PROGRAMAÇÃO WEB II

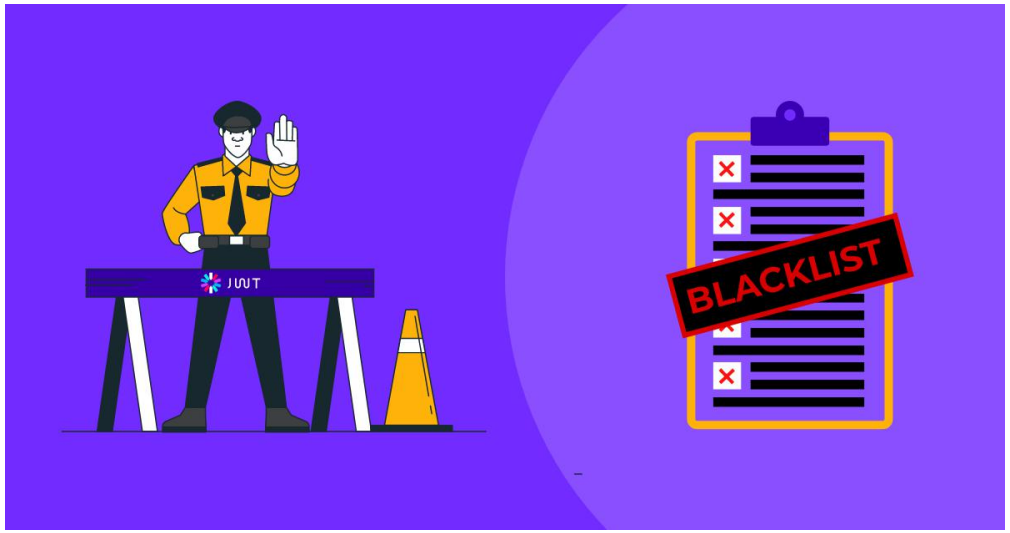
Curso Técnico Integrado em Informática

Lucas Sampaio Leite

Resolução do exercício...

INSTITUTO FEDERAL
Baiano

- Implemente a funcionalidade de logout:
  - Blacklist de tokens: Esta abordagem mantém uma lista de tokens revogados (em memória, Redis ou bancos de dados tradicionais).
  - Durante o processo de logout, o token ativo é adicionado a essa lista.
  - Em cada requisição a rotas protegidas, o sistema deve verificar se o token está presente na blacklist. Caso esteja, ele é considerado inválido.



Implementando o logout com blacklist de tokens.

INSTITUTO FEDERAL
Baiano

```python
jwt_blacklist = set()

@jwt.token_in_blocklist_loader
def check_if_token_revoked(jwt_header, jwt_payload):
    jti = jwt_payload["jti"]
    return jti in jwt_blacklist


@app.post("/logout")
@jwt_required()
def logout():
    jti = get_jwt()["jti"]
    jwt_blacklist.add(jti)
    return {"msg": "Logout realizado com sucesso"}, HTTPStatus.OK
```

Implementando o logout com blacklist de tokens.

INSTITUTO FEDERAL
Baiano

```python
jwt_blacklist = set()

@jwt.token_in_blocklist_loader
def check_if_token_revoked(jwt_header, jwt_payload):
    jti = jwt_payload["jti"]
    return jti in jwt_blacklist


@app.post("/logout")
@jwt_required()
def logout():
    jti = get_jwt()["jti"
    jwt_blacklist.add(jti)
    return {"msg": "Logout realizado com sucesso"}, HTTPStatus.OK
```

O decorador @jwt.token_in_blocklist_loader registra uma função de verificação automática. Sempre que o Flask-JWT-Extended recebe um token (em qualquer rota protegida com @jwt_required()), ele chama essa função para verificar se o token foi revogado.

Implementando o logout com blacklist de tokens.

