Projeto Final

Curso Elixir 2023 — IFRS — Campus Canoas

O aplicativo **Tarefas** modela uma lista de tarefas que podemos utilizar para registrar nossas tarefas pendentes, prioriza-las, lista-las, e marca-las como completadas. Também temos funcionalidades de apoio, como remover tarefas ou marcar tarefas como sem completar. Durante o curso, usamos Elixir para começar a desenvolver este aplicativo com interface de linha de comandos (CLI). O código visto em aula está disponível aqui.

Agora vamos extender o aplicativo criado para termos uma interface web, a través do protocolo HTTP. Para isso, usaremos o *framework* **Phoenix**, e faremos algumas adaptações na estrutura do nosso código.

Preliminar

No novo aplicativo **Tarefas**, as tarefas são representadas como structs do tipo Tarefa, que contém id, descrição e estado. O id é um valor que representa unívocamente uma tarefa; a descrição é um texto que descreve a tarefa; e o estado (completar ou sem_completar) indica se a tarefa foi completada ou não.
Leia sobre mapas e structs

O código relevante ao nosso aplicativo está organizado em cinco módulos: Tarefa, Tarefas, Tarefa.CLI e ControladorTarefas. Pede-se implementar as funções destes módulos para que o aplicativo funcione corretamente na interface de linha de comandos e na interface web.

Para construir o novo aplicativo, criamos um projeto Phoenix com o comando **mix phx.new tarefas**. Isto já esta feito. A base para a implementação deste projeto pode ser encontrada aqui. Será necessário editar unicamente os seguintes arquivos:

— backend/lib/tarefas/tarefa.ex
— backend/lib/tarefas.ex
— backend/lib/tarefas/cli.ex
— backend/lib/tarefas_web/controllers/controlador_tarefas.ex
— backend/lib/tarefas_web/router.ex

Execução

Os seguintes comando serão úteis para o desenvolvimento do projeto.
Na pasta **backend**:
— **mix test** executa os testes automatizados. Leia mais aqui.
— **mix escript.build** compila a interface de linha de comandos
— **mix phx.server** executa o servidor web Phoenix

Na pasta **backend**, depois de executar **mix escript.build**
— **./tarefas** executa a interface de linha de comandos em Linux/Mac
— **escript tarefas** executa a interface de linha de comandos em Windows

Na pasta **frontend**:
— **npm run server** executa o frontend

Implementando o módulo Tarefa

O módulo Tarefa possui a definição do struct Tarefa e funções para a manipulação de elementos deste tipo. **Funções neste módulo operam sobre uma única tarefa**. Não é necessário incluir aqui funções para operar sobre listas de tarefas. Pede-se implementar as seguintes funções no arquivo **backend/lib/tarefas/tarefa.ex**:

— **completar/1** (marca uma Tarefa como completada)
— **reiniciar/1** (marca uma Tarefa como sem completar)
— **completada?/1** (informa se uma Tarefa foi completada ou não)
— **codificar/1** (converte uma Tarefa em string)
— **decodificar/1** (converte uma string em Tarefa)
— **imprimir/1** (imprime uma Tarefa no console)

Leia atentamente a especificação destas funções (presente no código) para entender o que cada uma deve fazer. **Lembre-se de utilizar pattern matching na definição das funções.** Execute os testes deste módulo com o comando:

```
mix test backend/test/tarefas/tarefa_test.exs
```

Implementando o módulo Tarefas

O módulo Tarefas possui funções para a manipulação de listas de tarefas. **Funções neste módulo operam sobre listas de tarefas**. Não é necessário incluir aqui funções para operar sobre uma única tarefa. Pede-se implementar as seguintes funções no arquivo **backend/lib/tarefas.ex**:

— **decodificar/1** (converte uma string em lista de tarefas)
— **codificar/1** (converte uma lista de tarefas em string)
— **filtrar/2** (filtra uma lista de tarefas segundo o estado)
— **ordenar/1** (ordena uma lista de tarefas segundo o estado)
— **imprimir/1** (imprime uma lista de tarefas no console)
— **inserir/1** (insere uma Tarefa na lista de tarefas)
— **inserir/2** (insere uma Tarefa numa posição específica da lista de tarefas)
— **mover/3** (move uma Tarefa de uma posição a outra na lista de tarefas)
— **remover/2** (remove uma Tarefa da lista de tarefas)
— **completar/2** (marca uma Tarefa da lista de tarefas como completada)
— **reiniciar/2** (marca uma Tarefa da lista de tarefas como sem completar)

Lembre-se de:
— utilizar as funções já implementadas no módulo Tarefa
— utilizar **pattern matching na definição das funções**
— considerar que os **dados são imutáveis**
— utilizar guardas **with** para definir os tipos das variáveis. Leia mais aqui**.**
— utilizar as funções dos módulos Enum e List

Leia atentamente a especificação destas funções (presente no código) para entender o que cada uma deve fazer. Execute os testes deste módulo com o comando:

mix test backend/test/tarefas_test.exs

Implementando o módulo Tarefas.CLI

O módulo Tarefas.CLI possui o ponto de entrada para a interface de linha de comandos (a função **main**) e funções para o processamento dos comandos.

