**UNIT – 5**

**Django** is a Python-based **web framework** that allows you to create efficient **web applications**quickly. It is also called **batteries included framework** because Django provides built-in features for everything including Django Admin Interface, default database – SQLlite3, etc.

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

### ***What are the Features of Django?***

## Rapid Development

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

## Secure

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

## Scalable

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

## Fully loaded

Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

## Versatile

Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

## Open Source

Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

## Vast and Supported Community

Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

***What are the uses of Django?***

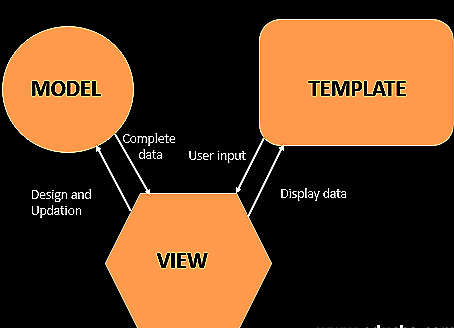
1. **Web Development**: Django is primarily used for building web applications, ranging from simple websites to complex web platforms.
2. **Efficient Development**: It provides a streamlined development process with built-in tools for tasks like URL routing, form handling, and database interaction, reducing development time.
3. **Scalability**: Django is capable of handling high levels of traffic and data due to its scalability features, making it suitable for both small-scale projects and large-scale applications.
4. **Security**: Django includes built-in security features such as protection against common web security threats like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
5. **Versatility**: It can be used for various types of web applications including e-commerce platforms, content management systems (CMS), social networking sites, and more.
6. **Community and Support**: Django has a large and active community of developers, providing extensive documentation, tutorials, and support resources.
7. **Customization**: Developers can extend Django's functionality through its modular structure, allowing for customization and integration with third-party libraries and frameworks.
8. **Maintainability**: Django promotes clean and maintainable code through its adherence to best practices and design patterns like Model-View-Template (MVT), making it easier to maintain and update projects over time.
9. **Cross-Platform Compatibility**: Django is platform-independent, meaning it can run on various operating systems, including Linux, macOS, and Windows.
10. **Open Source**: Django is an open-source framework, meaning it's free to use and can be modified and redistributed under the terms of the BSD license.

