

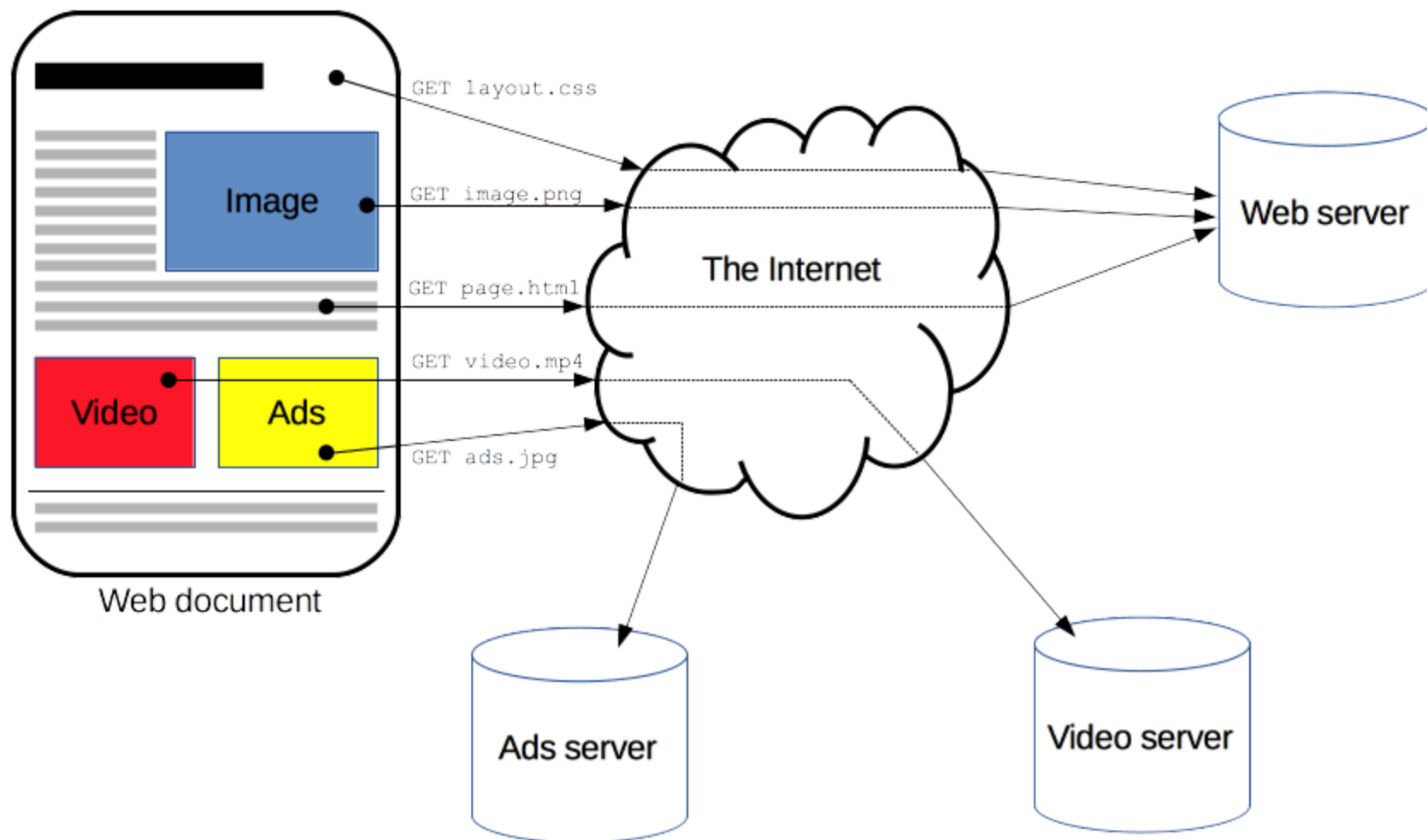
Programowanie Aplikacji Internetowych

API / Komunikacja między serwisami

Plan na dziś

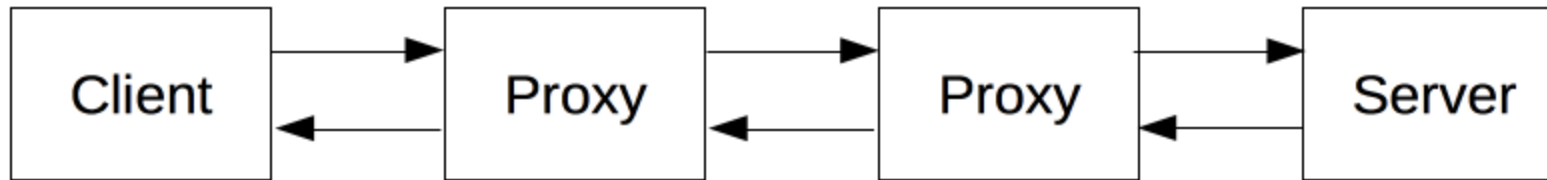
- http
- RPC
- REST
- GraphQL

HTTP



HTTP

Cała infrastruktura przystosowana do pracy z http:



HTTP

Demo:

```
curl -I www.google.com
```

```
curl -I -L google.com
```

HTTP - methods

Methods:

- GET
- POST
- PUT
- DELETE

HTTP - status code

Status code:

- 5xx: 500, 502
- 4xx: 404, 400, 401
- 3xx: 301, 302
- 2xx: 200, 201, 02

Warto wiedzieć, gdzie jest błąd.

WebSockets

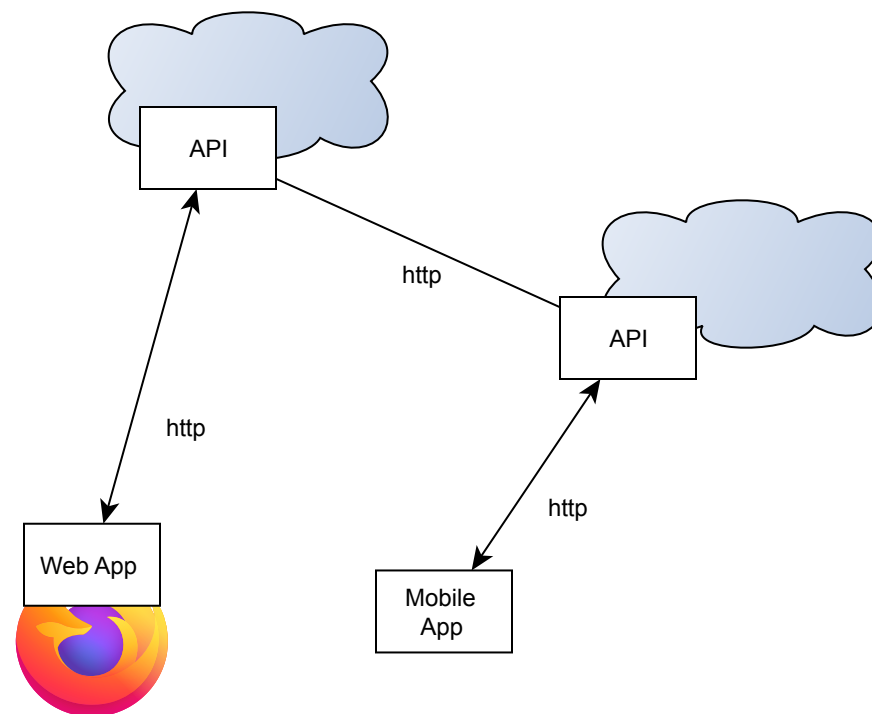
- dwustronnej szybkiej komunikacji
- Alternatywa dla *long polling*

Więcej później o [websocketach i socketio](#) później.

A co z serwisami?

- +/- Wiemy jak działają przeglądarki
- co z serwisami?

A co z serwisami?



Protokoły

Najpopularniejsze:

- (web) RPC
- REST API
- GraphQL

(web) RPC

- RPC (remote procedure call)
- po prostu wywołanie zewnętrznej funkcji

(web) RPC

Przykłady / o czym należy pamiętać wywołując zewnętrzny serwis:

- [example_py_call_rest_api](#)
- [example_js_call_rest_api](#)
- [example_js_call_rest_api](#)

REST API

- Inspiracja: jak działa komunikacja między przeglądką, a serwerem,
- Istniejąca infrastruktura,
- Najbardziej popularne podejście.

Przykład - Github

Przyjrzyjmy się bliżej API:

- [commits](#);
- [prs](#);
- [authentication](#).

Przykład - Github

Co warto sprawdzić:

- [verbs](#);
- [errors](#);
- [rate limiting](#).

Przykład - Github

Często mamy już dostępne biblioteki:

- oficjalne - github.com/octokit
- nieoficjalne - [google/go-github](https://www.google.com/search?q=go-github)

Jeśli budujesz API

Warto się wzorować na:

- shopify API - [przykład](#),
- twillio - <https://www.twilio.com/docs/usage/api>,
- ably - <https://ably.com/docs/api/rest-api#publish>,
- ○ pragmatyzm;
- ○ [dobre praktyki](#).

