Projekt i implementacja usługi internetowej do zarządzania zadaniami zgodnej z REST oraz współpracującej z nią aplikacji mobilnej

Juliusz Gonera

2012-06-09

Założenia projektowe

- REST
- Collection+JSON API
- Clojure
- Podejście funkcyjne
- Natywny klient mobilny (android)
- Postgresql

REST - wymagania

- podział odpowiedzialności wg modelu klient-serwer
- bezstanowość
- pamięć podręczna
- system warstwowy
- standardowy interfejs

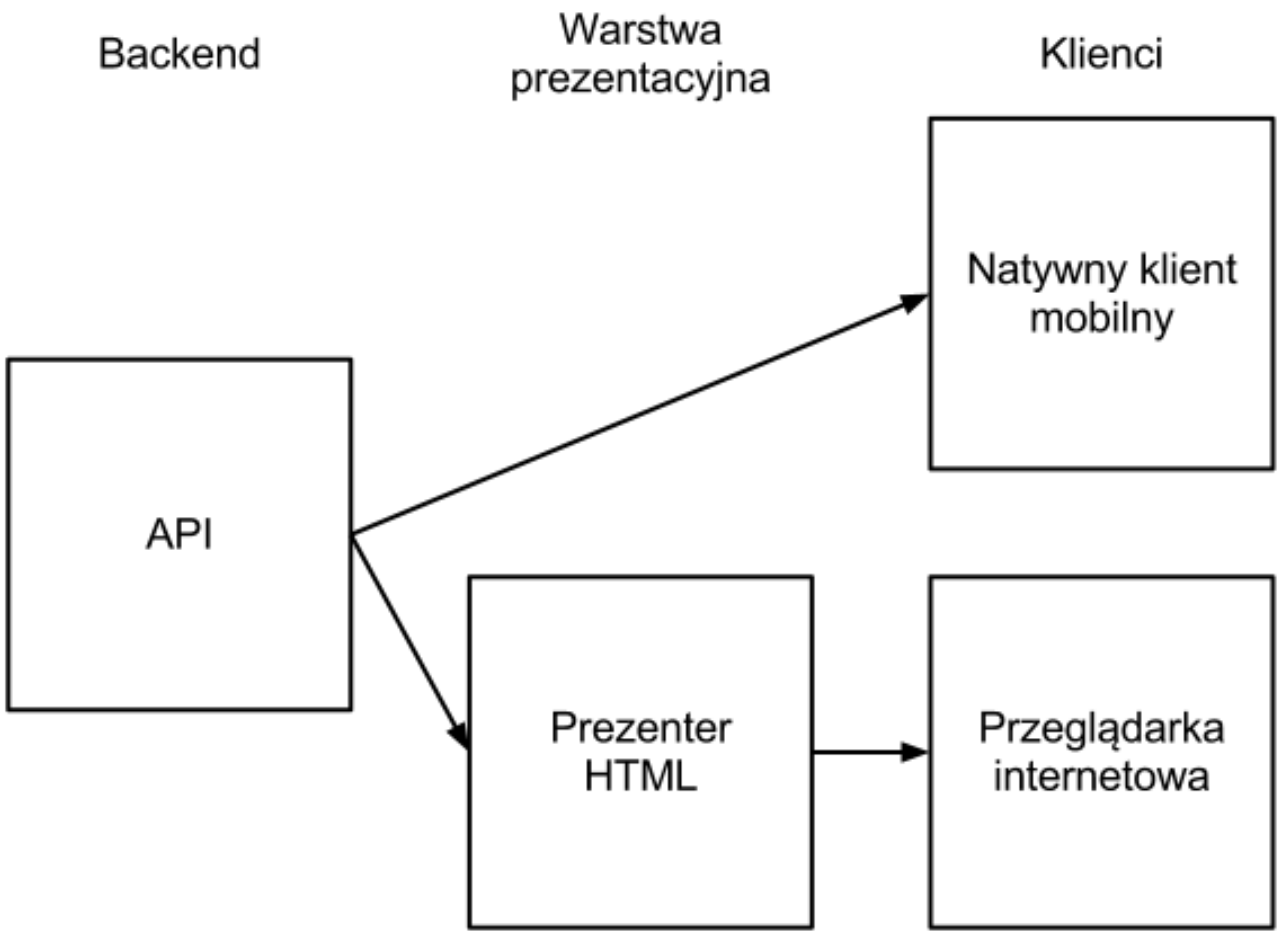
Podział odpowiedzialności

- serwer zawiera logikę aplikacji
- klient jest odpowiedzialny za prezentację danych.
- klient i serwer są rozwijane niezależnie od siebie.

Bezstanowość

- aplikacja nie przechowuje stanu po stronie serwera
- dane potrzebne do zrealizowania zapytania wysyłane za każdym razem
- umożliwia skalowanie horyzontalne systemu

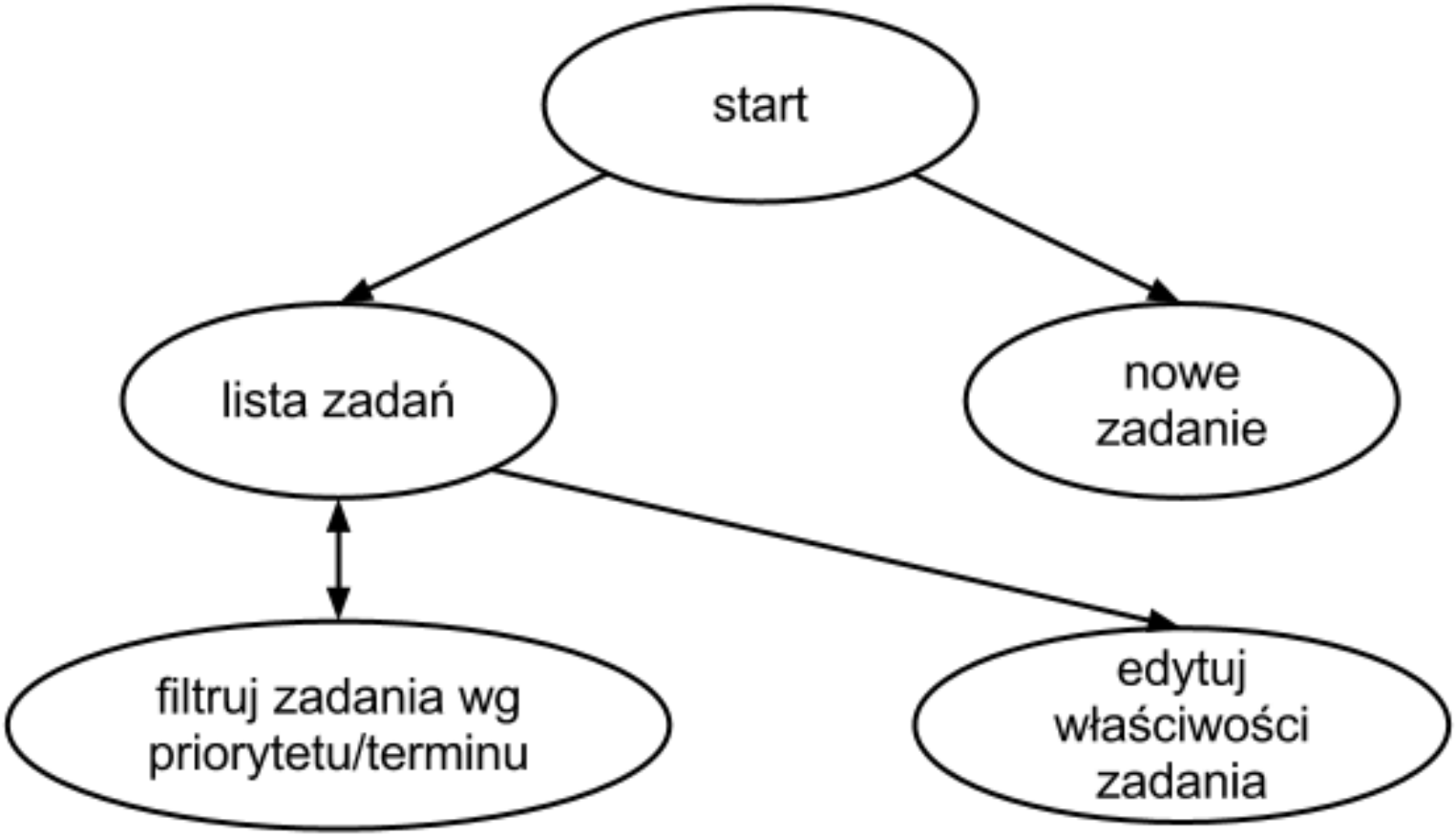
Warstwowy system aplikacji



Standardowy interfejs

- API definiuje zbiór nazw zasobów i operacje
- czasowniki HTTP wykorzystywane jako działania na zasobach
- przejścia stanu aplikacji klienckiej są dokonywane na podstawie opcji oferowanych w odpowiedzi serwera
- klient zna tylko główny URI usługi i zbiór zasobów udostępnianych
- odkrywalność, samodokumentowalność API

Workflow klienta



Zalety REST

- skalowalność
- zmniejszenie zależności między modułami
- odkrywalność, samodokumentowalność API
- łatwe budowanie klientów
- prostota

Wady

- redundancja przesyłanych danych
- model wyklucza przechowywanie stanu sesji po stronie serwera
- nie istnieje formalny standard opisu usług zgodnych z REST

Clojure i podejście funkcyjne

- dialekt LISP
- homoikoniczność, makra
- łatwe testowanie kodu
- bezpieczeństwo wątków
- filozofia - unikanie stanu, programowanie funkcyjne.
- zmiana stanu - sytuacja wyjątkowa

Wybrane funkcjonalności aplikacji

- ustalanie priorytetu zadanie
- działanie offline
- sortowanie wg priorytetu
- wiele list zadań
- synchronizacja list między urządzeniami
- agenda

Koniec

Hypermedia as the Engine of Application State

Collection+JSON

```
{"collection":
  {
    "version": "1.0"
    "href": URI,
    "links": [...],
    "items": [...],
    "queries": [...],
    "template": {OBIEKT},
    "error": {OBIEKT}
  }
 }
```

Konstrukcja zapytań

```
{ "queries" : [
   {"href" : URI,
      "rel" : "all",
      "prompt" : "Wszystkie zadania"},
   {"href": URI,
     "rel": "date-due",
     "prompt": "Zadania z podanym terminem zakończenia",
     "data": [{"name": "termin", "value": "", "prompt": "
   }, ]}
```

```
Zapytanie:
GET URI HTTP/1.1
Host: URI
Content-Type: application/vnd.collection+json
```

Konstrukcja form

```
{ "template":
  { "data": [
    {"name": "full-name", "value": "", "prompt": "Wpisz swo
    {"name": "email", "value": "", "prompt": "Wprowadź adre
  ]}
 }
```

Zapytanie:

```
POST /users/ HTTP/1.1
Host: URI
Content-Type: application/vnd.collection+json
{ "template" : { "data" : [ ...] } }
```

Kolekcja elementów - items

```
    items" :
    [
      {
        "href" : URI,
        "data" : [
{"prompt" : STRING, "name" : STRING, "value" : VALUE},
        ],
        "links" : [ARRAY]
      },
      ...
      {
        "href" : URI,
        "data" : [ARRAY],
        "links" : [ARRAY]
      }
```

Clojure - narzędzia abstrakcji

```
(defmacro validate-presence-of
  [& attributes]
  (let [model (symbol (last (clojure.string/split (str (ns-
    `(if (not (and ~@(map #(list `contains? `~model %) attr
        (throw (Exception. (str "Model " '~model
                                " validates presence of att

;; constructor
(defn new [{:keys [title body] :as task}]
  (validate-presence-of :title :body)
  task)
```