

Comparação de desenvolvimento de aplicações web com ASP.NET MVC 5 e Spring MVC 4

José Rafael Vasconcelos Cavalcante

January 5, 2011

1 Introdução

2 Configuração de ambiente de desenvolvimento

Neste capítulo será demonstrado o preparo do ambiente de desenvolvimento em um computador rodando o sistema operacional *Windows* 8.1 de 64 *bits* . Primeiramente, instala-se um sistema gerenciador de banco de dados para trabalhar tanto com a plataforma *Java / Spring MVC* quanto com a plataforma *ASP.NET MVC 5* . O banco de dados usado será o *MySQL Community Server* versão 5.6.22 (a versão mais atual até o momento de criação desta monografia). A *Integrated Development Environment* (*IDE*) utilizada para escrever código em *Java* , será o *Eclipse Luna* . O *Gradle* será usado como *build tool* e o *Apache Tomcat* como *container Java Enterprise Edition* (*JEE*). Para desenvolver em *C#* , será usado o *Visual Studio 2013 Community* . Considerando que o leitor já possui entendimentos sobre informática necessários para instalar programas no *Windows* , as instruções de instalação serão sucintas.

2.1 Instalação do MySQL

O download do instalador do MySQL Community foi feito no seguinte endereço <https://dev.mysql.com/downloads/windows/installer/5.6.html>. O instalador está disponível duas versões, web installer e off-line installer. O web installer é um arquivo pequeno que quando executado irá baixar os arquivos do MySQL para a máquina, o off-line installer é maior e vem com todos os arquivos necessários para a instalação do MySQL. Qualquer que seja o método de instalação escolhido, eles terão as mesmas opções.

Executando o instalador, é escolhida a opção Custom e na árvore de opções que aparecerá na tela a seguir, são escolhidos o MySQL Server, o MySQL Workbench, o

Connector/J (para Java) e o Connector/NET (para .NET), como mostrado na figura 1. Pode acontecer do instalador pedir para instalar o Microsoft Visual C++ 2013 como dependência do MySQL Workbench, se isso acontecer, o próprio instalador proverá um botão para instalar essa dependência.

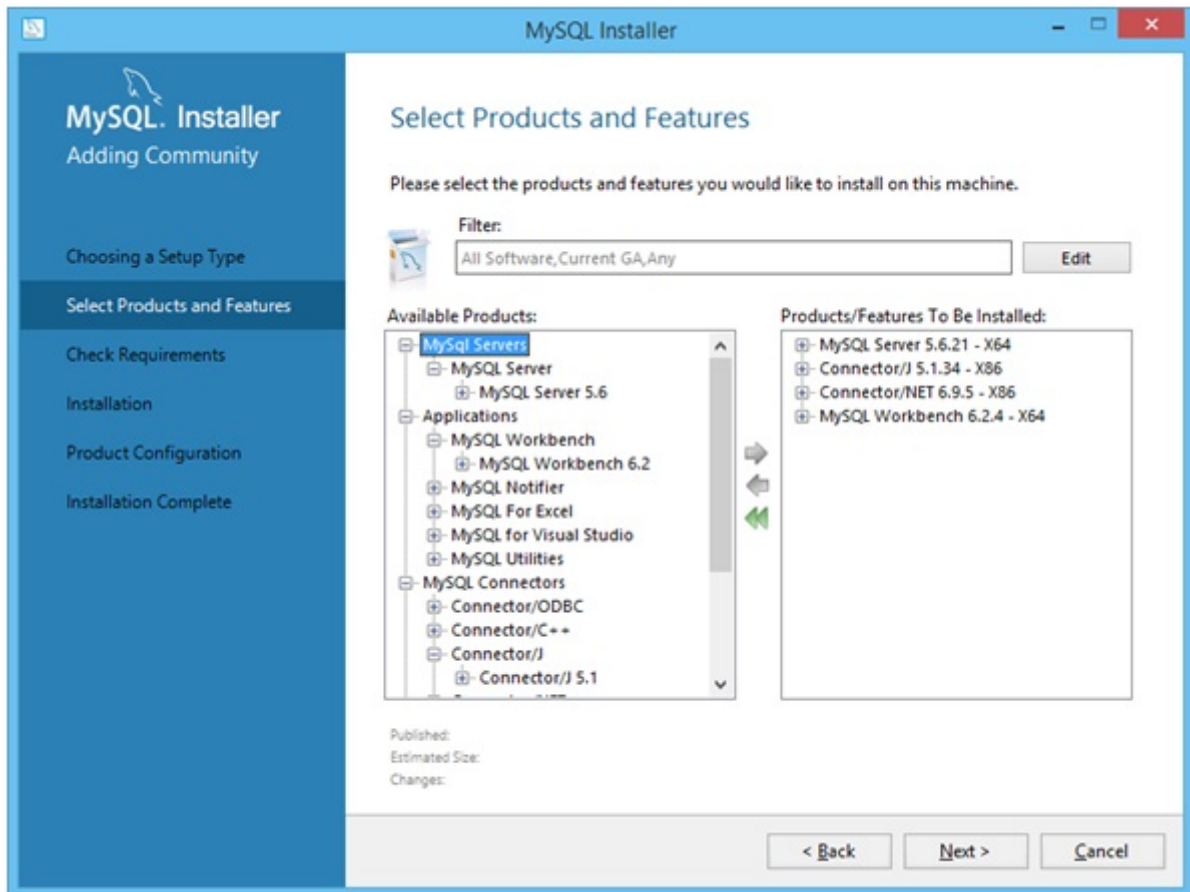


Figura 1: Opções de instalação do *MySQL*

Concluída a instalação, é hora de configurar o serviço do MySQL. Deixa-se selecionado o tipo de configuração como Development Machine e as configurações de rede padrão (protocolo TCP/IP, porta 3306). Quando for necessária a senha do usuário Root, será usada “1234”, é uma senha fraca que não se recomenda usar em ambiente de produção, mas serve para propósito de exemplo. Finaliza-se a configuração deixando marcados os restantes das opções de configuração como padrão do instalador.

Com o objetivo de testar o sucesso da instalação, o desenvolvedor pode executar o MySQL Workbench, como ilustrado na figura 2, e tentar se conectar à instância do MySQL.

