

Websphere Liberty - Usando Liberty Tools com código VS

"App Modernization não é só sobre runtimes". É Developer Experience ...



Última atualização: Setembro de 2024

Duração: 30 minutos

1. Objetivos

Neste exercício, você aprenderá como os desenvolvedores podem usar o Liberty no modo "dev" com o VS Code Integrated Development Environment para obter desenvolvimento iterativo eficiente, teste, ciclo de debug ao desenvolver aplicativos e microserviços baseados em Java.

No final deste laboratório você deverá ser capaz de:

- Usar a extensão Liberty Tools disponível no VS Code para desenvolver, de forma eficiente, testar e depurar aplicativos nativos da nuvem Java.
- Experimentar o recarregamento quente de código de aplicação e alterações de configuração usando o modo dev

2. Introdução ao Liberty Tools no Código VS

O Liberty Dev mode permite que você, como desenvolvedor, se concentre em seu código. Quando o Open Liberty está em execução no modo dev, seu código é compilado e implementado automaticamente para o servidor em execução, facilitando a iteração em suas alterações.

Neste laboratório, como desenvolvedor, você vai experimentar usando a extensão Open Liberty Tools em VS Code para trabalhar com o seu código, executar testes sob demanda, para que você possa obter feedback imediato sobre suas alterações.

Você também trabalhará com ferramentas integradas de depuração e anexará um depurador Java para depurar o seu aplicativo em execução.

A partir de uma perspectiva do desenvolvedor, trata-se de um enorme ganho de eficiência, uma vez que todas essas atividades de desenvolvimento de loop interno iterativo ocorrem sem nunca deixar o ambiente de desenvolvimento integrado (IDE).

2.1 Revise as extensões do Código VS e arquivos pom.xml usado para este projeto

O aplicativo de exemplo usado neste laboratório está configurado para ser construído com Maven. Todo projeto configurado pelo Maven contém um arquivo **pom.xml**, que define a configuração do projeto, dependências, plug-ins e etc.

O seu arquivo **pom.xml** está no diretório raiz do projeto e está configurado para incluir o liberty-maven-plugin, que permite instalar aplicativos em Open Liberty e gerenciar as instâncias do servidor.

Para começar, navegue até o diretório do projeto e revise o arquivo IDE extensões e pom.xml que é usado para o microserviço "**system**" que é fornecido em laboratório.

Primeiro, inclua a pasta do projeto em um Espaço de Trabalho do Código VS

1. **Fechar todas as janelas do Terminal e Guias Browser utilizadas em qualquer laboratório anterior.**
2. Use o Ícone de **Activities** para alternar para a barra de ferramentas, em seguida, clique no ícone Terminal para abrir uma janela do Terminal.



3. Clone o repo GitHub que inclui artefatos necessários para este laboratório:

```
mkdir -p /home/techzone/Student/labs
git clone https://github.com/openliberty/guide-getting-started.git /home/techzone/Student/labs/vscode
cd /home/techzone/Student/labs/vscode
```

Uma vez concluído, o repo de artefatos de laboratório local é clonado no seguinte diretório na VM de desktop.

/home/techzone/Student/labs/vscode

4. Navegue até o diretório do projeto e ative o VS Code no diretório do projeto “**start**”.

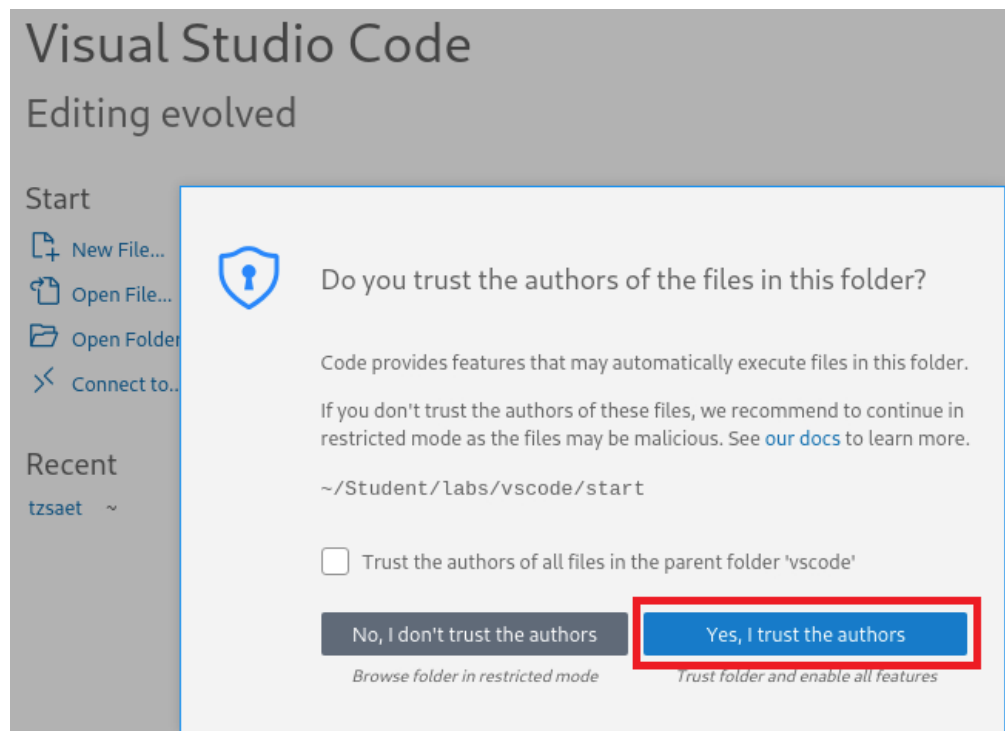
- a. Abra uma janela do terminal e mude para o seguinte diretório:

```
cd /home/techzone/Student/labs/vscode/start
```

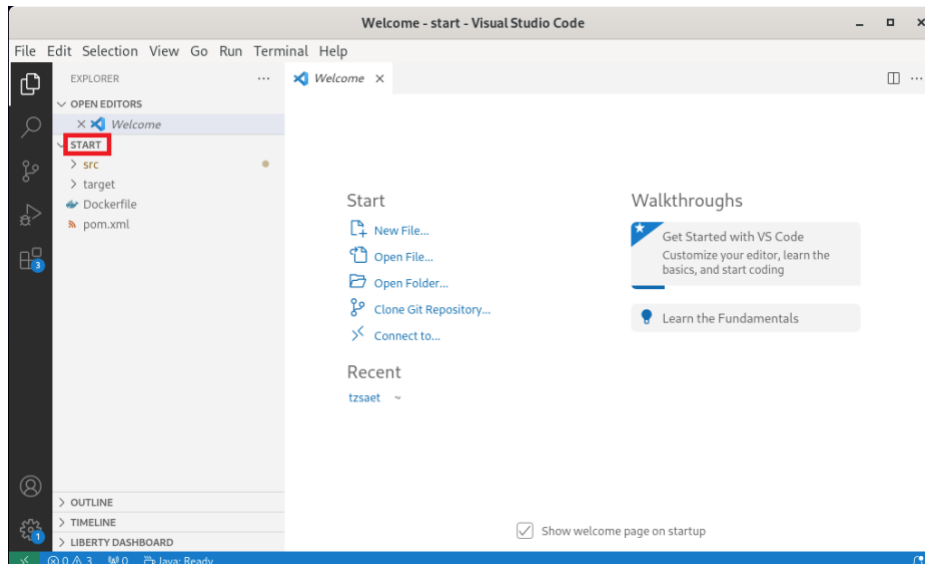
5. Ative o Código VS usando o diretório atual como a pasta raiz para a área de trabalho código.

```
code .
```

Click em “**Yes, I trust the authors**”!

