

**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

Figura 1 - Arquitetura de um módulo do sistema.....	11
Figura 2 - Arquitetura dos módulos do sistema.....	11
Figura 3 - Opção para criar projeto.....	13
Figura 4 - Tela para escolha do tipo do projeto como <i>POM Project</i>	14
Figura 5 - Tela de informações do novo projeto.....	14
Figura 6 - Criação de novo projeto como módulo.....	15
Figura 7 - Estrutura do projeto principal composto por outro projeto.....	15
Figura 8 - Criação de novo projeto como módulo.....	16
Figura 9 - Tela para escolha do tipo do projeto como <i>Web Application</i>	17
Figura 10 - Tela de configuração do <i>Server</i> e <i>Java EE Version</i>	17
Figura 11 - Tela para escolha do tipo do projeto como <i>Java Application</i> ou <i>OSGi Bundle</i>	18
Figura 12 - Tela para escolha do tipo do projeto como <i>Enterprise JavaBeans</i>	19
Figura 13 - Estrutura do projeto composta por um módulo do sistema.....	20
Figura 14 - Localização do arquivo <i>pom.xml</i> na estrutura do projeto.....	21
Figura 15 - Informações principais do arquivo <i>pom.xml</i>	22
Figura 16 - Informações do gerenciamento de dependências.....	23
Figura 17 - Configurações de plugins (parte 1).....	24
Figura 18 - Configurações de plugins (parte 2).....	25
Figura 19 - Configurações de plugins (parte 3).....	26
Figura 20 - Configurações de plugins (parte4).....	26
Figura 21 - Arquivo <i>osgi.properties</i>	27

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
BSD	<i>Berkeley Software Distribution</i>
CSS	<i>Cascading Style Sheet</i>
EJB	<i>Enterprise JavaBeans</i>
HTML	<i>HyperText Markup Language</i>
IDE	<i>Integrated Development Environment</i>
IoT	<i>Internet of Things</i>
JDBC	<i>Java Database Connectivity</i>
JEE	<i>Java Platform, Enterprise Edition.</i>
JMS	<i>Java Message Service</i>
JVM	<i>Java Virtual Machine</i>
NTT	<i>Nippon Telegraph and Telephone</i>
OSGi	<i>Open Services Gateway initiative</i>
SQL	<i>Structured Query Language</i>
SRA	<i>Systems Research and Applications Corporation</i>
TDC	<i>The Developers Conference</i>
UNIVAS	<i>Universidade do Vale do Sapucaí</i>
XHTML	<i>Extensible HyperText Markup Language</i>
XML	<i>Extensible Markup Language</i>
W3C	<i>World Wide Web Consortium</i>

SUMÁRIO

1 QUADRO METODOLÓGICO.....	5
1.1 Tipo de pesquisa.....	5
1.2 Contexto de pesquisa.....	5
1.3 Instrumentos.....	6
1.4 Procedimentos.....	8
1.4.1 Prototipação.....	8
1.4.2 Definição do software.....	9
1.4.3 Modelagem da arquitetura dos módulos.....	10
1.4.4 Desenvolvimento.....	12
REFERÊNCIAS.....	28

1 QUADRO METODOLÓGICO

Neste capítulo serão abordados e descritos os caminhos definidos e utilizados para conduzir a pesquisa. Serão apresentados o tipo de pesquisa, seu contexto, bem como os instrumentos e os procedimentos para o seu desenvolvimento.

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 respostas suficientes que satisfaçam a resolução de problemas (GIL, 2002).

Para atingir os objetivos deste trabalho, definiu-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 modular de *softwares*.

1.2 Contexto de pesquisa

Cada vez mais os *softwares* são utilizados em empresas, indústrias, computadores pessoais, *web* e dispositivos móveis, os quais são desenvolvidos por meio de práticas e

tecnologias existentes que auxiliam na sua criação, a não utilização de tais ferramentas torna o seu desenvolvimento e manutenção um processo desgastante e trabalhoso.

Esta pesquisa demonstra a utilização do *framework* OSGi, que possibilita o desenvolvimento do *software* em módulos, oferecendo melhor organização na sua estrutura, vantagens na manutenção e maior facilidade na expansão do mesmo, já que é construído em módulos.

Sempre que há a necessidade de adicionar, melhorar ou corrigir funcionalidades do *software*, é necessário interromper o funcionamento do mesmo para realizar o *deploy* de toda a aplicação novamente, para que então as novas funcionalidades passem a funcionar. A arquitetura de desenvolvimento modular utilizando a tecnologia OSGi propõe a resolução desses problemas, sendo possível aplicar tais funcionalidades sem comprometer todo o sistema, pois como o sistema é composto por vários módulos, passa a ser possível realizar o *deploy* de partes do sistema, parando somente o módulo necessário em vez de parar todo o sistema.

Softwares modularizados oferecem maior flexibilidade para atender empresas de diferentes áreas. Utilizando uma arquitetura modular com OSGi, os módulos desenvolvidos podem ser reutilizáveis, ou seja, os mesmos módulos podem ser utilizados em aplicações desenvolvidas para diferentes tipos de empresas. Por exemplo, um *software* é desenvolvido para determinado tipo de empresa que utiliza um módulo responsável pelo cadastro de clientes, ao desenvolver um novo *software* para outro tipo de empresa que também utilizará o cadastro de clientes, pode-se então reaproveitar o módulo de cadastro de clientes já existente.

Diante disso, torna-se útil o desenvolvimento de *softwares* utilizando a tecnologia OSGi, pois a mesma possibilita a criação de uma arquitetura modular que oferece vantagens na manutenção e expansão da aplicação, além de possibilitar o reaproveitamento de módulos.

1.3 Instrumentos

Durante o desenvolvimento desta pesquisa foi necessária 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.

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.

De acordo com Gil (2002), uma entrevista pode ser definida como uma técnica em que o pesquisador se apresenta frente ao investigado e lhe formula perguntas, com o objetivo de obtenção dos dados que interessam à pesquisa. A entrevista é, portanto, uma forma de diálogo assimétrica, em que uma das partes busca coletar dados e a outra se apresenta como fonte de informação.

Na maioria dos casos as pesquisas utilizam levantamentos bibliográficos e entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado, com o propósito de esclarecer e construir hipóteses sobre o problema confrontado. (COOK, SELTZER e WRIGHTSMAN, 1987).

Para se obter mais conhecimentos acerca do tema, foi realizada uma entrevista com Filipe Portes, já que este 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 TDC¹ em 2012 e 2014, apresentando uma visão ampla da tecnologia, explicando conceitos do desenvolvimento modular, suas características e seu funcionamento, além de demonstrar exemplos práticos.

A entrevista realizou-se em 1º de agosto de 2015 com duração de 2 horas, utilizando o *software* Skype². Através deste instrumento, foram sanadas dúvidas sobre a tecnologia OSGi e conceitos da arquitetura modular. A entrevista realizada foi de grande importância, pois se pode compreender melhor o funcionamento da tecnologia e sua arquitetura. Ainda houve o compartilhamento de experiências, projetos e padrões de desenvolvimento, que foram utilizados para obtenção de mais conhecimentos práticos acerca do tema da pesquisa. Com isso, percebeu-se falhas na arquitetura do *software* que havia sido criado, podendo então realizar as correções necessárias.

1 TDC – Abreviação para *The Developers Conference*

2 Skype – software gratuito que permite fazer chamadas de voz e vídeo, chat de mensagens e o compartilhamento de arquivos

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 definida uma sequência de procedimentos que foram executados de forma progressiva conforme os resultados obtidos por meio dos estudos realizados em cada tecnologia.

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 desenvolvida uma aplicação que demonstrasse a modularização de *softwares*.

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

A especificação OSGi é implementada por vários *frameworks*. Para os protótipos desta pesquisa utilizou-se o Equinox e o 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 servidor de aplicação GlassFish. Sendo assim, definiu-se também, a utilização do mesmo, 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* JavaBeans. 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, que demonstra 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 Principal:** responsável por controlar o fluxo do sistema e armazenar os recursos web (CSS, JavaScript, bibliotecas, imagens, etc) compartilhados por todos os outros módulos;
- **Módulo Sessão:** responsável por controlar os cadastros de usuários 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 à 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 do seu desenvolvimento. 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 composto por outros três módulos denominados 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 do sistema.

