Django

A Python Web Framework

What is Django?

- Python web framework
- Built in development server and database
- Let's you create reusable apps
- Comes with a lot of built-in functionality (e.g., administration sites)
- Open source

How can I use Django?

- Install Python from *python.org*
- Use the Python package manager *pip* to install Django
  - *python –m pip install django*
- Confirm that Django was installed correctly
  - *python –m django --version*

How to start a Django project

- *django-admin startproject mysite* (hyphens in the project name are not allowed!)
  - Creates basic folder structure and files
  - *manage.py* for interacting with the Django project
  - *urls.py*
  - *settings.py* (defines basic settings for the project)
- Verify the project is working by running the development server
  - *python manage.py runserver*

Create an app

- A Django project is just a container for different apps
- To create an app run *python manage.py startapp <app-name>*
  - Apps are created inside the folder of the *manage.py* file
  - A folder named *<app-name>* and necessary files will be created
  - *views.py* (contains the web pages i.e., the views of the app represented by Python functions)
  - *models.py* (holds the database models of the app)
  - *admin.py* (register database models for admin usage)

MVC

- Model View Controller
- Pattern for structuring software into three main components
  - Model: the data which is stored/modified/used in the application
  - View: interacts with the user; provides the UI (just HTML in case of the web)
  - Controller: interface between view and model; processes data from (HTTP-)Requests and updates the view
- Retrospect: Webengineering I
  - JavaBeans (model)
  - JSP (view)
  - Servlets (controller)
- Django encourages the developer to use this pattern
  - *models.py*
  - *templates*
  - *views.py*

Create a view

- You can add views to your app by defining functions in the *views.py* file
- These views/functions need to be wired to a specific URL inside a new *urls.py* file within the app folder
- The app specific *urls.py* must be included into the *urls.py* inside the project folder
- Instead of returning the HTML as a string you should use templates
  - Create a templates directory inside the app folder
  - Inside the templates directory create another directory with the name of the app (this is called *namespacing* templates)
  - You can add HTML files as templates and render them with *django.shortcuts.render()* in *views.py*
  - Make sure to add your app in the *settings.py*

Create a database model

- Django comes with a built in SQLite database
- A model is a single set of information and basically defines a table in your database (models can be created in the *models.py* file)
- Class variables declare database fields (there are a lot of predefined fields)
- To store the changes as *migrations* run the *makemigrations* command (similar to *git commit*)
  - Stages changes inside the *migrations* folder
  - To view the changes that would be applied: *sqlmigrate <app-name> <migration>*
- To apply changes run *migrate* (similar to *git push*)
- You can add data as an admin by creating a super user (*createsuperuser*) and visiting the */admin* site after registering the model in *admin.py*

Pros and Cons of Django

- Pros
  - Django is a “High Level” framework i.e., it offers a lot of inbuilt tools and utilities e.g., admin panel, user authentication or testing-libraries
  - Scalability - Django is built to handle millions of users
  - Clearly structured (MVC)
  - Less code because of reusable apps
- Cons
  - Django is known for taking up a lot of resources (less suited for small projects)
  - Slower compared to other backend frameworks

Recap

- Basic explanation of Django
- Installation of Django
- Running the development server
- Starting a project and creating an app
- Basic explanation of MVC
- Creating views and templates
- Using database models
- Managing data as an admin
- Creating a basic task tracker app