Urls, Templates, Models

Mapping URLs

- Rather than directly mapping URLs from the project (in CRM_project/urls.py) to the app, we can make our app more modular (and thus re-usable) by changing how we route the incoming URL to a view.

- To do this, we first need to modify the project's urls.py and have it point to the app to handle any specific crm app requests. We then need to specify how Rango deals with such requests.

Mapping URLs

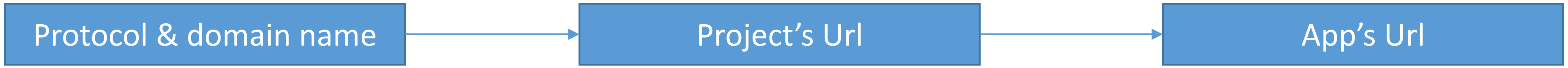
- First, open the **project's urls.py** file which is located inside your project configuration directory. Update module to look like the example below.

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

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

Mapping URLs

- Think of this as a chain that processes the URL string



- Given the changes we made above, we need to create a new file called urls.py in the crm_app directory. This new module will handle the remaining URL string. Update module to look like the example below.

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


urlpatterns = [
    path('', views.home, name="home"),
]
```

Templates and Media Files

- A template in Django is basically written in HTML, CSS, and Javascript in a .html file. Django framework efficiently handles and generates dynamically HTML web pages that are visible to the end-user.
- Django mainly functions with a backend so, in order to provide a frontend and provide a layout to our website, we use templates.

Why templates

- The layout from page to page within a website is often the same.
- Whether you see a common header or footer on a website’s pages, the repetition of page layouts aids users with navigation and reinforces a sense of continuity.
- Templates make it easier for developers to achieve this design goal, as well as separating application logic (code within your views) from presentational concerns (look and feel of your app).

Configuring the Templates Directory

- To get templates up and running within your Django app, you'll need to create two directories in which template files are stored.

- Within the **CRM_APP** directory, create a directory called, **templates**, and then within that directory, created another directory called **crm**. This may seem odd, but Django looks within the application directories for the templates directory, and then tries to find the specific templates for the named application.

Adding a Template

- Within your template directory for **crm_app** i.e. **crm_app/templates/crm/**, create a file called **home.html**. In this file, add the following HTML markup and Django template code.

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

urlpatterns = [
    path('', views.home, name="home"),
]
```

Adding a Template

- To use this template, we need to reconfigure the index() view that we created earlier in chapter 2. Instead of dispatching a simple response, we will change the function name to **home** and the view to dispatch our template

```
from django.shortcuts import render

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

Template Context

- A template context is a Python dictionary that maps template variable names with Python variables
- A Context is a dictionary with variable names as the key and their values as the value
- A Context is a mapping of variable names to values. When the template is rendered, these values are made available to the template engine and fill in "the holes" in your templates by replacing variables with their respective values.

Template Context

File	**Code**

(project) views.py	context_dictionary = {'**cars**': 'Ford, Ferrari, Mustang, Range Rover, Tesla'}
	
	def home(request):
	    return render(request, 'crm/home.html', context_dictionary)

(project) home.html	**{{cars}}**

Static files in Django

- Static files are the files that are not generated dynamically by the application; they are simply sent as it is to a client’s web browser. Example of static files are :Cascading Style Sheets (CSS), JavaScript and Images

Configuring the Static Media Directory

- To get templates up and running within your Django app, you'll need to create two directories in which static files are stored.

- Within the **CRM_APP** directory, create a directory called, **static**, and then within that directory, created another directory called **crm**. Just like for templates, Django looks within the application directories for the static directory, and then tries to find the specific templates for the named application. In the static folder, create another directory called **images**, and this is where you will save all your image files.

Configuring the Static Media Directory

- We need to tell Django about our new static directory. To do this, open the project's settings.py module. Within this file, we need to add a new variable pointing to our static directory. First, create a variable called STATIC_DIR at the top of settings.py, underneath BASE_DIR to keep your paths all in the same place.

File	Code
(project) settings.py	import os

	STATIC_URL = 'static/' ***(should already exist)***
	STATIC_DIR = os.path.join(BASE_DIR, 'static')
	STATICFILES_DIRS = [STATIC_DIR,]

Static Media Files and Templates

- Now that you have your Django project set up to handle static files, you can now make use of these files within your templates to improve their appearance and add additional functionality.

File	Code
(project) home.html	**{% load static %}** <!doctype html> <html lang="en"> <head> </head> <body> **<img src="{% static 'images/Logo.jpg' %}" alt="Picture of Nile" />** </body> </html>

Models and Databases

- Typically, web applications require a backend to store the dynamic content that appears on the app’s webpages. For Rango, we need to store pages and categories that are created, along with other details. The most convenient way to do this is by employing the services of a relational database. These will likely use the Structured Query Language (SQL) to allow you to query the data store.

- However, Django provides a convenient way in which to access data stored in databases by using an Object Relational Mapper (ORM)1. In essence, data stored withina database table is encapsulated via Django models.

Model

- A model is a Python object that describes the database table’s data. Instead of directly working on the database via SQL, Django provides methods that let you manipulate the data via the corresponding Python model object. Any commands that you issue to the ORM are automatically converted to the corresponding SQL statement on your behalf.