



Data
30/07/2019 09:25:25

Setor de Origem
PF - PF-DEPEX

Tipo
Ensino: Projeto de Ensino

Assunto
Solicitação de prorrogação do projeto de ensino PJE2018 PFU 0066.

Interessados
Jorge Luis Boeira Bavaresco, Rafael Marisco Bertei

Situação
Em trâmite

Trâmites

- 11/12/2019 09:40
Recebido por: IF-PROEN: Leonardo Olsen de Campos Silva
- 25/11/2019 20:32
Enviado por: IF-DIRPEI: Veridiana Krolow Bosenbecker
- 21/11/2019 14:33
Recebido por: IF-DIRPEI: Veridiana Krolow Bosenbecker
- 07/11/2019 14:25
Enviado por: IF-PROEN: Leonardo Olsen de Campos Silva
- 07/11/2019 14:24
Recebido por: IF-PROEN: Leonardo Olsen de Campos Silva
- 06/11/2019 21:19
Enviado por: PF-DIRGER: Alexandre Pitol Boeira
- 06/11/2019 17:39
Recebido por: PF-DIRGER: Alexandre Pitol Boeira
- 06/11/2019 10:29
Enviado por: PF-DEPEX: Maria Carolina Fortes
- 01/11/2019 00:14
Recebido por: PF-DEPEX: Claudio Andre Lopes de Oliveira
- 29/10/2019 21:20
Enviado por: PF-COCSIST: Rafael Marisco Bertei

- 24/10/2019 20:20
Recebido por: PF-COCERSIST: Rafael Marisco Bertei
- 24/10/2019 20:08
Enviado por: PF-DEPEX: Claudio Andre Lopes de Oliveira
- 24/10/2019 20:06
Recebido por: PF-DEPEX: Claudio Andre Lopes de Oliveira
- 22/10/2019 16:48
Enviado por: IF-DIRPEI: Ana Carolina Madeira Hise
- 21/10/2019 17:36
Recebido por: IF-DIRPEI: Magno Souza Grillo
- 21/10/2019 17:36
Enviado por: IF-PROEN: Magno Souza Grillo
- 21/10/2019 16:46
Recebido por: IF-PROEN: Magno Souza Grillo
- 20/10/2019 20:40
Enviado por: IF-DIRPEI: Veridiana Krolow Bosenbecker
- 20/10/2019 20:29
Recebido por: IF-DIRPEI: Veridiana Krolow Bosenbecker
- 11/10/2019 14:26
Enviado por: IF-PROEN: Magno Souza Grillo
- 11/10/2019 14:25
Recebido por: IF-PROEN: Magno Souza Grillo
- 10/10/2019 22:19
Enviado por: PF-DIRGER: Alexandre Pitol Boeira
- 10/10/2019 22:02
Recebido por: PF-DIRGER: Alexandre Pitol Boeira
- 10/10/2019 21:35
Enviado por: PF-DEPEX: Maria Carolina Fortes
- 10/10/2019 21:34
Recebido por: PF-DEPEX: Maria Carolina Fortes
- 10/10/2019 11:18
Enviado por: IF-PROEN: Magno Souza Grillo

- 10/10/2019 08:10
Recebido por: IF-PROEN: Magno Souza Grillo
- 07/10/2019 16:48
Enviado por: PF-DEPEX: Maria Carolina Fortes
- 07/10/2019 16:46
Recebido por: PF-DEPEX: Maria Carolina Fortes
- 07/10/2019 15:28
Enviado por: PF-COCERSIST: Rafael Marisco Bertei
- 15/08/2019 20:15
Recebido por: PF-COCERSIST: Rafael Marisco Bertei
- 15/08/2019 18:07
Enviado por: PF-DEPEX: Maria Carolina Fortes
- 15/08/2019 18:06
Recebido por: PF-DEPEX: Maria Carolina Fortes
- 15/08/2019 12:22
Enviado por: IF-DIRPEI: Magno Souza Grillo
- 14/08/2019 18:42
Recebido por: IF-DIRPEI: Magno Souza Grillo
- 01/08/2019 15:16
Enviado por: IF-PROEN: Rodrigo Nascimento da Silva
- 01/08/2019 15:16
Recebido por: IF-PROEN: Rodrigo Nascimento da Silva
- 31/07/2019 11:49
Enviado por: PF-DEPEX: Maria Carolina Fortes
- 31/07/2019 11:47
Recebido por: PF-DEPEX: Maria Carolina Fortes
- 30/07/2019 09:27
Enviado por: PF-DEPEX: Jorge Luis Boeira Bavaresco



MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE
PRÓ-REITORIA DE ENSINO

FORMULÁRIO PARA PRORROGAÇÃO DE PRAZO EM PROJETO DE ENSINO

REGISTRO SOB N° : PJE2018 PFU 0066

Informar número de registro do projeto de ensino ou processo eletrônico.

I.

IDENTIFICAÇÃO

a. Título do Projeto:

Desenvolvimento de sistemas com a plataforma Java EE

b. Coordenador do Projeto:

Jorge Luis Boeira Bavaresco

II.

SOLICITAÇÃO DE PRORROGAÇÃO DE PRAZO

Período de prorrogação:

Prorrogação de oito (08) meses, até dezembro 2019.

Justificativa:

Mudanças foram necessárias devido a atualizações das tecnologias utilizadas. Desta forma parte do trabalho deve ser atualizado ou refeito. Além deste fato, o prazo para conclusão teve atrasos devidos a licença capacitação do coordenador do projeto. Devido a este fatos é necessária a prorrogação do projeto de ensino para que seja possível a sua conclusão.

III. NOVO CRONOGRAMA DE EXECUÇÃO

Atividades	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6	Mês 7	Mês 8	Mês 9	Mês 10	Mês 11	Mês 12
1	X	X	X									
2				X	X	X						
3						X	X	X				

Descrição das atividades:

Atividade 1: Estudo para atualização das tecnologias vigentes. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 2: Escrita dos capítulos restantes do livro. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 3: Procedimentos para publicação e avaliação do projeto.

PARECERES NECESSÁRIOS NO PROCESSO DO SUAP

- PARECER COLEGIADO/COORDENAÇÃO/ÁREA.
- PARECER DIREÇÃO/DEPARTAMENTO DE ENSINO.
- PARECER DIREÇÃO/DEPARTAMENTO DE ADMINISTRAÇÃO E PLANEJAMENTO (Quando necessário).
- PARECER DIREÇÃO-GERAL DO CAMPUS.
- PARECER DA PRÓ-REITORIA DE ENSINO.

OBS:

Não esquecer de enviar o relatório com as atividades já realizadas. Utilizar modelo de Relatório Final.

Não esquecer de enviar a ata de aprovação da prorrogação pelo Colegiado/Coordenação do Curso.

30 de julho de 2019

Documento assinado eletronicamente por:

- **Jorge Luis Boeira Bavaresco, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 30/07/2019 09:22:01.

Este documento foi emitido pelo SUAP em 08/07/2019. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 28026

Código de Autenticação: ac9cd3d2d8





SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Encaminhamento de solicitação de prorrogação de projeto de ensino, para os devidos tramites.

Assinatura:

Despacho assinado eletronicamente por:

- Jorge Luis Boeira Bavaresco, Jorge Luis Boeira Bavaresco - PROFESSOR ENS BASICO TECN TECNOLOGICO, PF-DEPEX, em 30/07/2019 09:27:27.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

De acordo com a solicitação, considerando a importância do projeto para os estudantes envolvidos.

Assinatura:

Despacho assinado eletronicamente por:

- Maria Carolina Fortes, Maria Carolina Fortes - CHEFE DE DEPARTAMENTO - CD4 - PF-DEPEX, PF-DEPEX, em 31/07/2019 11:49:10.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

A DIRPEI para esclarecimentos

Assinatura:

Despacho assinado eletronicamente por:

- Rodrigo Nascimento da Silva, Rodrigo Nascimento da Silva - PROFESSOR ENS BASICO TECN TECNOLOGICO, IF-PROEN, em 01/08/2019 15:16:51.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Retorno processo contendo Solicitação de Prorrogação do Projeto de Ensino PJE2018PFU0066, pois a solicitação de prorrogação apresenta tempo superior ao máximo permitido e essa situação não é contemplada no Regulamento de Projetos do IFSul: Art. 10. (...) § 1º Entende-se por prorrogação a concessão de novo prazo, não superior a 50% do prazo originalmente previsto, especificamente para a finalização das atividades propostas no cronograma original. O Cronograma de execução original prevê 12 meses, dessa forma a prorrogação não pode ultrapassar 6 meses. Ainda, Informo que o processo não apresenta os despachos da coordenação do Curso e da Direção Geral do Campus, conforme solicita o Formulário. As solicitações de renovação também devem apresentar Relatório Final parcial das atividades.

Assinatura:

Despacho assinado eletronicamente por:

- Magno Souza Grillo, Magno Souza Grillo - ASSISTENTE EM ADMINISTRACAO, IF-DIRPEI, em 15/08/2019 12:22:00.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Para as devidas providências

Assinatura:









Despacho assinado eletronicamente por:

- Maria Carolina Fortes, Maria Carolina Fortes - CHEFE DE DEPARTAMENTO - CD4 - PF-DEPEX, PF-DEPEX, em 15/08/2019 18:07:46.

ATA REUNIÃO Colegiado do TSPI, NÚMERO 2 de 2019

Aos sete dias do mês de agosto de dois mil e dezenove, com início às dezesseis horas, na sala 513 do Instituto Federal Sul-Rio-Grandense, campus Passo Fundo, ocorreu a reunião do grupo Colegiado do TSPI.

A reunião foi presidida por Rafael Marisco Bertei, sendo a redação da ata realizada por Carmen Vera Scorsatto, tendo se feito presentes as seguintes pessoas:

1. Adilso Nunes de Souza
 2. Alexandre Tagliari Lazzaretti
 3. André Fernando Rollwagen 
 4. Anubis Graciela de Moraes Rossetto
 5. Carlisa Toebe
 6. Carlos Petry
 7. Carmen Vera Scorsatto
 8. Edimara Luciana Sartori 
 9. Elder Francisco Fontana Bernardi
 10. Jaqueline Pinzon 
 11. Jorge Luis Boeira Bavaresco
 12. Josué Toebe
 13. José Antônio Oliveira de Figueiredo 
 14. Lisandro Lemos Machado
 15. Maikon Cismoski dos Santos
 16. Maria Carolina Fortes 
 17. Rafael Marisco Bertei 
 18. Ricardo Vanni Dallasen
 19. Roberto Wiest
 20. Vanessa Lago Machado
 21. JOAO MAMIO LOBO BNEZIN 
DANIEL DEZFINI 
- Justificativa de Ausência:

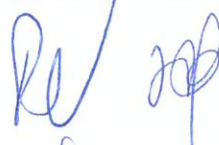
Alex Sebben da Cunha: Em atividades de trabalho.

Tendo a pauta definida conforme segue:

1. Projeto de Ensino prof. Jorge.
O projeto do Professor Jorge falou do seu projeto de produção de material didático e solicitou a aprovação na prorrogação do projeto de ensino.
2. Projeto de licença capacitação prof. Maikon
Todos aprovaram a licença de capacitação do professor de 18/09 ate 16/12
3. Cancelamento Compulsório Marcos Vinicius Mandeli do Amaral
De acordo com as regras da organização didática o aluno perdeu a vaga compulsoriamente.
4. Avaliação reabertura de curso aluno José Alcebides Almeida
O aluno já se matriculou na disciplina de projeto de Conclusão cinco vezes e reprovou por falta. Isto se enquadra na regras da organização didática.
5. Malotes somente quartas-feiras.
Agora será reduzido e envio de malote para Pelotas somente nas quartas feiras.
6. Processo seletivo -> vestibular e SISU
Existe uma discussão em todos os campus sobre a possibilidade de utilizar uma forma híbrida de ingresso aos cursos, Vestibular e Sisu.
7. RAD 2019/02
A Rad 2019/02 deve ser entregue até o final do mês de agosto.
Deverá ser utilizada a planilha antiga utilizada no semestre passado.
8. Situação orçamentária - encaminhamento Codir
A situação está bastante preocupante. Onde foi solicitado uma relação por parte da reitoria com todos os gastos até o final de 2019. Para deixar ciente o ministério da educação sobre essa situação que se encontra o IFSUL. Para que o MEC possa reavaliar a questão dos cortes realizados na área de educação.
9. Reunião de Avaliação e Diagnóstico de alunos com necessidades.
O Professor Josué Toebe, comenta sobre a avaliação e diagnósticos dos alunos, onde será realizado uma reunião com a finalidade de cada professor fazer um relato de seus alunos que necessitam ser diagnosticados para encaminhamentos futuros dos mesmos.
10. Plano de vagas 2020/2024.
Esta sendo feito um estudo de oferta de vagas para o período de 2020/2024
TSMI : 20
CC : 30
11. Cancelamento Compulsório Milena Pelissari
De acordo com as regras da organização didática o aluno perdeu a vaga

JP



lanetta p... s.

compulsoriamente.

12. Escolha representante comissão RAD
O representante da área da Informática: Professor André Rollwagen.

13. Cronogramas do PC2
Entrega de resumos : 17/09/2019
Seminários de andamento de 24/09/2019 ate 01/10/2019
Entrega final : 19/11/2019
Bancas de apresentação : 26/11/2019 a 03/12/2019
Entrega da versão final 10/12/2019

14. Cronograma do PCI
Entrega da proposta : 23/08/2019
Entrega de resumos : 13/09/2019
Seminários de andamento de 13/09/2019 ate 04/10/2019
Entrega final : 18/11/2019
Bancas de apresentação : 25/11/2019 a 06/12/2019
Entrega da versão final 13/12/2019

Sendo estes os assuntos tratados, o redator lavra a ata que, após lida e aprovada, será então assinada. Nada mais tendo a tratar segue assinado pela presidência da reunião e pelos demais presentes.

Prof. André Rollwagen
Milton C. de Aguiar
Luiz Carlos
Edimara L. Sartori
Luiz Carlos
Edimara L. Sartori
Luiz Carlos
Edimara L. Sartori

Documento Digitalizado Público

Ata da reunião com a solicitação da prorrogação do projeto

Assunto: Ata da reunião com a solicitação da prorrogação do projeto
Assinado por: Jorge Bavaresco
Tipo do Documento: Documento
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Mídia

Documento assinado eletronicamente por:

■ **Jorge Luis Boeira Bavaresco**, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO, em 20/08/2019 08:59:29.

Este documento foi armazenado no SUAP em 20/08/2019. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/verificar-documento-externo/> e forneça os dados abaixo:

Código Verificador: 48418

Código de Autenticação: 850cb9e608





RELATÓRIO FINAL DE PROJETO DE ENSINO

REGISTRO SOB N° : PJE2018 PFU 0066

Informar o número de registro do projeto de ensino ou processo eletrônico.

I.

IDENTIFICAÇÃO

a. Título do Projeto:

Desenvolvimento de sistemas com a plataforma Java EE

b. Resumo do Projeto:

O projeto trata da elaboração de um livro sobre o Desenvolvimento de sistemas com a plataforma Java EE, utilizando um estudo de caso para aplicação dos conceitos. O livro servirá de material didático para disciplinas do Curso de Ciência da Computação do campus Passo fundo, como Programação para Web e Programação para Web 2.

c. Classificação, Carga Horária, Equipe e Custo Global do Projeto:

Classificação e Carga Horária Total:			
<input type="checkbox"/> Curso/Mini-curso	<input type="checkbox"/> Palestra	<input type="checkbox"/> Evento	<input type="checkbox"/> Encontro <input type="checkbox"/> Fórum <input type="checkbox"/> Jornada
<input type="checkbox"/> Semana Acadêmica	<input type="checkbox"/> Olimpíada	<input type="checkbox"/> Clube	<input checked="" type="checkbox"/> outro - (especificar)
<input type="checkbox"/> Atividade Esportiva	<input type="checkbox"/> Monitoria	<input type="checkbox"/> Oficina	Produção de material didático.
<input checked="" type="checkbox"/> Ciências Exatas e da Terra	<input type="checkbox"/> Ciências Biológicas	<input type="checkbox"/> Engenharias	
<input type="checkbox"/> Ciências da Saúde	<input type="checkbox"/> Ciências Agrárias	<input type="checkbox"/> Ciências Sociais Aplicadas	
<input type="checkbox"/> Ciências Humanas	<input type="checkbox"/> Lingüística, Letras e Artes	<input type="checkbox"/> Outros	
Carga horária total do projeto: 240 horas			

Coordenador

Nome: Jorge Luis Boeira Bavaresco
Lotação: PF-DEPEX
SIAPE: 2969348

Demais membros		
Nome	Função	CH cumprida

Observação: a carga horária cumprida é a efetivamente realizada pelo membro ao final do Projeto não podendo ultrapassar a carga horária total do Projeto e a função pode ser Coordenador, Colaborador, Participante, Ministrante ou Palestrante.

Listar apenas os membros que serão certificados.

Custo Global do Projeto
O projeto não apresentou custos. Para a criação do material foram utilizados softwares gratuitos.

II. INTRODUÇÃO

O curso de Bacharelado em Ciência da Computação do campus Passo Fundo entrou em vigência no primeiro semestre letivo de 2017. Entre as disciplinas obrigatórias está a de Programação para a Web, no 5º semestre, e Programação para a Web II, sendo esta eletiva.

Os conceitos abordados em ambas as disciplinas descritas anteriormente podem ser implementados utilizando-se recursos e tecnologias presentes na plataforma Java EE. A literatura disponível em língua portuguesa sobre este tema está relativamente desatualizada, como pode-se perceber sobre alguns dos principais livros:

- GONCALVES, Antônio. **Introdução à plataforma Java (TM) EE 6 com o glassFish (TM) 3. 2º** Edição. Rio de Janeiro: Ciência Moderna, 2011.
- GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces. 3ª** Edição. Rio de Janeiro, RJ: Alta Books, 2012.
- BAUER, Christian; KING, Gavin. **Java persistence com hibernate. Rio de Janeiro, RJ: Ciência Moderna, 2007.**

Desde o lançamento destes livros até a data atual, diversas melhorias e inovações foram implantadas nas tecnologias relacionadas, deixando os alunos sem uma fonte de consulta atualizada em língua portuguesa. O presente projeto pretende elaborar um livro para suprir estas demandas e aplicar um método de ensino abordando estudos de caso, o que visa aproximar o aluno das demandas que ele terá no mundo acadêmico e

profissional.

III.
IV.

RESULTADOS OBTIDOS

Até o momento, os seguintes resultados foram obtidos:

- Estudo das tecnologias
- Criação de um estudo de caso para o desenvolvimento do livro
- Escrita dos seguintes capítulos do livro:
 - Introdução;
 - Configuração do ambiente de desenvolvimento;
 - Persistência de dados com JPA;
 - Enterprise Java Beans;

Para a conclusão do livro restam serem escritos os seguintes capítulos:

- Introdução a Java Server Faces;
- Desenvolvimento do estudo de caso com Java Server Faces e JPA;
- Relatórios com Jasper Reports;

IV.

FORMAS DE DISSEMINAÇÃO DOS RESULTADOS

O material será distribuído aos alunos dos Cursos de Ciência da Computação, durante as disciplinas de Programação para a Web e Programação para a Web II.

V.

CRONOGRAMA FINAL DE EXECUÇÃO

Atividades	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6	Mês 7	Mês 8	Mês 9	Mês 10	Mês 11	Mês 12
1	X	X	X									
2				X	X							
3					X	X						

Descrição das atividades:

Atividade 1: Estudo para atualização das tecnologias vigentes. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 2: Escrita dos capítulos restantes do livro. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 3: Procedimentos para publicação e avaliação do projeto.

VI.

REFERÊNCIAS BIBLIOGRÁFICAS

GONCALVES, Antônio. *Introdução à plataforma Java (TM) EE 6 com o glassFish (TM) 3*. 2. ed. Rio de Janeiro: Ciência Moderna, 2011.

GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3 ed. Rio de Janeiro, RJ: Alta Books, 2012.

BAUER, Christian; KING, Gavin. *Java persistence com hibernate*. Rio de Janeiro, RJ: Ciência Moderna, 2007.

ANEXOS (Listar os anexos)
1 -
2 -
3 -
4 -

PARECERES NECESSÁRIOS NO PROCESSO DO SUAP

- PARECER COLEGIADO/COORDENAÇÃO/ÁREA.
- PARECER DIREÇÃO/DEPARTAMENTO DE ENSINO.
- PARECER DIREÇÃO/DEPARTAMENTO DE ADMINISTRAÇÃO E PLANEJAMENTO (Quando necessário).
- PARECER DIREÇÃO-GERAL DO CAMPUS.
- PARECER DA PRÓ-REITORIA DE ENSINO.

20 de agosto de 2019

Documento assinado eletronicamente por:

- Jorge Luis Boeira Bavaresco, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO, em 20/08/2019 09:03:20.

Este documento foi emitido pelo SUAP em 20/08/2019. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 32428

Código de Autenticação: f074b6ee19





MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE
PRÓ-REITORIA DE ENSINO

FORMULÁRIO PARA PRORROGAÇÃO DE PRAZO EM PROJETO DE ENSINO

REGISTRO SOB N° : PJE2018 PFU 0066

Informar número de registro do projeto de ensino ou processo eletrônico.

I.

IDENTIFICAÇÃO

a. Título do Projeto:

Desenvolvimento de sistemas com a plataforma Java EE

b. Coordenador do Projeto:

Jorge Luis Boeira Bavaresco

II.

SOLICITAÇÃO DE PRORROGAÇÃO DE PRAZO

Período de prorrogação:

Prorrogação de oito (06) meses, até outubro 2019.

Justificativa:

Mudanças foram necessárias devido a atualizações das tecnologias utilizadas. Desta forma parte do trabalho deve ser atualizado ou refeito. Além deste fato, o prazo para conclusão teve atrasos devidos a licença capacitação do coordenador do projeto. Devido a este fatos é necessária a prorrogação do projeto de ensino para que seja possível a sua conclusão.

III.

NOVO CRONOGRAMA DE EXECUÇÃO

Atividades	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6	Mês 7	Mês 8	Mês 9	Mês 10	Mês 11	Mês 12
1	X	X	X									
2				X	X							
3					X	X						

Descrição das atividades:

Atividade 1: Estudo para atualização das tecnologias vigentes. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 2: Escrita dos capítulos restantes do livro. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 3: Procedimentos para publicação e avaliação do projeto.

PARECERES NECESSÁRIOS NO PROCESSO DO SUAP

- PARECER COLEGIADO/COORDENAÇÃO/ÁREA.
- PARECER DIREÇÃO/DEPARTAMENTO DE ENSINO.
- PARECER DIREÇÃO/DEPARTAMENTO DE ADMINISTRAÇÃO E PLANEJAMENTO (Quando necessário).
- PARECER DIREÇÃO-GERAL DO CAMPUS.
- PARECER DA PRÓ-REITORIA DE ENSINO.

OBS:

Não esquecer de enviar o relatório com as atividades já realizadas. Utilizar modelo de Relatório Final.

Não esquecer de enviar a ata de aprovação da prorrogação pelo Colegiado/Coordenação do Curso.

20 de agosto de 2019

Documento assinado eletronicamente por:

- **Jorge Luis Boeira Bavaresco, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 20/08/2019 09:02:37.

Este documento foi emitido pelo SUAP em 20/08/2019. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 32425

Código de Autenticação: df1b67bfbc





SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

De acordo.

Assinatura:

Despacho assinado eletronicamente por:

- Rafael Marisco Bertei, Rafael Marisco Bertei - COORDENADOR - FUC1 - PF-COCSIST, PF-COCSIST, em 07/10/2019 15:28:31.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

De acordo com a prorrogação do projeto de ensino.

Assinatura:

Despacho assinado eletronicamente por:

- Maria Carolina Fortes, Maria Carolina Fortes - CHEFE DE DEPARTAMENTO - CD4 - PF-DEPEX, PF-DEPEX, em 07/10/2019 16:48:20.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Avaliado. Falta parecer do Diretor Geral do Campus, o que é obrigatório segundo Regulamento de Projetos de Ensino.

Assinatura:

Despacho assinado eletronicamente por:

- Magno Souza Grillo, Magno Souza Grillo - ASSISTENTE EM ADMINISTRACAO, IF-PROEN, em 10/10/2019 11:18:01.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Ao Diretor geral, para parecer e posterior encaminhamento a PROEN

Assinatura:

Despacho assinado eletronicamente por:

- Maria Carolina Fortes, Maria Carolina Fortes - CHEFE DE DEPARTAMENTO - CD4 - PF-DEPEX, PF-DEPEX, em 10/10/2019 21:35:55.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Considerando os pareceres das chefias imediatas, esta administração não verifica óbice a esta prorrogação por 6 meses.

Assinatura:

Despacho assinado eletronicamente por:

- Alexandre Pitol Boeira, Alexandre Pitol Boeira - DIRETOR GERAL - CD2 - PF-DIRGER, PF-DIRGER, em 10/10/2019 22:19:20.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Encaminhado à Diretoria de Políticas de Ensino e Inclusão para Avaliação e Parecer.

Assinatura:

Despacho assinado eletronicamente por:

- Magno Souza Grillo, Magno Souza Grillo - ASSISTENTE EM ADMINISTRACAO, IF-PROEN, em 11/10/2019 14:26:52.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Analisando a justificativa apresentada pelo Coordenador: "Mudanças foram necessárias devido a atualizações das tecnologias utilizadas. Desta forma parte do trabalho deve ser atualizado ou refeito. Além deste fato, o prazo para conclusão teve atrasos devido a licença capacitação do coordenador do projeto. Devido a este fatos é necessária a prorrogação do projeto de ensino para que seja possível a sua conclusão." Avalio que o referido Projeto de Ensino "Desenvolvimento de sistemas com a plataforma Java EE" é de grande importância para os estudantes as justificativas apresentadas são relevantes, sendo assim, defiro o pedido de prorrogação do mesmo, apesar de ultrapassar o tempo estipulado no Regulamento de Projetos de Ensino, que é de até 50% do período previsto inicialmente.

Assinatura:

Despacho assinado eletronicamente por:

- Veridiana Krolow Bosenbecker, Veridiana Krolow Bosenbecker - DIRETOR - CD3 - IF-DIRPEI, IF-DIRPEI, em 20/10/2019 20:40:01.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Solicitação de Prorrogação Aprovada Pela Pró-Reitoria de Ensino. Retornamos o processo para inserção (Upload) do Relatório Final e do Formulário para Solicitação de Certificação, quando do encerramento do Projeto de Ensino.

Assinatura:

Despacho assinado eletronicamente por:

- Ana Carolina Madeira Hise, Ana Carolina Madeira Hise - ESTAGIARIO - ETG1 - IF-REIT, IF-DIRPEI, em 22/10/2019 16:48:45.



RELATÓRIO FINAL DE PROJETO DE ENSINO

REGISTRO SOB N° : PJE2018 PFU 0066

Informar o número de registro do projeto de ensino ou processo eletrônico.

I.

IDENTIFICAÇÃO

a. Título do Projeto:

Desenvolvimento de sistemas com a plataforma Java EE

b. Resumo do Projeto:

O projeto trata da elaboração de um livro sobre o Desenvolvimento de sistemas com a plataforma Java EE, utilizando um estudo de caso para aplicação dos conceitos. O livro servirá de material didático para disciplinas do Curso de Ciência da Computação do campus Passo fundo, como Programação para Web e Programação para Web 2.

c. Classificação, Carga Horária, Equipe e Custo Global do Projeto:

Classificação e Carga Horária Total:			
<input type="checkbox"/> Curso/Mini-curso	<input type="checkbox"/> Palestra	<input type="checkbox"/> Evento	<input type="checkbox"/> Encontro <input type="checkbox"/> Fórum <input type="checkbox"/> Jornada
<input type="checkbox"/> Semana Acadêmica	<input type="checkbox"/> Olimpíada	<input type="checkbox"/> Clube	<input checked="" type="checkbox"/> outro - (especificar)
<input type="checkbox"/> Atividade Esportiva	<input type="checkbox"/> Monitoria	<input type="checkbox"/> Oficina	Produção de material didático.
<input checked="" type="checkbox"/> Ciências Exatas e da Terra	<input type="checkbox"/> Ciências Biológicas	<input type="checkbox"/> Engenharias	
<input type="checkbox"/> Ciências da Saúde	<input type="checkbox"/> Ciências Agrárias	<input type="checkbox"/> Ciências Sociais Aplicadas	
<input type="checkbox"/> Ciências Humanas	<input type="checkbox"/> Lingüística, Letras e Artes	<input type="checkbox"/> Outros	
Carga horária total do projeto: 240 horas			

Coordenador

Nome: Jorge Luis Boeira Bavaresco
Lotação: PF-DEPEX
SIAPE: 2969348

Demais membros		
Nome	Função	CH cumprida

Observação: a carga horária cumprida é a efetivamente realizada pelo membro ao final do Projeto não podendo ultrapassar a carga horária total do Projeto e a função pode ser Coordenador, Colaborador, Participante, Ministrante ou Palestrante.

Listar apenas os membros que serão certificados.

Custo Global do Projeto
O projeto não apresentou custos. Para a criação do material foram utilizados softwares gratuitos.

II. INTRODUÇÃO

O curso de Bacharelado em Ciência da Computação do campus Passo Fundo entrou em vigência no primeiro semestre letivo de 2017. Entre as disciplinas obrigatórias está a de Programação para a Web, no 5º semestre, e Programação para a Web II, sendo esta eletiva.

Os conceitos abordados em ambas as disciplinas descritas anteriormente podem ser implementados utilizando-se recursos e tecnologias presentes na plataforma Java EE. A literatura disponível em língua portuguesa sobre este tema está relativamente desatualizada, como pode-se perceber sobre alguns dos principais livros:

- GONCALVES, Antônio. **Introdução à plataforma Java (TM) EE 6 com o glassFish (TM) 3. 2º** Edição. Rio de Janeiro: Ciência Moderna, 2011.
- GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces. 3ª** Edição. Rio de Janeiro, RJ: Alta Books, 2012.
- BAUER, Christian; KING, Gavin. **Java persistence com hibernate. Rio de Janeiro, RJ: Ciência Moderna, 2007.**

Desde o lançamento destes livros até a data atual, diversas melhorias e inovações foram implantadas nas tecnologias relacionadas, deixando os alunos sem uma fonte de consulta atualizada em língua portuguesa. O presente projeto pretende elaborar um livro para suprir estas demandas e aplicar um método de ensino abordando estudos de caso, o que visa aproximar o aluno das demandas que ele terá no mundo acadêmico e

profissional.

III.
IV.

RESULTADOS OBTIDOS

Com o desenvolvimento do projeto, foi escrito um livro sobre o desenvolvimento de sistemas com a plataforma Java EE, com um total de 97 páginas. Para a publicação o livro não se encontra totalmente pronto, necessitando de complementação em alguns capítulos. Pretende-se ao longo do tempo realizar-se constantes atualizações do seu conteúdo.

Atualmente o livro conta com os seguintes capítulos:

- Configuração do ambiente de desenvolvimento;
- Persistência com JPA;
- Layout responsivo com Java Server Faces;
- Composite: Criando seus próprios componentes;
- Internacionalização de mensagens;
- Enterprise Java Beans;

Futuramente, serão adicionados ao livro alguns capítulos:

- Desenvolvimento do estudo de caso com Java Server Faces e JPA;
- Relatórios com Jasper Reports;

O propósito do projeto de ensino foi atingido, que era fornecer aos alunos um material atualizado dando subsídio teórico e prático para as aulas. Como aprendizado ao se desenvolver o livro, ficou a experiência em relação ao tempo de execução e produção do projeto, o que irá ajudar a produzir estimativas e planejar de forma adequada futuros projetos. Ressalto contudo a produção de material personalizado para os alunos, pois fica mais adequado para uso no curso e com as necessidades locais, em relação a obras já disponíveis, algumas desatualizadas para as tecnologias atuais.

IV.

FORMAS DE DISSEMINAÇÃO DOS RESULTADOS

O material será distribuído aos alunos dos Cursos de Ciência da Computação, durante as disciplinas de Programação para a Web e Programação para a Web II.

V.

CRONOGRAMA FINAL DE EXECUÇÃO

Atividades	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6	Mês 7	Mês 8	Mês 9	Mês 10	Mês 11	Mês 12
1	X	X	X									
2				X	X							

3					X	X							
---	--	--	--	--	---	---	--	--	--	--	--	--	--

Descrição das atividades:

Atividade 1: Estudo para atualização das tecnologias vigentes. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 2: Escrita dos capítulos restantes do livro. Responsável: Jorge Luis Boeira Bavaresco.

Atividade 3: Procedimentos para publicação e avaliação do projeto.

