CSC301

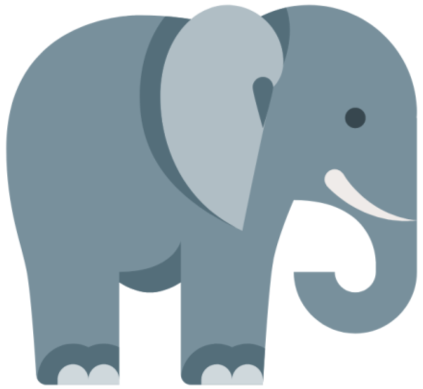
How Do Your Build Your Software?

- How many of you have inherited code from another CSC301 team?
- How easy is it to work with? Why?

Agenda

- Introduction to software architecture
- Introduction to APIs
    - Use
        - Examples
        - Postman
    - Design
        - HTTP
    - Multi-channel
    - Approaches to Integration

HOW DO YOU EAT AN ELEPHANT?

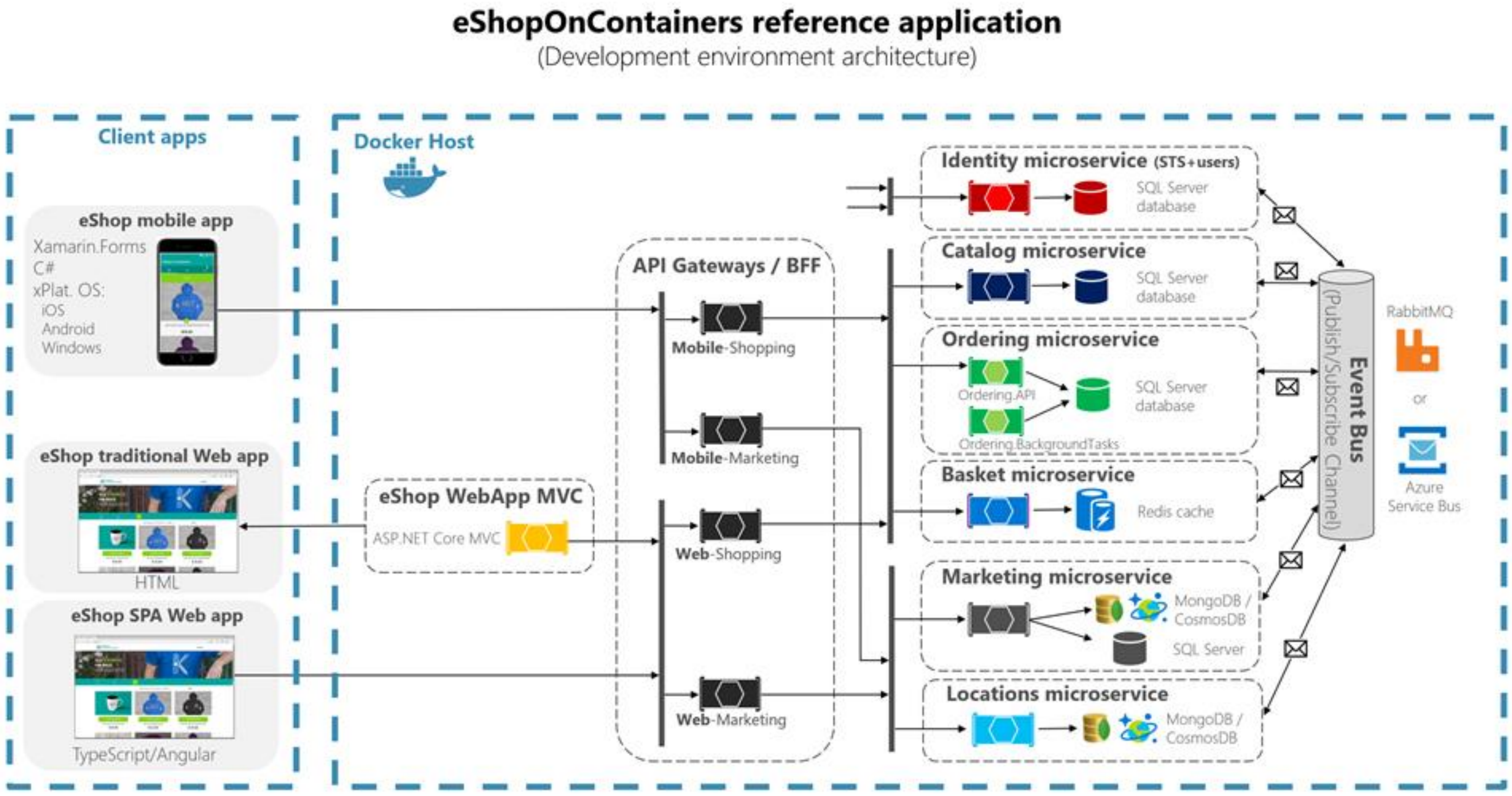


ONE BITE AT A TIME.

What is Software Architecture?

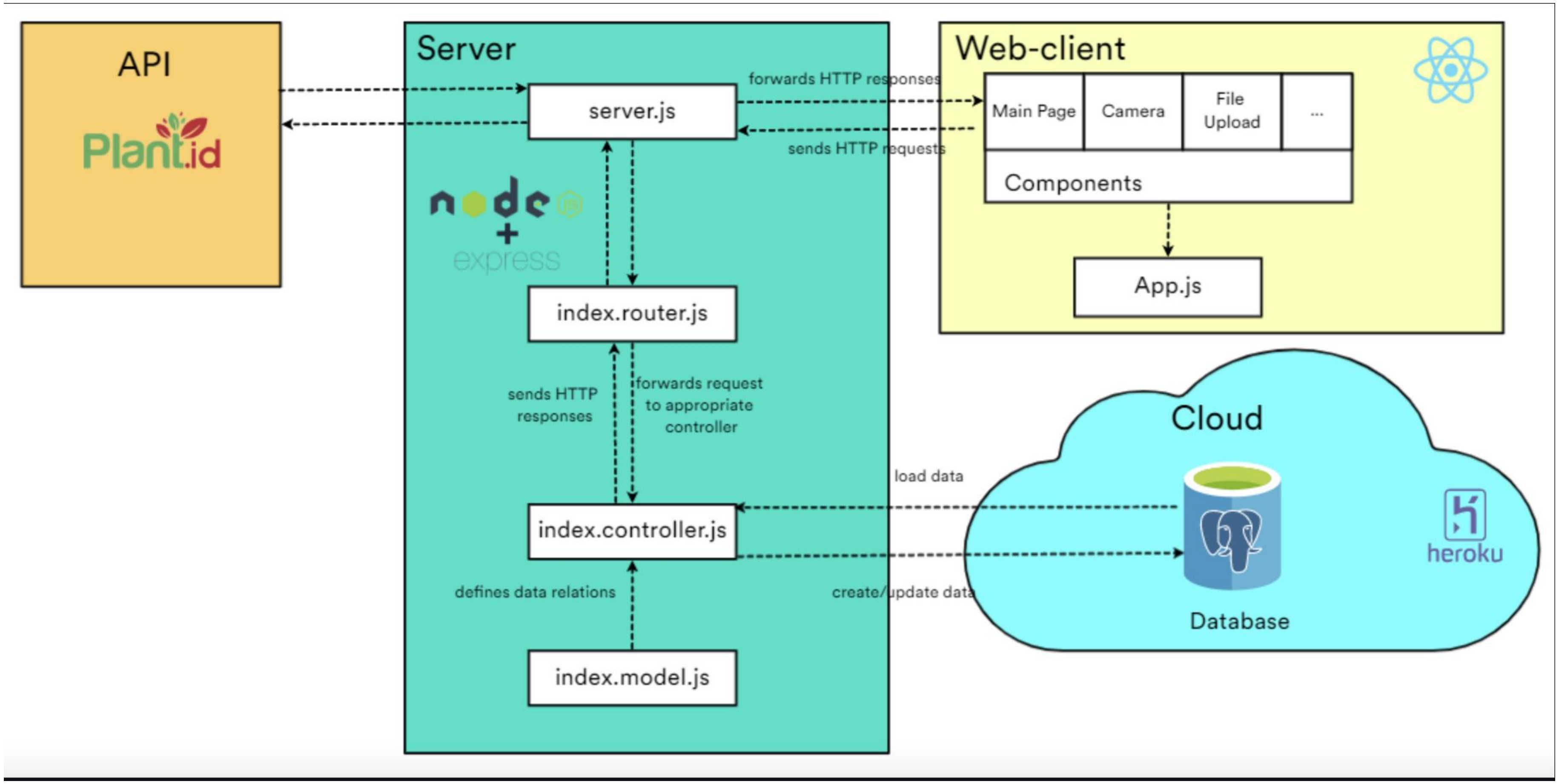
- How do you organize your software?
  - Behaviour
  - Structure
- 10 common software architecture patterns
- Most helpful few to start with

