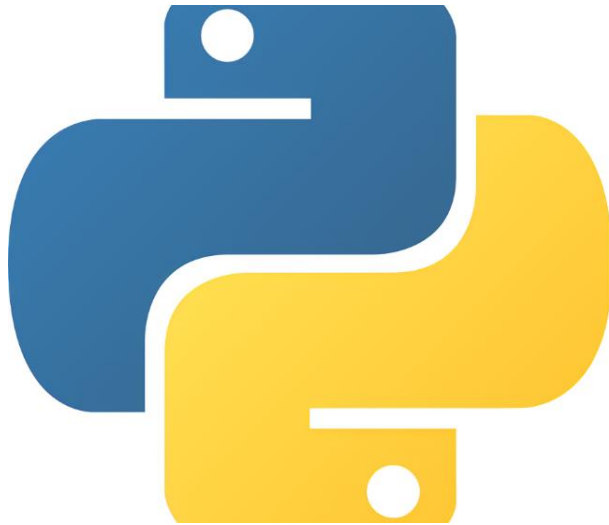


Unlimited Powerful Learning



Learn Core Python

PRESENTED BY : Corporate Trainer

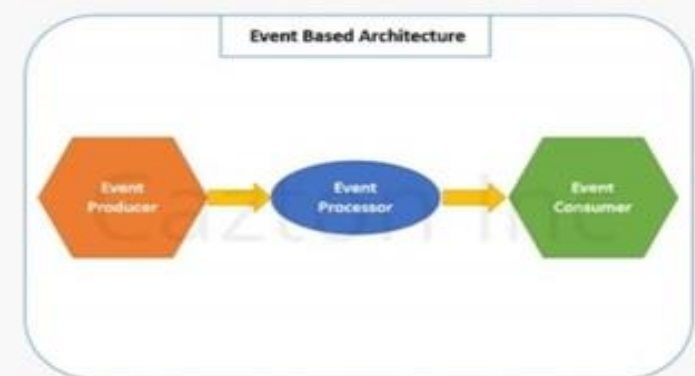
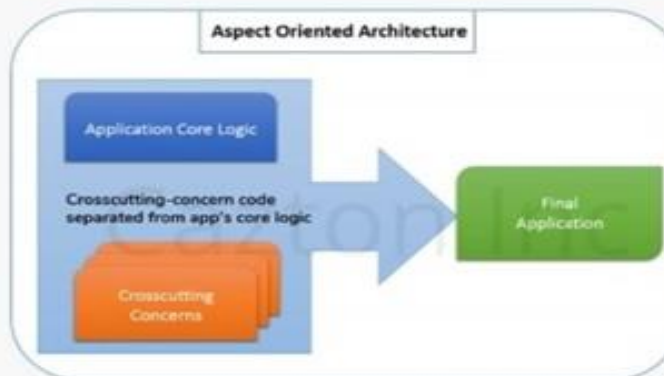
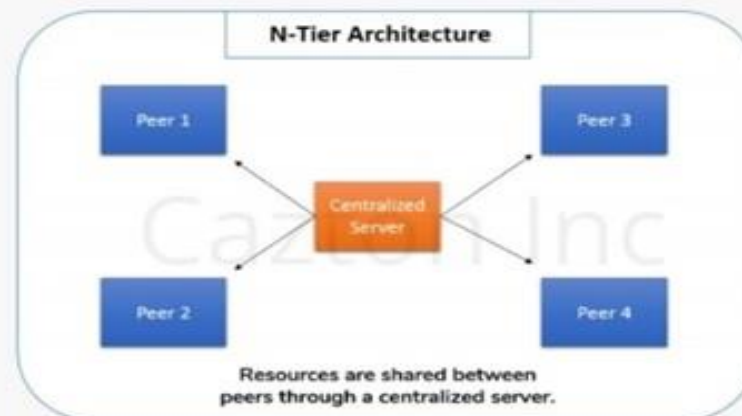
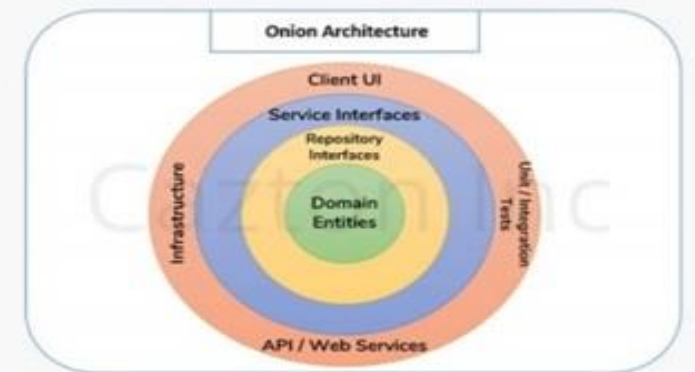
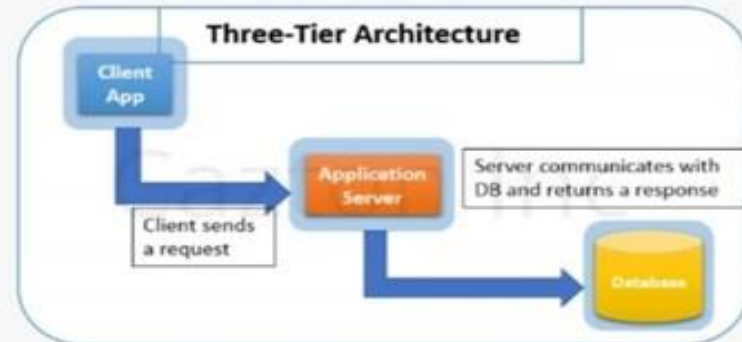
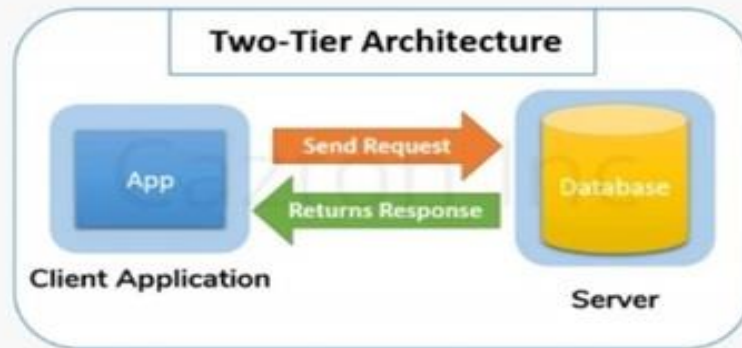
Mallikarjuna G D

gdmallikarjuna@gmail.com

- I. INTRODUCTION
- II. ARCHITECTURE DESIGN
- III. FRAMEWORK
- IV. CHOOSE RIGHT FRAMEWORK
- V. DJANGO FSD ARCHITECTURE
- VI. HISTORY
- VII. MVC DESIGN PATTERN
- VIII. INSTALLATION
- IX. PROGRAM STRUCTURE







- Process of defining structure, components, relationships, and behavior of a software system
- Aims to meet specific requirements and functionality

Overview

- High-level design decisions
- Organization and interaction of system components

Components Identification

- Identify major components/modules
- Based on functional requirements and domain knowledge
- Represent units of functionality or logical groupings

Architectural Styles

- Layered Architecture
- Client-Server Architecture
- Microservices Architecture
- Event-Driven Architecture

Decomposition and Abstraction

Decomposition:

- Break down system into smaller components
- Identify subsystems, define boundaries, establish interfaces

Abstraction:

- Hide unnecessary details
- Focus on essential aspects of components

Communication Protocols

- Define protocols and data formats for interactions
- Specify message formats, APIs, and network communication protocols

Data Management

- Design data architecture for storage, retrieval, and manipulation
- Select appropriate data storage technologies (e.g., relational, NoSQL databases)
- Design data models, schemas, and access pattern

Scalability and Performance

- Design for scalability and performance
- Implement techniques like load balancing, caching, asynchronous processing

Security and Reliability

- Implement security measures: access controls, encryption, authentication
- Ensure system reliability: error handling mechanisms, resilience to failures

Cross-Cutting Concerns

- Logging, monitoring, configuration management
- Addressed through reusable components or architectural aspects

Documentation and Governance

- Document design decisions, rationale, and guidelines
- Establish governance processes for managing changes and adherence to standards

Collaborative and Iterative Process

- Involves architects, developers, stakeholders, and others
- Balance conflicting goals: functionality, performance, scalability, maintainability

- A software framework is a platform for developing software applications.
- It provides a foundation on which software developers can build programs for a specific environment.

