ANÁLISE DE SISTEMAS: APP DE TAREFAS

- Crie um novo projeto

- Instale as dependências:
    - npx expo install @react-native-async-storage/async-storage
    - npm install @react-navigation/native
    - npx expo install react-native-screens react-native-safe-area-context
    - npm install @react-navigation/native-stack
    - npm install @react-navigation/bottom-tabs
    - npm install axios
    - npm install react-native-dropdown-picker
    - npx expo install react-native-gesture-handler
    - npx expo install react-native-reanimated
    - npm install react-native-root-toast
    - npm install uuid
    - npm install react-native-get-random-values
    - npm install --save @types/uuid

- Crie uma pasta src

- Dentro da pasta src crie as pastas components, contexts, routes, screens, types, utils.

- Na pasta utils crie um arquivo data.ts e dentro dele exporte o array de acordo com o modelo:

```
export const categories = [
  {
    id: 0,
    label: "A Fazer",
    value: "all",
    color: "#353535",
  },
  {
    id: 1,
    label: "Pessoal",
    value: "personal",
    color: "#219ebc",
  },
  {
    id: 2,
    label: "Trabalho",
    value: "work",
    color: "#ff006e",
  },
  {
    id: 3,
    label: "Estudo",
    value: "study",
    color: "#f77f00",
  },
  {
    id: 4,
    label: "Bem-estar",
    value: "life",
    color: "#ff8fab",
  },
  {
    id: 5,
    label: "Concluídas",
    value: "done",
    color: "#007200",
  },
];
```

- Na pasta contexts insira o mesmo arquivo do UserContext utilizado no app do carrinho de compras.

- Na pasta types crie um arquivo Task.ts e insira o conteúdo:

```
export interface Task {
  id: string;
  title: string;
  completed: boolean;
  category: string;
}

export interface Category {
  id: number;
  label: string;
  value: string;
  color: string;
}
```

- Na pasta types crie um arquivo User.ts e insira o conteúdo:

```ts
export interface UserDTO {
  id: number;
  username: string;
  email: string;
  firstName: string;
  lastName: string;
  gender: string;
  image: string;
  token: string;
}
```

- Na pasta screens crie os arquivos Login.tsx e User.tsx copiando os dados do app de carrinho de compras. Crie também um arquivo Home.tsx, no momento, retornando apenas um text.

- Na pasta routes crie os arquivos index.tsx, AppRoutes.tsx e AuthRoutes.tsx. Use a estrutura do app de carrinho de compras para montar as rotas. AuthRoutes terá apenas a tela de Login e AppRoutes terá uma StackRoutes com a tela Home e um Tab (chamado de AppRoutes) com a StackRoutes e a tela User.

- No arquivo App.tsx deixe-o assim:

```tsx
import "react-native-gesture-handler";
import { RootSiblingParent } from "react-native-root-siblings";
import { UserContextProvider } from "./src/contexts/UserContext";
import { Routes } from "./src/routes";
import { GestureHandlerRootView } from "react-native-gesture-handler";

export default function App() {
  return (
    <GestureHandlerRootView style={{ flex: 1 }}>
      <RootSiblingParent>
        <UserContextProvider>
          <Routes />
        </UserContextProvider>
      </RootSiblingParent>
    </GestureHandlerRootView>
  );
}
```

- Na pasta components crie um arquivo ItemCard.tsx. Nele vamos utilizar o componente swipeable para lidar com gestos. Precisamos ter um componente para renderizar o deslize para esquerda e outro para a direita:

```tsx
import { StyleSheet, Text, View, Alert } from "react-native";
import React from "react";
import { MaterialIcons } from "@expo/vector-icons";
import { Swipeable } from "react-native-gesture-handler";
import { Task } from "../types/Task";
import { categories } from "../utils/data/todos";

interface Props {
  task: Task;
  handleRemoveTask: (id: string) => void;
  handleDoneTask: (id: string) => void;
}

const ItemCard = ({ task, handleRemoveTask, handleDoneTask }: Props) => {
  const category = categories.filter((c) => c.value === task.category);

  const handleDelete = () => {
    Alert.alert("Tarefas", "Tem certeza que deseja excluir esta tarefa?", [
      {
        text: "Nãooooooo",
        style: "cancel",
      },
      { text: "Simmmm", onPress: () => handleRemoveTask(task.id) },
    ]);
  };

  const LeftAction = () => {
    return (
      <View style={styles.swipeLeft}>
        <MaterialIcons
          name="done"
          size={20}
          color="#fff"
          onPress={() => handleDoneTask(task.id)}
        />
      </View>
    );
  };

  const RightAction = () => {
    return (
      <View style={styles.swipeRight}>
        <MaterialIcons
          name="delete"
          size={20}
          color="#fff"
          onPress={handleDelete}
        />
      </View>
    );
  };

  return (
    <Swipeable renderLeftActions={LeftAction} renderRightActions={RightAction}>
      <View style={styles.container}>
        <View
          style={{
            borderStyle: "solid",
            height: "100%",
            borderLeftWidth: 6,
            borderColor: category[0].color,
            marginRight: 10,
          }}
        />
        <Text style={styles.title}>{task.title}</Text>
      </View>
    </Swipeable>
  );
};



export default ItemCard;
```

