УДК 004 **Информационные технологии**

Яровая Екатерина Владимировна, *магистранта Гродненского государственного университета им Я. Купалы, Беларусь, г. Гродно*

ПРИНЦИПЫ ПОСТРОЕНИЯ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Аннотация: В статье рассмотрим, как правильно реализовать архитектуру программного обеспечения. Обсудим самые популярные паттерны, используемые для построения программного обеспечения. Будут затронуты, какие проблемы могут быть при некорректном подходе реализации архитектуры. Плюсы и минусы использования паттернов микросервесной архитектуры и монолитной, в каких случаях и для каких проектов могут быть использованы эти патерны.

Ключевые слова: архитектурные паттерны, типы архитектур, микросервисная архитектура, монолитная архитектура.

Annotation: In this article we will consider how to implement the software architecture correctly. We will discuss the most popular patterns used to build software. We will touch upon what problems can occur if the architecture is implemented incorrectly. Pluses and minuses of using patterns of microserial architecture and monolithic, in what cases and for what projects these patterns can be used.

Keywords: architectural patterns, types of architectures, microservices architecture, monolithic architecture.

Многие разработчики слышали такое выражение, как архитектура программного обеспечения, может кто-то знает, что есть такая должность как архитектор. Только поймите меня правильно, мы не говорим про архитектора зданий, если будет упомянуто архитектору без дополнения, это все равно будет

архитектор программного обеспечения, для сокращения –архитектор.

Для начала нужно ответить на такой вопрос, что такое Архитектура? Архитектура приложения - это набор методов и шаблонов, дробление бизнес процессов на более мелкие, которые помогают команде разработки создавать хорошо структурированные приложения, также это можно понимать, как перечень неких фундаментальных решений, которые после принятия и согласования с трудом поддаются изменению [3]. В описание Архитектуры входит ряд задач такие как, описание компонентов системы и их взаимодействия между собой, также в задачу архитектора входит определение самых важных классов и функций которые будут составлять основу приложений или обязательны для выполнение задач, определять организацию данных, общение между сервисами, как будут данные храниться, какие механизмы будут запускаться при сбоях или как мы будем обрабатывать исключительные операции, какой уровень безопасности будет, но тут есть исключение, это входит в обязанности архитектора, если нету инженера по безопасности. В обязанности Архитектора также входит оценивание и масштабирование ресурсов, ресурсов в плане серверной части. И последнее, что входит в обязанности архитектора – стратегия и способы развития системы.

После обсуждения, что входит в обязанности архитектора, стоит обсудить, какие архитектурные паттерны существуют. Мы перечислим самые популярные паттерны, на некоторых остановимся, чтобы сильно не растягивать время чтения. И так какие же они бывают: многоуровневая архитектура (Layered), клиент-сервер(Client-server), ведущий-ведомый (master-slave), канал-фильтр (pipe-filter), посредник (broker), одноранговое соединение (peer-to-peer), шины событий(event-bus), модель-представление-контроллер (MVC), доска объявлений (blackboard), интерпретатор (interpreter) [2; 5]. Например, паттерн ведущий, ведомый, предназначен для снижения нагрузки на сервер, его принцип заключается в том, что есть главная база данных и так называемые ведомые, если вы делаете записи или обновляете данные, вы это делаете в главной базе, а она передает информацию об измененной записи в ведомые, но если вам надо

вычитать данные, это делается с ведомых баз данных, таким образам нагрузка уменьшается, а производительность увеличивается.

Перейдем к типам архитектуры, просьба не путать с паттернами. Файл серверная архитектура, такой тип уже устарел и в современном мире ее уже тяжело найти. Клиент-серверная, по сути весь веб основан на этом типе. Трехслойная архитектура и микросервисная, и еще очень много. Но самые важные – это клиент-серверная и микросервисные архитектуры.