Key Features

- Reusable code libraries
- Predefined structures and guidelines
- Standardized ways to build and deploy applications

Importance of Frameworks

- Efficiency: Saves time by providing pre-written code
- Consistency: Ensures uniformity across projects
- Scalability: Facilitates building scalable applications
- Maintainability: Makes code easier to maintain and update
- Security: Often includes built-in security features

Types of Frameworks

Web Frameworks:

- For developing web applications
- Examples: Django, Ruby on Rails, ASP.NET, SPRING BOOT

Mobile Frameworks:

- For developing mobile applications
- Examples: React Native, Flutter, Xamarin

Desktop Frameworks:

- For developing desktop applications
- Examples: Electron, Qt, WPF

Game Development Frameworks:

- For creating video games
- Examples: Unity, Unreal Engine, Godot

Mobile Frameworks

- Frameworks designed for building applications on mobile platforms.
- Examples: React Native, Flutter, Xamarin

- Project Requirements: Specific needs and goals of the project.
- Language Preference: Languages that the development team is comfortable with.
- Community and Support: Availability of resources, documentation, and community support.
- Scalability: Ability to handle growth and scale with the project.
- Performance: Efficiency and speed of the framework.

Benefits:

- Speed of Development: Accelerates the development process.
- Standardization: Promotes best practices and consistency.
- Security: Often includes built-in security measures.
- Community Support: Access to a wealth of resources and community assistance.
- Maintainability: Easier to maintain and update code

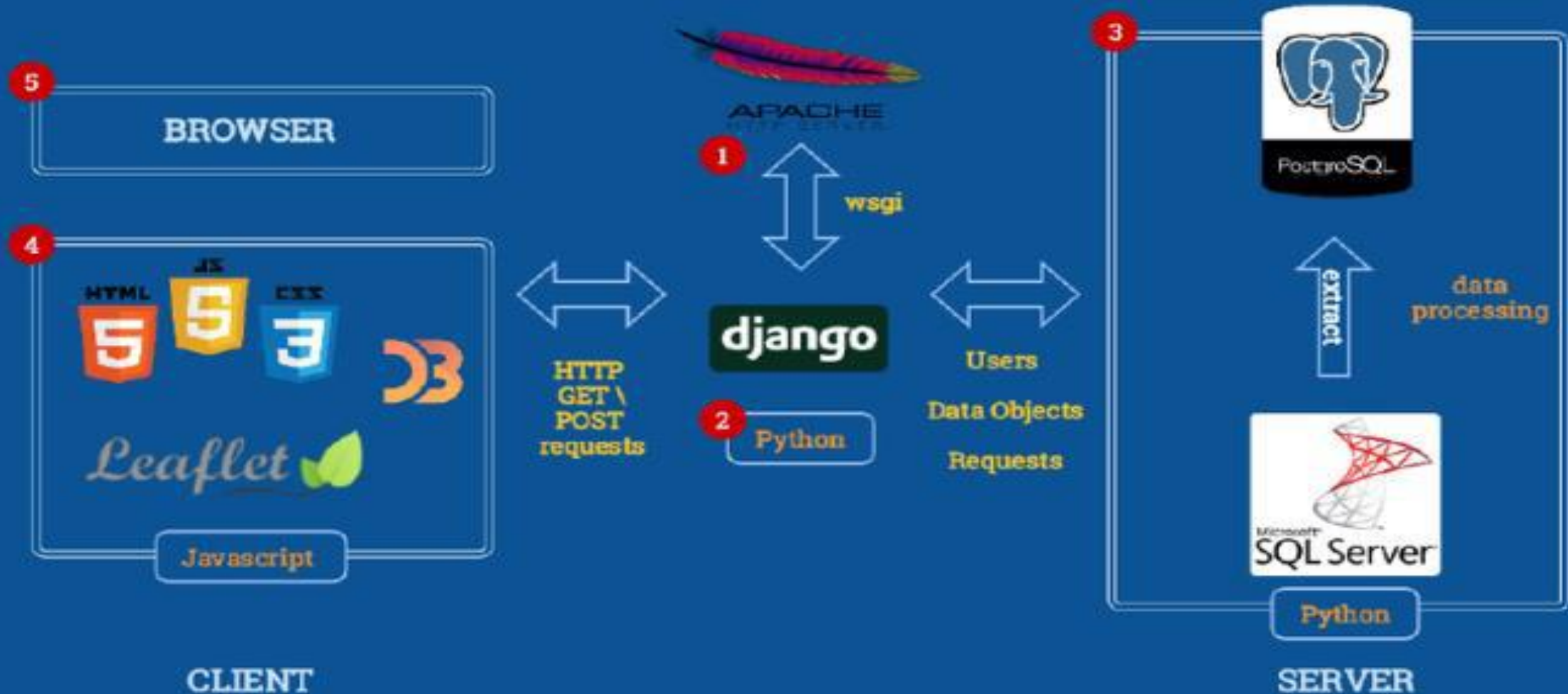
Drawbacks

- Learning Curve: Can be steep for beginners.
- Overhead: May add unnecessary complexity for simple projects.
- Flexibility: Can be restrictive if not suitable for specific project needs.

DJANGO FSD ARCHITECTURE

Philosophy:

- "Batteries-included" framework.
- Emphasis on reusability and less code.



Backend Development:

| Component | Description | Technology | Example |
|-----------------------------|--|-------------------|--|
| Model Layer | Define data models using Django's ORM system. | Django ORM | <pre>class Book(models.Model): title = models.CharField(max_length=100)</pre> |
| View Layer | Implement views to process HTTP requests and return HTTP responses. | Django Views | <pre>def book_list(request): books = Book.objects.all()</pre> |
| Controller (Business Logic) | Handle the business logic of the application by processing requests, validating input, and generating responses. | Django Views | <pre>def book_detail(request, pk): book = get_object_or_404(Book, pk=pk)</pre> |
| URL Routing | Define URL patterns to map incoming URLs to views. | Django URLconf | <pre>path('books/', views.book_list, name='book_list')</pre> |
| Middleware | Process requests and responses at various stages of the request/response cycle. | Django Middleware | <pre>MIDDLEWARE = ['django.middleware.security.SecurityMiddleware', ...]</pre> |

Frontend Development:

| Component | Description | Technology | Example |
|---------------------------------------|---|--|---|
| Templates | Create HTML templates using Django's template engine. | Django Template Engine | {{ book.title }} in book_list.html |
| Static Files Handling | Serve static files such as CSS, JavaScript, and images. | Django Static Files | STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")] |
| Integration with Front-end Frameworks | Integrate with frameworks like React.js, Vue.js, or Angular for building interactive UIs. | Django REST Framework, React.js/Vue.js/Angular | Using Django as a RESTful API backend and React for the front-end UI. |

Database Development:

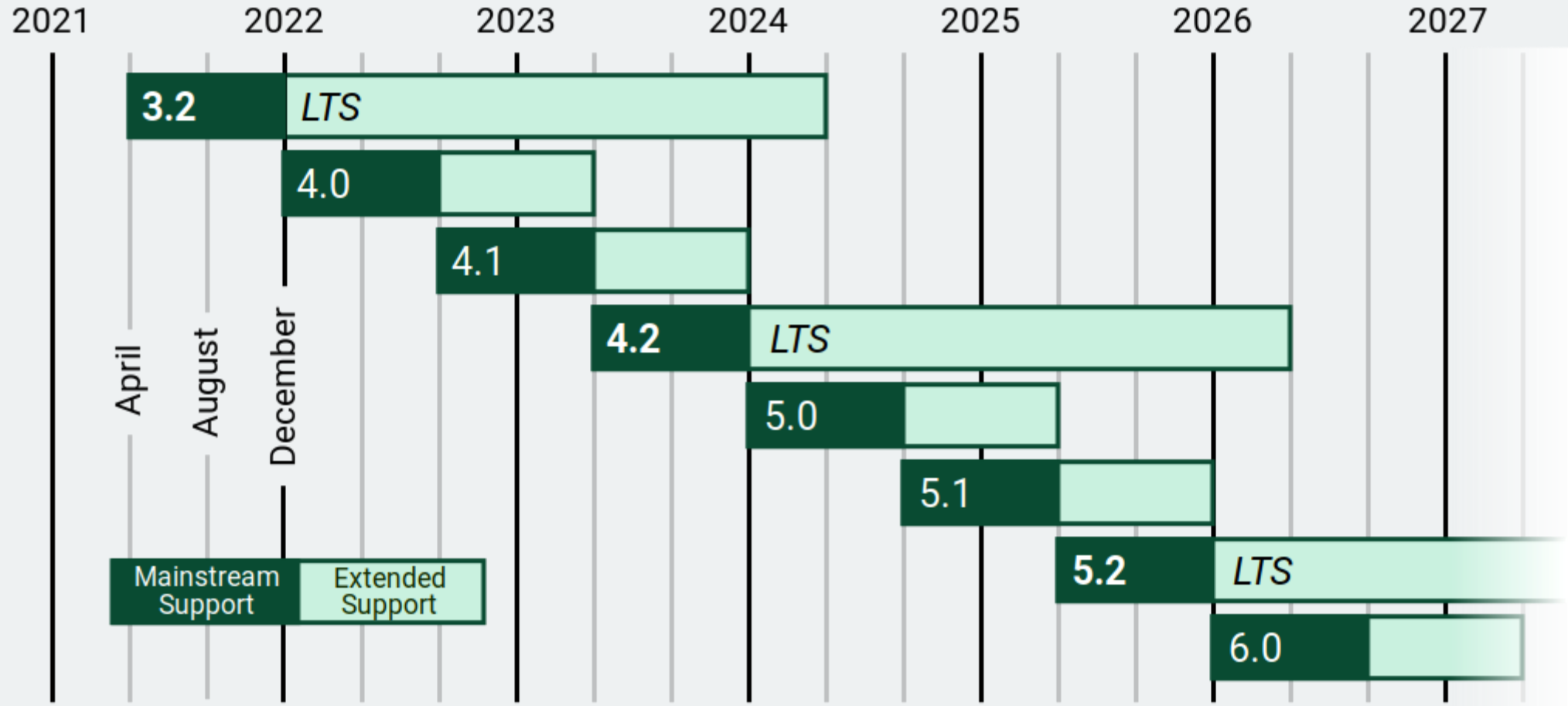
| Component | Description | Technology | Example |
|---------------------------|--|--|---|
| Multiple Database Support | Supports PostgreSQL, MySQL, SQLite, and Oracle. | Django ORM, PostgreSQL/MySQL/SQLite/Oracle | DATABASES = { 'default': { 'ENGINE': 'django.db.backends.postgresql', 'NAME': 'mydatabase', ... } } |
| Database Operations | Perform querying, inserting, updating, and deleting records. | Django ORM | Book.objects.filter(author='Author Name') |

Authentication and Authorization

| component | Description | Technology | Example |
|----------------------|--|--------------------|--|
| User Management | Built-in system for user authentication and authorization. | Django Auth System | <code>from django.contrib.auth.models import User</code> |
| Password Hashing | Secure password storage and hashing. | Django Auth System | <code>user.set_password('new_password')</code> |
| Session Management | Handle user sessions and cookies. | Django Sessions | <code>request.session['key'] = 'value'</code> |
| Permissions Handling | Manage user permissions and groups. | Django Auth System | <code>from django.contrib.auth.models import Permission</code> |

Deployment:

| component | Description | Technology | Example |
|---------------------------------|---|--|---|
| Deployment Platforms | Deploy to servers like Heroku, AWS, or DigitalOcean. | Heroku, AWS EC2, DigitalOcean | <code>git push heroku master</code> |
| Scaling Techniques | Scale horizontally by adding more instances behind a load balancer. | Load Balancer (Nginx/Apache), Docker, Kubernetes | Using AWS ELB (Elastic Load Balancing) to distribute traffic. |
| Environment Configuration | Manage environment-specific settings and secrets. | Environment Variables, django-environ | <code>DATABASE_URL = os.environ.get('DATABASE_URL')</code> |
| Static and Media Files Handling | Use services like AWS S3 for storing static and media files. | AWS S3, django-storages | <code>DEFAULT_FILE_STORAGE = storages.backends.s3boto3.S3Boto3Storage'</code> |



