(Event Sourcing + Laravel) Introduction

Miro Hudak, miroslav.hudak@continuity.sk, @mx0r

Laravel Meetup 4 - 18. jún 2020

O mne

- Software Engineer
- Programujem od začiatku 90. rokov (od QBasic-u)
- Skúsenosti s vývojom desktop, web aj mobilných aplikácií
- V súčasnosti sa venujem hlavne C#, Swift a PHP + softvérovej architektúre
- CTO & Team Lead
- … a taktiež ma baví fotografia, LEGO, počítačové hry…

Motivácia

Micro/services

- Komunikácia jednotlivých services medzi sebou
- Robustnosť systému, zachovanie funkcionality aj pri výpadku niektorých častí
- Geograficky distribuované systémy
- Disaster recovery

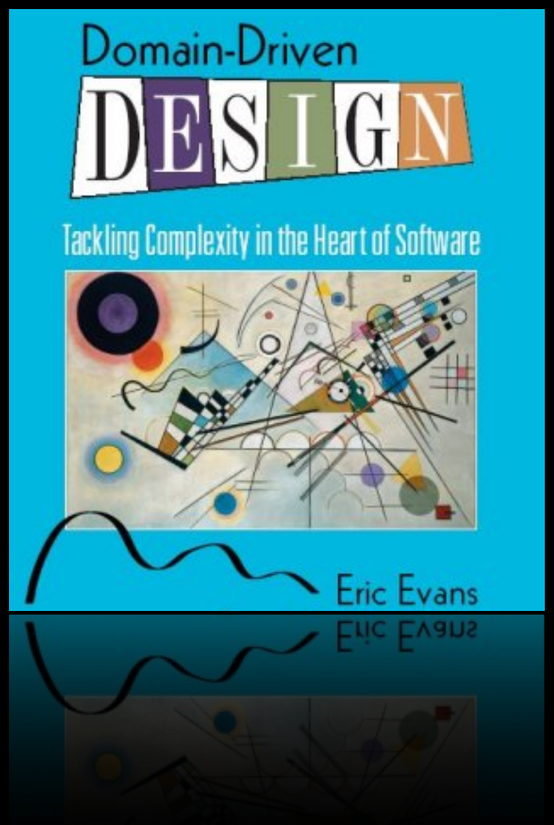
Databázy

- Väčšinou normalizované, nie optimalizované
- Zložité až nemožné "cestovanie v čase"
- Problematické poskytovanie reportov podľa neskorších požiadaviek klienta
- Problematická auditovateľnosť, nemožnosť zaručiť, že údaje v databáze neboli zmenené*

Event Sourcing?

DDD...

- ES je pattern z Domain-drive design (DDD)
  - základom architektúry riešenia sú biznis požiadavky
  - autor konceptu je Eric Evans, ktorý ho použil vo svojej knihe, ktorá koncept popisuje
- Definuje základné stavebné časti aplikácie:
  - Entity, Value Object, Aggregate, Domain Event, Service, Repository, Factory
- Nevýhodou je komplikovanejší vývoj, keď sa chceme držať všetkých princípov



... a Event Sourcing?

- Princíp, pri ktorom sú operácie nad systémom uložené v udalostiach - Events
- Command Query Responsibility Segregation (CQRS)
- Časti Event Sourcing architektúry:
  - Aggregate, Command, Event, State
  - Projection, Reaction

Časti ES architektúry

- Aggregate - konkrétny objekt, ktorý je upravovaný udalosťami
  - Command - príkaz na mutáciu agregátu
  - State - stav agregátu v danom momente v čase
  - Event - udalosť, ktorá sa stala v minulosti
- Projection - projekcia udalostí na "read model"
- Reaction - reakcia na prvý výskyt udalosti v systéme

