DJANGO – SYLLABUS

1. **Introduction to Django and web development:**

   Understand the basics of web development and the role of Django in building web applications.

2. **Setting up the development environment:**

   Learn how to set up a local development environment and install Django.

3. **Creating a Django project:**

   Create a new Django project and understand the project structure and settings.

4. **Building a basic web application:**

   Learn how to create views, templates, and URLs in Django and build a basic web application.

5. **Using the Django ORM:**

   Understand the basics of the Django Object-Relational Mapping (ORM) and learn how to interact with the database using models.

6. **Creating forms:**

   Learn how to create and validate forms using Django's built-in forms and the third-party library Django Crispy Forms.

7. **Authenticating and authorization:**

   Learn how to handle authentication and authorization in Django using the built-in user model and the Django REST framework.

8. **Advanced topics:**

   Learn about advanced features of Django such as caching, signals, and middleware.

9. **Deployment:**

   Learn how to deploy a Django application to a production environment, such as Heroku or AWS.

10. **Project work:**

    Work on a real-world project to practice the concepts learned during the course and develop a functional web application.

1) Introduction to Django and web development

"Introduction to Django and web development" is an important module that provides a foundation for understanding the basics of web development and the role of Django in building web applications.

Here are the key concepts that will be covered in this module:

What is web development:

Web development is the process of creating and maintaining websites and web applications. It involves a combination of client-side and server-side technologies to create a dynamic and interactive user experience.

Client-side vs server-side:

In web development, client-side refers to the technologies used to create the user interface and the client-side logic, such as HTML, CSS, and JavaScript. Server-side refers to the technologies used to handle the backend logic, such as database management, authentication, and authorization.

What is Django:

Django is an open-source web framework for building web applications using the Python programming language. It follows the Model-View-Controller (MVC) architectural pattern and provides a set of built-in features and tools for building web applications quickly and efficiently.

Django philosophy:

Django follows the "Don't Repeat Yourself" (DRY) principle and emphasizes the importance of reusing code and keeping the codebase as simple and clean as possible. It also follows the "Convention over Configuration" principle, which means that it has a set of conventions and defaults that developers can follow to make development faster and more consistent.

Benefits of using Django:

Django has a lot of built-in features and tools that make web development faster and more efficient. It also has a large and active community, which means there are a lot of resources and tutorials available to help developers learn and troubleshoot. Additionally, it is highly scalable and flexible, meaning that it can handle large amounts of traffic and can be used for a variety of different types of web applications.

By the end of this module, you should have a good understanding of the basics of web development and the role of Django in building web applications. You should also have an understanding of the key concepts and principles of Django, and how it compares to other web frameworks.

2) Setting up the development environment

"Setting up the development environment" is an important step in learning Django and building web applications. This module will guide you through the process of setting up your local development environment for building Django web applications. It's important to have a proper environment set up to avoid any compatibility issues or errors that might arise.

Here are the steps you need to follow to set up your development environment for Django:

Install Python: Django is built on Python, so the first step is to install the latest version of Python on your computer. You can download the latest version of Python from the official website (https://www.python.org/downloads/)

Install pip:

pip is a package manager for Python that allows you to install and manage Python packages. It is included with Python version 3.4 and later, but if you are using an earlier version, you will need to install it separately.

Create a virtual environment: A virtual environment is a tool used to isolate Python environments and avoid compatibility issues between different projects. It allows you to have different versions of Python and packages for different projects. You can create a virtual environment using the command `python -m venv myenv` on the command line.

Activate the virtual environment:

To activate the virtual environment, you will need to navigate to the folder where the virtual environment is located and use the command `source myenv/bin/activate` on MacOS or Linux or `myenv\Scripts\activate.bat` on Windows.

Install Django:

Once the virtual environment is activated, you can use pip to install Django using the command `pip install Django`. This will install the latest version of Django in your virtual environment.

Install a code editor or IDE: You will need a code editor or integrated development environment (IDE) to write and debug your code. Some popular choices include Visual Studio Code, PyCharm, and Sublime Text.

Test your installation:

To test your installation, you can create a new Django project by running the command `django-admin startproject myproject`. If the project is created successfully, it means your development environment is set up correctly.

By following these steps, you should have a development environment that is ready to start building Django web applications. Remember to always activate your virtual environment before working on a project, to avoid any compatibility issues.

