

Web Application Frameworks (COMP3011/COMP 6006)

Lecture 1: Introduction to the Unit and Django

Welcome to Country

“I acknowledge the Wadjuk Noongar people as the traditional custodians of the land on which Curtin University’s Bentley Campus sits, and would like to pay respect to elders past, present and emerging.”

Outline



Unit Introduction



System Pitfalls



Web Application Frameworks



Introduction to Django

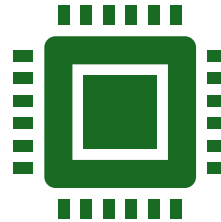
About Me

- Dr. Sheik Mohammad Mostakim Fattah (He/Him/His)
- Education:
 - Doctor of Philosophy in Computer Science (University of Sydney, Australia)
 - Master of Engineering in Information and Communication Engineering (Hankuk University of Foreign Studies, Republic of Korea)
 - Bachelor of Science (Hons) in Computer Science and Engineering (University of Dhaka)
- Email: sheik.fattah@curtin.edu.au (Best Communication Method)
- Teaching: Web Application Frameworks, Distributed Computing, Computer Project 1
- Research: Cloud Computing, Internet of Things, and Service Computing

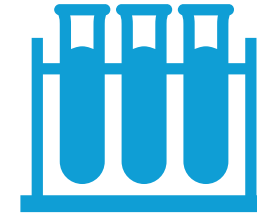
Learning Activities



Lecture: 1 X 2 Hours
(Weekly)



Computer Laboratory:
1 X 2 Hours (Weekly)



**No lab in the first
week**

Unit Learning Outcomes



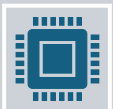
Evaluate and argue for architectural design approaches for developing web applications in terms of security, usability, performance, and other properties;



Create sophisticated client-side web applications based on modern frameworks and tool sets.



Create server-side back-ends to web applications to support complex functionality and secure data management;



Assess the workability, interoperability and quality of client-side and server-side aspects of web applications.

Learning Resources



Free, online tutorials and reference materials (Similar to Books) will be made available throughout the semester.



Blackboard is your friend

Recording of the lectures can be found in iLecture

Lecture slides & Lab's material: slides, source code templates

Assessment specification, clarification, and submission



If something is happening with this unit, it will be communicated via Blackboard, in one format or another.



Read the unit outline! It is available via OASIS



Useful
Knowledge

Python

JavaScript

HTML, CSS


Relational Database

No SQL Database

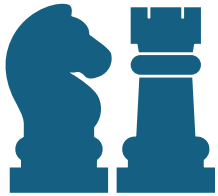
Expectations & Resources

- You will need a computer with free software to undertake this unit.
 - You will need an Integrated Development Environment (IDE); VS Code is recommended, but you are more than welcome to use something else;
 - Additional software is required throughout semester, we will discuss this when required – these are development libraries;
 - You can use the laboratory machines; however, it is easier to use your own!
- You are expected to attend the lecture and lab and undertake the activities and assessments throughout semester.
 - You may need more time for the practical activities outside of class;
 - You may wish to read up on the additional resources to undertake more practice.

Assessments

	Task	Value %	Date Due	Unit Learning Outcome(s) Assessed	Late Assessments Accepted?	Assessment Extensions Considered? *
1	Practical Work	25 %	Week: 9 Day: 28th April, 2025 Time: 23.59	1,2,3	No	 Yes
2	Web Design Assignment	25 %	Week: 4 Day: 21st March Time: 23.59	2,3,4	Yes	Yes
3	Final Examination	50 %	Week: TBD Day: TBD Time: TBD	1,2,3,4	No	Yes

Using AI Tools for Assignments



Examples: ChatGPT, deepseek,
Bard, and many more!!!



**Strictly prohibited for the
assignment!!!**



Check the Curtin Connect FAQ
Page for more details.