VI. REFERÊNCIAS BIBLIOGRÁFICAS

GONCALVES, Antônio. *Introdução à plataforma Java (TM) EE 6 com o glassFish (TM) 3*. 2. ed. Rio de Janeiro: Ciência Moderna, 2011.

GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3 ed. Rio de Janeiro, RJ: Alta Books, 2012.

BAUER, Christian; KING, Gavin. *Java persistence com hibernate*. Rio de Janeiro, RJ: Ciência Moderna, 2007.

ANEXOS (Listar os anexos)
1 -
2 -
3 -
4 -

PARECERES NECESSÁRIOS NO PROCESSO DO SUAP

- PARECER COLEGIADO/COORDENAÇÃO/ÁREA.
- PARECER DIREÇÃO/DEPARTAMENTO DE ENSINO.
- PARECER DIREÇÃO/DEPARTAMENTO DE ADMINISTRAÇÃO E PLANEJAMENTO (Quando necessário).
- PARECER DIREÇÃO-GERAL DO CAMPUS.
- PARECER DA PRÓ-REITORIA DE ENSINO.

Documento assinado eletronicamente por:

- **Jorge Luis Boeira Bavaresco, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 29/10/2019 15:11:36.

Este documento foi emitido pelo SUAP em 29/10/2019. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 40283

Código de Autenticação: 9c6e181229





MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE
PRÓ-REITORIA DE ENSINO

FORMULÁRIO PARA SOLICITAÇÃO DE CERTIFICAÇÃO

REGISTRO SOB N° : PJE2018 PFU 0066

Informar o número de registro do projeto de ensino ou processo eletrônico.

INFORMAÇÕES PARA CERTIFICAÇÃO

a. Título do Projeto:

Desenvolvimento de sistemas com a plataforma Java EE

b. Classificação, Carga Horária e Equipe

Classificação e Carga Horária Total:			
<input type="checkbox"/> Curso/Mini-curso	<input type="checkbox"/> Palestra	<input type="checkbox"/> Evento	<input type="checkbox"/> Encontro <input type="checkbox"/> Fórum <input type="checkbox"/> Jornada
<input type="checkbox"/> Semana Acadêmica	<input type="checkbox"/> Olimpíada	<input type="checkbox"/> Clube	<input checked="" type="checkbox"/> outro - (especificar)
<input type="checkbox"/> Atividade Esportiva	<input type="checkbox"/> Monitoria	<input type="checkbox"/> Oficina	Produção de material didático.
<input checked="" type="checkbox"/> Ciências Exatas e da Terra	<input type="checkbox"/> Ciências Biológicas	<input type="checkbox"/> Engenharias	
<input type="checkbox"/> Ciências da Saúde	<input type="checkbox"/> Ciências Agrárias	<input type="checkbox"/> Ciências Sociais Aplicadas	
<input type="checkbox"/> Ciências Humanas	<input type="checkbox"/> Lingüística, Letras e Artes	<input type="checkbox"/> Outros	
Carga horária total do projeto: 240 horas			
Mês/ano de início: Maio 2018			
Mês/ano de término: Outubro 2019			

Coordenador

Nome: Jorge Luis Boeira Bavaresco

Lotação: PF-COMSI

SIAPE: 2969348

Membros			
Nome	CPF	Função	CH cumprida
Jorge Luis Boeira Bavaresco	98329324087	Coordenador	240 horas

Observação: a carga horária cumprida é a efetivamente realizada pelo membro ao final do Projeto não podendo ultrapassar a carga horária total do Projeto e a função pode ser Coordenador, Colaborador, Participante, Ministrante ou Palestrante.

Listar apenas os membros que serão certificados.

Relatório Final:

Aprovado pela Pró-reitoria de Ensino em:

(Não esquecer de anexar o Relatório Final aprovado em todas as instâncias).

Coordenador do Projeto de Ensino deve assinar eletronicamente o documento e enviar para a PROEN

29 de outubro de 2019

Documento assinado eletronicamente por:

- **Jorge Luis Boeira Bavaresco, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 29/10/2019 15:19:34.

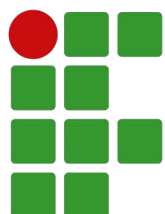
Este documento foi emitido pelo SUAP em 29/10/2019. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 40291

Código de Autenticação: e429949a59



Jorge Luis Boeira Bavaresco

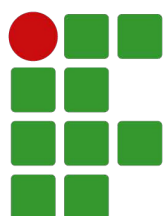


INSTITUTO FEDERAL
Sul-rio-grandense

Câmpus
Passo Fundo

Desenvolvimento de sistemas com Java EE

Jorge Luis Boeira Bavaresco



INSTITUTO FEDERAL
Sul-rio-grandense

Câmpus
Passo Fundo

Desenvolvimento de sistemas com Java EE

© 2019 Jorge Luis Boeira Bavaresco & Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
Qualquer parte desta publicação pode ser reproduzida, desde que citada a fonte.

BAVARESCO, J. L. B.

Desenvolvimento de sistemas com Java EE. / Jorge Luis Boeira Bavaresco. – Passo Fundo: Instituto Federal Sul-Rio-Grandense Campus Passo Fundo, 2019.

Bibliografia.

ISBN XXXX-XXXX-XX.

1. Java EE. 2. Java Server Faces. 3. Java.

Agradecimentos

À minha esposa Neuzer, pelo amor, inspiração e incentivo. Amo você.

Às minhas filhas Giovanna e Camila, pelo amor, alegria e inspiração.

Aos meus pais, João e Anita, pelo amor, carinho e valores que me deram ao longo da minha vida.

Lista de ilustrações

Figura 1 – Download do Java JDK 8	11
Figura 2 – Tela de boas vindas da instalação do Java	11
Figura 3 – Tela de escolha do diretório da JDK	12
Figura 4 – Tela de escolha do diretório da JRE	12
Figura 5 – Download do Netbeans 8.2	13
Figura 6 – Tela de boas vindas da instalação do Netbeans 8.2	13
Figura 7 – Tela de personalização do instalador do Netbeans 8.2	14
Figura 8 – Download do Payara Server.	14
Figura 9 – Tela de instalação do plugin do Payara.	15
Figura 10 – Adicionando servidor no netbeans	15
Figura 11 – Adicionar instância do servidor	15
Figura 12 – Localização do servidor	16
Figura 13 – Localização do domínio	16
Figura 14 – Iniciando o servidor Payara	17
Figura 15 – Tela do Payara no navegador.	17
Figura 16 – Download do PostgreSQL.	18
Figura 17 – Tela de boas vindas da instalação do PostgreSQL	18
Figura 18 – Escolha do diretório de instalação do PostgreSQL	19
Figura 19 – Escolha de componentes na instalação do PostgreSQL	19
Figura 20 – Escolha do diretório de dados do PostgreSQL	20
Figura 21 – Senha do usuário postgres	20
Figura 22 – Escolha da porta de funcionamento do SGBD	21
Figura 23 – Escolha da localidade do SGBD	21
Figura 24 – Tela do pgAdmin4	22
Figura 25 – Tela de download do JasperStudio	23
Figura 26 – Tela de seleção do diretório do JasperStudio	23
Figura 27 – Tela do gerenciador de bibliotecas	24
Figura 28 – Tela do gerenciador de bibliotecas - Adicionando arquivos da biblioteca	25
Figura 29 – Tela da biblioteca após o seu registro	25
Figura 30 – Diagrama de classes do estudo de caso	29
Figura 31 – Diagrama de classes do estudo de caso	30
Figura 32 – Classes Produto e Arquivo	35
Figura 33 – Tela de seleção de tipo de projeto (criação do projeto da camada de modelo)	40
Figura 34 – Tela de criação do projeto da camada de modelo	40
Figura 35 – Estrutura inicial do projeto da camada de modelo	41
Figura 36 – Adicionando bibliotecas ao projeto - primeira tela	41
Figura 37 – Adicionando bibliotecas ao projeto - segunda tela	42
Figura 38 – Adicionando bibliotecas ao projeto - terceira tela	42
Figura 39 – Adicionando bibliotecas ao projeto - quarta tela	42
Figura 40 – Adicionando JAR ao projeto	43
Figura 41 – Bibliotecas do projeto da camada de modelo.	43

Figura 42 – pgAdmin 4 - Tela para criar base de dados	44
Figura 43 – pgAdmin 4 - Tela com informações para criar o banco de dados	45
Figura 44 – pgAdmin 4 - Tela a listagem dos bancos de dados	45
Figura 45 – Registrando a conexão no Netbeans	46
Figura 46 – Registrando a conexão no Netbeans - Primeira tela do assistente	46
Figura 47 – Registrando a conexão no Netbeans - Segunda tela do assistente	47
Figura 48 – Conexão com o banco de dados registrada no Netbeans	47
Figura 49 – Criando a unidade de persistência - Tela 01	48
Figura 50 – Criando a unidade de persistência - Tela 02	48
Figura 51 – Biblioteca adiciona pela Netbeans a ser removida	49
Figura 52 – Classe Estado	50
Figura 53 – Tela de criação da Classe Estado	51
Figura 54 – Refatorar campos	52
Figura 55 – Selecionando campos para refatorar	52
Figura 56 – Selecionando campos para refatorar	53
Figura 57 – Exibição da listagem em telas grandes	67
Figura 58 – Exibição da listagem em telas pequenas	68
Figura 59 – Exibição do formulário em telas grandes	69
Figura 60 – Exibição do formulário em telas pequenas	70
Figura 61 – Pasta dos componentes dentro de resources	72
Figura 62 – Estrutura do projeto com os arquivos de internacionalização	77
Figura 63 – Código de bean de sessão <i>stateful</i>	83
Figura 64 – Injeção de dependências do EJB.	84

Lista de tabelas

Tabela 1 – Softwares utilizados no livro	10
Tabela 2 – Bibliotecas utilizadas no livro	24
Tabela 3 – Callbacks para um EJB Stateless	84
Tabela 4 – Callbacks para um EJB Stateless	85
Tabela 5 – Anotações do ciclo de vida de um Bean CDI do JSF	89
Tabela 6 – Atributos para a criação de expressões	91
Tabela 7 – Exemplos de expressões do serviço temporizador	92

Lista de códigos fonte

2.1	Treço da classe Estado	30
2.2	Anotação @Entity	31
2.3	Anotação @Table	31
2.4	Anotação @Id	31
2.5	Anotação @SequenceGenerator e @GeneratedValue	32
2.6	Anotação @GeneratedValue com strategy GenerationType.IDENTITY	32
2.7	Anotação @Column	32
2.8	Anotação @Temporal	33
2.9	Anotação para mapeamento de herança - trecho classe Usuário	33
2.10	Anotação para mapeamento de herança - trecho classe PessoaFisica	34
2.11	Anotação @ManyToOne e @JoinColumn	34
2.12	Anotação @OneToMany - mapeamento na classe Arquivo	35
2.13	Anotação @OneToMany - mapeamento na classe Produto	35
2.14	Anotação @ManyToMany	36
2.15	Código de criação da tabela contas receber	36
2.16	Treço da Classe ContaReceberID	36
2.17	Treço da Classe ContaReceber	37
2.18	Treço da Classe Estado com validações	38
2.19	Chamando a validação da Classe Estado	39
2.20	Arquivo persistence.xml criado pela IDE	49
2.21	Arquivo persistence.xml modificado	50
2.22	Classe Estado sem o padrão Java Beans	51
2.23	Classe Estado com o padrão Java Beans e método Equals sobrescrito	53
2.24	Classe Estado com as anotações da JPA e validações	54
2.25	Persistindo um objeto	56
2.26	Carregando um objeto	57
2.27	Atualizando um objeto	57
2.28	Removendo um objeto	58
2.29	Consulta com JPQL	59
3.1	Classe Pessoa	60
3.2	Classe ControlePessoa	62
3.3	Classe ConverterCalendar	63
3.4	Conteúdo do arquivo web.xml	64
3.5	Conteúdo do arquivo faces-config.xml	65
3.6	Conteúdo do arquivo template.xhtml	65
3.7	Conteúdo do arquivo index.xhtml	66
3.8	Conteúdo do arquivo listar.xhtml	66
3.9	Conteúdo do arquivo formulario.xhtml	68
4.1	campo.xhtml	72
4.2	Utilização do campo de formulário	73
4.3	painel.xhtml	73

4.4	Utilização do painel.xhtml	73
4.5	botaoSalvar.xhtml	74
4.6	Utilizando o botaoSalvar.xhtml	74
4.7	botaoRemover.xhtml	74
4.8	Utilizando o botaoRemover.xhtml	75
5.1	messages_pt_BR.properties	76
5.2	messages_en_US.properties	76
5.3	ValidationMessages_pt_BR.properties	77
5.4	ValidationMessages_en_US.properties	77
5.5	Idiomas no faces-config.xml	77
5.6	Mensagem de internacionalização na tela	78
5.7	Mensagem de internacionalização nas validações	78
5.8	ControleLocale.java	78
5.9	template.xhtml com a definição da localidade	79
5.10	Util.java	80
5.11	Exemplo de uso do método getMensagemInternacionalizada	81
6.1	Código fonte de um EJB Stateless	84
6.2	Código fonte de um EJB Stateful	86
6.3	Código fonte de um EJB Singleton	87
6.4	Código fonte de um EJB Singleton	88
6.5	Código fonte de um bean CDI do JSF	88
6.6	Código fonte de um bean CDI que será injetado	89
6.7	Código fonte onde bean CDI que foi injetado	90
6.8	EJB com métodos que usam temporizador	92

Sumário

Introdução	9
1 Configuração do ambiente de desenvolvimento	10
1.1 Softwares utilizados no livro	10
1.1.1 Instalação do Java JDK 8	10
1.1.2 Instalação do Netbeans 8.2	12
1.1.2.1 Configuração do Payara 5 na IDE Netbeans	14
1.1.3 Instalação do SGBD PostgreSQL	17
1.1.4 Instalação da IDE JasperStudio	22
1.2 Bibliotecas utilizadas no livro	24
1.2.1 Criação e registro das bibliotecas no Netbeans	24
2 Persistência com JPA	26
2.1 JPA - Java Persistence API	26
2.2 Framework de persistência Hibernate	27
2.2.1 Biblioteca Hibernate 5.3 JPA 2.2	27
2.3 Estudo de caso	27
2.4 Mapeamento objeto-relacional com anotações JPA	30
2.4.1 Anotações da JPA	30
2.4.1.1 Anotação @Entity	31
2.4.1.2 Anotação @Table	31
2.4.1.3 Anotação @Id	31
2.4.1.4 Anotação @SequenceGenerator e @GeneratedValue	31
2.4.1.5 Anotação @Column	32
2.4.1.6 Anotação @Temporal	33
2.4.1.7 Anotação @Inheritance	33
2.4.1.8 Anotações @ManyToOne e @JoinColumn	34
2.4.1.9 Anotação @OneToMany	34
2.4.1.10 Anotação @ManyToMany	35
2.4.1.11 Chave composta com anotação @Embeddable e @EmbeddedId	36
2.5 Validação de dados com Hibernate Validator	37
2.5.1 Anotações para validação	37
2.5.2 Anotando uma classe e invocando o processo de validação	38
2.6 Camada de modelo - Mapeamento Objeto-relacional do estudo de caso	39
2.6.1 Criando o projeto da camada de modelo	40
2.6.1.1 Adicionando as bibliotecas	41
2.6.1.2 Criando a Unidade de persistência (Persistence Unit)	44
2.6.2 Mapeamento de classe simples	50
2.6.3 Realizando operações de persistência	55
2.6.3.1 Persistindo um objeto	56
2.6.3.2 Carregando um objeto	57
2.6.3.3 Atualizando um objeto	57
2.6.3.4 Removendo um objeto	58
2.6.3.5 Realizando uma consulta com a JPQL	59
3 Layout responsivo com Java Server Faces	60
3.1 Bibliotecas necessárias	60
3.2 Desenvolvimento da aplicação	60
3.2.1 Camada de modelo	60
3.2.2 Camada de controle	62

3.2.3	Configurações dos arquivos web.xml e faces-config.xml	64
3.2.4	Template	65
3.2.5	Tela dos CRUDS	66
4	Composite : Criando seus próprios componentes	71
4.1	Documentação	71
4.2	Criando seus próprios componentes.	71
4.3	Empacotando seus próprios componentes em um arquivo JAR.	75
5	Internacionalização de mensagens (i18n)	76
5.1	Arquivos com as mensagens de internacionalização	76
5.2	Registro no arquivo faces-config.xml dos idiomas suportados	77
5.3	Associando os valores das mensagens com as chaves de internacionalização	78
5.4	Controle do JSF para os idiomas	78
5.5	Internacionalização de mensagens dentro de métodos ou classes	80
6	Enterprise Java Beans (EJB)	82
6.1	JAVA EE	82
6.1.1	EJB - Enterprise Java Beans	83
6.2	Tipos de EJB	84
6.2.1	Bean de sessão sem estado (@Stateless)	84
6.2.2	Bean de sessão com estado (@Stateful)	85
6.2.3	Bean de sessão Singular (@Singleton)	87
6.2.4	Usando EJBs	88
6.3	Controladores do JSF gerenciados pelo servidor de aplicações (CDI)	88
6.3.1	Injetando um bean CDI do JSF	89
6.4	Serviço temporizador	91
	Referências	94

Introdução

ESTE DOCUMENTO aborda o desenvolvimento de sistemas com a plataforma Java EE (*Java Enterprise Edition*) e suas tecnologias. Espera-se que ele seja um guia para desenvolvedores e estudantes, nortando o uso de boas práticas de desenvolvimento e produtividade, e apresente ao leitor as características e possibilidades da plataforma.

O livro inicia no [Capítulo 1](#), onde será tratado da configuração do ambiente de desenvolvimento, que são os softwares necessários para se reproduzir os exemplos apresentados no livro. No [Capítulo 2](#) é apresentada a tecnologia da JPA (Java Persistence API), e se inicia a construção do estudo de caso. No [Capítulo 3](#) será apresentada uma solução para o desenvolvimento de interfaces com layout responsivo usando Java Server Faces e algumas bibliotecas de componentes. Já no capítulo [Capítulo 4](#) criaremos nossos próprios componentes com o JSF. O [Capítulo 5](#) apresentará a internacionalização de mensagens. E o [Capítulo 6](#) realiza uma introdução à plataforma Java EE.

Configuração do ambiente de desenvolvimento

Este capítulo trata de conduzir o leitor na instalação e configuração dos softwares e bibliotecas utilizadas no desenvolvimento do aplicativo do estudo de caso do livro.

1.1 Softwares utilizados no livro

Para o desenvolvimento do livro, serão utilizados alguns softwares, entre eles a linguagem de programação Java, a IDE de desenvolvimento NetBeans, o sistema gerenciador de bancos de dados PostgreSQL, o servidor web Payara e a IDE de desenvolvimento de relatórios Jasper Studio. Na [Tabela 1](#) são listados os softwares utilizados com suas respectivas versões e endereço para download.

Tabela 1 – Softwares utilizados no livro

<i>Software</i>	Versão	Endereço para download
Java JDK 8	jdk-8u191	https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Netbeans IDE	8.2	https://netbeans.org/downloads/8.2/
SGBD PostgreSQL	11.1	https://www.openscg.com/bigsql/postgresql/installers.jsp
Servidor Web Payara	5	https://www.payara.fish/software/downloads/all-downloads/
IDE JasperStudio	6.6.0	https://community.jaspersoft.com/project/jaspersoft-studio/releases

As seções a seguir irão tratar da instalação de cada uma destas ferramentas.

1.1.1 Instalação do Java JDK 8

A versão do Java utilizada neste livro será a versão 8. Será abordada a instalação na plataforma windows. O arquivo para instalação pode ser obtido em <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>, conforme exibido na [Figura 1](#). A versão utilizada é a JDK 8 update 191 para windows de 64 bits. Realize o download conforme a sua versão do sistema operacional. O arquivo obtido (jdk-8u191-windows-x64.exe) contém o kit de desenvolvimento Java (JDK - *Java Development Kit*) e o *Java Runtime Environment*, que contém a máquina virtual para execução de aplicativos Java.

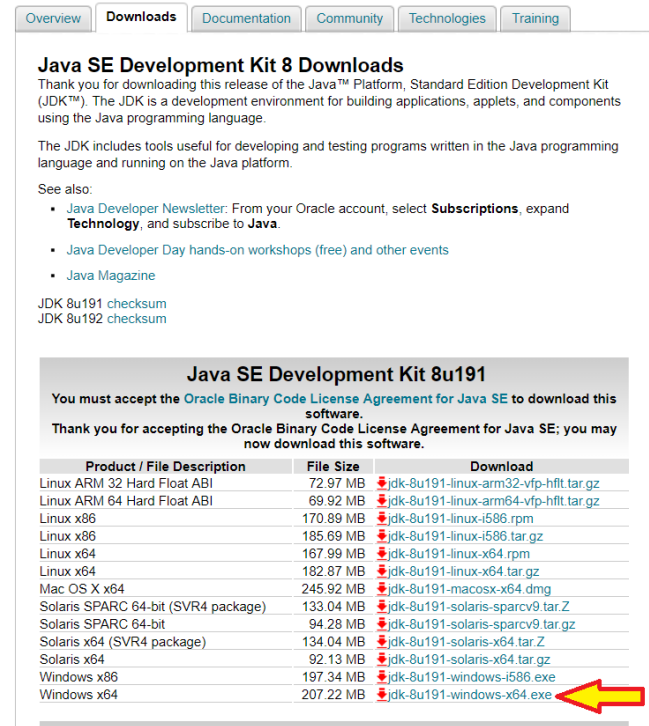


Figura 1 – Download do Java JDK 8

Após o download do arquivo execute-o. Uma tela de boas vindas será exibida (Figura 2), clique no botão next e será exibida a tela para escolher o diretório de instalação da JDK (Figura 3). Espere concluir esta etapa e o instalador irá solicitar o local de instalação do java JRE (Figura 4), basta clicar no botão próximo e aguardar o término da instalação.

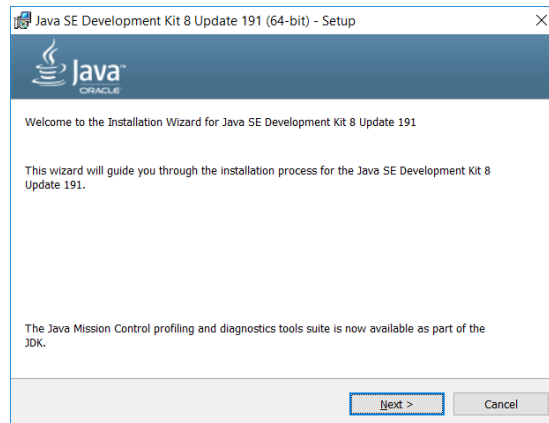


Figura 2 – Tela de boas vindas da instalação do Java

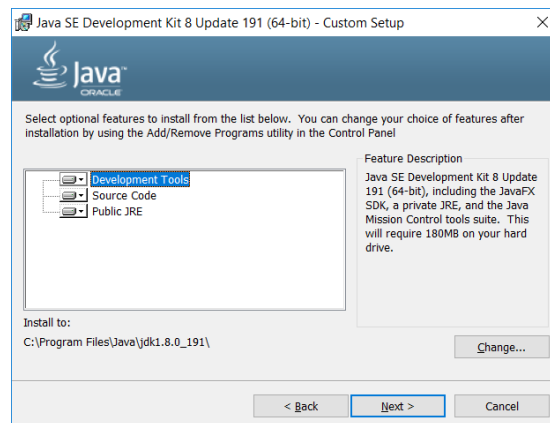


Figura 3 – Tela de escolha do diretório da JDK

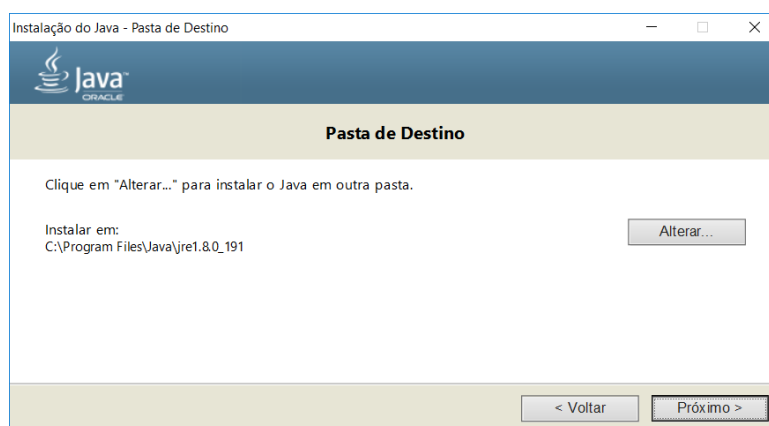


Figura 4 – Tela de escolha do diretório da JRE

1.1.2 Instalação do Netbeans 8.2

A IDE utilizada no desenvolvimento será a Netbeans IDE, versão 8.2, cujo download pode ser realizado em <https://netbeans.org/downloads/8.2/>. Obtenha a versão completa, como mostra a Figura 5.



Figura 5 – Download do Netbeans 8.2

Após o download execute o arquivo obtido (netbeans-8.2-windows.exe), e a tela de boas vindas se abrirá e nela selecione a opção personalizar (Figura 6).

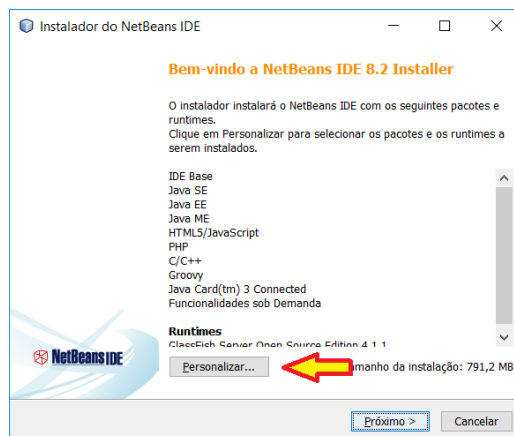


Figura 6 – Tela de boas vindas da instalação do Netbeans 8.2

Na tela que será exibida (Figura 7) desmarque as opções de Runtimes do GlassFish e Apache Tomcat, já que no livro o servidor utilizado vai ser o Payara Server, que será registrado posteriormente na IDE. Também é possível escolher outros componentes a serem instalados, e no caso deste livro podem ser desmarcadas as opções de PHP, C/C++, Groovy e Java Card. Após clique em ok e a tela anterior será exibida novamente, então clique em próximo.

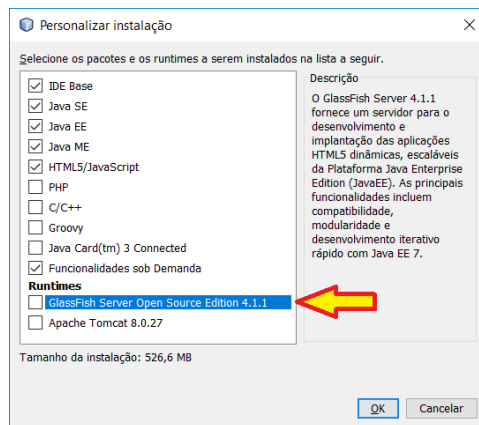


Figura 7 – Tela de personalização do instalador do Netbeans 8.2

Na tela seguinte aceite o contrato de licença e clique em próximo. Na tela que se abrirá é possível escolher o diretório de instalação, basta clicar no botão próximo novamente e finalizar a instalação.

1.1.2.1 Configuração do Payara 5 na IDE Netbeans

No livro será utilizado o servidor Payara 5, que implementa a especificação do Java EE 8, e pode ser obtido em <<https://www.payara.fish/software/downloads/all-downloads/>> (Figura 8). Ele é um servidor de código fonte aberto derivado do GlassFish. Realize o download e descompacte o arquivo em uma pasta do seu computador.

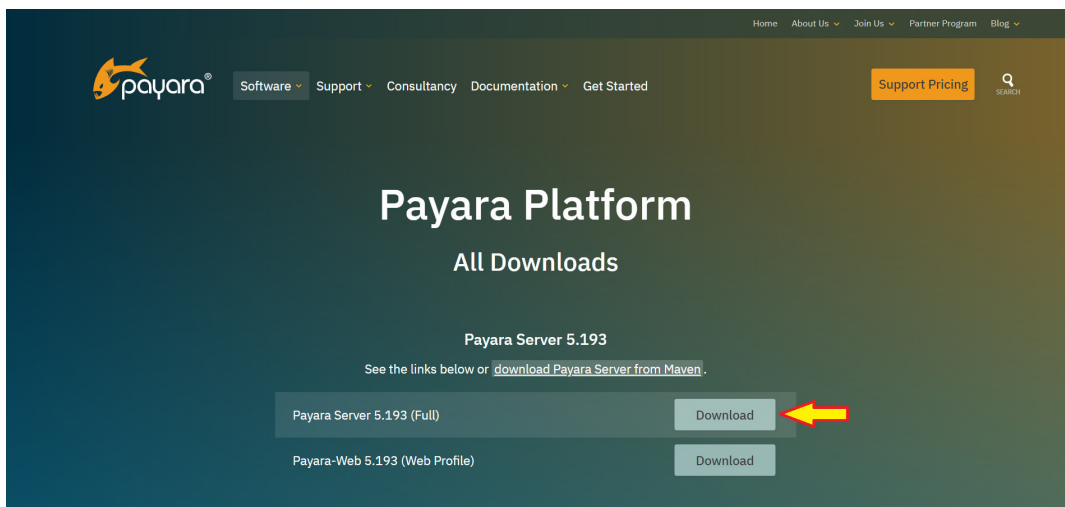


Figura 8 – Download do Payara Server.

Após, na IDE Netbeans, é necessário realizar a instalação de plugins para permitir o uso deste servidor. Para isso, acesse o menu ferramentas, e após a opção plugins. Selecione a guia Plug-ins e digite no campo de busca Payara. Selecione os cinco plugins disponíveis (Figura 9) e clique no botão de instalar. Na próxima tela clique no botão próximo, e na seguinte aceite os termos de licença e clique em instalar. Aceite os próximos diálogos e reinicie a IDE.

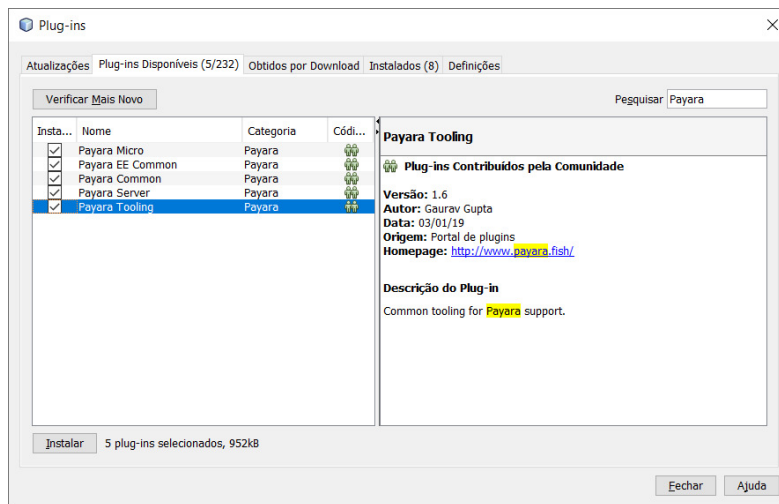


Figura 9 – Tela de instalação do plugin do Payara.

Após a IDE Netbeans reiniciar, acesse a guia serviços, de depois clique com o botão direito do mouse na opção servidores, e depois selecione a opção adicionar servidor (Figura 10).

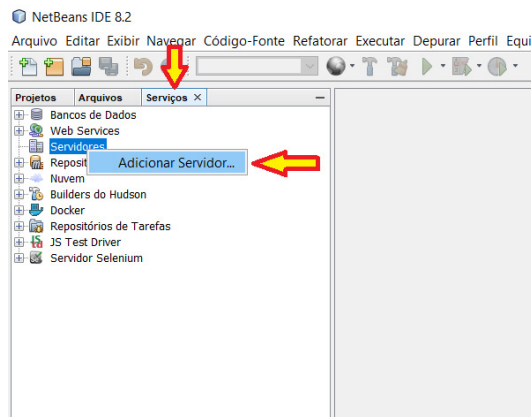


Figura 10 – Adicionando servidor no netbeans

Na tela que se abrirá selecione o servidor Payara e no nome informa Payara Server, como pode ser visualizado na Figura 11. Clique no botão próximo.

