Testing Notes

1. Django Tests

Django Tests is based on unittest. Shares the same syntax but extends the functionality into as below

1. Allows you to simulate a request to see what response is returned by a particular endpoint. Also allows you to check (a) status code of endpoint (b) context (c) what content is actually being returned

Note: When you are creating tests don’t need to be exhaustive. The idea is that when bugs are discovered, you will add an extra test to make sure that that bug will never turn up again

1. Selenium

Django Tests does not have the ability to test code that occurs on the client side. I.E. Any event handler that is attached to HTML elements cannot be checked whether that interactivity is correct.

Common way to conduct browser testing is via Selenium. Selenium can simulate a web browser and a user interacting with that browser via a web driver. A web driver allows a developer to programmatically control what goes on to a browser.

1. Selenium Set-Up

Each browser requires a special web driver to be installed. Please download these from the Selenium download page.

1. Selenium Commands

**File\_uri(uriName)**

Selenium needs a page’s URI in order to test it. Simple use the above method to get it.

**Driver.get(uri)**

This would tell the driver to actually go to that page

**driver.<htmlElement> and driver.page\_source**

Once you have the uri attached, then you can access any part of the DOM simply by just dot notation.

**Driver.find\_element\_by\_id(“increase)**

This works too to find HTML elements. You can assign this element to a python variable. If it is a button you can use increase.click().

1. CI/CD

Continuous Integration

* Frequent merges to main branch
  + Catch and resolve conflicts quickly
* Automated unit testing
  + Make sure each component – like function by function – works well.

Continuous Delivery

* Short release schedule
  + Incremental changes means faster time to market plus catching bugs earlier

1. Github Actions

Allows for certain actions to automatically trigger once something has been merged to a repo. I.E. A style check for code or running unit tests and having an email sent if they fail

Github actions is written in YAML. YAMlL has the following:

1. Key-value pairs
2. Arrays
3. Nesting between key and values i.e. within 1 key you can have 1 array which was values
4. Docker

Docker helps to standardize environments across servers and development machines to ensure that application actually works.

Docker creates a container where each container contains its own configurations. Applications are ran inside these containers instead on separate developer machines.

Docker is similar to a VM however a VM runs an entirely new OS while docker containers run on Host OS but has a layer in between applications and OS. Hence, they are lighter weight than VMs

Docker files describe instructions to create a docker image. Docker image contains all the configurations you want the container to have.

A common problem docker fixes is the issue of having a separate database. Databases in production are hosted on their own server such as in POSTGRSQL or MYSQL. However, your development database might be SQLite. Docker fixes this by having separate containers for your web application and your database.

Docker compose allows us to compose various service. I.E. allow 2 containers one running Postgres another running the web application to talk to one another

**Project Notes**

1. New Post

Index route should show a form that allows user to write text-base post. This post should contain 2 things (i) textarea for post content (ii) submit button with name post

Index route should also display the author’s post below

After writing a new post, then user should be redirected to their home page. Which should contain a feed of all their post

Tests required

1. Unit test on Django Model Method create\_post to ensure that tests are created
2. Integration test with Client to show that route is working and correct template is used with get request
3. Integration test with Client to show that form can successfully post to route and save data to model.
4. All Posts

New template where all posts should be shown in chronological order with most recent first. Each post should contain (i) username of poster (ii) post content (iii) date and time of post (iv) number of likes the post has which defaults to 0

Must apply pagination here as well

1. Username

Username should lead to profile page. Need the following information

* Number of followers of the user, number of people the user follows
* All posts in reverse chronological order i.e. most recent first
* IF current user is not host of page, have follow or unfollow button
  + Means require logic to determine whether (i) user is signed in (ii) user is viewing his own profile

1. Following

Same as All Posts but they can only see posts by users that the current user follows. Apply pagination here

1. Edit Post

Data Model

1. Tables to have are (i) User (ii) Post
2. User should contain the following
   1. Followers -> Many to many relationship. This should be self-referencing. A single user can have many followers. A follower can have many following.
   2. Following -> Same as above
   3. Check this
3. Posts
   1. Author -> User\_ID foreign key
   2. Post content -> Text
   3. Timestamp -> Date and Time
4. Likes
   1. User\_Id
   2. Post\_id
   3. Timestamp

APIs required

To send likes and unlike

To send follow or unfollow

OOP Design

1. User Object
   1. Given a post should like that post
   2. Given a post unlike that post
   3. Given another user, follow that user
   4. Given a user unfollower that user
2. Post object

Django Concepts

1. Making queries

QuerySet -> Reprsents a collection of objects from database. It can contain filters which narrow down the query results based on the given parameters. QuerySets roughly correspond to the SQL keyword SELECT and a filter is a limiting clause such as WHERE or LIMIT

Manager -> Every model has 1 manager and it is called objects by default. A manager is the main source of QuerySets for a model. They are basically an interface which database query operations are provided to Django models.

Queries therefore start like this

modelName.objects.filter(…..)

Example:

**>>>** Entry.objects.filter(

**...**  headline\_\_startswith='What'

**...** ).exclude(

**...**  pub\_date\_\_gte=datetime.date.today()

**...** ).filter(

**...**  pub\_date\_\_gte=datetime.date(2005, 1, 30)

**...** )

**Using get**

You can use the get method on manager if you know that only a single object will match your query. This would return a QuerySet containing a single element.

If there are no results, get will raise a DoesNotExists exception. If more than 1 it will raise a MultipleObjectsReturned

**Django Models**

1. On\_delete
2. Through
3. Related\_name

ForeignKey has a method called related\_name. This name specifies the name to use for the relation from related object back to the one that it is currently defined on. I.E. ForeignKey is defined on a Django model and it is used to create a many\_to\_one relationship with a another table where the model containing the foreign key is the parent. Therefore, using childTable.relatedKeyName would result in a row in the parent table that contains a reference to the child table.

It is also the default valkue for related\_query\_name which is the name to use for thereverse filter name from target model.

1. Get\_or\_create()

A method defined on Manager. Used to lookup an object with given arguments – can be empty wth models has default values for all fields – or can create one if unable to find one.

Returns a tuple of (object, created) where object is the retrieved or created object and created is a Boolean specifying whether a new object was created.

Meant to prevent duplicate objects from being created when requests are made in parallel and shortcut boilerplate code.

1. Create()

This is a method defined on Manager to create a new object, save it and put it on a related object set. Returns a newly created object. I.E. you don’t need to use the word save

**Django Model Form**

1. Validation

2 steps involved in validating ModelForm.

1. Validating the form
2. Validating the model instance

Validation for both is triggered implicitly when calling is\_valid() or accessing the errors attribute and explicitly when calling full\_clean() but using full\_clean() is normally not needed.

Any cleaning method will raise VAlidationError.

**Django Testing Tools**

1. Test Client

Client is a Python class that acts as a dummy Web browser. Allows testing of views and interact with Django powered app programmatically. Use the test client to

* Establish that the correct template is being rendered and the template is passed the correct context data

However, use Selenium to

* Test the rendered HTML and the behaviour of the web page such as their JS functionality.

1. How to Test Forms

Client can simply send post request to selected URL to test forms with information in the context. Then the context will check

1. How to Test Views
2. General Notes

**How does Django urls work?**