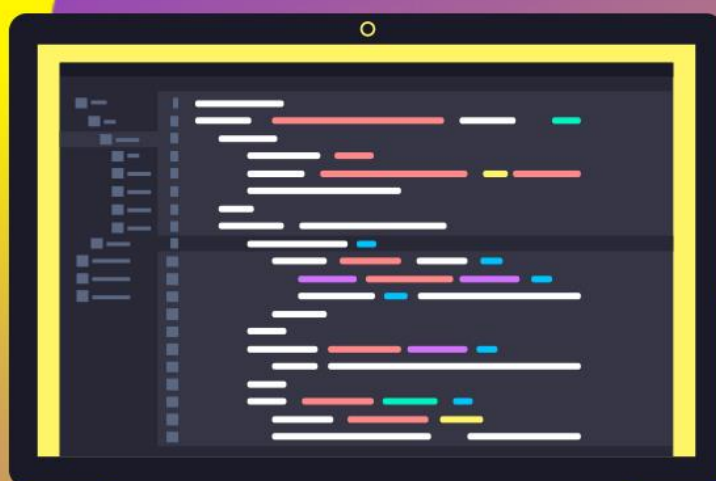


SERVICES ARCHITETURE, API  
E MOBILE ARCHITETURE

# DDD, SPRING BOOT, EXPRESS E NODEJS

CHRISTIANE DE PAULA REIS



## LISTA DE FIGURAS

Figura 2.1 – Strategic design e Tactical patterns in DDD. ....	6
Figura 2.2 – Contexto Limitado em DDD .....	7
Figura 2.3 – Verificando variável de ambiente JAVA_HOME. ....	10
Figura 2.4 – Verificando variável de ambiente Path .....	11
Figura 2.5 – Spring Initializr .....	12
Figura 2.6 – Spring Tool Suite .....	13
Figura 2.7 – Configurando um Spring Starter Project. ....	14
Figura 2.8 – Configurando dependências de um Spring Starter Project.....	15
Figura 2.9 – Criando um Maven Project .....	16
Figura 2.10 – Download Node .....	19
Figura 2.11 – Resultado da criação do arquivo package.json .....	22
Figura 2.12 – Resultado da instalação da biblioteca Express .....	22
Figura 2.13 – Resultado inclusão de dependências no arquivo package.json .....	22
Figura 2.14 – Resultado da execução da aplicação “myapp” em Node. ....	23
Figura 2.15 – Log de criação da aplicação esqueleto chamada “helloworld” .....	25
Figura 2.16 – Pasta da aplicação esqueleto “helloworld” .....	26
Figura 2.17 – Log de instalação das dependências para a aplicação “helloworld” ....	26
Figura 2.18 – Resultado da execução do comando DEBUG .....	27
Figura 2.19 – Resultado da execução da aplicação “helloworld” em Node .....	28
Figura 2.20 – Criando um projeto Maven Web no Eclipse .....	29
Figura 2.21 – Escolhendo o Archetype maven-archetype-webapp. ....	30
Figura 2.22 – Fornecendo os detalhes para o projeto .....	31
Figura 2.23 – Criando uma nova configuração para executar com Maven .....	34
Figura 2.24 – Criando a classe InicioServlet .....	35
Figura 2.25 – Resultado da execução da classe InicioServlet .....	36

## LISTA DE CÓDIGOS-FONTE

Código-Fonte 2.1 – Inserindo inicializador Spring Boot no arquivo pom.xml.....	16
Código-Fonte 2.2 – Instalando Node para Linux (Ubuntu) - comando 2. ....	20
Código-Fonte 2.3 – Instalando Node para Linux (Ubuntu) - comando 2. ....	20
Código-Fonte 2.4 – Comando para verificar qual a versão Node está instalada.....	20
Código-Fonte 2.5 – Comando para verificar qual a versão NPM está instalada. ....	21
Código-Fonte 2.6 – Comando para criar e navegar no diretório da aplicação “myapp”. ....	21
Código-Fonte 2.7 – Comando para criar um arquivo package.json. ....	21
Código-Fonte 2.8 – Comando para instalar a biblioteca Express na aplicação “myapp”. ....	22
Código-Fonte 2.9 – Listagem do arquivo index.js da aplicação “myapp”. ....	23
Código-Fonte 2.10 – Comando para iniciar o servidor chamando o Node. ....	23
Código-Fonte 2.11 – Comando para instalar o Express Application Generator através do NPM. ....	24
Código-Fonte 2.12 – Comando para criar uma aplicação chamada “helloworld”. ....	24
Código-Fonte 2.13 – Comando para entrar no diretório da aplicação “helloworld”. ..	25
Código-Fonte 2.14 – Comando para instalar dependências em “helloworld”. ....	26
Código-Fonte 2.15 – Executar a aplicação esqueleto “helloworld” no Windows. ....	26
Código-Fonte 2.16 – Executar a aplicação esqueleto “helloworld” no Linux/MacOS. .....	27
Código-Fonte 2.17 – Listagem do arquivo app.js da aplicação “helloworld”.....	27
Código-Fonte 2.18 – Comando para executar a aplicação “helloworld” com Node...28	
Código-Fonte 2.19 – Inserindo dependências no arquivo pom.xml – parte 2.....	32
Código-Fonte 2.20 – Inserindo dependências no arquivo pom.xml – parte 2.....	32
Código-Fonte 2.21 – Inserindo propriedades no arquivo pom.xml. ....	32
Código-Fonte 2.22 – Listagem final do arquivo pom.xml.....	33
Código-Fonte 2.23 – Criando o arquivo <i>application.properties</i> .....	33
Código-Fonte 2.24 – Listagem da classe InicioServlet.....	36

## SUMÁRIO

2 DDD, SPRING BOOT, EXPRESS E NODEJS .....	5
2.1 DDD, o que é?.....	5
2.2 Spring Boot.....	7
2.2.1 Componentes do Spring Boot .....	8
2.2.2 Pré-requisitos para trabalhar com Spring Boot.....	9
2.2.3 Spring Initializr .....	11
2.2.4 Spring Tool Suite (STS).....	13
2.2.5 Spring Boot Starter .....	15
2.3 Express e NodeJs .....	17
2.3.1 Express .....	17
2.3.2 Node.....	17
2.3.3 Afinal, o que é o NPM?.....	18
2.3.4 Configurar o Node/Express como ambiente de desenvolvimento .....	18
2.3.5 Instalando Node para Windows e MacOS .....	20
2.3.6 Instalando Node para Linux (Ubuntu).....	20
2.3.7 Verificar as versões instaladas do Node e NPM .....	20
2.3.8 Criando sua primeira aplicação “Hello World” em Node com Express .....	21
2.3.9 Instalando o Express Application Generator .....	24
2.3.10 Criando sua primeira aplicação esqueleto padrão MVC através do Express Generator .....	24
2.4 Um pouco sobre Maven .....	28
2.4.1 Criando e executando um projeto Maven Web no Eclipse .....	29
CONCLUSÃO.....	37
REFERÊNCIAS.....	38
GLOSSÁRIO .....	39

## 2 DDD, SPRING BOOT, EXPRESS E NODEJS

Neste capítulo você vai entender o que é Design Dirigido por Domínio (DDD, traduzido do inglês *Domain-Driven Design*) e como ele se encaixa na arquitetura de microsserviços (MSA - *MicroService Architecture*). Vai aprender também sobre o projeto Spring Boot do framework Spring e vai iniciar a implementação do seu primeiro microsserviço usando o Maven e o framework Express com Node.js. Siga em frente!

### 2.1 DDD, o que é?

DDD é uma modelagem para o desenvolvimento de software através de um conjunto de princípios e padrões de projeto e surgiu para ser uma espécie de guia para aplicar as melhores práticas de desenvolvimento que já existem. No DDD, um modelo deve ser único e internamente consistente, funcionando como linguagem ubíqua que representa o elo de comunicação entre o desenvolvedor de software e a pessoa que desempenha o papel de especialista em domínio.

**IMPORTANTE:** Domínio é o *core* do projeto. É o problema que será resolvido, ou seja, as regras de negócio do projeto.

Pela definição de Vernon, “DDD é primariamente sobre modelar uma Linguagem Ubíqua em um Contexto Delimitado”.

*“In short, DDD is primarily about modeling a Ubiquitous Language in an explicitly Bounded Context.”* (VERNON, 2016. p. 11)

O modelo DDD propõe uma conexão entre padrões estratégicos (como por exemplo, o protótipo do projeto) e ferramentas táticas fundamentais de programação (por exemplo, a implementação e o detalhamento do projeto).

## DOMAIN DRIVEN DESIGN

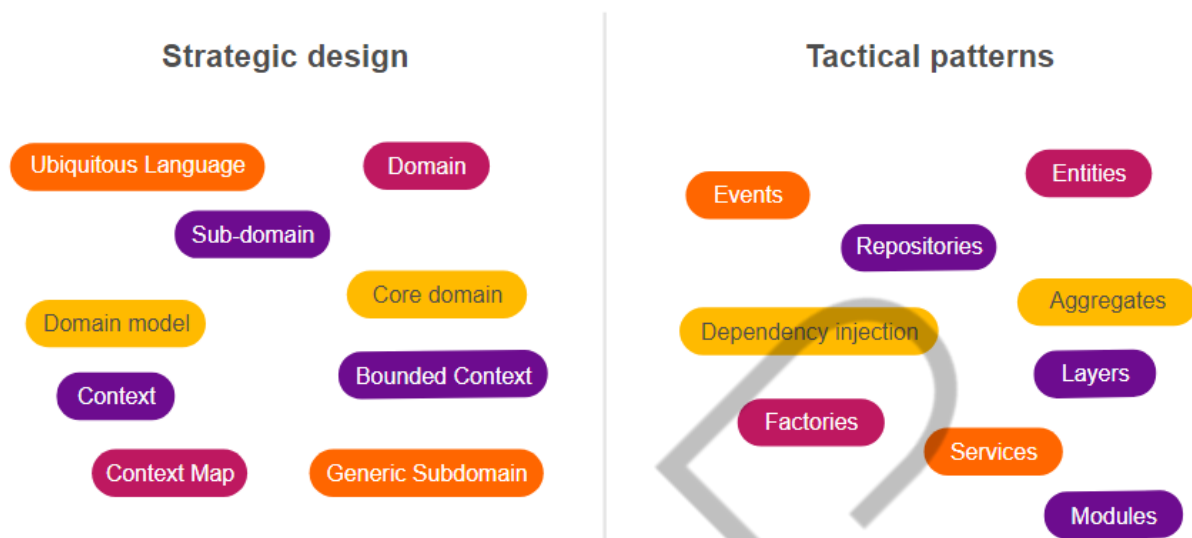


Figura 2.1 – Strategic design e Tactical patterns in DDD.  
Fonte: VERNON (2016)

Design estratégico é como fazer o rascunho antes de entrar nos detalhes da implementação. Destaca o que é estrategicamente importante para o seu negócio, como dividir o trabalho por importância e como fazer integrações da melhor maneira. (VERNON, 2016).

