FELIPE RODRIGUES DO PRADO
JOÃO PAULO NAKAJIMA PEREIRA

DESENVOLVIMENTO MODULAR DE SOFTWARE UTILIZANDO OSGI

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE
2015

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BSD	*Berkeley Software Distribution*
CSS	*Cascading Style Sheet*
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language*
ICC	Inatel *Competence Center*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
JDBC	Java *Database Connectivity*
JEE	Java *Platform, Enterprise Edition.*
JMS	Java *Message Service*
JPA	Java *Persistence* API
JVM	Java *Virtual Machine*
NTT	*Nippon Telegraph and Telephone*
OSGi	*Open Services Gateway initiative*
SENAC	Serviço Nacional de Aprendizagem Comercial
SQL	*Structured Query Language*
SRA	*Systems Research and Applications Corporation*
TDC	*The Developers Conference*
UNIFEI	Universidade Federal de Itajubá
UNIVAS	Universidade do Vale do Sapucaí
XHTML	*Extensible HyperText Markup Language*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 QUADRO METODOLÓGICO

Neste capítulo serão abordados e descritos os caminhos definidos e utilizados para conduzir o projeto até seu fim. Serão apresentados o tipo de pesquisa, seu contexto, bem como os instrumentos, os procedimentos, o orçamento e o cronograma para o desenvolvimento deste projeto.

1.1 Tipo de pesquisa

Pesquisa é um processo desenvolvido com o objetivo de se obter respostas para indagações propostas, através de conhecimentos existentes e a utilização de métodos, técnicas e procedimentos científicos. Uma pesquisa se faz necessária quando não existem repostas suficientes que satisfaçam a resolução de problemas (GIL, 2002).

Para atingir os objetivos deste projeto, desenvolveu-se uma pesquisa de abordagem aplicada, a qual é utilizada quando se desenvolve um produto real, com uma finalidade prática, que pode ser aplicada em determinado contexto. Conforme aponta Appolinário (2004), pesquisas aplicadas têm o objetivo de resolver problemas ou necessidades concretas e imediatas.

Aplicando esses conceitos e utilizando a pesquisa de forma aplicada por agregar maiores vantagens e demonstrar melhor os resultados obtidos, desenvolveu-se um *software* modularizado utilizando a tecnologia OSGi, com o objetivo de apresentar o paradigma da modularização e sua arquitetura, demonstrando suas características no desenvolvimento de softwares.

1.2 Contexto de pesquisa

Cada vez mais os *softwares* são utilizados em empresas, indústrias, computadores

pessoais, web e dispositivos móveis. Estes são desenvolvidos por meio de práticas e tecnologias existentes que auxiliam na sua criação, porém a não utilização de tais ferramentas tornam o seu desenvolvimento e manutenção um processo desgastante e trabalhoso.

Esta pesquisa demonstra a arquitetura e desenvolvimento modularizado utilizando a tecnologia OSGi, que oferece grande organização na estrutura do *software*, vantagens na manutenção e maior facilidade na expansão do mesmo, isso pelo fato dessa tecnologia permitir a adição e integração de novas funcionalidades em modo *on-the-fly*[1].

Softwares modularizados oferecem maior flexibilidade para atender empresas de diferentes áreas. Através do OSGi, os módulos podem ser desenvolvidos de forma independente, disponibilizando módulos específicos para cada empresa. Dessa maneira desenvolveu-se uma aplicação que pode ser adotada por empresas de diferentes segmentos no mercado, bastando apenas que exista um módulo que atenda a necessidade específica da área.

O objetivo desse projeto é apresentar o modelo de desenvolvimento modular, sua arquitetura, metodologias, engenharias, técnicas e tecnologias para o âmbito de *softwares* comerciais, provendo facilidade na manutenção e expansão do mesmo. Esta pesquisa, por apresentar uma análise e demonstração do desenvolvimento modular, servirá de base de conhecimento para estudantes, profissionais da área de tecnologia da informação e empresas que pretendem empregar essa forma de desenvolvimento.

1.3 Instrumentos

Durante o desenvolvimento deste projeto foi necessário a realização de reuniões entre os participantes, para a obtenção e organização das informações, divisão das tarefas e desenvolvimento do *software*, pois segundo Kioskea (2014), as reuniões são um meio para partilhar, num grupo de pessoas, um mesmo nível de conhecimento sobre um assunto ou um problema e para tomar decisões coletivamente. Além disso, decisões tomadas coletivamente, com representantes das diferentes entidades interessadas, serão facilmente aceitas por todos.

Os dados foram obtidos a partir de livros, artigos, documentações e reuniões online com pessoas que possuem conhecimento e experiência sobre o tema do projeto. Após

1 *On-the-fly:* Termo referente ao funcionamento do sistema, onde a nova funcionalidade é adicionada sem que seja necessário reiniciar toda a aplicação, com ela em pleno funcionamento.

aplicados estes instrumentos foi possível analisar e organizar as informações coletadas.

As reuniões entre os participantes possuem o objetivo de planejar a execução do projeto, tomar decisões quanto ao desenvolvimento do *software*, bem como as tecnologias e metodologias que foram utilizadas.

Para obtermos mais conhecimentos acerca do tema, realizou-se uma entrevista com Filipe Portes, pois o mesmo apresenta grande conhecimento teórico e prático sobre modularização de *softwares* utilizando OSGi. Filipe Portes, é graduado em Ciências da Computação pela Universidade Paulista, com mais de oito anos de experiência em Desenvolvimento e Arquitetura de Sistemas Java para *web*. Ministrou palestras sobre OSGi no TDC[2] em 2012 e 2014, apresentando uma visão ampla da tecnologia, explicando conceitos do desenvolvimento modular, características e seu funcionamento, além de demonstrar exemplos práticos.

A entrevista realizou-se em 1 de agosto de 2015, utilizando o *software* Skype[3]. Através desta, foram sanadas dúvidas sobre a tecnologia OSGi e conceitos da arquitetura modular. A entrevista realizada foi de grande importância, pois pôde-se compreender melhor o funcionamento da tecnologia e sua arquitetura..

Houve o compartilhamento de experiências, projetos e padrões de desenvolvimento, que foram utilizados para obtenção de mais conhecimentos práticos acerca do tema do projeto. Com isso, percebeu-se falhas na arquitetura do *software* que havia sido criado, podendo então realizar as correções necessárias..

1.4 Procedimentos

Nesta pesquisa desenvolveu-se uma aplicação que exemplifica os conceitos de uma arquitetura modular, utilizando como principais tecnologias, a linguagem de programação Java e a especificação OSGi, que oferece suporte a modularização de softwares. Para o seu desenvolvimento foi definido um cronograma com os principais procedimentos que foram executados de forma progressiva conforme os resultados obtidos por meio dos estudos realizados em cada tecnologia.

2 TDC – Abreviação para *The Developers Conference*
3 Skype – software gratuito que permite fazer de chamadas de voz e vídeo, chat de mensagens e o compartilhamento de arquivos.

1.4.1 Prototipação

Através dos resultados obtidos na realização de estudos sobre a especificação OSGi, desenvolveu-se protótipos a fim de verificar seu funcionamento. Foram desenvolvidos diferentes tipos de protótipos, com o objetivo de definir como seria desenvolvido uma aplicação que demonstrasse a modularização de softwares.

Desenvolveu-se protótipos que podem ser executados no ambiente do tipo *desktop* e *web*. De acordo com os resultados, definimos que a aplicação desenvolvida para exemplificar a modularização de software seria do tipo *web*.

A especificação OSGi é implementada por vários *frameworks*, dentre eles, os protótipos foram criados utilizando o Equinox e Apache Felix. Desta forma, definiu-se como implementação a ser utilizada para o desenvolvimento o *framework* Apache Felix, devido a sua excelente integração com o GlassFish. Sendo assim, definiu-se também, a utilização do GlassFish como servidor de aplicação, completando assim o escopo *back-end* para o desenvolvimento de uma aplicação *web*.

Com o desenvolvimento dos protótipos, baseado nos estudos realizados, observou-se que a tecnologia OSGi oferece suporte a projetos do tipo Java *Application*, *Web Application* e *Enterprise* Java *Bean*. Com isso, constatou-se a possibilidade de desenvolver módulos de diferentes tipos, que interagem entre si através de interfaces bem definidas, que são disponibilizadas como serviços ou exportadas dentro do contexto OSGi.

Foram desenvolvidos também protótipos de módulos responsáveis por disponibilizar a interface gráfica com o usuário, utilizando as tecnologias HTML, JavaScript, Angular JS, CSS e Bootstrap.

A prototipação foi um procedimento muito importante para o desenvolvimento desta pesquisa, pois além de se comprovar o funcionamento prático das teorias estudadas, observou-se que o desenvolvimento modular requer um planejamento complexo de sua arquitetura, proporcionando desacoplamento dos módulos e alta coesão, ou seja, os módulos podem ser desinstalados, parados e atualizados em tempo de execução sem afetar outros módulos, além de definir responsabilidades específicas para cada módulo.

