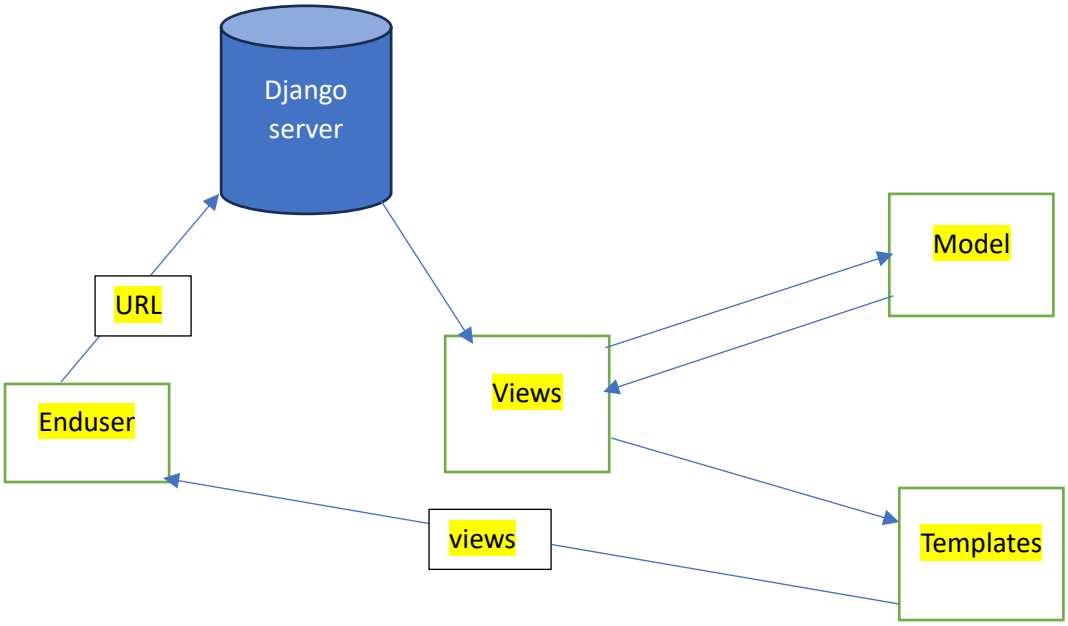
Architecture of Django Application



The architecture of a Django application typically follows the MVT (Model-View-Template).

1. **URLs:**
    - The URL dispatcher maps URLs to views, determining which view function or class should handle each incoming HTTP request.
    - In Django, you define URL patterns in a URLconf (URL configuration), which is a Python module that maps URL patterns to views.
    - The URL dispatcher matches incoming URLs against these patterns and calls the corresponding view to process the request.

2. **Models:**
    - Models represent the structure and behaviour of your application's data. Each model class typically corresponds to a database table, and instances of these classes represent individual records in the database.
    - Models define fields that represent attributes of the data, such as strings, integers, dates, etc. You can also define methods on models to encapsulate business logic related to the data.
    - Django's ORM (Object-Relational Mapping) handles interactions with the database, allowing you to query, insert, update, and delete records using Python code without writing SQL queries directly.

3. **Views:**
    - Views are Python functions or classes that receive HTTP requests from clients and return HTTP responses.
    - Views contain the business logic of your application, orchestrating interactions between models and templates to fulfill client requests.
    - Views handle tasks such as fetching data from the database, processing form submissions, rendering templates, and returning appropriate responses to clients.
    - In Django, views can be function-based or class-based. Function-based views are simple Python functions, while class-based views are Python classes that provide reusable behaviour through mixings and inheritance.

4. **Templates:**
    - Templates are responsible for generating dynamic HTML content to be rendered and displayed in a user's web browser.
    - Templates typically contain HTML markup along with placeholders and template tags that allow you to insert dynamic data and control the flow of the page.
    - Django's template engine processes templates, substituting placeholders with actual data and executing template tags to generate the final HTML output.

5. **Settings:**
    - Django's settings module contains configuration settings for your application, such as database settings, middleware configurations, installed apps, static file paths, etc.
    - Settings allow you to customize the behaviour of your Django application, such as enabling features, specifying database connections, setting debug mode, etc.

6. **Admin Interface:**
    - Django provides a built-in admin interface that allows you to manage application data through a web-based interface.
    - The admin interface is automatically generated based on your application's models, providing CRUD (Create, Read, Update, Delete) functionality for managing data.
    - You can customize the admin interface by registering models, defining custom admin classes, and overriding default templates and behaviour.