Введение

Разработка современных Web-приложений на базе алгоритмического языка *Python* предполагает применение специализированных встроенных фреймворков, таких как *Django*. Особенностью и преимуществом *Django* является наличие готовых решений для широкого класса задач, таких как:

- готовые компоненты для создания приложений, имеющих распространенную архитектуру MVC;
- объектно-реляционное отображение (ORM) для работы с базами данных, автоматически преобразующее операции над объектами Python в SQL-запросы и тем самым упрощающее взаимодействие с базой данных;
- удобные средства администрирования Web-приложений;
- инструменты для доступа к данным в современных форматах, обеспечивающих взаимодействие с крупнейшими владельцами информационных ресурсов через стандарт REST API;
- набор инструментов для построения модульных монолитных приложений, которые организованы в автономные, слабосвязанные приложения (apps), каждое из которых отвечает за определённую функциональную область (например, аутентификацию, блог, интернет-магазин). Чёткое разделение ответственности на уровне приложений и их взаимодействие через стандартные механизмы фреймворка (например, импорт моделей и сервисов) позволяет в будущем, при необходимости, относительно легко провести эволюционный рефакторинг и масштабировать систему, преобразовав эти модули в независимые микросервисы, взаимодействующие через REST API или брокеры сообщений;
- встроенные средства обеспечения безопасности данных, такие как:

- аутентификация пользователей;
- защита от CSRF-атак (выполнения действий от имени другого пользователя);
- защита от XSS-атак (когда злоумышленник пытается вставить вредоносный программный код через веб-сайт, который будет выполнен на устройстве пользователя, посетившего веб-страницу) с помощью экранирования;
- встроенные механизмы управления доступом к ресурсам на основе политики CORS (Cross-Origin Resource Sharing), которые обеспечивают безопасное междоменное взаимодействие, позволяя гибко настраивать разрешенные источники запросов, HTTP-методы и заголовки, что критически важно при разделении фронтенда и бэкенда или построении распределенных систем;
- комплексная защита от SQL-инъекций, достигаемая за счет применения ORM, которое использует параметризованные запросы, и встроенных форм с автоматическим экранированием пользовательского ввода, что полностью исключает возможность выполнения вредоносного кода на уровне базы данных.

В настоящем учебном пособии рассматриваются вышеуказанные решения на базе двух наиболее распространенных средств разработки *Visual Studio Code* и *PyCharm*. Пособие составлено в виде пошаговых инструкций для реализации примеров решения задач.

Авторы выражают благодарность за помощь в подготовке пособия Трошину Павлу Игоревичу, Антимонову Денису, Муразяну Матвею Рубеновичу.

1 Архитектурный паттерн Django — MTV

Архитектура *Django* основана на модификации классического паттерна *Model-View-Controller* (MVC), известной как *Model-Template-View* (MTV). Данный паттерн обеспечивает четкое разделение ответственности между компонентами приложения, что является залогом его масштабируемости и простоты поддержки.

Классический паттерн MVC разделяет приложение на три основные части.

Model (модель): инкапсулирует данные и бизнес-логику, обеспечивая доступ к состоянию приложения и методам для его изменения.

View (представление): отвечает за преобразование данных в форму, пригодную для представления пользователю (как правило, в виде HTML).

Controller (контроллер): принимает входные данные пользователя, взаимодействует с моделью и выбирает подходящее представление для формирования ответа.

Django формализует и уточняет эту структуру, перераспределяя обязанности контроллера.

Рассмотрим назначения и взаимодействия каждого компонента архитектуры MTV в сравнении с классикой MVC (табл. 1).

Понимание эволюции классического паттерна MVC в архитектуру *Django* MTV является фундаментальным для осознанного структурирования кода.

Django не отказывается от принципов MVC, а проводит их строгую ревизию и переименование компонентов для большей ясности в контексте веб-разработки. Ключевое изменение — *декомпозиция монолитного Controller* на две отдельные, четко специализированные сущности.

Роль классического контроллера в *Django* выполняет тандем из двух компонентов — View + URLconf:

Таблица 1

Сравнение архитектурных паттернов MVC и MTV

Компонент	MVC (классический)	MTV (*Django*)	Пояснение
Управление данными и бизнес-логикой	Model	Model	Назначение компонента идентично. Отвечает за структуру данных, валидацию, бизнес-правила и абстракцию базы данных (через ORM)
Определение внешнего представления	View	Template	Назначение компонента идентично. Отвечает за визуальный слой (HTML), используя шаблонизатор для динамической подстановки данных
Обработка логики запроса	Controller	View	View в *Django* перенимает логическую функцию Controller. Это код, который получает запрос, работает с Model, принимает решения и инициирует ответ
Маршрутизация и диспетчеризация	Часть Controller	URLconf	*Django* выносит функцию маршрутизации в отдельный компонент. URLconf (файл urls.py) выступает как контроллер маршрутизации, отвечая за сопоставление URL и View

- **URLconf** — это «мозг» управления запросами. Он анализирует URL и решает, *какой* обработчик (View) должен быть вызван. Он не содержит бизнес-логики;
- **View** — это «исполнительный» орган. Он получает запрос от URLconf, выполняет всю необходимую работу (обращается к Model, проводит вычисления) и решает, какой результат вернуть (результат рендеринга* шаблона, редирект, JSON).

Такое разделение ответственности (URLconf для маршрутизации, View для логики) делает архитектуру *Django* более модульной, тестируемой и понятной. Разработчик всегда точно знает, где искать правило маршрутизации (urls.py) и где искать код обработки конкретного запроса (views.py).

---

* Рендеринг шаблона — процесс подстановки данных в шаблон с последующим формированием итогового HTML-документа, который отправляется веб-сервером клиенту.

Взаимодействие компонентов

Жизненный цикл обработки HTTP-запроса в *Django* выглядит следующим образом.

1. **Запрос:** *Django* получает HTTP-запрос.
2. **Маршрутизация:** фреймворк определяет, какое View должно обработать запрос путем сопоставления запрошенного URL с паттернами, объявленными в URLconf.
3. **Обработка:** выбранное View выполняет всю необходимую логику: обращается к базе данных через Models, производит вычисления, работает с формами.
4. **Рендеринг:** для генерации контента View загружает соответствующий Template и передает ему данные для отображения (контекст). Template рендерится в конечный HTML.

Ответ: View возвращает сгенерированный HTML в виде HTTP-ответа клиенту.

Архитектура MTV является не просто переименованием MVC, а его практической адаптацией для веб-разработки. Ключевое отличие MTV — явное выделение маршрутизации (URLconf) в отдельный структурный компонент кода. Однако концептуально MTV содержит те же компоненты, что и MVC:

$$\text{Контроллер}_{mvc} = (\text{View} + \text{URLconf})_{mtv};$$
$$\text{View}_{mvc} = \text{Template}_{mtv}.$$

Такое выделение маршрутизации обеспечивает:

- *предсказуемость* — код организован по четким правилам;
- *тестируемость* — компоненты изолированы друг от друга;
- *сопровождаемость* — разработчику легко локализовать и изменить конкретную функциональность.

Последующие разделы детально раскроют каждый компонент архитектуры *Django*.