Hexagonal Architecture with Ruby on Rails

Šiašiakampė architektūra su Ruby on Rails

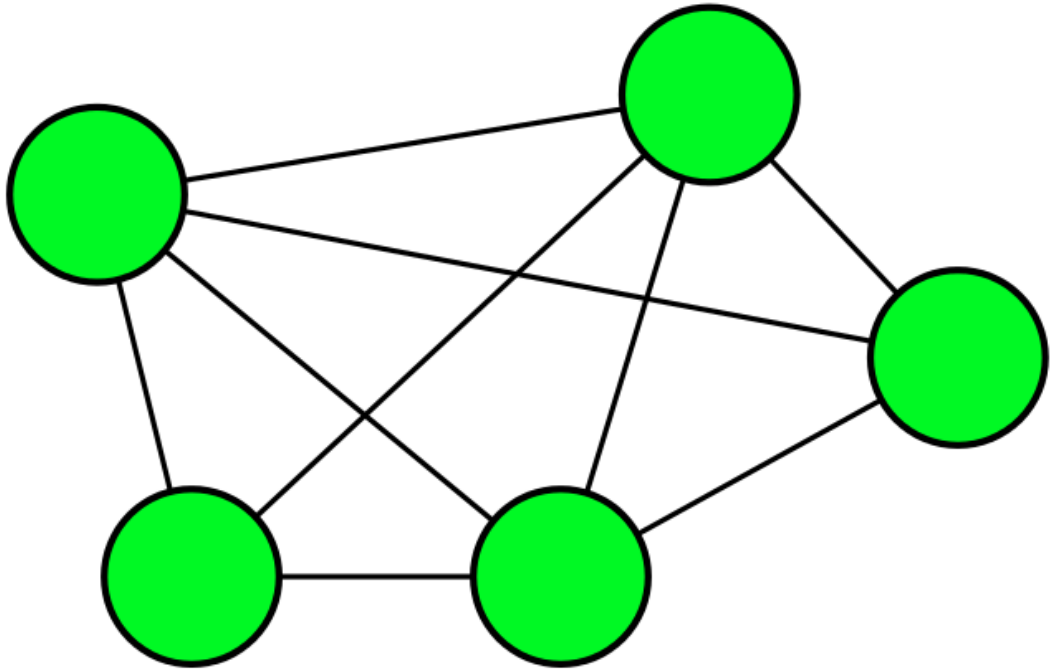
Vilius Luneckas

vilius.luneckas@gmail.com

#3 Ruby Meetup
2013

Turinys

Rails aplikacijos architektūra



Paveikslas pasiskolintas iš Matt Wynne „Hexaonal Rails“ prezentacijos

Rails aplikacijos architektūra

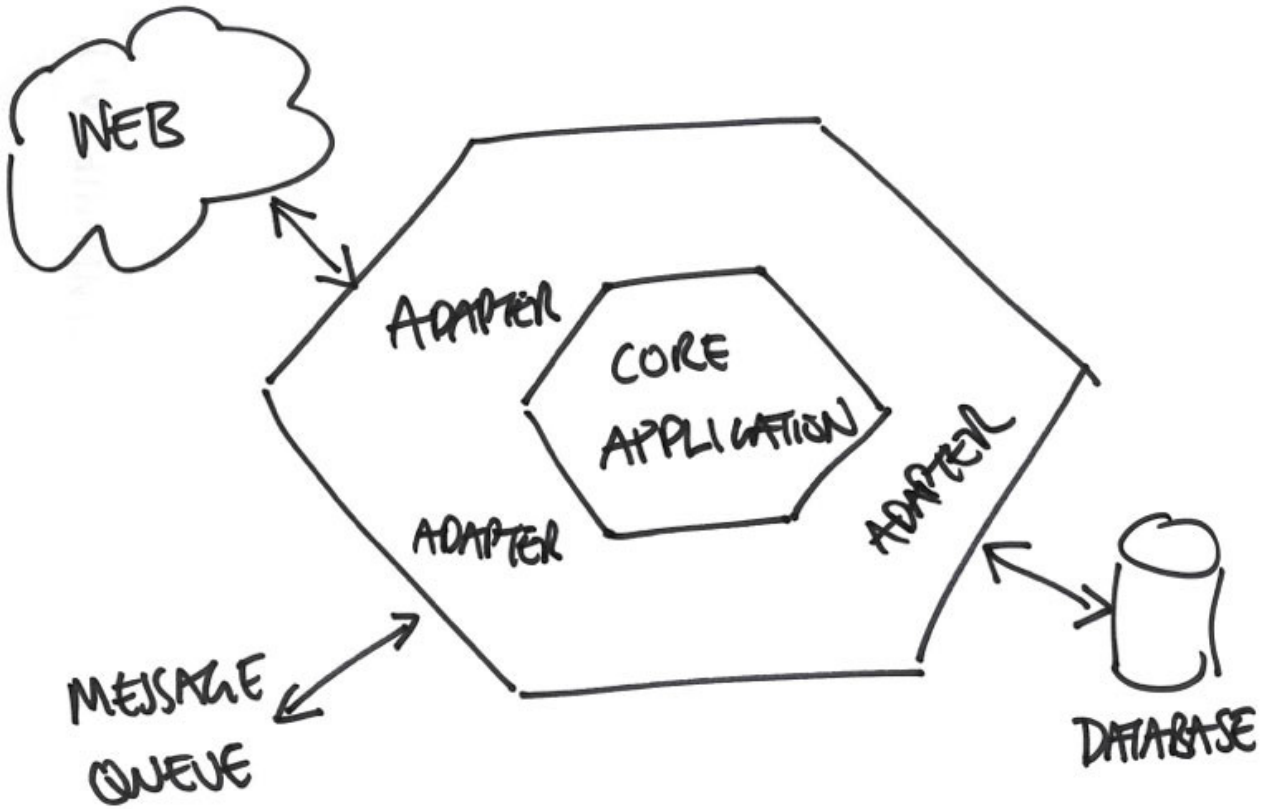
Privalumai:

- projekto pradžioje leidžia greitai kurti naują funkcionalumą;
- viskas yra lengvai pasiekiama.

Minusai:

- bėgant laikui kardinalus funkcionalumo keitimas yra sudėtingas;
- testai lėtėja;
- naujiems testas reikia paruošti vis daugiau susijusių objektų.

Hexagonal Architecture



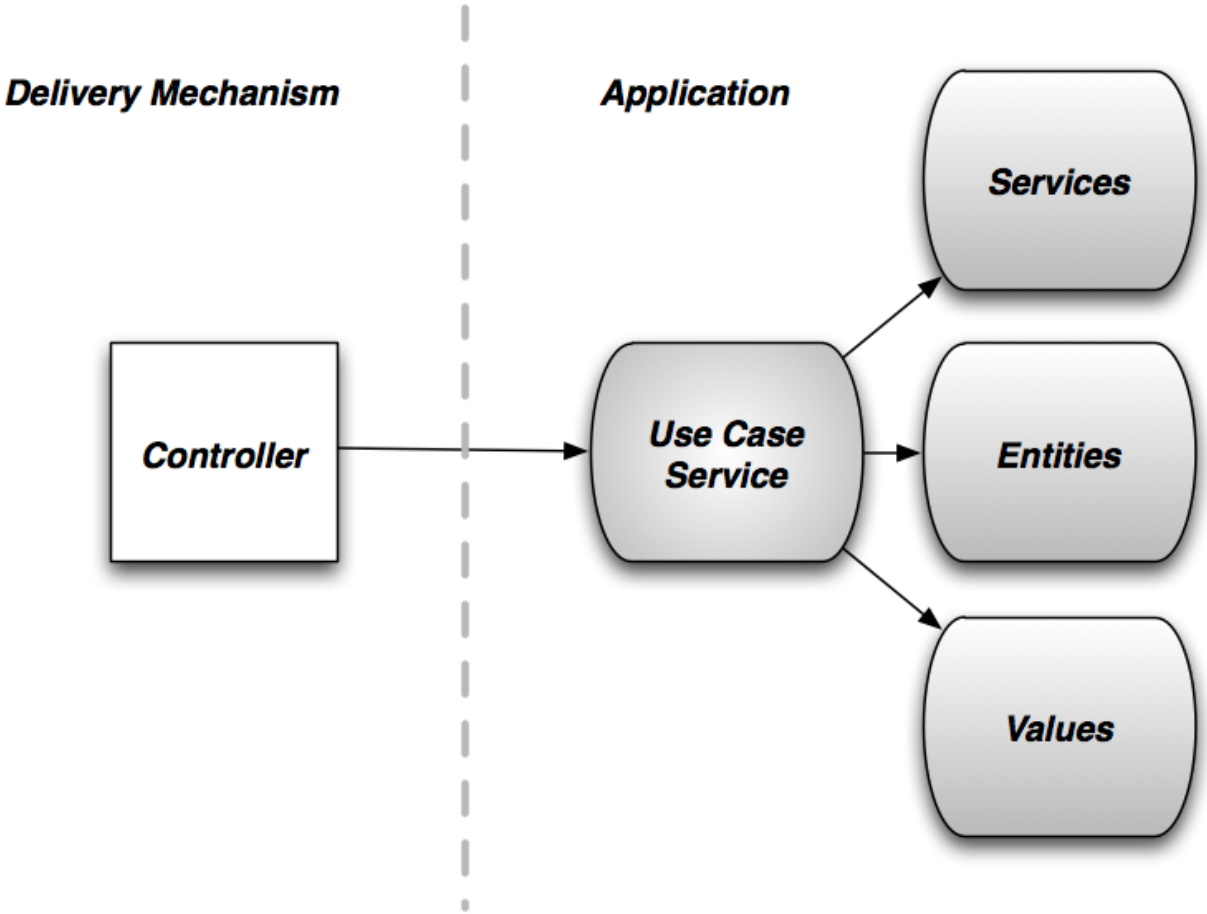
Paveikslas pasiskolintas iš Matt Wynne „Hexaonal Rails“ prezentacijos

Motyvacija

Atskirdami verslo logiką nuo infrastruktūros gauname šiuos privalumus:

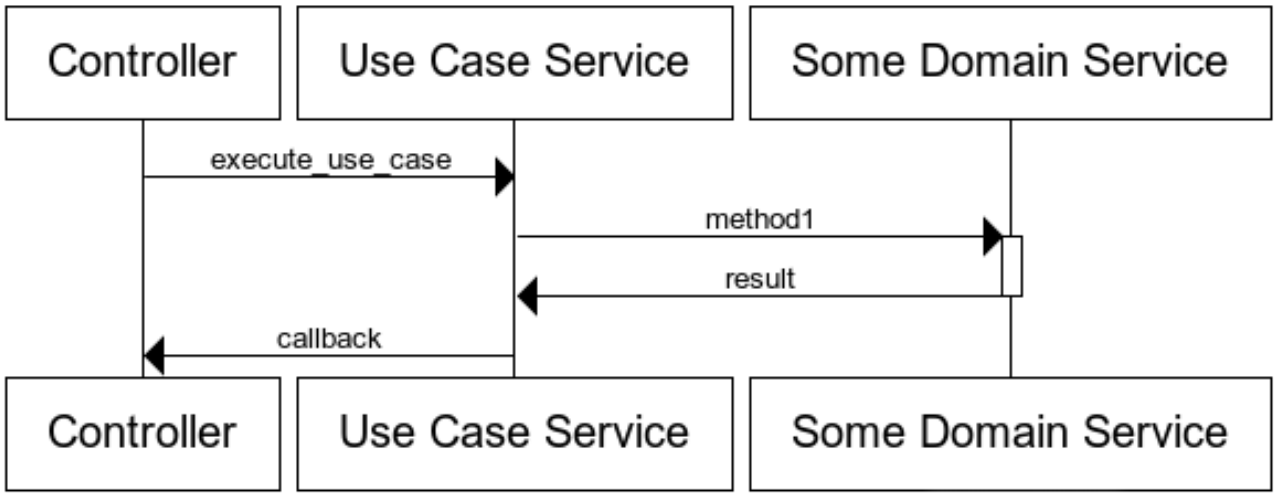
- **Suprantamesnį kodą**
  - Šešiakampio viduje – visas programinis kodas yra skirtas tik verslo logikai. Išgrynintas, suprantamas.
  - Išorėje – kodas rašomas verslo logikos duomenims ir įvykiams perduoti į išorę.
- **Infrastruktūros pakeičiamumą**;
- **Testų greitį**

#1 Use Case Service



Paveikslas pasiskolintas iš Victor Savkin straipsnio („Hexagonal Architecture for Rails Developers“)

#2 Passive Controllers



Paveikslas pasiskolintas iš Victor Savkin straipsnio („Hexagonal Architecture for Rails Developers“)

#3 Repositories
Jei pirmųjų dviejų negana

- ~~ActiveRecord~~ ???
- Objektai, atsakingi už bendravimą su DB
- EDR, git: `nulogy/edr`

Prieš

```
def update
  if params[:iteration]
    @card.iteration = @card.project.find_iteration(params[:iteration])
  end
  if params[:column]
    @card.column = @card.iteration.columns.find_by_name(params[:column])
  elsif @card.column.blank?
    @card.column = @card.iteration.columns.first
  end
  if params[:priority]
    Prioritiser.reprioritise_card(@card, params[:priority])
  end
  if @card.update_attributes(params[:card])
    redirect_to card_path(@card), notice: 'Card saved'
  else
    flash.now[:alert] = 'Ooops, something went wrong'
    render action: 'edit'
  end
end
```

Po

```
class PlanChange < Struct.new(:listener)
  def perform(card, iteration, priority, column_name, attributes)
    assign_iteration(card, iteration)
    assign_column(card, column_name)
    Prioritiser.prioritise_card(card, priority)
    if card.update_attributes(attributes)
      listener.card_update_succeeded(card)
    else
      listener.card_update_failed(card)
    end
  end

  # ... other methods removed for clarity ...
end
```

Po

```
class CardsController

  # ... other methods removed for clarity ...

  def card_update_succeeded(card)
    redirect_to card_path(card), notice: 'Card saved'
  end

  def card_update_failed(card)
    flash.now[:alert] = 'Ooops, something went wrong'
    render action: 'edit'
  end
end
```

Pavyzdys su #3 technika

```ruby
class OrdersController < ApplicationController
#...

  def create
    create_order = CreateOrder.new(OrderRepository, self)
    create_order.create current_user, params[:order]
  end

  def order_creation_succeeded order
    redirect_to order_path(order)
  end

  def order_creation_failed order
    @order = order
    render 'new'
  end

end

# Models
class User
  #...
end

class Order
  #...
end
```

Pavyzdys su #3 technika

```ruby
# Adapters
module OrderRepository
  extend self

  def save order
    DATABASE.put(:order, id, order)
  end
end

# Use Case Service
class CreateOrder
  def initialize order_repository, listener
    @order_repository = order_repository
    @listener = listener
  end

  def create user, params
    order = Order.new(params.merge(user: user))

    if order.valid?
      @order_repository.save(order)
      @listener.order_creation_succeeded order
    else
      @listener.order_creation_failed order
    end
  end
end
```

Kas yra kur?

- *Išoriniai „įrenginiai“* – UI, DB;
- *Adapteriai* – OrderController, OrderRepository;
- *Verslo logika* – User, Order modeliai ir CreateOrder service.

Apibendrinimas

Šis architektūros tipas padeda atskirti verslo logiką nuo infrastruktūros.

- Verslo logikos branduolys nieko nežino apie Ruby on Rails karkasą. Apie verslo transakcijų vykdimą, rezultatus praneša adaptariams.
- Kraštutiniu atveju sistema nieko nežino apie duomenų saugojimo būdą, tam naudoja repozitorijas.

Literatūra

- Refactoring with Hexagonal Rails, *Agile Planner*
- Hexagonal Architecture for Rails Developers, *Victor Savkin*
- Hexagonal Rails: Objects, Values and Hexagons, *Matt Wynne*

?!