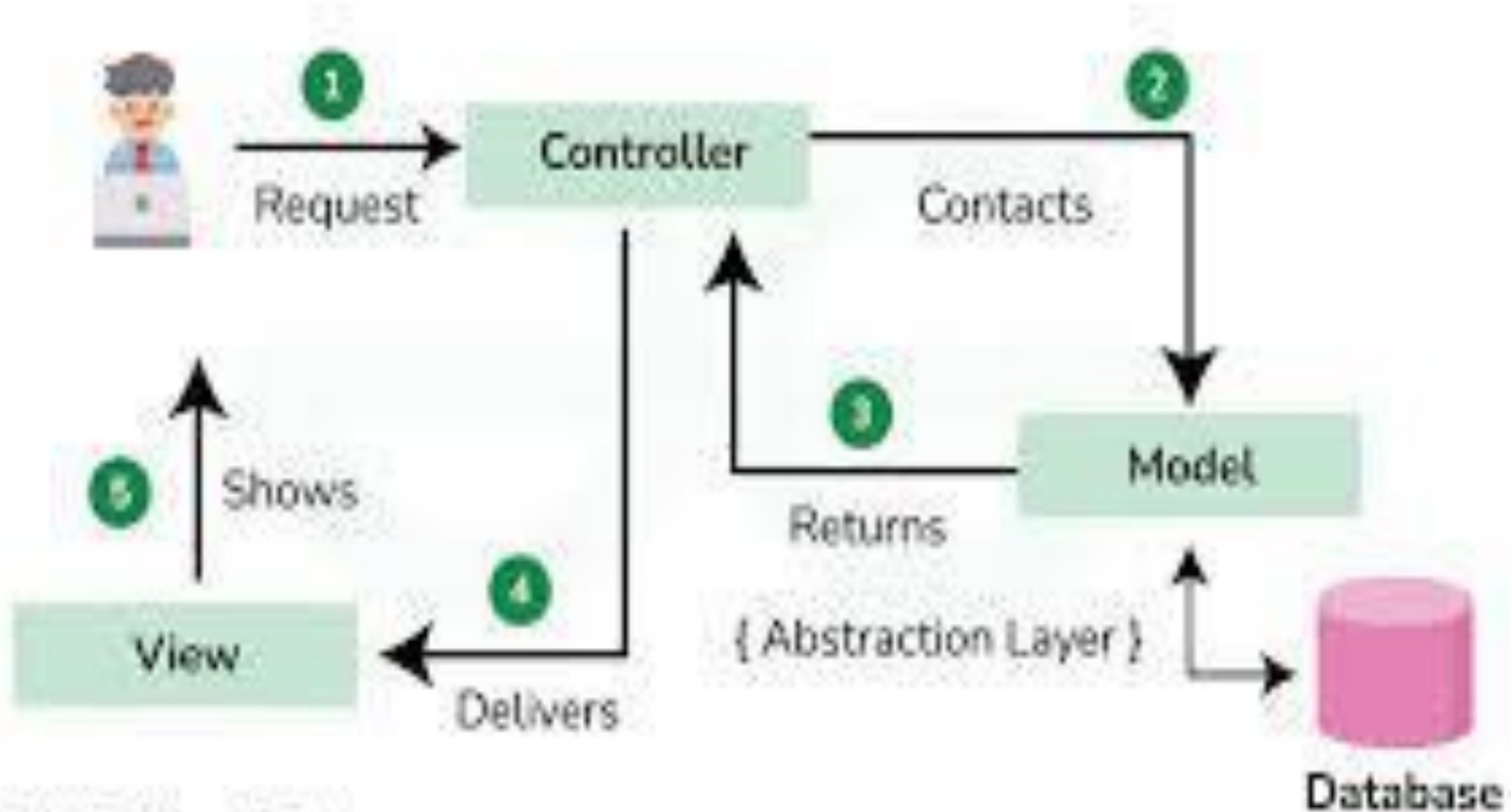
| Version | Release Year | Key Features |
|---------|--------------|--|
| 0.9 | 2005 | Initial public release |
| 0.91 | 2006 | Improved admin interface, new template filters |
| 0.95 | 2006 | Form library overhaul, enhanced documentation |
| 0.96 | 2006 | Session framework, generic views |
| 1 | 2008 | Stability guarantees, long-term support (LTS) |
| 1.1 | 2009 | Aggregates support, GeoDjango, improved testing |
| 1.2 | 2010 | Multiple database support, CSRF protection |
| 1.3 | 2011 | Class-based views, improved static files handling |
| 1.4 | 2012 | Time zone support, custom user model |
| 1.5 | 2013 | Custom template tags, configurable user model |
| 1.6 | 2013 | Improved testing tools, connection pooling |
| 1.7 | 2014 | Migrations framework, application loading refactor |
| 1.8 | 2015 | Built-in support for multiple template engines |
| 1.9 | 2015 | Automatic password validation, admin list filters |
| 1.1 | 2016 | Template-based widget rendering, conditional expressions |
| 1.11 | 2017 | Longer-term support, subquery expressions |
| 2 | 2017 | Python 3 only, new URL syntax, window expressions |
| 2.1 | 2018 | PostgreSQL 9.4+ JSONField support, easier testing |
| 2.2 | 2019 | Advanced options for database indexes, performance improvements |
| 3 | 2019 | ASGI support, MariaDB support, timezone-aware datetimes |
| 3.1 | 2020 | Support for customizing form rendering, asynchronous views |
| 3.2 | 2021 | Features new database backends, improved admin customization |
| 4.2 | 2023 | Psycopg 3 support, ENGINE as django.db.backends.postgresql supports both libraries |
| 5 | 2024 | Django 5.0 supports Python 3.10, 3.11, and 3.12 |

KEY FEATURE

Unlimited Powerful Learning



| Feature | Description |
|--|---|
| Batteries-Included | <ul style="list-style-type: none">Comprehensive set of built-in features and tools for web development. |
| Model-View-Controller (MVC) Architecture | <ul style="list-style-type: none">Follows MVC architectural pattern, organizing code for better maintainability. |
| ORM (Object-Relational Mapping) | <ul style="list-style-type: none">Simplifies database interactions by allowing developers to work with Python objects. |
| Admin Interface | <ul style="list-style-type: none">Built-in interface for CRUD operations on database records, highly customizable. |
| URL Routing | <ul style="list-style-type: none">Maps URLs to view functions/classes, providing clean and flexible URL organization. |
| Template Engine | <ul style="list-style-type: none">Generates dynamic HTML content using templates, supporting inheritance and tags. |
| Security Features | <ul style="list-style-type: none">Provides built-in protection against common web vulnerabilities like CSRF and XSS. |
| ORM Migrations | <ul style="list-style-type: none">Automates database schema changes, simplifying the process of evolving the database schema over time. |
| Internationalization and Localization | <ul style="list-style-type: none">Supports multilingual applications, with tools for translating text and handling language-specific content. |
| Scalability and Performance | <ul style="list-style-type: none">Designed to handle high traffic loads efficiently, offering features like caching and session management. |



MVC Design Pattern

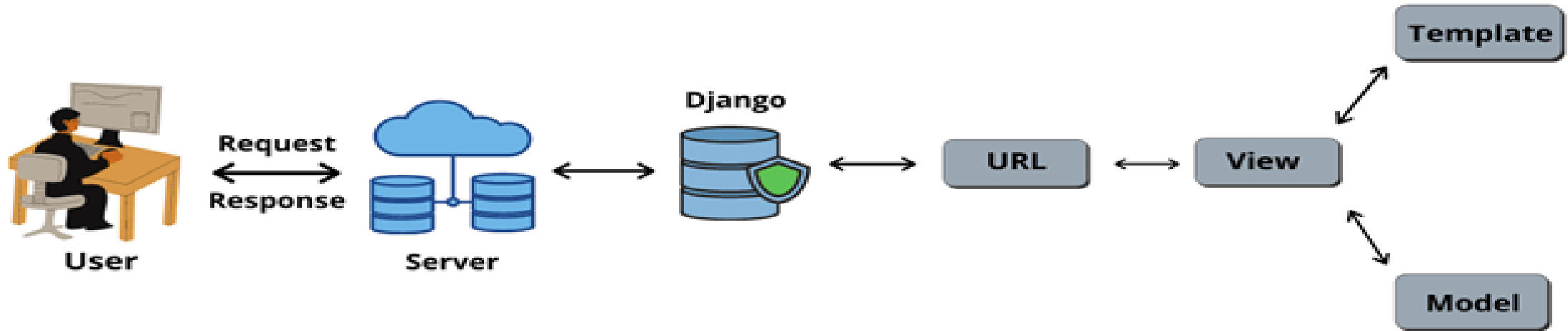
- **Separation of Concerns:** MVC separates the application into three distinct components, ensuring each component has a specific role and responsibility.
- **Modularity:** Each component can be developed, tested, and maintained independently, promoting code reusability and scalability.
- **Flexibility:** Changes in one component (e.g., the model) do not directly affect the other components (e.g., the view or controller), allowing for easier modifications and updates.

BENEFITS:

- **Maintainability:** Clear separation of concerns simplifies code maintenance and updates.
- **Scalability:** Modular design allows for easy scaling of individual components as the application grows.
- **Testability:** Components can be unit tested independently, facilitating comprehensive testing and debugging.

Application in Django:

- Django follows the MVC architectural pattern, although it's often referred to as the "Model-View-Template" (MVT) framework.
- Models represent data using Django's ORM, views handle user requests and responses, and templates render the HTML UI.



