SofaScore PHP akademija
11. predavanje

27. svibnja 2021.

Alen Murtić

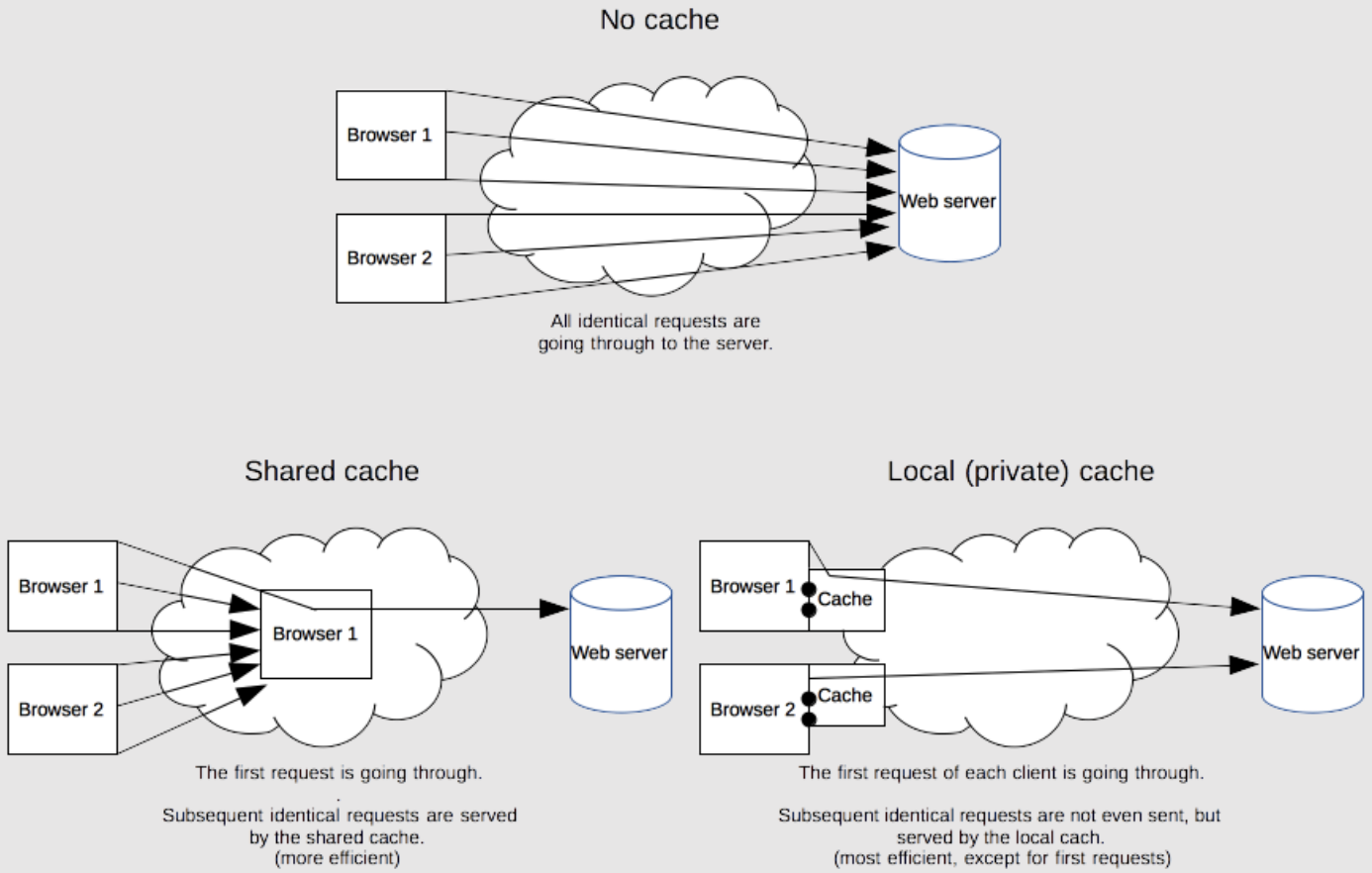
Sadržaj

- Koncept cachea
  - HTTP caching
  - ORM caching
- Red poslova (job queue)
  - Slušanje promjena nad entitetom
  - Izrada jednostavne verzije uz pomoć Doctrinea i Redisa
- Testiranje

Koncept cachea

- Pretpostavka: web poslužitelj svaki podatak definira jedinstvenim identifikatorom resursa (URI), npr. stranicu Hrvatske na SofaScoreu može se dohvatiti jedino putem https://www.sofascore.com/team/football/croatia/4715
- Ako više korisnika želi saznati iste javne podatke u kratkom vremenskom razdoblju, u kojem se oni sigurno neće promijeniti, svi ti zahtjevi se ne moraju izvršiti na PHP poslužitelju, već drugi korisnik može dobiti isti odgovor koji je dobio prvi
  - Kako to postići? **HTTP cachingom**
- HTTP cache je mehanizam privremenog spremanja rezultata HTTP zahtjeva tako da se svi zahtjevi ne moraju izvršiti na poslužitelju

