Advanced technologies: Django, Mapbox, Leaflet

What is a web framework

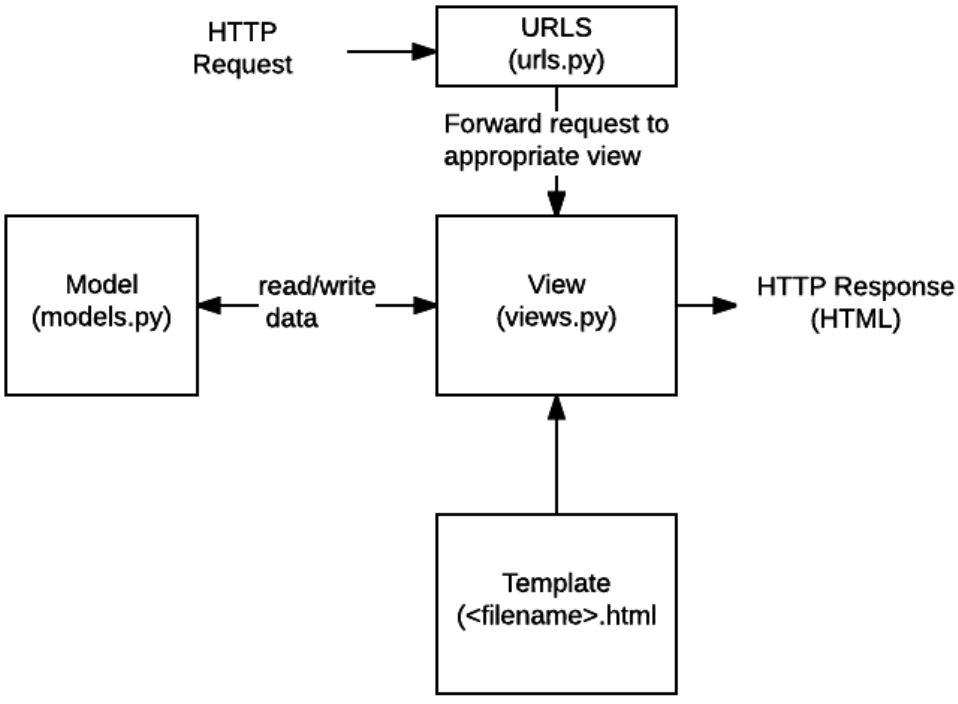
A web framework is a library that allows us to write web applications or services without having to handle low-level details such as protocols, sockets or process/thread management. They offer a standard way to build and structure an application without too much work. This allows us to focus on implementing features rather than configuration details.

Common tasks that web frameworks can handle include:

- URL routing
- HTML, XML, JSON, and other output format templating
- Database manipulation
- Session storage and retrieval

Django

Django is a python based web framework that follows the model-template-views architectural pattern.



The image above shows how Django handles a HTTP request.
URLS: Contains a URL mapper that redirects a HTTP request to the corresponding view function. The URL mapper can parse for patterns in the URL and pass these as data to the view.

View: Contains a handler function for each valid HTTP request. Returns a HTTP response. Access data required for a request via the model. Gets the format for the HTTP response via the template. Server-side scripting can be done in the functions to create an interactive website.
Model: Models are python objects. They provide the structure of an application's data and provide mechanisms to manage and query records in a database.
Template: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model.

As seen in the explanation above, the model-template-views pattern allows for separation of logic and code reuse. For example a single HTML template can be used by 2 different view functions populating the template with different data.

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, following consistent design principles.

Another feature of Django is that it has over 4000 packages that can be used to build and run an application. This means it is likely that a specific configuration required already exists and can be used instead of building from scratch.

Flask

Flask is a python micro web framework. It is defined as a microframework because it has no requirements for use of specific tools or libraries. It has no database abstraction layer compared to Django which has the model. Additionally, it doesn't support form validation or any other common tasks where other frameworks would have library support.

This is what makes Flask different from Django as it is very flexible, with its main approach stated in its documentation "Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don't". This flexibility comes at a cost to the developer as they must have prior knowledge on web development to know how to build up the framework themselves with their required extensions. However the benefits come at no bloat to the framework as everything included directly supports the application.

