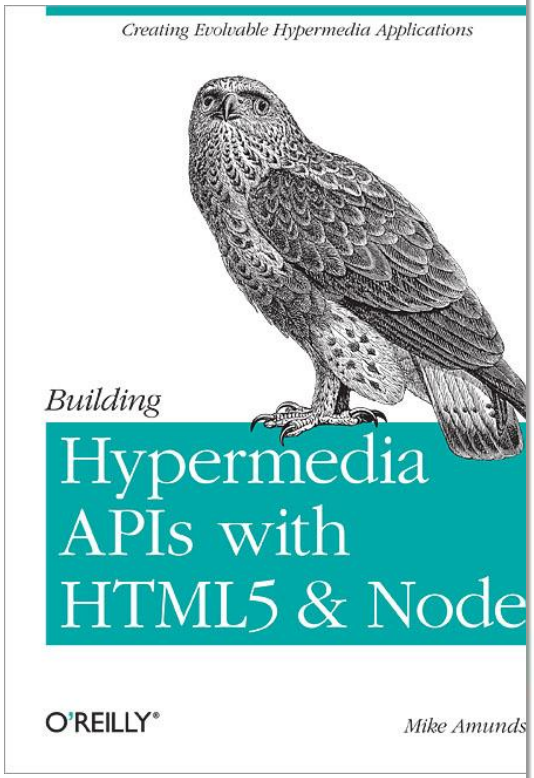
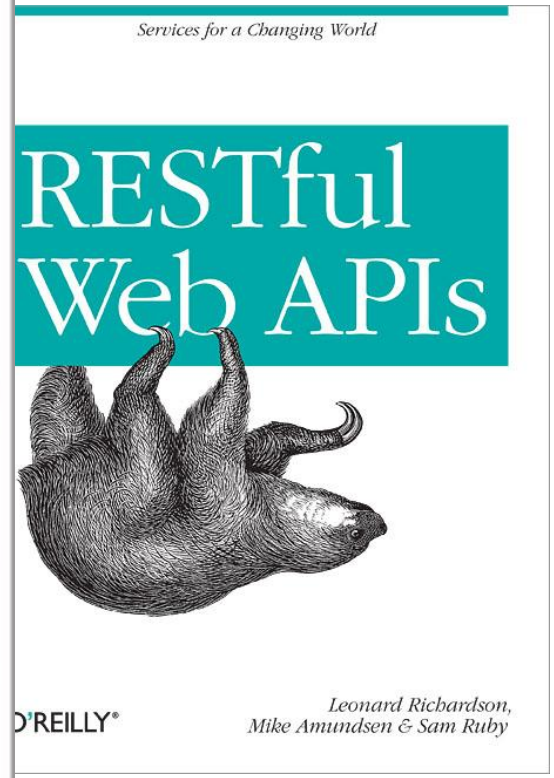

REST and API Styles
Mike Amundsen
API Academy
@mamund


Ctrl
PgUp
Mike Amundsen
@mamund


Creating Evolvable Hypermedia Applications
Building
Hypermedia
APIs with
HTML5 & Node
O'REILLY
Mike Amunds


O'REILLY
RESTful
Web Clients
ENABLING REUSE THROUGH HYPERMEDIA
Mike Amundsen


Services for a Changing World
RESTful
Web APIs
O'REILLY
Leonard Richardson,
Mike Amundsen & Sam Ruby

API and microservices best practices from industry experts

The API Academy provides insights and practical guidance on strategy, architecture and design for APIs and microservices.

Explore

Some of our most popular topics:

API Design
Microservices
API Strategy

Morning Agenda

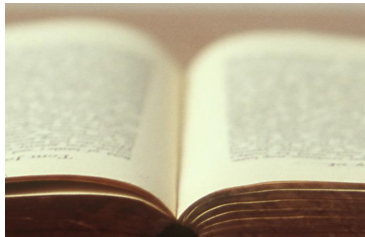
- REST: *a starting point*
- CRUD: *for the masses*
- BREAK
- GraphQL: *for the new kids*
- Hypermedia: *for the dedicated*
- Summary

REST: A Starting Point

REST - The Short Story

Fielding's Dissertation

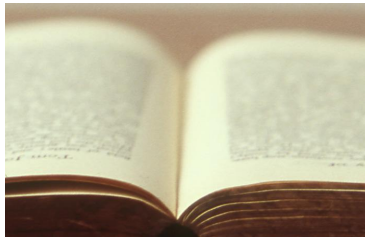
"This dissertation defines a framework for understanding software architecture via architectural styles and demonstrates how styles can be used to guide the architectural design of network-based application software."
- Fielding, 2000



