Технологии программирования. Знакомство с архитектурой.

весна 2023

Agenda

- Понятие архитектуры
- Пример простого веб-сервиса
- Интеграция (REST, SOAP, gRPC, GraphQL)
- Монолит и микросервисы
- Документирование архитектуры (C4, 4+1)

Компьютерная архитектура

“Компьютерная архитектура, как и всякая другая архитектура, это искусство определения требований пользователя к структуре, а затем проектирование таким образом, чтобы она как можно полнее соответствовала этим требованиям при заданных экономических и технологических ограничениях”.

(c) Ф. Брукс “Архитектурная философия”

Архитектурное проектирование

“Архитектурное проектирование - это проектирование сверху вниз, определяющее каждую деталь как функцию целого. С этой точки зрения архитектурное проектирование дополняет формальное определение: определить детали по общей структуре можно только в том случае, если метод описания позволяет совершенно свободно опускать детали и говорить о желаемых свойствах системы в целом до начала любой работы по объединению частей сооружения”.

(c) Г. Земанек

Принципы хорошей архитектуры по Блаау

Согласованность. Хорошая архитектура согласована, то есть частичное знание системы позволяет предсказать остальное.

Ортогональность. Этот принцип требует, чтобы функции были независимы друг от друга и специфицированы по отдельности.

Соответственность. Согласно этому принципу следует включать в архитектуру только те функции, которые соответствуют существенным требованиям к системе, другими словами, в хорошей архитектуре нет ненужных функций.

Принципы хорошей архитектуры по Блаау

Экономность. Никакая функция в описании архитектуры не должна в том или ином виде дублировать другую.

Прозрачность. Функции, найденные в процессе исполнения, должны быть известны пользователю.

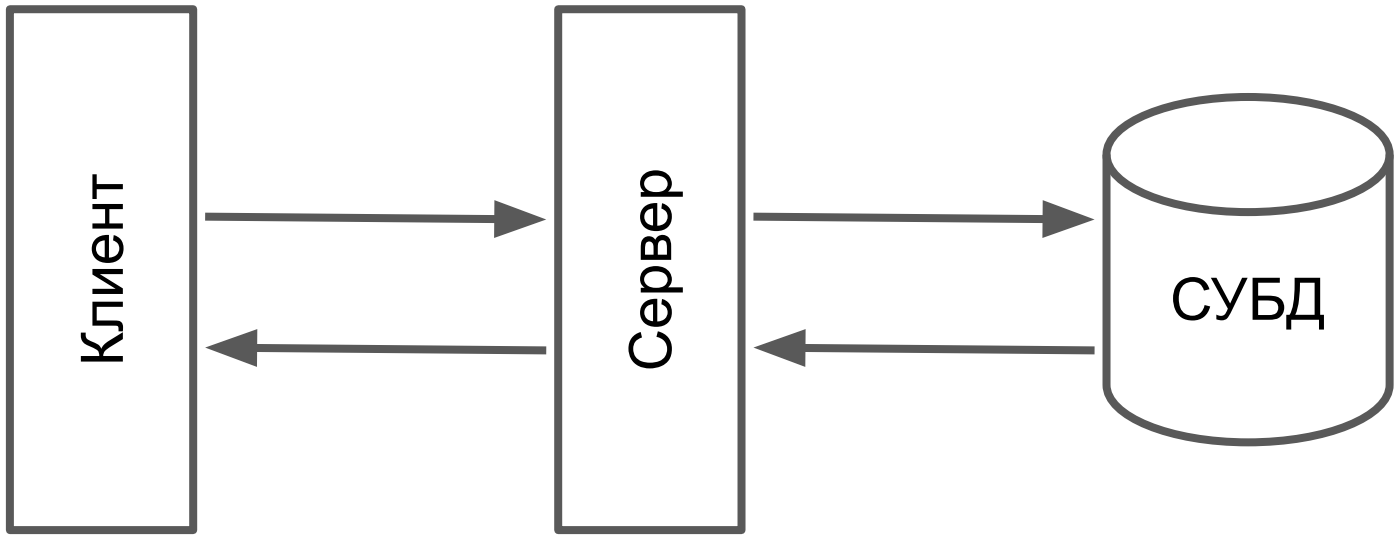
Общность. Если функция должна быть введена, её следует вводить в таком виде, чтобы она отвечала как можно большему числу назначений.

