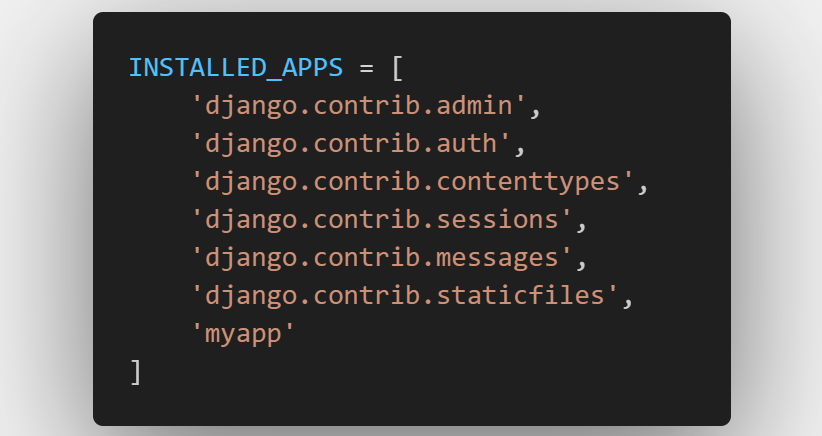
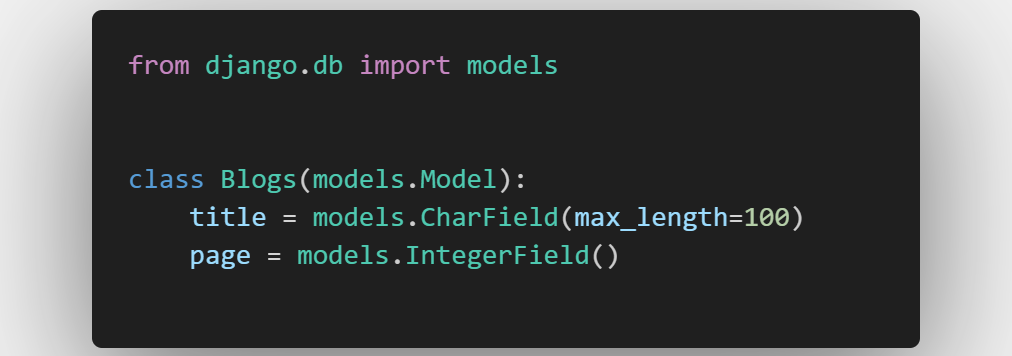
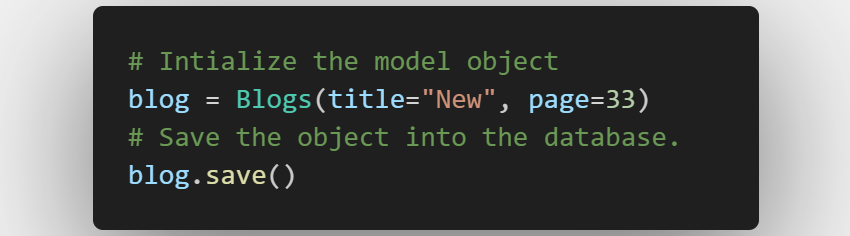
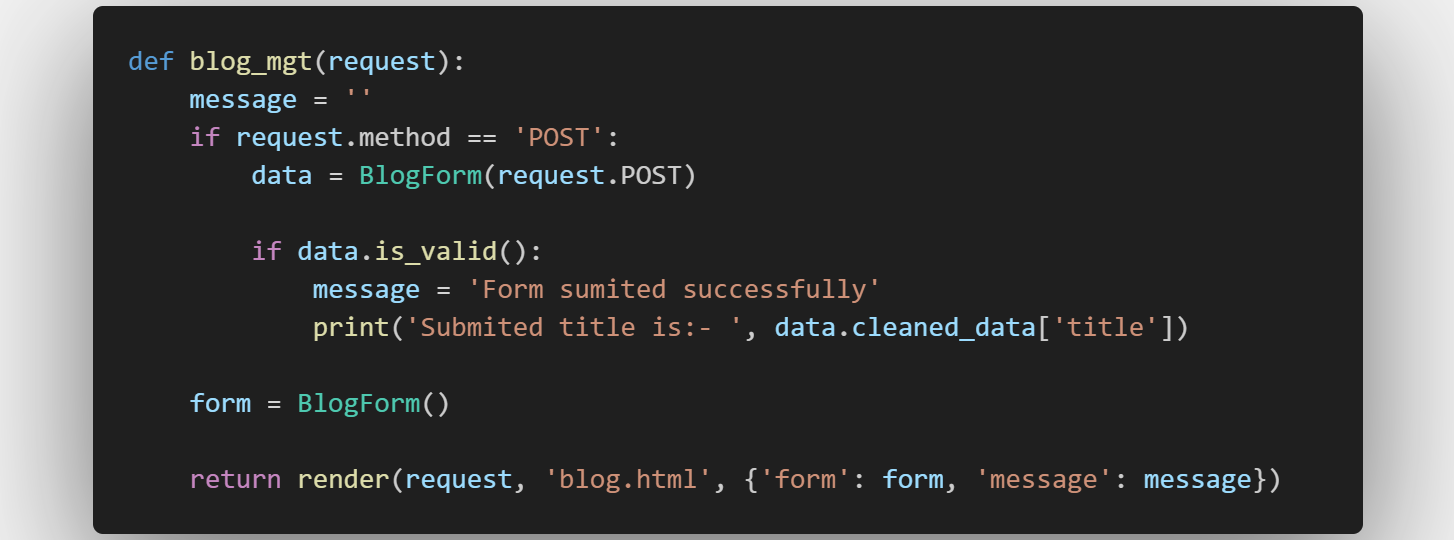
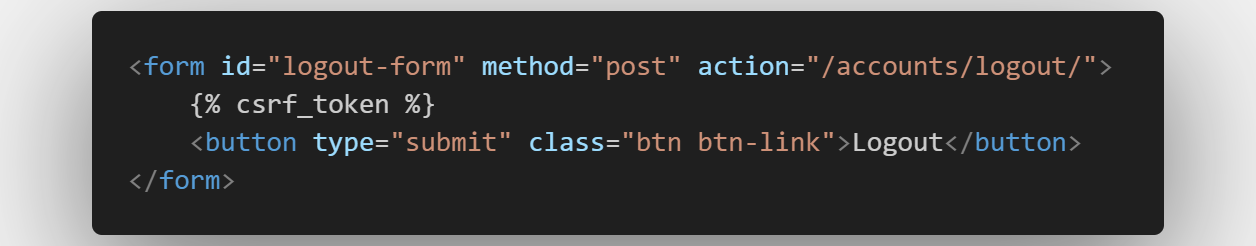
**Django**