Lecture + Lab Slides Format

- Lecture slides will contain **exercises** for the lab. So, you have only one document in each week.
- If you see **YOUR TURN**, it means you will practice it in the lab.
 - You will often be introduced to commands and processes within the workshop slides, which you may wish to run. You don't need to (please don't) run them unless you see **YOUR TURN**.
 - If you do try and include them within the practical, you may end up with a mess of code, some of which works and some of which doesn't;
 - The time you see in the corner is just a rough guide, it may take you (or all of us) longer than suggested – do not worry one bit!
 - **When you see something in carets (e.g. <example>), replace it with a relevant value as asked, including the carets themselves (i.e. <example> becomes myApp).**

System Pitfalls

- On some systems, you may have to use `python` rather than `python3`;
- You may have installed an earlier version of Django, depending on your system and what you have done previously.
 - Anything that is version 3 or later will be okay for this unit.
 - Take note of the output when you installed Django.
- You may need to remove and then reinstall Django; issue the terminal command: `python3 -m pip uninstall Django`.
- Details will be provided soon regarding how to install Python, in later slides.

Pitfalls on Windows Systems

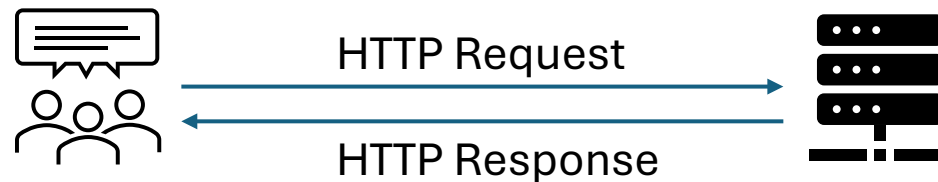
- (Generally) On Windows, note down any error messages that look similar to the following:
 - WARNING: The script django-admin.exe is installed in 'C:\Users\username\AppData\Roaming\Python\Python311-arm64\Scripts' which is not on PATH.
- To fix this, you will have to adjust your PATH, or each time below we refer to django-admin, specify the path in full, i.e.
"C:\Users\username\AppData\Roaming\Python\Python311-arm64\Scripts\django-admin.exe" into the PowerShell console/terminal, complete with leading dot and quotes.

Pitfalls on the Virtual Desktop Infrastructure (VDI)

- On the VDI/Lab Machines, django-admin won't be installed globally, nor will it persist between logons.
 - You'll have to “**python3 -m pip install**” Django each time you log in and want to use *django-admin* – but not a Django project itself;
 - Hence, once you have created a project, this is not an issue – until you create another, as it is rarely used for anything else;
 - All terminal/console commands for django-admin will need to be prepended, such that the command is as follows: “**~/local/bin/django-admin**”
- *By default, your 'home' folder that persists data is not at ~. Instead, it will be located at /mnt/home/<yourID>, where <yourID> is your Student ID.*
- *Keep this in mind when you create and save files, you will use them again!*
- *Due to a bug, this folder won't be created unless you have first logged into the Level 2 Linux machines in Building 314. Do this now, if you haven't already.*

Understanding the basic

- Let's assume that you want to build a website and publish it at mypage.com.
- It will have two parts:
 - Front-end - Loaded on the Client Machine
 - Back-end - Runs on a webserver
- Users will access the website using URL (Uniform Resource Locator)



Web Application Frameworks (WAFs)



Software Framework that provides a way to build and deploy web application



Simplify the web development process by automating routine tasks.



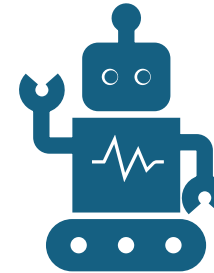
Key benefits: Efficiency, Security, Scalability, Community and Support

WAFs continued...



Common Features:

- MVC Architecture
- Database Management
- Session Management
- Security Features



Examples:

- Python: Django, Flask
- JavaScript: Express.js, Angular, React
- Ruby: Ruby on Rails
- PHP: Laravel, Symfony
- .Net: ASP.NET

Django: Python Web Framework

What is Django?

