Instant REST Services With RestExpress

Todd Fredrich

A Little About Me

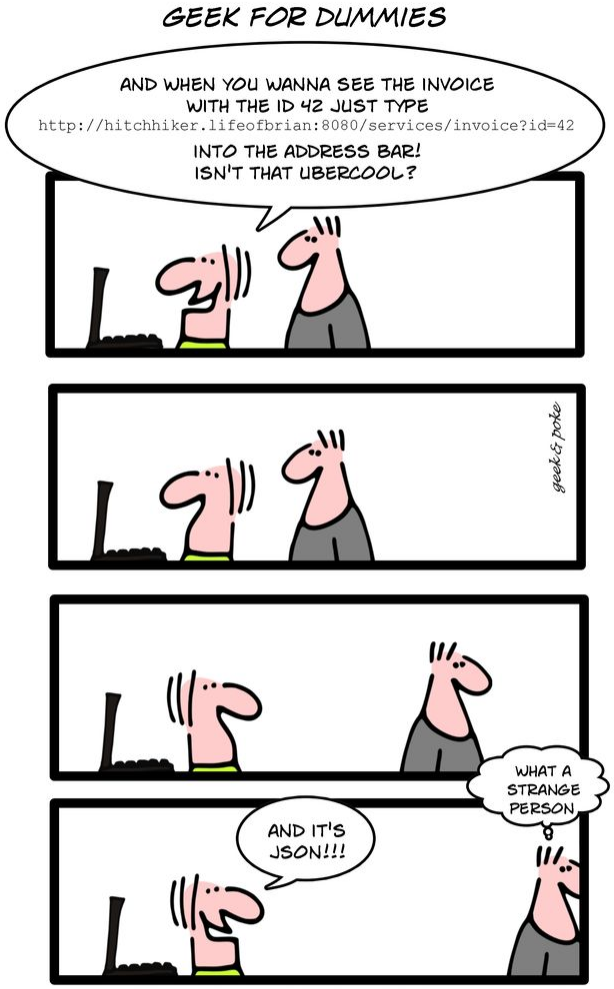
- Todd Fredrich
- Product Architect
- Pearson eCollege

- Java guy since 1998…
  - C/C++ before that

- 10+ years of services experience
  - SOAP over JMS anyone?
  - Axis/Axis2
  - 4+ years REST-ish experience

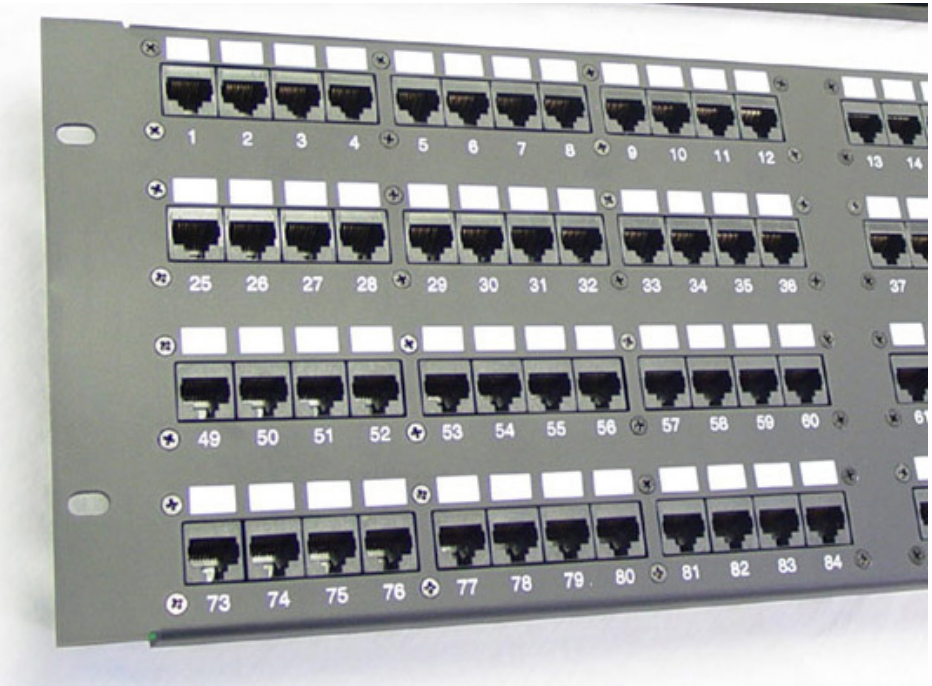
Quick Overview of REST

- REST == "**RE**presentational **S**tate **T**ransfer"
- Simply and architectural style (Roy T. Fielding)
- Embraces the web as it was meant to be used
- Resource-based
- Representations
- Six constraints:
  - Uniform interface
  - Stateless
  - Client-server
  - Cacheable
  - Layered system
  - Code on demand


GEEK FOR DUMMIES
AND WHEN YOU WANNA SEE THE INVOICE WITH THE ID 42 JUST TYPE
http://hitchhiker.lifeofbrian:8080/services/invoice?id=42
INTO THE ADDRESS BAR!
ISN'T THAT UBERCOOL?
geek & poke
WHAT A STRANGE PERSON
AND IT'S JSON!!!
CHAPTER 1: BE AWARE THAT NOT EVERYBODY SHARES YOUR ENTHUSIASM ABOUT RESTFUL APIS

Uniform Interface

- Identification of resources
- Resource manipulation via representations
- Hypermedia as the engine of application state (HATEOAS)
- Self-descriptive messages



