Django is a high-level Python Web framework that encourages rapid development and clean pragmatic design.

A web framework is a server-side application framework which is designed to support the development of dynamic websites. With the help of a framework, you don't have to handle the hassles of web development and its various components. Therefore, it makes the life of a web application developer much easier as they don't have to put extra effort is doing everything from very starting. There are various web development frameworks available in the market. Some of them are listed below:

Ex: React JS, Angular, Ruby on Rails,

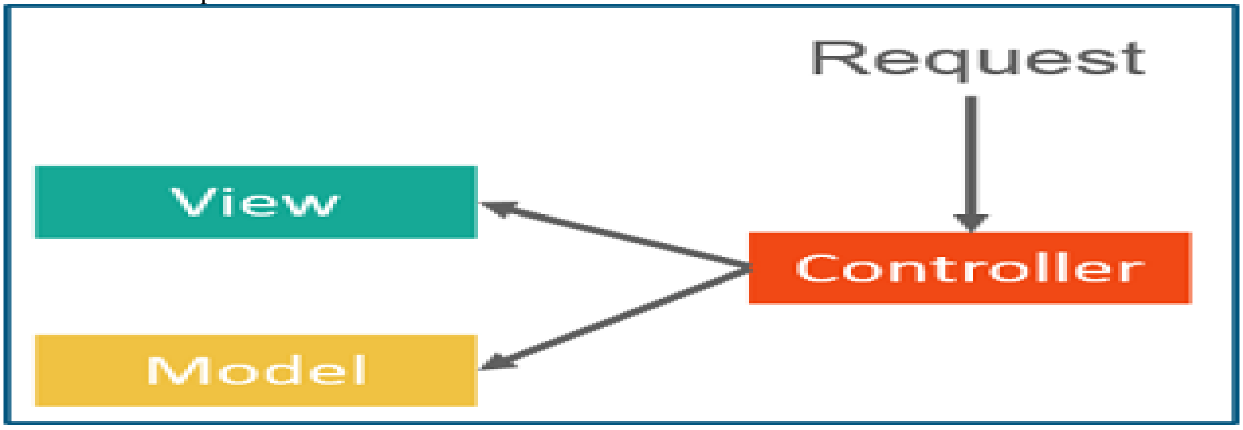
But Django is specifically designed to do coding with Python as its software language.

Django is Used to create website or webservices or API with three-layer structure (Full Stack Development Model)

It helps us in achieving code simplicity and less human intervention on development as well as maintenance.

Django Architecture

Django follows the MVT or Model View Template architecture which is based on the MVC or Model View Controller architecture. The main difference between these two is that Django itself takes care of the controller part.



Model — Model deals with database related activity of storing and retrieving.

Views — Visualizing or GUI part is what we Django terms as Views

Controller — Controller is a business logic which will interact with the model and the view.

Similar to MVC, Python Django deals with MVT.

MVT stands for Model View Template. In MVT, there is a predefined template for user interface. Let's take an example, say you want to write several static html forms like hello netzwerk, hello academy and so on. With template, you will be having only one file that prints hello along with the variable name. Now this variable will be substituted in that particular template using some jinja logic. That's the magic of template, you don't need to rewrite the code again n again!

To create the project: Part 1

1. Open a command shell (or a terminal window)
2. Navigate to where you want to store your Django
3. Create the new project using the **django-admin startproject** command as shown, and then change into the project folder:
4. **django-admin startproject netzwerk**
5. **cd netzwerk**

The django-admin tool creates a folder/file structure as follows:

```
netzwerk/
    manage.py
    netzwerk/
        __init__.py
        settings.py
        urls.py
        wsgi.py
        asgi.py
```

The *netzwerk* project sub-folder is the entry point for the website:

- **__init__.py** is an empty file that instructs Python to treat this directory as a Python package.
- **settings.py** contains all the website settings, including registering any applications we create, the location of our static files, database configuration details, etc.
- **urls.py** defines the site URL-to-view mappings. While this could contain *all* the URL mapping code, it is more common to delegate some of the mappings to particular applications, as you'll see later.
- **wsgi.py** is used to help your Django application communicate with the webserver. You can treat this as boilerplate.

