CSCI E-33a

CS50’s Web Programming with Python and JavaScript

Spring 2020

Barbara Karakyriakou · Teaching Fellow

- Email: karakyriakou@fas.harvard.edu
- Phone: 617 820 6930
- Section Meetings: Wednesdays 8:30pm-10:00pm EST
- Office Hours: Saturdays 1:00pm - 2:30pm EST

Section 3: SQL, Models, and Migrations

Agenda

- SQL
- Models
  - ➢RELATIONSHIPS – ForeignKey: many-to-one relationship
  - ➢RELATIONSHIPS – ManyToManyField: Many-to-many relationship
- Migrations
- Problem Set 2 Instructions

SQL

- **SQL** (pronounced "ess-que-el") stands for Structured Query Language. **SQL** is used to communicate with a database
- The standard SQL commands are "Select", "Insert", "Update", "Delete", "Create", and "Drop"

Models

- A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you’re storing. Generally, each model maps to a single database table
- Each model is a Python class that subclasses django.db.models.Model
- Each attribute of the model represents a database field.

Model Example

```
from django.db import models

class Student(models.Model):
      first_name = models.CharField(max_length=64)
      last_name = models.CharField(max_length=64)
      dob = models.DateField()

class StudentIDs(models.Model):
      student = models.ForeignKey(Student, on_delete=models.CASCADE)
       huid = models.IntegerField()
```

Relationships

ForeignKey

```
class ForeignKey(to, on_delete, **options)
```

A many-to-one relationship. Requires two positional arguments: the class to which the model is related and the on_delete option.

To create a recursive relationship – an object that has a many-to-one relationship with itself use

```
models.ForeignKey('self', on_delete=models.CASCADE)
```

Relationships

Many-to-many relationships

To define a many-to-many relationship, use ManyToManyField. You use it just like any other Field type: by including it as a class attribute of your model.

ManyToManyField requires a positional argument: the class to which the model is related.

For example, a Student takes multiple courses, but at the same time each course has multiple students—how would you represent that?

Many-to-many Relationship Example

```
from django.db import models

class Student(models.Model):

        # ...

        pass

class Course(models.Model):

        # ...

        students = models.ManyToManyField(Student)
```

Many-to-many Relationship Hints

- As with ForeignKey, you can also create recursive relationships (an object with a many-to-many relationship to itself) and relationships to models not yet defined.
- It’s suggested, but not required, that the name of a ManyToManyField (toppings in the example above) be a plural describing the set of related model objects.
- It doesn’t matter which model has the ManyToManyField, but you should only put it in one of the models – not both.

Migrations

- Migrations are Django’s way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema.
- They’re designed to be mostly automatic, but you’ll need to know when to make migrations, when to run them, and the common problems you might run into.
- The migration files for each app live in a “migrations” directory inside of that app, and are designed to be committed to, and distributed as part of its codebase.

Migrations Commands

There are several commands which you will use to interact with migrations

and Django's handling of database schema:

`migrate` responsible for applying and un-applying migrations.

`makemigrations` responsible for creating new migrations based on the

changes you have made to your models.

`sqlmigrate` displays the SQL statements for a migration.

`showmigrations` lists a project's migrations and their status.

Migrations Hints

- `makemigrations` is responsible for packaging up your model changes into individual migration files - analogous to commits
- `migrate` is responsible for applying those migrations to your database.