- Django is a high-level Python web framework
- It enables rapid development of secure and maintainable websites.
- It is free and open-source, designed to help developers build web applications quickly with minimal boilerplate code.
- (Parts of) many popular web applications are (were?) built with Django: Instagram, Spotify, Dropbox, YouTube, BitBucket.

Key Features

- MVC Architecture: Django follows the MTV (Model-Template-View) pattern, which is similar to MVC.
- Built-in Admin Panel: Automatically generates an admin interface for managing app data.
- Security Features: Prevents SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

Key Features

- ORM (Object-Relational Mapping) – Allows interaction with databases using Python instead of SQL.
- Scalability: Used by companies like Instagram, Pinterest, and Mozilla for handling large traffic.
- Built-in Authentication: Includes user authentication and session management.
- Batteries Included: Comes with tools for common web development tasks, like URL routing, form handling, and caching.

Your TURN (5 min): Practical Overview

- In the next few tutorials, you will be building **a personal website**. At this stage, your website will only have **a blog feature**; however, other features such as videos and forums may be added.
- *A blog is a list of articles, or blog posts, published on a single website and organized by date.* You can choose the theme of your blog (e.g. technology news, philosophy, reviews, etc.)
- Spend a couple of minutes considering what you could put as some example content in your blog. Write this down on paper, or in a word-processor document.

Your TURN (10 min): Set up pre-requisite (Part 1)

- This practical assumes that you have a code editor installed, alongside the latest version of Python 3.
 - If you are using the lab machines at Curtin, these will be installed for you.
- If not, you may wish to download and install the following for your system:
 - Visual Studio Code (latest version) from <https://code.visualstudio.com/> – as this is the code editor we will be using in the examples;
 - You will then need to also download Python (version 3.6 or later) from <https://www.python.org/> – rather than elsewhere such as the Windows Store.

Your TURN (10 min): Set up pre-requisite (Part 2)

- You will need to install Django (Latest) itself.
 - Instructions to do so are summarized below and provided at <https://docs.djangoproject.com/en/5.1/intro/install/>
 - If you are on a lab machine, you will have to install Django each time you need to use django-admin, as indicated in 'System Pitfalls'.
- Use the Python package manager to install Django:
 - `python3 -m pip install Django`
 - Recall your command may differ on your system from `python3`;

Model-View-Controller

- The **Model-View-Controller (MVC)** architecture is a commonly used software design pattern that separates an application into three main components:
 - Model – controls the organization and storage of data;
 - View – controls how data is displayed;
 - Controller – acts as an interface between the Model and the View. Determines what the user wants and returns data to the user.
- As we will see, Django somewhat confuses this.