Identification of Resources

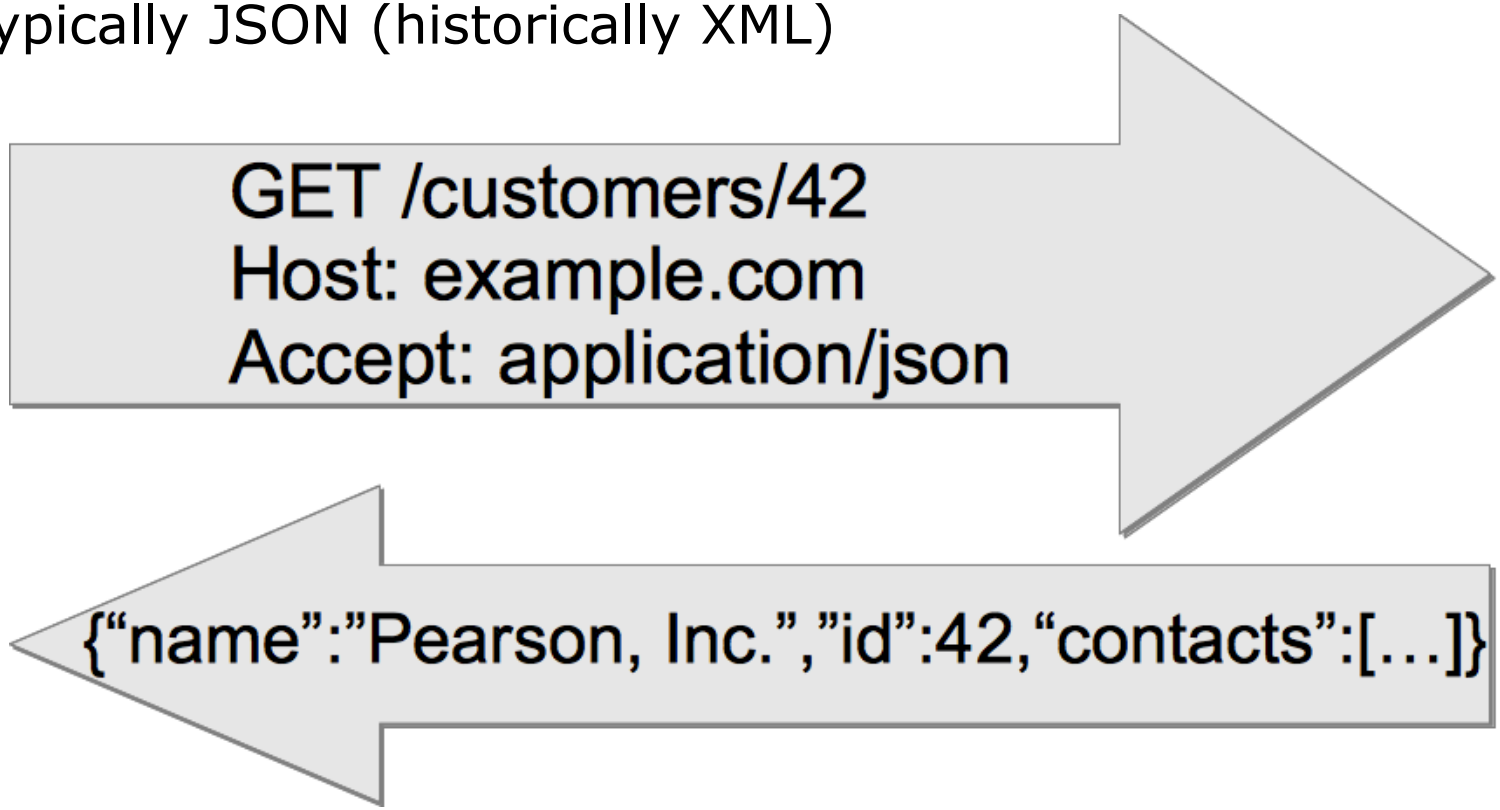
- Identified by URIs
  - Multiple URIs may refer to same resource.
  - Naming URIs is key to usability [BEST PRACTICE]
- Nouns vs. verbs (things vs. actions)

For example:

- http://example.com/customers/42
- http://example.com/customers/42/orders
- http://example.com/customers/42/orders/33
- http://example.com/processes/annual-tax-increase/2012

Resource Manipulation via Representations

- Part of the resource state
- Transferred between client and server
- Typically JSON (historically XML)



Hypermedia As The Engine Of Application State (HATEOAS)

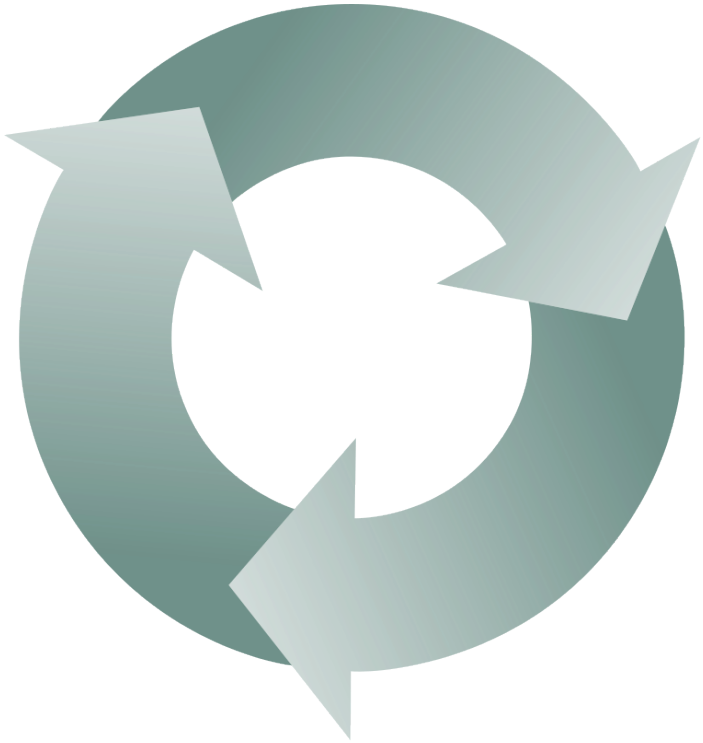
- More advanced model
- Recommend at a minimum, Location header on create:
  - Location: http://example.com/customers/43