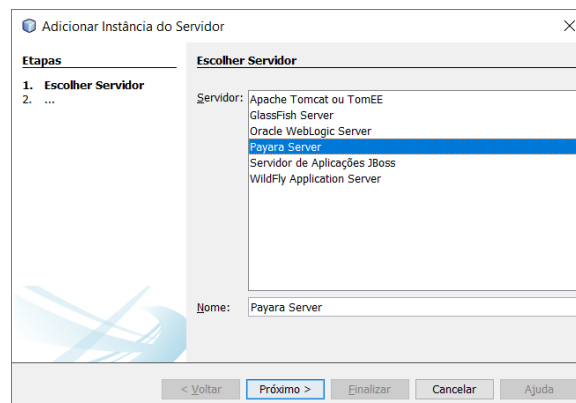


Figura 11 – Adicionar instância do servidor

Na tela seguinte é necessário selecionar o local onde foi descompactado o Payara (Figura 12). Para isso clique no botão procurar e selecione a pasta do Payara. Marque a opção de aceite do contrato de licença e clique em próximo.

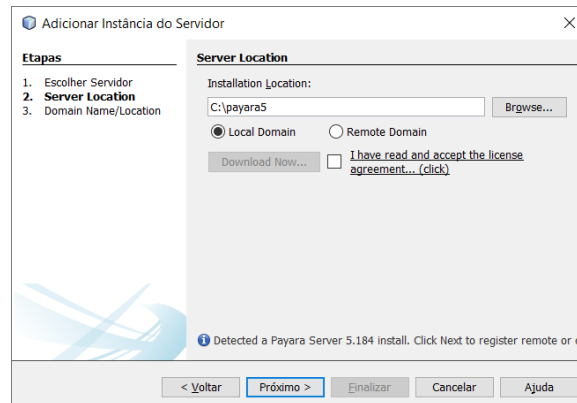


Figura 12 – Localização do servidor

A última tela de configuração do servidor é a tela de localização do domínio (Figura 13). Deixe as configuração padrão e clique em finalizar.

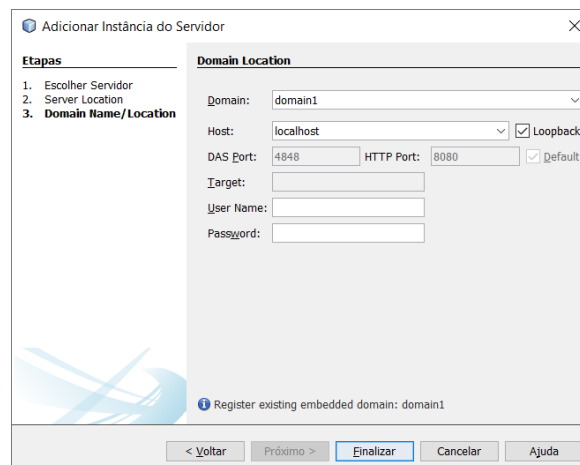


Figura 13 – Localização do domínio

Após concluído o registro do servidor na IDE, é necessário inicializa-lo para validar a instalação. Na guia serviços, acessa a opção servidores e clique com o botão direito do mouse no servidor Payara Server, e selecione a opção Start (Figura 13). O servidor será iniciado, caso seja solicitado autorização do firewall do windows autorize. Após o início do servidor acesse no navegador o endereço <<http://localhost:8080/>>, e se o servidor estiver rodando normalmente a tela da Figura 15 surgirá.

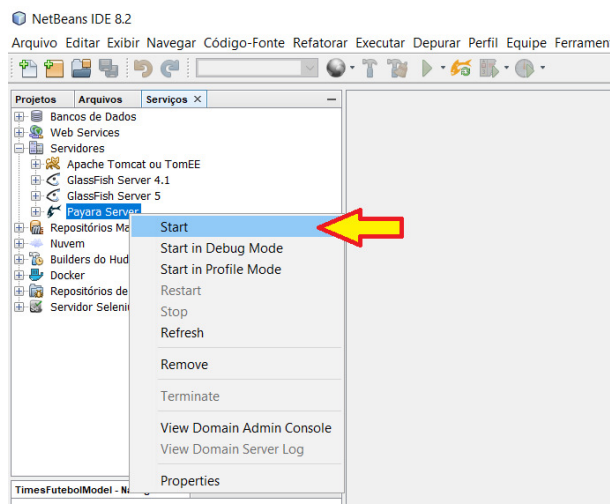


Figura 14 – Iniciando o servidor Payara

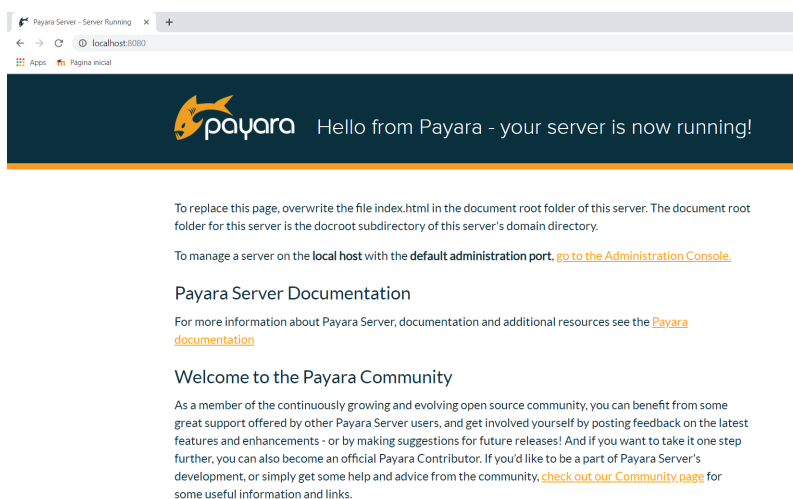


Figura 15 – Tela do Payara no navegador.

1.1.3 Instalação do SGBD PostgreSQL

O sistema gerenciador de banco de dados utilizado no livro será o PostgreSQL, não necessitando de uma versão em específico. Na data em que este livro está sendo produzido a versão atual é a 11.5, que pode ser obtida acessando o endereço <<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>>, onde deve ser efetuado o download da versão para windows, conforme mostra a Figura 16. O link para download direto é este: <<https://www.enterprisedb.com/thank-you-downloading-postgresql?anid=1256714>>.

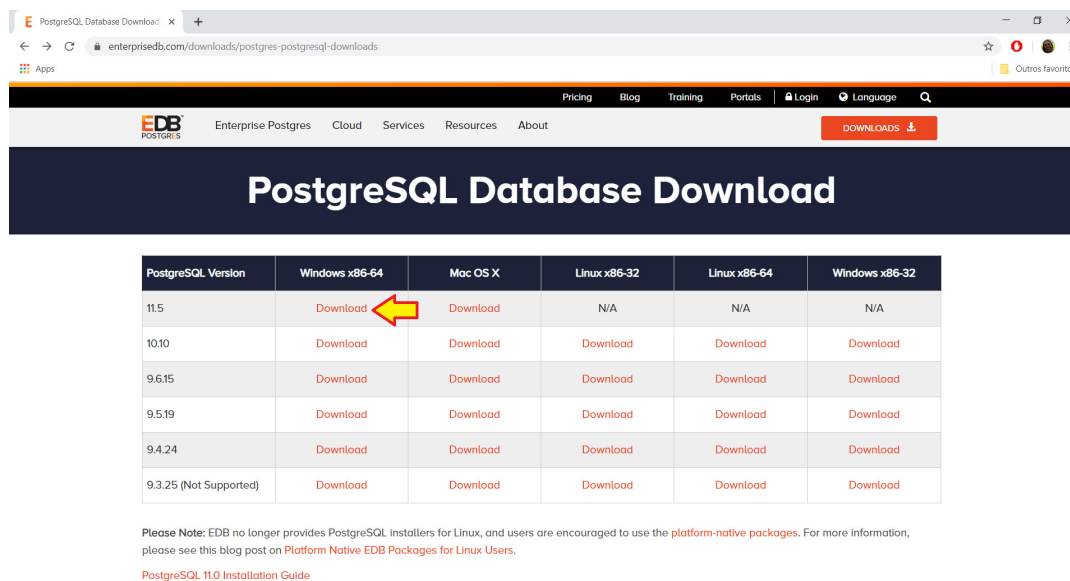


Figura 16 – Download do PostgreSQL.

Após o download ser concluído, execute o arquivo, e a tela de boas vindas será exibida (Figura 17), clique no botão next.

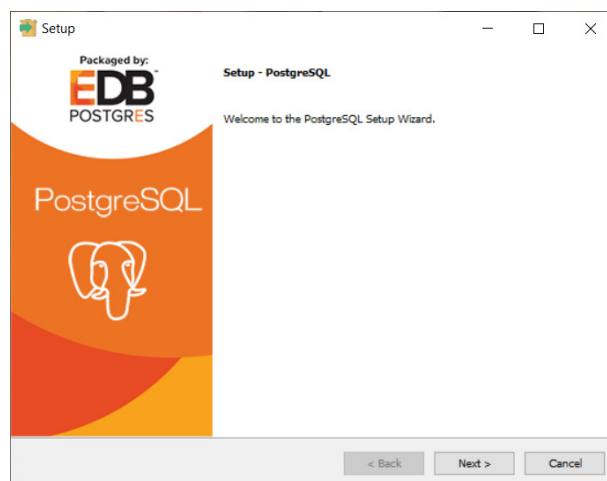


Figura 17 – Tela de boas vindas da instalação do PostgreSQL

Na próxima tela selecione o diretório para instalação e clique em next (Figura 18).

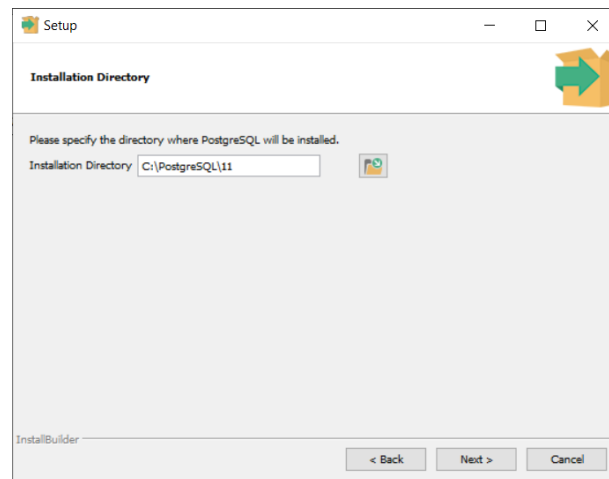


Figura 18 – Escolha do diretório de instalação do PostgreSQL

A tela seguinte apresenta a opção de selecionar componentes na instalação (Figura 19), onde deve-se selecionar além do PostgreSQL Server, o software pgAdmin4, que é um cliente gráfico para o PostgreSQL, onde pode-se criar bancos de dados, tabelas e executar comandos SQL. Após selecionar o pgAdmin4 clique em next.

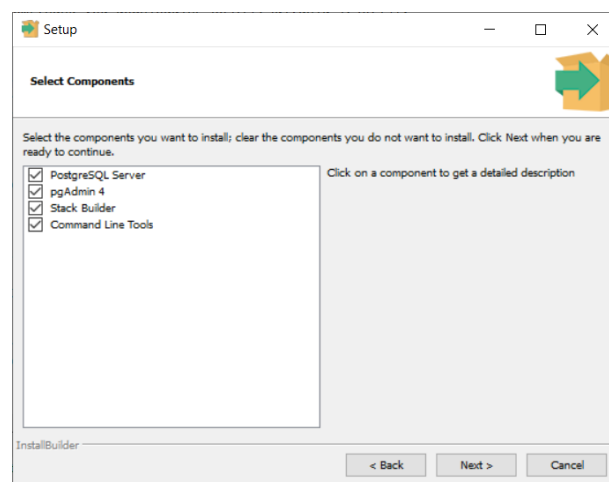


Figura 19 – Escolha de componentes na instalação do PostgreSQL

O próximo passo é para a escolha do diretório que armazena os dados do PostgreSQL, conforme apresentado na Figura 20. Escolha o diretório e clique em next.

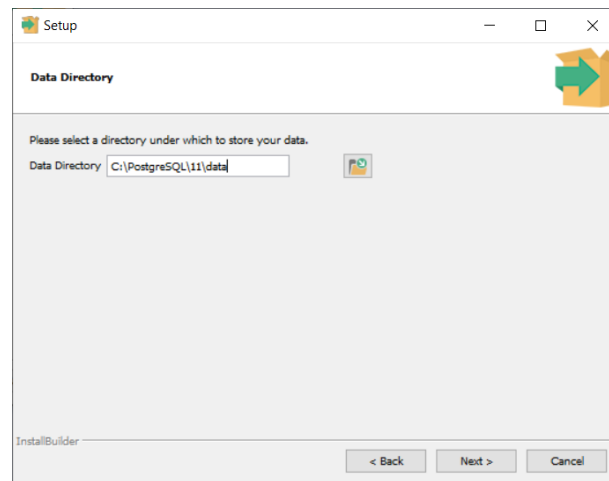


Figura 20 – Escolha do diretório de dados do PostgreSQL

Na tela apresentada na [Figura 20](#), deve-se informar a senha para o usuário postgres, que é o super usuário do banco. Informe postgres como senha, ou escolha uma outra. Clique em next, e na próxima tela pode-se escolher a porta TCP/IP em que o banco irá funcionar ([Figura 22](#)). Por padrão a porta 5432 é escolhida. Modifique caso seja necessário e clique em next.

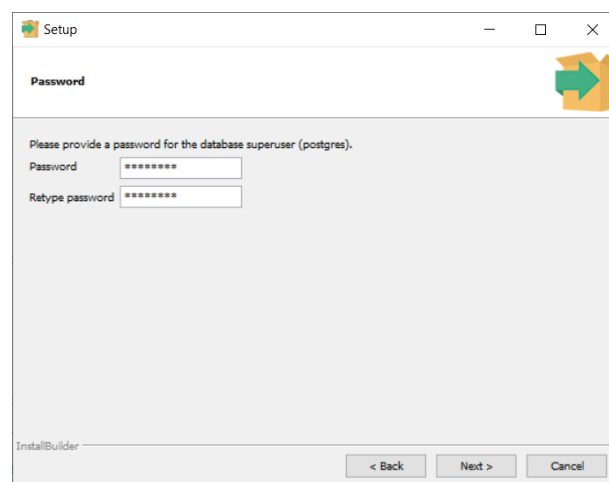


Figura 21 – Senha do usuário postgres

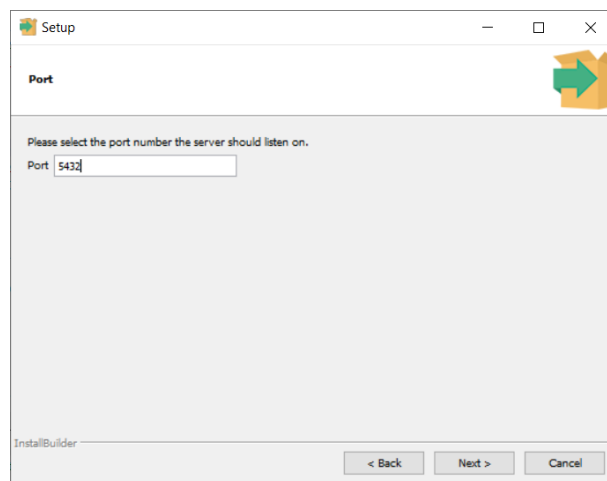


Figura 22 – Escolha da porta de funcionamento do SGBD

Em seguida será exibida uma tela para escolha da localidade padrão para o banco (Figura 23). Escolha a localidade desejada, ou utilize a seleção *default locale* para usar a localidade padrão do sistema operacional. Clique em next, e na próxima tela next novamente para finalizar a instalação.

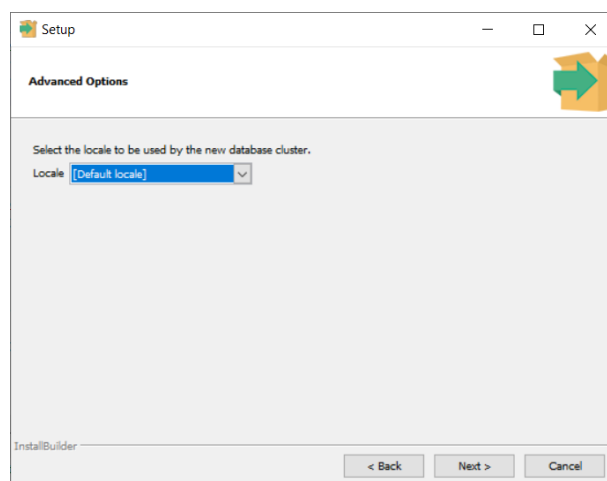


Figura 23 – Escolha da localidade do SGBD

Para validar a instalação, execute o aplicativo pgAdmin4, que pode ser encontrado no menu iniciar. Informe a senha cadastrada do usuário postgres criada durante a instalação para acessar o aplicativo. No menu a esquerda do aplicativo são listados os servidores registrados, clique no servidor PostgreSQL 11, e se ele exibir informações como a de um database com o nome postgres, como pode ser visualizado na Figura 24, o banco de dados está funcionando normalmente.

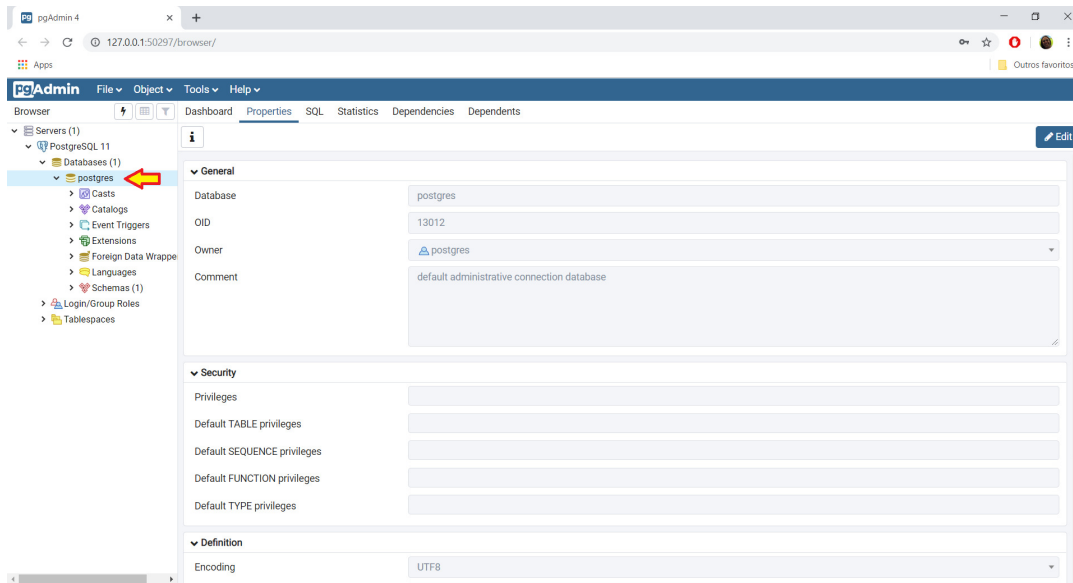


Figura 24 – Tela do pgAdmin4

1.1.4 Instalação da IDE JasperStudio

A IDE JasperStudio será utilizada para se criar relatórios. Ela pode ser obtida no endereço <<https://community.jaspersoft.com/project/jaspersoft-studio/releases>>. Selecione o versão 6.6.0 para o seu sistema operacional, conforme exibido na Figura 25.

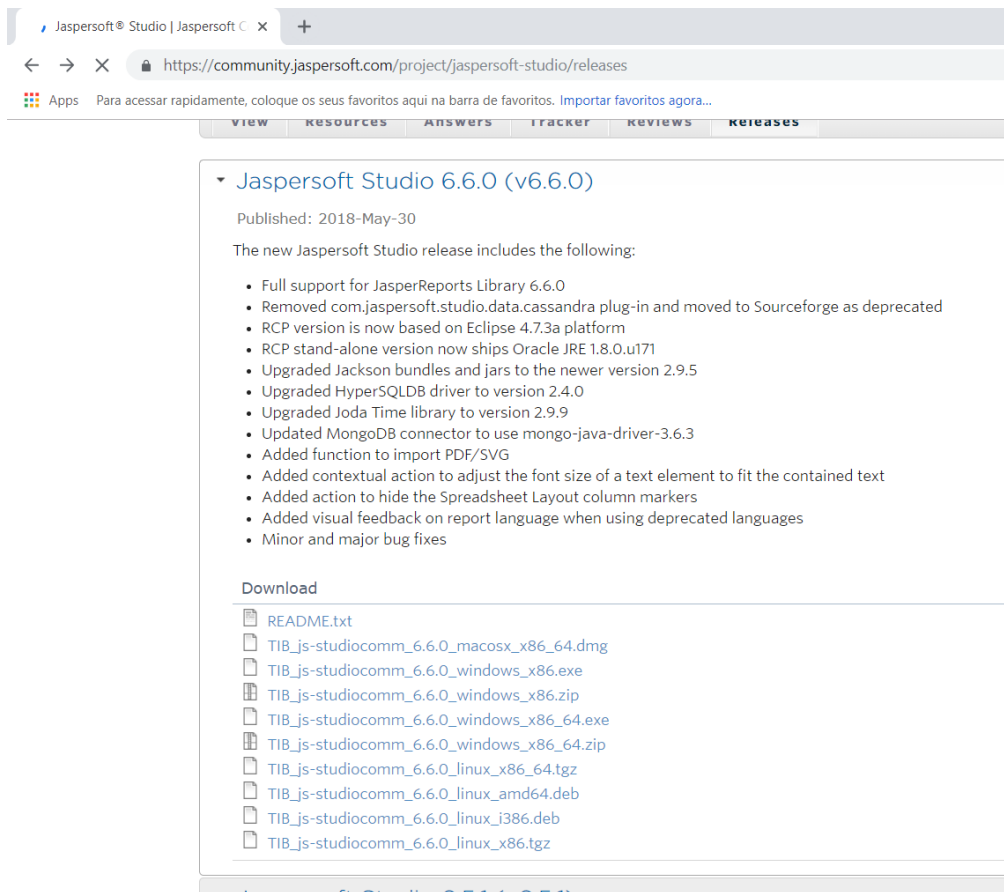


Figura 25 – Tela de download do JasperStudio

Após o download ser concluído, execute o arquivo. Na tela que se abrirá, clique no botão “I Agree”, e na próxima tela (Figura 26), selecione o local de instalação e clique em install. Após concluir a instalação será exibida uma tela onde está marcado para executar a IDE. Finalize esta tela e a IDE se abrirá, solicitando um diretório para criar a workspace da IDE, pode deixar o padrão que vem selecionado e a IDE estará instalada.

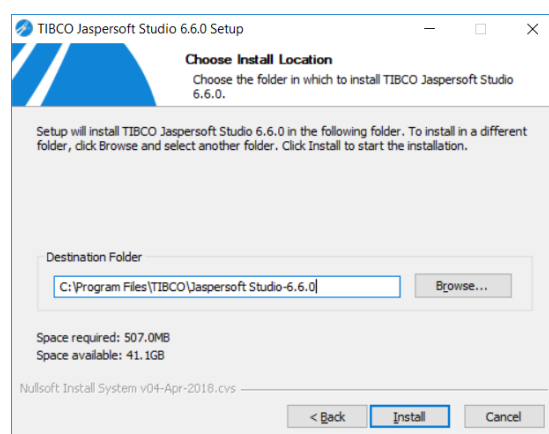


Figura 26 – Tela de seleção do diretório do JasperStudio

1.2 Bibliotecas utilizadas no livro

Nos projetos do estudo de caso do livro serão utilizadas algumas bibliotecas, que podem ser visualizadas na Tabela 2. As bibliotecas podem ser obtidas nos sites dos respectivos fornecedores, porém o link disponibilizado nesta obra se refere a conta do GitHub do autor. O motivo é manter as versões específicas utilizadas no livro disponíveis para download, que mantém a compatibilidade com os códigos fontes disponibilizados. Nestas bibliotecas já estão incluídas as dependências das mesmas.

Tabela 2 – Bibliotecas utilizadas no livro

Biblioteca	Versão	Endereço para download
Hibernate JPA 2.2	5.3.6	< https://github.com/jlbavaresco/Java-Libraries/blob/master/Hibernate%205.3.6%20JPA%202.2/Hibernate%205.3.6%20JPA%202.2.zip?raw=true >
Hibernate Validator	5.4.2	< https://github.com/jlbavaresco/Java-Libraries/blob/master/Hibernate%20Validator%205.4.2/Hibernate%20Validator%205.4.2.zip?raw=true >
Driver JDBC PostgreSQL	42.2.5	< https://github.com/jlbavaresco/Java-Libraries/blob/master/JDBCPostgreSQL/postgresql-42.2.5.jar?raw=true >
JasperReports	6.6.0	< https://github.com/jlbavaresco/Java-Libraries/blob/master/JasperReports6.6.0/JasperReports6.6.0.zip?raw=true >
Primefaces	7	< https://github.com/jlbavaresco/Java-Libraries/blob/master/PrimeFaces/primefaces-7.0.jar?raw=true >

1.2.1 Criação e registro das bibliotecas no Netbeans

Para facilitar e organizar o uso das bibliotecas, será necessário registrá-las na IDE Netbeans, o que irá facilitar a sua posterior utilização nos projetos envolvidos. Será registrada a biblioteca do Hibernate JPA, e o mesmo procedimento deverá ser realizado para as bibliotecas Hibernate Validator e Jasper Reports. Primeiramente a biblioteca deve ser obtida (<<https://github.com/jlbavaresco/Java-Libraries/blob/master/Hibernate%20Validator%205.4.2/Hibernate%20Validator%205.4.2.zip?raw=true>>) e descompactada.

O registro da biblioteca deve ser feito na IDE Netbeans acessando o menu Ferramentas e após bibliotecas. Na tela que se abrirá clique em nova biblioteca (Figura 27). Na tela seguinte informe “Hibernate 5.3.6 JPA 2.2”. Com a biblioteca selecionada no menu à esquerda, clique no botão “Adicionar JAR/Pasta” (Figura 27) e navegue até a pasta onde os arquivos da biblioteca foram descompactados. Selecione todos os arquivos e depois clique em OK.

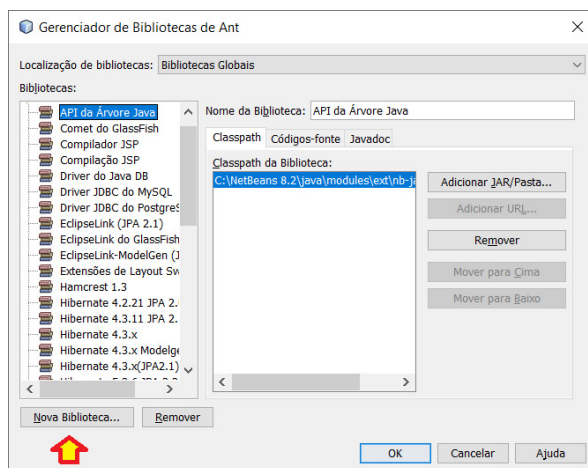


Figura 27 – Tela do gerenciador de bibliotecas

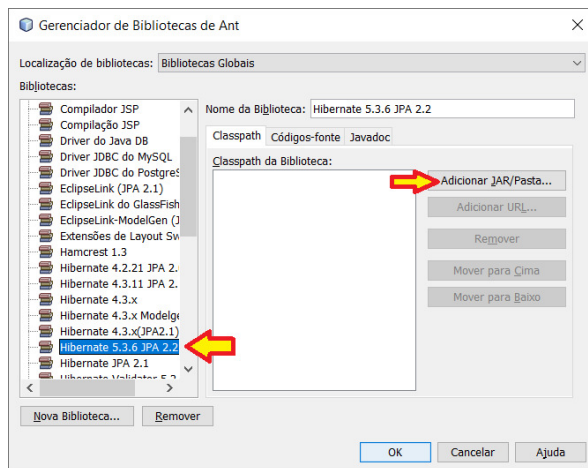


Figura 28 – Tela do gerenciador de bibliotecas - Adicionando arquivos da biblioteca

A tela de gerenciamento da biblioteca deve ficar igual a apresentada na [Figura 29](#) após a adição dos arquivos. Repita o procedimento para as bibliotecas do Hibernate Validator e Jasper Reports.

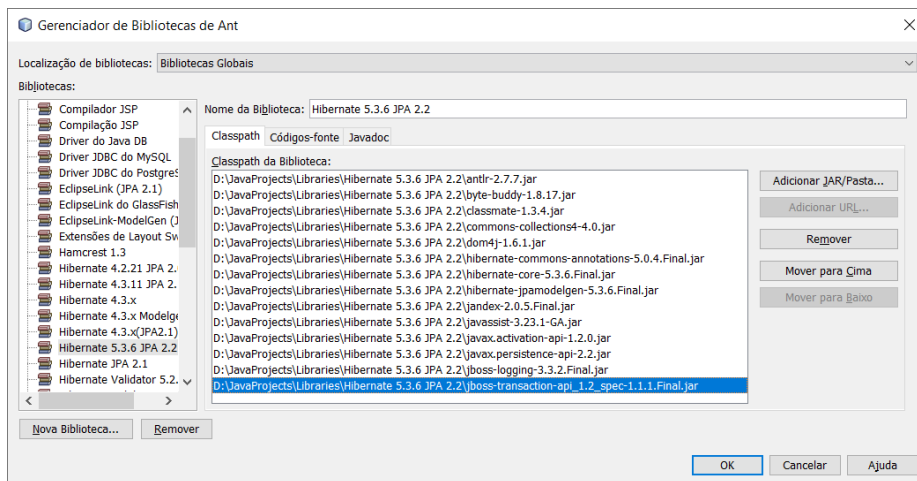


Figura 29 – Tela da biblioteca após o seu registro

Persistência com JPA

Neste capítulo será apresentada JPA (Java Persistence API), que pode ser usada para realizar o mapeamento objeto-relacional a fim de persistir dados de uma aplicação. Num primeiro momento serão abordados os conceitos da JPA e do framework Hibernate, e depois se apresentará as principais anotações para mapeamento objeto relacional. Para fixar os assuntos abordados será utilizado um estudo de caso para exemplificar os conceitos de forma prática.

2.1 JPA - Java Persistence API

A *Java Persistence API* (JPA) é uma especificação que foi criada para facilitar a persistência de dados, e é responsável por realizar o mapeamento objeto-relacional da camada de persistência, estabelecendo uma relação entre as tabelas relacionais do banco de dados e as classes de modelo (POJO). Desta forma, uma linha existente em uma tabela no banco de dados transforma-se em uma instância de um objeto na aplicação. Sendo assim, se obtêm uma união dos modelos orientado a objetos e do modelo relacional.

Um dos principais objetivos da versão 5 da plataforma *Java Enterprise Edition* (Java EE 5) é a facilidade de desenvolvimento. Mudanças surgiram na plataforma para tornar o desenvolvimento de aplicações Java mais fácil e com menos codificação, porém mantendo todas as funcionalidades existentes na versão anterior da plataforma. Esta simplificação atingiu a tecnologia *Enterprise JavaBeans* (EJB), e a versão do EJB 3.0 adicionou novos recursos a versão existente, EJB 2.1. Destaca-se a adição na tecnologia EJB a API *Java Persistence*, que surgiu para simplificar o modelo de persistência (BISWAS; ORT, 2006).

Um dos meios de persistência na tecnologia Java é por meio da *Java Database Connectivity* (JDBC), que é a api padrão para acesso a bases de dados relacionais. Com ela pode-se conectar a uma base de dados relacional e executar comandos em linguagem SQL (*Structured Query Language*), para tratar operações CRUD (*Create, Read, Update e Delete*), responsáveis pelas operações de inserção, alteração, exclusão e consulta de registros em um banco de dados relacional. Porém a utilização de JDBC pode tornar a manutenção de uma aplicação complexa, visto que exige do desenvolvedor escrever comandos SQL específicos para cada banco de dados utilizado, e qualquer modificação no modelo ou no banco de dados exige modificações nas consultas SQL escritas, fato que culminou para o surgimento de ferramentas de mapeamento objeto relacional (ORM).

Princípios de ORM envolvem o acesso à base de dados com ferramentas, que por sua vez fornecem uma versão orientada a objetos de bases relacionais e vice-versa. Essas ferramentas fornecem uma correspondência bidirecional entre a base de dados e os objetos (GONCALVES, 2011). Essas ferramentas são chamadas de provedores de persistência, e entre os mais populares pode-se citar o Hibernate¹, o

¹ <<http://hibernate.org/orm/>>

TopLink² e o Java Data Objects (JDO)³. Porém, cada uma destas ferramentas possui a configuração para utilização muito específica, o que dificultava o desenvolvimento, pois na necessidade de mudança de provedor, os mapeamentos realizados e configurações também precisavam ser modificados. A JPA surgiu para padronizar o mapeamento objeto-relacional com a sua inclusão na especificação EJB 3.0, e adotou muitos princípios do Hibernate culminando no surgimento da especificação da JPA 1.0.