REST in one slide

Properties

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability



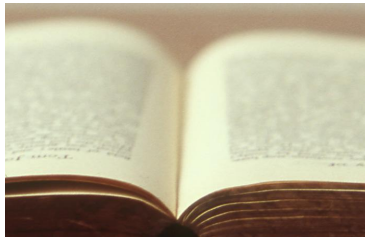
REST in one slide

Properties

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

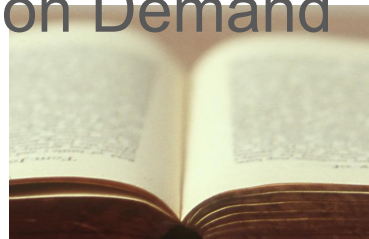
+ Requirements

- Low-Entry Barrier
- Extensibility
- Distributed Hypermedia
- Internet Scale



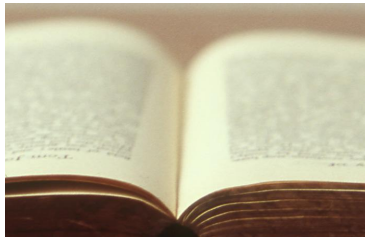
REST in one slide

Properties	**+**	**Requirements**	**=**	**Constraints**
● Performance		● Low-Entry Barrier		● Client-Server
● Scalability		● Extensibility		● Stateless
● Simplicity		● Distributed Hypermedia		● Cache
● Modifiability		● Internet Scale		● Uniform Interface
● Visibility				● Layered System
● Portability				● Code on Demand
● Reliability				



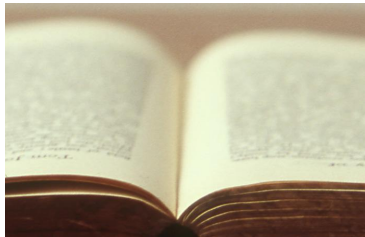
Affordances

"When I say hypertext, I mean the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions." - Fielding, 2008



Affordances

*"When I say hypertext, I mean the simultaneous presentation of information and controls such that **the information becomes the affordance** through which the user (or automaton) obtains choices and selects actions." - Fielding, 2008*



CRUD: For the Masses

HTTP APIs - The Shared Story

RESTful Web Services - 2007

"Our ultimate goal in this book is to reunite the programmable web with the human web. We envision a single interconnected network: a World Wide Web that runs on one set of servers, uses one set of protocols, and obeys one set of design principles."

- Richardson & Ruby, 2008

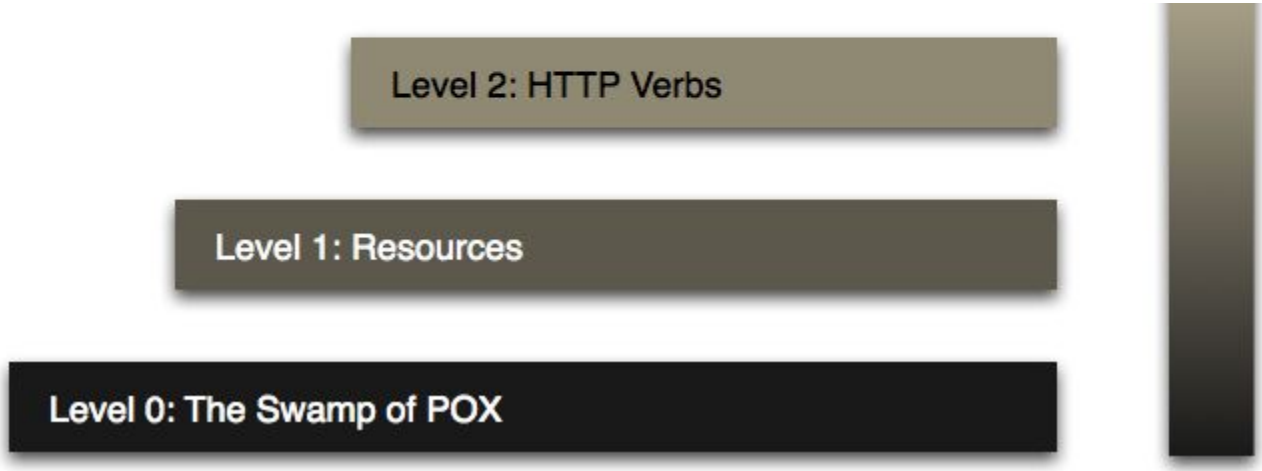
Richardson Maturity Model (RMM)

Richardson Maturity Model (RMM)