Example "Order":

{"id":33,"items_sold":3,"customer":{"href":
"http://example.com/customers/42"},
"links":[
{"rel":"self", "href":
"http://example.com/customers/42/orders/33"},
{"rel":"related","title":"order line-items","href":
"http://example.com/customers/32/orders/33/line-items"}
]}

Self-Descriptive Messages

- Visibility due to:
  - Standard HTTP methods
  - Understandable resource names (URIs)
  - Control data (HTTP headers)
  - Media Types & negotiations
  - Returned data (representations)

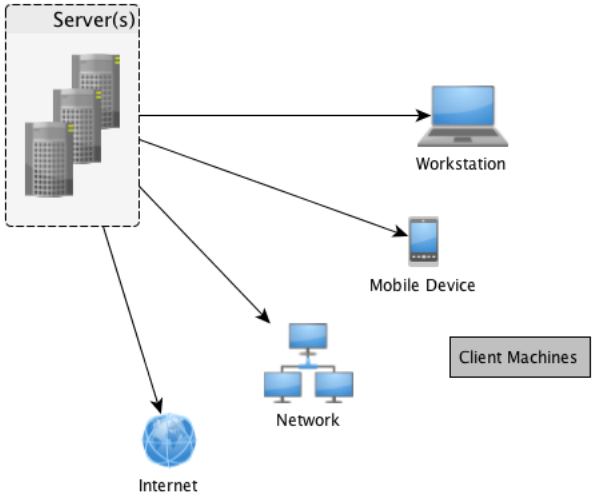


Stateless

- Server contains no client state
- Each request contains enough context to process the message
  - Self-descriptive messages
- Any session state is held on the client

Client-Server

- Assume a disconnected system
- Separation of concerns
- Uniform interface links the two

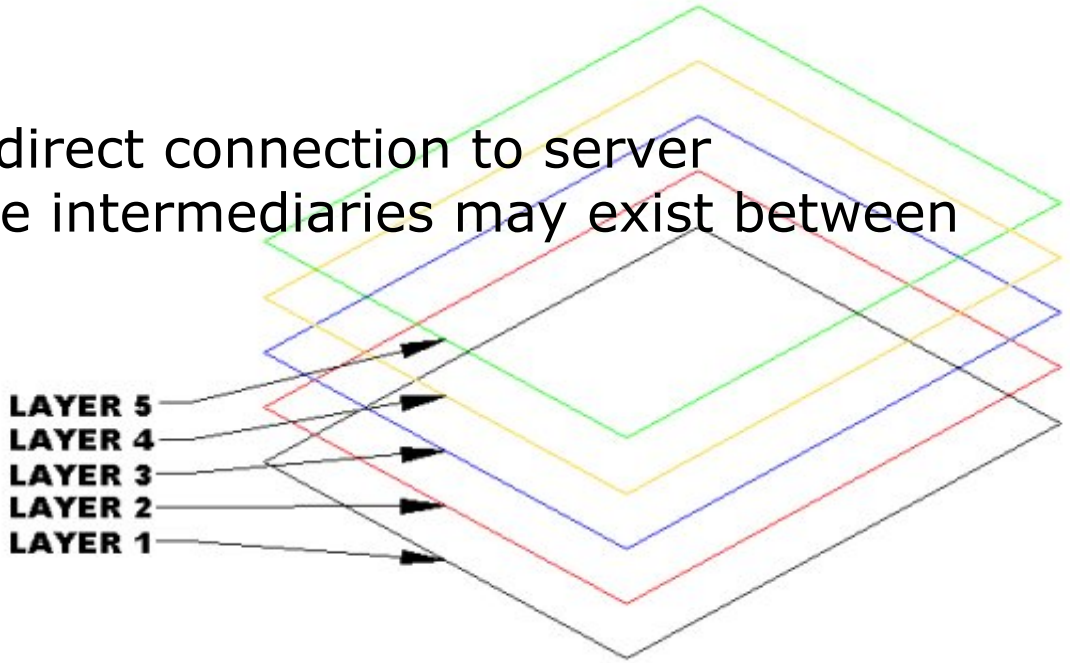


Cacheable

- Server responses (representations) are cacheable
  - Implicitly
  - Explicitly
  - Negotiated

Layered System

- Client can't assume direct connection to server
- Software or hardware intermediaries may exist between client and server
- Improves scalability



Code on Demand

- Server can temporarily extend client
- Transfer logic to client
- Client executes logic
- For example:
  - Java applets
  - Executable JavaScript
- The only optional constraint

Before REST (e.g. SOAP-RPC)

- Customer Management Service
  - addNewCustomer()
  - getCustomerList()
  - getCustomerInfo()
  - updateCustomer()
  - deleteCustomer()