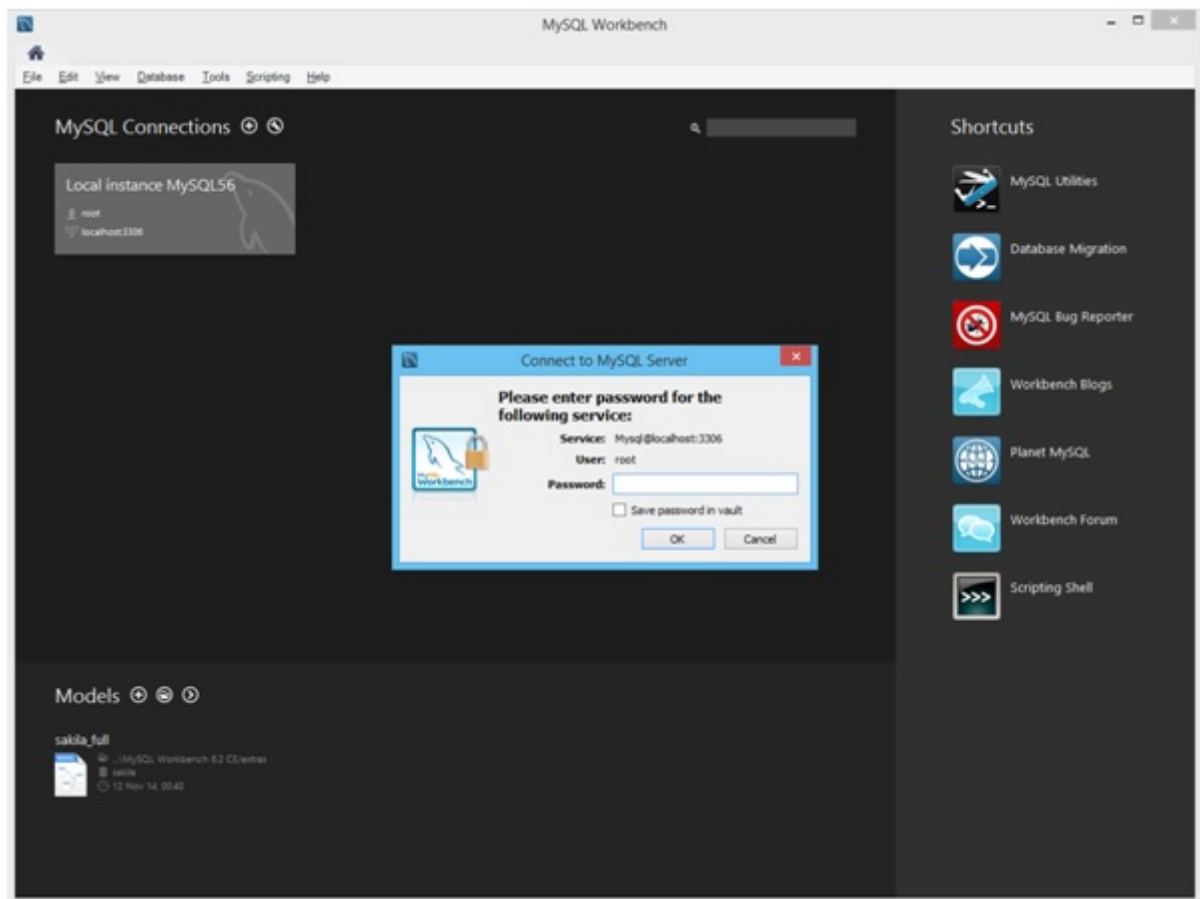


Figura 2: O *MySQL Workbench*

Para mais informações sobre o MySQL, visite a página oficial do projeto, <https://www.mysql.com/>.

2.2 Preparando o ambiente Java

Para desenvolver em Java, será utilizado o Eclipse Luna e o Java Development Kit 8 (JDK 8). Será usado o Gradle como build tool através de um plugin do Eclipse e o servidor web utilizado será o Apache Tomcat.

2.2.1 Instalando JDK 8

O instalador do JDK 8 pode ser adquirido no endereço <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>. Existem diversas versões para diversos sistemas operacionais, será usado nesse trabalho a versão para Windows de 64 bits.

Para fazer a instalação do JDK, foi executado o arquivo de instalação seguindo as suas instruções. A única configuração possível durante a instalação é a mudança da

sua pasta de destino, mas será mantido o diretório padrão como mostrado na figura 3.

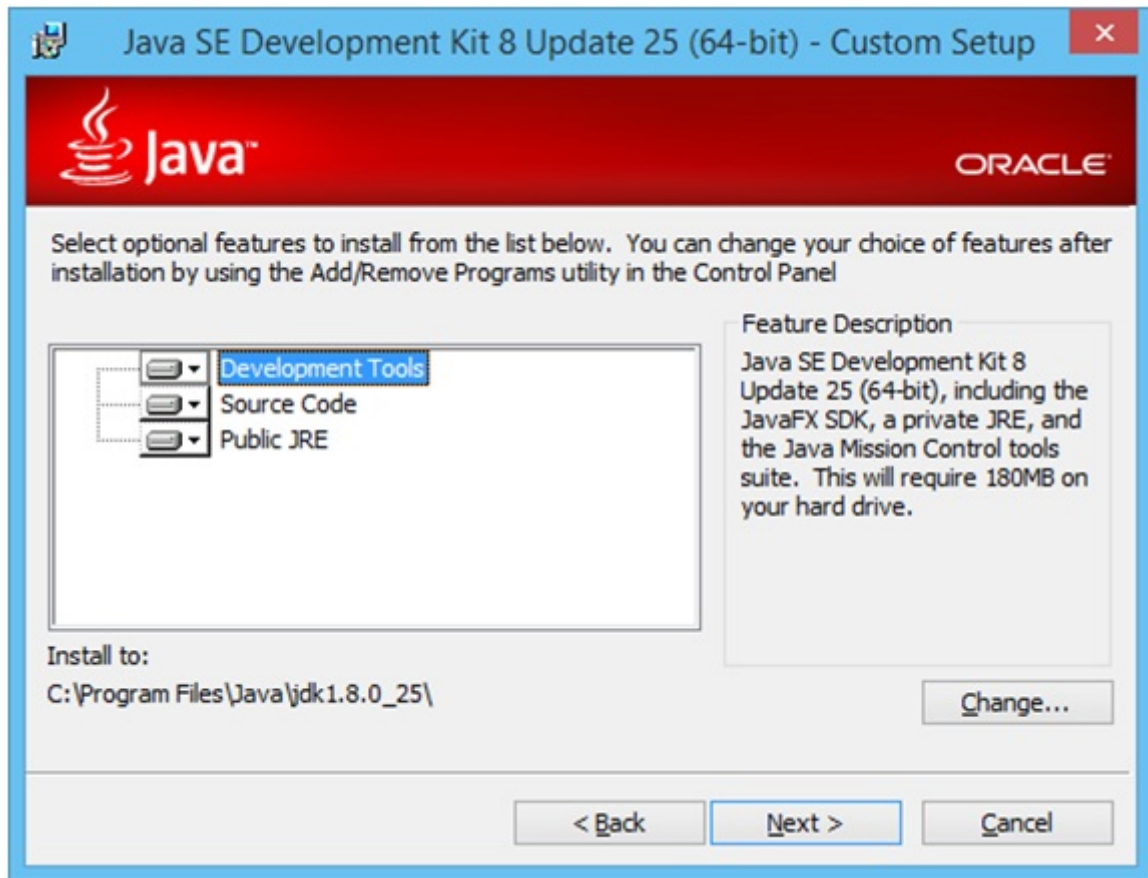


Figura 3: O instalador do JDK 8

Terminada a instalação, é necessário configurar a variável PATH para que o sistema encontre os arquivos do Java. Essas opções de configuração estão no painel de controle do Windows, no caminho Sistema/Configurações avançadas do sistema/-Variáveis de ambiente. Na janela de variáveis do sistema, é editada a variável PATH. Se adiciona o caminho onde o JDK foi instalado acrescido da pasta bin (C:\Program Files\Java\jdk1.8.0_25\bin) como mostrado na figura 4.

