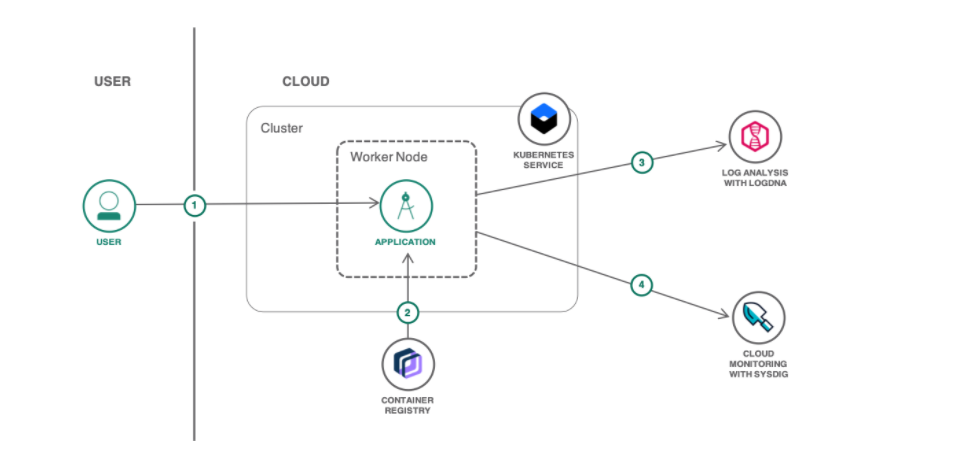
* ***Django***

[**What is Django?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_is_django)

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

**Django helps you write software that is:**

* **Complete**
* **Versatile**
* **Secure**
* **Scalable**
* **Maintainable**
* **Portable**

****

[**Where did it come from?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#where_did_it_come_from)

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework which was open-sourced as the Django project in July 2005.

## [How popular is Django?](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#how_popular_is_django)

There isn't any readily-available and definitive measurement of popularity of server-side frameworks (although you can estimate popularity using mechanisms like counting the number of GitHub projects and Stack Over flow questions for each platform). A better question is whether Django is "popular enough" to avoid the problems of unpopular platforms. Is it continuing to evolve? Can you get help if you need it? Is there an opportunity for you to get paid work if you learn Django?

## Getting to know the Django web framework:

Do you need more information to discover what is Django and how it's used? Do you want some hands-on experience working with a web app created using the Django web framework? IBM offers a [step-by-step tutorial](https://cloud.ibm.com/docs/Log-Analysis-with-LogDNA?topic=Log-Analysis-with-LogDNA-getting-started) for using [IBM Cloud](https://www.ibm.com/cloud) services to track activity and monitor the health of a Python-based [Kubernetes](https://www.ibm.com/cloud/learn/kubernetes) application written using the Django web framework and running on the IBM Cloud.

With all the Python web frameworks to choose from, why pick Django web framework? It might not be the easiest to use and it’s certainly not the newest. Nevertheless, the Django web framework might be the right fit when you’re building a web app that’s involves cross-site scripting and is expected to handle a large number of users or a complex set of features, such as API connectivity or user authentication. Based on the number of projects on GitHub, it's also very popular.

Anyone proficient in the Python programming language and its syntax should be able to start a project using Django web framework to build a web app. (Note: While most of Django core is Python, the admin and gis contrib apps contain JavaScript code.) However, intermediate to advanced Django developers can better capitalize on Django models, which are more sophisticated features.

## What is the Django Framework?

Django is an open-source framework for backend web applications based on Python one of the top web development languages. Its main goals are simplicity, flexibility, reliability, and scalability.

