kapsch >>>

JavaCro'16

Zahtjevno-Orijentirana Web Arhitektura

Vjeran Marčinko
Kapsch Carrier Com

kapsch

JavaCro'16

Eksplozija klijentskih uređaja

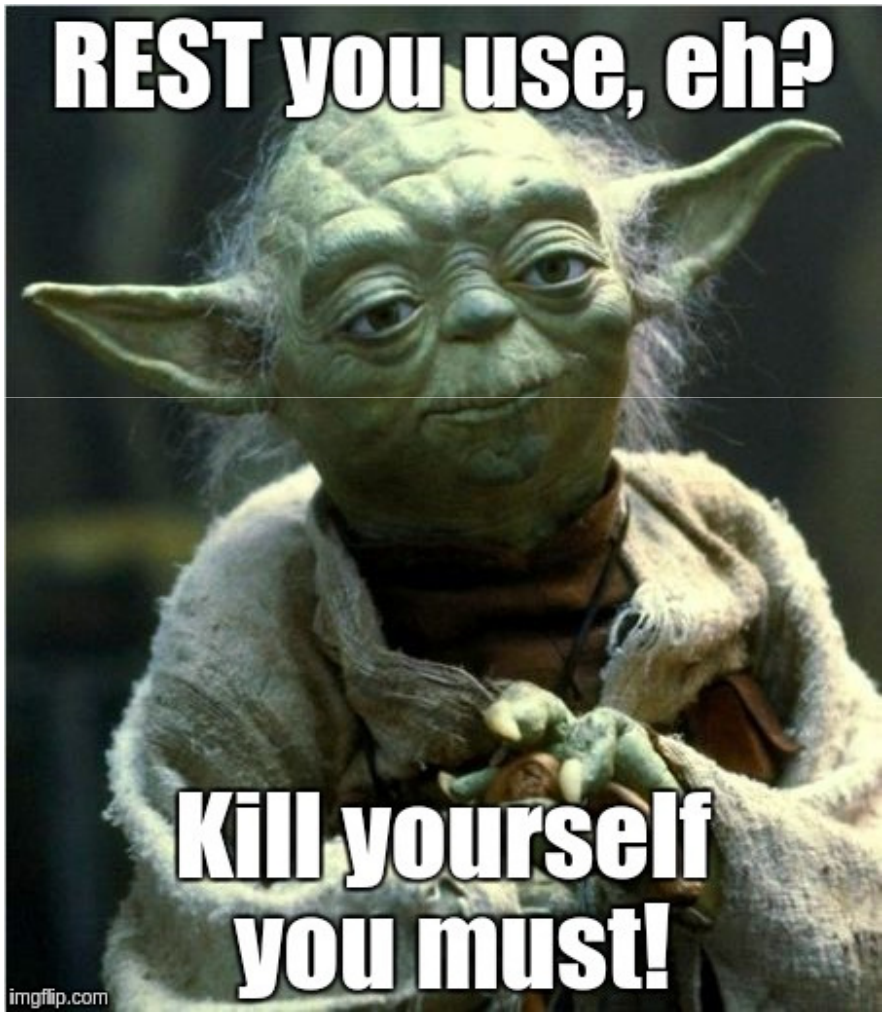
- Aplikacije na klijentima dohvaćaju podatke na serverima
- Kako implementirati serverske API-e?

REST API?

- REST definirao Roy Fielding u svojoj dizertaciji iz 2000. g.
- On je usko povezan sa razvojem weba i HTTP protokola u ‘90-tim

- HTTP je inicijalno zamišljen **za statični web sadržaj -** skupina resursa lociranim pod nekim URL-ovima
- No što je sa web aplikacijama?

REST za web aplikacije?



Problemi s REST API-ima...

- API je podijeljen u „logične” resurse
- Ti resursi, sa stajališta klijenta, vjerovatno:
  - Imaju nepotrebnih detalja
  - Imaju podatke u neodgovarajućem obliku
  - Nemaju sve potrebne detalje
- JOIN-ovi među resursima? Višestruki HTTP pozivi? Performanse? (pozivi često idu preko WAN-a!)
  - Grubi model weba: Map<String, Resource>
- Ne možete predvidjeti što klijenti žele – to može biti mješavina resursa, ili nečega što uopće nisu resursi itd...
- Kako posluživati N različitih klijenata sa jednim servisom?

„The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction.”