Pede-se implementar a função **processar/2**, no arquivo **backend/lib/tarefas/cli.ex** que atenderá aos seguintes comandos:

— **tarefas**
Lista as tarefas que ainda não foram completadas

— **tarefas todas**
Lista todas as tarefas
As completadas aparecem no final com o subtítulo —Tarefas Completadas—

— **tarefas** <descrição>
Cadastra uma tarefa nova na lista de tarefas, com a descrição fornecida
A tarefa é cadastrada na última posição
Ao inserir uma nova tarefa, é **imprescindível** definir o id com a função **@uuid.()**

— **tarefas** <descrição> <posição>
Cadastra uma tarefa nova na lista de tarefas, com a descrição fornecida
A tarefa é cadastrada na posição indicada
Ao inserir uma nova tarefa, é **imprescindível** definir o id com a função **@uuid.()**

— **tarefas mover** <origem> <destino>
Muda a posição da tarefa da posição origem na lista de tarefas para a posição destino

— **tarefas remover** <posição>
Remove a tarefa na posição indicada da lista de tarefas

— **tarefas completar** <posição>
Marca a tarefa na posição indicada como completada

— **tarefas reiniciar** <posição>
Marca a tarefa na posição indicada como sem completar

Ao implementar o módulo Tarefas.CLI, lembre-se de:

— utilizar as funções já implementadas nos módulos Tarefa e Tarefas

— utilizar **pattern matching na definição das funções**

— utilizar guardas **with** para definir os tipos das variáveis. Leia mais aqui**.**

Execute os testes deste módulo com o comando:

mix test backend/test/tarefas/cli_test.exs

Uma vez que os módulos Tarefa, Tarefas e Tarefas.CLI estiverem implementados corretamente, será possível compilar a interface de linha de comandos com o comando **mix escript.build** e executar o executável **tarefas** segundo as definições acima.

Implementando o módulo ControladorTarefas

O módulo ControladorTarefas possui funções para o processamento de comandos HTTP recebidos da web. Phoenix se responsabiliza pela recepção e pelo processamento destes comandos e por chamar as funções adequadas do módulo ControladorTarefas. Leia a secão **Utilização básica de Phoenix** neste documento.

Pede-se adicionar as definições no módulo Router, arquivo **backend/lib/tarefas_web/router.ex**, e implementar as funcões no módulo ControladorTarefas, arquivo **backend/lib/tarefas_web/controllers/controlador_tarefas.ex**, para atender aos seguintes requisitos:

— **GET /api/tarefas**
  Retorna todas as tarefas
  Utilizar a função **listar** do módulo ControladorTarefas

— **POST /api/tarefas {“descricao”: descricao}**
  Cadastra uma tarefa nova na lista de tarefas, com a descrição fornecida
  A tarefa é cadastrada na última posição
  Ao inserir uma nova tarefa, é **imprescindível** definir o id com a função **@uuid.()**
  Utilizar a função **inserir** do módulo ControladorTarefas

— **POST /api/tarefas/:origem/mover-a/:destino**
  Muda a posição da tarefa da posição origem na lista de tarefas para a posição destino
  Utilizar a função **mover** do módulo ControladorTarefas

— **POST /api/tarefas/completadas/:posicao**
  Marca a tarefa na posição indicada como completada
  Utilizar a função **completar** do módulo ControladorTarefas

— **DELETE /api/tarefas/completadas/:posicao**
  Marca a tarefa na posição indicada como sem completar
  Utilizar a função **reiniciar** do módulo ControladorTarefas

— **DELETE /api/tarefas/:posicao**
  Remove a tarefa na posição indicada da lista de tarefas
  Utilizar a função **remover** do módulo ControladorTarefas

Ao implementar o módulo ControladorTarefas, lembre-se de:

— utilizar as funções já implementadas nos módulos Tarefa e Tarefas

— utilizar **pattern matching na definição das funções**

— sempre retornar uma resposta

Execute os testes deste módulo com o comando:

mix test backend/test/tarefas/controllers/controlador_tarefas_test.exs

Uma vez que os módulos Tarefa, Tarefas e ControladorTarefas estiverem implementados corretamente, será possível interagir com a lista de tarefas mediante o *website* implementado na pasta frontend. Para isto, acesse a pasta **frontend**, execute o comando **npm run start** e abra a URL http://localhost:3000 no seu navegador.

