



FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA - UNIFOR
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PATRÍCIO HALLISSON LIMA MOREIRA - 2124734
RODRIGO CÉSAR SOARES MENDES - 2124690

ATIVIDADE FINAL

Fortaleza-CE
2023

PATRÍCIO HALLISSON LIMA MOREIRA
RODRIGO CÉSAR SOARES MENDES

ATIVIDADE FINAL

Elaboração da atividade final apresentada ao curso de Análise e Desenvolvimento de Sistemas da Universidade de Fortaleza para conclusão da atividade final do componente curricular Programação Funcional.

Orientador: Prof. Dr. Samuel Lincoln Magalhães Barrocas

Fortaleza
2023

SUMÁRIO

- 1 ETAPA 1 – PROPOSTA DE IDEIA DE SOFTWARE**Erro! Indicador não definido.
- 2 ETAPA 2 - PLANO DOCUMENTAL E ESPECIFICAÇÃO TÉCNICA.....**Erro! Indicador não definido.

1 DOCUMENTO DE REQUISITOS

1.1 Requisitos Funcionais

1.1.1 Registrar de Usuário

O usuário deve ser capaz de se cadastrar fornecendo um nome de usuário e senha. Este requisito está representado na função “**register_user**”.

1.1.2 Menu de opções

O sistema deve apresentar um menu com opções para o usuário escolher a ação que deseja executar. O menu de opções está representado na função “**main**”.

1.1.3 Login de Usuário:

O usuário deve ser capaz de fazer login usando suas credenciais (nome de usuário e senha). Este requisito está representado na função “**login_user**”.

1.1.4 Cadastrar um carro

Os administradores podem cadastrar um carro no sistema. O requisito de cadastro de carro no sistema está representado na função “**create_car**”.

1.1.5 Listar Carros Disponíveis:

O usuário deve ser capaz de listar os carros disponíveis para aluguel e está representado na função “**list_available_cars**”.

1.1.6 Alugar Carro:

O usuário deve ser capaz de alugar um carro disponível após fazer login. O requisito de alugar um carro está representado na função “**rent_car**”.

1.1.7 Atualização de Senha do Usuário:

Os usuários podem atualizar sua senha após fazer login. O requisito de atualização de senha do usuário está representado na função “**update_user**”.

1.1.8 Exclusão de Usuário:

Os administradores podem excluir um usuário do sistema. O requisito de exclusão de usuário está representado na função “**delete_user**”.

1.1.9 Listar todos os usuários cadastrados

Os administradores devem ser capazes de listar todos os usuários cadastrados no sistema. O requisito de listar todos os usuários cadastrados está representado na função “**read_users**”.

1.1.10 Listar Todos os Carros Cadastrados:

Os administradores podem listar todos os carros cadastrados no sistema. O requisito de listar todos os carros cadastrados está representado na função “**read-cars**”.

1.1.11 Atualizar Status de um Carro:

Os administradores podem atualizar o status de um carro (disponível ou alugado). O requisito de atualizar o status de um carro está representado na função “**update-car**”.

1.1.12 Excluir Carro:

Os administradores podem excluir um carro do sistema. O requisito de excluir um carro está representado na função “**delete_car**”.

1.2 Requisitos não-funcionais

1.2.1 Persistência de Dados em Arquivos de Texto:

Os dados dos usuários e carros são armazenados em arquivos de texto ("**users.json**" e "**cars.json**") para persistência. A persistência dos dados em arquivos de texto está implementada em todas as funções relacionadas a usuários e carros, onde os dados são lidos e gravados nos arquivos "**users.json**" e "**cars.json**".

1.2.2 Criptografia de Senha:

As senhas dos usuários são criptografadas antes de serem armazenadas para garantir a segurança das credenciais. A criptografia de senha está representada na função **encrypt_password**, onde as senhas dos usuários são criptografadas antes de serem armazenadas.

1.2.3 Integridade dos Dados:

A integridade dos dados é mantida por meio do armazenamento em arquivos de texto e manipulação cuidadosa dos dados dos usuários e carros.

1.2.4 Controle de Acesso para Usuários Comuns:

Os usuários comuns podem realizar operações como listar carros e alugar carros após fazer login. O controle de acesso para usuários comuns é implementado em todas as funções relacionadas a carros, onde apenas usuários logados podem listar e alugar carros.

1.2.5 Controle de Acesso para Administradores:

Os administradores têm permissões adicionais, como listar todos os carros, atualizar o status dos carros e excluir carros. O controle de acesso para administradores é implementado nas opções "**6 (excluir usuário)**", "**8 (atualizar status de um carro)**", "**9 (excluir um carro)**" e "**10 (criar um carro)**" em "**main**", onde apenas administradores podem executar essas ações.

1.3 Responsabilidades

O aluno Patrício Hallisson Lima Moreira ficou responsável pela documentação dos requisitos, já o aluno Rodrigo César Soares Mendes ficou encarregado do código.

1.4 Implementação dos conceitos de programação funcional

- Função lambda

A função lambda está incluída na função “*list_available_cars()*” para verificar o status de aluguel dos carros.

- List comprehension.

A list comprehension está sendo usada na função “*list_available_cars()*” para criar a lista de carros disponíveis.

- Função de continuação.

A função de continuação está implementada na função “*continue_operation()*”, que pausa a execução do programa até que o usuário pressione Enter para continuar

- Closure

A Closure foi implementada na função “*create_user_closure()*”, que retorna uma função interna que exibe uma mensagem de boas-vindas personalizada para o usuário logado.

- Função de alta ordem

A função de alta ordem está implementada na função *apply_to_each_user()*, que recebe uma lista de usuários e uma função como argumentos e aplica essa função a cada usuário na lista

- Monad

O monad “*ResultMonad*” foi utilizado no código para representar operações que podem falhar e retornar informações sobre o sucesso ou falha de uma operação.

Para a criação do monad foi utilizado o chatbot (chatGPT), obtendo-se a resposta abaixo:

Utilização do ChatGPT

Pergunta: Implemente um monad no nosso código principal

Resposta:

```
# Definindo o monad Maybe
class Maybe:
    def __init__(self, value):
        self.value = value

    def bind(self, func):
        if self.value is None:
            return Maybe(None)
        else:
            return func(self.value)

# Modificando a função login_user para retornar um objeto Maybe
def login_user(username, password):
    try:
        with open(USERS_FILE, 'r') as file:
            users = json.load(file)
    except FileNotFoundError:
        return Maybe(None)

    for user in users:
        if user['username'] == username:
            hashed_password = encrypt_password(password)
            if user['password'] == hashed_password:
                return Maybe(user.get('role', 'user'))
    return Maybe(None)

# Agora, podemos usar o monad Maybe para verificar o resultado do login
def main():
    # ...

    if choice == "2":
        username = input("Digite o nome de usuário: ")
        password = input("Digite a senha: ")
        maybe_role = login_user(username, password)

        if maybe_role.value is not None:
            logged_in_user_role = maybe_role.value
            logged_in_user = username
            print(f"Você está logado como {logged_in_user} ({logged_in_user_role}).")
        else:
            print("Login falhou.")

    # ...
```


1.5 Repositório no GitHub

<https://github.com/rodscesars/ProgramacaoFuncionalBD>