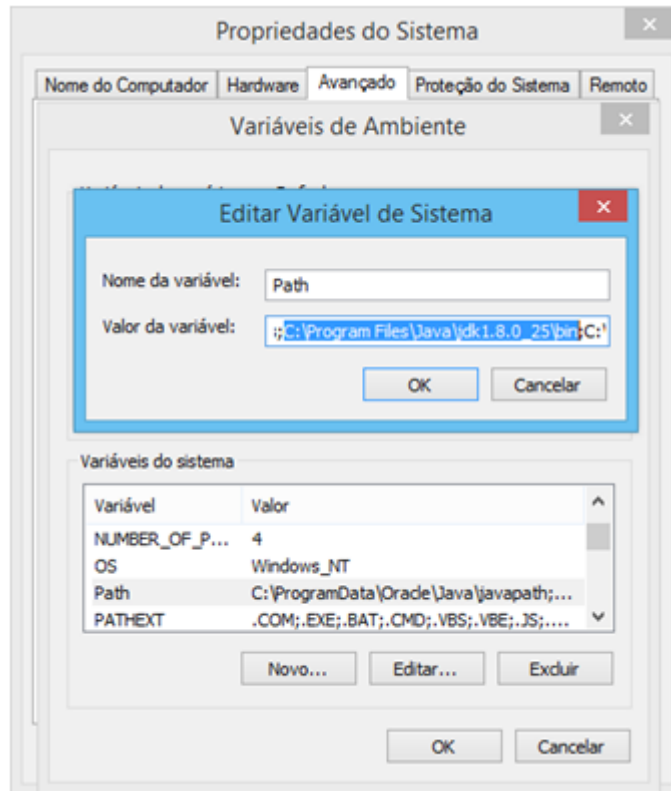


Figura 4: Configurando a variável PATH

Para testar se tudo foi instalado corretamente, abre-se uma janela do prompt de comando e digita-se o comando “java –version” (sem aspas). Se não existir problemas, será exibida na tela o número da versão do JDK instalado. Caso isso não aconteça, é aconselhável desinstalar o JDK e repetir o processo de instalação.

2.2.2 Instalando o Eclipse Luna

O Eclipse Luna para Desenvolvedores Java EE é encontrado no endereço <https://www.eclipse.org/downloads/>. Terminando o download, a pasta “eclipse” pode ser descompactada para qualquer diretório do computador. Coloca-se um atalho na sua área de trabalho para o executável do Eclipse (eclipse.exe) para facilitar o acesso.

Na primeira vez que o Eclipse é executado será exibida uma janela para configurar o Workspace padrão (uma pasta onde serão guardados projetos e configurações), como está ilustrado na figura 5.

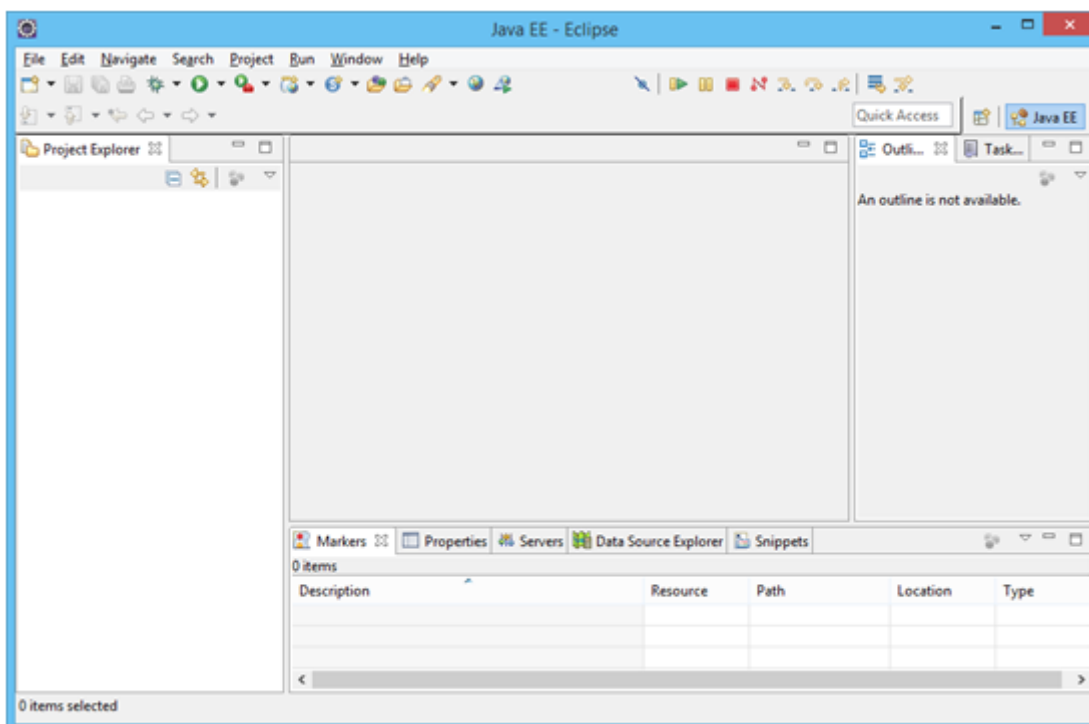


Figura 5: Eclipse recém instalado

2.2.3 Instalando o plugin do Gradle

O Gradle é a build tool que será utilizada nos exemplos do projeto *Java / Spring MVC*. Ele faz o mesmo trabalho que o ANT associado ao Ivy ou o Maven fazem, mas ele é considerado por alguns autores como o mais moderno em se tratando de build tools, pois seus scripts são escritos em Groovy em vez de XML e ele permite configurações que o Maven não permite. O Gradle está presente em todo ciclo de vida do software (ele gera artefatos, executa teste unitários, resolve dependências e executa integração contínua), mas será usada apenas uma pequena parte do que ele pode oferecer. Para mais informações sobre o Gradle acesse <https://www.gradle.org>.

No Eclipse Marketplace (repositório de plugins do Eclipse), faz-se uma pesquisa por “Gradle” na barra de buscas, entre os resultados está o Gradle IDE Pack. Esse plugin será usado nos exemplos desse trabalho. O Eclipse pede confirmação para instalação de todos os pacotes necessários, todos são selecionados e instalados. Aceita-se os termos de uso do Gradle e ao aparecer uma janela de alerta confirmando a instalação, clica-se em OK. Quando a instalação terminar, o Eclipse é reiniciado.

Para verificar se o plugin foi instalado com sucesso, a pasta Gradle deve aparecer na árvore de tipos de projetos, no menu de novos projetos, como pode ser observado na figura 6.

