

Autenticação de Usuários

Conteúdos

- ↗ Autenticação de Usuários
- ↗ Segurança de Senhas
- ↗ Flask-Login
 - ↗ Instalação e Configuração
 - ↗ Configuração de classe de usuário
 - ↗ Proteção de Rotas

Autenticação de Usuários

Autenticação de Usuários

- ~ As aplicações modernas oferecem experiências personalizadas.
- ~ Dessa maneira, é possível realizar o cadastro na aplicação e realizar os serviços oferecidos além de personaliações.
- ~ Para realizar o cadastro, é necessário fornecer algumas informações e principalmente uma senha.
- ~ É nesse ponto que precisamos focar a nossa atenção devido as questões de segurança.

Autenticação de Usuários

- O processo de cadastro de usuários e, posteriormente, seu ingresso na aplicação utilizando, digamos, um email e uma senha é chamado de **autenticação**.
- Na autenticação, buscamos validar credenciais de acesso que definem os usuários de forma única como emails, matrículas ou alguma outra informação que seja única.
- Um ponto importante neste processo é garantir que a senha do usuário seja bem protegida.

Autenticação de Usuários

Neste material, teremos 2 exemplos:

- ↗ Uso do pacote `werkzeug` para criptografia de senhas - [Exemplo 01](#)
- ↗ Uso do pacote `Flask-Login` para gerenciamento de sessões de usuário - [Exemplo 02](#)

Segurança de Senhas

Segurança de Senhas

- ~ Um aspecto importante sobre o sistema de autenticação de usuários é a forma de lidar com a senha do usuário.
- ~ É comum sistemas que retornam uma string com a senha literal do usuário diante de um pedido de recuperação.
- ~ Esta é uma falha de segurança muito importante.
- ~ Muitos usuários acabam repetindo senhas em diferentes serviços e logo teriam tudo exposto em caso de vazamento da senha.

Segurança de Senhas

- No [Exemplo 01](#), a aplicação mostra uma técnica de segurança importante que é a geração de **hash** a partir da senha do usuário.
- Quando geramos um **hash** não temos como desfazê-lo para saber qual o valor que o gerou.
- O que podemos fazer é o seguinte:
 - dada a informação original (por exemplo, a senha), checar se o hash foi gerado a partir dela.
- Este mecanismo adiciona uma boa camada de segurança já que o sistema não salva a senha do usuário.

Segurança de Senhas

- No [Exemplo 01](#), você salva nome e senha de usuários na sessão e pode verificar as senhas para cada um deles.
- A partir destas funcionalidades, é possível ver o processo de geração e verificação de hash através do módulo `werkzeug.security`.
- A seguir, os principais aspectos serão apresentados.
- Baixe o código e execute-o para avaliar as funcionalidades.

Segurança de Senhas

➤ Para usar o módulo `werkzeug.security` é necessário importar as funções que usaremos.

Exemplo

```
from werkzeug.security import generate_password_hash, check_password_hash
```

➤ Pelos nomes, já podemos ver o propósito de cada uma delas.

➤ `generate_password_hash` : é usada para gerar hashes, e;

➤ `check_password_hash` : verifica se um hash foi gerado a partir de uma senha informada.

➤ Na página seguinte, veja uma imagem da aplicação com um usuário adicionado.

Segurança de Senhas

Criptografia de Senhas

Cadastrar

enviar

Usuários

Nome: jose

Hash

bcrypt:32768:8:1\$Bq9SWJoQeCivJHGL\$cb739b6b83b1c3297f1b3e22770ec905bab5a35a4b4df07ad1961281e3bf674bd42882e3a4b1eb83c95da502fc0664fb7f70891f59745285fdbf5cc5db78db27

Adivinhar

Nome: jose

Hash

bcrypt:32768:8:1\$WeOv1xgFjggHHx7c\$7338366dff06f8e78b0a4d6df4c5b656c583be842defb46c742f5ed881c79c58eabc515fdf31482ff623cbb39e660090992bc55b0de38820daabf48a7177b1ff

Adivinhar

Fonte: própria

Programação de Sistemas para Internet - Prof. Romerito Campos - 2025

Segurança de Senhas

- O usuário pode enviar novos cadastros que serão processados pela view `index`.
- O ponto principal é a geração do hash a partir da `senha`

Exemplo

```
# trecho de cripto.py
dados = {
    'nome': nome,
    'senha': senha,
    'hash': generate_password_hash(senha)
}
lista.append( dados )
```

Segurança de dados

- Os dados do usuário serão mostrados na tela e você pode tentar adivinhar a senha. A view `adivinhar` recebe as requisições.