Essa arquitetura pode ser entendida melhor de acordo com a Figura 1 que demonstra a arquitetura de um módulo do sistema, além de suas camadas.

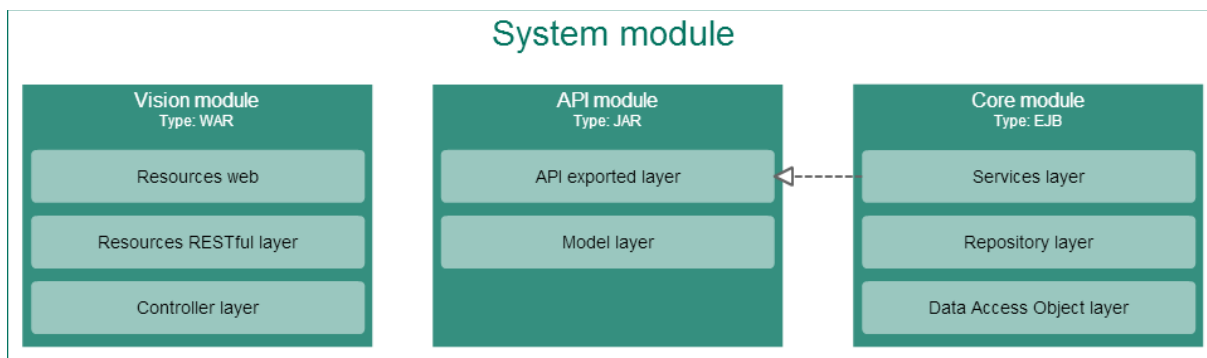


Figura 1: Arquitetura de um módulo do sistema. **Fonte:** Elaborado pelos autores

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 de todos os módulos que compõem o *software* desenvolvido.

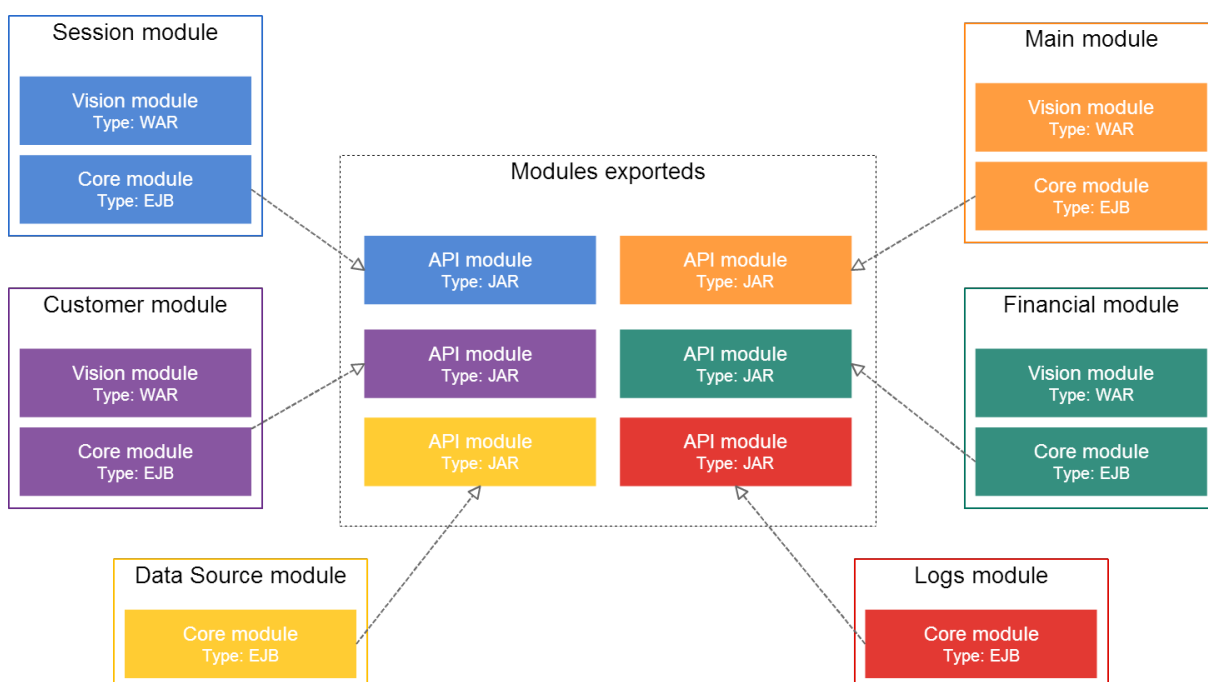


Figura 2: Arquitetura dos módulos do sistema. **Fonte:** Elaborado pelos autores

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*.

O *front-end* da aplicação é composto pelas telas 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*.

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;
- **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. Esta 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.

