Yii2: Структура большого проекта

Архитектура, надёжность, поддерживаемость

Александр Макаров

Yii core team

http://slides.rmcreative.ru/2017/yii2-big-projects/

Архитектура?

Чёткого определения нет :)

Ряд решений о том, как взаимодействуют отдельные части системы. Это, в основном, глобальные решения, которые тяжело изменить потом.

— На какие элементы разбить код?
— Как эти элементы взаимодействуют?

Архитектура — это про интерфейсы и контракты. Не про код и конкретную реализацию.

Что такое интерфейс?

Наследование знают все. А что такое инкапсуляция и полиморфизм? Какой их смысл?

Фреймворк — не архитектура

Он не построит архитектуру за вас.

Фреймворк **ничего не знает об элементах вашего приложения** и **не может определить интерфейсы для взаимодействия этих элементов**.

Но может дать начальный шаблон и инфраструктуру. В случае Yii это MVC.

basic/advanced — не догма и не шаблон полноценной архитектуры

Кстати, про MVC...

Controller

— Принимает данные извне (GET, POST, консольный ввод, ...).

— Отдаёт данные в нужном виде в Model и View.

— **Не реализует логику, не занимается форматированием или формированием ответа.**

View

— Получает подготовленные контроллером данные.
— Форматирует данные для ответа.
— **Никогда не работает с внешними данными, базой или пользовательским вводом напрямую.**

Model

— **Не Model в Yii! Не ActiveRecord!**
— **M в MVC - не один класс, а целый доменный слой (вся логика приложения).**
— Получает подготовленные контроллером данные, обрабатывает их, возвращает результат.
— **Никогда не работает с внешними данными или пользовательским вводом напрямую.**
— **Не занимается форматированием или формированием ответа.**

Паттерны проектирования — не архитектура.

Это проверенные варианты решения более-менее распространённых проблем. У каждого паттерна есть как плюсы, так и минусы.

Зачем нужна архитектура?

Для борьбы со сложностью.

Цель — сделать понятным каждый уровень абстракции.

— Для изменения под реалии бизнеса.
— Чтобы не приходилось всё переписывать с нуля.
— Сделать с первого раза правильно не просто.

Хорошая архитектура — это дорого. Плохая — еще дороже.

"If you think good architecture is expensive, try bad architecture"
– Brian Foote and Joseph Yoder

SOLID

Single Responsibility

Класс должен делать что-то одно.

Open-closed

Класс или модуль (несколько связанных классов) должен скрывать детали реализации (то есть как именно он работает внутри), но иметь чётко определённый интерфейс, который позволяет как использовать модуль или класс (public методы), так и расширять его наследованием (protected и public методы).

Liskov substitution

Принцип об иерархии и наследовании. Классическое определение очень запутанное. Можно проще.

Когда мы реализовываем новый класс, наследуясь от существующего, новый должен быть с тем же интерфейсом и вести себя абсолютно так же в тех же ситуациях. Так, чтобы программа работала, если подсунуть ей как родителя, так и наследника.

Interface segregation

Интерфейс не должен определять больше функциональности, чем используется за один раз. Это как Single Responsibility, только для интерфейсов. Если интерфейс описывает больше одной задачи, разбиваем его на несколько интерфейсов.

Dependency inversion

Класс должен объявлять зависимости (то, что ему нужно) через интерфейсы, но никогда не получать их самостоятельно.

Хорошие и плохие зависимости

- Cohesion - связность.
- Coupling - связанность.

Cohesion

Зависимость классов, которые служат единой цели.

Coupling

Зависимость классов, цели которых отличаются.

Cohesion - хорошо. Coupling - плохо.

— **Служащие одной цели классы** собираются в модули (Не модули Yii! Не конкретные классы! Логические рамки).
— **Внутри модуля** используем его классы напрямую. **Нет смысла абстрагироваться**.
— Используемые модулем классы, но **не связанные с ним**, используются **только через интерфейсы**.
— Модулю наплевать на то, как он получит зависимости.

Довольно много мест в Yii сделаны не по SOLID. На это есть причины.

Чтобы нарушать правила необходимо их знать и уметь соблюдать.

Попробуйте делать в своих проектах правильно.

Active Record

— Делает одновременно слишком много.
— Притягивает к себе бизнес-логику (ей не место в моделях AR).
— Позволяет достичь цели очень быстро.
— Удобен.

Переабстрагирование

Чрезмерное увлечение абстракцией может привести к тому, что каждый отдельный её уровень слишком прост, а взаимодействие уровней слишком сложно.

Документирование архитектурных решений

Что описывать?

— Модули: структура компонентов.
— Компоненты и коннекторы: взаимодействие компонентов.
— Размещение: физическое распределение компонентов по серверам.

Чем описывать?

– Текст
– UML

Как сделать приложение надёжным?

Исправляю одно, отваливается другое...

TDD/BDD

Тестировать.

Ключевые места хорошей архитектуры не сложно тестировать.

Архитектура зависит в большей степени от предметной области, а не от применяемых фреймворков и библиотек.

DDD

Domain Driven Design

Программист недостаточно знает автоматизируемый им процесс.

Есть человек, который знает его от и до. Это доменный эксперт.

— Большие проекты (> 6 месяцев)
— Цель — получить архитектуру, близкую к реальному процессу
— Использовать термины реального процесса

Структурные блоки

– Value object — значение, которое может включать в себя другие значения.
– Entity — объекты, которые можно идетифицировать. Взяв два объекта и сравнив их, можно определить, тот же это объект или нет.
– Aggregate root — группа объектов. Имеет главный Entity, без которого вся группа не имеет смысла.

Доменный слой не зависит ни от чего, только от PHP.

Главные паттерны

— Инфраструктура: Repository — сохраняет чистые объекты и загружает их. Чаще всего речь идёт про базу данных. Не AR!
— Factory — создаёт объекты.
— Доменный сервис — использует entity для какой-то полезной работы. Всё ещё не зависит ни от чего вне домена.
— Сервис приложения — предоставлятся фреймворком (components). Может использовать доменный сервис для работы с доменом.

Почитать

- Руководство Microsoft по проектированию архитектуры приложений
- Domain Driven Design Quickly (InfoQ)
- Catalog of Patterns of Enterprise Application Architecture
- Clean Architecture

Время вопросов!

- http://slides.rmcreative.ru/2017/yii2-big-projects/
- http://www.yiiframework.com/