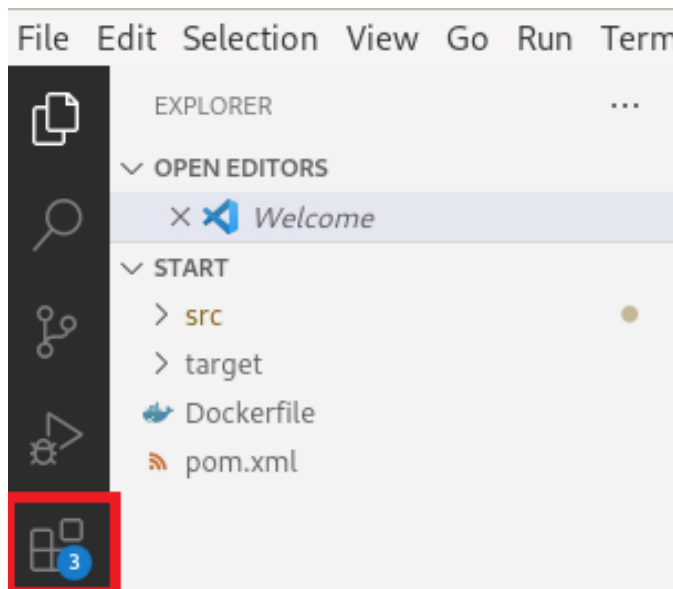


Na abertura da UI do VS Cod, a visualização Explorer é mostrada. A pasta "START" contém o código-fonte para o projeto.



6. Revise as extensões instaladas no VS Code que são usadas para este laboratório.

a. Clique no ícone **Extensions** na barra de navegação esquerda no Código VS.




b. Expandir a seção de extensões "INSTALADAS" para listar as extensões que estão atualmente instaladas neste ambiente. As extensões notáveis usadas neste laboratório são:

- Liberty Tools
- Tools for MicroProfile
- Language Support for Java
- Debugger for Java

- c. Clique na extensão "**Liberty Tools**" para visualizar seus detalhes.
- d. Observe a lista de comandos que são suportados pela extensão Liberty Tools.

Extension: Liberty Tools X




Liberty Tools

v24.0.3

Open Liberty openliberty.io | 39,606 | ★★★★★ (2)

Liberty Tools for Visual Studio Code

[Disable](#) [Uninstall](#) 

This extension is enabled globally.

[DETAILS](#) [FEATURE CONTRIBUTIONS](#) [CHANGELOG](#) [DEPENDENCIES](#) [RUNTIME STATUS](#)

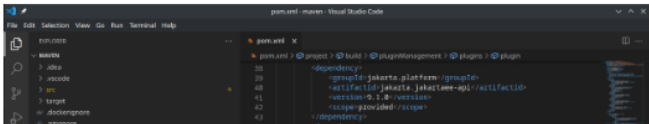
Liberty Tools for Visual Studio Code

Note: Requires **Visual Studio Code 1.78.0** or later, and **Java 17** or later.

This page provides an overview of Liberty Tools for Visual Studio Code. For minimum requirements information and detailed instructions on how to use Liberty Tools, check the [user-guide](#).

[VS MARKET](#) [V24.0.3](#) [LICENSE](#) [EPL-2.0](#)

Liberty Tools for Visual Studio Code offers features for developing cloud-native Java applications with [Open Liberty](#) and [WebSphere Liberty](#). Iterate fast with Liberty dev mode, code with assistance for MicroProfile & Jakarta EE APIs, and easily edit Liberty configuration files.



Categories

Programming Languages

Extension Resources

[Marketplace](#)
[Repository](#)
[License](#)
[Open Liberty](#)


More Info

Published 2019-09-15, 19:43:35
Last released 2024-03-11, 18:51:13
Last updated 2024-03-15, 06:08:08
Identifier open-liberty, liberty-dev-vscode-ext

- e. Clique em **Dependencies** da página de detalhes do Liberty Tools.

Observe o requisito do "**Tools for MicroProfile**" para suportar o desenvolvimento de Microserviços que utilizam APIs do MicroProfile com o Open Liberty.

Extension: Liberty Tools X




Liberty Tools

v24.0.3


Open Liberty openliberty.io | 39,606 | ★★★★★ (2)

Liberty Tools for Visual Studio Code


[Disable](#) [Uninstall](#) 

This extension is enabled globally.


[DETAILS](#) [FEATURE CONTRIBUTIONS](#) [CHANGELOG](#) [DEPENDENCIES](#) [RUNTIME STATUS](#)

>  **Tools for MicroProfile** [redhat.vscode-microprofile](#)

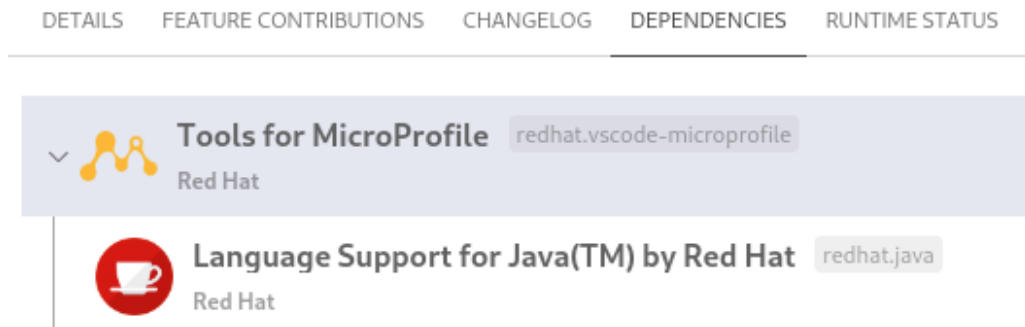
Red Hat

 **XML** [redhat.vscode-xml](#)

Red Hat

 **Debugger for Java** [vscjava.vscode-java-debug](#)

Microsoft

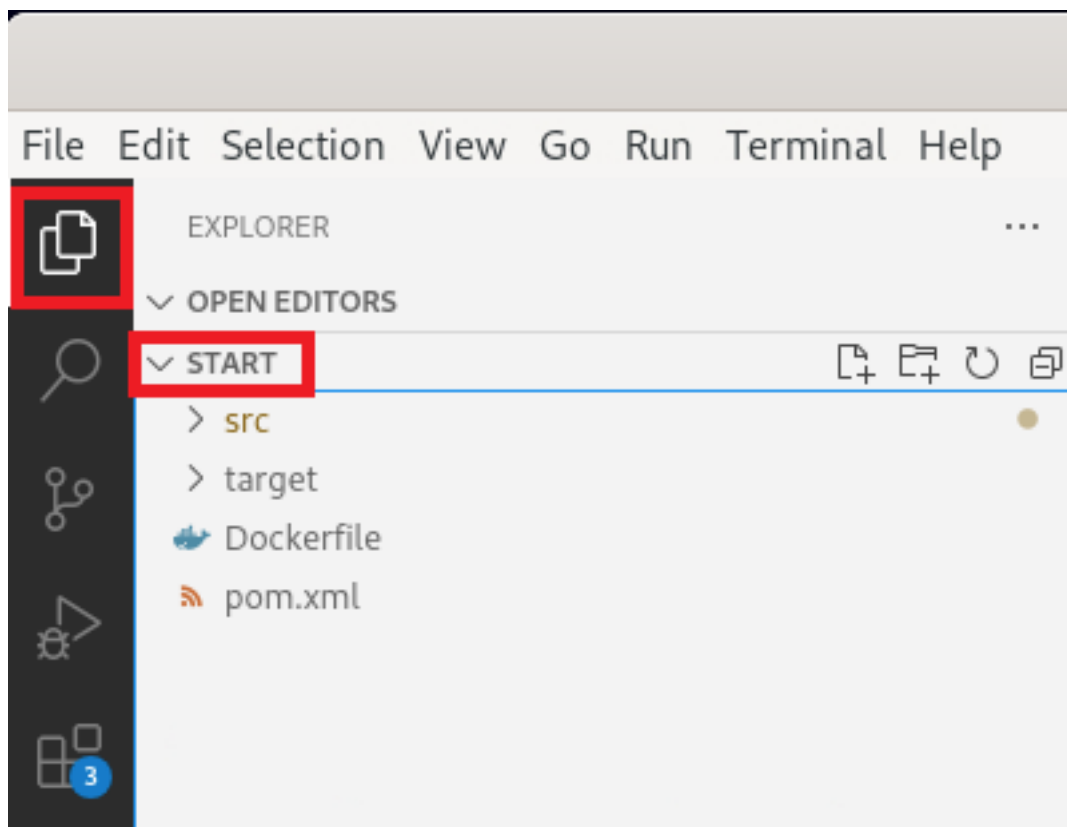


f. Feche a página de detalhes da **Liberty Tools Extension**

7. Revise o arquivo **pom.xml** usado para configurar e construir o microserviço **"system"**.

a. Clique no ícone Explorer localizado na barra de navegação esquerda no Código VS.

