

ⓘ This shared conversation is generated by AI, for reference only.

## Exemplo de auto complete em primefaces 14, java 17, Jakarta, Tomcat 11

Aqui está um exemplo completo de implementação do componente AutoComplete no PrimeFaces 14 com Java 17, Jakarta EE e Tomcat 11:

### 1. Estrutura do Projeto

text

```
src/
├─ main/
│   └─ java/
│       └─ com/
│           └─ exemplo/
│               └─ model/
│                   └─ Cidade.java
│               └─ controller/
│                   └─ AutoCompleteBean.java
│               └─ service/
│                   └─ CidadeService.java
└─ webapp/
    └─ WEB-INF/
        └─ web.xml
    └─ autocomplete.xhtml
```

### 2. Dependências Maven (pom.xml)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.exemplo</groupId>
  <artifactId>primefaces-autocomplete</artifactId>
  <version>1.0-SNAPSHOT</version>
```

```
<packaging>war</packaging>

<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <primefaces.version>14.0.0</primefaces.version>
</properties>

<dependencies>
  <!-- PrimeFaces -->
  <dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>${primefaces.version}</version>
    <classifier>jakarta</classifier>
  </dependency>

  <!-- Jakarta EE -->
  <dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>10.0.0</version>
    <scope>provided</scope>
  </dependency>

  <!-- CDI -->
  <dependency>
    <groupId>jakarta.enterprise</groupId>
    <artifactId>jakarta.enterprise.cdi-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>

  <!-- JSF -->
  <dependency>
    <groupId>jakarta.faces</groupId>
    <artifactId>jakarta.faces-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>
</dependencies>

<build>
  <finalName>primefaces-autocomplete</finalName>
  <plugins>
    <plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<version>3.4.0</version>
<configuration>
  <failOnMissingWebXml>>false</failOnMissingWebXml>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

### 3. Model (Cidade.java)

java

```
package com.exemplo.model;

import java.io.Serializable;
import java.util.Objects;

public class Cidade implements Serializable {

    private Integer id;
    private String nome;
    private String estado;

    public Cidade() {}

    public Cidade(Integer id, String nome, String estado) {
        this.id = id;
        this.nome = nome;
        this.estado = estado;
    }

    // Getters e Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }
}
```

```
public void setName(String nome) {
    this.nome = nome;
}

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

@Override
public String toString() {
    return nome + " - " + estado;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Cidade cidade = (Cidade) obj;
    return Objects.equals(id, cidade.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
}
```

## 4. Service (CidadeService.java)

java

```
package com.exemplo.service;

import com.exemplo.model.Cidade;
import jakarta.enterprise.context.ApplicationScoped;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@ApplicationScoped
```

```

public class CidadeService implements Serializable {

    private List<Cidade> cidades;

    public CidadeService() {
        inicializarDados();
    }

    private void inicializarDados() {
        cidades = new ArrayList<>();
        cidades.add(new Cidade(1, "São Paulo", "SP"));
        cidades.add(new Cidade(2, "Rio de Janeiro", "RJ"));
        cidades.add(new Cidade(3, "Belo Horizonte", "MG"));
        cidades.add(new Cidade(4, "Brasília", "DF"));
        cidades.add(new Cidade(5, "Salvador", "BA"));
        cidades.add(new Cidade(6, "Fortaleza", "CE"));
        cidades.add(new Cidade(7, "Curitiba", "PR"));
        cidades.add(new Cidade(8, "Manaus", "AM"));
        cidades.add(new Cidade(9, "Recife", "PE"));
        cidades.add(new Cidade(10, "Porto Alegre", "RS"));
        cidades.add(new Cidade(11, "Belém", "PA"));
        cidades.add(new Cidade(12, "Goiânia", "GO"));
        cidades.add(new Cidade(13, "Guarulhos", "SP"));
        cidades.add(new Cidade(14, "Campinas", "SP"));
        cidades.add(new Cidade(15, "São Luís", "MA"));
    }

    public List<Cidade> getCidades() {
        return cidades;
    }

    public List<Cidade> pesquisarCidades(String query) {
        String queryLower = query.toLowerCase();

        return cidades.stream()
            .filter(cidade -> cidade.getNome().toLowerCase().contains(queryLower) ||
                cidade.getEstado().toLowerCase().contains(queryLower))
            .limit(10)
            .collect(Collectors.toList());
    }
}

```

## 5. Controller (AutoCompleteBean.java)

java

```
package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

    private Cidade cidadeSelecionada;
    private String textoPesquisa;
    private List<Cidade> cidadesFiltradas;
    private String valorSimples;

    public List<Cidade> completarCidade(String query) {
        return cidadeService.pesquisarCidades(query);
    }

    public void aoSelecionarCidade() {
        if (cidadeSelecionada != null) {
            FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage("Cidade selecionada",
                    cidadeSelecionada.toString()));
        }
    }

    public void aoSelecionarValor() {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Valor selecionado: " + valorSimples));
    }

    public void limpar() {
        cidadeSelecionada = null;
        textoPesquisa = null;
        valorSimples = null;
    }
}
```

```
}

// Getters e Setters
public Cidade getCidadeSelecionada() {
    return cidadeSelecionada;
}

public void setCidadeSelecionada(Cidade cidadeSelecionada) {
    this.cidadeSelecionada = cidadeSelecionada;
}

public String getTextoPesquisa() {
    return textoPesquisa;
}

public void setTextoPesquisa(String textoPesquisa) {
    this.textoPesquisa = textoPesquisa;
}

public List<Cidade> getCidadesFiltradas() {
    return cidadesFiltradas;
}

public void setCidadesFiltradas(List<Cidade> cidadesFiltradas) {
    this.cidadesFiltradas = cidadesFiltradas;
}

public String getValorSimples() {
    return valorSimples;
}

public void setValorSimples(String valorSimples) {
    this.valorSimples = valorSimples;
}
}
```

## 6. Página XHTML (autocomplete.xhtml)

xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:p="primefaces">
```

```
<h:head>
    <title>Exemplo AutoComplete - PrimeFaces 14</title>
</h:head>

<h:body>
    <h:form>
        <p:growl id="growl" showDetail="true" life="3000"/>

        <p:panel header="Exemplos de AutoComplete" style="margin:20px">

            <h:panelGrid columns="3" cellpadding="10">

                <!-- AutoComplete Básico com Objeto -->
                <p:outputLabel for="basic" value="Básico (Objeto):"/>
                <p:autoComplete id="basic"
                    value="#{autoCompleteBean.cidadeSelecionada}"
                    completeMethod="#{autoCompleteBean.completarCidade}"
                    var="cidade"
                    itemLabel="#{cidade.nome} - #{cidade.estado}"
                    itemValue="#{cidade}"
                    converter="cidadeConverter"
                    forceSelection="true"
                    dropdown="true"
                    minQueryLength="2"
                    queryDelay="500"
                    placeholder="Digite uma cidade..."
                <p:ajax event="itemSelect"
                    listener="#{autoCompleteBean.aoSelecionarCidade}"
                    update="growl"/>
                </p:autoComplete>
                <h:outputText value="#{autoCompleteBean.cidadeSelecionada}" />

                <!-- AutoComplete com Template Personalizado -->
                <p:outputLabel for="template" value="Com Template: "/>
                <p:autoComplete id="template"
                    value="#{autoCompleteBean.cidadeSelecionada}"
                    completeMethod="#{autoCompleteBean.completarCidade}"
                    var="cidade"
                    itemValue="#{cidade}"
                    converter="cidadeConverter"
                    forceSelection="true"
                    placeholder="Buscar cidade..."
                    panelStyle="width:300px">

                    <p:ajax event="itemSelect"
                        listener="#{autoCompleteBean.aoSelecionarCidade}"
```



```

        update="growl"/>

        <f:facet name="item">
            <h:panelGrid columns="2" style="width:100%">
                <h:outputText value="#{cidade.nome}" style="font-weight:
t:bold"/>

                <h:outputText value="#{cidade.estado}" style="color:#66
6"/>

            </h:panelGrid>
        </f:facet>
    </p:autoComplete>

    <!-- AutoComplete para Valores Simples -->
    <p:outputLabel for="simple" value="Valor Simples: "/>
    <p:autoComplete id="simple"
        value="#{autoCompleteBean.valorSimples}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"
        itemLabel="#{cidade.nome}"
        itemValue="#{cidade.nome}"
        forceSelection="true"
        placeholder="Digite o nome da cidade...">
        <p:ajax event="itemSelect"
            listener="#{autoCompleteBean.aoSelecionarValor}"
            update="growl"/>
    </p:autoComplete>

    <!-- AutoComplete com Múltipla Seleção -->
    <p:outputLabel for="multiple" value="Múltipla Seleção: "/>
    <p:autoComplete id="multiple"
        value="#{autoCompleteBean.cidadesFiltradas}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"
        itemLabel="#{cidade.nome} - #{cidade.estado}"
        itemValue="#{cidade}"
        converter="cidadeConverter"
        multiple="true"
        forceSelection="true"
        placeholder="Selecione várias cidades...">
    </p:autoComplete>

    <!-- AutoComplete com Grupo -->
    <p:outputLabel for="group" value="Agrupado: "/>
    <p:autoComplete id="group"
        value="#{autoCompleteBean.cidadeSelecionada}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"

```

```

        itemLabel="#{cidade.nome}"
        itemValue="#{cidade}"
        converter="cidadeConverter"
        groupBy="#{cidade.estado}"
        placeholder="Cidades agrupadas por estado..."
    </p:autoComplete>

</h:panelGrid>

<p:separator/>

<p:commandButton value="Limpar"
    actionListener="#{autoCompleteBean.limpar}"
    update="@form"
    icon="pi pi-refresh"
    style="margin-top:10px"/>

<p:commandButton value="Mostrar Selecionado"
    update="growl"
    icon="pi pi-check"
    style="margin-top:10px; margin-left:10px"
    actionListener="#{facesContext.addMessage(null,
        facesMessage['Cidade selecionada: ' + autoCompleteBe
an.cidadeSelecionada])}"/>
</p:panel>
</h:form>
</h:body>
</html>

```

## 7. Converter (CidadeConverter.java)

```

java

package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.util.Optional;

```

```

@Named
@RequestScoped
@FacesConverter(value = "cidadeConverter", managed = true)
public class CidadeConverter implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Integer id = Integer.parseInt(value);
            Optional<Cidade> cidade = cidadeService.getCidades().stream()
                .filter(c -> c.getId().equals(id))
                .findFirst();

            return cidade.orElse(null);
        } catch (NumberFormatException e) {
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
        if (cidade == null) {
            return "";
        }

        return String.valueOf(cidade.getId());
    }
}

```

## 8. Configuração (web.xml)

xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee

```

```

https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
version="6.0">

<display-name>PrimeFaces 14 AutoComplete Example</display-name>

<!-- Configuração JSF -->
<context-param>
    <param-name>jakarta.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
</context-param>

<context-param>
    <param-name>jakarta.faces.FACELETS_SKIP_COMMENTS</param-name>
    <param-value>true</param-value>
</context-param>

<!-- PrimeFaces Theme -->
<context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>saga</param-value>
</context-param>

<!-- Servlet JSF -->
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>autocomplete.xhtml</welcome-file>
</welcome-file-list>

</web-app>

```

## 9. beans.xml (para CDI)

xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="https://jakarta.ee/xml/ns/jakartaee"

```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/beans_4_0.xsd"
version="4.0"
bean-discovery-mode="annotated">
</beans>
```

## Principais características implementadas:

1. **AutoComplete básico** com seleção de objetos
2. **Template personalizado** para exibição dos resultados
3. **Valores simples** (String)
4. **Múltipla seleção** de itens
5. **Agrupamento** por estado
6. **Converter** personalizado para objetos complexos
7. **Eventos AJAX** ao selecionar itens
8. **Validação** com forceSelection
9. **Dropdown** para visualizar todas as opções
10. **Delay na pesquisa** para melhor performance

## Como executar:

1. Clone o projeto
2. Execute `mvn clean package`
3. Copie o arquivo WAR gerado para a pasta `webapps` do Tomcat 11
4. Inicie o Tomcat
5. Acesse: `http://localhost:8080/primefaces-autocomplete/autocomplete.xhtml`

O exemplo demonstra as principais funcionalidades do componente AutoComplete do PrimeFaces 14, compatível com Java 17, Jakarta EE e Tomcat 11.