REST API

Zasady:

- Logical organization of resources
- Logical nesting
- Stateless
- cacheable data
- większości JSON-based

REST API

Projekty / standardy:

- [OpenAPI](#) - industrial standard;
- [json API](#) - popularniejsze z bardziej ustrukturyzowanego podejścia.

REST API

Wiele godzin rozstało przepalone na dyskusjach co to jest REST API i czy dane API jest rzeczywiście REST...

Ograniczenia REST API

- [wszystko albo nic](#);
- kilka(naście) requestów, żeby zebrać dane;
- a potem składanie.

Ograniczenia REST API

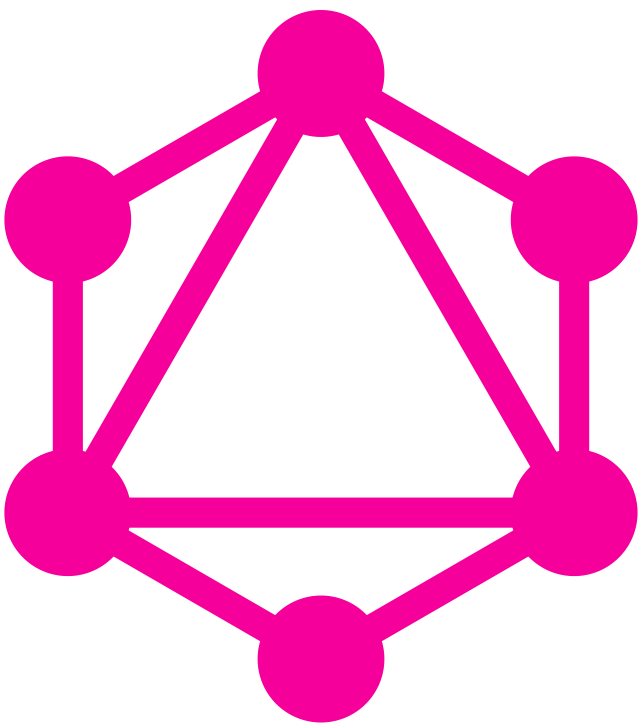
- za każdym razem backend musi pisać API dla frontendu (backed-for-frontend);
- czasami gonienie za nieuchwytnym celem.

Plusy REST API

- Cacheable;
- łatwe do zrozumienia;
- z OpenAPI, duża ilość narzędzi out-of-the-box, np., [browsable API](#);
- bez niespodzianek dla backendu.

Warto wiedzieć

- [OData](#) - less popular, aplikacje enterprise - np., Microsoft czy SAP;
- [activitypub](#) - protokół komunikacji [mastodona](#).



GraphQL

GraphQL

Value proposition:

- informacja o typie;
- frontend zadaje zapytanie;
- może obejmować kilka encji;
- może również wybrać które atrybuty nas interesują;
- aliasy, zmiany nazw;
- standard.

GraphQL

Value proposition:

- frontend/client może powiedzieć czego chce.

GraphQL

Minusy:

- problem z cache-owaniem;
- ciężar po stronie backendu;

Demo

Za [GraphQL docs](#), definicja serwisu GraphQL:

```
type Query {  
  me: User  
}  
  
type User {  
  id: ID  
  name: String  
}
```

Demo

Za [GraphQL docs](#), zapytanie klienta:

```
{  
  me {  
    name  
  }  
}
```

Demo

Za [GraphQL docs](#), możliwa odpowiedź:

```
{  
  "me": {  
    "name": "Natalia"  
  }  
}
```


Demo

- [insomnia](#)
-

GraphQL

- <https://graphql.org/>
- <https://graphql.org/learn/>

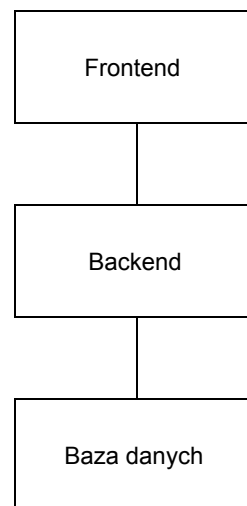
Narzędzia

- [insomnia](#) lub [postman](#)
- [curl](#)
- [jq](#)
- biblioteki [jmespath](#)

Warto wiedzieć

- gRPC

Architektura



Zauważ

- JS/TS w przeglądarce to też aplikacja,
- Docelowo - JS/TS powinna komunikować się przez API.

Dziękuję za uwagę

Backup slides

3-tier architecture

Jak hostować?

- PaaS: [vercel](#), [netify](#), [heroku](#);
- CaaS (AWS EKS, GCP) - container-as-a-service
- XaaS (AWS, GCP):
 - IaaS