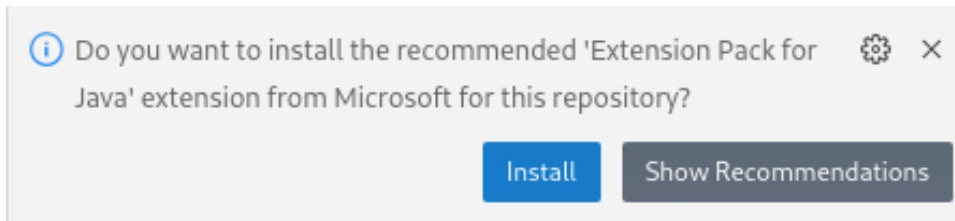
b. Expandir a pasta START se ela ainda não for expandida



c. Clique no arquivo **pom.xml** para abri-lo na área de janela do editor

d. Feche quaisquer caixas Pop-up perguntando se você deseja instalar extensões ou comutar visualizações.

Nota: Você pode ver pop-ups adicionais, basta fechá-los ou ignorá-los.



e. Observe o empacotamento do arquivo war de aplicação Java que é produzido a partir do Maven Build. O arquivo WAR produzido será nomeado **guide-getting-started** version 1.0-SNAPSHOT.

```
7      <groupId>io.openliberty.guides</groupId>
8      <artifactId>guide-getting-started</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <packaging>war</packaging>
```

f. Portas HTTP e HTTPS padrão são definidas, e substituídas no arquivo server.xml

```
17     <!-- Liberty configuration -->
18     <liberty.var.http.port>9080</liberty.var.http.port>
19     <liberty.var.https.port>9443</liberty.var.https.port>
```

g. O Plugin do Liberty Tools está ativado, com uma versão suportada de 3.10.3

```
67     <!-- Enable liberty-maven plugin -->
68     <plugin>
69       <groupId>io.openliberty.tools</groupId>
70       <artifactId>liberty-maven-plugin</artifactId>
71       <version>3.10.3</version>
72     </plugin>
```

h. O plugin para Testes em execução também é adicionado à configuração do Maven, que potencializam as dependências de teste também definidas no arquivo pom.xml.

```

83 |         <!-- Plugin to run functional tests -->
84 |         <plugin>
85 |             <groupId>org.apache.maven.plugins</groupId>
86 |             <artifactId>maven-failsafe-plugin</artifactId>
87 |             <version>3.3.0</version>
88 |             <configuration>
89 |                 <systemPropertyVariables>
90 |                     <http.port>${liberty.var.http.port}</http.port>
91 |                     <context.root>/</context.root>
92 |                 </systemPropertyVariables>
93 |             </configuration>
94 |         </plugin>

```

i. Feche o arquivo pom.xml

Informações:

Dica: Informações adicionais sobre o liberty-maven-plugin podem ser encontradas aqui: <https://github.com/OpenLiberty/ci.maven>

2.2 Usando o Liberty Tools no Código VS

Nesta seção do laboratório, você usará o Liberty Tools em Código VS para trabalhar com o seu código e executar testes sob demanda, para que você obtenha feedback imediato sobre suas alterações.

Importante:

Para o Liberty Tools (LIBERTY DASHBOARD)

O VS Code fornece extensões para Java para suportar os recursos de linguagem Java.

VS Code para Java suporta dois modos.

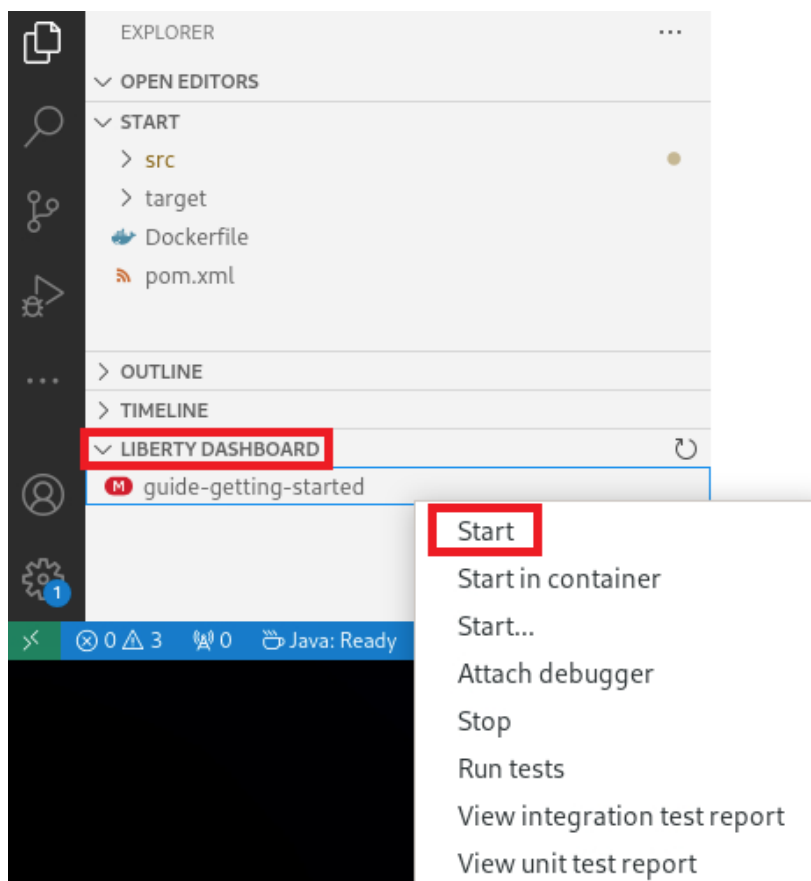
- *Lightweight mode*
- *Standard mode*

O VS Code possui uma configuração padrão chamada "modo híbrido" onde uma área de trabalho é aberta no modo Lightweight, mas conforme necessário, você é solicitado a mudar para o modo Standard.

*A Extensão Tools for MicroProfile, necessária para a extensão Liberty Tools, requer que a área de trabalho Java seja aberta no modo "**STANDARD**". Caso contrário o LIBERTY DASHBOARD não funcionará corretamente.*

Dica: neste ambiente de laboratório, a área de trabalho já está configurada para usar o modo Standard. Para obter mais detalhes sobre o VS Code for Java está disponível aqui: <https://code.visualstudio.com/docs/java/java-project>

1. Use o Liberty Dashboard para iniciar o Liberty Server no modo dev
 - a. No Código VS, expanda a seção LIBERTY DASHBOARD
 - b. Clique com o botão direito do mouse sobre **guide-getting-started** no servidor Liberty.
 - c. Selecione **Start** a partir do menu para iniciar o servidor



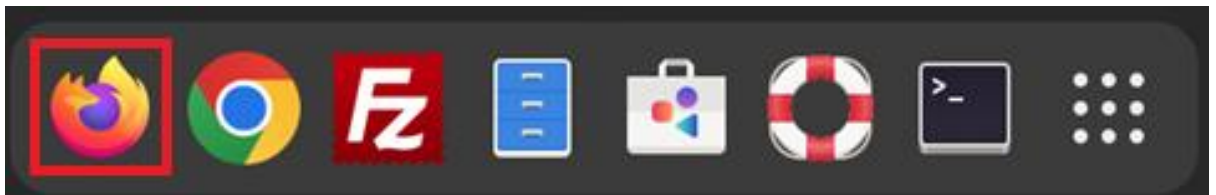
- d. A visualização do Terminal se abre, e você vê as mensagens de log do servidor conforme o início do servidor. Quando a mensagem a seguir aparece no Terminal, o servidor Liberty é iniciado.