– Roy Fielding (izumitelj REST-a)

Da li vam je to poznato?


```
SELECT count(*), a.city, a.post_code
FROM employee e, address a
INNER JOIN ON a.employee_id = e.id
WHERE e.active = true
GROUP BY a.city
```

Rješenje

- Klijenti specifiraju što žele od servisa i dobivaju točno ono što su zatražili
- Svi zahtjevi idu prema jednom endpointu (npr. jedan URL)
- Zahtjeva složeniji „interpreter” zahtjeva na serverskoj strani
- Veći „up-front” trošak, no dugoročno smanjen trošak održavanja serverskog API-a
- Poboljšanje performansi zbog „batch” odgovora

Implementacije

- Implementacije uglavnom prisutne u „full-stack” Javascript svijetu

Falcor / JSONGraph

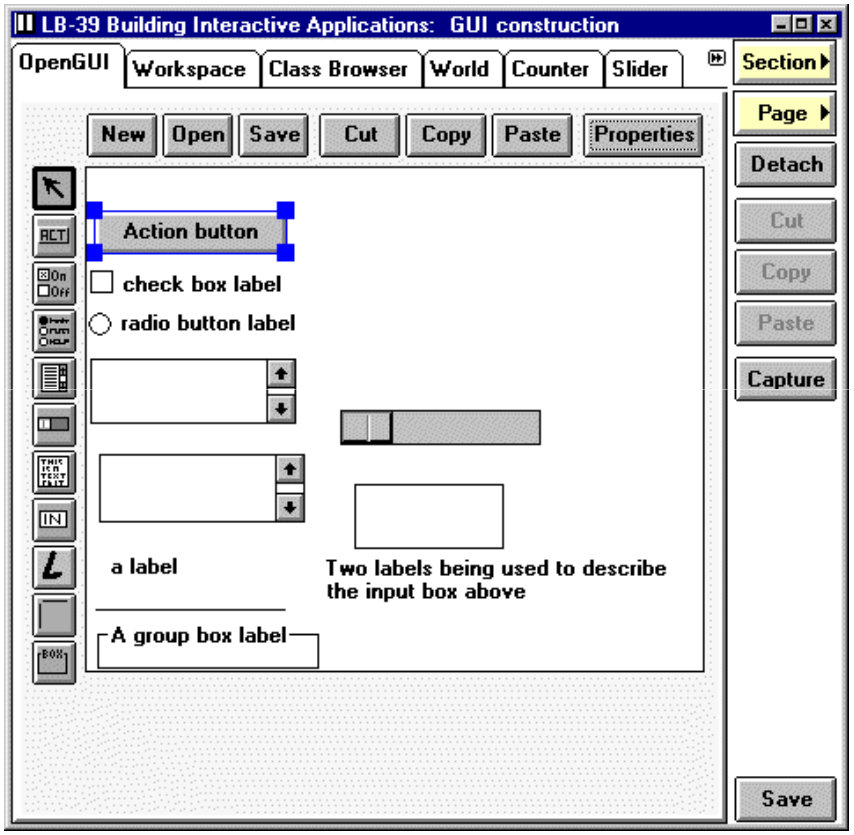
... iliti priča kako je NETFLIX eliminirao 90% njihovog „networking” koda

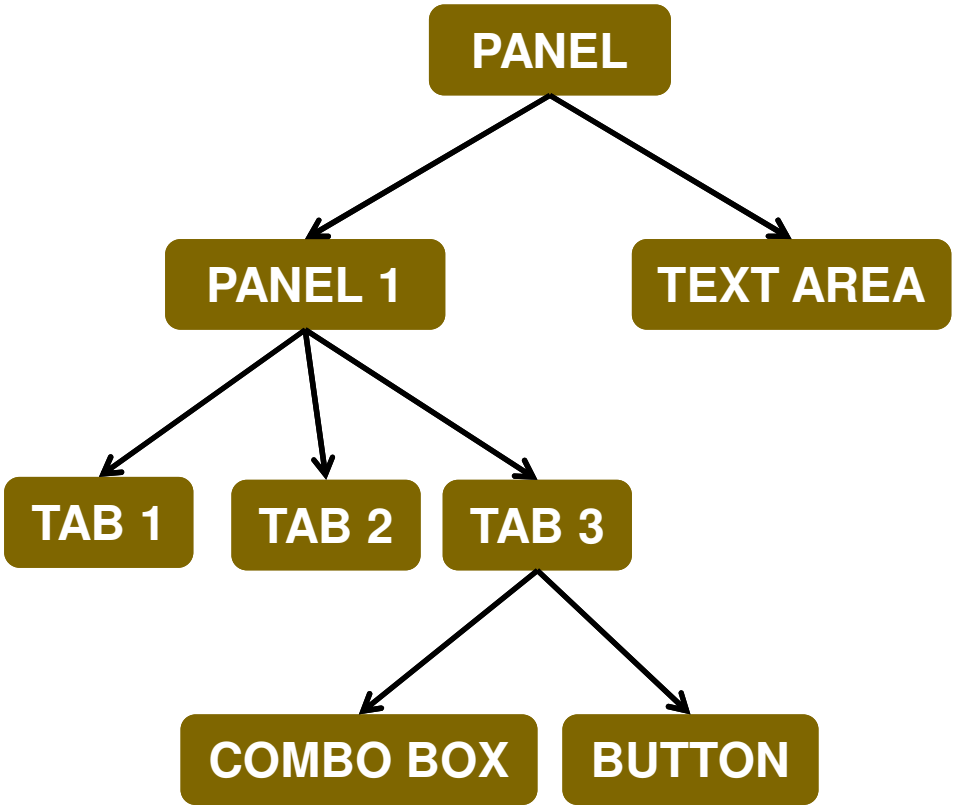
- JSONGraph je njihov „query language”