Принципы хорошей архитектуры по Блаау

Открытость. Пользователю должно быть позволено использовать функцию иначе, чем это предполагалось при проектировании.

Полнота. Введённые функции должны с учётом экономических и технологических ограничений как можно полнее соответствовать требованиям и пожеланиям пользователя.

Трёхзвенная архитектура “клиент-сервер”



Посмотрим на пример простого web-приложения

СУБД = PostgreSQL

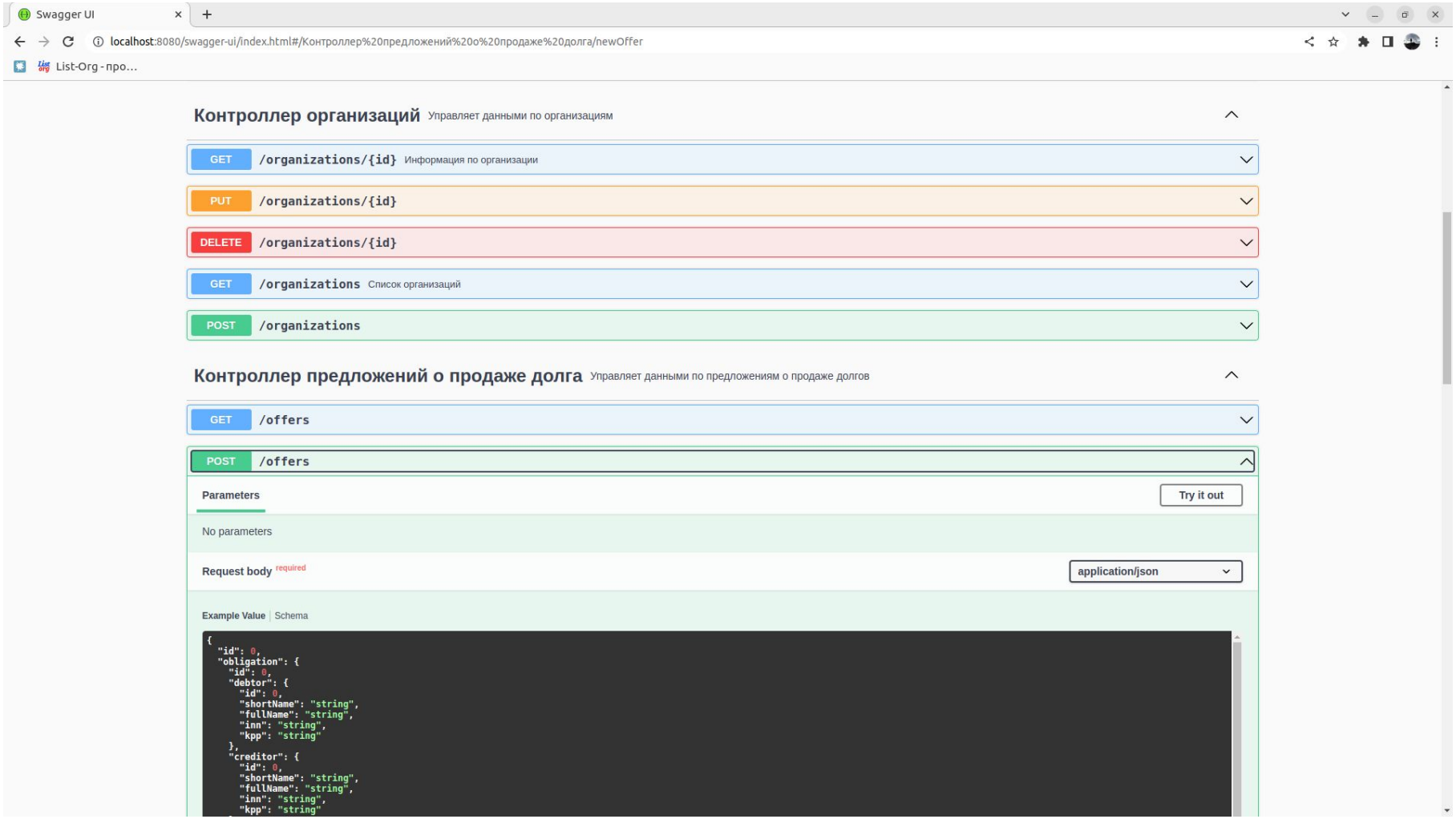
https://www.postgresql.org/docs/current/protocol.html