DeveloperStacks.com

| Component | Description |
|-----------|---|
| Model | Represents the application's data and business logic. It interacts with the database and manages data integrity. |
| View | Handles user input, processes requests, and returns appropriate responses. It acts as an intermediary between the model and the template. |
| Template | Renders the HTML UI by combining static templates with dynamic data provided by views. It presents the data to the user in a human-readable format. |

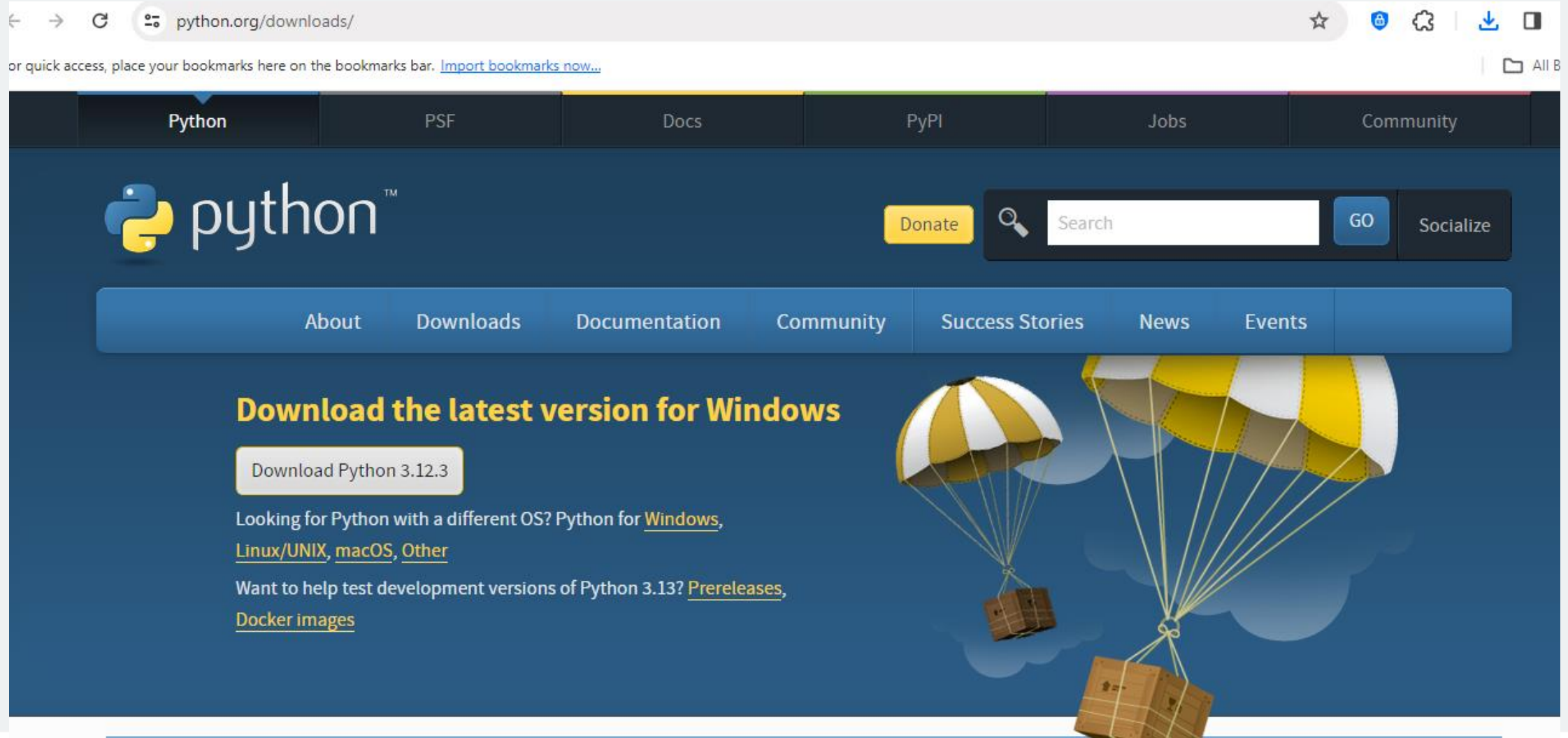
- Django follows the **MVT architectural pattern**, which is similar to MVC but with a slight variation in terminology.
- **Model**: Django's models interact with the database using its Object-Relational Mapping (ORM) system, simplifying database operations.
- **View**: Django's views handle HTTP requests and responses, processing data from models and passing it to templates for rendering.
- **Template**: Django's templates are HTML files that include placeholders for dynamic content. They are rendered with data provided by views to generate the final HTML output sent to the client

BENEFITS:

- **Separation of Concerns**: Clear separation of data, logic, and presentation layers enhances maintainability and code organization.
- **Modularity**: Each component can be developed and tested independently, promoting code reusability and scalability.
- **Rapid Development**: Django's built-in features and conventions streamline development, enabling quick creation of web applications.

