INSTALLATION AND CONFIGURATION OF PYTHON FRAMEWORKS

PRESENTATION

01 INTRODUCTION TO FRAMEWORK INSTALLATION

Installing and configuring frameworks properly is a crucial step in any development project. Frameworks provide a structured foundation that simplifies the development process, enhances productivity, and ensures consistency across the application.

Proper installation prevents:

- Compatibility issues.
- Security risks.
- Performance bottlenecks.

02 STEP-BY-STEP INSTALLATION

The installation process involves:

Step 1: Ensure Python is Installed

First, verify that Python is installed by running the following command in your terminal or command prompt:

```
python --version
```

Step 2: Create a Virtual Environment

Creating a virtual environment isolates your project dependencies, preventing conflicts between projects. Run this command to create a virtual environment:

```
python -m venv env
```

Step 3: Activate the Virtual Environment

Next, activate the virtual environment to ensure all installed packages are confined to this environment:

For **Linux:**

```
source env/bin/activate
```

For **Windows:**

```
.\env\Scripts\activate
```

Once activated, your terminal prompt will change, showing you're working within the virtual environment (typically env appears in parentheses).

Step 4: Install the Framework

With the virtual environment activated, install your desired framework using pip:

To install **Django:**

```
pip install django
```

To install **Flask:**

```
pip install flask
```

You can check the installation by running:

For **Django:**

```
django-admin --version
```

For **Flask:**

```
flask --version
```

03 BASIC CONFIGURATION: DJANGO

Django configuration includes modifying settings.py for:

1. **Database Connections:**

- Configure the database settings to connect to your preferred database (e.g., SQLite, PostgreSQL, MySQL).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

2. **Debugging Mode:**

- Enable or disable debugging mode.

Example for production (disable debugging).

```
DEBUG = False
```

3. **Allowed Hosts for Security:**

- Specify which hosts are allowed to serve your Django application.

```
ALLOWED_HOSTS = ['yourdomain.com', '127.0.0.1']
```

BASIC CONFIGURATION: FLASK

Flask configuration involves:

1. Creating a config.py file to manage settings:

```
class Config:
    DEBUG = False
    SECRET_KEY = 'your-secret-key-here'
    SQLALCHEMY_DATABASE_URI = 'sqlite:///app.db'
```

2. Using environment variables for sensitive information (like API keys and database credentials):

```
import os
SECRET_KEY = os.environ.get('SECRET_KEY')
DATABASE_URI = os.environ.get('DATABASE_URI')
```

04 WHY ENVIRONMENT VARIABLES MATTERS?

Using environment variables to store sensitive information (e.g., security keys, database URLs) is a best practice for secure application operation. This allows you to keep these details outside of the source code and adjust configurations for different environments (e.g., development, production).