CS471 - Web Technologies

HTML Fundamentals IV

Semester: 452

Lecture: 7

Outline

- More Form tags
- Django models

<select> tag

Demo
Next

- Another form tag for selection from a list of options.

```
<select name="selection">
  <option selected>None</option>
  <option value="select1">Selection 1</option>
  ...
</select>
```

Example:

```
<select name="selectedgenre">
  <option selected>None</option>
  <option value="fiction">Fiction</option>
  <option value="n-fiction">Non-fiction</option>
  <option value="novel">Novel</option>
</select>
```

<select> tag

Demo
Next

- To get the value of the input field, we do the following in action view

```
def search(request):
    ...
    choice = request.POST.get('selectedgenre')
```

<textarea> tag

Demo
Next

- Provide an input for multi-line text or sting
- Attributes for number of columns and rows can be supplied

```
<textarea name="elementname" rows="XX" cols="YY"></textarea>
```

For example:

```
<textarea name="description" rows="10" cols="30"></textarea>
```

Django models and migrations

- We covered V (view) and T (template) in MVT (MVC) paradigm
- Remember, you can link most of databases with you Django
    - By default, Django links your project with SQLite

- Models like images (abstract) of the structure of the database
- Django uses models and allow object-relational mapping (ORM) to interact with your tables in database

- settings.py have your database configuration

Django models and migrations

- Using Django and models, we can CRUD
  - Create, read, update and delete records in your database
  - In Django, records are referred to as objects
- For every Django app, you have a separated models.py
  - Use a class for a table
  - Use attributes for fields in the table
  - For fields constraints, use models helping functions
    - There are 3 categories
      - Field type
      - Option
      - Relationships: foreign key, many to many

Django models and migrations

- Assume we have to represent the data of book as follows

ID	Title	Authors	Price	Edition
1	Continuous Delivery	J.Humble and D. Farley	120.00	3
2	Reversing: Secrets of Reverse Engineer	E. Eilam	97.00	2
3	The Hundred-Page Machine Learning Book	Andriy Burkov	100.00	4

Here we have:

ID as integer (this is done automatically by Django)

Title as text

Authors as test

Price as real

Edition: as integer

Django models and migrations (cont.d)

Demo Next

- In models.py of bookmodule, we add this:

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length = 50)
    author = models.CharField(max_length = 50)
    price = models.FloatField(default = 0.0)
    edition = models.SmallIntegerField(default = 1)
```

Django models and migrations (cont.d)

Demo Next

- Now we will use manage.py to do the creation of that table in the database

First, make necessary migrations

```
python manage.py makemigrations
```

Then, apply the migrations to the database

```
python manage.py migrate
```

If we open the database ‘db.sqlite3’ file in ‘VS code’ or ‘DB Browse for SQLite’ apps you can see the table.

Django models and migrations (cont.d)

Demo Next

- To insert values into the table we do:
  - Use the constructor function

```
mybook = Book(title = 'Continuous Delivery',
author = 'J.Humble and D. Farley', edition = 1)
```

  - Use the create function

```
mybook = Book.objects.create(title = 'Continuous Delivery',
author = 'J.Humble and D. Farley', edition = 1)
```

- You must save the object once created

```
mybook.save()
```

- Note that the ID fields is automatically assigned by Django incrementally, if 'DEFAULT_AUTO_FIELD' variable is set in the settings.py

Use random data generator

Demo Next

- For development, you would need data (a lot of data)
  - You can use those websites that generate data for you
- One of them is www.mockaroo.com
- You can export the data to any file you want, I recommend SQL
- Then, import that data into your database
  - Of course, you need to create the schema and all your tables
  - Then do:
    - Open db.sqlite3
    - Go the execute SQL window
    - Paste the content of SQL downloaded file into the windows
    - Hit execute

Django models and migrations (cont.d)

- To retrieve an object from the database we use either
  - get() function: throws an exception if the record is not found
  - filter() function: returns a QuerySet (zero or more objects)

```
mybook=Book.objects.get(title = 'Continuous Delivery') # <-Single object

print(f"author of {mybook.title} is {mybook.author}")

# or

mybooks=Book.objects.filter(title__icontains='and') # <- multiple objects

for obj in mybooks:

    print(f"author of {obj.title} is {obj.author}")
```

QuerySet lookups

Demo Next

- We can use the following keywords with fields names to make conditions

exact	`User.objects.filter(name__exact = 'Khalid')` `# or just equal =`
contains	`Room.objects.filter(color__contains='green')`
isnull	`Student.objects.filter(grade__isnull = True)` `# or False`
startswith, endswith	`Book.objects.filter(title__startswith = 'T')`
gt (greater than) gte (greater or equal) lt (less than) lte (less than or equal)	`Student.objects.filter(grade__gte = 60)`

QuerySet lookups (cont.d)

- For extra filtering to the queryset, we can attach multiple filtering use either
  - filter()
  - exclude() # a function that does the opposite of filter()

Ex1:

```
books = Book.objects.filter(price__lte = 100).filter(edition__gte = 2)
```

Ex2:

```
books=Book.objects.filter(title__icontains='testing').exclude(price__lte = 100)
```

QuerySet lookups (cont.d)

- QuerySet can be limited/offsetted using a python subset array slicing
- Use [startindex:endindex]

```
# return objects of indices between 2 and 10 (inclusive)

books = Book.objects.filter(price__lte = 100)[2:10]
```

- You can also specificity only the start index and/or end index

```
books = Book.objects.filter(price__lte = 100)[2:] # objects from 2 until end of
queryset

books = Book.objects.filter(price__lte = 100)[:6] # objects from 0 until 6
```