A JPA é uma abstração que funciona sobre o JDBC, provendo independência de SQL. Classes e anotações da JPA se encontram no pacote `javax.persistence`. Dentre os componentes destacam-se uma API gerenciadora de entidades para realizar operações CRUD na base de dados, uma linguagem de consulta em JAVA (JPQL), mecanismos de transação e bloqueio providos pela JTA (Java Transaction API) e suporte a transações de recursos locais não providos pela JTA (GONCALVES, 2011). Outro fato importante é que a JPA independe de banco de dados e provedor de persistência, sendo que este é necessário para a sua utilização.

A versão 1.0 da JPA surgiu com uma abordagem leve, definindo padrões para sua utilização, muito baseada no provedor Hibernate. Ela permitia o mapeamento objeto-relacional por anotações ou por arquivos xml. Na versão da JPA 2.0 foram adicionados novos recursos, como coleções de tipos de dados simples e objetos embutíveis, possibilidade de ordenações de coleções a anotação `@OrderColumn`, tratamento da remoção de objetos orfãos quando o objeto pai é removido, adição de bloqueio pessimista, uma API de critérios para realização de consultas totalmente orientada a objetos, novas funcionalidades na sintaxe da linguagem de consulta JPQL e padronização de algumas propriedades do arquivo `persistence.xml` (GONCALVES, 2011). Na versão da JPA 2.1 destaca-se a possibilidade de mapeamento de procedimentos armazenados no banco de dados.

2.2 Framework de persistência Hibernate

A JPA é apenas uma especificação que padroniza a persistência no Java, sendo necessário o uso de algum framework de persistência para realizar o mapeamento objeto-relacional em nossas aplicações. Entre os frameworks que implementam a especificação da JPA está o Hibernate⁴. Com ele é possível mapear classes Java em tabelas de banco de dados e vice-versa. Ele suporta mapeamento de associações entre objetos, herança, polimorfismo, composição e coleções.

2.2.1 Biblioteca Hibernate 5.3 JPA 2.2

Neste livro será utilizada a versão 5.3 do Hibernate⁵ que implementa a especificação 2.2 da JPA. O download da biblioteca pode ser realizado em <https://github.com/jlbavaresco/Java-Libraries/raw/master/Hibernate%205.3.6%20JPA%202.2/Hibernate%205.3.6%20JPA%202.2.zip>.

2.3 Estudo de caso

Para exemplificar os conceitos abordados no livro, de desenvolvimento de sistemas com a plataforma Java EE, será implementado um estudo de caso, do desenvolvimento de um sistema para gerenciar ordens de serviço de uma empresa de conserto de eletrônicos. Sobre o levantamento de requisitos para este sistema, ele deverá manter dados dos usuários (colaboradores da empresa) e dos clientes, bem como as permissões de acesso que cada papel terá. O sistema também deverá manter dados das ordens de serviço, o que inclui os dados do equipamento, os serviços e produtos que foram utilizados, possíveis fotos do estado do equipamento e dados relacionados ao pagamento. É necessário também manter dados de produtos disponíveis, e para estes deve ser possível armazenar arquivos no sistema, como manuais, firmwares entre outros.

Sendo assim, foram identificados os principais requisitos que o sistema deverá implementar:

² <<http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>>

³ <https://db.apache.org/jdo/jdo_v_jpa_orm.html>

⁴ <<http://hibernate.org/orm/>>

⁵ <<http://hibernate.org/orm/releases/5.3/>>

- Manter dados de usuários;
- Manter dados de clientes;
- Manter permissões de usuários e clientes;
- Manter dados de ordens de serviço;
- Manter dados de serviços;
- Manter dados de produtos;
- Manter arquivos de produtos;
- Manter dados de equipamentos;
- Manter contas a receber;
- Proteger o acesso ao sistema com autenticação de usuários;
- Proteger o acesso a funcionalidades do sistema conforme a permissão do usuário;
- Gerar relatório de ordem de serviço;

Com base nos requisitos levantados, elaborou-se um diagrama de classes com as entidades necessárias na aplicação. Este diagrama pode ser visualizado na [Figura 30](#). Este diagrama será implementado na camada de modelo, onde será realizado o mapeamento objeto-relacional, estabelecendo uma relação entre os objetos da aplicação e o banco de dados, e depois na camada web onde se terá manutenções CRUD sobre as entidades do diagrama.

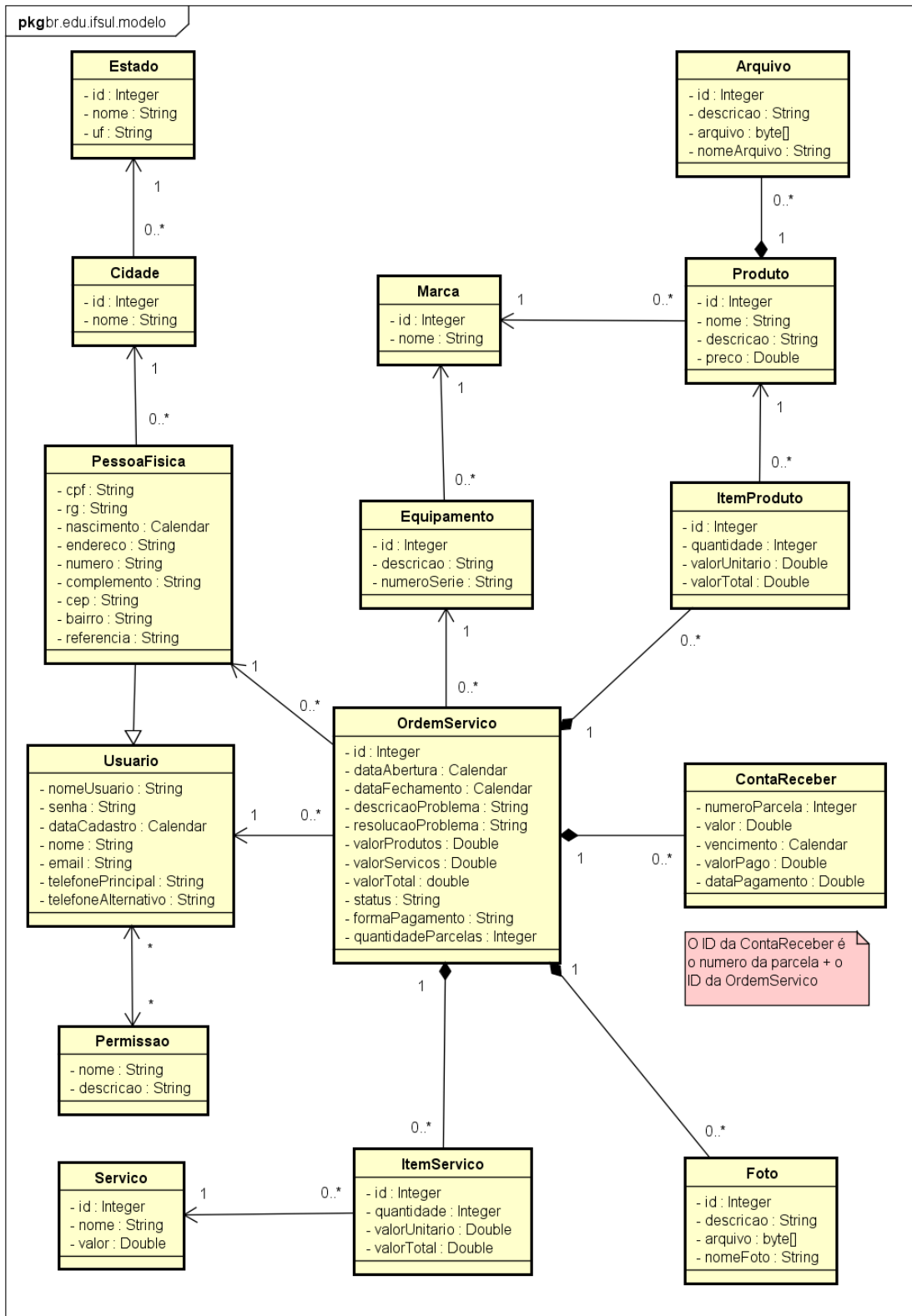


Figura 30 – Diagrama de classes do estudo de caso

2.4 Mapeamento objeto-relacional com anotações JPA

O mapeamento objeto relacional consiste no mapeamento das classes e atributos, de modo que as informações de uma classe sejam persistidas em uma tabela do banco de dados, e que os atributos de uma classe sejam persistidos em colunas de tabelas do banco de dados. Basicamente se indica para uma classe em qual tabela do banco suas informações, e para o atributo em qual coluna. Isso se faz por meio de anotações, as quais são colocadas na classe que se deseja mapear. É possível também que as tabelas do banco de dados sejam geradas automaticamente após o mapeamento objeto-relacional.

A [Figura 31](#) apresenta a classe estado com seus atributos e um exemplo de como ficaria a tabela para armazenar as suas informações. Já no [Código Fonte 2.1](#) podemos observar um trecho do código fonte da classe Estado com as anotações para realizar o mapeamento objeto-relacional.

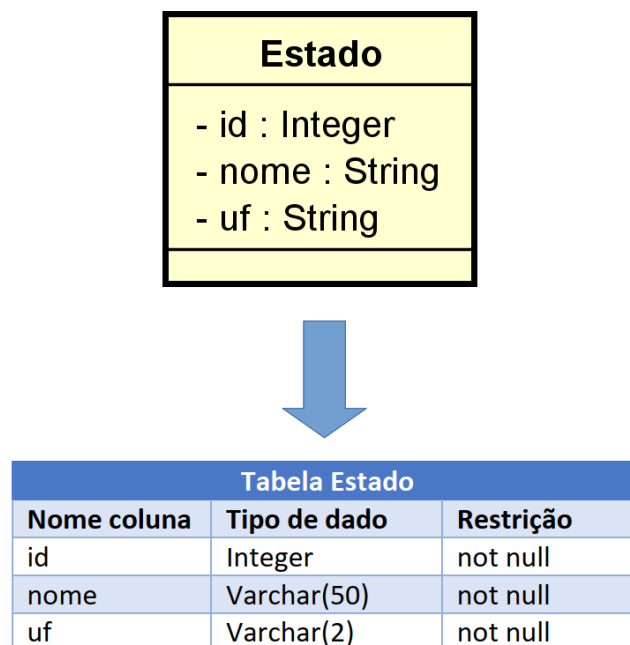


Figura 31 – Diagrama de classes do estudo de caso

```

1 @Entity
2 @Table(name = "estado")
3 public class Estado implements Serializable {
4     @Id
5     @SequenceGenerator(name = "seq_estado", sequenceName = "seq_estado_id", allocationSize = 1)
6     @GeneratedValue(generator = "seq_estado", strategy = GenerationType.SEQUENCE)
7     private Integer id;
8     @Column(name = "nome", length = 50, nullable = false)
9     private String nome;
10    @Column(name = "uf", length = 2, nullable = false)
11    private String uf;

```

Código Fonte 2.1 – Treço da classe Estado

2.4.1 Anotações da JPA

O mapeamento objeto-relacional pode ser realizado por meio de anotações nas classes. Estas anotações irão indicar como a JPA deve persistir as informações no banco de dados relacional. Esta sessão irá

apresentar algumas das principais anotações de mapeamento.

2.4.1.1 Anotação @Entity

Anotação que indica para a JPA que a classe em questão é uma entidade persistente, ou seja, o seu estado pode ser persistido em um banco de dados. Anotação de uso obrigatório. Um exemplo do seu uso pode ser visto no [Código Fonte 2.2](#), na linha 1. A anotação deve ser declarada antes do nome da classe.

```
1 @Entity
2 public class Estado implements Serializable {
3
4     private Integer id;
```

Código Fonte 2.2 – Anotação @Entity

2.4.1.2 Anotação @Table

Anotação que indica o nome da table que irá armazenar os dados dos objetos da classe mapeada. Um exemplo do seu uso pode ser visto no [Código Fonte 2.3](#), na linha 2. A anotação deve ser declarada antes do nome da classe.

```
1 @Entity
2 @Table(name = "estado")
3 public class Estado implements Serializable {
4
5     private Integer id;
```

Código Fonte 2.3 – Anotação @Table

2.4.1.3 Anotação @Id

Anotação de uso obrigatório, que indica qual o atributo que será o identificador da classe. Para a persistência de uma classe as únicas anotações obrigatórias são @Entity e @Id. O uso desta anotação em um atributo irá gerar a chave primária na tabela do banco de dados. Um exemplo do seu uso pode ser visto no [Código Fonte 2.4](#), na linha 5.

```
1 @Entity
2 @Table(name = "estado")
3 public class Estado implements Serializable {
4
5     @Id
6     private Integer id;
```

Código Fonte 2.4 – Anotação @Id

2.4.1.4 Anotação @SequenceGenerator e @GeneratedValue

São anotações que tratam da geração automática da chave primária na tabela do banco de dados. No [Código Fonte 2.5](#), na linha 2 é utilizada a anotação @SequenceGenerator. Ela trata da criação de uma sequencia no banco de dados, que terá o nome *seq_estado_id*, definida pelo atributo *sequenceName*. O atributo *allocationSize* define o incremento usado par gerar o id, neste caso com o valor 1. Já o atributo *name* desta anotação define o nome de uma variável, que será usada pela anotação @GeneratedValue.

No [Código Fonte 2.5](#), na linha 3 é utilizada a anotação @GeneratedValue. O atributo *generator* indica qual o gerador será utilizado, cujo valor aponta para o valor do atributo *name* de uma anotação @SequenceGenerator. Já o atributo *strategy* indica a estratégia para a geração do valor, que no caso utiliza sequencias do banco de dados. Com o uso em conjunto destas duas anotações ao se persistir uma instância de uma classe, o valor do id será gerado automaticamente pelo banco de dados.

```
1 @Id
2 @SequenceGenerator(name = "seq_estado", sequenceName = "seq_estado_id", allocationSize = 1)
3 @GeneratedValue(generator = "seq_estado", strategy = GenerationType.SEQUENCE)
4 private Integer id;
```

Código Fonte 2.5 – Anotação @SequenceGenerator e @GeneratedValue

Quando a geração do ID é realizada no banco de dados, utilizando-se do tipo “serial” no banco de dados PostgreSQL ou “autoincrement” no MySQL, não é necessário o uso da anotação @SequenceGenerator, somente da anotação @GeneratedValue utilizando a estratégia de geração IDENTITY, como pode ser visualizado na linha 2 do [Código Fonte 2.6](#). Ao se persistir uma classe com este tipo de mapeamento não se informa o ID, e ele será gerado pelo banco de dados.

```
1 @Id
2 @GeneratedValue(strategy = GenerationType.IDENTITY)
3 private Integer id;
```

Código Fonte 2.6 – Anotação @GeneratedValue com strategy GenerationType.IDENTITY

2.4.1.5 Anotação @Column

Anotação para realizar o mapeamento de um atributo de uma classe, relacionando-o a uma coluna do banco de dados. Com ela pode-se definir o nome da coluna e restrições, e no [Código Fonte 2.7](#) pode-se visualizar exemplos do seu uso. Abaixo seguem as definições de alguns dos atributos da anotação @Column:

- name: Define o nome da coluna no banco de dados usada para armazenar o valor do atributo.
- nullable: Define se a coluna irá permitir valores nulos ou não. Valor padrão false.
- length: Define o tamanho do campo quando se usar o tipo String. Na linha 1 do [Código Fonte 2.7](#), na propriedade length se utilizou o valor 50, desta forma a coluna será criada com o tipo de dado “varchar(50)”. Somente informar esta propriedade quando utilizar-se tipo String.
- unique: Define se a coluna permite ou não armazenar valor repetidos. Valor padrão false, se for utilizado true será criado um índice único associado a respectiva coluna no banco de dados.
- insertable: Define se o atributo será incluído em comando insert. Valor padrão true.
- updatable: Define se o atributo será incluído em comando update. Valor padrão true.
- columnDefinition: A JPA cria a coluna no banco de dados conforme o tipo utilizado no Java. Por exemplo caso se use String irá criar no banco de dados um tipo “varchar”. Em alguns casos é necessário informar o tipo que se deseja utilizar. O atributo columnDefinition serve para isso. No [Código Fonte 2.7](#), na linha 3 utilizou-se o valor “text”, para que a coluna descrição use este tipo, possibilitando informar um texto sem limite de caracteres. No mesmo código fonte, na linha 5, utilizou-se em columnDefinition o valor “numeric(12,2)” para se criar uma coluna que armazene números com 12 dígitos e duas casas decimais.

```
1 @Column(name = "nome", length = 50, nullable = false)
2 private String nome;
3 @Column(name = "descricao", columnDefinition = "text")
4 private String descricao;
5 @Column(name = "preco", nullable = false, columnDefinition = "numeric(12,2)")
6 private Double preco;
```

Código Fonte 2.7 – Anotação @Column

2.4.1.6 Anotação @Temporal

Em Java pode-se trabalhar com tipos de dados para data e horas como as classes `java.util.Date` e `java.util.Calendar`. Estes tipos de dados armazenam data, hora, minutos, milisegundos, etc. Para persistir este tipo de dados e para a JPA criar os campos de forma adequada a coluna no banco de dados, é necessário fazer uso da anotação `@Temporal` para se definir daquele tipo de dado o que se deseja persistir. Os valores possíveis são `TemporalType.DATE` (Código Fonte 2.8 linha 1), `TemporalType.TIME` (Código Fonte 2.8 linha 4) e `TemporalType.TIMESTAMP` (Código Fonte 2.8 linha 7), para persistir respectivamente data, hora e data e hora. Em alguns versões do Hibernate o valor `TemporalType.TIME` apresenta problemas, e se fazer necessário definir o tipo de dado pelo valor da propriedade `columnDefinition` como “time”, da propriedade `@Column`, como pode ser visualizado na linha 10 do Código Fonte 2.8.

```
1 @Temporal(TemporalType.DATE)
2 @Column(name = "nascimento", nullable = false)
3 private Calendar nascimento;
4 @Temporal(TemporalType.TIME)
5 @Column(name = "hora", nullable = false)
6 private Calendar hora;
7 @Temporal(TemporalType.TIMESTAMP)
8 @Column(name = "dataHora", nullable = false)
9 private Calendar dataHora;
10 @Column(name = "dataHora", nullable = false, columnDefinition = "time")
11 private Calendar outraHora;
```

Código Fonte 2.8 – Anotação @Temporal

2.4.1.7 Anotação @Inheritance

O mapeamento de herança no Java é um recurso simples de ser utilizado. No estudo de caso abordado no livro, apresentado no diagrama de classes da Figura 30, observa-se a classe `Usuário` que fornece herança para a classe `PessoaFisica`. O mapeamento da herança no Java se dá na classe `PessoaFisica`, simplesmente declarando na definição da classe `PessoaFisica` que ela estende de `Usuario` (“`PessoaFisica extends Usuario`” - Código Fonte 2.10 linha 5). Na JPA é necessário informar como esse relacionamento irá gerar o banco de dados.

Para utilizar a herança é necessário declarar a anotação `@Inheritance` na classe que fornece herança, informando qual estratégia será usada para gerar o banco de dados, cujo valor pode ser:

- `InheritanceType.JOINED`: Utiliza uma tabela para persistir os dados de cada classe.
- `InheritanceType.SINGLE_TABLE`: Mapeia os dados de toda a hierarquia em uma única tabela.
- `InheritanceType.TABLE_PER_CLASS`: Cada entidade é mapeada em uma tabela dedicada, porém os dados da entidade raiz (pai) também serão mapeados na classe filha.

No Código Fonte 2.9 pode-se observar o mapeamento da classe `Usuário`, com o uso da estratégia `JOINED`. Outro recurso que pode ser utilizado é a anotação `@DiscriminatorColumn`, onde pode-se informar uma coluna para guardar o tipo do registro que foi salvo. No caso deste código fonte observa-se na linha 5 a definição de uma coluna chamada “tipo” que irá armazenar o valor “US” quando for um usuário e “PF” quando for uma pessoa física. O valor que será armazenado é definido na anotação `@DiscriminatorValue`.

```
1 // Classe Usuário
2 @Entity
3 @Inheritance(strategy = InheritanceType.JOINED)
4 @Table(name = "usuario")
5 @DiscriminatorColumn(name = "tipo", discriminatorType = DiscriminatorType.STRING,length = 2)
6 @DiscriminatorValue(value = "US")
```

```
7 public class Usuario implements Serializable {
8     @Id
9     @Column(name = "nome_usuario", length = 20, nullable = false)
10    private String nomeUsuario;
```

Código Fonte 2.9 – Anotação para mapeamento de herança - trecho classe Usuário

```
1 // Classe Pessoa Física
2 @Entity
3 @Table
4 @DiscriminatorValue(value = "PF")
5 public class PessoaFisica extends Usuario implements Serializable {
6
7     @Column(name = "cpf", length = 14, nullable = false)
8     private String cpf;
```

Código Fonte 2.10 – Anotação para mapeamento de herança - trecho classe PessoaFisica

2.4.1.8 Anotações @ManyToOne e @JoinColumn

A anotação @ManyToOne deve ser usada quando se deseja mapear relacionamento muitos para um, ou quando se tem uma chave estrangeira no banco de dados. No estudo de caso do livro a classe Cidade possui um Estado, o que se tornará no banco de dados, na tabela cidade uma chave estrangeira apontando para o id da tabela estado. Em termos de classe, a classe Cidade possui um atributo do tipo Estado.

No **Código Fonte 2.11** na linha 1, a anotação @ManyToOne indica que o atributo estado faz referência a um relacionamento muitos para um. Na linha 2, utiliza-se a anotação @JoinColumn para indicar o nome da coluna na tabela cidade que irá armazenar o id do estado (name = "estado"), qual coluna da tabela estado ela irá referenciar (referencedColumnName = "id") e se permite valores nulos (nullable = false). Já na linha 3 o atributo foreignKey define o nome da chave estrangeira que será criada no banco de dados.

```
1 @ManyToOne
2 @JoinColumn(name = "estado", referencedColumnName = "id", nullable = false,
3     foreignKey = @ForeignKey(name = "fk_cidade_estado"))
4 private Estado estado;
```

Código Fonte 2.11 – Anotação @ManyToOne e @JoinColumn

2.4.1.9 Anotação @OneToMany

A anotação @OneToMany é utilizada para mapear relacionamento um para muitos, onde um objeto de origem faz referência a uma lista de objetos. Tal tipo de relacionamento pode ser visualizado no estudo de caso abordado no livro, na relação entre a classe Produto e a classe Arquivo. Um produto pode possuir um ou mais arquivos, sendo que uma instância de Arquivo não existe sem uma instância de Produto, por se tratar de um relacionamento de agregação por composição.

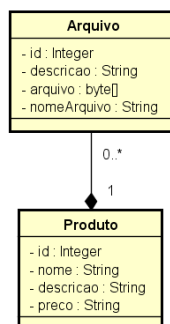


Figura 32 – Classes Produto e Arquivo

Para realizar o mapeamento do relacionamento, primeiro deve-se mapear o Produto na classe arquivo, cujo mapeamento pode ser visualizado no [Código Fonte 2.12](#).

```

1 @ManyToOne
2 @JoinColumn(name = "produto", referencedColumnName = "id", nullable = false,
3 foreignKey = @ForeignKey(name = "fk_arquivo_produto"))
4 private Produto produto;
  
```

Código Fonte 2.12 – Anotação @OneToMany - mapeamento na classe Arquivo

Depois deve-se mapear o relacionamento da classe Produto. O relacionamento irá se tornar uma lista do tipo Arquivo ([Código Fonte 2.13](#) linha 3). Após, deve-se usar a anotação @OneToMany, para que quando se carregar do banco de dados um objeto do tipo Produto, também se recupere a lista de arquivos, ou quando se persista um objeto Produto no banco de dados também se persista seus arquivos. A anotação possui alguns atributos:

- **mappedBy**: Indica no nome do atributo na classe da lista que estabelece a relação com a classe pai. No caso o valor “produto” é o nome do atributo na classe arquivo que referencia a classe Produto.
- **cascade**: Indica que operações de persistência serão propagadas para os filhos da lista. Os valores possível são CascadeType.ALL (todas as operações), CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REMOVE, CascadeType.REFRESH e CascadeType.DETACH
- **orphanRemoval**: Indica se os filhos da classe devem ser removidos caso a classe principal seja removida.
- **fetch**: Indica o momento em que a coleção é recuperada. O valor FetchType.LAZY faz com que a coleção seja recuperada tardiamente, somente quando a coleção for utilizada, ou quando se chama os métodos size ou contains da coleção. Já o valor FetchType.EAGER é usado quando se deseja recuperar previamente a coleção, quando se realiza uma consulta ou se recuperar um objeto da classe. O uso de FetchType.EAGER pode acarretar um desempenho pior em consulta por exemplo, por ter que recuperar mais objetos do banco de dados.

```

1 @OneToMany(mappedBy = "produto", cascade = CascadeType.ALL,
2 orphanRemoval = true, fetch = FetchType.LAZY)
3 private List<Arquivo> arquivos = new ArrayList<>();
  
```

Código Fonte 2.13 – Anotação @OneToMany - mapeamento na classe Produto

2.4.1.10 Anotação @ManyToMany

Anotação utilizada para se mapear relacionamentos do tipo muitos para muitos. No estudo de caso proposto pelo livro, observa-se tal tipo de relacionamento entre as classes Usuário e Permissão. Na implementação a classe Usuário terá uma lista de Permissões. Para realizar o mapeamento basta adicionar a

anotação `@ManyToMany`, como pode ser observado no [Código Fonte 2.14](#) na linha 1. Caso seja necessário indicar o nome da tabela associativa deve-se usar a anotação `@JoinTable`. Por fim, para que seja gerada a chave primária da tabela associativa automaticamente deve-se usar a classe `java.util.Set` para criar a coleção de permissões.

```

1 @ManyToMany(fetch = FetchType.LAZY)
2 @JoinTable(name = "permissoes",
3     joinColumns =
4     @JoinColumn(name = "nome_usuario", referencedColumnName = "nome_usuario", nullable = false)
5     ,
6     inverseJoinColumns =
7     @JoinColumn(name = "permissao", referencedColumnName = "nome", nullable = false))
private Set<Permissao> permissoes = new HashSet<>();

```

Código Fonte 2.14 – Anotação `@ManyToMany`

2.4.1.11 Chave composta com anotação `@Embeddable` e `@EmbeddedId`

A JPA permite realizar o mapeamento de uma chave primária composta em uma tabela do banco de dados. Isto permite mapear situações onde um ou mais atributos serão utilizados como chave primária da tabela. Um exemplo é o mapeamento da classe `ContaReceber`, onde o identificador será o código da ordem de serviço e o número da parcela. O SQL de criação da tabela pode ser visualizado no [Código Fonte 2.15](#).

Código Fonte 2.15 – Código de criação da tabela contas receber

```

1 CREATE TABLE conta_receber (
2     numero_parcela integer NOT NULL,
3     data_pagamento date,
4     valor numeric(10,2) NOT NULL,
5     valor_pago numeric(10,2),
6     vencimento date NOT NULL,
7     ordem_servico integer NOT NULL,
8     CONSTRAINT conta_receber_pkey PRIMARY KEY (numero_parcela, ordem_servico),
9     CONSTRAINT fk_item_servico_os FOREIGN KEY (ordem_servico)
10 );

```

Para se realizar o mapeamento, primeiro deve-se criar uma classe onde os dois atributos chaves ficarão. No caso será a classe `ContaReceberID`, que pode ser visualizado no [Código Fonte 2.16](#). Os atributos chave são definidos nas linhas 6 e 11, e na linha 1 usa-se a anotação `@Embeddable` para definir que esta classe será embutida ou utilizada em uma classe persistente.

```

1 @Embeddable
2 public class ContaReceberID implements Serializable {
3
4     @NotNull(message = "O número da parcela deve ser informado")
5     @Column(name = "numero_parcela", nullable = false)
6     private Integer numeroParcela;
7
8     @NotNull(message = "A Ordem de serviço deve ser informada")
9     @ManyToOne(fetch = FetchType.LAZY)
10    @JoinColumn(name = "ordem_servico", referencedColumnName = "id", nullable = false,
11    foreignKey = @ForeignKey(name = "fk_item_servico_os"))
private OrdemServico ordemServico;

```

Código Fonte 2.16 – Trecho da Classe `ContaReceberID`

Na classe que utilizará a chave composta, no caso a classe `ContaReceber`, exibida no [Código Fonte 2.17](#), basta definir o atributo do tipo da classe `ContaReceberID` (linha 6) e indicar que ele será o ID, com a anotação `@EmbeddedId` (linha 5).

```
1 @Entity
2 @Table(name = "conta_receber")
3 public class ContaReceber implements Serializable {
4
5     @EmbeddedId
6     private ContaReceberID id;
```

Código Fonte 2.17 – Trecho da Classe ContaReceber

2.5 Validação de dados com Hibernate Validator

A Bean Validation API foi introduzida no Java EE 6, e a partir da sua especificação (JSR 303) o Hibernate Validator surgiu como implementação de referência. A versão atual é a 2.0 e faz parte da plataforma Java EE 8.

O funcionamento do framework é simples, basta adicionar restrições a atributos, métodos ou classes na forma de anotações, para tornar possível verificar os dados. É possível por exemplo, adicionar restrições em atributos de classes para verificar os valores informados, mesmo não se tratando de uma classe que será persistida em um banco de dados. Por ser parte da especificação da plataforma Java EE, se integra a outros frameworks como JSF e a JPA.

A ideia principal é facilitar a validação, no caso deste trabalho, as restrições ficarão na camada de modelo, e todas as camadas que utilizarem ela não precisam reescrever as validações, tornando possível centralizar a validação no modelo e reutilizar em uma camada web ou camada de serviços.

2.5.1 Anotações para validação

A validação dos dados se dá por meio de anotações. Todas as anotações possuem um atributo *message*, onde pode-se informar a mensagem que será exibida ao usuário caso o valor do atributo não seja considerado válido. A lista completa das anotações disponíveis pode ser verificada no site com a documentação do Hibernate Validator⁶. Abaixo segue uma lista das principais anotações:

- **@Length(min=, max=)**: Verifica se o tamanho da string esta dentro dos limites.
- **@Max(value=)**: Verifica se o valor é maior ou igual. Aplicado a representações de números, armazenados até mesmo em String.
- **@Min(value=)**: Verifica se o valor é menor ou igual. Aplicado a representações de números, armazenados até mesmo em String.
- **@NotNull**: Checa se o valor não é null.
- **@NotEmpty**: Checa se não é null e não vazia.
- **@Past**: Checa se uma data esta no passado.
- **@Future**: Checa se uma data esta no futuro.
- **@Pattern(regex="regexp", flag=)** ou **@Patten(@Pattern(...))**: Checa se a propriedade obedece à expressão regular.
- **@Range(min=, max=)**: Confere se o valor está entre o mínimo e o máximo (incluindo), aplicado a valores numéricos, mesmo armazenados em String.
- **@Size(min=, max=)**: Confere se a quantidade de elementos esta entre o mínimo e o máximo (incluído), aplicado a Array, Collection, Map.
- **@AssertFalse**: Confere se o valor de uma propriedade boolean é false.
- **@AssertTrue**: Confere se o valor de uma propriedade boolean é true

⁶ <https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#validator-defineconstraints-spec>

- **@Email**: Confere se a string é um e-mail válido.
- **@CreditCardNumber**: Confere se a string é um número de cartão de crédito válido.
- **@Digits**: Verifica quantidade de dígitos ante e depois do separador de casa decimal (integerDigits e factionalDigits).
- **@EAN**: Verifica se a string é um código EAN ou UPC-A válido
- **@SafeHtml**: Verifica se a String não contem fragmentos de códigos maliciosos como <script/>
- **@URL**: Verifica se a string informada é uma url valida. Pode ser informado o protocolo host e porta aceitos.
- **@CNPJ**: Verifica se a string informada é um CNPJ válido
- **@CPF**: Verifica se a string informada é um CPF válido
- **@TituloEleitoral**: Verifica se a String informada é um titulo eleitoral válido

2.5.2 Anotando uma classe e invocando o processo de validação

Para anotar uma classe, adicionando a validação com a Bean Validation API, basta adicionar as anotações de validação antes da declaração do atributo que se deseja validar. No exemplo da classe Estado (Código Fonte 2.18) é necessário validar os atributos nome e uf, para verificar se eles não são nulos, não estão informados em branco, e se o tamanho da string informada não ultrapassa o limite de caracteres estabelecida.

