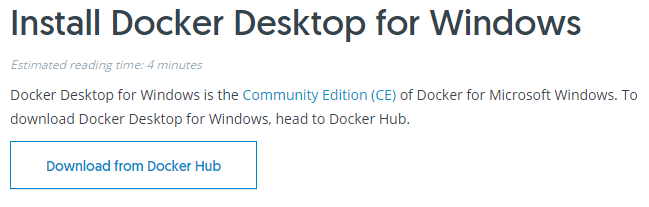
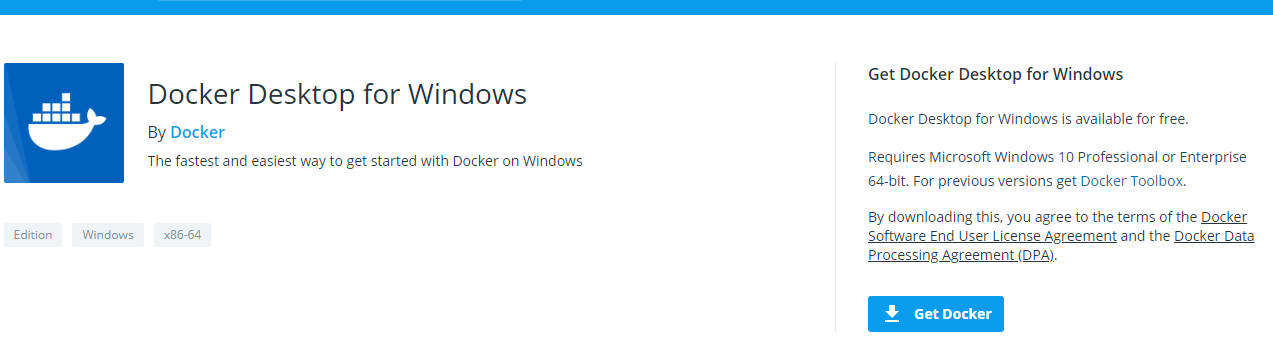
**MOBILE AND INTERNET COMPUTING - SMART IRRIGATION**

**To Download docker desktop for the windows**

* Go to this link(<https://docs.docker.com/docker-for-windows/install/>)



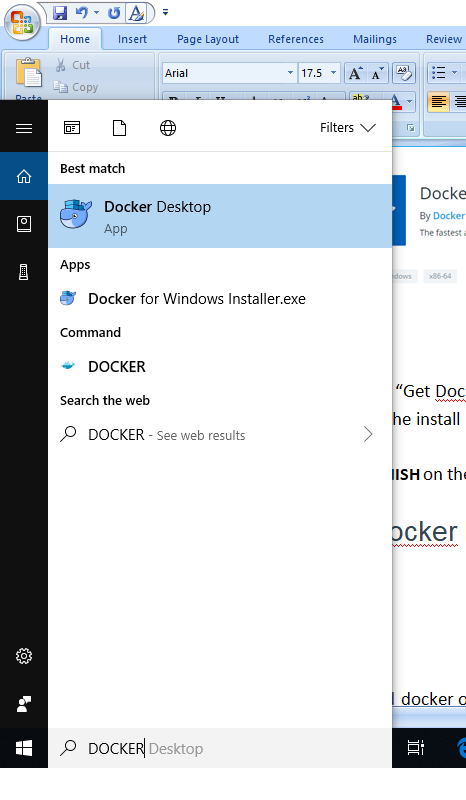
* Click on “Download from Docker Hub”, there you will find



* Click on “Get Docker”. Then docker desktop.exe will be downloaded.
* Follow the install wizard to accept the license, authorize the installer, and proceed with the install.
* Click **FINISH** on the setup complete dialog to launch Docker.

Start Docker Desktop for Windows

* Docker doesn’t start automatically after installation. You need to search for it in search box. Select Docker desktop for windows and click it



* Installed docker on my computer.
* Tested the docker environment.
* Test your installation of docker in command prompt.

