Django Framework

Topics to be covered

- About Django
- A good Web framework addresses these common concerns
- Why Django for Web Development?
- History Django
- Django with Python Overview
- Python Features
- Data Structure
- Django with MVC
- Django Working Architecture
- Django Uses
- Django Views
- Django HttpResponse
- URL
- Routers in Django
- Django Templates
- Django Forms
- Django with REST APIs
- Testing with Django
- Models

Continue..

Topics

- QuerySet
- Django Lookups
- Django Q-objects
- Cookies
- Sessions
- User Authentication
- Generic Views
- Django Caching
- Django Web Application Security

What is Django?

•Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.
•The web framework for perfectionists with deadlines.
•Primary Focus

Dynamic and database driven websites
Content based websites
Example Websites (Washingtonpost,eBay, craigslist) Google AppEngine

•A Web framework is a set of components that provide a standard way to develop websites fast and easily.
•Django's primary goal is to ease the creation of complex database-driven websites.

A good Web framework addresses these common concerns:

- It provides a method of mapping requested URLs to code that handles requests.
- It makes it easy to display, validate and redisplay HTML forms.
- It converts user-submitted input into data structures that can be manipulated conveniently.
- It helps separate content from presentation via a template system
- It conveniently integrates with storage layers
- It lets you work more productively, at a higher level of abstraction

Django does all of these things well — and introduces a number of features that raise the bar for what a Web framework should do.

Why Django for Web Development?

- Lets you divide code modules into logical groups to make it flexible to change
  MVC design pattern (MVT)
- Provides auto generated web admin to ease the website administration
- Provides pre-packaged API for common user tasks
- Provides you template system to define HTML template for your web pages to avoid code duplication
  DRY Principle
- Allows you to define what URL be for a given Function
  Loosely Coupled Principle
- Allows you to separate business logic from the HTML
  Separation of concerns
- Everything is in python (schema/settings)

History Django

- Named after famous Guitarist “Django Reinhardt” .
- Developed by Adrian Holovatyand Jacob Kaplan-moss at World Online News for efficient development .
- Open sourced in 2005
- First Version released September 3, 2008

Django with Python Overview

The framework is written in Python, a beautiful, concise, powerful, high-level programming language.
To develop a site using Django, we write Python code that uses the Django libraries.

…that encourages rapid development…

Regardless of how many powerful features it has, a Web framework is worthless if it doesn't save our time.
Django's philosophy is to do all it can to facilitate hyper-fast development.
With Django, we build Web sites in a matter of hours, not days; weeks, not years.

Python Features

- Python is an **interpreted language**,
  which means there's no need to compile code.
  Just write your program and execute it.
  In Web development, this means you can develop code and immediately see results by hitting "reload" in your Web browser.
- Python is **dynamically typed**,
  which means you don't have to worry about declaring data types for your variables.
- Python syntax is **concise yet expressive**,
  which means it takes less code to accomplish the same task than in other,
  more verbose, languages such as Java.
  One line of python usually equals 10 lines of Java.
- Python offers **powerful introspection and meta-programming** features, which make it possible to inspect and add behavior to objects at runtime.

Beyond the productivity advantages inherent in Python, Django itself makes every effort to encourage rapid development,clean and pragmatic design

Finally, Django strictly maintains a clean design throughout its own code and makes it easy to follow best Web-development practices in the applications you create.

Data structure

Data strucutre is an structures which can hold some data together. In other words they are used to store a collection of related data.

There are four built-in data structures in Python:
- list
- tuple
- dictionary
- set

List

In python, A list is a data structure that holds an ordered collection of items i.e. you can store a sequence of items in a list.

In your python shell, define a variable called "django" and store some elements value into it.
empty

listdjango = []

list of integersdjango = [1, 2, 3]

list with mixed datatypesdjango = [1, "Hello", 3.4]

nested listdjango = ["mouse", [8, 4, 6]]

Accessing Values in Lists
To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example:
>>> django = [1, 2, 3]>>> print django[0]

Tuple

Tuples are sequences, just like lists.

The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values.

Let's define a tuple:
>>> best_os = ("ubuntu", "macos", "solaris" "windows")

Accessing tuple is same as list.

Since, we defined a tuple, let's access the best operating system in same python shell.
>>> print best_os[0]ubuntu>>>

Dictionary

Dictionary is another data type built into python.

Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.
Tuples can be used as keys if they contain only strings, numbers, or tuples;

if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.

You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

Set

In simplest terms set can be defined as - set is an unordered collections of unique elements.

Or Sets are lists with no duplicate entries.

Let's say you want to collect a list of words used in a paragraph:
>>> print set("Hello Django is an open initiatives for python classes.".split())set(['initiatives', 'for', 'python', 'is', 'an', 'classes.', 'Django', 'open', 'Hello'])

Django with MVC

Django follows the “model-view-controller” (MVC) architecture.

Simply put, this is a way of developing software so that the code for defining and accessing data (the model) is separate from the business logic (the controller), which in turn is separate from the user interface (the view).

Django supports the MVC pattern.

The MVC pattern is a software architecture pattern that separates data presentation from the logic of handling user interactions(in other words, saves you stress:),

It has been around as a concept for a while, and has invariably seen an exponential growth in use since its inception.

It has also been described as one of the best ways to create client-server applications, all of the best frameworks for web are all built around the MVC concept

Continue..

Overview MVC

Model: This handles your data representation, it serves as an interface to the data stored in the database itself, and also allows you to interact with your data without having to get perturbed with all the complexities of the underlying database.

View: As the name implies, it represents what you see while on your browser for a web application or In the UI for a desktop application.

Controller: provides the logic to either handle presentation flow in the view or update the model's data i.e it uses programmed logic to figure out what is pulled from the database through the model and passed to the view, also gets information from the user through the view and implements the given logic by either changing the view or updating the data via the model , To make it more simpler, see it as the engine room.

Here is a rundown of steps involved in an MVC blog application.

1. Web browser or client sends the request to the web server, asking the server to display a blog post.
2. The request received by the server is passed to the controller of the application.
3. The controller asks the model to fetch the blog post.
4. The model sends the blog post to the controller.
5. The controller then passes the blog post data to the view.
6. The view uses blog post data to create an HTML page.
7. At last, the controller returns the HTML content to the client.