This means Flask is the opposite in terms of Django's Batteries-included approach, where only a simple core is included. Tools and libraries needed are added onto the core as extensions.

Opinionated and Unopinionated frameworks

Opinionated frameworks are those with opinions about the "right way" to handle any particular task. They often support rapid development in a particular domain because the right way to do anything is usually well-understood and well-documented. However they can be less flexible at

solving problems outside their main domain, and tend to offer fewer choices for what components and approaches they can use.

Unopinionated frameworks have fewer restrictions on the best way to glue components together to achieve a goal, or even what components should be used. They make it easier for developers to use the most suitable tools to complete a particular task, albeit at the cost that you need to find those components yourself.

Django is a somewhat opinionated framework. This means it provides a set of components to handle most web development tasks and preferred ways to use them. However due to its decoupled architecture we can still pick and choose different options or add new ones ourselves.

On the other hand Flask is an unopinionated framework, following its definition as a microframework, where it is very flexible in allowing a developer to build any type of website. However, it comes at the cost of having to find components and the knowledge of how to make multiple components work together.

Why I chose Django over Flask

Comparing the 2 frameworks, they both have different approaches to web development. Django is a powerful framework providing features such as ORM (object relational mapping) and uses data models out of the box. Additionally using a preferred architecture of MVT (model view template) which makes structuring code easier to apply the DRY (don't repeat yourself) principle. This contrasts with Flask which only requires the use of a default template engine which can be easily changed. It doesn't have any of the mentioned Django features out of the box. Allowing for more flexibility and modularity but requiring previous knowledge in web development.

One of the main reasons in choosing Django is due to this project being the first time I have done a web project in python. Due to having to learn everything about web development, choosing Django made the most sense as it is more beginner friendly than Flask. This is because Flask requires knowing what components are needed to build an application as out of the box the core is very simple and requires extensions to make a basic application. As a beginner having to find and know how to put together various components is too steep a learning curve. Whereas compared to Django it follows a Batteries-included philosophy, therefore following a basic tutorial is much easier and a basic application can be made faster.

Another reason for choosing Django is because it is much more popular with an active developer community compared to Flask. This means finding tutorials or questions is much easier. Compared to Flask, its community is not as big as Django and therefore finding answers to questions and basic tutorials will take more time. This made me choose Django as being my first web project in python, having a large community will mean more resources online to learn from with more basic tutorials that don't rely on assumed knowledge.

Leaflet

Leaflet is a javascript library for creating interactive maps. Its main feature being lightweight (39KB of JS) and mobile friendly. By default, Leaflet only supports displaying a map via raster tiles. Raster tiles are images rendered on a server and then displayed on a website. A benefit of using raster tiles is that it works on all types of devices as it is just displaying an image. However there are multiple downsides to using raster tiles. It is not possible to show/hide objects in the client, no way to customize or add new styles on the client side and zooming/moving the map displays temporary empty sections as the required tiles are loaded in.

Since Leaflet is lightweight, its customization options on a map are limited. To get around this it allows for plugins for various features. However these plugins are third party and therefore documentation isn't guaranteed. Additionally, different plugins compatibility with each other is unknown.

Mapbox

Mapbox is a provider of online custom maps. It is the creator of the library Mapbox GL-JS. Opposite to Leaflet, Mapbox GL-JS displays maps via vector tiles. Vector tiles have many benefits. This includes being able to change the map's style dynamically client side, no rendering delays when the map is moved and a small size due to vector tiles being binary files containing information on how to generate the map instead of an image. However there are some downsides to the flexibility vector tiles provide. Because rendering the map occurs client-side, it could cause performance issues on slower devices. Additionally, the library being used is specific for desktop users, therefore mobile users are not accounted for and a separate library must be used.

Because Mapbox is a commercial company, Mapbox GL-JS is a powerful library with many customization options for the map. The default vector tiles provided by Mapbox can be modified on their website to suit a specific need. Additionally, custom made vector data can be created on the website and uploaded to an account. This vector data can then be accessed through API calls.

