Python in Web Development

Recap

- Request / Response
- Django Templates

LAB 8

Django Models and friends

AGENDA

- Database 101: relations, transactions
- Django Models and Managers
- Schema & Data Changes

Database 101

- Field types: TextField, Charfield, Datefield…
- Relations
  - Keys: primary, foreign, surrogate, natural
  - OneToMany (ForeignKey)
  - OneToOne
  - ManyToMany
- Transactions
  - Why you need them
- Engines: PostgreSQL vs MySQL vs JSON

ORM Advantages

- Advantages:
  - portable across db engines
  - can create python expressions
  - security against SQL injection
  - no longer required to write SQL
  - migrations
  - Traceability in code and versioning
  - Can go forwards and/or backwards in migrations

ORM Disadvantages

- Disadvantages:
  - not so scalable (if you're google, pinterest...)
  - adds a bit of overhead -> complexity
  - for optimal performance you do need to look at SQL queries from the ORM

Django Models

- Object mappings to database tables
- Easy to manipulate with the Django ORM
- Model instances can/know to talk to the DB
  - CRUD: Create, Read, Update, Delete
  - .save() , update(), .delete(), .filter()
  - Verifies constraints

- Just inherit from django.db.models.Model to inject know-how
  - Metaclasses all the way down

Example

```
# models.py

class CartItem(models.Model):

    item_name = models.CharField(max_length=50)
    Item_quantity = models.IntegerField(default=0)

     def __unicode__(self):
         return '{}x{}'.format(self.item_name, self.item_quantity)

mycart = CartItem.objects.get(pk=42)
mycart.item_name = 'Gaming Mouse'
mycart.save()
```

QuerySet and Model Managers

- Default is the *objects* attribute under Model
- For operations across multiple rows
- Custom code for collection logic should be in custom manger: ex. public blog entries
- Managers delegate almost all calls to querysets
- QuerySet is a composable lazy evaluated expression
  - can chain expressions: Q and F objects
  - Minilanguage: .filter(field1__<field2>__...=)
- Methods: filter and get, exists, all, count, values, update, delete

Django Migrations

- Two kinds: Schema and Data Migrations

1. Schema Migration
   - What to do when you need to add a new field to a table?
     - Alter table add column item_price: NO!!!
     - You write a schema migration
2. Data Migration
   - When you need to handle changes in data

Django Migrations

- Run
  - python manage.py showmigrations
  - python manage.py makemigrations
  - python manage.py migrate [app_name] [0007]

- To go backwards you specify a previous migration number
- No need to worry if you modified the schema when changing environments!
- When writing data-migrations always provide a backwards migration code (strongly advised).

ORM - Cheatsheet

- ModelName.objects.get(expression)
- ModelName.objects.filter(expression)
- instance = ModelName(**kwargs)
  - instance.save()
- instance.delete()

```
class MyModel(models.Model):

    field = models.CharField(max_length=50)
    ...
```

Workshop

- models: Question, Choice, Poll
- Question fields:
  - question_text (max 200)
  - pub_date for publication date
- Choice fields:
  - question relation to Question
  - choice_text (max 200)
  - votes
- Poll fields:
  - name (max 200)
  - questions relation to Question

TODO: update models.py and makemigrations

Quiz time :)

https://goo.gl/forms/OVlrIc65MT8LOX1Y2

Thank you!