- **asgi.py** is a standard for Python asynchronous web apps and servers to communicate with each other. ASGI is the asynchronous successor to WSGI and provides a standard for both asynchronous and synchronous Python apps (whereas WSGI provided a standard for synchronous apps only). It is backward-compatible with WSGI and supports multiple servers and application frameworks.

The **manage.py** script is used to create applications, work with databases, and start the development web server.

To create the project: Part 2

By default, Django will be provided with default application called "admin". Which can help us login with default login template.

To use the admin login form page, we have to create credentials for the same.

Before creating the username and password for the default admin page, we need to double ensure to do the following

1. **navigate to same netzwerk folder directory where project got created**
2. **python manage.py migrate**

Point 2 will help us in loading all the default configuration created by Django project (netzwerk) into the DB(sqlite3)

Sqlite3 is the default engine selected by Django.

If needed, we can change the database to any engine of wish (oracle or mongo)

DB action via model.py will be carried out by mechanism called "Django ORM"

Which object relation mapping concept similar to sql alchemy.

Once migration is done successfully, please do execute the following command to create the default super user for admin.

3. **Python manage.py createsuperuser**

Point 3 will prompt us to enter "username", "password" and "mailid". Password will be masked on typing.

To Execute both the basic Django and this Admin API we need to run this application as server.

Running the application can be done by following command

4. **Python manage.py runserver**

Above point 4 will run your current application in your localhost 127.0.0.1 with default port number

To keep your application running, double check to maintain this command runs all the time.

By default, Django will run only in localhost(127.0.0.1). If you need to change that into system IP, then find out the IP of your system and update the same in settings.py file.

To create the project: Part 3

Next, run the following command to create the *student* application that will live inside our *netzwerk* project. Make sure to run this command from the same folder as your project's **manage.py**:

```
python manage.py startapp student
```

The updated project directory should now look like this:

```
netzwerk/
    manage.py
    netzwerk/
    student/
        admin.py
        apps.py
        models.py
        tests.py
        views.py
        __init__.py
        migrations/
```

migrations.py file is to deals with updating the configuration created as part of model files

(DB) into DataBases.

__init__.py is used to create class like template to share the data across the layers of the

Django application.

Now that the application has been created, we have to register it with the project so that it will be included when any tools are run (like adding models to the database for example). Applications are registered by adding them to the INSTALLED_APPS list in the project settings.

Open the settings**.py**, and find the definition for the INSTALLED_APPS list. Then add a new line at the end of the list, as shown below:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Add our new application
    'netzwerk.apps.NetzwerkConfig,
]
```

NetzwerkConfig is the class name that got created automatically within apps.py python file during the application creation.

Following were the place where we can configure our DB engine. As described early in this page, Django uses sqlite3 as the default db.

We'll use the SQLite database for this example, because we don't expect to require a lot of concurrent access on a demonstration database, and it requires no additional work to set up! You can see how this database is configured in **settings.py**:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Hooking up the URL mapper

The website is created with a URL mapper file (**urls.py**) in the project folder. While you can use this file to manage all your URL mappings, it is more usual to defer mappings to the associated application.

Open **netzwerk/netzwerk/urls.py** and note the instructional text which explains some of the ways to use the URL mapper.

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

The URL mappings are managed through the urlpatterns variable, which is a Python *list* of path() functions. Each path() function either associates a URL pattern to a *specific view*, which will be displayed when the pattern is matched, or with another list of URL pattern testing code (in this second case, the pattern becomes the "base URL" for patterns defined in the target module). The urlpatterns list initially defines a single function that maps all URLs with the pattern *admin/* to the module admin.site.urls , which contains the Administration application's own URL mapping definitions.

.# Use include() to add paths from the student application