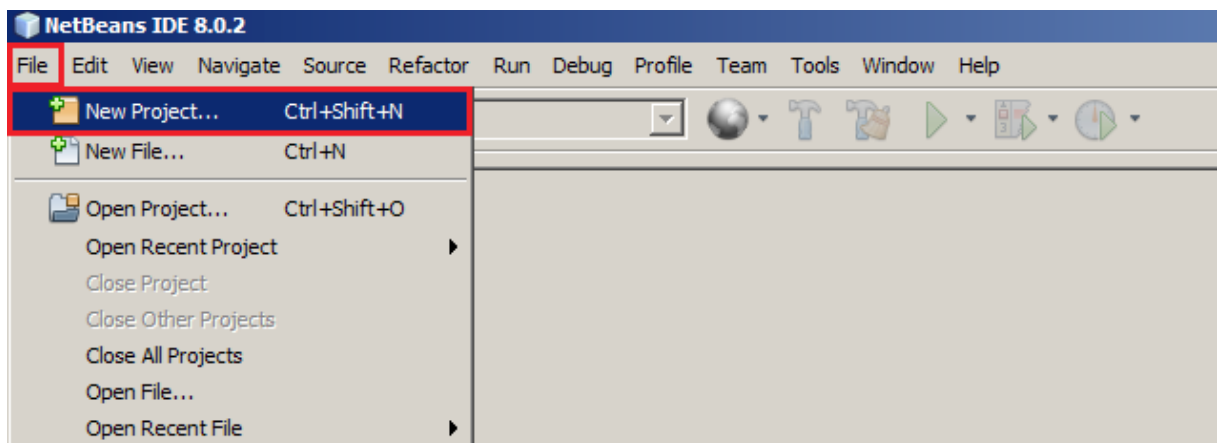


Figura 3 - Opção para criar projeto. Fonte: Elaborado pelos autores

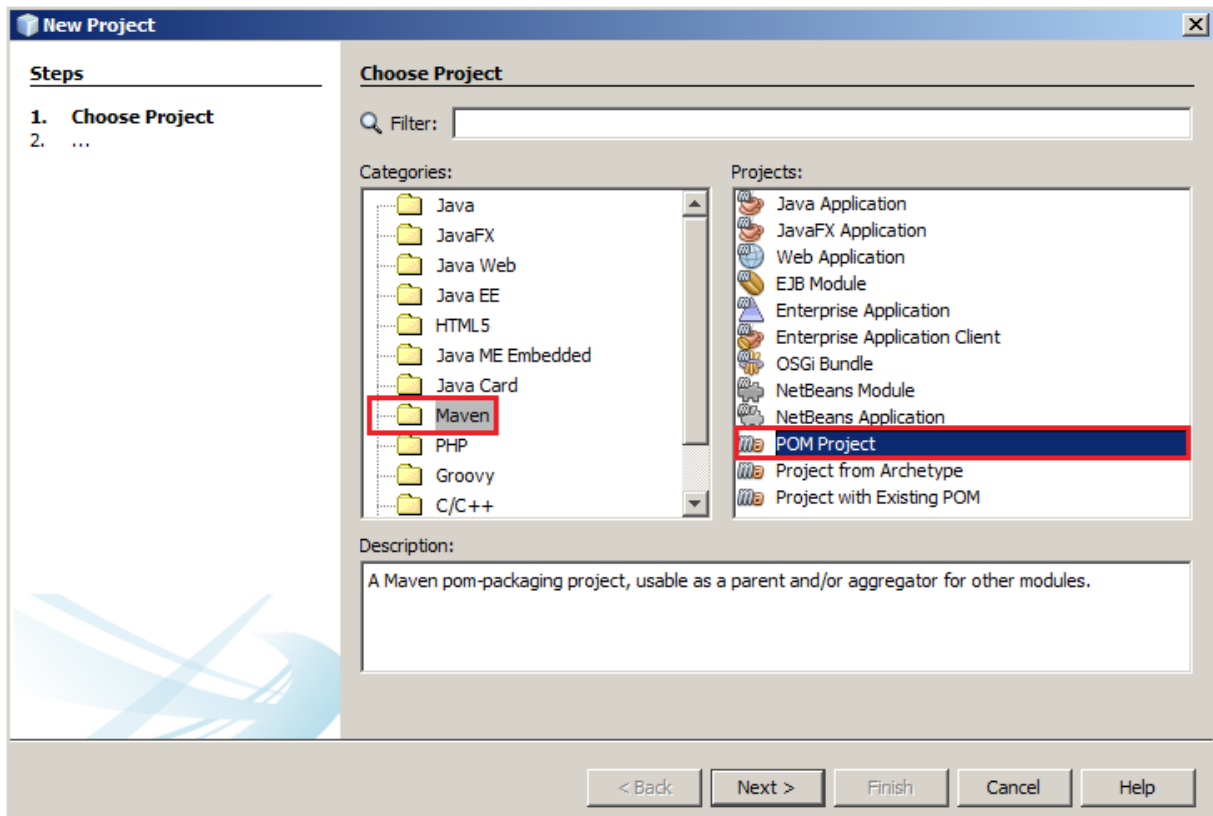


Figura 4 - Tela para escolha do tipo do projeto como POM Project. **Fonte:** Elaborado pelos autores

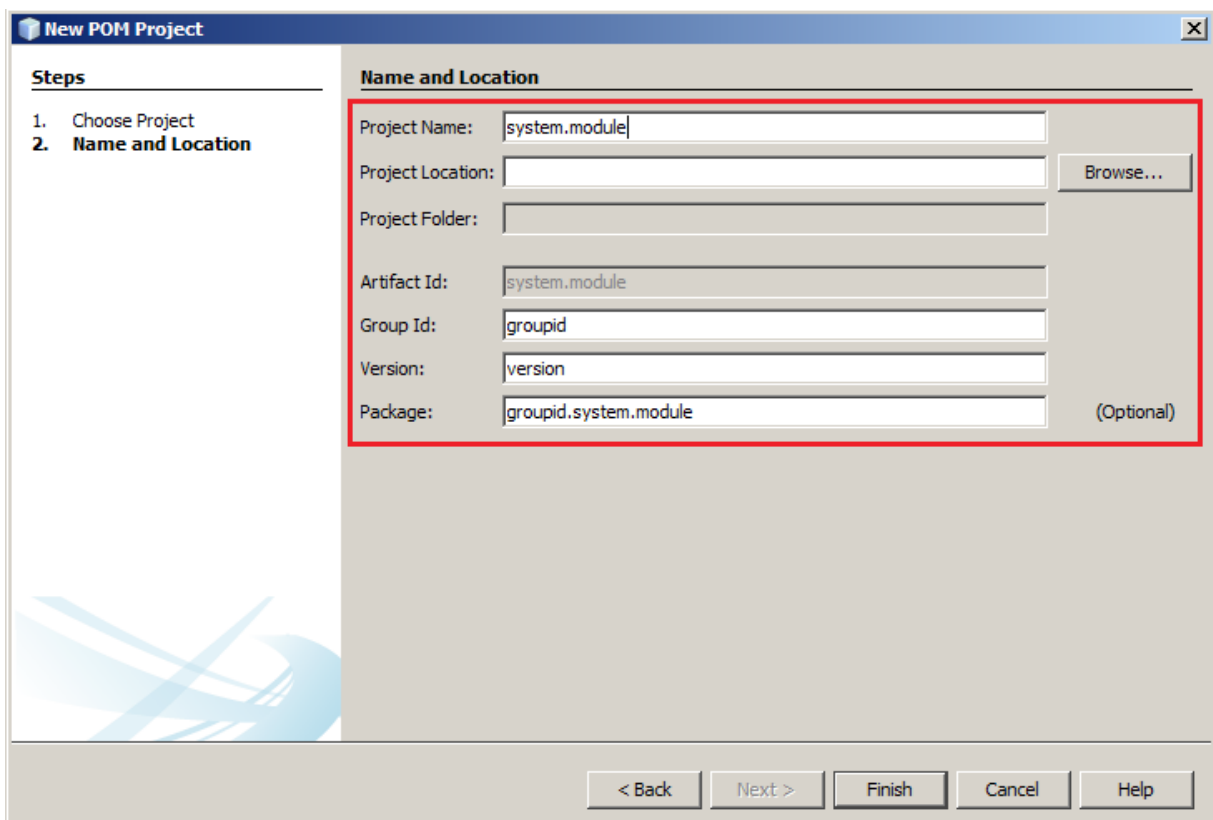


Figura 5 - Tela de informações do novo projeto. **Fonte:** Elaborado pelos autores

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.