3) Creating a Django project

"Creating a Django project" is an essential module that will guide you through the process of creating a new Django project, which is the foundation for building web applications with Django.

Here are the steps you need to follow to create a new Django project:

Activate your virtual environment:

Before you create a new project, make sure that you have activated the virtual environment you set up in the previous module. This will ensure that you are using the correct version of Python and Django.

Run the startproject command:

To create a new Django project, you will need to use the command `django-admin startproject projectname`. This command creates a new directory with the same name as the project, and inside it, creates the basic structure of a Django project.

Understand the project structure:

Once the project is created, you should see the following files and folders:

manage.py: This is a command-line utility that lets you interact with the project.

projectname: This is the main folder that contains the settings and the main urls of your project.

projectname/settings.py: This file contains the project-wide settings and configurations.

projectname/urls.py: This file contains the project-wide urls and directs the request to the correct views.

projectname/wsgi.py: This file is used for deploying the project to a production environment.

Run the development server:

Once the project is created, you can run the development server by using the command `python manage.py runserver` on the command line. This will start the server and make your project available at http://127.0.0.1:8000/

Test your project:

Open your browser and navigate to http://127.0.0.1:8000/. You should see a "Welcome to Django" page, which confirms that your project is set up correctly and the server is running.

By following these steps, you will have created a new Django project and understand the basic structure of the project. From here, you can start building views, templates and URLs for your web application. Remember to always activate your virtual environment before working on a project, to avoid any compatibility issues.

4) Building a basic web application

"Building a basic web application" is an important module that will guide you through the process of building a simple web application with Django, which will cover the core concepts of the framework.

Here are the steps you need to follow to build a basic web application with Django:

Create a view:

A view is a Python function that handles a specific request from the user and returns a response. To create a view, you need to create a new file called views.py in your project folder and define a function that takes a request as an argument and returns a response.

Create a URL pattern:

A URL pattern is a set of rules that define how the URLs of your application should look like. To create a URL pattern, you need to open the urls.py file in your project folder and define a new URL pattern that maps to the view you created in step 1.

Create a template:

A template is a file that contains the HTML, CSS, and JavaScript that will be rendered in the user's browser. To create a template, you need to create a new folder called templates in your project folder and create a new file with the HTML code.

Render the template:

To render the template in the user's browser, you need to call the render function from the views.py file and pass it the request and the template as arguments.

Test the application:

Once you have completed the above steps, you can run the development server using the command `python manage.py runserver` and test the application by navigating to the URL defined in the url patterns.

By following these steps, you will have built a basic web application with Django, which will cover the core concepts of the framework such as views, URLs, and templates. You will also have an understanding of how the different components of a Django web application work together.

It's important to note that this is just a simple example of building a web application with Django, and in real-world projects, it could be more complex. However, this example should give you a good understanding of how the different components of a Django web application interact with each other and how to build one.

5) Using the Django ORM

"Using the Django ORM" is an important module that will guide you through the process of interacting with databases using Django's built-in Object-Relational Mapping (ORM) feature.

Here are the key concepts that will be covered in this module:

What is ORM:

ORM is a technique that allows developers to interact with databases using an object-oriented programming model, rather than writing raw SQL queries. Django's ORM allows developers to define their models as Python classes and use them to create, read, update, and delete records in the database.

Define models:

To use the ORM, you first need to define your models, which represent the different tables in the database. Models are defined in the models.py file of your app and each model represents a table in the database. You will define the fields of the model, which correspond to the columns of the table, and the relationships between the models, which correspond to the relationships between the tables.

Create a database:

Once the models are defined, you need to create the database tables by running the command `python manage.py makemigrations` followed by `python manage.py migrate`. This will create the necessary tables in the database based on the models defined.

Perform CRUD operations:

Once the database is set up, you can use the ORM to perform CRUD operations. The ORM provides a set of methods to create, read, update, and delete records in the database using the defined models.

Advanced features:

Django's ORM also provides advanced features such as filtering, ordering, and aggregating records, and managing relationships between models. You can use the ORM to filter records based on certain criteria, order the records in a certain way, and aggregate data from multiple records.

By the end of this module, you will have a good understanding of how to use Django's ORM to interact with databases, define models, and perform CRUD operations. You will also have an understanding of some of the advanced features provided by the ORM.

It's important to note that this is just an overview of the Django ORM and there's much more to learn in order to master it. However, this module should give you a good understanding of the basic concepts and how to use it in a Django web application.