1.4.2 Definição do *software*

Após estudos e criação de protótipos, definiu-se uma aplicação básica contendo cinco módulos, para demonstrar uma arquitetura modular utilizando a tecnologia OSGi através do *framework* Apache Felix. O objetivo do *software* é apenas demonstrar o funcionamento da arquitetura assim como características da tecnologia OSGi.

- **Módulo Usuário:** responsável por controlar os cadastros do usuário e autenticação dos mesmos no sistema;
- **Módulo Clientes:** responsável por controlar os cadastros de clientes, com integração ao módulo financeiro;
- **Módulo Financeiro:** responsável por controlar os lançamentos de contas a pagar e a receber, com integração ao módulo de clientes;
- **Módulo Logs:** responsável por registrar logs do sistema em arquivos, com integração entre todos os outros módulos.
- **Módulo Data Source:** responsável por fornecer e controlar acesso ao banco de dados, com integração entre todos os outros módulos.

A aplicação foi definida para execução no ambiente *web*, sendo implantada no servidor de aplicação Glassfish. Isso se justifica devido a grande flexibilidade que os sistemas *web* oferecem, podendo ser executado em qualquer sistema operacional, assim como qualquer dispositivo que tenha suporte ao *browser*.

1.4.3 Modelagem da arquitetura dos módulos

A arquitetura de um *software* é uma das principais partes no desenvolvimento de uma aplicação. Dessa maneira, criou-se uma arquitetura modular que fornece aos módulos flexibilidade para que possam ser desinstalados, parados e atualizados enquanto o restante do sistema está em funcionamento.

Modelou-se uma arquitetura em que os módulos do sistema são formados por outros módulos menores. Normalmente, cada módulo do *software* está formado por outros três módulos denominados como módulo de visão, responsável por conter as telas de interação com o usuário, módulo API, que contém funcionalidades e interfaces, que são compartilhadas e expostas como serviços, e o módulo core, que contém a implementação das interfaces do módulo API e a lógica de negócio da aplicação.

Com os módulos dispostos dessa maneira, o único módulo que gera dependência para outros módulos é o módulo API, com isso o módulo de visão e *core* podem ser desinstalados, parados e atualizados sem comprometer o restante do *software*. Definiu-se então, que o módulo API é a base principal para os módulos de visão e core, não podendo ser desinstalado sistema.

Essa arquitetura pode ser entendida melhor de acordo com a Figura 1 que demonstra a estrutura definida para os módulos.

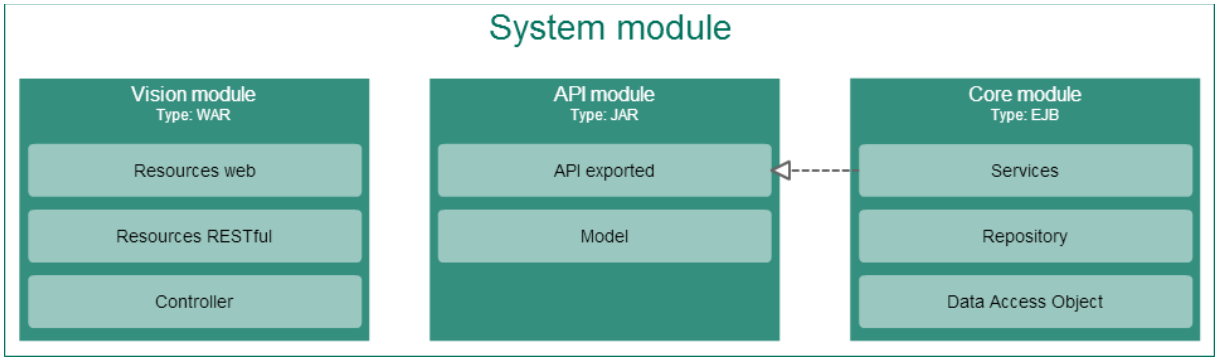

Figura 1 - Arquitetura de um módulo do sistema. **Fonte: Elaborado pelo autor**

A arquitetura dos módulos foi desenvolvida com base nos estudos, práticas pesquisadas, e de acordo com necessidade da aplicação. Porém isso pode mudar dependendo da aplicação que se deseja desenvolver.

A Figura 2 demonstra como foi definida a arquitetura completa de todos os módulos que compõem o software desenvolvido.

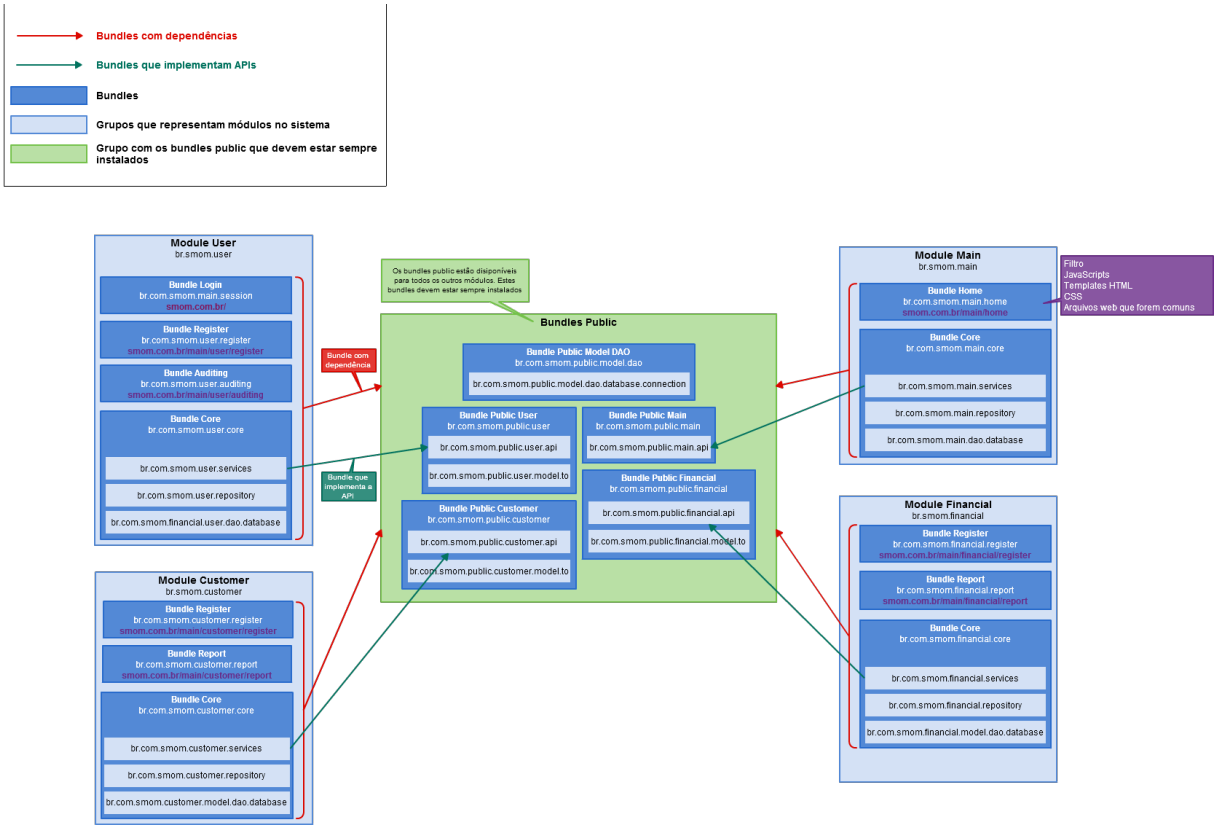

Figura 2 - Arquitetura completa dos módulos do sistema. **Fonte: Elaborado pelo autor**

Através da modelagem dessa arquitetura, foi possível desenvolver um *software* em que os módulos não dependam diretamente da implementação de outros módulos. Isso se fez devido a criação de interfaces bem definidas que estão nos módulos APIs. Resultando em um *software* desacoplado e também de alta coesão devido a cada módulo ter sua responsabilidade bem definida.

1.4.4 Desenvolvimento

O *software* desenvolvido está separado por duas partes que possuem responsabilidades específicas, o *back-end* e *front-end*. Os módulos foram divididos de forma que estas partes fiquem bem definidas. O *back-end* é composto pelos módulos que contém regras de negócio, recursos e serviços. Enquanto o *front-end* é responsável por realizar uma interface entre o sistema e o usuário consumindo as funcionalidades do *back-end*.

Como dito anteriormente, basicamente cada módulo do sistema é composto por outros

três módulos menores, o módulo de visão, API e *core*. No módulo de visão, estão os recursos RESTful e controle de fluxo. No módulo API estão definidos as interfaces que são expostas como serviços e funcionalidades, ambas estão disponíveis para qualquer outro módulo. E por fim, no módulo *core* está a implementação das interfaces, assim como toda lógica de negócio e controle do módulo. Todos esses fatores compõem a parte *back-end* da aplicação.

Os módulos foram desenvolvidos utilizando a IDE de desenvolvimento NetBeans 8.0.2 acompanhado do padrão Maven que propõe uma estrutura para cada tipo de projeto, além de possibilitar que projetos sejam criados em uma estrutura hierárquica que, possibilita criar um projeto principal que controle outros projetos que o compõe. O Maven já vem integrado ao NetBeans, desta forma não foi necessária nenhuma configuração para a utilização do mesmo.

