UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE SISTEMA WEB DE GESTÃO DE PESSOAS PARA O TRIBUNAL REGIONAL ELEITORAL DE MATO GROSSO

WEINNE WILLAN MOREIRA SANTOS

CUIABÁ – MT
2015

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE SISTEMA WEB PARA GESTÃO DE PESSOAS PARA O TRIBUNAL REGIONAL ELEITORAL DE MATO GROSSO

WEINNE WILLAN MOREIRA SANTOS

Relatório apresentado no Instituto de Computação da Universidade Federal de Mato Grosso para obtenção do título de Bacharel em Sistemas de Informação.

CUIABÁ – MT
2015

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

WEINNE WILLAN MOREIRA SANTOS

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Sistemas de Informação como uma das exigências para obtenção do título de Bacharel em Sistemas de Informação da Universidade Federal de Mato Grosso

Aprovado por:

Prof. MSc. KAREN DA SILVA FIGUEIREDO

Instituto de Computação

EDGARD BELTRÃO LEONEL

Tribunal Regional Eleitoral

Prof. MSc. FERNANDO CASTILHO

Instituto de Computação

DEDICATÓRIA

À Deus, o SENHOR,

À minha família pelo apoio

AGRADECIMENTOS

Primeiramente agradeço a Deus, o SENHOR, por ter me fortalecido quando estive cansado e revigorado quando estive sem forças. Agradeço também aos meus pais e meus irmãos por terem incentivado meus estudos e me dado todas as oportunidades.

Agradeço aos professores do Instituto de Computação, pelo conhecimento transmitido e pelo esforço em exercer sua vocação, para o enriquecimento da formação de todos os alunos, tanto ética quanto acadêmica e profissional. Agradeço em especial a professora Karen Figueiredo, por me auxiliar e responder minhas dúvidas com paciência.

Agradeço aos amigos que fiz no curso de Sistemas de Informação, por estarem sempre presentes nessa etapa e por toda a vida. Agradeço em especial os amigos Thais Bueno e Julian Rodrigues, além do aluno de Ciência da Computação Eduardo Borges, por serem verdadeiros amigos, para todas as horas.

Agradeço aos meus colegas do Tribunal Regional Eleitoral, cuja parceria enriqueceu minha formação profissional e me deu experiência que me será útil por toda a vida. Agradeço especialmente o Edgard Beltrão Leonel, meu supervisor no estágio e amigo.

Agradeço a Aliança Bíblica Universitária do Brasil (ABUB) e a ABU Cuiabá, por terem sido parte importante da minha vida acadêmica, me presenteado com amigos de alma, e me dado um propósito na universidade. Seria impossível mencionar todos os que foram importantes, mas agradeço em especial a Luciana Fernandes, Andrei Mantesso Coimbra e Cezar Macegoza, por terem me discipulado e ensinado através de suas vidas.

Agradeço a todos os amigos que me me acompanharam, ajudaram, me presentearam com risadas e momentos bons, e foram pacientes com meus erros, permanecendo ao meu lado. Agradeço também a Aniely, minha namorada, que não me deixou ter dúvidas de que continuaria com esse trabalho até o fim.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

CPU	Central Processing Unit – Unidade Central de Processamento
PMI	Project Management Institute
PMBOK	Project Management Body of Knowledge
UML	Unified Modeling Language
SGP-Web	Sistema de Gestão de Pessoas
SBD	Seção de Banco de Dados
CP	Coordenadoria de Pessoal
JSF	Java Server Faces
CDI	Contexts and Dependency Injection for the JavaTM Platform
JPA	Java Persistence API
EJB	Enterprise Java Beans
Java EE	Java Enterprise Edition
EPF	Ecipse Proccess Framework
XP	Extreme Programing

RESUMO

Este trabalho apresenta o desenvolvimento do Sistema de Gestão de Pessoas (SGP-Web), um sistema que será utilizado pela Coordenadoria de Pessoal do Tribunal Regional Eleitoral de Mato Grosso para gerir a frequência dos servidores do quadro. O trabalho propõe também a utilização do Processo Praxis, como uma solução para a lacuna causada pela falta de um processo formal de desenvolvimento de *software* na Seção de Banco de Dados (SBD), que apresenta um meio termo entre a liberdade e a colaboração das metodologias ágeis, a formalidade do método clássico e o controle do Processo Unificado. Pretende-se durante o trabalho a documentação do *software*, desde o levantamento de requisitos, a criação de modelos, diagramas e a descrição de casos de uso, e o desenvolvimento do software, utilizando a especificação *Java EE*, as ferramentas *Netbeans IDE, Enterprise Architect, Apache Subversion e Apache Maven*.

Palavras-chave: Processos, Praxis, Orientação a Objetos, Engenharia de Software

INTRODUÇÃO

A Coordenadoria de Pessoal (CP) do Tribunal Regional Eleitoral de Mato Grosso é a seção responsável pelo acompanhamento da frequência dos servidores do quadro, assim como os afastamentos, justificativas, e a reflexão desses dados na folha de pagamento. A gerência de dados da CP é feita por diversos sistemas que compõem um conjunto que podemos chamar de Frequência Nacional. Esses sistemas acessam um banco de dados central, gerido pelo Tribunal Superior Eleitoral.

Um dos sistemas utilizados é o Sistema de Gestão de Pessoas (SGP-Web). O SGP-Web é acessado tanto pela coordenadoria quanto por outros servidores do quadro que podem ver no sistema informações relativas a sua frequência, férias, folgas e substituições em sua seção.

O SGP-Web é desenvolvido utilizando uma versão anterior da especificação Java EE, JSF 1.2, Hibernate e JBoss Seam. Algumas decisões acerca de sua arquitetura e dependências passaram a prejudicar o desenvolvimento e a falta de documentação adequada e atualizada fez com que o *software* se tornasse difícil de manter. Outros problemas foram identificados, denrte eles a ausência de *webservices* oferecidos pelo sistema, o que dificulta a integração com outros sistemas do tribunal; o código fortemente acoplado, com interfaces pouco definidas e dependência de um *framework* muito inflexível, que assume muitas responsabilidades em si e causa reescrita de código em toda situação não prevista; e a ausência de uma documentação robusta, o que dificulta o entendimento de todas as regras de negócio.

O objetivo deste trabalho foi o desenvolvimento de um novo *software*, que possua os mesmos requisitos do SGP-Web, utilizando a especificação Java EE mais atual e o padrão de arquitetura *Enterprise Web Application*, o que diminui o acoplamento entre os módulos do sistema e facilita sua expansão e integração com outros sistemas do Frequência Nacional. O desenvolvimento do *software* foi guiado pelo Processo Praxis, com suas disciplinas e fases descritas no livro-texto (PAULA FILHO, 2009) e no hipertexto gerado pelo *Eclipse Proccess Framework*.

Durante o estágio o aluno aprendeu a trabalhar bem com o gerenciador de dependências Maven, a ferramenta de modelagem *Enterprise Architect*, o ambiente

de desenvolvimento Netbeans, o gerenciador de versões Subversion, assim como a desenvolver para a plataforma Java EE, utilizando suas diversas especificações, como JSF, CDI, JPA e EJB.

A partir desta introdução, este trabalho está organizado da seguinte forma: no Capítulo 1 é apresentada a revisão de literatura com as definições de *Processos de software*, *Processo Praxis* e *Mapeamento Objeto-relacional*; no Capítulo 2 são apresentados os materiais, técnicas e métodos utilizados na realização deste trabalho, tais como: *Enterprise Architect*, *Apache Maven*, *Subversion*, *IDE Netbeans* e *Java EE*, além das fases do Processo Praxis que envolvem o *Desenvolvimento*; no Capítulo 3 são apresentados os resultados atingidos e no Capítulo 4 são apresentadas as dificuldades encontras no decorrer deste trabalho. Por fim, no Capítulo 5 é apresentada a conclusão obtida e no Capítulo 6 as referências bibliográficas utilizadas como apoio a este trabalho.

1. REVISÃO DE LITERATURA

Para realização e compreensão das atividades propostas são necessárias algumas considerações sobre o referencial teórico do trabalho. Assim, nas subseções seguintes são abordados os seguintes conceitos: *Processos de Software*, *Processo Praxis* e *Mapeamento Objeto-Relacional*. Estes conhecimentos teóricos foram adquiridos ao longo do curso de Sistemas de Informação e através de estudos dirigidos no ambiente de trabalho e sob a orientação acadêmica recebida durante o período do estágio descrito neste relatório.

1.1. Processos de *Software*

Processo, metodologia ou procedimento de *software* é um "conjunto de atividades que leva à produção de um produto de *software*" (SOMMERVILLE, 2007).

> Os procedimentos definem a sequência em que os métodos serão aplicados, os produtos (*deriverables*) que se exige que sejam entregues (documentos, relatórios, formularios, etc.), os controles que ajudam a assegurar a qualidade e a coordenar as mudanças e os marcos de referência que possibilitam aos gerentes de *software* avaliar o progresso. (PRESMANN, 1995, p. 32)

Segundo o PMBOK (PMI, 2004, p. 38), um processo de *software* é "um conjunto de ações e atividades inter-relacionadas realizadas para obter um conjunto especificado de produtos, resultados ou serviços". Em Gerência de Projetos, pode-se dizer que um processo "é uma receita que é seguida por um projeto e que o projeto concretiza uma abstração, que é o processo". (PAULA FILHO, 2009).

Segundo Presmann, na engenharia de software podemos identificar um "conjunto de etapas que envolve métodos, ferramentas e procedimentos", que podemos chamar de "paradigmas de engenharia de software" (PRESMANN, 1995), ou "modelo de ciclo de vida" (PAULA FILHO, 2009). Os paradigmas mais discutidos são o *ciclo de vida clássico, ou cascata, ciclos de vida em espiral, ciclos de vida baseados em prototipações ou liberações*, e *ciclos de vida da quarta geração* (PRESMANN, 1995).

O paradigma mais utilizado é ciclo de vida clássico, ou modelo cascata. De acordo com Presmann (1995, p. 32), ele "requer uma abordagem sistemática, sequencial, ao desenvolvimento do software, que se inicia no nível do sistema e avança ao longo da análise, projeto, codificação, teste e manutenção". O ciclo de vida clássoco abrange as seguntes atividades: *engenharia de sistemas, análise, projeto, codificação, teste e manutenção* (PRESMANN, 1995). Esse paradigma é o mais amplamente usado da engenharia de software, porém alguns problemas tem sido apontados por críticos ao modelo, ao questionarem sua aplicabilidade em todas as situações em que é difícil o projeto seguir o fluxo sequencial, ou o cliente declarar todas as exigências explicitamente, e esperar pacientemente pelo produto final (PRESMANN, 1995).

Se seguido de forma literal, pode ser um processo bem rígido e burocrático, em que as atividades de requisitos, análise e projeto precisam ser bem dominadas, pois, teoricamente, o processo não prevê a correção posterior de problemas nas fases anteriores (PAULA FILHO, 2009).

O paradigma de prototipação é baseado em várias entregas de pacotes, que podem ser protótipos ou mesmo uma parte funcional da aplicação. Segundo Presmann (1995, p.35), a "prototipação é um processo que capacita o desenvolvedor a criar um modelo do software que será implementado", podendo assumir uma das três formas: a forma de um protótipo gráfico que retrata a interação homem-máquina de uma forma que capacita o usuário a entender quanta interação ocorrerá; um protótipo de trabalho que implementa algum subconjunto da função exigida do software desejado; ou um programa existente que executa parte ou toda a função desejada, mas que tem outras características que serão melhoradas em função de um novo esforço de desenvolvimento (PRESMANN, 2995). O paradigma de prototipação abrange as atividades: *coleta e refinamento de requisitos, projeto rapido, construção do protótipo, avaliação do protótipo pelo cliente, refinamento do protótipo e engenharia do produto* (PRESMANN, 1995).

O modelo espiral reúne características do ciclo de vida clássico e do paradigma de prototipação. Segundo Presmann (1995, p. 38), o ciclo de vida espiral define quatro importantes atividades: *planejamento, análise dos riscos, engenharia* e *avaliação do produto pelo cliente*.

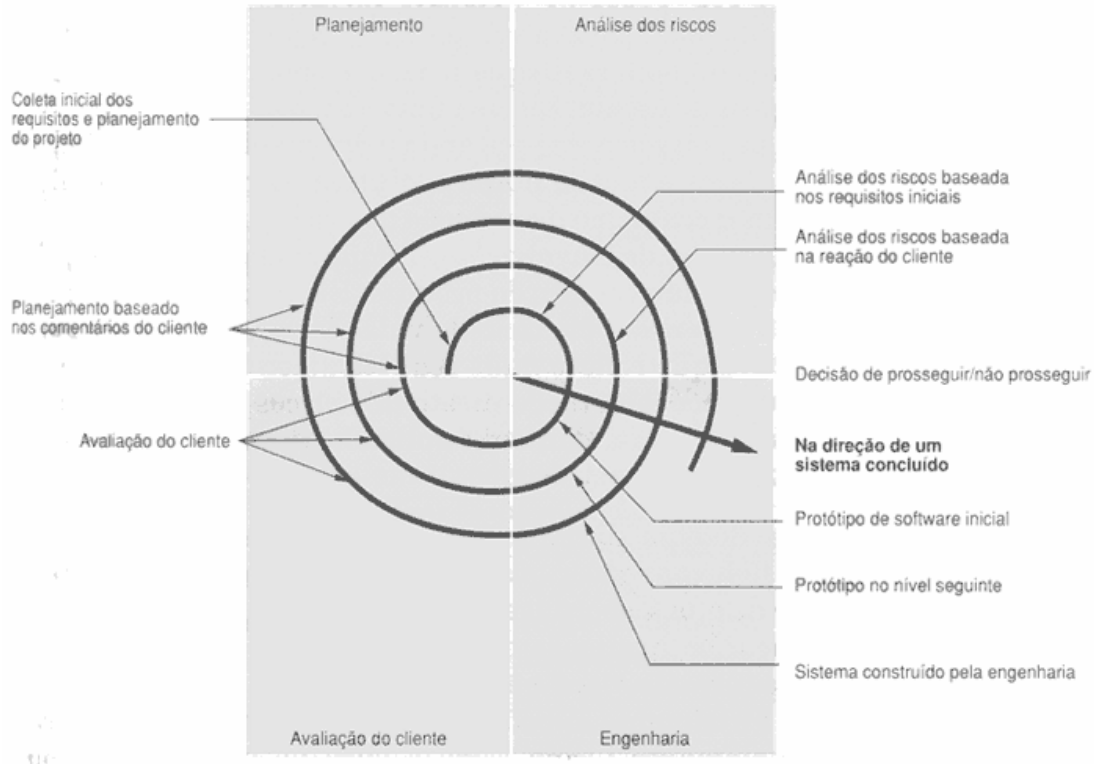


Figura 1 - O modelo espiral

Fonte: PRESMANN, 1995

A Figura 1 demonstra um ciclo de vida espiral, onde podemos ver a sua dimensão radial. A cada iteração em volta da espiral, começando pelo centro, uma versão mais completa do *software* é desenvolvida.

O modelo espiral é mais utilizado em processos ageis, como o SCRUM e o XP. O Processo Praxis, metodologia de software escolhida para este projeto, é um modelo semi-espiral. Segundo Wilson de Pádua Paula Filho (2009, p. 95), "o modelo espiral puro dificilmente é usado, mesmo em processos ágeis como o XP. Sempre existe uma fase de iniciação, na qual é feita pelo menos uma definição mínima dos requisitos do produto, para delimitar seu escopo, e uma fase de transição, na qual o produto completo é implantado em seu ambiente definitivo".

Figura 2 – Ciclo de vida semi-espiral

Fonte: PAULA FILHO, 2009

Por fim, os ciclos de vida baseados em técnicas de quarta geração são metodologias diversas que tem em comum o fato de permitir que "o desenvolvedor especifique alguma característica do software num nível elevado" (PRESMANN, 1995). Uma ferramenta fica encarregada de gerar o código-fonte, baseado nessa especificação.

1.2. Processo Praxis

Praxis é um processo desenhado para dar suporte a projetos didáticos, em disciplinas de engenharia de software de cursos de informática e em programas de capacitação profissional em engenharia de software. A sigla Praxis significa *PRocesso para Aplicativos eXtensíveis InterativoS*, refletindo uma ênfase no desenvolvimento de aplicativos gráficos interativos, baseados na tecnologia orientada a objetos (PAULA FILHO, 2009). O processo, suas disciplinas e exemplos a serem

usados nos artefatos podem ser encontrados no livro *engenharia de software: fundamentos, métodos e padrões*, de Wilson Pádua de Paula Filho.

O Processo Praxis tem um ciclo de vida semi-espiral, que pode ser entendido conforme a Figura 3.

Figura 3 – Ciclo de vida do Processo Praxis

Fonte: PAULA FILHO, 2014

Fora da espiral ficam as fases de **Iniciação** e **Transição**. A espiral é executada dentro das iterações de **Elaboração** e **Construção**, que ocupam a maior parte do esforço do projeto. Em cada **Iteração** dessas fases, é desenvolvido um conjunto de **Casos de uso**. Cada fase pode ter mais de uma iteração, se necessário.

Iniciação: fase em que procura atingir o consenso entre as partes interessadas sobre escopo e objetivos do projeto. O fluxo da fase de Iniciação pode ser observado na Figura 4.

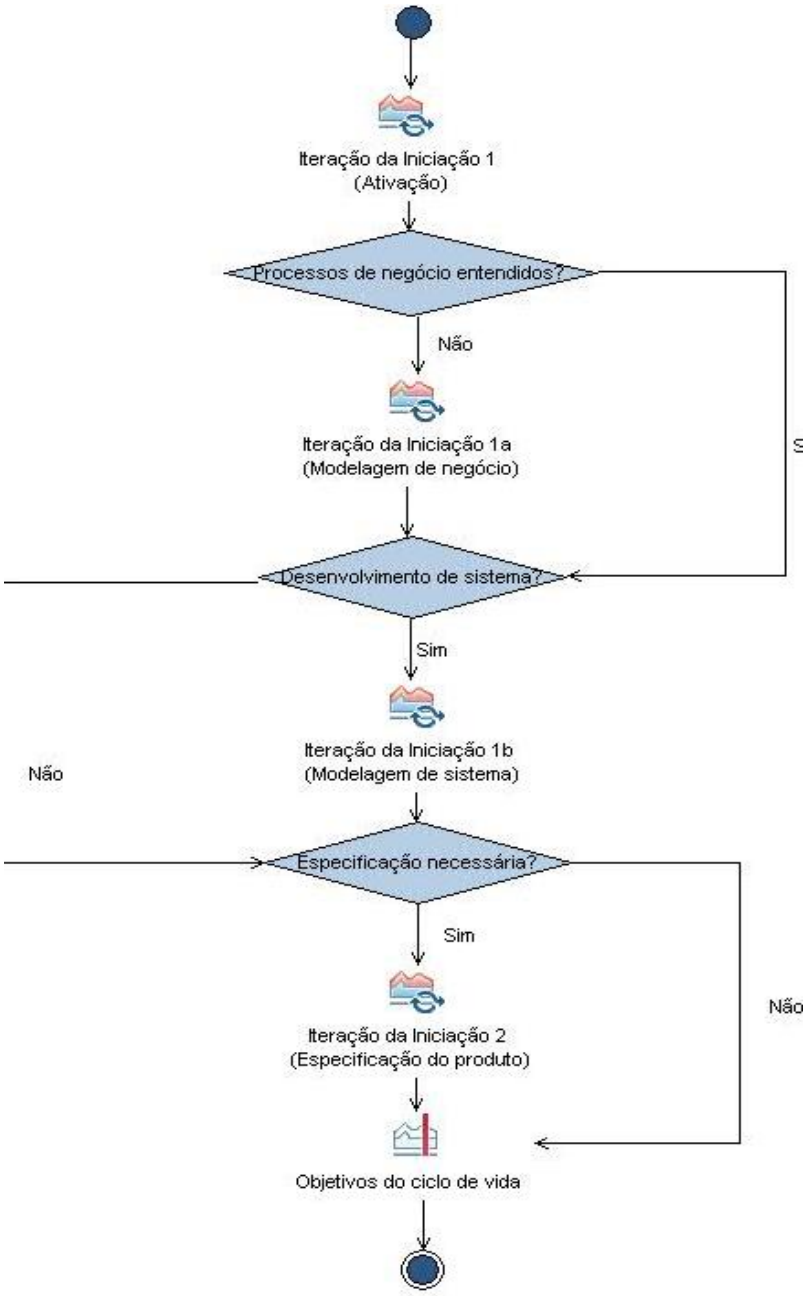


Figura 4 – Fluxo da fase de Iniciação

Fonte: PAULA FILHO, 2014

Elaboração: fase na qual se atinge uma arquitetura sólida e uma visão bem fundamentada dos riscos que permitirá o desenvolvimento estável e bem controlado do restante do projeto. O fluxo da fase de Elaboração está ilustrado na Figura 5.

