REST-rajapintoja käytännössä

Digitin koodikerho 2021-11-22

Eero Ruohola & Markus Blomqvist

www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

CHAPTER 5

Representational State Transfer (REST)

This chapter introduces and elaborates the Representational State Transfer (REST) architectural style for distributed hypermedia systems, describing the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, while contrasting them to the constraints of other architectural styles. REST is a hybrid style derived from several of the network-based architectural styles described in Chapter 3 and combined with additional constraints that define a uniform connector interface. The software architecture framework of Chapter 1 is used to define the architectural elements of REST and examine sample process, connector, and data views of prototypical architectures.

5.1 Deriving REST

The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture. By examining the impact of each constraint as it is added to the evolving style, we can identify the properties induced by the Web's constraints. Additional constraints can then be applied to form a new architectural style that better reflects the desired properties of a modern Web architecture. This section provides a general overview of REST by walking through the process of deriving it as an architectural style. Later sections will describe in more detail the specific constraints that compose the REST style.

5.1.1 Starting with the Null Style

There are two common perspectives on the process of architectural design, whether it be for buildings or for software. The first is that a designer starts with nothing--a blank slate, whiteboard, or drawing board--and builds-up an architecture from familiar components until it satisfies the needs of the intended system. The second is that a designer starts with the system needs as a whole, without constraints, and then incrementally identifies and applies constraints to elements of the system in order to differentiate the design space and allow the forces that influence system behavior to flow naturally, in harmony with the system. Where the first emphasizes creativity and unbounded vision, the second emphasizes restraint and understanding of the system context. REST has been developed using the latter process. Figures 5-1 through 5-8 depict this graphically in terms of how the applied constraints would differentiate the process view of an architecture as the incremental set of constraints is applied.

The Null style (Figure 5-1) is simply an empty set of constraints. From an architectural perspective, the null style describes a system in which there are no distinguished boundaries between components. It is the starting point for our description of REST.

Figure 5-1. Null Style

5.1.2 Client-Server

The first constraints added to our hybrid style are those of the client-server architectural style (Figure 5-2), described in Section 3.4.1. Separation of concerns is the principle behind the client-server constraints. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains.

Figure 5-2. Client-Server

5.1.3 Stateless

We next add a constraint to the client-server interaction: communication must be stateless in nature, as in the client-stateless-server (CSS) style of Section 3.4.3 (Figure 5-3), such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

Roy Thomas Fielding, University of California Irvine

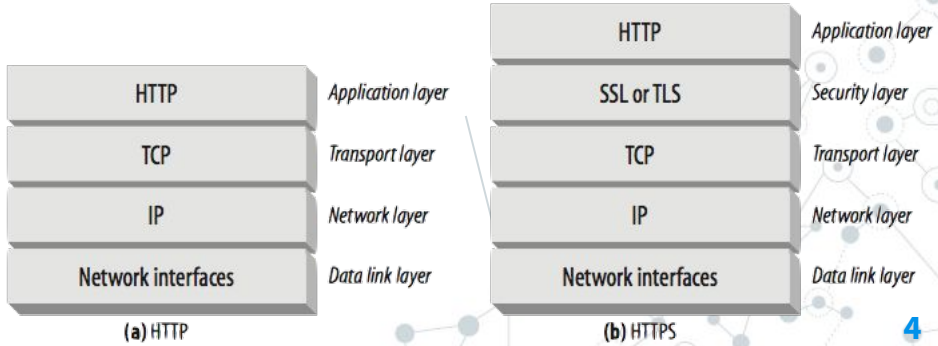
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Ohjelmointirajapinta

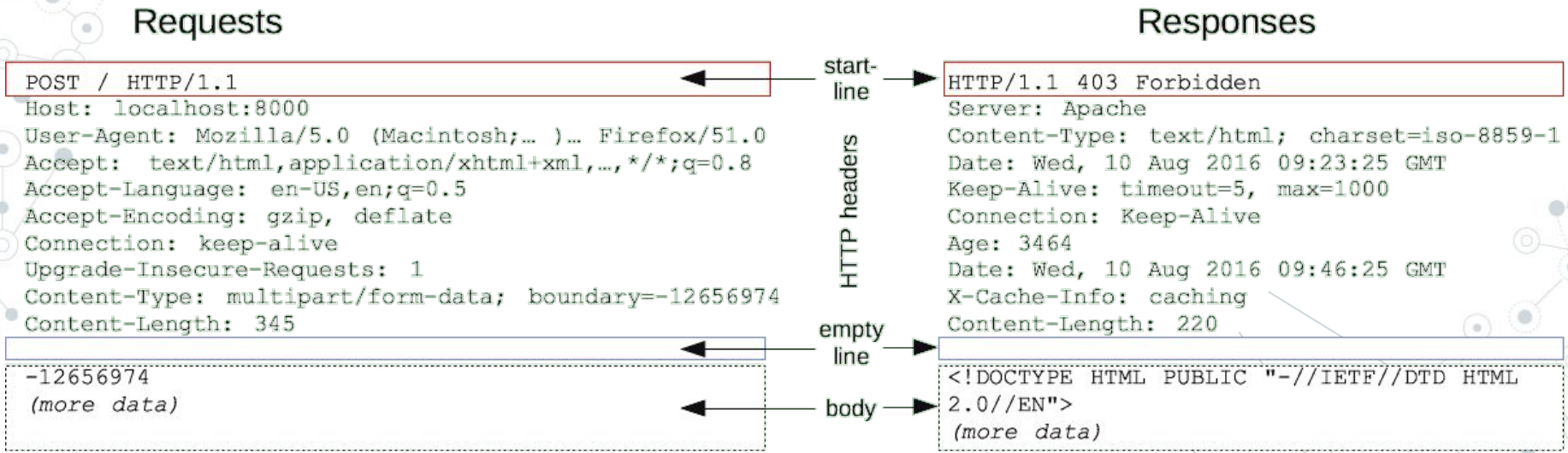
- Ohjelmointirajapinta == Application Programming Interface == API
- Ohjelmointirajapinnat ovat softakomponenttien välillä olevia väyliä, jotka mahdollistavat eri komponenttien kommunikoinnin keskenään.
- Esimerkki: Web-sovelluksen käyttöliittymä hakee tietoa palvelimelta jonkin rajapinnan yli, tämä rajapinta käyttää usein REST-arkkitehtuuria.