A estrutura oferecida pelo Maven é muito útil para uma aplicação modular, pois, como cada módulo é um projeto, o mesmo permite melhor controle da hierarquia dos módulos, além de todas as configurações de geração dos projetos estarem definidas em um projeto principal.

A estrutura do projeto está separada por um projeto principal do tipo POM *Project,* que contém todos os outros projetos. O mesmo é composto por outros projetos do mesmo tipo, que representam os módulos do sistema, que por fim, são compostos pelos projetos do tipo *Web Application* (módulo de visão), Java *Application* (módulo de API) e *Enterprise* JavaBeans (módulo *core*).

O projeto do tipo POM *Project* foi criado selecionando a opção **Maven → POM Project** após selecionar a opção **File → New Project** no menu principal do NetBeans, como demonstrado nas figuras 3 e 4. Em seguida, como demonstra a figura 5, foi definido o nome do projeto e escolhido o local onde o mesmo seria salvo. Além dessas informações, foi definido também o **Artifact Id, Group Id**, **Version** e **Package**.

- **Artifact Id:** nome único para o projeto dentro do contexto do Maven que será o nome final do módulo;
- **Group Id:** grupo definido para os projetos;
- **Version:** versão definida para o projeto, que será a versão do módulo;
- **Package:** nome inicial para a estrutura de pacotes do projeto. Está informação é opcional para projetos do tipo POM *Project*.

As opções descritas acima são utilizadas pelo Maven para controlar o projeto, além de serem utilizadas para geração final do projeto de cada módulo.

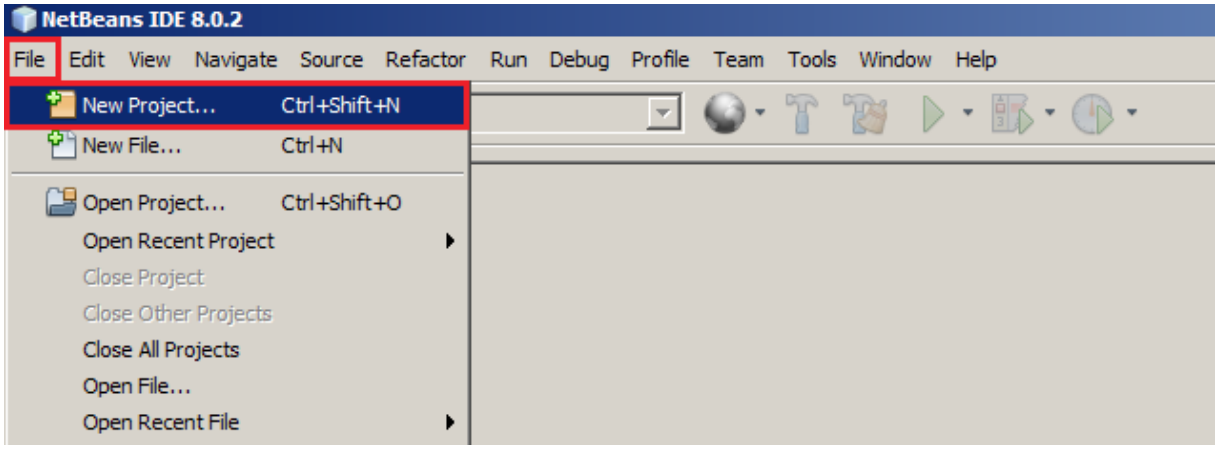

Figura 3 - Opção para criar projeto. **Fonte: Elaborado pelo autor**

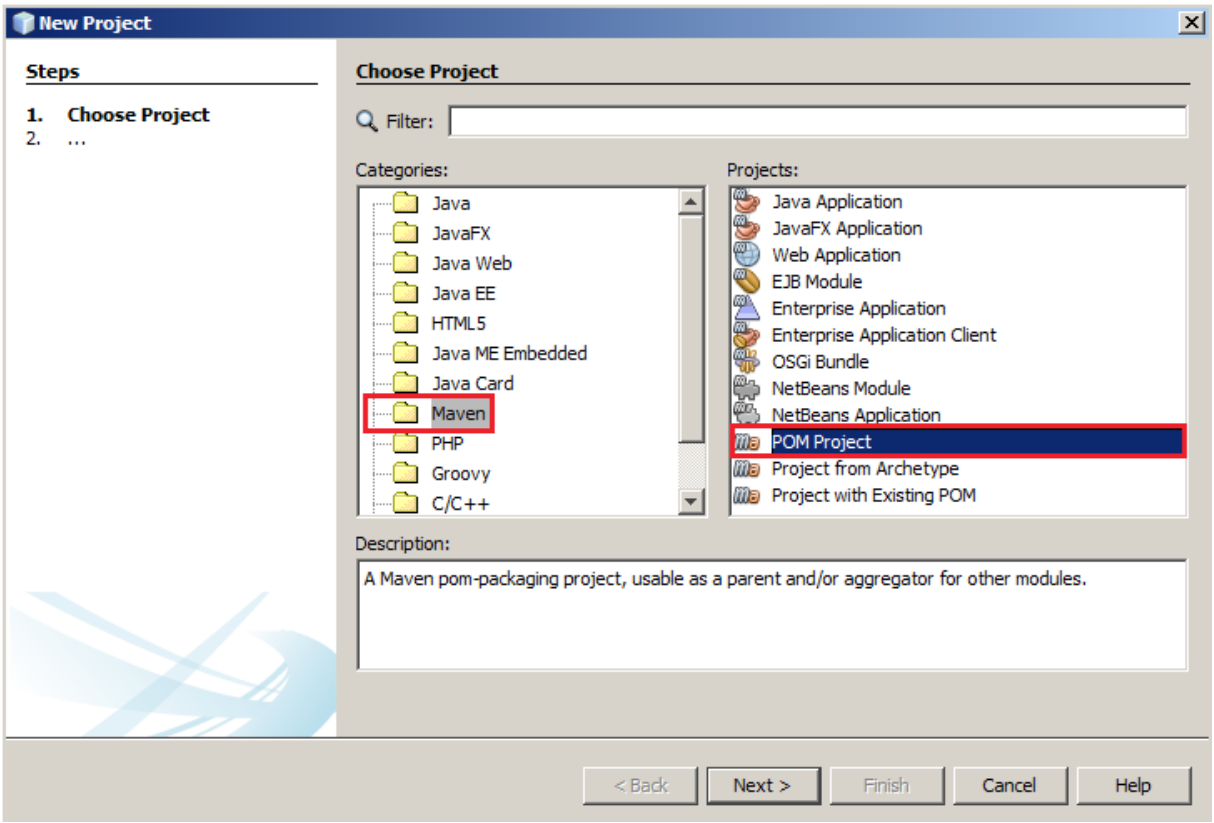

Figura 4 - Tela para escolha do tipo do projeto como POM *Project.* **Fonte: Elaborado pelo autor**