Figura 5 – Fluxo da fase de Elaboração

Fonte: PAULA FILHO, 2014

Construção: fase na qual se completa o desenvolvimento de uma versão completamente operacional do produto que atende aos requisitos especificados. A fase de Construção tem seu fluxo descrito na Figura 6.

Figura 6 – Fluxo da fase de Construção

Fonte: PAULA FILHO, 2014

Transição: fase na qual o produto é colocado à disposição de uma comunidade de usuários para testes finais, treinamento e uso inicial. O fluxo da fase de Transição pode ser observado na Figura 7.

Figura 7 – Fluxo da fase de Transição

Fonte: PAULA FILHO, 2014

No Processo Praxis, os fluxos de trabalho se chamam *disciplinas*. Todos os processos, subprocessos e disciplinas são descritos por meio do *Eclipse Proccess Framework*.

O Eclipse Proccess Framework (EPF) é uma ferramenta desenvolvida pela Eclipse Foundation que "visa produzir uma estrutura de processo de engenharia de software personalizável, com exemplos e ferramentas, suportando uma ampla variedade de tipos de projetos e estilos de desenvolvimento" (ECLIPSE FOUNDATION, 2015).

O criador do processo Praxis utilizou o EPF para exemplificar e possibilitar um acesso rápido aos fluxos do processo, com diagramas de fluxo, divisão de tarefas e exemplos. O hipertexto gerado pelo EPF é complementar ao livro-texto (PAULA FILHO, 2009), e deve ser usado em conjunto.

A Seção de Banco de Dados do Tribunal Regional Eleitoral não tem nenhum padrão estabelecido para o processo de desenvolvimento e manutenção de software, embora essa seja uma de suas atribuições. A cultura da equipe é, até então, de projetos individuais, e desenvolvimento a partir de chamados de service-desk. Não há quaisquer mecanismos de controle, métodos de análise e projeto, para o desenvolvimento de software. Porém, alguns desenvolvedores demonstram o interesse em estabelecer um processo, e mudar essa cultura de projetos individuais para um ambiente de alta produtividade, com colaboração e proximidade maior com os usuários de software.

A Seção de Análise e Desenvolvimento tem promovido mudanças nesse sentido, com a adoção de algumas práticas do SCRUM, como a definição de sprints e reuniões diárias. Contudo, viu-se a necessidade da adoção de um processo que combine a colaboração e interação com usuários que as metodologias ágeis oferecem, com a formalidade, a documentação mais extensa e que abrange a complexidade dos processos envolvidos, encontrado no método clássico. Esse foi um fator que nos levou a decidir, nesse projeto, pela adoção experimental do Processo Praxis.

1.3. Mapeamento Objeto-Relacional

O Mapeamento Objeto-Relacional (*Object/Relational Mapping*, ORM) é um padrão de arquitetura para aplicações corporativas que visa diminuir a impedância entre os paradigmas de orientação a objetos e o esquema de entidade relacional ao criar uma camada intermediária entre memória e persistência. Segundo Martin Fowler, o ORM é "uma camada de software que separa os objetos na memória do banco de dados", com a responsabilidade de "transferir dados entre os dois e também isolá-los um do outro" (FOWLER, 2015).

O paradigma da orientação a objetos busca mapear e modelar os elementos de um sistema a ser desenvolvido da mesma forma como são percebidos no mundo real (CAMPOS E SANTUCI, 2009). Esses elementos são chamados objetos, que por sua vez são instância de uma classe, ou entidade, que possui atributos e métodos. Uma instância de um objeto pode atingir diversos estados, em sua interação e colaboração com outros objetos para um objetivo comum.

A programação orientada a objetos difere muito do esquema entidade relacional e precisamos pensar das duas maneiras para fazer um único sistema. Para representarmos as informações no banco, utilizamos tabelas e colunas. As tabelas geralmente possuem chave primária (PK) e podem ser relacionadas por meio da criação de chaves estrangeiras (FK) em outras tabelas. Quando trabalhamos com uma aplicação Java, seguimos o paradigma orientado a objetos, onde representamos nossas informações por meio de classes e atributos. Além disso, podemos utilizar também herança, composição para relacionar atributos, polimorfismo, enumerações, entre outros. Esse buraco entre esses dois paradigmas gera bastante trabalho: a todo momento devemos "transformar" objetos em registros e registros em objetos (CAELUM, 2014).

Enquanto o paradigma de orientação a objetos é predominante no desenvolvimento de software; na modelagem, gerenciamento e manutenção das bases de dados, na qual os dados serão criados/manipulados e recuperados, o modelo de relacional é a solução mais adotada (ARAUJO, 2013). No modelo relacional, os elementos do mundo real são representados como entidades no formato de tabelas de dados interconectadas por relacionamentos.

O paradigma orientado a objetos é baseado em princípios de engenharia de software comprovados. O paradigma relacional , no entanto, é baseado em princípios matemáticos comprovados. Porque os paradigmas subjacentes são diferentes, as duas tecnologias não funcionam perfeitamente em conjunto. A diferença de impedância se torna aparente quando você olha para a abordagem preferida para acessar: acessando objetos por meio de suas relações no paradigma orientado a objetos, enquanto junta as linhas de tabela no paradigma relacional. Esta diferença fundamental resulta em uma combinação não-ideal dos dois paradigmas (AMBLER, 2014).

Desenvolver camadas de acesso a dados pode ser muito trabalhoso, principalmente pela necessidade de se ter que escrever código SQL, além de se precisar lidar com a diferença de paradigmas entre o modelo de objetos e o relacional. Conhecer e utilizar um framework de ORM simplifica a implementação e pode minimizar o esforço de realizar tarefas que se repetem para cada classe persistente (ARAÚJO, 2013).

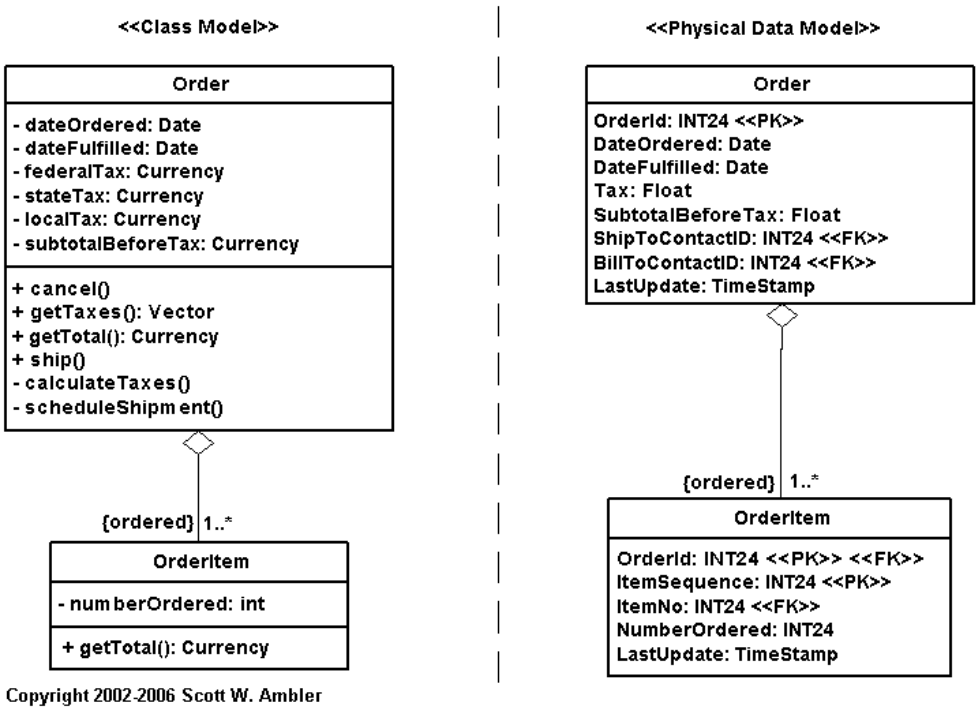


Figura 8 – Diferença entre modelos de classe e físico

Fonte: AMBLER, 2014

2. MATERIAIS, TÉCNICAS E MÉTODOS

Para execução das tarefas reportadas neste relatório de estágio foram utilizados diversos recursos tecnológicos e metodológicos. A seguir serão abordados os principais materiais, técnicas e métodos utilizadas para a realização destas tarefas. São abordados primeiramente as duas primeiras fases do Processo Praxis, que envolvem a documentação do problema e da solução: *Iniciação: Requisitos, Contexto e Escopo* e *Elaboração: Desenho e Casos de Uso*, e logo após as ferramentas e roteiro do desenvolvimento, nas subseções: *Gerenciamento de Versão com Subversion, Plataforma Java EE, Controle de Dependências com Maven, Ambiente de Desenvolvimento Netbeans* e *Desenvolvimento*.

2.1. Iniciação: Requisitos, Contexto e Escopo

A Iniciação é a fase que procura atingir o consenso entre as partes interessadas sobre o escopo e objetivos do projeto (PAULA FILHO, 2014). Ela é formada por duas iterações: **Ativação** (que pode ser estendida em Modelagem de Negócio e Modelagem de Sistema) e **Especificação do Produto** (opcional), conforme podemos ver na Figura 9.

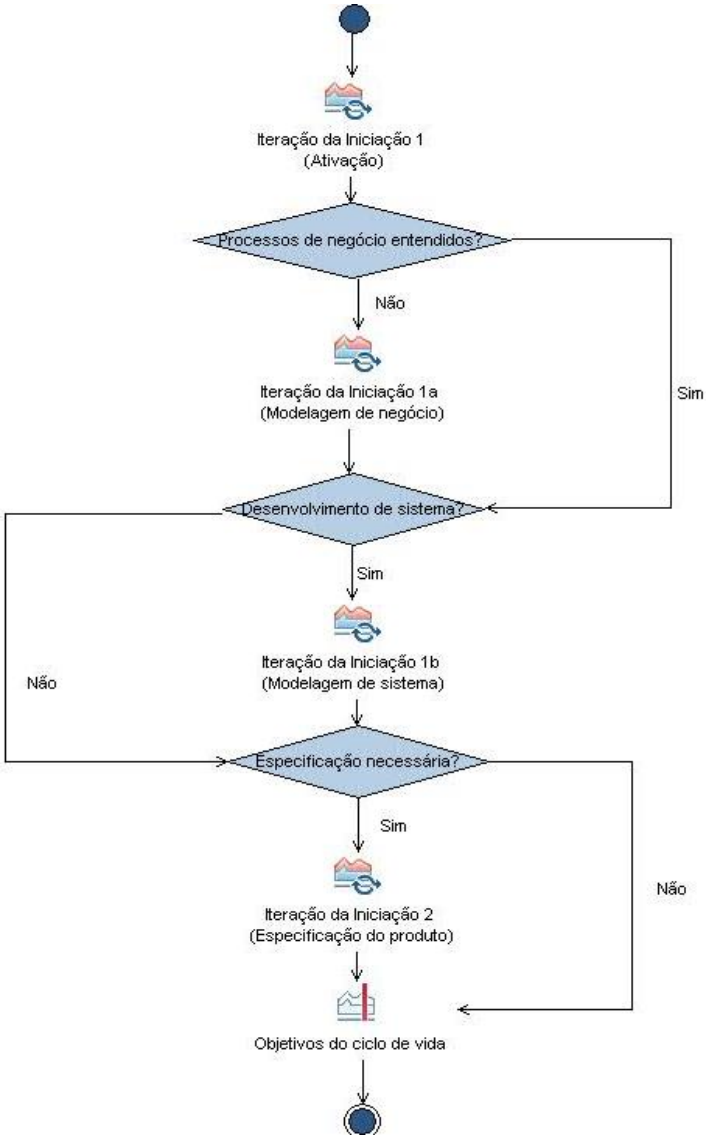


Figura 9 – Fluxo da fase de Iniciação

Fonte: PAULA FILHO, 2014

A iteração de Ativação reúne principalmente as atividades relacionadas ao levantamento de requisitos, escopo e funcionalidades esperadas. Podemos ver o fluxo dessa iteração na Figura 10. Seu principal fluxo de atividades é a Identificação dos Requisitos.

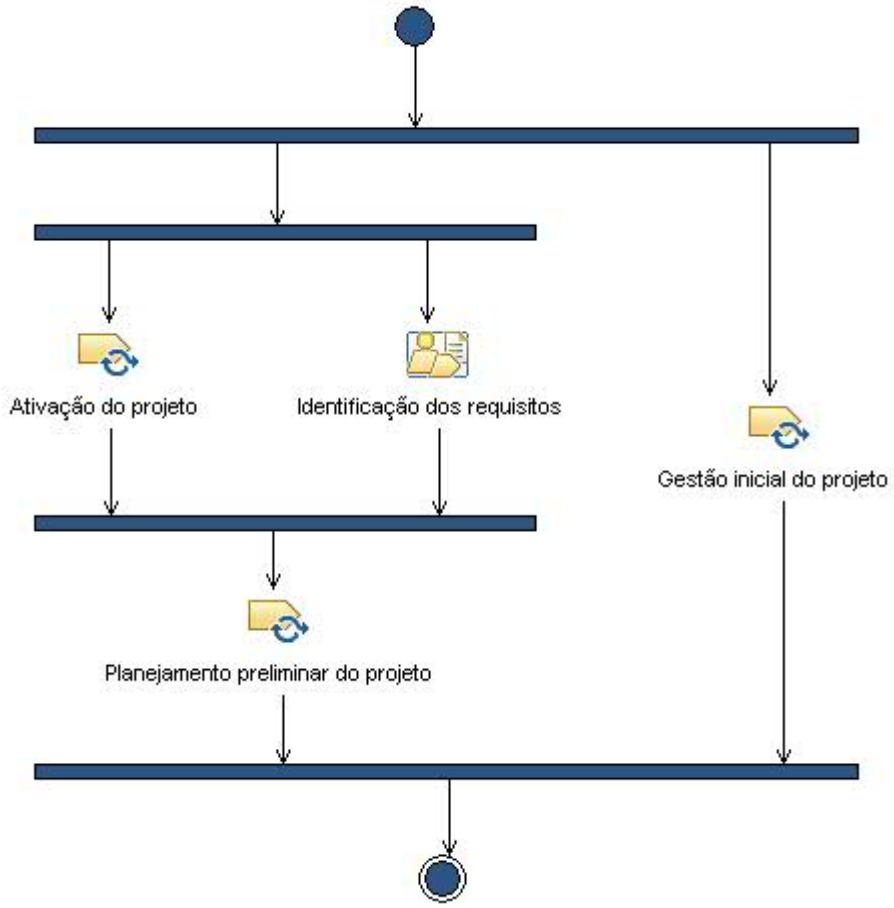


Figura 10 – Ativação do Projeto

Fonte: PAULA FILHO, 2014

A atividade de Identificação de Requisitos, conforme a Figura 11, parte do fluxo de Iniciação, compreende as tarefas: **Determinação do contexto**, **Definição do escopo, Definição dos requisitos e Priorização de requisitos**, descritas a seguir:

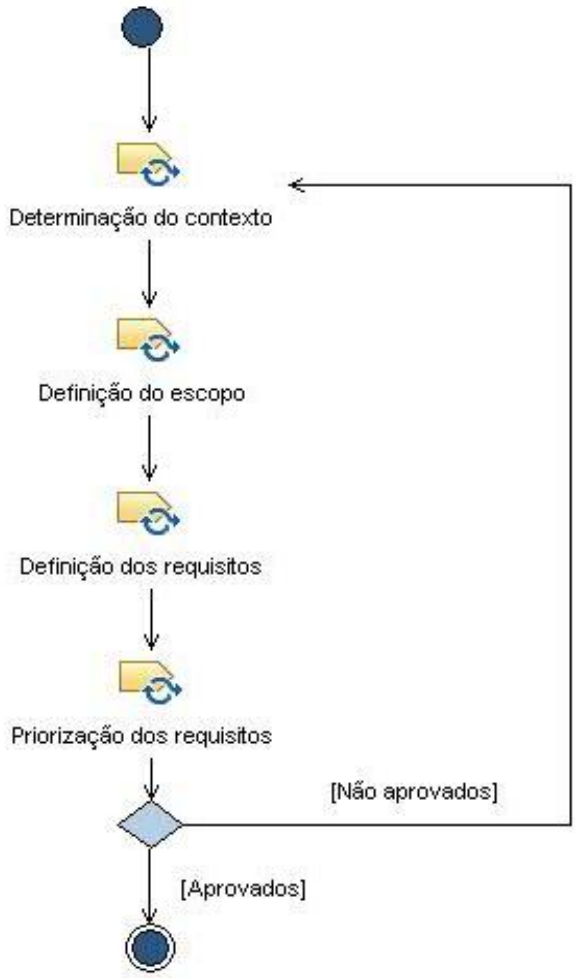


Figura 11 – Fluxo de atividades de Identificação de Requisitos

Fonte: PAULA FILHO, 2009

Determinação do Contexto: Levantamento dos aspectos dos processos de negócio ou de um sistema maior que sejam relevantes para a determinação dos requisitos da aplicação.

Definição do Escopo: Delimitação dos problemas que a aplicação se propõe a resolver.

Definição dos Requisitos: Produção de uma lista de todos os requisitos funcionais e não funcionais.

Priorização de Requisitos: Determinação das prioridades relativas dos requisitos.

