Информатика

Баранов Максим Александрович

План занятия

1) Основы WEB-разработки

2) Django

3) Первый проект

Фреймворки

Фреймворк – программная платформа, определяющая структуру разрабатываемого программного решения.

- Фреймворк облегчает разработку компонентов программного решения.
- Фреймворк влияет на выбор того или иного паттерна (шаблона) проектирования, например, использование MVC схемы разделения данных.

Фреймворки Python

- **Django**
- **Pyramid**
- **TurboGears**
- **Flask**
- **Tornado**

Фреймворки Python. Pyramid

Особенности

- Инструментарий для работы со статичными ассетами.
- Генерация URL
- Предикаты

Пример использования: Substance-D, Charte.ca

Фреймворки Python. Pyramid

плюсы

- Гибкость
- Работа с Ajax-запросами
- Поддержка SQLAlchemy

МИНУСЫ

- Сложность подготовки к работе
- Излишняя нагруженность для простых приложений

Фреймворки Python. TurboGears

Особенности

- Интеграция с библиотеками JS (MochiKit) для работы
- Одновременная поддержка нескольких БД
- поддерживает SQLAlchemy, Genshi, WebOb и Repoze

Пример использования: Apache Allura от SourceForge

Фреймворки Python. TurboGears

плюсы

- Гибкость и расширяемость
- Open Source проект

МИНУСЫ

- Сложность работы

Фреймворки Python. Flask

Особенности

- Встроенный дебаггер
- Шаблоны Jinja2
- Безопасность работы с куки на клиенте
- Поддержка юнит-тестов

Пример использования: Netflix

Фреймворки Python. Flask

плюсы

- Быстрое прототипирование
- Множество возможностей за счет более низкоуровневой работы

МИНУСЫ

- Сложность работы
- Высокий порог вхождения

Фреймворки Python. Tornado

Особенности

- Основная «фишка» - асинхронность работы
- Работа в реальном времени
- Возможности аутентификации
- Может выдерживать проблемы 10 000 соединений

Пример использования: Uploadcare

Фреймворки Python. Tornado

плюсы

- Поддержка множественных пользовательских соединений
- Работа в реальном времени
- Поддержка переводов и локализации

МИНУСЫ

- Сложность работы
- Высокий порог вхождения

Фреймворки Python. Django

Особенности

- Контроль версий для БД (миграции).
- Маршрутизация URL.
- Поддержка веб-серверов, аутентификации, интернационализации

Пример использования: Spotify

Фреймворки Python. Django

плюсы

- Множество библиотек
- Сообщество и документация
- Масштабируемость

минусы

- Проблемы при работе с WebSockets
- Готовые библиотеки могут снижать гибкость

Фреймворки Python. Django

1. **Выполнив команду**

>> django-admin.py startproject first

Был создан проект first, где:

- *manage.py* ***– скрип управления проектом (добавление новых приложений, сборка файлов и пр.);***
- *__init__.py* ***необходим для определения директории/ пакета;***
- *settings.py* ***– глобальные настройки проекта (пути, БД, подключения и пр.);***
- *urls.py* ***– файл привязок url (по какому адресу вызывается тот или иной скрипт);***
- *wsgi.py* ***– WSGI-приложение для работы с web-сервером***

```
./first
├── first
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Фреймворки Python. Django

1. **Выполнив команду**

>> ./manage.py startapp main

Было создано приложение main, где:

- *__init__.py* ***необходим для определения директории/пакета;***
- *models.py* ***– модели приложения***
- *tests.py* ***– шаблон для тестов;***
- *views.py* ***– представления приложения***

```
first
├── first
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── main
│   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── manage.py
```

Фреймворки Python. Django

Порядок работы

1. ***Браузер запрашивает URL, сервер передает запрос в Django***
2. ***Django формирует объект HttpRequest***
3. ***Запускаются методы process_request Middleware-классов****
4. ***Поиск соответствия в файле uels.py***
5. ***Запускаются методы process_request Middleware-классов***
6. ***Передается Request с параметрами (при их определении в файле urls.py) в найденное представление***
7. ***Выполняется функция представления, возвращается объект HttpResponse***
8. ***Запускаются методы process_request Middleware-классов***
9. ***Django возвращает ответ на web-сервер, а затем – в браузер***

Фреймворки Python. Django

Middleware-классы

Выполняют работу по модификации запросов и ответов
Например, django.contrib.auth добавляет к каждому запросу поле с именем пользователя, который осуществил данный запрос

Фреймворки Python. Django

Существует проект *example_project* с приложением *pages* и файлом шаблона *home.html*

Структура может выглядеть следующим образом: в приложении *pages* создается каталог *templates*, а в нем каталог *pages*, в котором размещается файл шаблона *home.html*

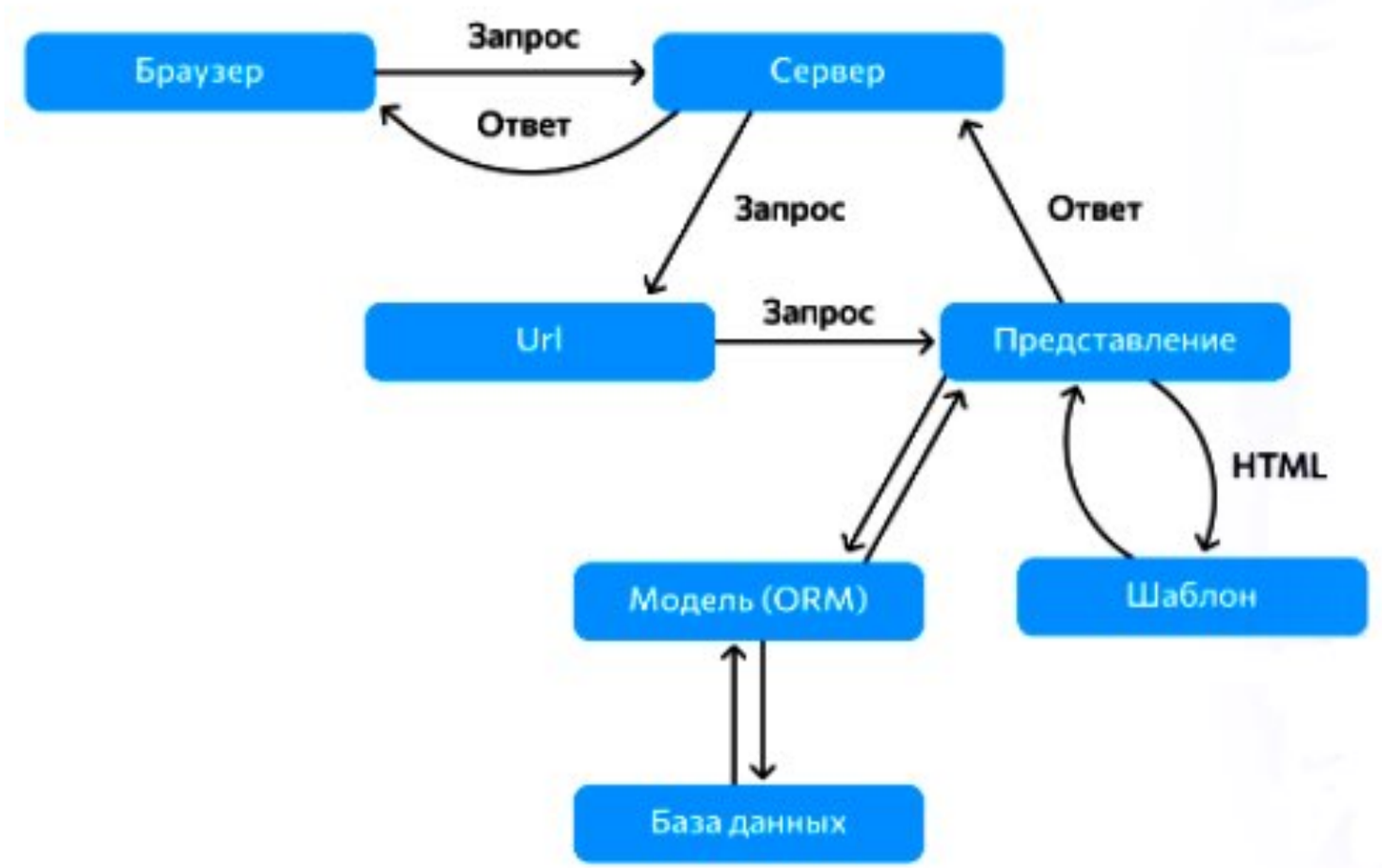
```
├── example_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│   └── pages
│       ├── __init__.py
│       ├── admin.py
│       ├── apps.py
│       ├── models.py
│       ├── tests.py
│       └── views.py
│       ├── templates
│           ├── pages
│               ├── home.html
└── manage.py
```

** загрузчик шаблонов сначала ищет их в приложениях, а затем при настроенном DIRS, в каталоге templates проекта.*

Полнота
Многогранность
Безопасность
Масштабируемость
Переносимость

Фреймворки Python. Django. MVC/MTV

Model (Модель)
View (Представление)
Controller (Контроллер)

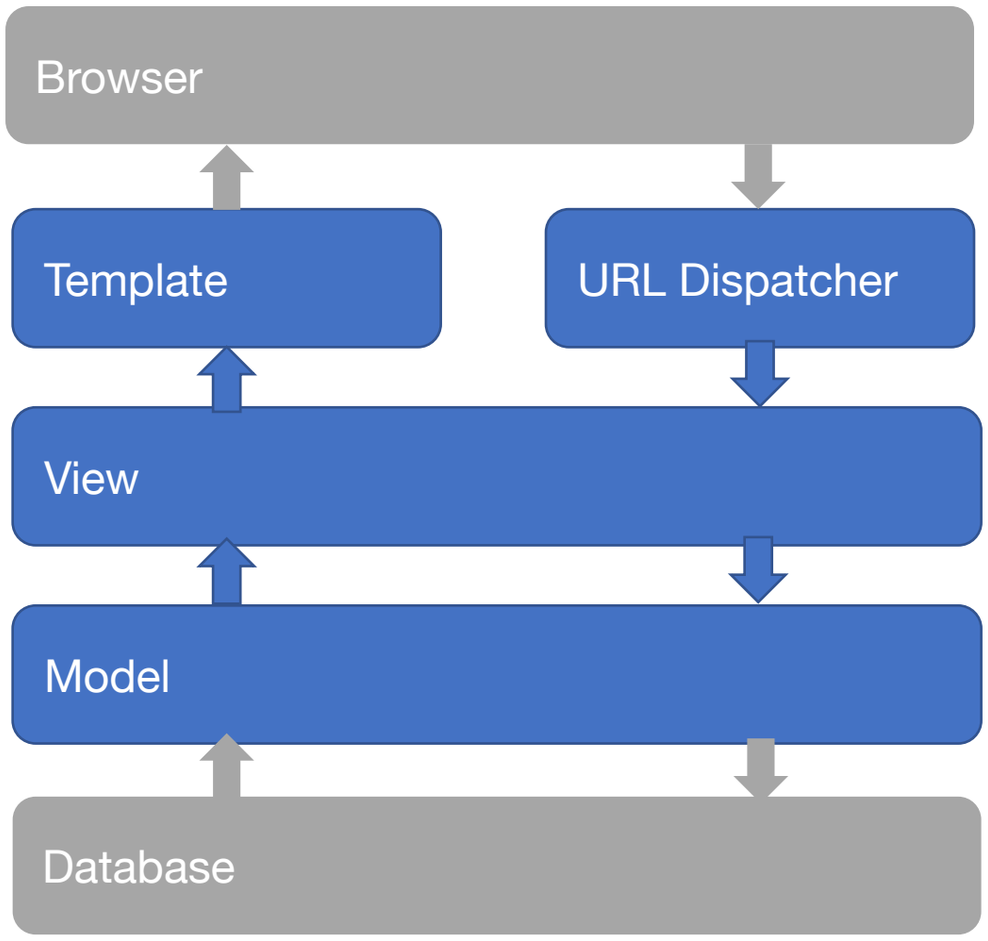


Model (модель)
Template (шаблон)
View (представление)

Django MVT

получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ.

представляет логику представления в виде сгенерированной разметки html.

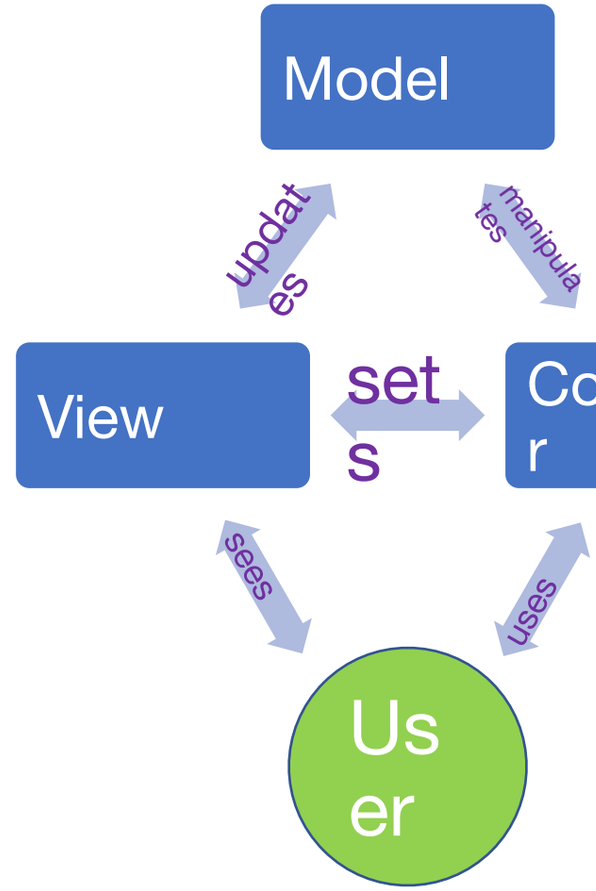


при получение запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.

описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.

Model-View-Controller (MVC)

— архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.



Шаблон *MVC* позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

Модель (Model)

Представление (View)

Поведение (Controller)

Model-View-Template (MVT)

— *модификация распространённого в веб-программировании паттерна MVC.*

Model, уровень доступа к данным. Здесь сосредоточена вся информация о данных: как получить к ним доступ, как осуществлять контроль, каково их поведение, каковы отношения между данными.

Template (шаблон), уровень отображения. Здесь принимаются решения, относящиеся к представлению данных: как следует отображать данные на веб-странице или в ином документе.

View, уровень логики. Здесь расположена логика доступа к модели и выбора подходящего шаблона (или шаблонов). Это мост между моделями и шаблонами.