Model-View-Controller

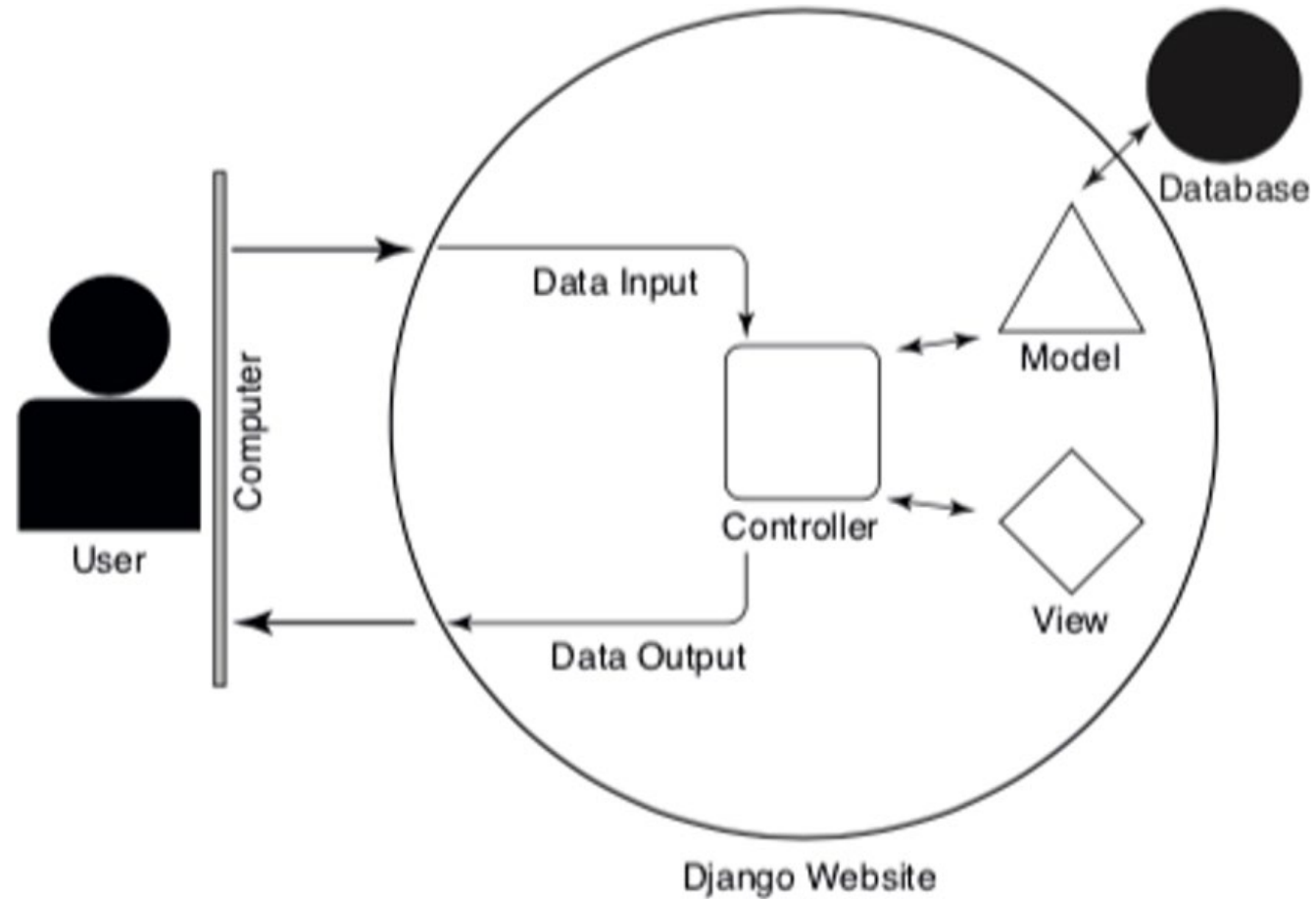
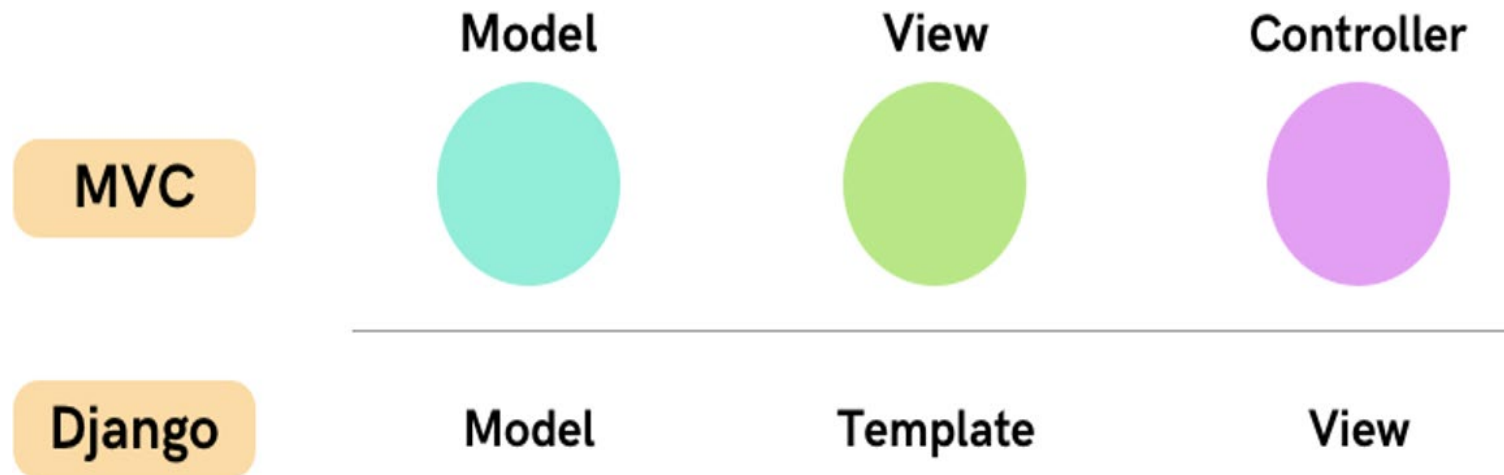