1. Open a terminal window (Command Prompt or PowerShell, but not PowerShell ISE).
2. Run docker --version to ensure that you have a supported version of Docker:
3. > docker --version
4. Docker version 18.03.0-ce, build 0520e24
5. Pull the [hello-world image](https://hub.docker.com/r/library/hello-world/) from Docker Hub and run a container:
6. > docker run hello-world
7. docker : Unable to find image 'hello-world:latest' locally
8. ...
9. latest:
10. Pulling from library/hello-world
11. ca4f61b1923c:
12. Pulling fs layer
13. ca4f61b1923c:
14. Download complete
15. ca4f61b1923c:
16. Pull complete
17. Digest: sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1
18. Status: Downloaded newer image for hello-world:latest
19. Hello from Docker!
20. This message shows that your installation appears to be working correctly.
21. ...
22. List the hello-world image that was downloaded from Docker Hub:
23. > docker image ls
24. List the hello-world container (that exited after displaying “Hello from Docker!”):
25. > docker container ls --all
26. Explore the Docker help pages by running some help commands:
27. > docker --help
28. > docker container --help
29. > docker container ls --help

> docker run --help

**DOCKERIZING THE DJANGO APPLICATION AWS DOCKER**

Steps to be followed:

1. Create an empty project directory
2. Create a new file called “Dockerfile” in the project directory
3. Add the following content to the Dockerfile.
4. FROM python:3
5. ENV PYTHONUNBUFFERED 1
6. RUN mkdir /code
7. WORKDIR /code
8. COPY requirements.txt /code/
9. RUN pip install -r requirements.txt
10. COPY . /code/

4) Create a requirements.txt in your project directory.

5) Add the required software in the file.

Django>=2.0,<3.0

psycopg2>=2.7,<3.0

**6)** sudo apt-get install virtualenv python3-pip

7) pip install --upgrade virtualenv

8) virtualenv -p python3 .venv

9)source .venv/bin/activate

10) pip install -r config/app/requirements.txt

11) django-admin.py startproject myproject app(after creating the project I copied the contents from the project created on local app)

**DOCKER IMAGE**

*# Install docker and docker-compose*

sudo apt-get install docker docker-compose

*# Add current user to Docker group (so we can run docker not from sudo)*

usermod -aG docker **${**USER**}**

*# Reboot*

sudo reboot

Configure the docker file with following details ( To expose the port to acsess the docker from local web browser)

*# Creating image based on official python3 image*

**FROM** python:3

*# Your contacts, so people blame you afterwards*

**MAINTAINER** Pavel Gasanov <pogasanov@gmail.com>

*# Sets dumping log messages directly to stream instead of buffering*

**ENV** PYTHONUNBUFFERED 1

*# Creating and putting configurations*

**RUN** mkdir /config

**ADD** config/app /config/

*# Installing all python dependencies*

**RUN** pip install -r /config/requirements.txt

*# Open port 8000 to outside world*

**EXPOSE** 8000

*# When container starts, this script will be executed.*

*# Note that it is NOT executed during building*

**CMD** ["sh", "/config/on-container-start.sh"]

*# Creating and putting application inside container*

*# and setting it to working directory (meaning it is going to be default)*

**RUN** mkdir /app

**WORKDIR** /app

**ADD** app /app/

When container is started, it should make migrations and update database accordinly and then start gunicorn server.

Create a file config/app/on-container-start.sh

*# Create migrations based on django models*

python manage.py makemigrations

*# Migrate created migrations to database*

python manage.py migrate

*# Start gunicorn server at port 8000 and keep an eye for app code changes*

*# If changes occur, kill worker and start a new one*

gunicorn --reload myproject.wsgi:application -b 0.0.0.0:8000

Now the project would be running on the local server with the port mentioned

**PYTHON\_DJANGO- BACKEND FRAMEWORK**

The framework used in the backend is Python-Django. The reason why Django is used because of its its ability to deliver high-quality code and transparent writing provides the right platform to customers for business and even for the developers.