Why Web APIs?



eShopOnContainers reference application
(Development environment architecture)
Client apps
eShop mobile app
Xamarin.Forms
C#
xPlat. OS:
iOS
Android
Windows
eShop traditional Web app
HTML
eShop SPA Web app
TypeScript/Angular
Docker Host
eShop WebApp MVC
ASP.NET Core MVC
API Gateways / BFF
Mobile-Shopping
Mobile-Marketing
Web-Shopping
Web-Marketing
Identity microservice (STS+users)
SQL Server database
Catalog microservice
SQL Server database
Ordering microservice
Ordering.API
Ordering.BackgroundTasks
SQL Server database
Basket microservice
Redis cache
Marketing microservice
MongoDB / CosmosDB
SQL Server
Locations microservice
MongoDB / CosmosDB
Event Bus
(Publish/Subscribe Channel)
RabbitMQ
or
Azure Service Bus

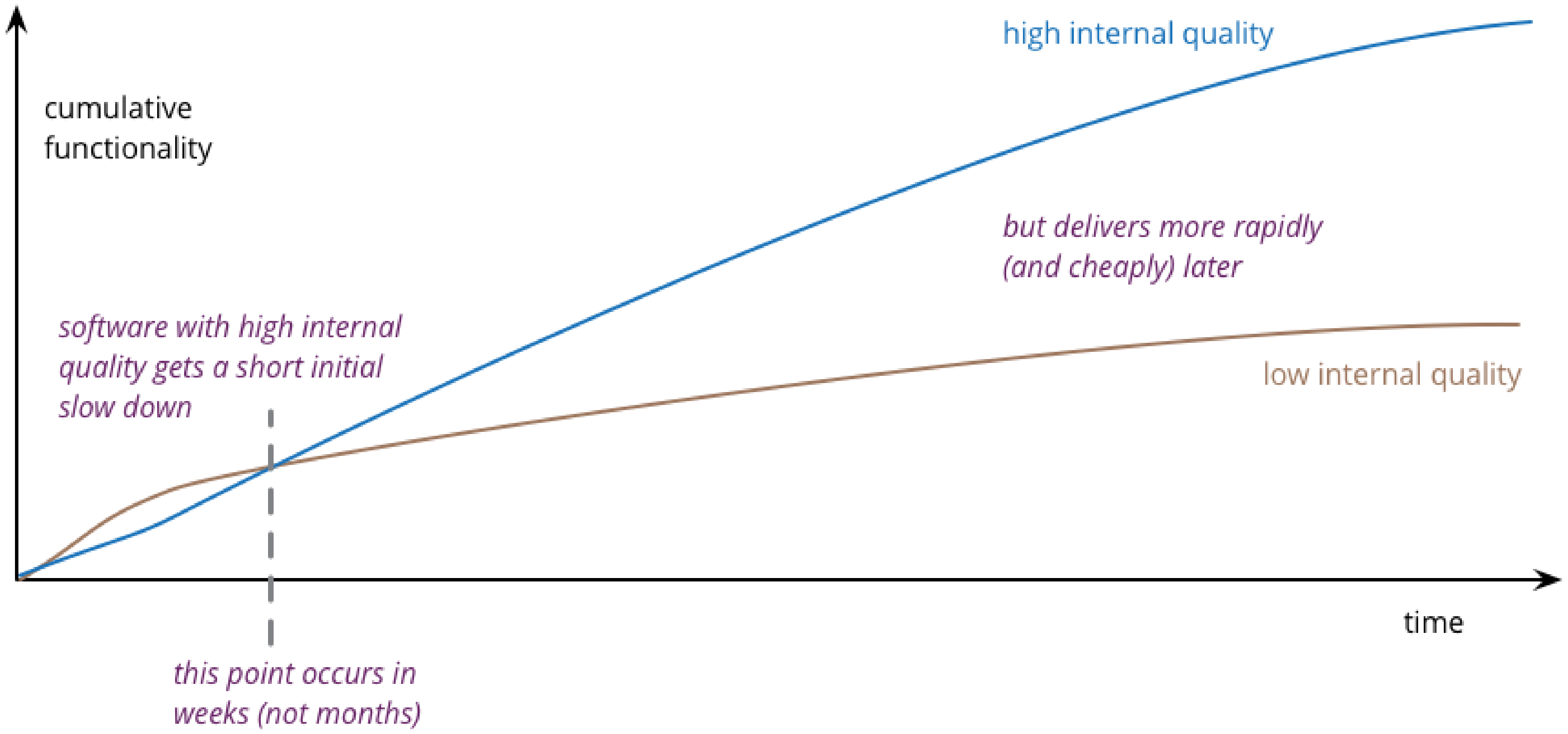
Software architecture



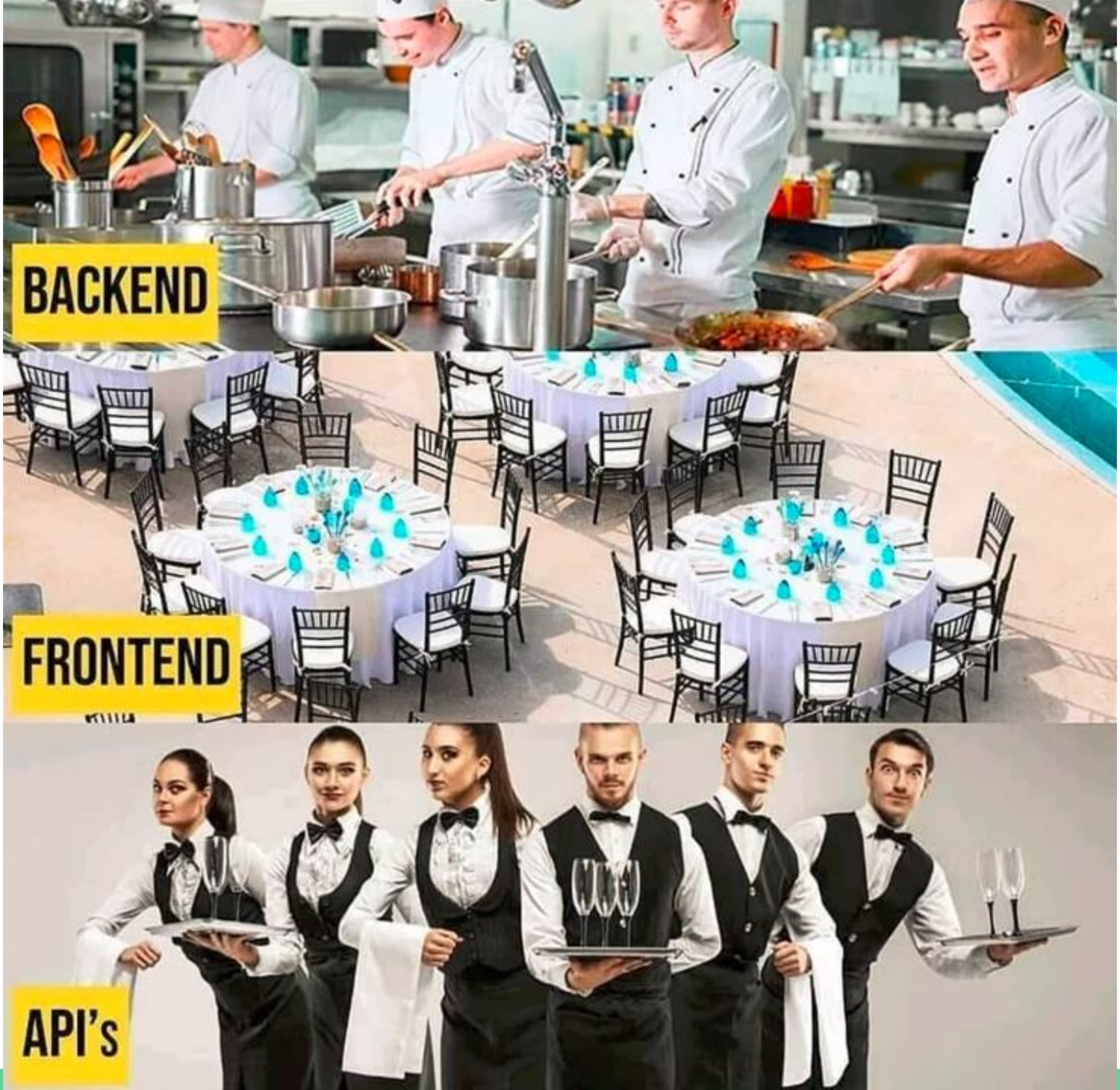
Design Patterns

- Catalog of Design Patterns
- Short video on 10 design patterns

Value of Architecture



How do software modules talk to each other?


BACKEND
FRONTEND
API's

Exercise 1

You are building a system for a Canadian business that operates in Canada and the United States with income and costs in USD and CAD. You are required to calculate all the revenues in CAD to report to the CRA based on the rate defined by the Bank of Canada.