Diagram sourced from "Django Unleashed" by Andrew Pinkham.

MVC in Django

- It is called MVT – (Model-View-Template)



Creating a Django Project

- To create a project, we issue the Terminal command:

```
[user@pc]$ django-admin startproject <projectname>
```

- This creates a new project named `<projectname>` in the current directory. We will use `exampleproject` for this on future slides.
- In the process, this will create a bunch of 'boilerplate' files that we will edit and add to which we can build our project from.

Django Project Directory Structure

- ❑ `exampleproject`: project container;
 - ❑ `exampleproject`: separate project folder, inside the above;
 - ❑ `__init__.py`: tells Python to treat this directory as a package;
 - ❑ `asgi.py`: used to communicate with server software;
 - ❑ `settings.py`: contains your projects' settings;
 - ❑ `urls.py`: contains route definitions for the project;
 - ❑ `wsgi.py`: used to communicate with server software.
 - ❑ `db.sqlite3`: your database, it won't exist yet;
 - ❑ `manage.py`: used to manage your project – various utility functions like development web server, database migration.

Django Project Directory Structure

- Of the files highlighted on the last slide, the only one we will be editing is `urls.py`.
 - This enables us to define which URL's map to which page-creating functions.
 - The rest (generally) can stay as-is.
- We will be creating new files to handle the creation of our views and models.
 - Compare again above Models and Views in Django-speak vs MVC-speak.
- You will see `manage.py` a lot, however.
 - Think of it as our 'control center' for using Django.

Django Project Composition

- Django projects are organized in a structured way, making them modular and scalable. The key idea is that a Django project consists of one or more Apps, each handling a specific functionality.
- Django Project vs. Django App
 - Project: The overall container for your web application. It holds configurations, settings, and multiple apps.
 - App: A specific feature or module within the project, such as authentication, blog, or a chat system.

Think of a project as a company and each app as a department handling different responsibilities.

- Example: Social Media Web Application
 - Authentication, News Feed, Chat, Profile.
- Simpler web applications might use a single app. Complex applications can have multiple apps working together.

Your Turn (5 min): Creating a new Django Project (Part 1)

- With Django correctly installed, developers can access the `django-admin` command-line tool. This command, an alias to the `django-admin.py` script, provides subcommands to automate Django behavior.
- In the Terminal, identify using `pwd` (or a similar command) where you are currently located and use `cd` (or a similar command) to move to an appropriate folder to work in, that will be persisted.
 - Windows users may wish to investigate using `chdir` to achieve both of these.

Your Turn (5 min): Creating a new Django Project (Part 2)

- Once you have done this, to create a project, run the following command in the command-line terminal, substituting in your own `<projectname>`:

```
[user@pc]$ django-admin startproject <projectname>
```

- `<projectname>` can be any name of your choosing for your blog. However, a good name might be considered `blogproject`.
- The above command generates the folder structure for your project.