Project 2

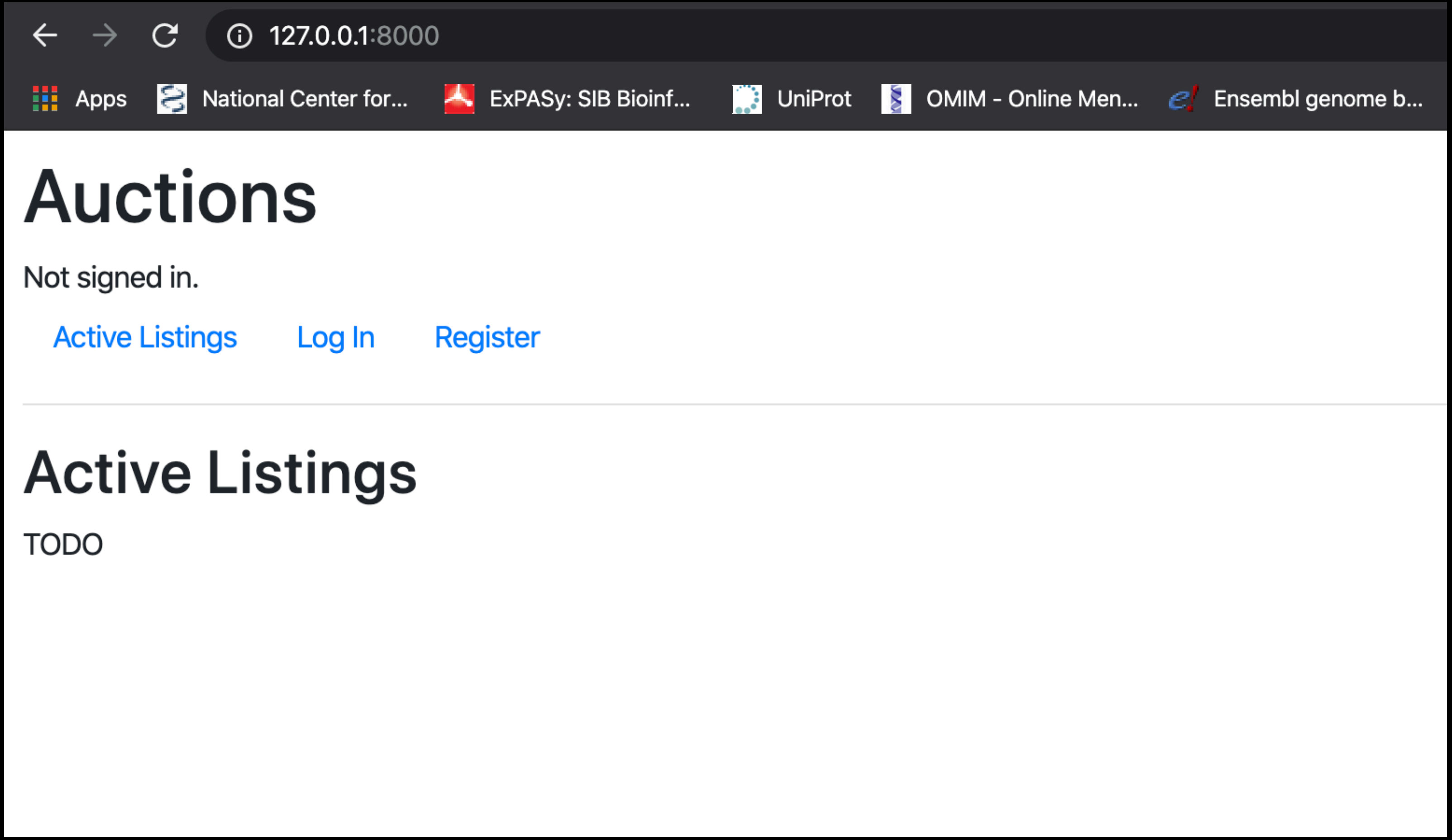
Commerce

- Download the project zip folder and unzip it
- Move the project to a location that you have easy access through your terminal
- Once you open the folder you will find two subfolders, auctions, commerce, and a manage.py file
- Run `python(3) manage.py makemigrations auctions` to make migrations for the auctions app.
- Run `python(3) manage.py migrate` to apply migrations to your database.

Project 2

Next steps

- Start up the Django web server: `python manage.py runserver`


127.0.0.1:8000

Auctions

Not signed in.

Active Listings Log In Register

Active Listings

TODO

Project 2

TO DO

models.py

Add three models in addition the User model that already exists.

- auction listings

```
class Listing(models.Model):
```

    (here you will add fields)

- bids

```
class Bid(models.Model):
```

    (here you will add fields)

- comments (made on auction listings)

```
class Comment(models.Model):
```

    (here you will add fields)

Project 2

TO DO

models.py

You may also want to add a model for the Categories requirement

- categories (list of all listing categories)

```
class Category(models.Model):
```

(here you will add fields)

Project 2

TO DO

views.py

Import your new models to your views file!

```
      from .models import User, Bid, Listing, etc.
```

Next create views

- Functions in your views.py file
- Paths in your urls.py
- Add the necessary html files in your templates folder

Project 2

TO DO

views.py

- A function for viewing a listing: `def listing(request, listing_id):`
- A function to place a bid: `def bid(request, listing_id):`
- A function to create a listing: `def create(request):`
- A function for users' viewing their watchlist: `def watchlist(request):`
- A function to add to a watchlist: `def add_watchlist(request):`
- A function to remove from a watchlist: `def remove_watchlist(request):`

Project 2

TO DO

Templates

- index.html (modify)
- categories.html
- create.html
- listing.html

https://docs.djangoproject.com/en/3.0/topics/forms/

https://docs.djangoproject.com/en/3.0/topics/forms/modelforms/#modelform

Project 2

TO DO

admin.py

```
from .models import Bid, Category, Comment, Listing

admin.site.register(Bid)
admin.site.register(Category)
admin.site.register(Comment)
admin.site.register(Listing)
```

Project 2

TO DO

- **Django Admin Interface**: Via the Django admin interface, a site administrator should be able to view, add, edit, and delete any listings, comments, and bids made on the site.

To create a superuser run:

```
$python(3) manage.py createsuperuser
```

The super user would then be able to login and access the admin site at:
http://127.0.0.1:8000/admin/

Q&A