6) Creating forms

"Creating forms" is an important module that will guide you through the process of creating forms in Django, which allows users to input data into the web application.

Here are the key concepts that will be covered in this module:

What are forms:

Forms are a way for users to input data into the web application. In Django, forms are created as classes that define the fields and validation rules for the form.

Using Django forms:

Django provides built-in form classes such as **CharField, EmailField,** and **PasswordField** that can be used to create forms. You can also create custom forms by subclassing the Form class and defining your own fields and validation rules.

Rendering forms:

Once the form is created, it needs to be rendered in a template so that the user can interact with it. You can use the **{{ form.as_p }}** or **{{ form.as_table }}** template tags to render the form as a series of paragraphs or table rows respectively. You can also use **{{form.as_p}}** or **{{form.as_table}}** to render forms.

Form validation:

When a user submits a form, the data needs to be validated to ensure that it is correct and complete. Django's built-in form classes have built-in validation for common fields such as email and password. You can also add custom validation by defining the **clean()** method in your form class.

Handling form submissions:

Once the form is submitted, the data needs to be handled by the view. The view can access the data by calling the **cleaned_data** attribute of the form object. You can access the form data by calling the **cleaned_data** attribute of the form object.

Using third-party libraries:

Django also has a number of third-party libraries available for creating forms such as Django Crispy Forms and Django Bootstrap Forms, which provide additional features such as styling and layout options. These libraries can provide you with additional options for styling and layout.

By the end of this module, you will have a good understanding of how to create forms in Django, how to handle form submissions and validate the data. You will also have an understanding of how to use third-party libraries to enhance the functionality of the forms

7) Authenticating and authorization

"Authenticating and authorization" is an important module that will guide you through the process of securing your web application by implementing authentication and authorization in Django.

Here are the key concepts that will be covered in this module:

What is authentication:

Authentication is the process of verifying a user's identity by checking their credentials, such as a username and password. Django provides built-in authentication views and forms that can be used to authenticate users.

What is authorization:

Authorization is the process of granting or denying access to certain resources or actions based on a user's role or permissions. Django provides built-in tools for managing user roles and permissions.

Using Django's built-in authentication views:

Django provides built-in views and forms for handling user registration, login, logout, and password management. These views can be easily integrated into your web application and customized to fit your needs.

Creating custom authentication views:

In some cases, you may need to create custom authentication views to handle specific requirements. Django's authentication views can be easily subclassed and customized to fit your needs.

Implementing authentication with third-party services:

Django also supports authentication with third-party services such as Google, Facebook, and Twitter. This can be achieved by using third-party libraries such as **python-social-auth** and **django-allauth.**

Managing user roles and permissions:

Django provides built-in tools for managing user roles and permissions, such as the built-in User model, the Permission and Group models, and the @user_passes_test and @permission_required decorators. You can use these tools to manage user roles and permissions, and control access to certain resources or actions based on the user's role or permissions.

By the end of this module, you will have a good understanding of how to implement authentication and authorization in Django using built-in views and forms, and how to customize them to fit your needs. You will also have an understanding of how to use third-party libraries to implement authentication with third-party services, and how to manage user roles and permissions.

8) Advanced topics

"Advanced topics" is a module that will cover more advanced concepts and features of Django that are useful for building more complex and feature-rich web applications.

Here are the key concepts that will be covered in this module:

Handling file uploads:

Django provides built-in support for handling file uploads, allowing you to handle images, videos, and other types of files in your web application.

Sending email:

Django provides a built-in email library that can be used to send emails from your application, such as password reset emails or confirmation emails.

Using the Django REST framework:

The Django REST framework is a powerful tool for building RESTful web services in Django. It provides views, serializers, and other tools for building RESTful APIs.

Caching:

Caching can be used to speed up your web application by storing the results of expensive operations in memory, reducing the number of database queries and other operations that need to be performed.

Deployment:

To make your web application available to the public, you will need to deploy it to a web server. Django provides support for deployment to a variety of platforms, such as Heroku, AWS, and GCP.

Debugging and troubleshooting:

Debugging and troubleshooting are essential skills for any web developer. Django provides a number of tools and techniques for debugging and troubleshooting issues in your web application, such as the built-in debug toolbar and logging.

By the end of this module, you will have a good understanding of how to handle file uploads, sending email, creating RESTful APIs, caching and debugging your Django web application. You will also have an understanding of how to deploy your application to a web server and troubleshoot issues.