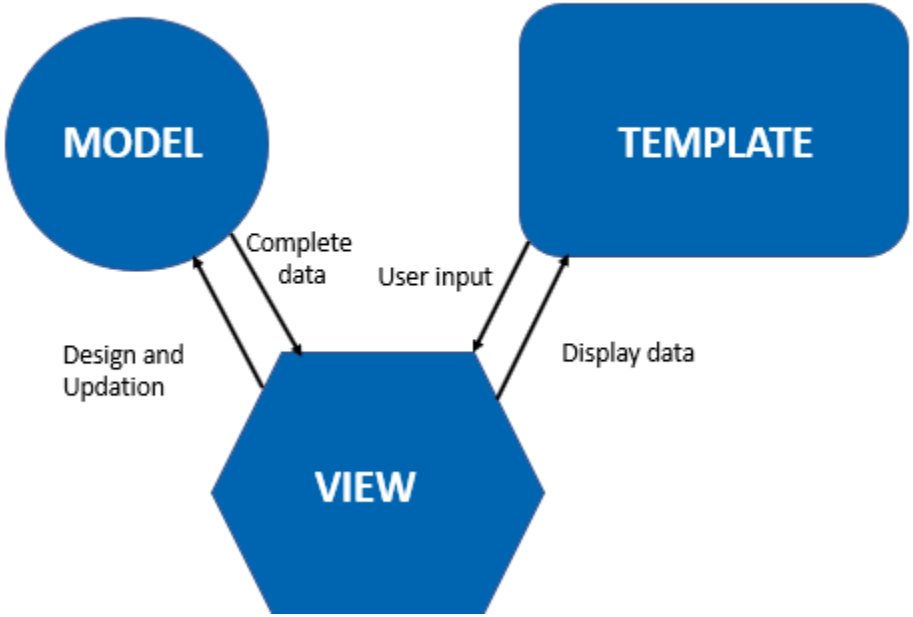
The MVC pattern not only helps us to create and maintain a complex application.
It really shines when it comes to separation of concerns.

For example, In a web development company, there are web designers and there are developers. The job of web designers is to create views. The developers take these views and incorporate them with models and controllers.

Django MTV

- Django follows MVC pattern very closely but it uses slightly different terminology.
- Django is essentially an MTV (Model-Template-View) framework.
- Django uses the term Templates for Views and Views for Controller.
- In other words, in Django views are called templates and controllers are called views.
- Hence our HTML code will be in templates and Python code will be in views and models.

MTV Architecture Components (Model, Template, and View)



Django Model

In Django, the model does the linking to the database and each model gets mapped to a single table in the database.

These fields and methods are declared under the file models.py
With this linking to the database, we can actually each and every record or row from that particular table and can perform the DML operations on the table.

Django.db.models.
The model is the subclass that is used here.
We can use the import statement by defining as from django.db import models.
So after defining our database tables, columns and records; we are going to get the data linked to our application by defining the mapping in settings.py file under the INSTALLED_APPS.

Django Template

This template helps us to create a dynamic website in an easy manner.

The dynamic website deals with dynamic data.

Dynamic data deals with a scenario where each user is displayed with their personalized data; as Facebook feeds, Instagram feeds, etc.
The configuration of the template is done in settings.py file under INSTALLED_APPS. So python code would search for the files under the template subdirectory.

We can create an HTML file or import any dynamic web page from the browser and place it under the template folder.
And after that our usual linking of this file in urls.py and views.py to get a response is mandatory.

In this way after linking all these together and running the server, we can get our web application ready.

Django View

This is the part where actually we would be mentioning our logic.

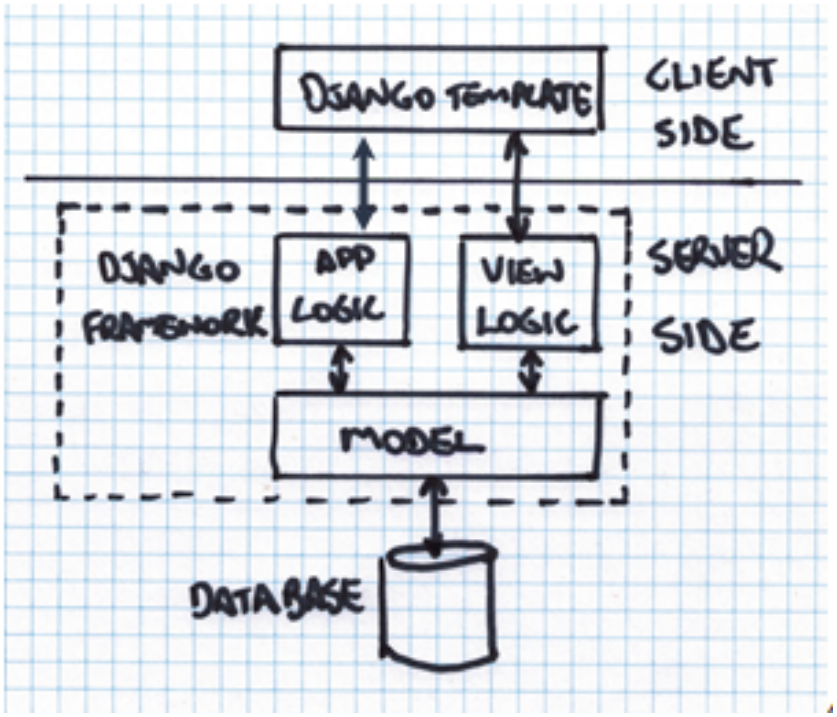
The coding is done through the python file views.py

This view also sends responses to the user when the application is used, to understand briefly, we can say that this view.py can deal with HttpResponse.

Now, after creating a view, how can we link it to our application?
How do you think that the system is going to understand to display a particular view?

This can be done by mapping the views.py in urls.py file. As already mentioned, urls.py keeps track of all those different pages that we created and hence map each of them.

Django Working Architecture



Continue..

Django Working Architecture

We have some components and two regions i.e., server side and client side.

Here you will notice that the View is on the server-side part while the template is on the client side.

Now, when we request for the website, the interface through which we use to make that request via our browser was the Template.

Then that request transmits to the server for the management of view file.
Django is literally a play between the requests and responses.

So whenever our Template is updating it's the input (request) we sent from here which on the server was seen by the View.

And, then it transports to the correct URL. It's one of the important components of Django MTV architecture.
There, the URL mapping in Django is actually done in regular expressions.

These expressions are much more understandable than IP addresses.

Continue..

Django Working Architecture

Now after the sending of a request to the correct URL, the app logic applies and the model initiates to correct response to the given request.

Then that particular response is sent back to the View where it again examines the response and transmits it as an HTTP response or desired user format.

Then, it again renders by the browser via Templates.

Django Uses

1. Django is time-tested

It's been 13 years Django started developing its framework and the first release of open source commit as it was under development quite a long time before release. During these years it had many releases some of them have new features other releases focuses on security enhancements etc. Django is the first framework to respond to new issues and vulnerabilities and alter other frameworks to make patches to frameworks. The Latest release of it is focusing on new features and boundary case problems.

2. Application Development

Django was developed by online news operation team with an aim to create web applications using the Python programming language. The framework has templates, libraries, and APIS which work together. In general, applications developed using Django can be upgraded with minimal cost, changes, and additions and it make a lot of web development easier.