```
from django.urls import include
```

```
urlpatterns += [
    path(student/', include(‘student.urls')),
]
```

Copy to Clipboard

Note: Note that we included the import line (from django.urls import include) with the code that uses it (so it is easy to see what we've added), but it is common to include all your import lines at the top of a Python file.

Now let's redirect the root URL of our site (i.e. 127.0.0.1:8000) to the URL 127.0.0.1:8000/student/. This is the only app we'll be using in this project. To do this, we'll use a special view function, RedirectView, which takes the new relative URL to redirect to (/student/) as its first argument when the URL pattern specified in the path() function is matched (the root URL, in this case).

Add the following lines to the bottom of the file:

```
#Add URL maps to redirect the base URL to our application
from django.views.generic import RedirectView
urlpatterns += [
    path('', RedirectView.as_view(url='student/', permanent=True)),
]
```

Leave the first parameter of the path function empty to imply '/'. If you write the first parameter as '/' Django will give you the following warning when you start the development server:

Django does not serve static files like CSS, JavaScript, and images by default, but it can be useful for the development web server to do so while you're creating your site. As a final addition to this URL mapper, you can enable the serving of static files during development by appending the following lines.

Add the following final block to the bottom of the file now:

```
# Use static() to add url mapping to serve static files during development (only)
from django.conf import settings
from django.conf.urls.static import static

urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
urlpatterns = [
    path('admin/', admin.site.urls),
    path('catalog/', include('catalog.urls')),
    path('', RedirectView.as_view(url='catalog/')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

As a final step, create a file inside your catalog folder called **urls.py**, and add the following text to define the (empty) imported urlpatterns. This is where we'll add our patterns as we build the application.

```
from django.urls import path
from . import views

urlpatterns = [ ‘’,views.index]
```

Views.index is the direct the route to the exact GUI which we will be projecting to the end user.

To project something, when user clicks on the http://127.0.0.1:8000/student. Go to Views.py and create simple HttpResponse with some ordinary text content.

Views.py

```
From django.urls import HttpResponse

def index(request):
   return HttpResponse(“My Views to the User Click”)
```

Before we do run the program, we should first run a *database migration*. This updates our database (to include any models in our installed applications) and removes some build warnings.

Running database migrations

Django uses an Object-Relational-Mapper (ORM) to map model definitions in the Django code to the data structure used by the underlying database. As we change our model definitions, Django tracks the changes and can create database migration scripts (in **/netzwerk/student/migrations/**) to automatically migrate the underlying data structure in the database to match the model.

When we created the website, Django automatically added a number of models for use by the admin section of the site (which we'll look at later). Run the following commands to define tables for those models in the database (make sure you are in the directory that contains **manage.py**):

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Copy to Clipboard

The makemigrations command *creates* (but does not apply) the migrations for all applications installed in your project. You can specify the application name as well to just run a migration for a single project. This gives you a chance to check out the code for these migrations before they are applied. If you're a Django expert, you may choose to tweak them slightly!

The migrate command is what applies the migrations to your database. Django tracks which ones have been added to the current database.

during development to give it a convenient quick test. By default it will serve the site to your

local computer (http://127.0.0.1:8000/), but you can also specify other computers on your network to serve to. For more information see django-admin and manage.py:

runserver (Django docs).

Run the *development web server* by calling the runserver command (in the same directory as **manage.py**):

```
$ python3 manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 01, 2022 - 04:08:45
Django version 4.0.2, using settings 'locallibrary.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Once the server is running, you can view the site by navigating to http://127.0.0.1:8000/ in your local web browser. You should see a site error page.

Advantages:

1. Implemented in python
2. Rapid-Development
3. Secured
4. Better CDN and Content Management
5. Fast and Scalable

Dis-Advantages:

1. Monothic
2. Structure is complex
3. URL’s Mapping