+ SQL

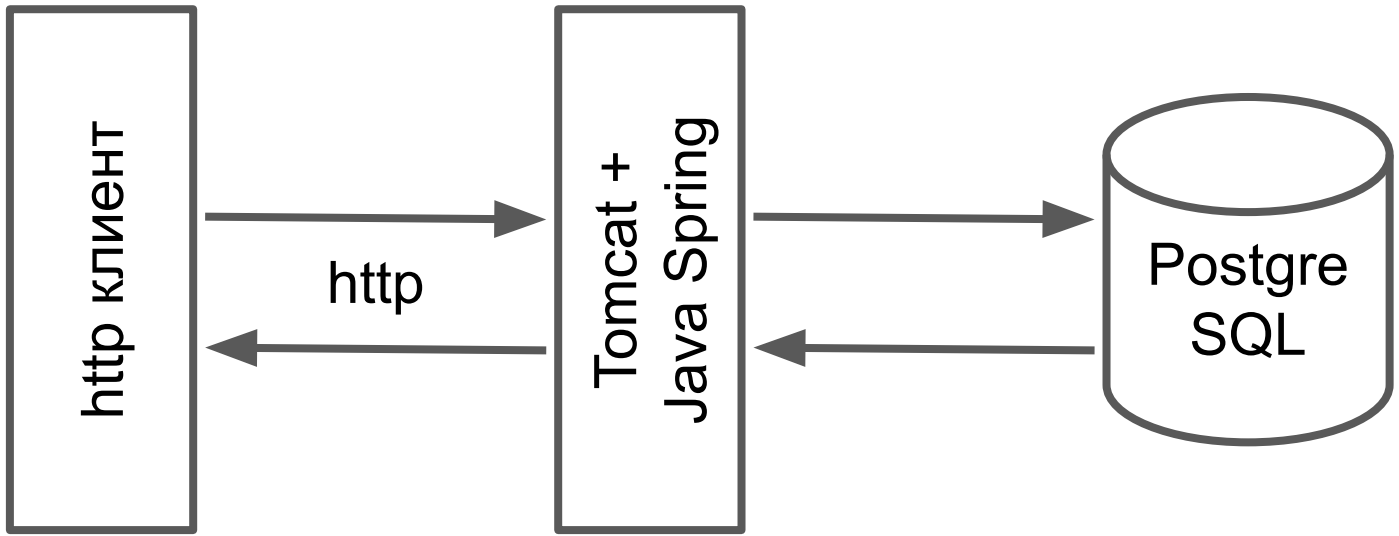
Backend = Java + Spring boot

http + REST

Open API Specification (Swagger)


Swagger UI
localhost:8080/swagger-ui/index.html#/Контроллер%20предложений%20о%20продаже%20долга/newOffer
List-Org - про...
Контроллер организаций
Управляет данными по организациям
GET /organizations/{id} Информация по организации
PUT /organizations/{id}
DELETE /organizations/{id}
GET /organizations Список организаций
POST /organizations
Контроллер предложений о продаже долга
Управляет данными по предложениям о продаже долгов
GET /offers
POST /offers
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"id": 0,
"obligation": {
"id": 0,
"debtor": {
"id": 0,
"shortName": "string",
"fullName": "string",
"inn": "string",
"kpp": "string"
},
"creditor": {
"id": 0,
"shortName": "string",
"fullName": "string",
"inn": "string",
"kpp": "string"

Трёхзвенная архитектура “клиент-сервер”