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS guide-getting-started (liberty dev) + - [ ] [ ] ... ^ x

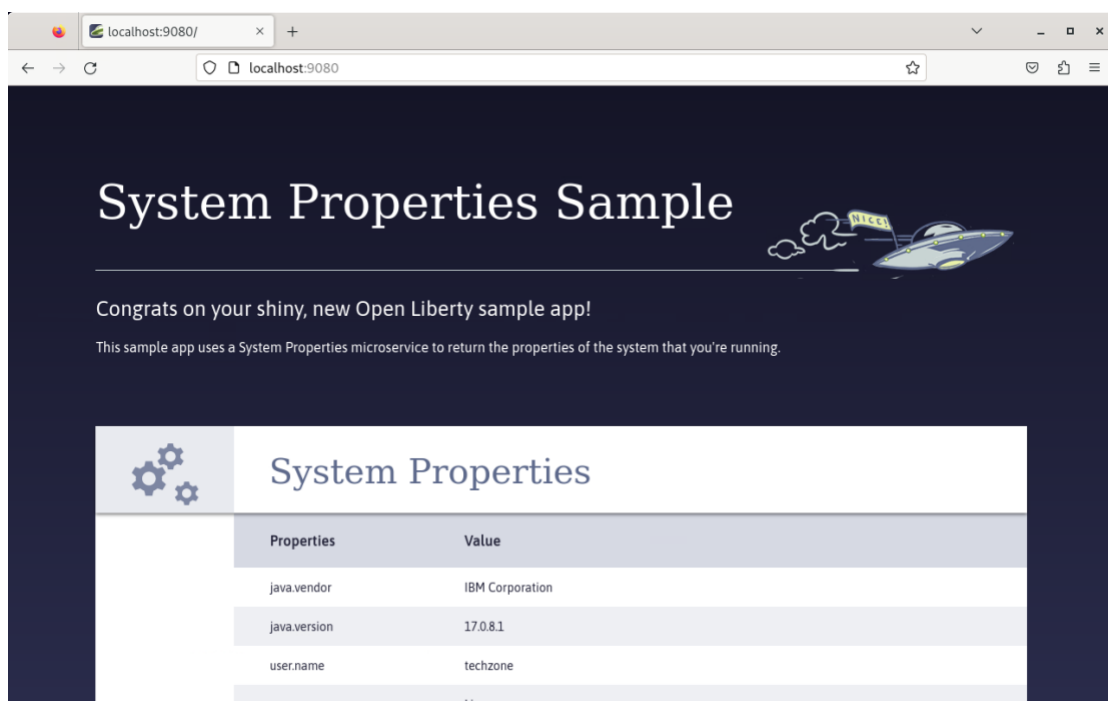
[INFO] *      Liberty server HTTP port: [ 9080 ]
[INFO] *      Liberty server HTTPS port: [ 9443 ]
[INFO] *      Liberty debug port: [ 7777 ]
[INFO] *****
[INFO] Source compilation was successful.
[INFO] Tests compilation was successful.
[INFO] [AUDIT ] CWWKT0017I: Web application removed (default_host): http://rhel9-base.gym.lan:9080/
[INFO] [AUDIT ] CWWKZ0009I: The application guide-getting-started has stopped successfully.
[INFO] [AUDIT ] CWWKS4104A: LTPA keys created in 11.494 seconds. LTPA key file: /home/techzone/Student/labs/vsco
de/start/target/liberty/wlp/usr/servers/defaultServer/resources/security/ltpa.keys
[INFO] [AUDIT ] CWWKT0016I: Web application available (default_host): http://rhel9-base.gym.lan:9080/
[INFO] [AUDIT ] CWWKZ0003I: The application guide-getting-started updated in 0.797 seconds.
```

2. Execute o aplicativo de exemplo **"System Properties Sample"** a partir de um navegador web

a. Use o ícone Atividades para alternar para a barra de ferramentas, em seguida, clique no ícone do Firefox para abrir uma janela do navegador Firefox.



b. Acesse <http://localhost:9080> para verificar o aplicativo está em execução.



2.3 Experiência do Desenvolvedor Usando o Liberty Tools no VS Code

O aplicativo de exemplo Propriedades do Sistema está em funcionamento no servidor Liberty. Em seguida, como desenvolvedor, você deseja implementar um **health check** para a aplicação.

A experiência do desenvolvedor é *frictionless*, já que toda mudança de configuração e código que o desenvolvedor realize, são detectados automaticamente e o servidor e o aplicativo são atualizados dinamicamente no servidor em execução para refletir o código e a configuração atualizados.

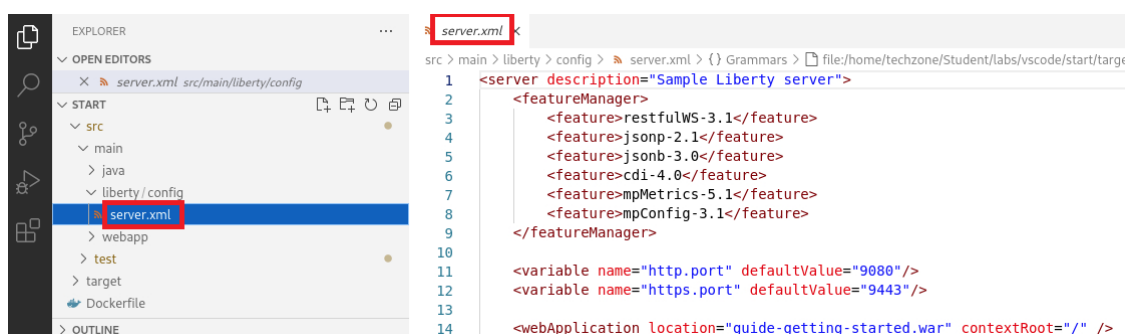
Vamos explorar exemplos da experiência de desenvolvimento muito eficiente ao implementar alguma nova capacidade em nosso serviço.

Neste exemplo, você irá utilizar o recurso **mpHealth-4.0** no Open Liberty, que implementa a API do MicroProfile **mpHealth-4.0**, para utilizar health check para a aplicação.

O recurso mpHealth-4.0 fornece um endpoint **/health** que representa um status binário, UP ou DOWN, dos microserviços instalados.

Para saber mais sobre o recurso mpHealth MicroProfile, visite: <https://www.openliberty.io/docs/24.0.0.6/health-check-microservices.html>

1. Atualize o arquivo de configuração do servidor Liberty (**server.xml**) para incluir o recurso **mpHealth-4.0** para começar a implementar as verificações de funcionamento para o aplicativo.
 - a. Na visualização do VS Code Explorer, navegue até **START-> src-> main-> liberty / config**
 - b. Clique em **server.xml** para abrir o arquivo na área de janela do editor



- c. Adicione o recurso **mpHealth-4.0** no arquivo server.xml utilizando o texto abaixo:

```
<feature>mpHealth-4.0</feature>
```

server.xml X

src > main > liberty > config > server.xml > server

```
1  <server description="Sample Liberty server">
2      <featureManager>
3          <feature>restfulWS-3.1</feature>
4          <feature>jsonp-2.1</feature>
5          <feature>jsonb-3.0</feature>
6          <feature>cdi-4.0</feature>
7          <feature>mpMetrics-5.1</feature>
8          <feature>mpConfig-3.1</feature>
9          <feature>mpHealth-4.0</feature>
10     </featureManager>
--
```

- d. Salve e feche o arquivo server.xml

Quando o arquivo server.xml é salvo, as alterações de configuração são detectadas, e o servidor é atualizado dinamicamente, instalando o novo recurso e atualizando o aplicativo no servidor em execução.

2. Visualize as mensagens na visualização do Terminal, mostrando o recurso sendo instalado e o aplicativo sendo atualizado.

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS guide-getting-started (liberty dev) + - [] [] ... ^ X

