INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO SUL
Campus Bento Gonçalves

Curso Superior de Análise e Desenvolvimento de Sistemas
Prática Profissional Integrada I
Prof. Dr. Maurício Covolan Rosito
PROVA P1

INSTRUÇÕES GERAIS

- Esta é uma prova prática, individual e com consulta.
- O valor total desta prova corresponde à 50% da nota da disciplina.
- O prazo de entrega da prova é 07/03/2021 às 23:55h.
  - As questões devem estar em um arquivo compactado (.zip).
  - O arquivo compactado da prova deve ser submetido pelo Moodle (não serão aceitos trabalhos enviados por e-mail).
  - Envios submetidos após esta data não serão aceitos.
- Os critérios observados e exigidos nesta prova são os seguintes:
  - funcionalidade de acordo com o solicitado
  - compreensão dos conceitos abordados
  - pontualidade
  - clareza na demonstração da solução

CENÁRIO: Vamos simular que nossa empresa foi contratada para desenvolver um sistema em React para fazer a gestão da lista de emails de contato dos alunos do IFRS-Campus Bento Gonçalves. Vamos utilizar os conhecimentos relacionados a React Hooks, useReducer, Context API e React Router.

QUESTÃO 1: [5,0 PONTOS] Nossa aplicação será composta de alguns componentes de apresentação e um contêiner. Também haverá um objeto de contexto para gerenciar o estado dos contatos. Como nossa árvore de estados será um pouco mais complexa, teremos que usar o hook useReducer.

Desta forma, crie um projeto chamado provap1-contatos. Em seguida, crie o objeto de contexto de estado no arquivo ContactContextProvider.js com o seguinte código:

```
import React, { useReducer, createContext } from "react";

export const ContactContext = createContext();

const initialState = {
 contacts: [
   {
     id: 98,
     name: "Diana Prince",
     email: "diana@us.army.mil"
   },
   {
     id: 99,
     name: "Bruce Wayne",
     email: "bruce@batmail.com"
   },
   {
     id: 100,
     name: "Clark Kent",
     email: "clark@metropolitan.com"
```

```
    }
  ]
};

const reducer = (state, action) => {
  switch (action.type) {
    case "ADD_CONTACT":
      return {
        contacts: [...state.contacts, action.payload]
      };
    case "DEL_CONTACT":
      return {
        contacts: state.contacts.filter(
          contact => contact.id !== action.payload
        )
      };
    default:
      throw new Error();
  }
};

export const ContactContextProvider = props => {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <ContactContext.Provider value={[state, dispatch]}>
      {props.children}
    </ContactContext.Provider>
  );
};
```

Observe que o código `export const ContactContext = createContext();` é responsável por criar um contexto novo para a nossa aplicação. Com nosso contexto inicializado, nós precisávamos fornecer o Provider dele, o `<ContactContext.Provider>`, para o nosso componente ContactContextProvider. Assim, podemos ler o `value` do nosso Provider em qualquer componente que estiver dentro do componente ContactContextProvider (fornecidos via `props.children`), independente do nível que o componente esteja.

Ainda, veja que nós criamos uma função redutora chamada `reducer`. Dentro do nosso componente ContactContextProvider, nós aplicamos o hook `useReducer`, enviando nossa função redutora (`reducer`) como primeiro argumento e um estado inicial (`initialState`) como segundo argumento. O hook, por sua vez, retorna um array com dois atributos:

- state: O estado atual do redutor;

- dispatch: Uma função auxiliar para que você possa despachar novas ações para nosso redutor.

Cabe lembrar que toda vez que nós despachamos uma ação com `dispatch`, a nossa função redutora `reducer` é chamada, e um novo estado é retornado através dela. Veja que usamos um `type` para identificar a ação e um `payload` para identificar o contato.

Ainda, cabe ressaltar que o método `filter()` do Javascript recebe como parâmetro uma função de *callback*, onde o retorno dado será um novo array com os elementos que passaram na validação realizada.

