CQRS and Event Sourcing

with Event Horizon

Todays Agenda

- CQRS
- Event Sourcing
- Break!
- Event Horizon
- Case: TodoMVC
- Case: Rapideye by Great Beyond

Max Ekman, Looplab AB

- Owner & Software Engineer
- Consulting firm
- System Architecture
- Cloud and Backend
- Golang
- Google Cloud Platform
- Scrum

CQRS

What is CQRS?

Command and Query Responsibility Segregation

Different requirements for viewing and modifying data

Simple in practice, but steep learning curve

The Bible: http://cqrs.nu/Faq

Domain

Describes a real business domain

Encapsulates business logic

Uses non-tech terminology

Bounded context - a fairly independent part of a business

Entities and Aggregates

An item in the domain

Has a unique ID

Items can have the same properties as other items

Aggregates encapsulate the logic for a single entity

Stores the items for later queries

Commands

Describes an action that *should* happen - e.g. BuyItem

The CUD in CRUD - create, update, delete

Contains all info about the action - ID, Who bought it, when etc.

Processed by an Aggregate

Succeeds or fails, decided by the Aggregate

Validated before it even gets to the Aggregate

Query Engine

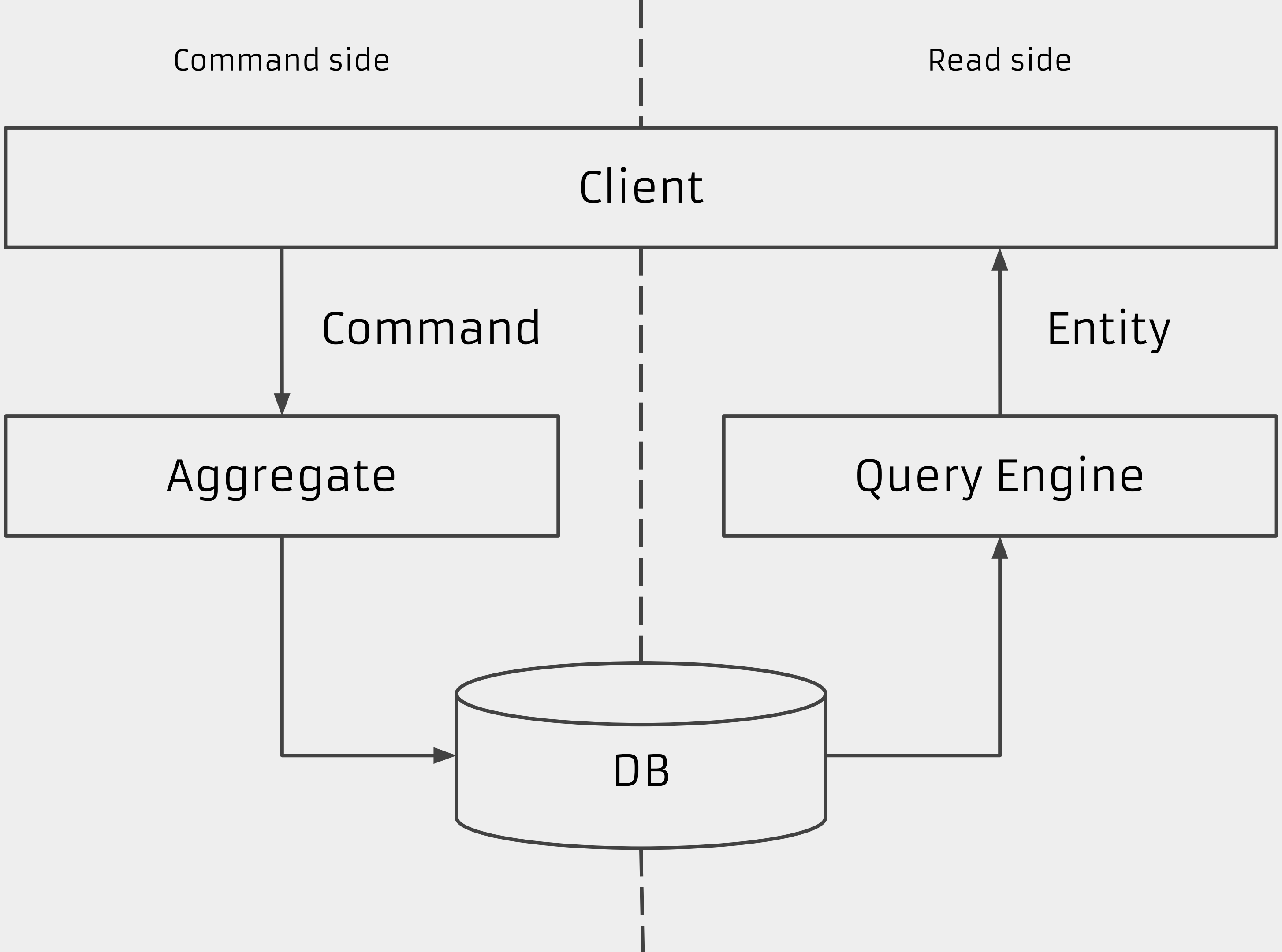
Read only access to entities

The R in CRUD - read

Projected by the aggregate or a dedicated projector

Multiple read models per aggregate allowed and encuraged

Sometimes called materialized views


Command side
Read side
Client
Command
Entity
Aggregate
Query Engine
DB

Pros and Cons

- + Separation of concerns
- + Full control over data mutation
- + Easy to validate actions
- + Uses domain terminology
- + Creates a clear API
- - More complex, at least in the beginning
- - Harder to iterate UI, usually
- - Can be too strict

Event Sourcing

What is Event Sourcing?

Describes what happens in a domain as continuous timeline

Traditionally only the latest state is known

Tracks the history of everything

Enables data replay, undo/redo, diffs etc.

More complex

Events

Describes an event that *has* happened - e.g. ItemBought

Contains all info about the what happened - ID, Who bought it, when etc.

Processed by event handlers - store data, send mails etc.