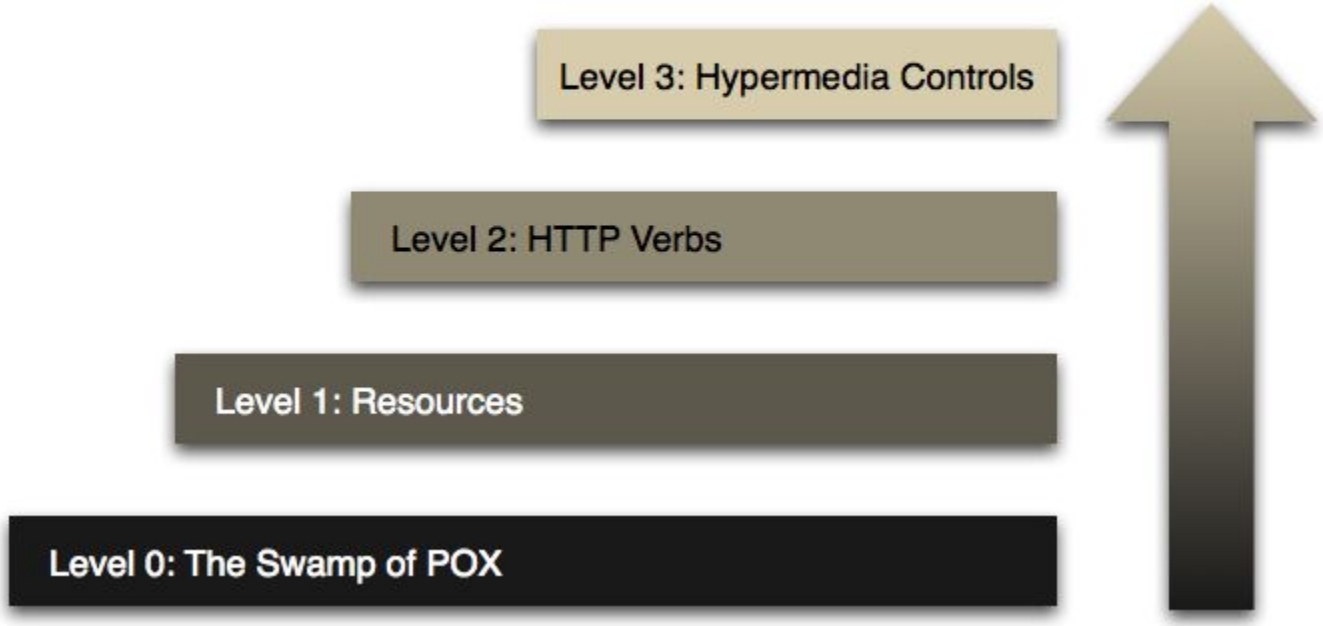
Level 0: The Swamp of POX

Richardson Maturity Model (RMM)

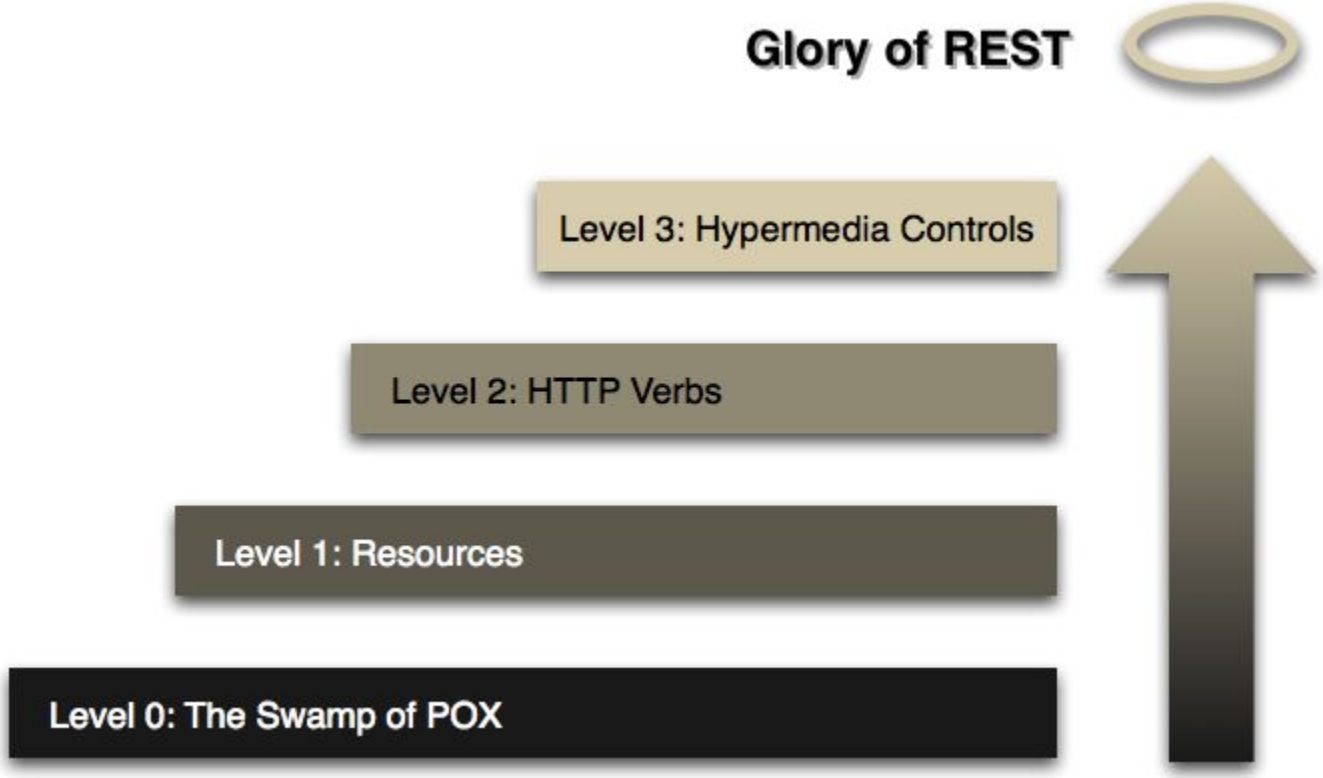
Richardson Maturity Model (RMM)