Inclua no exemplo um faces converter

Aqui está o exemplo completo incluindo o **FacesConverter** para o AutoComplete do PrimeFaces 14:

# 1. Model (Cidade.java) - Atualizado

```
java

package com.exemplo.model;

import java.io.Serializable;
import java.util.Objects;

public class Cidade implements Serializable {

    private Integer id;
    private String nome;
    private String estado;
    private Integer populacao;
    private String regiao;

    public Cidade() {}

    public Cidade(Integer id, String nome, String estado, Integer populacao, String
regiao) {
        this.id = id;
        this.nome = nome;
        this.estado = estado;
        this.populacao = populacao;
        this.regiao = regiao;
    }

    // Getters e Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public String getEstado() {
        return estado;
    }
```

```
}

public void setEstado(String estado) {
    this.estado = estado;
}

public Integer getPopulacao() {
    return populacao;
}

public void setPopulacao(Integer populacao) {
    this.populacao = populacao;
}

public String getRegiao() {
    return regiao;
}

public void setRegiao(String regiao) {
    this.regiao = regiao;
}

public String getNomeCompleto() {
    return nome + " - " + estado + " (" + regiao + ")";
}

@Override
public String toString() {
    return nome + " - " + estado;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Cidade cidade = (Cidade) obj;
    return Objects.equals(id, cidade.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
}
```

## 2 Service (CidadeService.java) - Atualizado

```
java
```

```
package com.exemplo.service;

import com.exemplo.model.Cidade;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Named
@ApplicationScoped
public class CidadeService implements Serializable {

    private List<Cidade> cidades;

    public CidadeService() {
        inicializarDados();
    }

    private void inicializarDados() {
        cidades = new ArrayList<>();
        cidades.add(new Cidade(1, "São Paulo", "SP", 12300000, "Sudeste"));
        cidades.add(new Cidade(2, "Rio de Janeiro", "RJ", 6748000, "Sudeste"));
        cidades.add(new Cidade(3, "Belo Horizonte", "MG", 2523000, "Sudeste"));
        cidades.add(new Cidade(4, "Brasília", "DF", 3055000, "Centro-Oeste"));
        cidades.add(new Cidade(5, "Salvador", "BA", 2887000, "Nordeste"));
        cidades.add(new Cidade(6, "Fortaleza", "CE", 2687000, "Nordeste"));
        cidades.add(new Cidade(7, "Curitiba", "PR", 1963000, "Sul"));
        cidades.add(new Cidade(8, "Manaus", "AM", 2219000, "Norte"));
        cidades.add(new Cidade(9, "Recife", "PE", 1655000, "Nordeste"));
        cidades.add(new Cidade(10, "Porto Alegre", "RS", 1488000, "Sul"));
        cidades.add(new Cidade(11, "Belém", "PA", 1499000, "Norte"));
        cidades.add(new Cidade(12, "Goiânia", "GO", 1536000, "Centro-Oeste"));
        cidades.add(new Cidade(13, "Guarulhos", "SP", 1392000, "Sudeste"));
        cidades.add(new Cidade(14, "Campinas", "SP", 1214000, "Sudeste"));
        cidades.add(new Cidade(15, "São Luís", "MA", 1109000, "Nordeste"));
        cidades.add(new Cidade(16, "Maceió", "AL", 1025000, "Nordeste"));
        cidades.add(new Cidade(17, "Campo Grande", "MS", 906000, "Centro-Oeste"));
        cidades.add(new Cidade(18, "Natal", "RN", 890000, "Nordeste"));
        cidades.add(new Cidade(19, "Teresina", "PI", 868000, "Nordeste"));
        cidades.add(new Cidade(20, "João Pessoa", "PB", 817000, "Nordeste"));
    }

    public List<Cidade> getCidades() {
```



```

        return cidades;
    }

    public List<Cidade> pesquisarCidades(String query) {
        String queryLower = query.toLowerCase();

        return cidades.stream()
            .filter(cidade -> cidade.getNome().toLowerCase().contains(queryLower) ||
                cidade.getEstado().toLowerCase().contains(queryLower) ||
                cidade.getRegiao().toLowerCase().contains(queryLower))
            .limit(10)
            .collect(Collectors.toList());
    }

    public Cidade encontrarPorId(Integer id) {
        return cidades.stream()
            .filter(c -> c.getId().equals(id))
            .findFirst()
            .orElse(null);
    }
}

```

### 3. FacesConverter (CidadeConverter.java) - 3 VERSÕES DIFERENTES

#### Versão 1: Converter com CDI (Recomendado para PrimeFaces 14 + Jakarta)

```

java

package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;

```

```

@Named
@FacesConverter(value = "cidadeConverter", managed = true)
public class CidadeConverter implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            // Tenta converter o ID
            Integer id = Integer.parseInt(value);
            return cidadeService.encontrarPorId(id);
        } catch (NumberFormatException e) {
            // Se não for um número, tenta encontrar pelo toString
            return cidadeService.getCidades().stream()
                .filter(c -> c.toString().equals(value))
                .findFirst()
                .orElse(null);
        }
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
        if (cidade == null || cidade.getId() == null) {
            return "";
        }
        return String.valueOf(cidade.getId());
    }
}

```

## Versão 2: Converter com @FacesConverter (Sem CDI)

```

java

package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;

```

```
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import java.util.List;

@FacesConverter("cidadeConverterSimples")
public class CidadeConverterSimples implements Converter<Cidade> {

    private CidadeService getCidadeService() {
        FacesContext context = FacesContext.getCurrentInstance();
        return context.getApplication().evaluateExpressionGet(context, "#{cidadeService}", CidadeService.class);
    }

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Integer id = Integer.parseInt(value);
            return getCidadeService().encontrarPorId(id);
        } catch (NumberFormatException e) {
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
        if (cidade == null || cidade.getId() == null) {
            return "";
        }
        return String.valueOf(cidade.getId());
    }
}
```

## Versão 3: Converter com Map (Mais eficiente para muitos dados)

```
java

package com.exemplo.converter;

import com.exemplo.model.Cidade;
```

```
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.util.HashMap;
import java.util.Map;

@Named
@FacesConverter(value = "cidadeConverterMap", managed = true)
public class CidadeConverterMap implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    private Map<Integer, Cidade> cache = new HashMap<>();

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Integer id = Integer.parseInt(value);

            // Verifica se já está em cache
            if (cache.containsKey(id)) {
                return cache.get(id);
            }

            // Busca e adiciona ao cache
            Cidade cidade = cidadeService.encontrarPorId(id);
            if (cidade != null) {
                cache.put(id, cidade);
            }

            return cidade;
        } catch (NumberFormatException e) {
            return null;
        }
    }

    @Override
```

```

    public String getAsString(FacesContext context, UIComponent component, Cidade c
idade) {
        if (cidade == null || cidade.getId() == null) {
            return "";
        }

        // Adiciona ao cache se não existir
        if (!cache.containsKey(cidade.getId())) {
            cache.put(cidade.getId(), cidade);
        }

        return String.valueOf(cidade.getId());
    }
}

```

## 4. Controller (AutoCompleteBean.java) - Atualizado

java

```

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

    private Cidade cidadeSelecionada;
    private Cidade cidadeSelecionada2;
    private Cidade cidadeSelecionada3;
    private String valorSimples;
    private List<Cidade> cidadesMultiplas;
    private List<Cidade> cidadesMultiplas2;

```

```
public AutoCompleteBean() {
    cidadesMultiplas = new ArrayList<>();
    cidadesMultiplas2 = new ArrayList<>();
}

// Método para o autoComplete básico
public List<Cidade> completarCidade(String query) {
    return cidadeService.pesquisarCidades(query);
}

// Método específico para o converter com Map
public List<Cidade> completarCidadeMap(String query) {
    return cidadeService.pesquisarCidades(query);
}

// Evento ao selecionar cidade
public void aoSelecionarCidade() {
    if (cidadeSelecionada != null) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Cidade selecionada (converter padrão)",
                cidadeSelecionada.getNomeCompleto()));
    }
}

public void aoSelecionarCidade2() {
    if (cidadeSelecionada2 != null) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Cidade selecionada (converter simples)",
                cidadeSelecionada2.getNomeCompleto()));
    }
}

public void aoSelecionarCidade3() {
    if (cidadeSelecionada3 != null) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Cidade selecionada (converter com Map)",
                cidadeSelecionada3.getNomeCompleto() + " - Pop: " +
                cidadeSelecionada3.getPopulacao()));
    }
}

public void aoSelecionarValor() {
    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage("Valor simples selecionado: " + valorSimples));
}

public void mostrarMultiplas() {
```

```
StringBuilder sb = new StringBuilder("Cidades selecionadas: ");
for (Cidade c : cidadesMultiplas) {
    sb.append(c.getNome()).append(", ");
}
FacesContext.getCurrentInstance().addMessage(null,
    new FacesMessage(sb.toString()));
}

public void limpar() {
    cidadeSelecionada = null;
    cidadeSelecionada2 = null;
    cidadeSelecionada3 = null;
    valorSimples = null;
    cidadesMultiplas.clear();
    cidadesMultiplas2.clear();
}

// Getters e Setters
public Cidade getCidadeSelecionada() {
    return cidadeSelecionada;
}

public void setCidadeSelecionada(Cidade cidadeSelecionada) {
    this.cidadeSelecionada = cidadeSelecionada;
}

public Cidade getCidadeSelecionada2() {
    return cidadeSelecionada2;
}

public void setCidadeSelecionada2(Cidade cidadeSelecionada2) {
    this.cidadeSelecionada2 = cidadeSelecionada2;
}

public Cidade getCidadeSelecionada3() {
    return cidadeSelecionada3;
}

public void setCidadeSelecionada3(Cidade cidadeSelecionada3) {
    this.cidadeSelecionada3 = cidadeSelecionada3;
}

public String getValorSimples() {
    return valorSimples;
}

public void setValorSimples(String valorSimples) {
```





```

    }
</style>
</h:head>

<h:body>
    <h:form>
        <p:growl id="growl" showDetail="true" life="5000" autoUpdate="true"/>

        <p:panel header="Exemplos de AutoComplete com Diferentes Converters"
            style="margin:20px">

            <!-- Converter com CDI (Recomendado) -->
            <div class="converter-title">1. Converter com CDI (Recomendado) - cidadeConverter</div>
            <h:panelGrid columns="3" cellpadding="10" styleClass="panel-grid">
                <p:outputLabel for="cdiConverter" value="Selecione uma cidade:"/>
                <p:autoComplete id="cdiConverter"
                    value="#{autoCompleteBean.cidadeSelecionada}"
                    completeMethod="#{autoCompleteBean.completarCidade}"
                    var="cidade"
                    itemLabel="#{cidade.nome} - #{cidade.estado}"
                    itemValue="#{cidade}"
                    converter="cidadeConverter"
                    forceSelection="true"
                    dropdown="true"
                    minQueryLength="2"
                    queryDelay="300"
                    placeholder="Digite uma cidade..."
                    panelStyle="width:300px">

                    <p:ajax event="itemSelect"
                        listener="#{autoCompleteBean.aoSelecionarCidade}"
                        update="growl"/>

                    <f:facet name="item">
                        <h:panelGrid columns="2" style="width:100%">
                            <h:outputText value="#{cidade.nome}" style="font-weight:bold"/>
                            <h:outputText value="#{cidade.estado}" style="color:#666666; text-align:right"/>
                        </h:panelGrid>
                    </f:facet>
                </p:autoComplete>

                <h:outputText value="Selecionado: #{autoCompleteBean.cidadeSelecionada}"
                    style="color:green; font-style:italic"/>

```

```

</h:panelGrid>

<!-- Converter Simples (Sem CDI) -->
<div class="converter-title">2. Converter Simples - cidadeConverterSimp
les</div>

<h:panelGrid columns="3" cellpadding="10" styleClass="panel-grid">
    <p:outputLabel for="simpleConverter" value="Selecione uma cidade:"/
>

    <p:autoComplete id="simpleConverter"
        value="#{autoCompleteBean.cidadeSelecioneada2}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"
        itemLabel="#{cidade.nome} - #{cidade.estado}"
        itemValue="#{cidade}"
        converter="cidadeConverterSimples"
        forceSelection="true"
        dropdown="true"
        minLength="2"
        placeholder="Digite uma cidade..."

    <p:ajax event="itemSelect"
        listener="#{autoCompleteBean.aoSelecionarCidade2}"
        update="growl"/>
</p:autoComplete>

    <h:outputText value="Selecionado: #{autoCompleteBean.cidadeSelecione
ada2}"
        style="color:green; font-style:italic"/>
</h:panelGrid>

<!-- Converter com Map (Cache) -->
<div class="converter-title">3. Converter com Map (Cache) - cidadeConve
rterMap</div>

<h:panelGrid columns="3" cellpadding="10" styleClass="panel-grid">
    <p:outputLabel for="mapConverter" value="Selecione uma cidade:"/>
    <p:autoComplete id="mapConverter"
        value="#{autoCompleteBean.cidadeSelecioneada3}"
        completeMethod="#{autoCompleteBean.completarCidadeMa
p}"
        var="cidade"
        itemLabel="#{cidade.nome} - #{cidade.estado} (Pop: #
{cidade.populacao})"
        itemValue="#{cidade}"
        converter="cidadeConverterMap"
        forceSelection="true"
        dropdown="true"
        minLength="2"

```

```

placeholder="Digite uma cidade...">

<p:ajax event="itemSelect"
        listener="#{autoCompleteBean.aoSelecionarCidade3}"
        update="growl"/>

<f:facet name="item">
    <h:panelGrid columns="3" style="width:100%">
        <h:outputText value="#{cidade.nome}" style="font-weight:bold"/>
        <h:outputText value="#{cidade.estado}" style="color:#666"/>
        <h:outputText value="#{cidade.regiao}" style="color:#2b7dbc; font-size:0.9em"/>
    </h:panelGrid>
</f:facet>
</p:autoComplete>

<h:outputText value="Selecionado: #{autoCompleteBean.cidadeSelecionada3.nome} - Pop: #{autoCompleteBean.cidadeSelecionada3.populacao}"
                style="color:green; font-style:italic"/>
</h:panelGrid>

<!-- Múltipla seleção com converter -->
<div class="converter-title">4. Múltipla Seleção com Converter</div>
<h:panelGrid columns="3" cellpadding="10" styleClass="panel-grid">
    <p:outputLabel for="multipleConverter" value="Selecione várias cidades:"/>
    <p:autoComplete id="multipleConverter"
        value="#{autoCompleteBean.cidadesMultiplas}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"
        itemLabel="#{cidade.nome} - #{cidade.estado}"
        itemValue="#{cidade}"
        converter="cidadeConverter"
        multiple="true"
        forceSelection="true"
        placeholder="Selecione várias cidades..."
        maxResults="10">
    </p:autoComplete>

    <p:commandButton value="Mostrar Selecionadas"
        actionListener="#{autoCompleteBean.mostrarMultiplas}"
        update="growl"
        icon="pi pi-check"/>
</h:panelGrid>