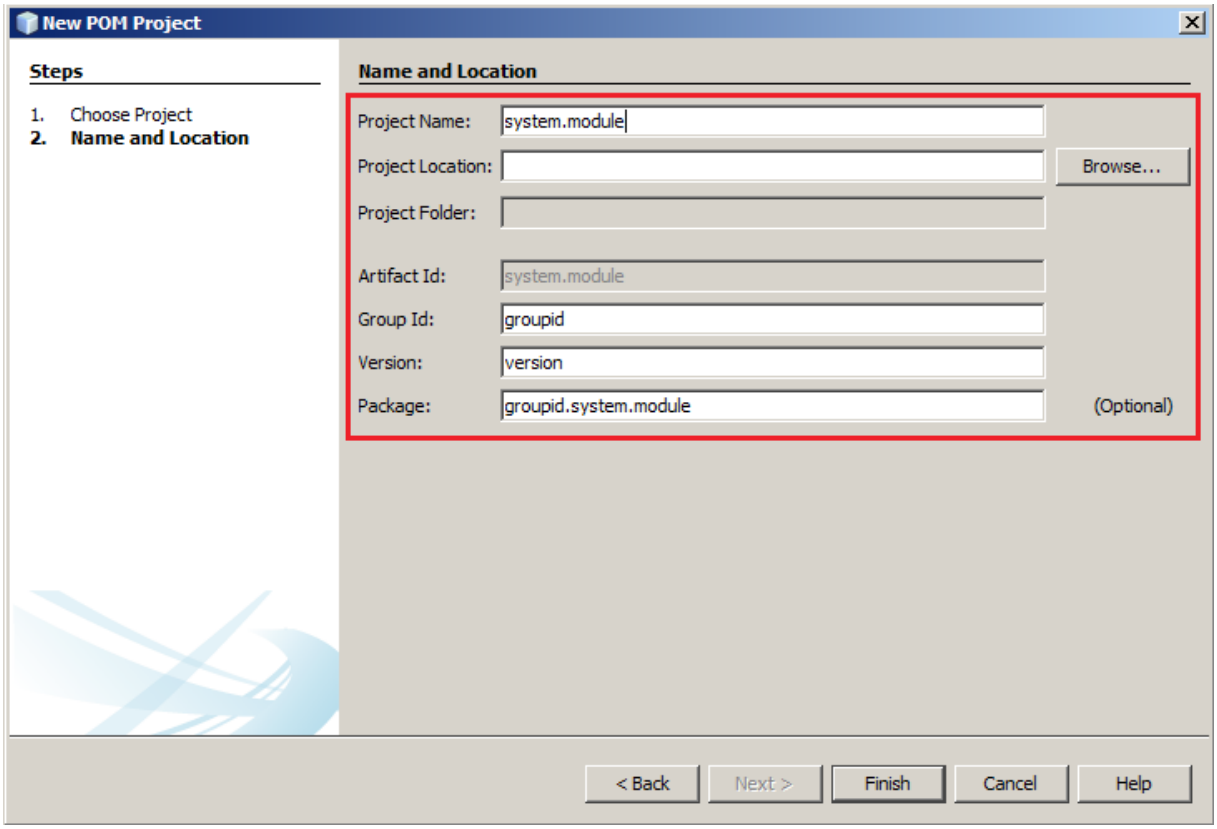

Figura 5 - Tela de informações do novo projeto. **Fonte: Elaborado pelo autor**

Os projetos que representam os módulos do sistema, também são do tipo POM *Project* e foram criados através da opção **Modules → Create New Module** localizada no projeto criado anteriormente, conforme é demonstrado na Figura 6. Após escolhida essa opção basta seguir os procedimentos da Figura 4 e 5.

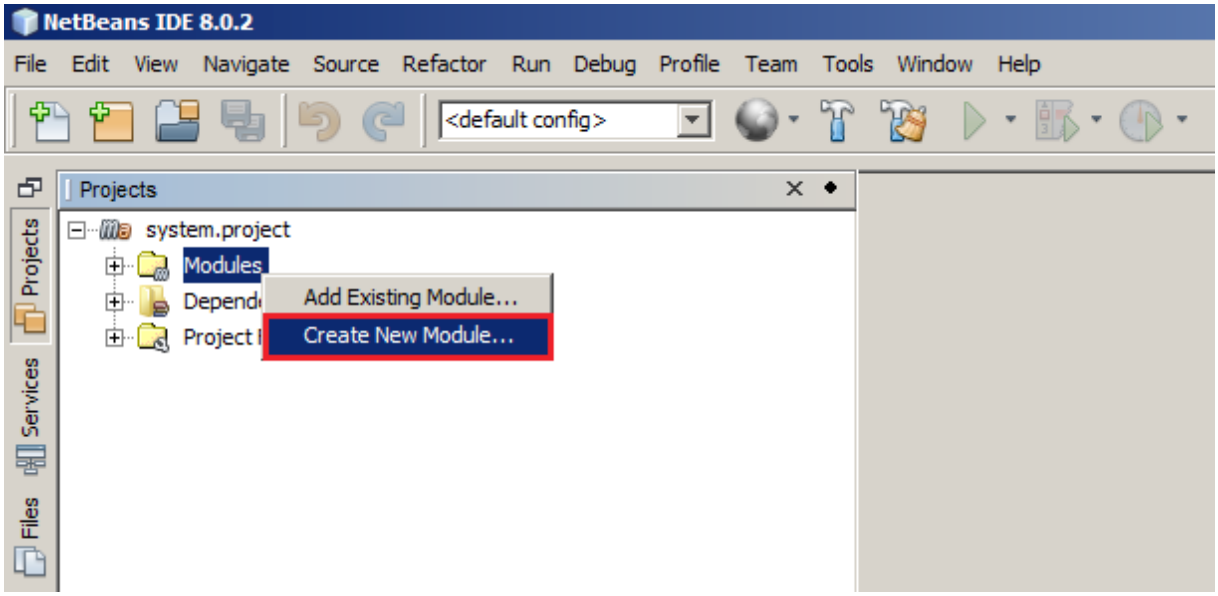

Figura 6 - Criação de novo projeto como módulo. **Fonte: Elaborado pelo autor**

Após fazer as etapas descritas até aqui, o projeto ficou com uma estrutura conforme demonstrada na Figura 7.

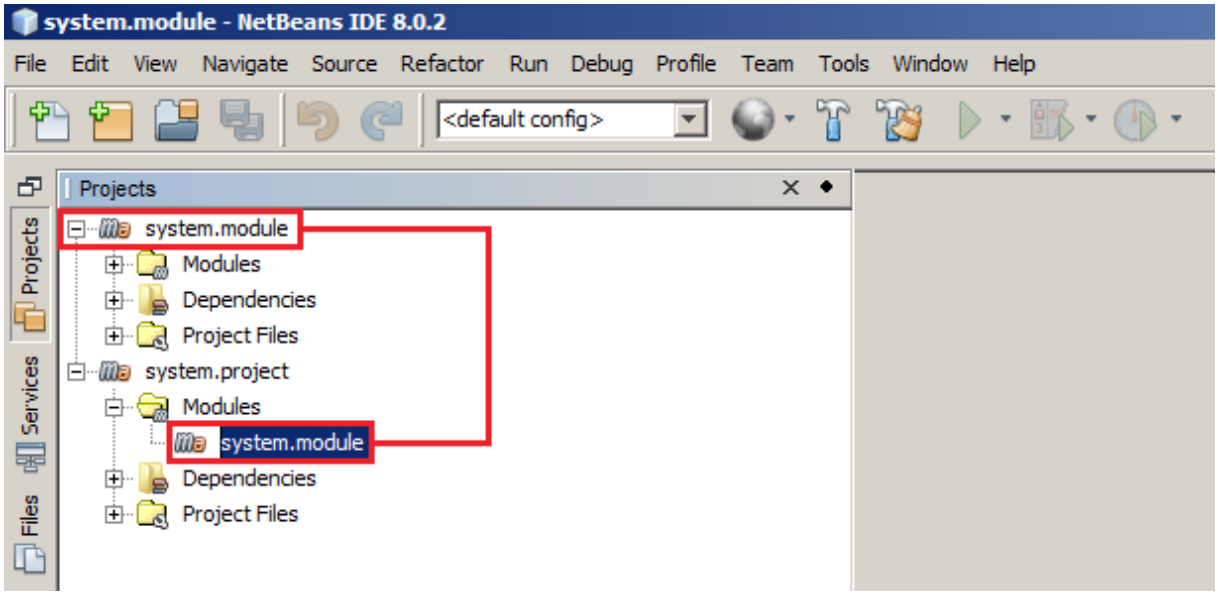

Figura 7 - Estrutura do projeto principal composto por outro projeto. **Fonte: Elaborado pelo autor**

A criação dos projetos do tipo *Web Application* foi realizada através a opção **Modules → Create New Module** no projeto que representa os módulos do sistema, conforme demonstrado na Figura 8. Em seguida foi escolhido a opção **Maven → Web Application** conforme a Figura 9. Após isto, basta definir as informações do novo projeto conforme a

Figura 5 e, por fim, é necessário definir o GlassFish como servidor de aplicação na opção **Server** e escolher Java EE 7 Web para a opção **Java EE Version** como mostra a Figura 10.