Continue..

Django Uses

3. Easy to Use

Django uses Python programming language which is a popular language in 2015 and now most choosing language by programmers who are learning to code and applications of Django framework is widely used as it is free and open-source, developed and maintained by a large community of developers. It means we can find answers to the problems easily using Google.

4. Operating System Dependent

Django framework runs on any platform like PC, Windows, Mac, Linux etc. It provides a layer between the developer and database called ORM (object-relational mapper) which makes it possible to move or migrate our applications to other major databases with few lines of code change.

Continue..

Django Uses

5. Excellent Documentation for real-world application

Applications of Django has one of the best documentation for its framework to develop different kinds of real-world applications whereas many other frameworks used an alphabetical list of modules, attributes, and methods. This is very useful for quick reference for developers when we had confused between two methods or modules but not for fresher's who are learning for the first time. It's a difficult task for Django developers to maintain the documentation quality as it is one of the best open source documentation for any framework.

Continue..

Django Uses

6. Scalable and Reliable

As Django is a well-maintained web application framework and widely used across the industries so cloud providers taking all measures to provide support to run Django applications easily and quickly on cloud platforms. It means, once Django applications deployed then it can be managed by an authorized developer with a single command in a cloud environment. As Django developers are working in the same development environment for a long time so they will grow and expertise in these areas which means applications developed, websites created are getting better day by day, more functional, efficient and reliable.

7. Community Support

Django community is one of the best communities out there as it is governed by the Django software foundation which had some rules like for event there is a code of conduct. Django communities will have IRC and mailing list most welcome, even it may have bad appeals it will rectify immediately. Django offers stability, packages, documentation and a good community.

Continue..

Django Uses

8. DRY – Don’t repeat yourself

Django framework follows don’t repeat yourself principle as it concentrates on getting most out of each and every line of code by which we can spend less time on debugging or code re-orientation etc. In general DRY code means all uses of data change simultaneously rather than a need to be replicated and its fundamental reason to use of variables and functions in all programming.

Continue..

Django Uses

9. Batteries of Django

Django framework has everything to build a robust framework with main features as below:
Template layers,
Forms, development process,
Views layers, security,
Model layers, python compatibility,
Localization, performance, and optimization
Geographic framework, common tools for web application development
Other core functionalities required for websites.
As Django can be used to build any type of website with help of its frameworks like content management, Wikipedia pages, social networking applications, chat applications, and websites like Mozilla, Instagram, Pinterest, BitBucket etc. Django can work with any client-server applications and able to deliver content in any form (HTML, text, JSON, XML, RSS etc.)

Continue..

Django Uses

10. Django Benefits

With the uses of Django framework, we can develop and deploy web applications within hours as it takes care of much of the hassle of web development. Django is very fast, fully loaded such as it takes care of user authentication, content administration, security as Django takes it very seriously and helps to avoid SQL injection, cross-site scripting etc. and scalable as applications can be scalable to meet high demands and used to build any type of applications that's why we call it as versatile framework. We can build different applications from content management to social networking websites using Django framework. It offers lots of resources and good documentation which helps new learners to learn and experienced people for reference.

Continue..

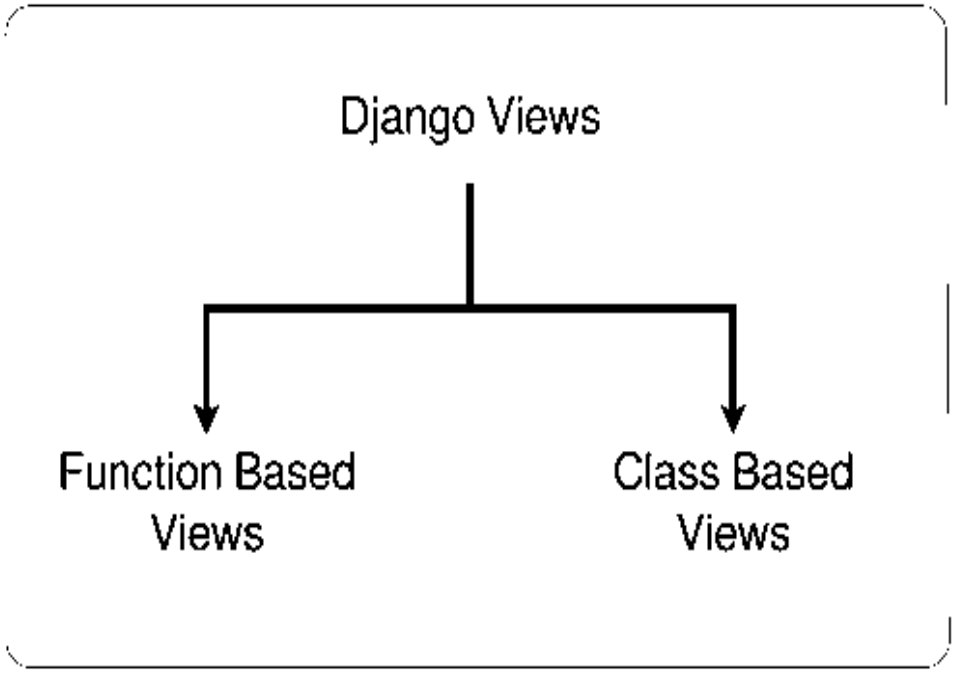
Django Views

A view function, or “view” for short, is simply a Python function that takes a web request and returns a web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, etc.

Types of Views

Django views are divided into two major categories :-

- Function Based Views
- Class Based Views



Function Based Views

Function based views are writer using a function in python which receives as an argument HttpRequest object and returns an HttpResponse Object.

Function based views are generally divided into 4 basic strategies, i.e., CRUD (Create, Retrieve, Update, Delete).

CRUD is the base of any framework one is using for development.

Class Based Views

- Class-based views provide an alternative way to implement views as Python objects instead of functions.

- They do not replace function-based views, but have certain differences and advantages when compared to function-based views.

- Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.

Django HttpResponse

- HttpResponse is a response class with string data.

- While HttpRequest is created by Django, HttpResponse is created by programmer.

- HttpResponse has subclasses including JsonResponse, StreamingHttpResponse, and FileResponse.

HttpRequest in Django

- **Request and Response Objects** –

- Django uses request and response objects to pass state through the system.

- When a page is requested, Django creates an HttpRequest object that contains metadata about the request.
- Then Django loads the appropriate view, passing the HttpRequest as the first argument to the view function.
- Each view is responsible for returning an HttpResponse object.