What data do you need? How do you get it?

Let’s do an exercise

Other than getting data, what are the other possible actions we may want to do?

Activity 2- Project Component Design

Take 10 minutes to think of different data and components you need for your project

1. Any external systems providing you services or data?
2. Do you need to provide services or data to an external system?
3. How about communications inside the modules of your software (e.g., frontend -> backend -> database)?

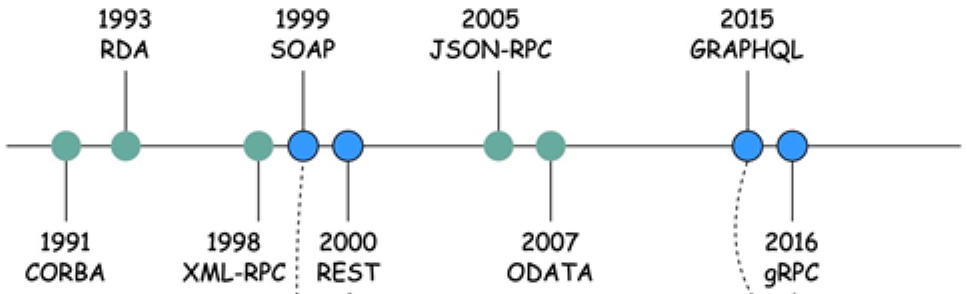
   What are the request and response format for each one? How about errors and exceptions?

