Desafios Técnicos Rocket II
Rocket Talents
Trilha Liferay

Instruções gerais:

- O desafio será aberto ao final da call do dia 20/08/2024 e a entrega será no dia 16/09/2024, até 23h59. Durante esse período, dúvidas pontuais podem ser tiradas via Chat da Google no grupo com os padrinhos.
- O desafio consiste em 1) realizar os exercícios, registrar no Github e enviar o link para avaliação; 2) Apresentar o que foi feito para o Comitê de Padrinhos (Data a Marcar).
- A apresentação deve abranger todos os tópicos requeridos.
- Importante: O cumprimento do prazo (16/09/2024) faz parte da avaliação e eventuais intercorrências devem ser avisadas com antecedência.

Instruções do Desafio:

Envie o link do git contendo os tópicos solicitados e apresente em reunião tópicos mencionados abaixo, demonstrando o que foi feito, com duração máxima de 15 minutos. Certifique-se de demonstrar e explicar os conceitos de forma clara e concisa.

A avaliação será baseada na qualidade dos códigos, na clareza da explicação, na precisão das informações e na demonstração prática dos conceitos. Certifique-se de ter internet, câmera e luz adequadas na hora da apresentação.

Esta avaliação testará seu conhecimento prático do Liferay, qualidade de código e sua capacidade de comunicar eficazmente os conceitos aos outros. Boa sorte!

Desafio: Intranet

Objetivo: Criar um dashboard personalizado que mostra informações relevantes para os usuários.

Front-end:

1 - Dashboard Personalizado

Objetivo: Criar um dashboard personalizável com tipos diferentes de layouts onde os usuários possam inserir componentes desejados através de uma interface de "Drag and Drop".

Detalhes:

1. **Tipos de Layout**:
    - **4 tipos de Grid Layout**: Uma grade simples onde os componentes podem ser arranjados em linhas e colunas.
2. **Drag and Drop**:
    - Deixar que os usuários possam arrastar, soltar, adicionar, mover e remover componentes.
3. **Componentes adaptáveis**:
    - Implementar estilos coerentes aos componentes para se adequar ao layout escolhido.
    - implementar a opção de collapse para os componentes se o usuário desejar.
4. **Estilização**:
    - Utilizar CSS, SASS, Bootstrap 4 para estilizar o componente de forma atrativa e consistente.
    - Implementar temas (claro/escuro) para melhorar a experiência do usuário.

2 - Componente Tarefas Pendentes

Objetivo: Criar uma seção de tarefas pendentes que se integre com o backend para exibir e gerenciar as tarefas do usuário.

Detalhes:

1. **Visual**:
    - Uma lista de tarefas com diferentes estados (pendente, em progresso, concluída).
    - Filtros para visualizar tarefas por status, data de vencimento, prioridade, etc.
    - Opção de marcar tarefas como concluídas ou editar detalhes da tarefa.
    - Design responsivo e intuitivo, utilizando frameworks como Bootstrap ou Material UI.
2. **Integração com o Backend**:
    - Utilizar a API desenvolvida pelo backend para obter, criar, atualizar e deletar tarefas.
    - Assegurar que as tarefas exibidas pertençam ao usuário logado.
    - Manter o estado das tarefas sincronizado entre o frontend e o backend.
3. **Estilização**:
    - Utilizar CSS, SASS, Bootstrap 4 para estilizar o componente de forma atrativa e consistente com o restante do dashboard.

3 - Componente Dados de cotação (Utilizando os Dados da API de Cotação)

Objetivo: Criar um componente estilizado que exiba dados obtidos de uma API de cotação que será integrada pelos backends (ex.: cotação de moedas, ações, etc.).

Detalhes:

1. **Obtenção de Dados**:
    - Fazer requisições a uma API de cotação (como a API de câmbio da Open Exchange Rates ou uma API de mercado financeiro) para obter os dados em tempo real.
    - Implementar tratamento de erros para lidar com falhas nas requisições.
2. **Visualização dos Dados**:
    - Exibir os dados de maneira clara e informativa, utilizando gráficos, tabelas ou outros componentes visuais.
    - Implementar diferentes modos de visualização (ex.: gráfico de linha para mostrar variação ao longo do tempo, tabela para comparações entre diferentes cotações).
3. **Estilização**:
    - Utilizar CSS, SASS, Bootstrap 4 para estilizar o componente de forma atrativa e consistente com o restante do dashboard.

Back-end:

1 - Integrar o Liferay com uma API de cotação do Dólar

Passo 1: Escolher a API de Cotação de Dólar

Primeiramente, você precisa selecionar uma API para obter a cotação do dólar. Algumas opções populares são:

- Open Exchange Rates
- CurrencyLayer
- Alpha Vantage

Essas APIs fornecem dados em JSON, que é fácil de integrar no Liferay.

Passo 2: Obter as Credenciais da API

Crie uma conta na API escolhida e obtenha a **chave de API** (API key). Você precisará dela para autenticar suas solicitações.