HttpRequest and HttpResponse Object

HttpRequest Attributes – Django

ATTRIBUTE	DESCRIPTION
HttpRequest.scheme	A string representing the scheme of the request (HTTP or HTTPs usually).
HttpRequest.body	It returns the raw HTTP request body as a byte string.
HttpRequest.path	It returns the full path to the requested page does not include the scheme or domain.
HttpRequest.path_info	It shows path info portion of the path.
HttpRequest.method	It shows the HTTP method used in the request.
HttpRequest.encoding	It shows the current encoding used to decode form submission data.
HttpRequest.content_type	It shows the MIME type of the request, parsed from the CONTENT_TYPE header.

HttpRequest and HttpResponse Object

HttpRequest Attributes – Django

ATTRIBUTE	DESCRIPTION
HttpRequest.content_params	It returns a dictionary of key/value parameters included in the CONTENT_TYPE header.
HttpRequest.GET	It returns a dictionary-like object containing all given HTTP GET parameters.
HttpRequest.POST	It is a dictionary-like object containing all given HTTP POST parameters.
HttpRequest.COOKIES	It returns all cookies available.
HttpRequest.FILES	It contains all uploaded files.
HttpRequest.META	It shows all available Http headers.
HttpRequest.resolver_match	It contains an instance of ResolverMatch representing the resolved URL.

HttpRequest and HttpResponse Object

HttpRequest Methods – Django

ATTRIBUTE	DESCRIPTION
HttpRequest.get_host()	It returns the original host of the request.
HttpRequest.get_port()	It returns the originating port of the request.
HttpRequest.get_full_path()	It returns the path, plus an appended query string, if applicable.
HttpRequest.build_absolute_uri (location)	It returns the absolute URI form of location.
HttpRequest.get_signed_cookie (key, default=RAISE_ERROR, salt=”, max_age=None)	It returns a cookie value for a signed cookie, or raises a django.core.signing.BadSignature exception if the signature is no longer valid.
HttpRequest.is_secure()	It returns True if the request is secure; that is, if it was made with HTTPS.
HttpRequest.is_ajax()	It returns True if the request was made via an XMLHttpRequest.

What is a URL?

URL stands for Uniform Resource Locator.

It is the address used by your server to search for the right webpage.

URLs in Django

How does Django Server interpret URLs?

Django interprets URLs in a rather different way, the URLs in Django are in the format of regular expressions, which are easily readable by humans than the traditional URLs of PHP frameworks.

Regular Expressions

A Regular Expression also called RegEx is a format for search pattern in URLs which is much cleaner and easy to read for the humans and is very logical. That's important because it makes the process of SEO much easier then it would be with the traditional URL approach which contains much more special characters.

Routers in Django

Routers are used with **ViewSets** in django rest framework to auto config the urls.

Routers provides a simple, quick and consistent way of wiring ViewSet logic to a set of URLs.

Router automatically maps the incoming request to proper viewset action based on the request method type(i.e GET, POST, etc). Let's start using routers.

Two types of Routers in Django REST

knowledge hut

1. SimpleRouter
    - It includes actions for all actions (i.e `list` , `create` , `retrieve` , `update` , `partial_update` and `destroy`).
    - Let's see how it configures the urls

URL Style	HTTP Method	Action	URL Name
{prefix}/	GET	list	{basename}-list
	POST	create	
{prefix}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=False)` decorated method	{basename}-{url_name}
{prefix}/{lookup}/	GET	retrieve	{basename}-detail
	PUT	update	
	PATCH	partial_update	
	DELETE	destroy	
{prefix}/{lookup}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=True)` decorated method	{basename}-{url_name}

Two types of Routers in Django REST

2. DefaultRouter

- It also includes all actions just like `SimpleRouter` and it additionally provides a default root view which returns a response containing hyperlinks to all the list views.
- Let's see how it configures the urls

URL Style	HTTP Method	Action	URL Name
[.format]	GET	automatically generated root view	api-root
{prefix}/[.format]	GET	list	{basename}-list
	POST	create	
{prefix}/{url_path}/[.format]	GET, or as specified by `methods` argument	`@action(detail=False)` decorated method	{basename}-{url_name}
{prefix}/{lookup}/[.format]	GET	retrieve	{basename}-detail
	PUT	update	
	PATCH	partial_update	
	DELETE	destroy	
{prefix}/{lookup}/{url_path}/[.format]	GET, or as specified by `methods` argument	`@action(detail=True)` decorated method	{basename}-{url_name}

Django Templates

Template Fundamentals –

- Django provides a convenient way to generate dynamic HTML pages by using its template system. A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Why Django Template?

- In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser. We know that HTML is a static markup language, while Python is a dynamic programming language. Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

Loading Template

To load the template, call get_template() method as we did below and pass template name

Template Tags

In a template, Tags provide arbitrary logic in the rendering process. For example, a tag can output content, serve as a control structure e.g. an "if" statement or a "for" loop, grab content from a database etc.

Template Filters

- Filters the contents of the block through one or more filters. Multiple filters can be specified with pipes and filters can have arguments, just as in variable syntax. It is to be noted that the block includes all the text between the filter and endfilter tags

Template Inheritance in Django

The include tag allows us to include the contents of a template inside another template making the inheritance of template possible. Here is the syntax of the include tag:

{% include template_name %}

The template_name could be a string or a variable.

Rendering of Templates

A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This article revolves around how to render an HTML page from Django using views. Django has always been known for its app structure and ability to manage applications easily. Let’s dive in to see how to render a template file through a Django view.

DJANGO FORMS

- Unless you're planning to build websites and applications that do nothing but publish content, and don't accept input from your visitors, you're going to need to understand and use forms.

•Django provides 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 HTML forms.
- It describes a form and how it works and appears.
- It is similar to the ModelForm class that creates a form by using the Model, but it does not require the Model.
- Each field of the form class map to the HTML form <input> element and each one is a class itself, it manages form data and performs validation while submitting the form.

Commonly used fields

Commonly used fields and their details are given in the below table

Name	Class	HTML Input	Empty value
BooleanField	class BooleanField(**kwargs)	CheckboxInput	False
CharField	class CharField(**kwargs)	TextInput	Whatever you've given as empty_value.
ChoiceField	class ChoiceField(**kwargs)	Select	'' (an empty string)
DateField	class DateField(**kwargs)	DateInput	None
DateTimeField	class DateTimeField(**kwargs)	DateTimeInput	None
DecimalField	class DecimalField(**kwargs)	NumberInput	None
EmailField	class EmailField(**kwargs)	EmailInput	'' (an empty string)
FileField	class FileField(**kwargs)	ClearableFileInput	None
ImageField	class ImageField(**kwargs)	ClearableFileInput	None

Form Authentication

Django authentication provides both authentication and authorization together and is generally referred to as the authentication system, as these features are somewhat coupled

The primary attributes of the default user are:

- username
- password
- email
- first_name
- last_name

Changing passwords

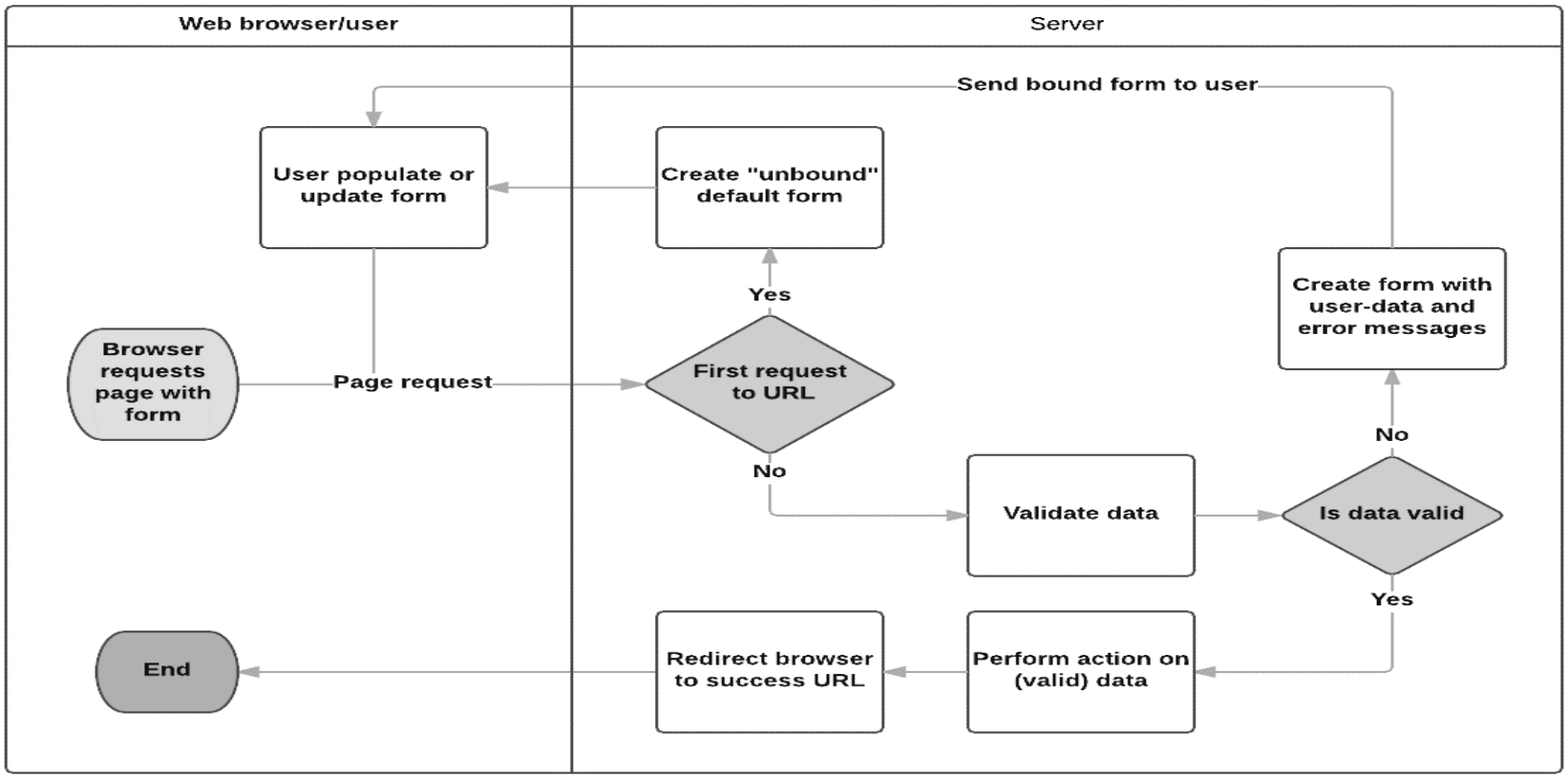
- Django does not store raw (clear text) passwords on the user model, but only a hash. You can also change a password programmatically, using set_password():

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

Django Form Processing Technique

- Display the default form the first time it is requested by the user.
  - The form may contain blank fields (e.g. if you're creating a new record), or it may be pre-populated with initial values (e.g. if you are changing a record, or have useful default initial values).
  - The form is referred to as *unbound* at this point, because it isn't associated with any user-entered data (though it may have initial values).
- Receive data from a submit request and bind it to the form.
  - Binding data to the form means that the user-entered data and any errors are available when we need to redisplay the form.
- Clean and validate the data.
  - Cleaning the data performs sanitization of the input (e.g. removing invalid characters that might be used to send malicious content to the server) and converts them into consistent Python types.
  - Validation checks that the values are appropriate for the field (e.g. are in the right date range, aren't too short or too long, etc.)
- If any data is invalid, re-display the form, this time with any user populated values and error messages for the problem fields.
- If all data is valid, perform required actions (e.g. save the data, send an email, return the result of a search, upload a file, etc.)
- Once all actions are complete, redirect the user to another page

Django Form Processing Technique



Django with REST APIs

- REST framework is a collaboratively funded project. If you use REST framework commercially, we strongly encourage you to invest in its continued development by signing up for a paid plan.

- The initial aim is to provide a single full-time position on REST framework.

- Every single sign-up makes a significant impact towards making that possible.

Reason for using REST Framework

Django REST framework is a powerful and flexible toolkit for building Web APIs.
Some reasons you might want to use REST framework:

- The Web browsable API is a huge usability win for your developers.
- Authentication policies including optional packages for OAuth1a and OAuth2.
- Serialization that supports both ORM and non-ORM data sources.
- Customizable all the way down - just use regular function-based views if you don't need the more powerful features.
- Extensive documentation, and great community support

Testing with Django

Automated testing is an extremely useful bug-killing tool for the modern Web developer.
You can use a collection of tests – a **test suite** – to solve, or avoid, a number of problems:
When you're writing new code, you can use tests to validate your code works as expected.
When you're refactoring or modifying old code, you can use tests to ensure your changes haven't affected your application's behavior unexpectedly.

Testing a Web application is a complex task, because a Web application is made of several layers of logic – from HTTP-level request handling, to form validation and processing, to template rendering.

With Django's test-execution framework and assorted utilities, you can simulate requests, insert test data, inspect your application's output and generally verify your code is doing what it should be doing.

Testing with Django

The preferred way to write tests in Django is using the unittest module built-in to the Python standard library. This is covered in detail in the Writing and running tests document.
You can also use any *other* Python test framework; Django provides an API and tools for that kind of integration.

Unit Testing with Django

- **What is Unit Testing?**

- Unit Testing is the first level of software testing where the smallest testable parts of a software are tested. This is used to validate that each unit of the software performs as designed. The unittest test framework is python's xUnit style framework.