Para realizar a validação para verificar se o atributo não é nulo, usa-se a anotação `@NotNull`, como pode ser verificado no Código Fonte 2.18 na linha 23. A validação será realizada e quando ela for violada, ao se informar um valor nulo no atributo, será gerada uma mensagem de erro, que pode ser personalizada no atributo desta anotação chamado *message*. Basicamente o valor informado é o que será exibido ao usuário. Caso não seja informado será exibida a mensagem padrão da anotação.

Na linha 24 do Código Fonte 2.18 usa-se a anotação `@NotBlank`, para evitar que o usuário informe uma *string* vazia que passa na validação de não nulo.

A anotação utilizada para verificar o tamanho da *string* é a `@Length`, e seu uso pode ser visualizado na linha 24 do Código Fonte 2.18. Nela se informa o valor 50 para o atributo *max*, impondo assim a limitação de 50 caracteres para o atributo. No atributo *message*, usa-se o coringa *{max}* para que na mensagem seja usado o valor informado no atributo *max*.

```
1 package br.edu.ifsul.modelo;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5 import javax.persistence.Column;
6 import javax.persistence.Entity;
7 import javax.persistence.GeneratedValue;
8 import javax.persistence.GenerationType;
9 import javax.persistence.Id;
10 import javax.persistence.SequenceGenerator;
11 import javax.persistence.Table;
12 import javax.validation.constraints.NotNull;
13 import org.hibernate.validator.constraints.Length;
14 import org.hibernate.validator.constraints.NotBlank;
15
16 @Entity
17 @Table(name = "estado")
18 public class Estado implements Serializable {
19     @Id
20     @SequenceGenerator(name = "seq_estado", sequenceName = "seq_estado_id", allocationSize = 1)
21     @GeneratedValue(generator = "seq_estado", strategy = GenerationType.SEQUENCE)
```

```
22 private Integer id;
23 @NotNull(message = "O nome não pode ser nulo")
24 @NotBlank(message = "O nome não pode ser em branco")
25 @Length(max = 50, message = "O nome não pode ter mais que {max} caracteres")
26 @Column(name = "nome", length = 50, nullable = false)
27 private String nome;
28 @NotNull(message = "A UF não pode ser nula")
29 @NotBlank(message = "A UF não pode ser em branco")
30 @Length(max = 2, min = 2, message = "A UF deve ter {max} caracteres")
31 @Column(name = "uf", length = 2, nullable = false)
32 private String uf;
33
34 public Estado() {
35 }
36
37 // métodos getters, setters e equal e hash code omitidos
38 }
```

Código Fonte 2.18 – Trecho da Classe Estado com validações

O processo de validação será chamado automaticamente ao tentar persistir o objeto com a JPA, ou ao se tentar submeter um formulário no JSE, para classes que possuam as anotações de validação. Caso seja necessário chamar a validação manualmente, é necessário obter um objeto do tipo *Validator* (Código Fonte 2.19 linha 10), e depois usar o método que esta classe possui chamado *validate*, onde deve se informar por parâmetro o objeto que se deseja validar. O retorno deste método é um *Set* do tipo *ConstraintViolation*, tipado para a classe que se vai validar, que irá conter uma lista das violações de validação que ocorreram o objeto informado. Depois basta percorrer esta lista para ter acesso a estes erros. O processo completo pode ser visualizado no Código Fonte 2.19.

```
1 import br.edu.ifsul.modelo.Estado;
2 import java.util.Set;
3 import javax.validation.ConstraintViolation;
4 import javax.validation.Validation;
5 import javax.validation.Validator;
6
7 public class ValidarEstado {
8     public static void main(String[] args) {
9         Estado obj = new Estado();
10        Validator validador = Validation.buildDefaultValidatorFactory().getValidator();
11        Set<ConstraintViolation<Estado>> erros = validador.validate(obj);
12        for (ConstraintViolation<Estado> erro : erros){
13            System.out.println("Erro: "+erro.getMessage());
14        }
15    }
16 }
```

Código Fonte 2.19 – Chamando a validação da Classe Estado

2.6 Camada de modelo - Mapeamento Objeto-relacional do estudo de caso

Nesta seção será realizada a criação de um projeto, para realizar o mapeamento objeto-relacional do estudo de caso contido no diagrama de classes apresentado na Figura 30. Neste livro para facilitar o reaproveitamento de código, será criado um projeto específico para a camada de modelo, que contém os mapeamentos objeto-relacional, e outro projeto, para a camada web, que reaproveita o código fonte do primeiro. Esta é uma boa prática de programação, que deixa os projetos com um fraco acoplamento.

O código fonte completo do projeto da camada de modelo está disponível no GITHUB, no endereço <<https://github.com/jlbavaresco/OSEletronicosModel>>.

2.6.1 Criando o projeto da camada de modelo

Para a criação do projeto da camada de modelo, será utilizada a IDE NetBeans, cuja obtenção e instalação é descrita na [subseção 1.1.2](#). Para criar o novo projeto, com a IDE aberta, clique no botão Novo Projeto ou no menu Arquivo -> Novo Projeto. O diálogo apresentado na [Figura 33](#) irá aparecer, onde deve ser selecionado em categoria a opção Java, e em projetos a opção Aplicação Java. Após clique no botão próximo.

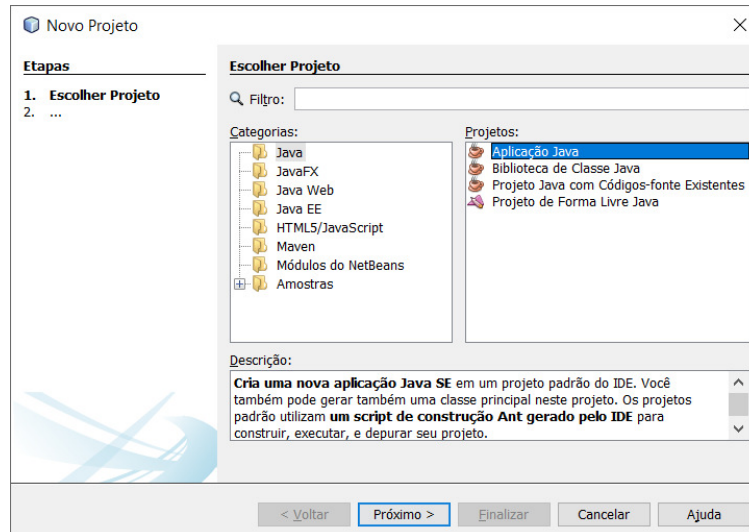


Figura 33 – Tela de seleção de tipo de projeto (criação do projeto da camada de modelo)

Na tela seguinte ([Figura 34](#)), informe o nome do projeto “OSEletronicosModel”, marque a opção “Usar Pasta Dedicada Para Armazenar Bibliotecas”. Isto fará com que as bibliotecas adicionadas ao projeto sejam mantidas em uma pasta chamada “lib” dentro da pasta onde o projeto será criado e armazenado. Após clique em finalizar. A estrutura inicial do projeto pode ser visualizada na [Figura 35](#).

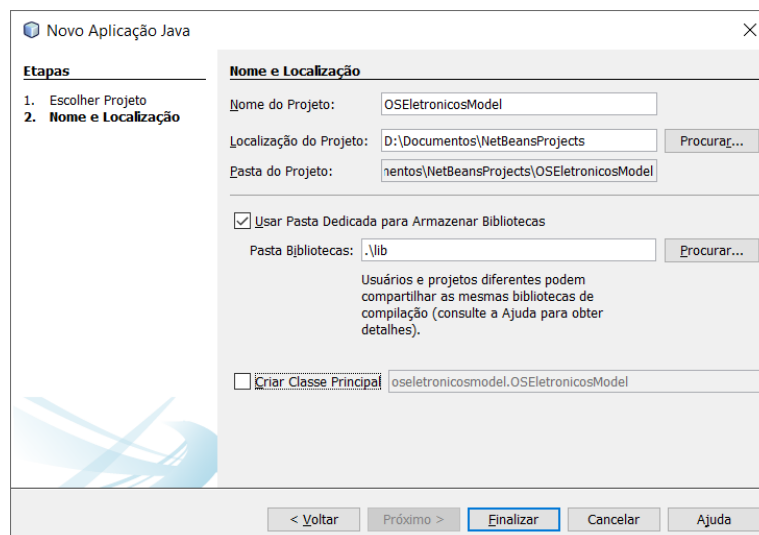


Figura 34 – Tela de criação do projeto da camada de modelo

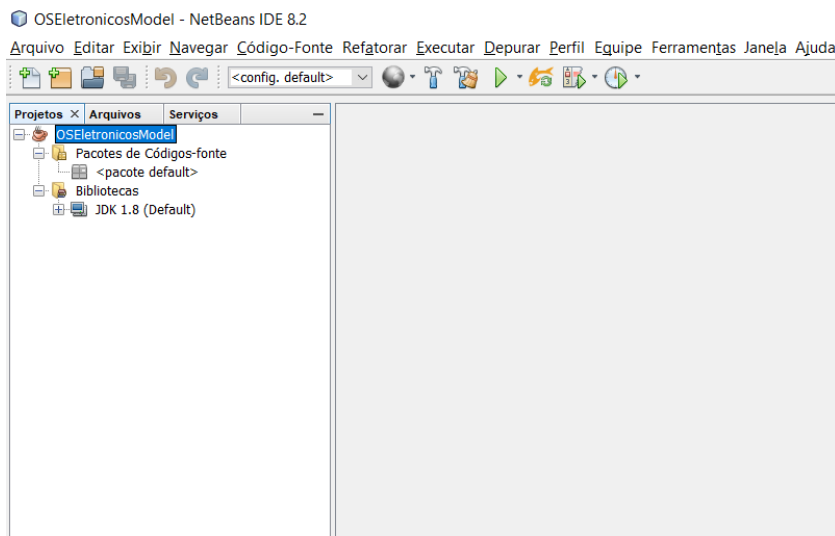


Figura 35 – Estrutura inicial do projeto da camada de modelo

2.6.1.1 Adicionando as bibliotecas

No projeto da camada de modelo, além da estrutura de classes já disponível na linguagem Java, serão necessárias algumas bibliotecas que implementam outras funcionalidades, como é o caso da biblioteca Hibernate JPA 2.2 5.3.6, Hibernate Validator 5.4.2 e o driver JDBC do PostgreSQL versão 42.2.5, cujos links de download se encontram na [Tabela 2](#). A configuração das bibliotecas é descrita na [subseção 1.2.1](#) e deve ser realizada antes dos próximos passos.

Para adicionar uma biblioteca ao projeto clique com o botão direito do mouse nas bibliotecas do projeto, e selecione a opção “Adicionar biblioteca” (Figura 36). Na tela que se abrirá (Figura 37), clique na opção “Importar”.

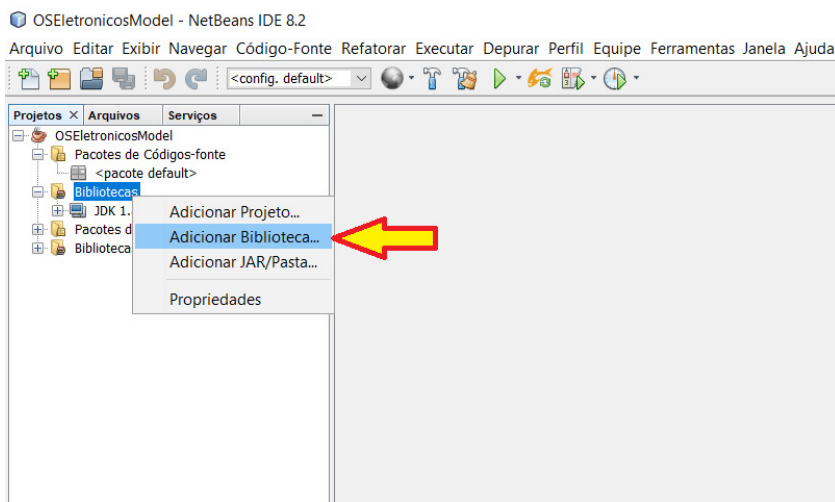


Figura 36 – Adicionando bibliotecas ao projeto - primeira tela

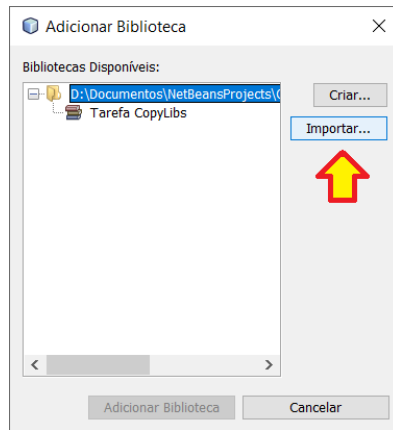


Figura 37 – Adicionando bibliotecas ao projeto - segunda tela

Será exibida uma tela com as bibliotecas que estão cadastradas no Netbeans, as que já existem por padrão na IDE e as que o usuário criou. Selecione a biblioteca que foi criada para o Hibernate (“Hibernate JPA 5.3.6 2.2”) e clique no botão importar biblioteca (Figura 38). Será exibida a tela anterior com a biblioteca (Figura 39), e nela clique no botão “Adicionar biblioteca”. Repita o procedimento para a biblioteca “Hibernate Validator 5.4.2”.

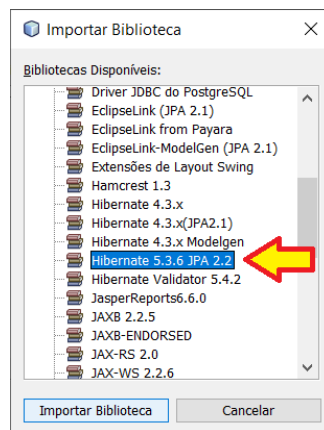


Figura 38 – Adicionando bibliotecas ao projeto - terceira tela

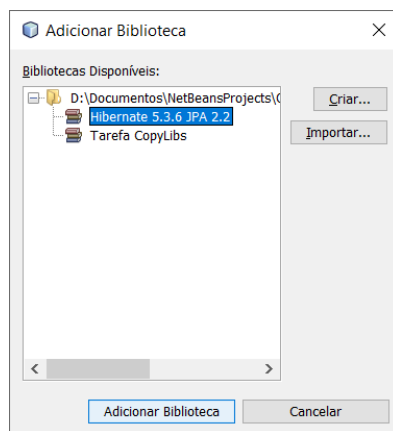


Figura 39 – Adicionando bibliotecas ao projeto - quarta tela

Além das bibliotecas adicionadas anteriormente, é necessário adicionar o driver JDBC do PostgreSQL. Primeiro obtenha o arquivo do driver JDBC (Tabela 2). Depois clique com o botão direito no mouse nas bibliotecas do projeto, selecione a opção “Adicionar JAR/Pasta”. Será aberto um diálogo, navegue até a pasta onde se encontra o arquivo do driver, selecione-o e clique no botão “Abrir” (Figura 40).

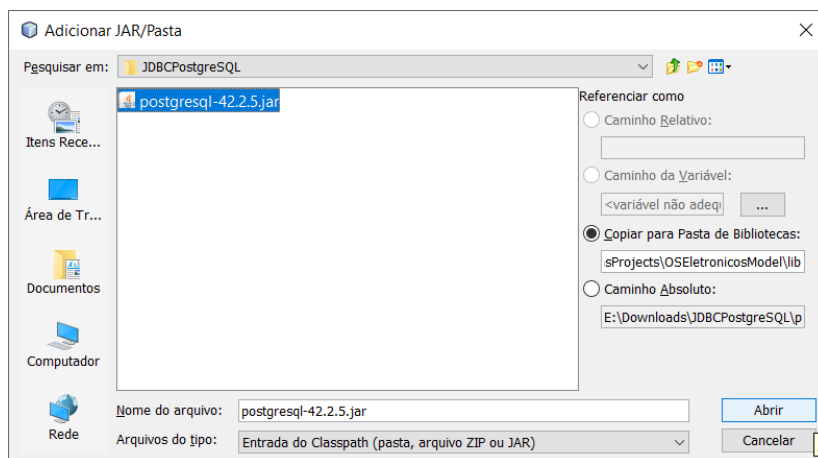


Figura 40 – Adicionando JAR ao projeto

Após a inclusão das bibliotecas e do driver JDBC do PostgreSQL no projeto, ao se expandir as bibliotecas clicando no sinal de “+”, a aparência deve ser igual a apresentada na Figura 41.

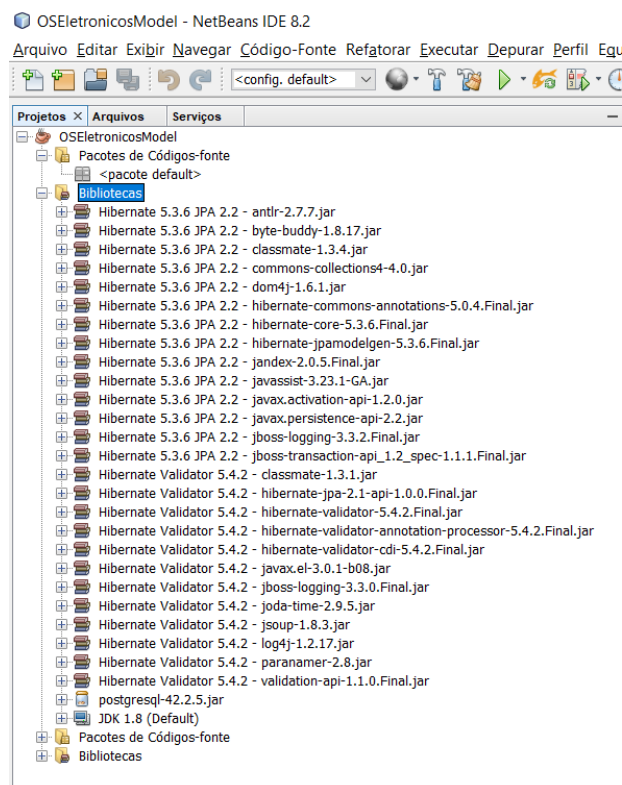


Figura 41 – Bibliotecas do projeto da camada de modelo.

2.6.1.2 Criando a Unidade de persistência (Persistence Unit)

Para utilização da JPA é necessário se ter uma unidade de persistência, que nada mais é que um elemento dentro de um arquivo chamado *persistence.xml*, e que pode conter definições como por exemplo, o nome do banco de dados utilizado, nome de usuário e senha do banco de dados, entre outras definições. Como utilizaremos a JPA para criar as tabelas do banco de dados, somente é necessária a criação do mesmo, antes da criação do arquivo *persistence.xml*.

Para criar o banco de dados, acesse o software pgAdmin 4. Na tela que será apresentada, clique com o botão direito do mouse em *Databases*, depois em *Create*, e *Database* (Figura 42). Na tela que se abrirá (Figura 43), informe no campo *Database* o valor “oseletronicos” e clique em *save*. Se o banco foi criado corretamente ele aparecerá na lista do pgAdmin 4, conforme exibido na Figura 44.

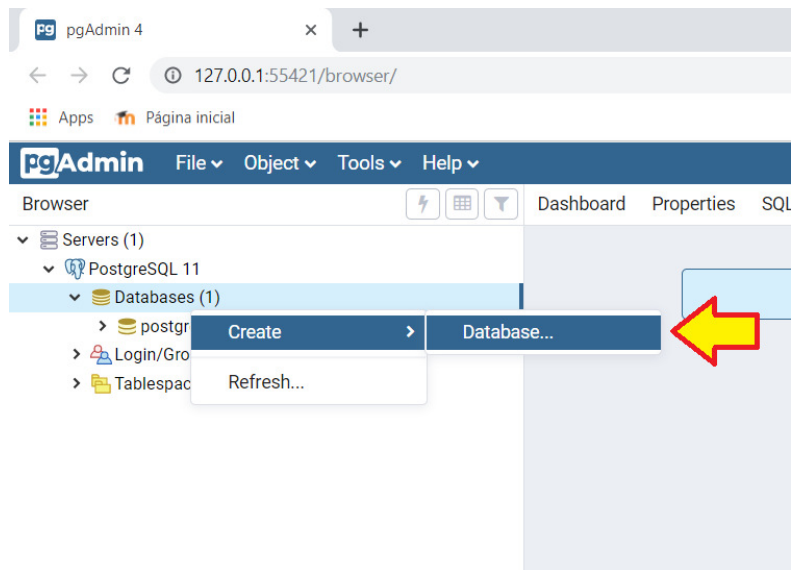


Figura 42 – pgAdmin 4 - Tela para criar base de dados

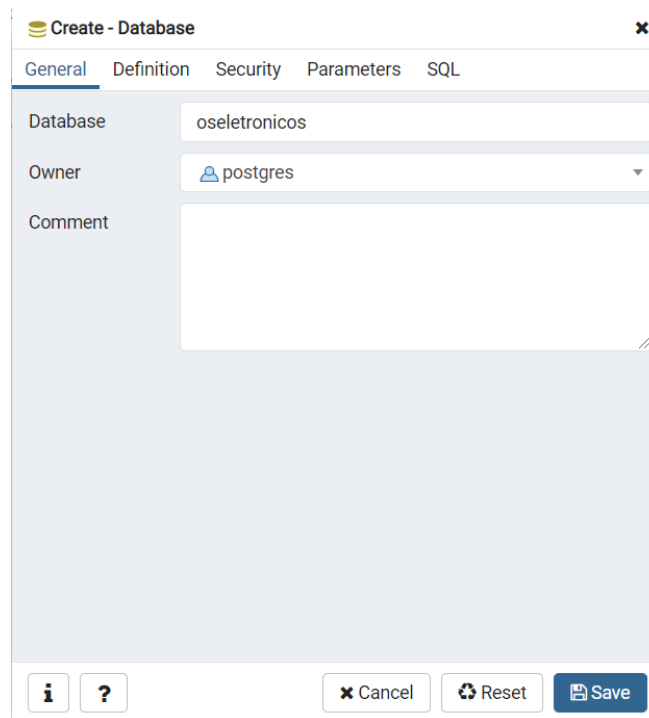


Figura 43 – pgAdmin 4 - Tela com informações para criar o banco de dados

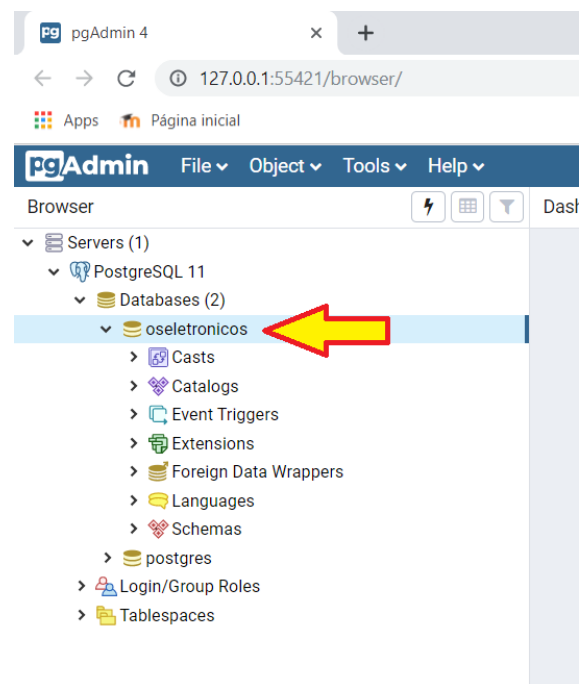


Figura 44 – pgAdmin 4 - Tela a listagem dos bancos de dados

Após a criação do banco de dados no PostgreSQL, é necessário que se registre a conexão no Netbeans. Para isso, acesse a guia serviços, e depois clique com o botão direito do mouse em banco de dados, e selecione a opção “Nova Conexão” (Figura 45).

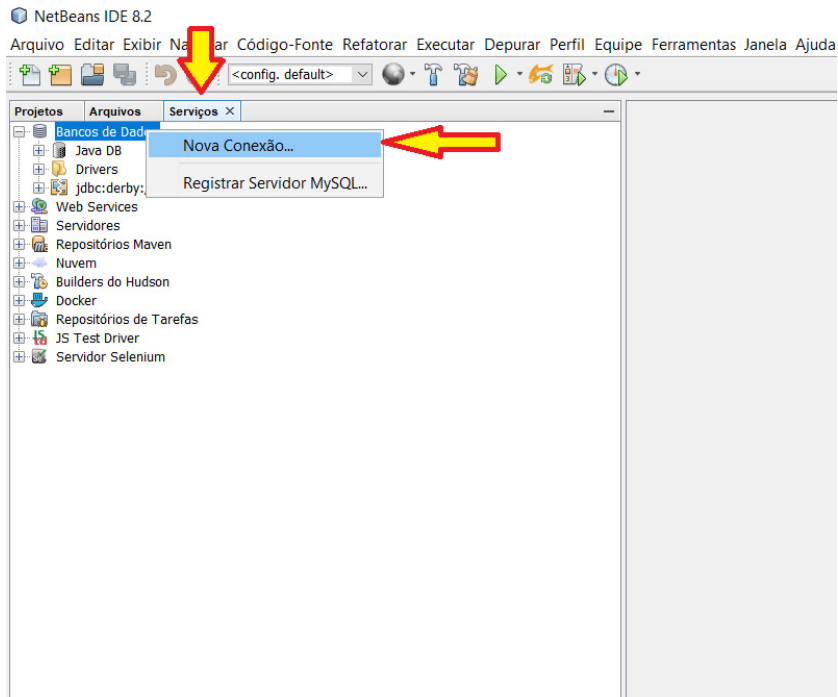


Figura 45 – Registrando a conexão no Netbeans

Será iniciado um assistente para registro da conexão. Na primeira tela (Figura 46), selecione o Driver do PostgreSQL, e clique no botão próximo. Na próxima tela, informe no campo banco de dados o nome “oseletronicos”, que é o nome do banco criado anteriormente. Na opção *Host*, o valor *localhost* corresponde ao endereço do servidor local, caso tenha o banco de dados remoto informe o endereço do servidor. Na opção porta, o valor “5432” corresponde a porta padrão do SGDB PostgreSQL, caso tenha sido alterada informe o valor. Informe o seu usuário e senha do banco de dados nos campos correspondentes. Após clique em “Testar conexão”, e caso os valores informados estejam corretos será exibida a mensagem “Conexão Bem-sucedida” (Figura 47). Clique no próximo, e na tela que será exibida selecione o esquema *public*, a após clique em finalizar. A tela da Figura 48 mostra a conexão registrada, o que permite por exemplo, executar comandos SQL no banco de dados dentro da IDE.

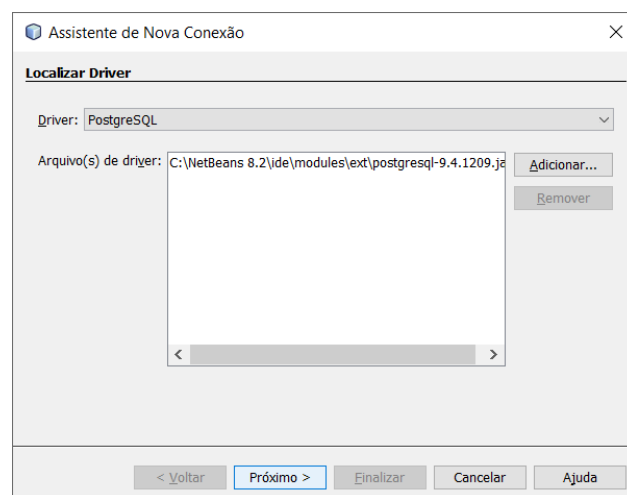


Figura 46 – Registrando a conexão no Netbeans - Primeira tela do assistente

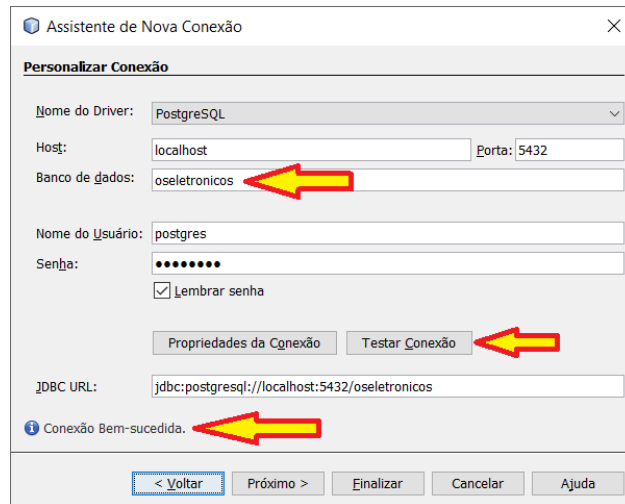


Figura 47 – Registrando a conexão no Netbeans - Segunda tela do assistente

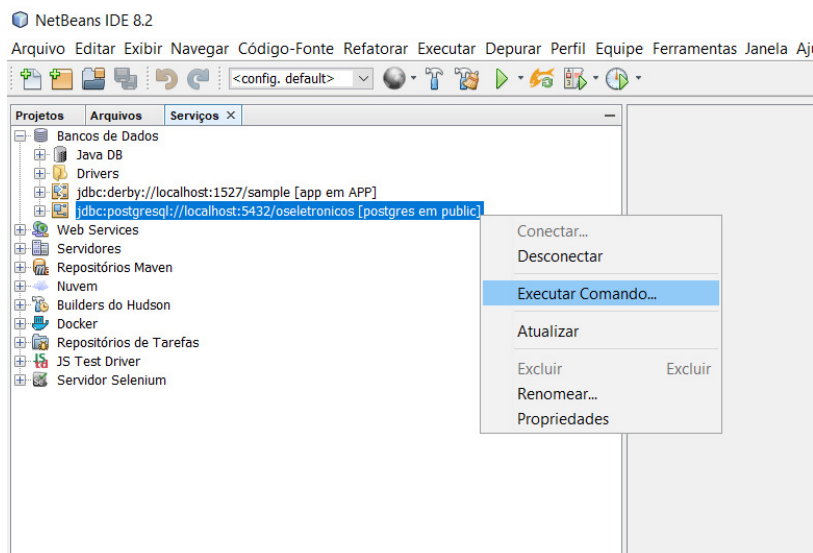


Figura 48 – Conexão com o banco de dados registrada no Netbeans

Com a conexão registrada, é possível criar a unidade de persistência. Acesse o menu arquivo, depois a opção Novo arquivo. Na tela que se abrirá (Figura 49) selecione a categoria Persistência, e em tipo de arquivo selecione Unidade de Persistência e clique no botão próximo. Será exibida a tela da Figura 50, e nela na opção biblioteca de persistência selecione “Hibernate (JPA 2.1)”, e na conexão com o banco de dados selecione a conexão que foi registrada. O nome da unidade de persistência pode ser deixado o padrão, bem como a opção de estratégia de geração de tabela. Clique em finalizar.

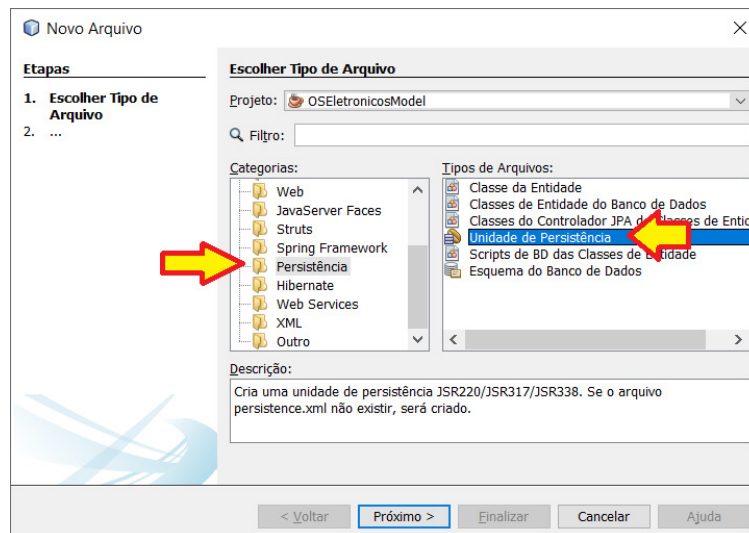


Figura 49 – Criando a unidade de persistência - Tela 01

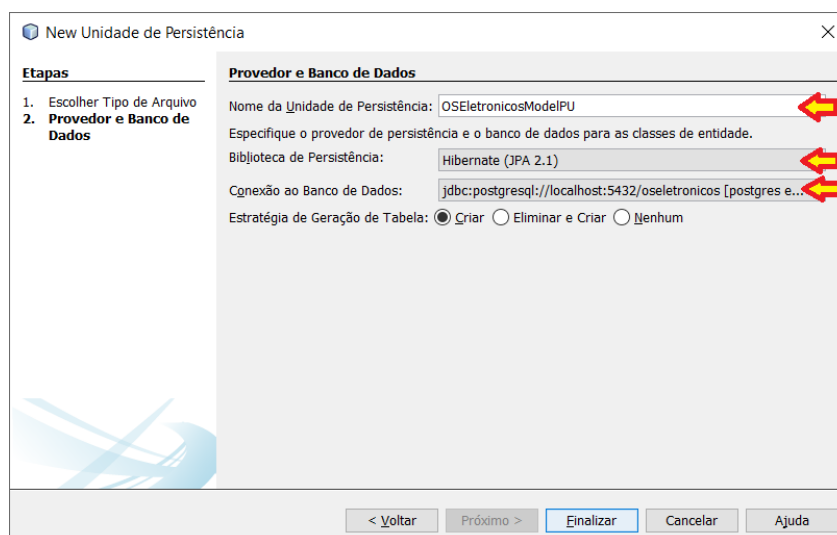


Figura 50 – Criando a unidade de persistência - Tela 02

Após a criação da unidade de persistência, o Netbeans pode adicionar mais uma biblioteca do Hibernate, como exibido na [Figura 51](#). Caso isso tenha acontecido, remova a biblioteca adicionada.