HTTP cache



Više na: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

HTTP cache

- Definira ga response header **cache-control**
- Ako definiramo A kao browser, a B kao server, postoje dvije vrste HTTP cachinga, ovisno cachira li se zahtjev na uređaju koji ga je poslao ili na uređaju koji je odgovorio
  - Lokalni cache - A pošalje zahtjev, B odgovori, a njegov odgovor sadržava header: **cache-control: max-age:x**
    - A sprema taj odgovor
    - Sljedećih x sekundi A odgovor za taj URI traži kod sebe umjesto da šalje zahtjev prema B
  - Server cache – A pošalje zahtjev, B odgovori, a njegov odgovor saržava header **cache-control: s-maxage: y,max-age: x** ili samo **cache-control: max-age:x**
    - B sprema odgovor na y sekundi u prvom slučaju ili x u drugom (tada je lokalni cache jednak poslužiteljskom) i umjesto da pokreće PHP aplikaciju, kod sljedećeg zahtjeva za isti URI, B odgovor potraži u cacheu
    - Nužno ga je dodatno implementirati, on ne radi *out-of-the-box*

ORM cache

- Pogledajmo sljedeći programski kod:

```
$this->em->getRepository(Goal::class)->find(3);
$this->em->getRepository(Goal::class)->find(3);
$this->em->getRepository(Goal::class)->find(3);
```

  - Tri puta se dohvati repozitorij entiteta Goal koji u bazi pokuša pronaći Goal s identifikatorom = 3
  - Umjesto da radi tri upita na bazu, Doctrine kod prvog upita u svoju internu mapu spremi 3 => referenca na popunjeni Goal objekt te kod drugog i trećeg upita samo vrati isti objekt iz mape
    - Ta mapa je Doctrineov cache
- ORM caching može biti izuzetno koristan kod lijenog učitavanja (lazy loadinga)

Slušanje promjena nad entitetom

- Problem: na 5+ mjesta u kodu se mijenja vrijednost atributa nekog objekta, a za svaku promjenu tog atributa treba odraditi neku proceduru
  - Rješenje: detektirati svaku promjenu entiteta i zasebno odraditi tu proceduru
- CRUD – create, read, update, delete – CUD radi promjene
  - Doctrine nudi slušanje svake od tih akcija samo za objekte koje spremamo u bazu
    - Zapravo neposrebno prije i poslije svake od akcija
  - Create – prePersist, postPersist
  - Update – preUpdate, postUpdate
  - Delete – preRemove, postRemove
- U bazu se propagiraju samo promjene iza kojih je pozvana metoda **flush**
  - Zato postoje i preFlush, postFlush, onFlush događaji
- Moguće je provjeriti staru i novu vrijednost atributa

Job queue koncept

- Ako je izvođenje programa prekinuto zbog detektirane promjene, treba li odmah pozvati željenu proceduru? NE.
  - Prekida rad trenutne operacije koja bi se izvršila kasnije i sporije
  - Ako i ta procedura dodatno mijenja entitete u bazi, poziv flusha prije nego što se izvršio prethodni flush može dovesti do grešaka u radu aplikacije
- Umjesto izvršavanja procedure u istom procesu, njezini argumenti se ubace u prioritetni red poslova (job queue)
  - Ona će se izvršiti negdje i nekad u zasebnom procesu
  - SofaScore koristi https://beanstalkd.github.io/, ali lako je implementirati vlastiti jednostavni red poslova
- U red poslova mogu se ubaciti i rascijepani dugi procesi

Red poslova uz pomoć Redisa

- Ideja: implementacija reda poslova tako da je ključ spojeno ime servisa koji posao izvršava i njegove metode, a vrijednost lista argumenata pojedinog posla
  - Npr. U aplikaciji postoji servis se zove TeamService, on ima metodu calculateRecentForm koja prima identifikator tima za koji treba izračunati formu
    - U Redis to spremamo pod ključem TeamService.calculateRecentForm s vrijednostima npr. 1, 2, 5, 10
  - Pojednostavljenje načina na koji radi beanstalkd

Symfony servisi i dependency injection

- Servis je način organizacije koda
- Symfony je počeo koristeći ServiceContainer koncept, ali novije verzije forsiraju dependency injection
  - Servisi se ne mogu dohvatiti iz containera, oni su privatni
  - Ovisnost jednog servisa o drugom definira kroz konstruktor
  - U services.yml datoteci treba postaviti sljedeće parametre:

```yaml
services:
    _defaults:
        autowire: true      # Automatically injects dependencies in your services.
        autoconfigure: true # Automatically registers your services as commands, event subscribers, etc.
        public: false       # Allows optimizing the container by removing unused services; this also means
                            # fetching services directly from the container via $container->get() won't work.
                            # The best practice is to be explicit about your dependencies anyway.
```