Интеграция

- REST (Representational State Transfer)
- GraphQL (query language for APIs)
- RPC (Remote Procedure Call)
- SOAP (Simple Object Access Protocol)
- gRPC

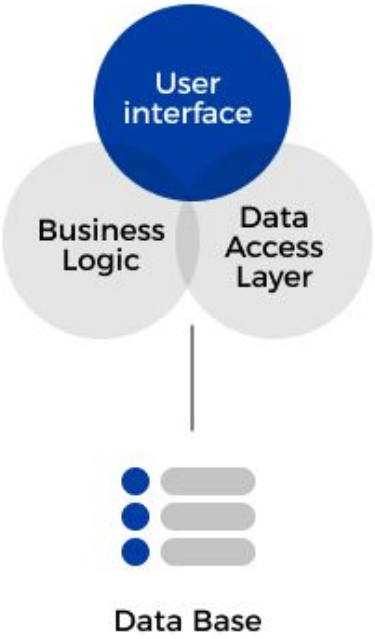
thriftly.io protocol comparison	First released	Formatting type	Key strength
SOAP	Late 1990s	XML	Widely used and established
REST	2000	JSON, XML, and others	Flexible data formatting
JSON-RPC	mid-2000s	JSON	Simplicity of implementation
gRPC	2015	Protocol buffers by default; can be used with JSON & others also	Ability to define any type of function
GraphQL	2015	JSON	Flexible data structuring
Thrift	2007	JSON or Binary	Adaptable to many use cases

Монолитная и микросервисная архитектура

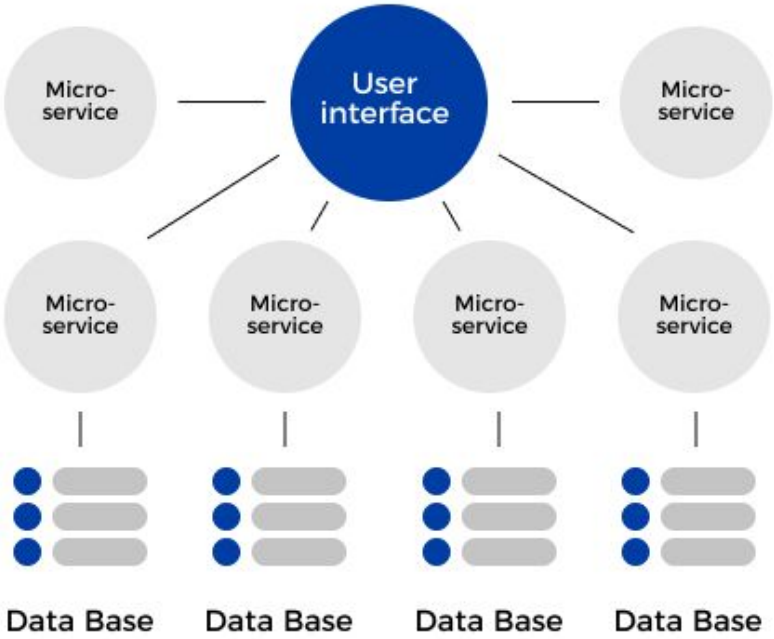
Монолитное приложение - единое общее приложение, работающее автономно, реализующее все функции.

Микросервисное приложение - система небольших независимых служб / приложений, каждая из которых реализует одну функцию.

MONOLITHIC ARCHITECTURE



MICROSERVICE ARCHITECTURE



Монолитная архитектура	Микросервисная архитектура
Цельное большое приложение	“Оркестр” микро приложений
Единый язык программирования	У каждого сервиса может быть свой язык программирования
Сложный код	Простой код
Сложно масштабировать	Масштабируемость
Быстрый старт	Расходы на интеграцию

Монолит и миросервисы. Что почитать?

