Hypermedia APIs

Who's that?

- Bastian Krol
- @bastiankrol
- github.com/basti1302
- Consultant at codecentric
- Background: Java, JavaScript, Node.js, Enterprisy stuff, Backends

Disclaimer

I'm not a hypermedia expert!

¯_(ツ)_/¯

Outline

- An example API
- Why does it matter?
- The hypermedia constraint
- REST vs. hypermedia
- Media Types (aka the hypermedia zoo)
- Implementation / server side
- Implementation / client side

The Bank Account API

Root URL: `https://api.example.com/`

`GET /accounts:` **List of all accounts (for which the current user is authorized)**

`GET /accounts/12345:` **Returns the account overview for account 12345**

`GET /accounts/12345/transactions:` **A list of all transactions for this account (paginated)**

`GET /accounts/12345/transactions?offset=100:` **To use pagination, attach the offset parameter to the URL**

`GET /accounts/12345/transactions?amount={amount}&reference={text}&partner={recipient/sender}:` **Search for transactions/filter transactions**

`POST /accounts/12345/transfer:` **Transfer money from your account to someone else**

With Hypermedia:

Root URL: `https://api.example.com/`

… this might be all the documentation you need :D

Example

```
GET /
```

Example (2)

```
GET /
HTTP/1.1 200 OK
Content-Type: application/vnd.mason+json
Link:
<https://api.example.com/profiles/account>;rel="profile"
```

Example (3)

```
GET /
HTTP/1.1 200 OK
Content-Type: application/vnd.mason+json
Link: <https://api.example.com/profiles/account>;rel="profile"

{
  "@links": {
    "self": {
      "href": "/"
    },
    "accounts": {
      "href": "/accounts"
    }
  }
}
```

Example (3)

```
GET /accounts

{
  "accounts": [{
      "account_id": "12345",
      "@links": { "self": { "href": "/accounts/12345" } }
    },
    {
      "account_id": "12346",
      "@links": { "self": { "href": "/accounts/12346" } }
    }]
}
```

Example (4)

```
GET /accounts/12345
{
  "id": "12345",
  "balance": 1302.42,
  "@links": {
    "self": { "href": "/accounts/12345" },
    "transactions": { "href": "/accounts/12345/transactions" }
  },
  "@actions": {
    "transfer": {
      "title": "Transfer money to another account",
      "type": "json",
      "href": "/accounts/12345/transfer",
      "method": "POST",
      "schemaUrl": "/schemas/transfer"
    }
  }
}
```

Example (5)

```
GET /accounts/12345
{
  "id": "12345",
  "balance": -302.42,
  "@links": {
    "self": { "href": "/accounts/12345" },
    "transactions": { "href": "/accounts/12345/transactions" }
  }
}
```

Why Bother?

Hypermedia APIs enable...

- ... less coupling between client and server
- ... evolving APIs
- ... generic clients (media type browsers)
- ... discoverability

Why Bother? (2)

Hypermedia APIs...

- ... can re-use generic client libs
- ... need only very small API-specific clients
- ... are easier to adopt
- ... result in less one-off “standards” – less duplicated efforts

Why Bother? (3)

Hypermedia APIs...

- ... need less out-of-band information (less human readable documentation)
- ... provide more machine readable information directly in the API
- ... tackle the semantic challenge

The Hypermedia Constraint

aka HATEOAS

aka Hypermedia as the engine of application state

Application State → Client

- application state = client state = current location
- only the client keeps application state
- server never tracks application state of clients

The Hypermedia Constraint (2)

URL Space → Server

- server offers links
- client follows links (to change the application state)
- client has no hardcoded URLs (except root URL)

Resource State

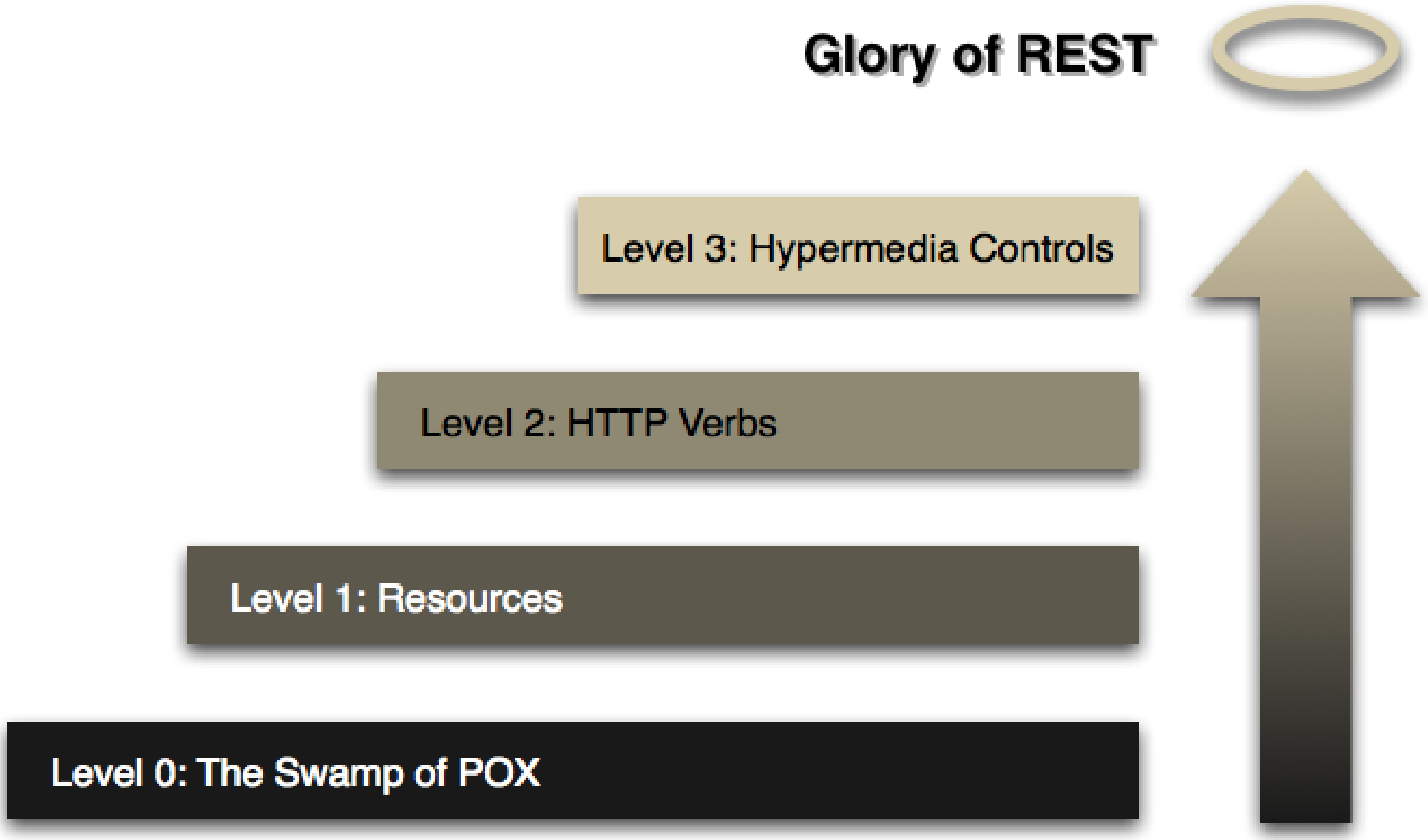
Resource state → Server

- server keeps resource state
- server offers hypermedia controls to change resource state
- client changes resource indirectly
  - by using hypermedia controls
  - by sending representations

REST vs. Hypermedia (Terminology)

- You can do Hypermedia without REST
- You can't do REST without Hypermedia
- Nearly every so-called REST API does not do Hypermedia – and is by definition not a REST API *</pedantic-nitpicking>*
- Unfortunately, the term REST is fubar

Richardson Maturity Model



Trade Offs

- Isn't this more effort than the usual HTTP CRUD API?
- Yes, probably
- Is it worth the effort?
- It depends :-)

Criteria For Using Hypermedia

- Public facing API or internal?
- Multiple clients or only one client?
- Client and Server implemented by the same team, or at least by the same company?
- Is the API expected to be used for years?
- Might the API need to change?
- Can you afford to break existing clients?

Who's Doing It?

- GitHub
- Twitter
- Amazon (AppStream API)
- Netflix
- NPR (PMP)

The Hypermedia Zoo

- HAL
- Collection-JSON
- Mason
- Siren
- Uber

The Hypermedia Zoo (2)

- HTML Microformats
- HTML Microdata
- AtomPub

- ... and more

The Hypermedia Zoo (3)

- Why so many?
- Which one to use?

Implementating Hypermedia

Server Side

- Express
- express-resource
- Restify
- Percolator
- Others: Koa, Hapi, ...

Client Side

Libs for working with media types:

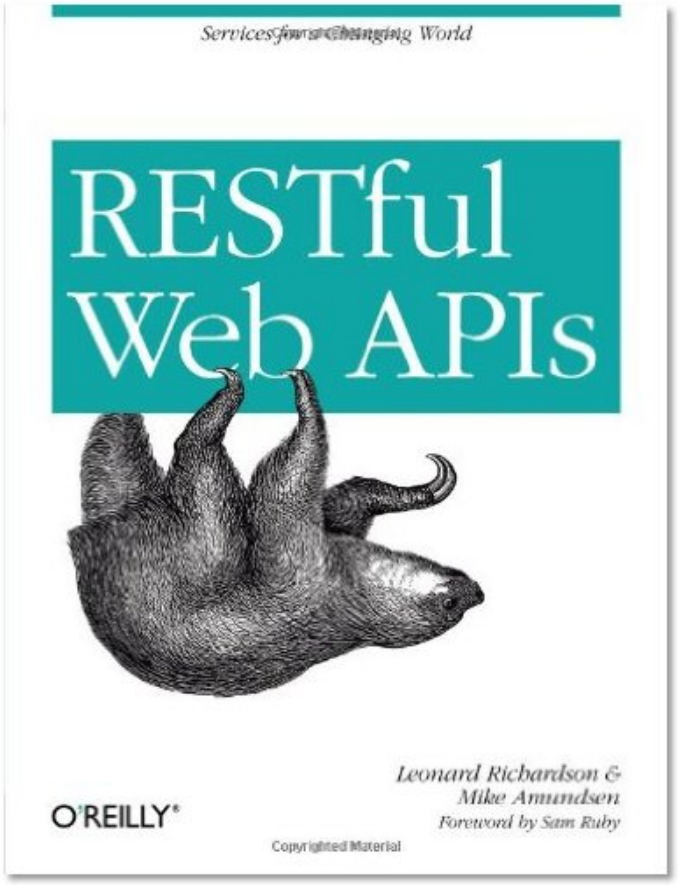
- HAL: Halfred, Halbert, Dave, express-hal, hyperagent
- Siren: siren, siren-writer
- Collection+JSON: collection-json
- Microformats: microformat-node, semantic-schema-parser
- Mason: -
- Uber: -

Client Side (2)

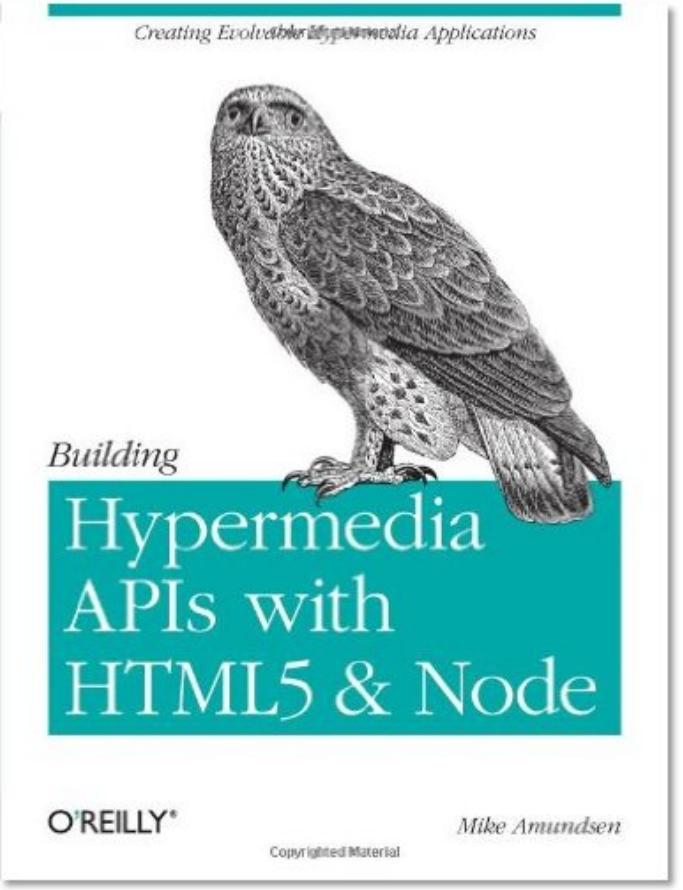
General client libs

- rest.js – helps with request/response lifecycle
- Traverson – follow a path of link relations

Further Reading



Richardson, Amundsen



Amundsen

That's It

Thanks!

Questions?