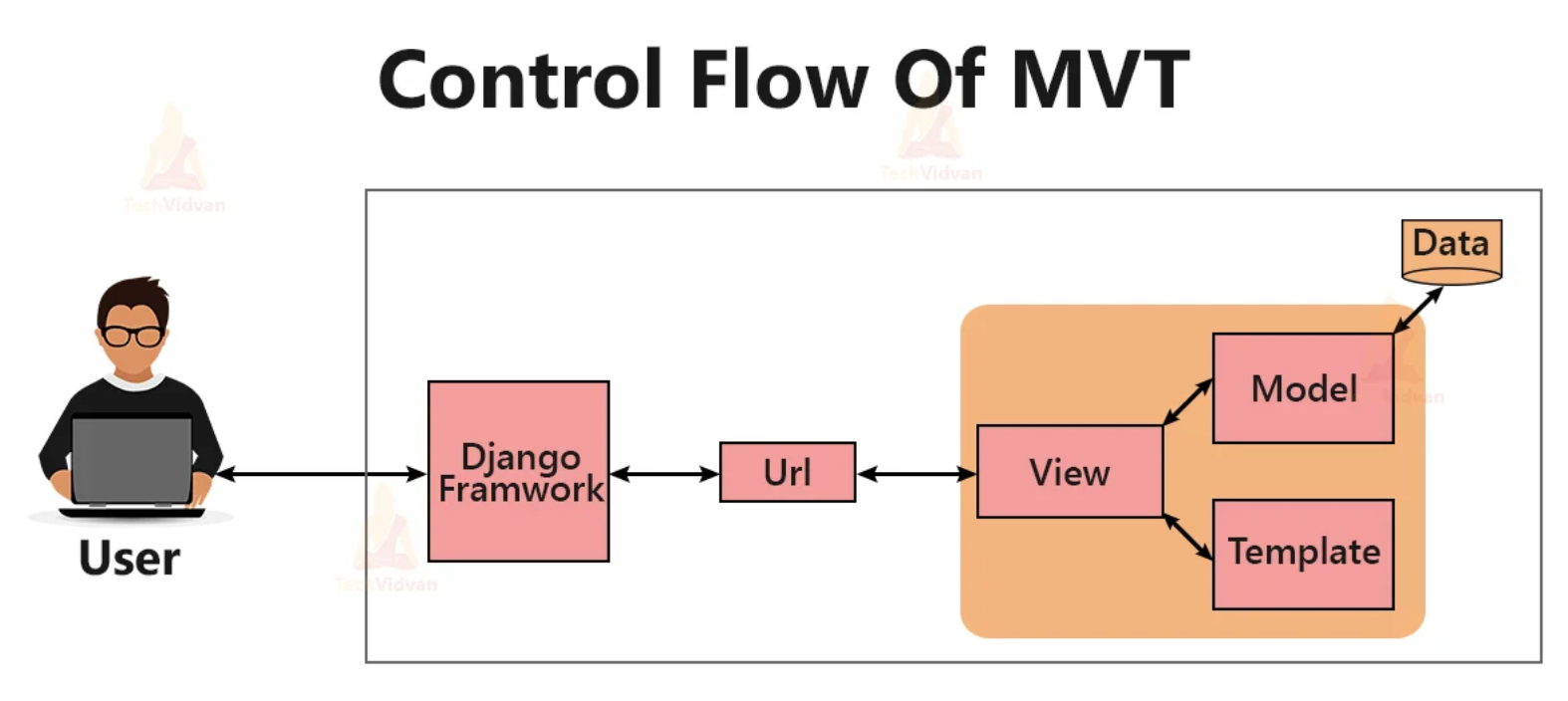
***Django architecture***

Django is based on **MVT (Model-View-Template)** architecture. MVT is a software design pattern for developing a web application. Django is an open-source framework that is used for backend development of web application.

**MVT Structure has the following three parts –**

* **Model: This handles the data of your application, like storing information in a database.**
* **View: Views are where you put the code to respond to web requests. They're like the logic behind what users see on a webpage.**
* **Template: Templates are used to generate the HTML that is sent to the user's browser. They define how the information from the views should be displayed on the webpage.**





## How GUI is created in python? What is Tkinter?

**Tkinter**is a Python Package for creating GUI applications. Python has a lot of GUI frameworks, but Tkinter is the only framework that’s built into the Python standard library.

Tkinter has several strengths; it’s cross-platform, so the same code works on Windows, macOS, and Linux.

Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern shine is unnecessary, and the top priority is to build something functional and cross-platform quickly.

### **Use Cases of Tkinter**

**1. Creating windows and dialog boxes**: Tkinter can be used to create windows and dialog boxes that allow users to interact with your program. These can be used to display information, gather input, or present options to the user.

To create a window or dialog box, you can use the***Tk()*** function to create a root window, and then use functions like ***Label***, ***Button***, and ***Entry*** to add widgets to the window.

**2. Building a GUI for a desktop application**: Tkinter can be used to create the interface for a desktop application, including buttons, menus, and other interactive elements.

To build a GUI for a desktop application, you can use functions like ***Menu***, ***Checkbutton***, and ***RadioButton*** to create menus and interactive elements and use layout managers like ***pack*** and ***grid*** to arrange the widgets on the window.

**3.** **Adding a GUI to a command-line program**: Tkinter can be used to add a GUI to a command-line program, making it easier for users to interact with the program and input arguments.

To add a GUI to a command-line program, you can use functions like ***Entry*** and ***Button*** to create input fields and buttons, and use event handlers like ***command*** and ***bind*** to handle user input.