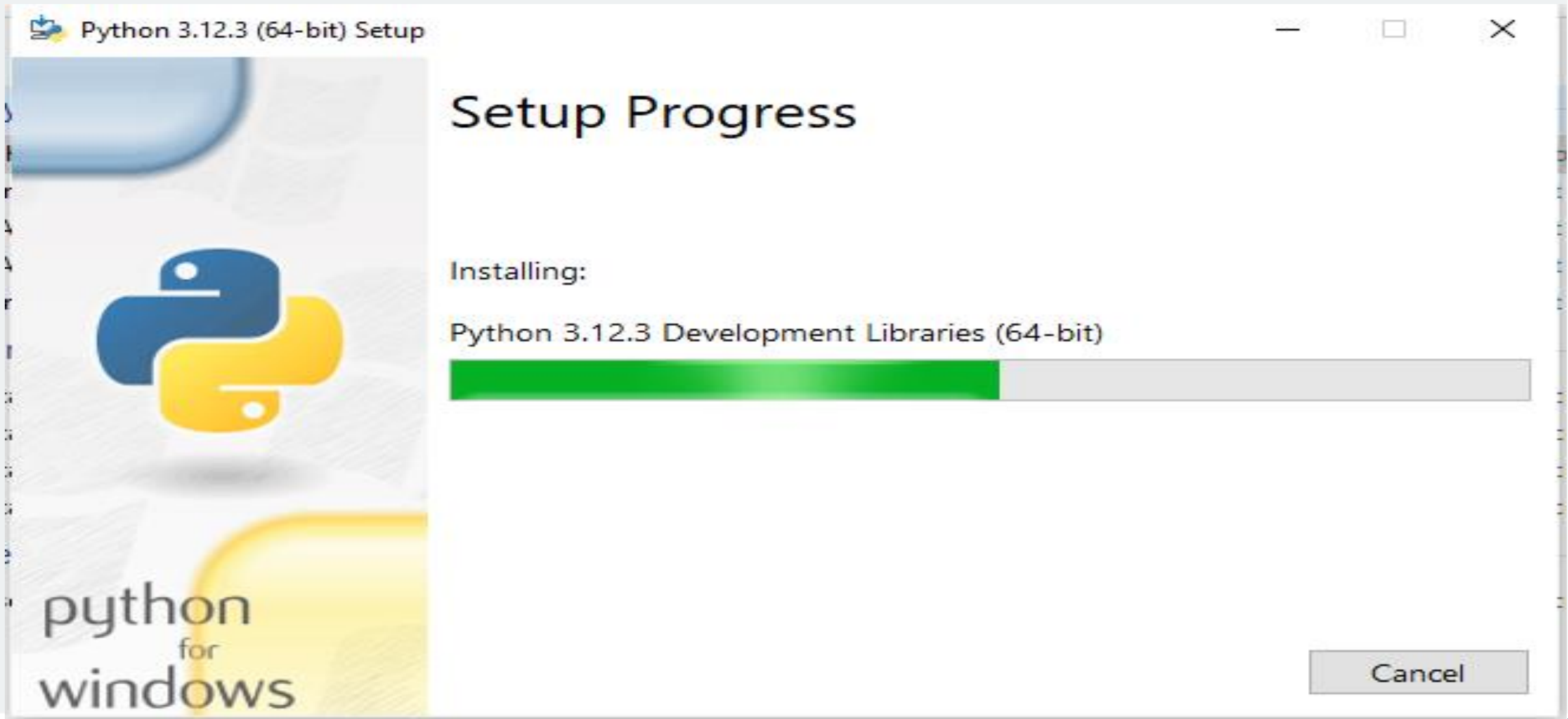
Step 1: Install Python

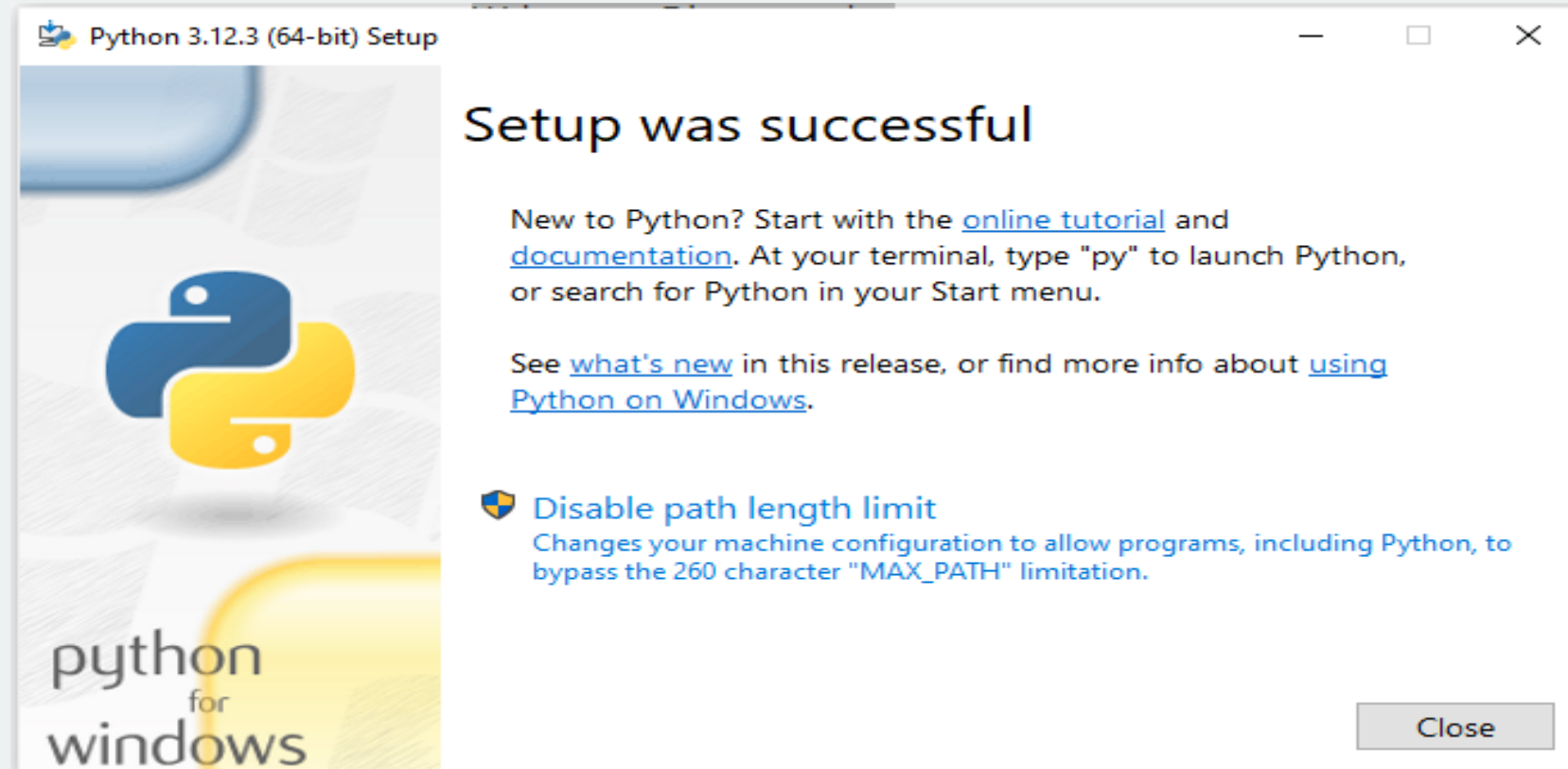
- Download Python Installer: Visit <https://www.python.org/downloads/>
- Choose the appropriate Python version for your operating system.
- Download the installer



Step 2: Run Python Installer:

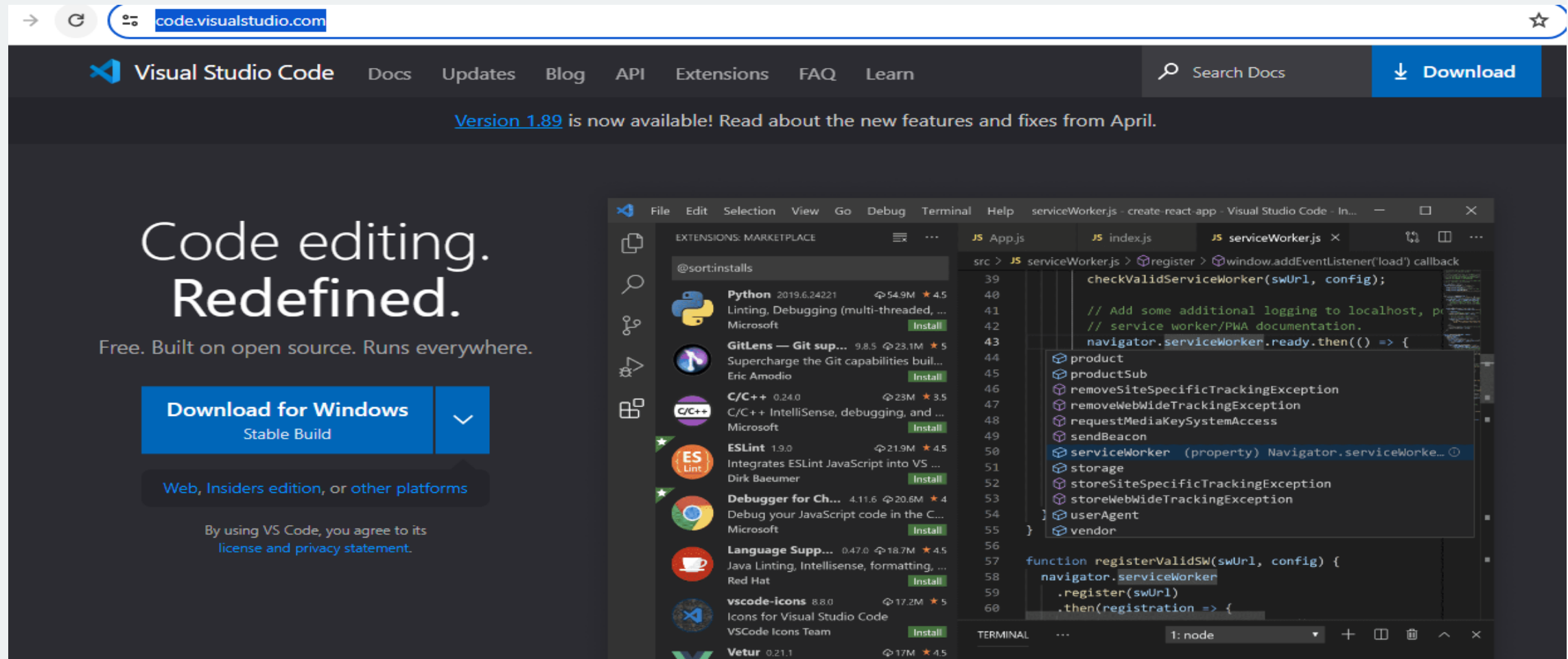
- Double-click the installer to launch the installation wizard.
- Follow on-screen instructions to complete the installation.
- Ensure Python is added to PATH during installation.





Step 3: Download VS Code Installer:

- Visit <https://code.visualstudio.com/>
- Download the installer for Windows.
- Run VS Code Installer:.