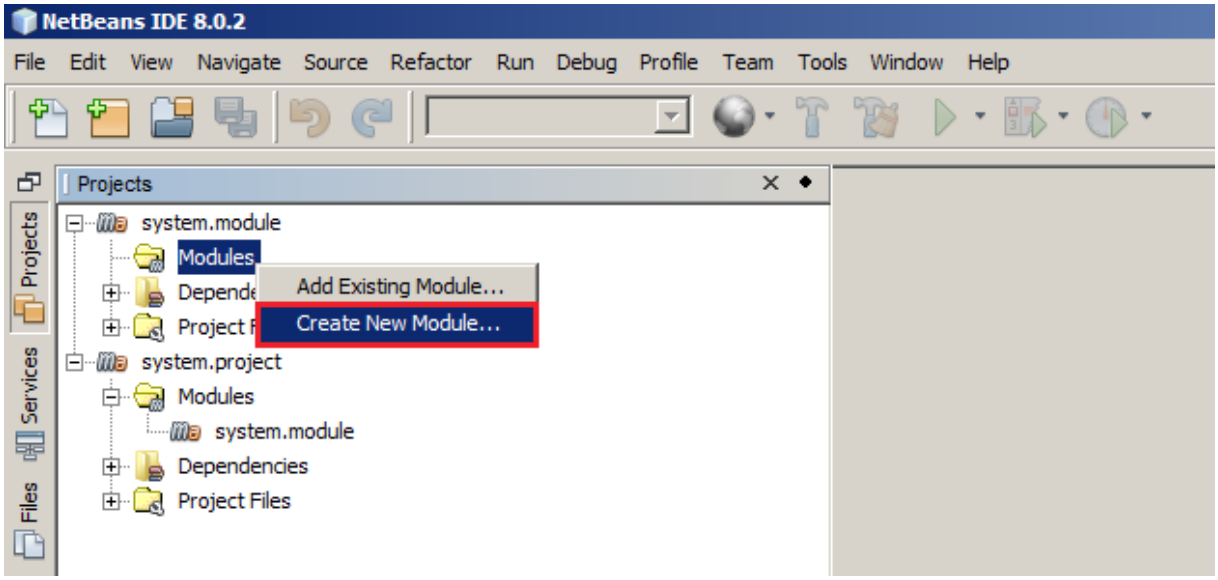

Figura 8 - Criação de novo projeto como módulo. **Fonte: Elaborado pelo autor**

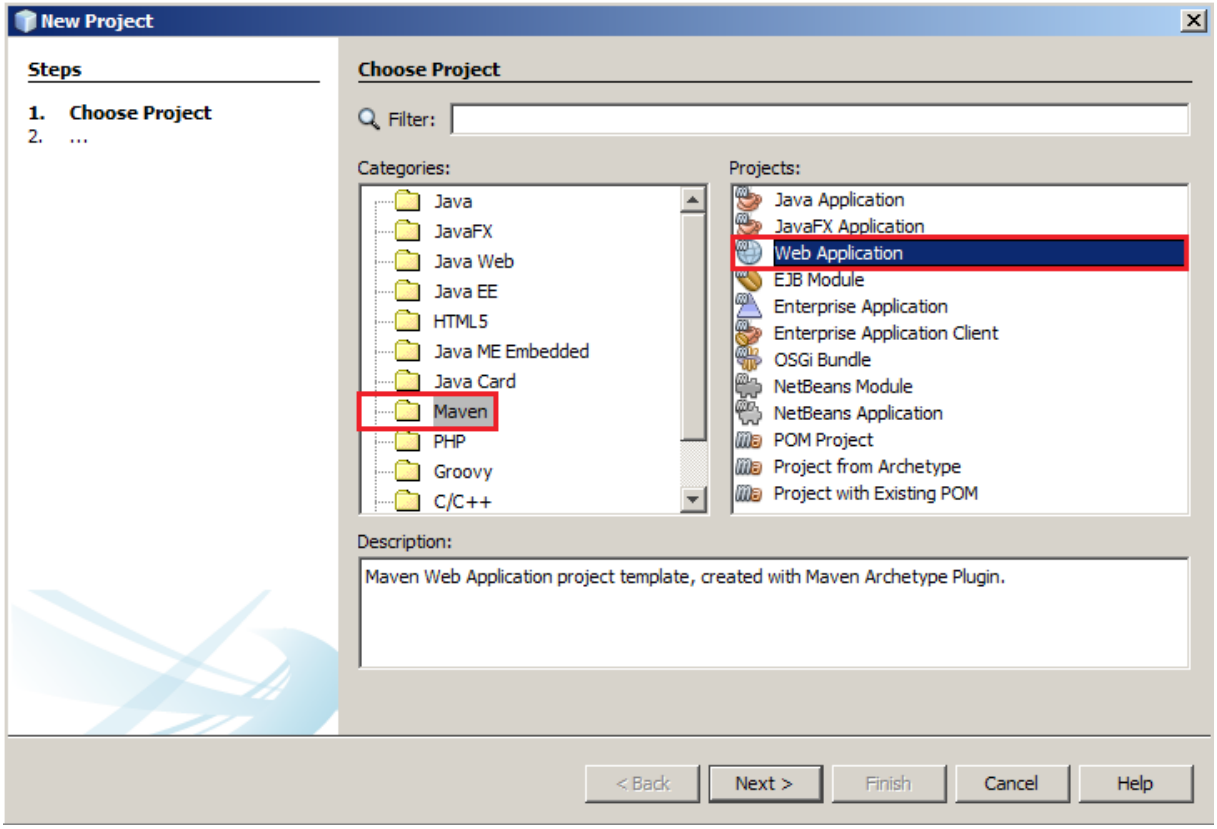

Figura 9 - Tela para escolha do tipo do projeto como *Web Application.* **Fonte: Elaborado pelo autor**

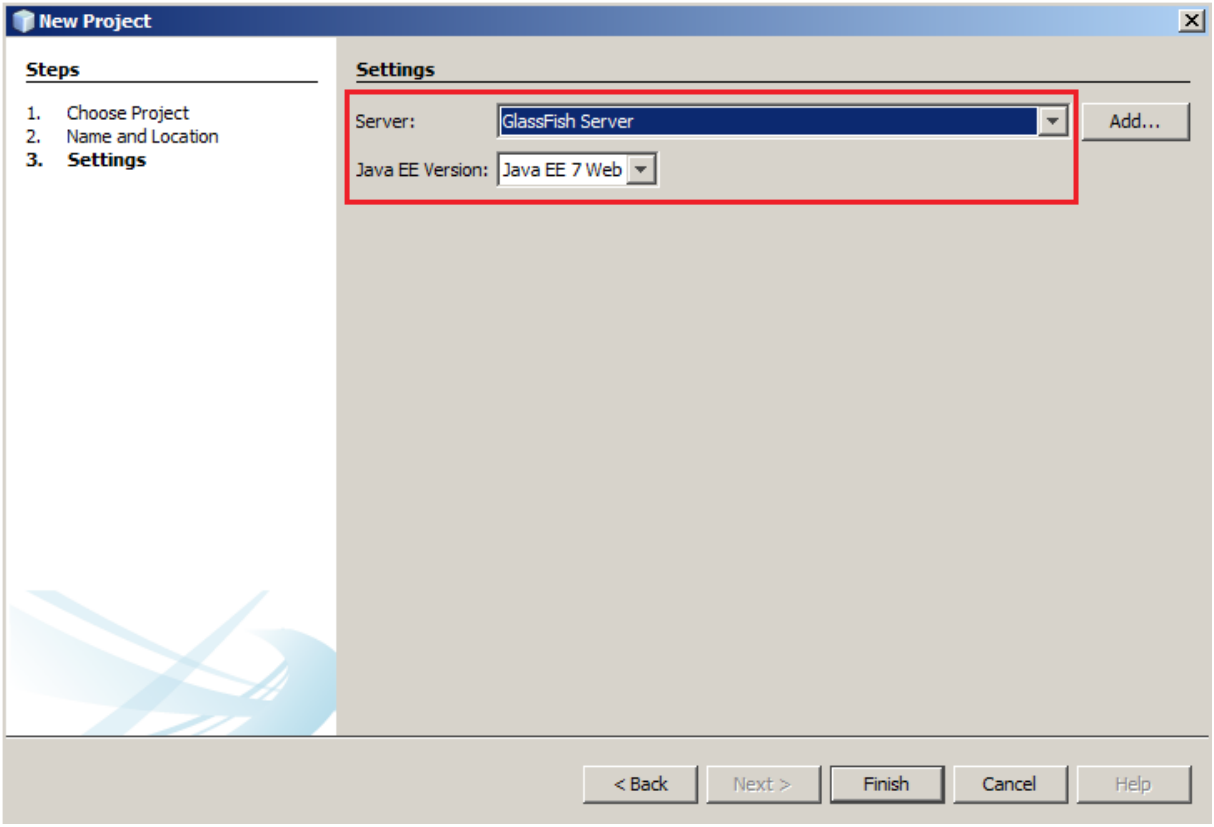

Figura 10 - Tela de configuração do *Server* e Java EE *Version.* **Fonte: Elaborado pelo autor**

Para a criação dos projetos do tipo Java *Application* basta seguir os passos da Figura 8, em seguida escolher a opção **Maven → Java Application** ou **Maven → OSGi Bundle** conforme demonstra a Figura 11. Após isto, basta definir as informações do novo projeto conforme demonstrado na Figura 5.