**4. Creating custom widgets**: Tkinter includes a variety of built-in widgets, such as buttons, labels, and text boxes, but it also allows you to create your own custom widgets.

To create a custom widget, you can define a class that inherits from the ***Widget*** class and overrides its methods to define the behavior and appearance of the widget.

**5. Prototyping a GUI**: Tkinter can be used to quickly prototype a GUI, allowing you to test and iterate on different design ideas before committing to a final implementation.

To prototype a GUI with Tkinter, you can use the ***Tk()*** function to create a root window, and then use functions like ***Label***, ***Button***, and ***Entry*** to add widgets to the window and test different layouts and design ideas.

## Getting Started with Tkinter

**1.** Import tkinter package and all of its modules.  
**2.** Create a root window. Give the root window a title(using **title()**) and dimension(using **geometry()**).  All other widgets will be inside the root window.   
**3.** Use **mainloop()** to call the endless loop of the window. If you forget to call this nothing will appear to the user. The window will wait for any user interaction till we close it.

## Tkinter Widgets

Tkinter is the GUI library of Python, it provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called **Widgets.**The list of commonly used **Widgets**are mentioned below –

| **Widget** | **Description** |
| --- | --- |
| **Label** | The Label widget is used to provide a single-line caption for other widgets. It can also contain images. |
| **Button** | The Button widget is used to display buttons in your application. |
| **Entry** | The Entry widget is used to display a single-line text field for accepting values from a user. |
| **Menu** | The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton. |
| **Canvas** | The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application. |
| **Checkbutton** | The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time. |
| **Frame** | The Frame widget is used as a container widget to organize other widgets. |
| **Listbox** | The Listbox widget is used to provide a list of options to a user. |
| **Menubutton** | The Menubutton widget is used to display menus in your application. |
| **Message** | The Message widget is used to display multiline text fields for accepting values from a user. |
| **Radiobutton** | The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time. |
| **Scale** | The Scale widget is used to provide a slider widget. |
| **Scrollbar** | The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes. |
| **Text** | The Text widget is used to display text in multiple lines. |
| **Toplevel** | The Toplevel widget is used to provide a separate window container. |
| **LabelFrame** | A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. |
| **tkMessageBox** | This module is used to display message boxes in your applications. |
| **Spinbox** | The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values. |
| **PanedWindow** | A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically. |

**Geometry Management**

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place. Their description is mentioned below –

| **Widget** | **Description** |
| --- | --- |
| **pack()** | This geometry manager organizes widgets in blocks before placing them in the parent widget. |
| **grid()** | This geometry manager organizes widgets in a table-like structure in the parent widget. |
| **place()** | This geometry manager organizes widgets by placing them in a specific position in the parent widget. |

**Here is a list of some common Tkinter events which are generally used for making the application interactive.**

* **<Button>** − Use the Button event in a handler for binding the Mouse wheels and Buttons.
* **<ButtonRelease>** − Instead of clicking a Button, you can also trigger an event by releasing the mouse buttons.
* **<Configure>** − Use this event to change the widgets properties.
* **Destroy** − Use this event to kill or terminate a particular widget.
* **<Enter>** − It actually works like <return> event that can be used to get the focus on a widget with mouse Pointer
* **<Expose>** − The event occurs whenever a widget or some part of the application becomes visible that covered by another window in the application.
* **<Focus In>** − This event is generally used to get the focus on a particular widget.
* **<Focus Out>** − To move the focus from the current widget.
* **<Key Press>** − Start the process or call the handler by pressing the key.
* **<KeyRelease>** − Start the process or call an event by releasing a key.
* **<Leave>** − Use this event to track the mouse pointer when user switches from one widget to another widget.
* **<Map>** − Use Map event to show or display any widget in the application.
* **<Motion>** − Track the event whenever the mouse pointer moves entirely within the application.
* **<Unmap>** − A widget can be unmapped from the application. It is similar to hiding the widget using **grid\_remove()**.
* **<Visibility>** − An event can happen if some part of the application gets visible in the screen.