A disciplina de requisitos reúne as atividades que visam obter o enunciado completo, claro e preciso dos requisitos de um produto de software. Esses requisitos devem ser levantados pela equipe do projeto, em conjunto com representantes do cliente, usuários e, possivelmente, especialistas da área de aplicação (PAULA FILHO, 2009).

O resultado da fase de Identificação de Requisitos é um anexo ao Modelo do Problema, chamado de Descrição Geral do Contexto, como visto na Figura 12.

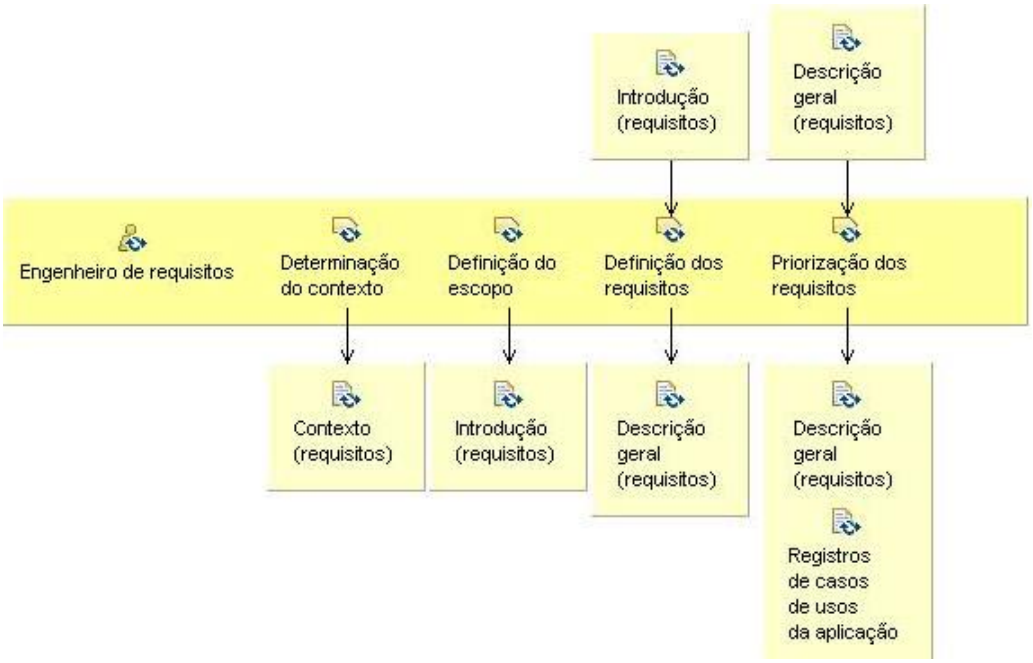


Figura 12 – Artefatos gerados pela Identificação de Requisitos

Fonte: PAULA FILHO, 2014

O **Modelo do Problema** no processo Praxis é o principal artefato de descrição dos requisitos de um projeto de software. As características que devem ser descritas nesse modelo incluem informações sobre a **funcionalidade, interfaces externas, desempenho e restrições.**

2.2. Elaboração: Desenho e Casos de Uso

Elaboração é a fase na qual se atinge uma arquitetura sólida e uma visão bem fundamentada dos riscos, que permitirá o desenvolvimento estável e bem controlado do restante do projeto (PAULA FILHO, 2014). Ela é composta por *n* iterações, divididas em pacotes ou entregas.

Figura 13 – Fluxo da fase de Elaboração

Fonte: PAULA FILHO, 2014

Cada iteração de elaboração é composta pelas tarefas **Abertura da Iteração**, **Desenvolvimento de Caso de Uso**, **Suporte ao Desenvolvimento** e **Fechamento da Iteração** (PAULA FILHO, 2014).

No presente trabalho, o resultado dessa fase foi a elaboração de diagramas de classe, desenho arquitetônico interno, diagramas de caso de uso e descrição dos casos de uso.

Para a a criação dos diagramas de classe, arquitetura e casos de uso, foi necessária a escolha de um software de modelagem. O software atualmente em uso pela equipe é o Sparx Systems Enterprise Architect, "uma plataforma visual para projetar e construir sistemas de software, para a modelagem de processos de negócios, e para mais generalizada fins de modelagem" (SPARX SYSTEMS, 2014).

De acordo com Wilson de Pádua Paula Filho, "o desenho arquitetônico interno é a definição, em nível estratégico, do desenho interno, compreendendo os principais aspectos de organização, estrutura e colaborações das classes que serão usadas para implementar a solução" (PAULA FILHO, 2014). Já os casos de uso representam funções completas do produto, e devem "descrever a funcionalidade

completa do produto, sem lacunas e sem superposições". Pode-se dizer que cada caso de uso representa uma "fatia de funcionalidade" (PAULA FILHO, 2009).

Os casos de uso formam o ponto de partida para:

- determinar classes, atributos e operações, na disciplina de **Análise**;
- desenhar o uso detalhado do produto, na disciplina de **Desenho**;
- especificar os testes de aceitação, na disciplina de **Testes**;
- escrever os procedimentos, na disciplina de **Implementação**.

A lista de casos de uso da aplicação definida na fase de Iniciação é detalhada na fase atual, na **Especificação do Problema do Caso de Uso**. No desenvolvimento da solução do caso de uso, desenhamos o diagrama de casos de uso e descrevemos detalhadamente.

2.3. Gerenciamento de Versão com *Subversion*

Subversion é um "sistema de controle de versão de código aberto" (APACHE SUBVERSION, 2015). Isto é, o Subversion gerencia arquivos e diretórios, e as modificações feitas neles ao longo do tempo. O Subversion pode funcionar em rede, o que lhe possibilita ser usado por diversas pessoas em diferentes computadores. O Subversion é um sistema centralizado de compartilhamento de informação. Em seu núcleo está um repositório, que armazena informação numa árvore de arquivos e diretórios. Um numero variado de clientes se conecta ao repositório e então lê ou escreve nestes arquivos, e os grava no repositório, tornando a informação disponível para outros clientes (COLLINS-SUSSMAN, FITZPATRICK, PILATO, 2007).

O Subversion é o sistema de controle de versão utilizado no TRE-MT. Foi cogitada a ideia de uma migração para o GIT, sistema de controle de versão *opensource* baseado em operações locais (GITHUB, 2014), mas a equipe chegou a conclusão de que seria necessário mais estudos e planejamento para essa mudança. O Subversion tem como vantagem a sua facilidade de configuração e a desvantagem é a sua arquitetura centralizada dependente de um servidor central. No entanto, ele atende as necessidades do tribunal.

2.4. Plataforma Java EE

Java é uma linguagem de programação e plataforma computacional lançada pela Sun Microsystems em 1995, com o objetivo de resolver os principais problemas

em linguagens de programação na década de 90, tais como gerenciamento de ponteiros, memória, falta de bibliotecas, e a necessidade de reescrever parte do código ao mudar de sistema operacional (CAELUM, 2015). Hoje é uma linguagem amplamente usada na Justiça Eleitoral, tanto para desenvolvimento *desktop* quanto *web*.

O Java EE (Java Enterprise Edition) consiste de uma série de especificações bem detalhadas, dando uma receita de como devem ser implementados os serviços de infraestrutura de uma aplicação *web*, tais como: persistência em banco de dados, transação, acesso remoto, *webservices*, gerenciamento de *threads*, gerenciamento de conexões HTTP, cache de objetos, gerenciamento da sessão *web*, balanceamento de carga, entre outros (CAELUM, 2015).

No projeto, preferimos a especificação Java EE sempre que possível, com o objetivo de haver o mínimo de dependência de bibliotecas de terceiros. Foi utilizado o JSF como *framework* MVC, CDI para injeções de dependências, EJB para desenvolvimento das classes de negócio e JPA (Hibernate) para persistência.

O sistema foi implementado com uma Aplicação Corporativa (*Enterprise Application*). Um aplicativo corporativo é uma coleção de aplicativos *web* e módulos EJB (Enterprise Java Beans) que estão configurados para trabalhar em conjunto, quando implantado em um servidor de aplicativos Java EE. O aplicativo corporativo contém informações sobre como os módulos devem trabalhar em conjunto (ORACLE, 2014). Nosso projeto tem dois módulos EJB, que contém as classes de negócio e entidades, e um módulo WEB, com os componentes necessários para a criação de páginas.

2.5. Controle de Dependências com Maven

O Maven (MAVEN, 2015) é uma ferramenta de gerenciamento de projetos, construção e implantação de projetos, que auxilia no processo de gerenciamento de dependências, *build*, geração de relatórios e de documentação (CAVALCANTI, 2008).

Cada um dos módulos da Aplicação Corporativa tem um arquivo de configuração do Maven, *pom.xml*, onde são especificadas as dependências do

módulo. O Maven encarrega-se de gerenciar estas dependências, obtendo a partir de um repositório remoto e realizando a construção dos módulos.

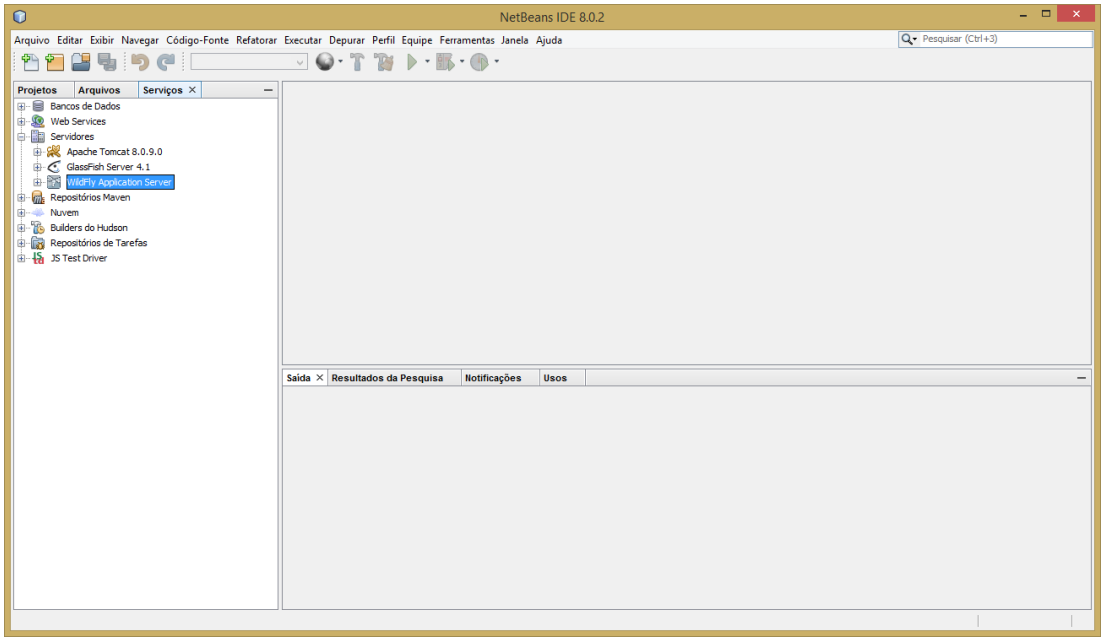
2.6. Ambiente de Desenvolvimento Netbeans

O NetBeans IDE é um ambiente de desenvolvimento - uma ferramenta para programadores, que permite escrever, compilar, depurar e instalar programas. O IDE é completamente escrito em Java, mas pode suportar qualquer linguagem de programação (NETBEANS, 2015).

O Netbeans foi escolhido como IDE por diversos motivos, como o suporte a linguagem Java, a disponibilidade de *plugins*, a integração com o Java EE, Maven e o servidor de aplicações WildFly 8.0, o mesmo utilizado em produção na intranet do TRE-MT.

2.7. Desenvolvimento

Para iniciar o desenvolvimento, primeiro foi necessária a instalação e configuração do ambiente de desenvolvimento Netbeans 8.0.2, o gerenciador de dependências Apache Maven e o servidor de aplicações Wildfly 8.0, e a integração



entre essas ferramentas através de plugins do Netbeans.

Figura 14 – Netbeans com *plugin* do servidor WIldfly

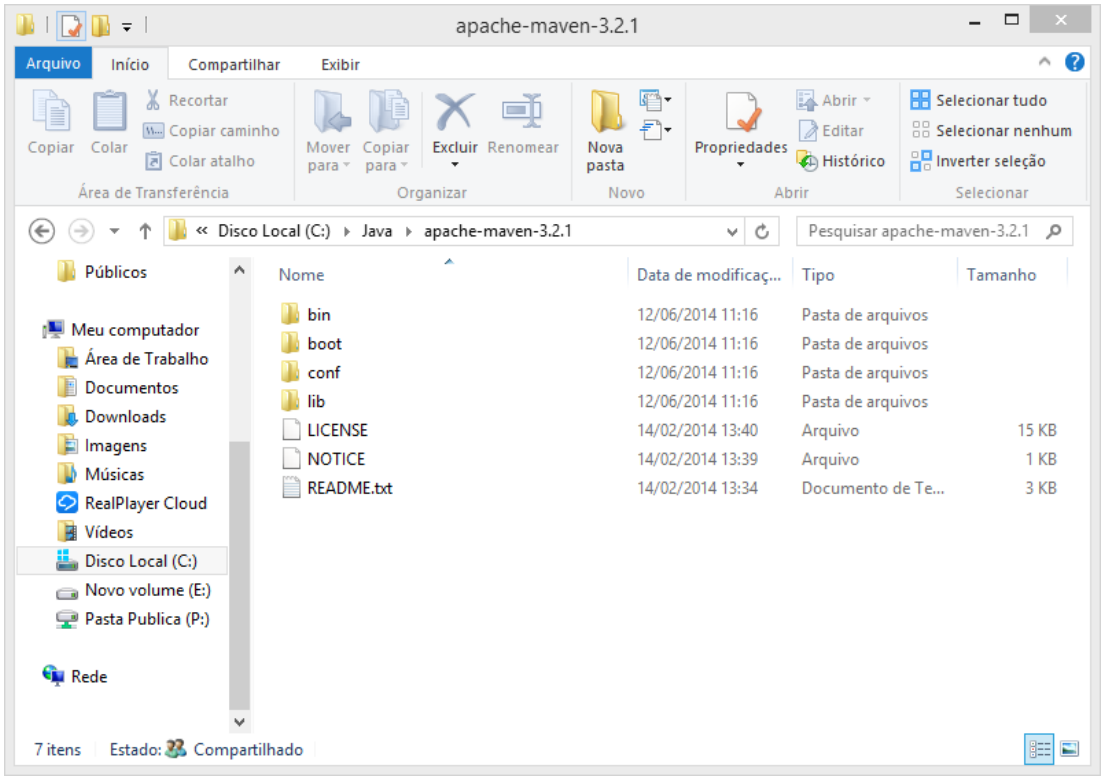


Figura 15 – Diretório de instalação do Apache Maven

Após a instalação e configuração desses componentes, utilizando o Netbeans foi aberto um projeto enterprise pré-existente, feito com base em uma aplicação anterior, já com os módulos previamente criados. Os módulos são *sgp-ear* (aplicação enterprise), *sgp-ejb* (aplicação EJB), *sgp-common* (aplicação JAR) e *sgp-web* (aplicação Web), como na figura a seguir.

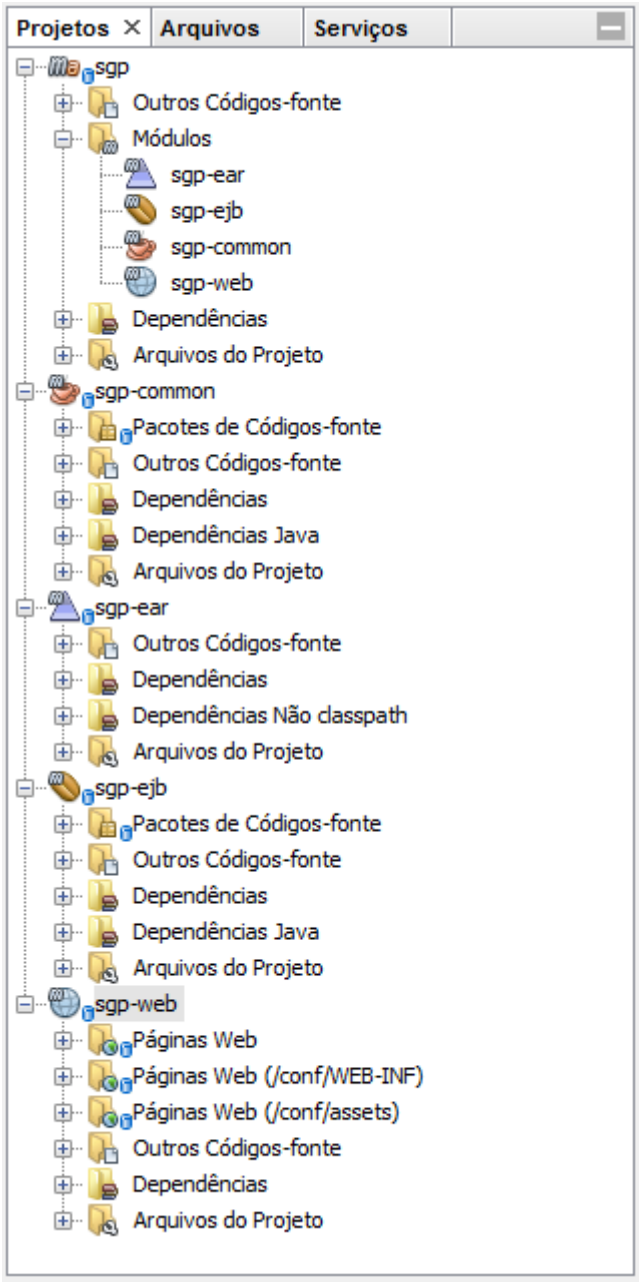


Figura 16 – Módulos da aplicação enterprise SGP

Para configurar as dependências do módulo, entre si e em relação à bibliotecas externas, é necessária a edição do arquivo *pom.xml*. Esse arquivo deve ficar na raiz da pasta do projeto (Arquivos do Projeto).