```

```

<!-- Valor Simples (sem converter) -->
<div class="converter-title">5. Valor Simples (Sem Converter)</div>
<h:panelGrid columns="3" cellpadding="10" styleClass="panel-grid">
    <p:outputLabel for="simpleValue" value="Digite o nome:"/>
    <p:autoComplete id="simpleValue"
        value="#{autoCompleteBean.valorSimples}"
        completeMethod="#{autoCompleteBean.completarCidade}"
        var="cidade"
        itemLabel="#{cidade.nome}"
        itemValue="#{cidade.nome}"
        forceSelection="true"
        placeholder="Digite o nome da cidade..."
    <p:ajax event="itemSelect"
        listener="#{autoCompleteBean.aoSelecionarValor}"
        update="growl"/>
</p:autoComplete>

    <h:outputText value="Valor: #{autoCompleteBean.valorSimples}"
        style="color:green; font-style:italic"/>
</h:panelGrid>

<p:separator/>

<p:commandButton value="Limpar Todos"
    actionListener="#{autoCompleteBean.limpar}"
    update="@form"
    icon="pi pi-refresh"
    style="margin-top:10px"/>
</p:panel>

<!-- Pannel explicativo sobre converters -->
<p:panel header="Sobre os Converters" style="margin:20px">
    <h:panelGrid columns="1">
        <h:outputText value="• Converter com CDI: Usa @Inject para acessar
o service diretamente. Recomendado para PrimeFaces 14 + Jakarta."/>
        <h:outputText value="• Converter Simples: Sem CDI, obtém o service
via EL. Útil quando não se pode usar CDI."/>
        <h:outputText value="• Converter com Map: Implementa cache para mel
hor performance em listas grandes."/>
        <h:outputText value="• Multiple: Demonstra o converter funcionando
com múltipla seleção."/>
        <h:outputText value="• Importante: O converter deve ser declarado n
o atributo 'converter' do p:autoComplete."/>
    </h:panelGrid>
</p:panel>
</h:form>

```

```
</h:body>  
</html>
```

## 6. Adicionar suporte a validação (Opcional)

java

```
package com.exemplo.validator;  
  
import com.exemplo.model.Cidade;  
import jakarta.faces.application.FacesMessage;  
import jakarta.faces.component.UIComponent;  
import jakarta.faces.context.FacesContext;  
import jakarta.faces.validator.FacesValidator;  
import jakarta.faces.validator.Validator;  
import jakarta.faces.validator.ValidatorException;  
  
@FacesValidator("cidadeValidator")  
public class CidadeValidator implements Validator<Cidade> {  
  
    @Override  
    public void validate(FacesContext context, UIComponent component, Cidade value)  
        throws ValidatorException {  
  
        if (value == null) {  
            return;  
        }  
  
        // Valida se a cidade é das regiões permitidas  
        if (!"Sudeste".equals(value.getRegiao()) &&  
            !"Sul".equals(value.getRegiao())) {  
  
            FacesMessage msg = new FacesMessage("Validação falhou",  
                "Apenas cidades do Sudeste e Sul são permitidas");  
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);  
            throw new ValidatorException(msg);  
        }  
    }  
}
```

## Como usar no XHTML com validação:

xhtml

```
<p:autoComplete id="comValidator"
    value="#{autoCompleteBean.cidadeSelecionada}"
    completeMethod="#{autoCompleteBean.completarCidade}"
    converter="cidadeConverter"
    validator="cidadeValidator"
    forceSelection="true">
    <f:ajax execute="@this" render="msg"/>
</p:autoComplete>
<p:message for="comValidator"/>
```

## Principais diferenças entre os converters:

### 1. **cidadeConverter** (com CDI):

- Usa @Inject
- Requer @FacesConverter(managed=true)
- Mais limpo e moderno
- Recomendado para Jakarta EE

### 2. **cidadeConverterSimples** (sem CDI):

- Não usa CDI
- Obtém service via EL
- Compatível com versões mais antigas

### 3. **cidadeConverterMap** (com cache):

- Melhor performance
- Mantém cache local
- Útil para listas grandes

## Observações importantes:

- O atributo `managed=true` no `@FacesConverter` é necessário para injetar dependências com CDI
- O converter deve ser referenciado pelo valor do atributo `value` da anotação
- Para objetos complexos, sempre implemente `equals()` e `hashCode()`
- O cache no converter pode melhorar significativamente a performance
- Teste ambos os métodos para ver qual se adapta melhor ao seu caso de uso

## Simplifique o exemplo usando o caso com muitos dados

Aqui está um exemplo simplificado e otimizado para lidar com **muitos dados** no AutoComplete do PrimeFaces 14:

### 1. Model (Cidade.java) - Otimizado

java

```
package com.exemplo.model;

import java.io.Serializable;
import java.util.Objects;

public class Cidade implements Serializable {

    private Integer id;
    private String nome;
    private String estado;
    private String regiao;

    public Cidade() {}

    public Cidade(Integer id, String nome, String estado, String regiao) {
        this.id = id;
        this.nome = nome;
        this.estado = estado;
        this.regiao = regiao;
    }

    // Getters e Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

```
}

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getRegiao() {
    return regiao;
}

public void setRegiao(String regiao) {
    this.regiao = regiao;
}

public String getNomeCompleto() {
    return nome + " - " + estado;
}

@Override
public String toString() {
    return nome + " - " + estado;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Cidade cidade = (Cidade) obj;
    return Objects.equals(id, cidade.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
}
```

## 2. Service (CidadeService.java) - Com muitos dados simulados

java



```
package com.exemplo.service;

import com.exemplo.model.Cidade;
import jakarta.annotation.PostConstruct;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Named
@ApplicationScoped
public class CidadeService implements Serializable {

    private List<Cidade> cidades;
    private static final int TOTAL_CIDADES = 10000; // 10 mil cidades simuladas

    @PostConstruct
    public void init() {
        inicializarDados();
    }

    private void inicializarDados() {
        cidades = new ArrayList<>(TOTAL_CIDADES);
        String[] estados = {"SP", "RJ", "MG", "RS", "PR", "SC", "BA", "PE", "CE",
"DF"};
        String[] regioes = {"Sudeste", "Sul", "Nordeste", "Centro-Oeste", "Norte"};
        String[] cidadesBase = {
            "São Paulo", "Rio de Janeiro", "Belo Horizonte", "Porto Alegre", "Curit
iba",
            "Florianópolis", "Salvador", "Recife", "Fortaleza", "Brasília",
            "Manaus", "Belém", "Goiânia", "Campo Grande", "Cuiabá",
            "São Luís", "Natal", "João Pessoa", "Maceió", "Aracaju",
            "Vitória", "Niterói", "Santos", "Campinas", "Ribeirão Preto"
        };

        // Gera 10.000 cidades combinando os dados base
        int id = 1;
        for (int i = 0; i < TOTAL_CIDADES; i++) {
            String nomeBase = cidadesBase[i % cidadesBase.length];
            String estado = estados[i % estados.length];
            String regioao = regioes[i % regioes.length];

            // Adiciona sufixo para criar variações
            String nome = nomeBase + " " + (i / cidadesBase.length + 1);
        }
    }
}
```

```

        cidades.add(new Cidade(id++, nome, estado, regioao));
    }
}

/**
 * Método otimizado para busca com muitos dados
 * Usa parallelStream para melhor performance
 */
public List<Cidade> pesquisarCidades(String query) {
    if (query == null || query.length() < 2) {
        return new ArrayList<>();
    }

    String queryLower = query.toLowerCase().trim();

    // Usa parallelStream para busca em grandes volumes
    return cidades.parallelStream()
        .filter(cidade -> cidade.getNome().toLowerCase().contains(queryLower) ||
            cidade.getEstado().toLowerCase().contains(queryLower))
        .limit(20) // Limita resultados para melhor performance
        .collect(Collectors.toList());
}

/**
 * Busca por ID (otimizado com map)
 */
public Cidade encontrarPorId(Integer id) {
    if (id == null) return null;

    return cidades.stream()
        .filter(c -> c.getId().equals(id))
        .findFirst()
        .orElse(null);
}

public List<Cidade> getCidades() {
    return cidades;
}
}

```

### 3. Converter (CidadeConverter.java) - Com cache otimizado

java

```
package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

@Named
@FacesConverter(value = "cidadeConverter", managed = true)
public class CidadeConverter implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    // Cache thread-safe para melhor performance
    private static final Map<Integer, Cidade> CACHE = new ConcurrentHashMap<>();

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Integer id = Integer.parseInt(value);

            // Verifica cache primeiro
            Cidade cached = CACHE.get(id);
            if (cached != null) {
                return cached;
            }

            // Busca e adiciona ao cache
            Cidade cidade = cidadeService.encontrarPorId(id);
            if (cidade != null) {
                CACHE.put(id, cidade);
            }
        }
    }
}
```

```

        return cidade;
    } catch (NumberFormatException e) {
        return null;
    }
}

@Override
public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
    if (cidade == null || cidade.getId() == null) {
        return "";
    }

    // Adiciona ao cache se não existir
    CACHE.putIfAbsent(cidade.getId(), cidade);

    return String.valueOf(cidade.getId());
}
}

```

## 4. Controller (AutoCompleteBean.java) - Simplificado

java

```

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

    private Cidade cidadeSelecionada;
    private List<Cidade> cidadesMultiplas;
}

```

```
private String resultado;

public AutoCompleteBean() {
    cidadesMultiplas = new ArrayList<>();
}

/**
 * Método de busca otimizado para muitos dados
 */
public List<Cidade> buscarCidades(String query) {
    long inicio = System.currentTimeMillis();
    List<Cidade> resultados = cidadeService.pesquisarCidades(query);
    long fim = System.currentTimeMillis();

    System.out.println("Busca por '" + query + "' levou " + (fim - inicio) + "ms - " +
        resultados.size() + " resultados");

    return resultados;
}

/**
 * Evento ao selecionar cidade
 */
public void aoSelecionar() {
    if (cidadeSelecionada != null) {
        resultado = "Selecionado: " + cidadeSelecionada.getNomeCompleto();
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Sucesso", resultado));
    }
}

/**
 * Mostra as cidades selecionadas (múltiplas)
 */
public void mostrarSelecionadas() {
    if (cidadesMultiplas.isEmpty()) {
        resultado = "Nenhuma cidade selecionada";
    } else {
        StringBuilder sb = new StringBuilder("Cidades: ");
        for (Cidade c : cidadesMultiplas) {
            sb.append(c.getNome()).append(", ");
        }
        resultado = sb.toString();
    }

    FacesContext.getCurrentInstance().addMessage(null,
```

```

        new FacesMessage("Resultado", resultado));
    }

    /**
     * Limpa todas as seleções
     */
    public void limpar() {
        cidadeSelecionada = null;
        cidadesMultiplas.clear();
        resultado = null;
    }

    // Getters e Setters
    public Cidade getCidadeSelecionada() {
        return cidadeSelecionada;
    }

    public void setCidadeSelecionada(Cidade cidadeSelecionada) {
        this.cidadeSelecionada = cidadeSelecionada;
    }

    public List<Cidade> getCidadesMultiplas() {
        return cidadesMultiplas;
    }

    public void setCidadesMultiplas(List<Cidade> cidadesMultiplas) {
        this.cidadesMultiplas = cidadesMultiplas;
    }

    public String getResultado() {
        return resultado;
    }
}

```

## 5. Página XHTML (autocomplete.xhtml) - Versão simplificada

xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:p="primefaces">

```

```
<h:head>
  <title>AutoComplete com muitos dados - PrimeFaces 14</title>
  <style>
    .info-box {
      background-color: #e8f4f8;
      border-left: 4px solid #2b7dbc;
      padding: 10px;
      margin: 10px 0;
    }
  </style>
</h:head>

<h:body>
  <h:form>
    <p:growl id="growl" showDetail="true" life="3000"/>

    <p:panel header="AutoComplete Otimizado para Grandes Volumes de Dados"
      style="margin:20px">

      <div class="info-box">
        <strong>Informação:</strong> Banco de dados simulado com
        <strong>10.000 cidades</strong>. A busca é otimizada usando
        parallelStream e cache no converter.
      </div>

      <h:panelGrid columns="2" cellpadding="10" style="width:100%">

        <!-- AutoComplete Simples -->
        <p:outputLabel for="simple" value="Seleção Simples:"/>
        <p:autoComplete id="simple"
          value="#{autoCompleteBean.cidadeSelecionada}"
          completeMethod="#{autoCompleteBean.buscarCidades}"
          var="cidade"
          itemLabel="#{cidade.nome} - #{cidade.estado}"
          itemValue="#{cidade}"
          converter="cidadeConverter"
          forceSelection="true"
          dropdown="true"
          minQueryLength="2"
          queryDelay="300"
          maxResults="20"
          placeholder="Digite pelo menos 2 caracteres..."
          panelStyle="width:350px">

          <p:ajax event="itemSelect"
            listener="#{autoCompleteBean.aoSelecionar}"
```

```

        update="growl resultado"/>
</p:autoComplete>

<!-- AutoComplete Múltiplo -->
<p:outputLabel for="multiple" value="Seleção Múltipla:"/>
<p:autoComplete id="multiple"
    value="#{autoCompleteBean.cidadesMultiplas}"
    completeMethod="#{autoCompleteBean.buscarCidades}"
    var="cidade"
    itemLabel="#{cidade.nome} - #{cidade.estado} ({#cida
de.regiao})"

    itemValue="#{cidade}"
    converter="cidadeConverter"
    multiple="true"
    forceSelection="true"
    minQueryLength="2"
    maxResults="20"
    placeholder="Selecione várias cidades..."
    panelStyle="width:350px">
</p:autoComplete>

<!-- Botões de ação -->
<p:outputLabel value="Ações:"/>
<h:panelGroup>
    <p:commandButton value="Mostrar Seleccionadas"
        actionListener="#{autoCompleteBean.mostrarSeleci
onadas}"

        update="growl resultado"
        icon="pi pi-check"
        style="margin-right:10px"/>

    <p:commandButton value="Limpar"
        actionListener="#{autoCompleteBean.limpar}"
        update="@form"
        icon="pi pi-refresh"/>
</h:panelGroup>

<!-- Resultado -->
<p:outputLabel for="resultado" value="Resultado:"/>
<h:outputText id="resultado"
    value="#{autoCompleteBean.resultado}"
    style="font-weight:bold; color:#2b7dbc"/>

</h:panelGrid>

<!-- Estatísticas de performance -->
<p:separator/>