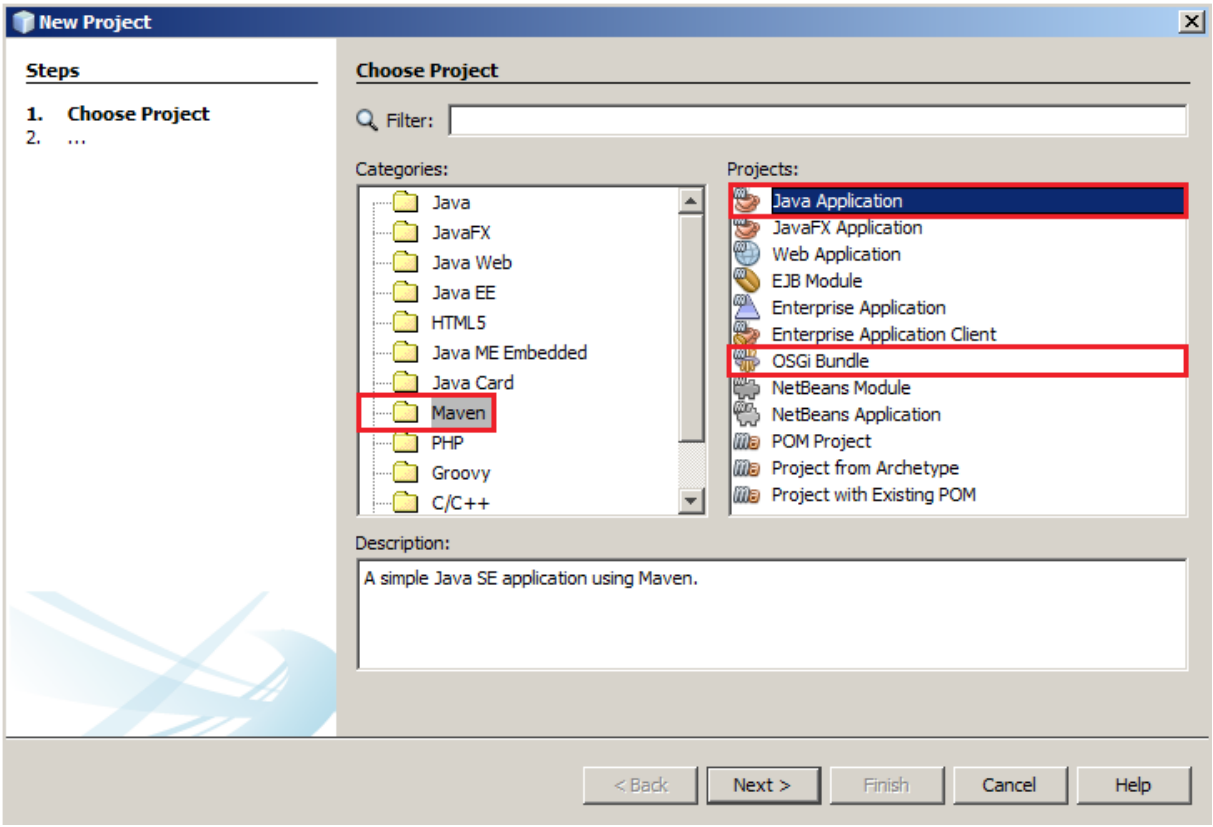

Figura 11 - Tela para escolha do tipo do projeto como Java *Application* ou OSGi *Bundle.* **Fonte: Elaborado pelo autor**

Os projetos do tipo *Enterprise* JavaBeans foram criados conforme os passos demonstrados na Figura 8**,** em seguida escolhida a opção **Maven → EJB Module** conforme Figura 12. Após isto basta seguir os passos da Figura 5 e Figura 10.

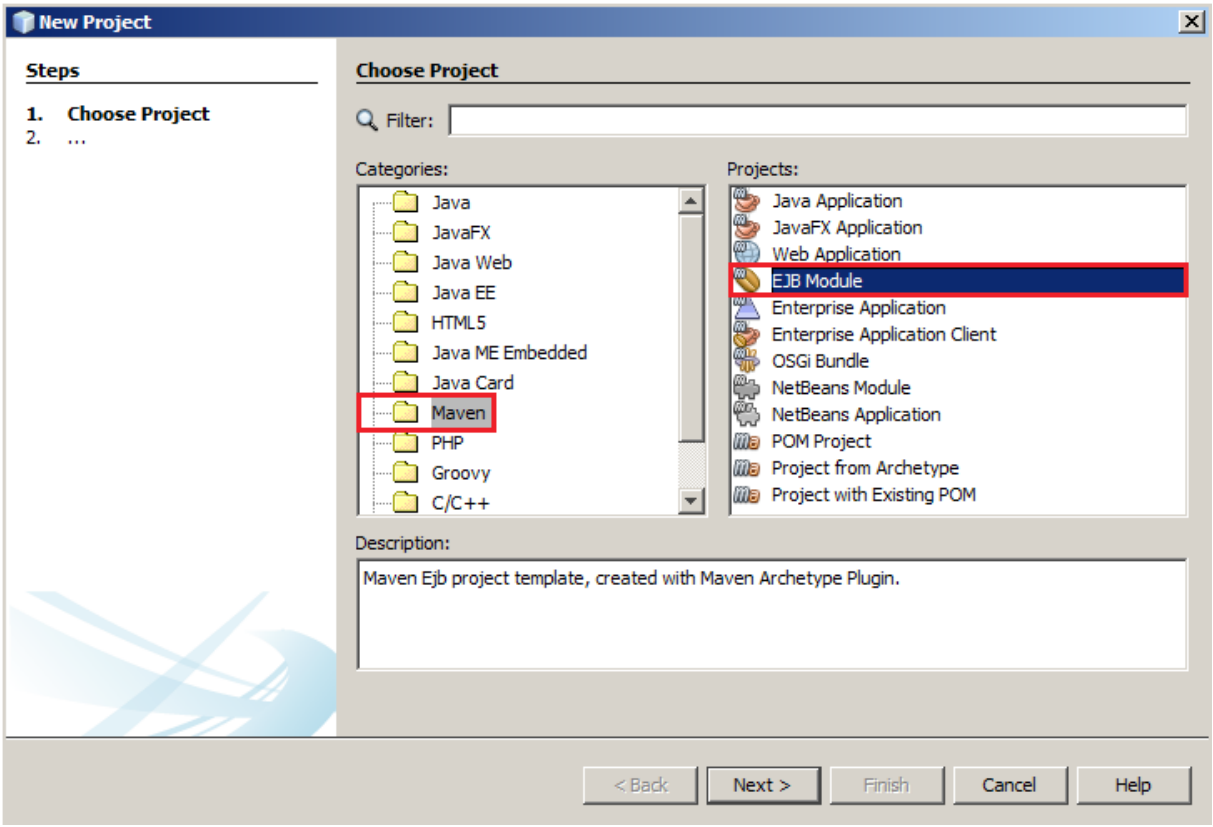

Figura 12 - Tela para escolha do tipo do projeto como *Enterprise* JavaBeans. **Fonte: Elaborado pelo autor**

Após realizada todas essas etapas, a estrutura do projeto ficou composta por um projeto do tipo POM Project que representa todo o sistema. Este projeto é composto por outros projetos do tipo POM *Project* que representa os módulos do sistema, que está composto pelos projetos do tipo *Web Application*, Java *Application* e *Enterprise* JavaBeans que são respectivamente os módulos de visão, API e *core*. Essa estrutura é demonstrada na Figura 13.

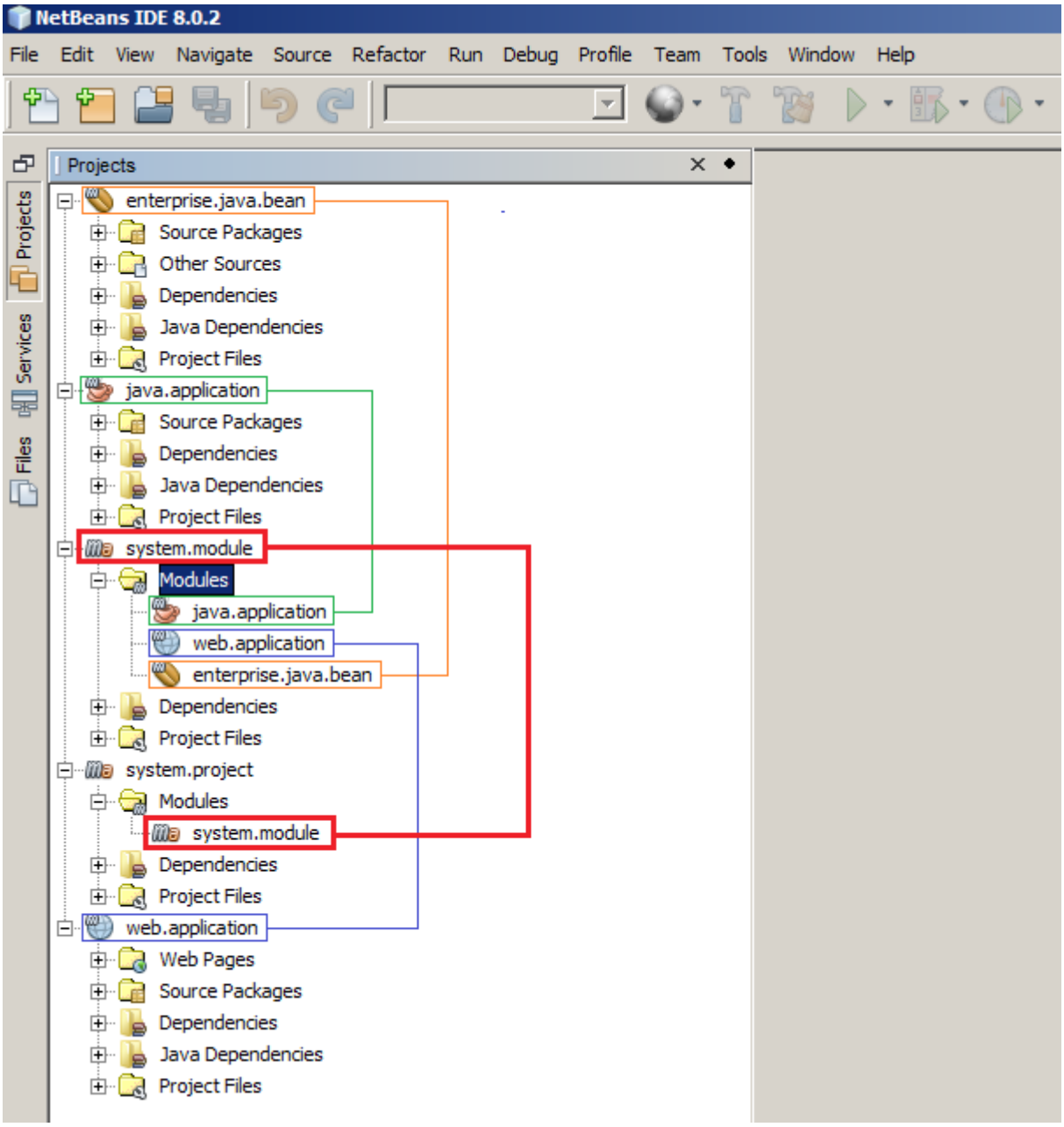

Figura 13 - Estrutura do projeto composta por um módulo do sistema. **Fonte: Elaborado pelo autor**