Handling should preferably never fail - but it does happen

Projections

The process of transforming events to a state

In practice it does CRUD to the read models

Allows re-projecton of events

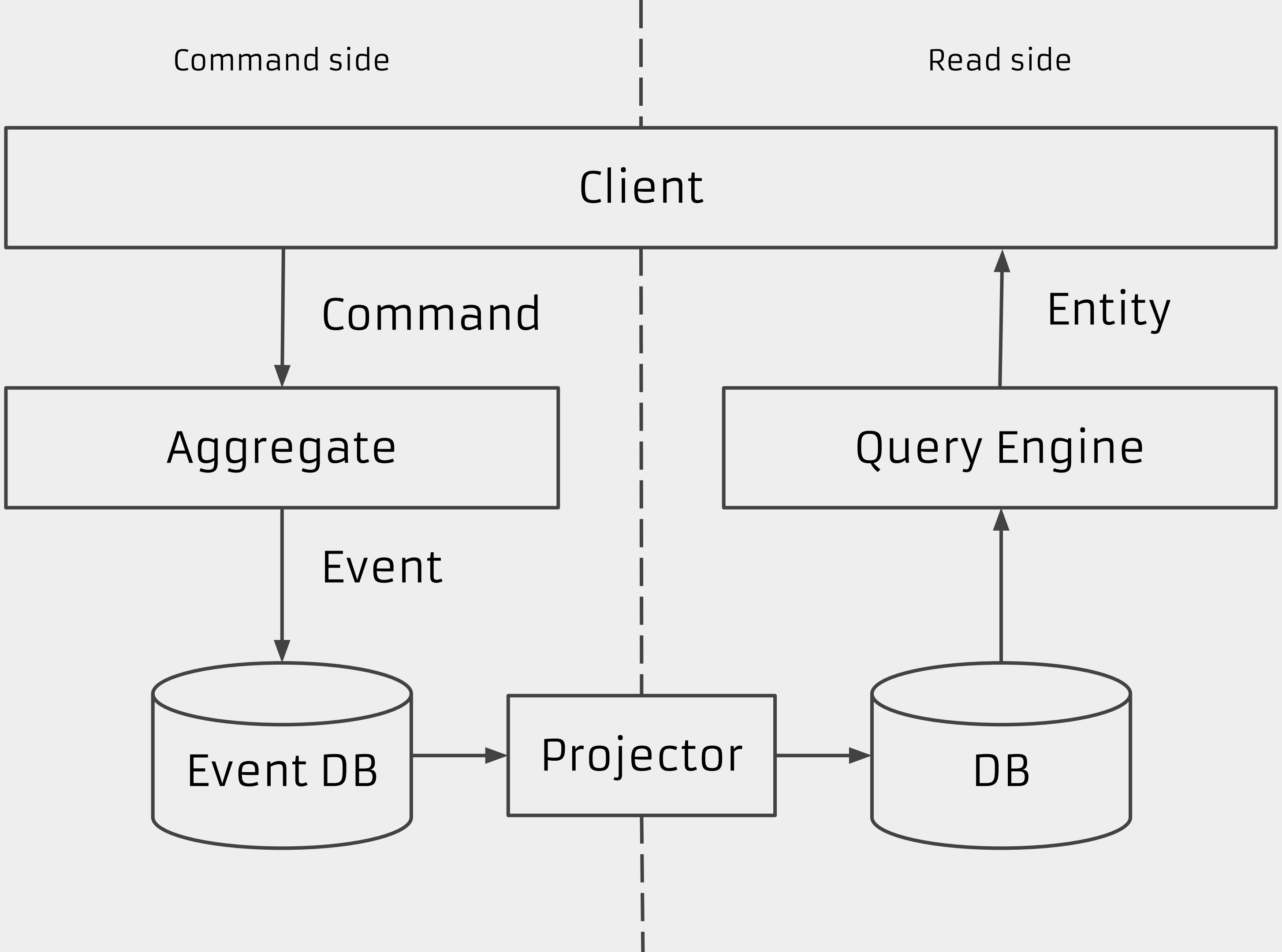
Flexibility when migrating models

Observers

Useful for passive reactions to events

Sending mail and other notifications are typical examples

Can be used to notify clients and UIs about changes