А что считать хорошей архитектурой, как все в мире программирования определение хорошей архитектуры не самое однозначное, она звучит так - хорошая архитектура это та, которая вам говорит для чего предназначено ваше приложение. Она также должна быть эффективна, например, калькулятор считает, а не рисует картинки на экране, гибкость при планирование систему мы не можем привязываться к примеру определенной базе данных. Расширяемость системы, тут все просто, добавление новых функций, не нарушая основную структуру. Масштабируемость процесс – возможность декомпозировать сложные большие бизнес процессы на несколько команд, а потом куски собрать в единое без особых проблем. Переиспользование, придерживание паттернов DRY, или избегать по максимум дублирование кода [4]. Сопровождение, хорошо налаженное логирование мониторинга, простое устранение критических ошибок.

Теперь немного подробней о некоторых самых популярных архитектурных паттернов. Первое что мы обсудим, это клиент-серверная архитектура, у нее тип многоуровневая, суть заключается в том, что есть сервер, который каким-то способам получает данные или запросы, как-то их обрабатывает и возвращает ответ. Сервер может хранить данные или просто валидировать. Если говорить, а вебе как правило они общаются по HTTP протоколу. Трехступенчатая архитектура очень похожа на клиент-серверную, только добавляется база данных, которая дает возможность хранить какие-то данные.

Мы обсудили физическое представление, но есть и логическая так

называемая концепция слоев [1]. Чаще всего выделяют 3 основных слоя представления, сюда относится то, что мы показываем пользователю, бизнес-логика, там находятся модели сервиса и третьим слоем являются источники данных, это могут быть веб или базы данных, все что угодно, там же происходит обработка и переработка данных для моделей. Также слои могут увеличивать, дробиться или претерпевать какие-то другие изменения.

Монолитная архитектура представляет собой что-то единое целое: база данных, логика, бизнес процесс, и мы не можем удалить какую-то часть, так как все завязано друг на друге, но есть модульные монолиты, которые можно подключать или отключать. Плюсы монолитов можно отметить: простое развертывание, разработка, весь код лежит рядом, минимум интеграций, производительность, опять же не надо использовать другие сервисы, упрощенное тестирование и отладка, снижение скорости разработки, но это проблема возникает после того, как приложение разрослось до огромных размеров, очень сложное масштабирование, с надежностью тоже могут быть проблемы, из-за одной ошибки все приложение может привести к поломке, недостаточно гибки, развертывание, да это плюс и минус одновременно, большие монолиты могут устанавливаться часами.

Микросервисная архитектура, тут противоположность монолитной архитектуре, каждый сервис выполняет конкретную бизнес задачу. Преимуществами такого подхода является: непрерывное развертывание ускоряет выпуск релизов, независимость развертывания, можно обновлять один модуль, не изменяя всю структуру, гибкость технологий для каждого бизнес процесса, можно выбрать свой язык программирования, высокая надежность так как при ошибке ломается только один модуль, а не все приложение. Из недостатков можно отметить такие как: разрастание процесса разработки, один человек не сможет отследить откуда ошибка и приходиться много коммуницировать с другими командами, расходы на инфраструктуру так как под каждый микросервис требуется свой сервер, отсутствие стандартизации, каждая команда может использовать свои языки и технологии.

После обсуждения двух доминирующих паттернов архитектуры микросервисы и моналит, что же выбрать? Для этого нужна ответь на пару вопросов. Ожидается ли высокая нагрузка на ваше приложение, и вам не понадобится масштабировать, то монолит прекрасно подойдет. Стартап? Монолит прекрасно подойдет, сократит время разработки и затраты на разработку. Частое обновление процессов, тут будет хорошее решение выбрать микросервисную архитектуру. Резко меняется трафик? Тут надо использовать принцип работы холодного старта если использовать монолитное приложение, но микросервис позволит увеличить количество только тех сервисов, на которые наиболее загружены.

Библиографический список:

1. Shklar, L. Web Application Architecture: principles, protocols, and practices // 2010.
2. Пьюривал С., Основы разработки веб-приложений // Бестселлеры O'Reilly. 2015.
3. Роберт М., Чистая архитектура. Искусство разработки программного обеспечения // Архитектор ПО. 2018.
4. Сэм Ньюман, Building Microservices: Designing Fine-Grained Systems // Технолог. 2014.
5. Martin Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems // исследователь. 2017. № 1.