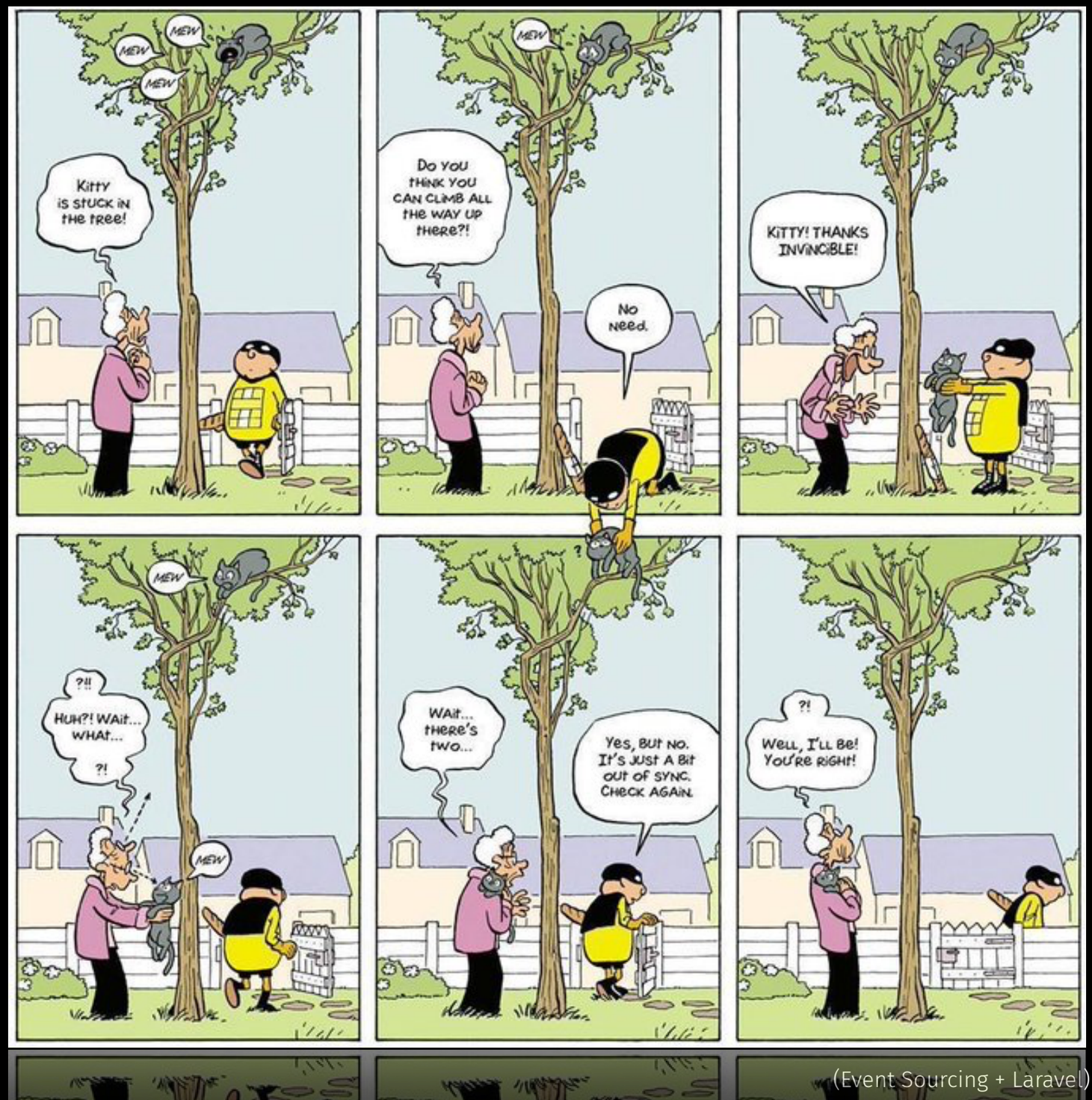
Výhody

- Asynchrónna komunikácia medzi jednotlivými services vo väčšom riešení*
- Cestovanie v čase!
  - možnosť vytvoriť nové projekcie z historických údajov
  - možnosť zistiť stav systému v konkrétnom čase v minulosti
- Jednoduchšia zmena databázového modelu v prípade potreby
- Nemožnosť upraviť už uložené eventy

Nevýhody

- Komplikovanejší vývoj, viac písania, avšak je možné napísať si generátory na základné veci a vytvoriť si vždy scaffolding
- Distribuovaná aplikácia
- Eventual Consistency
- Potreba dokumentácie
- Nemožnosť upraviť už uložené eventy

Eventual Consistency?!