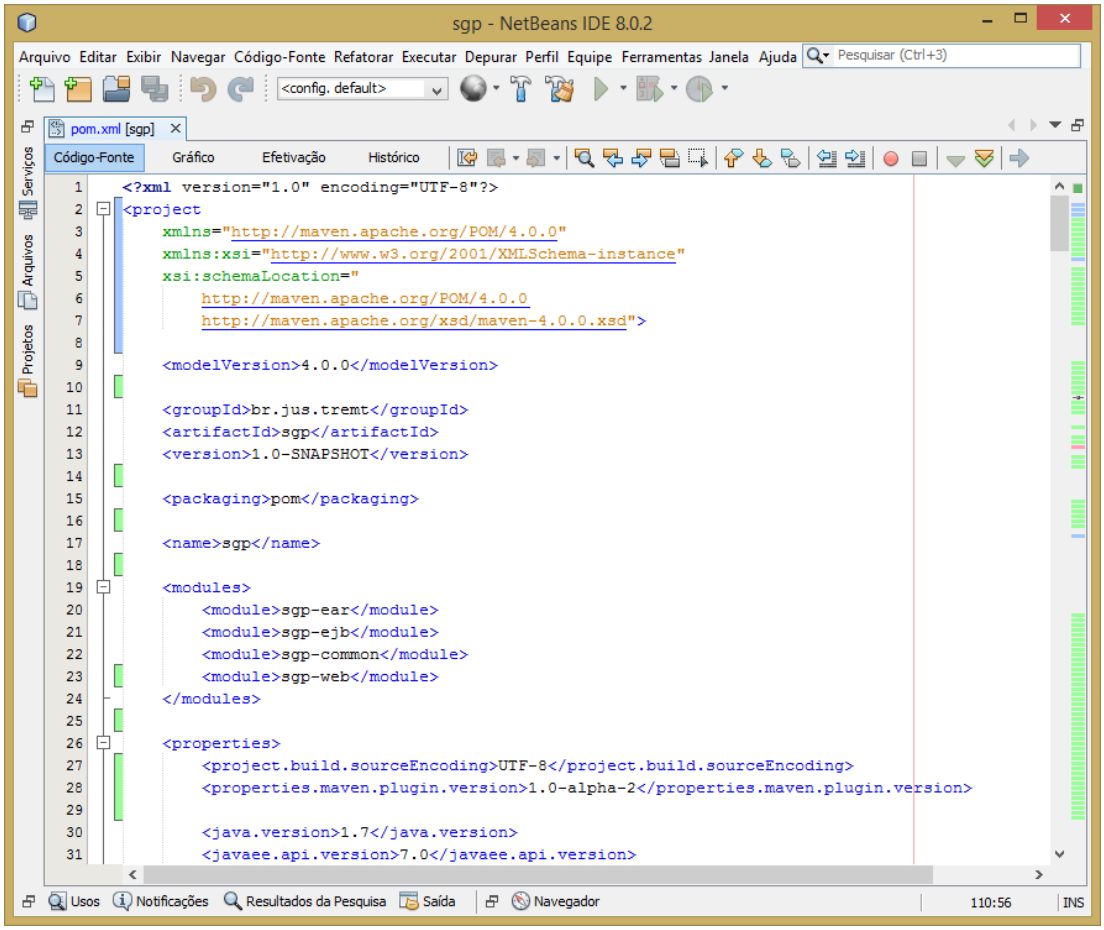


Figura 17 – Arquivo de configuração do projeto Maven

No projeto principal, são definidas configurações globais, tais como os módulos, a versão do projeto, e a versão das bibliotecas utilizadas, os perfis de execução (produção, desenvolvimento), os repositórios de onde as bibliotecas serão obtidas, e definições acerca da construção do projeto. Na seção *<dependencies>* são definidas as bibliotecas necessárias, e é a seção que mais é alterada ao longo do desenvolvimento do projeto.

Em cada módulo é necessária a definição de suas dependências internas, sem a necessidade de determinar a versão (que é obtida do projeto principal). Caso haja dependências entre os módulos, também devem ser definidos. Na figura 18 temos o arquivo *pom.xml* do módulo sgp-ejb. Note que ele traz como dependência o módulo sgp-common, no qual se encontram as entidades de negócio.

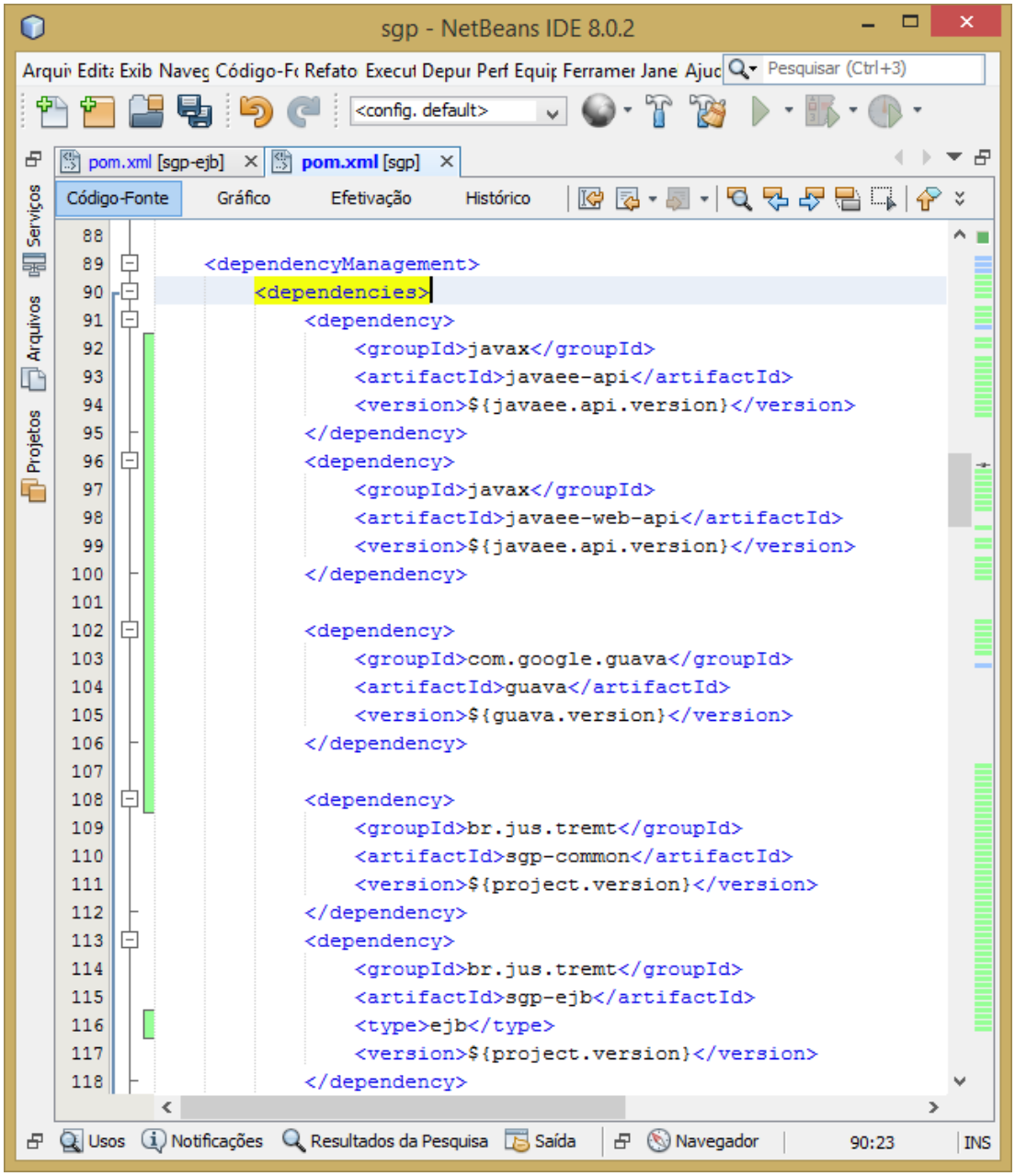


Figura 18 – Seção de dependências da configuração do projeto Maven

Uma das dependências mais importantes é o pacote *br.jus.tre-mt.reuse*. Esse pacote contém um conjunto de interfaces, tipos e *helpers*, desenvolvidos pela equipe, para auxiliar a criação de módulos MVC.

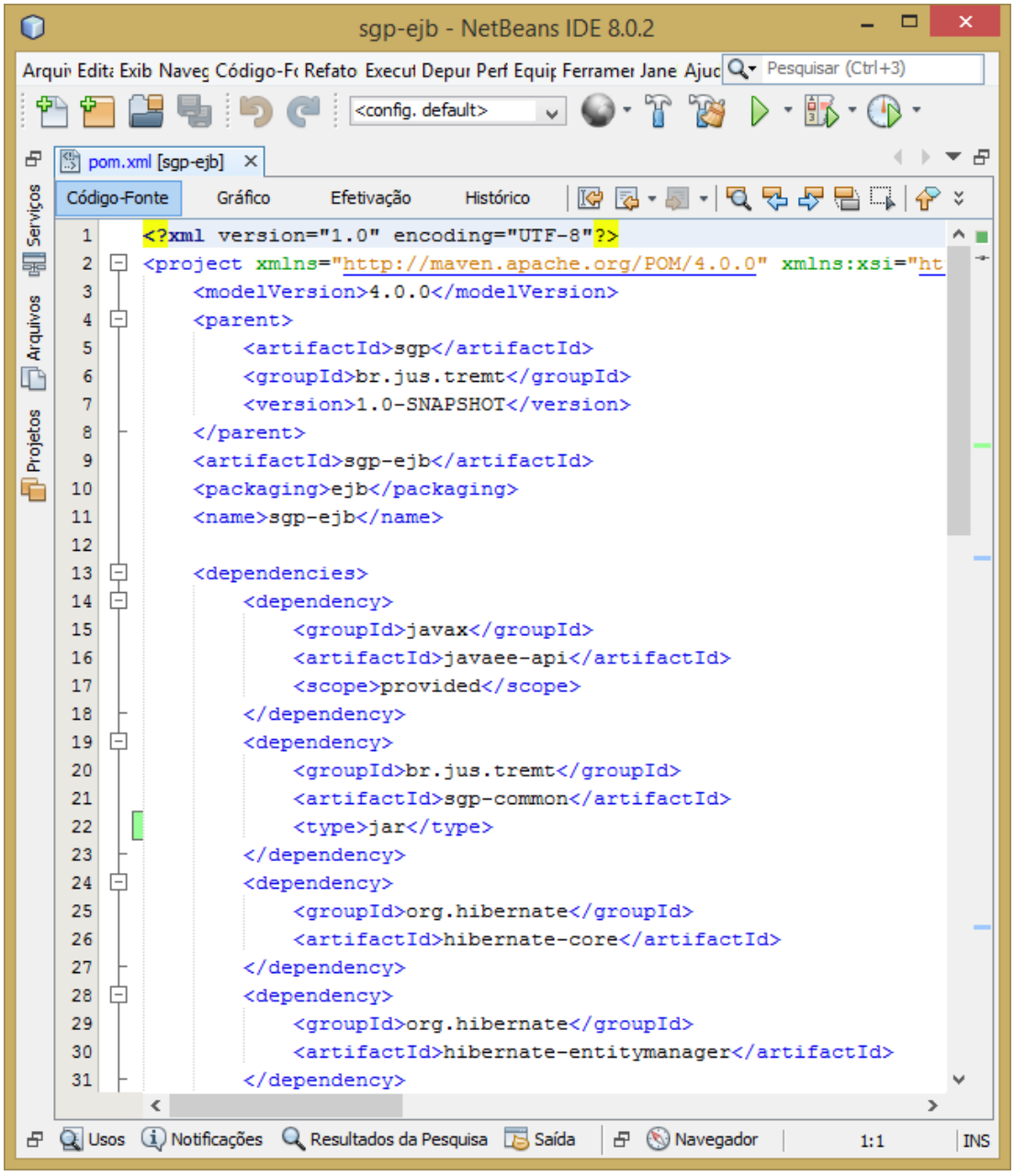


Figura 19 – Configuração do módulo do projeto Maven

Inicialmente é necessária apenas uma configuração básica. A IDE auxiliará a inclusão de mais bibliotecas, conforme necessário, e as dependências serão revisadas regularmente ao longo do desenvolvimento.

O mapeamento das classes de negócio, feito no diagrama de classes, agora deve ser codificado. O Enterprise Architect auxilia nesse processo ao gerar códigos-fonte relacionados a cada uma das classes, nos poupando de boa parte do trabalho repetitivo. Resta após isso o mapeamento objeto-relacional.

O mapeamento das entidades foi feito utilizando o padrão Java Persistence API (JPA), na implementação Hibernate. O JPA é uma especificação Java EE criada com inspiração no Hibernate, cuja implementação abstrai o seu código SQL, toda a camada JDBC e o SQL será gerado em tempo de execução. É uma implementação de Mapeamento Objeto-Relacional.

As entidades são mapeadas com anotações, como @Entity, @Table, @Column, @Id, @EmbeddedId @Temporal e @Enum. Os relacionamentos são mapeados com anotações como @JoinColumn, @OneToOne, @ManyToTone, @OneToMany e @ManyToMany. Cada anotação tem propriedades a serem definidas, que auxiliam na criação das consultas e operações SQL.

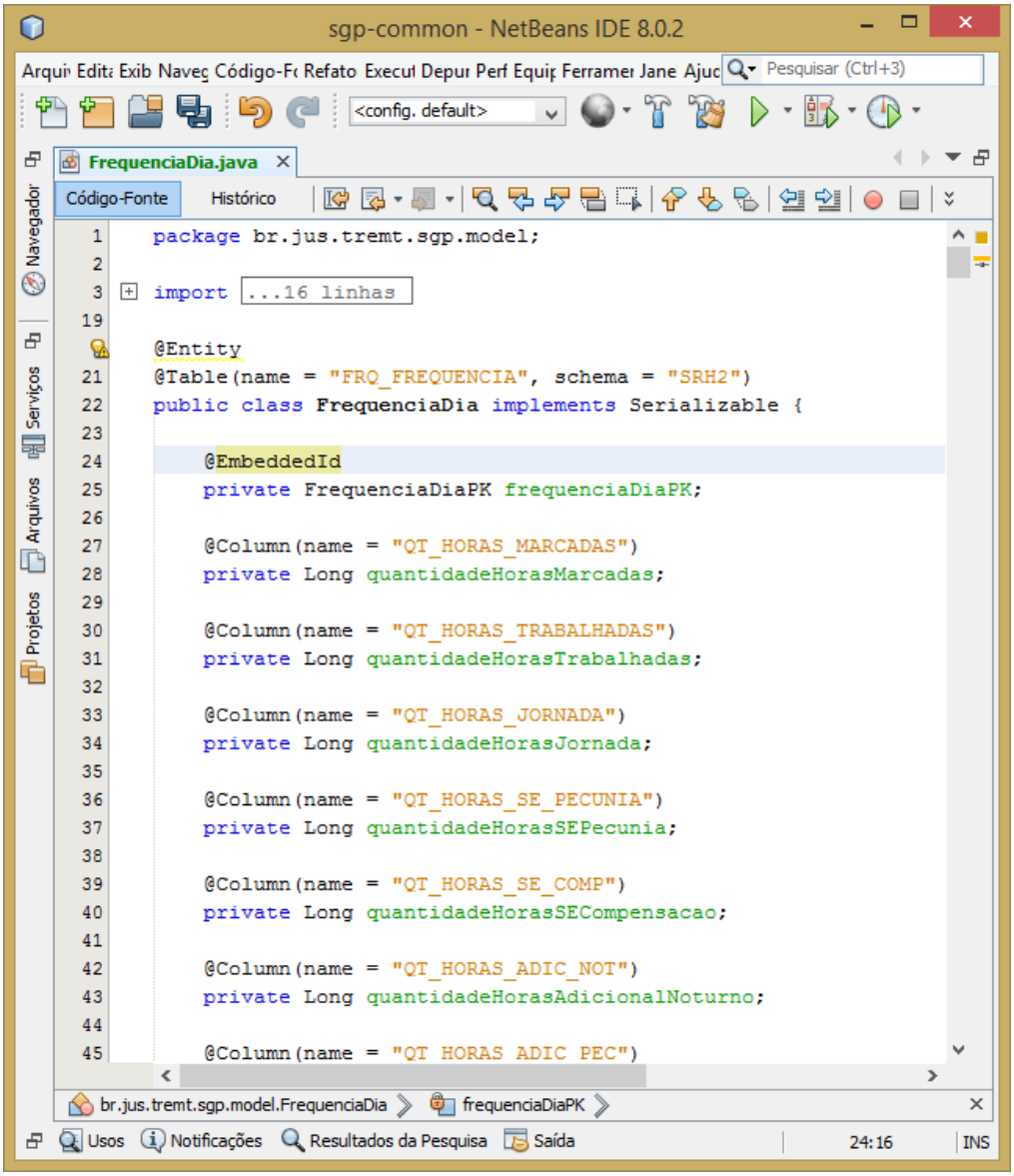


Figura 20 – Anotações do JPA em uma entidade mapeada

Assim como a configuração de dependências, o mapeamento é revisado ao longo do desenvolvimento. É importante também configurar o arquivo *standalone.xml* do servidor, no qual é configurado a conexão com o banco de dados Oracle, o arquivo *persistence.xml* (sgp-ejb), no qual é configurada a unidade de persistência, com a qual será instanciado o *entityManager*, o auxiliar do JPA para as operações necessárias.

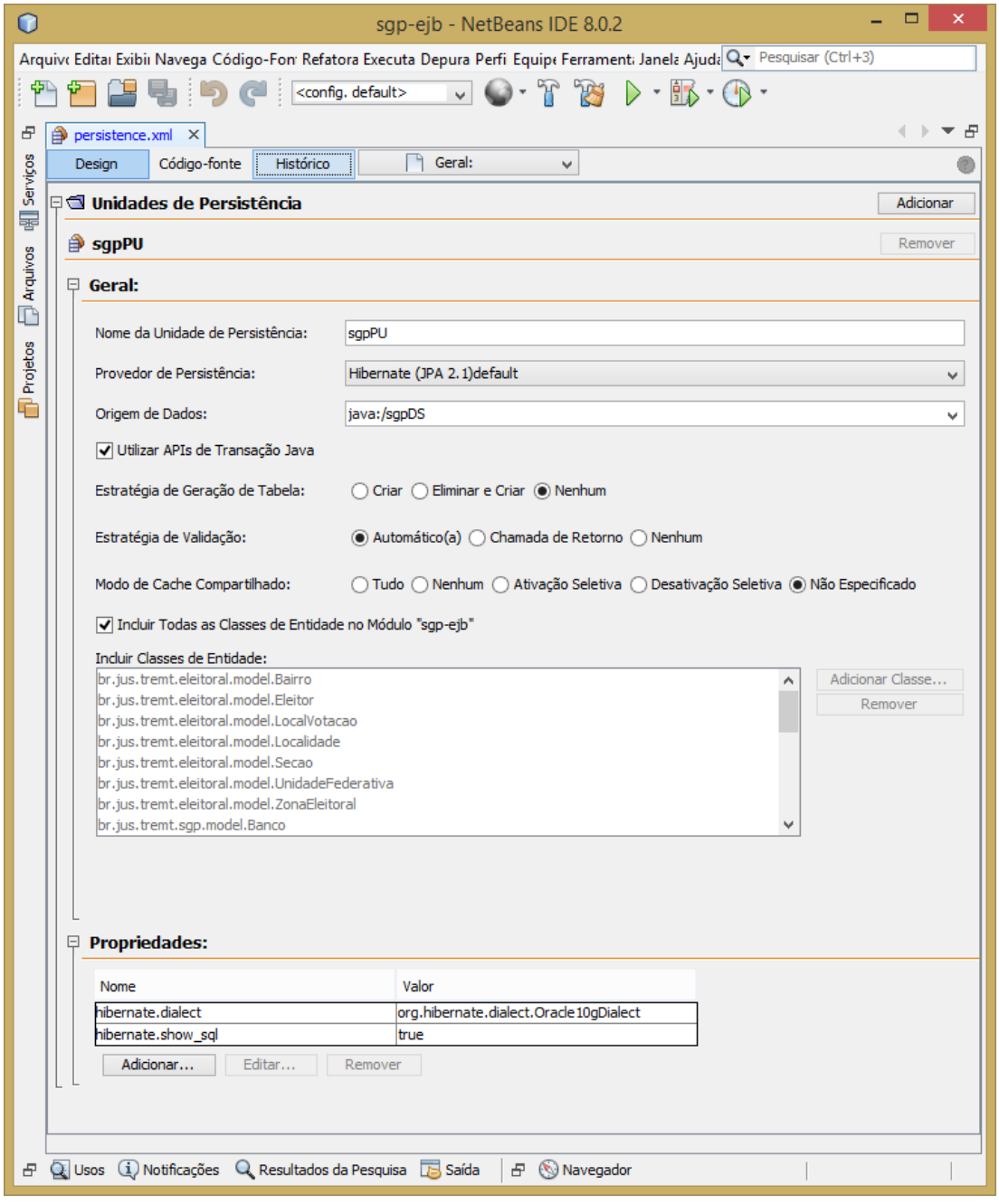


Figura 21 – Configuração da Unidade de Persistência

O próximo passo do desenvolvimento foi a criação das classes EJB, que são as funcionalidades em si. Os pacotes foram divididos tendo como referência os casos de uso. Para cada pacote de funcionalidade, seguindo a arquitetura definida no modelo arquitetural interno, temos vários *beans* que implementam interfaces ou herdam classes do *micro-framework*, sendo elas classes de controle (*Controller*), negócio (*Business*), ou repositório (*Repository*).