Hyper Text Transfer Protocol (HTTP)

- HTTP on verkon sovelluskerroksessa sijaitseva tiedonsiirtoprotokolla.
- Mahdollistaa hypertekstin lähettämisen verkon yli. Hypertekstiä on esimerkiksi selaimen renderöimä HTML.
- Turvallisempi versio HTTPS (=HTTP Secure) nykyään lähes aina käytössä.
- Client ⇄ HTTP ⇄ Server
- (Selain ⇄ HTTP ⇄ Palvelin)



HTTP-viestin rakenne



HTTP-metodit

- POST - Luo resurssin, käytetään myös yleisesti, jos muuta sopivaa metodia ei ole
- GET - Hakee resurssin
- HEAD - Sama kuin GET, mutta ei palauta bodya vaan pelkät headerit
- PUT - Päivittää koko resurssin
- PATCH - Päivittää osan resurssista
- DELETE - Poistaa resurssin
- (OPTIONS) - Kysyy sallitut metodit ja muita tietoja polusta
- (CONNECT) - Pyytää jatkuvan yhteyden avaamista
- (TRACE) - Palauttaa pyynnön sellaisenaan debuggausta varten

Polut

- Kauppalistasovelluksessa on resursseina List-objekteja ja Product-objekteja:
- `/lists` - kaikki kauppalistat
- `/lists/1` - ensimmäinen kauppalista
- `/lists/1/products` - ensimmäisen kauppalistan kaikki tuotteet
- `/lists/1/products/1` - ensimmäisen kauppalistan ensimmäinen tuote

Statuskoodit 1/2

- `1xx` - Pyyntö vastaanotettu, mutta ei vielä käsitelty
  - Harvinaisempia, ei yleensä käytössä
- `2xx` - Onnistunut pyyntö
  - `200` - OK, standardivastaus onnistumiselle
  - `201`- Resurssi luotu
- `3xx` - Sisältö löytyy jostain muualta
  - `301` - Polku on muuttunut eikä vanhaa polkua tule enää käyttää
  - `307` - Väliaikainen uudelleenohjaus, vanha polku edelleen ok

Statuskoodit 2/2

- `4xx` - Virheellinen kysely
  - `400` - Standardivastaus virheelliselle pyynnölle, paremman koodin puutteessa
  - `401` - Kirjautuminen vaaditaan
  - `403` - Ei oikeutta resurssiin
  - `404` - Resurssia ei löytynyt
  - `418` - I’m a teapot
- `5xx` - Virhe palvelimella
  - `500` - Tuntematon virhe
  - `503` - Palvelu ei saatavilla

Headerit

- Sisältävät lisätietoa HTTP-viestistä (kyselystä tai vastauksesta).
- Key-value eli avain-arvo pareja, erottimena kaksoispiste.
- Esimerkki: Selain lähettää `Accept-Language: fi` -headerillä kyselyn mukana käyttäjän kielivalinnan ja palvelin vastaa kyseisen suomenkielisellä käännöksellä resurssista ja asettaa vastausviestiin `Content-Language: fi` -headerin.

Body

- Viestin sisältö tekstinä. Useimmiten jotain rakenteellista kieltä, kuten JSON, HTML tai XML.
- HTML kuvaa verkkosivua.
- JSON tai XML kuvaa mitä tahansa rakenteellista dataa.
- Kuvat yms. binääridata kulkee myös tekstimuotoon muutettuna kyselyn bodyssa.

Request: `GET /lists/1/products`

Response body:

```
[
    {
        "name": "Maito",
        "quantity": 1
    },
    {
        "name": "Leipä",
        "quantity": 1
    },
    {
        "name": "Olut",
        "quantity": 24
    }
]
```

JavaScript Object Notation (JSON)

- Yleisin formaatti datan lähettämiseen kommunikoitaessa rajapintojen kanssa
- Objektit koostuvat key-value -pareista, joissa key on string (merkkijono), ja value mitä tahansa datatyppiä

- Datatyypit:
  - String: `"Olut"`
  - Number: `9001 123.45`
  - Boolean: `true false`
  - Null: `null`
  - Array: `[1, 2, 3]`
  - Object: `{"balance": 1.20}`

Hyvä REST-rajapinta:

- Käyttää loogisia statuskoodeja, polkuja ja HTTP-metodeja
  - Ei: `POST /lists/1/delete-item/1` → Status: `404`
  - Kyllä: `DELETE /lists/1/items/1` → Status: `204`
- Palauttaa informatiivisia virheviestejä
- Palauttaa yhdenmukaista dataa
- On (automaatti) testattu
- On dokumentoitu
- On versioitu oikein
- Ei paljasta ei-julkista tietoa (esim. tietokannan virheitä)
- Varautuu siihen, että kuka tahansa voi kutsua sitä → ei luota siihen, että oma sovellus käyttää sitä aina oikein tai turvallisesti

github.com/DigitKoodit/koodikerho-rest