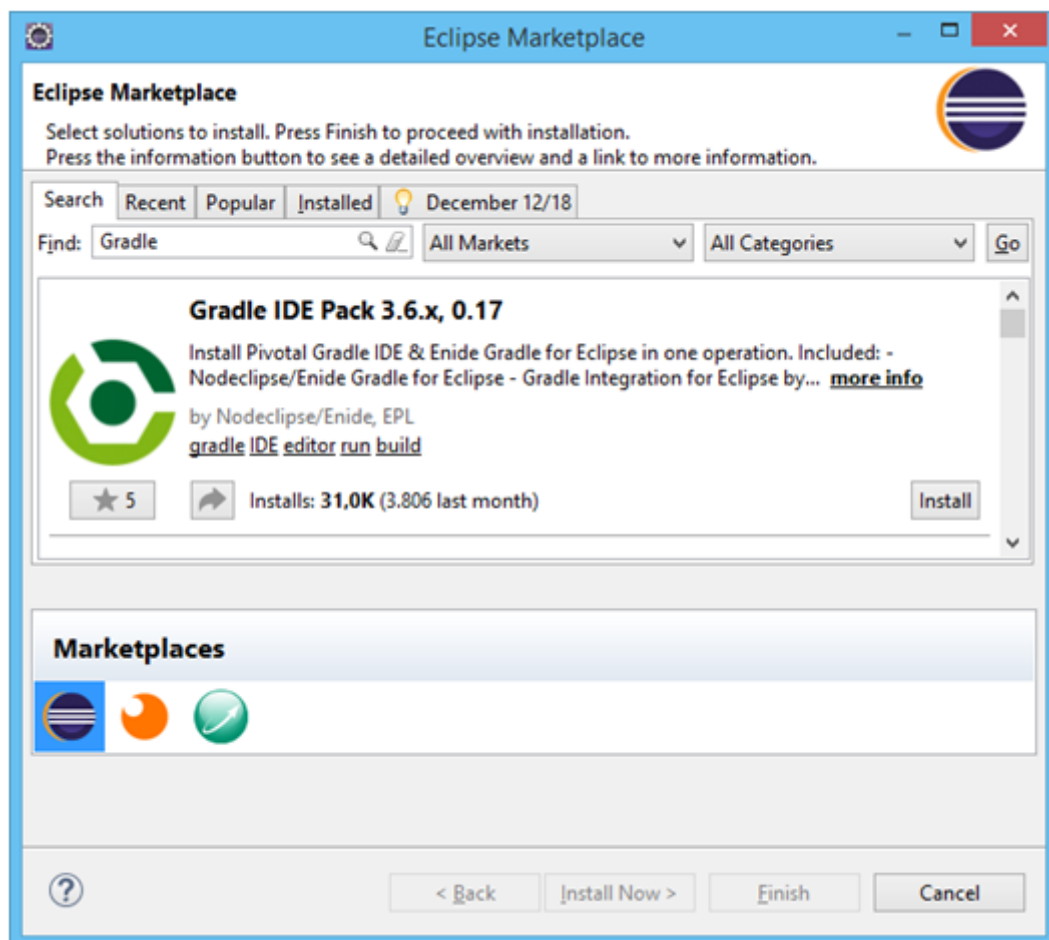


Figura 6: Instalando o plugin do Gradle

2.2.4 Instalando o Apache Tomcat 8 como servidor de desenvolvimento do Eclipse

O Java Enterprise Edition é um conjunto de especificações que precisam ser implementadas por um container (um servidor de aplicação ou servidor web) que irá executar efetivamente a aplicação. Existem diversos containers disponíveis no mercado, sendo o GlassFish o próprio container da Oracle. Nesse trabalho será usado o Apache Tomcat, pois o Eclipse tem integração nativa com ele. O Tomcat pode ser gerenciado pela aba de servidores do Eclipse.

O instalador do Tomcat 8 pode ser encontrado no endereço <https://tomcat.apache.org/download-80.cgi>. Para os exemplos, usa-se a distribuição para Windows de 64 bits no formato zip. A pasta apache-tomcat-8.0.15 pode ser descompactada para qualquer diretório no disco rígido (como exemplo será usada a raiz do disco C:).

Na aba “Servers” do Eclipse possui um link auto descritivo para adicionar um novo servidor. Clicando no link e expandindo pasta “Apache”, é escolhido o Tomcat 8 na árvore de opções, o que pode ser observado na figura 7. Na janela seguinte, em “Tomcat installation directory”, o botão “browse...” é usado para escolher o caminho de instalação do Tomcat (C:\apache-tomcat-8.0.15 no nosso exemplo). Clicando no botão

“Finish”, o Tomcat está pronto para uso com o Eclipse.

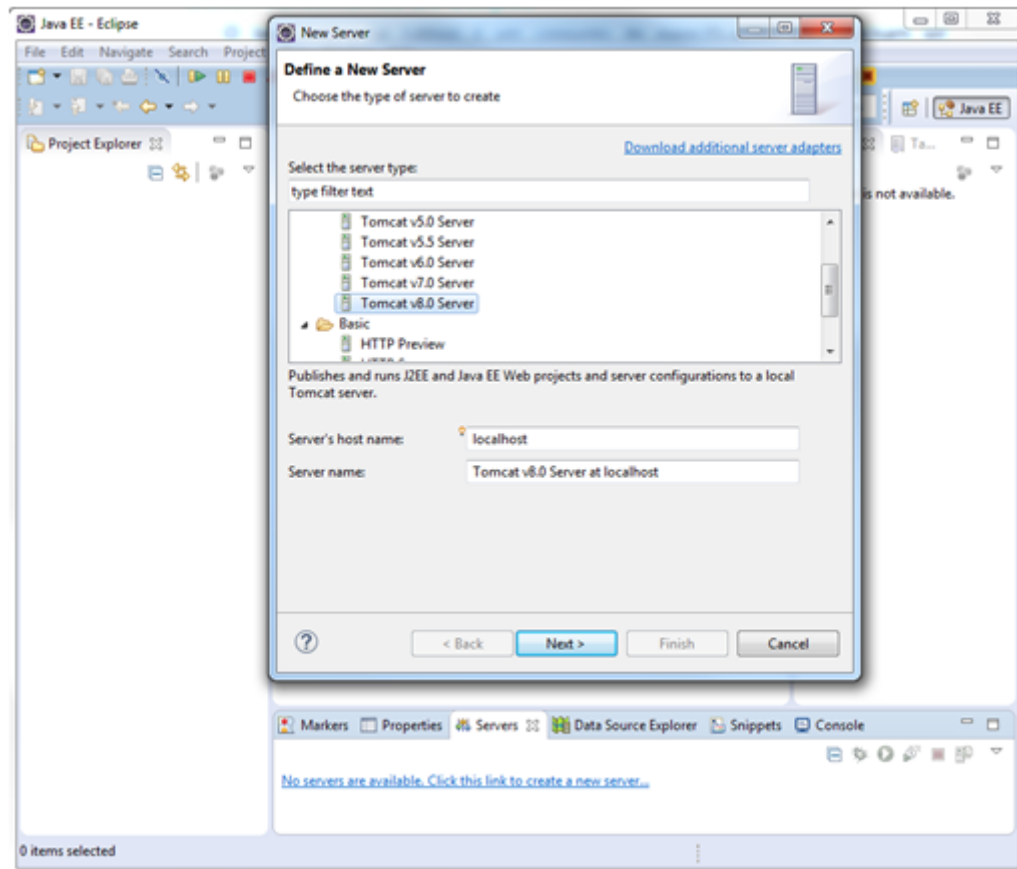


Figura 7: Janela pra adicionar servidores no Eclipse

2.3 Instalando o Visual Studio Community 2013

O instalador do Visual Studio Community 2013 pode ser encontrado no endereço <http://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx>. Na figura 8 temos uma ilustração do instalador em questão. Esse é um instalador online, ele irá baixar os arquivos do Visual Studio e opcionais selecionados à medida que a instalação for progredindo. Uma imagem do DVD de instalação offline também está disponível na sessão de downloads do site <http://www.visualstudio.com>.