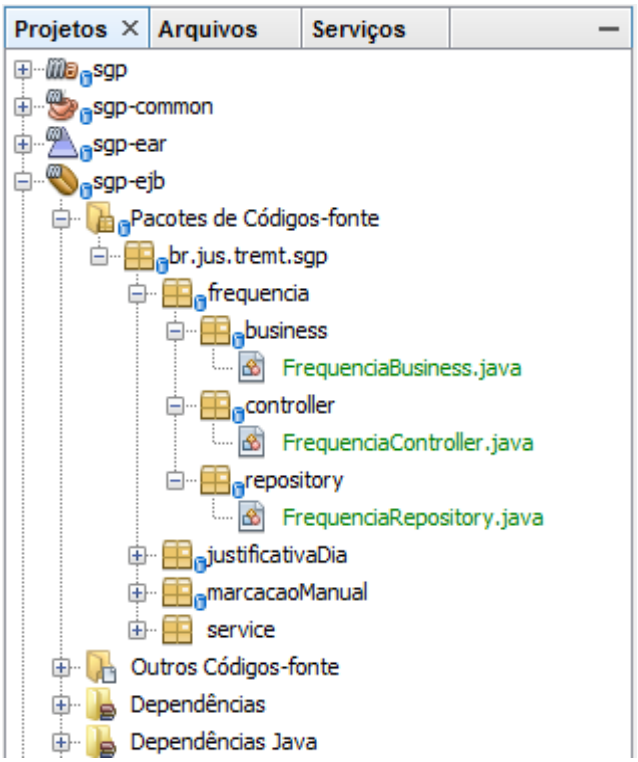


Figura 22 – Estrutura de um pacote de funcionalidade

As classes de controle implementam os métodos necessários para receber e responder requisições da visão, e para gerenciar as injeções de dependência e o escopo.

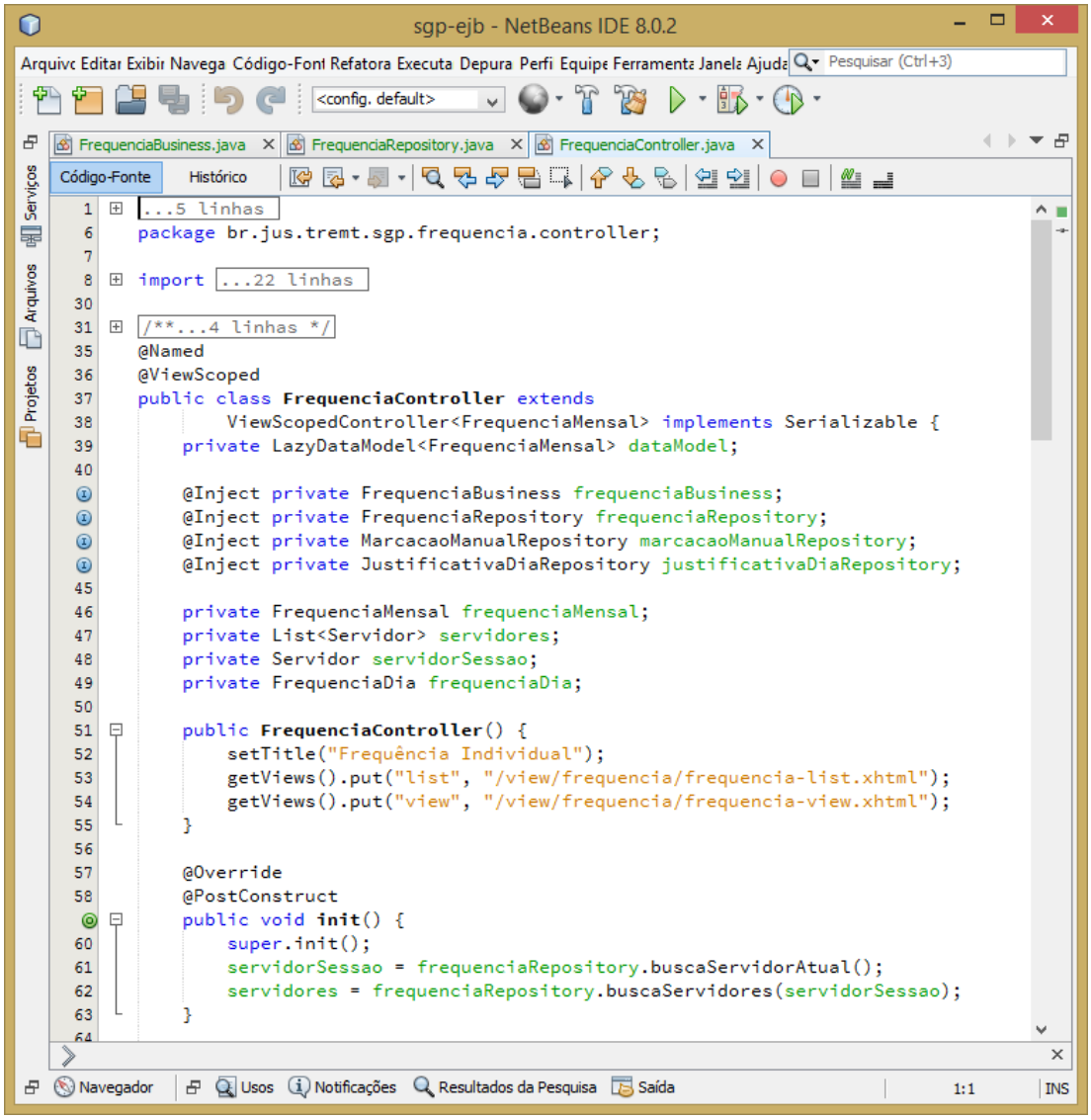


Figura 23 – Classe de controle

As classes de negócio contêm as validações necessárias e a manipulação de dados de acordo com as regras de negócio definidas.

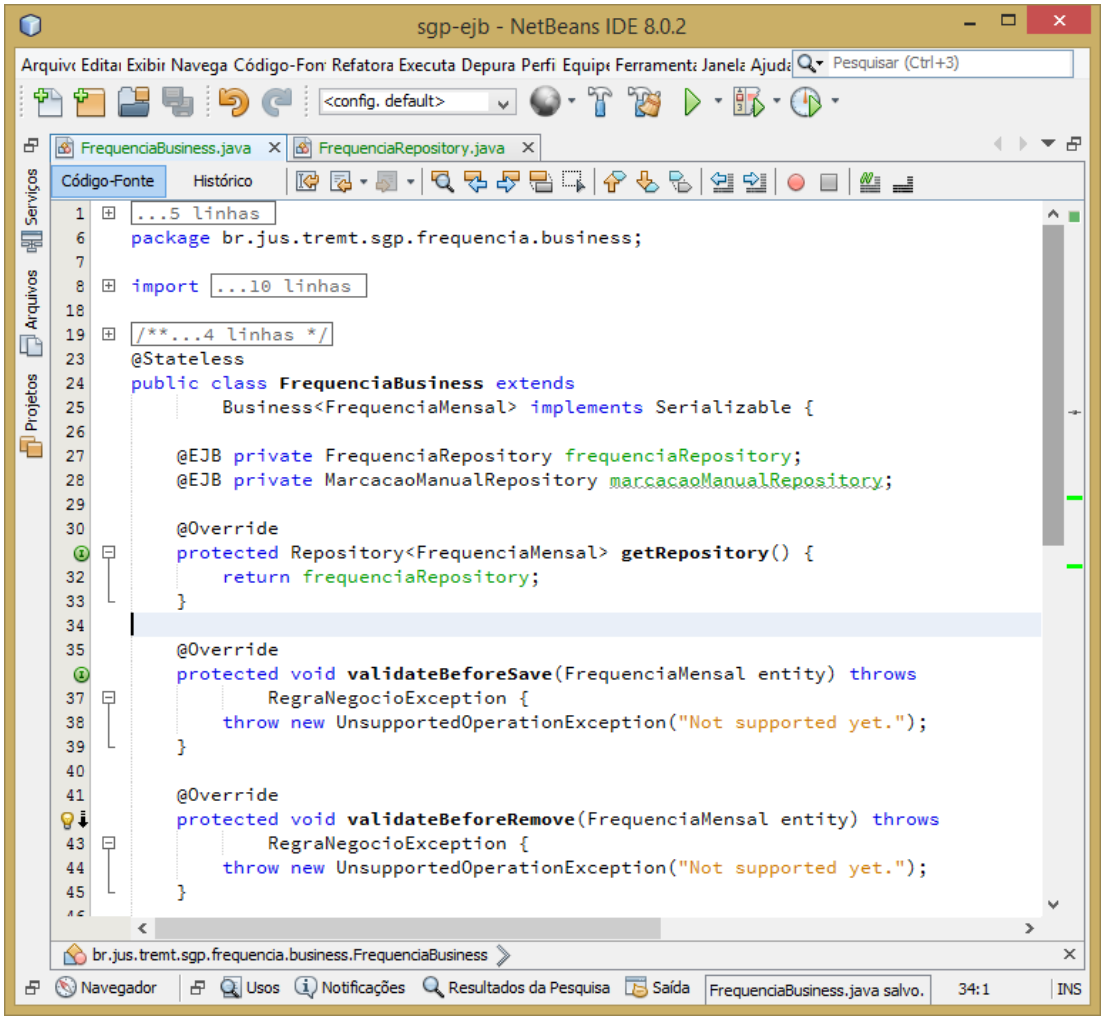


Figura 24 – Classe de negócio

As classes de repositório agrupam as consultas e a persistência, sendo também uma abstração para métodos do JPA.

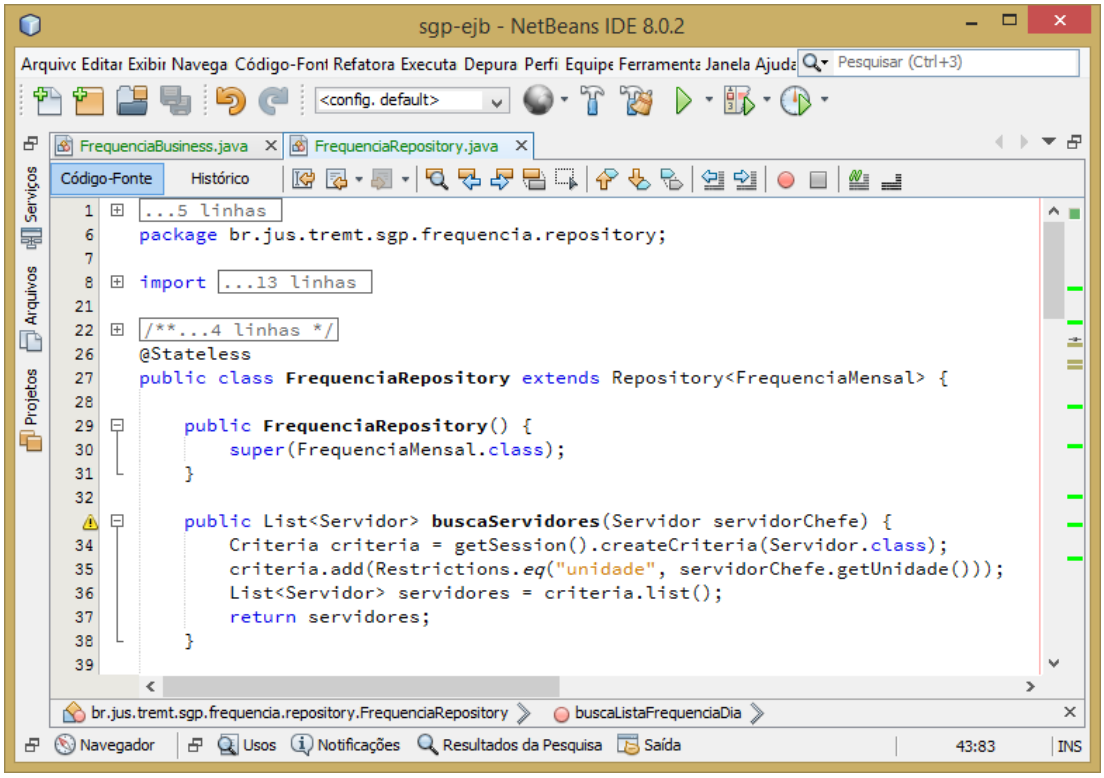


Figura 25 – Classe de repositório

Para a injeção de dependências em tempo de execução, utilizamos o padrão *Context and Dependency Injection* (CDI), tendo o Weld como implementação. Weld é a implementação de referência (RI) do CDI, que é o padrão do Java EE para injeção de dependência e gerenciamento de ciclo de vida contextual que integra de forma limpa e simples com a plataforma Java EE. Qualquer servidor de aplicação Java EE 6 fornece suporte para o CDI (DEMEDIA, 2014).

Para utilizar o Weld foi necessário configurar o arquivo *beans.xml*. Isso é necessario para que as classes sejam devidamente mapeadas e instanciadas sempre que requisitadas.

Já a integração do CDI, para definir as injeções e os contextos de cada classe, é feita a por meio de *Java Annotations*, metadados que são escritos sobre linhas de código com informações ao compilador, processadas em tempo de execução. Com o CDI devidamente configurado, ao longo da criação das classes EJB utilizamos anotações como @Named, @Stateless, @Inject e @EJB para gerenciar as

dependências, e @ViewScoped para gerenciar o escopo em que os dados permanecerão em tempo de execução.

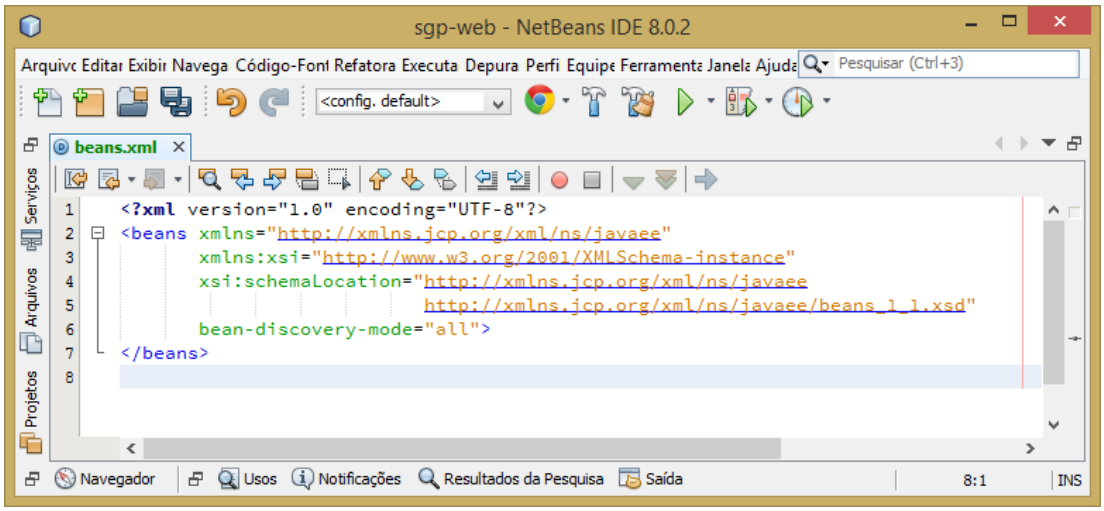


Figura 26 – Arquivo de configuração do CDI

A anotação @Named transforma a classe em um ManagedBean e pode ser acessada utilizando EL (Expression Language) em qualquer uma das visões (DEVMEDIA, 2014). A anotação @ViewScoped informa que os dados permanecerão disponíveis em tempo de execução apenas enquanto uma determinada visão estiver aberta. Para cada visão, uma nova instância será criada.

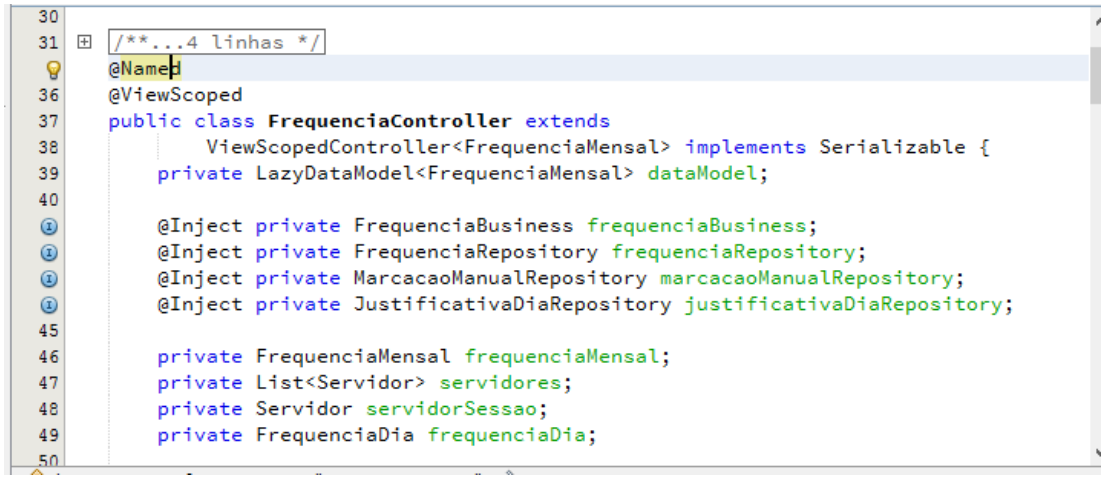


Figura 27 – Anotações de injeção do CDI

A anotação @Inject e @EJB são utilizadas para injeção de dependências nos atributos de um *bean*. As anotações @Stateless, @Stateful e @Singleton dizem respeito ao ciclo de vida de uma *session bean*, uma classe onde é implementada regras de negócio (K19, 2014).