O padrão do design estratégico no DDD é implementar um software pensando em grandes modelos divididos em diferentes contextos limitados (*bounded context*), porém interligados em determinados pontos.

Bounded Context is a central pattern in Domain-Driven Design. It is the focus of DDD's strategic design section which is all about dealing with large models and teams. DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their interrelationships. (FOWLER, 2014).

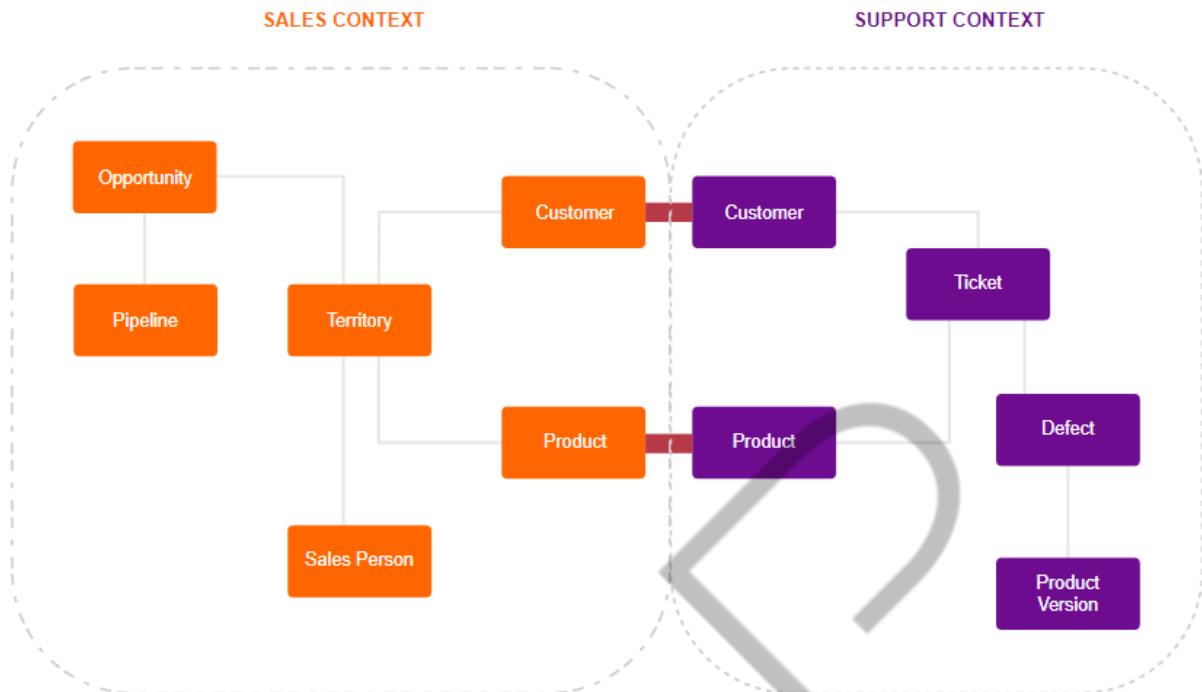


Figura 2.2 – Contexto Limitado em DDD  
Fonte: Fowler (2014)

É através dos princípios do DDD que empresas lidam, por exemplo, com uma migração de um monólito para microsserviços.

A disseminação dos microsserviços foi beneficiada por fatores como os conceitos de entrega contínua, automação da infraestrutura, escalabilidade de sistemas, desenvolvimento ágil, concepção orientada ao domínio, do inglês *Domain-driven design* – DDD. (EVANS, 2004).

## 2.2 Spring Boot

*Spring Boot* é um projeto do *framework Spring* responsável pelo processo de configuração e publicação de aplicações, que torna possível implantar e executar rapidamente uma aplicação em produção com esforço mínimo do desenvolvedor. Tem como princípios:

- Experiência de início de projeto (*getting started experience*) extremamente rápida e direta;
- Dispõe de diversos requisitos não funcionais pré-configurados como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado etc.; o que facilita bastante a vida do desenvolvedor;

- Não geração de código e reduzir a zero a necessidade de arquivos XML (*Extensible Markup Language*);
- Visão opinativa (*opinionated*) sobre a melhor forma de configuração dos projetos Spring, no entanto flexível para permitir substituição conforme requisitos do projeto.

### 2.2.1 Componentes do Spring Boot

O *Spring Boot* conta com três componentes para prover tais facilidades de configuração e publicação de aplicações, que são:

- **Spring Boot Starter:** são inicializadores (*starters*) que encapsulam todas as dependências relativas à criação de um determinado projeto Spring Boot. Alguns dos inicializadores mais populares do Spring Boot, são:
- **spring-boot-starter-web** é utilizado para desenvolver serviços web RESTful usando Spring MVC e já vem com o Tomcat como container da aplicação integrado, ou seja, fornece um ambiente de execução pronto com as funcionalidades que permitem construir a API (*Application Programming Interface*) do microsserviço em cima de controladores RESTful HTTP.
- **spring-boot-starter-jersey** é uma alternativa ao spring-boot-starter-web que usa o Apache Jersey em vez do Spring MVC.
- **spring-boot-starter-jdbc** é utilizado para definição do conjunto de conexões do JDBC. Baseia-se na implementação do conjunto de conexões do JDBC do Tomcat.
- **Spring Boot Autoconfiguration:** fornece configurações-padrão e ainda permite personalizar uma configuração; é responsável por gerenciar o processo de configuração de uma aplicação Spring Boot. Continuando com o exemplo de criação de um projeto web, as diretivas para resolução de *views* e *resolvers* são fornecidas pelo Spring Boot Autoconfiguration.
- **Spring Boot Actuator:** é o componente responsável pelo provisionamento de *endpoints* e obtenção de métricas da aplicação, ou seja, é responsável por operacionalizar os microsserviços fornecendo a estrutura e os *endpoints*



RESTful HTTP, expondo métricas, parâmetros de configuração e mapeamento de componentes internos, que são úteis para depuração. Por exemplo, efetua o provisionamento no servidor web de uma configuração efetuada pelo Spring Boot Autoconfiguration, ainda no caso de trabalhar com a criação de uma aplicação web.

### 2.2.2 Pré-requisitos para trabalhar com Spring Boot

Como pré-requisitos para trabalhar com Spring Boot, você deve ter instalado:

- JDK 11 para Windows, Mac OS X ou Linux/Unix.
- Eclipse IDE para Windows, Mac OS X ou Linux/Unix.
- Apache Maven ou Gradle para Windows, Mac OS X ou Linux/Unix.
- Git para Windows, Mac OS X ou Linux/Unix.

Nos nossos exemplos, vamos codificar em linguagem JAVA e trabalhar com JDK 12.0.4, Eclipse IDE 2019-06 R, Apache Maven 3.6.1 e Git 2.23.0, todos para sistema operacional Windows.

Se você já possui todas essas ferramentas instaladas em sua máquina, vamos, antes, nos certificar de que as variáveis de ambiente estão configuradas corretamente. Para fazer isso no Windows, clique com o botão direito em Este Computador > Propriedades > Configurações Avançadas do Sistema > Variáveis de Ambiente.

Verifique se você já possuiu uma variável chamada JAVA\_HOME, com o local onde seu JDK foi instalado, como na imagem abaixo:

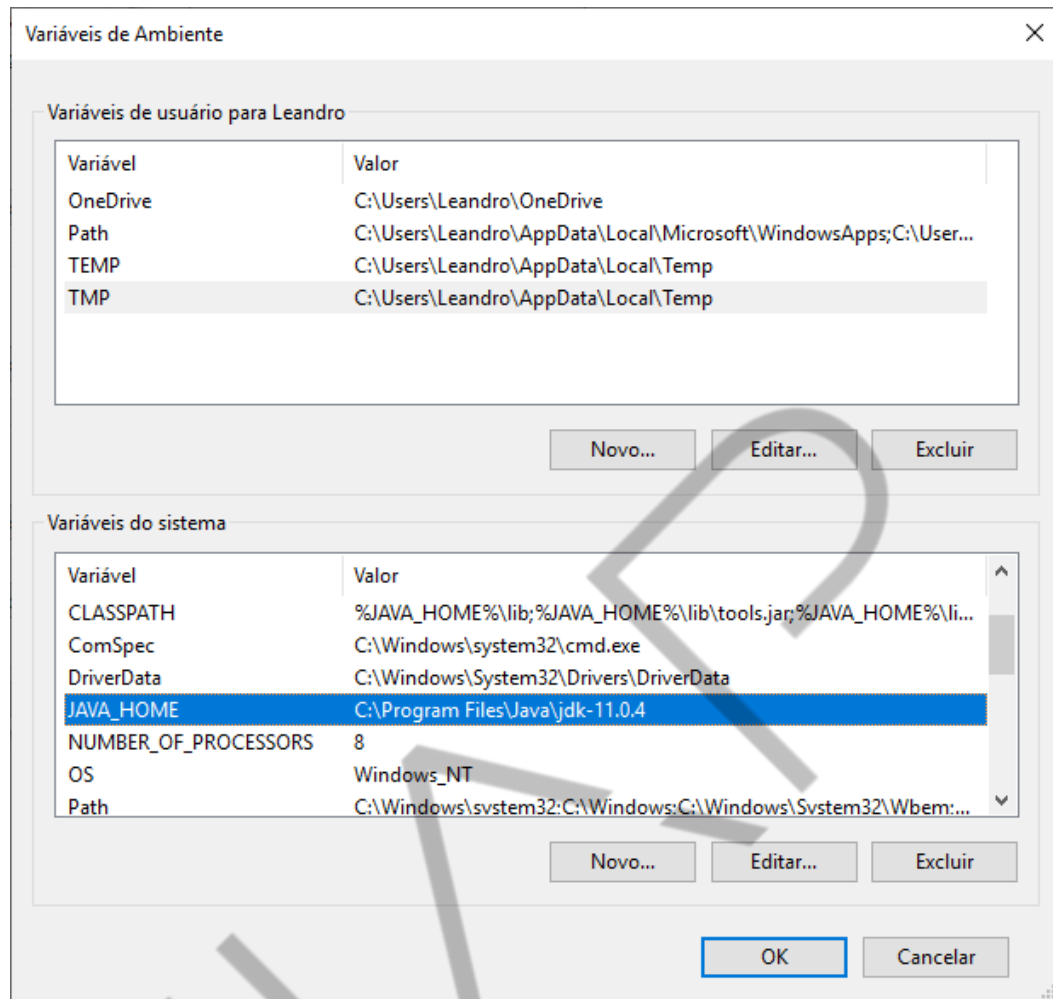


Figura 2.3 – Verificando variável de ambiente JAVA\_HOME.  
Fonte: Elaborado pela autora (2019)

Caso não houver, adicione esta variável com o caminho do seu JDK.