- Order Management Service
  - createOrder()
  - updateOrder()
  - getOrderList()
  - getOrderInfo()
  - cancelOrder()
  - addOrderItem()
  - removeOrderItem()

/customers

POST – create a new customer
GET – get a list of customers
PUT – not used
DELETE – not used

/customers/{customerId}

POST – not used
GET – get a list of customers
PUT – update customer delete
DELETE – remove the identified customer

Order Management

/customers/{customerId}/orders

POST – create a new order
GET – get a list of orders for a customer
PUT – not used
DELETE – not used

/orders

POST – create a new order
GET – get a list of orders
PUT – not used
DELETE – not used

/customers/{customerId}/orders/{orderId}

POST – not used
GET – get order details
PUT – update order details
DELETE – cancel the identified order

/orders/{orderId}

POST – not used
GET – get order details
PUT – update order details
DELETE – cancel the identified order

Line Item Management

/orders/{orderId}/line-items

POST – create a new line item on an order
GET – get a list of line items for an order
PUT – not used
DELETE – not used

/line-items

POST – create a new line item on an order
GET – get a list of line items for an order
PUT – not used
DELETE – not used

/orders/{orderId}/line-items/{lineItemId}

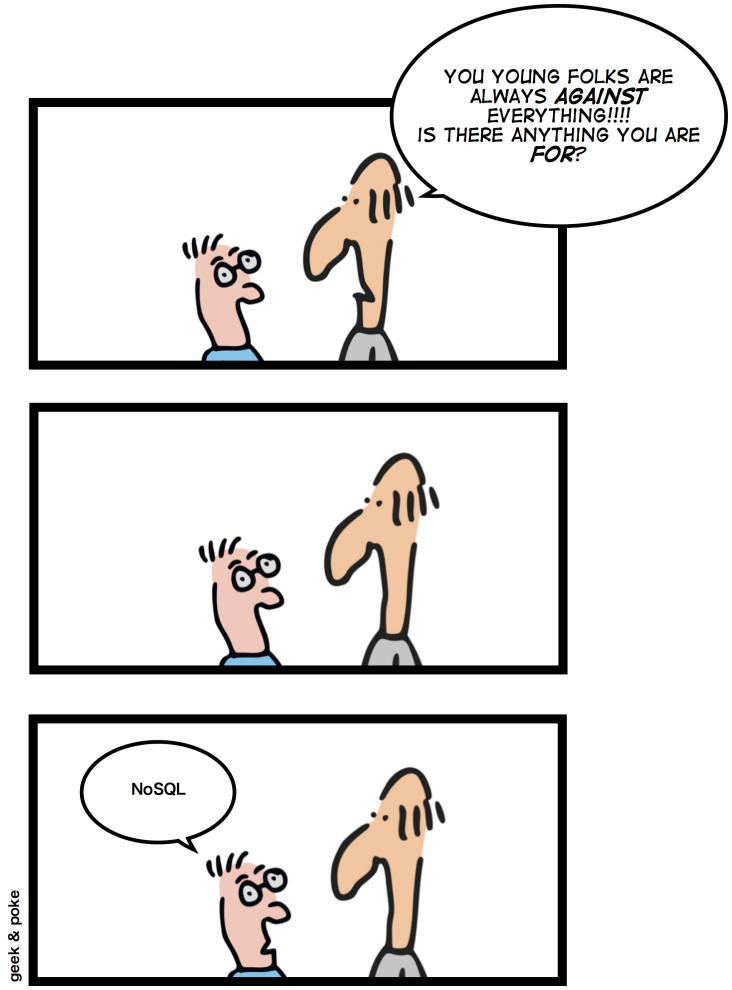
POST – not used
GET – get line item details
PUT – update line item details
DELETE – remove the identified line item

/line-items/{lineItemId}

POST – not used
GET – get line item details
PUT – update line item details
DELETE – remove the identified line item

REST Best Practice Quick Tips

- Use HTTP verbs to mean something
  - GET (read)
  - PUT (update)
  - DELETE (uh… delete/remove)
  - POST (create, plus others)
  - OPTIONS (for documentation)
- Sensible resource names (nouns only)
- JSON (+ XML)
- Fine-grained resources
- Support caching
- Support pagination (limit, offset, with sensible limit default)
- Consider connectedness (HATEOAS)
  - Location headers on create (minimal)
  - Links and pagination


YOU YOUNG FOLKS ARE ALWAYS AGAINST EVERYTHING!!!! IS THERE ANYTHING YOU ARE FOR?
NoSQL
geek & poke

Why NoSQL?

- Scalability
- Options:
  - Key/value – Voldemort, Redis, Riak
  - Document – CouchDB, MongoDB
  - Graph – Neo4J, Titan, FlockDB
  - Column-family – Dynamo, BigTable, Cassandra

MongoDB Strengths

- Easy to install and use
- Searchable/indexable
- Clusters and sharding
- Native BSON storage