```



```

<h:panelGrid columns="2" style="margin-top:20px">
    <p:outputLabel value="Total de registros:"/>
    <h:outputText value="10.000 cidades" style="font-weight:bold"/>

    <p:outputLabel value="Cache ativo:"/>
    <h:outputText value="Sim (ConcurrentHashMap)" style="color:green; font-weight:bold"/>

    <p:outputLabel value="Busca paralela:"/>
    <h:outputText value="Sim (parallelStream)" style="color:green; font-weight:bold"/>

    <p:outputLabel value="Limite de resultados:"/>
    <h:outputText value="20 por busca" style="font-weight:bold"/>
</h:panelGrid>

</p:panel>

<!-- Dicas de uso -->
<p:panel header="Dicas para lidar com muitos dados" style="margin:20px">
    <h:panelGrid columns="1">
        <h:outputText value="✓ Use minQueryLength='2' para evitar buscas desnecessárias"/>
        <h:outputText value="✓ Use queryDelay para evitar buscas a cada tecla"/>
        <h:outputText value="✓ Limite os resultados com maxResults"/>
        <h:outputText value="✓ Use cache no Converter para evitar buscas repetidas"/>
        <h:outputText value="✓ Considere usar parallelStream para grandes volumes"/>
        <h:outputText value="✓ Em produção, use índices no banco de dados"/>
    </h:panelGrid>
</p:panel>
</h:form>
</h:body>
</html>

```

## 6. beans.xml (para ativar CDI)

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```

xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/beans_4_0.xsd"
version="4.0"
bean-discovery-mode="annotated">
</beans>

```

## Principais otimizações implementadas:

1. **Muitos dados:** 10.000 registros simulados
2. **Busca paralela:** Uso de `parallelStream()` para melhor performance
3. **Cache eficiente:** `ConcurrentHashMap` no converter
4. **Limitação de resultados:** `maxResults="20"` para não sobrecarregar
5. **Delay inteligente:** `queryDelay="300"` para evitar buscas a cada tecla
6. **Mínimo de caracteres:** `minQueryLength="2"` para buscas significativas
7. **Thread-safe:** Cache com `ConcurrentHashMap` para ambientes concorrentes

## Como testar:

1. O serviço já inicializa com 10.000 cidades
2. Digite pelo menos 2 caracteres para ver os resultados
3. Observe a performance mesmo com muitos dados
4. O cache evita buscas repetidas ao converter IDs

Esta versão é mais simples, focada apenas no essencial para lidar com grandes volumes de dados de forma eficiente.

Um exemplo buscando os dados de uma entidade jpa mysql

Q Read 3 web pages 🐾🧠🧠

Aqui está um exemplo completo de integração do AutoComplete do PrimeFaces 14 com JPA e MySQL, usando as melhores práticas para lidar com muitos dados no banco.

## 1. Configuração do Projeto (pom.xml)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.exemplo</groupId>
  <artifactId>primefaces-jpa-autocomplete</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <primefaces.version>14.0.0</primefaces.version>
  </properties>

  <dependencies>
    <!-- PrimeFaces -->
    <dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>${primefaces.version}</version>
      <classifier>jakarta</classifier>
    </dependency>

    <!-- Jakarta EE -->
    <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>10.0.0</version>
      <scope>provided</scope>
    </dependency>

    <!-- JPA -->
    <dependency>
      <groupId>jakarta.persistence</groupId>
      <artifactId>jakarta.persistence-api</artifactId>
      <version>3.1.0</version>
    </dependency>

    <!-- Hibernate (implementação JPA) -->
    <dependency>
      <groupId>org.hibernate.orm</groupId>
```

```

        <artifactId>hibernate-core</artifactId>
        <version>6.3.1.Final</version>
    </dependency>

    <!-- MySQL Connector -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.2.0</version>
    </dependency>

    <!-- HikariCP (Pool de conexões) -->
    <dependency>
        <groupId>com.zaxxer</groupId>
        <artifactId>HikariCP</artifactId>
        <version>5.0.1</version>
    </dependency>
</dependencies>

<build>
    <finalName>primefaces-jpa-autocomplete</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.4.0</version>
        </plugin>
    </plugins>
</build>
</project>

```

## 2. Entidade JPA (Cidade.java)

```
java
```

```
package com.exemplo.model;
```

```
import jakarta.persistence.*;
```

```
import java.io.Serializable;
```

```
import java.util.Objects;
```

```
@Entity
```

```
@Table(name = "cidades")
```

```
@NamedQueries({
```

```
    @NamedQuery(name = "Cidade.buscarPorNomeOuEstado",
```

```
        query = "SELECT c FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro OR
```

```
LOWER(c.estado) LIKE :filtro ORDER BY c.nome"),
    @NamedQuery(name = "Cidade.buscarPorId",
        query = "SELECT c FROM Cidade c WHERE c.id = :id")
})
public class Cidade implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome", length = 100, nullable = false)
    private String nome;

    @Column(name = "estado", length = 2, nullable = false)
    private String estado;

    @Column(name = "populacao")
    private Integer populacao;

    @Column(name = "regiao", length = 50)
    private String regiao;

    @Column(name = "ativo")
    private Boolean ativo = true;

    // Construtores
    public Cidade() {}

    public Cidade(String nome, String estado, String regiao) {
        this.nome = nome;
        this.estado = estado;
        this.regiao = regiao;
        this.ativo = true;
    }

    // Getters e Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }
}
```

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

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public Integer getPopulacao() {
    return populacao;
}

public void setPopulacao(Integer populacao) {
    this.populacao = populacao;
}

public String getRegiao() {
    return regiao;
}

public void setRegiao(String regiao) {
    this.regiao = regiao;
}

public Boolean getAtivo() {
    return ativo;
}

public void setAtivo(Boolean ativo) {
    this.ativo = ativo;
}

public String getNomeCompleto() {
    return nome + " - " + estado;
}

@Override
public String toString() {
    return nome + " - " + estado;
}

@Override
```

```

    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Cidade cidade = (Cidade) o;
        return Objects.equals(id, cidade.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}

```

### 3. Persistence Unit (persistence.xml)

xml

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
    xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
        https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">

    <persistence-unit name="autocompletePU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <class>com.exemplo.model.Cidade</class>

        <properties>
            <!-- Configuração da conexão MySQL -->
            <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/exemplo_db?useSSL=false&serverTimezone=UTC&rewriteBatchedStatements=true"/>
            <property name="jakarta.persistence.jdbc.user" value="root"/>
            <property name="jakarta.persistence.jdbc.password" value="sua_senha"/>

            <!-- Configurações do Hibernate -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.jdbc.batch_size" value="30"/>

```

```

        <!-- Pool de conexões HikariCP -->
        <property name="hibernate.connection.provider_class" value="org.hiberna
te.hikaricp.internal.HikariCPConnectionProvider"/>
        <property name="hibernate.hikari.maximumPoolSize" value="20"/>
        <property name="hibernate.hikari.minimumIdle" value="5"/>
        <property name="hibernate.hikari.idleTimeout" value="300000"/>
    </properties>
</persistence-unit>
</persistence>

```

## 4. DAO/Repository (CidadeDAO.java)

```
java
```

```

package com.exemplo.dao;

import com.exemplo.model.Cidade;
import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Named
@ApplicationScoped
public class CidadeDAO implements Serializable {

    private EntityManagerFactory emf;
    private EntityManager em;

    @PostConstruct
    public void init() {
        emf = Persistence.createEntityManagerFactory("autocompletePU");
        em = emf.createEntityManager();

        // Verifica se precisa popular dados iniciais
        if (contarCidades() == 0) {
            popularDadosIniciais();
        }
    }
}

```



```
@PreDestroy
public void destroy() {
    if (em != null && em.isOpen()) {
        em.close();
    }
    if (emf != null && emf.isOpen()) {
        emf.close();
    }
}

/**
 * Busca cidades pelo nome ou estado (otimizado com índices)
 */
public List<Cidade> buscarPorNomeOuEstado(String query, int maxResults) {
    String filtro = query.toLowerCase() + "%";

    TypedQuery<Cidade> typedQuery = em.createNamedQuery("Cidade.buscarPorNomeOuEstado", Cidade.class)
        .setParameter("filtro", filtro)
        .setMaxResults(maxResults)
        .setHint("org.hibernate.cacheable", true); // Habilita cache de consulta

    return typedQuery.getResultList();
}

/**
 * Busca cidade por ID com cache
 */
public Cidade buscarPorId(Long id) {
    return em.find(Cidade.class, id);
}

/**
 * Busca cidades por IDs (para múltipla seleção)
 */
public List<Cidade> buscarPorIds(List<Long> ids) {
    if (ids == null || ids.isEmpty()) {
        return List.of();
    }

    return em.createQuery("SELECT c FROM Cidade c WHERE c.id IN :ids", Cidade.class)
        .setParameter("ids", ids)
        .getResultList();
}
```

```
/**
 * Busca paginada para casos onde o AutoComplete não é suficiente
 */
public Map<String, Object> buscarPaginado(String filtro, int pagina, int tamanhoPagina) {
    Map<String, Object> resultado = new HashMap<>();

    // Consulta para dados
    TypedQuery<Cidade> query = em.createQuery(
        "SELECT c FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro ORDER BY c.nome",
        Cidade.class)
        .setParameter("filtro", "%" + filtro.toLowerCase() + "%")
        .setFirstResult(pagina * tamanhoPagina)
        .setMaxResults(tamanhoPagina);

    // Consulta para total de registros
    TypedQuery<Long> countQuery = em.createQuery(
        "SELECT COUNT(c) FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro",
        Long.class)
        .setParameter("filtro", "%" + filtro.toLowerCase() + "%");

    resultado.put("dados", query.getResultList());
    resultado.put("total", countQuery.getSingleResult());

    return resultado;
}

/**
 * Conta total de cidades no banco
 */
private Long contarCidades() {
    return em.createQuery("SELECT COUNT(c) FROM Cidade c", Long.class)
        .getSingleResult();
}

/**
 * Popula dados iniciais para teste
 */
private void popularDadosIniciais() {
    EntityTransaction tx = em.getTransaction();
    try {
        tx.begin();

        // Dados do IBGE (amostra)
        String[][] cidadesIBGE = {
            {"São Paulo", "SP", "Sudeste", "12300000"},

```

```

        {"Rio de Janeiro", "RJ", "Sudeste", "6748000"},
        {"Brasília", "DF", "Centro-Oeste", "3055000"},
        {"Salvador", "BA", "Nordeste", "2887000"},
        {"Fortaleza", "CE", "Nordeste", "2687000"},
        {"Belo Horizonte", "MG", "Sudeste", "2523000"},
        {"Manaus", "AM", "Norte", "2219000"},
        {"Curitiba", "PR", "Sul", "1963000"},
        {"Recife", "PE", "Nordeste", "1655000"},
        {"Goiânia", "GO", "Centro-Oeste", "1536000"},
        {"Porto Alegre", "RS", "Sul", "1488000"},
        {"Belém", "PA", "Norte", "1499000"},
        {"Guarulhos", "SP", "Sudeste", "1392000"},
        {"Campinas", "SP", "Sudeste", "1214000"},
        {"São Luís", "MA", "Nordeste", "1109000"},
        {"São Gonçalo", "RJ", "Sudeste", "1084000"},
        {"Maceió", "AL", "Nordeste", "1025000"},
        {"Duque de Caxias", "RJ", "Sudeste", "919000"},
        {"Campo Grande", "MS", "Centro-Oeste", "906000"},
        {"Natal", "RN", "Nordeste", "890000"}
    };

    for (String[] dados : cidadesIBGE) {
        Cidade cidade = new Cidade(dados[0], dados[1], dados[2]);
        cidade.setPopulacao(Integer.parseInt(dados[3]));
        em.persist(cidade);
    }

    tx.commit();
    System.out.println("Dados iniciais inseridos com sucesso!");

} catch (Exception e) {
    if (tx.isActive()) {
        tx.rollback();
    }
    e.printStackTrace();
}
}
}