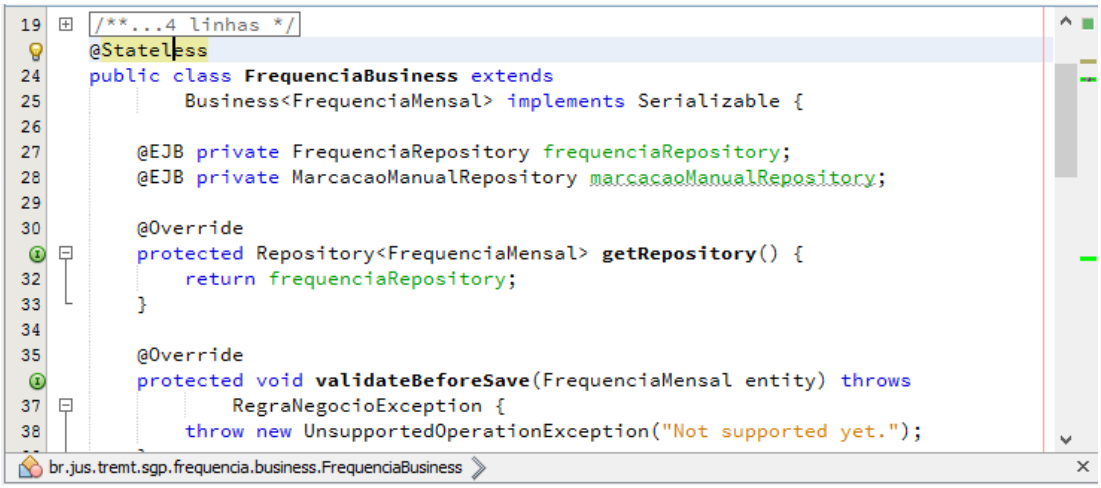


Figura 28 – Anotações de Session Beans do CDI

Após a implementação das regras de negócio nas classes EJB, iniciamos o desenvolvimento do módulo Web. Para isso, utilizamos o JSF (Java Server Faces) para a criação de páginas e componentes, e o PrettyFaces para configuração de URLs amigáveis.

O JavaServer Faces (JSF) é um framework de interface de usuário (IU) para aplicações Java Web, projetado para facilitar significativamente a trabalhosa tarefa de escrever e manter os aplicações que são executadas em um servidor de aplicações Java e renderizar as IUs de volta a um cliente de destino (NETBEANS, 2015). Para utilizar o JSF, é necessário criar o arquivo de configuração *faces-config.xml*, que contém definições sobre internacionalização, mensagens e constantes, conforme a Figura 29.

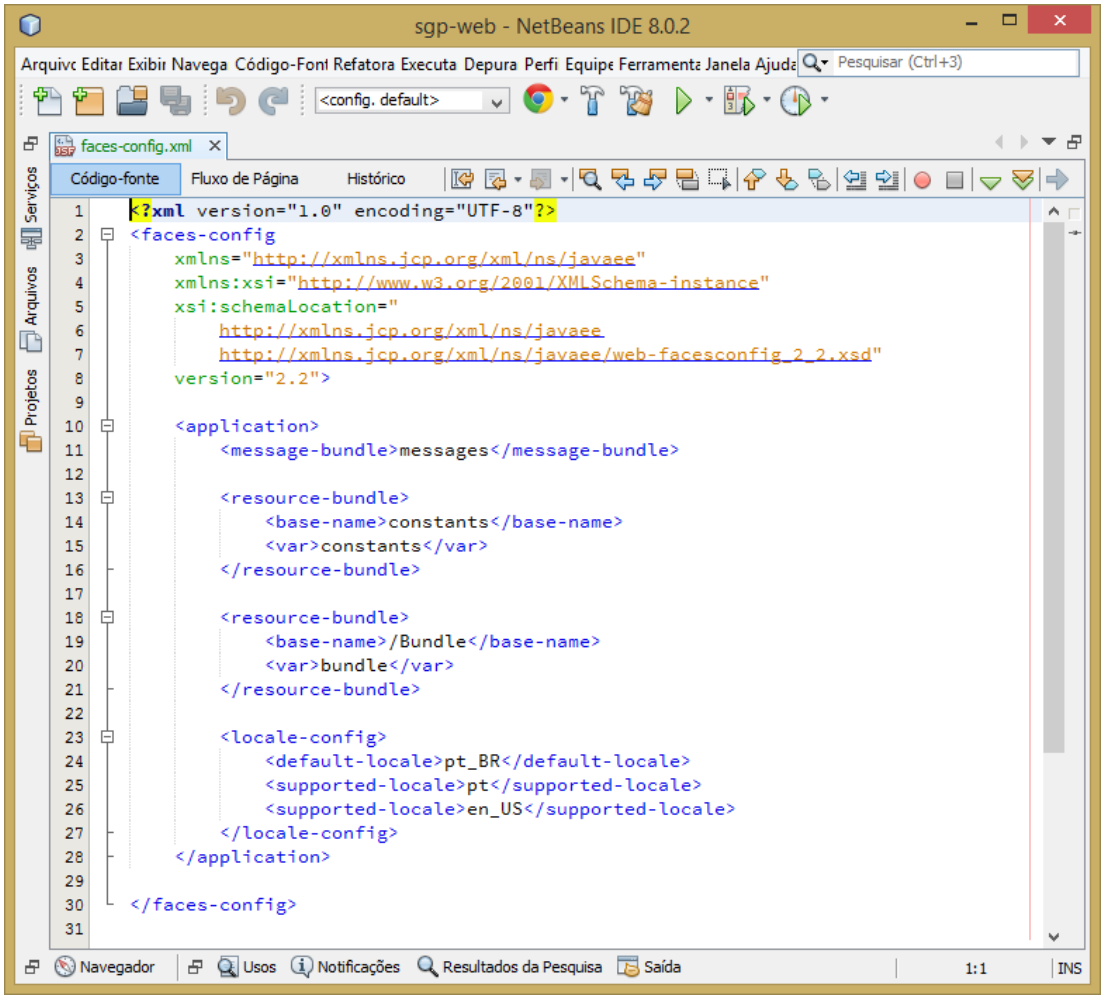


Figura 29 – Arquivo de configuração do JSF

PrettyFaces é uma biblioteca que permite a criação de URLs amigáveis e que podem ser guardadas (PRETTYFACES, 2015), e que são mapeadas no arquivo *pretty-config.xml*, conforme a Figura 30.

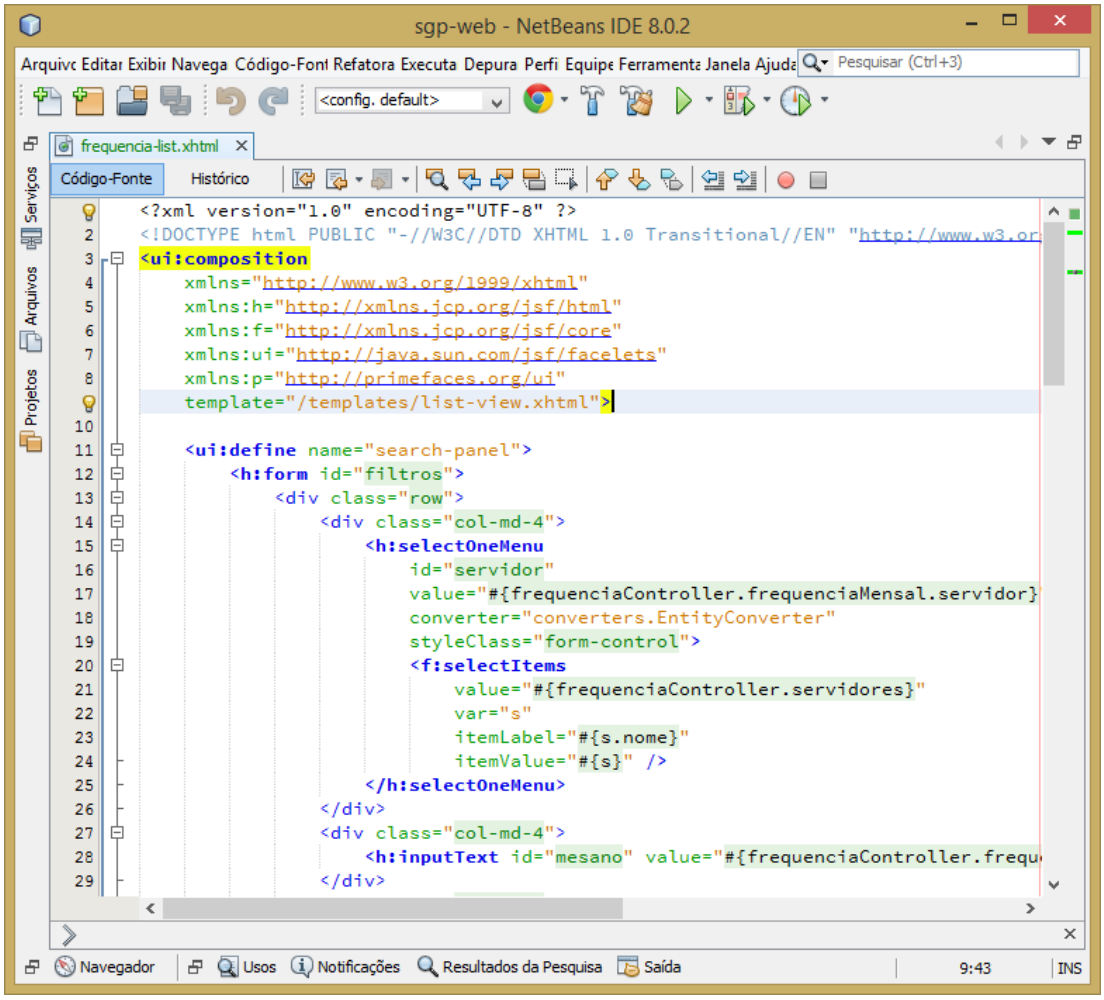


Figura 30 – Página web em XHTML com tags do JSF

Após a configuração, desenvolvemos cada uma das visões, usando como base templates fornecidos pelo pacote *br.jus.tre-mt.reuse*. No desenvolvimento, utilizamos componentes da implementação de referência do JSF, o Mojarra, e elementos do Twitter Bootstrap, um framework CSS e JS que oferece classes e estruturas prontas para criação de páginas web.

3. RESULTADOS

Os resultados obtidos com esse trabalho são a documentação do SGP-Web e parte do software concluída, que contém as funcionalidades Atestação de Frequência Individual, Cadastro de Justificativa, Cadastro de Marcação Manual e Homologação de Frequência.

A documentação do software se resume na *Descrição Geral do projeto, Cadastro de Requisitos* e *Plano de Liberações*, resultados da fase de Iniciação; e o *Modelo Arquitetônico, Diagrama de Classes, Diagrama de Casos de Uso* e *Descrição de Casos de Uso*, resultados da fase de Elaboração.

A pesquisa para a elaboração dos artefatos foi feita a partir do manual do software de gestão em uso atualmente, em reuniões com a Coordenadoria de Pessoal e a partir do uso do do sistema atual, além de um estudo da integração com os demais sistemas relacionados a Frequência Nacional fornecidos pelo Tribunal Superior Eleitoral. Os requisitos foram confirmados pela Coordenadoria e por toda a equipe.

O resultado da fase de Ativação são artefatos da disciplina de Requisitos. Os documentos basicamente apresentam tabelas com informações importantes sobre o projeto, desde seu escopo, objetivo, justificativas, principais funções funcionalidades e definições importantes. Na Tabela 1 podemos ver como exemplo as restrições do projeto.

Número de ordem	Restrição	Descrição
1	Segurança	O sistema deve restringir o acesso através de senhas individuais para cada usuário, utilizando a autenticação CAS
2	Legal	O sistema deverá estar de acordo com os regulamentos e portarias de sua época
3	Expansibilidade	O sistema deve ser feito de forma modular, de forma que permita mudanças constantes por conta das portarias que podem afetar a especificação
4	Ambiente	O sistema deve ser acessível em navegadores Web, sendo eles: IE 8+, Mozilla Firefox em sua versão mais recente e Google Chrome em sua versão mais recente

Tabela 1 – Restrições do Software

Na Tabela 2 temos os atores, conforme especificado na *Descrição Geral do Projeto*.

Número de ordem	Ator	Definição
1	Servidor	Servidor do TRE-MT (sede e cartórios)
2	Titular de Seção	titular de seção, com responsabilidade de homologar frequência, justificativa, escala, substituição e férias de servidores
3	Administrador	Servidor do STI que pode configurar e editar dados de forma ampla

Tabela 2 - Atores

Na Tabela 3 temos como exemplo a lista de Casos de Uso da Aplicação.

ID	Nome	Categoria	Prioridade	Estabilidade	Estado	Status
CUA1	Atestação de Frequência Individual	Base	Essencial	Alta	Identificado	Original
CUA2	Marcação de Ponto Manual	Base	Essencial	Alta	Identificado	Original
CUA3	Cadastro de Justificativa	Base	Essencial	Média	Identificado	Original
CUA4	Validação de Frequência	Base	Essencial	Média	Identificado	Original
CUA5	Homologação de Frequência	Base	Essencial	Alta	Identificado	Original

Tabela 3 – Casos de Uso da Aplicação

Na Tabela 4 temos os requisitos não-funcionais, conforme especificado no artefato *Cadastro de Requisitos*.

ID	Nome	Categoria	Prioridade	Estabilidade	Complexidade	Estado	Status
RNF1	O sistema deve restringir o acesso através de senhas individuais para cada usuário, utilizando a autenticação CAS	Técnico	Essencial	Alta	Média	Identificado	Original
RNF2	O sistema deverá estar de acordo com os regulamentos e portarias de sua época	Qualidade	Essencial	Média	Baixa	Identificado	Original
RNF3	O sistema deve ser feito de forma modular, de forma que permita mudanças constantes por conta das portarias que podem afetar a especificação	Qualidade	Desejável	Média	Alta	Identificado	Original
RNF4	O sistema deve ser acessível em navegadores Web, sendo eles: IE 8+, Mozilla Firefox em sua versão mais recente e Google Chrome em sua versão mais recente	Técnico	Essencial	Alta	Baixa	Identificado	Original
RNF5	Implementação JEE	Técnico	Essencial	Alta	Baixa	Identificado	Original
RNF6	Modelagem UML 2.0	Técnico	Essencial	Alta	Média	Identificado	Original

Tabela 4 – Requisitos não-funcionais

Na fase de Elaboração, como resultado das tarefas da disciplina de Análise, foi gerado o diagrama de classes, com definição de atributos, conforme a Figura 31. O foco da análise foi identificar as entidades de dados, visto que no que tange a arquitetura do sistema todos os módulos terão uma estrutura parecida (visão, controle, negócio, repositório). O mapeamento das entidades foi feito a partir de um estudo do banco de dados do Frequência Nacional, um conjunto de aplicações fornecidas pelo Tribunal Superior Eleitoral para o controle de frequência dos servidores (SGRH e SRH).

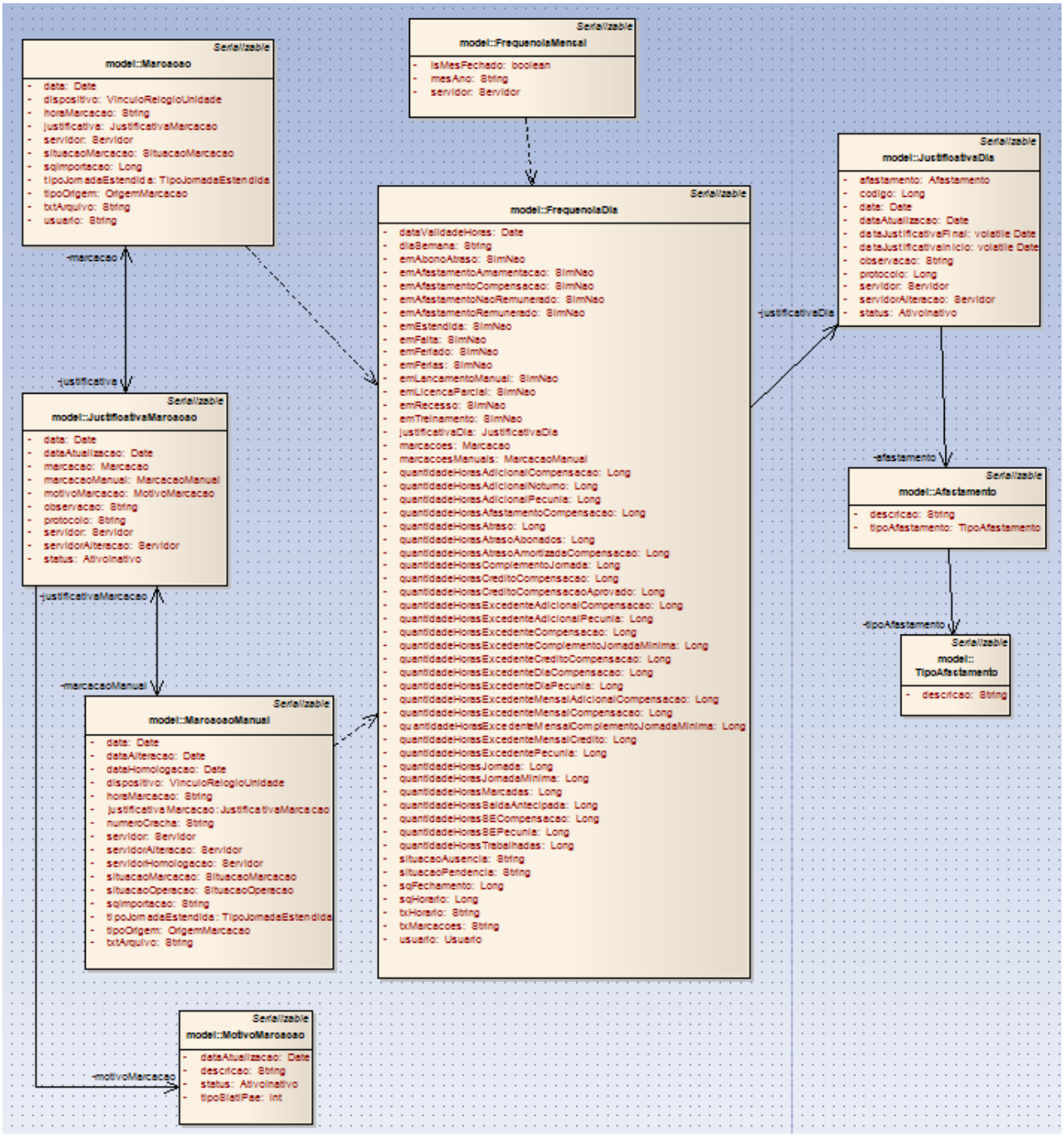


Figura 31 – Diagrama de classes de entidade

No desenvolvimento da solução do caso de uso, foi criado o diagrama de casos de uso relativo aos módulos de frequência, conforme a Figura 32, e o artefato **Descrição de casos de uso**, conforme o exemplo na Tabela 5.