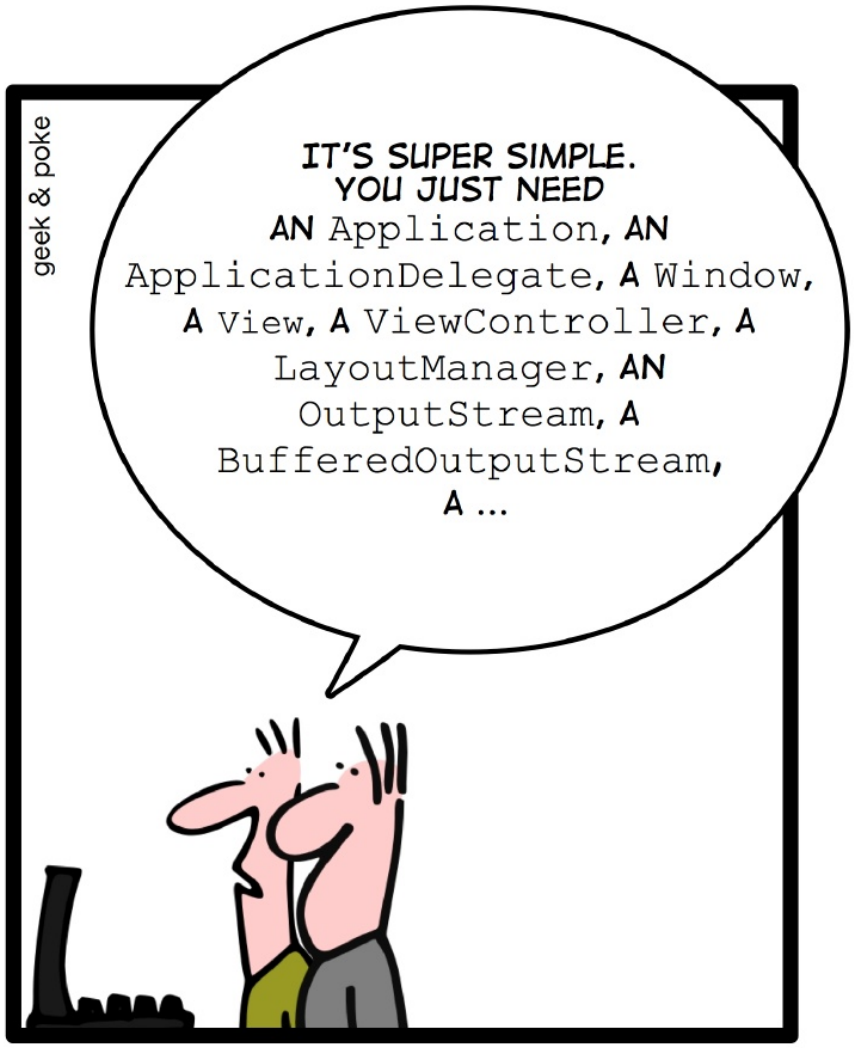
What is RestExpress?

- Simplest thing that could possibly work…
- Java-based REST framework
- Uses Netty NIO framework as its HTTP server
- Convention over configuration
- ISO8601 time point handling in JSON/XML
- Heavily influenced by:

The Default RestExpress Stack

- HTTP: Netty NIO
- JSON Processing: GSON (can easily plugin Jackson)
- XML Processing: XStream (can plugin Jackson)
- O/R Mapping: Morphia
- MongoDB Driver: Mongo Java Driver
- Domain Validation: Syntaxe
- ISO 8601 Date Support: DateAdapterJ
- Intra-JVM Messaging: Domain-Eventing
- Hypermedia Linking: HyperExpress
- RSS/Atom Feed Support: AtomExpress

Available at: https://github.com/RestExpress
Scaffold (kickstart) Application at:
https://github.com/RestExpress/RestExpress-Scaffold/downloads


geek & poke
IT'S SUPER SIMPLE.
YOU JUST NEED
AN Application, AN
ApplicationDelegate, A Window,
A View, A ViewController, A
LayoutManager, AN
OutputStream, A
BufferedOutputStream,
A ...

HELLO WORLD

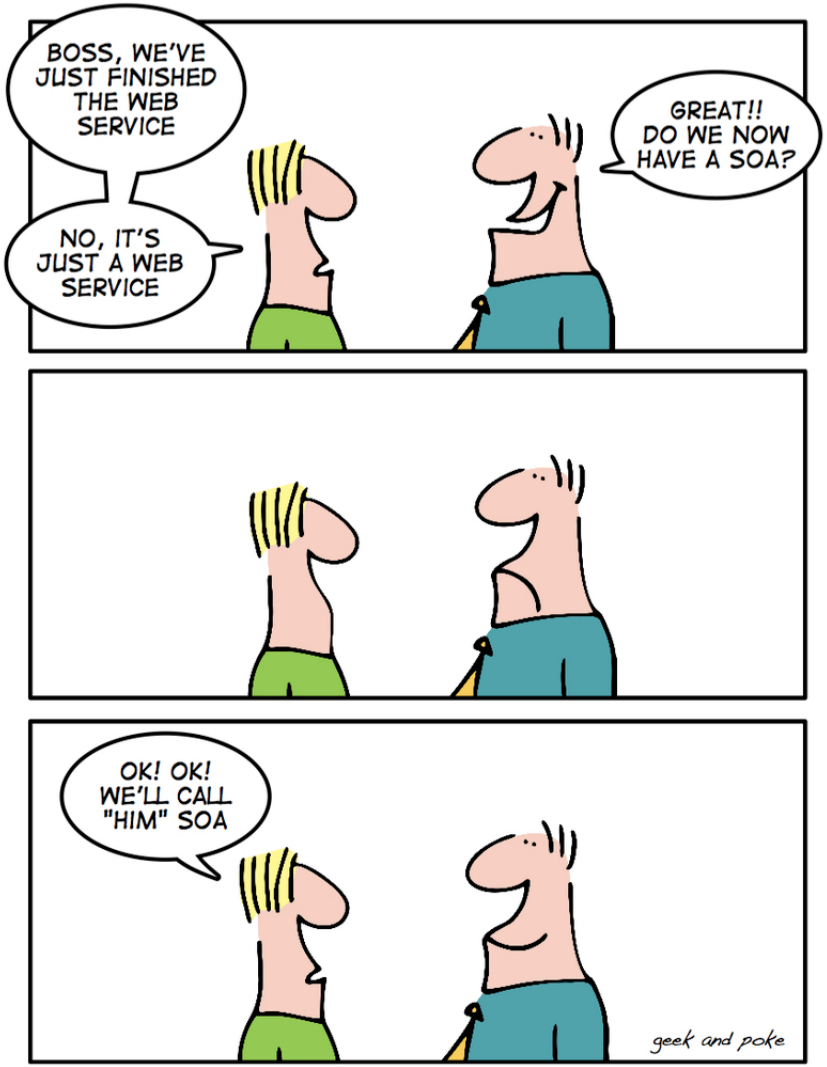
A Minimal RestExpress Server

```
public class Echo {
    public static void main(String[] args) {
        RestExpress server = new RestExpress();

        server.uri("/echo", new Object() {
            public String read(Request request, Response response) {
                String value = request.getRawHeader("echo");
                response.setContentType("text/xml");

                if (value == null) {
                    return "<echo><error>no value specified</error></echo>";
                }
                else {
                    return String.format("<echo><value>%s</value></echo>", value);
                }
            }
        })
        .method(HttpMethod.GET)
        .noSerialization();

        server.bind(8000);
        server.awaitShutdown();
    }
}
```