```

## 5. Service (CidadeService.java)

```

java

package com.exemplo.service;

import com.exemplo.dao.CidadeDAO;

```

```
import com.exemplo.model.Cidade;
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.List;

@Named
@RequestScoped
public class CidadeService implements Serializable {

    @Inject
    private CidadeDAO cidadeDAO;

    /**
     * Método chamado pelo AutoComplete
     */
    public List<Cidade> pesquisarCidades(String query) {
        if (query == null || query.length() < 2) {
            return List.of();
        }

        long inicio = System.currentTimeMillis();
        List<Cidade> resultados = cidadeDAO.buscarPorNomeOuEstado(query, 20);
        long fim = System.currentTimeMillis();

        System.out.println("Busca JPA por '" + query + "' levou " + (fim - inicio)
+
            "ms - " + resultados.size() + " resultados");

        return resultados;
    }

    /**
     * Busca cidade por ID (usado pelo converter)
     */
    public Cidade buscarPorId(Long id) {
        if (id == null) return null;
        return cidadeDAO.buscarPorId(id);
    }

    /**
     * Busca múltiplas cidades por IDs
     */
    public List<Cidade> buscarPorIds(List<Long> ids) {
        return cidadeDAO.buscarPorIds(ids);
    }
}
```

```
}  
}
```

## 6. Converter (CidadeConverter.java)

java

```
package com.exemplo.converter;  
  
import com.exemplo.model.Cidade;  
import com.exemplo.service.CidadeService;  
import jakarta.faces.component.UIComponent;  
import jakarta.faces.context.FacesContext;  
import jakarta.faces.convert.Converter;  
import jakarta.faces.convert.FacesConverter;  
import jakarta.inject.Inject;  
import jakarta.inject.Named;  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
  
@Named  
@FacesConverter(value = "cidadeConverter", managed = true)  
public class CidadeConverter implements Converter<Cidade> {  
  
    @Inject  
    private CidadeService cidadeService;  
  
    // Cache com limite para evitar memory leak  
    private static final Map<Long, Cidade> CACHE = new ConcurrentHashMap<>();  
    private static final int CACHE_MAX_SIZE = 1000;  
  
    @Override  
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {  
        if (value == null || value.isEmpty()) {  
            return null;  
        }  
  
        try {  
            Long id = Long.parseLong(value);  
  
            // Verifica cache primeiro  
            Cidade cached = CACHE.get(id);  
            if (cached != null) {  
                return cached;  
            }  
        }  
    }  
}
```

```

        // Busca no banco via service
        Cidade cidade = cidadeService.buscarPorId(id);

        // Adiciona ao cache se encontrou
        if (cidade != null) {
            // Mantém o cache com tamanho controlado
            if (CACHE.size() >= CACHE_MAX_SIZE) {
                CACHE.clear(); // Simplificação: limpa tudo quando atinge limit
            }
            CACHE.put(id, cidade);
        }

        return cidade;
    } catch (NumberFormatException e) {
        return null;
    }
}

@Override
public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
    if (cidade == null || cidade.getId() == null) {
        return "";
    }

    // Adiciona ao cache para conversões futuras
    if (!CACHE.containsKey(cidade.getId()) && CACHE.size() < CACHE_MAX_SIZE) {
        CACHE.put(cidade.getId(), cidade);
    }

    return String.valueOf(cidade.getId());
}
}

```

## 7. Controller (AutoCompleteBean.java)

```

java

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;

```

```
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

    private Cidade cidadeSelecionada;
    private List<Cidade> cidadesMultiplas;
    private String estatisticas;

    public AutoCompleteBean() {
        cidadesMultiplas = new ArrayList<>();
    }

    /**
     * Método chamado pelo AutoComplete (liga com o service JPA)
     */
    public List<Cidade> buscarCidades(String query) {
        return cidadeService.pesquisarCidades(query);
    }

    /**
     * Evento ao selecionar uma cidade
     */
    public void aoSelecionarCidade() {
        if (cidadeSelecionada != null) {
            String msg = "Cidade selecionada: " + cidadeSelecionada.getNomeCompleto() +
                " | População: " + String.format("%,d", cidadeSelecionada.getPopulacao()) +
                " | Região: " + cidadeSelecionada.getRegiao();

            FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(FacesMessage.SEVERITY_INFO, "Selecionado", msg));

            estatisticas = msg;
        }
    }
}
```

```
/**
 * Mostra cidades selecionadas (múltiplas)
 */
public void mostrarMultiplas() {
    if (cidadesMultiplas == null || cidadesMultiplas.isEmpty()) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_WARN, "Aviso", "Nenhuma cidade selecionada"));
        estatisticas = "Nenhuma cidade selecionada";
        return;
    }

    StringBuilder sb = new StringBuilder("Cidades selecionadas (" + cidadesMultiplas.size() + "):\n");
    long populacaoTotal = 0;

    for (Cidade c : cidadesMultiplas) {
        sb.append("• ").append(c.getNomeCompleto()).append("\n");
        populacaoTotal += c.getPopulacao() != null ? c.getPopulacao() : 0;
    }

    sb.append("\nPopulação total: ").append(String.format("%,d", populacaoTotal));

    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Múltiplas cidades", sb.toString()));

    estatisticas = sb.toString();
}

/**
 * Limpa todas as seleções
 */
public void limpar() {
    cidadeSelecionada = null;
    cidadesMultiplas.clear();
    estatisticas = null;

    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Limpo", "Seleções removidas"));
}

// Getters e Setters
public Cidade getCidadeSelecionada() {
```



```
        return cidadeSelecionada;
    }

    public void setCidadeSelecionada(Cidade cidadeSelecionada) {
        this.cidadeSelecionada = cidadeSelecionada;
    }

    public List<Cidade> getCidadesMultiplas() {
        return cidadesMultiplas;
    }

    public void setCidadesMultiplas(List<Cidade> cidadesMultiplas) {
        this.cidadesMultiplas = cidadesMultiplas;
    }

    public String getEstatisticas() {
        return estatisticas;
    }
}
```

## 8. Página XHTML (autocomplete.xhtml)

xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:p="primefaces">

    <h:head>
        <title>AutoComplete com JPA/MySQL - PrimeFaces 14</title>
        <style>
            .info-panel {
                background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
                color: white;
                padding: 15px;
                border-radius: 8px;
                margin-bottom: 20px;
            }
            .stats-box {
                background-color: #f0f8ff;
                border-left: 4px solid #2b7dbc;
                padding: 10px 15px;
                margin: 15px 0;
            }
        </style>
    </h:head>
</html>
```

```

        font-family: monospace;
        white-space: pre-wrap;
    }
</style>
</h:head>

<h:body>
    <h:form>
        <p:growl id="growl" showDetail="true" life="5000" autoUpdate="true"/>

        <div class="info-panel">
            <h2> AutoComplete com JPA e MySQL</h2>
            <p>Exemplo integrado com banco de dados MySQL usando JPA/Hibernate</p>
        </div>

        <p:panel header="Busca de Cidades no Banco de Dados" style="margin:20px">

            <h:panelGrid columns="2" cellpadding="10" columnClasses="label,field" style="width:100%">

                <!-- Seleção Simples -->
                <p:outputLabel for="simple" value="Seleção Simples:"/>
                <p:autoComplete id="simple"
                    value="#{autoCompleteBean.cidadeSelecionada}"
                    completeMethod="#{autoCompleteBean.buscarCidades}"
                    var="cidade"
                    itemLabel="#{cidade.nome} - #{cidade.estado}"
                    itemValue="#{cidade}"
                    converter="cidadeConverter"
                    forceSelection="true"
                    dropdown="true"
                    minLength="2"
                    queryDelay="400"
                    maxResults="15"
                    placeholder="Digite pelo menos 2 letras..."
                    panelStyle="width:400px"
                    emptyMessage="Nenhuma cidade encontrada">

                <p:ajax event="itemSelect"
                    listener="#{autoCompleteBean.aoSelecionarCidade}"
                    update="growl stats"/>

                <f:facet name="item">
                    <h:panelGrid columns="3" style="width:100%">
                        <h:outputText value="#{cidade.nome}" style="font-weight:bold"/>

                        <h:outputText value="#{cidade.estado}" style="color:#66

```

```
6"/>
```

```

        <h:outputText value="#{cidade.regiao}" style="color:#2b
7dbc; font-size:0.9em"/>

```

```

        </h:panelGrid>
    </f:facet>
</p:autoComplete>

```

```

<!-- Seleção Múltipla -->
<p:outputLabel for="multiple" value="Seleção Múltipla:"/>
<p:autoComplete id="multiple"
    value="#{autoCompleteBean.cidadesMultiplas}"
    completeMethod="#{autoCompleteBean.buscarCidades}"
    var="cidade"
    itemLabel="#{cidade.nome} - #{cidade.estado}"
    itemValue="#{cidade}"
    converter="cidadeConverter"
    multiple="true"
    forceSelection="true"
    minQueryLength="2"
    maxResults="15"
    placeholder="Selecione várias cidades..."
    panelStyle="width:400px">

```

```

    <f:facet name="item">
        <h:panelGrid columns="3" style="width:100%">
            <h:outputText value="#{cidade.nome}" style="font-weight:
t:bold"/>
            <h:outputText value="#{cidade.estado}" style="color:#66
6"/>
            <h:outputText value="#{cidade.populacao}" style="color:
green; font-size:0.9em">
                <f:convertNumber type="number" groupingUsed="true"/
>

```

```

        </h:outputText>
    </h:panelGrid>
</f:facet>
</p:autoComplete>

```

```

<!-- Botões de ação -->
<p:outputLabel value="Ações:"/>
<h:panelGroup>
    <p:commandButton value="Mostrar Selecionadas"
        actionListener="#{autoCompleteBean.mostrarMultip
las}"
        update="growl stats"
        icon="pi pi-check"
        style="margin-right:10px"/>

```

```

        <p:commandButton value="Limpar"
            actionListener="#{autoCompleteBean.limpar}"
            update="@form"
            icon="pi pi-refresh"/>
    </h:panelGroup>

</h:panelGrid>

<p:separator/>

<!-- Estatísticas e Resultados -->
<p:outputPanel id="stats">
    <h:panelGroup rendered="#{not empty autoCompleteBean.estatisticas}">
        <div class="stats-box">
            <h:outputText value="#{autoCompleteBean.estatisticas}" escape="false"/>
        </div>
    </h:panelGroup>
</p:outputPanel>

<!-- Informações do Banco -->
<p:fieldset legend="📊 Informações do Banco" toggleable="true" toggleSpeed="500">
    <h:panelGrid columns="2" style="width:100%">
        <h:outputText value="Tecnologia:"/>
        <h:outputText value="MySQL + JPA/Hibernate" style="font-weight:bold"/>

        <h:outputText value="Pool de conexões:"/>
        <h:outputText value="HikariCP (20 conexões)" style="font-weight:bold"/>

        <h:outputText value="Cache no converter:"/>
        <h:outputText value="ConcurrentHashMap (1000 itens)" style="font-weight:bold"/>

        <h:outputText value="Busca no banco:"/>
        <h:outputText value="NamedQuery com índice" style="font-weight:bold"/>

        <h:outputText value="Dados iniciais:"/>
        <h:outputText value="Cidades do IBGE (população)" style="font-weight:bold"/>
    </h:panelGrid>
</p:fieldset>

```

```
        </p:panel>
    </h:form>
</h:body>
</html>
```

## 9. Script SQL para criar o banco

sql

```
-- Cria o banco de dados
CREATE DATABASE IF NOT EXISTS exemplo_db
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

USE exemplo_db;

-- Cria a tabela de cidades
CREATE TABLE IF NOT EXISTS cidades (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    estado VARCHAR(2) NOT NULL,
    populacao INT,
    regiao VARCHAR(50),
    ativo BOOLEAN DEFAULT TRUE,
    INDEX idx_nome (nome),
    INDEX idx_estado (estado),
    INDEX idx_regiao (regiao)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Índice composto para busca otimizada
CREATE INDEX idx_busca ON cidades (nome, estado);

-- Exemplo de consulta otimizada
EXPLAIN SELECT * FROM cidades
WHERE LOWER(nome) LIKE 'sao%'
ORDER BY nome
LIMIT 20;
```

## 10. Configuração do DataSource no Tomcat 11 (context.xml)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/autocompleteDS"
    auth="Container"
    type="javax.sql.DataSource"
    factory="com.zaxxer.hikari.HikariJNDIFactory"
    driverClassName="com.mysql.cj.jdbc.Driver"
    jdbcUrl="jdbc:mysql://localhost:3306/exemplo_db?useSSL=false&serv
erTimezone=UTC"
    username="root"
    password="sua_senha"
    maximumPoolSize="20"
    minimumIdle="5"
    idleTimeout="300000"
    connectionTimeout="30000"
    maxLifetime="1800000"
    leakDetectionThreshold="60000"/>
</Context>
```

## 11. Versão alternativa usando LazyDataModel (PrimeFaces 11+) <sup>3</sup>

O PrimeFaces 11 introduziu o `JpaLazyDataModel` para facilitar a integração com JPA <sup>3</sup> :

```
java

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.dao.CidadeDAO;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import org.primefaces.model.LazyDataModel;
import org.primefaces.model.SortOrder;
import java.util.List;
import java.util.Map;

@Named
public class LazyCidadeBean {

    @Inject
    private CidadeDAO cidadeDAO;
```

```

private LazyDataModel<Cidade> lazyModel;
private Cidade cidadeSelecionada;

public LazyCidadeBean() {
    lazyModel = new LazyDataModel<Cidade>() {
        @Override
        public List<Cidade> load(int first, int pageSize,
                                   String sortField, SortOrder sortOrder,
                                   Map<String, Object> filters) {

            String filtro = (String) filters.get("nome");
            if (filtro == null) filtro = "";

            // Usa o DAO para busca paginada
            Map<String, Object> resultado = cidadeDAO.buscarPaginado(filtro,
                                                                       first / pageSize, pageSize);

            setRowCount(((Long) resultado.get("total")).intValue());

            return (List<Cidade>) resultado.get("dados");
        }
    };
}

// Getters e Setters
public LazyDataModel<Cidade> getLazyModel() {
    return lazyModel;
}

public Cidade getCidadeSelecionada() {
    return cidadeSelecionada;
}

public void setCidadeSelecionada(Cidade cidadeSelecionada) {
    this.cidadeSelecionada = cidadeSelecionada;
}
}

```

## Configuração no XHTML para LazyDataModel:

xhtml

```

<p:autoComplete id="lazyModel"
    value="#{lazyCidadeBean.cidadeSelecionada}"
    lazyModel="#{lazyCidadeBean.lazyModel}"
    var="cidade"

```

```
itemLabel="#{cidade.nome} - #{cidade.estado}"
itemValue="#{cidade}"
converter="cidadeConverter"
forceSelection="true"
completeMethod="#{lazyCidadeBean.buscarCidadesLazy}"
maxResults="15"
placeholder="Busca lazy com paginação..."/>
```


## Principais vantagens desta implementação:

1. **Integração real com MySQL:** Usa JPA/Hibernate para persistência
2. **Named Queries:** Consultas otimizadas e centralizadas
3. **Cache inteligente:** ConcurrentHashMap no converter evita buscas repetidas <sup>2</sup>
4. **Pool de conexões:** HikariCP para alta performance
5. **Índices no banco:** Otimização das consultas LIKE
6. **Dados iniciais:** Popula automaticamente com cidades do IBGE
7. **Múltipla seleção:** Suporte a seleção de várias cidades
8. **Estatísticas em tempo real:** Mostra informações das seleções
9. **Opção LazyDataModel:** Para casos com muitos dados <sup>3</sup>

## Para testar:

1. Crie o banco MySQL: `CREATE DATABASE exemplo_db;`
2. Ajuste a senha no `persistence.xml` e `context.xml`
3. Execute `mvn clean package`
4. Deploy no Tomcat 11
5. Acesse: `http://localhost:8080/primefaces-jpa-autocomplete/autocomplete.xhtml`

O exemplo já popula automaticamente a tabela com dados do IBGE na primeira execução, permitindo testes imediatos.