Verifique também se o Path do seu sistema já contém o caminho dessas ferramentas que foram instaladas, como:

- O JDK apontando para a pasta bin

- O Maven apontando para a pasta bin

- O Git apontando para a pasta cmd

Para fazer isso no Windows, clicar duas vezes na variável Path. Caso não houver, adicione todos os caminhos, como na imagem abaixo:

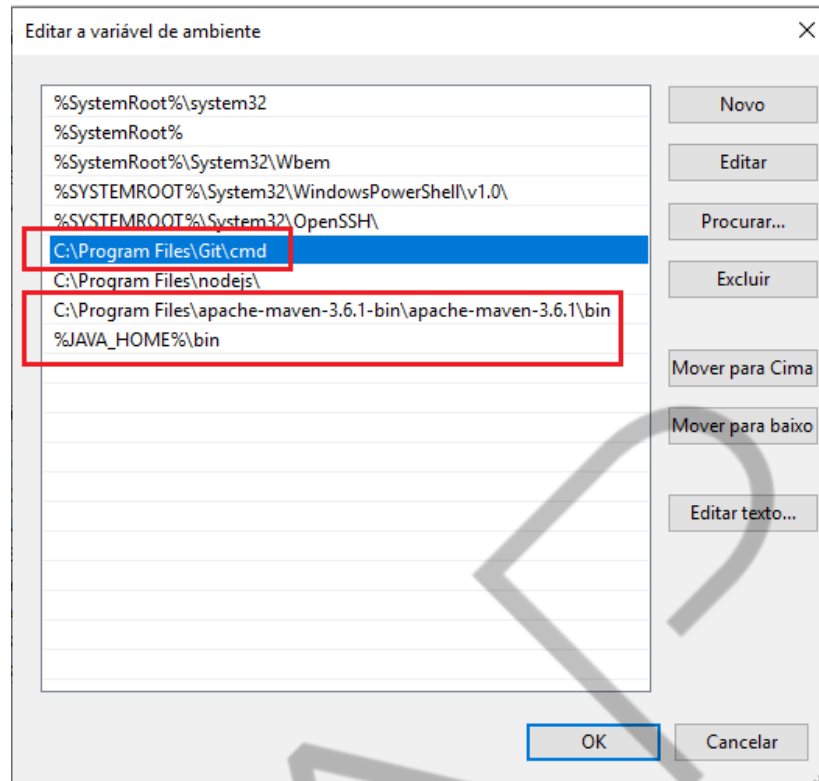


Figura 2.4 – Verificando variável de ambiente Path  
Fonte: Elaborado pela autora (2019)

Veja a seguir as opções para criar projetos de inicialização Spring com Eclipse e Maven.

### 2.2.3 Spring Initializr

1. Você deve acessar o Web Site Spring Initializr (<https://start.spring.io/>), disponibilizado pelo framework do Spring, que permite selecionar com alguns cliques os metadados básicos (bibliotecas) que você deseja utilizar no seu projeto e gera, ao final, um projeto Spring-boot Maven ou Gradle pré-configurado em linguagem Java, Kotlin ou Groovy, já contendo os componentes especificados.

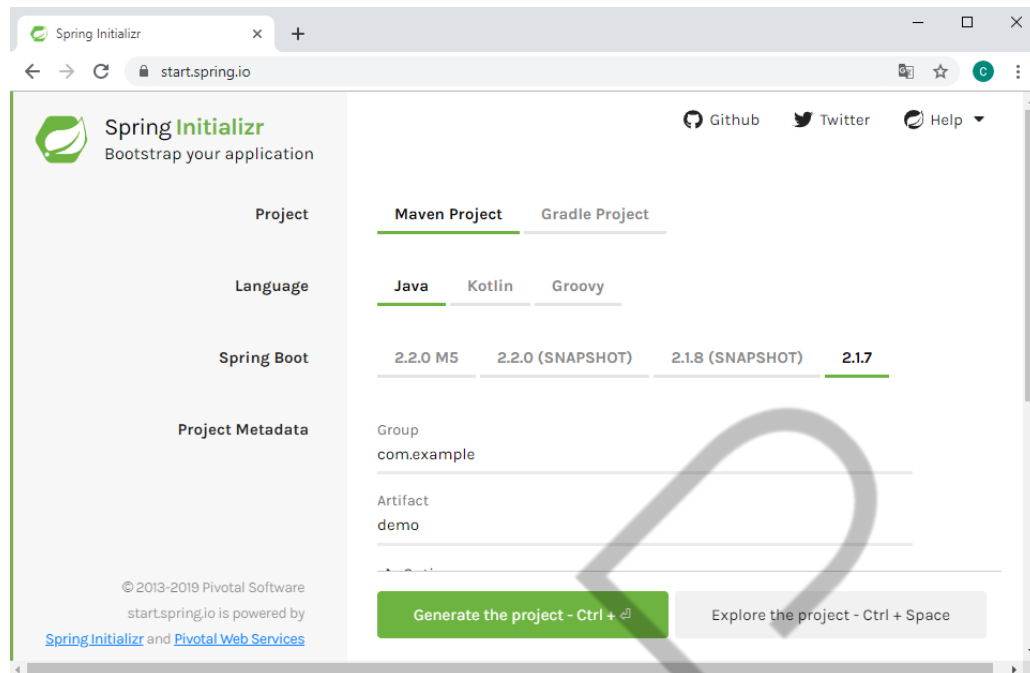


Figura 2.5 – Spring Initializr  
Fonte: Start.spring.io (2019)

2. Descompactar o arquivo ZIP localmente.
3. Após clicar em Generate Project, um arquivo ZIP será baixado para o seu computador, agora você precisa descompactar o arquivo em uma pasta local.
4. No Eclipse, clique em Arquivo > Importar > Maven > Projeto Maven existente
5. Navegue ou digite o caminho da pasta para onde você extraiu o arquivo zip na próxima tela.
6. 4Baixar as dependências do projeto
7. Depois de clicar em Concluir, o Maven levará algum tempo para baixar todas as dependências e inicializar o projeto.
8. Você pode conferir no diretório src/main/java o código fonte da classe *init* que foi gerada para o projeto.

## 2.2.4 Spring Tool Suite (STS)

O **STS** possui um assistente de criação de projetos para facilitar o trabalho de configuração. Você pode fazer o download da [instalação completa do STS](#) versão 3.9.9.RELEASE ou instalar o plug-in do STS Eclipse para criar um projeto Spring Boot Maven diretamente do Eclipse.

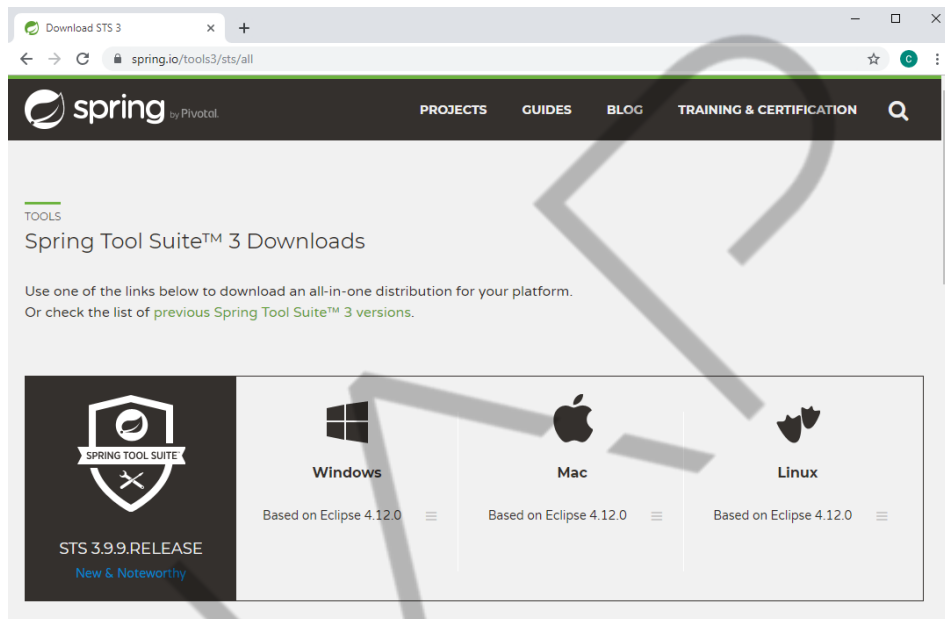
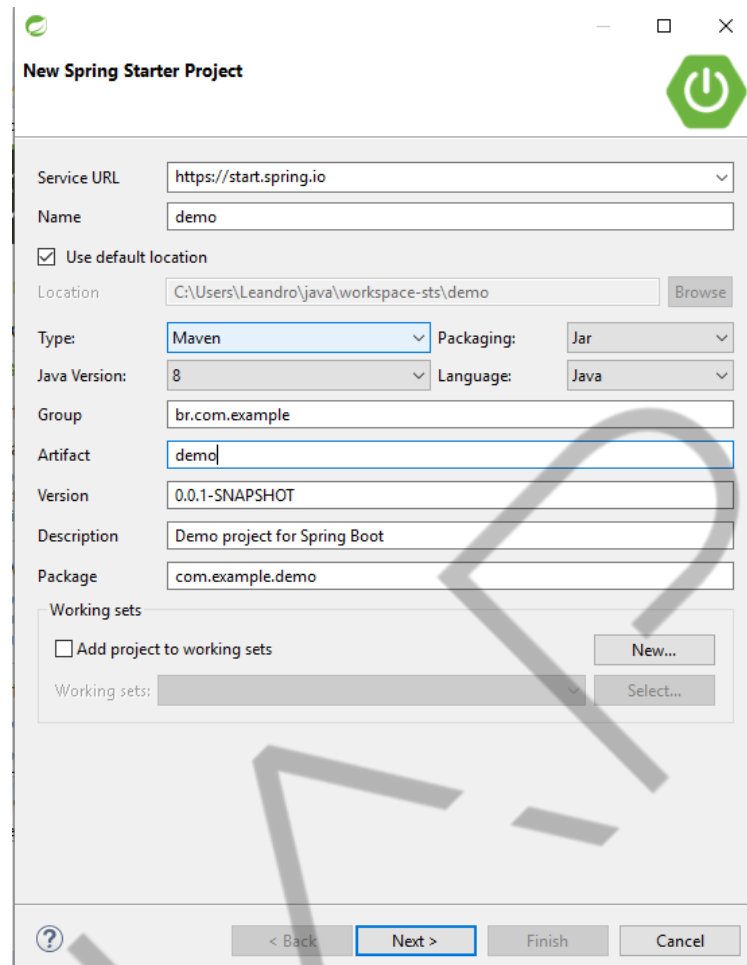