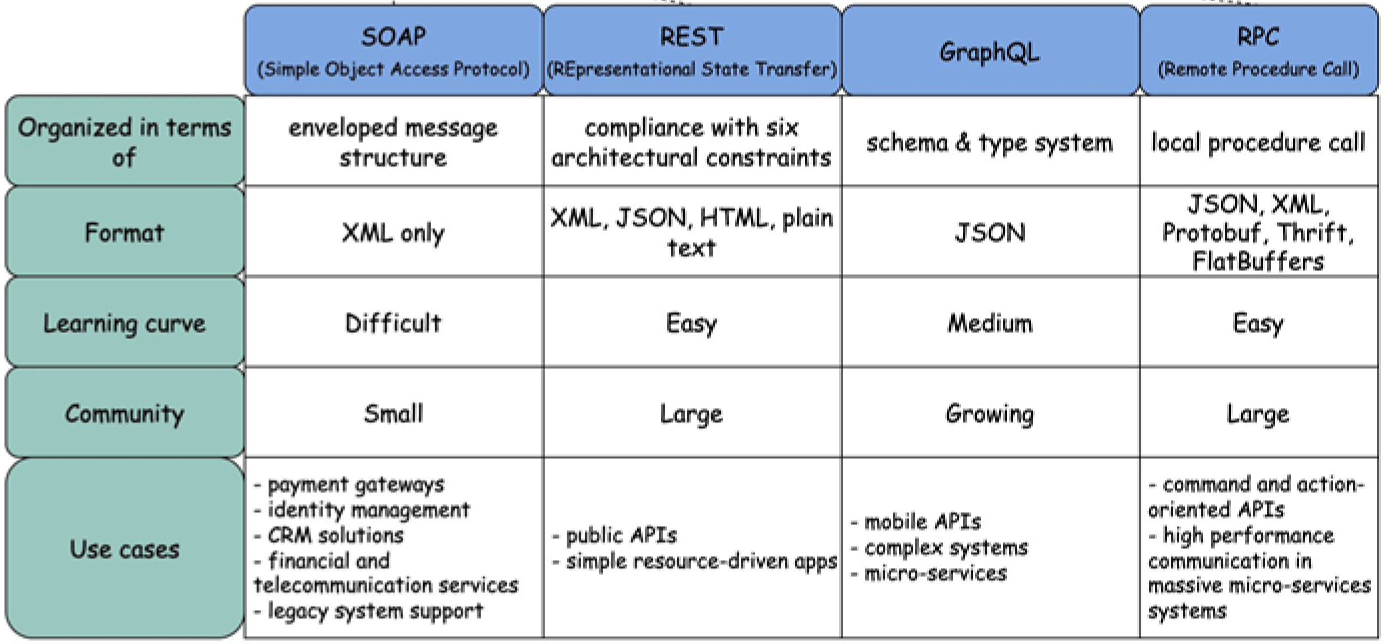
API Design

APIs to the rescue!

API Architectural Styles Comparison

Source: altexsoft





	SOAP (Simple Object Access Protocol)	REST (REpresentational State Transfer)	GraphQL	RPC (Remote Procedure Call)
Organized in terms of	enveloped message structure	compliance with six architectural constraints	schema & type system	local procedure call
Format	XML only	XML, JSON, HTML, plain text	JSON	JSON, XML, Protobuf, Thrift, FlatBuffers
Learning curve	Difficult	Easy	Medium	Easy
Community	Small	Large	Growing	Large
Use cases	- payment gateways - identity management - CRM solutions - financial and telecommunication services - legacy system support	- public APIs - simple resource-driven apps	- mobile APIs - complex systems - micro-services	- command and action-oriented APIs - high performance communication in massive micro-services systems