```

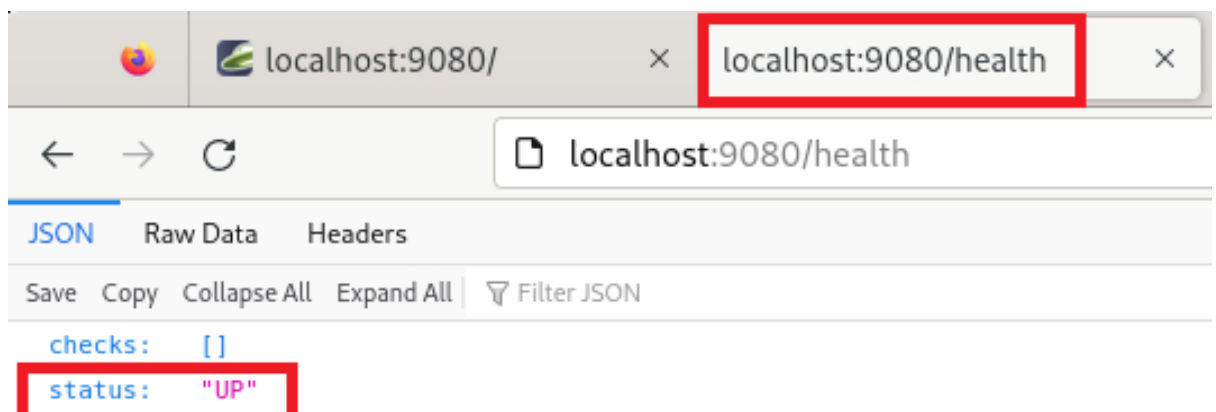
/target/liberty/wlp/usr/servers/defaultServer/configDropins/overrides/liberty-plugin-variable-config.xml
[INFO] [AUDIT] CWWKT0017I: Web application removed (default host): http://rhel9-base.gym.lan:9080/
[INFO] [AUDIT] CWWKZ0009I: The application guide-getting-started has stopped successfully.
[INFO] [AUDIT] CWWKG0017I: The server configuration was successfully updated in 0.904 seconds.
[INFO] [AUDIT] CWWKT0016I: Web application available (default host): http://rhel9-base.gym.lan:9080/health/
[INFO] [AUDIT] CWWKF1037I: The server added the [mpHealth-4.0] features to the existing feature set.
[INFO] [AUDIT] CWWKF0012I: The server installed the following features: [cdi-4.0, distributedMap-1.0, jndi-1.0,
json-1.0, jsonb-3.0, jsonp-2.1, monitor-1.0, mpConfig-3.1, mpHealth-4.0, mpMetrics-5.1, restfulWS-3.1, restfulWSC
lient-3.1, ssl-1.0, transportSecurity-1.0].
[INFO] [AUDIT] CWWKF0008I: Feature update completed in 1.084 seconds.
[INFO] [AUDIT] CWWKT0016I: Web application available (default host): http://rhel9-base.gym.lan:9080/
[INFO] [AUDIT] CWWKZ0003I: The application guide-getting-started updated in 0.727 seconds.

```

Uma vez que as alterações são salvas, e o servidor é atualizado automaticamente, o novo endpoint **/health** está disponível.

3. A partir do navegador da Web no VM acesse o terminal **/health** para visualizar o status de funcionamento do aplicativo.

```
http://localhost:9080/health
```



Atualmente, a verificação básica de saúde fornece um status simples indicando se o serviço está em execução, mas não se ele é saudável.

Nas próximas etapas, você implementará a verificação **liveness** que implementa a lógica que reúne informações de uso da memória e do cpu e informa o serviço DOWN na verificação de funcionamento se os recursos do sistema excedem um determinado limite.

Você também implementará uma verificação **readiness** que verifica a configuração de propriedade externa no arquivo server.xml, que é usado para colocar o serviço em modo de manutenção. E se o serviço estiver em modo de manutenção, o serviço é marcado DOWN a partir da verificação de funcionamento.

4. Copiar uma implementação do **SystemReadinessCheck.java** para o projeto

- a. Use o Ícone de **Activities** para alternar para a barra de ferramentas, em seguida, clique no ícone Terminal para abrir uma janela do Terminal.



- c. Execute o comando a seguir para copiar o SystemReadinessCheck.java para o projeto.

```
cp
/home/techzone/Student/labs/vscode/finish/src/main/java/
io/openliberty/sample/system/SystemReadinessCheck.java
/home/techzone/Student/labs/vscode/start/src/main/java/i
o/openliberty/sample/system/SystemReadinessCheck.java
```

Informações:

Para os fins do laboratório, o comando copy acima copia um readiness do projeto "finished", para o projeto de trabalho atual.

5. Revisar a implementação **SystemReadinessCheck.java**

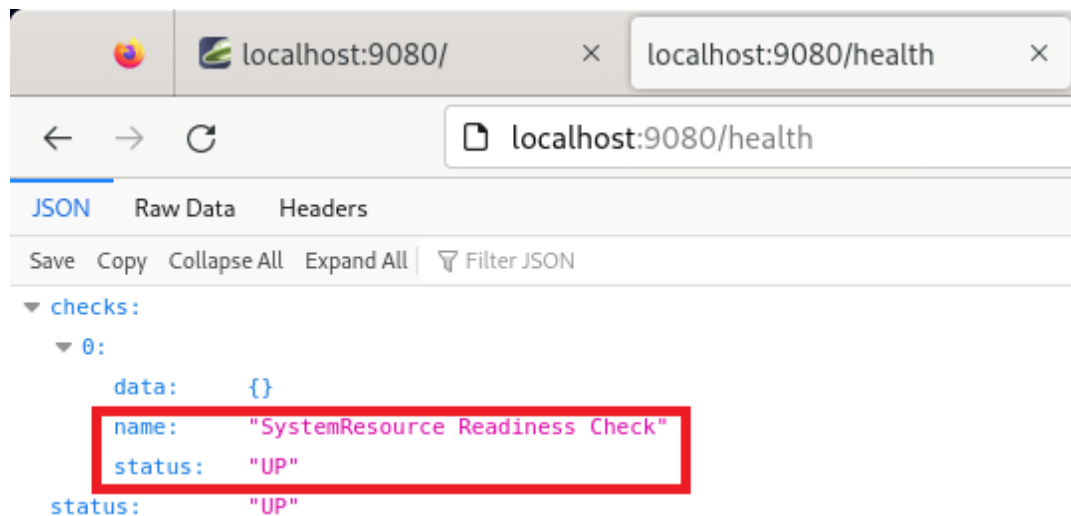
- a. Retornar para a visualização do VS Code Explorer
- b. Navegar para **START> main> java / io / openliberty / sample / system**
- c. Clique no arquivo **SystemReadinessCheck.java** para abri-lo na área de janela do editor

```
24 @Readiness
25 @ApplicationScoped
26 // tag::systemReadinessCheck[]
27 public class SystemReadinessCheck implements HealthCheck {
28
29     private static final String READINESS_CHECK = SystemResource.class.getSimpleName()
30         + " Readiness Check";
31
32     @Inject
33     @ConfigProperty(name = "io.openliberty.guides.system.inMaintenance")
34     Provider<String> inMaintenance;
35
36     @Override
37     public HealthCheckResponse call() {
38         if (inMaintenance != null && inMaintenance.get().equalsIgnoreCase("true")) {
39             return HealthCheckResponse.down(READINESS_CHECK);
40         }
41         return HealthCheckResponse.up(READINESS_CHECK);
42     }
43
44 }
45 // end::systemReadinessCheck[]
```

O **SystemReadinessCheck** simplesmente avalia o ConfigProperty " **inMaintenance**", que é implementado através do recurso mpConfig MicroProfile, e configurado no arquivo **server.xml** do Liberty Server.

- Se a propriedade " **inMaintenance**" for configurada como " **false**" a verificação readiness configura o status health para **UP**.
 - Se a propriedade " **inMaintenance**" for configurada como " **true**" o status é configurado como **DOWN**.
6. A partir do Web Browser na VM, reexecute o terminal **/health** para visualizar o status de funcionamento do aplicativo.