Entrega do projeto

Para entregar o projeto, você deverá:

— criar uma conta no GitHub

— fazer um fork do repositório https://github.com/andres-vidal/elixir-course-pt

— atualizar os arquivos do repositorio com suas implementações:

  — backend/lib/tarefas/tarefa.ex

  — backend/lib/tarefas.ex

  — backend/lib/tarefas/cli.ex

  — backend/lib/tarefas_web/controllers/controlador_tarefas.ex

  — backend/lib/tarefas_web/router.ex

— criar um Pull Request desde o seu repositorio ao original

Requisitos da entrega:

— o prazo tentativo é de duas semanas e meia (13 de março)

— os testes devem executar corretamente, confira com **mix test**

— a implementação deve ser correta

— a interface de linha de comandos deverá funcionar corretamente

— a interface web deverá funcionar corretamente

— poderá haver uma reentrega para incluir correções

Utilização básica de Phoenix

Poderíamos dizer que Phoenix se encarrega de **decodificar** as mensagens HTTP, chamar a função **processar** com os parâmetros adequados e **codificar** a resposta da mensagem. O módulo ControladorTarefas é análogo à função Tarefas.CLI.processar/2 e o módulo Router é onde se define qual função de ControladorTarefas deverá ser chamada como resposta a cada mensagem HTTP.
Leia sobre controladores e roteamento.

Definindo uma rota e uma função de processamento em Phoenix

Por exemplo, para processar o comando **GET /api/tarefas** é necessário escrever a linha **get "/tarefas", ControladorTarefas, :listar** dentro do bloco de código definido pela instrução **scope "/api", TarefasWeb do … end**. E definir a função **listar/2** no módulo ControladorTarefas, que recebe uma conexão **conn** e um mapa de parâmetros. Observe que o nome **listar** é arbitrário: pode ser escolhido qualquer nome para a função, sempre e quando a definição no módulo Router coincida com a definição no módulo ControladorTarefas.

Acessando os parâmetros de uma mensagem HTTP em Phoenix

Há varias maneiras de receber parâmetros numa mensagem HTTP. Neste projeto usaremos duas: no corpo da mensagem e na URL da mensagem.

Parâmetros no corpo da mensagem HTTP são inseridos pelo cliente que emite a mensagem, e podemos acessar eles em Phoenix desde o ControladorTarefas mediante o objeto de parâmetros da função de processamento.

Por exemplo, a mensagem **POST /api/tarefas {"descricao": "Fazer os exericicios"}** será processada por Phoenix se **post "/tarefas", ControladorTarefas, :inserir** estiver definido no *scope /api* do módulo Router e se definirmos uma função **inserir/2** no módulo ControladorTarefas. A descrição da nova tarefa poderá ser acessada pelo segundo parâmetro da função inserir:

```
def inserir(conn, params) do
  %{“descricao” => descricao} = params
  …
end
```

ou equivalentemente:

```
def inserir(conn, %{“descricao” => descricao}) do
  …
end
```

Por outro lado, podemos receber parâmetros na URL da mensagem HTTP, adicionando dois pontos (:) antes da porcão da URL que queremos fazer variável. Este tipo de URL se chamam URLs dinâmicas.

Por exemplo, a mensagem **DELETE /api/tarefas/:posicao** será processada por Phoenix se **delete “/tarefas/:posicao”, ControladorTarefas, :remover** estiver definido no *scope /api* do módulo Router e se definirmos uma função **remover/2** no módulo ControladorTarefas. A variável posicao poderá então ser acessada do no objeto de parâmetros, da mesma forma que no exemplo anterior.

Retornando uma resposta em Phoenix

Uma resposta HTTP pode ter varias formas. Neste projeto utilizaremos respostas no formato JSON e sem controle de fluxo, pelo qual sempre retornaremos o estado 200 OK. Para retornar uma resposta OK em Phoenix, retornamos o valor da função **json/2** em cada função de processamento do módulo ControladorTarefas. Esta função recebe uma conexão no primeiro parâmetro e os dados a ser retornados no segundo parâmetro.

Por exemplo, para retornar uma resposta vazia chamamos **json(conn, “”)** na última linha da função de processamento. Para retornar dados armazenados numa variável **resposta**, chamamos **json(conn, resposta)** na última linha da função de processamento.