Tech Stack Boot Camp
Day 1

REST, RestExpress, Eventing & SubPub

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

REST

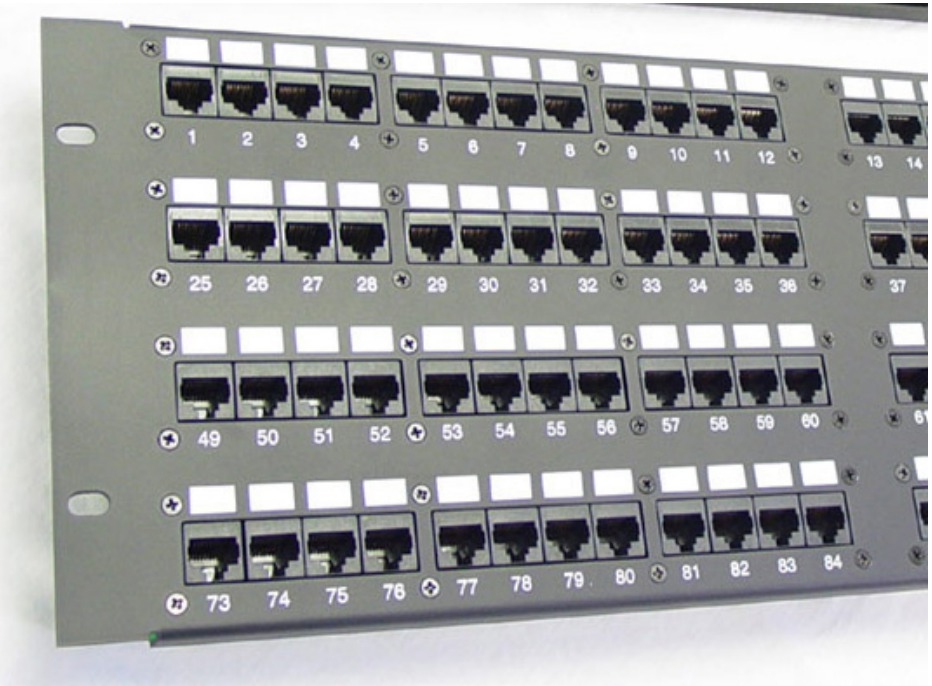
http://www.RestApiTutorial.com

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

Uniform Interface

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



Identification of Resources

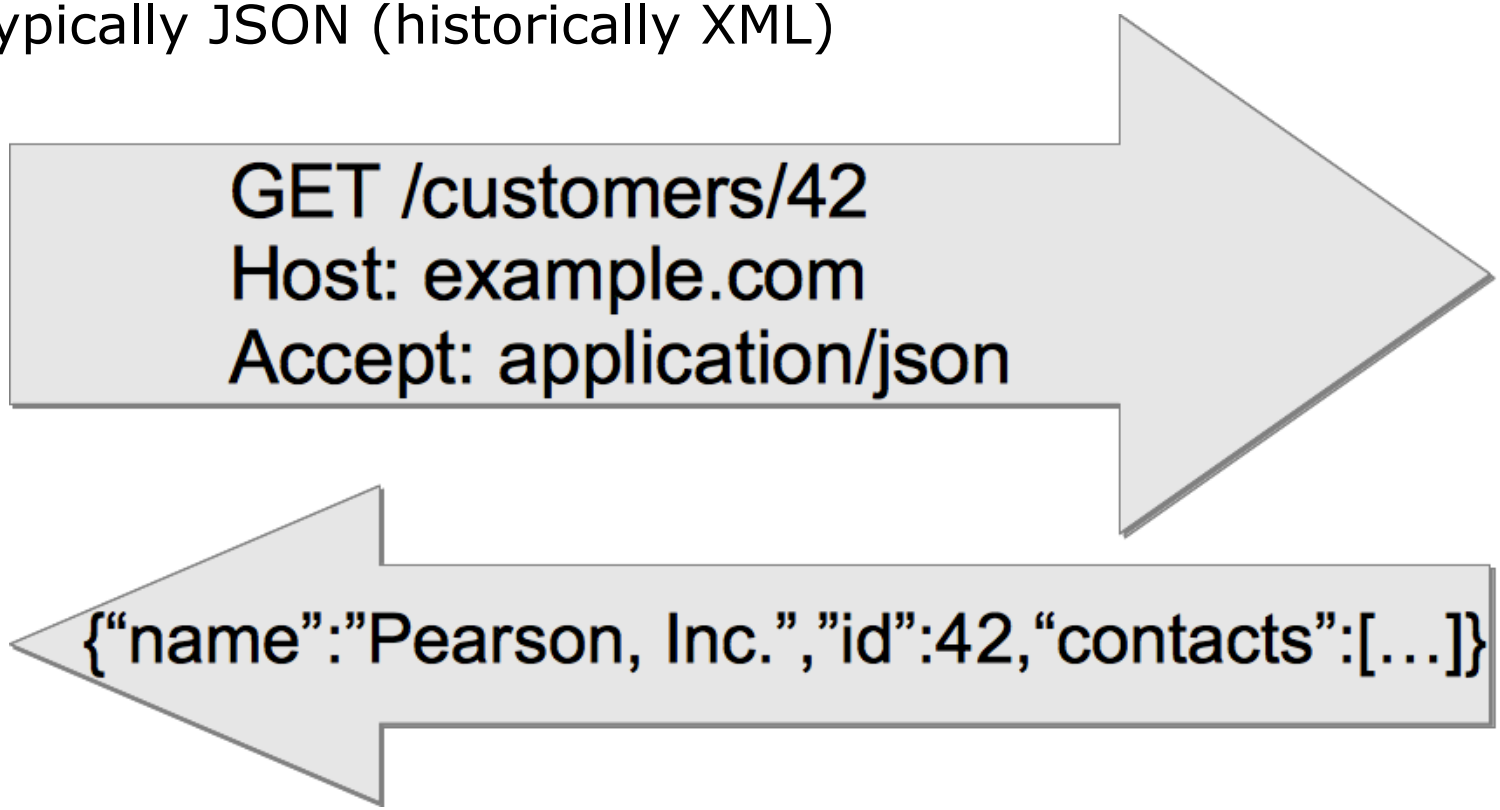
- Identified by URIs
  - Multiple URIs may refer to same resource.
  - Naming URIs is key to usability [BEST PRACTICE]
- Nouns vs. verbs (things vs. actions)