Figura 2.6 – Spring Tool Suite  
Fonte: Start.spring.io (2019)

1. No STS, clique em File > New > Spring Starter Project

Forneça os detalhes para o seu projeto: Name, Group, Version



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

Figura 2.7 – Configurando um Spring Starter Project.  
Fonte: Elaborado pela autora (2019)

Clique em Next

2. Selecione as dependências (inicializadores) desejadas para seu projeto.

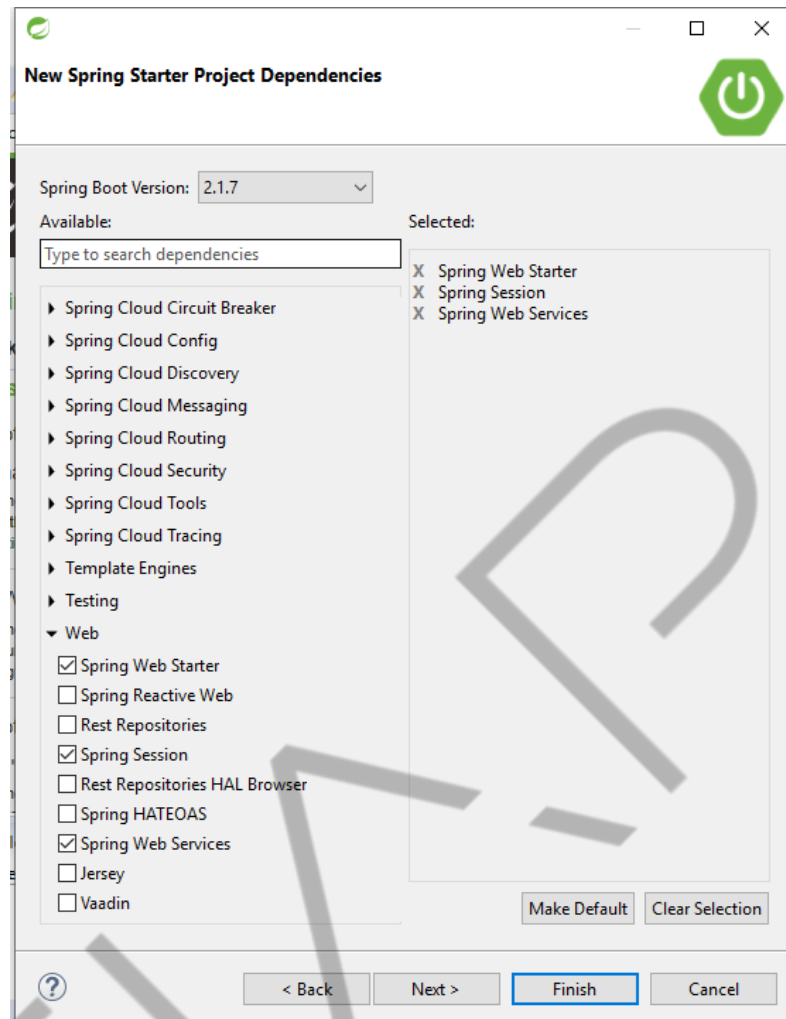


Figura 2.8 – Configurando dependências de um Spring Starter Project.  
Fonte: Elaborado pela autora (2019)

Clique em Finish.

O Maven (indicado em Type) deve levar algum tempo para baixar as dependências e inicializar o projeto.

### 2.2.5 Spring Boot Starter

Criar manualmente um projeto Maven através do Eclipse e adicionar as dependências do Spring Boot Starter no arquivo pom.xml.

1. No Eclipse, clique em Arquivo> Novo> Projeto Maven

Escolha “Create a simple project”

Navegue ou digite o caminho da pasta da sua workspace.

Forneça os detalhes para o seu projeto: ID do grupo, ID do Artefato e Versão  
Clique em Concluir

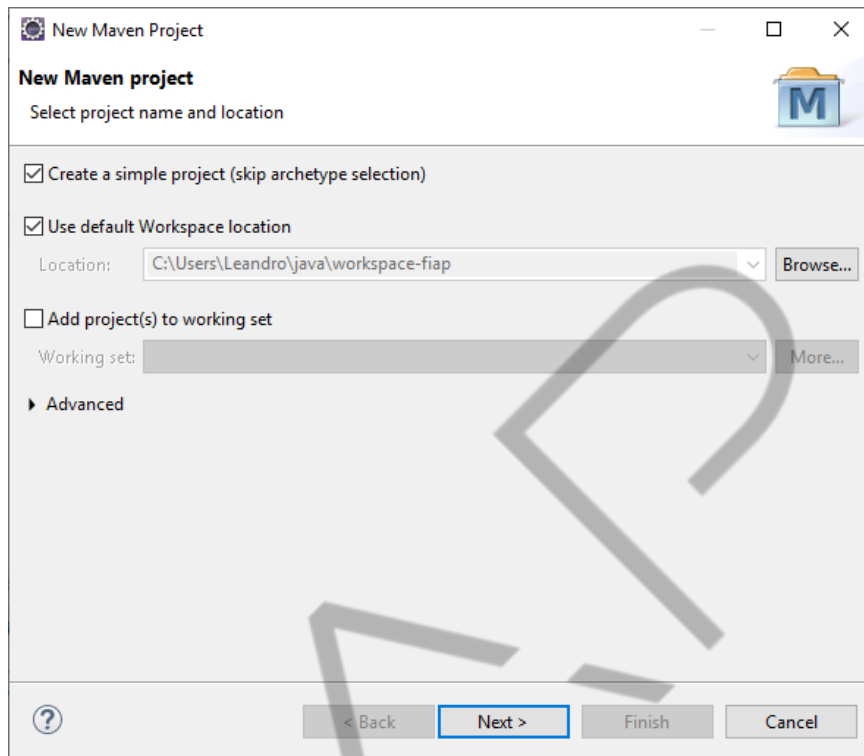


Figura 2.9 – Criando um Maven Project  
Fonte: Elaborado pela autora (2019)

2. Isso criaria um projeto básico do Maven com zero dependências
3. No arquivo pom.xml, adicione os inicializadores apropriados ao Spring Boot

Para trabalhar com o starter web do Spring Boot, inserir a seguinte dependência:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Código-Fonte 2.1 – Inserindo inicializador Spring Boot no arquivo pom.xml  
Fonte: Elaborado pela autora (2019)



## 2.3 Express e NodeJs

### 2.3.1 Express

Express é um framework *javascript* para aplicações web, muito utilizado em programações Node e está disponível no registro NPM (*Node Package Manager*) como um módulo Node.js. Express é não opinativo e minimalista.

- Fornece mecanismos para adicionar um *middleware* em qualquer ponto da requisição, dando maior flexibilidade para os desenvolvedores quanto aos problemas no desenvolvimento web;
- Dispõe de bibliotecas para trabalhar com login de usuários, cookies, sessões, cabeçalho de segurança, parâmetros de URL, dados em requisições POST etc.;
- Permite criar sua própria estrutura de pastas dentro do diretório principal;
- Permite estruturar sua aplicação da maneira que você desejar.

### 2.3.2 Node

O Node (ou Node.js) é um ambiente em tempo de execução *open-source* e multiplataforma que possibilita criar aplicações e ferramentas *server-side* em JavaScript. Como benefícios relevantes de se trabalhar com o Node, destacamos:

**Flexibilidade:** Por ser gerenciado pelo NPM, o Node.js é visto como uma plataforma com potencial para ser utilizada em qualquer situação.

**Portabilidade:** Suportado em sistemas operacionais Microsoft Windows, OSX, Linux, Solaris, FreeBSD, OpenBSD, WebOS e NonStop.

**Performance:** Node foi concebido com a proposta de otimizar a escalabilidade em aplicações web e a taxa de transferência em rede. É voltado para o desenvolvimento de aplicações em rede e foi projetado para lidar com alta concorrência em tempo real.

**Produtividade:** Os desenvolvedores trabalham com código JavaScript simples e não precisam se preocupar com alterações de código entre servidor web e navegador.

Node conta com suporte das principais empresas de produtos e serviços *cloud* do mercado, como a AWS, Microsoft Azure e Google Cloud e se encaixa perfeitamente para a implementação de serviços e componentes de Arquiteturas de Microsserviços (*Microservices Architecture - MSA*) e Serverless, devido às características de leveza e flexibilidade.

### 2.3.3 Afinal, o que é o NPM?

**NPM** (*Node Package Manager*) é a ferramenta mais importante para otimizar seu fluxo de trabalho com o JavaScript, funcionando como gerenciador de pacotes do Node.js.

É usado para buscar pacotes (bibliotecas JavaScript) que uma aplicação precisa nos ambientes de desenvolvimento, teste e produção, com finalidade de reutilização de código e pode ser usado para executar testes e ferramentas usadas no processo de desenvolvimento.

**IMPORTANTE:** NPM é o maior repositório de softwares do mundo.

### 2.3.4 Configurar o Node/Express como ambiente de desenvolvimento

O ambiente de desenvolvimento Express inclui em seu computador local uma instalação do Node.js, o pacote de gerenciamento NPM e o Gerador de Aplicações Express (*Express Application Generator*). Entretanto, não inclui um servidor web de desenvolvimento separado, ou seja, a aplicação cria e executa seu próprio servidor web.

**Gerador de Aplicações Express** sua instalação é opcional, porém facilita muito a criação de aplicações WEB através de esqueletos que seguem o padrão MVC (*Model-View-Controller*) e promove uma estrutura de aplicação modular.

### Pré-Requisitos:

- Ter previamente instalado em seu computador algum editor de texto ou IDE para edição de código.
- Ter previamente instalado em seu computador alguma ferramenta de controle de versão ou para registrar o histórico de edições do seu arquivo, como, por exemplo, o [Git](#).
- Ter habilidade com terminal e linha de comando.