Em seguida, crie o componente container das informações dos contatos no arquivo ContactView.js:

```
import React from "react";
import ContactForm from "./ContactForm";
import ContactTable from "./ContactTable";
import { ContactContextProvider }  from "./ContactContextProvider";

function ContactView() {
 return (
   <ContactContextProvider>
     <div>
       <h3>Contatos</h3>
       <ContactForm />
       <ContactTable />
     </div>
   </ContactContextProvider>
 );
}
export default ContactView;
```

Observe que ele cria uma instância do componente ContactContextProvider, passando para este componente pai os seguintes componentes filhos: ContactForm e ContactTable. O componente ContactForm será responsável por adicionar novos contatos no estado global da aplicação. O componente ContactTable será responsável por listar e excluir os contatos do estado global.

Agora, crie o componente de apresentação das informações dos contatos em formato de tabela no arquivo ContactTable.js:

```
import React, { useContext } from "react";
import  { ContactContext }  from "./ContactContextProvider";

function ContactTable() {
 const [state, dispatch] = useContext(ContactContext);
  const delContact = id => {
```

```
  dispatch({
    type: "DEL_CONTACT",
    payload: id
  });
};

const onRemoveUser = () => {
  if(state.contacts[0]!==undefined) {
    // Exclui o primeiro registro
    const firstId = state.contacts[0].id;
    delContact(firstId);
  }
  else {
    alert("Não existem mais contatos");
  }
};

const rows = state.contacts.map(contact => (
  <tr key={contact.id}>
    <td>{contact.id}</td>
    <td>{contact.name}</td>
    <td>{contact.email}</td>
  </tr>
));

return (
  <div>
    <p>Listagem de Contatos</p>
    <table border="1">
        <thead>
          <tr>
              <th>Id</th>
              <th>Nome</th>
              <th>Email</th>
          </tr>
      </thead>
      <tbody>{rows}</tbody>
      <tfoot>
        <tr>
          <th colSpan="4">
            <button onClick={onRemoveUser}>Remover</button>
          </th>
```

```
          </tr>
        </tfoot>
      </table>
    </div>
  );
}
export default ContactTable;
```

Veja que o código `const [state, dispatch] = useContext(ContactContext);` assina o estado de `contacts` e acessa a função `dispatch`, onde é passado o `type` correspondente à ação de exclusão (`DEL_CONTACT`) de um contato e o `payload` (que será o `id` do contato a ser excluído)

Observe que através do código `state.contacts.map(...)` é possível percorrer todos os contatos que estão no estado e apresentá-los em formato de tabela. Ainda, ao clicar no botão para remover um contato, a função `onRemoveUser` exclui sempre o primeiro contato da listagem (se existir).

Agora, crie outro componente de apresentação em ContactForm.js:

```
import React, { useContext } from "react";
import _ from "lodash";
import  { ContactContext }  from "./ContactContextProvider";

function ContactForm() {
 const [state, dispatch] = useContext(ContactContext);

 const onAddContact = () => {
   dispatch({
     type: "ADD_CONTACT",
     payload: { id: _.uniqueId(10), name: "Teste",
                email: "teste@email.com" }
   });
 };

 return (
   <div>
     <p>Adicionar Novo Contato</p>
     <button onClick={onAddContact}>Novo Contato</button>
     <hr/>
   </div>
 );
}
export default ContactForm;
```

Veja que o código `const [state, dispatch] = useContext(ContactContext);` assina o estado de `contacts` e acessa a função `dispatch`, onde é passado o `type` correspondente à ação de adição (`ADD_CONTACT`) de um contrato e o `payload` (que será um objeto com dados fixos). Aqui, cabe dizer que o método `_.uniqueId` gera um ID exclusivo. Se um prefixo for fornecido, neste caso 10, o ID será anexado a ele (fica, por exemplo, 101).

Agora, para testar esta aplicação, insira o seguinte código no App.js:

```
import React from "react";
import ContactView from "./ContactView";

function App() {
 return (
   <div>
     <h1>Gestão de contatos</h1>
     <ContactView />
   </div>
 );
}
export default App;
```

Veja o resultado no navegador quando clicamos no botão para adicionar um novo contato:

QUESTÃO 2: [3,0 PONTOS] Com os conhecimentos adquiridos relacionados ao uso de formulários com Hooks, ajuste o componente ContactForm.js para que o usuário possa informar o nome e o email.

QUESTÃO 3: [2,0 PONTOS] Crie uma novo componente chamado Detalhes.js. Ao clicar em um contato listado na tabela, deve ser redirecionado para este novo componente que irá mostrar os detalhes (id, nome e email) deste contato. Use os conhecimentos de rotas com React.