Exemplo

```
# trecho de cripto.py
if user['nome'] == nome and check_password_hash(user['hash'], senha):
    flash('Você acertou a senha', 'success')
    return redirect(url_for('index'))
```

- O trecho acima vai verificar se o `hash` foi gerado a partir da `senha` que você supõe ser a correta.

Flask-Login

Flask-Login

- A extensão `Flask-Login` tem o objetivo de gerenciar as sessões de usuário no sistema de autenticação. Veja mais [aqui](#).
- Desta maneira, ao fazer login no sistema o usuário será lembrado a medida que navega pelas páginas sem a necessidade fazer login novamente (a menos que a sessão expire)
- Podemos juntar os recursos do `Flask-Login` com o módulo `werkzeug.security` para oferecer uma experiência mais segura.
- Neste exemplo, vamos manter uma lista de usuários simulando banco de dados.

Flask-Login

- ~ Alguns aspectos devem ser levados em consideração
 - ~ É necessário instalar o pacote
 - ~ É necessário ter um **Modelo** preparado para ser usado no Flask-Login
 - ~ Vamos proteger as rotas de acesso indevido (usuários não logados)
 - ~ Vamos preparar o ambiente de cadastro e login de usuário de acordo com o recursos oferecidos pelo flask
- ~ Um ponto importante é que usaremos um Modelo. Entretanto, não adotaremos banco de dados.

Instalação e Configuração

Instalação e Configuração

➤ A primeira coisa a ser realizada é a instalação do pacote Flask-Login

Exemplo

```
pip install Flask-Login
```

➤ Este pacote oferece o gerenciamento de login/logout de usuários e controle de acesso a rotas protegidas.

➤ É necessário incorporar outros pacotes de funcionalidades para tornar a autenticação de usuários mais completa

Instalação e Configuração

- ~ Utilizarei o [Exemplo 02](#) para apresentar os principais conceitos do Flask-Login
- ~ No código de exemplo, utilizaremos ma dicionário de usuários salvo na sessão, através do objeto `session`.
- ~ A aplicação permite cadastrar usuários, fazer login e logout. Além disso, algumas rotas são bloqueadas para acesso sem login.
- ~ Dois pontos fundamentais da configuração são:
 - ~ Realizar o `import` das classes e funções da extensão
 - ~ Preparar o **Modelo** de usuário da aplicação

Instalação e Configuração

➤ Considerando que o pacote foi instalado podemos utilizá-lo da seguinte forma:

Exemplo

```
from flask_login import LoginManager, login_user, login_required, logout_user
login_manager = LoginManager() # fazer integração com Flask
app = Flask(__name__)
login_manager.init_app(app) # inicializar o app associado a Flask-Login
app.config['SECRET_KEY'] = 'ULTRAMEGADIFICIL'
```

➤ Vejamos os detalhes desta etapa

Login Manager

➤ A classe LoginManager é responsável por integrar o Flask ao Flask-Login

Exemplo

```
login_manager = LoginManager()
```

➤ Esta classe vai gerenciar a sessão do usuário e a maneira como recuperamos o usuário da sessão: a recuperação é feita com base no valor de `id`.

➤ O Flask-Login utiliza o objeto session durante o gerenciamento que ele realiza. Logo devemos definir a chave secreta da aplicação:

Exemplo

```
app.config['SECRET_KEY'] = 'ULTRAMEGADIFICIL'
```

Modelo User

- O próximo passo é definir um Modelo **User**.
- Naturalmente, a ideia de Modelo remete ao uso de banco de dados. Entretanto, não vamos utilizar neste exemplo.
- Mesmo sem utilizar banco de dados é necessário ter a definição da classe **User**: requisito de configuração do Flask-Login.
- O Modelo criado para este exemplo tem funções que permite consultar usuários no "Banco" que vamos utilizar.

Modelo User

➤ Para implementar o modelo, um módulo python será adicionado. Veja no código a pasta `models` com o arquivo `__init__.py`. Neste arquivo, é definida a classe `User`.

➤ A classe `User` herda da classe `UserMixin`, que é definida no pacote Flask-Login:

Exemplo

```
class User(UserMixin)
```

➤ Dessa maneira, o Flask-Login tem uma classe `User` para utilizar no gerenciamento de uma sessão de usuário.