HTTP Requests

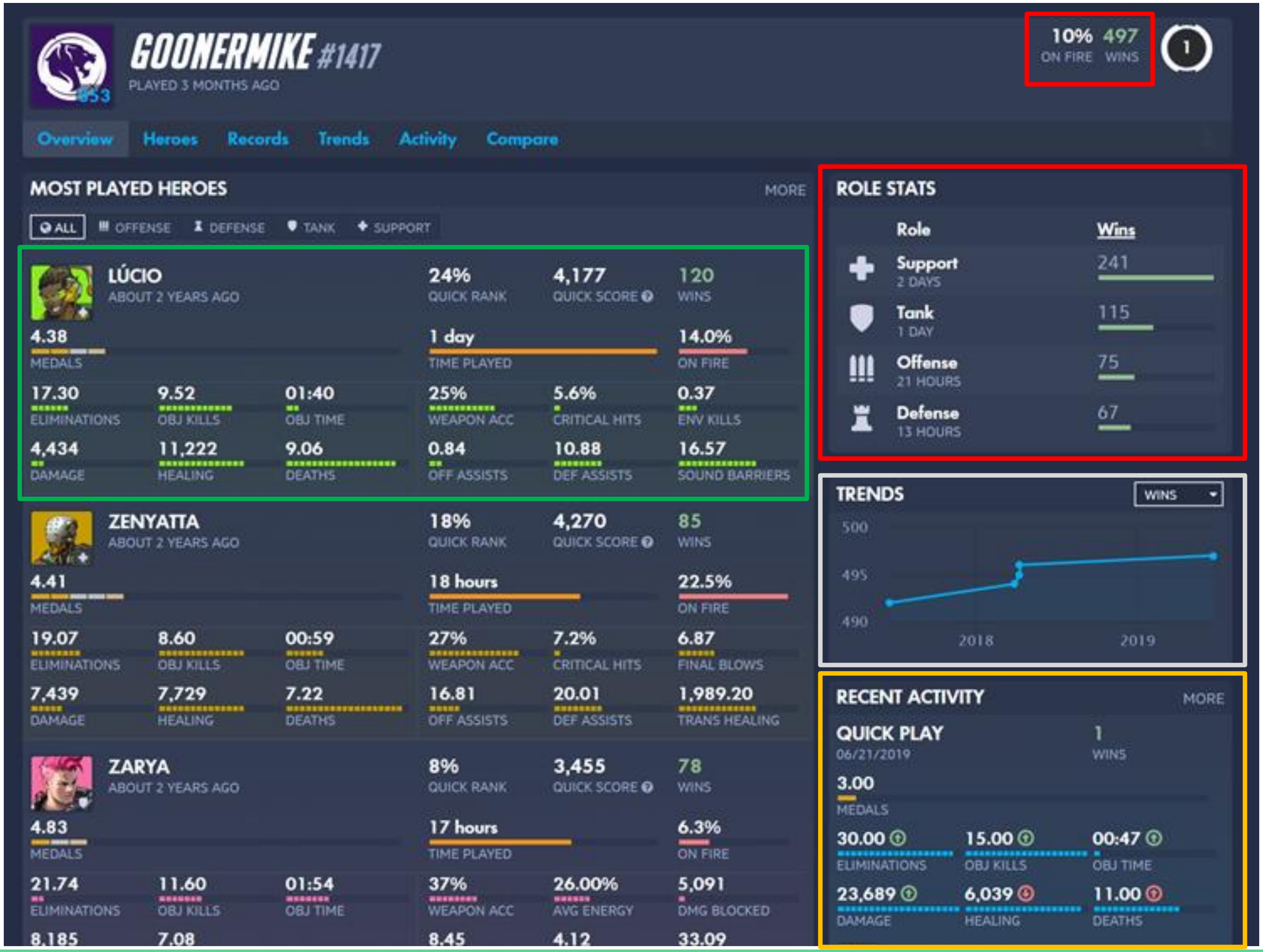
- **Self Descriptive Messages**
  - Messages must provide guidance about their contents (see previous)
  - Messages must use a HTTP method to convey processing semantics
  - Messages are sent to a URL
- GET – get information
- DELETE – delete resource
- POST – create new subordinate resource, “process this”
- PUT – replace resource
- PATCH – update *partial* state of resource
- OPTIONS – get list of supported HTTP methods

HTTP Status Codes You Need To Know

Code	Description
200	Everything went as expected
201	Use to indicate that a resource was *created*
301	Redirect. Use this when you change URLs to a resource
400	Invalid syntax
401	Unauthorized – request is missing required authn headers
403	Forbidden – valid & authenticated, but missing permissions
404	Whatever you're looking for is not here
500	Something went horribly wrong…

Another REST API Example (by Michael Davison)

Overwatch Player Stats



GOONERMIKE #1417
PLAYED 3 MONTHS AGO

10% ON FIRE | 497 WINS

Overview | Heroes | Records | Trends | Activity | Compare

MOST PLAYED HEROES — MORE

ALL | OFFENSE | DEFENSE | TANK | SUPPORT

LÚCIO
ABOUT 2 YEARS AGO

24% QUICK RANK	4,177 QUICK SCORE	120 WINS			
4.38 MEDALS	1 day TIME PLAYED	14.0% ON FIRE			
17.30 ELIMINATIONS	9.52 OBJ KILLS	01:40 OBJ TIME	25% WEAPON ACC	5.6% CRITICAL HITS	0.37 ENV KILLS
4,434 DAMAGE	11,222 HEALING	9.06 DEATHS	0.84 OFF ASSISTS	10.88 DEF ASSISTS	16.57 SOUND BARRIERS