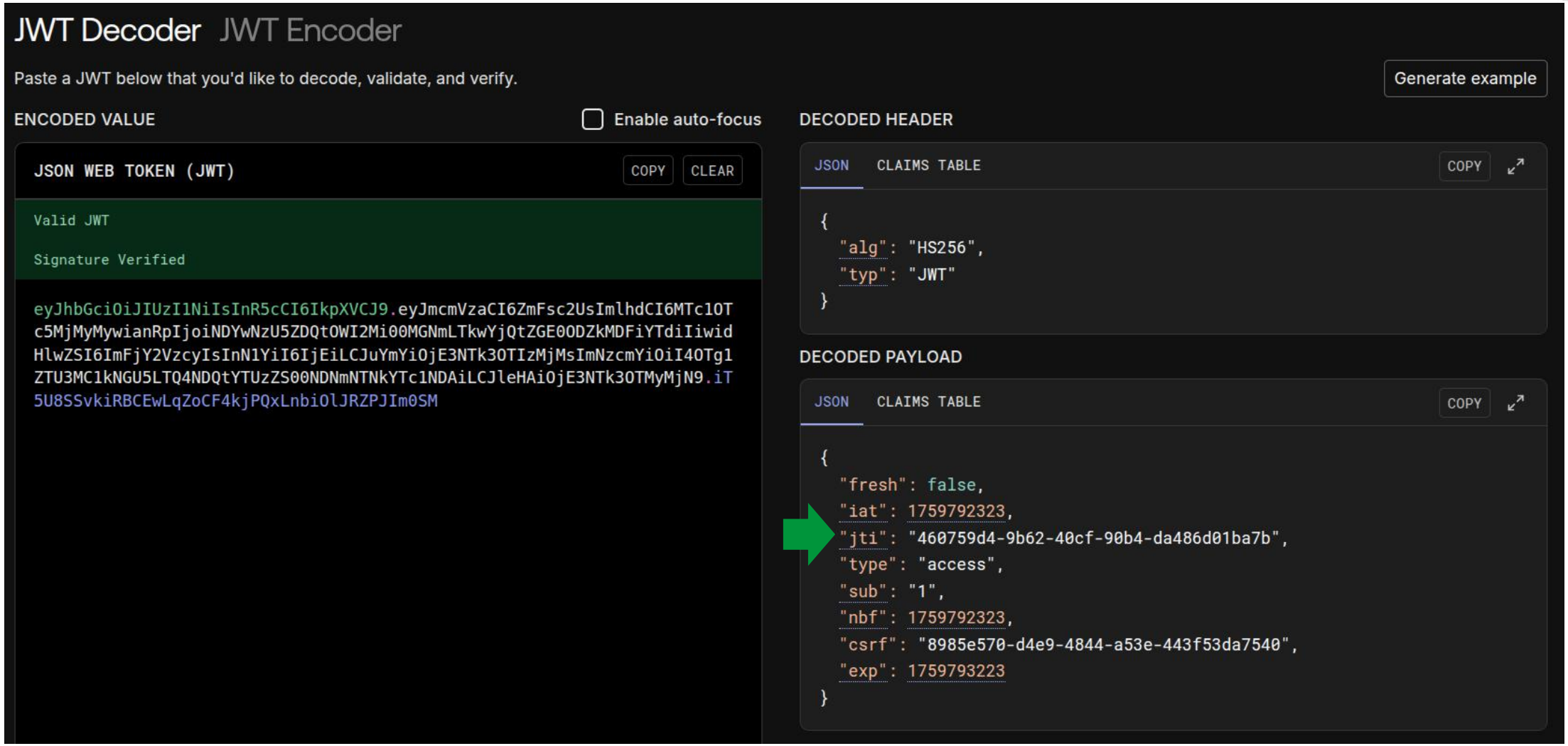
INSTITUTO FEDERAL
Baiano

```python
jwt_blacklist = set()

@jwt.token_in_blocklist_loader
def check_if_token_revoked(jwt_header, jwt_payload):
    jti = jwt_payload["jti"]
    return jti in jwt_blacklist


@app.post("/logout")
@jwt_required()
def logout():
    jti = get_jwt()["jti"]
    jwt_blacklist.add(jti)
    return {"msg": "Logout realizado com sucesso"}, HTTPStatus.OK
```

O parâmetro jwt_payload contém os dados decodificados do token.
A chave "jti" (JWT ID) é um identificador único gerado automaticamente para cada token.