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)

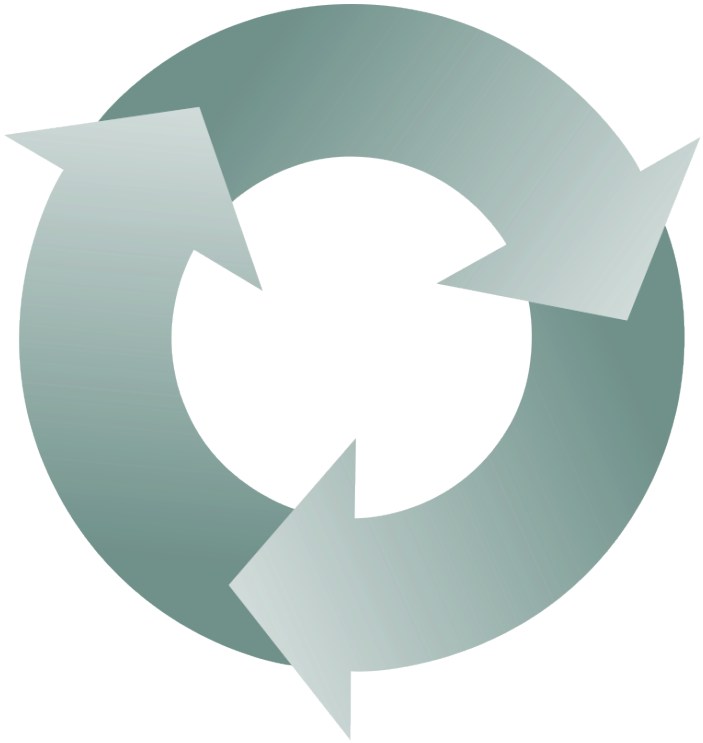
- More advanced model
- Recommend at a minimum, Location header on create:
  - Location: http://example.com/customers/43

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)