ZENYATTA
ABOUT 2 YEARS AGO

18% QUICK RANK	4,270 QUICK SCORE	85 WINS			
4.41 MEDALS	18 hours TIME PLAYED	22.5% ON FIRE			
19.07 ELIMINATIONS	8.60 OBJ KILLS	00:59 OBJ TIME	27% WEAPON ACC	7.2% CRITICAL HITS	6.87 FINAL BLOWS
7,439 DAMAGE	7,729 HEALING	7.22 DEATHS	16.81 OFF ASSISTS	20.01 DEF ASSISTS	1,989.20 TRANS HEALING

ZARYA
ABOUT 2 YEARS AGO

8% QUICK RANK	3,455 QUICK SCORE	78 WINS			
4.83 MEDALS	17 hours TIME PLAYED	6.3% ON FIRE			
21.74 ELIMINATIONS	11.60 OBJ KILLS	01:54 OBJ TIME	37% WEAPON ACC	26.00% AVG ENERGY	5,091 DMG BLOCKED
8,185	7.08		8.45	4.12	33.09

ROLE STATS

Role	Wins
Support 2 DAYS	241
Tank 1 DAY	115
Offense 21 HOURS	75
Defense 13 HOURS	67

TRENDS — WINS

RECENT ACTIVITY — MORE

QUICK PLAY 06/21/2019 — 1 WINS

3.00 MEDALS		
30.00 ELIMINATIONS	15.00 OBJ KILLS	00:47 OBJ TIME
23,689 DAMAGE	6,039 HEALING	11.00 DEATHS

Overwatch Player Stats

- Player
    - Player performance snapshot
    - Contains stats for
        - Most played avatars
        - Role (Attack, Support, Tank)
        - Performance Trends
        - Recent history
- Matches
    - Global list of matches played
    - Captures detailed match telemetry

Player and Match data are modeled as two separate services or APIs in this example

Why? I believe they are separate domains due to their vastly different operating characteristics

Overwatch Stats API

1. Resources

- Player
- Player’s List of Most Played Heroes
- Player Trends
  - Wins, Losses, etc.
- Recent Player Activity
- (Player) List of Matches Completed
- List of Matches (Global)
- Match
- Match Events

Overwatch Stats API

1. Resources with URLs

- Player - **/api/players/GoonerMike#1417**
- Player's List of Most Played Heroes - - **/api/players/GoonerMike#1417/{mode}/mostplayed?count={}&type={}**
- Player Trends - **/api/players/GoonerMike#1417/{mode}/trends?restype={}**
    - Wins, Losses, etc.
- Recent Player Activity - **/api/players/GoonerMike#1417/recent?mode={}**
- List of Matches Completed - **/api/players/matches**
- List of Matches - **/api/matches**
- Match - **/api/matches/match/{matchid}**
- Match Events – **/api/matches/match/{matchid}/events**

Overwatch Stats API

2. HTTP Verbs

- Player - **GET**
- Player's List of Most Played Heroes - - **GET**
- Trends - **GET**
  - Wins, Losses, etc.
- Activity – **GET**
- List of Matches Completed – **POST**
- Matches – **GET, POST**
- Match – **GET**
- Match Events – **GET, POST**

3. How do we build them?

You Have Options

- Postman Learn By API (basic training)
- AWS API Gateway Tutorial or AWS DevOps guide
- Getting Started with Heroku
- Stripe API is a comprehensive example (excellent documentation)

Activity 3

1. What resources do you need for your project?
2. What actions do you need to support?
    a. What will the request look like?
    b. What parameters can be sent?
    c. What will the response look like?

Let’s do it here

API Documentation

- Swagger is probably the most popular documentation approach