1. What is Django
   * Django is a high-level Python full-stack web framework that encourages rapid development and clean, pragmatic design.
   * Django follows the model–template–views (MVT) architectural pattern.
     + **Models**: Represent data structures and interact with the database.
     + **Views**: Handle user requests and determine how to respond.
     + **Templates**: Define the presentation layer (HTML) of the application.
2. Django Project vs Application
   * Project
     + A Django project acts as the top-level directory that houses your entire web application. It includes configurations, and settings.
     + A project can contain multiple apps.
   * Application
     + A Django app is a self-contained module designed to accomplish a specific task within the overall project.
     + You can use an app in multiple projects, and you can distribute apps, because they don’t have to be tied to a given Django project (installation).
3. Installing django
   * To create a Django project, first, you need to install Django on your operating system.
   * **pip install Django**
   * After installation check django on your system. **django-admin --version**
4. Create django project
   * Once you have installed Django on your operating system, now you can create a project.
   * Commmad:- **django-admin startproject <project\_name>**
   * Navigate to the project directory and start the development server for test - '**python** **manage.py runserver'**.
5. Create an Application
   * Django, it’s a common practice to create a new app for each distinct feature or functionality you want to add to your project.
   * Creating separate apps for each feature helps to keep your code organized and modular.
   * It allows you to reuse apps across different projects, and it makes it easier to maintain and scale your code as your project grows.
   * Command - **python manage.py startapp <app\_name>**
   * Register the app with our Django project by updating **INSTALLED\_APPS** tuple in the **settings.py**.
   * 
6. Creating Views
   * Django views are Python functions that takes http requests and returns http response.
   * 
7. URL Mapping in the application
   * To call the view, we need to map it to a URL - and for this we need a URL configuration.
   * To create a URL configuration in the app directory, create a file called **urls.py**.
   * A computer screen shot of a program code

     Description automatically generated
   * The next include your app **urls** in the **root** URL configuration.
   * A screen shot of a computer code

