***O que são os models Django?***

*Os models(modelos) no Django representam a estrutura de dados do seu aplicativo. Eles são a maneira de mapear as tabelas banco de dados com classes Python. Ou seja, cada modelo no Django define a estrutura de uma tabela no banco de dados, e cada atributo dessa classe representa uma coluna nessa tabela.****Como um model é definido?***

*Um model é uma classe que herda de models. Model. Vamos ver um exemplo básico que seria um modelo para o post do blog.  
  
from django.db import models  
  
class Post(models.Model):*

*title = models.CharField(max\_length=200) # Título do post  
 content = models.TextField() # Conteúdo do post  
 author = models.CharField(max\_length=100) # Nome do autor  
 created\_at = models.DateTimeField(auto\_now\_add=True) # Data de criação  
 updated\_at = models.DateTimeField(auto\_now=True) # Data de atualização  
  
 def \_\_str\_\_(self):  
 return self.title****1. Como devo planejar meus models?*** *Compreender o domínio do problema:*

*Antes de definir os models, você precisa entender exatamente o que seu projeto vai fazer e quais são os dados principais que você vai armazenar e manipular.  
  
Pergunta-se:  
- Qual é o objetivo do projeto?  
- Quais são os principais objetos ou entidades que fazem parte do seu sistema? Por exemplo, em um blog as principais entidades seriam:  
  
Post(Postagem do Blog)  
Comentário  
Autor ou Usuário*

**2. Identificar as entidades principais e seus relacionamentos**

Após entender o que sua aplicação precisa fazer, identifique as entidades principais e os relacionamentos entre elas. Vamos supor que você está criando um blog, algumas entidades poderiam ser:

* Post: Representa uma postagem de blog.
* Comentário: Representa os comentários que os usuários deixam nos posts.
* Usuário/Autor: Representa a pessoa que está escrevendo o post ou comentando.

Depois de identificar as entidades, pense em como elas se relacionam:

* Um Post pode ter vários Comentários (relação 1 para muitos).
* Um Usuário pode escrever vários Posts (relação 1 para muitos).
* Um Post pode ser escrito por apenas um Autor (relação muitos para 1).

**3. Determinar os campos e os tipos de dados**

Agora que você sabe quais são as entidades principais e como elas se relacionam, defina quais informações precisam ser armazenadas para cada entidade. Pergunte-se o que cada entidade precisa para funcionar.

Por exemplo, para um Post em um blog:

* Título (CharField, limite de caracteres)
* Conteúdo (TextField, texto longo)
* Data de Criação (DateTimeField)
* Data de Atualização (DateTimeField)
* Autor (Chave estrangeira para a entidade Usuário)

Cada campo tem um tipo correspondente no Django (CharField, TextField, DateTimeField, etc.). Você também pode definir opções como o comprimento máximo (max\_length), se o campo é obrigatório (null=False) ou se deve ter um valor padrão.

**4. Pensar nos relacionamentos entre as models**

Django usa chaves estrangeiras (ForeignKey), relações muitos para muitos (ManyToManyField), ou relações de um para um (OneToOneField) para definir relacionamentos entre models.

Pergunte-se:

* Existe uma relação um para muitos (um autor pode escrever vários posts)?
  + Use ForeignKey.

Exemplo:

author = models.ForeignKey(User, on\_delete=models.CASCADE)

Existe uma relação **muitos para muitos** (por exemplo, um post pode ter várias categorias e uma categoria pode ter vários posts)?

* Use ManyToManyField.

Exemplo:

categories = models.ManyToManyField(Category)

Existe uma relação um para um (um perfil é associado a exatamente um usuário)?

* Use OneToOneField.

Exemplo:

user = models.OneToOneField(User, on\_delete=models.CASCADE)