Após cumprido os pré-requisitos, siga a sequência de passos:

1. instalar o Node.js e o NPM

Última Versão Node LTS: [10.16.3 \(inclui NPM 6.9.0\)](#)

**DICA:** Você deve sempre trabalhar com a versão Node LTS - *Long Term Supported* (suporte a longo prazo) mais recente, por ser mais estável do que o release "atual".

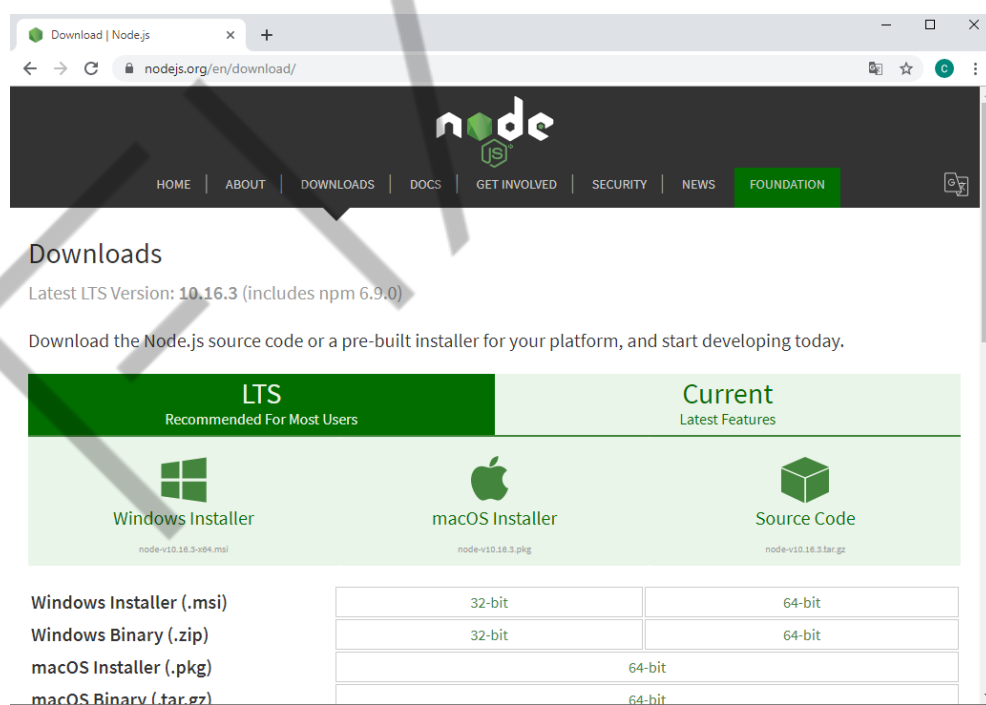


Figura 2.10 – Download Node  
Fonte: Nodejs.org (2019)

2. instalar o Gerador de Aplicações Express através do NPM

**Versão Express ou Gerador de Aplicações Express:** você deve sempre usar a versão mais recente, no nosso caso, vamos trabalhar com a versão 4.16.1.

Veja, a seguir, como configurar e testar o ambiente Node/Express no Windows, Linux (Ubuntu) e macOS, e também como instalar o Gerador de Aplicações Express.

### 2.3.5 Instalando Node para Windows e MacOS

9. Baixe o instalador em [Node](#).

10. Instale o Node clicando duas vezes no arquivo baixado e siga os prompts de instalação.

### 2.3.6 Instalando Node para Linux (Ubuntu)

Obter a versão LTS do Node através do repositório de distribuições binárias do Ubuntu, executando no terminal os seguintes comandos:

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E  
bash -
```

Código-Fonte 2.2 – Instalando Node para Linux (Ubuntu) - comando 2.  
Fonte: Elaborado pela autora (2019)

```
sudo apt-get install -y nodejs
```

Código-Fonte 2.3 – Instalando Node para Linux (Ubuntu) - comando 2.  
Fonte: Elaborado pela autora (2019)

**DICA:** Não instalar diretamente dos repositórios normais do Ubuntu, pois eles contêm versões desatualizadas do Node.

### 2.3.7 Verificar as versões instaladas do Node e NPM

Executar o comando "*version*" no terminal / *prompt* de comando e verificar se uma *string* de versão é retornada:

```
node -v
```

Código-Fonte 2.4 – Comando para verificar qual a versão Node está instalada.

Fonte: Elaborado pela autora (2019)

```
npm -v
```

Código-Fonte 2.5 – Comando para verificar qual a versão NPM está instalada.

Fonte: Elaborado pela autora (2019)

### 2.3.8 Criando sua primeira aplicação “Hello World” em Node com Express

Vamos fazer download de um pacote através do NPM, salvar nas dependências do projeto e configurar para ser utilizado em uma aplicação Node.

As dependências são gerenciadas usando um arquivo de definição de texto simples chamado package.json. Esse arquivo deve conter tudo o que o NPM precisa para buscar e executar sua aplicação.

#### 2. Crie e entre no novo diretório para sua aplicação chamada “myapp”

```
mkdir myapp  
cd myapp
```

Código-Fonte 2.6 – Comando para criar e navegar no diretório da aplicação “myapp”.

Fonte: Elaborado pela autora (2019)

2. Execute o comando <npm init> para criar um arquivo package.json para sua aplicação.

```
npm init
```

Código-Fonte 2.7 – Comando para criar um arquivo package.json.

Fonte: Elaborado pela autora autor (2019)

Esse comando solicita algumas informações como nome e versão da sua aplicação; e o nome do arquivo de ponto de entrada inicial (por padrão, é index.js). Por enquanto, apenas aceite os padrões teclando <enter>. Ao final, é possível verificar os padrões que foram definidos e a licença do arquivo package.json para confirmar, basta informar “yes”.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Figura 2.11 – Resultado da criação do arquivo package.json

Fonte: Elaborado pela autora (2019).

3. Agora instale a biblioteca Express no diretório da aplicação *myapp*. O pacote será salvo automaticamente na lista de dependências do seu arquivo package.json.

```
npm install express
```

Código-Fonte 2.8 – Comando para instalar a biblioteca Express na aplicação “myapp”.

Fonte: Elaborado pela autora (2019)

```
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN myapp@1.0.0 No description
npm WARN myapp@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 3.841s
found 0 vulnerabilities
```

Figura 2.12 – Resultado da instalação da biblioteca Express

Fonte: Elaborado pela autora (2019)

O Express já foi incluído na sessão de dependências do seu arquivo package.json

```
{
  "name": "myapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {},
  "description": ""
}
```

Figura 2.13 – Resultado inclusão de dependências no arquivo package.json

Fonte: Elaborado pela autora (2019)

4. Crie um arquivo chamado **index.js** na raiz do diretório da aplicação "myapp" com o seguinte conteúdo:

```
01 var express = require('express')
02 var app = express()
03
04 app.get('/', function (req, res) {
05   res.send('Hello World!')
06 })
07
08 app.listen(8000, function () {
09   console.log(Exemplo de app ouvindo na porta 8000!')
10 })
```

Código Fonte 2.9 – Listagem do arquivo index.js da aplicação “myapp”.  
Fonte: Elaborado pela autora (2019)

O código da aplicação da web Express "HelloWorld" importa o módulo "express" na linha 1, utiliza esse módulo para criar um servidor (uma aplicação) que ouve solicitações HTTP na porta 8000 (linha 8), e imprime uma mensagem no console (linha 9) informando qual porta você pode usar para testar o servidor. A função app.get(), na linha 4, só responde a solicitações HTTP GET com o caminho de URL especificado ('/'), neste caso chamando uma função para enviar nossa mensagem “Hello World!”, na linha 5.

5. Inicie o servidor chamando o Node em seu prompt de comando, com o script que você acabou de criar.

```
node index.js
```

Código-Fonte 2.10 – Comando para iniciar o servidor chamando o Node.  
Fonte: Elaborado pela autora (2019)

6. Navegue pela URL <http://127.0.0.1:8000/>. Se tudo estiver funcionando, o navegador deve simplesmente exibir a mensagem "Hello World!".

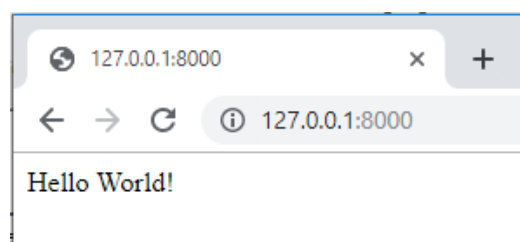


Figura 2.14 – Resultado da execução da aplicação “myapp” em Node.  
Fonte: Elaborado pela autora (2019)

Agora você já tem sua primeira aplicação em Node, criada através do Express, funcionando no *localhost* - porta 8000.

### 2.3.9 Instalando o Express Application Generator

2. Instale o gerador de aplicações através do NPM.

```
npm install express-generator -g
```

Código-Fonte 2.11 – Comando para instalar o Express Application Generator através do NPM.

Fonte: Elaborado pela autora (2019)

A flag -g instala a ferramenta globalmente para que você possa chamá-la de qualquer lugar.

### 2.3.10 Criando sua primeira aplicação esqueleto padrão MVC através do Express Generator

2. Execute o comando para criar sua primeira aplicação “esqueleto” no padrão MVC, de nome “helloworld” (verifique a pasta onde será criada sua nova aplicação).