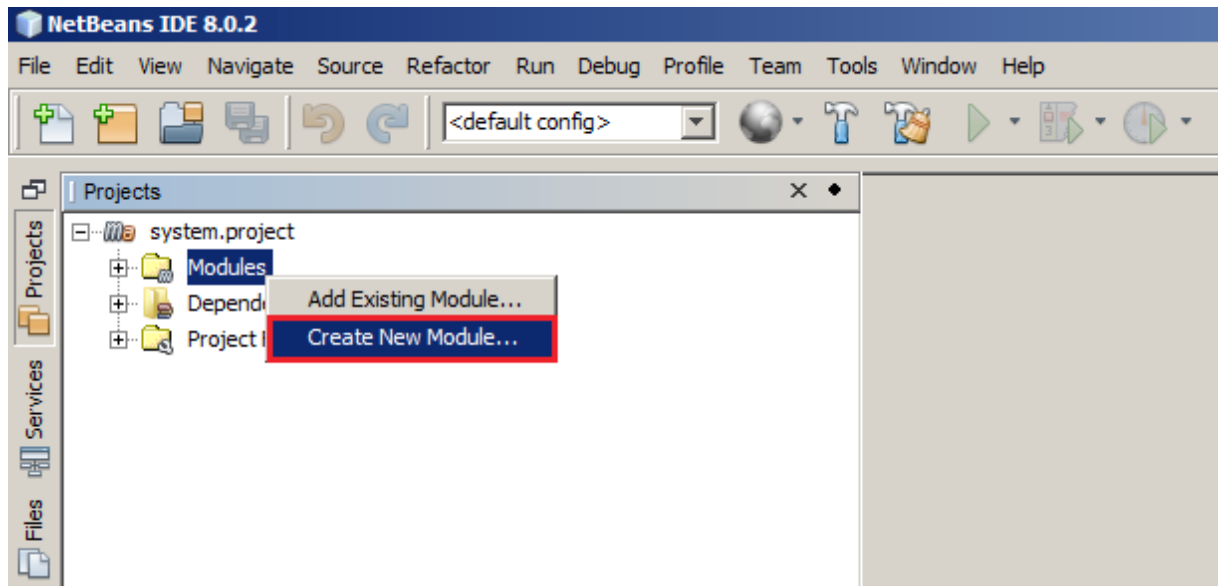


Figura 6 - Criação de novo projeto como módulo. Fonte: Elaborado pelos autores

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

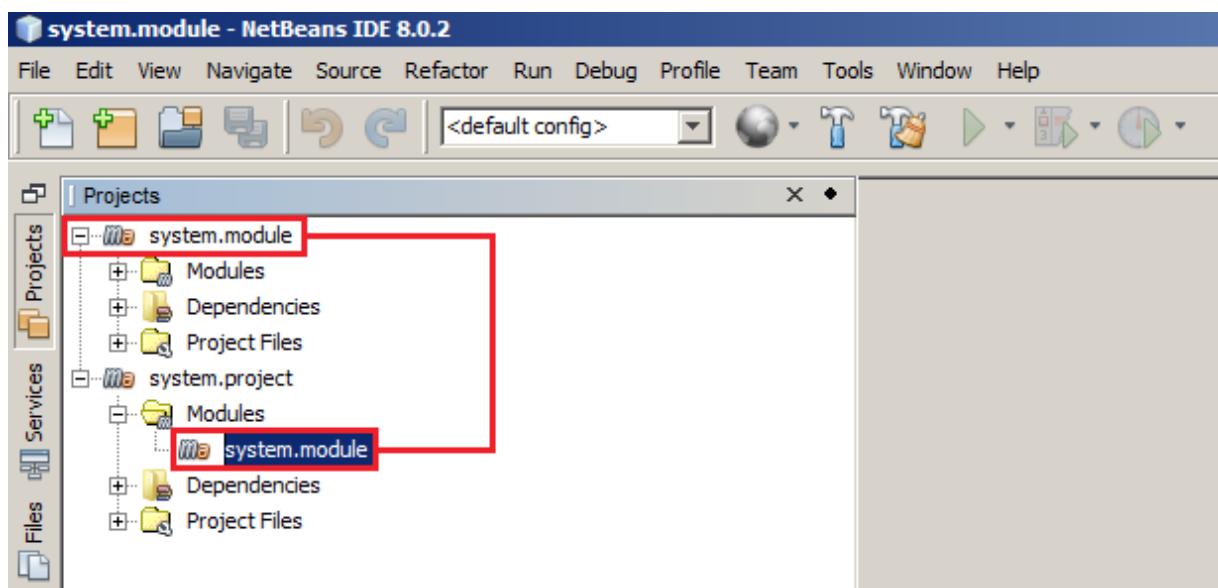


Figura 7 - Estrutura do projeto principal composto por outro projeto. Fonte: Elaborado pelos autores

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 escolhida 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.

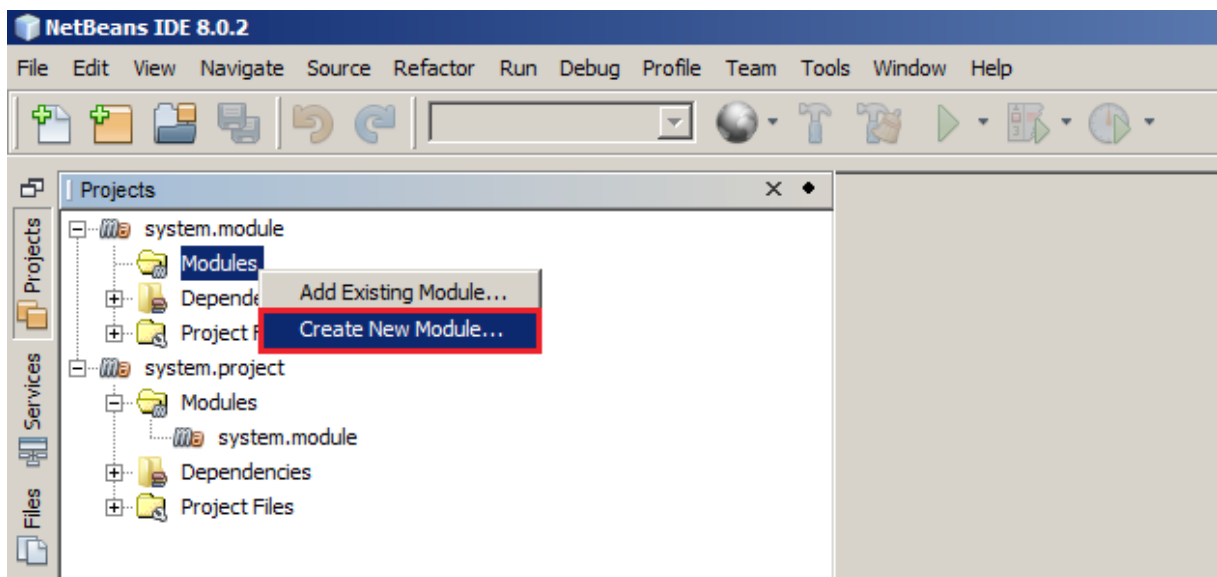


Figura 8 - Criação de novo projeto como módulo. **Fonte:** Elaborado pelos autores

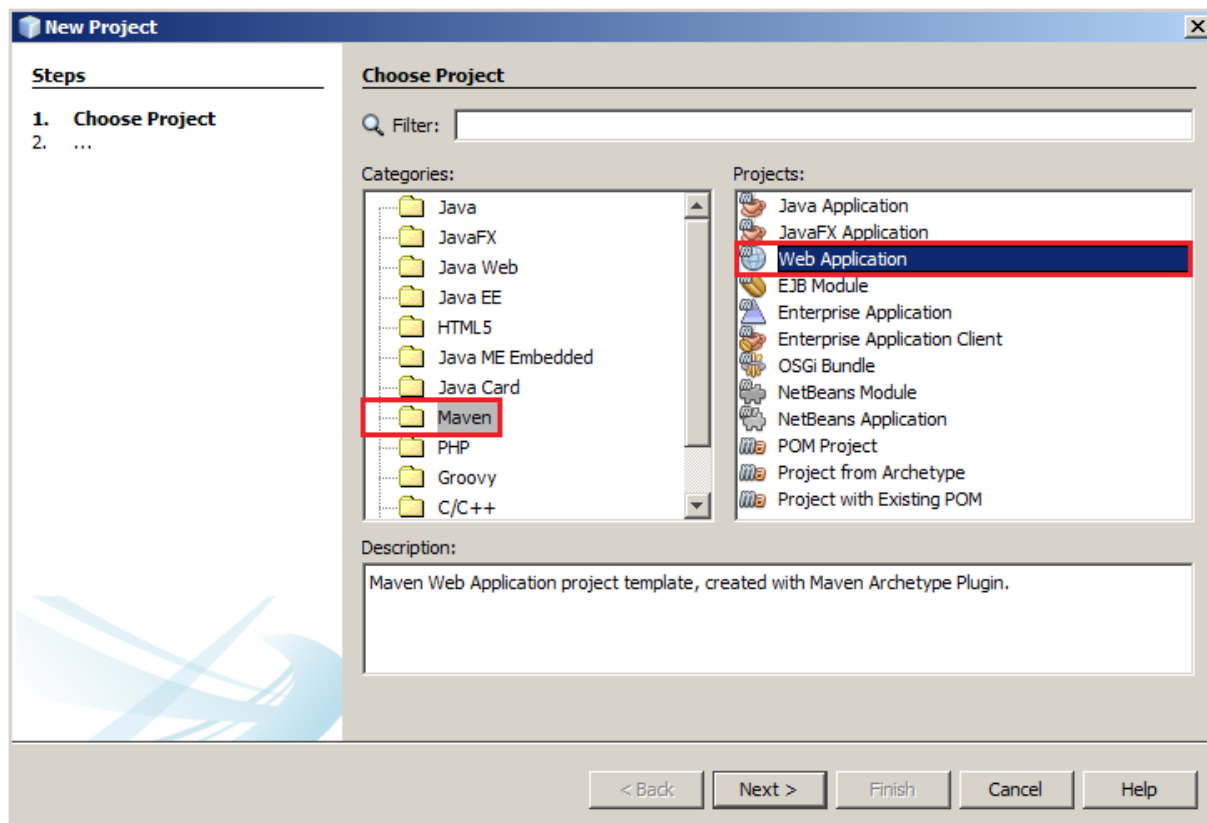


Figura 9 - Tela para escolha do tipo do projeto como *Web Application*. Fonte: Elaborado pelos autores

