FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Googol – Meta 2

Projeto de Sistemas Distribuídos

Projeto desenvolvido por:

André Moreira, nº2020239416

João Pinto, nº2020220907

Índice

Arquitetura do Sistema

1 - Modelo MVC

O modelo MVC (Model-View-Controller) é uma arquitetura de software amplamente utilizada no desenvolvimento de aplicativos web.

Model:

Representa os dados e a lógica de negócios da aplicação. No código fornecido, a classe webpage atua como o model, pois contém a lógica relacionada à interação com o Search Module e à manipulação dos resultados de pesquisa.

View:

A view é responsável pela apresentação dos dados ao usuário e pela interação com ele. Ela exibe as informações fornecidas pelo model e captura as ações do usuário. No código, os arquivos HTML no diretório de templates (como home.html, login.html, etc.) representam as views. Esses arquivos definem a estrutura e o conteúdo das páginas exibidas no navegador.

Controller:

O controller atua como intermediário entre o model e as views. Ele recebe as solicitações do usuário, interage com o model para obter os dados necessários e atualiza as views correspondente para exibir os resultados. No código, a classe webpage anotada com @Controller representa o controller. Ela define os mapeamentos de URL para as diferentes páginas, processa as solicitações recebidas e atualiza o model ou a view conforme necessário.

2 – Processos e Threads

Nesta meta não foram criados processos e Threads, apenas mantemos o que tínhamos da meta 1. Contudo, é necessário ter em conta que para o WebServer ficar operacional é preciso iniciar o *SpringBootApplication* de modo a ter toda a interface web disponível.

3 – Websockets

O *WebSocket* é utilizado para comunicação bidirecional em tempo real entre clientes e servidores por meio de conexões persistentes e constantes.

O *WebSocketClient* é o componente responsável por enviar mensagens para o *WebSocket,* por parte da Webpage. Ele utiliza a classe *SimpMessagingTemplate* para enviar mensagens para o destinatário, que por sua vez é o local onde os clientes estão subscritos. O *SimpMessagingTemplate* é injetado no construtor do *WebSocketClient* por meio da anotação *Autowired*. Através do método *sendMessage*(*String destination*, *ProgramStatus status*), a webpage comas informações provenientes do Search Module via RMI, pode enviar uma mensagem para um destino específico. Program Status é um *record* para poder enviar toda a informação do sistema de forma simples.

A classe WebSocketServer é uma configuração responsável por configurar o servidor WebSocket. Ela é anotada com *@Configuration* para indicar que é uma classe de configuração do Spring. Além disso, a anotação *@EnableWebSocketMessageBroker* é utilizada para habilitar a funcionalidade de mensagens baseada em WebSocket no servidor.

A implementação do WebSocket na página da web é realizada por meio do JavaScript. A função connect() é chamada quando o usuário clica no botão "*Connect*". Ela cria uma instância do objeto SockJS passando o endpoint do WebSocket ("/my-websocket") e, em seguida, utiliza o objeto Stomp para se conectar ao WebSocket. Após a conexão ser estabelecida com sucesso, a função setConnected(true) é chamada para habilitar os elementos da interface do usuário relacionados à conexão e assina o tópico "/topic/messages" para receber mensagens. A função *disconnect()* é chamada quando o usuário clica no botão "*Disconnect*". Ela desconecta o cliente do WebSocket e desabilita os elementos da interface do usuário relacionados à conexão. A função *showMessage(barrels, downloaders, searches)* é responsável por exibir as mensagens recebidas na tabela da página da web.

Integração do SpringBoot com o RMI

Para dar início à meta 2, foi criado um projeto do zero com recurso ao Website *Spring Initializr,* que nos devolvia, conforme as nossas configurações, uma aplicação de SpringBoot para ter acesso à interface web e todas as outras funcionalidades de forma mais interativa e intuitiva. Após este procedimento, migrámos todos os outros ficheiros da meta 1 para o novo projeto para ter fácil acesso a todos os componentes.