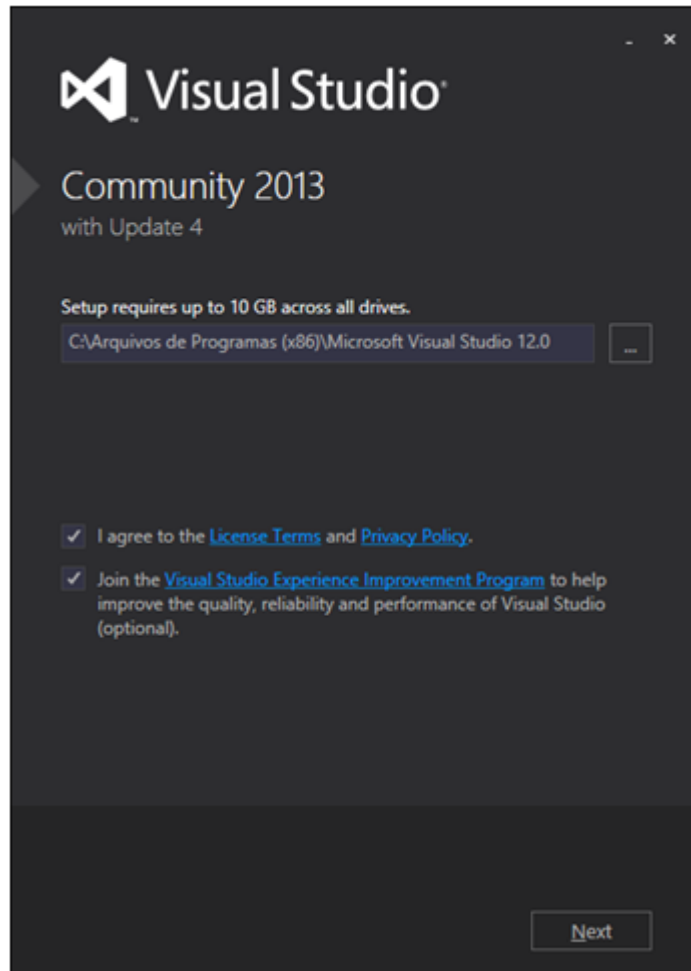


Figura 8: Instalador do Visual Studio Community 2013

Clicando no botão “Next”, a próxima tela que o instalador irá exibir uma lista de componentes opcionais como o kit de desenvolvimento do Windows Phone 8 e do Silverlight. Desses componentes opcionais, é aconselhável instalar pelo menos o Microsoft Web Developer Tools para facilitar o desenvolvimento de aplicações web.

Após a instalação, o desenvolvedor pode utilizar sua conta da Microsoft como perfil no Visual Studio e publicar suas aplicações no Microsoft Azure, porém isso é opcional. Quando se executa o Visual Studio pela primeira vez, ilustrado na figura 9, o desenvolvedor pode escolher as opções de desenvolvimento e um esquema de cores que irá usar. Como exemplo, é escolhida a opção de desenvolvimento “Web Development”.

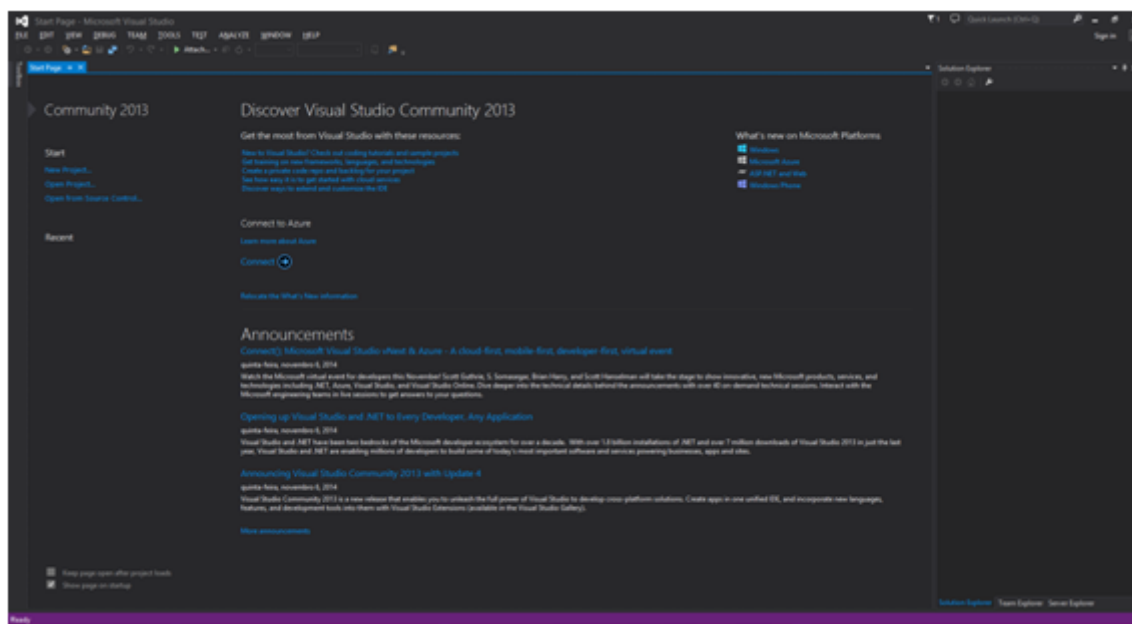


Figura 9: Tela inicial do Visual Studio Community 2013

O Visual Studio possui sua própria ferramenta de geração de builds (MsBuild), gerenciador de pacotes para obter bibliotecas de terceiros (Nuget) e um servidor de desenvolvimento minimalista baseado no Internet Information Services (servidor web do Windows Server) para executar e depurar aplicações web.

2.4 Conclusão

A preparação de um ambiente de desenvolvimento *Java / Spring MVC* requer mais passos, dentre eles instalar o kit de desenvolvimento do Java, uma IDE (Eclipse), um container Java Enterprise Edition (Tomcat) e uma build tool (Gradle), enquanto todo o software necessário para se desenvolver com .NET é adquirido em um único instalador.

As vantagens do ambiente Java é que ele fornece ao desenvolvedor mais opções de como configurar seu ambiente, além disso o tamanho em megabytes do software necessário é consideravelmente menor do que o ambiente .NET. O desenvolvedor tem a liberdade de escolher outras build tools disponíveis no mercado (Ex: Ant, Maven), outras IDEs (Ex: Netbeans) e outros servidores Java EE (Ex: Jetty, Glassfish). O lado negativo dessa liberdade é que, com essa variedade de opções, o desenvolvedor tem que pesquisar mais sobre cada solução até decidir como vai montar seu ambiente de desenvolvimento. Então depois de escolher quais produtos irá utilizar, pode ser que precise de mais algum tempo estudando como eles interagem.

A vantagem do ambiente .NET é a facilidade de obter todo o software necessário para o desenvolvimento em um único pacote, o .NET Framework, Visual Studio Community 2013 e demais ferramentas. A desvantagem é que esse pacote pode conter componentes que o desenvolvedor não precisa ou deseja, baixando arquivos desne-

cessários e tomando espaço em disco.