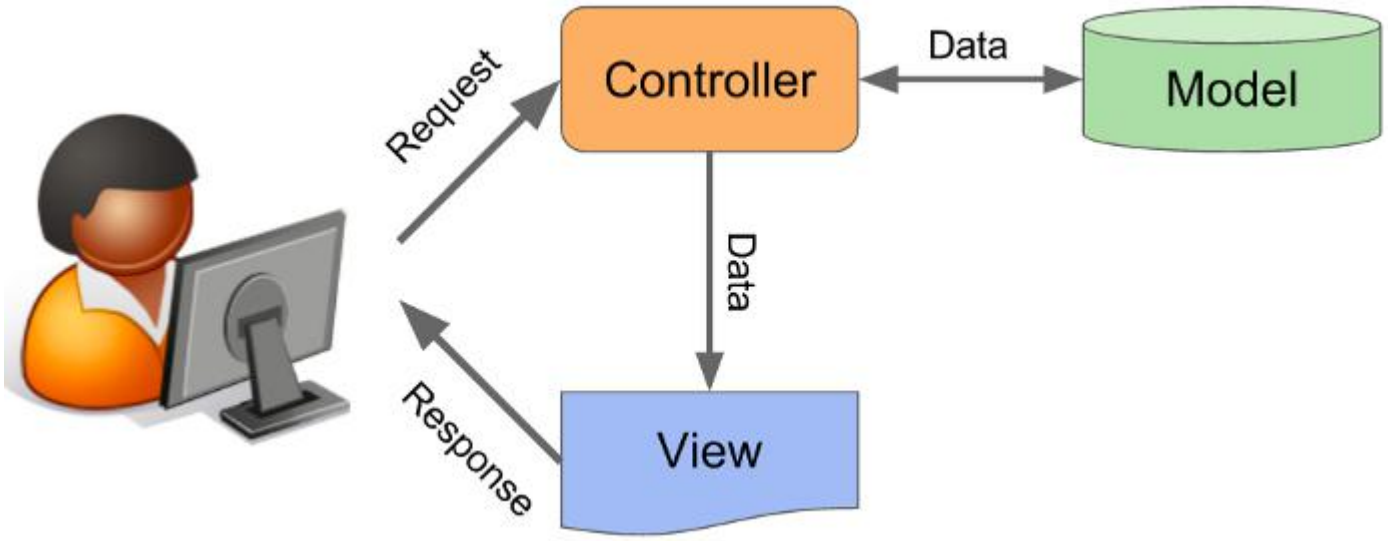
Implementando o logout com blacklist de tokens.