     Description automatically generated
8. Template System
   * In Django, a template is a text document or a Python string marked-up using the Django template language (DTL).
   * It’s used to generate dynamic HTML content.
   * Django’s template engine offers a mini-language to define the user-facing layer of the application.
   * The syntax of the Django template language involves four constructs.
     + Variables
       - A variable outputs a value from the context.
       - Variables are surrounded by **{{ var\_name }}** like this.
       - E.g. **User role is {{ user\_role }}**
     + Tags
       - Tags provide arbitrary logic in the rendering process.
       - Tags are surrounded by **{%** and **%}** like this:
       - Tags lets you perform the following operations: **if** condition, **for** loop, template inheritance and more.
       - Just like in Python you can use **if, else** and **elif** in your template
     + Filters
       - Filters help you modify variables at display time.
       - Filters structure looks like the following: **{{ var|filters }}**
       - E.g. **{{ customer\_name| lower}}** − Converts the customer name to lowercase.
     + Comments
       - Comments look like this: **{# this is commented #}**
       - OR **{% comment %} this is commented {% endcomment %}**
   * **Create django template.**
     + **NB**. Before configuring templates, make sure you have added the app to **INSTALLED\_APPS** in settings.py.
     + Django looks for a **templates** folder within each app to load templates.
     + Create html file in **templates** folder.
     + The Render Function
       - It plays a crucial role in presenting dynamic web pages by bringing together various elements.
       - It takes three parameters.
         * **Request** − The initial request.
         * **The path to the template** − This is the path relative to template file.
         * **Dictionary of parameters** − A dictionary that contains all variables needed in the template.
       - How to pass context to the template
         * The render function accepts the context parameter to pass variables to the templates.
9. Database
   * Django officially supports the following databases (refer official documentation <https://docs.djangoproject.com/en/5.0/ref/databases/>)
     + PostgreSQL
     + MariaDB
     + MySQL
     + Oracle
     + SQLite
   * In this tutorial, we will use MySQL.
     + For use MySQL you need to install the MySQL driver
       - **MySQL Connector/Python** is a pure Python driver from Oracle that does not require the MySQL client library or any Python modules outside the standard library.
         * **pip install mysql-connector-python**
         * Egine**:** **mysql.connector.django**
       - **Mysqlclient** - mysqlclient library for Python does use the MySQL C client library.
         * **pip install mysqlclient**
         * Engine: **django.db.backends.mysql**
       - Change database connection in **settings.py**
   * 
10. Django - Models
    * A Django model is a Python class that represents a database table or collection.
    * It is used to define the structure and behavior of the data that will be stored in the database.
    * Each attribute of the model class represents a field in the corresponding database table.
    * Each model maps to a single database table.
    * **NB**. Before configuring model, make sure you have added the app to **INSTALLED\_APPS** in settings.py.
    * Defining a Model
      + A model is defined as a subclass of **django.db.models.Model** class
      + Create model class in **models.py.**
    * 
    * Prepare migrations.
      + **python manage.py makemigrations** - the command that creates new migrations based on the changes detected in your models.
    * Apply migrations changes.
      + **python manage.py migrate** - command in Django used to apply migrations that have been generated by the **makemigrations** command.
    * Save model to DB.
    * 
    * Query from Database
    * 
11. Form Processing
    * In Django, forms are a fundamental mechanism for collecting user input in web applications.
    * They provide a structured approach to handling user interactions, data validation, and rendering HTML forms.
    * They provide a range of tools and libraries to help you build forms to accept input from site visitors, and then process and respond to the input.
    * Django provides a **Form** class which is used to create form objects. These form objects contain fields, defined as class variables, that map to **HTML form <input>** elements.
    * Django form is really like Django model.
    * 
    * Using Form in a View
      + Django forms supported only **GET** and **POST** http methods.
        - **GET** - usually for loading the form initially.
        - **POST** - this method is used for any request that could change the state of the system.
          * Data passed via POST can be accessed via the **request.POST** dictionary.
          * **NB**: You should add CSRF protection to the POST request by including the **{% csrf\_token %}** tag at the beginning of the form in your html template.
    * 
12. Django - Admin Interface
    * Django offers a built-in admin interface for administrative activities.
    * It automatically creates a user interface based on your defined models.
    * To access the admin interface, you need to have a superuser account.
      + To create super user –
        - **python manage.py createsuperuser**
    * To access admin interface - **<app-doman>/admin**
    * Register A Model In Django Admin
      + To register a model to Django’s admin, first import the model into the admin.py file of the same Django app as the models.py file.
      + Then use **admin.site.register(ModelName)** to register.
    * 
13. Setting up your authentication
    * Django provides almost everything you need to create authentication pages to handle login, log out, and password management.
    * This includes a URL mapper, views and forms, **but it does not include the templates**, so we have to create our own templates.
    * Add authentication **URLS.**
      + Add the following in root URL configuration to enable Django auth URL maps.
        - **path(/accounts/', include('django.contrib.auth.urls'))**
        - On navigate to **/accounts/** you will see the **auth** **URLS** mapping
    * Django looks for authentication templates in the **/registration/** directory.
      + So create the directory (**/registration/**) in templates folder of the project root folder.
      + **NB**: Make sure that the **templates** directory from the root folder is added to the DIRS list of the **TEMPLATES** setting in **settings.py**.
        - Django does not automatically look for templates in a **templates** directory at the root of your project.
        - By default, Django will only look for templates within each application’s templates directory.
        - 
        - On successful login, Django will redirect to **http://127.0.0.1:8000/accounts/profile/** by default.
          * To change this behavior add **LOGIN\_REDIRECT\_URL** in settings.py

**LOGIN\_REDIRECT\_URL = '<full-path-to-redirect>'**

* + Protect unauthorized access.
    - To protect views from being accessed by unauthorized users, add **@login\_required** decoration on the views function from the module **django.contrib.auth.decorators.login\_required**.
    - Also, you can do the same thing manually by checking on **request.user.is\_authenticated.**
    - But the decorator is much more convenient!
  + 
  + Logout user
    - To log out the currently authenticated user, send a POST request to the **/accounts/logout/** URL.
    - NB: also customize logout redirect URL by adding the variable **LOGOUT\_REDIRECT\_URL** in **settings.py**.
      * **LOGOUT\_REDIRECT\_URL = '<full-path-to-redirect>'**
  + 

**THANK YOU!!**

**Bantayehu Fikadu**

**March 2024**