9) Deployment

“Deployment" is an important module that will guide you through the process of deploying your Django web application to a production environment, making it accessible to the public.

Here are the key concepts that will be covered in this module:

Choosing a hosting platform:

There are many hosting platforms available for deploying Django web applications, such as Heroku, AWS, and GCP. Each platform has its own set of features and costs, and you will need to choose the one that best fits your needs and budget.

Configuring the production environment:

To prepare your application for deployment, you will need to configure the production environment, such as setting up a database, configuring security settings, and optimizing performance.

Deploying the application:

Once the production environment is configured, you can deploy your application to the hosting platform. This typically involves creating a new application on the platform, configuring environment variables, and deploying the code.

Managing dependencies:

Managing dependencies is important in production environment, you will need to ensure that all the required dependencies are installed and updated, such as the Django framework, third-party libraries, and database connector.

Monitoring and scaling:

Once your application is deployed, you will need to monitor it to ensure that it is running smoothly and scaling it as needed to handle increased traffic or load.

Handling errors and troubleshooting:

Even with monitoring and scaling, issues may arise with your deployed application. Django provides various tools and techniques for debugging and troubleshooting issues, such as log files and error tracking tools.

By the end of this module, you will have a good understanding of how to deploy your Django web application to a production environment, configure the production environment, manage dependencies and monitor and scale your application. Additionally, you will also have an understanding of how to handle errors and troubleshoot issues that may arise in the production environment.

10)Project work

"Project work" is an important module that will guide you through the process of building a complete web application using the concepts and techniques learned in previous modules.

Here are the key concepts that will be covered in this module:

Defining the project scope:

The first step in the project work is to define the scope of the project. This includes determining the requirements, goals, and objectives of the project, as well as identifying the target audience and any constraints.

Planning the project:

Once the project scope is defined, you can begin planning the project. This includes creating a project plan, outlining the project deliverables, and identifying the tasks and milestones required to complete the project.

Building the web application:

Using the concepts and techniques learned in previous modules, you will begin building the web application. This includes designing the database, creating the models, views, and templates, and implementing any additional functionality such as forms, authentication, and authorization.

Testing and debugging:

As you build the web application, you will need to test and debug it to ensure that it is functioning correctly. This includes writing test cases, testing the application, and troubleshooting any issues that arise.

Deploying the application:

Once the web application is complete and tested, you will deploy it to a production environment, making it accessible to the public.

Maintaining and updating the application:

After the web application is deployed, you will need to maintain and update it to ensure that it continues to function correctly and meet the needs of the users. This includes fixing any bugs that are discovered, updating dependencies, and adding new features as needed.

By the end of this module, you will have a good understanding of how to build a complete web application using Django, and you will have a solid foundation in web development. Additionally, you will have a better understanding of how to plan, build, test, deploy, and maintain web applications using Django.

11)Django References

Here is a list of useful Django references for your Django post:

1) **Official Django Documentation:**

   https://docs.djangoproject.com/en/3.2/ - a comprehensive guide to all aspects of Django, including tutorials and examples.

2) **Django Girls Tutorial:**

   https://tutorial.djangogirls.org/ - a beginner-friendly tutorial for building a simple web application using Django.

3) **Django for Beginners:**

https://djangoforbeginners.com/ - an online book that covers Django from the ground up, including installation and creating your first app.

4) **Django Project Github:**

https://github.com/django/django - the official repository of Django, where you can find the source code and contribute to the development of the framework.

5) **Django Packages:**

https://djangopackages.org/ - a directory of reusable Django apps and packages, including authentication, forms, and more.

6) **DjangoCon US:**

https://djangocon.us/ - an annual conference for Django developers, where you can learn about the latest developments in the framework and network with other developers.

7) **Django Rest Framework**:

https://www.django-rest-framework.org/ - a powerful and flexible toolkit for building RESTful APIs with Django.

8) **Django Channels:**

https://channels.readthedocs.io/en/latest/ - an extension for Django that enables real-time functionality, such as WebSockets and background tasks.

9) **Django Debug Toolbar:**

https://django-debug-toolbar.readthedocs.io/en/latest/ - a package for debugging Django apps, including tools for profiling, caching, and more.

10) **DjangoCon Europe:**

https://djangocon.eu/ - a European conference for Django developers, with talks, tutorials, and workshops.