INSTITUTO FEDERAL
Baiano



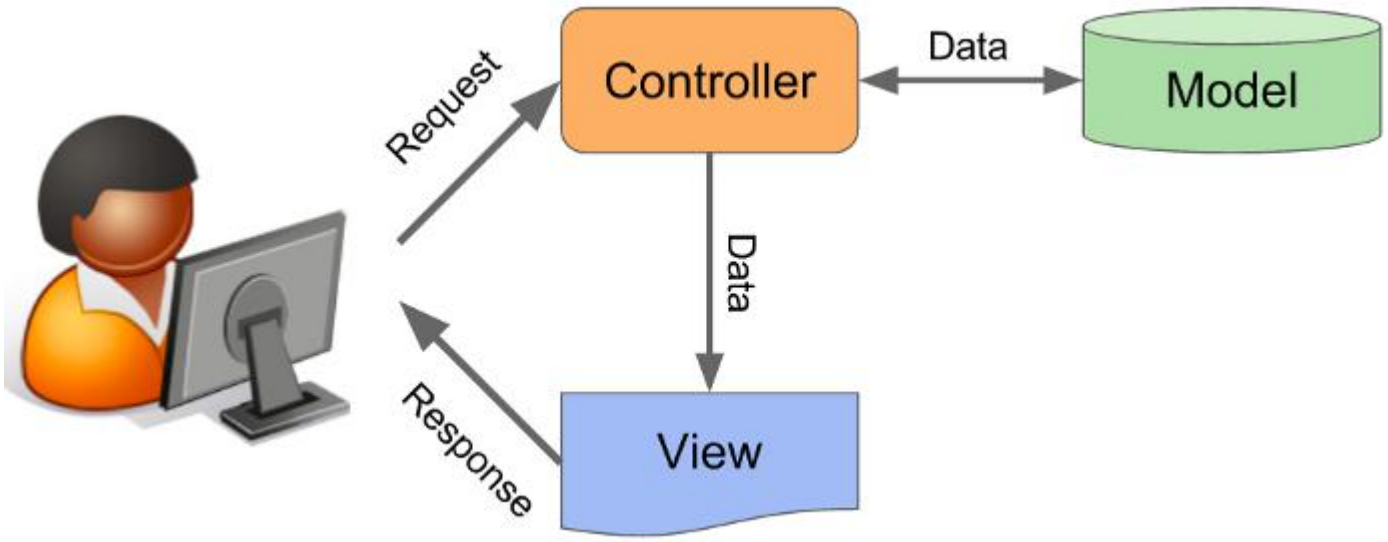
MVC

- O MVC (Model-View-Controller) é um padrão de arquitetura que organiza a aplicação em três camadas principais: Model, View e Controller.



MVC

- O MVC (Model-View-Controller) é um padrão de arquitetura que organiza a aplicação em três camadas principais: Model, View e Controller.



No nosso projeto Flask, mesmo sendo uma API REST (sem páginas HTML tradicionais), essa separação continua existindo, só que de forma adaptada.

Model (Modelo)

INSTITUTO FEDERAL
Baiano

- Representa os dados e as regras de negócio da aplicação.
  - Em um projeto Flask, geralmente fica na pasta models/ (user.py, role.py, post.py).
- Responsabilidade: definir as tabelas do banco, os atributos e regras:
  - ex.: senha criptografada, relacionamento entre usuário e role.
- Exemplo:
  - models/user.py define o modelo User, com atributos (username, email, password_hash) e métodos (set_password, check_password).

View (Visão)

INSTITUTO FEDERAL
Baiano

- É a camada responsável por mostrar a resposta ao usuário.
  - Em uma API REST, a View não é HTML, mas sim o JSON retornado nos endpoints.
- Responsabilidade: apenas exibir dados de forma compreensível, sem lógica de negócio.
- Exemplo:
  - O endpoint /auth/login retorna um JSON com o access_token ou uma mensagem de erro.

Controller (Controlador)

INSTITUTO FEDERAL
Baiano

- É a camada que liga a View ao Model, processando requisições e decidindo o que fazer.
  - Ex.: os blueprints na pasta controllers/ (auth.py, user.py, roles.py).
- Responsabilidade:
  - Receber a requisição HTTP (ex.: POST /auth/login).
  - Chamar os models quando necessário (ex.: buscar usuário no banco).
  - Retornar a resposta para a camada View.
- Exemplo:
  - No controllers/user.py, a função create_user() recebe dados do usuário, cria um novo User no banco e retorna a resposta formatada pela View.

Fluxo no projeto

INSTITUTO FEDERAL
Baiano

- Como funciona o fluxo no MVC?
    1. O usuário (ex.: cliente via Postman, frontend React, etc.) envia uma requisição POST /users.
    2. A View (resposta JSON) ainda não existe. Primeiro a requisição chega no Controller (user.py).
    3. O Controller processa os dados, cria uma instância do Model User e salva no banco de dados.
    4. O Model confirma a persistência do novo usuário.
    5. O Controller decide qual resposta retornar e a delega para a View (views/user.py).
    6. A View se concretiza no JSON enviado ao cliente, com as informações do usuário recém-criado e uma mensagem de sucesso.