Creating a map

To create a map on Mapbox/Leaflet, a div element is created in HTML. Then in javascript a Mapbox/Leaflet map object is created and bound to that div. Then the map object is given a tile layer from a source. In the tutorials they recommend using Mapbox vector tiles. Just from these simple steps an interactive map is created.

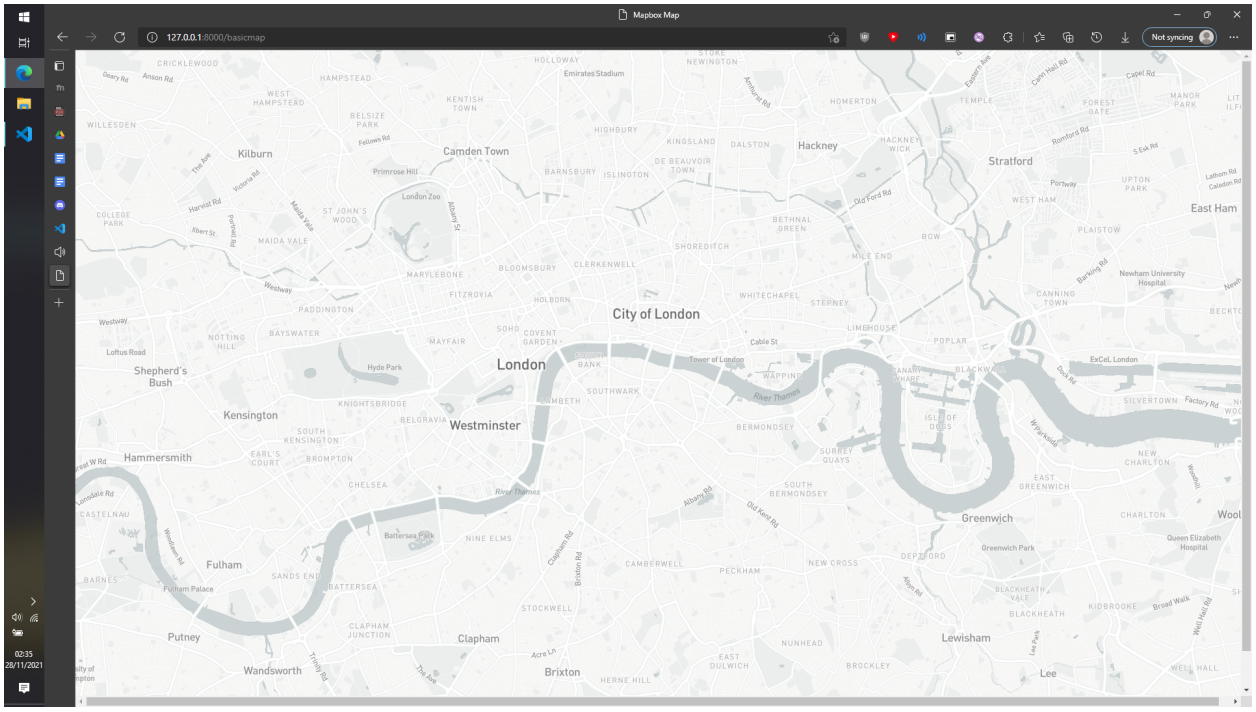
The process of creating a basic map is similar for both libraries because Leaflet was initially developed by someone who previously worked on Mapbox.js (older version of Mapbox GL-JS).

Why I chose Mapbox over Leaflet

As explained above, Mapbox has much more customization available than Leaflet. This is one of the main reasons I chose Mapbox because I don't know the scope or what features will be required for the final project. Therefore using Mapbox will mean I don't have to make a late switch between libraries when I find out it doesn't have a specific feature I need. I believe learning the more complicated system of Mapbox is worth it compared to the risk of switching late if Leaflet doesn't work.

When comparing Leaflet and Mapbox GL-J in terms of official documentation, Leaflet's documentation is very basic and doesn't cover any advanced use of the library. This means having to search through the internet for tutorials and examples. Additionally because it relies on plugins for more advanced features, the documentation will not be standardised. This is another reason why I chose Mapbox because the official documentation is extensive and covers most use cases.