```
http://localhost:9080/health
```



Informações:

Você percebeu que, ao implementar o novo código de verificação de prontidão no aplicativo, que você não teve que reiniciar o aplicativo ou Liberty Server?

O Liberty Tools detectou as alterações de código no projeto, e atualizou dinamicamente o aplicativo no servidor em execução.

7. Copiar uma implementação do **SystemLivenessCheck.java** para o projeto

a. Abrir uma janela do Terminal na VM

b. Execute o comando a seguir para copiar o SystemLivenessCheck.java para o projeto

```
cp
/home/techzone/Student/labs/vscode/finish/src/main/java/
io/openliberty/sample/system/SystemLivenessCheck.java
/home/techzone/Student/labs/vscode/start/src/main/java/i
o/openliberty/sample/system/SystemLivenessCheck.java
```

Informações:

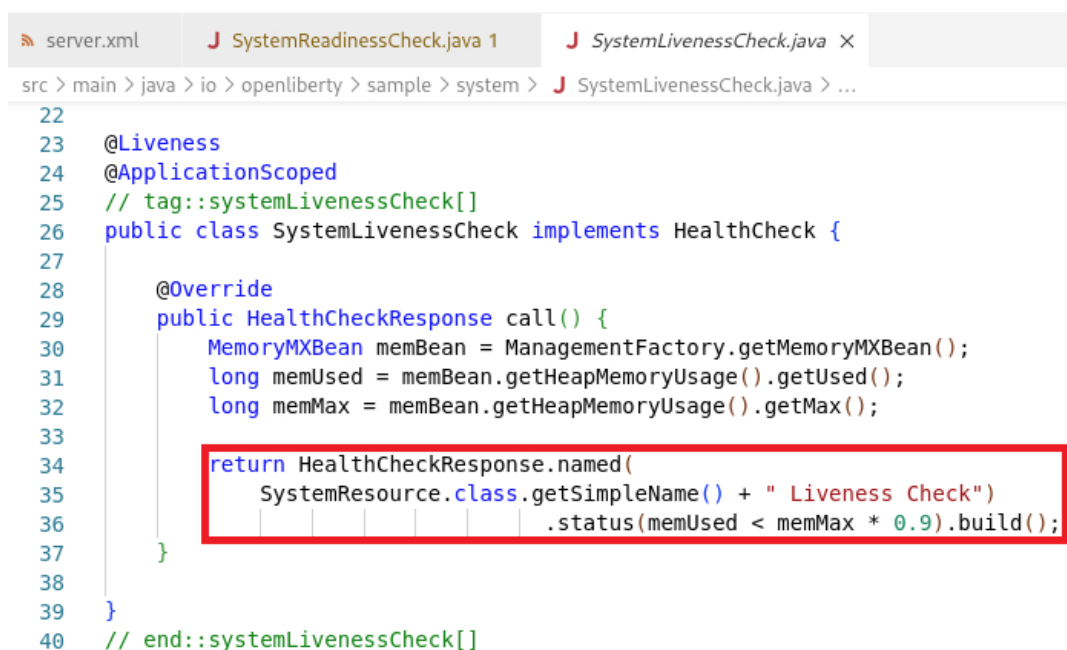
Para os fins do laboratório, o comando copy acima copia um liveness check implementado do projeto "finished", para o projeto de trabalho atual.

8. Revise a implementação **SystemLivenessCheck.java**

a. Retorne para a visualização do VS Code Explorer

b. Navegue para **START-> main-> java / io / openliberty / sample / system**

c. Clique no arquivo **SystemLivenessCheck.java** para abri-lo na área de janela do editor



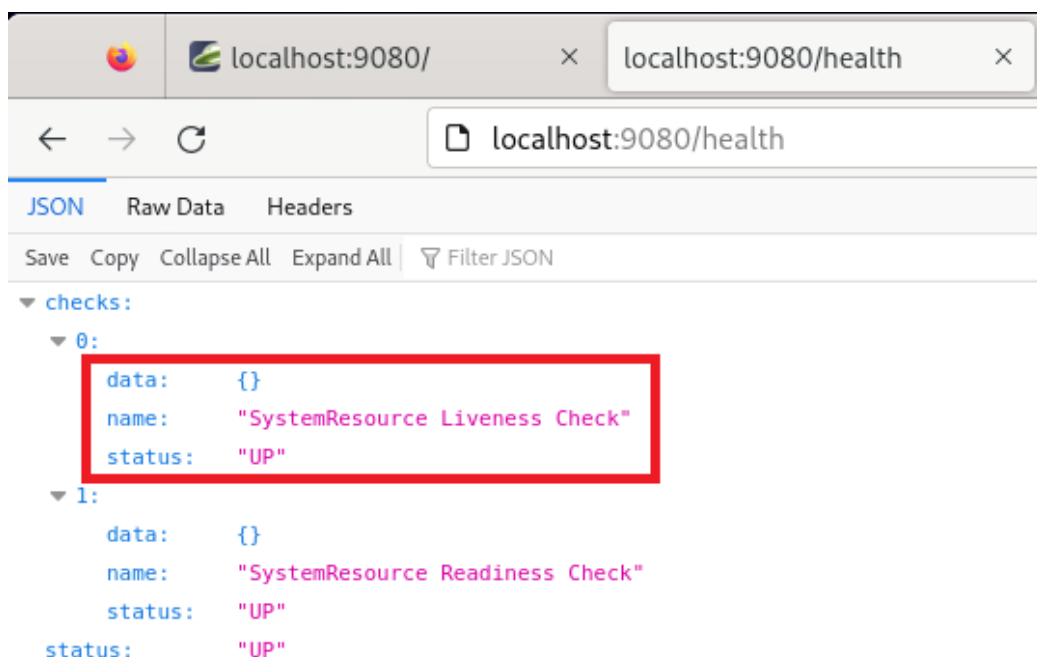
```
server.xml  SystemReadinessCheck.java 1  SystemLivenessCheck.java X
src > main > java > io > openliberty > sample > system > SystemLivenessCheck.java > ...
22
23 @Liveness
24 @ApplicationScoped
25 // tag::systemLivenessCheck[]
26 public class SystemLivenessCheck implements HealthCheck {
27
28     @Override
29     public HealthCheckResponse call() {
30         MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
31         long memUsed = memBean.getHeapMemoryUsage().getUsed();
32         long memMax = memBean.getHeapMemoryUsage().getMax();
33
34         return HealthCheckResponse.named(
35             SystemResource.class.getSimpleName() + " Liveness Check")
36             .status(memUsed < memMax * 0.9).build();
37     }
38
39 }
40 // end::systemLivenessCheck[]
```


O **SystemLivenessCheck** avalia os recursos de "memória" e "cpu" utilizados.

- Se a "memória" usada for menor que 90%, o **liveness** probe estabelece o status para **UP**.
- Se a "memória" usada for maior que 90%, o **liveness** probe estabelece o status para **DOWN**.

9. A partir do Web Browser na VM, reexecute o terminal **/health** para visualizar o status de funcionamento do aplicativo.

```
http://localhost:9080/health
```



Observação: no caso em que há múltiplas verificações de funcionamento sendo executadas, como no nosso exemplo, **TODAS** as verificações de funcionamento devem ter o status **UP** para que o serviço seja marcado **UP**.

Então, o que acontece quando mudamos a propriedade de **inManutenção** para **"true"**?

Vamos modificar a configuração externa para configurar o serviço em modo de manutenção e ver os resultados das verificações de funcionamento.

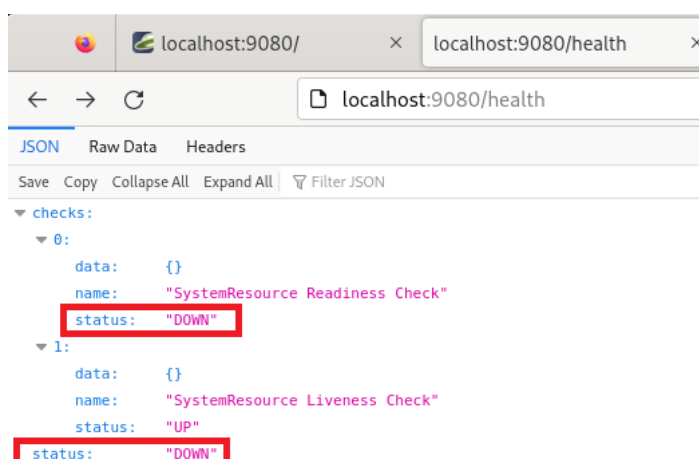
10. Modifique a propriedade inManutenção no arquivo server.xml

- Retorne para o console VS Code e navegue até **START-> src-> main-> liberty / config**.
- Clique em **server.xml** para abrir o arquivo no editor
- Modificar o valor da variável **inMaintenance** para **"true"** conforme ilustrado abaixo:
- Salvar o arquivo **server.xml**. A configuração do servidor é atualizada dinamicamente para refletir a atualização.