- Na pasta components crie um arquivo CategoryItem.tsx. Este arquivo recebe como props o item (do tipo Category), handleSelectCategory (recebe string e retorna void) e selectedCategory (string). Deve retornar um TouchableOpacity com um texto com o nome da categoria. Implemente a lógica para colocar uma borda com cor de destaque para a categoria selecionada. No touchable opacity chamar no onPress a função handleSelectCategory mandando o valor do item.

- Na tela Home.tsx:
    - Inserir: import "react-native-get-random-values";
    - Criar os seguintes valores de estado/contexto:

```
const { user } = useContext(UserContext);
const [open, setOpen] = useState(false);
const [categoryValue, setCategoryValue] = useState(null);
const [selectedCategory, setSelectedCategory] = useState("all");
const [taskInput, setTaskInput] = useState("");
const [taskList, setTaskList] = useState<Task[]>([]);
const [filteredTasks, setFilteredTask] = useState<Task[]>([]);
```

    - Criar uma função storeTasks que deve receber o array de tarefas e salvar em AsyncStorage.

    - Criar uma função getTasks para recuperar os valores das tarefas salvas em AsyncStorage, sentando o setTaskList com o valor recuperado.

    - Criar uma função getData que deve chamar o getTasks com await.

    - Crie um useEffect sem dependência chamado o getData.

    - Crie uma função handleAddTask que deve verificar se os inputs possuem valor, criar um clone da lista de tarefas, adicionar a nova tarefa ao array e chamar o storeTasks com o novo valor.Em seguida chamar o getData e limpar os inputs.

    - Crie uma função handleRemoveTask que deve receber como parâmetro o id da tarefa. Faça um filter para retornar os elementos diferentes do id da tarefa, chame o storeTasks com o array filtrado e o getData.

    - Crie uma função handleDoneTask que deve receber o id da tarefa, faça um clone da tasklist, encontre o index do elemento no array, altere a propriedade completed para true e chame storeTasks com o valor do clone e o getData.

    - Crie uma funçãohandleSelectedCategory que recebe um type: string, dentro dela set o setSelectedCategory com o valor recebido por parâmetro e faça um switch/case com o type. Caso seja ‘all’ deve filtrar por todas as tarefas que não estejam concluídas. Caso seja ‘done’, deve filtrar pelas tarefas concluídas. No default filtre onde a categoria do elemento seja igual ao type recebido na função. Em todos os casos chame a função setFilteredTask com o valor retornado no filtro.

- Inserir um input para o usuário digitar o texto da nova tarefa. O value deve ser taskInput e o onChangeText deve ser setTaskInput.

- Inserir um DropDownPicker:

```
<DropDownPicker
  style={styles.dropdown}
  open={open}
  value={categoryValue}
  items={categories.filter(
    (c) => c.value !== "all" && c.value !== "done"
  )}
  setOpen={setOpen}
  setValue={setCategoryValue}
  placeholder="Escolha uma categoria"
  theme="DARK"
  placeholderStyle={{
    color: "#ccc",
    fontSize: 16,
  }}
  listItemLabelStyle={{
    color: "#fff",
    fontSize: 16,
    paddingLeft: 15,
  }}
  dropDownContainerStyle={{
    backgroundColor: "#11212D",
  }}
  selectedItemContainerStyle={{
    backgroundColor: "#1c2541",
  }}
  selectedItemLabelStyle={{
    fontWeight: "bold",
    fontSize: 16,
    color: "#fff",
  }}
/>
```

- Inserir um ícone de enviar.

- O DropDown e o ícone devem estar em uma view com direction row.
- Inserir uma Flatlist com os dados da categoria, renderizando o componente CategoryItem.

- Inserir uma Flatlist com o data recebendo filteredTasks e renderizando o componente ItemCard.

- Faça as devidas customizações de estilo.

- No Canva, crie uma logo bem bacana para o app. Baixe o arquivo e insira na pasta assets.No arquivo app.json nas linhas que constam como ./assets/icon.png , ./assets/splash.png e ./assets/adaptive-icon.png, substitua pelo seu arquivo.

- Envie o repositório do Github ao professor.