Design: Service Layer

SWDV-691 Chris Patterson

I decided to use the AWS version of MySQL for the database. Since I'm not an experienced web developer, I wanted to use a backend that was a little more comfortable for me. I settled on Django as the backend service provider. Django has a built in MySQL connector. Working through these routes, I've found problems in the db that I will need to go back and correct as well.

Here's what I have on urls.py:

```
from django.contrib import admin
from django.urls import path, include
from . import views

urlpatterns = [
    path('', views.nologin_home),
    path('register/', views.register),
    path('users/<int:id>', views.user),
    path('tools/<int:id>', views.tool),
    path('search_by_name/<str:name>', views.search_by_name),
    path('create_user/', views.create_user),
    path('tool_form/', views.tool_form),
    path('preview_tool/<int:id>', views.preview_tool),
    path('create_tool/', views.create_tool),
    path('contact/<int:id>', views.contact),
    path('search/', views.search),
    path('borrow_tool/', views.borrow_tool),
    path('rate_user/<int:id>', views.rate_user) # stretch
]
```

Here's what I have on views:

```
from django.shortcuts import render

# Create your views here.

# might be able to replace these two with built-in
#####
def nologin_home(request):
    # The home page for users not logged in
    return render(request, 'diyexch_app/login.html' )
#####


# db queries

def create_user(request):
    # ...insert user in the db
    return render(request, 'diyexch_app/success.html', other_stuff)
```

```python
# @login_required
def view_account(request, id):
    # ... GET account info from db
    return render(request, 'diyexch_app/account.html', other_stuff)


# @login_required
def user(request, id):
    if request.method == 'GET':
        # get the user
        return render(request, 'diyexch_app/user_home.html', other_stuff)

    elif request.method == 'DELETE':
        # delete the user
        return render(request, 'diyexch_app/success.html', other_stuff)

    elif request.method == 'PUT':
        # updatr the user
        return render(request, 'diyexch_app/success.html', other_stuff)


# @login_required
def tool(request, id):

    if request.method == 'GET':
        # get the tool from db
        return render(request, 'diyexch_app/tool_home.html')

    elif request.method == 'DELETE':
        # delete the tool
        return render(request, 'diyexch_app/success.html')

    elif request.method == 'PUT':
        # update the tool
        return render(request, 'diyexch_app/success.html')


# @login_required
def create_tool(request,):
    # ... insert the tool into the db
    return render(request, 'diyexch_app/success.html')


# @login_required
def borrow_tool(request, tool_id):
    # logic to update the db
```

```
    return render(request, 'diyexch_app/success.html')


# @login_required
def search_by_name(request, name):
    # ... lookup list by name, return the search results
    return render(request, 'diyexch_app/search.html', {})


# @login_required
def contact(request, id):
    # contact the user
    return render(request, 'diyexch_app/conact.html', other_stuff)


# @login_required
def search(request):
    # provide some random preview tools on the initial search (filler)
    return render(request, 'diyexch_app/search.html', {})


# strictly page routes

def register(request):
    # ...register a user
    return render(request, 'diyexch_app/user_form.html' )


# @login_required
def tool_form(request):
    # displays the form used to create a new tool
    return render(request, 'diyexch_app/tool_form.html', {})


# @login_required
def preview_tool(request, other_stuff):
    # displays the preview of a tool before keeping it
    return render(request, 'diyexch_app/tool_preview.html', {})
```

I'm still getting familiar with the Django framework, so I know the CRUD and page routing will change as I begin to implement the project.

What I did as to make dummy HTML pages to represent the real pages in my MVP. Then I added routing urls for the operations and set up a function in views to handle each one of them. Obviously the functions are not logic complete, but I know where they will be going to.

For reference, object data models:

```python
from django.db import models
from django.core.validators import MaxValueValidator, MinValueValidator

# Create your models here.

class User(models.Model):
    email = models.CharField(max_length=255)
    username = models.CharField(max_length=100)
    cc_fullname = models.CharField(max_length=255, null=True)
    cc_number = models.CharField(max_length=22, null=True)
    cc_exp = models.CharField(max_length=5, null=True)
    zip_code = models.CharField(max_length=10)
    phone = models.CharField(max_length=16)
    img_url = models.CharField(max_length=255, null=True)
    tac_agree = models.BooleanField(blank=False)


class Tool(models.Model):
    ownerID = models.ForeignKey(User, on_delete=models.CASCADE)
    tool_value = models.DecimalField(max_digits=10, decimal_places=2)
    for_sale = models.BooleanField()
    description = models.CharField(max_length=500)
    img_url = models.CharField(max_length=255, null=True)
    name = models.CharField(max_length=255)
    visible = models.BooleanField(default=True)


class Borrow_tx(models.Model):
    # Use "delete" code, logic, or SP to make sure borrower cannot be deleted while borrowing tool,
    # make sure that tool cannot be deleted while borrowed
    toolID = models.ForeignKey(Tool, on_delete=models.RESTRICT)
    borrowerID = models.IntegerField()
    timestamp = models.DateTimeField()
    returned = models.BooleanField(default=False)
    # stretch
    rating_from_owner = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)])
    rating_from_borrower = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)])
```