Роберт Мартин (дядюшка Боб)

“Чистая архитектура. Искусство разработки программного обеспечения”

Microservice Architecture

https://microservices.io/

Документирование архитектуры

UML

4+1 Architectural view model

Architecture Decision Records

The C4 Model

Dependency diagrams

Application Map

Модель представления архитектуры “4 + 1”

Architectural Blueprints—The “4+1” View Model of Software Architecture (c) Philippe Kruchten
https://www.win.tue.nl/~wstomv/edu/2ip30/references/Kruchten4+1.pdf

Этот способ визуализации архитектуры программного приложения основан на 5 представлениях / ракурсах приложения, сообщающих нам, какие диаграммы можно использовать для документирования каждого из этих представлений.

Conceptual / Logical

Physical / Operational

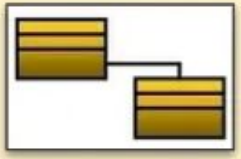
Functional

Non-functional

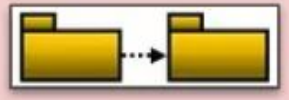
Logical / Structural view

Perspective: Analysts, Designers
Stage: Requirement analysis
Focus: Object oriented decomposition
Concerns: Functionality
Artefacts:

- Class diagram
- Object diagram
- Composite structure diagram



Implementation / Developer view



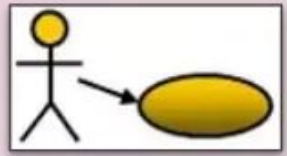
Perspective: Developers, Proj. mngs.
Stage: Design
Focus: Subsystem decomposition
Concerns: Software management
Artefacts:

- Component diagram
- Package diagram

Use Case/Scenario view

Perspective: End users
Stage: Putting it alltogether
Concerns: Understandability, usability
Focus: Feature decomposition
Artefacts:

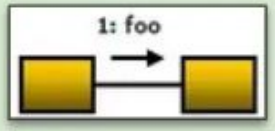
- Use-case diagram
- User stories



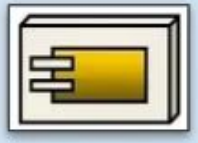
Process / Behaviour view

Perspective: System Integrators
Stage: Design
Focus: Process decomposition
Concerns: Performance, scalability, throughput
Artefacts:

- Sequence diagram
- Communication diagram
- Activity diagram
- State (machine) diagram
- Interaction overview diagram
- Timing diagram



Deployment / Physical view



Perspective: System Engineers
Stage: Design
Focus: Map software to hardware
Concerns: System topology, delivery, installation, communication
Artefacts:

- Deployment diagram
- Network topology (not UML)

4+1

1. Логическое / Структурное устройство (Logical/Structural view)

Отображает функциональность, предоставляемую системой, и то, как код разработан для обеспечения такой функциональности;

2. Реализация (Implementation/Developer view)

Отображает статическую организацию кода, компонентов, модулей и пакетов;

4+1

3. Поведение (Process/Behaviour view)

Фокусируется на поведении системы во время выполнения, на том, как системные процессы взаимодействуют, параллелизме, синхронизации, производительности и т. д.;

4. Физическое устройство (Deployment/Physical view)

Иллюстрирует физическую организацию приложения с точки зрения того, «какой код работает на каком оборудовании»;

4+1

5. Сценарии использования (Use Case/Scenario view)

Архитектура в целом объясняется с помощью нескольких вариантов использования, которые представляют собой просто последовательность действий. Часть архитектуры развивается из таких вариантов использования.