* **Fast:** This has been designed in a way to help the developers make an application as fast as possible. From idea, production to release, Django helps in making it both cost effective and efficient. Thus it becomes an ideal solution for developers having a primary focus on deadlines.
* **Fully Loaded:** It works in a way that includes dozens of extras to help with user authentication, site maps, content administration, RSS feeds and much more such things. These aspects help in carrying out the web development process completely.
* **Secure:** When you are doing it in Django, it is ensured that developers don’t commit any mistakes related to security. Some of the common mistakes include SQL injection, cross-site request forgery, clickjacking and cross-site scripting. To manage effectively usernames and passwords, the user authentication system is the key.
* **Scalable:** To meet the heaviest traffic demand, the benefits of Django framework can be seen. Therefore, the busiest sites use this medium to quickly meet the traffic demands.
* **Versatile:** Content management, scientific computing platforms, and even big organizations, all these aspects are very efficiently managed by the use of Django.

**HARDWARE REQUIREMENTS FOR DJANGO**

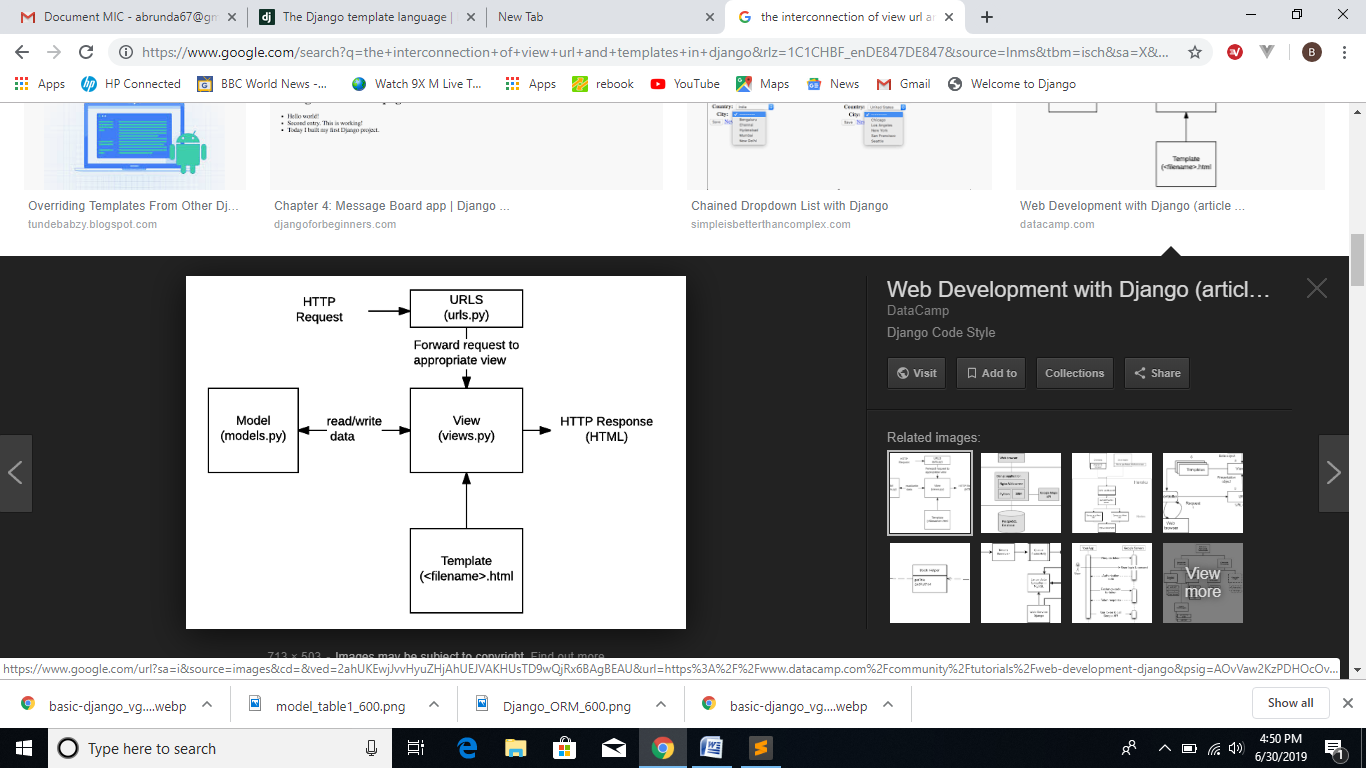
1. Install python latest version
2. Install pip
3. Text editor- Sublime