Your Turn (5 min): Creating a new Django Project (Part 3)

- Inside the newly created folder by the name of your project, you will find new files and folders for your project.
- `cd` to the root project directory and use the `tree` command (or a similar one – e.g. `ls -lR` or `find`) to list the directory:

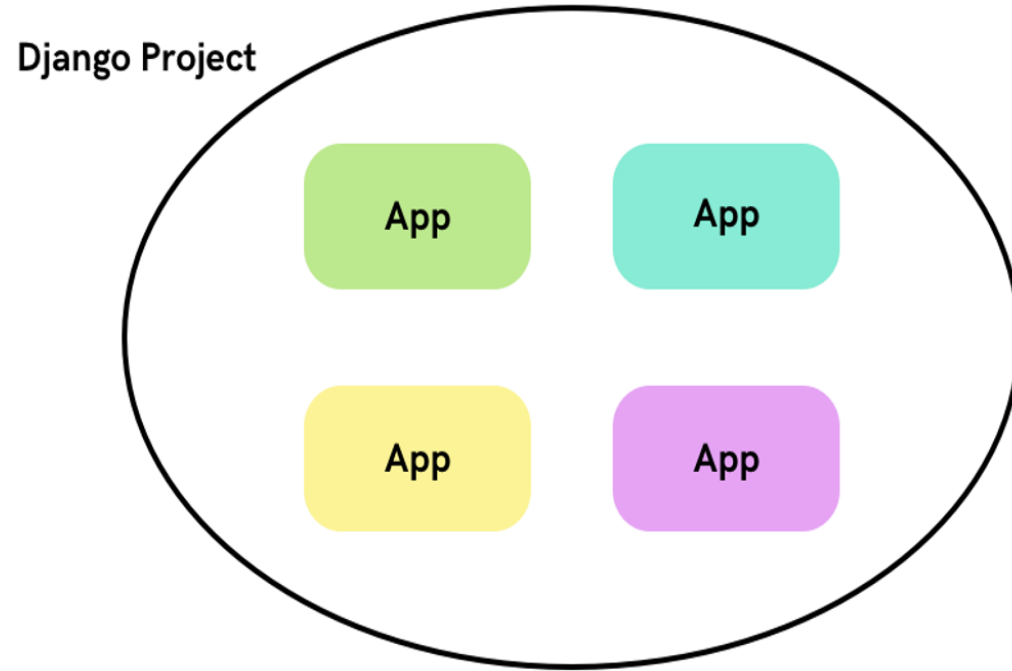
```
[user@pc]$ cd <projectname>
```

```
[user@pc]$ ls -lR
```

Your Turn (5 min): Creating a new Django Project (Part 4)

- Answer the following questions on paper or in a word processor document:
 - What is the purpose of the `__init__.py` file?
 - What does the `settings.py` file do?
 - What is the purpose of `urls.py` file?
 - Describe what the `wsgi.py` file does.
 - What does the `manage.py` file do?

Django Project vs Django App



Creating Django App

- In your project directory, to create a new App, issue the command:

```
[user@pc]$ python3 manage.py startapp <appname>
```

- You may need to a different command to python3, as previously described in 'System Pitfalls'.
- This creates a new App named <appname> in your current project. We will use `exampleapp` for this in the slides below.

Running Django App

- Once you have at least one App, you can start the local development web server by issuing the command:

```
[user@pc]$ python3 manage.py runserver
```

- This will, by default, run on port 8000 of your local machine.
- On most systems, a browser window will open automatically and direct you there.

Django Project Directory Structure, Enhanced (App vs Project)

- What's the difference between this and the last one?

- ❑ `exampleproject`: project container;

- ❑ `exampleapp`: app container directory ('app folder');

- ❑ `__init__.py`: tells Python to treat this directory as a package;

- ❑ `admin.py`: used to connect the App with Django's admin and configure the App for it;

- ❑ `apps.py`: contains App configuration settings, used with `settings.py`;

- ❑ `models.py`: defines the model (classes) used with your App;

- ❑ `tests.py`: defines unit test functions for the App (we won't);

- ❑ `urls.py`: contains route definitions for your App – needs to be created;

- ❑ `views.py`: contains the App's view (controller) functions that process requests and generates responses.

- ❑ `exampleproject`: unchanged as before.