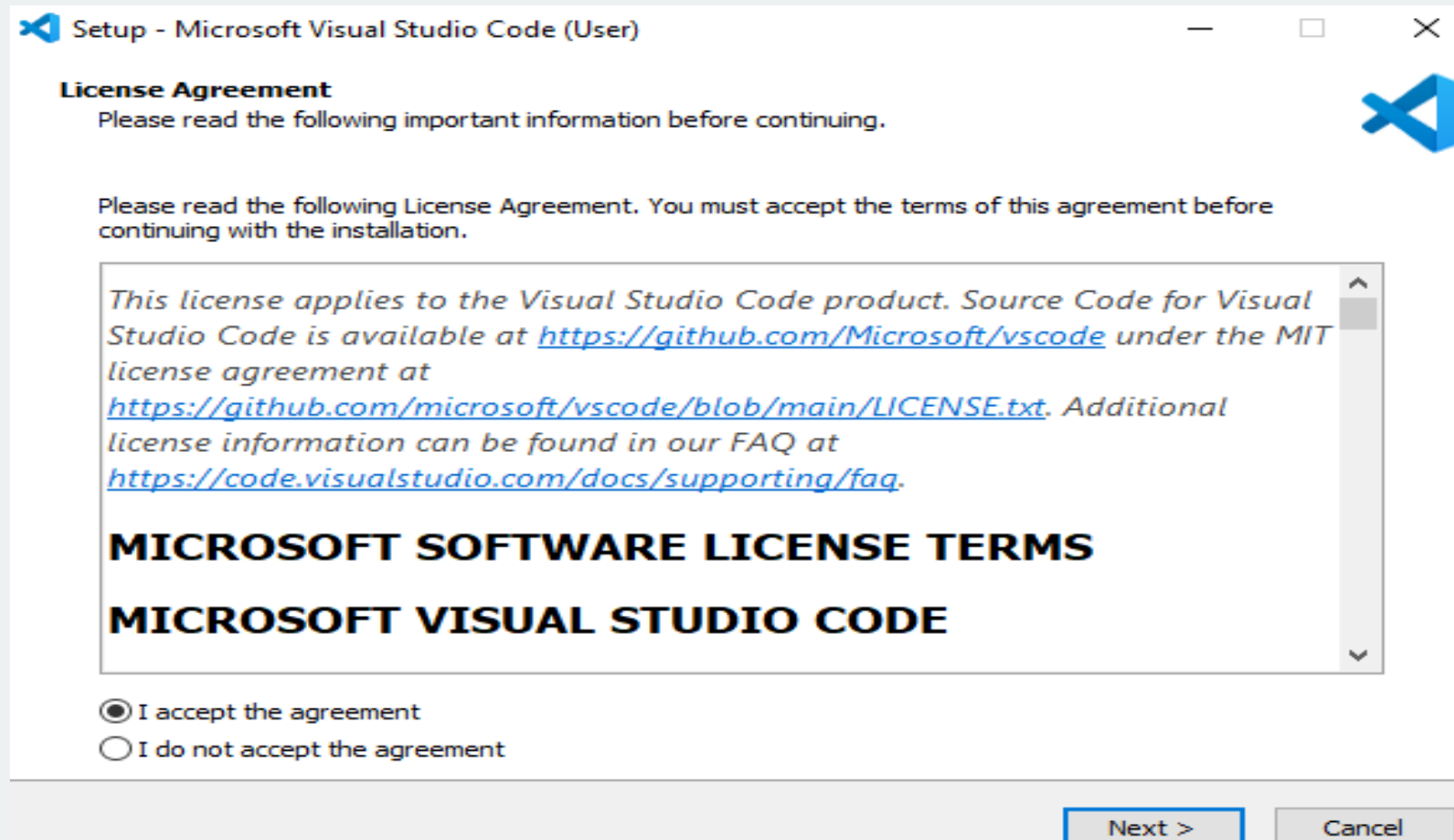
The screenshot shows the Visual Studio Code website in a web browser. The browser's address bar displays `code.visualstudio.com`. The website header includes links for Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, and Learn, along with a search bar and a 'Download' button. A banner below the header announces 'Version 1.89 is now available! Read about the new features and fixes from April.'

The main content area features the text 'Code editing. Redefined.' and 'Free. Built on open source. Runs everywhere.' Below this is a large blue button labeled 'Download for Windows' with 'Stable Build' underneath it. A smaller link below the button says 'Web, Insiders edition, or other platforms'. At the bottom, a note states 'By using VS Code, you agree to its license and privacy statement.'

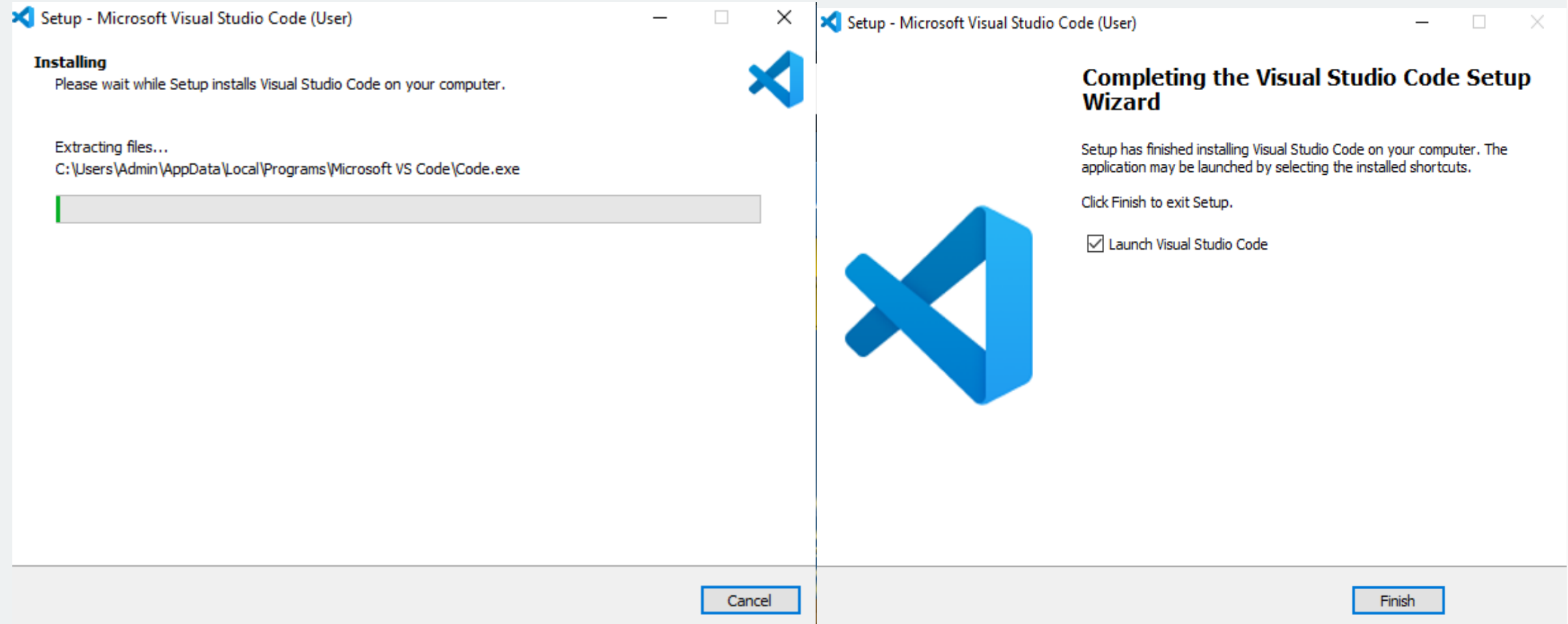
Overlaid on the bottom right of the website is a preview of the Visual Studio Code application interface. The interface shows the 'EXTENSIONS: MARKETPLACE' sidebar on the left with a list of extensions including Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support for Java, and VS Code Icons. The main editor area displays a JavaScript file named 'serviceWorker.js' with code for registering a service worker. The bottom status bar shows '1: node' in the terminal.