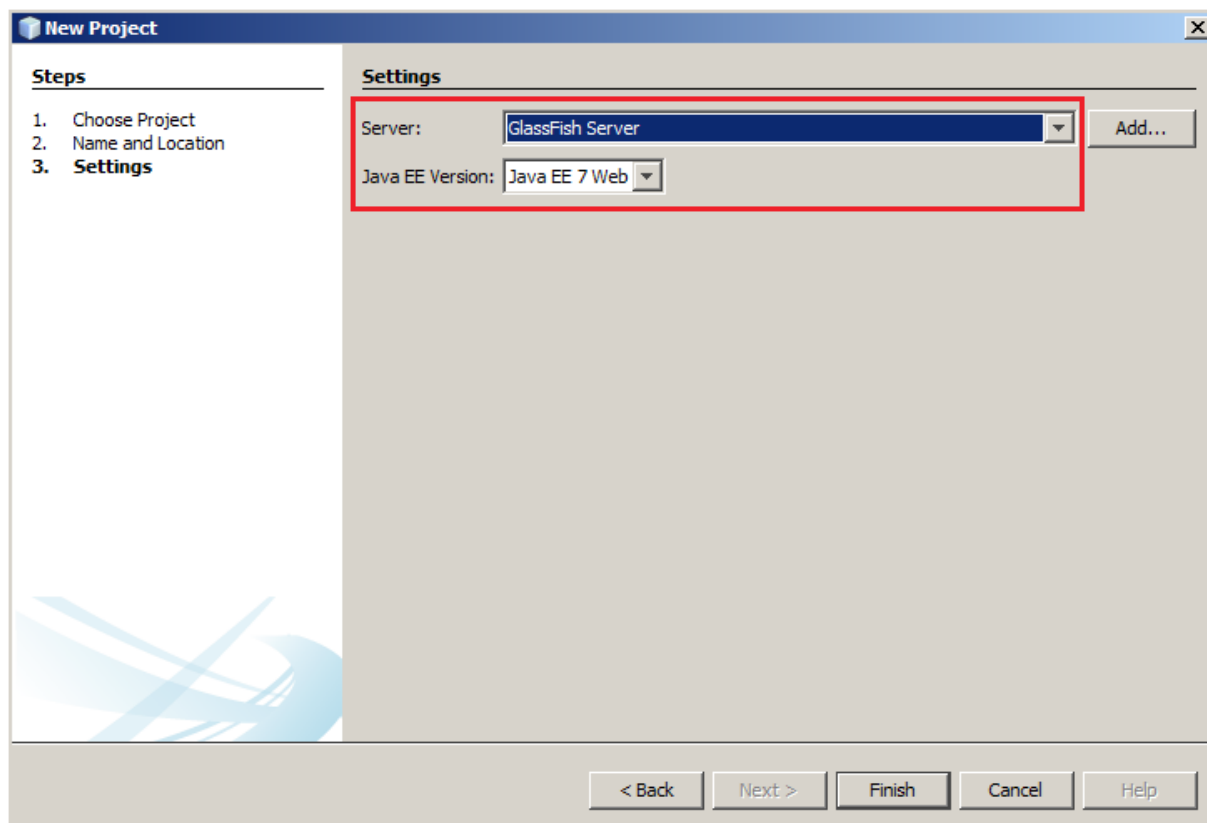


Figura 10 - Tela de configuração do *Server* e *Java EE Version*. Fonte: Elaborado pelos autores

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.

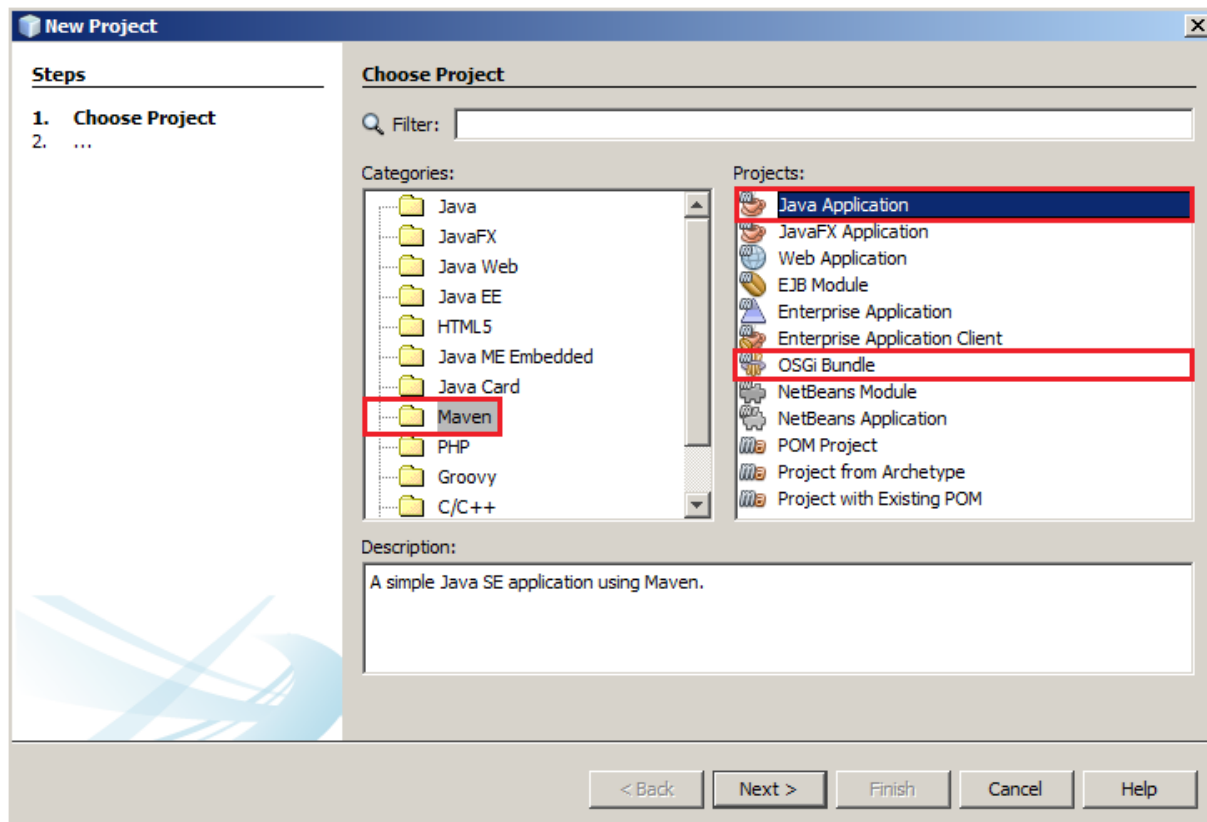


Figura 11 - Tela para escolha do tipo do projeto como *Java Application* ou *OSGi Bundle*. **Fonte:** Elaborado pelos autores

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. Em seguida, basta seguir os passos da Figura 5 e Figura 10.