**STEPS TO CREATE A DJANGO PROJECT**

1. python -V
2. easy\_install pip
3. pip install virtualenv
4. mkdir django\_project
5. cd django15\_project
6. virtualenv env
7. source env/bin/activate
8. pip install django==1.5.12
9. django.get\_version()
10. To create a django project = django-admin.py startproject my\_django15\_project
11. To start the app in Django = python manage.py startapp
12. To start the server python manage.py runserver

**CONFIGURING THE APP CREATED- VIEWS,URLS,TEMPLATES AND MODELS**

The flow of request and response in django is as follows:



**VIEWS**

* They take a request as their first argument
* Django returns a response object which includes the rendered template or text.
* Django has “Class Based Views”.Define a class and that class has a bunch of methods, and those methods interact, and at the end they hand back a response object.
* An incoming request to a piece of code that will began with a route. A route is the path that a request takes to a view. URLs. We’ve seen in our Django project in our site directory our urls.py file. We set up a static URL there to point to our media directory.
* Django creates urlpatterns. These are basically just lists of calls to the url function. url() comes from django.conf.urls and it is responsible for wiring together 2 things:
* The pattern that is being used to match an incoming request.
* The view that will be pointed at.

**TEMPLATES**

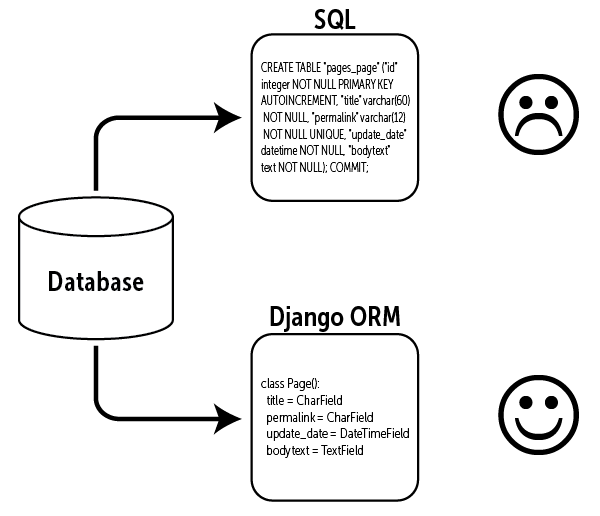
A template is simply a text file. It can generate any text-based format (HTML, XML, CSV, etc.).

A template contains **variables,** which get replaced with values when the template is evaluated, and **tags**, which control the logic of the template.

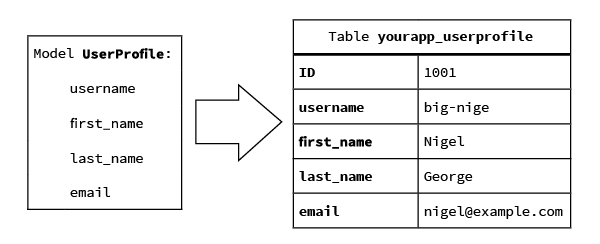
* Create a directory called “templates in the project directory”. Create a html file to display the contents and make the correction in the views.py and the urls.py respectively.

