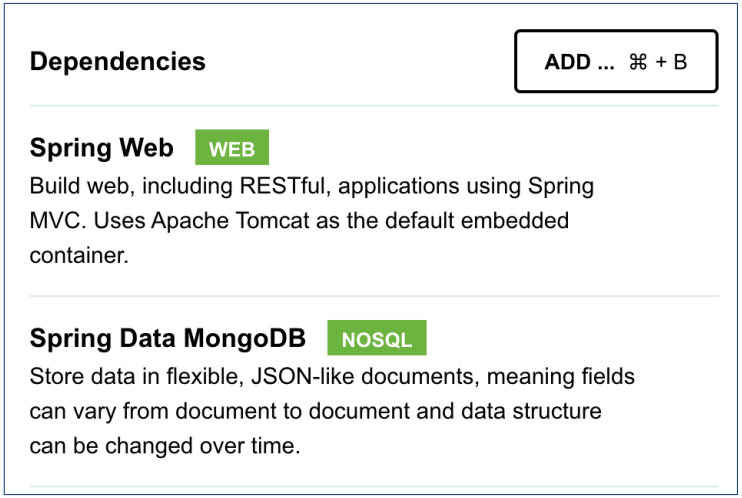


https://www.mongodb.com/try/download/compass



Passos:

- Criar o projeto com todas as suas dependências



ARQUITETURA DO PROJETO