No próximo capítulo será demonstrado como criar um projeto de uma aplicação web nas duas plataformas.

3 Criando projetos web no padrão MVC

Nesse capítulo será mostrado como criar um novo projeto para uma aplicação web que utiliza o padrão MVC nas plataformas *Java / Spring MVC* e *ASP.NET MVC 5*. Para Java serão usadas as bibliotecas Spring Framework, que cuidará da arquitetura MVC e injeção de dependências, e o Hibernate, que cuidará da persistência e acesso a dados. Na plataforma .NET será utilizado o ASP.NET MVC 5, que cuida de toda estrutura de uma aplicação web MVC, o Entity Framework 6 para acesso a dados e o Ninject para injeção de dependências.

3.1 Criando um projeto do Gradle no Eclipse

O modelo de projeto usado nos exemplos desse trabalho será o Gradle Project. Esse modelo de projeto está localizado na pasta Gradle na árvore de novos projetos do Eclipse. Na figura 10 podemos observar a localização do modelo e a criação do novo projeto.

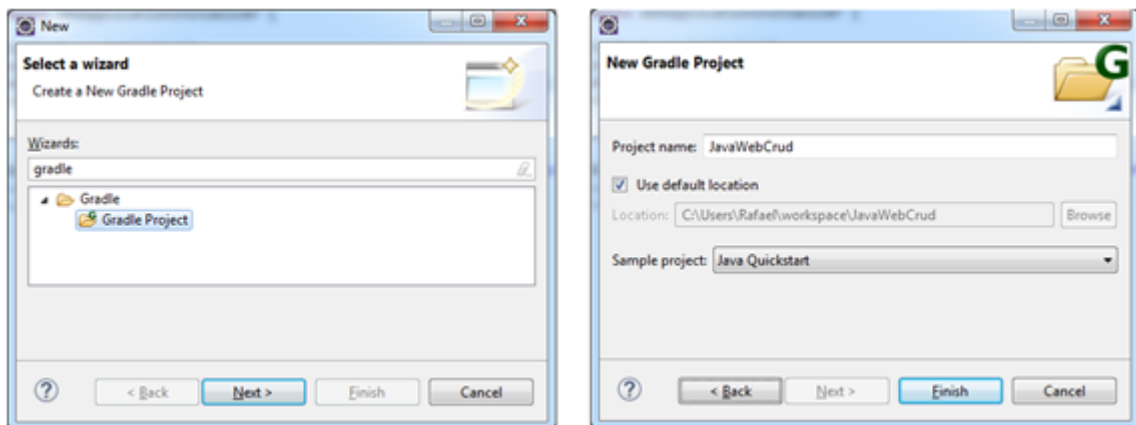


Figura 10: Criando um projeto do Gradle

Será criado um projeto com a estrutura mostrada na figura 11.

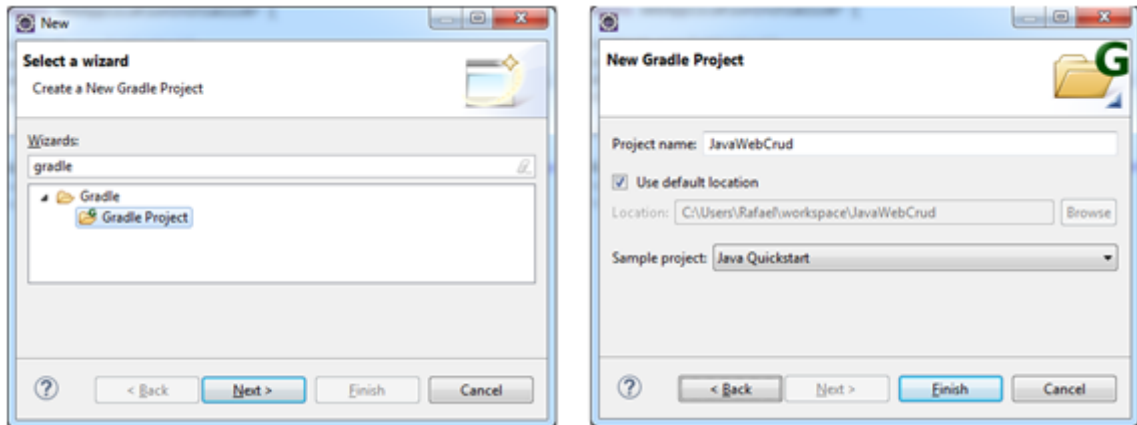


Figura 11: Estrutura inicial de um projeto do Gradle

As pastas “src/main/java” e “src/main/resources” devem armazenar, respectivamente, código fonte Java e recursos utilizados na aplicação. As pastas “src/test/java” e “src/test/resources” são utilizadas para testes unitários. As pastas de testes não serão utilizadas nos nossos exemplos e serão removidas. A pasta build é para onde irão todos os artefatos gerados pelo projeto.

O arquivo build.gradle é o arquivo de definição de projeto do Gradle. Nele podem ser criados scripts para controlar a construção de artefatos, adquirir bibliotecas de terceiros, configurar testes unitários e realizar várias outras tarefas. Os scripts do Gradle são escritos em Groovy, outra linguagem compatível com a máquina virtual Java.

```

apply plugin: 'java'
apply plugin: 'eclipse'

sourceCompatibility = 1.5
version = '1.0'
jar {
    manifest {
        attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': version
    }
}

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'commons-collections', name: 'commons-collections', version: '3.2'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}

test {
    systemProperties 'property': 'value'
}

```

```

uploadArchives {
    repositories {
        flatDir {
            dirs 'repos'
        }
    }
}
}

```

Quadro 1: O arquivo build.gradle

A estrutura do projeto e o arquivo gradle.build gerados devem ser modificados para servir à uma aplicação web. Será adicionada uma nova pasta, chamada de WebContent, para armazenar conteúdo específico para web (páginas jsp/html, arquivos javascript e css). Dentro dela deve ser criada uma pasta chamada WEB-INF para armazenar páginas jsp. Por padrão, usuários de sistemas web Java Enterprise Edition não possuem acesso direto à pasta WEB-INF, então é um lugar seguro para armazenar páginas. Na figura 12 pode ser observada a estrutura do projeto com a pasta WEB-INF.

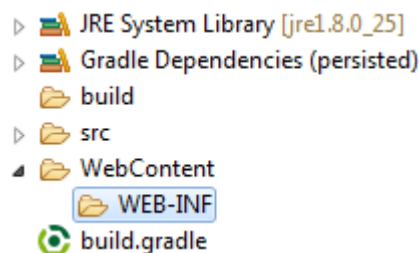


Figura 12: Estrutura do projeto com a pasta WebContent

Em seguida, é necessário modificar o arquivo gradle.build. O Gradle dispõe de vários plugins que facilitam seu uso em diversas situações e servem a propósitos específicos. Para gerar arquivos .war (artefatos de aplicações web) e informar ao Gradle que a pasta WebContent armazenará o conteúdo específico para web, será utilizado o plugin chamado war. Para fazer com que o Eclipse veja o projeto como um projeto para web que possa ser enviado para o Tomcat, será utilizado o plugin eclipse-wtp. O início do arquivo gradle.build deverá ficar como no exemplo mostrado no quadro 2.

```

apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'eclipse-wtp'

project.webAppDirName = 'WebContent'
...