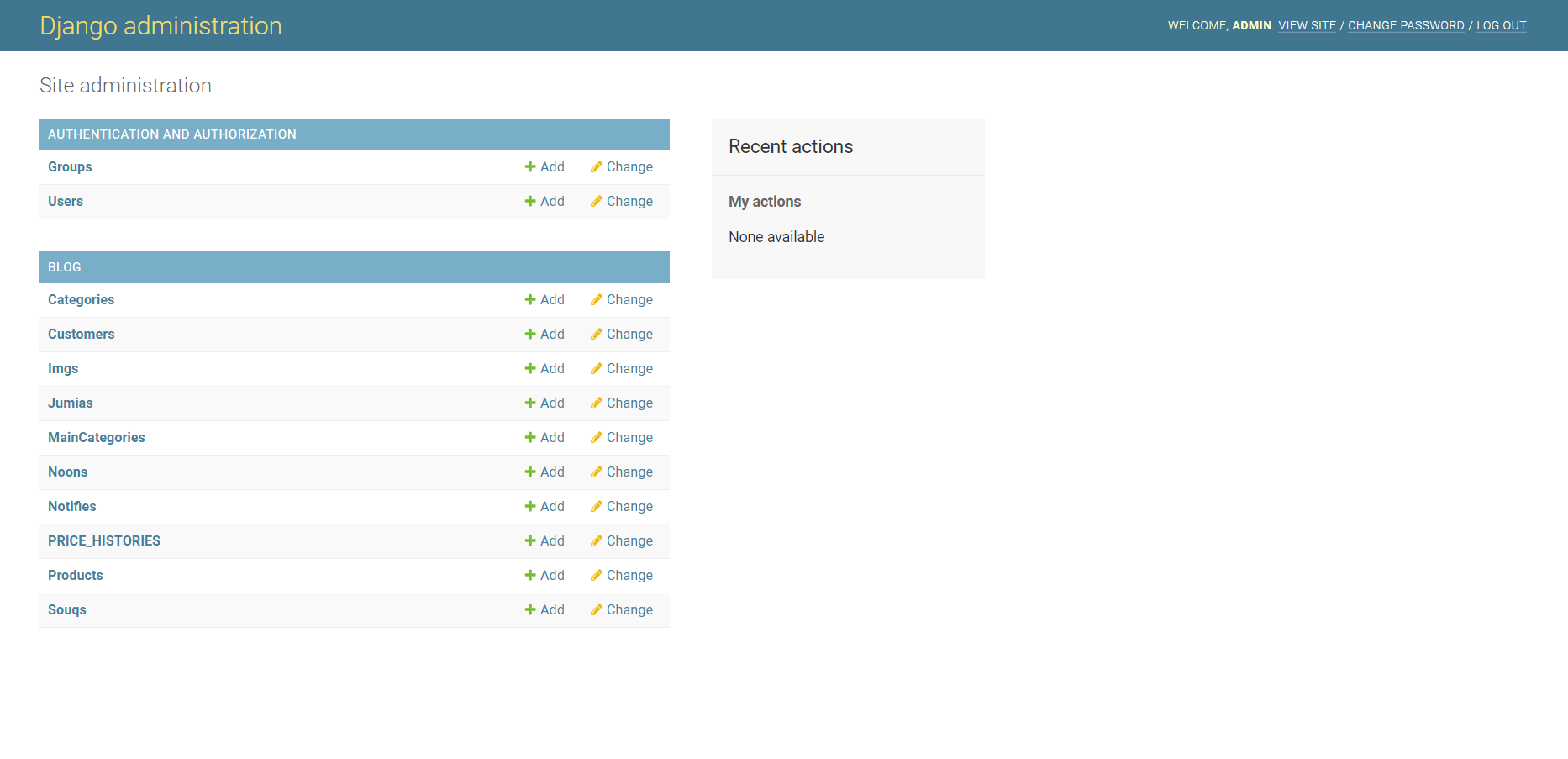
Django has its own naming system for all functions and components e.g., HTTP responses are called “**views**”. It also has an admin panel, which is deemed easier to work with than in Lavarel or Yii and other technical features, including:

* Simple syntax
* Its own web server
* MVC (Model-View-Controller) core architecture
* Batteries included (comes with all the essentials needed to solve solving common cases)
* An ORM (Object Relational Mapper)
* HTTP libraries
* Middleware support
* A Python unit test framework

## Why do you need a framework?

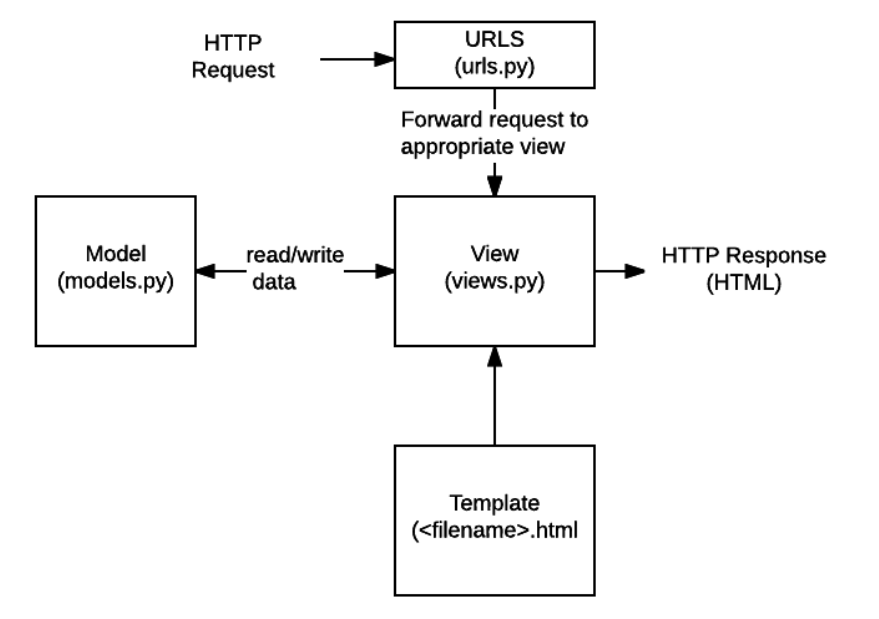
To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that you want it to serve you a web page.