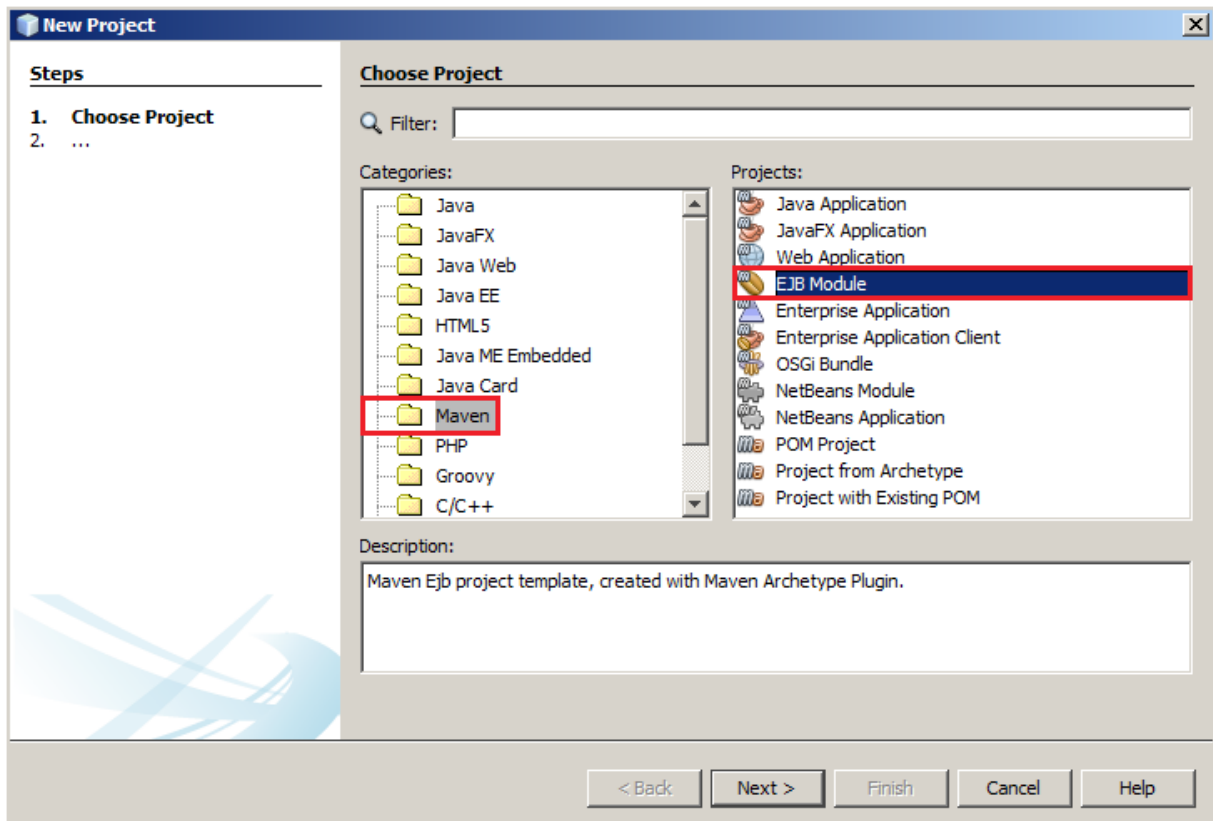


Figura 12 - Tela para escolha do tipo do projeto como *Enterprise JavaBeans*. **Fonte:** Elaborado pelos autores

Após realizadas todas essas etapas, a estrutura do projeto ficou composta por um projeto do tipo *POM Project* que representa todo o sistema. O mesmo está composto por outros projetos do tipo *POM Project* que representa os módulos do sistema, que estão compostos 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.

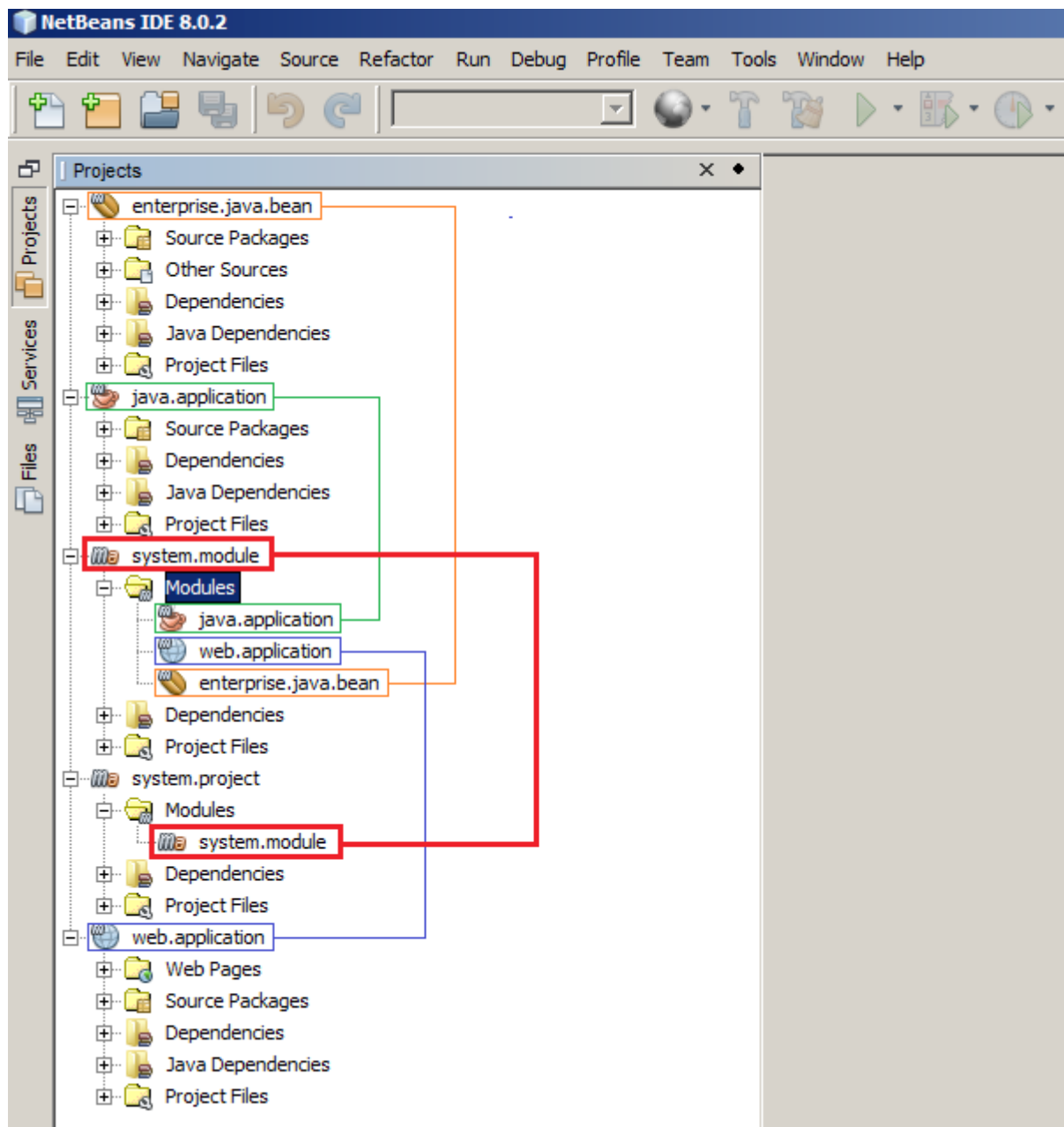


Figura 13 - Estrutura do projeto composta por um módulo do sistema. Fonte: Elaborado pelos autores

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, gerenciamento de dependências e *plugins*. Ao criar os projetos, o NetBeans se encarrega de criar esse arquivo dentro do mesmo conforme a Figura 14. É através desse arquivo que o Maven gerencia o projeto.