Example - A django application displaying a Mapbox map



This example website will return a map from the url http://127.0.0.1:8000/basicmap. This map is interactive, allowing for moving the map via dragging the mouse and zooming in and out. The Django development server will be used to return the html, css and js from the url.

urls.py

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

```
urlpatterns = [
    path("basicmap", views.map, name="map"),
]
```

This is the file used to map urls to their corresponding view function. The array urlpatterns contains all mappings. The path function returns an element for inclusion in urlpatterns. The path function's first argument is the route which is a string containing a url pattern, in this case it's the url ending with /basicmap. The next argument is the view function that will be run in response to the url. In this case it will be running the map() function in the views.py file. Finally an optional argument for a name is used which allows for easier referencing to this mapping.

views.py

```
from django.http import HttpResponse
from django.shortcuts import render

def map(request):
    return render(
        request,
        'map/map.html'
    )
```

This file contains view functions that are run in response to a web request and return a web response. This response can be anything, e.g. the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document. View functions can contain any logic that is necessary to return a response. View functions take a HttpRequest argument, which is called request here.

For the view function map, it runs in response to the url http://127.0.0.1:8000/basicmap. I have used the Django function render, which takes the request and a html file that may contain placeholders for values. Then the Django templating engine makes the substitutions if necessary and returns the html file. This provides separation of code as the view function only works with data values and the template only works with the markup.

map.html

```
<!DOCTYPE html>

<head>
    <title>Mapbox Map</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script
src='https://api.mapbox.com/mapbox-gl-js/v2.4.1/mapbox-gl.js'></script>
    {% load static %}
```

```
    <link rel="stylesheet" type="text/css" href="{% static 'map/style.css'
%}" />

</head>

<body>
    <div id='map'></div>
    <script src="{% static 'map/basicMap.js' %}"></script>
</body>

</html>
```

This file is the template that is used in the view function map. Here there are placeholders for values being used. In this case {% load static %} is used to signify that the template engine should fill out the static file pathway. “{% static ‘map/style.css’%}” is the template and will be filled out when rendered in the view function. In this example the string “{% static ‘map/style.css’%}” will be filled out to “map/static/map/style.css” by the templating engine. This same process is used to load the file basicMap.js.

In the head of the file, the library for mapbox gl-js is loaded by using a hosted version of it found at the link https://api.mapbox.com/mapbox-gl-js/v2.4.1/mapbox-gl.js.

Finally a div element with the id map is created. This will be the container for the map object created by Mapbox that displays an interactive map.

basicMap.js

```
mapboxgl.accessToken =
"pk.eyJ1IjoiYWdyYWlzIiwiYSI6ImNrdHVnOWU0azBoeGIycnF0dGE2dHp4dXYifQ.gIMMHin
rqMqWUQSQCAP4bw";
const map = new mapboxgl.Map({
  container: "map", // container ID
  style: "mapbox://styles/mapbox/light-v10", // style URL
  center: [-0.09, 51.505], // starting position [lng, lat]
  zoom: 12, // starting zoom
  minZoom: 10,
});
```

This script is run at the place it is called in the html file. Mapbox gl-js provides a mapboxgl.Map class. This class is instantiated with arguments seen in the code above. The container used is the div element map shown in the html file above. Mapbox provides a set of styles to use. Here the light v10 style is used on the default vector tileset Mapbox provides.

A mapboxgl.accessToken has its value set. This token is uniquely generated for my account and is used to allow the program to retrieve vector data from mapbox servers.

style.css

```
html, body {
    margin:0px;
  }

#map {
    height: 100vh;
    width: 100vw;
}
```

The style files role here is to make the map div cover the whole screen. This is done by making every element in the html file have a 0px margin. Then the map div element's height and width are set to 100vh and 100vw. Vh and vw are short for viewport height and width with each unit being 1%. The viewport is the visible area on the screen. Therefore setting the height and width of the map div to 100% of the viewport height and width.