HOW TO GET A SOA

Our Project: Blogging Service Suite

- Possibly overdone, but easily understood
- New technology stack hard enough

- 3 domain objects: Blog, Entry, Comment
- CRUD in MongoDB
- Domain validation (“requiredness”)
- Error handling (proper HTTP statuses)
- Caching support
- Location header on create
- Pagination
- Sorting & filtering

Maybe Even:

- Relational Links
- Asynchronous cascade-delete

Starting the Project

- Working Example Code at:
  - https://github.com/tfredrich/PTS-2012-Blogging
  - Zip file: https://github.com/tfredrich/PTS-2012-Blogging/downloads
- **Normally:** Download the Scaffolding Project
  - https://github.com/RestExpress/RestExpress-Scaffold/downloads

- Unzip the file into a directory
- Rename the 'kickstart' directory, if desired
- Import into Eclipse IDE
- Install and run MongoDB
- 'ant run' in console (or run as Java App in Eclipse IDE)
- http://localhost:8081/routes/metadata.json
- http://localhost:8081/routes/metadata.xml

Your response should look something like this:

curl -i localhost:8081/routes/metadata

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: 927

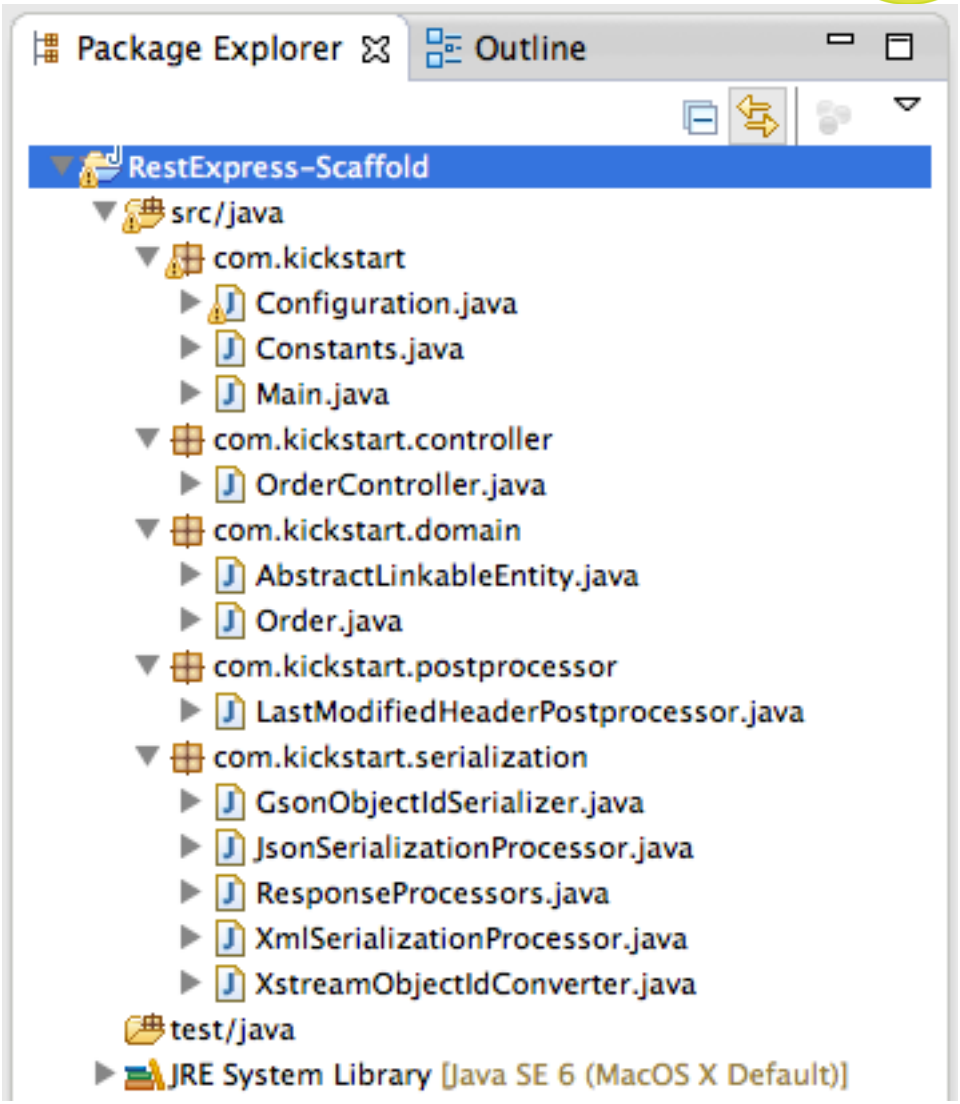
{"name":**"RESTful Kickstart"**,"port":**8081**,"supportedFormats":
["json","xml"],"defaultFormat":"json","routes":[{"uri":{"pattern":**"/orders.
{format}"**,"parameters":["format"]},"defaultFormat":"json","methods":
["POST","GET"],"isSerialized":true},{"name":"KickstartOrderUri","uri":{"pattern":**"/
orders/{orderId}.{format}"**,"parameters":
["orderId","format"]},"defaultFormat":"json","methods":
["GET","PUT","DELETE"],"isSerialized":true},{"name":"allRoutesMetadata","uri":
{"pattern":**"/routes/metadata.{format}"**,"parameters":
["format"]},"defaultFormat":"json","methods":["GET"],"isSerialized":true},
{"name":"singleRouteMetadata","uri":{"pattern":**"/routes/{routeName}/metadata.
{format}"**,"parameters":["routeName","format"]},"defaultFormat":"json","methods":
["GET"],"isSerialized":true},{"name":"routesConsole","uri":{"pattern":**"/routes/
console.html"**},"supportedFormats":["html"],"defaultFormat":"json","methods":
["GET"],"isSerialized":false}]}