Como já acontecia na meta 1, o *search module* atua como um servidor RMI, mas neste caso, os clientes RMI não são os utilizadores, mas sim a aplicação Web. Foram adaptados os métodos da Classe *Interface* para este novo modelo para poder receber e processar toda a informação proveniente do *search module*. A classe *webpage* implementa a interface da meta 1 *Hello_C_I* para ter acesso aos métodos remotos. No construtor da classe, é feita a ligação RMI e a posterior subscrição. Sempre que a webpage recebe informação do *search module,* esta é processada de forma ligeiramente diferente para satisfazer as necessidades de uma página web, mas essencialmente tem o mesmo funcionamento da anterior interface de cliente.

Tal como acontecia na Meta 1, o Search Module recebe agora as mensagens provenientes do WebServer e processa toda essa informação de igual forma. Cada funcionalidade depois tem um resultado específico e diferente da meta 1 devido a agora termos acesso a uma interface web. Foi facilitada a visualização dos resultados e melhorada a interação entre clientes e programa.

É necessário colocar o endereço IP da máquina onde está a correr o Search Module para o WebServer ter conhecimento do endereço onde está o servidor RMI.

Integração com serviço REST

Nesta meta, foi pedida a adição de 2 funcionalidades relacionadas com o Hacker News. Para as desenvolver tivemos de utilizar a API REST e toda a documentação disponibilizada pelo Hacker News.

Para a primeira tarefa de indexar os URLs das top stories que contenham os termos da pesquisa, simplesmente adicionámos um checkbox no html para verificar o desejo de executar esta tarefa ou não, por parte do utilizador. Se selecionar, envia a mensagem para o Search Module por RMI tal como das outras vezes, e dentro do Search Module é feita a procura das top stories do momento com recurso a esta ligação ***https://hacker-news.firebaseio.com/v0/topstories.json.*** Obtendo a lista de top stories, verifica-se se o título da publicação contem os termos da pesquisa, e se sim, indexa-se o url correspondente.

Na segunda tarefa de indexar os stories de um utilizador, criou-se uma página HTML para receber o input de um nome de utilizador do Hacker News. Envia-se esse username para o Search Module e é processado para receber as suas informações via ***"https://hacker-news.firebaseio.com/v0/user/"+utilizador+".json".*** Obtendo a lista de stories apenas será necessário indexar os URLs correspondentes.

Testes realizados

1. Login com user não registado: passed, mostra mensagem de erro a dizer que username ou password estão errados.

2. Register com carateres inválidos: passed, diz que não pode conter carateres inválidos.

3. Register com username já utilizado: passed, diz que username já se encontra utilizado.

4. Register com username e password válidos: passed, envia para a homepage e diz que registo foi realizado com sucesso.

5. Login a user válido: passed, envia para a homepage e diz que login foi realizado com sucesso.

6. Logout sem login realizado: passed, diz que necessita ter login realizado.

7. Logout com login realizado: passed, diz que logout foi realizado com sucesso.

8. Indexar url invalido: passed, downloader diz que houve erro na procura.

9. Indexar url válido: passed, url é indexado com sucesso.

10. Fornecer termos em branco: passed, diz que é obrigatório fornecer o termo.

11. Fornecer carateres inválidos na pesquisa de termos: passed, diz que não pode conter carateres inválidos.

12. Fornecer como número de parâmetros um valor inválido: passed, não permite fornecer parâmetros, só aparecendo um botão para voltar para a página inicial.

13. Realizar pesquisa por URL sem login realizado: passed, diz que necessita ter login realizado.

14. Indexar User stories com user correto: passed, stories do user são indexadas com sucesso.

15. Indexar User stories com user que não existe: passed, search engine diz que não foi encontrado o user.

16. Fornecer termos corretos: passed, envia para uma página que mostra os resultados da pesquisa.

17. Caixa de Indexar e premida na pesquisa de termos: passed, as pesquisas são indexadas.