As an example, looking up a user by their ID number so their home page can be rendered:

```
path('users/<int:id>', views.user)
```

`GET method -> HTTP://mysite.com/users/2`

This would call:

```
def user(request, id):
    if request.method == 'GET':
        try:
            user_data = UserModel.objects.get(id)
        except UserModel.DoesNotExist:
```

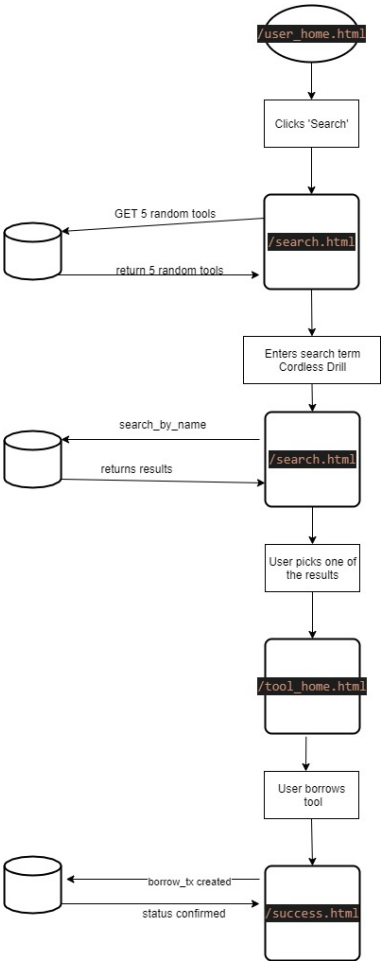
raise Http404('Data does not exist')

return render(request, 'diyexch_app/user_home.html', user_data)

The db logic isn't coded, but it would then (SELECT * FROM user WHERE id=2) and return a User object whose data can be sent to the /user_home.html page and rendered.

If the user ID wasn’t found, it would raise a 404 error.

Flow example – User Borrows a Tool:



A user decides they need a tool.

From their home page, they click on Search.

They are directed to the search page, which incorporates five random tool results as background filler.

The user wants a different tool, so searches by name, "Cordless Drill".

As search_by_name is called, it returns the matching result list.

The results are now displayed in the search page rather than the filler.

The user clicks on a tool and is taken to the tool's home page.

The user decides to borrow that tool.

A borrow_tx is created and the tool status is updated in the db.

On success, the user is taken to the success page.

Paths and Notes			
Path	View	Page	Notes
''	nologin_home	/login.html or /user_home.html	For users who are not logged in, takes them to login. Otherwise, user_home
'register/'	register	/user_form.html	Lets a user register to use app
'users/<int:id>'	user	/user_home.html or /success.html	Depends on the HTTP Req logic of GET, PUT, or DELETE
'tools/<int:id>'	tool	/tool_home.html or /success.html	Depends on the HTTP Req logic of GET, PUT, or DELETE
'search_by_name/<str:name>'	search_by_name	/search.html	Looks up matches from provided string, returns result in search page
'create_user/'	create_user	/success.html	Inserts user into db, displays success page on success
'tool_form/'	tool_form	/tool_form.html	Form to enter new tool info
preview_tool/<int:id>'	preview_tool	/tool_preview.html	Displays tool for user acceptance prior to final db commit
'create_tool/'	create_tool	/success.html	Inserts new tool into db, displays success page on success
'contact/ <int:id> '	contact	/conact.html	Contact form populated from the lookup of provided user ID
'search/'	search	/search.html	Route to "first time" search page with random items loaded as filler until user can perform search.
'borrow_tool/'	borrow_tool	/success.html	Inserts new borrow_tx in db, displays success page on success
'rate_user/<int:id>' (stretch)	user	/success.html	Update ('PUT') new data to borrow_tx and user rating in db