Or in XML:

curl -i localhost:8081/routes/metadata.xml

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 2161

<service>
  <name>RESTful Kickstart</name>
  <port>8081</port>
  <supportedFormats>
    <string>json</string>
    <string>xml</string>
  </supportedFormats>
  <defaultFormat>json</defaultFormat>
  <routes>
    <route>
      <uri>
        <pattern>/orders.{format}</pattern>
        <parameters>
          <string>format</string>
        </parameters>
...
```


Package Explorer
Outline
RestExpress-Scaffold
src/java
com.kickstart
Configuration.java
Constants.java
Main.java
com.kickstart.controller
OrderController.java
com.kickstart.domain
AbstractLinkableEntity.java
Order.java
com.kickstart.postprocessor
LastModifiedHeaderPostprocessor.java
com.kickstart.serialization
GsonObjectIdSerializer.java
JsonSerializationProcessor.java
ResponseProcessors.java
XmlSerializationProcessor.java
XstreamObjectIdConverter.java
test/java
JRE System Library [Java SE 6 (MacOS X Default)]

```
public class Main
{
    public static void main(String[] args) throws Exception
    {
        Configuration config = loadEnvironment(args);
        RestExpress server = new RestExpress()
            .setName(config.getName())
            .setDefaultFormat(config.getDefaultFormat())
            .putResponseProcessor(Format.JSON, ResponseProcessors.json())
            .putResponseProcessor(Format.XML, ResponseProcessors.xml())
            .addMessageObserver(new SimpleConsoleLogMessageObserver());

        defineRoutes(config, server);

        new RoutesMetadataPlugin()                          // Support basic discoverability.
            .register(server)
            .parameter(Parameters.Cache.MAX_AGE, 86400);    // Cache for 1 day (24 hours).

        mapExceptions(server);
        server.bind(config.getPort());
        server.awaitShutdown();
    }

    private static void defineRoutes(Configuration config, RestExpress server)
    {
        server.uri("/orders.{format}", config.getOrderController())
            .method(HttpMethod.POST)
            .action("readAll", HttpMethod.GET);

        server.uri("/orders/{orderId}.{format}", config.getOrderController())
            .method(HttpMethod.GET, HttpMethod.PUT, HttpMethod.DELETE)
            .name(Constants.KICKSTART_ORDER_URI)
            .parameter(Parameters.Cache.MAX_AGE, 3600);     // Cache for 3600 seconds (1 hour).
    }
```

Our First Domain Object: Blog

Using the Order-related objects as templates

- Route(s)
  - POST|GET /blogs.{format}
  - GET|PUT|DELETE /blogs/{blogId}.{format}
- Domain object: *Blog*
- Repository (+ Interface?)
- Controller

- Blog data elements
- Validation
- Modeling relationships (e.g. entries)

```java
@Entity("blogs")
public class Blog
extends AbstractMongodbEntity
{
    @StringValidation(name = "Blog Title", required = true)
    private String title;
    private String description;

    public Blog()
    {
    }

    public String getTitle()
    {
        return title;
    }

    public void setTitle(String title)
    {
        this.title = title;
    }

    public String getDescription()
    {
        return description;
    }

    public void setDescription(String description)
    {
        this.description = description;
    }
}
```

What's Next: Entry

- Routes
  - /blogs/{blogId}/entries.{format}
  - /blogs/{blogId}/entries/{entryId}.{format}
- Domain object: Entry
- Controller
- Service (?)
- Repository

- Data elements
- Validation
- Modeling relationships (e.g. comments)

