

## Flask-login: um pacote recheado de recursos

Um dos recursos mais comuns de uma aplicação web é permitir que o usuário faça login. No contexto de uma aplicação web em Flask, esse processo de login envolve etapas importantes que podem ser implementados e gerenciados utilizando o pacote flask-login (uma extensão do Flask).

Com o flask-login é possível realizar gerenciamento de sessões, proteção de rotas entre outras funcionalidades, e principalmente autenticação, manipulação e segurança das credenciais e senhas dos usuários.

## Autenticação e validação

### Validação: O que é e onde ela é feita?

Numa aplicação a validação é o processo de garantir que os dados inseridos por um usuário em um formulário (ou em qualquer outro lugar) na aplicação estejam corretos e atendam aos critérios definidos.

A validação de dados é feita em formulário usando bibliotecas como Flask-WTF ou WTForms. Os formulários podem conter validadores para garantir que os dados inseridos pelo usuário sejam válidos antes de serem processados pelo servidor.

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField,
TextAreaField, SelectField
from wtforms.validators import DataRequired, Length, Email,
ValidationError
```

Isso faz com que os dados inseridos nos formulários estejam corretos e atendam aos critérios definidos: formato, comprimento, unicidade (por exemplo, nomes de usuário únicos).

No exemplo abaixo a classe **RegistrationForm** utiliza validadores integrados do WTForms para garantir que os campos username, email e password atendam a critérios específicos, como serem obrigatórios, terem um comprimento mínimo/máximo e terem um formato de e-mail válido. Esses validadores são definidos dentro da lista **validators** associada a cada campo do formulário.

```
class RegistrationForm(FlaskForm):
    def validate_nome(self, check_user):
        user = User.query.filter_by(nome=check_user.data).first()
        if user:
            raise ValidationError("Usuário já existe! Cadastre outro nome de usuário")

    def validate_email(self, check_email):
        email = User.query.filter_by(email=check_email.data).first()
        if email:
            raise ValidationError("Email já existe! Cadastre outro email")
```

```

username = StringField(label='Username', render_kw={"placeholder":
"Username"}, validators=[DataRequired(), Length(min=2, max=30)])
email = StringField(label='Email', render_kw={"placeholder": "Email"},
validators=[DataRequired(), Email()])
password = PasswordField(label='Password', render_kw={"placeholder":
"Password"}, validators=[DataRequired(), Length(min=4, message="A senha deve
ter no mínimo 4 caracteres.")])
submit = SubmitField(label='Register')

```

Dentro da classe **RegistrationForm**, os métodos **validate\_nome** e **validate\_email** podem ser considerados validadores também. Eles são exemplos de validadores personalizados que são utilizados para validar dados específicos no formulário antes de serem processados pelo servidor.

O método `validate_nome(self, check_user)` verifica se o nome de usuário fornecido já existe no banco de dados.

O método `validate_email(self, check_email)`: verifica se o endereço de e-mail fornecido já está em uso por outro usuário.

**Os validadores estão mais associados ao processo de registro de novos usuário do que ao processo de login de um usuário que já está cadastrado?**

Sim, pois durante o registro de novos usuários é necessário garantir que os dados inseridos pelos usuários sejam válidos e atendam aos critérios definidos antes de serem persistidos no banco de dados.

```

class LoginForm(FlaskForm):
    email = StringField(label='Email', validators=[DataRequired()])
    senha = PasswordField(label='Password', validators=[DataRequired()])
    submit = SubmitField(label='Log In')

```

A validação no processo de login é menos complexa, uma vez que se concentra principalmente na verificação das credenciais do usuário em comparação com os registros existentes no banco de dados. Isso geralmente envolve verificar se o nome de usuário e a senha fornecidos correspondem aos armazenados no banco de dados para um usuário específico. Além de garantir que os campos de entrada não estejam vazios.

### Autenticação: O que é e onde ela é feita?

A autenticação é o processo de verificar a identidade de um usuário, geralmente através de credenciais como nome de usuário e senha. É usado para garantir que apenas usuários autorizados tenham acesso a certas partes da aplicação ou a certos recursos.

Ela é geralmente feita na camada de controle de acesso da aplicação. Isso pode envolver a verificação das credenciais do usuário em um banco de dados, por exemplo.

Em linhas gerais a autenticação verifica a identidade do usuário com base nas credenciais fornecidas durante o login. Isso normalmente envolve a comparação das credenciais (nome de usuário/senha) com as informações armazenadas no banco de dados para confirmar a identidade do usuário.

Se em `forms.py`, definimos os validadores associados aos campos do formulário, na `routes.py` é onde implementamos a **lógica de autenticação**.

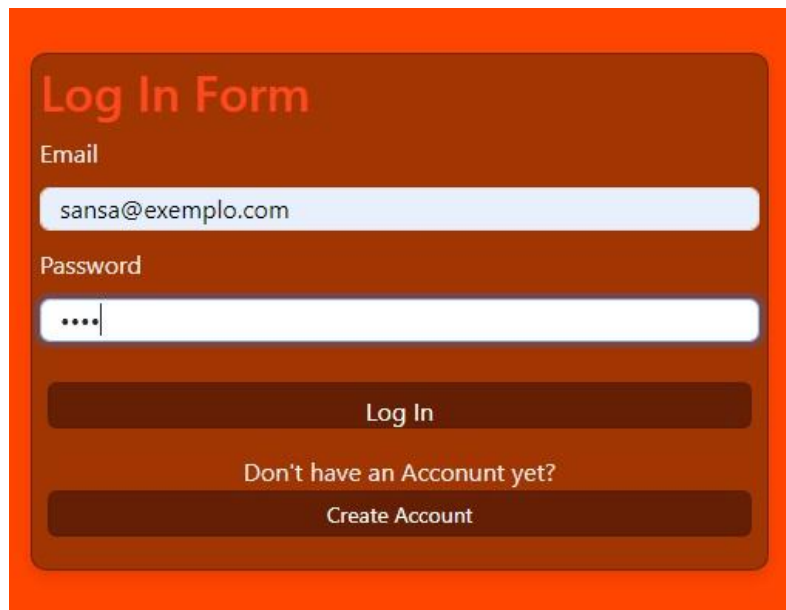
```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        usuario = User.query.filter_by(email=form.email.data).first()
        if usuario and usuario.verificar_senha(form.senha.data):
            login_user(usuario)
            flash(f"Sucesso! Seu usuário é {usuario.username}",
category="success")
            return redirect(url_for('user')) # Redireciona para a
página inicial após o login bem-sucedido
        else:
            flash("Usuário ou senha incorretos. Por favor, tente
novamente.", category="danger")
            return render_template('login.html', form=form)
```

No exemplo acima, instanciamos o formulário responsável pelo login do usuário (LoginForm). Quando o usuário acessa a rota `/login`, o Flask renderiza a página HTML associada a essa rota, que contém um formulário de login.

Formulário associado a rota:

```
class LoginForm(FlaskForm):
    email = StringField(label='Email', validators=[DataRequired()])
    senha = PasswordField(label='Password', validators=[DataRequired()])
    submit = SubmitField(label='Log In')
```

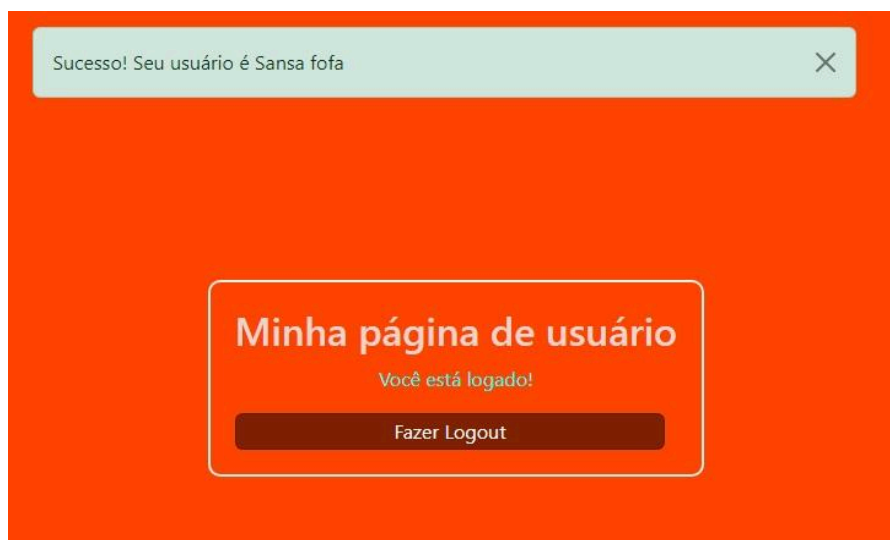
Renderização do template associado ao formulário:

A login form template with a dark orange background. It features a title "Log In Form" in a bold, dark font. Below the title are two input fields: "Email" with the text "sansa@exemplo.com" and "Password" with masked characters "....". A "Log In" button is positioned below the password field. Below the button is a link "Don't have an Account yet?" and a "Create Account" button.

Quando o usuário preenche o formulário e o envia, esses dados são **encapsulados** em um objeto do tipo `LoginForm`. Podemos então usar os métodos como **`validate_on_submit()`** para verificar se o formulário foi submetido e passou na validação, e acessar os dados dos campos do formulário (por exemplo, `form.email.data`, `form.senha.data`) para processá-los conforme necessário.

#### Processamento dos dados do formulário:

Com os dados do formulário acessíveis, a gente pode processá-los conforme necessário na rota. No exemplo, os dados do e-mail e da senha são usados para buscar o usuário correspondente no banco de dados. Se um usuário correspondente for encontrado e a senha estiver correta, o usuário é autenticado e redirecionado para a página inicial do aplicativo.

A user page template with a dark orange background. At the top, there is a light green notification box with the text "Sucesso! Seu usuário é Sansa fofa" and a close button. Below the notification, there is a white box with a dark orange border containing the title "Minha página de usuário", the text "Você está logado!", and a "Fazer Logout" button.

Esses dois componentes trabalham juntos para fornecer uma funcionalidade completa de registro e login em seu aplicativo Flask.