- OOP concepts supported by unittest framework are:

- **Test fixture**
- **Test case**
- **Test suite**
- **Test runner**

Testing Tools

Django provides a small set of tools that come in handy when writing tests.
The test client

The test client is a Python class that acts as a dummy Web browser, allowing you to test your views and interact with your Django-powered application programmatically.

Some of the things you can do with the test client are:
Simulate GET and POST requests on a URL and observe the response – everything from low-level HTTP (result headers and status codes) to page content.

See the chain of redirects (if any) and check the URL and status code at each step.

Test that a given request is rendered by a given Django template, with a template context that contains certain values.

Note that the test client is not intended to be a replacement for Selenium or other “in-browser” frameworks. Django’s test client has a different focus.

Testing Tools

In short:
Use Django's test client to establish that the correct template is being rendered and that the template is passed the correct context data.

Use in-browser frameworks like Selenium to test *rendered* HTML and the *behavior* of Web pages, namely JavaScript functionality.

Django also provides special support for those frameworks; see the section on LiveServerTestCase for more details.
A comprehensive test suite should use a combination of both test types.

Advanced Testing

The request factory
class RequestFactory
The RequestFactory shares the same API as the test client.

However, instead of behaving like a browser, the RequestFactory provides a way to generate a request instance that can be used as the first argument to any view.

This means you can test a view function the same way as you would test any other function – as a black box, with exactly known inputs, testing for specific outputs.

The API for the RequestFactory is a slightly restricted subset of the test client API:
It only has access to the HTTP methods get(), post(), put(), delete(), head(), options(), and trace().

These methods accept all the same arguments *except* for follow. Since this is just a factory for producing requests, it's up to you to handle the response.

It does not support middleware. Session and authentication attributes must be supplied by the test itself if required for the view to function properly.

Debugging

- **pdb (The Python Debugger) and ipdb (IPython pdb): pdb** is like the JavaScript debugger. You can drop an import pdb; pdb.set_trace() in your code and it will provide a REPL for you to step through your code. **ipdb** is like an enhanced version pdb. Useful features include syntax highlighting and tab completion. It is enormously helpful to be able to inspect objects.

- **Debugging Django in Docker:** If you are using docker, you might notice that dropping a pdb or ipdb into your code does nothing. To fix this, you'll have to run docker with service ports enabled. For example:

- docker-compose -f docker-compose.yml run --rm --service-ports name_of_container

Models

- A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

- The basics:

- Each model is a Python class that subclasses django.db.models.Model.
- Each attribute of the model represents a database field.
- With all of this, Django gives you an automatically-generated database-access API; see Making queries.

Django Model Fields

- Fields: The most important part of a model – and the only required part of a model – is the list of database fields it defines. Fields are specified by class attributes. Be careful not to choose field names that conflict with the models API like clean, save, or delete

Model Field options

- Each field takes a certain set of field-specific arguments (documented in the model field reference). For example, CharField (and its subclasses) require a max_length argument which specifies the size of the VARCHAR database field used to store the data.

- Some of the commonly used ones are –

- null
- blank
- choices
- default
- help_text
- Primary key
- unique

Table Naming Conventions

- Use lowercase table names for MariaDB and MySQL: It is strongly advised that you use lowercase table names when you override the table name via db_table, particularly if you are using the MySQL backend. See the MySQL notes for more details.

- Table name quoting for Oracle: In order to meet the 30-char limitation Oracle has on table names, and match the usual conventions for Oracle databases, Django may shorten table names and turn them all-uppercase. To prevent such transformations, use a quoted name as the value for db_table:
- db_table = '"name_left_in_lowercase"‘

- Such quoted names can also be used with Django’s other supported database backends; except for Oracle, however, the quotes have no effect. See the Oracle notes for more details.

- order_with_respect_to implicitly sets the ordering option: Internally, order_with_respect_to adds an additional field/database column named _order and sets the model’s ordering option to this field. Consequently, order_with_respect_to and ordering cannot be used together, and the ordering added by order_with_respect_to will apply whenever you obtain a list of objects of this model.

- Changing order_with_respect_to: Because order_with_respect_to adds a new database column, be sure to make and apply the appropriate migrations if you add or change order_with_respect_to after your initial migrate.

Creating Django Models

- Create a Django Application
- Add the model
- Update Settings
- Make Migrations (as per requirement)

Adding the Django Application to Your Project:

To add a new Django application to an existing project, the steps are –

- On the main menu, choose Tools | Run manage.py task

- In the Enter manage.py task name dialog, type startapp. Note suggestion list that appears under the dialog after entering the first letter, and shrinks as you type to show the exact match only.

- In the dialog that opens, type the name of the new Django application

Inserting data to a Django Table

- You need to create an instance of the model class, instantiate it with the appropriate values (name and address) and then call save(), which composes the appropriates SQL INSERT statement under the hood.

Understanding QuerySets

Object-relational mappers (or ORMs for short), such as the one that comes built-in with Django, make it easy for even new developers to become productive without needing to have a large body of knowledge about how to make use of relational databases.

They abstract away the details of database access, replacing tables with declarative model classes and queries with chains of method calls

Slicing a Queryset

- Slicing: A QuerySet can be sliced, using Python's array-slicing syntax. Slicing an unevaluated QuerySet usually returns another unevaluated QuerySet, but Django will execute the database query if you use the "step" parameter of slice syntax, and will return a list.

- Slicing a QuerySet that has been evaluated also returns a list.

Ordering Querysets

- order_by(*fields): By default, results returned by a QuerySet are ordered by the ordering tuple given by the ordering option in the model's Meta. You can override this on a per-QuerySet basis by using the order_by method.

- Example:

```
ordered_authors = Author.objects.order_by('-score', 'last_name')[:30]
```

Django Lookups

- Django offers a wide variety of built-in lookups for filtering (for example, exact and icontains).

- Lookups define the corresponding ajax views used by the auto-completion fields and widgets.

- They take in the current request and return the JSON needed by the jQuery auto-complete plugin.

Types of Lookups

Django-selectable uses a registration pattern similar to the Django admin. Lookups should be defined in a lookups.py in your application's module. Once defined you must register in with django-selectable. All lookups must extend from selectable.base.LookupBase which defines the API for every lookup.

- LookupBase.get_query(request, term)
- LookupBase.get_item_label(item)
- LookupBase.get_item_id(item)
- LookupBase.split_term(term)
- LookupBase.get_item_value(item)
- LookupBase.get_item(value)
- LookupBase.create_item(value)
- LookupBase.format_item(item)
- LookupBase.format_results(self, raw_data, options)
- LookupBase.paginate_results(results, options)

Django Q-objects:

- Q object encapsulates a SQL expression in a Python object that can be used in database-related operations. Using Q objects we can make complex queries with less and simple code.