```
@Entity("blog_entries")
public class BlogEntry
extends AbstractMongodbEntity
{
    @Indexed
    @StringValidation(name="Blog ID", required=true)
    private String blogId;

    @StringValidation(name="Title", required=true)
    private String title;

    @StringValidation(name="Entry Content", required=true)
    private String content;

    @Indexed
    @StringValidation(name="Author", required=true)
    private String author;

    public String getBlogId()
    {
        return blogId;
    }

    public void setBlogId(String blogId)
    {
        this.blogId = blogId;
    }

    public String getTitle()
    {
        return title;
    }
```

What's Next: Comment

- Routes
  - /blogs/{blogId}/entries/{entryId}/comments.{format}
  - /blogs/{blogId}/entries/{entryId}/comments/{commentId}.{format}
- Domain object: Entry
- Controller
- Service (?)
- Repository

- Data elements
- Validation

```
@Entity("comments")
public class Comment
extends AbstractMongodbEntity
{
    @Indexed
    @StringValidation(name="Blog Entry ID", required=true)
    private String blogEntryId;

    @StringValidation(name="Author", required=true)
    private String author;

    @StringValidation(name="Comment Content", required=true)
    private String content;

    public String getBlogEntryId()
    {
        return blogEntryId;
    }

    public void setBlogEntryId(String blogEntryId)
    {
        this.blogEntryId = blogEntryId;
    }

    public String getAuthor()
    {
        return author;
    }

    public void setAuthor(String author)
    {
        this.author = author;
    }
```

Pagination Support

Best practices:

- Query-string parameters, *limit* and *offset*
- Or Range header (e.g. 'Range: items=0-19')
- Response header: Content-Range: 0-19/50

QueryRange class

- QueryRange.parseFrom(Request r)
- QueryRange.parseFrom(Request r, int limit)
- QueryRange.asContentRange(int count)

Filtering Support

Best practices:

- Query-string parameter, “filter”
- Name/value separator, double colons (“::”)
- Pair separator, verticle bar (“|”)
- Example:
  - …/customers?filter=city::Denver|country::USA

QueryFilter Class

- QueryFilter.parseFrom(Request r)
- QueryFilter.hasFilters()
- QueryFilter.iterate(FilterCallback c)

FilterCallback Interface

- filterOn(FilterComponent c)

Sorting/Ordering Support

Best practices:

- Query-string parameter, "sort"
- Implied order is 'ascending'
- Descending indicator is a prefix dash ("-")
- sort separator, verticle bar ("|")
- Example:
  - .../orders?sort=customer_name|-total

QueryOrder Class

- QueryOrder.parseFrom(Request r)
- QueryFilter.isSorted()
- QueryFilter.iterate(OrderCallback c)

OrderCallback Interface

- orderBy(OrderComponent c)

Hypermedia Linking

- HyperExpress project
- Classes
  - Link
  - LinkableObject – wrapper for class links
  - LinkableCollection – wrapper for collections

- Interface
  - Linkable – defines the interface for LinkableObject and LinkableCollection

- Helpers
  - LinkUtils
  - MapStringFormat

Error Handling

- RestExpress uses *RuntimeException*
- server.mapException(from<Throwable>, to<ServiceException>)

Mapping Exceptions to HTTP Statuses

- ServiceException → 500
- BadRequestException → 400
- ConflictException → 409
- ForbiddenException → 403
- HttpSpecificationException → 500
- MethodNotAllowedException → 405
- NotFoundException → 404
- UnauthorizedException → 401

Need Another?

- Extend ServiceException
- Call super(<integer http status code>) in constructor.

Wrapped Responses

Why?

- AJAX (browser) clients
- Error conditions

What?

- Wrap response data in envelope
- JSEND
- Success: {“status”:”success”,”code”:201,”data”:”<json>”}
- Error: {“status”:”error”,”code”:400,”data”:”BadRequestException”, “message”:”Could not parse JSON input”}
- Failure: {“status”:”fail”,”code”:500,”data”:”NullPointException”}

How?

- server.putResponseProcessor(String format, ResponseProcesor rp)
- ResultWrapper.fromResponse(Response r)

RestExpress Stack Links

- https://netty.io/
- http://code.google.com/p/google-gson/
- http://xstream.codehaus.org/
- http://code.google.com/p/morphia/)
- https://github.com/mongodb/mongo-java-driver/downloads
- https://github.com/RestExpress/Syntaxe
- https://github.com/tfredrich/DateAdapterJ
- https://github.com/tfredrich/Domain-Eventing
- https://github.com/RestExpress/HyperExpress

- https://github.com/RestExpress
- https://github.com/RestExpress/RestExpress-Scaffold

Additional Resources:

- *REST API Design Rulebook*, Mark Masse, 2011, O'Reilly Media, Inc.
- *RESTful Web Services*, Leonard Richardson and Sam Ruby, 2008, O'Reilly Media, Inc.
- *RESTful Web Services Cookbook*, Subbu Allamaraju, 2010, O'Reilly Media, Inc.
- *REST in Practice: Hypermedia and Systems Architecture*, Jim Webber, et al., 2010, O'Reilly Media, Inc.
- *Service Design Patterns*, Robert Daigneau, 2012, Pearson Education, Inc.
- *SOA with REST*, Thomas Erl, et. al., 2013, SOA Systems Inc.
- *NoSQL Distilled*, Pramod J. Sadalage and Martin Fowler, 2013, Pearson Education, Inc.
- http://www.RestApiTutorial.com/
- https://github.com/RestExpress