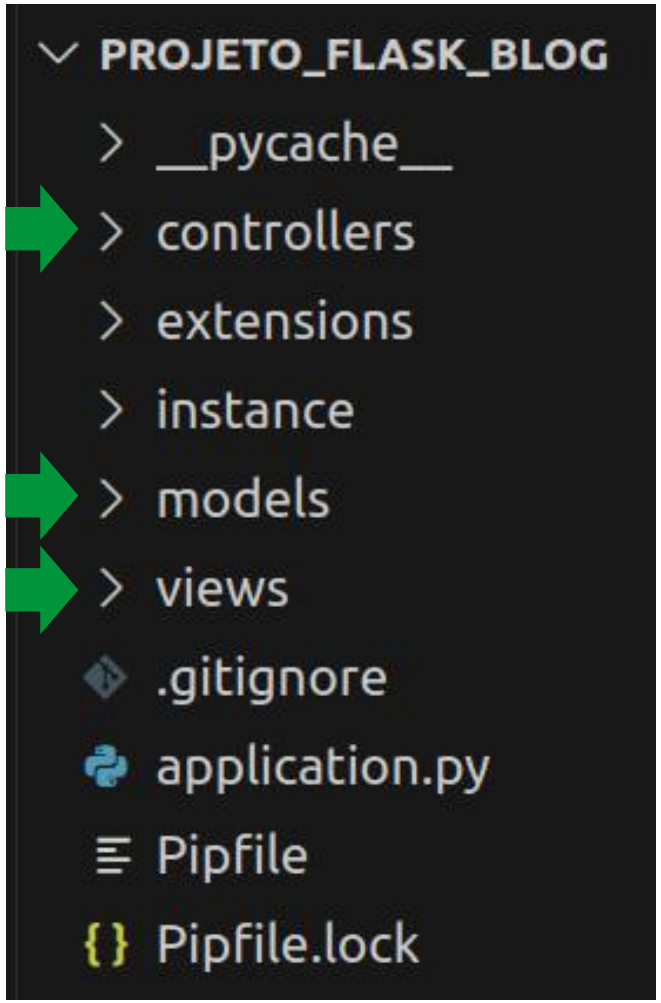
Benefícios do MVC no projeto

INSTITUTO FEDERAL
Baiano

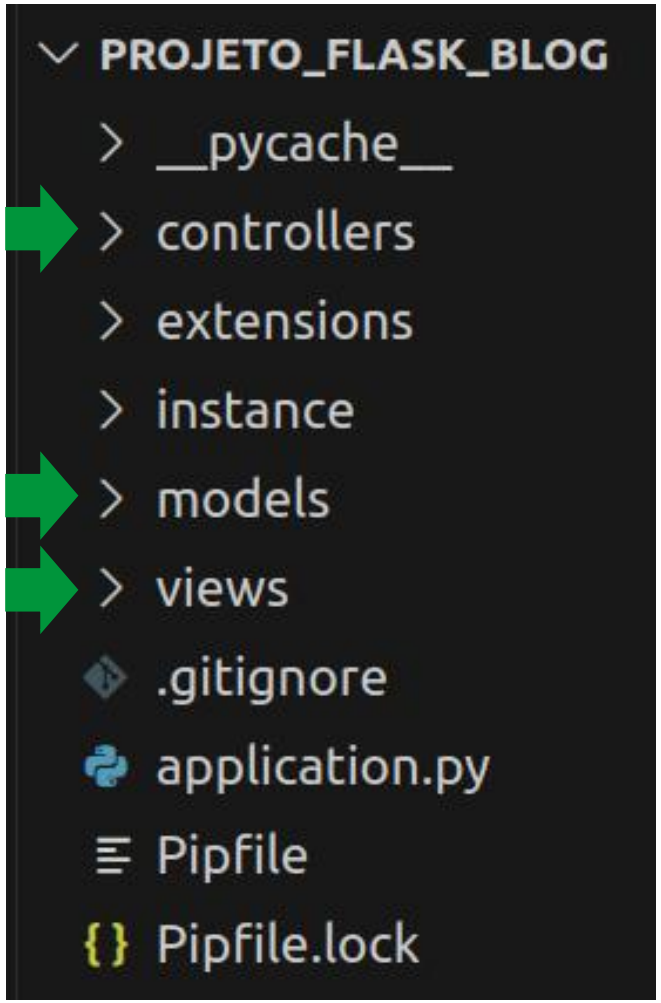
- Separação de responsabilidades:
  - models/ → só regras de dados e persistência (ORM, validações, relacionamentos).
  - views/ → só regras de apresentação da resposta (formatação do que vai ser devolvido ao cliente, geralmente JSON).
  - controllers/ → só regras de entrada/saída (receber requisição HTTP, chamar models, decidir a lógica do fluxo).
  - extensions/ → só inicialização de bibliotecas externas (SQLAlchemy, JWT, etc.).
- Reuso: o mesmo User pode ser usado em endpoints diferentes (auth e user).
- Organização: facilita manutenção, testes e futuras expansões.