```
express helloworld
```

Código-Fonte 2.12 – Comando para criar uma aplicação chamada “helloworld”.

Fonte: Elaborado pela autora (2019)



```
warning: the default view engine will not be jade in future releases
warning: use '--view=jade' or '--help' for additional options

create : helloworld\
create : helloworld\public\
create : helloworld\public\javascripts\
create : helloworld\public\images\
create : helloworld\public\stylesheets\
create : helloworld\public\stylesheets\style.css
create : helloworld\routes\
create : helloworld\routes\index.js
create : helloworld\routes\users.js
create : helloworld\views\
create : helloworld\views\error.jade
create : helloworld\views\index.jade
create : helloworld\views\layout.jade
create : helloworld\app.js
create : helloworld\package.json
create : helloworld\bin\
create : helloworld\bin\www

change directory:
> cd helloworld

install dependencies:
> npm install

run the app:
> SET DEBUG=helloworld:* & npm start
```

Figura 2.15 – Log de criação da aplicação esqueleto chamada “helloworld”  
Fonte: Elaborado pela autora (2019)

No console, é possível acompanhar o progresso da compilação. No final da saída, o gerador fornece instruções sobre como instalar as dependências (conforme listado no arquivo package.json) e como iniciar sua aplicação (as instruções acima são para Windows; no Linux / macOS, elas serão um pouco diferente).

Sua aplicação esqueleto será criada pelo NPM em uma subpasta do local atual que você escolheu (neste caso, nossa aplicação exemplo foi criada no diretório *myapp*, que criamos anteriormente).

## 2. Entre no novo diretório da sua aplicação “helloworld”

```
cd helloworld
```

Código-Fonte 2.13 – Comando para entrar no diretório da aplicação “helloworld”.  
Fonte: Elaborado pela autora (2019)

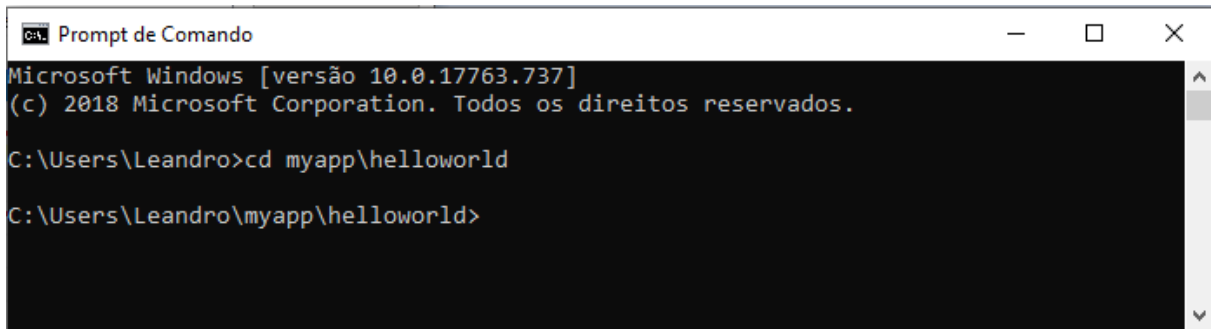


Figura 2.16 – Pasta da aplicação esqueleto “helloworld”

Fonte: Elaborado pela autora (2019)

### 3. Instale todas as dependências para sua aplicação “helloworld” usando o NPM

```
npm install
```

Código-Fonte 2.14 – Comando para instalar dependências em “helloworld”.

Fonte: Elaborado pela autora (2019)

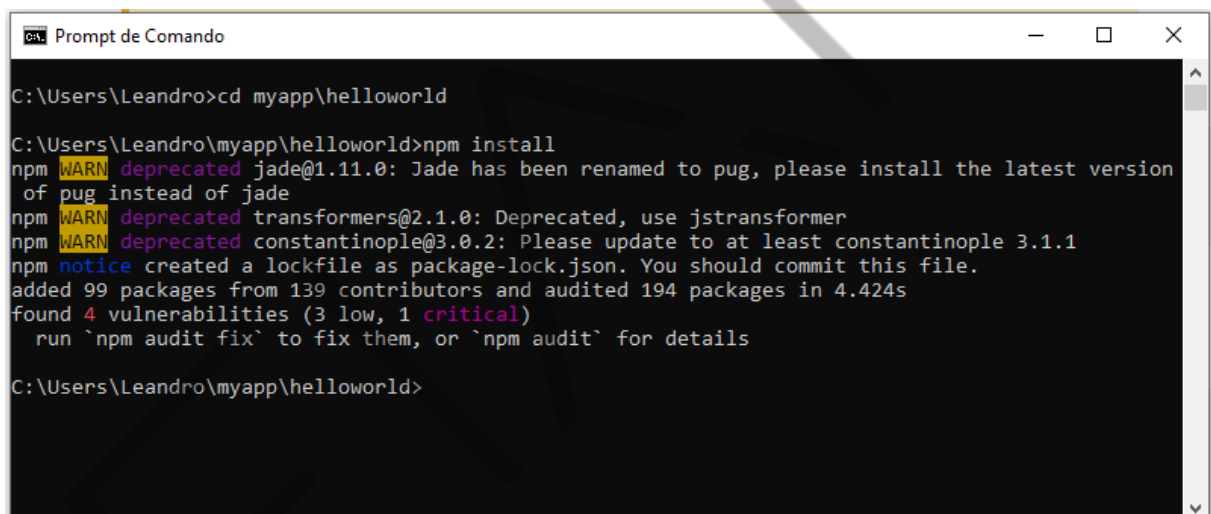


Figura 2.17 – Log de instalação das dependências para a aplicação “helloworld”

Fonte: Elaborado pela autora (2019)

Neste ponto, temos um projeto esqueleto completo. O site ainda não faz muito, mas vale a pena executá-lo para mostrar como funciona.

### 4. Execute sua aplicação (veja os comandos para Windows e Linux / macOS).

#### Comando para Windows

```
SET DEBUG=helloworld:* & npm start
```

Código-Fonte 2.15 – Executar a aplicação esqueleto “helloworld” no Windows.

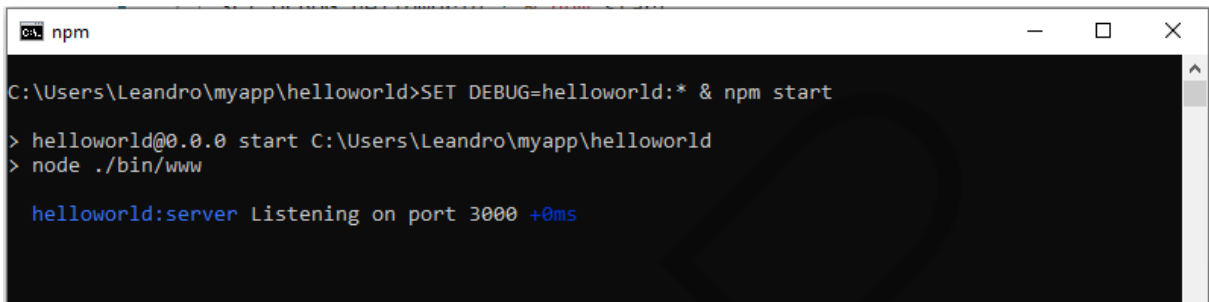
Fonte: Elaborado pela autora (2019)

#### Comando para Linux/macOS

```
DEBUG=helloworld:* npm start
```

Código Fonte 2.16 – Executar a aplicação esqueleto "helloworld" no Linux/MacOS.  
Fonte: Elaborado pela autora (2019)

O comando DEBUG cria um registro seu arquivo package.json, como mostrado abaixo:



```
npm
C:\Users\Leandro\myapp\helloworld>SET DEBUG=helloworld:* & npm start
> helloworld@0.0.0 start C:\Users\Leandro\myapp\helloworld
> node ./bin/www

helloworld:server Listening on port 3000 +0ms
```

Figura 2.18 – Resultado da execução do comando DEBUG  
Fonte: Elaborado pela autora (2019)

5. Volte para a raiz do diretório da aplicação "myapp" e crie um arquivo chamado **app.js** com o seguinte conteúdo:

```
01 const express = require('express')
02 const app = express()
03 const port = 3000
04
05 app.get('/', (req, res) => res.send('Hello World
Express Application Generator!'))
06
07 app.listen(port, () => console.log(`Exemplo app
executando na porta ${port}!`))
```

Código-Fonte 2.17 – Listagem do arquivo app.js da aplicação "helloworld".  
Fonte: Elaborado pela autora (2019)

O código da aplicação Express Generator "HelloWorld" importa o módulo "express" na linha 1, utiliza esse módulo para criar um servidor (uma aplicação) que ouve solicitações HTTP na porta 3000 (linha 7), e imprime uma mensagem no console informando qual porta você pode usar para testar o servidor. A função `app.get()`, na linha 5, só responde a solicitações HTTP GET com o caminho de URL especificado (`/`), neste caso, chamando uma função para enviar nossa mensagem "Hello World!".

6. Inicie o servidor chamando o Node em seu prompt de comando, com o script que você acabou de criar.