Модель C4

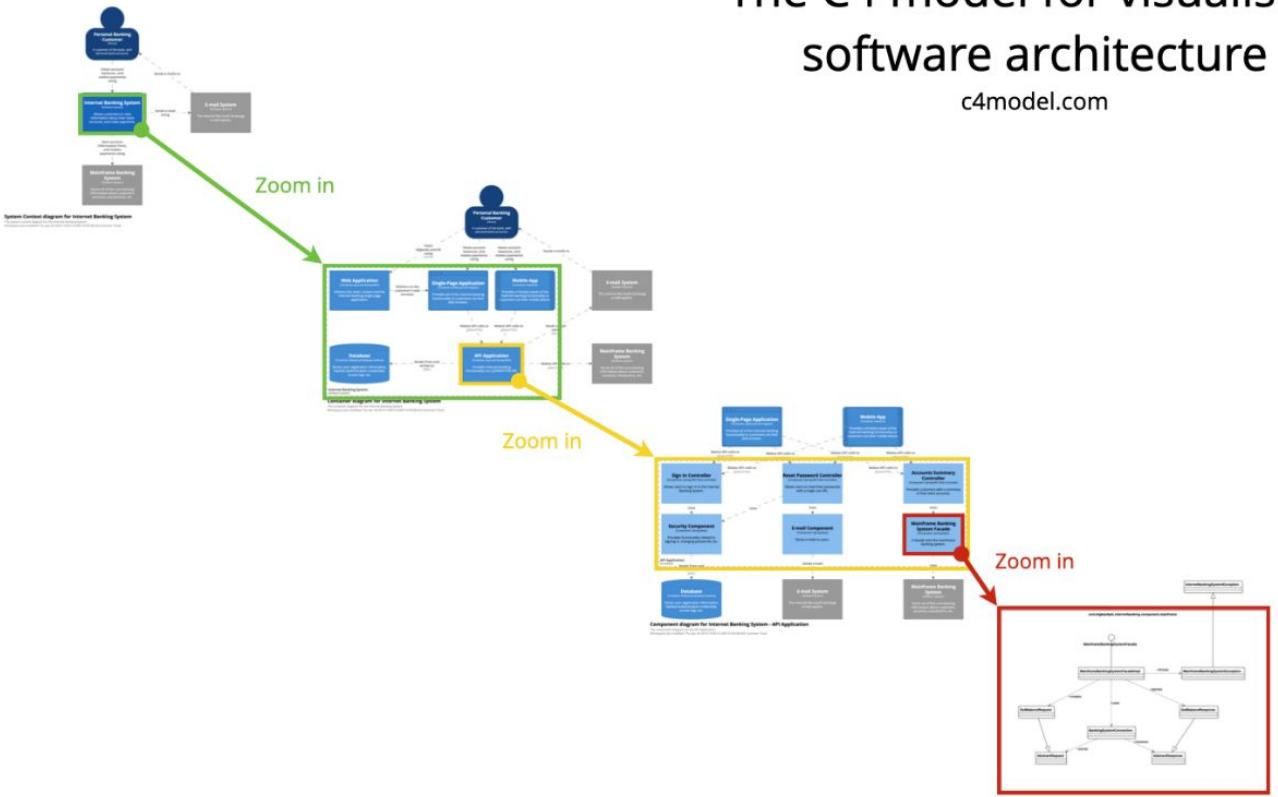
The C4 Model for Software Architecture (c) Simon Brown
https://www.infoq.com/articles/C4-architecture-model/

Context Container Component Code model

Идея состоит в том, чтобы использовать 4 различных уровня масштабирования для документирования архитектуры программного обеспечения

The C4 model for visualising software architecture

c4model.com



Level 1
Context

Level 2
Containers

Level 3
Components

Level 4
Code

Модель C4

Уровень 1: System Context diagram

Уровень 2: Container diagram

Уровень 3: Component diagram

Уровень 4: Code diagram

Уровень 1: System Context diagram

Самая высокоуровневая диаграмма.

Главная цель - описать контекст, в котором работает система.

Здесь описываются функции, которые приносят пользу пользователям. Уровень включает в себя и сам продукт, и другие взаимосвязанные с ними системы. В центре внимания не технологии, протоколы и другие низкоуровневые детали, а пользователи, роли, акторы, программные системы.

Уровень 2: Container diagram

Масштаб высокоуровневых строительных блоков.

Диаграмма показывает общую форму архитектуры, распределение функций и обязанностей.

Контейнер – это отдельно развертываемый объект или среда выполнения, которая зачастую работает в собственном пространстве процессов. Связь между контейнерами обычно принимает форму межпроцессного взаимодействия.

Это не про docker!

Уровни 3 и 4

Уровень 3: Component diagram

Показывает устройство контейнера.

Уровень 4: Code diagram

Описывает структуру кода внутри компонента.

Скорость устаревания диаграмм C4

Из-за иерархической природы каждая диаграмма будет изменяться с разной скоростью.

Архитектура

Продолжение следует?