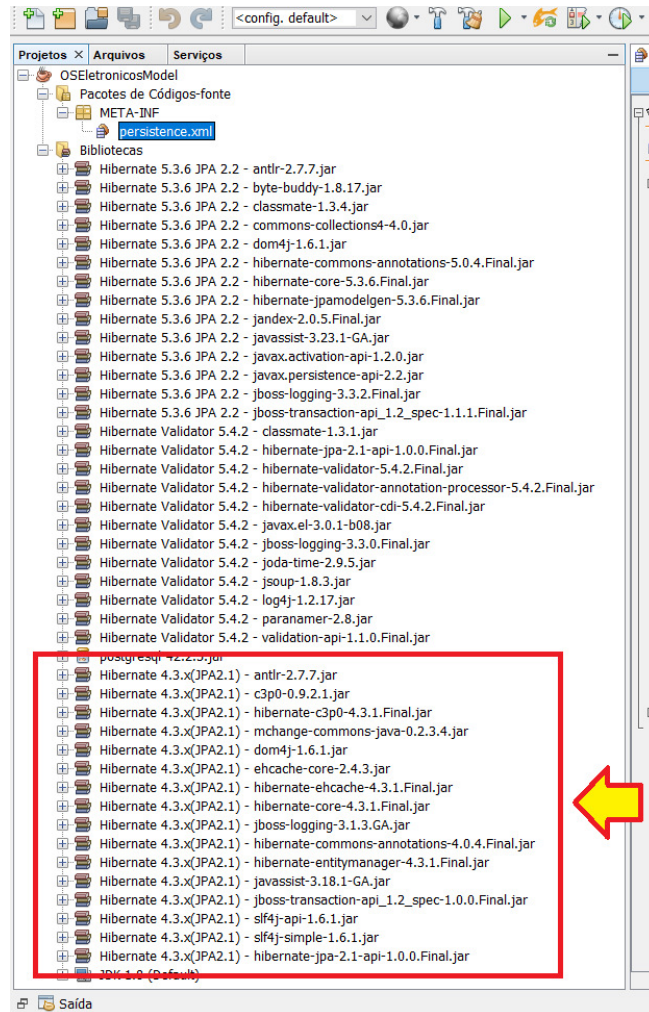


Figura 51 – Biblioteca adiciona pela Netbeans a ser removida

Abra o arquivo “persistence.xml”, e clique na opção código fonte. O **Código Fonte 2.20** será exibido, e nele já necessários alguns ajustes. O primeiro é remover o conteúdo da linha 11, que contém a propriedade “javax.persistence.schema-generation.database.action”. Ele será substituído por uma propriedade do Hibernate.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
3   <persistence-unit name="OSEletronicosModelPU" transaction-type="RESOURCE_LOCAL">
4     <provider>org.hibernate.ejb.HibernatePersistence</provider>
5     <properties>
6       <property name="javax.persistence.jdbc.url"
7         value="jdbc:postgresql://localhost:5432/oseletronicos"/>
8       <property name="javax.persistence.jdbc.user" value="postgres"/>
9       <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
10      <property name="javax.persistence.jdbc.password" value="postgres"/>
11      <property name="hibernate.cache.provider_class"
12        value="org.hibernate.cache.NoCacheProvider"/>
13      <property name="javax.persistence.schema-generation.database.action" value="create"/>

```

```

12     </properties>
13     </persistence-unit>
14 </persistence>

```

Código Fonte 2.20 – Arquivo persistence.xml criado pela IDE

O [Código Fonte 2.21](#) exibe o arquivo “persistence.xml” com as alterações necessárias. A primeira é no elemento provider, que deve ser alterado para “org.hibernate.jpa.HibernatePersistenceProvider” que é o nome da classe utilizada pela versão 5.3.6 do Hibernate. A alteração pode ser visualizada no [Código Fonte 2.21](#) linha 11. Adicione no arquivo as propriedades que se encontram entre as linhas 11 e 15, que são o dialeto do PostgreSQL, opção para não realizar o autocommit, a opção “hibernate.hbm2ddl.auto” com o valor “update” para criar e atualizar as tabelas do banco de dados, e as opções para exibir e formatar o SQL gerado pelo hibernate no console. Com este arquivo configurado, já pode-se iniciar o mapeamento das classes do Estudo de caso.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
3     <persistence-unit name="OSEletronicosModelPU" transaction-type="RESOURCE_LOCAL">
4         <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
5         <properties>
6             <property name="javax.persistence.jdbc.url"
7                 value="jdbc:postgresql://localhost:5432/oseletronicos"/>
8             <property name="javax.persistence.jdbc.user" value="postgres"/>
9             <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
10            <property name="javax.persistence.jdbc.password" value="postgres"/>
11            <property name="hibernate.cache.provider_class"
12                value="org.hibernate.cache.NoCacheProvider"/>
13            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
14            <property name="hibernate.connection.autocommit" value="false"/>
15            <property name="hibernate.hbm2ddl.auto" value="update"/>
16            <property name="hibernate.show_sql" value="false"/>
17            <property name="hibernate.format_sql" value="false"/>
18        </properties>
19    </persistence-unit>
20 </persistence>

```

Código Fonte 2.21 – Arquivo persistence.xml modificado

2.6.2 Mapeamento de classe simples

Com o projeto devidamente configurado, o banco de dados criado, e a unidade de persistência criada e modificada com os parâmetros adequados, pode-se iniciar o mapeamento objeto-relacional do estudo de caso. A primeira classe a ser criada será a classe Estado, por não possuir relacionamentos. A classe pode ser visualizada em seu diagrama da UML na [Figura 52](#).

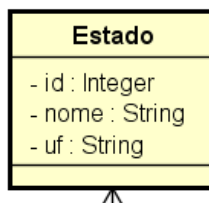


Figura 52 – Classe Estado

Para criar a classe, acesse o menu arquivo, depois a opção novo Arquivo. Na tela que se abrirá, em categorias selecione Java, e em tipo de arquivo selecione classe Java. Na tela seguinte (Figura 53) informe no nome da classe o valor “Estado”, e no pacote digite “br.edu.ifsul.modelo”. Após clique em finalizar.

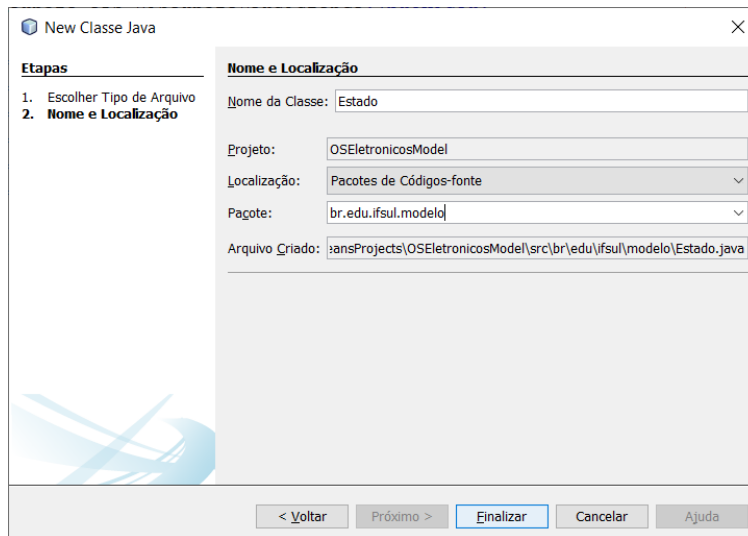


Figura 53 – Tela de criação da Classe Estado

Após a criação do classe, os seus atributos devem ser adicionados, conforme o especificado no diagrama da Figura 53. O código fonte da classe com os atributos pode ser visualizado no Código Fonte 2.22.

```
1 package br.edu.ifsul.modelo;
2
3 public class Estado {
4
5     Integer id;
6     String nome;
7     String uf;
8
9 }
```

Código Fonte 2.22 – Classe Estado sem o padrão Java Beans

Todas as classes persistentes deve adotar o padrão Java Beans, que é utilizado nas especificações da plataforma Java EE, e na maioria dos seus *frameworks*. Para ser considerado um Java bean uma classe deve ter 3 características:

- Implementar a interface *Serializable*;
- Possuir um construtor sem argumentos;
- Encapsular os atributos com a visibilidade *private* e possuir os métodos assessores (*getters* e *setters*) públicos;

Para implementar a classe *Serializable*, basta adicionar a expressão “*implements Serializable*” na declaração da classe. Já o método construtor sem argumentos consiste em um método publico com o mesmo nome da classe. Na IDE Netbens, para encapsular os campos, clique com o botão direito do mouse na classe, e vá até a opção Refatorar -> Encapsular campos (Figura 54). Na tela que se abrirá (Figura 55) clique no botão “Selecionar tudo” para selecionar todos os campos, e depois acione o botão “Refatorar”.

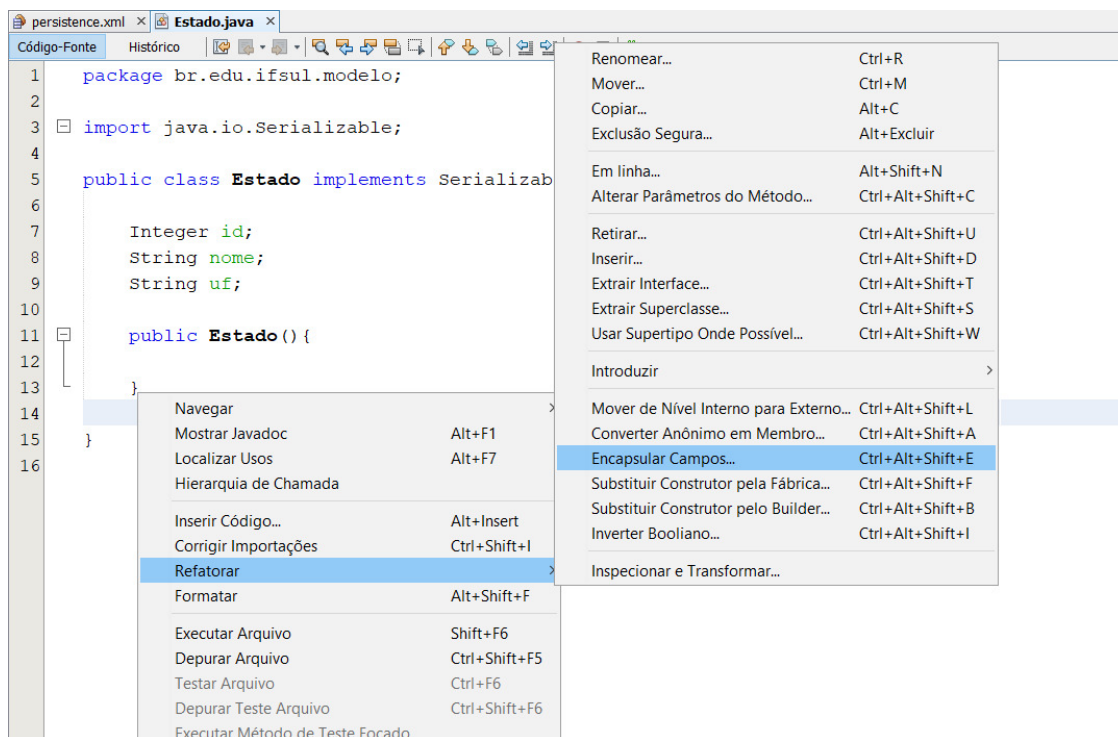


Figura 54 – Refatorar campos

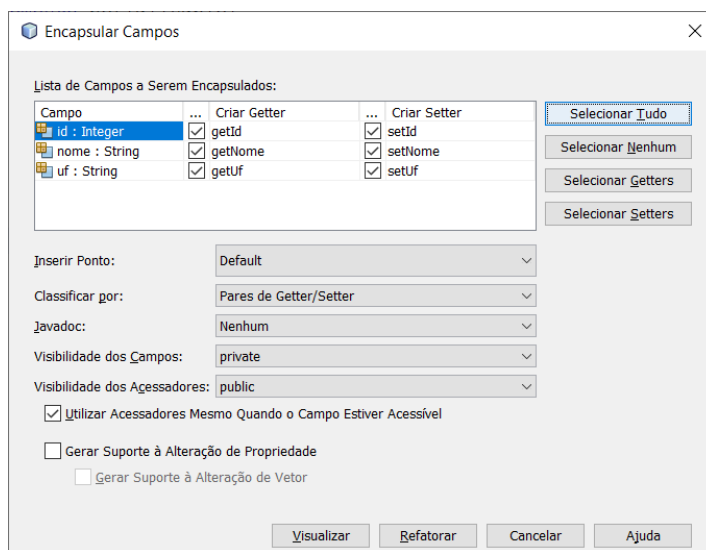


Figura 55 – Selecionando campos para refatorar

Além de implementar o padrão Java Bean, todas as classes persistentes devem ter os métodos equals e hashCode sobrescritos, para que a comparação entre objetos da mesma classe seja realizada comparando-se o atributo que será a chave primária no banco de dados. Isso possibilitará o correto funcionamento de conversores e outras funcionalidades nos sistemas, como a seleção de um objeto em uma lista. Para realizar isso, clique com o botão direito no mouse na classe, e vá na opção “Inserir código”, e depois selecione a opção “equal() e hashCode()”. Na tela que será exibida, selecione o campo “id” nos campos incluídos em equals e em hashCode, conforme ilustra a Figura 56, e clique em gerar.

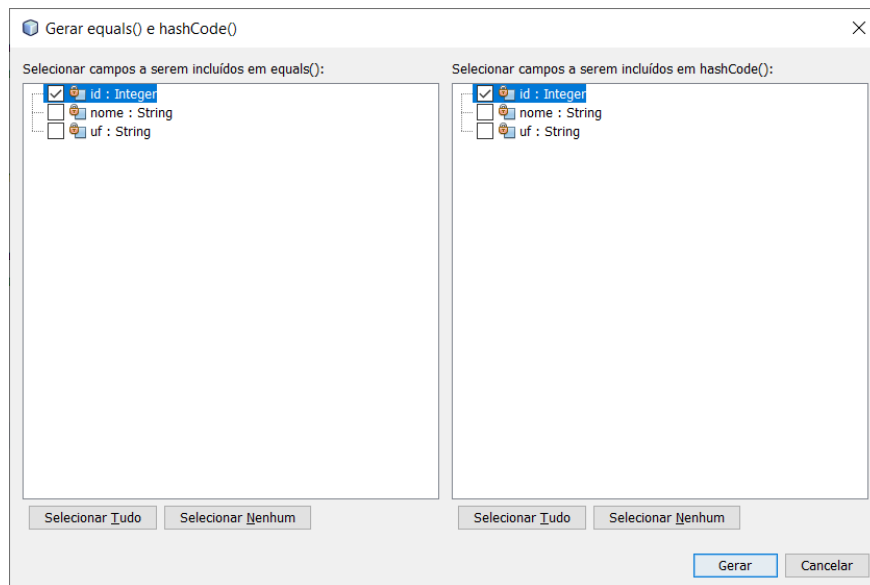


Figura 56 – Selecionando campos para refatorar

O fonte da classe Estado com o padrão Java Beans e os métodos equals e hashCode sobrescritos pode ser visualizado no [Código Fonte 2.23](#). Agora é necessário adicionar os mapeamentos da JPA e as anotações para validação dos dados.

```
1 package br.edu.ifsul.modelo;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 public class Estado implements Serializable{
7
8     private Integer id;
9     private String nome;
10    private String uf;
11
12    public Estado(){
13
14    }
15
16    public Integer getId() {
17        return id;
18    }
19
20    public void setId(Integer id) {
21        this.id = id;
22    }
23
24    public String getNome() {
25        return nome;
26    }
27
28    public void setNome(String nome) {
29        this.nome = nome;
30    }
31 }
```

```
32     public String getUf() {
33         return uf;
34     }
35
36     public void setUf(String uf) {
37         this.uf = uf;
38     }
39
40     @Override
41     public int hashCode() {
42         int hash = 7;
43         hash = 83 * hash + Objects.hashCode(this.id);
44         return hash;
45     }
46
47     @Override
48     public boolean equals(Object obj) {
49         if (this == obj) {
50             return true;
51         }
52         if (obj == null) {
53             return false;
54         }
55         if (getClass() != obj.getClass()) {
56             return false;
57         }
58         final Estado other = (Estado) obj;
59         if (!Objects.equals(this.id, other.id)) {
60             return false;
61         }
62         return true;
63     }
64 }
65 }
```

Código Fonte 2.23 – Classe Estado com o padrão Java Beans e método Equals sobrescrito

As principais anotações para o mapeamento objeto-relacional são descritas na [seção 2.4](#). As anotações necessárias para realizar o mapeamento da classe Estado, bem como validar os seus atributos pode ser visualizado no [Código Fonte 2.24](#). Na linha 17, é utilizada a anotação `@Entity` para definir que a classe é uma entidade da JPA, e na linha 18 com a anotação `@Table` se define que a classe será armazenada em uma tabela chamada “estado”. A definição do atributo que será a chave primaria na tabela ocorre na linha 21, com o uso na anotação `@Id`, e nas linhas 22 e 23 define-se a forma que será gerado o valor do atributo id, neste caso usando-se uma sequência no banco de dados. O mapeamento do restante dos atributos é realizado com a anotação `@Column`, como pode-se visualizar nas linhas 29 e 35, para os atributos “nome” e “uf” respectivamente. Nele se definem o nome da coluna no banco de dados, o seu tamanho e se irá permitir valores nulos. Para a validação dos atributos, para o nome e uf, que são strings, serão utilizadas as anotações `@NotNull` (para não permitir valores nulos), `@NotBlank` (para não permitir strings em branco) e `@Length` (para não deixar o tamanho da string exceder o limite definido). Elas devem ser adicionadas antes dos atributos.

```
1 package br.edu.ifsul.modelo;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5 import javax.persistence.Column;
6 import javax.persistence.Entity;
```

```
7 import javax.persistence.GeneratedValue;
8 import javax.persistence.GenerationType;
9 import javax.persistence.Id;
10 import javax.persistence.SequenceGenerator;
11 import javax.persistence.Table;
12 import javax.validation.constraints.NotNull;
13 import org.hibernate.validator.constraints.Length;
14 import org.hibernate.validator.constraints.NotBlank;
15
16
17 @Entity
18 @Table(name = "estado")
19 public class Estado implements Serializable {
20
21     @Id
22     @SequenceGenerator(name = "seq_estado", sequenceName = "seq_estado_id", allocationSize = 1)
23     @GeneratedValue(generator = "seq_estado", strategy = GenerationType.SEQUENCE)
24     private Integer id;
25
26     @NotNull(message = "O nome não pode ser nulo")
27     @NotBlank(message = "O nome não pode ser em branco")
28     @Length(max = 50, message = "O nome não pode ter mais que {max} caracteres")
29     @Column(name = "nome", length = 50, nullable = false)
30     private String nome;
31
32     @NotNull(message = "A UF não pode ser nula")
33     @NotBlank(message = "A UF não pode ser em branco")
34     @Length(max = 2, min = 2, message = "A UF deve ter {max} caracteres")
35     @Column(name = "uf", length = 2, nullable = false)
36     private String uf;
37
38     public Estado(){
39
40     }
41
42     /*
43     Métodos get, set, equals e hashCode omitidos
44     */
45
46 }
```

Código Fonte 2.24 – Classe Estado com as anotações da JPA e validações

2.6.3 Realizando operações de persistência

Após a configuração da unidade de persistência, e os mapeamentos com anotações realizados na classe Estado, é possível realizar operações de persistência com objetos da classe. Primeiro se instancia um objeto do tipo *EntityManagerFactory*, que é criado pela classe *Persistence*, por meio do método *createEntityManagerFactory*, que recebe por parâmetro o nome da unidade de persistência utilizada. Após é necessário uma instância de uma *EntityManager*. Depois inicia-se uma transação com o banco de dados usando a *EntityManager*, cria-se o objeto desejado, e executa-se a operação de persistência com a *EntityManager*. Por fim, para efetivar a gravação no banco de dados, realiza-se um *commit* da transação com a *EntityManager*. Os passos necessários são descritos abaixo:

1. Instanciar uma *EntityManagerFactory*;

2. Instanciar uma *EntityManager*;
3. Iniciar uma transação com o banco de dados;
4. Criar o objeto desejado;
5. Realizar uma operação de persistência;
6. Realizar um *commit* na transação com o banco de dados;
7. Chamar o método *close()* da *EntityManager* e da *EntityManagerFactory*;

Nas próximas seções serão criados exemplos de como realizar as operações mais comuns de persistência, com o código fonte completo para cada operação.

2.6.3.1 Persistindo um objeto

O [Código Fonte 2.25](#) exibe um exemplo completo de como persistir um objeto, e trata-se de uma classe Java principal (que possui o método *main*), que pode ser criada acessando o menu “Arquivo”, opção “Novo Arquivo”. Depois seleciona-se categoria “Java”, e em tipo de arquivo “Classe Java Principal”.

Para tornar possível utilizar a JPA na classe criada, nas linhas 13 e 14 do [Código Fonte 2.25](#), é instanciado um objeto do tipo *EntityManagerFactory*, passando-se o valor “OSEletronicosModelPU”⁷ para o método *createEntityManagerFactory* da classe *Persistence*. Com ele possível criar a *EntityManager* na linha 15. Entre as linhas 17 e 19 cria-se um objeto do tipo *Estado*, e informa-se os seus atributos. Já na linha 21, se inicia uma transação com o banco de dados. A classe *EntityManager* possui um método chamado *persist*, que recebe por parâmetro o objeto que se deseja persistir. Desta forma na linha 23 chama-se a persistência do objeto criado. Por fim na linha 25 realiza-se um *commit* da transação, e nas linhas 27 e 28 os objetos *em* e *emf* são fechados, para não ficarem carregados em memória.

```
1 package br.edu.ifsul.testes;
2
3 import br.edu.ifsul.modelo.Estado;
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7
8
9 public class TestePersistirEstado {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf =
14             Persistence.createEntityManagerFactory("OSEletronicosModelPU");
15         EntityManager em = emf.createEntityManager();
16
17         Estado e = new Estado();
18         e.setNome("Rio Grande do Sul");
19         e.setUf("RS");
20
21         em.getTransaction().begin();
22
23         em.persist(e);
24
25         em.getTransaction().commit();
26
27         em.close();
28         emf.close();
```

⁷ O valor é o nome da unidade de persistência, existente no arquivo *persistence.xml* ([Código Fonte 2.21](#))


```
29     }
30
31 }
```

Código Fonte 2.25 – Persistindo um objeto

2.6.3.2 Carregando um objeto

É possível recuperar um objeto do banco de dados, usando-se o método *find* da classe *EntityManager*. Ele recebe dois parâmetros: o primeiro é o nome da classe persistente do objeto que será recuperado, e o segundo é o valor da chave primária do objeto existente no banco de dados. Na linha 17 do [Código Fonte 2.26](#) executa-se o método *find*, onde se informa o valor “Estado.class” no primeiro parâmetro, e o valor “1” no segundo parâmetro, que é o valor de uma chave primária existente na tabela estado. E na linha 19 os dados do objeto carregado são exibidos no console.

```
1 package br.edu.ifsul.testes;
2
3 import br.edu.ifsul.modelo.Estado;
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7
8
9 public class TesteCarregarEstado {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf =
14             Persistence.createEntityManagerFactory("OSEletronicosModelPU");
15         EntityManager em = emf.createEntityManager();
16
17         Estado e = em.find(Estado.class, 1);
18
19         System.out.println("ID: " + e.getId() + " Nome: " + e.getNome() + " UF: " + e.getUf());
20
21         em.close();
22         emf.close();
23     }
24
25 }
```

Código Fonte 2.26 – Carregando um objeto

2.6.3.3 Atualizando um objeto

Para se atualizar um objeto, primeiro é necessário carregar um objeto do banco de dados, processo descrito na [subseção 2.6.3.2](#). Depois de modificados os valores dos atributos dos objetos ([Código Fonte 2.27](#) linhas 18 e 19), chama-se o método *merge* da classe *EntityManager* ([Código Fonte 2.27](#) linha 23), passando-se o objeto a ser modificado no banco de dados. A operação de persistência deve ser realizada com o início e *commit* de uma transação.

```
1 package br.edu.ifsul.testes;
2
3 import br.edu.ifsul.modelo.Estado;
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
```

```
7
8
9 public class TesteAtualizarEstado {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf =
14             Persistence.createEntityManagerFactory("OSEletronicosModelPU");
15         EntityManager em = emf.createEntityManager();
16
17         Estado e = em.find(Estado.class, 2);
18         e.setNome("Santa Catarina");
19         e.setUf("SC");
20
21         em.getTransaction().begin();
22
23         em.merge(e);
24
25         em.getTransaction().commit();
26
27         em.close();
28         emf.close();
29     }
30
31 }
```

Código Fonte 2.27 – Atualizando um objeto

2.6.3.4 Removendo um objeto

O [Código Fonte 2.28](#) trata da remoção de um objeto do banco de dados. Primeiro é necessário carregar o objeto que deseja ser removido, e isto é feito na linha 17, onde se carrega o objeto de código “2”. Depois de se iniciar a transação, chama-se o método *remove* na linha 21, passando-se o objeto carregado. Após se realiza o *commit* da transação para efetivar a gravação no banco de dados.

```
1 package br.edu.ifsul.testes;
2
3 import br.edu.ifsul.modelo.Estado;
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7
8
9 public class TesteRemoverEstado {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf =
14             Persistence.createEntityManagerFactory("OSEletronicosModelPU");
15         EntityManager em = emf.createEntityManager();
16
17         Estado e = em.find(Estado.class, 2);
18
19         em.getTransaction().begin();
20
21         em.remove(e);
22

```

```
23     em.getTransaction().commit();
24
25     em.close();
26     emf.close();
27 }
28
29 }
```

Código Fonte 2.28 – Removendo um objeto

2.6.3.5 Realizando uma consulta com a JPQL

Para se realizar consultas com a JPA, pode-se utilizar uma linguagem de consulta a objetos, chamada JPQL (Java Persistence Query Language). Nela, na cláusula *from*, informa-se o nome da classe persistente, e não da tabela do banco de dados. Além disso, é possível usar a cláusula *where* e *order by*.

O [Código Fonte 2.29](#) realiza uma consulta no banco de dados, para a classe “Estado”. A consulta é realizada na linha 18, onde se informa o valor “from Estado” para o método *createQuery* da *EntityManager*, e depois executa-se o método *getResultList()* para recuperar os objetos resultantes da consulta. A JPA é responsável por transformar este código em SQL para executá-lo no banco de dados. Neste caso, a consulta “from Estado” resulta em um SQL “select * from estado”;

```
1 package br.edu.ifsul.testes;
2
3 import br.edu.ifsul.modelo.Estado;
4 import java.util.List;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8
9
10 public class TesteListarEstado {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf =
15             Persistence.createEntityManagerFactory("OSEletronicosModelPU");
16         EntityManager em = emf.createEntityManager();
17
18         List<Estado> lista = em.createQuery("from Estado").getResultList();
19
20         for (Estado e : lista){
21             System.out.println("ID: " + e.getId() + " Nome: "
22                 + e.getNome() + " UF: " + e.getUf());
23         }
24
25         em.close();
26         emf.close();
27     }
28
29 }
```

Código Fonte 2.29 – Consulta com JPQL

Layout responsivo com Java Server Faces

Este capítulo trata do desenvolvimento de interfaces de sistemas web com layout responsivo utilizando o framework Java Server Faces e suas bibliotecas de componentes Primefaces e BootsFaces. Será apresentado como prova de conceito o desenvolvimento de uma aplicação web que realiza uma manutenção CRUD para uma classe.

Entre os pontos abordados estão o desenvolvimento de uma classe de modelo, do controlador do Java Server Faces, templates para a criação do layout e das telas da manutenção CRUD, como a tela de listagem dos registros e a tela de edição.

3.1 Bibliotecas necessárias

As bibliotecas abaixo listadas serão utilizadas no projeto, sendo necessário a obtenção das mesmas. Elas também devem ser adicionadas ao *classpath* do projeto.

- *Java Server Faces* implementação Mojarra versão 2.2.13: <<https://maven.java.net/content/repositories/releases/org/glassfish/javax.faces/2.2.13/javax.faces-2.2.13.jar>>
- *Primefaces* versão 6.0: <<http://search.maven.org/remotecontent?filepath=org/primefaces/primefaces/6.0/primefaces-6.0.jar>>
- Temas do *Primefaces*: <<http://repository.primefaces.org/org/primefaces/themes/all-themes/1.0.10/all-themes-1.0.10.jar>>
- *BootsFaces* versão 0.9.1 (Tema do *Bootstrap*): <<http://www.bootsfaces.net/download/BootsFaces-OSP-0.9.1-dist-default.jar>>

3.2 Desenvolvimento da aplicação

Nesta seção serão apresentados os componentes de software da aplicação.

3.2.1 Camada de modelo

A camada de modelo possui somente uma classe, que é a classe *Pessoa*, exibida no código fonte 3.1. A classe possui 4 atributos e utiliza o padrão *Java Beans*.

```
1 package br.edu.ifsul.modelo;  
2
```

```
3 import java.io.Serializable;
4 import java.util.Calendar;
5 import java.util.Objects;
6
7 /**
8 *
9 * @author Jorge Luis Boeira Bavaresco
10 * @email jorge.bavaresco@passofundo.ifsul.edu.br
11 * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
12 */
13 public class Pessoa implements Serializable {
14     private Integer id;
15     private String nome;
16     private String email;
17     private Calendar nascimento;
18
19     public Pessoa() {
20     }
21
22     public Pessoa(Integer id, String nome, String email, Calendar nascimento) {
23         this.id = id;
24         this.nome = nome;
25         this.email = email;
26         this.nascimento = nascimento;
27     }
28
29     @Override
30     public int hashCode() {
31         int hash = 5;
32         hash = 53 * hash + Objects.hashCode(this.id);
33         return hash;
34     }
35
36     @Override
37     public boolean equals(Object obj) {
38         if (this == obj) {
39             return true;
40         }
41         if (obj == null) {
42             return false;
43         }
44         if (getClass() != obj.getClass()) {
45             return false;
46         }
47         final Pessoa other = (Pessoa) obj;
48         if (!Objects.equals(this.id, other.id)) {
49             return false;
50         }
51         return true;
52     }
53
54     // Metodos Get e Set omitidos
55 }
```

Código Fonte 3.1 – Classe Pessoa

3.2.2 Camada de controle

A camada de controle possui os ManagedBeans do JSF, e classes que serão utilizadas pela interface.