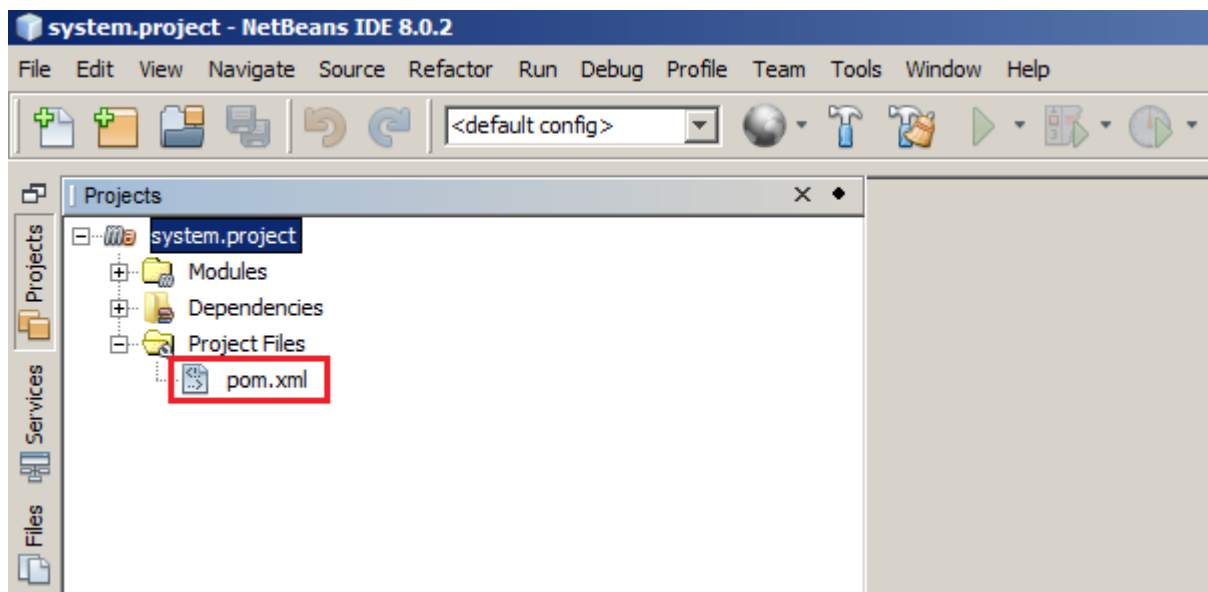
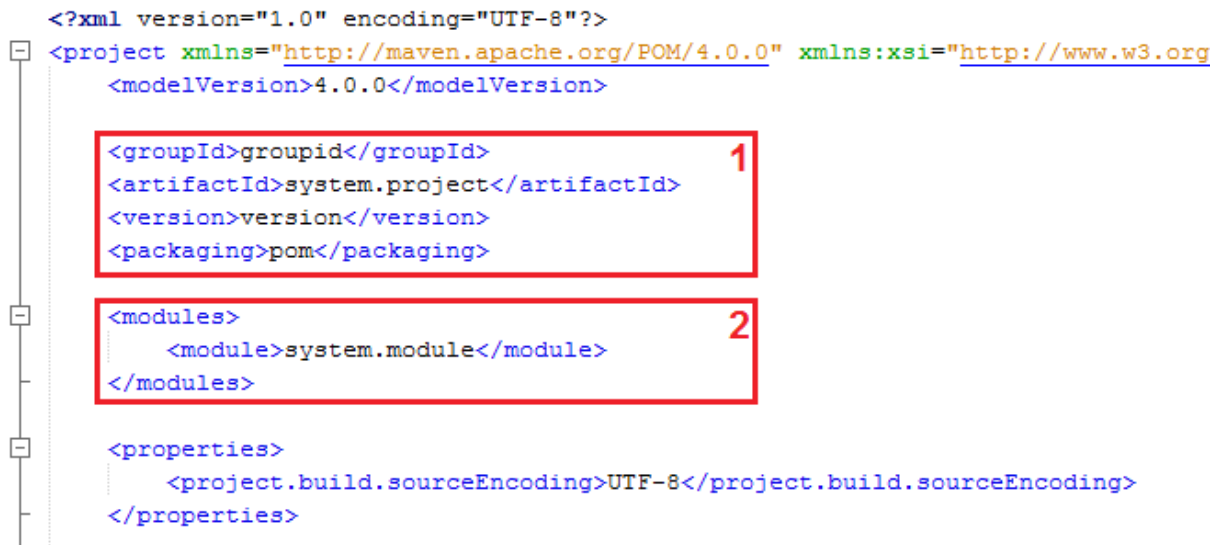


Figura 14 - Localização do arquivo pom.xml na estrutura do projeto. Fonte: Elaborado pelos autores

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, as configurações principais estão configuradas no mesmo e disponibilizadas para os outros projetos.

Para o projeto do tipo POM *Project* que representa todo o sistema, o arquivo **pom.xml** foi dividido 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, conforme 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>
  <artifactId>system.project</artifactId>
  <version>version</version>
  <packaging>pom</packaging>

  <modules>
    <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 pelos autores

As informações destacadas pelo retângulo vermelho de número 1 na Figura 15 são referentes aos dados informados na criação do projeto conforme demonstrado na Figura 5. O retângulo número 2 destaca a tag **<modules>** que identifica os projetos que compõe esse projeto.

Na Figura 16, conforme destacado pelo retângulo número 3, inicia-se o gerenciamento das dependências do Maven através da tag **<dependencyManagement>**. Dependências são bibliotecas e *frameworks* normalmente disponibilizadas por terceiros que são utilizados dentro do projeto, além dos módulos do sistema que foram desenvolvidos, que após construídos passam a se tornar dependências.

O Maven possui um repositório onde estão armazenados todas bibliotecas e *frameworks*, com isso ao informarmos no arquivo **pom.xml** alguma dependência conforme mostra o retângulo número 4, o Maven se encarrega de fazer o *download* e registrar tais dependências no projeto. Quando se trata de módulos que foram criados, o Maven busca essa dependência dentro da arquitetura do projeto e registra o mesmo.

Ao utilizarmos a tag **<dependencyManagement>**, conforme destacado no retângulo número 3, é indicado ao Maven que essas dependências serão utilizadas por outros projetos, desta maneira, o mesmo não fará o *download* dessas dependências para esse projeto, mas sim para o projeto que indicarem essa dependência em seu **pom.xml** e estiverem dentro da hierarquia desse projeto. Essa tag é muito útil devido ao versionamento das dependências, pois é definida somente uma vez a versão de cada dependência através da tag **<version>**, com isso, o projeto que for utilizar essa dependência não precisará informar sua versão. Desta

maneira, caso seja necessário trocar a versão de uma dependência, basta trocar a versão da mesma no **pom.xml** do projeto principal que sua versão será alterada em todos os outros projetos que compõe este projeto.

A *tag* **<scope>** define como a dependência será fornecida ao projeto no ambiente de execução, o valor **provided** indica que o servidor de aplicação onde o projeto será implantado é quem fornecerá essa dependência. Com o desenvolvimento modular praticamente todas as dependências têm a *tag* **<scope>** com o valor **provided**, pois são dependências que já devem estar instaladas no *framework* Apache Felix que está integrado com o servidor de aplicação GlassFish.

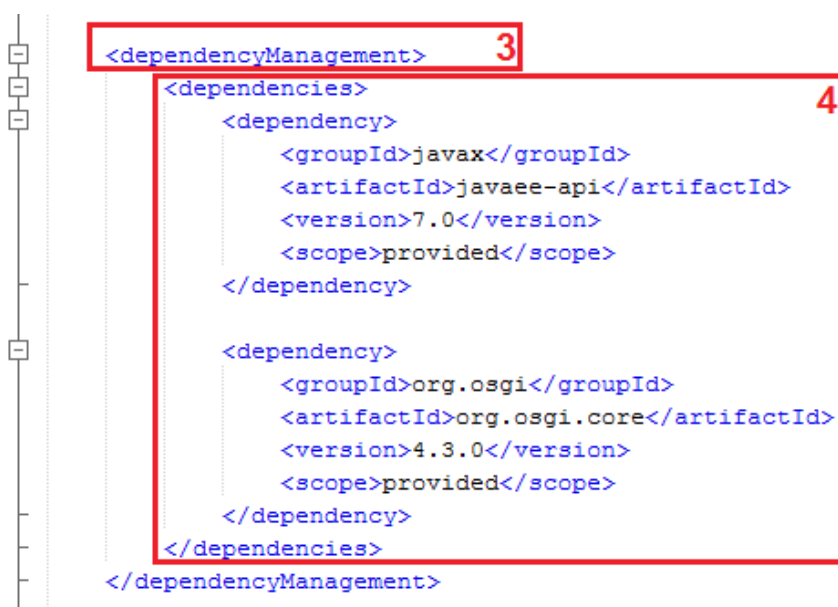


Figura 16 - Informações do gerenciamento de dependências. Fonte: Elaborado pelos autores

O Maven também possibilita a utilização de *plugins* que são utilizados para executarem tarefas durante a construção do projeto. Estão sendo utilizados 5 *plugins* para a geração dos projetos.