Richardson Maturity Model (RMM)



Richardson Maturity Model (RMM)



Richardson Maturity Model (RMM)

Level 2: HTTP Verbs

CRUD in one slide

Resource	Method	Representation	Status Codes
Employee	GET	Employee Format	200, 301, 410
Employee	PUT	Employee Format	200, 301, 400, 410
Employee	DELETE	N/A	200, 204
All Employees	GET	Employee List Format	200, 301
All Employees	POST	Employee Format	201, 400

Gregorio's Four Questions (2006)

1. What are the URIs?
2. What are the formats?
3. What methods are supported at each URI?
4. What status codes could be returned?

Common CRUD Guidance

- URI design is the primary task

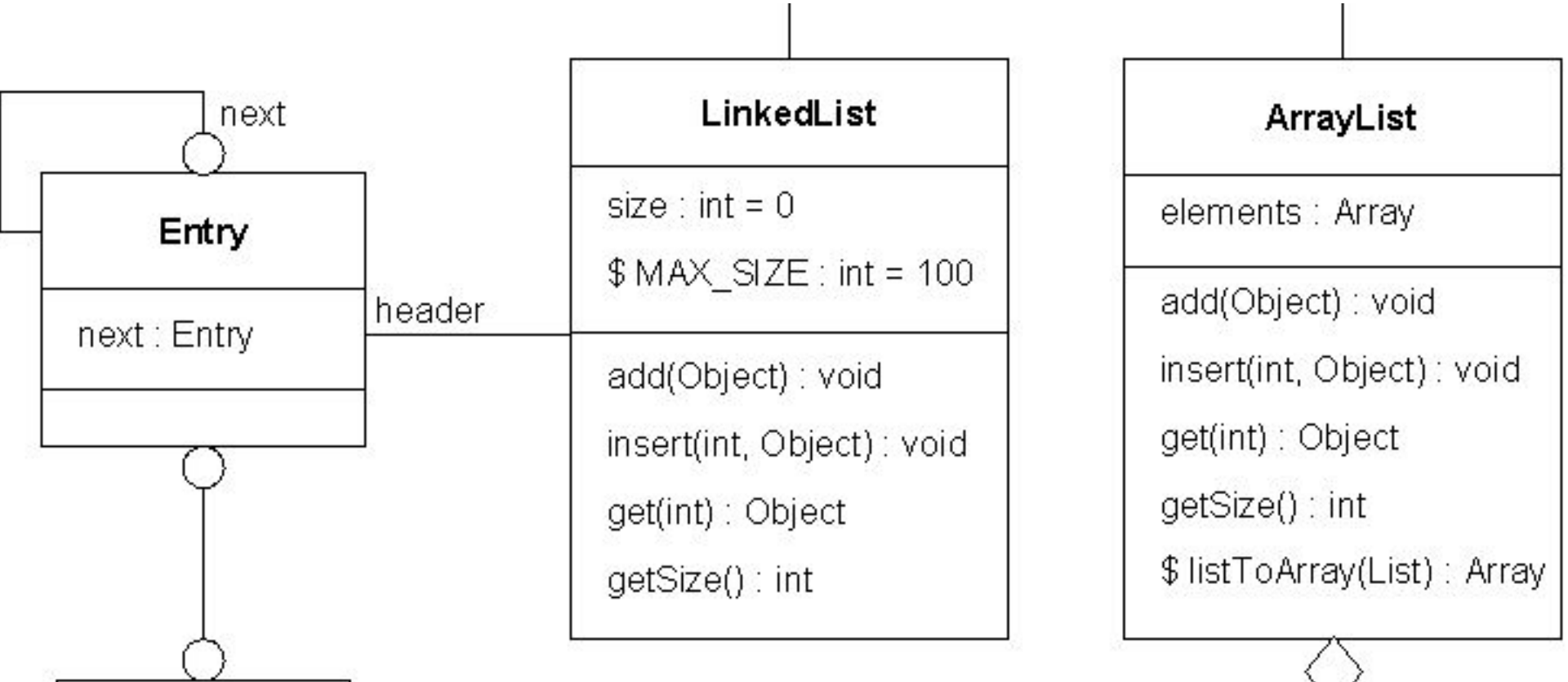
  `http://{server}/{collection}/{id}`
- Focus on serializing domain objects

  `{customer: {name:"mike",...}}`
- Use URIs to express object relationships

  `http://example.com/users/abc/friends/xyz`
- Use controller URIs to handle service tasks

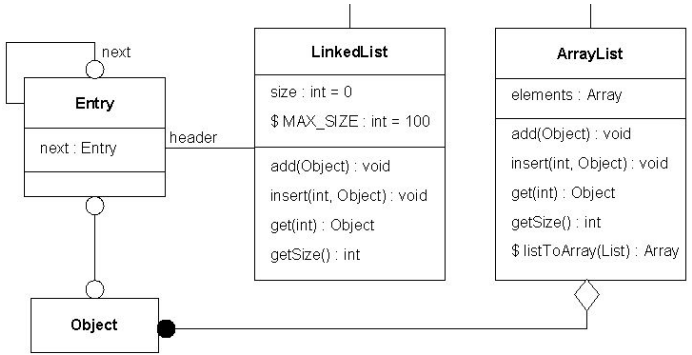
  `http://example.com/reports/1`

URI-Style Clients (CRUD)



URI-Style Clients (CRUD)

- CRUD Server Responses
- HTML SPA Container
- URIs, Objects, and Actions



URI-Style Clients (CRUD)

- CRUD Server Responses

```
- tasks: [
    - {
        id: 0,
        text: "this is some item"
      },
    - {
        id: 1,
        text: "this is another item"
      },
    - {
```

CRUD Server Responses

URI-Style Clients (CRUD)

- HTML SPA Container

```
<body>
  <h1>Tasks CRUD</h1>

  <!-- data goes here -->
  <ul id="list-data"></ul>

  <input id="add" type="button" value="Add" class=
  <input id="search" type="button" value="Search"
  <input id="list" type="button" value="List" clas

</body>
```

URI-Style Clients (CRUD)

- URIs, Objects, and Actions

```
var g = {};
g.msg = {};

g.addUrl = '/tasks/';
g.listUrl = '/tasks/';
g.searchUrl = '/tasks/search?text={@text}';
g.completeUrl = '/tasks/complete/';
```

URI-Style Clients (CRUD)

- URIs, Objects, and Actions

```
// fill in the list
elm = document.getElementById('list-data');
if(elm) {
  elm.innerHTML = '';
  for(i=0,x=g.msg.tasks.length;i<x;i++) {
    li = document.createElement('li');
    li.id = g.msg.tasks[i].id;
    li.title = 'click to delete';
    li.appendChild(document.createTextNode(g.msg.tasks[i].te
    li.onclick = completeItem;

    elm.appendChild(li);
  }
}
```

URI-Style Clients (CRUD)

- URIs, Objects, and Actions

```
// handle "search"
function searchList() {
  var text;

  text = prompt('Enter search:');
  if(text) {
    makeRequest(g.searchUrl.replace('{@text}',encodeURIComponent(tex
  }
}

// handle "add"
function addToList() {
  var text;

  text = prompt('Enter text:');
```

URI-Style Clients (CRUD)

- Composed HTML

```
<ul id="list-data">
  <li id="0" title="click to delete">this is some item</li>
  <li id="1" title="click to delete">this is another item</li>
  <li id="2" title="click to delete">this is one more item</li>
  <li id="3" title="click to delete">this is possibly an item</li>
</ul>
<input id="add" type="button" value="Add" class="button">
<input id="search" type="button" value="Search" class="button">
<input id="list" type="button" value="List" class="button">
```

URI-Style Clients (CRUD)

- JS Summary

```
g.addUrl = '/tasks/';
g.listUrl = '/tasks/';
g.searchUrl = '/tasks/search?text={@text}';
g.completeUrl = '/tasks/complete/';

// prime system
function init() {⟺}

// handle "list"
function refreshList() {⟺}

// handle "search"
function searchList() {⟺}

// handle "add"
function addToList() {⟺}

// handle "complete"
function completeItem() {⟺}

/* parse the returned document */
function showList() {⟺}

function initButtons() {⟺}
function clickButton() {⟺}

// handle network request/response
function makeRequest(href, context, body) {⟺}
function processResponse(ajax, context) {⟺}
```

URI-Style Client

Exercise: Design the CRUD ToDo API

GraphQL: For the New Kids

Query APIs (GraphQL)

Learning GraphQL - 2018

"GraphQL is a query language for your APIs. It’s also a runtime for fulfilling queries with your data. The GraphQL service is transport agnostic but is typically served over HTTP." -- Eve Porcello and Alex Banks

Learning GraphQL - 2018

*"GraphQL is **a query language** for your APIs. It's also a runtime for fulfilling queries with your data. The GraphQL service is transport agnostic but is typically served over HTTP." -- Eve Porcello and Alex Banks*

Learning GraphQL - 2018

*"GraphQL is **a query language** for your APIs. It's also **a runtime** for fulfilling queries with your data. The GraphQL service is transport agnostic but is typically served over HTTP." -- Eve Porcello and Alex Banks*

GraphQL Basics

- Queries
- Schemas
- Resolvers
- Mutations

GraphQL Queries

```
{
 todoSearch(title : "tasks") {
    id, title, completed
 }
}
```

GraphQL Schemas

```
# Todo object

type ToDo {

  # Unique ID of the record

  id: String

  # Title of the record

  title: String

  # Completed flag (true/false)

  completed:Boolean

}
```

GraphQL Resolvers

```
todoStatus : function({completed}) {
  var output = [], item;
  for(var x in storage) {
    if(storage[x].completed==completed) {
      item = {};
      item.id = storage[x].id;
      item.title = storage[x].title;
      item.completed = storage[x].completed;
      output.push(item);
    }
  }
  return output;
},
```

GraphQL Mutations

```
const mutation = new GraphQLObjectType({
  name: 'Mutation',
  fields: {
    create: {
      type: todoType,
      args: {
        id: { type: GraphQLString },
        title: { type: GraphQLString },
        completed: { type: GraphQLInt }
      },
      async resolve (parentValue, { title, body, userId }) {
        const response =
          await fetch('https://jsonplaceholder.typicode.com/todo', {
          method: 'POST',
          body: JSON.stringify({
            id,
            title,
            completed
          }),
          headers: {
            'Content-type': 'application/json; charset=UTF-8'
          }
        })
        const todo = await response.json
        return todo
```

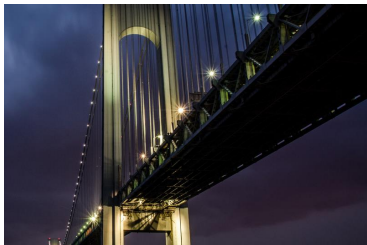
Exercise: Design the GraphQL ToDo API

Hypermedia: For the Dedicated

Hypermedia APIs - The Linked Story

RESTful Web APIs - 2013

*"**RESTful Web Services** covered hypermedia, but it wasn't central to the book. It was possible to skip the hypermedia parts of the book and still design a functioning API. By contrast, **RESTful Web APIs** is effectively a book about hypermedia." - Richardson & Amundsen*



H-Factors (2010)

Link Factors

- LO
- LE
- LT
- LN
- LI

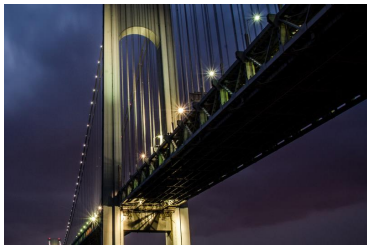
```
<a href="http://www.example.org/search" title="view search page">Search</a>
```

```
<img src="http://www.example.org/images/logo" title="company logo" />
```

```
<form method="get">
  <label>Search term:</label>
  <input name="query" type="text" value="" />
  <input type="submit" />
</form>
```

```
<form method="post" action="http://www.example.org/my-keywords"/>

  <label>Keywords:</label>
  <input name="keywords" type="text" value="" />
  <input type="submit" />
</form>
```

```
function delete(id)
{
  var client = new XMLHttpRequest();
  client.open("DELETE", "/records/"+id);
}
```



H-Factors

Control Factors

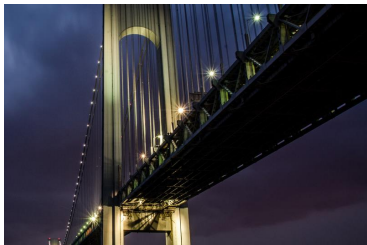
- CR
- CU
- CM
- CL

```
<xsl:include href="http://www.exmaple.org/newsfeed" accept="application/rss" />
```

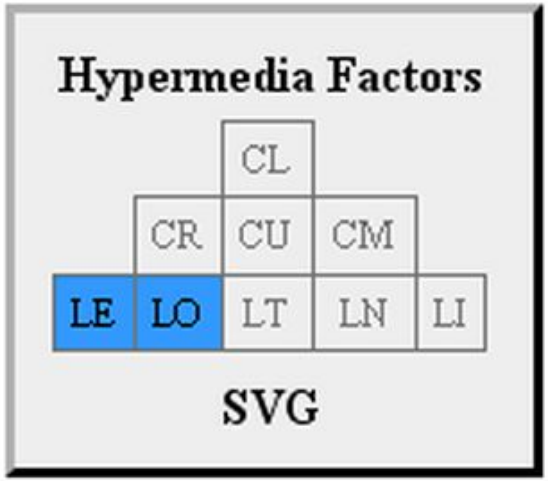
```
<form method="post" action="http://www.example.org/my-keywords" enctype="application/x-www-form-urlencoded" />

  <label>Keywords:</label>
  <input name="keywords" type="text" value="" />
  <input type="submit" />
</form>
```

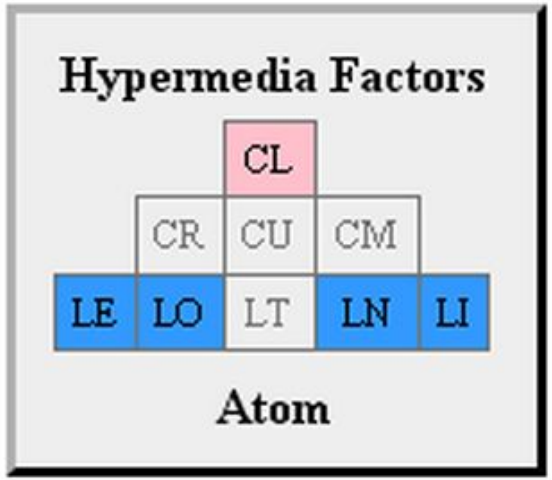
```
<form method="post" action="http://www.example.org/my-keywords" />
  <label>Keywords:</label>

  <input name="keywords" type="text" value="" />
  <input type="submit" />
</form>
```

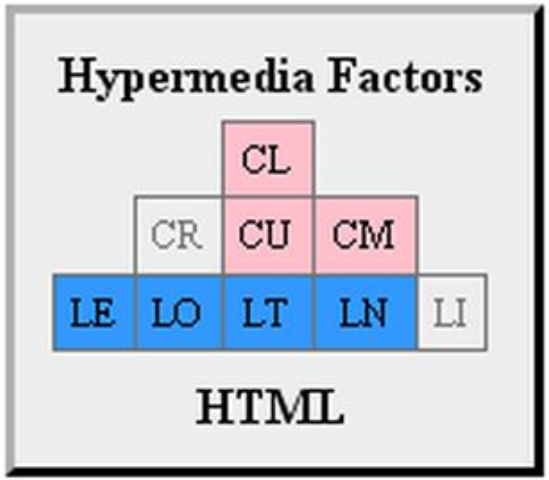
```
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>

  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>

  <content>Some text.</content>
  <link rel="edit" href="http://example.org/edit/first-post.atom"/>
</entry>
```



H-Factors

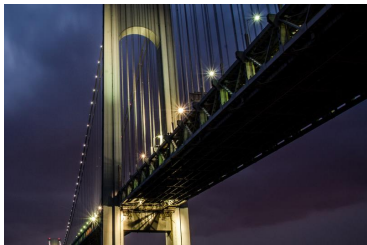
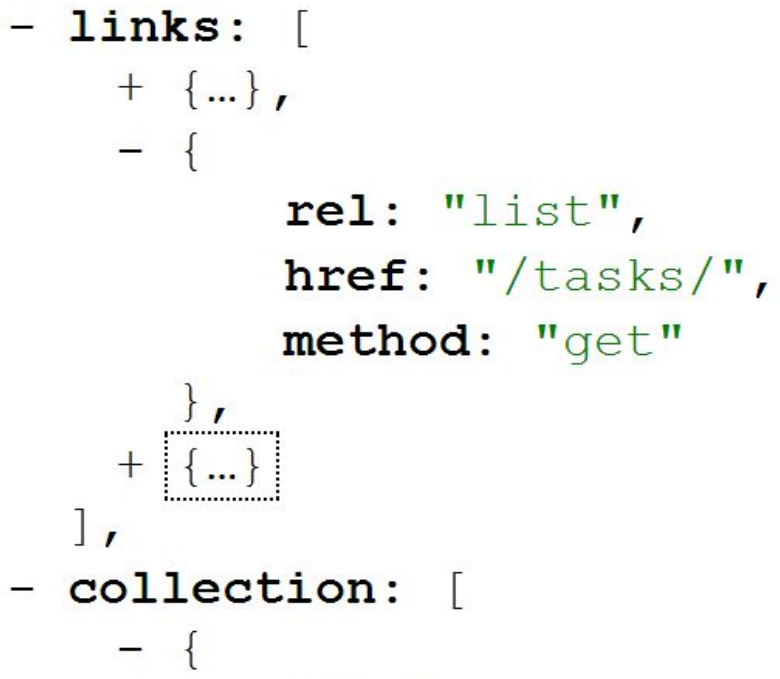

Hypermedia Factors
CL
CR
CU
CM
LE
LO
LT
LN
LI
SVG


Hypermedia Factors
CL
CR
CU
CM
LE
LO
LT
LN
LI
Atom


Hypermedia Factors
CL
CR
CU
CM
LE
LO
LT
LN
LI
HTML

Hypermedia in one slide

- ```
  {"link" :
    {"rel" : "help", "href" : "..."}
  }
  ```
- ```
  {"image" :
    {"rel" : "logo", "href" : "..."}
  }
  ```
- ```
  {"form" :
    {"rel" : "edit",
     "href" : "...",
     "method" : "put",
     "data" : [{"name":"...","value":"..."}]
    }
  ```


  ```

Hypermedia Client (REST)

Hypermedia Client (REST)

- REST Server Responses
- HTML FSM Container
- Media Types and Controls

Hypermedia Client (REST)

- REST Server Responses



REST Server Responses

Hypermedia Client (REST)

- HTML FSM Container

```
<body>
  <h1>Tasks Hypermedia</h1>

  <!-- response data goes here -->
  <ul id="data"></ul>

  <!-- actions go here -->
  <div id="actions"></div>

</body>
<script src="tasks.js" type="text/ja
```

Hypermedia Client (REST)

- Media Types and Controls

```
/* parse the response */
function showResponse() {
  var elm, li, i, x;

  // fill in the list
  elm = document.getElementById('data');
  if(elm) {
    elm.innerHTML = '';
    for(i=0,x=g.msg.collection.length;i<x;i++) {
      li = document.createElement('li');
      li.appendChild(document.createTextNode(g.msg.collection[i]

      // see if we have an affordance here
      try {
        if(g.msg.collection[i].link.rel==='complete') {
          if(g.msg.collection[i].link.data) {
            li.setAttribute('data', g.msg.collection[i].link.dat
```

Hypermedia Client (REST)

- Media Types and Controls

```
// handle possible hypermedia controls
function showControls() {
  var elm, inp, i, x;

  // find and render controls
  elm = document.getElementById('actions');
  if(elm) {
    elm.innerHTML = '';
    for(i=0,x=g.msg.links.length;i<x;i++) {
      inp = document.createElement('input');
      inp.type = "button";
      inp.className = "button";
      inp.id = g.msg.links[i].rel;
      inp.setAttribute('method',g.msg.links[i].method);
```

Hypermedia Client (REST)

- Composed HTML

```
<h1>Tasks Hypermedia</h1>
<!-- response data goes here -->
▼<ul id="data">
  <li data="id" dvalue="0" id="complete" href="/tasks/complete/" method="post">this is some item</li>
  <li data="id" dvalue="1" id="complete" href="/tasks/complete/" method="post">this is another
  item</li>
  <li data="id" dvalue="2" id="complete" href="/tasks/complete/" method="post">this is one more
  item</li>
  <li data="id" dvalue="3" id="complete" href="/tasks/complete/" method="post">this is possibly an
  item</li>
</ul>
<!-- actions go here -->
▼<div id="actions">
  <input type="button" class="button" id="add" method="post" href="/tasks/" value="add" data="text">
  <input type="button" class="button" id="list" method="get" href="/tasks/" value="list">
  <input type="button" class="button" id="search" method="get" href="/tasks/search" value="search"
  data="text">
```

Hypermedia Client (REST)

- Summary JS

```
var thisPage = function() {

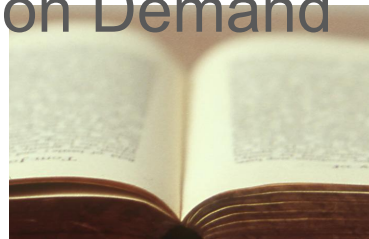
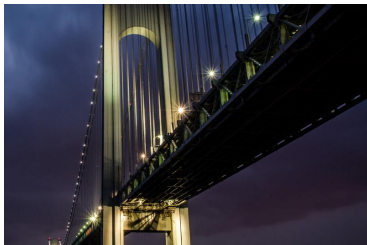
  var g = {};
  g.msg = {};
  g.listUrl = '/tasks/';

  // prime the system
  function init() {⟷}

  /* parse the response */
  function showResponse() {⟷}

  // handle possible hypermedia controls
  function showControls() {⟷}
  function clickButton() {⟷}

  // handle network request/response
  function makeRequest(href, context, body) {⟷}
  function processResponse(ajax, context) {⟷}

  var that = {};
  that.init = init;
  return that;
};
```

Exercise: Design the Hypermedia ToDo API

Summary

REST in one slide

Properties	**+**	**Requirements**	**=**	**Constraints**
● Performance		● Low-Entry Barrier		● Client-Server
● Scalability		● Extensibility		● Stateless
● Simplicity		● Distributed Hypermedia		● Cache
● Modifiability		● Internet Scale		● Uniform Interface
● Visibility				● Layered System
● Portability				● Code on Demand
● Reliability				



CRUD in one slide

- URI design is the primary task

  `http://{server}/{collection}/{id}`
- Focus on serializing domain objects

  `{customer: {name:"mike",...}}`
- Use URIs to express object relationships

  `http://example.com/users/abc/friends/xyz`
- Use controller URIs to handle service tasks

  `http://example.com/reports/1`

GraphQL in one slide

- Queries
- Schemas
- Resolvers
- Mutations

Hypermedia in one slide

- ```
  {"link" :
    {"rel" : "help", "href" : "..."}
  }
  ```
- ```
  {"image" :
    {"rel" : "logo", "href" : "..."}
  }
  ```
- ```
  {"form" :
    {"rel" : "edit",
     "href" : "...",
     "method" : "put",
     "data" : [{"name":"...","value":"..."}]
    }
  ```


  ```

REST and API Styles

- REST Style
  - System Properties and Implementation Constraints
- URI-Style Clients (CRUD)
  - URIs, Objects, and Actions
- Query Clients (GraphQL)
  - Schema, Resolvers, Mutations
- Hypermedia Clients (REST)
  - Media Types and Controls

What's Next?


REST and API Styles
Mike Amundsen
API Academy
@mamund