Invincible by Pascal Jousselin

Event Sourcing!

Event

- Popisuje udalosť v systéme
- Napríklad príjazd auta do parkovacieho domu, platbu, odjazd auta z parkovacieho domu
- Udalosti budú v systéme uložené a nebudú nikdy zmazané
- Môže byť priradená k agregátu - napr. parkovaciemu domu

```
class CarEntered
    implements ShouldBeStored
{
    /** @var string */
    public $licensePlate;

    function __construct(
      string $licensePlate)
    {
        $this→licensePlate
          = $licensePlate;
    }
}
```

Aggregate

- Trieda, ktorá popisuje objekt v DDD a ktorá má stav
- Stav je, podobne ako v iných patternoch, aktuálny stav objektu v danom čase (nemusí byť nutne prítomnosť)
- Špecifikuje príkazy, ktoré sa dajú nad agregátom vykonať
- Popisuje, akým spôsobom udalosti menia jeho stav

```
class CarParkAggregate
    extends AggregateRoot
{
    //region --- Commands ---

    public function create(
      string $name): self
    {
        return $this→recordThat(
          new CarParkCreated(
            $name));
    }

...
```

Projection

- Transformácia (projekcia) udalosti na read model
- Read model je denormalizovaná forma údajov, optimalizovaná pre konkrétne použitie
- Projekcie je možné vytvoriť aj neskôr ako je uložená udalosť do event store a následne udalosti nad týmto projektorom prehrať
- Nie je nutné aby projekcia obsahovala všetky údaje zo stavu agregátu, iba potrebné

```
class CarParkProjector
  implements Projector
{
  function onCarParkCreated(
    CarParkCreated $event,
    string $aggregateUuid)
  {
    $carPark = new CarPark([
      'uuid' ⇒ $aggregateUuid,
      'name' ⇒ $event→name]);
    $carPark→save();
  }

...
```

Read / Write

- V ideálnom prípade sa číta z projekcie, avšak je možné čítať aj z agregátu, prichádza pritom k prehrávaniu udalostí nad agregátom a ich následné vykonávanie
    - Je však možné (a odporúčané) vytváranie snapshotov, čo sú - ako názov hovorí - momentálne stavy v čase po aplikácii N eventov
    - Snapshoty je možné robiť okamžite, alebo priebežne napríklad raz za deň - v podstate agregátová cache
    - Následne sa pri vyžiadaní agregátu zoberie posledný stav zo snapshotu a aplikujú sa iba udalosti, ktoré vznikli neskôr
- Zápis sa robí iba do agregátu pomocou udalostí, ktoré sú následne okamžite* transformované do read modelu pomocou projekcií

Poďme k praxi...

Demo aplikácia

- Jednoduchá evidencia áut v parkovacom dome
- K dispozícii na GitHub-e: https://github.com/mx0r/lm4-event-sourcing
- Základný Laravel 7
- spatie/laravel-event-sourcing

```
composer create-project ↩
  laravel/laravel ↩
  lm4-event-sourcing

composer require ↩
  spatie/laravel-event-sourcing

./artisan serve
```

API

- Jednoduché API rozhranie:
  - `GET /api/car-parks`
  - `POST /api/car-parks`
  - `POST /api/car-parks/{guid}/cars`
  - `PUT /api/car-parks/{guid}/cars/{license-plate}/pay`
  - `DELETE /api/car-parks/{guid}/cars/{license-plate}`
  - `GET /api/car-parks/reports`

Live Demo

Otázky?

Zaujímavé odkazy

- Kleppmann, M: Designing Data-Intensive Applications
  https://dataintensive.net

- Fowler, M; CQRS & more
  https://martinfowler.com/bliki/CQRS.html

- Boggard, J; Six Little Lines of Fail:
  https://www.youtube.com/watch?v=VvUdvte1V3s

Ďakujem!

- Demo aplikácia: https://github.com/mx0r/lm4-event-sourcing
- V prípade ďalších otázok môžete napísať na miroslav.hudak@continuity.sk alebo twitter `@mx0r`
- Ďakujem za pozornosť!