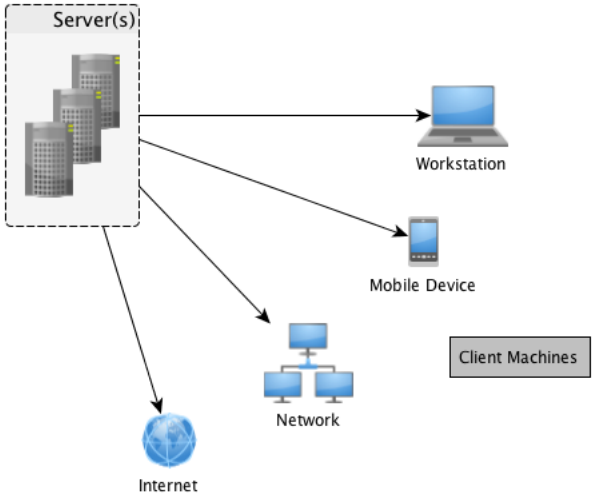


Stateless

- 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

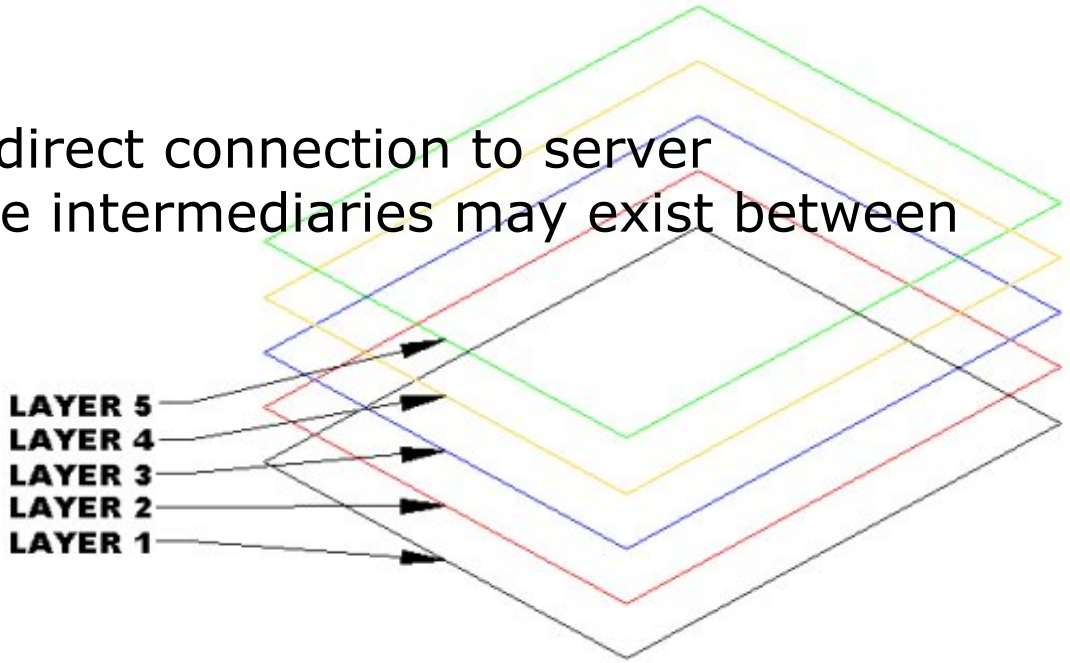


Cacheable

- 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 API Best Practices

- Use HTTP verbs to mean something
  - GET (read)
  - PUT (update)
  - DELETE (uh… delete/remove)
  - POST (create, plus others)
  - OPTIONS (for documentation)
- Sensible, hierarchical resource names (nouns only)
- Return JSON (also support XML)
- Fine-grained resources (with CRUD, if applicable)
- Support caching (via returned HTTP headers)
- Support pagination (limit, offset, with sensible limit default)
- Embrace connectedness (HATEOAS)
  - Location headers on create (minimal)