 3 web pages

Usando repository

Aqui está um exemplo completo usando o padrão **Repository** com Spring Data JPA (mesmo sem Spring, podemos implementar nosso próprio repositório genérico)



integrado com PrimeFaces 14, Jakarta e MySQL.

## 1. Dependências (pom.xml)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.exemplo</groupId>
  <artifactId>primefaces-repository</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <primefaces.version>14.0.0</primefaces.version>
  </properties>

  <dependencies>
    <!-- PrimeFaces -->
    <dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>${primefaces.version}</version>
      <classifier>jakarta</classifier>
    </dependency>

    <!-- Jakarta EE -->
    <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>10.0.0</version>
      <scope>provided</scope>
    </dependency>

    <!-- JPA -->
    <dependency>
      <groupId>jakarta.persistence</groupId>
      <artifactId>jakarta.persistence-api</artifactId>
```

```

        <version>3.1.0</version>
    </dependency>

    <!-- Hibernate -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.3.1.Final</version>
    </dependency>

    <!-- MySQL -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.2.0</version>
    </dependency>

    <!-- HikariCP -->
    <dependency>
        <groupId>com.zaxxer</groupId>
        <artifactId>HikariCP</artifactId>
        <version>5.0.1</version>
    </dependency>
</dependencies>

<build>
    <finalName>primefaces-repository</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.4.0</version>
        </plugin>
    </plugins>
</build>
</project>

```

## 2. Entidade JPA (Cidade.java)

```
java
```

```

package com.exemplo.model;

import jakarta.persistence.*;
import java.io.Serializable;
import java.time.LocalDateTime;

```

```
import java.util.Objects;

@Entity
@Table(name = "cidades",
    indexes = {
        @Index(name = "idx_nome", columnList = "nome"),
        @Index(name = "idx_estado", columnList = "estado"),
        @Index(name = "idx_busca", columnList = "nome, estado")
    })
@NamedQueries({
    @NamedQuery(name = "Cidade.findByNomeContainingIgnoreCase",
        query = "SELECT c FROM Cidade c WHERE LOWER(c.nome) LIKE LOWER(CONCAT('%', :nome, '%'))"),
    @NamedQuery(name = "Cidade.findByNomeOrEstado",
        query = "SELECT c FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro OR LOWER(c.estado) LIKE :filtro ORDER BY c.nome"),
    @NamedQuery(name = "Cidade.countByNomeOrEstado",
        query = "SELECT COUNT(c) FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro OR LOWER(c.estado) LIKE :filtro")
})
public class Cidade implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String nome;

    @Column(nullable = false, length = 2)
    private String estado;

    @Column(length = 50)
    private String regiao;

    private Integer populacao;

    @Column(name = "data_cadastro")
    private LocalDateTime dataCadastro;

    private Boolean ativo = true;

    // Construtores
    public Cidade() {
        this.dataCadastro = LocalDateTime.now();
    }
}
```

```
public Cidade(String nome, String estado, String regioao, Integer populacao) {
    this.nome = nome;
    this.estado = estado;
    this.regiao = regioao;
    this.populacao = populacao;
    this.dataCadastro = LocalDateTime.now();
    this.ativo = true;
}

// Métodos de negócio
public String getNomeCompleto() {
    return nome + " - " + estado;
}

public String getInformacaoCompleta() {
    return String.format("%s - %s (%s) - Pop: %,d",
        nome, estado, regioao, populacao != null ? populacao : 0);
}

// Getters e Setters
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getNome() {
    return nome;
}

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

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getRegiao() {
    return regioao;
}
```

```
public void setRegiao(String regiao) {
    this.regiao = regiao;
}

public Integer getPopulacao() {
    return populacao;
}

public void setPopulacao(Integer populacao) {
    this.populacao = populacao;
}

public LocalDateTime getDataCadastro() {
    return dataCadastro;
}

public void setDataCadastro(LocalDateTime dataCadastro) {
    this.dataCadastro = dataCadastro;
}

public Boolean getAtivo() {
    return ativo;
}

public void setAtivo(Boolean ativo) {
    this.ativo = ativo;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Cidade cidade = (Cidade) o;
    return Objects.equals(id, cidade.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}

@Override
public String toString() {
    return nome + " - " + estado;
}
```

```
}  
}
```

### 3. Interface Repository Genérica (IRepository.java)

```
java  
  
package com.exemplo.repository;  
  
import java.io.Serializable;  
import java.util.List;  
import java.util.Optional;  
  
/**  
 * Interface genérica para o padrão Repository  
 */  
public interface IRepository<T, ID extends Serializable> {  
  
    T salvar(T entity);  
  
    Optional<T> buscarPorId(ID id);  
  
    List<T> buscarTodos();  
  
    List<T> buscarTodos(int limite, int offset);  
  
    void deletar(T entity);  
  
    void deletarPorId(ID id);  
  
    long contar();  
  
    boolean existe(ID id);  
}
```

### 4. Implementação do Repository (CidadeRepository.java)

```
java  
  
package com.exemplo.repository;  
  
import com.exemplo.model.Cidade;  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;
```

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.*;

@Named
@ApplicationScoped
public class CidadeRepository implements IRepository<Cidade, Long>, Serializable {

    private EntityManagerFactory emf;
    private EntityManager em;

    @PostConstruct
    public void init() {
        emf = Persistence.createEntityManagerFactory("repositoryPU");
        em = emf.createEntityManager();

        // Verifica se precisa popular dados iniciais
        if (contar() == 0) {
            popularDadosIniciais();
        }
    }

    @PreDestroy
    public void destroy() {
        if (em != null && em.isOpen()) {
            em.close();
        }
        if (emf != null && emf.isOpen()) {
            emf.close();
        }
    }

    // Métodos específicos do CidadeRepository
    public List<Cidade> buscarPorNomeOuEstado(String filtro, int maxResults) {
        String filtroLike = filtro.toLowerCase() + "%";

        TypedQuery<Cidade> query = em.createNamedQuery("Cidade.findByNomeOuEstado",
Cidade.class)
            .setParameter("filtro", filtroLike)
            .setMaxResults(maxResults)
            .setHint("jakarta.persistence.cache.retrieveMode", CacheMode.USE);

        return query.getResultList();
    }
}
```

```
public List<Cidade> buscarPorRegiao(String regiao) {
    return em.createQuery("SELECT c FROM Cidade c WHERE c.regiao = :regiao", Ci
dade.class)
        .setParameter("regiao", regiao)
        .getResultList();
}

public List<Cidade> buscarPopulosas(int minimoHabitantes) {
    return em.createQuery("SELECT c FROM Cidade c WHERE c.populacao > :populaca
o", Cidade.class)
        .setParameter("populacao", minimoHabitantes)
        .getResultList();
}

public Map<String, Object> buscarPaginado(String filtro, int pagina, int tamanh
oPagina) {
    Map<String, Object> resultado = new HashMap<>();
    String filtroLike = "%" + filtro.toLowerCase() + "%";

    // Busca paginada
    TypedQuery<Cidade> query = em.createNamedQuery("Cidade.findByNomeContaining
IgnoreCase", Cidade.class)
        .setParameter("nome", filtro)
        .setFirstResult(pagina * tamanhoPagina)
        .setMaxResults(tamanhoPagina);

    // Total de registros
    TypedQuery<Long> countQuery = em.createQuery(
        "SELECT COUNT(c) FROM Cidade c WHERE LOWER(c.nome) LIKE :filtro", L
ong.class)
        .setParameter("filtro", "%" + filtro.toLowerCase() + "%");

    resultado.put("dados", query.getResultList());
    resultado.put("total", countQuery.getSingleResult());

    return resultado;
}

public List<Cidade> buscarPorIds(List<Long> ids) {
    if (ids == null || ids.isEmpty()) {
        return Collections.emptyList();
    }

    return em.createQuery("SELECT c FROM Cidade c WHERE c.id IN :ids", Cidade.c
lass)
        .setParameter("ids", ids)
        .getResultList();
}
```



```
}

// Implementação dos métodos genéricos
@Override
public Cidade salvar(Cidade cidade) {
    EntityManager tx = em.getTransaction();
    try {
        tx.begin();
        if (cidade.getId() == null) {
            em.persist(cidade);
        } else {
            cidade = em.merge(cidade);
        }
        tx.commit();
        return cidade;
    } catch (Exception e) {
        if (tx.isActive()) {
            tx.rollback();
        }
        throw new RuntimeException("Erro ao salvar cidade", e);
    }
}

@Override
public Optional<Cidade> buscarPorId(Long id) {
    Cidade cidade = em.find(Cidade.class, id);
    return Optional.ofNullable(cidade);
}

@Override
public List<Cidade> buscarTodos() {
    return em.createQuery("SELECT c FROM Cidade c ORDER BY c.nome", Cidade.class)
        .getResultList();
}

@Override
public List<Cidade> buscarTodos(int limite, int offset) {
    return em.createQuery("SELECT c FROM Cidade c ORDER BY c.nome", Cidade.class)
        .setFirstResult(offset)
        .setMaxResults(limite)
        .getResultList();
}

@Override
public void deletar(Cidade cidade) {
```

```
EntityTransaction tx = em.getTransaction();
try {
    tx.begin();
    if (!em.contains(cidade)) {
        cidade = em.merge(cidade);
    }
    em.remove(cidade);
    tx.commit();
} catch (Exception e) {
    if (tx.isActive()) {
        tx.rollback();
    }
    throw new RuntimeException("Erro ao deletar cidade", e);
}
}

@Override
public void deletarPorId(Long id) {
    buscarPorId(id).ifPresent(this::deletar);
}

@Override
public long contar() {
    return em.createQuery("SELECT COUNT(c) FROM Cidade c", Long.class)
        .getSingleResult();
}

@Override
public boolean existe(Long id) {
    return buscarPorId(id).isPresent();
}

// Método para popular dados iniciais
private void popularDadosIniciais() {
    EntityTransaction tx = em.getTransaction();
    try {
        tx.begin();

        String[][] cidadesIBGE = {
            {"São Paulo", "SP", "Sudeste", "12300000"},
            {"Rio de Janeiro", "RJ", "Sudeste", "6748000"},
            {"Brasília", "DF", "Centro-Oeste", "3055000"},
            {"Salvador", "BA", "Nordeste", "2887000"},
            {"Fortaleza", "CE", "Nordeste", "2687000"},
            {"Belo Horizonte", "MG", "Sudeste", "2523000"},
            {"Manaus", "AM", "Norte", "2219000"},
            {"Curitiba", "PR", "Sul", "1963000"},
        };
    }
}
```

```

        {"Recife", "PE", "Nordeste", "1655000"},
        {"Goiânia", "GO", "Centro-Oeste", "1536000"},
        {"Porto Alegre", "RS", "Sul", "1488000"},
        {"Belém", "PA", "Norte", "1499000"},
        {"Guarulhos", "SP", "Sudeste", "1392000"},
        {"Campinas", "SP", "Sudeste", "1214000"},
        {"São Luís", "MA", "Nordeste", "1109000"},
        {"São Gonçalo", "RJ", "Sudeste", "1084000"},
        {"Maceió", "AL", "Nordeste", "1025000"},
        {"Campo Grande", "MS", "Centro-Oeste", "906000"},
        {"Natal", "RN", "Nordeste", "890000"},
        {"Teresina", "PI", "Nordeste", "868000"}
    };

    for (String[] dados : cidadesIBGE) {
        Cidade cidade = new Cidade(
            dados[0],
            dados[1],
            dados[2],
            Integer.parseInt(dados[3])
        );
        em.persist(cidade);
    }

    tx.commit();
    System.out.println("✅ Dados iniciais inseridos com sucesso!");

} catch (Exception e) {
    if (tx.isActive()) {
        tx.rollback();
    }
    e.printStackTrace();
}
}
}