```
src > main > liberty > config > server.xml > ...
1  <server description="Sample Liberty server">
2      <featureManager>
3          <feature>restfulWS-3.1</feature>
4          <feature>jsonp-2.1</feature>
5          <feature>jsonb-3.0</feature>
6          <feature>cdi-4.0</feature>
7          <feature>mpMetrics-5.1</feature>
8          <feature>mpConfig-3.1</feature>
9          <feature>mpHealth-4.0</feature>
10     </featureManager>
11
12     <variable name="http.port" defaultValue="9080"/>
13     <variable name="https.port" defaultValue="9443"/>
14
15     <webApplication location="guide-getting-started.war" contextRoot="/" />
16
17     <mpMetrics authentication="false"/>
18
19     <httpEndpoint host="*" httpPort="${http.port}"
20         httpsPort="${https.port}" id="defaultHttpEndpoint"/>
21
22     <variable name="io_openliberty_guides_system_inMaintenance" value="true"/>
23 </server>
```

11. A partir do Web Browser na VM, reexecute o endpoint **/health** para visualizar o status de funcionamento do aplicativo.

http://localhost:9080/health

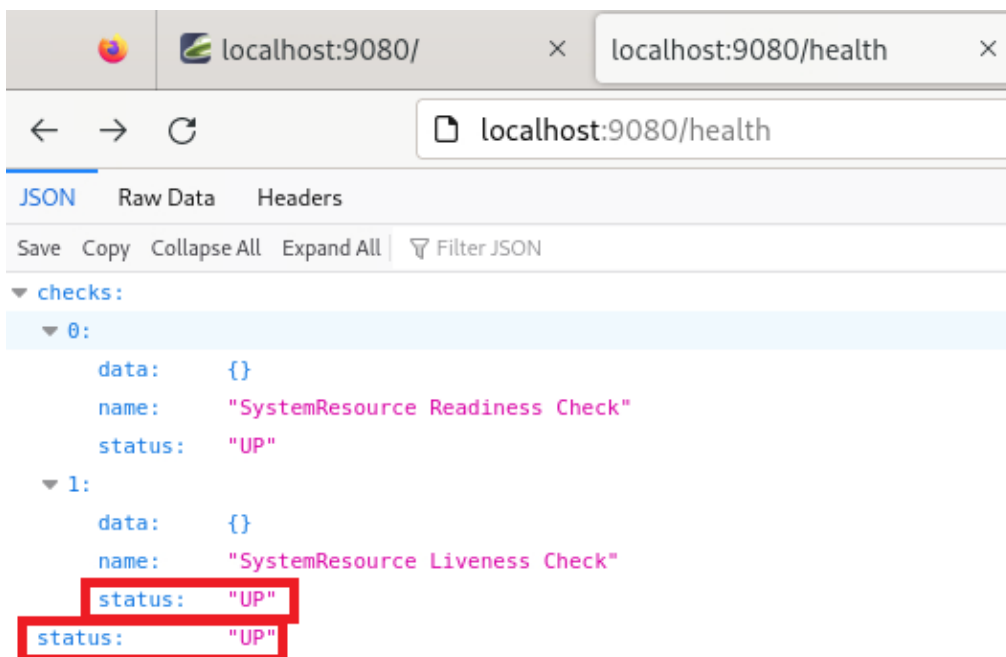


12. No arquivo **server.xml**, altere a variável **inMaintenance** de volta para **false**

- a. Salve o arquivo server.xml
- b. Feche a visualização do editor server.xml