```
node app.js
```

Código-Fonte 2.18 – Comando para executar a aplicação “helloworld” com Node.

Fonte: Elaborado pela autora (2019)

7. Navegue pela URL <http://127.0.0.1:3000/> para ver a execução da sua aplicação “Hello World”, criada com o Express Generator.

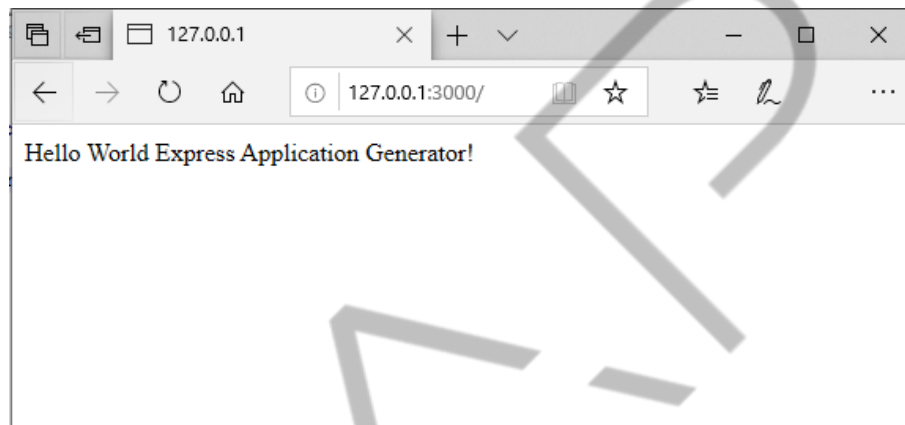


Figura 2.19 – Resultado da execução da aplicação “helloworld” em Node

Fonte: Elaborado pela autora (2019)

Agora você já tem sua primeira aplicação esqueleto em Node no padrão MVC, criada através do Express Generator, funcionando no *localhost* - porta 3000. Essa aplicação possui a toda a estrutura de arquivos padrão e pode ser utilizada para suas futuras implementações.

## 2.4 Um pouco sobre Maven

Apache Maven, ou Maven, é uma ferramenta de automação de compilação que foi construído utilizando uma arquitetura baseada em *plugin*. Essa ferramenta é usada primariamente em projetos Java, mas também pode ser aplicada para construir e gerenciar projetos escritos em C#, Ruby, Scala e outras linguagens.

O Maven baixa bibliotecas Java e seus *plugins* dinamicamente de um ou mais repositórios e armazena-os em uma área de cache local (que fica dentro da estrutura de arquivos da sua aplicação). É possível atualizar esse cache local de artefatos baixados com artefatos criados especificamente para seu projeto. Repositórios públicos também podem ser atualizados.

### 2.4.1 Criando e executando um projeto Maven Web no Eclipse

Nesse passo, você vai criar seu primeiro projeto web utilizando o Maven e agregar os componentes do Spring Boot.

Vamos utilizar o Tomcat, além das ferramentas já indicadas anteriormente, ao longo desse capítulo.

TomCat versão 9:

<<https://tomcat.apache.org/download-90.cgi>>.

#### 1. Crie um projeto Maven Web no Eclipse

Acesse o menu File > New > Project > Maven > Maven Project.

**NÃO** Selecione a opção Create a simple project (skip archetype selection) – para que seja possível escolher o Maven Archetype do seu projeto.

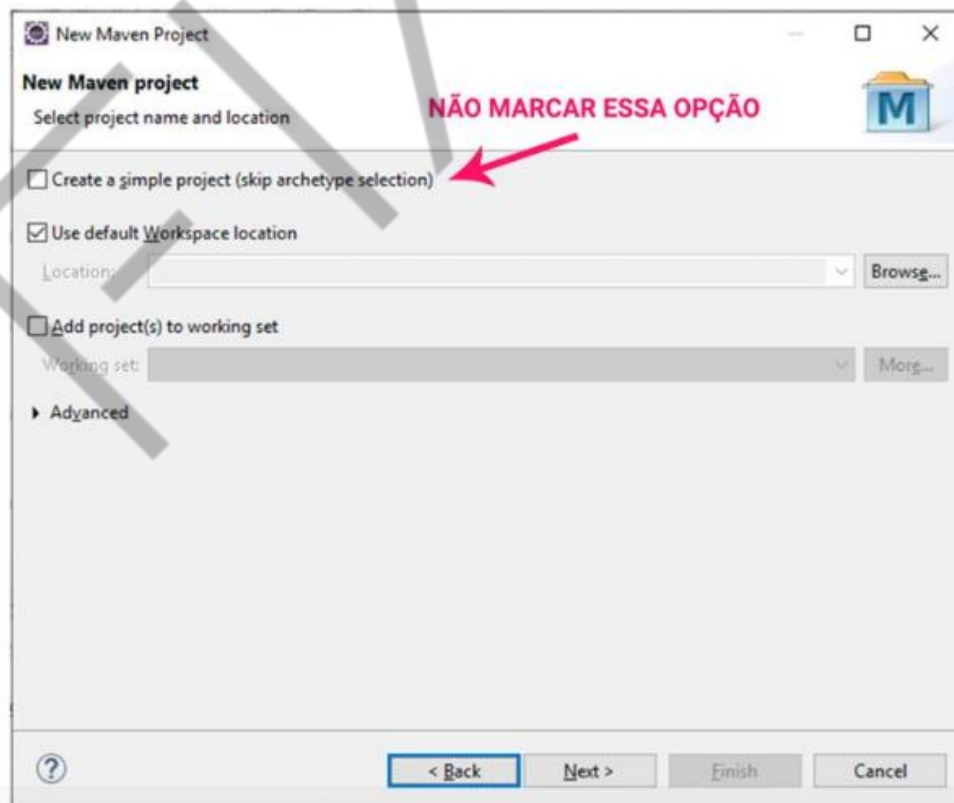


Figura 2.20 – Criando um projeto Maven Web no Eclipse  
Fonte: Elaborado pela autora (2019)

Clique em “Next”.

## 2. Escolha o Archetype maven-archetype-webapp

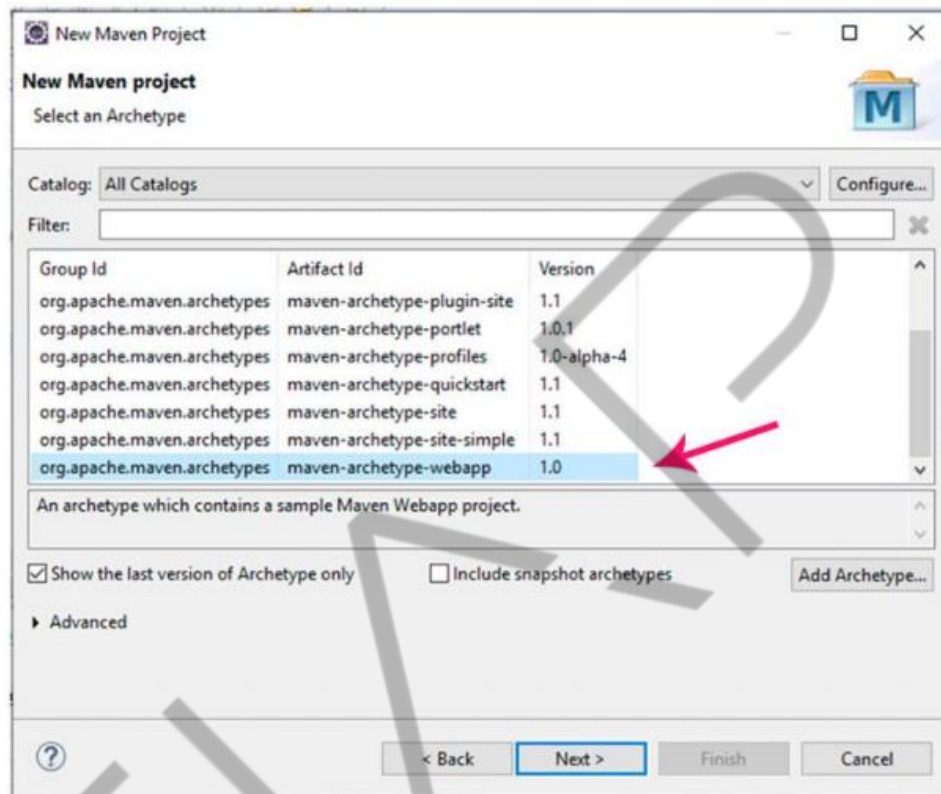


Figura 2.21 – Escolhendo o Archetype maven-archetype-webapp.

Fonte: Elaborado pela autora (2019)

Clique em “Next”.

## 3. Forneça os detalhes para o seu projeto: ID do grupo, ID do Artefato e Versão

Group Id – **br.com.microservice-exercicio1** (é a estrutura de pastas que você deseja criar para seu projeto, será replicado automaticamente em Package).

Artifact Id – **microserviceWebApp** (é o nome da sua aplicação jar).

Version – manter a versão que vier preenchida (pode ser alterada se for o caso de distribuir o seu microserviço em outra versão).

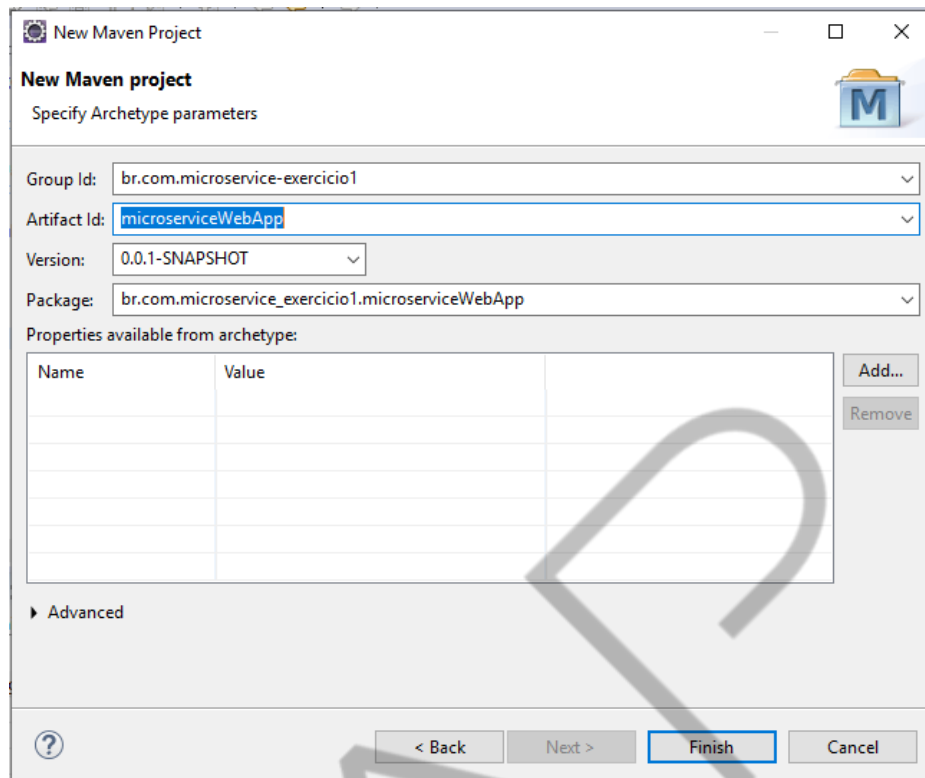


Figura 2.22 – Fornecendo os detalhes para o projeto  
Fonte: Elaborado pela autora (2019)

Clique em “Finish”.

#### 4. Configure as dependências, as propriedades e plugins no arquivo pom.xml.

- a) Para incluir as dependências da API: Abrir o arquivo pom.xml > clicar na aba Dependencies > preencher com os valores abaixo (uma de cada vez) > clicar em OK ou inserir o trecho diretamente no arquivo pom.xml.

O projeto web precisa das dependências de javax.servlet-api e javax.servlet.jsp-api

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.3.3</version>
  <scope>provided</scope>
```

```
</dependency>
```

Código-Fonte 2.19 – Inserindo dependências no arquivo pom.xml – parte 2.  
Fonte: Elaborado pela autora (2019)

- b) Para trabalhar com o starter web do Spring Boot, inserir a seguinte dependência:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Código-Fonte 2.20 – Inserindo dependências no arquivo pom.xml – parte 2.  
Fonte: Elaborado pela autora (2019)

- c) Agora inclua as propriedades.

Abrir o arquivo pom.xml > clicar na aba Overview > Properties > Create > preencher com os valores abaixo (uma de cada vez) > clicar em OK ou inserir o trecho diretamente no arquivo pom.xml.