```

## 5. Service usando Repository (CidadeService.java)

```

java

package com.exemplo.service;

import com.exemplo.model.Cidade;
import com.exemplo.repository.CidadeRepository;
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Inject;

```

```
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.List;
import java.util.Optional;

@Named
@RequestScoped
public class CidadeService implements Serializable {

    @Inject
    private CidadeRepository cidadeRepository;

    /**
     * Método para o AutoComplete - busca no banco via Repository
     */
    public List<Cidade> pesquisarCidades(String query) {
        if (query == null || query.trim().length() < 2) {
            return List.of();
        }

        long inicio = System.currentTimeMillis();

        // Usa o método otimizado do repository
        List<Cidade> resultados = cidadeRepository.buscarPorNomeOuEstado(query, 2
0);

        long fim = System.currentTimeMillis();
        System.out.printf("🔍 Busca por '%s': %d resultados em %dms%n",
            query, resultados.size(), (fim - inicio));

        return resultados;
    }

    /**
     * Busca cidade por ID
     */
    public Cidade buscarPorId(Long id) {
        if (id == null) return null;

        Optional<Cidade> resultado = cidadeRepository.buscarPorId(id);
        return resultado.orElse(null);
    }

    /**
     * Busca múltiplas cidades por IDs
     */
    public List<Cidade> buscarPorIds(List<Long> ids) {
```

```
        return cidadeRepository.buscarPorIds(ids);
    }

    /**
     * Busca cidades por região
     */
    public List<Cidade> buscarPorRegiao(String regiao) {
        return cidadeRepository.buscarPorRegiao(regiao);
    }

    /**
     * Salva uma nova cidade
     */
    public Cidade salvarCidade(Cidade cidade) {
        return cidadeRepository.salvar(cidade);
    }

    /**
     * Deleta uma cidade
     */
    public void deletarCidade(Long id) {
        cidadeRepository.deletarPorId(id);
    }

    /**
     * Conta total de cidades
     */
    public long contarCidades() {
        return cidadeRepository.contar();
    }
}
```

## 6. Converter (CidadeConverter.java)

java

```
package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;
```

```
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;

@Named
@FacesConverter(value = "cidadeConverter", managed = true)
public class CidadeConverter implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    // Cache com estatísticas
    private static final Map<Long, Cidade> CACHE = new ConcurrentHashMap<>();
    private static final int CACHE_MAX_SIZE = 500;
    private static final AtomicInteger CACHE_HITS = new AtomicInteger(0);
    private static final AtomicInteger CACHE_MISSES = new AtomicInteger(0);

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Long id = Long.parseLong(value);

            // Tenta buscar no cache primeiro
            Cidade cidade = CACHE.get(id);

            if (cidade != null) {
                CACHE_HITS.incrementAndGet();
                return cidade;
            }

            // Cache miss - busca no banco via repository
            CACHE_MISSES.incrementAndGet();
            cidade = cidadeService.buscarPorId(id);

            // Adiciona ao cache se encontrou
            if (cidade != null) {
                // Limpa cache se necessário
                if (CACHE.size() >= CACHE_MAX_SIZE) {
                    CACHE.clear();
                    System.out.println("🧹 Cache limpo - " + CACHE.size() + " itens removidos");
                }
            }
        }
    }
}
```

```

        CACHE.put(id, cidade);
    }

    return cidade;

} catch (NumberFormatException e) {
    return null;
}
}

@Override
public String getAsString(FacesContext context, UIComponent component, Cidade cidade) {
    if (cidade == null || cidade.getId() == null) {
        return "";
    }

    // Adiciona ao cache para conversões futuras (se não existir)
    CACHE.putIfAbsent(cidade.getId(), cidade);

    return String.valueOf(cidade.getId());
}

/**
 * Método para estatísticas do cache
 */
public static String getCacheStats() {
    return String.format("Cache: %d itens, Hits: %d, Misses: %d, Hit Ratio: %.2f%%",
        CACHE.size(),
        CACHE_HITS.get(),
        CACHE_MISSES.get(),
        CACHE_HITS.get() > 0 ?
            (double) CACHE_HITS.get() / (CACHE_HITS.get() + CACHE_MISSES.get())
            * 100 : 0);
}
}

```

## 7. Controller (AutoCompleteBean.java)

```

java

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;

```

```
import com.exemplo.converter.CidadeConverter;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

    private Cidade cidadeSelecionada;
    private List<Cidade> cidadesMultiplas;
    private String estatisticas;
    private String regiaoFiltro;
    private Long totalCidades;

    public AutoCompleteBean() {
        cidadesMultiplas = new ArrayList<>();
    }

    /**
     * Método chamado pelo AutoComplete (usa o repository)
     */
    public List<Cidade> buscarCidades(String query) {
        return cidadeService.pesquisarCidades(query);
    }

    /**
     * Busca por região (exemplo de método específico do repository)
     */
    public void buscarPorRegiao() {
        if (regiaoFiltro != null && !regiaoFiltro.isEmpty()) {
            List<Cidade> cidades = cidadeService.buscarPorRegiao(regiaoFiltro);
            FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage("Região " + regiaoFiltro,
                    "Encontradas " + cidades.size() + " cidades"));
        }
    }

    /**
```



```
* Evento ao selecionar uma cidade
*/
public void aoSelecionarCidade() {
    if (cidadeSelecionada != null) {
        String msg = String.format("✅ Cidade: %s - %s | População: %,d | Região: %s",
            cidadeSelecionada.getNome(),
            cidadeSelecionada.getEstado(),
            cidadeSelecionada.getPopulacao() != null ? cidadeSelecionada.getPopulacao() : 0,
            cidadeSelecionada.getRegiao());

        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO, "Selecionada", msg));

        atualizarEstatisticas();
    }
}

/**
 * Mostra cidades selecionadas (múltiplas)
 */
public void mostrarMultiplas() {
    if (cidadesMultiplas == null || cidadesMultiplas.isEmpty()) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_WARN, "Aviso", "Nenhuma cidade selecionada"));
        return;
    }

    StringBuilder sb = new StringBuilder();
    long populacaoTotal = 0;

    for (Cidade c : cidadesMultiplas) {
        sb.append("• ").append(c.getNomeCompleto()).append("\n");
        populacaoTotal += c.getPopulacao() != null ? c.getPopulacao() : 0;
    }

    String msg = String.format("📍 %d cidades selecionadas\nPopulação total: %,d",
        cidadesMultiplas.size(), populacaoTotal);

    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Múltiplas cidades", msg));
}
```

```
/**
 * Atualiza estatísticas
 */
private void atualizarEstatisticas() {
    totalCidades = cidadeService.contarCidades();
    estatisticas = String.format("📊 Total no banco: %,d cidades | %s",
        totalCidades, CidadeConverter.getCacheStats());
}

/**
 * Limpa seleções
 */
public void limpar() {
    cidadeSelecionada = null;
    cidadesMultiplas.clear();
    regioaoFiltro = null;
    atualizarEstatisticas();

    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Limpo", "Seleções removidas"));
}

// Getters e Setters
public Cidade getCidadeSelecionada() {
    return cidadeSelecionada;
}

public void setCidadeSelecionada(Cidade cidadeSelecionada) {
    this.cidadeSelecionada = cidadeSelecionada;
}

public List<Cidade> getCidadesMultiplas() {
    return cidadesMultiplas;
}

public void setCidadesMultiplas(List<Cidade> cidadesMultiplas) {
    this.cidadesMultiplas = cidadesMultiplas;
}

public String getEstatisticas() {
    if (estatisticas == null) {
        atualizarEstatisticas();
    }
    return estatisticas;
}
```

```
public String getRegiaoFiltro() {
    return regiaoFiltro;
}

public void setRegiaoFiltro(String regiaoFiltro) {
    this.regiaoFiltro = regiaoFiltro;
}

public Long getTotalCidades() {
    if (totalCidades == null) {
        totalCidades = cidadeService.contarCidades();
    }
    return totalCidades;
}
}
```

## 8. Página XHTML (autocomplete.xhtml)

xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:p="primefaces">

<h:head>
<title>AutoComplete com Repository - PrimeFaces 14</title>
<style>
    :root {
        --primary: #4361ee;
        --secondary: #3f37c9;
        --success: #4cc9f0;
        --info: #4895ef;
        --warning: #f72585;
    }

    .header-panel {
        background: linear-gradient(135deg, var(--primary), var(--secondary));
        color: white;
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 20px;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
    }
```

```

.stats-panel {
    background: #f8f9fa;
    border-left: 4px solid var(--success);
    padding: 15px;
    margin: 20px 0;
    border-radius: 5px;
    font-family: 'Courier New', monospace;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}

.label {
    font-weight: bold;
    color: #495057;
}

.badge {
    background: var(--info);
    color: white;
    padding: 3px 8px;
    border-radius: 12px;
    font-size: 0.8em;
}
</style>
</h:head>

<h:body>
    <h:form>
        <p:growl id="growl" showDetail="true" life="5000" autoUpdate="true"/>

        <div class="header-panel">
            <h1>🏠 AutoComplete com Padrão Repository</h1>
            <p>Integração com JPA/MySQL usando Repository Pattern</p>
            <p class="badge">Total no banco: #{autoCompleteBean.totalCidades} cidad
es</p>
        </div>

        <p:panel header="🔍 Busca de Cidades" style="margin:20px">

            <h:panelGrid columns="2" cellpadding="10" columnClasses="label,field" s
tyle="width:100%">

                <!-- Seleção Simples -->
                <p:outputLabel for="simple" value="Seleção Simples:"/>
                <p:autoComplete id="simple"
                    value="#{autoCompleteBean.cidadeSelecionada}"
                    completeMethod="#{autoCompleteBean.buscarCidades}"

```

```

var="cidade"
itemLabel="#{cidade.nome} - #{cidade.estado}"
itemValue="#{cidade}"
converter="cidadeConverter"
forceSelection="true"
dropdown="true"
minQueryLength="2"
queryDelay="400"
maxResults="15"
placeholder="Digite o nome da cidade..."
panelStyle="width:450px"
emptyMessage="Nenhuma cidade encontrada">

<p:ajax event="itemSelect"
    listener="#{autoCompleteBean.aoSelecionarCidade}"
    update="growl stats"/>

<f:facet name="item">
    <h:panelGrid columns="4" style="width:100%">
        <h:outputText value="#{cidade.nome}" style="font-weight:bold"/>
        <h:outputText value="#{cidade.estado}" style="color:#666"/>
        <h:outputText value="#{cidade.regiao}" style="color:var(--info)"/>
        <h:outputText value="#{cidade.populacao}" style="color:var(--success)">
            <f:convertNumber type="number" groupingUsed="true"/>
    </h:outputText>
</h:panelGrid>
</f:facet>
</p:autoComplete>

<!-- Seleção Múltipla -->
<p:outputLabel for="multiple" value="Seleção Múltipla:"/>
<p:autoComplete id="multiple"
    value="#{autoCompleteBean.cidadesMultiplas}"
    completeMethod="#{autoCompleteBean.buscarCidades}"
    var="cidade"
    itemLabel="#{cidade.nome} - #{cidade.estado}"
    itemValue="#{cidade}"
    converter="cidadeConverter"
    multiple="true"
    forceSelection="true"
    minQueryLength="2"
    maxResults="15"

```

```

placeholder="Selecione várias cidades..."
panelStyle="width:450px">

<f:facet name="item">
    <h:panelGrid columns="3" style="width:100%">
        <h:outputText value="#{cidade.nome}" style="font-weight:bold"/>

        <h:outputText value="#{cidade.estado}" style="color:#666"/>

        <h:outputText value="#{cidade.regiao}" style="color:var(--info)"/>
    </h:panelGrid>
</f:facet>
</p:autoComplete>

<!-- Filtro por Região -->
<p:outputLabel for="regiao" value="Filtro por Região:"/>
<h:panelGroup>
    <p:selectOneMenu id="regiao" value="#{autoCompleteBean.regiaoFiltro}" style="width:200px">
        <f:selectItem itemLabel="Selecione..." itemValue=""/>
        <f:selectItem itemLabel="Sudeste" itemValue="Sudeste"/>
        <f:selectItem itemLabel="Sul" itemValue="Sul"/>
        <f:selectItem itemLabel="Nordeste" itemValue="Nordeste"/>
        <f:selectItem itemLabel="Centro-Oeste" itemValue="Centro-Oeste"/>

        <f:selectItem itemLabel="Norte" itemValue="Norte"/>
        <p:ajax listener="#{autoCompleteBean.buscarPorRegiao}" update="growl"/>
    </p:selectOneMenu>
</h:panelGroup>

<!-- Botões -->
<p:outputLabel value="Ações:"/>
<h:panelGroup>
    <p:commandButton value="Mostrar Múltiplas"
        actionListener="#{autoCompleteBean.mostrarMultiplas}"
        update="growl"
        icon="pi pi-check"
        style="margin-right:10px">
        <p:confirm header="Confirmação" message="Mostrar cidades selecionadas?"
            icon="pi pi-question-circle"/>
    </p:commandButton>

    <p:commandButton value="Limpar"

```

```

        actionListener="#{autoCompleteBean.limpar}"
        update="@form"
        icon="pi pi-refresh"
        style="margin-right:10px"/>

        <p:commandButton value="Atualizar Stats"
            actionListener="#{autoCompleteBean.atualizarEsta
tisticas}"

            update="stats"
            icon="pi pi-chart-bar"/>
    </h:panelGroup>

</h:panelGrid>

<p:separator/>

<!-- Painel de Estatísticas -->
<p:outputPanel id="stats">
    <h:panelGroup rendered="#{not empty autoCompleteBean.estadistica
s}">