Relay / GraphQL

- Facebook izumio Relay i njegov query language GraphQL
- Na serveru treba imati GraphQL server koji interpretira zahtjev i dohvaća specifirane podatke

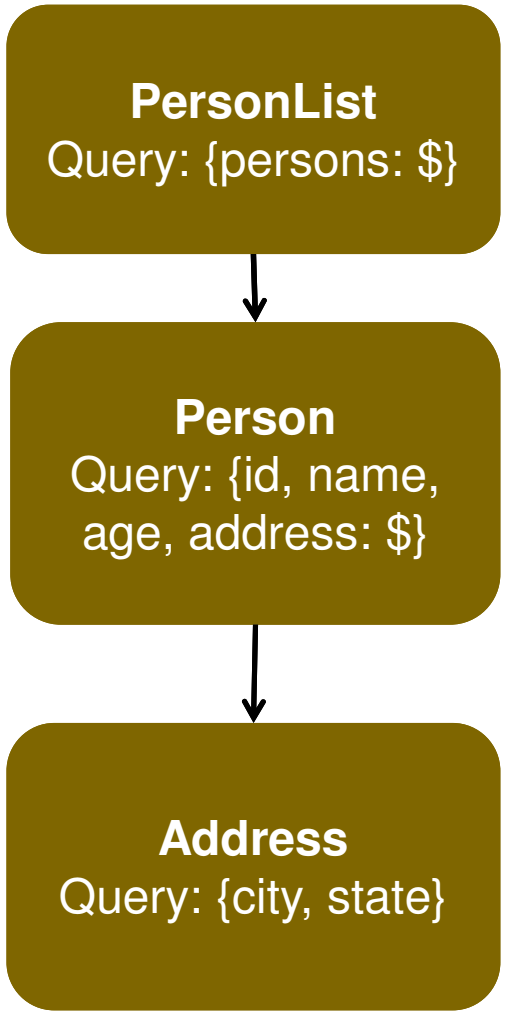
Struktura sučelja





- Sučelje aplikacije je „drvo” UI komponenti

Struktura sučelja



REQUEST:

```
{
persons: {
        id,
        name,
        age,
        address: {
                city,
                address
        }
}
```

RESPONSE:

```
{
”persons”: [
        { ”id”: 11,
          ”name”: ”John”,
          ”age”: 31,
          ” address ”: {
              ”city”: ”Dallas”,
              ”state”: ”Texas” }
        },
        { ”id”: 14,
          ”name”: ”Michael”,
          ....
        }]}
```

Primjer React komponente sa Relay/GraphQL-om

```
var friendInfo = React.createClass({
    statics: {
        queries: {
            user: function () {
                return graphql {
                    name,
                    mutual_friends { count }
                };} }
    }
    render: function () { return {
        <div>
            <span>{this.props.user.name}</span>
            <span>Mutual Friends:</span>
            <span> {this.props.user.mutual_friends.count}</span>
        </div>
    }}
}
```

QUERY DIO

RENDER DIO

„~~LISP~~ **programmers know the value of everything and the cost of nothing.**” – Alan Perlis

Poteškoće

- Network caching
  - REST je tu jak
- Autorizacija
  - nešto kompleksnija zbog bogate specifikacije zahtjevanih podataka
- Preveliki inicijalni trošak implementacije arhitekture?
- Neraširenost trenutnih implementacija

Zaključak

- Klijenti specifiraju točno što žele, u najmanje detalje
- Konvertirajuća logika na klijentu nestaje
- Front-end tim ne opterećuje back-end tim
- ”Batching” odgovora od servera poboljšava performanse


That's all Folks!