Passo 3: Criar um Módulo no Liferay

Agora você vai criar um módulo no Liferay para fazer as chamadas à API.

1. **Crie um Módulo no Liferay:**
    - No terminal, vá até o diretório do seu workspace do Liferay.

Execute o comando para criar um novo módulo:
basho
```
blade create -t mvc-portlet -p com.example.currencyconverter -c
CurrencyConverterPortlet currency-converter
```

    - Isso criará um portlet básico.

Configurar Dependências do Gradle: Adicione as bibliotecas necessárias ao `build.gradle` para permitir o envio de requisições HTTP. Uma biblioteca comum é o `Apache HttpComponents` ou o `OkHttp`.
Adicione a seguinte dependência ao arquivo `build.gradle`:

```
dependencies {

    compileOnly group: "org.apache.httpcomponents", name:
"httpclient", version: "4.5.13"

}
```

Passo 4: Fazer a Chamada à API

Criar uma classe `CurrencyConverterPortlet.java`, desenvolva o código para realizar a chamada à API de cotação de dólar.

1. **Chamada de API**: Você pode usar a biblioteca `HttpClient` ou outra de sua escolha para fazer uma requisição GET à API.

2 . **Tratar a Resposta JSON:**

    - Após obter a resposta da API, você pode analisar o JSON e extrair a cotação do dólar.

Passo 5: Exibir a Cotação no JSP

Agora, no arquivo `view.jsp`, exiba a cotação do dólar:

Resumo

- **Escolha uma API** de cotação de dólar.

- **Obtenha as credenciais** da API.
- **Crie um módulo no Liferay** para integrar a API.
- **Faça a chamada à API** no código Java.
- **Exiba os dados** da cotação no JSP.
- **Empacote e implante** o portlet no Liferay.

2 - Desenvolver um CRUD de tarefas pendentes utilizando o Service Builder no Liferay.

1. **Criar o Módulo de Serviço:**

No terminal, dentro do workspace, crie um módulo de serviço usando o Blade CLI:
bash

```
blade create -t service-builder -p com.example.task -c Task
task-service
```

- Isso criará o módulo `task-service` com o pacote `com.example.task`.

Passo 2: Definir o Modelo no Service.xml

1. **Modificar o `service.xml`:**
   - Navegue até o arquivo `service.xml` no módulo `task-service` (`task-service/src/main/resources/META-INF/service.xml`).
   - Defina o modelo da entidade `Task` (tarefa) da seguinte maneira:

EXEMPLO XML:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder
7.4.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_7_4_0.dtd">

<service-builder package-path="com.example.task">
```

```
    <namespace>Task</namespace>

    <entity name="Task" local-service="true" remote-service="false">

        <column name="taskId" type="long" primary="true" />

        <column name="title" type="String" />

        <column name="description" type="String" />

        <column name="dueDate" type="Date" />

        <column name="completed" type="boolean" />

        <order by="dueDate ASC" />

    </entity>

</service-builder>
```

- As colunas representam os atributos da entidade **Task**: `taskId`, `title`, `description`, `dueDate`, e `completed`.

Passo 3: Gerar o Código com o Service Builder

1. **Executar o Service Builder:**

No terminal, execute o comando para gerar o código do Service Builder:
bash
```
./gradlew buildService
```

- Isso gerará as classes necessárias para a persistência no banco de dados e serviços de CRUD.

Passo 4: Implementar o Portlet MVC para o CRUD

Agora que o modelo está definido e o Service Builder gerou o código, é hora de criar o portlet para gerenciar as tarefas pendentes.

1. **Criar um Portlet MVC:**

Crie um portlet no workspace usando o Blade CLI:
bash
```
blade create -t mvc-portlet -p com.example.task -c TaskPortlet
task-web
```

2. **Configurar Dependências no `build.gradle`:**

No módulo `task-web`, adicione dependências para o módulo de serviço:
gradle

```
dependencies {

    compileOnly project(":modules:task-service")

}
```

3. **Atualizar o `TaskPortlet.java`:**
    - No arquivo `TaskPortlet.java`, implemente a lógica de exibir, criar, editar e deletar tarefas.

Passo 5: Criar a Interface JSP

1. **Criar o `view.jsp`:**
    - No arquivo `view.jsp`, exiba as tarefas e inclua um formulário para criar novas tarefas:

Passo 6: Empacotar e Implantar

1. **Empacotar o Serviço e o Portlet:**

Execute o seguinte comando para empacotar e implantar ambos os módulos:
bash

```
./gradlew build
```

Resumo

- Defina a entidade `Task` com o **Service Builder**.
- Gere o código de serviços com o Service Builder.
- Implemente um portlet para gerenciar as tarefas pendentes.
- Desenvolva a interface JSP para permitir a criação e listagem de tarefas.
- Implante os módulos e teste o CRUD na interface do Liferay.