        <div class="stats-panel">
            <h:outputText value="#{autoCompleteBean.estadisticas}" esca
pe="false"/>

        </div>
    </h:panelGroup>
</p:outputPanel>

<!-- Seção de informações do Repository -->
<p:fieldset legend="📦 Padrão Repository" toggleable="true" toggleSpeed
="500">

    <h:panelGrid columns="2" style="width:100%">
        <h:outputText value="Repository:"/>
        <h:outputText value="CidadeRepository (implementa IRepository)"
style="font-weight:bold"/>

        <h:outputText value="Métodos específicos:"/>
        <h:outputText value="buscarPorNomeOuEstado(), buscarPorRegiao
()" style="font-weight:bold"/>

        <h:outputText value="Cache no converter:"/>
        <h:outputText value="ConcurrentHashMap com estatísticas" style
="font-weight:bold"/>

        <h:outputText value="Pool de conexões:"/>
        <h:outputText value="HikariCP (via persistence.xml)" style="fon
t-weight:bold"/>

```

```

        <h:outputText value="Named Queries:"/>
        <h:outputText value="Otimizadas com índices no banco" style="font-weight:bold"/>
    </h:panelGrid>
</p:fieldset>

</p:panel>

<p:confirmDialog global="true" showEffect="fade" hideEffect="fade">
    <p:commandButton value="Sim" type="button" style="margin-right:10px"
        onclick="PF('confirmDialog').hide()"/>
    <p:commandButton value="Não" type="button"
        onclick="PF('confirmDialog').hide()"/>
</p:confirmDialog>
</h:form>
</h:body>
</html>

```

## 9. persistence.xml (atualizado)

```

xml

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
    xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
        https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">

    <persistence-unit name="repositoryPU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.Hibernate

```

Simplifique usando só a versão comm filtro por nome

Aqui está uma versão **simplificada** usando apenas o filtro por nome, com o padrão Repository:

## 1. Entidade (Cidade.java)

```

java

package com.exemplo.model;

```



```
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.Objects;

@Entity
@Table(name = "cidades")
public class Cidade implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String nome;

    @Column(nullable = false, length = 2)
    private String estado;

    public Cidade() {}

    public Cidade(String nome, String estado) {
        this.nome = nome;
        this.estado = estado;
    }

    // Getters e Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
```

```
        this.estado = estado;
    }

    public String getNomeCompleto() {
        return nome + " - " + estado;
    }

    @Override
    public String toString() {
        return nome + " - " + estado;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Cidade cidade = (Cidade) o;
        return Objects.equals(id, cidade.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

## 2. Repository (CidadeRepository.java)

java

```
package com.exemplo.repository;

import com.exemplo.model.Cidade;
import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.List;
import java.util.Optional;

@Named
@ApplicationScoped
public class CidadeRepository implements Serializable {
```

```
private EntityManagerFactory emf;
private EntityManager em;

@PostConstruct
public void init() {
    emf = Persistence.createEntityManagerFactory("cidadesPU");
    em = emf.createEntityManager();
    popularDadosIniciais();
}

@PreDestroy
public void destroy() {
    if (em != null && em.isOpen()) em.close();
    if (emf != null && emf.isOpen()) emf.close();
}

/**
 * Busca cidades por nome (ignorando maiúsculas/minúsculas)
 */
public List<Cidade> buscarPorNome(String nome, int limite) {
    String sql = "SELECT c FROM Cidade c WHERE LOWER(c.nome) LIKE LOWER(:nome)
ORDER BY c.nome";

    return em.createQuery(sql, Cidade.class)
        .setParameter("nome", nome + "%")
        .setMaxResults(limite)
        .getResultList();
}

/**
 * Busca cidade por ID
 */
public Optional<Cidade> buscarPorId(Long id) {
    Cidade cidade = em.find(Cidade.class, id);
    return Optional.ofNullable(cidade);
}

/**
 * Conta total de cidades
 */
public long contar() {
    return em.createQuery("SELECT COUNT(c) FROM Cidade c", Long.class)
        .getSingleResult();
}

private void popularDadosIniciais() {
    if (contar() > 0) return;
}
```

```

String[][] cidades = {
    {"São Paulo", "SP"}, {"Rio de Janeiro", "RJ"}, {"Belo Horizonte", "M
G"},

    {"Brasília", "DF"}, {"Salvador", "BA"}, {"Fortaleza", "CE"},
    {"Curitiba", "PR"}, {"Manaus", "AM"}, {"Recife", "PE"},
    {"Porto Alegre", "RS"}, {"Belém", "PA"}, {"Goiânia", "GO"},
    {"Guarulhos", "SP"}, {"Campinas", "SP"}, {"São Luís", "MA"},
    {"Maceió", "AL"}, {"Campo Grande", "MS"}, {"Natal", "RN"},
    {"Teresina", "PI"}, {"João Pessoa", "PB"}, {"Aracaju", "SE"},
    {"Cuiabá", "MT"}, {"Florianópolis", "SC"}, {"Vitória", "ES"}
};

EntityTransaction tx = em.getTransaction();
try {
    tx.begin();
    for (String[] dados : cidades) {
        em.persist(new Cidade(dados[0], dados[1]));
    }
    tx.commit();
} catch (Exception e) {
    if (tx.isActive()) tx.rollback();
    e.printStackTrace();
}
}
}

```

### 3. Service (CidadeService.java)

```

java

package com.exemplo.service;

import com.exemplo.model.Cidade;
import com.exemplo.repository.CidadeRepository;
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.List;

@Named
@RequestScoped
public class CidadeService implements Serializable {

    @Inject

```

```

private CidadeRepository cidadeRepository;

/**
 * Método chamado pelo AutoComplete
 */
public List<Cidade> buscarPorNome(String query) {
    if (query == null || query.trim().length() < 2) {
        return List.of();
    }
    return cidadeRepository.buscarPorNome(query.trim(), 15);
}

/**
 * Busca cidade por ID (usado pelo converter)
 */
public Cidade buscarPorId(Long id) {
    if (id == null) return null;
    return cidadeRepository.buscarPorId(id).orElse(null);
}
}

```

## 4. Converter (CidadeConverter.java)

java

```

package com.exemplo.converter;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.faces.component.UIComponent;
import jakarta.faces.context.FacesContext;
import jakarta.faces.convert.Converter;
import jakarta.faces.convert.FacesConverter;
import jakarta.inject.Inject;
import jakarta.inject.Named;

@Named
@FacesConverter(value = "cidadeConverter", managed = true)
public class CidadeConverter implements Converter<Cidade> {

    @Inject
    private CidadeService cidadeService;

    @Override
    public Cidade getAsObject(FacesContext context, UIComponent component, String value) {

```

```

        if (value == null || value.isEmpty()) {
            return null;
        }

        try {
            Long id = Long.parseLong(value);
            return cidadeService.buscarPorId(id);
        } catch (NumberFormatException e) {
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Cidade c
idade) {
        if (cidade == null || cidade.getId() == null) {
            return "";
        }
        return String.valueOf(cidade.getId());
    }
}

```

## 5. Controller (AutoCompleteBean.java)

```
java
```

```

package com.exemplo.controller;

import com.exemplo.model.Cidade;
import com.exemplo.service.CidadeService;
import jakarta.enterprise.context.RequestScoped;
import jakarta.faces.application.FacesMessage;
import jakarta.faces.context.FacesContext;
import jakarta.inject.Inject;
import jakarta.inject.Named;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Named
@RequestScoped
public class AutoCompleteBean implements Serializable {

    @Inject
    private CidadeService cidadeService;

```

```
private Cidade cidadeSelecionada;
private List<Cidade> cidadesMultiplas = new ArrayList<>();

/**
 * Chamado pelo AutoComplete
 */
public List<Cidade> buscarCidades(String query) {
    return cidadeService.buscarPorNome(query);
}

/**
 * Quando seleciona uma cidade
 */
public void aoSelecionar() {
    if (cidadeSelecionada != null) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("Selecionado", cidadeSelecionada.getNomeCompleto
()));
    }
}

/**
 * Mostra as cidades selecionadas
 */
public void mostrarMultiplas() {
    if (cidadesMultiplas.isEmpty()) {
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_WARN, "Aviso", "Nenhuma cida
de selecionada"));
        return;
    }

    StringBuilder sb = new StringBuilder("Cidades: ");
    for (Cidade c : cidadesMultiplas) {
        sb.append(c.getNome()).append(", ");
    }

    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage("Múltiplas", sb.toString()));
}

/**
 * Limpa tudo
 */
public void limpar() {
    cidadeSelecionada = null;
    cidadesMultiplas.clear();
}
```

```
}

// Getters e Setters
public Cidade getCidadeSelecionada() {
    return cidadeSelecionada;
}

public void setCidadeSelecionada(Cidade cidadeSelecionada) {
    this.cidadeSelecionada = cidadeSelecionada;
}

public List<Cidade> getCidadesMultiplas() {
    return cidadesMultiplas;
}

public void setCidadesMultiplas(List<Cidade> cidadesMultiplas) {
    this.cidadesMultiplas = cidadesMultiplas;
}
}
```

## 6. Página XHTML (index.xhtml)

xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:p="primefaces">

    <h:head>
        <title>AutoComplete Simplificado</title>
    </h:head>

    <h:body>
        <h:form>
            <p:growl id="growl" showDetail="true" life="3000"/>

            <p:panel header="Busca de Cidades" style="margin:20px; width:600px">

                <h:panelGrid columns="2" cellpadding="10">

                    <!-- Seleção Simples -->
                    <p:outputLabel for="simples" value="Seleção Simples:"/>
                    <p:autoComplete id="simples">
```



```

        value="#{autoCompleteBean.cidadeSelecionada}"
        completeMethod="#{autoCompleteBean.buscarCidades}"
        var="cidade"
        itemLabel="#{cidade.nome} - #{cidade.estado}"
        itemValue="#{cidade}"
        converter="cidadeConverter"
        forceSelection="true"
        dropdown="true"
        minQueryLength="2"
        placeholder="Digite o nome..."
        panelStyle="width:300px">

    <p:ajax event="itemSelect"
        listener="#{autoCompleteBean.aoSelecionar}"
        update="growl"/>
</p:autoComplete>

<!-- Seleção Múltipla -->
<p:outputLabel for="multi" value="Seleção Múltipla:"/>
<p:autoComplete id="multi"
    value="#{autoCompleteBean.cidadesMultiplas}"
    completeMethod="#{autoCompleteBean.buscarCidades}"
    var="cidade"
    itemLabel="#{cidade.nome} - #{cidade.estado}"
    itemValue="#{cidade}"
    converter="cidadeConverter"
    multiple="true"
    forceSelection="true"
    minQueryLength="2"
    placeholder="Selecione várias..."
    panelStyle="width:300px">
</p:autoComplete>

<!-- Botões -->
<p:outputLabel value="Ações:"/>
<h:panelGroup>
    <p:commandButton value="Mostrar Múltiplas"
        actionListener="#{autoCompleteBean.mostrarMultip
las}"
        update="growl"
        icon="pi pi-check"
        style="margin-right:10px"/>

    <p:commandButton value="Limpar"
        actionListener="#{autoCompleteBean.limpar}"
        update="@form"
        icon="pi pi-refresh"/>

```

```

        </h:panelGroup>

    </h:panelGrid>

</p:panel>
</h:form>
</h:body>
</html>

```

## 7. persistence.xml (simples)

xml

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
    xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
    https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">

    <persistence-unit name="cidadesPU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>com.exemplo.model.Cidade</class>

        <properties>
            <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/exemplo_db?useSSL=false&serverTimezone=UTC"/>
            <property name="jakarta.persistence.jdbc.user" value="root"/>
            <property name="jakarta.persistence.jdbc.password" value="sua_senha"/>

            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>

```

## 8. Script SQL para criar o banco

sql

```
CREATE DATABASE IF NOT EXISTS exemplo_db;
USE exemplo_db;

CREATE TABLE IF NOT EXISTS cidades (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    estado VARCHAR(2) NOT NULL,
    INDEX idx_nome (nome)
);
```

## 9. beans.xml (para ativar CDI)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
        https://jakarta.ee/xml/ns/jakartaee/beans_4_0.xsd"
    version="4.0"
    bean-discovery-mode="annotated">
</beans>
```

## Estrutura final do projeto:

text







```
src/
├─ main/
│   └─ java/
│       └─ com/
│           └─ exemplo/
│               └─ model/
│                   └─ Cidade.java
│               └─ repository/
│                   └─ CidadeRepository.java
│               └─ service/
│                   └─ CidadeService.java
│               └─ converter/
│                   └─ CidadeConverter.java
│               └─ controller/
│                   └─ AutoCompleteBean.java
└─ webapp/
    └─ WEB-INF/
```

```
|      |  └─ web.xml
|      |  └─ beans.xml
|      └─ META-INF/
|      └─ persistence.xml
└─ index.xhtml
```

## Como testar:

1. Crie o banco MySQL: `CREATE DATABASE exemplo_db;`
2. Ajuste a senha no `persistence.xml`
3. Execute `mvn clean package`
4. Copie o WAR para o Tomcat 11
5. Acesse: `http://localhost:8080/seu-app/index.xhtml`

## Características desta versão simplificada:

-  Apenas filtro por nome
-  Repository pattern implementado
-  24 cidades iniciais populadas automaticamente
-  Suporte a seleção simples e múltipla
-  Converter funcional
-  Código mínimo e direto ao ponto