YOUR TURN (5min): Creating Django apps

- In Django, a project is made up of one or more apps.
 - From Python's perspective, an app is simply a package (Python files can be modules, and a directory of modules is a package).
 - Your website, for example, can have multiple apps such as a blog, forum, video section, etc.
- To create the blog app, run the following command:

```
[user@pc]$ python3 manage.py startapp blogapp
```

- Use the `ls -lR` command to list the directory:

```
[user@pc]$ ls -lR
```

YOUR TURN (5min): Creating Django apps (Questions)

- Answer the following questions on paper or in a word processor document:
 - Describe what the `admin.py` file does.
 - What does the `apps.py` file do?
 - What is the purpose of the `models.py` file?
 - What does the `tests.py` file do?
 - Describe what the `urls.py` file does.
 - What is the purpose of the `views.py` file?

YOUR TURN (5 min): Starting the web server

- Django requires a database before it can run. Use the following command to create the database, ensuring you are in your project folder:

```
[user@pc]$ python3 manage.py migrate
```

- After successful migration, run the following command to start the web server locally:

```
[user@pc]$ python3 manage.py runserver
```

- By default, the server runs on port 8000 on the IP address 127.0.0.1, also known as localhost.
- So, navigate to <http://127.0.0.1:8000/> in the browser to view the local web server.

Implementing Views

- View functions are defined in `views.py`, for each App.
- A view function processes a request (something requests a URL via POST or GET) and generates a response.
- Hence, every view function should return a `HttpResponse` object and take a parameter named `request`. A very basic example:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, World!")
```

How Django processes a Request

- When a user requests a page from your Django-powered site – e.g., `localhost:8000/start`
 - Django loads the `urls.py` file in the Project folder and looks for the variable `urlpatterns` within the file;
 - Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL (`/start`);
 - Once one of the URL patterns matches, Django `imports` and calls the given view.
 - If nothing matches, an error is returned.

Mapping Views to URL's

- In your App directory, create a file named urls.py.

```
from django.urls import path from .  
import views  
  
urlpatterns = [  
    path( '', views.index, name='index' )  
]
```

- The first two lines import the path class from the urls module as well as views.py from the current directory respectively.

Mapping Views to URL's

- `urlpatterns` is our URL configuration – a list of structures describing the URLs and their associated views.
- Each element of the list is a call to `path` that takes three parameters: `route` (the URL fragment), `view` (a reference to the associated view function) and `name_url` (a name for the path).

Updating the Project urls.py

- The Project `urls.py` configuration must be updated when a new App is created.
- Edit the `urls.py` file from the Project directory:

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('example/', include('<appname>.urls'))
]
```

- This way, we connect the `urls.py` file for `<appname>` to the wider project.

Update the project urls.py

- The first pattern will route admin/ to the built-in Admin view;
- The second pattern will route anything starting with `example/` to the `<appname>.urls` Python file – i.e. the `urls.py` file in `<appname>`.
- Directly visiting `/` will give us an error message at this point in time – that's okay. We must remember to visit `example/`.

Making Django aware of Apps

- When you create an App, Django is not automatically aware of it. To register your App with Django, open `settings.py` from the project directory (root directory) and add the following line in the `INSTALLED_APPS` list:

```
INSTALLED_APPS = [  
    ...,  
    '<appname>.apps.ExampleConfig'  
]
```

- The name of the config class (`ExampleConfig`) can be found in the particular App's `apps.py` file, located within its folder (in this case, the app is named `<appname>` and is in said folder).