- For example, this Q object filters whether the question starts wiht 'what':

- from django.db.models import Q
- Q(question__startswith='What')

- Q objects are helpfull for complex queries because they can be combined using logical operators and(&), or(|), negation(~)

Cookies in Django

•Sometimes you might want to store some data on a per-site-visitor basis as per the requirements of your web application.

•Always keep in mind, that cookies are saved on the client side and depending on your client browser security level, setting cookies can at times work and at times might not.

What are Sessions?

- The session is a semi-permanent and two-way communication between the server and the browser.

- **Sessions framework** can be used to provide persistent behaviour for anonymous users in the website.Sessions are the mechanism used by Django for you store and retrieve data on a per-site-visitor basis.Django uses a cookie containing a special session id.

- To enable the session in the django, you will need to make sure of two things in settings.py:

- MIDDLEWARE_CLASSES has 'django.contrib.sessions.middleware.SessionMiddleware' activated
- INSTALLED_APPS has 'django.contrib.sessions' added.

Handling Sessions in Django

- Django provides full support for anonymous sessions.

•The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the sending and receiving of cookies. Cookies contain a session ID – not the data itself (unless you're using the cookie based backend).

User authentication in Django

- Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

- The Django authentication system handles both authentication and authorization.

•Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do.

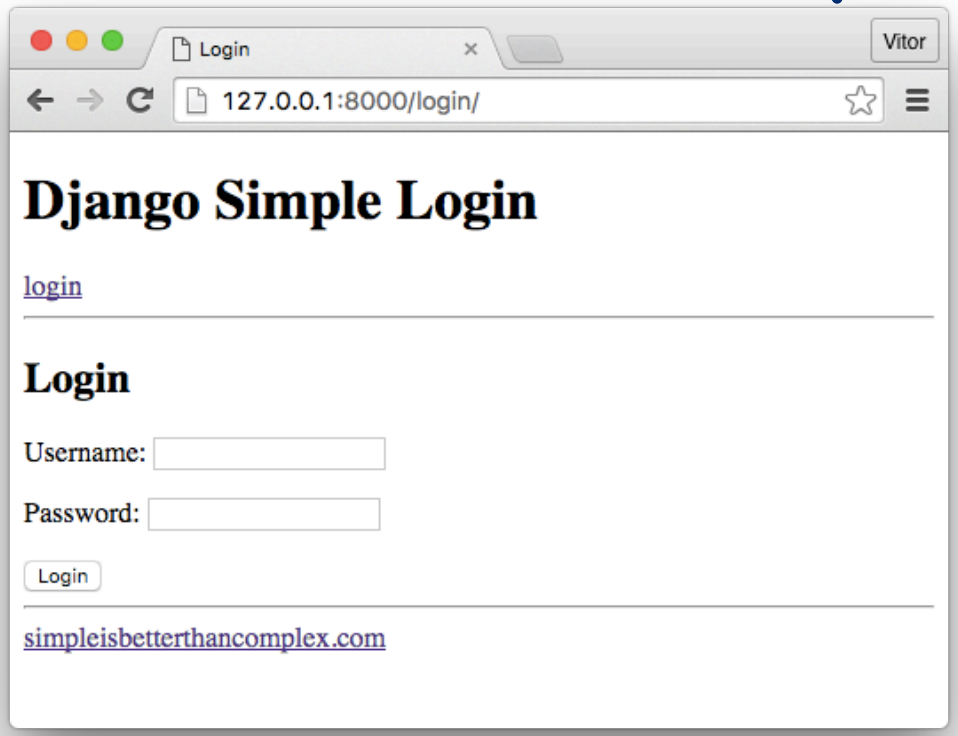
• Here the term authentication is used to refer to both tasks.

Django Login and Logouts

The steps are:

i. **Configure the URL routes**

ii. ***Building your own Login template***



Managing Permissions in Django

Different ways can be implemented to manage permissions in Django apps. These are –

1.Native Django Permissions
2.Django Gurdian
3.Django Role Permission

GENERIC VIEWS

Simple Generic Views –

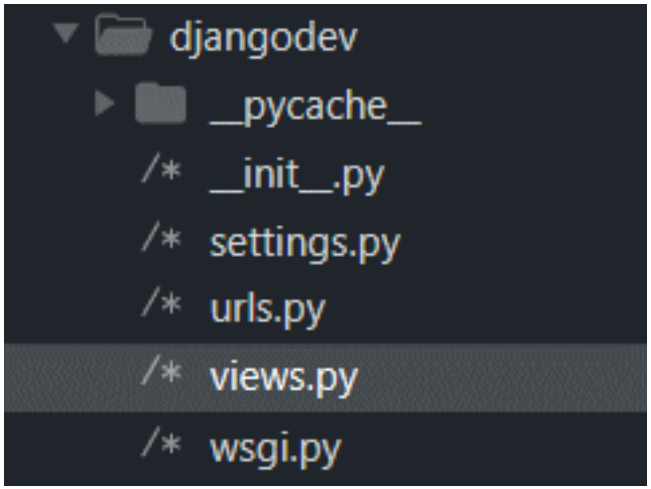
- In some cases, writing views, as we have seen earlier is really heavy. Imagine you need a static page or a listing page. Django offers an easy way to set those simple views that is called generic views.

- Unlike classic views, generic views are classes not functions. Django offers a set of classes for generic views in django.views.generic, and every generic view is one of those classes or a class that inherits from one of them.

Redirect Page in Django

- Django has a specially made redirects function, just for this sole purpose. As we have learned in previous tutorials, every request has a response. Requests are always Http Requests, but there can be different Responses like we returned HttpResponse in our first views.py.

- We will now return, an HttpResponseRedirect instance or class object to redirect the User. You need to write redirect in place of render in the views file. Let's edit views.py file.



Redirect Page in Django (cont.…)

The redirect():

- This function is capable of taking more than just URLs, you can redirect your users to models, redirect to a particular URL, while also passing the key values and other needed information. But, function-based redirects are most easy to use and are quite sufficient.

Generic Views Editing

- django.views.generic.edit.CreateView
- django.views.generic.edit.UpdateView
- django.views.generic.edit.DeleteView

Django Caching

Caching is one of those methods which a website implements to become faster. It is cost efficient and saves CPU processing time.

what kind of websites will actually benefit from caching? The answer is dynamic websites.

What are Dynamic Websites?
Dynamic websites are a collection of webpages which are generated dynamically.

Continue..

Django Data Caching

•Caching is important for a dynamic website.
•Since Django is a dynamic framework, it comes with built-in options for you to manage your cache.
•Django allows caching on different caching spaces as well as on different parts of a website.
•There can be certain parts of the website which can demand more CPU time and granularity to implement caching on them individually.
•This makes caching efficient.

Continue..

Django Caching –**Memcached**

The fastest, most efficient type of cache supported natively by Django, **Memcached** is an entirely memory-based cache server, originally developed to handle high loads at LiveJournal.com and subsequently open-sourced by Danga Interactive.

