# SofaScore PHP akademija
# 12. predavanje

2. lipnja 2021.

Alen Murtić

# Sadržaj

- Izrada JSON API-ja
  - REST
  - Serijalizacija
  - ParamConverter
- Sigurnost

# REST

- Skraćenica za Representational State Transfer
- Stil programske arhitekture
- Nije isto kao HTTP
  - REST nije tehnologija nego princip
  - Web servis izveden HTTP-om može biti RESTful
- Ključna apstrakcija je **resurs**
  - Svaka informacija koja se može imenovati je resurs, npr. ova prezentacija je resurs
  - Resurs može biti u različitim formatima – tekst, slika, video, …
- Metode pristupa resursima: GET, PUT, POST, DELETE, HEAD, PATCH, OPTIONS
  - Detaljne specifikacije pojedine metode: https://restfulapi.net/http-methods/
- Preduvjet lakše izrade RESTful API-ja je dobro dizajniran model baze podataka
- U praksi se teško držati apsolutno svih principa

# Serijalizacija

- Pretvorba podataka u format koji se može spremiti ili slati mrežom
- Podrazumijeva mogućnost rekonstrukcije – **deserijalizacije**
  - Hash vrijednost podatka se može spremiti i slati mrežom, ali se originalni podatak ne može rekonstruirati
- Formati: CSV, XML, **JSON**
- Zašto JSON? De-facto standard za HTTP Web API-je
- JSON osnovni tipovi: number, string, boolean, null, **array** (lista), **object** (asocijativno polje)

# Serijalizacijske grupe

- Serijalizacija nekog objekta je rekurzivna – ako postoji objekt koji je atribut tog objekta i on će biti serijaliziran
  - Ako dovoljno povežemo entitete, na kraju možemo imati samo rutu /all koja je ispis cijele baze
    - Direktna suprotnost REST principu
  - Rekurzivnost je dobra značajka kada se ispravno koristi
- Serijalizacijske grupe – serijaliziraju se samo oni atributi koji su definirani kao serijalizacijske grupe pojedine akcije controllera
  - Anotacija analogna @ORM anotaciji, npr. @Groups({"common"})
  - Serijalizacija je i dalje rekurzivna, ako je pojedini atribut zapravo objekt koji ima svoje atribute

# Kako odabrati serijalizacijske grupe?

- Imenovanje je proizvoljno
- Common, id ili nešto slično - serijalizacijska grupa u kojoj su najosnovniji podaci o entitetu ("id" je logično samo identifikator, a "common" npr. identifikator i ime) koji se rijetko mijenjaju
- Načelno je dobra praksa ne pretjerivati s brojem serijalizacijskih grupa jednog entiteta, npr. možemo imati 3 razine – common, entitet_extended, entitet_full
  - Nisu dovoljne samo ekstended i full zato što iako je neki atribut dohvaćen common grupom, rekurzivno će se serijalizirati atributi iz sve tri njegove grupe ako ruta vraća sve tri razine
- Ključan element dizajna: caching

# Serijalizacija i caching

- Potpuni REST API se savršeno cachira
  - Sve informacije o entitetu osim identifikatora se mogu dobiti samo na entitet/{id} GET ruti
  - Svaka promjena nad entitetom invalidira cache točno jedne rute
- API koji nije potpuni REST treba dizajnirati i analizirati za dobar caching
  - Odustajanje od REST principa je u pravilu zbog smanjenja broja HTTP zahtjeva s frontend platformi kod GET ruta tako da se serijalizira visak podataka
    - Npr. u popisu igrača na utakmici mozemo serijalizirati ime i primarnu poziciju samog igrača, to teško da će se promijeniti dok traje cache takve rute
  - Ne smijemo serijalizirati previše podataka
    - Dohvat nepotrebnih podataka iz baze
    - Višak prometa frontend platformama zbog podataka koje ne koriste
    - Ako se nešto promijeni, teško je invalidirati cache svih ruta na kojima se podatak nalazi

# ParamConverter

- Tipični tok pojedine akcije u controlleru:
  - Primi $id, dohvati repozitorij entiteta, dohvati objekt s identifikatorom $id, ako ne postoji baci 404 (not found) iznimku, nešto napravi s objektom
    - Do "nešto napravi" se sve ponavlja u svakoj akciji, za neke i više puta jer primaju više različitih identifikatora
    - "Code bloat" - nepotreban višak programskog koda
- ParamConverter
  - Umjesto da programer piše višak koda, samo mora dodati anotaciju poput:

    *@ParamConverter("manager", class=Manager::class, options={"managerId" = "id"})*
    - Prvi string ime je varijable koja je argument metode u controlleru, ovaj option znači da se parametar managerId u ruti mapira na id atribut klase Manager
  - Ako je parametar rute "id", niti ne moramo pisati ovu anotaciju, sve će raditi samo dodavanjem Manager $manager varijable kao argumenta metode
- /match/{id}/player/{playerId}/statistics je bolje od /match/player/statistics?id=&playerId= zato što se lakše cachira

# Symfony i JSON API

- Symfony ima klasu *Symfony\Bundle\FrameworkBundle\Controller\AbstractController* koja sadrži osnovne potrebne značajke
  - Unutar akcije možemo vratiti new JsonResponse ili $this->json
    - Nije baš najljepši kod, ali radi
    - Bolje koristiti FOSRestBundle ili naslijediti AbstractController u neki svoj apstraktni
- Na sve 404 iznimke možemo vratiti isti 404 JSON odgovor
  - Bilo da ih generira ParamConverter ili *throw new NotFoundHttpException();*

# Podjela odgovornosti backenda i frontenda

- JSON API omogućuje da više različitih platformi koristi isti API
  - Ključno pitanje: što raditi na backendu, a što na platformama?
- Sve računski teške operacije moraju biti na backendu
- Ako postoji mogućnost da puno korisnika koristi stare verzije frontenda (npr. Android ekosustav), sve što se naknadno mora moći promijeniti je na backendu
- Statičke informacije (npr. prijevodi imena država ili zastave) trebaju biti na frontendu, a backend slati samo kodove
- Cilj: optimizacija korisničkog iskustva, količine prenesenih podataka i opterećenja servera
- Uglavnom nije egzaktna znanost, sve ima svojih prednosti i mana

# Sigurnost u Symfonyju

- Samo korištenje Symfonyja u odnosu na čisti PHP je velik napredak u pogledu sigurnosti
  - Doctrine značajno smanjuje mogućnost SQL injectiona
    - SQL injection je ubacivanje zloćudnih SQL izraza u korisničke forme
  - Nema "kopanja" po superglobalnim varijablama poput $_SERVER
    - Iako se to može, ali bolje ne
- Prvo pravilo sigurnosti: "Don't reinvent the wheel"
  - Svi koji nisu sigurnosni eksperti ne bi trebali pisati vlastite implementacije sigurnosnih algoritama nego koristiti već implementirana rješenja
    - U idealnom slučaju da napišemo 100% ispravan algoritam (što vjerojatno nećemo), taj dio koda ćemo vjerojatno zaboraviti jer radi
      - Algoritam može postati nesiguran
      - Javno dostupne biblioteke netko redovito revidira
  - Symfony nudi sve bitnije algoritme

# Symfony security bundle

- Početna točka dodavanja autentikacije i autorizacije u sustav
- Osnovni algoritam (https://symfony.com/doc/current/security.html):
  - 1. Definiranje User klase - više opcija
    - MakerBundle -> make:user komanda ili FOSUserBundle su najbolje dvije opcije
  - 2. Definicija UserProvider klase
    - Prethodne dvije opcije stvore osnovna rješenja kreiranjem User klase
  - 3. Definicija encodiranja lozinki
  - 4. Definicija vatrozida (firewalla), tj. principa zaštite
- Svaki korisnik može imati 0+ rola (uloga) koje mu omogućuju određena prava u aplikaciji
  - Struktura uloga je najčešće hijerarhijska (stablasta/piramidalna) - jedna uloga može biti sačinjena od više drugih
- Pravila se definiraju u *security.yml* datoteci

# 2. projekt

- Zadatak iz SofaScore domene
  - Sustav za neku vrstu administracije ili prikaza podataka
  - Svi zahtjevi aplikacije će biti definirani u projektnom zadatku
- Cjelokupno znanje stečeno na akademiji s fokusom na Symfony
  - Naravno prilagođene kompleksnosti
  - Fokus na bitne i detaljno obrađene teme
- Puno opcija za skupljanje maksimalnog broja bodova
  - Nije potrebno riješiti sve točke

# Hvala na pozornosti!


- Alen Murtić, alen.murtic@sofascore.com
- Karlo Knežević, knezevic.karlo1@gmail.com