```

Quadro 2: Arquivo gradle.build com novos plugins

Com tudo devidamente configurado, é necessário atualizar o tipo de projeto para que o Eclipse o veja como um projeto de uma aplicação web. Pressionando Ctrl+Alt+Shift+R o Eclipse abrirá uma caixa de texto onde poderão ser executadas tarefas (tasks) do Gradle. Uma dessas tarefas é o comando “eclipse” que gera arquivos adicionais para que o projeto possa ser detectado como uma aplicação web. O comando é executado como na figura 13 e o Eclipse agora poderá publicar a aplicação no Tomcat. gradle-command.png

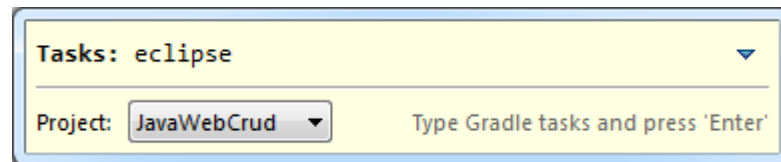


Figura 13: Gradle Task Quick Launcher

Após executar esse comando, a estrutura do projeto mudará um pouco. As pastas destinadas a conter código Java e bibliotecas usadas no projeto serão movidas para uma pasta chamada Java Resources.

3.1.1 Adicionando o projeto ao Tomcat

Para adicionar uma aplicação web ao servidor Tomcat configurado no eclipse, na aba servers dá-se um duplo clique no servidor para abrir suas configurações, e na aba “modules” clica-se no botão “Add Web Module”. Escolhido o projeto web que se deseja adicionar, o caminho da aplicação pode ser configurado. A figura 14 ilustra esse procedimento. Com a configuração padrão do Tomcat, a aplicação estará disponível no endereço com o seguinte formato: `http://<endereço-ip>:8080/<caminhodaaplicação>`. Esse endereço será referenciado no restante do trabalho como raiz da aplicação.

Iniciando o Tomcat, clicando nos botões “Start the server” ou “Start the server in debug mode”, e acessando o endereço da aplicação, será mostrado uma página de erro 404. Isso acontece porque ainda não existem as configurações do servlet padrão, do Spring MVC e ainda não foi criado nenhum controller e nenhuma view. Nesse capítulo ainda será demonstrado como configurar o servlet padrão e o Spring MVC, no capítulo 3 será demonstrado como se criam controllers e views. tomcatproject.png

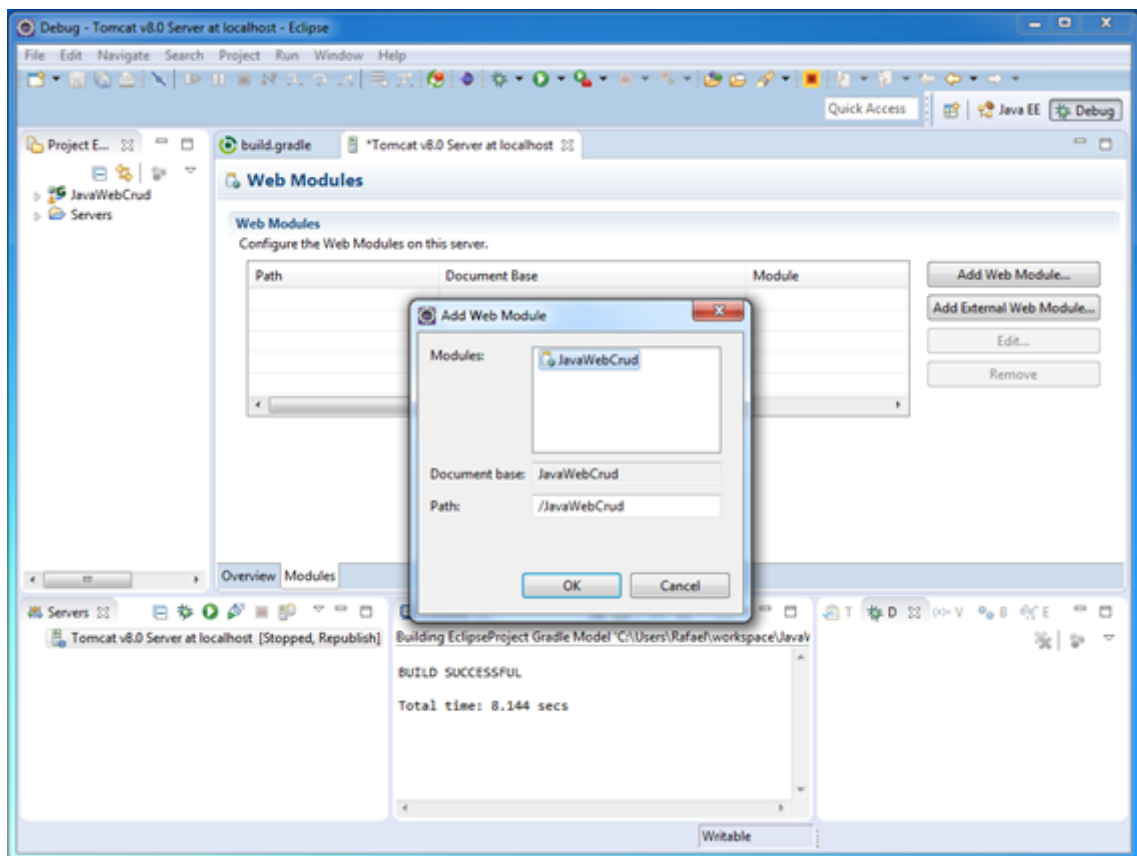


Figura 14: Adicionando um projeto web ao Tomcat

3.1.2 Adicionando dependências ao projeto

Abaixo estão listadas as bibliotecas que serão usadas ao projeto *Java / Spring MVC*. O Gradle pode adquirir essas bibliotecas e suas dependências do repositório central do Maven.

- *Java Servlet API 3.1.0* - Biblioteca base para programar em Java para web usando servlets.
- *Spring Web MVC 4.1.4 RELEASE* - Biblioteca para criar um projeto MVC com Spring.
- *Spring Transaction 4.1.4 RELEASE* - Gerenciador de transações com o banco de dados.
- *Spring Object/Relational Mapping 4.1.4 RELEASE* - Gerenciador de entidades e mapeamento objeto/relacional.
- *Jackson Databind 2.5.1* - Serializa e de serializa objetos Java no formato JSON.
- *MySQL Java Connector 5.1.34* - Comunicação com o banco de dados MySQL.
- *Hibernate JPA Support 4.3.8 Final* - Adaptador do Hibernate para padrões Java Persistence API.

- *Hibernate Validator Engine 5.1.3 Final* - Biblioteca para validar dados de entidades.
- *Hibernate c3p0 Integration 4.3.8* - Gerenciador de conexões com o banco de dados.

Os detalhes sobre as funcionalidades das bibliotecas de comunicação com o banco de dados e mapeamento objeto/relacional serão abordados no capítulo 6.