It is used by sites such as Facebook and Wikipedia to reduce database access and dramatically increase site performance.

Memcached runs as a daemon and is allotted a specified amount of RAM.

All it does is provide a fast interface for adding, retrieving and deleting data in the cache.

All data is stored directly in memory, so there's no overhead of database or filesystem usage.

Continue..

Django Caching –**Memcached**

After installing Memcached itself, you'll need to install a Memcached binding.

There are several Python Memcached bindings available; the two most common are **python-memcached** and **pylibmc**.

To use Memcached with Django:

Set **BACKEND** to django.core.cache.backends.memcached.MemcachedCache or django.core.cache.backends.memcached.PyLibMCCache (depending on your chosen memcached binding)

Set **LOCATION** to ip:port values, where ip is the IP address of the Memcached daemon and port is the port on which Memcached is running, or to a unix:path value, where path is the path to a Memcached Unix socket file.

Types of Caching in Django

There are different options of caching in Django:

1. Database Caching

Database Caching is also one of the viable options when there is a fast database server. This type of Caching is more common and is most easily applicable.

2. File-System Caching

The File-based Caching means storing caches as individual files. The engine we use for file-system will serialize files. That makes it a better way to manage files.

3. Local-Memory Caching

Django has a default caching system in the form of local-memory caching. It is very powerful and robust. This system can handle multi-threaded processes and is efficient.

4. Dummy Caching

Django also provides you with dummy caching. This implementation allows you to develop a website in a production environment.

5. Custom Cache System

Django supports a wide variety of caching implementations. If you want something that isn't natively supported by Django like making your own cache backend, then you might want this setting which is really helpful.

Django Caching Options

In Django there are four different caching options in descending order of granularity:
1) The **per-site cache** is the simplest to setup and caches your entire site.
2) The **per-view cache** lets you cache individual views.
3) **Template fragment caching** lets you specify a specific section of a template to cache.
4) The **low-level cache API** lets you manually set, retrieve, and maintain specific objects in the cache.

Data Caching for Performance

- To cache something is to save the result of an expensive calculation, so that you don't perform it the next time you need it. Following is a pseudo code that explains how caching works

```
given a URL, try finding that page in the cache

if the page is in the cache:
   return the cached page
else:
   generate the page
   save the generated page in the cache (for next time)
   return the generated page
```

Data Caching for Performance (cont.…)

- Django comes with its own caching system that lets you save your dynamic pages, to avoid calculating them again when needed. The good point in Django Cache framework is that you can cache −

- The output of a specific view.

- A part of a template.

- Your entire site

Data Caching for Performance (cont….)

- Setting Up Cache in Memory

The cache system requires a small amount of setup. Namely, you have to tell it where your cached data should live – whether in a database, on the filesystem or directly in memory. This is an important decision that affects your cache's performance; yes, some cache types are faster than others.
Your cache preference goes in the **CACHES** setting in your settings file.

Data Caching for Performance (cont.…)

Caching a View

If you don’t want to cache the entire site you can cache a specific view. This is done by using the **cache_page** decorator that comes with Django.
Let us say we want to cache the result of the **viewArticles** view −

```
from django.views.decorators.cache import cache_

page @cache_page(60 * 15)

def viewArticles(request, year, month):

text = "Displaying articles of : %s/%s"%(year, month)

return HttpResponse(text)
```

As you can see **cache_page** takes the number of seconds you want the view result to be cached as parameter.
In our example above, the result will be cached for 15 minutes.

Django Web Application Security

Django has effective protections against a number of common threats, including XSS and CSRF attacks.

Common Threat/Protection

Cross site scripting (XSS)
XSS is a term used to describe a class of attacks that allow an attacker to inject client-side scripts *through* the website into the browsers of other users. This is usually achieved by storing malicious scripts in the database where they can be retrieved and displayed to other users, or by getting users to click a link that will cause the attacker's JavaScript to be executed by the user's browser.

Django's template system protects you against the majority of XSS attacks by escaping specific characters that are "dangerous" in HTML.

Continue
..

Common Threat/Protection

Cross site request forgery (CSRF) protection
CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent. For example consider the case where we have a hacker who wants to create additional authors for our LocalLibrary.

Django generates a user/browser specific key and will reject forms that do not contain the field, or that contain an incorrect field value for the user/browser.
To use this type of attack the hacker now has to discover and include the CSRF key for the specific target user. They also can't use the "scattergun" approach of sending a malicious file to all librarians and hoping that one of them will open it, since the CSRF key is browser specific.
Django's CSRF protection is turned on by default. You should always use the {% csrf_token %} template tag in your forms and use POST for requests that might change or add data to the database.

Continue
..

Common Threat/Protection

SQL injection protection

SQL injection vulnerabilities enable malicious users to execute arbitrary SQL code on a database, allowing data to be accessed, modified, or deleted irrespective of the user's permissions. In almost every case you'll be accessing the database using Django's querysets/models, so the resulting SQL will be properly escaped by the underlying database driver. If you do need to write raw queries or custom SQL then you'll need to explicitly think about preventing SQL injection.

Continue
..

Common Threat/Protection

Clickjacking protection

In this attack a malicious user hijacks clicks meant for a visible top level site and routes them to a hidden page beneath. This technique might be used, for example, to display a legitimate bank site but capture the login credentials in an invisible <iframe> controlled by the attacker. Django contains clickjacking protection in the form of the X-Frame-Options middleware which, in a supporting browser, can prevent a site from being rendered inside a frame.

Continue
..

Common Threat/Protection

Enforcing SSL/HTTPS

SSL/HTTPS can be enabled on the web server in order to encrypt all traffic between the site and browser, including authentication credentials that would otherwise be sent in plain text (enabling HTTPS is highly recommended). If HTTPS is enabled then Django provides a number of other protections you can use: SECURE_PROXY_SSL_HEADER can be used to check whether content is secure, even if it is incoming from a non-HTTP proxy.

SECURE_SSL_REDIRECT is used to redirect all HTTP requests to HTTPS.

Use HTTP Strict Transport Security (HSTS). This is an HTTP header that informs a browser that all future connections to a particular site should always use HTTPS. Combined with redirecting HTTP requests to HTTPS, this setting ensures that HTTPS is always used after a successful connection has occurred. HSTS may either be configured with SECURE_HSTS_SECONDS and SECURE_HSTS_INCLUDE_SUBDOMAINS or on the Web server.

Use ‘secure’ cookies by setting SESSION_COOKIE_SECURE and CSRF_COOKIE_SECURE to True. This will ensure that cookies are only ever sent over HTTPS.

Continue
..

Common Threat/Protection

Host header validation

Use ALLOWED_HOSTS to only accept requests from trusted hosts.

Continue
..

Thank you!