```
21 |  
22 |     <variable name="io_openliberty_guides_system_inMaintenance" value="false"/>  
23 | </server>
```

13. Retorne ao endpoint **/health** para verificar se o serviço agora está marcado UP novamente.



2.4 Executando Testes usando o Liberty Tools no VS Code

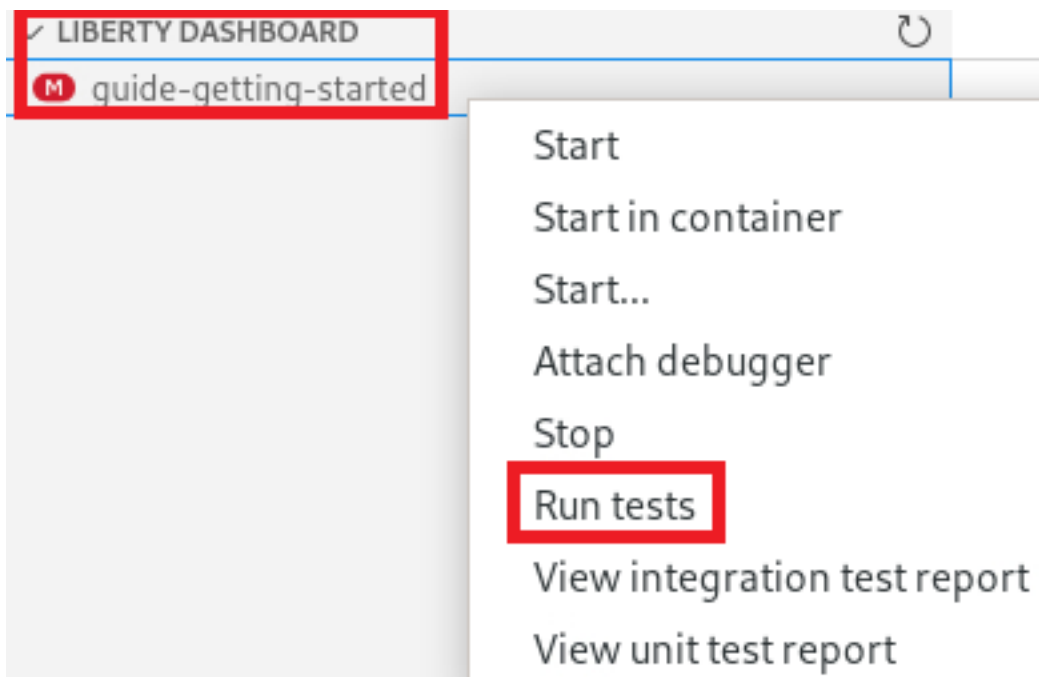
Nesta seção do laboratório, você fará algumas alterações simples no código da aplicação de exemplo e executará os casos de teste diretamente a partir da IDE do Código VS usando os recursos integrados no Liberty Tools.

Para simular uma mudança de quebra no código do aplicativo, você modificará o caminho para o terminal de serviço de **/properties** para **/all-properties**.

Como o caso de teste tente executar o serviço do sistema usando o caminho **/properties**, o caso de teste falhará e retornará um Código HTTP de 404, em vez do código de resposta esperado de 200.

Uma vez que o desenvolvedor está propositalmente introduzindo essa mudança, o caso de teste precisa ser atualizado para refletir o novo caminho para o serviço para que os testes passem.

1. Use o Liberty Dashboard para Executar os testes (**Run Tests**) para o serviço de exemplo de Propriedades do Sistema.
 - a. No Código VS, expanda a seção LIBERTY DASHBOARD
 - b. Clique com o botão direito do mouse sobre o servidor liberty **guide-getting-started**
 - d. Selecione **Run Tests** a partir do menu para executar os testes



- d. No terminal no VS Code, você verá os resultados dos testes. Um teste foi executado, e um teste com status PASSED.

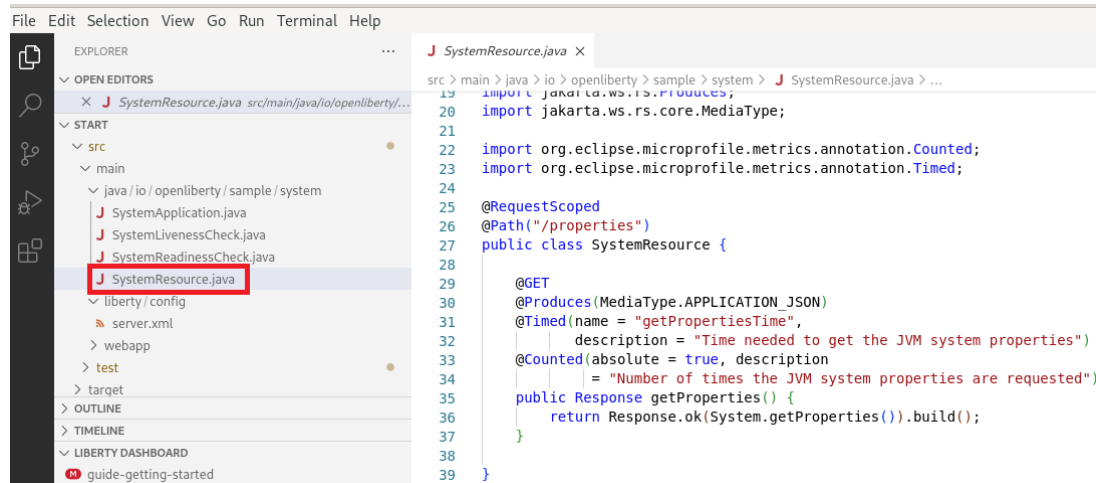
```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS guide-getting-started (liberty dev) + - [] ... ^
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running it.io.openliberty.sample.PropertiesEndpointIT
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.205 s -- in it.io.openliberty.sample.PropertiesEndpointIT
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] Rendering content with org.apache.maven.skis:maven-default-skin:jar:1.3 skin.
[INFO] Integration tests finished.
[INFO] To run tests on demand, press Enter.
```

Em seguida, como desenvolvedor no projeto, você foi solicitado a alterar o código para especificar um caminho diferente para o serviço de "**properties**". Fazendo isso, tem um impacto nos testes. Nas próximas etapas, você fará a alteração de código, e atualizará os testes para combinar com os novos resultados esperados.

2. Abra o editor **SystemResource.java** no VS Code

a. Na visualização do VS Code Explorer, expanda **START-> src-> main-> java / io / openliberty / sample / system**

b. Clique em **SystemResource.java** para abri-lo no editor



3. Atualize o `@Path` para o serviço de propriedades do sistema para especificar um caminho de serviço diferente

a. A partir do editor, faça a seguinte alteração no arquivo `systemResource.java`:

Altere a linha destacada:

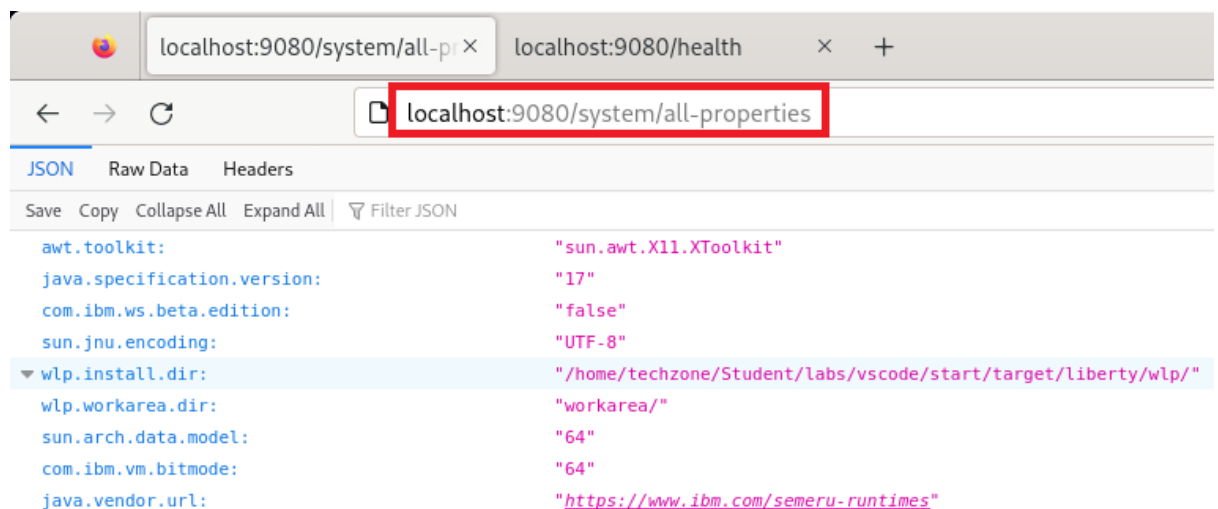
```
25 @RequestScoped
26 @Path("/properties")
27 public class SystemResource {
28
29     @GET
30     @Produces(MediaType.APPLICATION_JSON)
31     @Timed(name = "getPropertiesTime",
32           description = "Time needed to get the JVM system properties")
33     @Counted(absolute = true, description
34           = "Number of times the JVM system properties are requested")
35     public Response getProperties() {
36         return Response.ok(System.getProperties()).build();
37     }
38
39 }
```

Atualizado para ler: @Path("/all-properties")

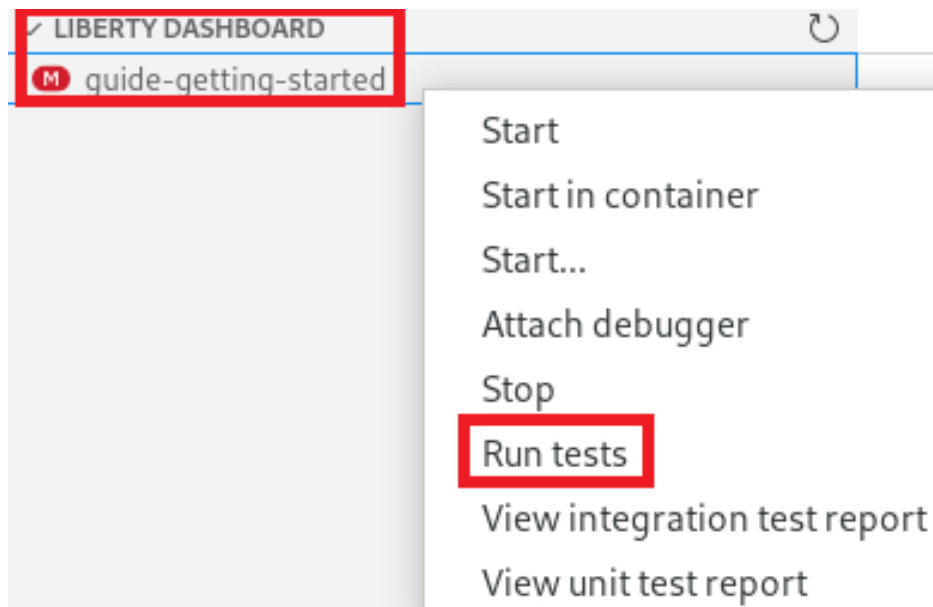
```
25 @RequestScoped
26 @Path("/all-properties")
27 public class SystemResource {
28 |
```

- b. SALVE o arquivo. O servidor Liberty e o aplicativo são atualizados dinamicamente.
 - c. Feche a visualização do editor para o arquivo SystemResource.java
4. A partir do navegador da Web, execute o serviço usando a URL do novo terminal

<http://localhost:9080/system/all-properties>



5. Use o Liberty Dashboard para Executar Testes (Run Tests) para o serviço de exemplo de Propriedades do Sistema.
- a. No Código VS, expanda a seção LIBERTY DASHBOARD
 - b. Clique com o botão direito do mouse sobre o servidor liberty **guide-getting-started**
 - c. Selecione **Run Test** a partir do menu para iniciar o servidor



d. Alternativamente, é possível executar os testes simplesmente pressionando a tecla ENTER na janela do Terminal. Dê uma tentativa. Os testes agora FALHAM.

A screenshot of the VS Code Terminal window. The 'TERMINAL' tab is selected and highlighted with a red box. The output shows the following: [INFO] Results: [INFO] [INFO] [ERROR] Failures: [ERROR] PropertiesEndpointIT.testGetProperties:42 Incorrect response code from http://localhost:9080/ ==> expected: <200> but was: <404> [INFO] [ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0 [INFO] [INFO] Rendering content with org.apache.maven.skins:maven-default-skin:jar:1.3 skin. [ERROR] Integration tests failed: There are test failures. Please refer to /home/techzone/Student/labs/vscode/start/target/failsafe-reports for the individual test results. Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream. [INFO] To run tests on demand, press Enter.

Parabéns! Você usou com sucesso a extensão Liberty Tools para o Código VS iniciar o Open Liberty em modo de desenvolvimento, fazer alterações em sua aplicação e na configuração do servidor Liberty enquanto o servidor estiver em alta, executar testes e visualizar resultados sem deixar o editor.

Como você explorou a rápida e eficiente experiência de desenvolvimento de loop interno usando o Liberty Tools e VS Code IDE, seu código foi compilado e implementado automaticamente para o seu servidor em execução, facilitando a iteração em suas alterações.