Para fazer com que o Gradle adquira as bibliotecas necessárias e suas dependências, a seção “dependencies” do arquivo build.gradle é alterada como mostra o quadro 3.

```
dependencies {
    compile 'javax.servlet:javax.servlet-api:3.1.0'
    compile 'org.springframework:spring-webmvc:4.1.4.RELEASE'
    compile 'org.springframework:spring-tx:4.1.4.RELEASE'
    compile 'org.springframework:spring-orm:4.1.4.RELEASE'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.5.1'
    compile 'mysql:mysql-connector-java:5.1.34'
    compile 'org.hibernate:hibernate-entitymanager:4.3.8.Final'
    compile 'org.hibernate:hibernate-validator:5.1.3.Final'
    compile 'org.hibernate:hibernate-c3p0:4.3.8.Final'
}
```

Quadro 3: Adicionando dependências ao projeto

Com o arquivo gradle.build alterado e salvo, o comando para baixar as dependências para o projeto estará localizado no menu de contexto do projeto (clcando com o botão direito do mouse sobre ele) no caminho Gradle/Refresh Dependencies, que pode ser observado na figura 15. O Gradle irá adquirir todas as bibliotecas especificadas no arquivo build.gradle e suas dependências automaticamente. O Spring MVC, por exemplo, depende do Spring Core para funcionar e o Hibernate JPA Support precisa do Hibernate Core. Todas as dependências do projeto ficam visíveis na seção Libraries/Gradle Dependencies.

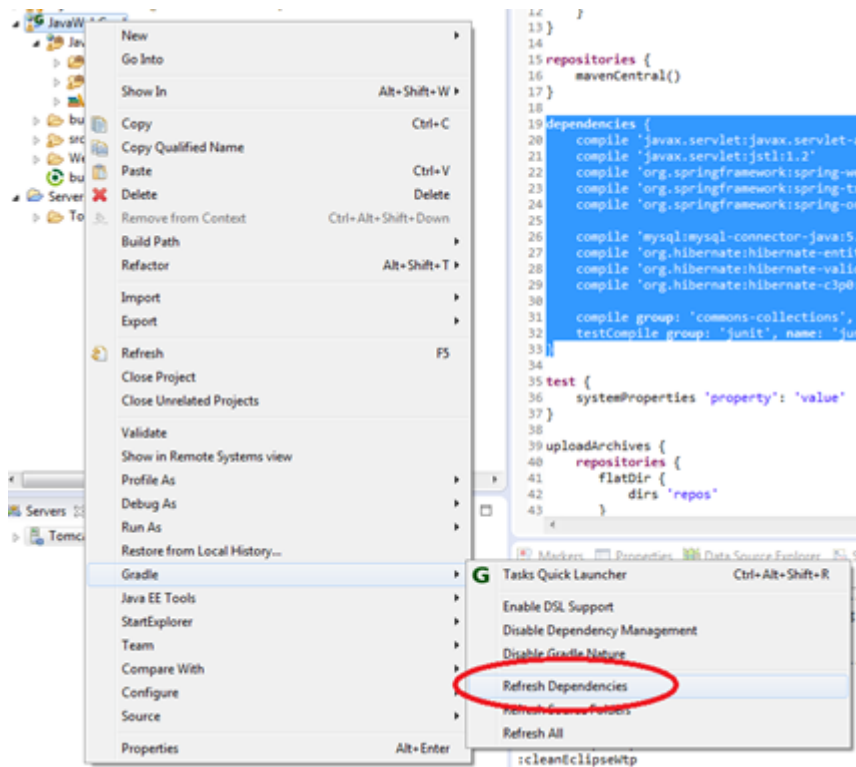


Figura 15: Atualizando dependências de um projeto Gradle

3.1.3 Configurando a aplicação web

Existem duas maneiras de se configurar uma aplicação Java, usando arquivos XML ou escrevendo a configuração em Java. Nos exemplos dessa seção, será usada a segunda opção.

Os arquivos de configura são armazenados em um pacote chamado `br.uece.webCrud.config`. Dentro desse pacote são criadas duas classes, `AppInitializer` e `SpringMvcConfig`, como mostra a figura 15.

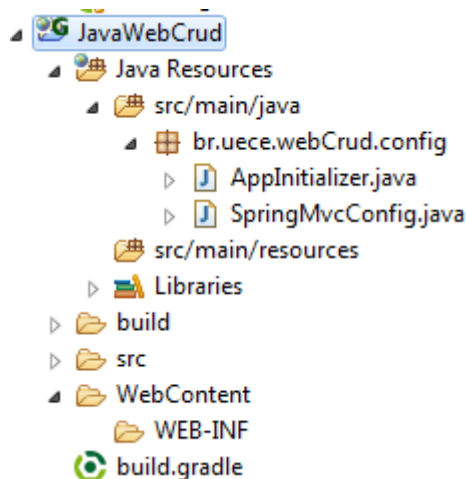


Figura 16: Estrutura do projeto *Java / Spring MVC* com pacote para arquivos de configuração

A primeira configuração que deve ser feita é usar a classe `DispatcherServlet` como o servlet padrão e configurar o contexto da aplicação (configurações específicas do Spring Framework). Servlets são classes Java que processam requisições para aplicação, elas podem servir páginas para o usuário, retornar objetos JSON, redirecionar requisições para outros servlets, ETC. Para mais informações sobre servlets recomenda-se o livro Murach's Java Servlets and JSP.

A classe `AppInitializer` herda da interface `WebApplicationInitializer`. Para inicializar a aplicação, o Spring Framework irá procura classes que herdaram dessa interface dentro dos pacotes. O conteúdo da classe deve ficar como no quadro 5.

A interface `WebApplicationInitializer` possui a assinatura de apenas um método, `onStartup`. Na implementação desse método é escrito código para configurar tanto o contexto da aplicação quando o servlet primário.

Primeiro é criado um objeto do tipo `AnnotationConfigWebApplicationContext` e é usado o seu método `register` para a adicionar a classe `SpringMvcConfig` como uma classe de configuração do Spring. A classe `AnnotationConfigWebApplicationContext` habilita o uso de anotações Java para configurar demais classes, o conteúdo da classe `SpringMvcConfig` será exposto mais adiante.

É criado então um servlet do tipo `DispatcherServlet` que recebe o contexto como parâmetro e é usada a classe `ServletRegistration` para registrar o servlet à aplicação. O método `addMapping` recebe como parâmetro o caractere `/`, isso quer dizer que esse servlet será usado para todas a páginas web e demais caminhos dentro da aplicação. O método `setLoadOnStartup` recebe como parâmetro o valor 1, para configurar nosso servlet como o primeiro que o servidor deve usar.

```
package br.uece.webCrud.web.config;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
```

```

import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class AppInitializer implements WebApplicationInitializer {

    public void onStartUp(ServletContext servletContext)
        throws ServletException {
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();
        context.register(SpringMvcConfig.class);

        DispatcherServlet springServlet = new DispatcherServlet(context);

        ServletRegistration.Dynamic springServletRegistration = servletContext.addServlet(
            springServletRegistration.addMapping("/");
            springServletRegistration.setLoadOnStartup(1);
        }
}

```

Quadro 4: Classe AppInitializer