Após realizar a criação de toda a estrutura do projeto, configurou-se o arquivo **pom.xml** do Maven, que contém informações sobre propriedades do projeto e o gerenciamento de dependências e plugins. Quando os projetos são criados, o NetBeans se encarrega de criar esse arquivo dentro do projeto conforme a Figura 14. É através desse arquivo que o Meven gerencia o projeto.

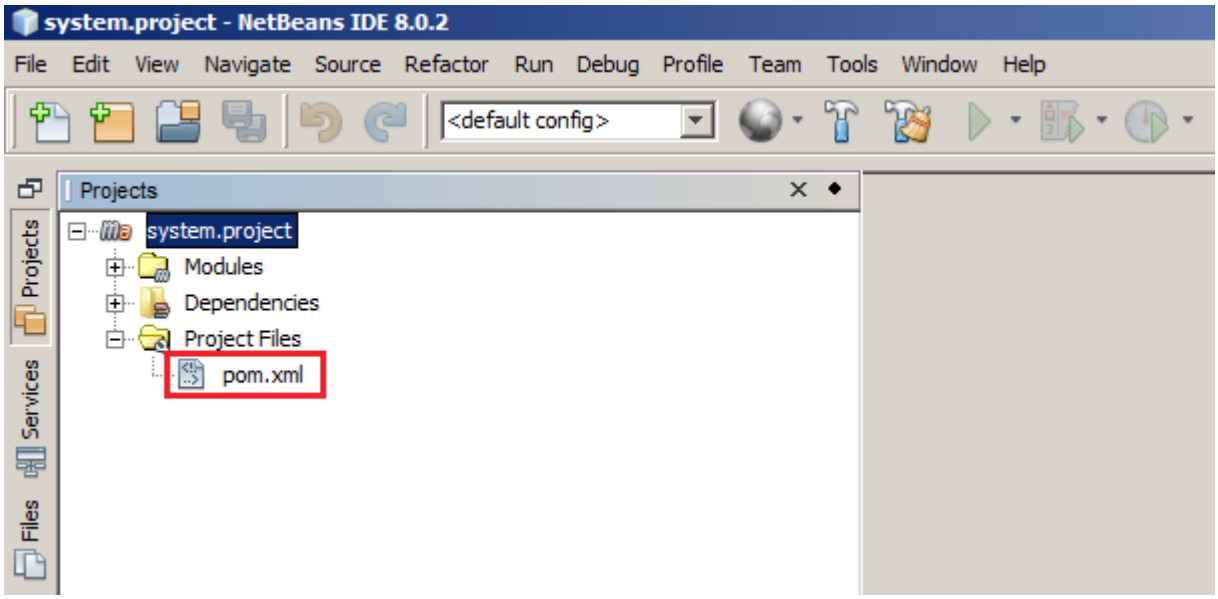

Figura 14 - Localização do arquivo pom.xml na estrutura do projeto. **Fonte: Elaborado pelo autor**

Cada projeto criado possui seu arquivo **pom.xml** que contém suas propriedades, porém como o Maven disponibiliza uma hierarquia que possibilita que propriedades do arquivo **pom.xml** do projeto principal possa ser utilizada pelos projetos que o compõe.

Para o projeto do tipo POM *Project* que representa todo o sistema, o arquivo **pom.xml** foi divido em três partes, as informações principais do projeto que estão demonstradas na Figura 15, as configurações de dependências demonstradas na Figura 16 e as configurações de *plugins* e construção do projeto, confome demonstrado nas Figuras 17, 18, 19 e 20.

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

    <groupId>groupid</groupId>                                    1
    <artifactId>system.project</artifactId>
    <version>version</version>
    <packaging>pom</packaging>

    <modules>                                                     2
        <module>system.module</module>
    </modules>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

Figura 15 - Informações principais do arquivo pom.xml. **Fonte: Elaborado pelo autor**

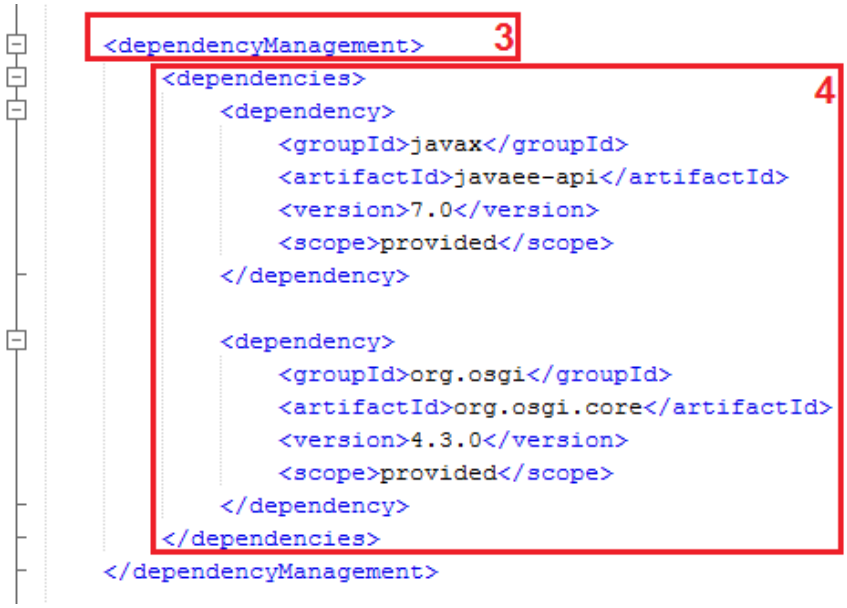

Figura 16 - Informações do gerenciamento de dependências. **Fonte: Elaborado pelo autor**


```
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <version>2.5.4</version>
                <extensions>true</extensions>
                <configuration>
                    <supportedProjectTypes>
                        <supportedProjectTypes>bundle</supportedProjectTypes>
                        <supportedProjectTypes>ejb</supportedProjectTypes>
                        <supportedProjectTypes>jar</supportedProjectTypes>
                        <supportedProjectTypes>war</supportedProjectTypes>
                    </supportedProjectTypes>
                    <instructions>
                        <_include>osgi.properties</_include>
                    </instructions>
                </configuration>
                <executions>
                    <execution>
                        <id>bundle-manifest</id>
                        <phase>process-classes</phase>
                        <goals>
                            <goal>manifest</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>bundle-install</id>
                        <phase>install</phase>
                        <goals>
                            <goal>install</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

Figura 17 - Configurações de plugins (parte 1). **Fonte: Elaborado pelo autor**

```
7
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <version>2.5.1</version>
    <configuration>
        <ejbVersion>3.1</ejbVersion>
        <archive>
            <manifestFile>
                ${project.build.outputDirectory}/META-INF/MANIFEST.MF
            </manifestFile>
            <manifestEntries>
                <Export-EJB>ALL</Export-EJB>
            </manifestEntries>
        </archive>
    </configuration>
</plugin>
```