Command side
Read side
Client
Command
Entity
Aggregate
Query Engine
Event
Event DB
Projector
DB

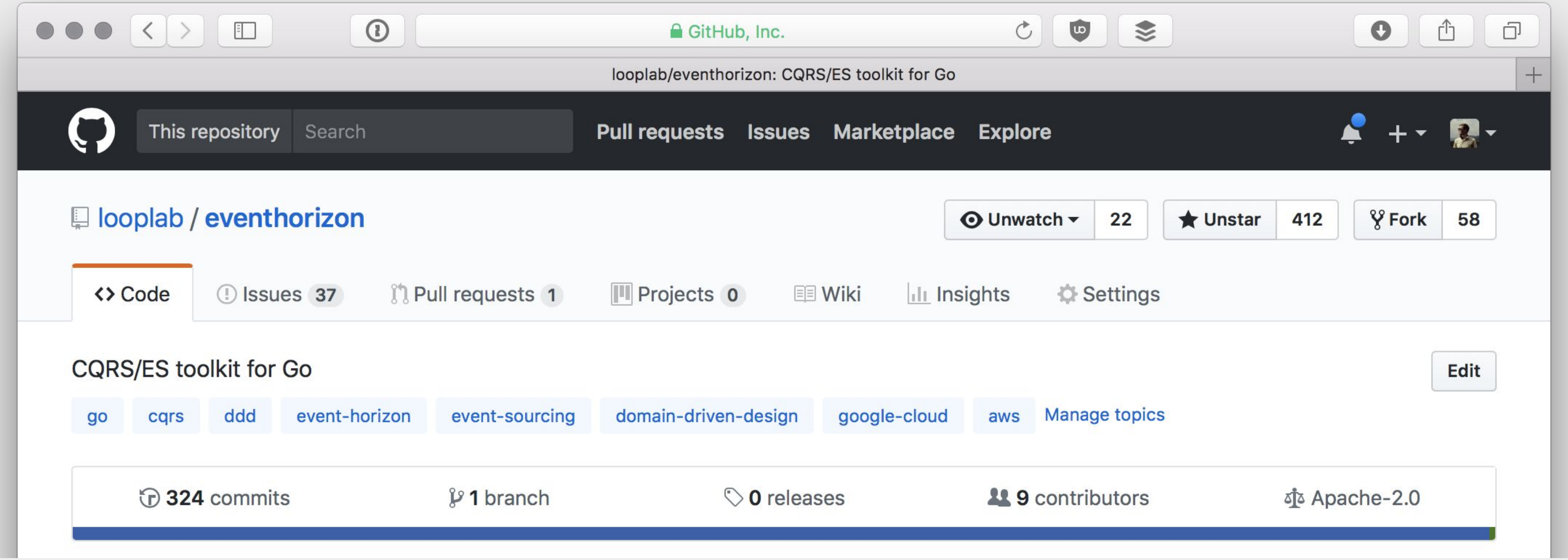
Pros and Cons

+ Full history of all data
+ Multiple projections/views
+ Easy to do undo/redo, audit logs etc.
+ Reactions on events; mail etc.
+ Uses domain terminology
- Migrations are hard, really hard
- Events that can fail are tricky to do correctly
- Even higher complexity, lots of moving parts

Break!

