**Quantl – Django Development Notes**

**Repository Structure**

quantlpro/

manage.py

quantlpro/

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

MVP/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

urls.py

templates/

Project databased will be stored here

HTML pages & rendering done here

HTML URLs add here (link views.py function to see the HTML pages)

All \*.html files

**Starting Django**

Some key commands for Django

1. Start Django server, navigate to working directory (C:/[web-app-project]). Make sure you’re in a root directory where file manage.py is located

…/> py manage.py runserver

1. To access your page in browser:

<http://localhost:8000/MVP/>

**Django Tutorials Links**

|  |
| --- |
| <https://www.djangoproject.com/> |
| <https://docs.djangoproject.com/en/3.2/intro/tutorial01/> |
| <https://docs.djangoproject.com/en/3.2/> |

**Django Creating a Project (Already done for MVP)**

1. To start a new project we need to auto generate some part of the code

django-admin startproject quantlpro

1. Now navigate to dir (i.e. C/) where project is created and start the server

py manage.py runserver

1. Now create an application within the project  
     
   py manage.py startapp MVP
2. Once the server running use the following link in browser to test

http://127.0.0.1:8000/MVP/

**Database Setup in Django Development Environment**

By default Django uses SQLite however for our application we will be using more scalable database type such as PostgreSQL or MongoDB. In such case: install the appropriate [database bindings](https://docs.djangoproject.com/en/3.2/topics/install/#database-installation) and change the following keys in the [**DATABASES**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-DATABASES) 'default' item to match your database connection settings

* [**ENGINE**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-DATABASE-ENGINE) – Either **'django.db.backends.sqlite3'**, **'django.db.backends.postgresql'**, **'django.db.backends.mysql'**, or **'django.db.backends.oracle'**. Other backends are [also available](https://docs.djangoproject.com/en/3.2/ref/databases/#third-party-notes).
* [**NAME**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-NAME) – The name of your database. If you’re using SQLite, the database will be a file on your computer; in that case, [**NAME**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-NAME) should be the full absolute path, including filename, of that file. The default value, **BASE\_DIR / 'db.sqlite3'**, will store the file in your project directory.

Database Setup documentation

<https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-DATABASES>

**$** python manage.py migrate

The [migrate](https://docs.djangoproject.com/en/3.2/ref/django-admin/#django-admin-migrate) command looks at the [INSTALLED\_APPS](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-INSTALLED_APPS) setting and creates any necessary database tables according to the database settings in your mysite/settings.py file and the database migrations shipped with the app (we’ll cover those later). You’ll see a message for each migration it applies. If you’re interested, run the command-line client for your database and type \dt (PostgreSQL), SHOW TABLES; (MariaDB, MySQL), .schema (SQLite), or SELECT TABLE\_NAME FROM USER\_TABLES; (Oracle) to display the tables Django created.

**Migrations**

Migrations are how Django stores changes to your models (and thus your database schema) - they’re files on disk. You can read the migration for your new model if you like; it’s the file **polls/migrations/0001\_initial.py**. Don’t worry, you’re not expected to read them every time Django makes one, but they’re designed to be human-editable in case you want to manually tweak how Django changes things.

There’s a command that will run the migrations for you and manage your database schema automatically - that’s called [**migrate**](https://docs.djangoproject.com/en/3.2/ref/django-admin/#django-admin-migrate), and we’ll come to it in a moment - but first, let’s see what SQL that migration would run. The [**sqlmigrate**](https://docs.djangoproject.com/en/3.2/ref/django-admin/#django-admin-sqlmigrate) command takes migration names and returns their SQL:

Every time we make a change in our **models.py** and update **settings.py**, we need to run the migrations command to publish our changes so Django updates our model.

1. Change Models. Py
2. When changes made to models.py run migrations command to publish database changes using command:

py manage.py makemigrations polls

1. Once done now run migrate again to create newly defined models in our database

python manage.py migrate

The [**migrate**](https://docs.djangoproject.com/en/3.2/ref/django-admin/#django-admin-migrate) command takes all the migrations that haven’t been applied (Django tracks which ones are applied using a special table in your database called **django\_migrations**) and runs them against your database - essentially, synchronizing the changes you made to your models with the schema in the database.

Migrations are very powerful and let you change your models over time, as you develop your project, without the need to delete your database or tables and make new ones - it specializes in upgrading your database live, without losing data. We’ll cover them in more depth in a later part of the tutorial, but for now, remember the three-step guide to making model changes:

**Annex A: Background**

**Web Development Framework (GUI Interface)**

For development of web application (GUI) we have two options:

1. Flask
2. **Django (ideal for database driven web applications)**

Hosting can either can done on Google App Engine or Platform.sh (see above)

It is decided that we will not be making the MVP locally (i.e. using GUI tools in Tkinter python). MVP will be developed as a Web Application, and we want a solution that is scalable, populator, well supported and documented.