```
8
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>2.6</version>
    <configuration>
        <archive>
            <manifestFile>
                ${project.build.outputDirectory}/META-INF/MANIFEST.MF
            </manifestFile>
        </archive>
    </configuration>
</plugin>
```

Figura 18 - Configurações de plugins (parte 2). **Fonte: Elaborado pelo autor**

Figura 19 - Configurações de plugins (parte 3). **Fonte: Elaborado pelo autor**

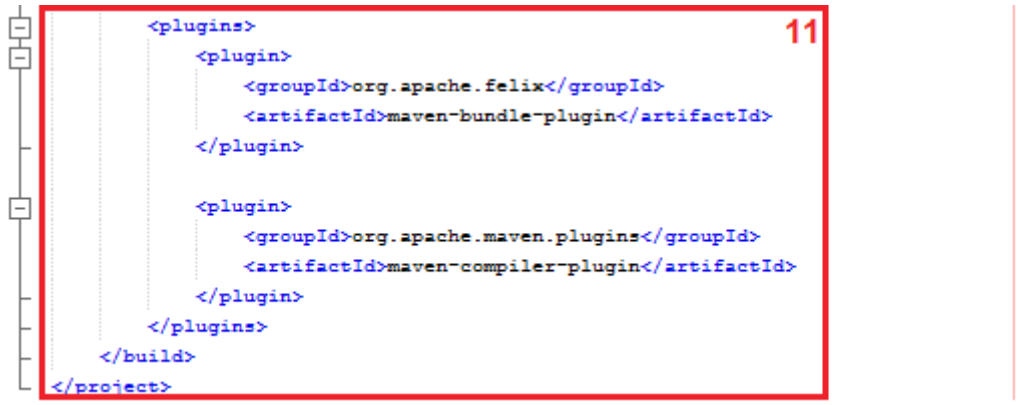

Figura 20 - Configurações de plugins (parte4). **Fonte: Elaborado pelo autor**

O *front-end* da aplicação é composto pelas telas que realizam a interface entre o sistema e o usuário, desenvolvidas utilizando as tecnologias HTML, CSS, Bootstrap, JavaScript, Angular JS e JQuery. Essa parte do sistema está disposta nos módulos de visão que são projetos do tipo *Web Application*.

1.4.5 Ferramentas e configurações

REFERÊNCIAS

APACHE. **OSGi Frequently Asked Questions**. 2015. Disponível em http://felix.apache.org/documentation/tutorials-examples-and-presentations/apache-felix-osgi-faq.html. Acesso em 16 de junho, 2015.

APACHE. **What is maven**. 2015. Disponível em http://maven.apache.org/what-is-maven.html. Acesso em 16 de junho, 2015.

APPOLINÁRIO, Fábio. **Dicionário de metodologia:** um guia para a produção do conhecimento científico. São Paulo: Atlas, 2004.

BORBA, Paulo. **Aspectos de Modularização**. 2015. Disponível em http://www.di.ufpe.br/~java/graduacao961/aulas/aula4/aula4.html. Acesso em 21 de junho, 2015.

BARTLETT, Neil. **OSGi In Practice**. 2009.

BOSSCHAERT, David. **OSGi in Depth**. Shelter Island: Manning Publications Co, 2012

CAELUM. **Apostila Desenvolvimento Web com HTML, CSS e JavaScript**. 2015. Disponível em https://www.caelum.com.br/apostila-html-css-javascript/javascript-einteratividade-na-web/. Acesso em 08 de março, 2015.

CAELUM. **Apostila Java e Orientação a Objetos**. 2015. Disponível em http://www.caelum.com.br/apostila-java-orientacao-objetos/. Acesso em 08 de março, 2015.

CLARO, Daniela Barreiro; SOBRAL, João Bosco Mangueira. **Programação em Java**. Santa Catarina: Copyleft Pearson Education, 2008.

COSTA, Gabriel. **O que é bootstrap?** 2014. Disponível em http://www.tutorialwebdesign.com.br/o-que-e-bootstrap/. Acesso em 09 de agosto, 2015.

DEITEL, Harvey Matt; DEITEL, Paul John. **Java How to Program**. 8. ed. Edson Furmankiewicz. São Paulo: Pearson Prentice Hall, 2010.

DEVMEDIA. **Introdução ao Spring Framework**. 2015. Disponível em http://www.devmedia.com.br/introducao-ao-spring-framework/26212. Acesso em 10 de março, 2015.

DEVMEDIA. **Novidades do GlassFish 3.1**. 2011. Disponível em: http://www.devmedia.com.br/novidades-do-glassfish-3-1-artigo-java-magazine-91/21124. Acesso em 19 de junho, 2015.

DURHAM, Alan; JOHNSON, Ralph. A Framework for Run-time Systems and its Visual

Programming Language. In: **OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS**. San Jose, CA, 1996, p. 20-25.

FERNANDES, Leonardo. **OSGi e os benefícios de uma Arquitetura Modular**. 37.ed. 2009. p. 27-35.

GAMA, Kiev. **Uma visão geral sobre a plataforma OSGi**. 2008. Disponível em https://kievgama.wordpress.com/2008/11/24/um-pouco-de-osgi-em-portugues/. Acesso em 09 de março, 2015.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 2002.

GONÇALVES, Antonio. **Beginning Java EE 6 Platform with GlassFish 3**. Nova York: Springer Science+Businnes Media, LCC. 2010.

JERSEY. **Jersey**: RESTful Web Services in Java. 2015. Disponível em https://jersey.java.net/. Acesso em 10 de agosto, 2015.

KIOSKEA. **Condução de reunião**. 2014. Disponível em http://pt.kioskea.net/contents/579-conducao-de-reuniao. Acesso em 16 de abril, 2015.

KNOERNSCHILD, Kirk. **Java Application Architecture:** Modularity Patterns with Examples Using OSGi. Crawfordsville: Pearson Education, 2012.

LIRA, Douglas; SCHMITZ, Daniel. **AngularJS na prática**. São Paulo: Leanpub, 2014.

LUCENA, Fábio Nogueira de. **Introdução ao Equinox**. 2010. Disponível em https://code.google.com/p/exemplos/wiki/equinox. Acesso em 09 de março, 2015.

MADEIRA, Marcelo. **OSGi – Modularizando sua aplicação**. 2009. Disponível em https://celodemelo.wordpress.com/2009/11/12/osgi-modularizando-sua-aplicacao/. Acesso em 09 de março, 2015.

MALCHER, Marcelo Andrade da Gama. **OSGi Distribuído:** deployment local e execução remota. Monografia de Seminários de Sistemas Distribuídos. Pontifícia Universidade Católica do Rio de Janeiro, 2008.

MARIE, Victor. **Bootsrapt e formulários HTML5.** 2015. Disponível em http://www.caelum.com.br/apostila-html-css-javascript/bootstrap-e-formularios-html5/. Acesso em 09 de agosto, 2015.

MAUJOR. **Site sobre CSS e Padrões Web:** Por que CSS?. 2015. Disponível em http://www.maujor.com/index.php. Acesso em 08 de março, 2015.

MAYWORM, Marcelo. **OSGi Distribuída:** Uma Visão Geral. 42.ed. 2010. p. 60-67.

MILANI, André. PostgreSQL: **Guia do Programador**. São Paulo: Novatec Editora, 2008.

Mozilla Developer Network. **CSS**. 2015. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em 08 de março, 2015.

Mozilla Developer Network. **HTML**. 2014. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em 07 de março, 2015.

Mozilla Developer Network. **JavaScript**. 2015. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em 08 de março, 2015.

Mozilla Developer Network. **JavaScript language resources**. 2014. Disponível em https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources. Acesso em 08 de março, 2015.

OLIVEIRA, Eric. **Aprenda AngularJS com estes 5 Exemplos Práticos.** 2013. Disponível em http://javascriptbrasil.com/2013/10/23/aprenda-angularjs-com-estes-5-exemplos-praticos/. Acesso em 09 de agosto, 2015.

OSGI ALLIANCE. **OSGi**. 2015. Disponível em http://www.osgi.org/Main/HomePage. Acesso em 08 de março, 2015.

ORACLE. **Difference between GlassFish Open Source and Commercial Editions**. 2011. Disponível em https://blogs.oracle.com/GlassFishForBusiness/entry/difference_between_glassfish_open_source. Acesso em 20 de junho, 2015.

SAUDATE, Alexandre. **REST**: Construa API's inteligentes de maneira simples. São Paulo: Casa do Código, 2013.

SANTOS FILHO, Walter dos. **Introdução ao Apache Maven**. Belo Horizonte: Eteg Tecnologia da Informação Ltda, 2008.

SANTOS, Wagner Roberto dos. **RESTful Web Services e a API JAX-RS**. 2009. Disponível em http://www.univale.com.br/unisite/mundo-j/artigos/35RESTful.pdf. Acesso em 08 de agosto, 2015.

SILVA, Maurício Samy. **CSS3:** Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec Editora, 2012.

SILVA, Maurício Samy. **HTML5**: A linguagem de marcação que revolucionou a web. São Paulo: Novatec Editora, 2011.

SILVA, Maurício Samy. **JavaScript:** Guia do Programador. São Paulo: Novatec Editora, 2010.

SOUZA, Arthur Câmara; AMARAL, Hugo Richard; LIZARDO, Luis Eduardo O. **PostgreSQL:** uma alternativa para sistemas gerenciadores de banco de dados de código aberto. In: Anais do Congresso Nacional Universidade, EAD e Software Livre, 2012.

STERN, Eduardo Hoelz. **PostgreSQL** - Introducao e Conceitos. 2003. Disponível em http://www.devmedia.com.br/artigo-sql-magazine-6-postgresql-introducao-e-conceitos/7185. Acesso em 10 de agosto, 2015.

USP. **Fundamentos do projeto de software**. 2015. Disponível em http://www.pcs.usp.br/~pcs722/98/Objetos/bases.html. Acesso em 21 de junho, 2015.

W3C. **About W3C**. 2015. Disponível em http://www.w3.org/Consortium/. Acesso em 07 de março, 2015.

W3C. **What is CSS?**. 2015. Disponível em http://www.w3.org/Style/CSS/. Acesso em 08 de março, 2015.