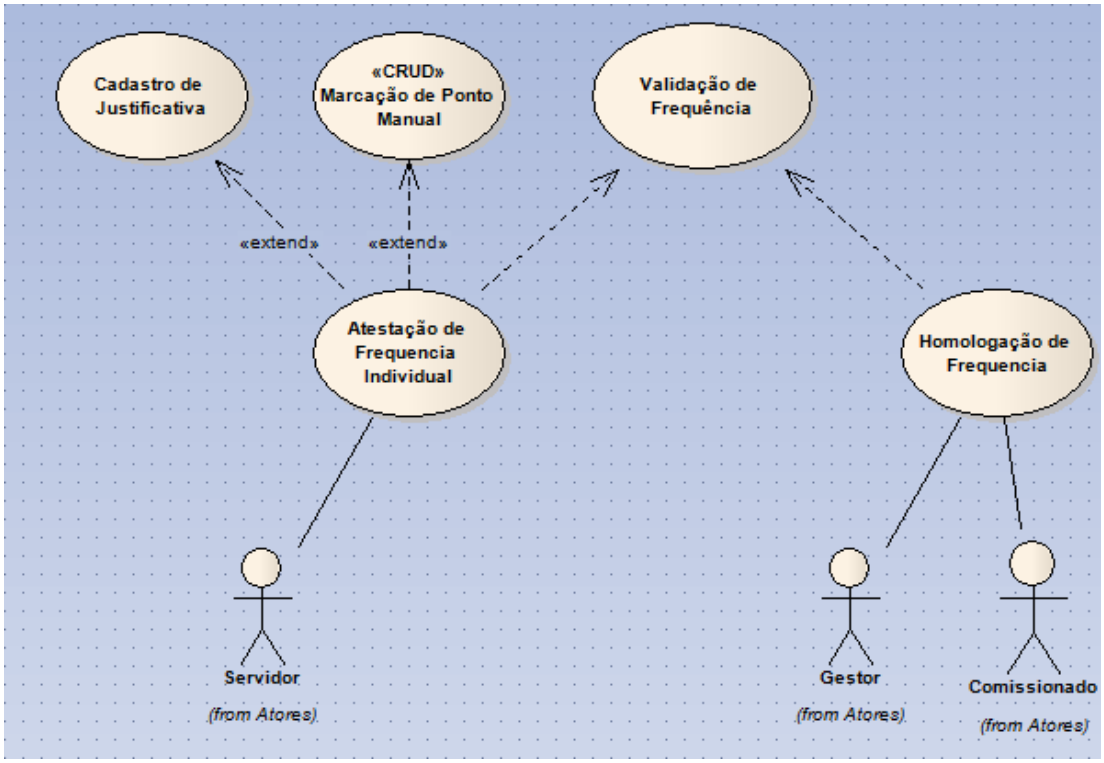


Figura 32 – Diagrama de Casos de Uso

Caso de Uso	**Atestação de Frequência Individual**
Objetivo	Consultar as marcações em relógio de ponto, as marcações manuais, procurar por pendências, e enviar a frequência mensal para validação pelo Comissionado.
Pré-condições	Consultar as marcações em relógio de ponto, as marcações manuais, procurar por pendências, e enviar a frequência mensal para validação pelo Comissionado.
Fluxo Principal	1. O Servidor abre o módulo **Atestação de Frequência Individual** 2. O **Sistema** exibe a frequência mensal do mês atual, com suas marcações e pendências. As frequências serão ordenadas por dias, e cada dia poderá ser colorido em caso de feriado, afastamento, justificativa e fim de semana.
Subfluxos	**Frequência por mês:** 1. O Servidor informa um mês específico e aciona o comando "Pesquisar".

	2. O **Sistema** exibe as frequências diárias do mês informado.
Fluxos Alternativos	**Fluxo alternativo Cadastro de Justificativa** **Pré-condição:** O Servidor selecionou uma frequência mensal que está na situação *ABERTO*. 1. O Servidor seleciona uma das frequências e aciona o comando "Ver detalhes". 2. O **Sistema** exibe informações a respeito das marcações e justificativas relacionadas a frequência selecionada. 3. O Servidor aciona o comando "Nova Justificativas". 4. O **Sistema** executa o caso de uso **Cadastro de Justificativa** **Fluxo alternativo Marcação Manual de Ponto** **Pré-condição:** O Servidor selecionou uma frequência mensal que está na situação *ABERTO*. 5. O Servidor seleciona uma das frequências e aciona o comando "Ver detalhes". 6. O **Sistema** exibe informações a respeito das marcações e justificativas relacionadas a frequência selecionada. 7. O Servidor aciona o comando "Nova Marcações". 8. O **Sistema** executa o caso de uso **Marcação Manual de Ponto**

Tabela 5 – Descrição do caso de uso

Na fase de Elaboração também foi desenvolvido o modelo arquitetural. Como um diagrama de componentes, como na Figura 33. A arquitetura do sistema é baseada num *micro-framework* desenvolvido internamente, tendo como base o JSF e o CDI, que tem como paradigma o modelo MVC, e oferece interfaces e classes extensíveis para criação de classes de controle, negócio, serviço (acesso ao repositório de dados) e os modelos de entidade.

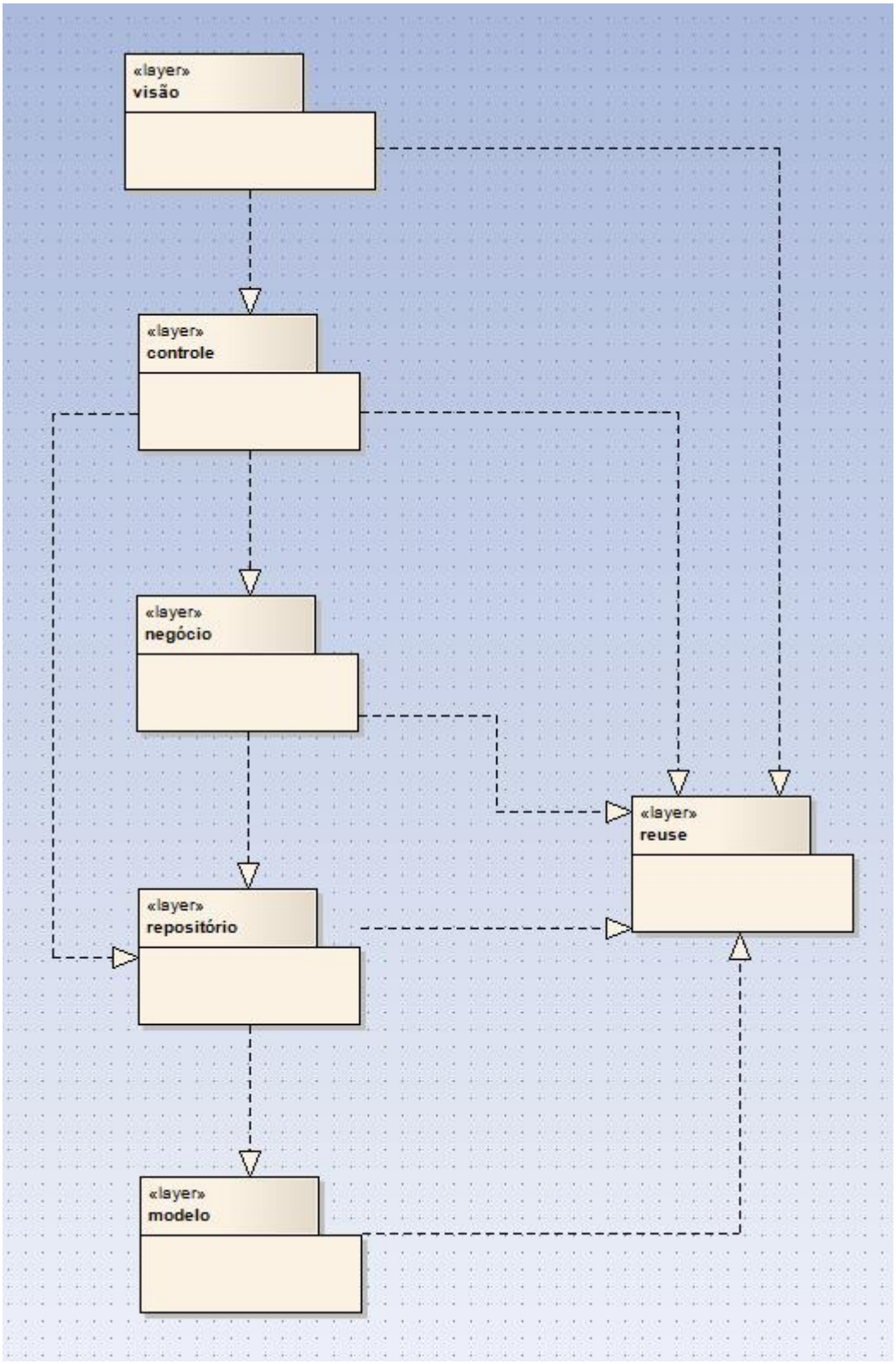


Figura 33 – Diagrama de componentes

Por fim, o desenvolvimento teve como resultado as funcionalidades Atestação de Frequência Individual, Cadastro de Justificativa, Cadastro de Marcação Manual e Homologação de Frequência.

A primeira tela da Atestação de Frequência é uma lista de marcações de ponto do mês atual, e filtros para alterar o servidor (da mesma seção) e o período (mês e ano), como na Figura 34. No exemplo, é exibida a frequência de maio de 2014 de um servidor, a partir do banco de testes.

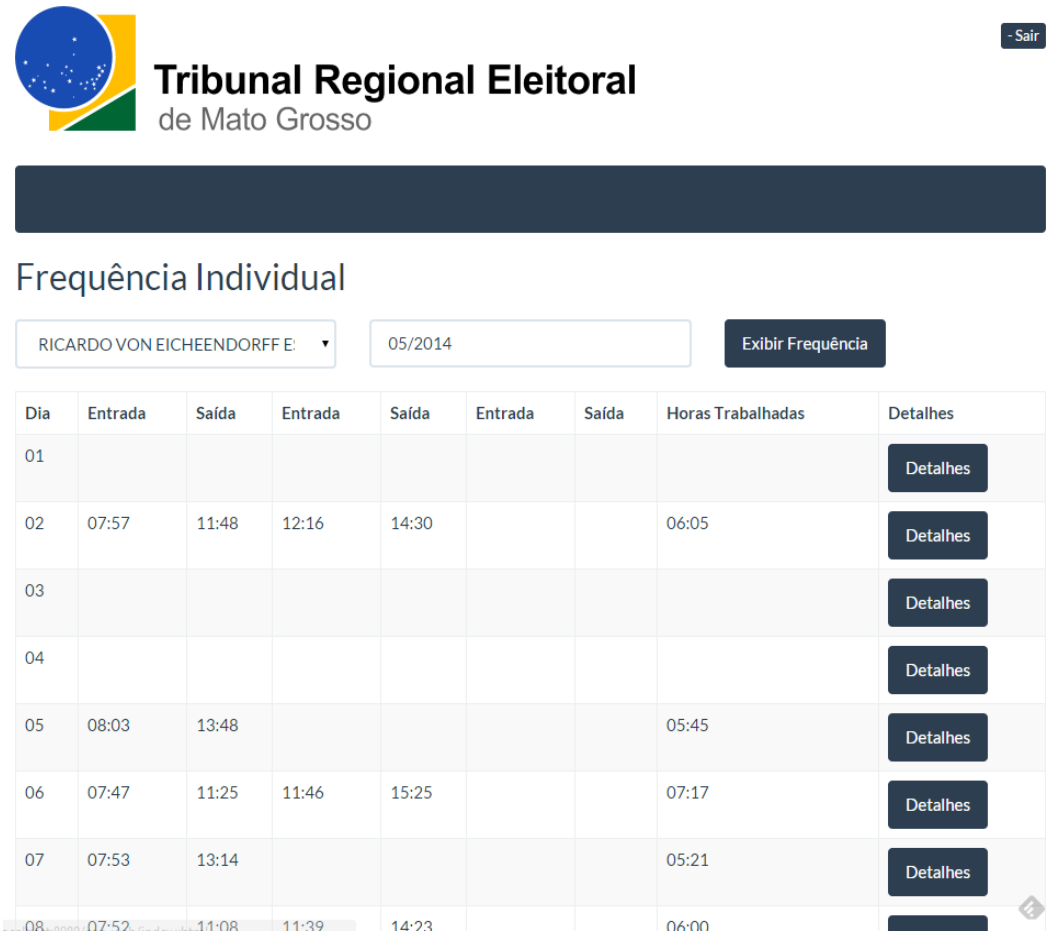


Figura 34 – Atestação de Frequência Individual

Para cada dia, há uma ação Detalhes, na qual se encontra informações e cadastros relativos a este dia, como na Figura 35. Na tela de Detalhes, temos informações sobre a frequência do dia selecionado, quantidade de horas trabalhadas, marcações feitas em relógio de ponto e cadastro de Marcação Manual e Justificativa.

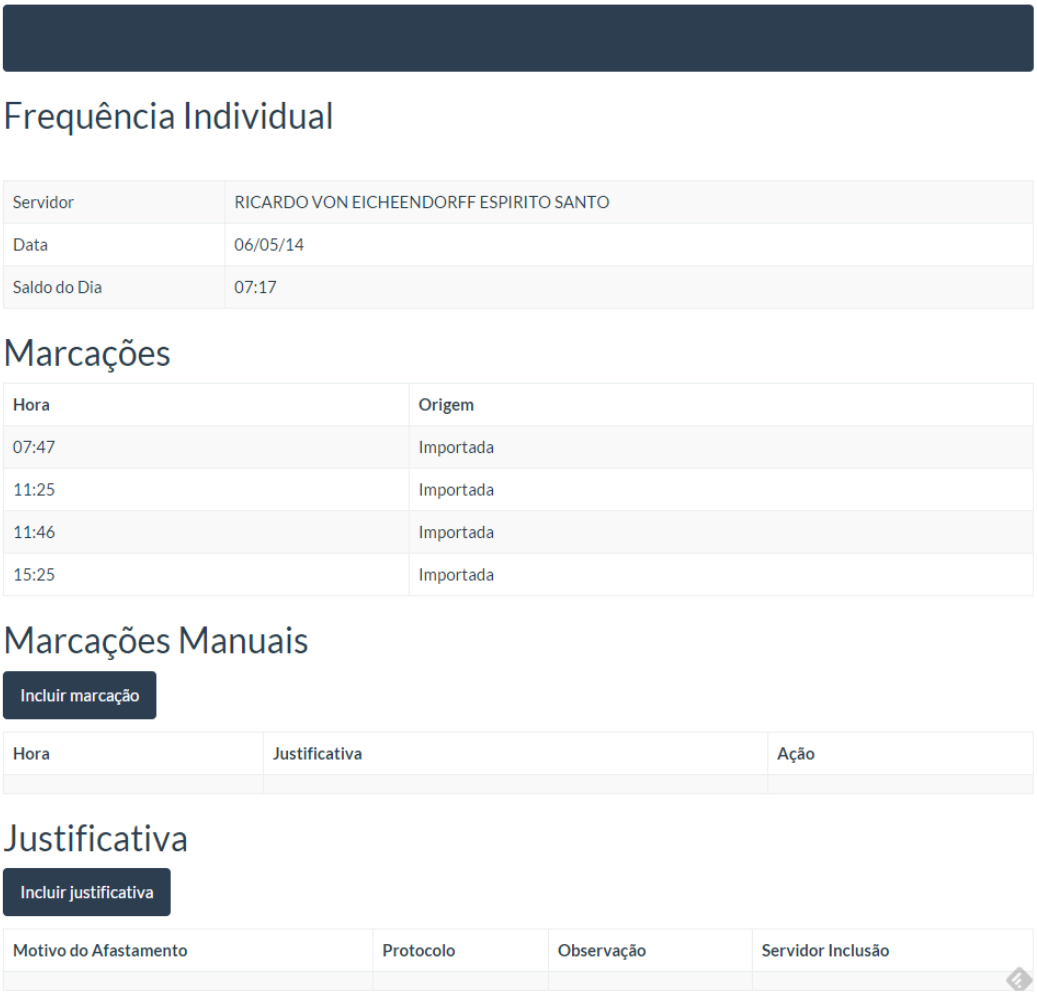


Figura 35 – Detalhe de uma frequência

Ao clicar em “Incluir Marcação”, “Incluir Justificativa” ou “Editar”, um modal é aberto com o cadastro do item, como podemos ver nas Figuras 36 e 37.