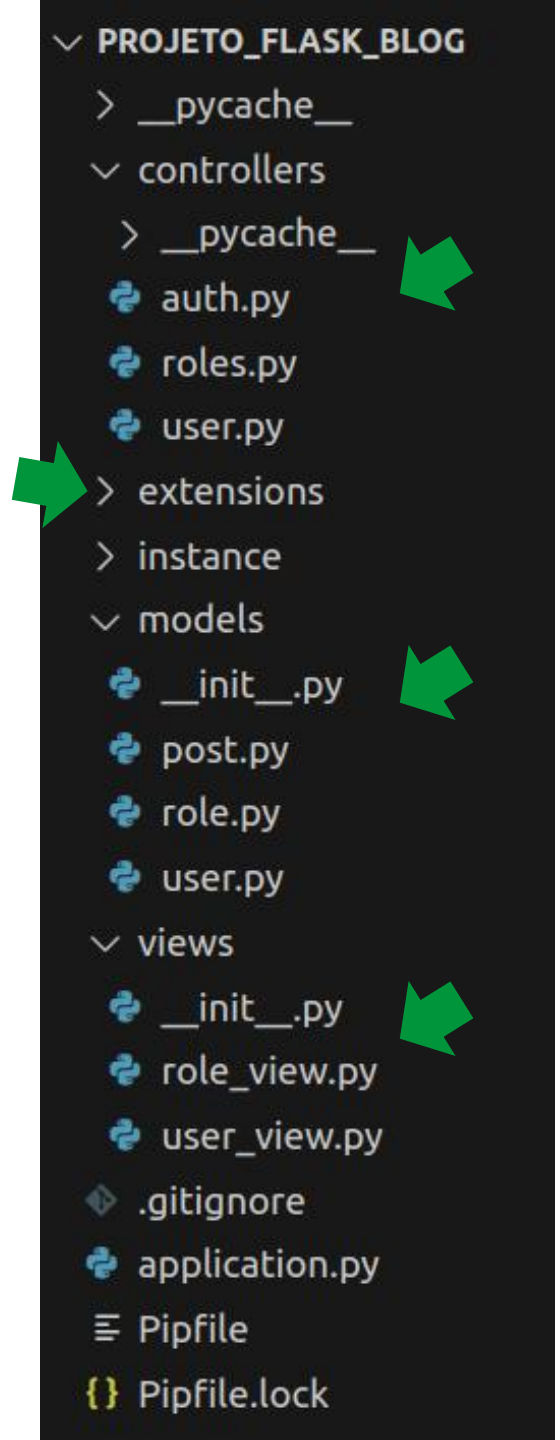
Aplicando MVC ao projeto

INSTITUTO FEDERAL
Baiano



Aplicando MVC ao projeto

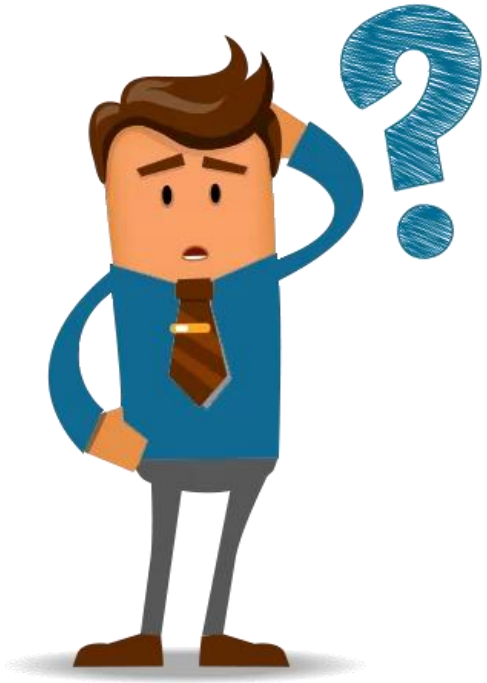




INSTITUTO FEDERAL
Baiano

Dúvidas

INSTITUTO FEDERAL
Baiano



PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite