GeekBrains

Урок 2

Архитектура Python-приложений

Виды и стили архитектур приложений

На этом уроке разберём

Что такое архитектура ПО и зачем она нужна?

Виды архитектур Python-приложений

Почему мы изучаем архитектуру и паттерны проектирования

1. Архитектура поможет сделать ваш код красивым, отсутствие превратит в антипаттерн.
2. Принципы, как и у людей: либо они есть, либо их нет.
3. Паттерны сэкономят кучу вашего времени и избавят от уже набитых до вас шишек.

План урока

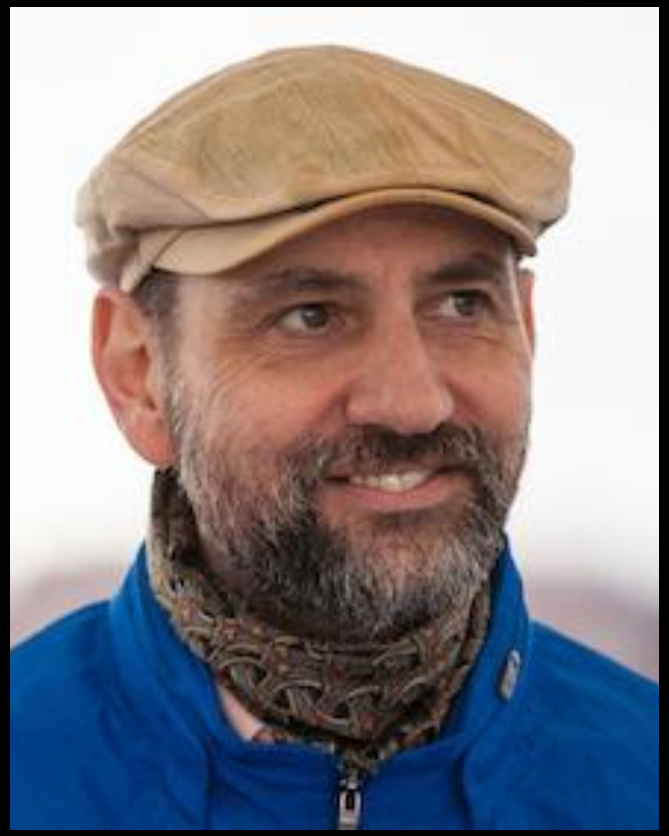
1. Что такое архитектура программного обеспечения.
2. Виды архитектуры приложений.
3. WSGI-сервер (продолжение). Параметры запроса.

Что такое архитектура программного обеспечения

Архитектура

Архитектура (ἀρχιτέκτων), от ἀρχι — главный, τέκτων — плотник, строитель.

Архитектура



.... согласие в вопросе идентификации главных компонентов системы и способов их взаимодействия, а также выбор таких решений, которые интерпретируются как основополагающие и не подлежащие изменению в будущем.

Мартин Фаулер https://martinfowler.com

Мотивация

Традиция	Программирование	Функция	Мотивация в единицу времени
Архитектор	Архитектор	Задаёт архитектуру системы	$$$
Проектировщик	Developer (Senior)	Проектирует систему. используя крупные блоки — паттерны	$$
Строитель	Coder / программист (Junior)	Реализует	$

Архитектура: преимущества

Повышает скорость разработки

Повышает качество разработки

Снижает риски и вероятность провала

Снижает стоимость разработки

Архитектура: основные принципы

Создавайте с расчётом на будущее

Учитывайте вновь возникающие требования

Используйте UML

Архитектура: основные принципы

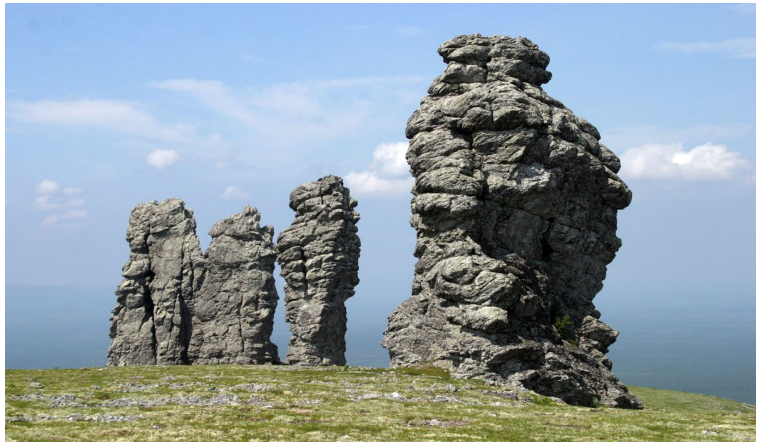
Используйте модели и визуализацию

Используйте DDD (Domain Driven Design)

Выявляйте ключевые инженерные решения

Усложняйте дизайн постепенно

Эрозия архитектуры



Архитектура: восстановление

Набор методов выделения архитектурной информации на основе анализа нижележащих слоев дизайна, в том числе из программного кода.

Виды архитектур

Архитектура «клиент-сервер»

Плюсы

- Высокая безопасность
- Простое администрирование
- Простота обслуживания

Минусы

- Тенденция тесного связывания данных и бизнес-логики приложения.
- Зависимость от центрального сервера.

Многослойная архитектура

Плюсы

- Каждый слой самодостаточен и независим
- Многовариантность реализации
- Стандартизация слоя

Минусы

- Слои разбивают приложение по функциям, а не по смыслу
- Каскад изменений в слоях
- Чрезмерное расслоение может понизить производительность

Проектирование на основе предметной области

Плюсы

- Упрощает обмен информацией
- Упрощает модификацию при изменении внешних условий
- Объекты модели предметной прекрасно тестируются

Минусы

- Тяжело «ложится» на реляционные БД

Сервисно-ориентированная архитектура (SOA)

Плюсы

- Разделяет функции на отдельные слабо связанные блоки.
- Стандартизованные интерфейсы.
- Мультиплатформенность. Быстрое изменение функционала.

Минусы

- Требует инфраструктуры

Архитектура – шина сообщений

- Реализует систему связи между взаимодействующими программными приложениями
- Концентрация обмена сообщениями

WSGI-сервер

Разделение запросов get и post

Данные get-запроса

Данные post-запроса

Практическое задание

В этой самостоятельной работе тренируем умения:

Разделять get и post запрос внутри wsgi-фреймворка

Получать и декодировать параметры post-запроса

Зачем:

Чтобы уметь обрабатывать
разные типы веб-запросов

Последовательность действий:

1. Добавить в свой WSGI-фреймворк возможность обработки post-запроса.
2. Добавить в свой WSGI-фреймворк возможность получения данных из post-запроса.
3. Дополнительно можно добавить возможность получения данных из get-запроса.
4. В проект добавить страницу контактов, на которой пользователь может отправить нам сообщение (пользователь вводит тему сообщения, его текст, свой email).
5. После отправки реализовать сохранение сообщения в файл, либо вывести сообщение в терминал (базу данных пока не используем)

Спасибо!

Каждый день вы становитесь лучше :)