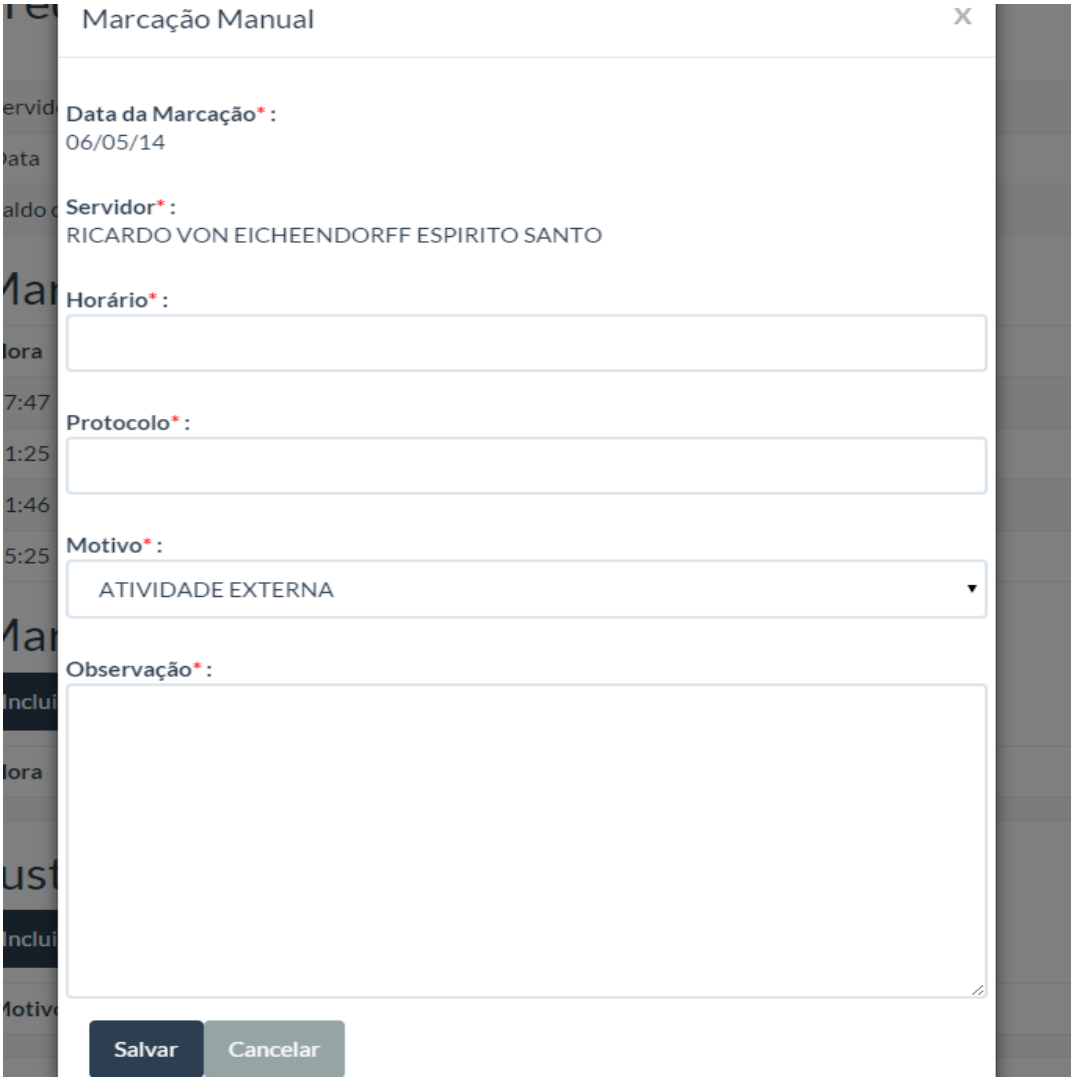


Figura 36 – Cadastro de Marcação Manual

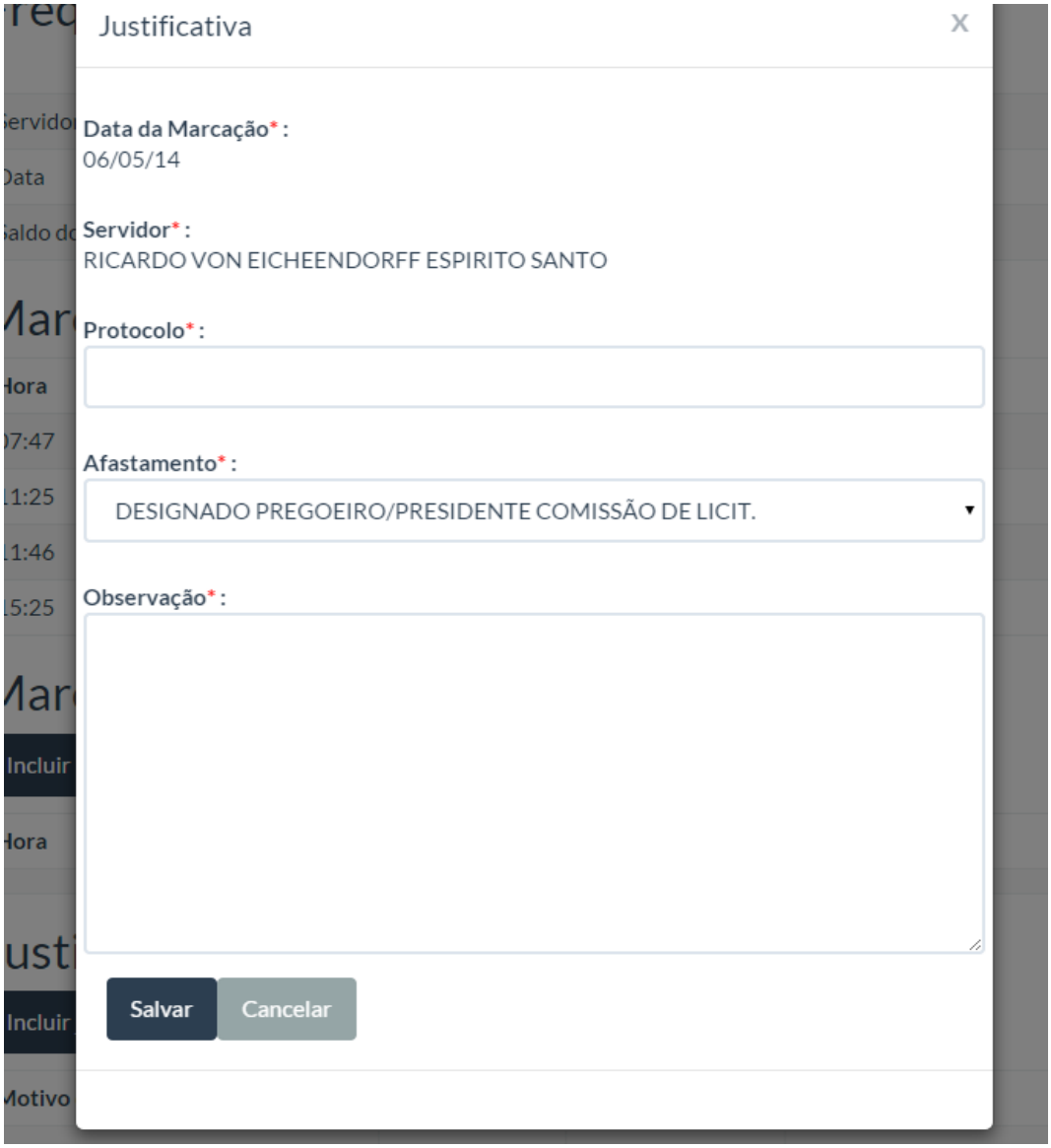


Figura 37 – Cadastro de Justificativa

4. DIFICULDADES ENCONTRADAS

Durante o desenvolvimento lidamos com diversas dificuldades, devido a falta de conhecimento prévio de algumas ferramentas, como o Apache Maven e o CDI, e a complexidade de lidar com um banco de dados legado, nacional, já populado e sem a possibilidade de realizar alterações. Entretanto, estas dificuldades foram superadas através de estudos e pesquisas realizadas pelo aluno durante o estágio, procurando sempre a melhor solução para os problemas encontrados e aprender novos conhecimentos técnicos e teóricos.

Graças às escolhas de arquitetura, e o desenvolvimento utilizando o parão *Enterprise Application*, é possível ter um sistema que pode ser facilmente estendido, além de possibilitar que no futuro se crie *webservices* a partir das classes EJB, oferecidos a outras aplicações, possibilitando uma maior integração futura entre os sistemas do Frequência Nacional.

O uso do CDI, no lugar de um *framework* de terceiros para a injeção de dependências de classes, a definição de interfaces que atendem a necessidades variadas, e a busca pelo desenvolvimento de classes independentes, genéricas, diminui os problemas de forte acoplamento encontrados no sistema legado, além de diminuir os custos futuros de manutenção.

Buscamos assim, desenvolver um código limpo, reutilizável, com baixo acoplamento e que pode ser estendido em serviços para outras aplicações e clientes.

5. CONCLUSÕES

O trabalho exposto se baseia em conceitos de Engenharia de Software, Processos de Desenvolvimento de Software e Desenvolvimento Orientado a Objetos utilizando a especificação Java EE.

O objetivo do trabalho foi o desenvolvimento de uma ferramenta de gestão de pessoas para o Tribunal Regional Eleitoral de Mato Grosso (TRE-MT), utilizando: (i) a especificação Java EE e ferramentas integradas no desenvolvimento; e (ii) a documentação do sistema guiada pelo Processo Praxis.

Com o estudo dos conceitos, especificação e das ferramentas utilizadas, foi possível alcançar todos os objetivos do trabalho. Como futuras ações deste trabalho é possível destacar a utilização do Processo Praxis como um experimento do uso de uso de um processo de software num ambiente que requer mudança cultural para a implantação definitiva de uma metodologia.

Este estudo pode ser estendido em trabalhos futuros, de forma a beneficiar a SBD a adotar medidas que aumentem sua eficiência em desenvolvimento de software. O primeiro trabalho é analisar a efetividade do Processo Praxis para o desenvolvimento de um software de tamanho médio ou pequeno. Os sistemas desenvolvidos pela Seção de Banco de Dados são de pequena escala, e não necessitam de uma documentação extensa; e a implantação de qualquer processo exigiria uma mudança de uma cultura de desenvolvimento individual para a de colaboração e gerência de projetos. O Processo Praxis é um processo robusto e, apesar de permitir simplificações, é pensado para grandes equipes de desenvolvimento, e a complexidade de assimilá-lo pode fazer com que a equipe considere-o custoso. É sugerido comparar o Processo Praxis com outras metodologias, sejam clássicas ou ágeis, analisando tanto a eficiência quanto a facilidade de adoção.

O segundo trabalho é discutir o uso do Java como plataforma para sistemas pequenos e médios. O uso do Java no TRE-MT é resultado de um alinhamento com outros tribunais e o serviço público em geral. Há muitos sistemas legados

desenvolvidos em Java, e a infraestrutura já é preparada para a plataforma. A plataforma Java é robusta, e oferece um grande número de bibliotecas, *frameworks* e ferramentas. Porém, considerando que a Seção de Banco de Dados se concentra no desenvolvimento de pequenos e médios sistemas, com alteração constante, e conta com uma equipe reduzida, talvez a escolha por essa plataforma se torne custosa para a equipe. É sugerido comparar a plataforma Java com outras plataformas e linguagens modernas, tais como Python e Ruby, além do C#, que é tão robusto quanto o Java.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, Scott. Mapping Objects to Relational Databases: O/R Mapping in Detail. Disponível em: <http://www.agiledata.org/essays/mappingObjects.html> (acessado em 23 de dezembro de 2014).

AMBLER, Scott. The Object-Relational Impedance Mismatch. Disponível em: <http://www.agiledata.org/essays/impedanceMismatch.html> (acessado em 23 de dezembro de 2014).

ARAUJO, C. Introdução à Persistência com Hibernate – Parte 1 - Easy Java Magazine 4. Disponível em: <http://www.devmedia.com.br/introducao-a-persistencia-com-hibernate-parte-1-easy-java-magazine-4/20138> (acessado em 23 de dezembro de 2014).

CAELUM. Java e Orientação a Objetos. [s.l.: s.n.], Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/>. (acessado em: 1 jan 2015).

CAELUM. Java para Desenvolvimento Web. [s.l.: s.n.], Disponível em: <https://www.caelum.com.br/apostila-java-web/>. (acessado em: 1 jan. 2015).

CAVALCANTI, Lucas. Processo de build com o Maven - blog.caelum.com.br. [s.l.: s.n.], 2008, Disponível em: <http://blog.caelum.com.br/processo-de-build-com-o-maven//>. (acessado em: 1 jan. 2015).

DEVMEDIA, Criando e configurando um projeto Web - JSF 2, Primefaces 3 e CDI, Disponível em <http://www.devmedia.com.br/criando-e-configurando-um-projeto-web-jsf-2-primefaces-3-e-cdi/25251>. (acessado em 2 jan 2015).

FOWLER, Martin, P of EAA: Data Mapper, Disponível em <http://martinfowler.com/eaaCatalog/dataMapper.html>. (acessado em 10 de fev 2015).

GITHUB, Git, Disponível em: <http://git-scm.com> (acessado em 12 de janeiro de 2015)

NETBEANS, Bem-Vindo ao NetBeans, Disponível em <https://netbeans.org/index_pt_PT.html>. (acessado em 2 jan 2015).

NETBEANS, Introdução ao JavaServer Faces 2.x, Disponível em <https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html>. (acessado em 2 jan 2015).

PAULA FILHO, Wilson de Pádua, 2009, Engenharia de Software: Fundamentos, Métodos e Padrões. Terceira Edição. Rio de Janeiro. LTC

PAULA FILHO, Wilson de Pádua. Praxis. Disponível em: <http://homepages.dcc.ufmg.br/~wilson/praxis/> (acessado em 06 de dezembro de 2014).

PRESMANN, Roger S. 1995, Engenharia de Software. Terceira Edição. São Paulo. Makron Books.

PRETTYFACES, Introdução ao JavaServer Faces 2.x, Disponível em <http://ocpsoft.org/prettyfaces>. (acessado em 2 jan 2015).

PROJECT MANAGEMENT INTITUTE. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos. 3. ed. Pennsylvania: PMI, 2004.

APÊNDICES

Descrição de Casos de Uso

Caso de uso Atestação de Frequência Individual

Objetivo

Consultar as marcações em relógio de ponto, as marcações manuais, procurar por pendências, e enviar a frequência mensal para validação pelo Comissionado.

Precondições

O Servidor deve estar cadastrado numa seção.

Fluxo principal

1. O Servidor abre o módulo **Atestação de Frequência Individual**
2. O **Sistema** exibe a frequência mensal do mês atual, com suas marcações e pendências. As frequências serão ordenadas por dias, e cada dia poderá ser colorido em caso de feriado, afastamento, justificativa e fim de semana.

Subfluxos

Subfluxo Frequência por Mês

1. O Servidor informa um mês específico e aciona o comando "Pesquisar".
2. O **Sistema** exibe as frequências diárias do mês informado.

Fluxos alternativos

Fluxo alternativo Cadastro de Justificativa

Precondições	O Servidor selecionou uma frequência mensal que está na situação *ABERTO*.
Passos	1. O Servidor seleciona uma das frequências e aciona o comando "Ver detalhes".

2. O **Sistema** exibe informações a respeito das marcações e justificativas relacionadas a frequência selecionada.
3. O Servidor aciona o comando “Nova Justificativas”.
4. O **Sistema** executa o caso de uso **Cadastro de Justificativa**.

Fluxo alternativo Marcação Manual de Ponto

Precondições	O Servidor selecionou uma frequência mensal que está na situação ABERTO.
Passos	1. O Servidor seleciona uma das frequências e aciona o comando “Ver detalhes”. 2. O **Sistema** exibe informações a respeito das marcações e justificativas relacionadas a frequência selecionada. 3. O Servidor aciona o comando “Nova Marcações”. 4. O **Sistema** executa o caso de uso **Marcação Manual de Ponto**.

Caso de uso Cadastro de Justificativa

Objetivo

Listar e cadastrar justificativas para a ausência de marcações de ponto em dias úteis

Precondições

1. O Servidor selecionou uma frequência mensal que está na situação *ABERTO*.
2. O Servidor acionou o comando “Ver Detalhes” em uma das frequências.
3. O Servidor acionou o comando “Ver Justificativas”

Fluxo principal

1. O **Sistema** exibe, na aba **Justificativas**, uma justificativa cadastrada ou um

formulário de cadastro de justificativa, executando assim o subfluxo **Cadastro de Justificativa**.

Subfluxos

Subfluxo Nova Justificativa

1. O Servidor informa os dados da justificativa, motivo, protocolo (número do processo PAe ou SIATI) e observação e aciona o comando "Salvar".
2. O **Sistema** exibe o aviso "A justificativa foi salva".

Subfluxo Edição de Justificativa

1. O Servidor aciona o comando "Editar"
2. O Servidor informa os dados da justificativa, motivo, protocolo (número do processo PAe ou SIATI) e observação e aciona o comando "Salvar".
3. O **Sistema** exibe o aviso "A justificativa foi salva".

Fluxos alternativos

Fluxo alternativo Protocolo da Justificativa

Precondições	O Servidor informa um protocolo de justificativa relativo a um processo no PAe ou SIATI.
Passos	1. O **Sistema** verificará o protocolo informado. Se não houver processo no PAe ou SIATI, o sistema avisará e não criará a justificativa. Se não for informado processo, não será feita a validação. 2. O **Sistema** exibe o aviso "A justificativa foi salva".

Caso de uso Marcação de Ponto Manual

Objetivo

Criar marcações manuais de ponto, a serem autorizadas pelo Comissionado, em casos de ausência de marcações em relógio de ponto.

Precondições

1. O Servidor selecionou uma frequência mensal que está na situação *ABERTO*.
2. O Servidor acionou o comando “Ver Detalhes” em uma das frequências.
3. O Servidor acionou o comando “Ver Justificativas”

Fluxo principal

1. O **Sistema** exibe, na aba **Marcações**, uma lista de marcações registradas

Subfluxos

Subfluxo Nova Marcação

1. O Servidor aciona o comando “Novo”
2. O Servidor insere os dados relativos a marcação, horário, protocolo, motivo e justificativa, e aciona o comando “Salvar”
3. O **Sistema** exibe o aviso “A justificatia foi salva”.

Subfluxo Editar Marcação

1. O Servidor seleciona uma marcação e aciona o comando “Editar”
2. O Servidor insere os dados relativos a marcação, horário, protocolo, motivo e justificativa, e aciona o comando “Salvar”.
3. O **Sistema** exibe o aviso “A justificativa foi salva”.

Fluxos alternativos

Fluxo alternativo Protocolo de Marcação Manual

Precondições	O Servidor informa um protocolo de marcação relativo a um processo no PAe ou SIATI.
Passos	1. O **Sistema** verificará o protocolo informado. Se não houver processo no PAe ou SIATI, o sistema avisará e não criará a justificativa. Se não for informado processo, não será feita a validação.

2. O **Sistema** exibe o aviso “A marcação foi salva”.

Caso de uso Homologação de Frequência

Objetivo

O Comissionado consulta as frequências mensais dos Servidores de sua seção, e atesta a veracidade das marcações manuais de ponto, para posterior fechamento por parte do Gestor.

Precondições

1. O perfil **Ocupante de Cargo Comissionado** deve estar selecionado.

Fluxo principal

1. O Comissionado abre o módulo **Homologação de Frequência**.
2. O **Sistema** exibe a frequência mensal do mês atual do próprio Comissionado, com suas marcações e pendências. As frequências serão ordenadas por dias, e cada dia poderá ser colorido em caso de feriado, afastamento, justificativa e fim de semana.

Subfluxos

Subfluxo Frequência por Mês

1. O Servidor informa um mês específico, e aciona o comando “Pesquisar”.
2. O **Sistema** exibe as frequências diárias do mês informado.

Subfluxo Frequência por Servidor

1. O Servidor informa um servidor de sua seção, e aciona o comando “Pesquisar”.
2. O **Sistema** exibe as frequências diárias do servidor informado.

Subfluxo Deferimento de Marcações

1. O Comissionado escolhe uma frequência aciona o comando “Ver Detalhes”
2. O **Sistema** então exibe, na seção **Marcações**, uma lista de marcações registradas
3. O Comissionado seleciona as marcações que deseja deferir ou indeferir.
4. O Comissionado aciona o comando “Deferir” ou “Indeferir”.
5. O **Sistema** exibe o aviso “A marcação foi deferida”.

Deferir todas as marcações

1. Na tela de frequência mensal, o Comissionado aciona o comando “Deferir todas as marcações”
2. O **Sistema** exibe uma tela de confirmação.
3. O Comissionado seleciona a opção “Sim” para confirmar.
4. O **Sistema** avisa que a tarefa foi concluída.

Fluxos alternativos

Validação de Pendências feita pelo sistema

Precondições	Há dias sem marcações, ou pendências no mês
Passos	1. O Sistema exibe o aviso “Há pendências na frequência mensal”.

Caso de uso Validação de Frequência

Objetivo

O Servidor ou o Comissionado validam a frequência mensal, e o **Sistema** registra a data da validação.

Precondições

1. O Servidor ou o Comissionado selecionou uma frequência mensal que está na situação *ABERTO*.

Fluxo principal

1. O Servidor ou o Comissionado abrem a frequência mensal pelos módulos **Atestação de Frequência Individual** ou **Homologação de Frequência**
2. O Servidor ou o Comissionado aciona o comando “Validar”

Subfluxos

Subfluxo Verificação de Frequência

1. O **Sistema** verifica a frequência mensal por pendências, como ausência de batidas e necessidade de marcações e justificativas.
2. Caso encontradas pendências, o **Sistema** exibe o aviso “Foram encontradas pendências na frequência mensal”.

Subfluxo Validação da Frequência

1. Caso não sejam encontradas pendências, o **Sistema** exibe o aviso “Frequência validada com sucesso”.
2. O **Sistema** exibe, no lugar do comando “Validar”, a data em que a frequência foi validada.