YOUR TURN (5min): Connecting Django apps to the project

- Django is not aware of your `blog` app. To connect your apps to the project, you must inform the project of the existence of these apps.
- To do this, open `<projectname>/settings.py` and append to the list with your new app:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    '<appname>.apps.ExampleConfig'  
]
```

- Do note that Django built-in apps begin with `django.contrib`.
- Run the local web server again to make sure everything is working properly. You should be greeted with the same page in your browser.

YOUR TURN (10min): Creating views

- Your website has one app at this stage (`blog`); however, it does not have a view (`function`) corresponding to it.
- On paper or in a word processor document, describe the MVC architecture.
- Go to the `blogapp` app directory and locate the `views.py` file. This file will contain the view implementation of your blog app, once written.
- In your App's `views.py` file, create an `index` function that returns a `HttpResponse` object with the message `Hello World`.
 - Use the lecture notes to guide you and be careful with your `imports`.

YOUR TURN (5min): Mapping views to URLs (Part 1)

- When a user requests a page from a Django-powered website (e.g. `http://127.0.0.1:8000/blog`), Django does the following:
 - Loads the project `urls.py` file and looks for the `urlpatterns` variable.
 - Runs through each URL pattern in sequential order and stops at the first one that matches the requested URL.
 - Once the URL has been matched, Django imports and calls the given view.
- In your App's directory, create a new file called `urls.py` and update it as such:

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

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

YOUR TURN (5min): Mapping views to URLs (Part 2)

- When a new App is created, the Project urls.py file must be updated. Edit the Project urls.py file as such:

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

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

- Start the web server and navigate to `http://127.0.0.1:8000/blog` in your browser to check that your app works properly. A reminder of how to do so:

```
[user@pc]$ python3 manage.py runserver
```

If all works properly, you should see a blank page with Hello World on it.

Creating Models

- Models are Python classes that represent application data.

```
class Book(models.Model):  
    title = models.CharField(max_length=255)  
    author = models.CharField(max_length=255)  
    description = models.CharField(max_length=255)  
    rating = models.ForeignKey(Rating,  
                              on_delete=models.CASCADE)  
  
class Rating(models.Model):  
    numReviews =  
    models.IntegerField() avgRating =  
    models.FloatField()
```

Creating Models

- We created two models, one to represent a `Book` and another to represent a `Rating of a Book`.
- There is a relationship between the two Models; each `Book` can have a `singleRating`; i.e. `1 : 1` cardinality.
 - Must it? By default, in Django, yes, but we can easily change this.
- A full list of the field types that a model can contain is available at:
<https://docs.djangoproject.com/en/5.1/topics/db/models/#fields>
 - Have a good read over these – it will provide useful!
 - You will need them next week when we build our own models.

Creating and Executing Migrations (Part 1)

- Once you have added your App to your Django project, you can then generate migrations to allow you to use the SQLite database to store instances of your Model, i.e. create individual Books.
- To do this, you will need to generate a migration that describes your model to enable the database to generate, update and understand the schema.
- To create your migrations, issue the Terminal command:

```
[user@pc]$ python3 manage.py makemigrations
```

- This generates the SQL code to create the database tables corresponding to your models.

Creating and Executing Migrations (Part 2)

- The next step is to actually run the migrations to update the database.
 - If you do not have a database, Django will create one called 'db.sqlite3';
 - If you already have a database, Django will (attempt to) modify your database to sync with your models.
- Issue the following command to run the migrations:

```
[user@pc]$ python3 manage.py migrate
```

- This will create the table in database using the commands generated by makemigrations.

SQLite Database after Migration

Name	Type
▼ Tables (13)	
▶ auth_group	
▶ auth_group_permissions	
▶ auth_permission	
▶ auth_user	
▶ auth_user_groups	
▶ auth_user_user_permissions	
▼ books_book	
id	integer
title	varchar(255)
author	varchar(255)
description	varchar(255)
rating_id	integer
▼ books_rating	
id	integer
numReviews	integer
avgRating	real
▶ django_admin_log	
▶ django_content_type	
▶ django_migrations	
▶ django_session	
▶ sqlite_sequence	

Summary of Commands

- To create a new Django project:

```
[user@pc]$ django-admin startproject <projectname>
```

- To create a new App in a Django project:

```
[user@pc]$ python3 manage.py startapp <appname>
```

- To start the local development server:

```
[user@pc]$ python3 manage.py runserver
```

Summary of Commands

- To generate migrations:

```
[user@pc]$ python3 manage.py makemigrations
```

- To sync your models with your database (i.e., execute migrations)

```
[user@pc]$ python3 manage.py migrate
```