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
  <java.version>11</java.version>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <maven.compiler.target>${java.version}</maven.compiler.target>
  <spring-boot-dependencies.version>2.2.7.RELEASE</spring-boot-
dependencies.version>
  <microservice.name>microservice-exercicio1</microservice.name>
</properties>
```

Código-Fonte 2.21 – Inserindo propriedades no arquivo pom.xml.  
Fonte: Elaborado pela autora (2019)

- d) Após a inclusão das dependências e propriedades, o arquivo pom.xml deve ficar dessa forma:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.microservice-exercicio1</groupId>
  <artifactId>microserviceWebApp</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>microserviceWebApp Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <parent>
    <groupId>org.springframework.boot</groupId>
```



```

        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.3.RELEASE</version>
        <relativePath />
    </parent>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.3</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    <build>
        <finalName>microserviceWebApp</finalName>
    </build>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>11</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <spring-boot-dependencies.version>2.2.7.RELEASE</spring-boot-
dependencies.version>
        <microservice.name>microservice-exercicio1</microservice.name>
    </properties>
</project>

```

Código-Fonte 2.22 – Listagem final do arquivo pom.xml.  
Fonte: Elaborado pela autora (2019)

## 5. Configurar o microsserviço

Crie um arquivo *application.properties* no diretório *src/main/resources* com as seguintes propriedades:

```

application.name=microserviceWebApp
server.port=80

```

Código-Fonte 2.23 – Criando o arquivo *application.properties*.  
Fonte: Elaborado pela autora (2019)

## 6. Configurar o build do seu microserviço para usar o Maven

- Para executar seu projeto Maven no Eclipse, vá até o Menu do Eclipse, escolha a opção Run > Run Configuration.
- No Assistente para Configuração de Execução (lado esquerdo da tela), clique duas vezes em Maven Build para criar uma nova configuração e forneça as informações de configuração (Name, Base directory, Goals), conforme mostrado na imagem abaixo.

Name: microserviceWebApp

Base directory: clicar em Workspace e escolher a microserviceWebApp

Goals: *clean install*

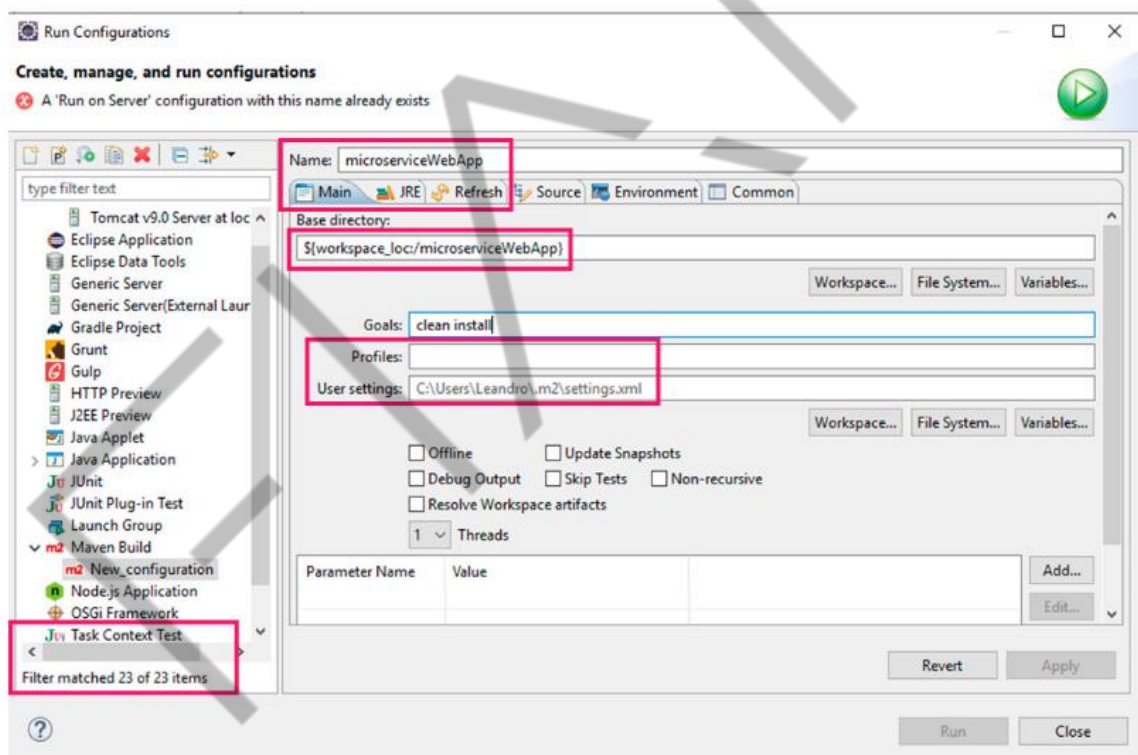


Figura 2.23 – Criando uma nova configuração para executar com Maven  
Fonte: Elaborado pela autora (2019)

- Clique em Run (o Maven será configurado na sua aplicação).

## 7. Criar uma classe Java

Nesse passo, crie uma classe chamada **InicioServlet** no pacote **microserviceWebApp/src/main/resources**

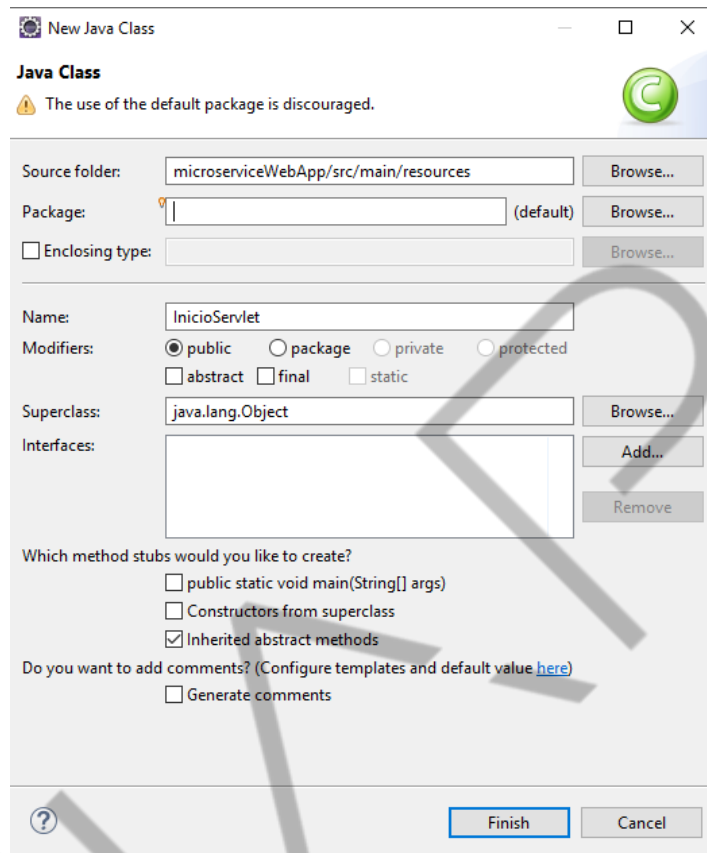


Figura 2.24 – Criando a classe InicioServlet  
Fonte: Elaborado pela autora (2019)

Escreva o seguinte código:

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet ( "/inicio" )
public class InicioServlet extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet (HttpServletRequest req , HttpServletResponse
resp) throws ServletException , IOException {
        resp.setContentType("text/plain" );
        resp.getWriter().write ("Início! Exemplo de Projeto Maven
Web.");
    }
}
```

```
protected void doPost (HttpServletRequest req , HttpServletResponse  
resp) throws ServletException , IOException {  
    doGet(req, resp);  
}  
}
```

Código-Fonte 2.24 – Listagem da classe InicioServlet  
Fonte: Elaborado pela autora (2019)

## 8. Executar seu primeiro projeto Maven Web

Execute e veja o resultado digitando o endereço:  
<http://localhost:8080/microserviceWebApp/inicio>

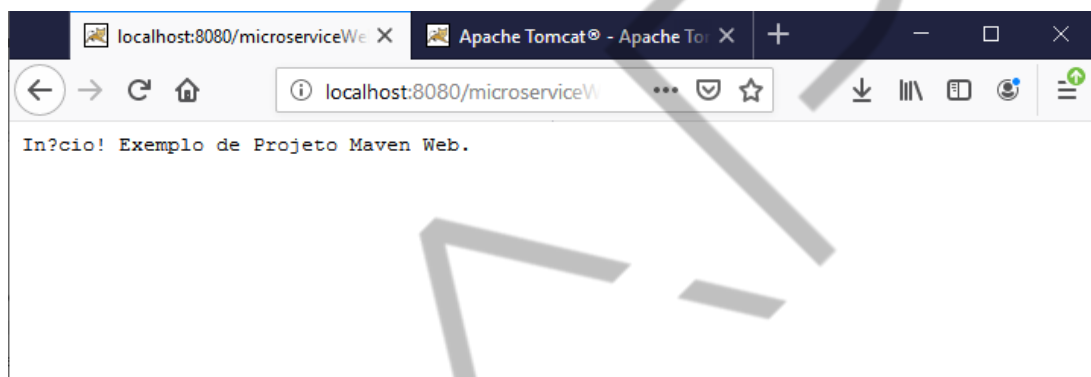


Figura 2.25 – Resultado da execução da classe InicioServlet  
Fonte: Elaborado pela autora (2019)

Pronto, agora seu ambiente de desenvolvimento está configurado!

## CONCLUSÃO

Este capítulo foi recheado de aprendizado não é mesmo? Até aqui você já conseguiu fazer seu “Hello Word” em Node através do Express e já conseguiu executar seu primeiro projeto Maven Web através do Eclipse. Além de saber gerar o esqueleto para uma aplicação Node que vai seguir o padrão MVC.

Mas vamos além!

No próximo capítulo, você vai aprender sobre importantes mecanismos para conectar o universo de aplicações baseadas em microsserviços.

## REFERÊNCIAS

EVANS, Eric. **Domain Driven Design**: Tackling Complexity in the Heart of Software. Addison Wesley, 2004. Page 529.

ECLIPSE. **Eclipse IDE Web Site**. Disponível em: <https://www.eclipse.org/downloads/packages/>. Acesso em: 10 out. 2019.

EXPRESS. **Express Web Site**. Disponível em <https://expressjs.com/>. Acesso em: 10 out. 2019.

FOWLER, Martin. **BoundedContext**. 2014. Disponível em: <https://martinfowler.com/bliki/BoundedContext.html>. Acesso em: 10 out. 2019.

Git. **Git Web Site**. Disponível em <https://git-scm.com/>. Acesso em: 10 out. 2019.

Maven. **Maven Web Site**. Disponível em: <https://maven.apache.org/download.cgi>. Acesso em: 10 out. 2019.

Node. **Node Web Site**. Disponível em: <https://nodejs.org/en/>. Acesso em: 10 out. 2019.

VERNON, Vaughn. **Domain-Driven Design Distilled**. Addison-Wesley Professional; Edição: 1, 2016.

VERNON, Vaughn; FURMANKIEWICZ, Edson. **Implementando Domain Driven Design**. Elsevier/Alta Books, 2019.

## GLOSSÁRIO

<b>API - Application Programming Interface</b>	API é uma estrutura de códigos de programação padronizados para permitir a conexão entre sistemas.
--	--

AVANADO