Event Horizon

github.com/looplab/eventhorizon



CQRS/ES toolkit for Go

Project Overview

A toolkit of components

Written in Golang

Not tied to specific DBs or other infrastructure

3-4 years old, but not API stable or feature complete yet

Fairly mature drivers for MongoDB and Redis

Components

Basic types - Command, Event, UUID, handler interfaces like http.Handler

Event storage

Read model repository

Aggregate handling

Projection

Utilities

Commands

```
type Command interface {
    AggregateID() UUID                  // Which entity to target.
    AggregateType() AggregateType   // What part of the domain to act on.
    CommandType() CommandType       // What to do to the entity.
}
```

Commands, cont.

```
type Create struct {
    ID eh.UUID   `json:"id"`
    Brand string `json:"brand"`
}
func (c *Create) AggregateType() eh.AggregateType { return “car” }
func (c *Create) AggregateID() eh.UUID            { return c.ID }
func (c *Create) CommandType() eh.CommandType     { return “create” }
```

Events

```
type Event interface {
    EventType() EventType           // What happened.
    Data() EventData                // All data about what happened.
    Timestamp() time.Time           // When it happened.
    ...
}
```

Events, cont.

```
type Event interface {
    ...
    AggregateType() AggregateType   // In which part of the domain.
    AggregateID() UUID              // Which entity did it happen to.
    Version() int                   // Where in the timeline did it happen.
}
```

Events, cont.

```
Created = eh.EventType("car:created")


type CreatedData struct {
    Brand string `json:"brand" bson:"brand"`
}
```

Aggregate

```
type Aggregate struct {...}


// Transforms commands into events.
func (a *Aggregate) HandleCommand(ctx context.Context, cmd eh.Command) error


// Updates the aggregate state from events.
func (a *Aggregate) ApplyEvent(ctx context.Context, event eh.Event) error
```

Model

```
type Car struct {
    ID    int    `json:"id"    bson:"id"`
    Brand string `json:"brand" bson:"brand"`
}

func (c *Car) EntityID() eh.UUID { return c.ID }
```

Projector

```
type Projector struct {...}


// Projects an event onto an entity, returning the modified entity.
func (p *Projector) Project(ctx context.Context,
    event eh.Event, entity eh.Entity)
    (eh.Entity, error)
```

Roadmap

More drivers, GCP Datastore and Pub/Sub

Less complex setup of domains

Easier to understand async event handling (actor model?)

Production utilities; migration, reply etc.

Production hardening

Case: TodoMVC

Intro

Domain overview

System overview

Demo

Further reading

Overview

The classic TodoMVC from todomvc.com

Frontend in Elm!

Event driven UI

Backend using both CQRS and event sourcing

Uses three new HTTP utilities for Event Horizon

Uses MongoDB for storage and a in-process event bus

Domain

Commands - Create, AddItem, SetItemDescription, CheckItem etc.

Events - Created, ItemAdded, ItemDescriptionSet, ItemChecked etc.

Aggregate and Projector

Model:

- TodoList - One list, singleton in the example
- TodoItem - One for each checkable row

Demo!

Case: Rapideye

by Great Beyond

Great Beyond - greatbeyond.se

Hybrid agency doing web, film, print etc.