A classe ControlePessoa é um *managed bean* do JSF e é responsável por responder as ações do usuário e armazenar a lista de pessoas na sessão. A classe pode ser visualizada no código fonte 3.2. Ela também implementa o padrão *Java Beans*. O atributo lista armazena as pessoas em memória. Já o atributo objeto, que é do tipo Pessoa armazena o objeto que está sendo editado no momento. E o atributo editando é utilizado pela interface para determinar se exibe a listagem das pessoas ou o formulário de edição, conforme a ação que está sendo executada.

```
1
2 package br.edu.ifsul.controle;
3
4 import br.edu.ifsul.modelo.Pessoa;
5 import java.io.Serializable;
6 import java.util.ArrayList;
7 import java.util.Calendar;
8 import java.util.List;
9 import javax.faces.bean.ManagedBean;
10 import javax.faces.bean.SessionScoped;
11
12 /**
13 *
14 * @author Jorge Luis Boeira Bavaresco
15 * @email jorge.bavaresco@passofundo.ifsul.edu.br
16 * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
17 */
18 @ManagedBean(name = "controlePessoa")
19 @SessionScoped
20 public class ControlePessoa implements Serializable {
21     private List<Pessoa> lista = new ArrayList<>();
22     private Pessoa objeto;
23     private Boolean editando;
24
25     public ControlePessoa() {
26         lista.add(new Pessoa(1, "jorge", "jorge.bavaresco@passofundo.ifsul.edu.br", Calendar.
27             getInstance()));
28         lista.add(new Pessoa(2, "Joao", "joao@gmail.com", Calendar.getInstance()));
29         editando = false;
30     }
31
32     public String listar(){
33         editando = false;
34         return "/pessoa/listar?faces-redirect=true";
35     }
36
37     public void novo(){
38         objeto = new Pessoa();
39         editando = true;
40     }
41
42     public void alterar(Pessoa obj){
43         objeto = obj;
44         editando = true;
45     }
46
47     public void excluir(Pessoa obj){
```

```
47     lista.remove(obj);
48 }
49
50 public void salvar(){
51     if (objeto.getId() == null){
52         objeto.setId(lista.size()+1);
53         lista.add(objeto);
54     }
55     editando = false;
56 }
57
58 // Metodos Get e Set omitidos
59 }
```

Código Fonte 3.2 – Classe ControlePessoa

A classe ConverterCalendar (código fonte 3.3) é um conversor do JSF, e converte a data que vem da interface do usuário como String em um objeto da classe Calendar.

```
1
2 package br.edu.ifsul.converters;
3
4 import java.io.Serializable;
5 import java.text.SimpleDateFormat;
6 import java.util.Calendar;
7 import javax.faces.component.UIComponent;
8 import javax.faces.context.FacesContext;
9 import javax.faces.convert.Converter;
10 import javax.faces.convert.FacesConverter;
11
12 /**
13 *
14 * @author Jorge Luis Boeira Bavaresco
15 * @email jorge.bavaresco@passofundo.ifsul.edu.br
16 * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
17 */
18 @FacesConverter(value = "converterCalendar")
19 public class ConverterCalendar implements Serializable, Converter {
20
21     SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
22
23     // Converte da tela para um objeto
24     @Override
25     public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
26         if (string == null){
27             return null;
28         }
29         Calendar retorno = Calendar.getInstance();
30         try {
31             retorno.setTime(sdf.parse(string));
32             return retorno;
33         } catch (Exception e){
34             return null;
35         }
36     }
37
38     // Converter do objeto para a tela
39     @Override
```

```
40 public String getAsString(FacesContext fc, UIComponent uic, Object o) {
41     Calendar obj = (Calendar) o;
42     return sdf.format(obj.getTime());
43 }
44
45 }
```

Código Fonte 3.3 – Classe ConverterCalendar

3.2.3 Configurações dos arquivos web.xml e faces-config.xml

Nesta seção são apresentados os conteúdos dos arquivos web.xml e faces-config.xml. Ambos se encontram dentro do diretório de páginas web, e nele dentro do diretório WEB-INF.

O código fonte 3.4 apresenta o conteúdo do arquivo web.xml. Entre as linhas 3 e 6 está sendo definido um parâmetro de contexto para a utilização do tema do Primefaces. No caso será utilizado o tema do bootstrap, mesmo tema utilizado pela biblioteca do BootsFaces. Outro ajuste que deve ser realizado é no padrão de URL do servlet do JSF (linha 18), para que interprete as tags do JSF em arquivos xhtml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
3     <context-param>
4         <param-name>primefaces.THEME</param-name>
5         <param-value>bootstrap</param-value>
6     </context-param>
7     <context-param>
8         <param-name>javax.faces.PROJECT_STAGE</param-name>
9         <param-value>Development</param-value>
10    </context-param>
11    <servlet>
12        <servlet-name>Faces Servlet</servlet-name>
13        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
14        <load-on-startup>1</load-on-startup>
15    </servlet>
16    <servlet-mapping>
17        <servlet-name>Faces Servlet</servlet-name>
18        <url-pattern>*.xhtml</url-pattern>
19    </servlet-mapping>
20    <session-config>
21        <session-timeout>
22            30
23        </session-timeout>
24    </session-config>
25    <welcome-file-list>
26        <welcome-file>index.xhtml</welcome-file>
27    </welcome-file-list>
28 </web-app>
```

Código Fonte 3.4 – Conteúdo do arquivo web.xml

No código fonte 3.5 é apresentado o código fonte do arquivo faces-config.xml. Nele está sendo definido um caso de navegação explícito, para o arquivo index.xhtml, com o nome de outcome chamado inicio. Ele é necessário para a utilização em componentes do BootsFaces que não possuem a chamada de ações.


```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <faces-config version="2.2"
3 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
6   <application>
7     <navigation-rule>
8       <from-view-id>*/</from-view-id>
9       <navigation-case>
10        <from-outcome>inicio</from-outcome>
11        <to-view-id>/index.xhtml</to-view-id>
12      </navigation-case>
13    </navigation-rule>
14  </application>
15 </faces-config>

```

Código Fonte 3.5 – Conteúdo do arquivo faces-config.xml

3.2.4 Template

Nesta seção será apresentada a criação de um arquivo com o template para ser utilizado pela aplicação. Para a criação do template será utilizado facelets.

No arquivo template.xhtml (código fonte 3.6) é definido um layout responsivo com o bootsfaces. O componente `b:container` organiza os componentes, e os elementos contidos dentro do componente `b:navBar` organizam o menu. Na linha 15 o componente `b:navLink` utiliza o outcome `inicio` definido no arquivo faces-config.xml para ir para a página inicial, sendo necessário a sua utilização desta forma já que o componente não possui uma configuração para a chamada de uma ação de controladores do JSF. Já o componente `b:navCommandLink` nas linhas 17 e 18 utiliza a ação definida no managed bean para realizar a navegação. Ainda temos a definição de regiões editáveis com o componente `ui:insert`, para utilização em páginas que venham a fazer uso do template. A utilização dos componentes do BootsFaces torna o design do template responsivo.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:h="http://xmlns.jcp.org/jsf/html"
5   xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
6   xmlns:b="http://bootsfaces.net/ui">
7   <h:head>
8     <title><ui:insert name="titulo">titulo da pagina</ui:insert></title>
9   </h:head>
10  <h:body>
11    <b:container >
12      <h:form id="formMenu">
13        <b:navBar brand="Gerencial" >
14          <b:navBarLinks>
15            <b:navLink value="Inicio" outcome="inicio"/>
16            <b:dropMenu value="Cadastros" >
17              <b:navCommandLink value="Pessoas" ajax="false"
18                action="#{controlePessoa.listar()}/>
19            </b:dropMenu>
20          </b:navBarLinks>
21        </b:navBar>
22      </h:form>

```

```

23     <ui:insert name="conteudo">
24         Aqui vai o conteudo
25     </ui:insert>
26 </b:container>
27 </h:body>
28 </html>

```

Código Fonte 3.6 – Conteúdo do arquivo template.xhtml

A utilização do template pode ser visualizado no código fonte 3.7. O elemento `ui:composition` define a utilização do template, e o elemento `ui:define` define o conteúdo das regiões editáveis.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
5     <ui:composition template="/templates/template.xhtml">
6         <ui:define name="titulo">Sistema com BootsFaces</ui:define>
7         <ui:define name="conteudo">
8             </ui:define>
9     </ui:composition>
10 </html>

```

Código Fonte 3.7 – Conteúdo do arquivo index.xhtml

3.2.5 Tela dos CRUDS

A telas dos CRUD serão criadas em dois arquivos, dentro do diretório `peessoa`, que fica na raiz das páginas web. Eles são o arquivo `listar.xhtml` (código fonte 3.8) e `formulario.xhtml` (código fonte 3.9).

O código fonte 3.8 é responsável por criar uma tela de listagem. Para a exibição dos dados é utilizado o componente `p:dataTable` do *Primefaces*, e a utilização do atributo `reflow` com o valor `true` na linha 18 é responsável por permitir que o layout do `dataTable` se ajuste a telas com tamanhos diferentes, como pode-se visualizar nas figuras 57 e 58. Na linha 13 podemos visualizar outro detalhe importante, que é a exibição ou não do conteúdo contido dentro do elemento `h:panelGroup`, por meio do atributo `rendered` com o valor `#{!controlePessoa.editando}`. Desta forma se controla com o atributo `editando` do controlador `controlePessoa` a exibição ou não dos elementos conforme se está editando uma pessoa. Apesar do layout estar dividido em dois arquivos, está sendo realizado o include do formulário na linha 47.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
5     xmlns:b="http://bootsfaces.net/ui"
6     xmlns:p="http://primefaces.org/ui"
7     xmlns:h="http://xmlns.jcp.org/jsf/html"
8     xmlns:f="http://xmlns.jcp.org/jsf/core">
9     <ui:composition template="/templates/template.xhtml">
10         <ui:define name="titulo">Manutenção de pessoas</ui:define>
11         <ui:define name="conteudo">
12             <h:form id="formListagem" >
13                 <h:panelGroup rendered="#{!controlePessoa.editando}">
14                     <p:messages/>
15                     <p:commandButton value="Novo" icon="ui-icon-plus"
16                                     actionListener="#{controlePessoa.novo()}"
17                                     update="formEdicao formListagem" />

```

```

18     <p:dataTable value="#{controlePessoa.lista}" var="obj" reflow="true"
19         paginatorTemplate="{CurrentPageReport} {FirstPageLink} {PreviousPageLink}
20             {PageLinks} {NextPageLink} {LastPageLink}"
21         paginator="true" rows="5">
22         <p:column headerText="ID">
23             <p:outputLabel value="#{obj.id}"/>
24         </p:column>
25         <p:column headerText="Nome">
26             <p:outputLabel value="#{obj.nome}"/>
27         </p:column>
28         <p:column headerText="Email" >
29             <p:outputLabel value="#{obj.email}"/>
30         </p:column>
31         <p:column headerText="Nascimento" >
32             <p:outputLabel value="#{obj.nascimento}"/>
33             <f:converter converterId="converterCalendar"/>
34         </p:outputLabel>
35         </p:column>
36         <p:column headerText="Ações">
37             <div align="center">
38                 <p:commandButton icon="ui-icon-pencil"
39                     actionListener="#{controlePessoa.alterar(obj)}"
40                     process="@form" update=":formEdicao formListagem"
41                     />
42                 <p:commandButton icon="ui-icon-trash"
43                     actionListener="#{controlePessoa.excluir(obj)}"
44                     process="@form" update=":formListagem" />
45             </div>
46         </p:column>
47     </p:dataTable>
48 </h:panelGroup>
49 </h:form>
50 <ui:include src="/pessoa/formulario.xhtml"/>
51 </ui:define>
52 </ui:composition>
53 </html>

```

Código Fonte 3.8 – Conteúdo do arquivo listar.xhtml





ID	Nome	Email	Nascimento	Ações
1	jorge	jorge.bavaresco@passofundo.ifsul.edu	08/08/2016	 
2	João	joao@gmail.com	08/08/2016	 

Figura 57 – Exibição da listagem em telas grandes

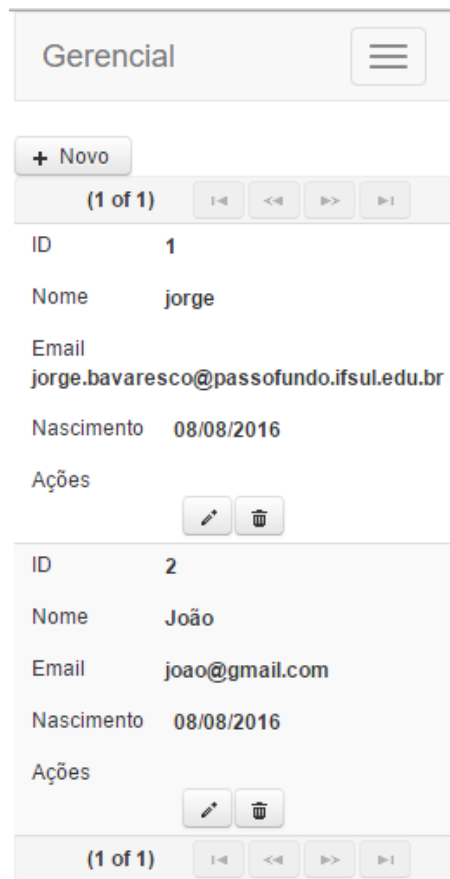


Figura 58 – Exibição da listagem em telas pequenas

No código fonte 3.9 é criado o formulário de edição da pessoa. O que torna o layout responsivo é a utilização da classe *ui-fluid* na *div* da linha 9, juntamente com a definição dos valores para os atributos *columnClasses*, *layout* e *styleClass* para *ui-grid-col-2*, *ui-grid-col-20*, *grid* e *ui-panelgrid-blank* respectivamente, do elemento *p:panelGrid* nas linhas 11 e 12. O resultado pode ser visualizado nas figuras 59 e 60.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:h="http://xmlns.jcp.org/jsf/html"
5     xmlns:p="http://primefaces.org/ui"
6     xmlns:f="http://xmlns.jcp.org/jsf/core">
7 <h:form id="formEdicao">
8     <h:panelGroup rendered="#{controlePessoa.editando}">
9         <div class="ui-fluid">
10             <p:messages/>
11             <p:panelGrid columns="2" columnClasses="ui-grid-col-2,ui-grid-col-20"
12                 layout="grid" styleClass="ui-panelgrid-blank">
13                 <f:facet name="header">
14                     <p:outputLabel value="Edição de Pessoas" />
15                 </f:facet>
16                 <p:outputLabel for="txtID" value="ID" />
17                 <p:inputText id="txtID" value="#{controlePessoa.objeto.id}" size="10" />
18                 <p:outputLabel for="txtNome" value="Nome" />

```

```
19     <p:inputText id="txtNome" value="#{controlePessoa.objeto.nome}"
20               placeholder="Obrigatório" size="40" maxlength="40"
21               required="true"
22               requiredMessage="O Nome deve ser informado" />
23     <p:outputLabel for="txtEmail" value="Email" />
24     <p:inputText id="txtEmail" value="#{controlePessoa.objeto.nome}"
25               placeholder="Obrigatório" size="50" maxlength="50"
26               required="true"
27               requiredMessage="O Email deve ser informado" />
28     <p:outputLabel for="txtNascimento" value="Nascimento" />
29     <p:calendar value="#{controlePessoa.objeto.nascimento}" required="true"
30               requiredMessage="O email deve ser informado"
31               mask="true" pattern="dd/MM/yyyy" id="txtNascimento"
32               placeholder="Obrigatório">
33         <f:converter converterId="converterCalendar" />
34     </p:calendar>
35     <p:commandButton value="Salvar" icon="ui-icon-disk"
36                     actionListener="#{controlePessoa.salvar}"
37                     update="formEdicao formListagem" />
38 </p:panelGrid>
39 </div>
40 </h:panelGroup>
41 </h:form>
</html>
```

Código Fonte 3.9 – Conteúdo do arquivo formulario.xhtml

Gerencial Inicio Cadastros ▾

Edição de Pessoas

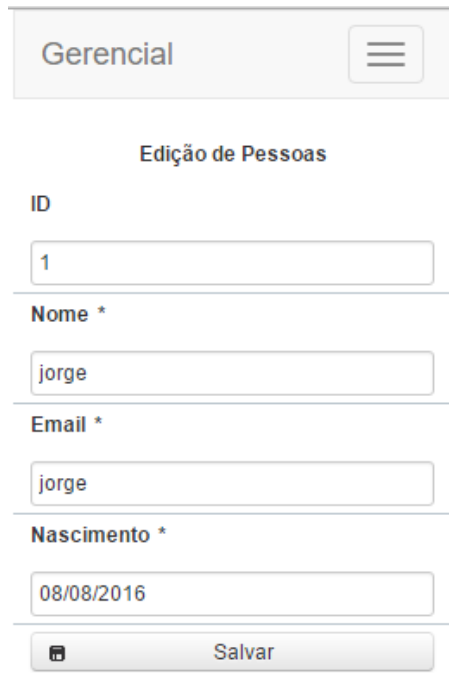
ID

Nome *

Email *

Nascimento *

Figura 59 – Exibição do formulário em telas grandes



The image shows a mobile-optimized web interface. At the top, there is a header bar with the text "Gerencial" on the left and a hamburger menu icon on the right. Below the header, the main content area is titled "Edição de Pessoas". The form consists of several input fields, each with a label above it: "ID" with the value "1", "Nome *" with the value "jorge", "Email *" with the value "jorge", and "Nascimento *" with the value "08/08/2016". The asterisks indicate required fields. At the bottom of the form is a button with a floppy disk icon and the text "Salvar".

Figura 60 – Exibição do formulário em telas pequenas

Composite : Criando seus próprios componentes

O Java Server Faces é um framework baseado em componentes, e possui diversas bibliotecas de componentes, como o Primefaces([PRIMEFACES, 2016](#)), RichFaces([RICHFACES, 2016](#)) e BootsFaces([BOOTSFACES, 2016](#)). Eles herdam as características dos componentes do JSF adicionando funcionalidades. É possível implementar componentes para serem reutilizados, processo que é facilitado a partir da versão 2.0 do JSF que possui uma biblioteca de tags para implementação de componentes compostos, que emprega o prefixo *composite*. Neste capítulo serão apresentados exemplos do seu uso.

4.1 Documentação

O livro Core Java Server Faces ([GEARY; HORSTMANN, 2012](#)) possui o capítulo 9, chamado “Componentes Compostos” que possui uma documentação bem ampla sobre o assunto. Outra fonte de pesquisa é a documentação de composite components da oracle¹ e a documentação de Facelets tag libs².

4.2 Criando seus próprios componentes.

Para a criação dos componentes, primeiro é necessário criar um pasta dentro das paginas web chamada resources, e dentro dela uma pasta que irá abrigar os nossos componentes. Esta pasta também será o prefixo dos componentes. A estrutura pode ser visualizada na figura 61.

¹ <<https://docs.oracle.com/javaee/7/tutorial/jsf-facelets005.htm>>

² <<https://docs.oracle.com/javaee/7/javaserver-faces-2-2/vldocs-facelets/toc.htm>>

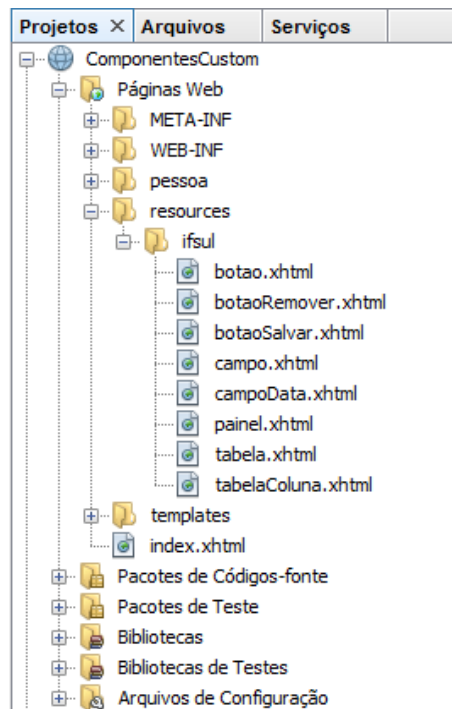


Figura 61 – Pasta dos componentes dentro de resources

Serão criados componentes baseados nos exemplos desenvolvidos no capítulo 3. Iremos criar componentes para facilitar a reutilização e facilitar o desenvolvimento.

O primeiro componente criado é um campo de texto. Primeiro deve ser criado um arquivo chamado `campo.xhtml` dentro de `ifsul`, desta forma o nome do componente será “campo”. Ele pode ser visualizado no código fonte 4.1. São necessário dois elementos principais, que são `cc:interface` e `cc:implementation`. Em `cc:interface` são definidos que atributos o elemento terá, que podem ser valor como os textos exibidos no componentes, ou ações que serão executadas. No caso este componente irá facilitar a utilização do componente `inputText` do Primefaces. Na linha 6 o atributo `shortDescription` recebe um valor para a descrição do componente que será exibido no JavaDoc da IDE. Já o elemento `cc:attribute` como na linha 7 defino atributos que posteriormente serão utilizados na implementação. Por fim o elemento `cc:implementation` implementa o componente, onde podemos visualizar o componente `inputText` utilizando o atributo `valor` da interface na linha 15, com o texto “`{cc.attrs.valor}`”.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:cc="http://xmlns.jcp.org/jsf/composite"
5     xmlns:p="http://primefaces.org/ui">
6   <cc:interface shortDescription="Campo para usar em formulários">
7     <cc:attribute name="valor" required="true"/>
8     <cc:attribute name="tamanho" required="true"/>
9     <cc:attribute name="obrigatorio" required="true"/>
10    <cc:attribute name="obrigatorioMensagem" required="true"/>
11    <cc:attribute name="placeholder" required="true"/>
12    <cc:attribute name="somenteleitura" required="true"/>
13  </cc:interface>
14  <cc:implementation>
15    <p:inputText value="{cc.attrs.valor}"
16                size="{cc.attrs.tamanho}" maxLength="{cc.attrs.tamanho}"

```



```

17         required="#{cc.attrs.obrigatorio}"
18         requiredMessage="#{cc.attrs.obrigatorioMensagem}"
19         placeholder="#{cc.attrs.placeholder}"/>
20     </cc:implementation>
21 </html>

```

Código Fonte 4.1 – campo.xhtml

A maneira de utilizar o componente criado pode ser visualizada no código fonte 4.2

```

1 <facesul:campo id="txtNome" valor="#{controlePessoa.objeto.nome}" tamanho="40"
2     obrigatorio="true" obrigatorioMensagem="O nome deve ser informado"
3     placeholder="Obrigatorio" somenteLeitura="false"/>

```

Código Fonte 4.2 – Utilização do campo de formulário

O próximo componente a ser criado irá facilitar o uso do painel responsivo do Primefaces, exibido no código fonte 4.3. Como o painel deverá ter um espaço onde podem ser inseridos elementos na sua utilização, deve ser usado o elemento *cc:insertChildren* na linhas 19 e 21, delimitando um espaço para inserção de filhos.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5     xmlns:cc="http://xmlns.jcp.org/jsf/composite"
6     xmlns:p="http://primefaces.org/ui"
7     xmlns:f="http://xmlns.jcp.org/jsf/core">
8     <cc:interface shortDescription="Painel responsivo">
9         <cc:attribute name="colunas" required="true"/>
10        <cc:attribute name="titulo" required="true"/>
11    </cc:interface>
12    <cc:implementation >
13        <div class="ui-fluid">
14            <p:panelGrid columns="#{cc.attrs.colunas}"
15                columnClasses="ui-grid-col-2, ui-grid-col-20"
16                layout="grid" styleClass="ui-panelgrid-blank">
17                <f:facet name="header">
18                    <p:outputLabel value="#{cc.attrs.titulo}"/>
19                </f:facet>
20                <cc:insertChildren>
21                </cc:insertChildren>
22            </p:panelGrid>
23        </div>
24    </cc:implementation>
25 </html>

```

Código Fonte 4.3 – painel.xhtml

O exemplo de utilização do painel pode ser visualizado no código fonte 4.4.

```

1 <facesul:painel colunas="2" titulo="Edição de Pessoas">
2     <p:outputLabel for="txtID" value="ID"/>
3     <p:inputText id="txtID" value="#{controlePessoa.objeto.id}" readOnly="true"
4         size="10"/>
5     <p:outputLabel for="txtNome" value="Nome"/>
6     <facesul:campo id="txtNome" valor="#{controlePessoa.objeto.nome}" tamanho="40"
7         obrigatorio="true" obrigatorioMensagem="O nome deve ser informado"
8         placeholder="Obrigatorio" somenteLeitura="false"/>

```

```
9 </ifsul:painel>
```

Código Fonte 4.4 – Utilização do painel.xhtml

Também podem ser criados componentes que realizam a execução de ações. No código fonte 4.5 podemos visualizar a criação de um botão que realiza a execução de um método void. É necessário definir o tipo de assinatura do método na linha 7, utilizando o atributo *method-signature* com o valor *void action()*. Assim podemos passar para o componente na linha 14 a ação a ser executada. Também se informa o texto a ser exibido no componente, bem como os elementos *process* e *update*.

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:cc="http://xmlns.jcp.org/jsf/composite"
5     xmlns:p="http://primefaces.org/ui">
6     <cc:interface>
7         <cc:attribute name="acao" required="true" method-signature="void action()" />
8         <cc:attribute name="texto" required="true"/>
9         <cc:attribute name="update" required="false"/>
10        <cc:attribute name="process" required="false"/>
11    </cc:interface>
12    <cc:implementation>
13        <p:commandButton value="#{cc.attrs.texto}" icon="ui-icon-disk"
14                        actionListener="#{cc.attrs.acao}"
15                        update="#{cc.attrs.update}"
16                        process="#{cc.attrs.update}"/>
17    </cc:implementation>
18 </html>
```

Código Fonte 4.5 – botaoSalvar.xhtml

A utilização do botão criado pode ser visualizada no código fonte 4.6.

```
1 <ifsul:botaoSalvar acao="#{controlePessoa.salvar()}"
2     texto="Salvar" update="formEdicao :formListagem"/>
```

Código Fonte 4.6 – Utilizando o botaoSalvar.xhtml

Também podem ser criados componentes para executar ações que recebem parâmetros, como por exemplo um botão para excluir, que pode ser visualizado no código fonte 4.7. Basta informar na definição da assinatura do método na linha 7 o valor “void action(java.lang.Object)”. A utilização do componente pode ser visualizada no código fonte 4.8.

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:cc="http://xmlns.jcp.org/jsf/composite"
5     xmlns:p="http://primefaces.org/ui">
6     <cc:interface>
7         <cc:attribute name="acao" required="true" method-signature="void
8             action(java.lang.Object)" />
9         <cc:attribute name="update" required="false"/>
10        <cc:attribute name="process" required="false"/>
11    </cc:interface>
12    <cc:implementation>
13        <p:commandButton icon="ui-icon-trash"
14                        actionListener="#{cc.attrs.acao}"
15                        process="#{cc.attrs.process}" update="#{cc.attrs.update}"/>
```

```
15     </cc:implementation>
16 </html>
```

Código Fonte 4.7 – botaoRemoveer.xhtml

```
1 <if:botaoRemoveer acao="#{controlePessoa.excluir(obj)}"
2     process="@form" update=":formListagem" />
```

Código Fonte 4.8 – Utilizando o botaoRemoveer.xhtml

4.3 Empacotando seus próprios componentes em um arquivo JAR.

Os componentes criados podem ser empacotados em um arquivo JAR, tornando possível a sua distribuição para utilização por outros desenvolvedores.

Para fazer isto, basta realizar os seguintes passos:

1. Com a IDE Netbeans, crie um novo projeto Biblioteca de classe Java da categoria Java;
2. Crie dentro dos pacotes de código fonte uma pasta chamada META-INF, e dentro dela uma pasta chamada resources;
3. Dentro da pasta resources deve ser criada a pasta que irá armazenar os componentes, os quais devem ser criados dentro dela. Seguindo o padrão usado neste capítulo, pode ser criada uma pasta com o nome ifsul;
4. Limpe e construa o projeto. Isso pode ser feito clicando com o botão direito do mouse e selecionando a opção limpar e construir. O arquivo jar contendo os componentes será criado na pasta dist do seu projeto;

Internacionalização de mensagens (i18n)

Neste capítulo será apresentada a internacionalização de mensagens (i18n) utilizando o Java Server Faces. Veremos como internacionalizar as mensagens da interface do usuário, as mensagens da bean validation API e também mensagens geradas dentro de métodos das classes.

5.1 Arquivos com as mensagens de internacionalização

Os arquivos de internacionalização consistem em arquivos de texto simples, com a extensão `.properties`, onde se associa uma chave a um valor. Desta forma o valor de cada chave pode ser criado em arquivos com idiomas diferentes. As mensagens da interface do usuário são criadas em arquivos com um nome que fica definido no arquivo `faces-config.xml` dentro de `WEB-INF`, seguido do idioma. Desta forma serão criados os arquivos `messages_pt_BR.properties` (código fonte 5.1) e `messages_en_US.properties` (código fonte 5.2). Nestes arquivos estão definidas mensagens para os idiomas português e inglês, com chaves de internacionalização (exemplo: `pessoa.form.header`) e valores para as chaves (exemplo: Edição de Pessoas).

```
1 pessoa.form.header=Edição de Pessoas
2 pessoa.form.nome=Nome
3 pessoa.form.email=Email
4 pessoa.form.nascimento=Nascimento
5 form.salvar=Salvar
6 pessoa.salvar.sucesso=Pessoa persistida com sucesso
```

Código Fonte 5.1 – `messages_pt_BR.properties`

```
1 pessoa.form.header=Edit People
2 pessoa.form.nome=Name
3 pessoa.form.email=Email
4 pessoa.form.nascimento=Birth
5 form.salvar=Save
6 pessoa.salvar.sucesso=Person successfully persisted
```

Código Fonte 5.2 – `messages_en_US.properties`

Para as mensagens de internacionalização da bean validation API devem ser criados arquivos com o nome `ValidationMessages` mais o nome do idioma. Desta forma foram definidos os arquivos para os idiomas português (código fonte 5.3) e inglês (código fonte 5.4).