Step 3: Run VS Code Installer:

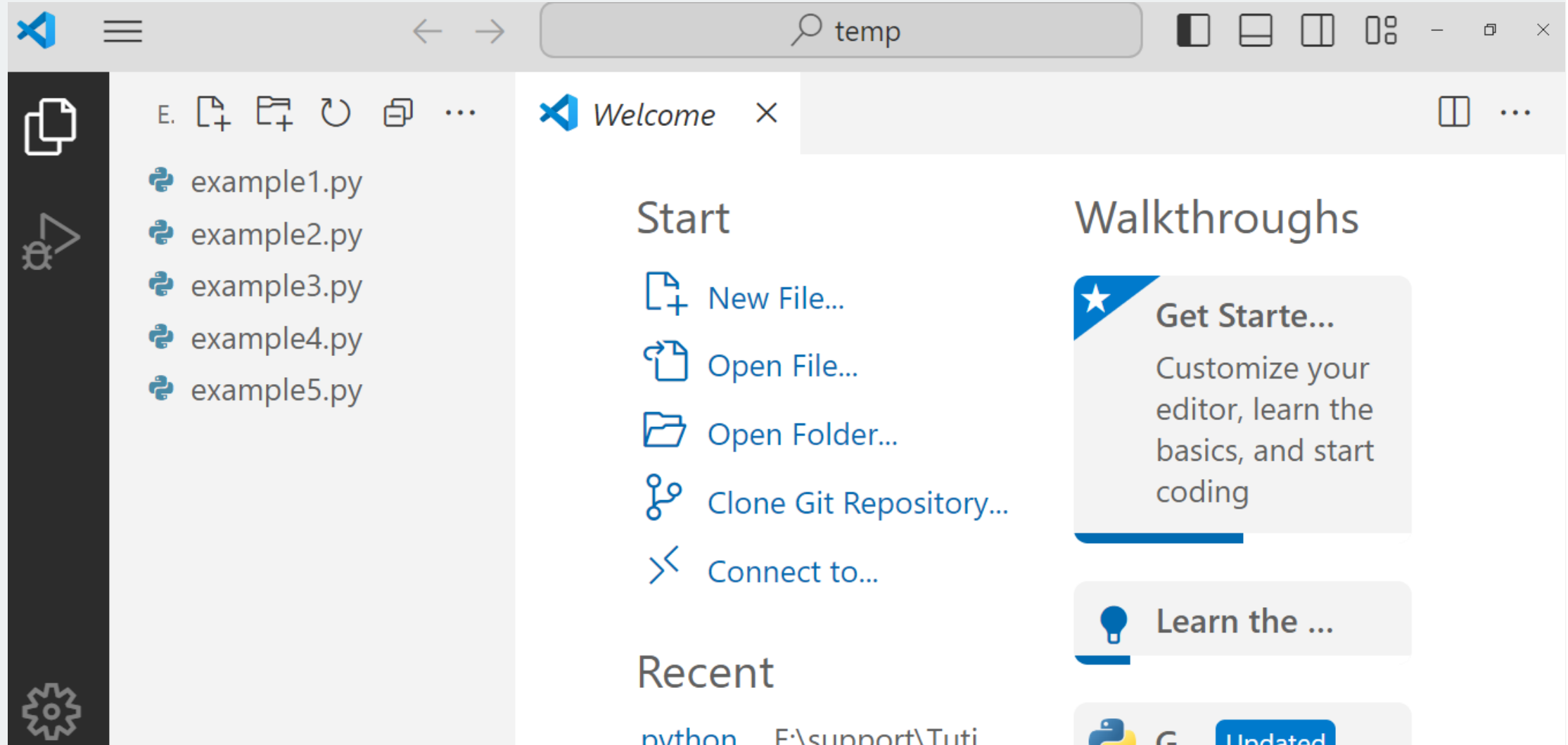
- Double-click the installer to launch the installation wizard.
- Follow on-screen instructions to complete the installation.



Run VS Code Installer:.



Run VS Code Installer:.



Step 4: Install Django

Open Command Prompt:

Press Win + R, type cmd, and press Enter.

Install Django:

Type `pip install django` and press Enter.

Wait for Django and its dependencies to install.

Verify `django-admin --version`

It shows version then Django is installed

Step 5: Create a Virtual Environment:

Following command installs the virtualenvwrapper-win package using pip.

`pip install virtualenvwrapper-win`

Following command creates a new virtual environment named "mylearn" using virtualenvwrapper-win.

`mkvirtualenv mylearn`

Following command activates the virtual environment named "mylearn". When a virtual environment is activated

`workon mylearn`

Step 6 Create a Django Project

Navigate to Desired Directory:

Use cd command to navigate to the directory where you want to create the project.

Create Django Project:

Type `django-admin startproject myproject` and press Enter.

Replace myproject with your desired project name.

Step 7: Run First Django Program

Navigate to Project Directory:

Use cd myproject to navigate to the project directory.

Run Development Server:

Type `python manage.py runserver` and press Enter.

Open a web browser and go to

<http://127.0.0.1:8000/> to view your Django project.

→ ↺ ⓘ 127.0.0.1:8000 ☆

django

View [release notes](#) for Django 5.0



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



Django Documentation
Topics, references, & how-to's

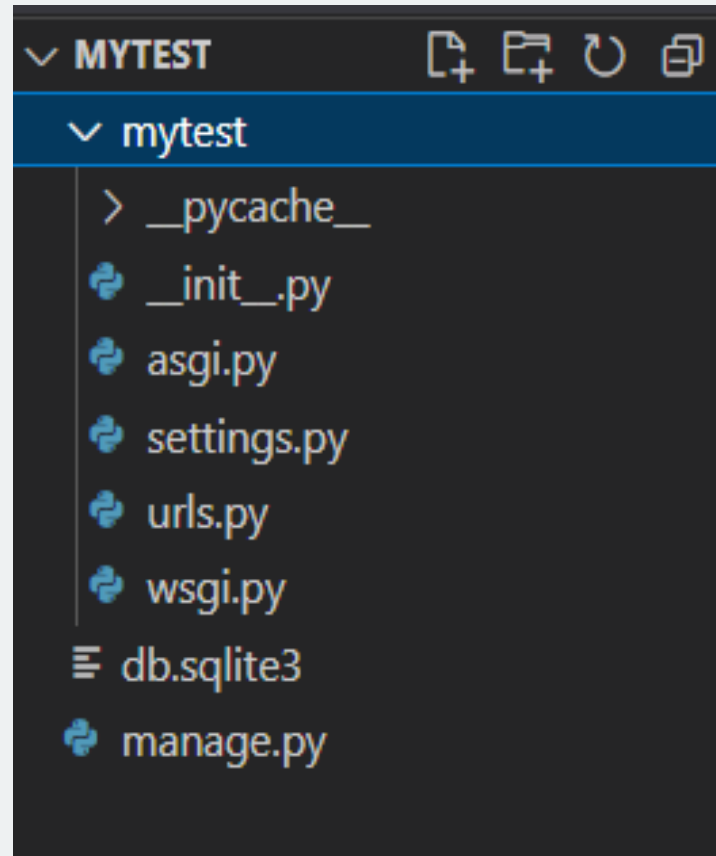


Tutorial: A Polling App
Get started with Django



Django Community
Connect, get help, or contribute

django-admin startproject mytest



| Directory/File | Description |
|-------------------|---|
| Project Directory | • Main directory for the Django project |
| manage.py | • Command-line utility for interacting with the Django project |
| settings.py | • Configuration settings for the Django project |
| urls.py | • URL patterns for the project |
| wsgi.py | • Entry point for WSGI-compatible web servers |
| asgi.py | • Entry point for ASGI-compatible web servers |
| Apps | • Directory for organizing Django applications |
| models/ | • Contains Python classes representing data models |
| views/ | • Contains logic for processing requests and generating responses |
| urls.py | • URL patterns specific to the app |
| templates/ | • HTML files defining the presentation layer |
| static/ | • Directory for CSS, JavaScript, images, etc. |
| migrations/ | • Contains Python files defining changes to the database schema |
| static_root/ | • Directory where static files are collected during deployment |
| media_root/ | • Directory for storing user-uploaded files |



THANK YOU