Helps actors in society to communicate

Activism in Sthlm - political and non-political campaigns, strategy

Tech in GBG - organization tools, hosting, digital campaigns

Rapideye

A organization management tool

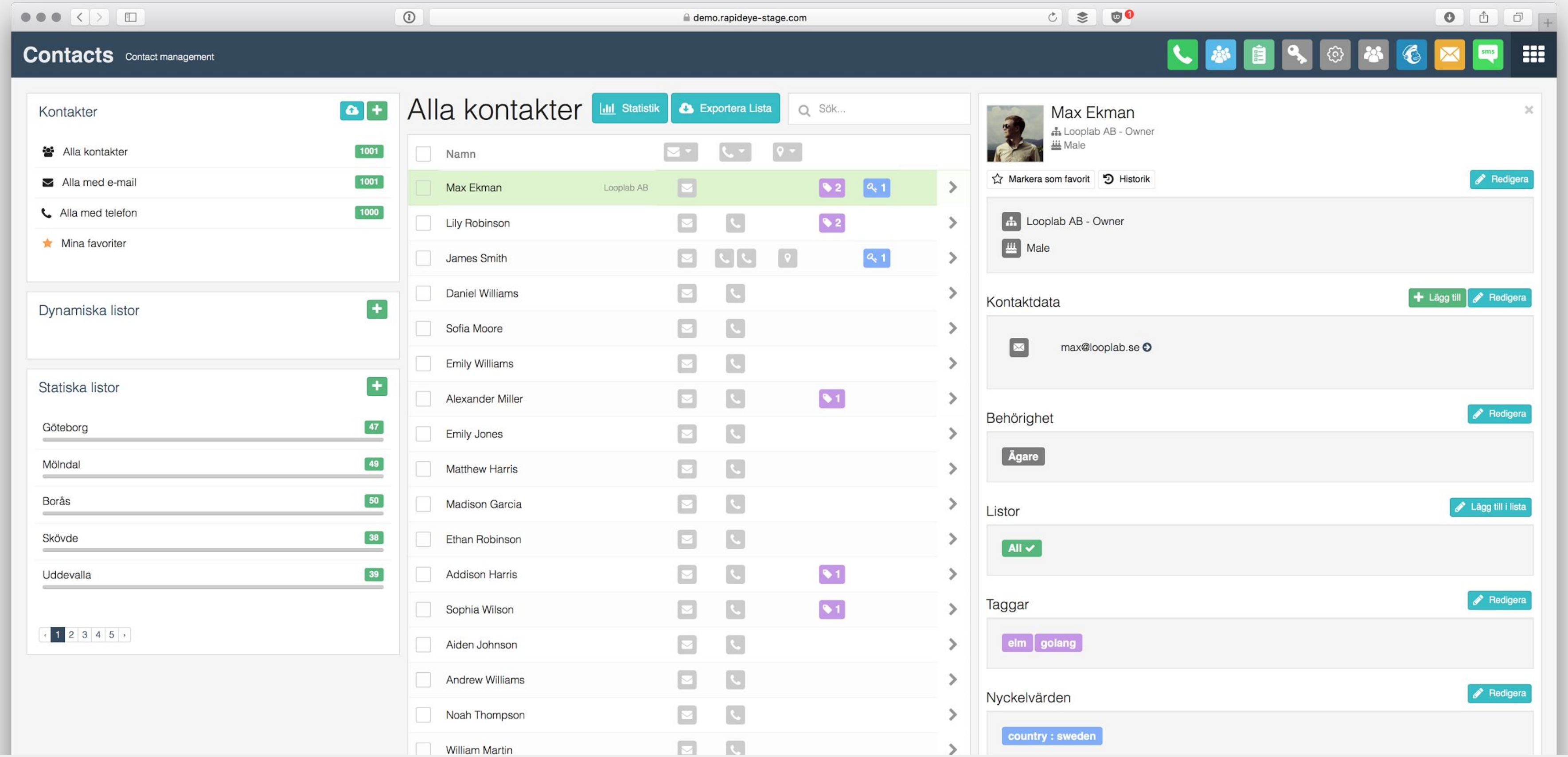
Handles members, communications, campaigns etc.

Modular system

Sold B2B according to client demands

Currently in use by a few clients

Beta tested as call center app in Kyrkovalet 2017


demo.rapideye-stage.com

Contacts Contact management

Kontakter

Alla kontakter	1001
Alla med e-mail	1001
Alla med telefon	1000
Mina favoriter	

Dynamiska listor

Statiska listor

Göteborg	47
Mölndal	49
Borås	50
Skövde	38
Uddevalla	39

‹ 1 2 3 4 5 ›

Alla kontakter Statistik Exportera Lista Sök...

Namn					
Max Ekman	Looplab AB			2	1
Lily Robinson				2	
James Smith					1
Daniel Williams					
Sofia Moore					
Emily Williams					
Alexander Miller				1	
Emily Jones					
Matthew Harris					
Madison Garcia					
Ethan Robinson					
Addison Harris				1	
Sophia Wilson				1	
Aiden Johnson					
Andrew Williams					
Noah Thompson					
William Martin					

Max Ekman
Looplab AB - Owner
Male

Markera som favorit Historik Redigera

Looplab AB - Owner
Male

Kontaktdata Lägg till Redigera

max@looplab.se

Behörighet Redigera

Ägare

Listor Lägg till i lista

All ✓

Taggar Redigera

elm golang

Nyckelvärden Redigera

country : sweden

Benefits of CQRS / ES

Free auditlog of all events

Reactive UI, across browsers and devices

Domain terminology understandable by non-tech people

Easy to do notifications, mail etc.

Challanges

Migrations!

System efficiency with large data sets

Complexity

Thanks!

hello@looplab.se