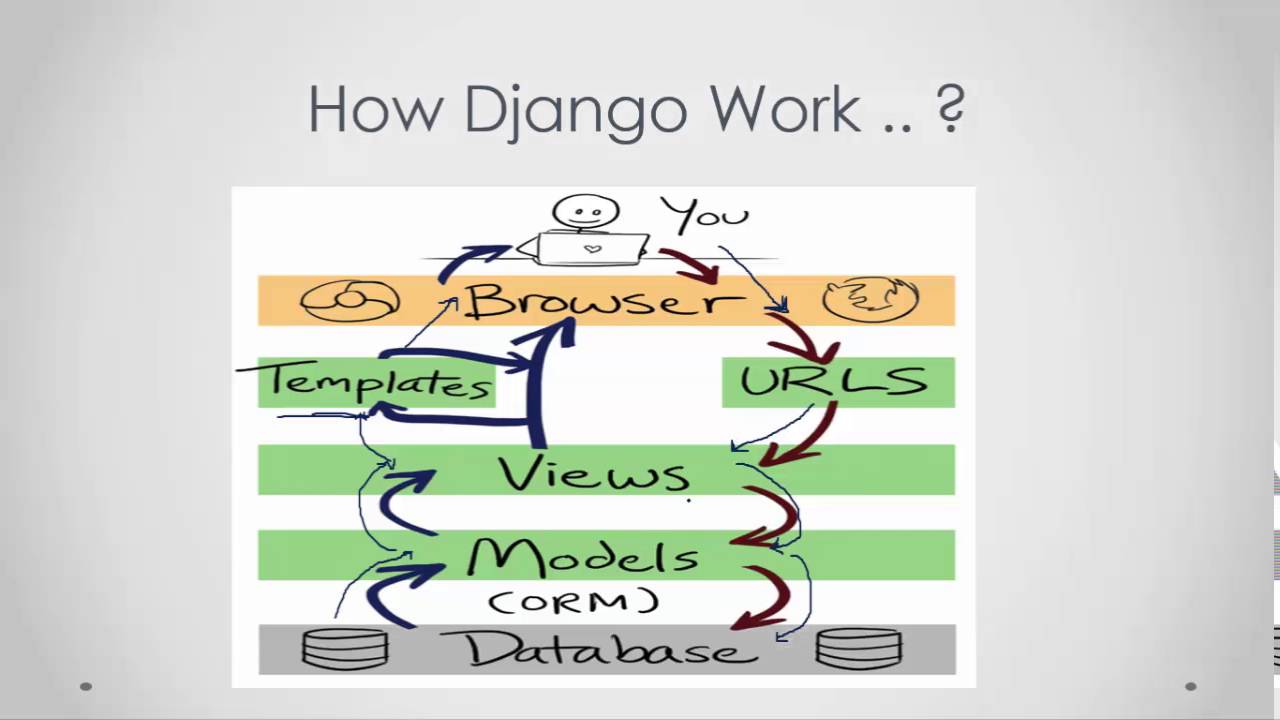
Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server reads the letter and then sends a response with a webpage. But when you want to send something, you need to have some content. And Django is something that helps you create the content.



## [What does Django code look like?](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_does_django_code_look_like)

In a traditional data-driven website, a web application waits for HTTP requests from the web browser or other client. When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.





* **URLs:**While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
* **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
* **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
* **Templates:** 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*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

## What happens when someone requests a website from your server?

When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested. It takes a web page address first and tries to figure out what to do. This part is done by Django's **url resolver**(note that a website address is called a URL – Uniform Resource Locator – so the name url resolver makes sense). It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function which is called **view**.

In the view function all the interesting things are done we can look at a database to look for some information.

So instead of diving too much into details, we will start creating something with Django and we will learn all the important parts along the way

## [What else can you do?](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_else_can_you_do)

* **Forms**: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
* **User authentication and permissions**: Django includes a robust user authentication and permission system that has been built with security in mind.
* **Caching**: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
* **Administration site**: The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
* **Serialising data**: Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service or when creating a website in which the client-side code handles all the rendering of data.
* Sources
* <https://tutorial.djangogirls.org/en/django/>
* <https://djangostars.com/blog/why-we-use-django-framework/>
* <https://www.ibm.com/cloud/learn/django-explained>
* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>