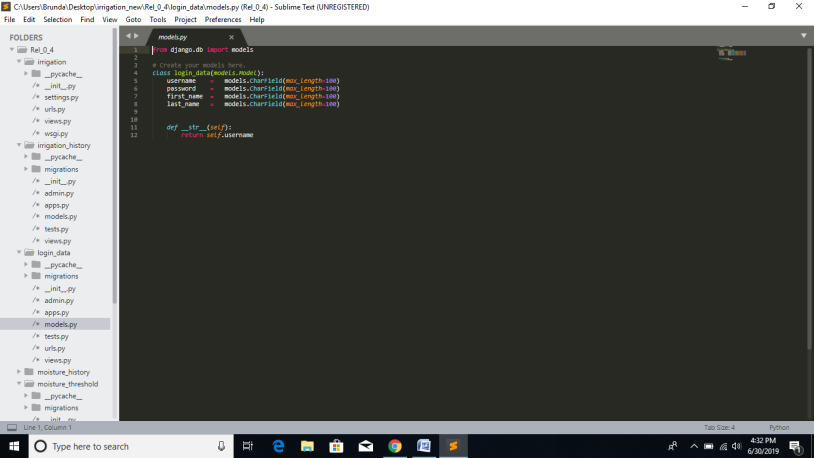
**MODELS**

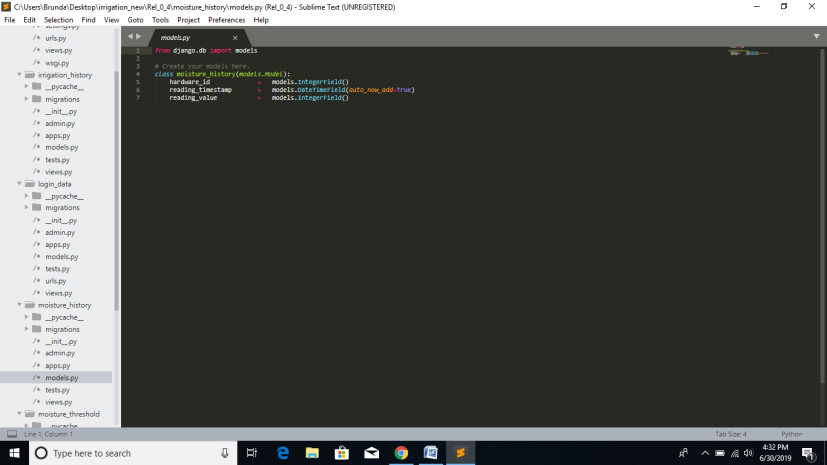
Django uses in-built SQLITE database. Django’s models provide an Object-relational Mapping (ORM) to the underlying database. ORM is a powerful programming technique that makes working with data and relational databases much easier. An ORM tool, on the other hand, provides a simple mapping between an object (the ‘O’ in ORM) and the underlying database.

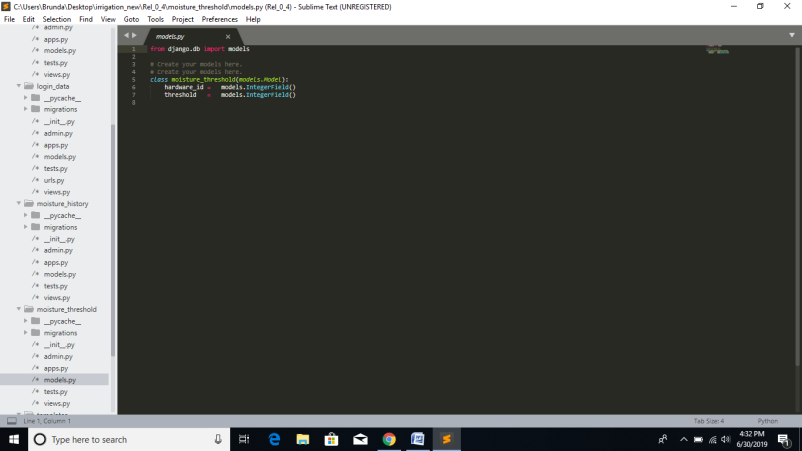


The model also links related information in the database



Models created In our project are as follows for login\_data, irrigation\_history, moisture\_history, 





**How to add models to database?**

Python manage.py makemigrations irrigation

Python manage.py migrate irrigation

Python manage.py sqlmigrate irrigation 0001\_initial (0001\_initial is the file that will be created after the migrations has been applied)

**Accessing the data from the database and entering the values.**

* Python manage.py shell
* From irrigation.models import irrigation (accessing the models from the project)
* From irrigation.models import login\_data
* Login\_data1 = login\_data(username =”brunda”, password = “aralappa”, frist\_name = “brunda”, last\_name = “aralappa”)
* Login\_data1.save()

**Retrieving all the record from the database.**

* Login\_list= login\_data.objects.all()
* Login\_list
* Login\_data.objects.get(username = “brunda”)

The values can also be entred and retrived by the django admin by creating the super user:

Python manage.py createsuperuser