**Module 3: Building a data model**

* **T**he first thing to do when creating Django project is figuring out what is the data I need in the application.
* In E-commerce apps I need following basic tables or let’s say shared across all e-commerce apps.

1. Product.
2. Category.
3. Customer.
4. Order.
5. Label (something not specifically related to products or customers).

* I should not invent the schema out of my head, it will be based on business requirements.

**Organize modules in app.**

* + Each app inside Django should do one and only one thing and separated from other apps.
    - The philosophy is each app do one thing at a time (loosely couple) and focused on one thing (cohesion).
  + There are couple of ways to build and organize model inside Django app.

1. Create all models inside one app: problem with this approach is when it grows larger it gets messy and harder to maintain
2. Create each one dependently:
   1. problem with this approach when these apps depending on each other.
   2. when user for example going to need to use product will also be forced to use label while label is something not related to product.
3. The solution is to avoid these two weaknesses is to create one core app contain the core models and different app that contain the label:
   1. so, user now will optionally use label inside the app.
      1. Minimum coupling and high cohesion.

**Creating** **Models in apps**.

* To create app models, I go to the app > models
* All app models related to the app should be created in this file.
* To create a db model I create a class and inherit it from models.Model.
* Now when I define fields, I need to give it.

**Relationships**

* There are 3 relationships represent model fields.

1. One to one.
   1. When I create one to one relation I have to add option primary\_key=True because if I don’t, Django will allow adding more addresses,

And I will end up having 1 to many relationship, to avoid this I make the relation in child as primary key so I don’t allow multi addresses.

* 1. A good example of one to one is user and email: each user can have only one email, and each email can have only one user.

1. One to many:
   1. Example of one to many is: order and user: where order belong to one customer and customer can have multi orders.
2. Many to many
   1. A good example of many to many is order and product: an order can have multiple products and product can be in multiple orders.
   2. We can represent m2m relation in association class, the way to handle this is to create for example a productItem class:
      1. productItem relation with product is: product can have many productItem, while productItem can be only 1 product and I add quantity field.
      2. productItem relation with order is: productItem can be in 1 order while order can have many productItem.

* In database there is a parent and child in any relationship, meaning that the child can’t exist without parent.
  + The parent is when I say order can have(parent) many items(child)
  + User(parent) can have only one email(child).
* The relation should be added in the child model.
* On\_delete=Option have 4 values:
  + Models.CASCADE : means when parent deleted, the child should be deleted as well.
  + Models.SET\_NULL: means don’t delete child and set parent related field to null.
  + Models.SET\_DEFAULT: when parent deleted set value to default.
  + Models.PROTECT: means don’t allow parent delete when child is protected.
* The delete option depends on the requirements.

**Circular dependency**

* It occurs when I have two relations between two models.
* Django create a reverse relation, so when I add another relation, the name will be doublicate
* They become dependent on each other.
* To solve it I add relation to parent class and set behavior to SET\_NULL because I don’t want to end up deleting the model.
* And make it nullable to make the value of the field null.
* Give it related\_name = “+” to prevent reverse or related\_name = “name different than the name stored in the other class”.

**Generic relation**

* To make optionally needed apps minimum coupled and highly cohesion in Django I use generic relation.
* If I don’t use this method, I will have to import the parent model in the optional app and become depending on it.
* The generic way provided by ContentType app that is installed when Django project is created.
* It can be used to identify the models I want to add in a generic way.

HOW TO IMPLEMENT?

* First go to optionally used module.
* from Django.contrib.ContentTypes.models Import models and from Django.contrib.ContentTypes.fields import GenericForiengKey
* then define the model
* added fields to model:
  + the relation of the parent class.
  + Content\_type = models.ForeignKey(ContentType, on\_delete=option)
  + Id = PositiveIntengerField(). // the limitation of the generic type is that it assumes that the id is integer.
  + Content\_object = the actual content of the object