➤ Neste exemplo, vamos lidar com três propriedades no Modelo: `id`, `email`, `password`

Modelo User

➤ Tendo o Modelo User sido definido, ele é importado no arquivo da aplicação:

```
from models import User
```

Exemplo

➤ Ele vai ser utilizado na próxima configuração importante do Flask-Login que é a definição da função `load_user`

```
@login_manager.user_loader  
def load_user(user_id):  
    return User.get(user_id)
```

Exemplo

➤ O Flask-Login vai utilizar esta função `load_user` para recuperar os dados de um usuário a partir de `user_id` salvo na sessão (lembre-se que este é seu papel)

Modelo User

- Antes de prosseguirmos nos detalhes o exemplo, vale reforçar que a classe `UserMixin` possui 4 métodos que podem ser sobrescritos pela classe `User` que criamos:
 - `is_authenticated` : checar se o usuário está autenticado
 - `is_active` : verifica se ele está ativo
 - `is_anonymous` : verificar se é usuário anônimo
 - `get_id` : obter o identificador do usuário pelo qual se busca ele em alguma fonte de dados(Banco de dados, por exemplo)

Modelo User

- No [Exemplo 02](#), foi necessário redefinir a função: `get_id()` e criar uma nova função chamada `get()`.
- Primeiro, a função `get_id` foi redefinida de maneira que poderemos considerar o `email` do usuário (que deve ser único) como sendo seu `id`, logo a implementação é:

Exemplo

```
def get_id(self):  
    return self.email
```



Normalmente, o id de usuário é a informação que definimos na tabela do banco de dados como id

Modelo User

~ A método `get()` é utilizadao procurar por um usuário dentro do dicionário que guardamos no objeto `session`.

Exemplo

```
@classmethod
def get(cls, user_id):
    lista = session.get('usuarios')
    if user_id in lista.keys():
        user = User(email=user_id, password=lista[user_id])
        user.id = user_id
        return user
```

~ Note que este código poderia ser executado fora da classe `User` e do método `get()`. Entretanto, esta forma permite melhor organização do código.

Modelo User

- Recapitulando a função `load_user`, agora você compreend que a execução de `User.get()` é uma chamada para função `get` do slide anterior.

Exemplo

```
@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)
```

- `load_user` é essencial para o funcionamento do Flask-Login
- Há outros métodos na classe `User`. Eles são importantes, mas o Flask-Login poderia funcionar sem eles.

Proteção de Rotas

➤ Podemos partir para a próxima etapa que é proteger as rotas que desajarmos. No arquivo `app.py` veja o código:

Exemplo

```
@app.route('/dashboard')
@login_required
def dash():
    return render_template('pages/dash.html')
```

➤ Importarmos `login_required` de Flask-Login e decoramos a rota

➤ Agora apenas usuários logados pode acessar esta rota.

Cadastro de Usuário

~ O cadastro de usuário é simples. Os formulários e manipulação da lista de usuários continuam iguais aos exemplos anteriores, apenas estão encapsulados na classe

User.

~ Há uma rota para registro e o trecho mais importante é o seguinte:

Exemplo

```
#trecho de app.py | view register
usuarios = User.all()
if email not in usuarios.keys():
    user = User(email=email, password=password)
    user.save()
    # 6 - logar o usuário após cadastro
    login_user(user)
    flash("Cadastro realizado com sucesso")
    return redirect(url_for('dash'))
```

- Obtemos a lista de usuários com `User.all()`.
- Em seguida, no if, verificamos se existe o novo email na lista de emails dos usuários.
- Caso não exista, um objeto `user` é criado com `email` e `senha`.
- Em seguida, ele salva os dados do novo usuário na lista de usuário com `user.save()`
- O usuário é logado por meio de `login_user(user)`, que é uma função da extensão Flask-Login
- O usuário é redirecionado para o dashboard.

Login de Usuário

~ O login de usuário também é simples. Destaca-se da função de login o código abaixo:

Exemplo

```
user = User.find(email)
if check_password_hash(user._hash, password):
    login_user(user)
    flash("Você está logado")
    return redirect(url_for('dash'))
```

~ Buscamos usuário pelo email com `User.find()`

~ Em seguida, testamos a senha do usuário e em caso positivo ele é logado com `login_user()`

- Neste ponto, aplica-se a função `check_password_hash` considerando a hash do `user` recuperado da lista de usuários e a senha enviada pelo formulário.
- Caso haja sucesso, então a função `login_user` da extensão Flask-Login é usada e o usuário está logado.
- O usuário é redirecionado
- Caso contrário, ele recebe de volta o formulário

Logout de usuário

➤ O logout de usuários também utiliza uma função da extensão Flask-Login

➤ Observe o código abaixo:

Exemplo

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('index'))
```

➤ A função `logout_user()` foi aplicada e a sessão encerrada.

Acessando o usuário logado

➤ A Extensão Flask-Login permite consultar o usuário logado através do objeto `current_user`

```
from flask_login import current_user
```

Exemplo

➤ Na página `index.html` temos um código que mostrar os links de cadastro e login de usuário se não houver ninguém logado.

➤ Veja o código a seguir