```

1 pessoa.nome.notNull=0 nome da pessoa deve ser informado
2 pessoa.nome.length=0 nome não pode ter mais que {max} caracteres
3 pessoa.nome.notblank=0 nome da pessoa não pode ser em branco

```

Código Fonte 5.3 – ValidationMessages_pt_BR.properties

```

1 pessoa.nome.notNull=The name of the person should be informed
2 pessoa.nome.length=The name can not be more than {max} characters
3 pessoa.nome.notblank=The person's name can not be blank

```

Código Fonte 5.4 – ValidationMessages_en_US.properties

Todos os arquivos devem ser criados na raiz dos pacotes de código fonte. A figura 62 ilustra a estrutura dos arquivos no projeto.

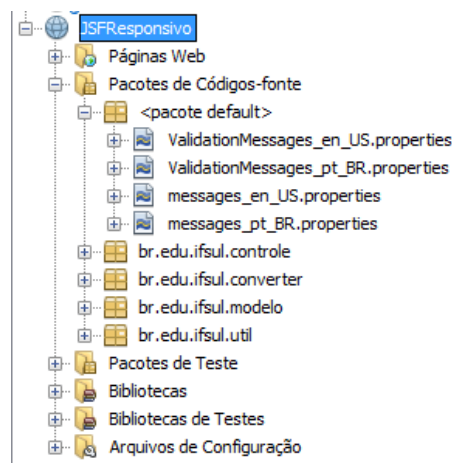


Figura 62 – Estrutura do projeto com os arquivos de internacionalização

5.2 Registro no arquivo faces-config.xml dos idiomas suportados

Os idiomas suportados devem ser registrados no arquivo faces-config.xml que fica dentro da pasta WEB-INF em páginas web. O conteúdo deste arquivo pode ser visualizado no código fonte 5.5. Entre as linhas 14 e 18, são informados os idiomas suportados e o idioma padrão da aplicação. Já entre as linhas 19 e 22, informa-se o nome dos arquivos de internacionalização (linha 20), e uma variável para associar as chaves de internacionalização (linha 21).

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <faces-config version="2.2"
3     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6         http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
7     <application>
8         <navigation-rule>
9             <from-view-id/*</from-view-id>
10            <navigation-case>
11                <from-outcome>inicio</from-outcome>
12                <to-view-id>index.xhtml</to-view-id>
13            </navigation-case>
14        </navigation-rule>
15        <locale-config>
16            <default-locale>pt_BR</default-locale>

```

```

16     <supported-locale>en_US</supported-locale>
17     <supported-locale>pt_BR</supported-locale>
18 </locale-config>
19 <resource-bundle>
20     <base-name>messages</base-name>
21     <var>msgs</var>
22 </resource-bundle>
23 </application>
24 </faces-config>

```

Código Fonte 5.5 – Idiomas no faces-config.xml

5.3 Associando os valores das mensagens com as chaves de internacionalização

Em todos os textos das telas deve-se utilizar as chaves de internacionalização. Para associar um texto de uma tela com uma chave de internacionalização deve-se usar o seguinte código `msgs['chave']`, onde `msgs` é o nome definido no `faces-config.xml` e `chave` é a chave de internacionalização. O código fonte 5.6 exemplifica seu uso.

```

1 <p:outputLabel value="#{msgs['pessoa.form.header']}" />

```

Código Fonte 5.6 – Mensagem de internacionalização na tela

Para as anotações da bean validation API basta se informar o nome da chave de internacionalização entre chaves, como pode ser visualizado no código fonte `i18nBean`.

```

1 @NotNull(message = "{pessoa.nome.notnull}")
2 @Length(max = 50, message = "{pessoa.nome.length}")
3 @NotBlank(message = "{pessoa.nome.notblank}")
4 private String nome;

```

Código Fonte 5.7 – Mensagem de internacionalização nas validações

5.4 Controle do JSF para os idiomas

É necessário criar um controlador do JSF para selecionar o idioma que o usuário deseja. ele pode ser visualizado no código fonte 5.8. Na linha 21 é definido um atributo para se armazenar o idioma escolhido pelo usuário, que inicialmente é o português, e métodos para se alterar o idioma.

```

1
2 package br.edu.ifsul.controle;
3
4 import java.io.Serializable;
5 import java.util.Locale;
6 import javax.faces.bean.ManagedBean;
7 import javax.faces.bean.SessionScoped;
8 import javax.faces.component.UIViewRoot;
9 import javax.faces.context.FacesContext;
10
11 /**
12  *
13  * @author Jorge Luis Boeira Bavaresco
14  * @email jorge.bavaresco@passofundo.ifsul.edu.br
15  * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
16  */
17 @ManagedBean(name = "controleLocale")

```

```

18 @SessionScoped
19 public class ControleLocale implements Serializable {
20
21     private Locale currentLocale = new Locale("pt", "BR");
22
23     public ControleLocale() {
24     }
25
26     public void englishLocale() {
27         UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();
28         currentLocale = Locale.US;
29         viewRoot.setLocale(currentLocale);
30     }
31
32     public void portugueseLocale() {
33         UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();
34         currentLocale = new Locale("pt", "BR");
35         viewRoot.setLocale(currentLocale);
36     }
37
38     public Locale getCurrentLocale() {
39         return currentLocale;
40     }
41
42     public void setCurrentLocale(Locale currentLocale) {
43         this.currentLocale = currentLocale;
44     }
45 }

```

Código Fonte 5.8 – ControleLocale.java

Em todas as telas é necessário associar o valor da view (f:view) com a localidade do controlador, tornando possível que o JSF identifique como associar as mensagens de internalização com as telas. Isso pode ser visualizado na linha 8 do código fonte 5.9. Entre as linhas 24 e 29 são definidos os menus para que se escolha os idiomas.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:h="http://xmlns.jcp.org/jsf/html"
5     xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
6     xmlns:b="http://bootsfaces.net/ui"
7     xmlns:f="http://xmlns.jcp.org/jsf/core">
8 <f:view encoding="ISO-8859-1" contentType="text/html" locale="#{controleLocale.currentLocale}">
9     <h:head>
10        <title><ui:insert name="titulo">titulo da pagina</ui:insert</title>
11    </h:head>
12    <h:body>
13        <b:container>
14            <h:form id="formMenu">
15                <b:navBar brand="IFSUL">
16                    <b:navBarLinks>
17                        <b:navLink value="Inicio" outcome="inicio" />
18                        <b:dropMenu value="Cadastros">
19                            <b:navCommandLink value="Pessoas"
20                                ajax="false"
21                                action="#{controlePessoa.listar()}" />

```

```

22         </b:dropMenu>
23         <b:dropMenu value="Idiomas">
24             <b:navCommandLink value="Português"
25                 ajax="false"
26                 action="#{controleLocale.portugueseLocale()}" />
27             <b:navCommandLink value="Inglês"
28                 ajax="false"
29                 action="#{controleLocale.englishLocale()}" />
30         </b:dropMenu>
31     </b:navbarLinks>
32 </b:navBar>
33 </h:form>
34 <ui:insert name="conteudo">
35
36     </ui:insert>
37 </b:container>
38 </h:body>
39 </f:view>
40 </html>

```

Código Fonte 5.9 – template.xhtml com a definição da localidade

5.5 Internacionalização de mensagens dentro de métodos ou classes

Diferente das telas e da beans validation API, quando se quer internacionalizar mensagens dentro de métodos ou classes, é necessário buscar o valor da chave de internacionalização manualmente. Para facilitar esta tarefa a classe Util.java (código fonte 5.10) foi criada, contendo o método getMessageInternacionalizada, que recebe como parâmetro a chave de internacionalização e retorna o seu valor.

```

1 package br.edu.ifsul.util;
2
3 import java.util.Locale;
4 import java.util.ResourceBundle;
5 import javax.faces.context.FacesContext;
6
7 /**
8  *
9  * @author Jorge Luis Boeira Bavaresco
10 * @email jorge.bavaresco@passofundo.ifsul.edu.br
11 * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
12 */
13 public class Util {
14
15     public static String getMessageInternacionalizada(String messageID) {
16         FacesContext facesContext = FacesContext.getCurrentInstance();
17         String msg = "";
18         Locale locale;
19         if (facesContext != null) {
20             locale = facesContext.getViewRoot().getLocale();
21         } else {
22             locale = new Locale("pt", "BR");
23         }
24         ResourceBundle bundle
25             = ResourceBundle.getBundle("messages", locale);
26         try {
27             msg = bundle.getString(messageID);

```



```
28     } catch (Exception e) {
29         System.out.println("Não encontrou a mensagem");
30         msg = messageID;
31     }
32     return msg;
33 }
34
35 }
```

Código Fonte 5.10 – Util.java

Para se utilizar a internacionalização dentro de um método recuperando o valor de uma chave basta usar o método *getMessageInternacionalizada* como na linha 2 do código fonte 5.11.

```
1 FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO,
2     Util.getMessageInternacionalizada("pessoa.salvar.sucesso"), "");
```

Código Fonte 5.11 – Exemplo de uso do método getMessageInternacionalizada

Enterprise Java Beans (EJB)

Este capítulo fará uma introdução aos Enterprise Java Beans, que são uma arquitetura de componentes do lado do servidor para a plataforma Java EE (*Enterprise Edition*). Com eles é possível implementar componentes responsáveis pela lógica de negócio, além de explorar mais profundamente a plataforma Java EE.

6.1 JAVA EE

Java EE surgiu no final dos anos 1990 e trouxe para a linguagem Java uma plataforma de software robusta voltada para o desenvolvimento empresarial. Aplicações empresariais normalmente precisam acessar dados, aplicar lógica funcional, adicionar camadas de apresentação e se comunicar com sistemas externos, e o Java EE surgiu para atender estas demandas (GONCALVES, 2011).

Padrões abertos são uma força do Java EE, pois aplicações escritas com JPA, EJB, JSE, JMS, serviços web SOAP ou RESTful são portáteis entre os servidores de aplicação. A primeira versão do Java EE (J2EE) focava em questões que as companhias enfrentavam em 1999, como componentes distribuídos. A plataforma evoluiu para responder a necessidades técnicas como estas, oferecendo padrões de trabalho com as especificações, tornando-se ao longo dos anos, mais simples, mais fácil de usar e mais portátil (GONCALVES, 2011).

Fazem parte do Java EE serviços como:

- Enterprise Java Beans (EJB)
- Java Transaction API (JTA)
- Java Persistence API (JPA)
- Validação
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- Java Mail
- JavaBeans Activation Framework (JAF)
- Processamento de XML
- Java EE Connector Architecture (JCA)
- Serviços de segurança
- Serviços WEB
- Injeção de dependências

- Gerenciamento
- Distribuição

Essas especificações serão descritas detalhadamente nos tópicos seguintes.

6.1.1 EJB - Enterprise Java Beans

A tecnologia de *Enterprise Java Beans* (EJB) é uma arquitetura de componentes do lado do servidor para a plataforma Java EE (*Enterprise Edition*). Essa que permite o desenvolvimento simplificado de aplicativos distribuídos, transacionais, seguros e portáteis (ORACLE, 2015).

EJBs são componentes do lado do servidor que, além de encapsular a lógica de negócio, cuidar de aspectos transacionais e de segurança, possuem uma pilha integrada para transmissão de mensagens, acesso remoto, serviços web (SOAP e REST) e injeção de dependências. Ainda controlam o ciclo de vida dos componentes entre outras funcionalidades. Outra vantagem dos EJBs é que eles se integram com as tecnologias do Java Se e Java EE, como JDBC, JavaMail, *Java Transaction API* (JTA), *Java Messaging Service* (JMS), *Java Authentication and Authorization Service* (JAAS), *Java Naming and Directory Interface* (JNDI) e *Remote Method Invocation* (RMI). Esse fato justifica a sua utilização para a construção de camadas de lógica funcional (GONCALVES, 2011).

Os EJBs utilizam um modelo de programação que combina facilidade de uso e robustez, permitindo reuso e escalabilidade para aplicações corporativas. Isso ocorre pelo fato de utilizar anotações em um único Plain Old Java Object (POJO) que é distribuído em um contentor. Um contentor EJB é um ambiente de tempo de execução que oferece serviços como gerenciamento de transações, controle de concorrência, segurança e outras funcionalidades. Dessa forma, o desenvolvedor pode se preocupar com a lógica do negócio enquanto o contentor lida com a implementação técnica das funcionalidades. O contentor apenas precisa implementar a especificação EJB, que atualmente encontra-se na versão 3.1 (GONCALVES, 2011).

Já beans de sessão são usados para encapsular a lógica funcional de alto nível e são considerados uma parte importante da tecnologia EJB. Eles podem ser sem estado (*Stateless*), com estado (*Stateful*) ou singular (*Singleton*). Os beans de sessão sem estado não contém estado de conversação entre método, e qualquer instância pode ser usada por qualquer cliente. Os beans de sessão com estado mantém estado de conversação e uma instância pode ser usada por um único cliente. Já o bean de sessão singular é instanciado uma única vez na aplicação e compartilhado por todos os clientes (GONCALVES, 2011).

Beans de sessão com e sem estado podem ser utilizados na criação de componentes da lógica de negócio. Na figura 63 pode-se observar o trecho de código fonte de um beans de sessão com estado. A sua definição se dá por meio da anotação `@Stateful` no POJO, dessa forma o contentor EJB sabe que classe deve considerada um bean de sessão.

```
@Stateful
public class ExperimentDAO<T> extends GenericDAO<Experiment> implements Serializable {

    public ExperimentDAO() {
        super();
        super.setPersistentClass(Experiment.class);
        // inicializar as ordenações possíveis
        super.getListOrder().add(
            new Order("id", "beans.label.id", "="));
        super.getListOrder().add(
            new Order("description", "beans.label.description", "like"));
        // definir qual a ordenação padrão no caso o segundo elemento da lista (índice 1)
        super.setCurrentOrder((Order) super.getListOrder().get(0));
        // inicializando o filtro
        super.setFilter("");
        // inicializando o conversor da ordem
        super.setConverterOrder(new ConverterOrder(super.getListOrder()));
    }
}
```

Figura 63 – Código de bean de sessão *stateful*.

O recurso de injeção de dependências do EJB também pode ser utilizado na criação de componentes da lógica de negócio. Dessa forma, o container instancia automaticamente um bean de sessão e gerencia o seu ciclo de vida. Para definir quando o container deve injetar automaticamente um bean de sessão, deve-se usar a anotação `@EJB` na definição do atributo que é o bean de sessão. O seu uso pode ser visualizado na figura 64.

```
@ManagedBean(name = "controlExperiment")
@SessionScoped
public class ControlExperiment implements Serializable {

    @EJB
    private ExperimentDAO<Experiment> dao;
    private Experiment object;
    @EJB
    private PeopleDAO<People> daoPeople;
    @EJB
    private OrganizationDAO<Organization> daoOrganization;
    @EJB
    private ModelSimulationDAO<ModelSimulation> daoModelSimulation;
    private Boolean editando;
    @EJB
    private CultivarDAO<Cultivar> daoCultivar;
```

Figura 64 – Injeção de dependências do EJB.

6.2 Tipos de EJB

Os EJBs podem ser definidos em três tipos: Stateless, Stateful e Singleton. Nesta seção será explicado a característica de cada tipo.

6.2.1 Bean de sessão sem estado (@Stateless)

Não armazena estado conversacional, e uma única instancia pode atender chamadas de diversos clientes. Porém não atende duas chamadas ao mesmo método simultaneamente, processa uma de cada vez. O container pode criar várias instancias do mesmo EJB para atender mais rapidamente as chamadas. Uma classe é definida como um EJB sem estado utilizando a anotação `@Stateless` na classe.

Callback é um mecanismo que atua no ciclo de vida de um EJB, onde podem ser definidas regras de negócio associadas a etapas deste ciclo de vida. Um EJB Stateless pode utilizar os seguintes Callbacks (Tabela 3):

Tabela 3 – Callbacks para um EJB Stateless

Anotação	Descrição
<code>@PostConstruct</code>	Método é invocado quando o bean é criado pela primeira vez.
<code>@PreDestroy</code>	Método é invocado quando o bean é destruído.

O código fonte 6.1 exibido um EJB Stateless. A difinição do tipo do EJB está na linha 16, com o uso da anotação `@Stateless`. Nas linhas 23 e 28 são utilizadas as anotações `@PostConstruct` e `@PreDestroy`, executando os métodos ao criar ou destruir a instância do EJB.

```
1 package br.edu.ifsul.ejb;
2
3 import java.io.Serializable;
4 import java.text.SimpleDateFormat;
5 import java.util.Calendar;
```

```

6 import javax.annotation.PostConstruct;
7 import javax.annotation.PreDestroy;
8 import javax.ejb.Stateless;
9
10 /**
11  *
12  * @author Jorge Luis Boeira Bavaresco
13  * @email jorge.bavaresco@passofundo.ifsul.edu.br
14  * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
15  */
16 @Stateless
17 public class BeanHora implements Serializable {
18
19     public BeanHora(){
20         System.out.println("Usou o construtor");
21     }
22
23     @PostConstruct
24     public void iniciar(){
25         System.out.println("Usou o método iniciar");
26     }
27
28     @PreDestroy
29     public void destruir(){
30         System.out.println("Usou o método destruir");
31     }
32
33
34     public String getDataHoraServidor(){
35         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss:SSS");
36         return sdf.format(Calendar.getInstance().getTime());
37     }
38 }
39 }

```

Código Fonte 6.1 – Código fonte de um EJB Stateless

6.2.2 Bean de sessão com estado (@Stateful)

Um bean de sessão stateful é utilizado quando se tem a necessidade de manter estado conversacional, normalmente usado quando necessita associar a instancia de um EJB a sessão de um usuário, por exemplo. Existe uma instancia para cada usuário do sistema. Um bean stateful pode ser definido com a anotação @Stateful.

Um EJB Stateful pode utilizar os seguintes Callbacks (Tabela 4):

Tabela 4 – Callbacks para um EJB Stateless

Anotação	Descrição
@PostConstruct	Método é invocado quando o bean é criado pela primeira vez.
@PreDestroy	Método é invocado quando o bean é destruído.
@PrePassivate	Método é invocado quando o bean é movido para o pool de beans (Apassivação).
@PostActivate	Método é invocado quando o bean é removido do pool de beans do servidor e ativado novamente.

No código fonte 6.2 pode-se visualizar um EJB stateful, com o uso da anotação @Stateful na linha 19. Na linha 20 e utilizada a anotação @StatefulTimeout, que define por quanto tempo o EJB continuará

existindo após a inatividade da sessão. Define-se a unidade (horas, minutos, segundos) e o valor. Neste caso o método anotado com `@PreDestroy` será executado após 10 minutos de inatividade da sessão.

```
1
2 package br.edu.ifsul.ejb;
3
4 import java.io.Serializable;
5 import java.util.HashSet;
6 import java.util.Set;
7 import javax.annotation.PostConstruct;
8 import javax.annotation.PreDestroy;
9 import javax.ejb.PostActivate;
10 import javax.ejb.PrePassivate;
11 import javax.ejb.Stateful;
12
13 /**
14  *
15  * @author Jorge Luis Boeira Bavaresco
16  * @email jorge.bavaresco@passofundo.ifsul.edu.br
17  * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
18  */
19 @Stateful
20 @StatefulTimeout(unit = TimeUnit.MINUTES,value = 10)
21 public class BeanCarrinho implements Serializable {
22     private Set<String> carrinho = new HashSet<>();
23
24     public BeanCarrinho() {
25     }
26
27     public Set<String> getCarrinho() {
28         return carrinho;
29     }
30
31     public void setCarrinho(Set<String> carrinho) {
32         this.carrinho = carrinho;
33     }
34
35     @PostConstruct
36     public void iniciar(){
37         System.out.println("Usou o método iniciar");
38     }
39
40     @PreDestroy
41     public void destruir(){
42         System.out.println("Usou o método destruir");
43     }
44
45     @PrePassivate
46     public void antesAtivar(){
47         System.out.println("Usou o método antesAtivar");
48     }
49
50     @PostActivate
51     public void depoisAtivar(){
52         System.out.println("Usou o método depoisAtivar");
53     }
54 }
```

54 }

Código Fonte 6.2 – Código fonte de um EJB Stateful

6.2.3 Bean de sessão Singular (@Singleton)

Beans de sessão singleton podem ser utilizados para compartilhar dados entre todos os usuários de uma aplicação, e possuem somente um instância. Pode ser usando em soluções como sistemas de chat, compartilhamento de produtos, sistemas colaborativos, etc.

Um EJB Singleton pode utilizar os mesmos callbacks de um EJB Stateless, descritos na Tabela 3.

No código fonte 6.3 pode-se visualizar um EJB singleton, com o uso da anotação @Singleton na linha 16. Na linha seguinte está sendo utilizada a anotação @Startup, que serve para inicializar o Bean no momento em que a aplicação é carregada, anotação que é opcional.

```
1 package br.edu.ifsul.ejb;
2
3 import java.io.Serializable;
4 import java.util.HashSet;
5 import java.util.Set;
6 import javax.annotation.PostConstruct;
7 import javax.annotation.PreDestroy;
8 import javax.ejb.Singleton;
9
10 /**
11  *
12  * @author Jorge Luis Boeira Bavaresco
13  * @email jorge.bavaresco@passofundo.ifsul.edu.br Instituto Federal
14  * Sul-Rio-Grandense Campus Passo Fundo
15  */
16 @Singleton
17 @Startup
18 public class BeanListaProdutos implements Serializable {
19
20     private Set<String> produtos = new HashSet<>();
21
22     public BeanListaProdutos() {
23     }
24
25     @PostConstruct
26     public void iniciar() {
27         System.out.println("Usou o método iniciar");
28     }
29
30     @PreDestroy
31     public void destruir() {
32         System.out.println("Usou o método destruir");
33     }
34
35     public Set<String> getProdutos() {
36         return produtos;
37     }
38
39     public void setProdutos(Set<String> produtos) {
40         this.produtos = produtos;
41     }
42 }
```

42 }

Código Fonte 6.3 – Código fonte de um EJB Singleton

6.2.4 Usando EJBs

Para utilizar um EJB definido, basta adicionar um atributo do tipo do EJB em uma classe e anotar ele com a anotação @EJB, como pode ser visualizado no código fonte 6.4 na linha 19.

```
1 package br.edu.ifsul.controle;
2
3 import br.edu.ifsul.ejb.BeanHora;
4 import java.io.Serializable;
5 import javax.ejb.EJB;
6 import javax.enterprise.context.RequestScoped;
7 import javax.inject.Named;
8
9 /**
10 *
11 * @author Jorge Luis Boeira Bavaresco
12 * @email jorge.bavaresco@passofundo.ifsul.edu.br Instituto Federal
13 * Sul-Rio-Grandense Campus Passo Fundo
14 */
15 @Named("controleBeanHora")
16 @RequestScoped
17 public class ControleBeanHora implements Serializable {
18
19     @EJB
20     private BeanHora beanHora;
21
22     public BeanHora getBeanHora() {
23         return beanHora;
24     }
25
26     public void setBeanHora(BeanHora beanHora) {
27         this.beanHora = beanHora;
28     }
29
30 }
```

Código Fonte 6.4 – Código fonte de um EJB Singleton

6.3 Controladores do JSF gerenciados pelo servidor de aplicações (CDI)

O servidor de aplicações pode gerenciar o ciclo de vida de controladores do JSF, oferecendo um modelo mais flexível que o conceito de beans gerenciados (@ManagedBean). Isso foi definido na JSR 299 (*Context and dependency injection*) cuja sigla é CDI (GEARY; HORSTMANN, 2012).

Um bean CDI é usada da mesma forma que um bean gerenciado do JSF, somente usando a anotação @Named (pacote javax.inject) no lugar da anotação @ManagedBean. Desta forma fica definido que instancias da classe serão gerenciadas pelo container EJB. Um exemplo de uso de um bean CDI pode ser visualizado no código fonte 6.5.

```
1
2 package br.edu.ifsul.controle;
3
4 import br.edu.ifsul.ejb.BeanHora;
```



```

5 import java.io.Serializable;
6 import javax.ejb.EJB;
7 import javax.enterprise.context.RequestScoped;
8 import javax.inject.Named;
9
10 /**
11  *
12  * @author Jorge Luis Boeira Bavaresco
13  * @email jorge.bavaresco@passofundo.ifsul.edu.br
14  * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
15  */
16
17 @Named("controleBeanHora")
18 @RequestScoped
19 public class ControleBeanHora implements Serializable {
20     @EJB
21     private BeanHora beanHora;
22
23     public BeanHora getBeanHora() {
24         return beanHora;
25     }
26
27     public void setBeanHora(BeanHora beanHora) {
28         this.beanHora = beanHora;
29     }
30
31 }

```

Código Fonte 6.5 – Código fonte de um bean CDI do JSF

As anotações do ciclo de vida do bean do JSF CDI devem usar pacotes diferentes dos pacote `javax.faces.bean`, que era usado com a anotação `@ManagedBean`. A importação necessária para cada escopo escolhido pode ser visualizada na Tabela 5.

Tabela 5 – Anotações do ciclo de vida de um Bean CDI do JSF

<i>Anotação do ciclo de vida</i>	<i>Importação necessária</i>
<code>@RequestScoped</code>	<code>import javax.enterprise.context.RequestScoped;</code>
<code>@ViewScoped</code>	<code>import javax.faces.view.ViewScoped;</code>
<code>@SessionScoped</code>	<code>import javax.enterprise.context.SessionScoped;</code>
<code>@ApplicationScoped</code>	<code>import javax.enterprise.context.ApplicationScoped;</code>

6.3.1 Injetando um bean CDI do JSF

Uma tarefa bastante comum nos controladores do JSF é fazer acesso a outro controlador do JSF. Como nos beans CDI o controle das instancias é realizado pelo servidor esta tarefa é facilitada com o uso da anotação `@Inject`.

O código fonte 6.6 apresenta um exemplo de um bean CDI que armazena o nome de um usuário na sessão. Já no código fonte 6.7 é necessário acessar a informação do nome do usuário na sessão, que está dentro da instância do `controleUsuario` (código fonte 6.6). Para isto basta definir um atributo do tipo `ControleUsuario` (código fonte 6.7 linha 24) e anotar ele com a anotação `@Inject` (código fonte 6.7 linha 23). Desta forma a instancia deste atributo será injetada pelo container EJB.

```

1 package br.edu.ifsul.controle;
2
3 import java.io.Serializable;
4 import javax.enterprise.context.SessionScoped;

```

```
5 import javax.inject.Named;
6
7 /**
8  *
9  * @author Jorge Luis Boeira Bavaresco
10 * @email jorge.bavaresco@passofundo.ifsul.edu.br Instituto Federal
11 * Sul-Rio-Grandense Campus Passo Fundo
12 */
13 @Named(value = "controleUsuario")
14 @SessionScoped
15 public class ControleUsuario implements Serializable {
16
17     private String usuario;
18
19     public ControleUsuario() {
20     }
21
22     public String informarUsuario() {
23         return "index";
24     }
25
26     public String getUsuario() {
27         return usuario;
28     }
29
30     public void setUsuario(String usuario) {
31         this.usuario = usuario;
32     }
33 }
```

Código Fonte 6.6 – Código fonte de um bean CDI que será injetado

```
1
2 package br.edu.ifsul.controle;
3
4 import br.edu.ifsul.ejb.BeanHora;
5 import java.io.Serializable;
6 import javax.ejb.EJB;
7 import javax.enterprise.context.RequestScoped;
8 import javax.inject.Inject;
9 import javax.inject.Named;
10
11 /**
12  *
13  * @author Jorge Luis Boeira Bavaresco
14  * @email jorge.bavaresco@passofundo.ifsul.edu.br
15  * Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
16  */
17
18 @Named("controleBeanHora")
19 @RequestScoped
20 public class ControleBeanHora implements Serializable {
21     @EJB
22     private BeanHora beanHora;
23     @Inject
24     private ControleUsuario controleUsuario;
25 }
```

```

26
27     public String getUsuario(){
28         return controleUsuario.getUsuario() != null ? controleUsuario.getUsuario() : "Usuario n
                ão informado";
29     }
30
31     public BeanHora getBeanHora() {
32         return beanHora;
33     }
34
35     public void setBeanHora(BeanHora beanHora) {
36         this.beanHora = beanHora;
37     }
38
39 }

```

Código Fonte 6.7 – Código fonte onde bean CDI que foi injetado

6.4 Serviço temporizador

A especificação do Java EE possui uma facilidade para realizar agendamento de tarefas chamada serviço temporizador. Ele surgiu na especificação do EJB 2.1, e na versão do EJB 3.1 que se obteve uma melhoria drástica neste serviço, que se assemelha a outras ferramentas ou recursos como por exemplo o CRON do UNIX, porém com mais funcionalidades. Este serviço é executado dentro do conteúdos EJB, para classes que são beans de sessão (EJB).

O serviço temporizador pode ser útil para executar tarefas, como por exemplo, de enviar um email para aniversariantes. Pode-se, desta forma agendar que um método seja executado todos os dias a meia noite, fazendo uma consulta na base de dados e enviando email para quem faz aniversário no dia.

Beans sem estado (@Stateless) e beans Singleton (@Singleton) podem ser registrados pelo serviço temporizador, porém beans com estado (@Stateful) não podem ser usados. Os temporizadores são criados automaticamente pelo contentor no momento da distribuição, se o bean tiver métodos anotados com @Schedule.

As expressões usadas pelo serviço temporizador são baseadas na sintaxe de calendário do utilitário CRON do UNIX juntamente com o uso da anotação @Schedule. Para a criação de expressões podem ser usados os atributos apresentados na Tabela 6, e na Tabela 7 (GONCALVES, 2011) pode-se observar alguns exemplos de expressões do temporizador.

Tabela 6 – Atributos para a criação de expressões

Atributo	Valores permitidos
second	[0,59]
minute	[0,59]
hour	[0,23]
dayOfMonth	[1,31] ou "1st", "2nd", "3rd", . . . , "0th", "31st" ou "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" ou "Last" (o último dia do mês) ou -x (numero x de dias antes do fim do mês)
month	[1,12] ou "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
dayOfWeek	[0,7] ou "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" — "0" e "7" para domingo
year	ano com quatro dígitos

Tabela 7 – Exemplos de expressões do serviço temporizador

<i>Exemplo</i>	<i>Expressão</i>
Toda quarta-feira à meia-noite	dayOfWeek="Wed"
Toda quarta-feira à meia-noite	second="0", minute="0", hour="0", dayOfMonth="*", month="*", dayOfWeek="Wed", year=""
Todos os dias às 6:55	minute="55", hour="6", dayOfWeek="Mon-Fri"
A cada minuto de todas as horas	minute="*", hour=""
A cada segundo de todas as horas de todos os dias	second="*", minute="*", hour=""
Todas as segundas e sextas-feiras, 30 segundos após o meio dia	second="30", hour="12", dayOfWeek="Mon, Fri"
A cada cinco minutos da hora	minute="*/5", hour=""
A cada cinco minutos da hora	minute="0,5,10,15,20,25,30,35, 40,45,50,55", hour=""
A ultima segunda-feira de dezembro às 15h	hour="15", dayOfMonth="Last Mon", month="Dec"

O código fonte 6.8 exibe um EJB que possui métodos que criam temporizadores. Na linha 21 usa-se a anotação `@Schedule(minute = "*", hour = "*")` para definir um temporizador que será executado todos os minutos em todas as horas. Entre as linhas 27 e 30 são definidos dois temporizadores para o mesmo método com o uso da anotação `@Schedules`, para executar o método as 2 horas, e as 14 horas das quartas-feiras. Com o uso destas anotações o contentor EJB cria os temporizadores no momento da distribuição (deploy) da aplicação no servidor.

```

1 package br.edu.ifsul.ejb;
2
3 import java.io.Serializable;
4 import javax.ejb.Schedule;
5 import javax.ejb.Schedules;
6 import javax.ejb.Singleton;
7
8 /**
9  *
10 * @author Jorge Luis Boeira Bavaresco
11 * @email jorge.bavaresco@passofundo.ifsul.edu.br
12 * @organization Instituto Federal Sul-Rio-Grandense Campus Passo Fundo
13 */
14 @Singleton
15 public class BeanTemporizador implements Serializable {
16
17     public BeanTemporizador() {
18
19     }
20
21     //Executa a cada minuto de cada hora de todos os dias
22     @Schedule(minute = "*", hour = "*")
23     public void metodoAgendado01(){
24
25     }
26
27     @Schedules({ // Cria dois temporizadores
28         @Schedule(hour = "2"), // executa as 2 horas
29         @Schedule(hour = "14", dayOfWeek = "Wed") // executa as duas horas da quarta
30     })
31     public void metodoAgendado02(){

```

```
32  
33     }  
34 }
```

Código Fonte 6.8 – EJB com métodos que usam temporizador

Referências

BISWAS, R.; ORT, E. The java persistence api-a simpler programming model for entity persistence. *Sun, May*, 2006. Citado na página 26.

BOOTSFACES. *BootsFaces*. 2016. Disponível em: <<http://www.bootsfaces.net/>>. Acesso em 15 Agosto 2016. Citado na página 71.

GEARY, D.; HORSTMANN, C. *Core JavaServer Faces*. [S.l.]: Altabooks, 2012. Citado 2 vezes nas páginas 71 e 88.

GONCALVES, A. *Introdução à Plataforma Java EE 6 com GlassFish 3*. [S.l.]: Editora Ciência Moderna, 2011. v. 2. Citado 5 vezes nas páginas 26, 27, 82, 83 e 91.

ORACLE. *Enterprise JavaBeans Technology*. 2015. Disponível em: <<<http://www.oracle.com/technetwork/java/javase/ejb/index.html>>>. Acesso em 28 Agosto 2015. Citado na página 83.

PRIMEFACES. *Primefaces*. 2016. Disponível em: <<http://www.primefaces.org>>. Acesso em 15 Agosto 2016. Citado na página 71.

RICHFACES. *RichFaces*. 2016. Disponível em: <<http://richfaces.jboss.org/>>. Acesso em 15 Agosto 2016. Citado na página 71.

Documento Digitalizado Público

Material resultante do projeto de ensino.

Assunto: Material resultante do projeto de ensino.
Assinado por: Jorge Bavaresco
Tipo do Documento: Documento
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Mídia

Documento assinado eletronicamente por:

■ **Jorge Luis Boeira Bavaresco**, JORGE LUIS BOEIRA BAVARESCO - PROFESSOR ENS BASICO TECN TECNOLOGICO, em 29/10/2019 15:24:35.

Este documento foi armazenado no SUAP em 29/10/2019. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsul.edu.br/verificar-documento-externo/> e forneça os dados abaixo:

Código Verificador: 66100

Código de Autenticação: 8b111f2a7f





SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

De acordo, considerando o relatório final.

Assinatura:

Despacho assinado eletronicamente por:

- Maria Carolina Fortes, Maria Carolina Fortes - CHEFE DE DEPARTAMENTO - CD4 - PF-DEPEX, PF-DEPEX, em 06/11/2019 10:29:21.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

De acordo, considerando, além dos pareceres das chefias imediatas, o parecer da pró-reitoria de ensino.

Assinatura:

Despacho assinado eletronicamente por:

- Alexandre Pitol Boeira, Alexandre Pitol Boeira - DIRETOR GERAL - CD2 - PF-DIRGER, PF-DIRGER, em 06/11/2019 21:19:13.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Encaminhado à Diretoria de Políticas de Ensino e Inclusão para Avaliação e Parecer.

Assinatura:

Despacho assinado eletronicamente por:

- Leonardo Olsen de Campos Silva, Leonardo Olsen de Campos Silva - ASSISTENTE EM ADMINISTRACAO, IF-PROEN, em 07/11/2019 14:25:24.



SERVIÇO PÚBLICO FEDERAL

Instituto Federal Sul-rio-grandense

Despacho:

Aprovo o relatório final do Projeto de Ensino "Desenvolvimento de sistemas com a plataforma Java EE", pois o referido projeto cumpriu com êxito os objetivos propostos. Encaminho para os devidos registros e certificação.

Assinatura:

Despacho assinado eletronicamente por:

- Veridiana Krolow Bosenbecker, Veridiana Krolow Bosenbecker - DIRETOR - CD3 - IF-DIRPEI, IF-DIRPEI, em 25/11/2019 20:32:06.