- Pojedini servisi ipak mogu biti javni, ali za njih eksplicitno treba definirati public: true

Testovi

- PHPUnit je nezavni programski okvir za testiranje PHP aplikacije
  - Koristi se i za testiranje Symfony PHP aplikacija
    - Integracija u symfony bundleu
- U projektu iz primjera dovoljno je pozvati:
  - *vendor/bin/simple-phpunit* i composer će automatski instalirati bundle, ako već ne postoji
    - Istom naredbom se pokreću svi testovi
- Testovi su organizirani u klase koje nasljeđuju Symfony\Bundle\FrameworkBundle\Tests\TestCase ili neku izvedenu klasu
- Metode koje testiraju nešto trebaju biti javne i njihovo ime mora počinjati riječju test
  - Sadrže pozive poput self::assertEquals(true, 'aaa' === 'aaa') gdje je prvi argument očekivana vrijednost, a drugi rezultat onoga što želimo testirati
- Kontinuirano testiranje: za svaku promjenu u kodu glavnog (master) brancha, trebaju prolaziti svi testovi koji postoje u projektu – alati poput Jenkinsa

Mock objekti

- Da bi mogli testirati neki kod koji ovisi o vanjskom sustavu, potrebno je koristiti taj vanjski sustav
  - Vanjski sustav može biti spor, imati maksimalni broj korištenja u nekom vremenu ili plaćen po jednom korištenju
  - Pad vanjskog sustava uzrokuje pad testova koji ovise o njemu
  - Rješenje: mockovi (lažni objekti)
- Mock objekti
  - Uvode predvidljivost u sustav testiranja
  - Programer definira njihovo ponašanje - ponajprije dozvoljene metode
    - Mogućnost definicije rezultata poziva metode za pojedini argument
  - Nedostaci: zahtijevaju uloženo vrijeme i uvode "code bloat" u testove
  - Mock klase X je u testnom okruženju kod provjere tipova varijabli ekvivalentan njenoj instanci

Test fixtures

- Da bi mogli testirati neki kod koji ovisi o podacima, potrebni su podaci
  - Pisanje tih podataka kao mock objekata može biti zamorno
- Test fixtures su podaci u bazi podataka privremeno popunjenoj za izvršavanje testova
- Podrška korištenjem DoctrineFixturesBundlea
- Metode setUp(BeforeClass) i tearDown(AfterClass)
- Lakše modeliranje stvarnog ponašanja u odnosu na mockove

Tipovi testova

- Funkcionalni
  - Unit (jedinični) - mali dijelovi koda
  - Integracijski – rad s vanjskim servisima ili vanjskim kodom
  - Sistemski – cjeloviti proizvod
  - Regresijski – provjera radi li kod nakon promjene
- Nefunkcionalni
  - Performanse, stres, testovi pouzdanosti, lokalizacijski, sigurnosni

Unit testovi

- Cilj: testiranje malih, što nezavisnijih dijelova koda
- Mogu biti vrlo jednostavni i granično besmisleni, ali imaju svoju svrhu
- Nužno testirati:
  - Statičke helper metode
  - Settere koji imaju logiku van svojeg naziva (npr. ako se igraču promijeni momčad, briše mu se broj na dresu, a metoda se samo zove *setTeam*)
- Idealno bi bilo testirati sve jedinične dijelove koda

Integracijski testovi

- Test stvarnog ponašanja korištenja nekog vanjskog elementa (servisa ili koda)
- Načelno se vanjski element ne bi trebao mockati
  - Inače testiramo mockanje, a ne integraciju – takav test je bolje ne imati nego oslanjati se na njegovu "točnost"
  - Prihvatljivo je da takvi testovi budu sporiji
- Kad se jednom testira integracija tog vanjskog elementa, mockanje je dozvoljeno u drugim testovima, čak i poželjno jer vjerojatno ubrzava izvođenje istih

Sistemski i regresijski testovi

- Sistemski - testiranje cijelog proizvoda ili većih povezanih dijelova tog proizvoda
- Regresijski – testiraju kod nakon promjene
  - Možemo testirati unatražnju kompatibilnost (backwards compatibility)
    - Iako o njoj možda ne ovisimo mi nego druge platforme
- Primjer testa koji je istovremeno sistemski i regresijski:
  - Dohvat podataka s neke rute i provjera zadovoljavaju li podaci neke zahtjeve
    - Sistemski – inicijalizacija projekta, pristup ruti, poslovna logika i manipuliranje podacima, serijalizator
    - Regresijski – ako nešto nije kao prije, test pada

Sljedeće (završno) predavanje

- Izrada JSON API-ja
- Sigurnost
- Kratki uvod u završni projekt

Hvala na pozornosti!

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