Os *plugins* podem ser gerenciados de forma parecida com o gerenciamento de dependências explicado anteriormente. Ao informar a *tag* **<pluginManagement>**, conforme destacada com o retângulo número 5 da Figura 17, o Maven passa a gerenciar todas os plugins dentro da mesma, porém os *plugins* ainda não estão disponíveis para os outros projetos. Para que isso aconteça é necessário informar uma *tag* **<plugins>** fora da *tag* **<pluginManagement>** conforme destacado com o retângulo de número 11 na Figura 20. O plugin informado na *tag* **<plugins>**, é uma referência ao plugin **maven-bundle-plugin** que

está dentro da *tag* `<pluginManagement>`. Ao fazer isso somente este plugin fica disponível para os demais projetos.



Figura 17 - Configurações de plugins (parte 1). Fonte: Elaborado pelos autores

O *plugin*, **maven-bundle-plugin** destacado no retângulo número 6 da Figura 17, é o principal plugin utilizado no desenvolvimento da aplicação, pois é responsável por construir o projeto como um módulo, ou seja, construir um *bundle*³ que é reconhecido como um módulo no contexto OSGi.

Neste plugin, são configurados, através da *tag* `<supportedProjectTypes>`, quais os tipos de projetos que podem ser gerados no momento da construção do pacote. Como

³ *Bundle* – são arquivos disponibilizados em um contêiner OSGi, que podem ser remotamente instalados, inicializados, atualizados, finalizados ou desinstalados dinamicamente, em tempo de execução e sem a necessidade de reiniciar todo o sistema (FERNANDES, 2009).

explicado anteriormente, a aplicação está composta por projetos do tipo *Web Application*, *Java Application* e *Enterprise JavaBeans*, para que esses projetos sejam gerados corretamente, é necessário então que essa *tag* tenha os valores **bundle**, **ejb**, **jar** e **war**.

Além de configurar os tipos de projetos que serão gerados por esse plugin, é necessário que estejam configurados no arquivo **pom.xml** os plugins referentes a cada tipo de projeto. Para projetos do tipo *Enterprise JavaBeans* é utilizado o plugin **maven-ejb-plugin**, conforme demonstrado no retângulo número 7 da Figura 18. Para os projetos do tipo *Java Application* ou *OSGi Bundle* é utilizado o plugin **maven-jar-plugin**, destacado pelo retângulo número 8. E para os projetos do tipo *Web Application* é utilizado o plugin **maven-war-plugin**, conforme destaca o retângulo 9 da Figura 19.



Figura 18 - Configurações de plugins (parte 2). Fonte: Elaborado pelos autores



Figura 19 - Configurações de plugins (parte 3). Fonte: Elaborado pelos autores

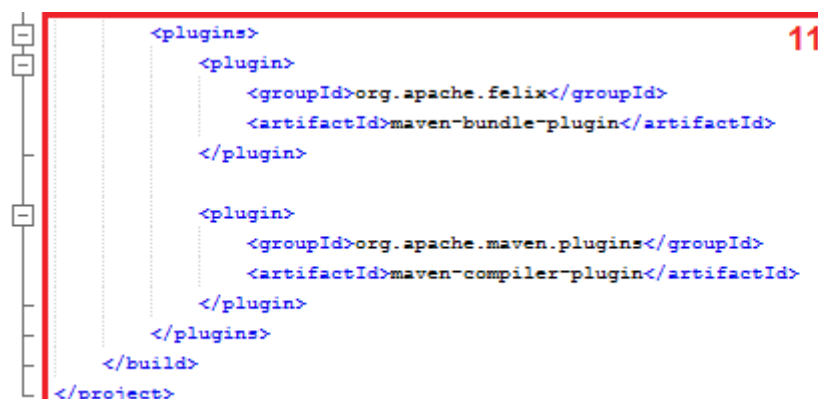


Figura 20 - Configurações de plugins (parte4). Fonte: Elaborado pelos autores

Durante a construção do projeto, o plugin **maven-bundle-plugin** gera o arquivo MANIFEST.MF que contém as informações necessárias para que o *framework* Apache Felix possa disponibilizar o projeto como um módulo. Devido a esse arquivo possuir propriedades específicas de cada projeto, foi necessário realizar outra configuração nesse *plugin*, conforme mostra o retângulo número 6 da Figura 17, uma instrução de inclusão do arquivo **osgi.properties** é configurada através da *tag* `<_include>`. Desta maneira, durante a

construção do projeto, o plugin buscará esse arquivo na raiz do projeto e colocará essas informações no arquivo MANIFEST.MF. A Figura 21 demonstra o conteúdo do arquivo.

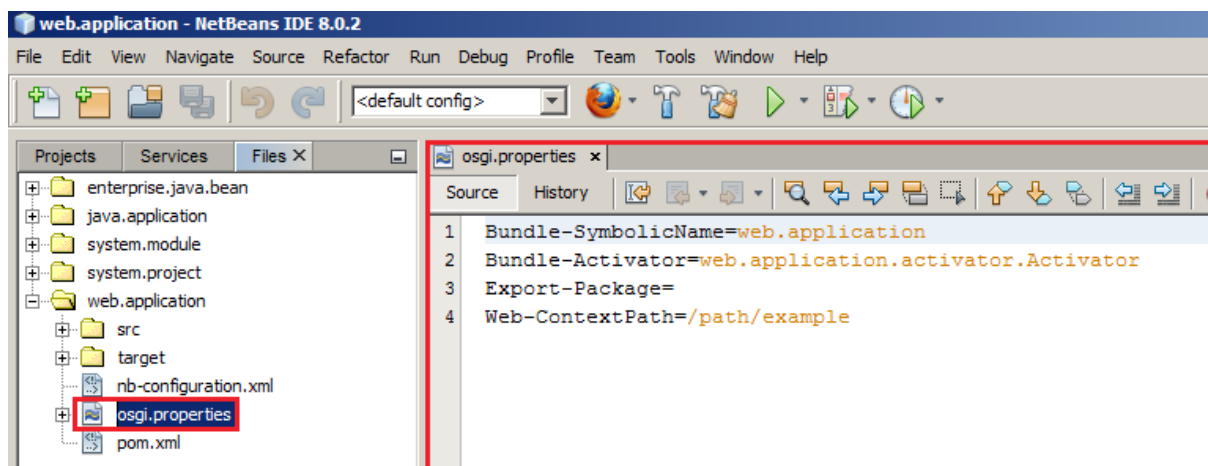


Figura 21 - Arquivo osgi.properties. Fonte: Elaborado pelos autores

O arquivo **osgi.properties** deve ser adicionado manualmente na raiz de cada projeto informando cada propriedade específica de acordo com seu tipo.

- **Bundle-SymbolicName:** é indicado o nome definido para o módulo;
- **Bundle-Activator:** é indicado o caminho da classe que implementa a interface BundleActivator.java do *framework* OSGi;
- **Export-Package:** é indicado quais pacotes são expostos por esse módulo;
- **Web-ContextPath:** é indicado o caminho da URL de acesso caso o módulo seja do tipo *Web Application*.

A utilização do **maven-bundle-plugin** auxiliou muito na geração dos módulos, pois além das configurações demonstradas anteriormente, o plugin se encarrega de preencher diversas outras propriedades do arquivo MANIFEST.MF, o que teria sido muito trabalhoso se realizado manualmente.

Após configurar adequadamente toda a estrutura do projeto, iniciou-se a programação das classes que compõe cada projeto, e também a criação das interfaces e implementações que são expostos pelos módulos.

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-e-interatividade-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 Manguiera. **Programação em Java**. Santa Catarina: Cengage Learning, 2008.

COOK, Stuart; SELTZER, Claire; WRIGHTSMAN, Lawrence Samuel. **Métodos de pesquisa nas relações sociais**. São Paulo: EPU, 1987.

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. COOK, Stuart; SELLTIZ, Claire; WRIGHTSMAN, Lawrence Samuel. Métodos de pesquisa nas relações sociais. São Paulo: EPU, 1987.