- See http://www.RestApiTutorial.com Best Practices Guide

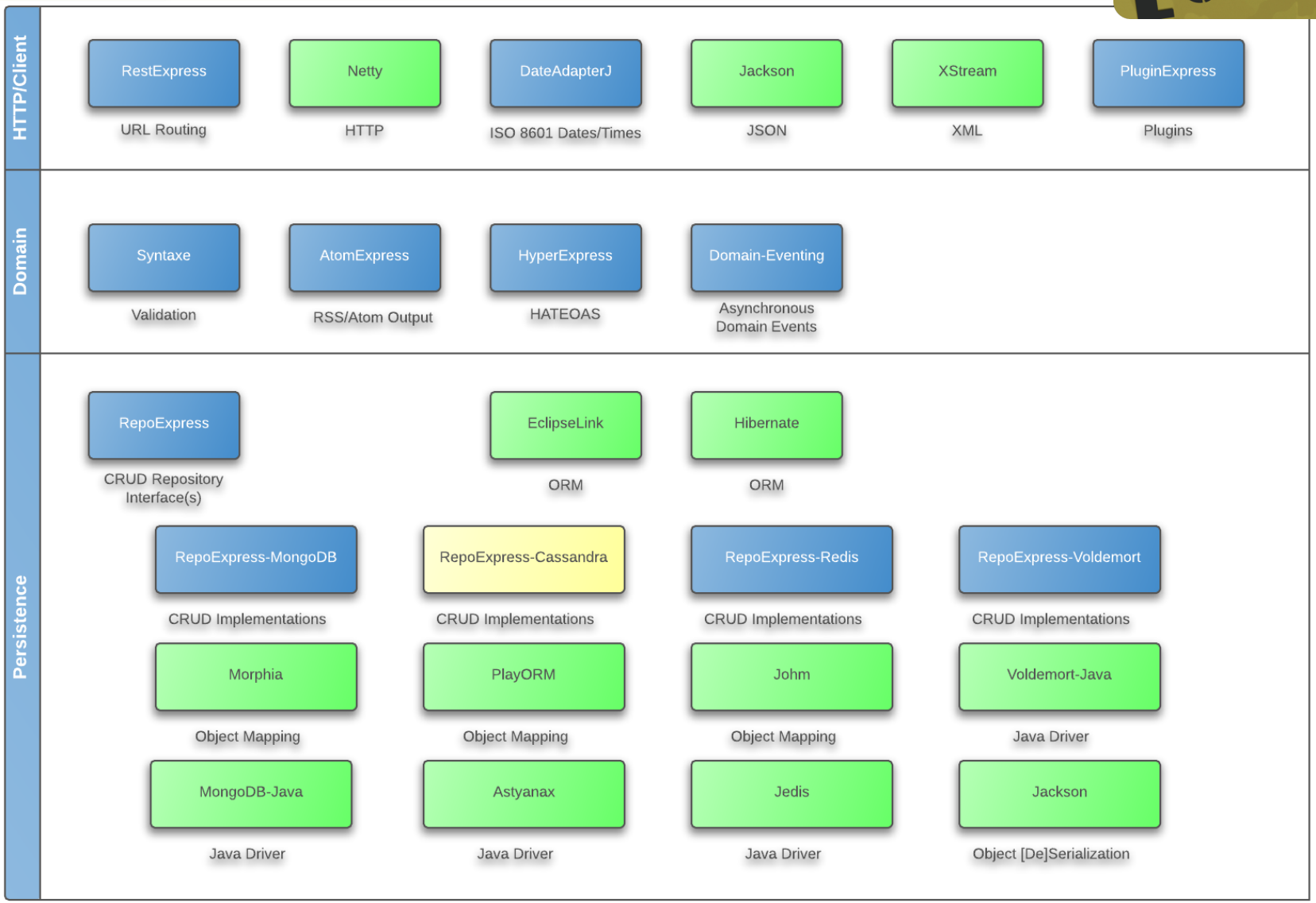
RestExpress

https://github.com/RestExpress/RestExpress

What is RestExpress?

- Simplest thing that could possibly work…
- Java-based REST framework
- For JSON & XML CRUD-style services
- Uses Netty NIO framework as its HTTP server
- Convention over configuration
- ISO8601 time point handling in JSON/XML


BOOT
CAMP
TECH
STACK


HTTP/Client
RestExpress
URL Routing
Netty
HTTP
DateAdapterJ
ISO 8601 Dates/Times
Jackson
JSON
XStream
XML
PluginExpress
Plugins
Domain
Syntaxe
Validation
AtomExpress
RSS/Atom Output
HyperExpress
HATEOAS
Domain-Eventing
Asynchronous
Domain Events
Persistence
RepoExpress
CRUD Repository
Interface(s)
EclipseLink
ORM
Hibernate
ORM
RepoExpress-MongoDB
CRUD Implementations
RepoExpress-Cassandra
CRUD Implementations
RepoExpress-Redis
CRUD Implementations
RepoExpress-Voldemort
CRUD Implementations
Morphia
Object Mapping
PlayORM
Object Mapping
Johm
Object Mapping
Voldemort-Java
Java Driver
MongoDB-Java
Java Driver
Astyanax
Java Driver
Jedis
Java Driver
Jackson
Object [De]Serialization

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();
    }
}
```

Our Project: Little-LMS

- Everyone here knows the domain
- New technology stack is hard enough

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

Phase 1: Courses

- Domain object: Course
  - ID (is supplied by AbsractLinkableEntity)
  - Name, required
  - Description
- Routes:
  - POST /courses.{format}
  - GET, PUT, DELETE /courses/{courseId}.{format}

Let’s Do It!

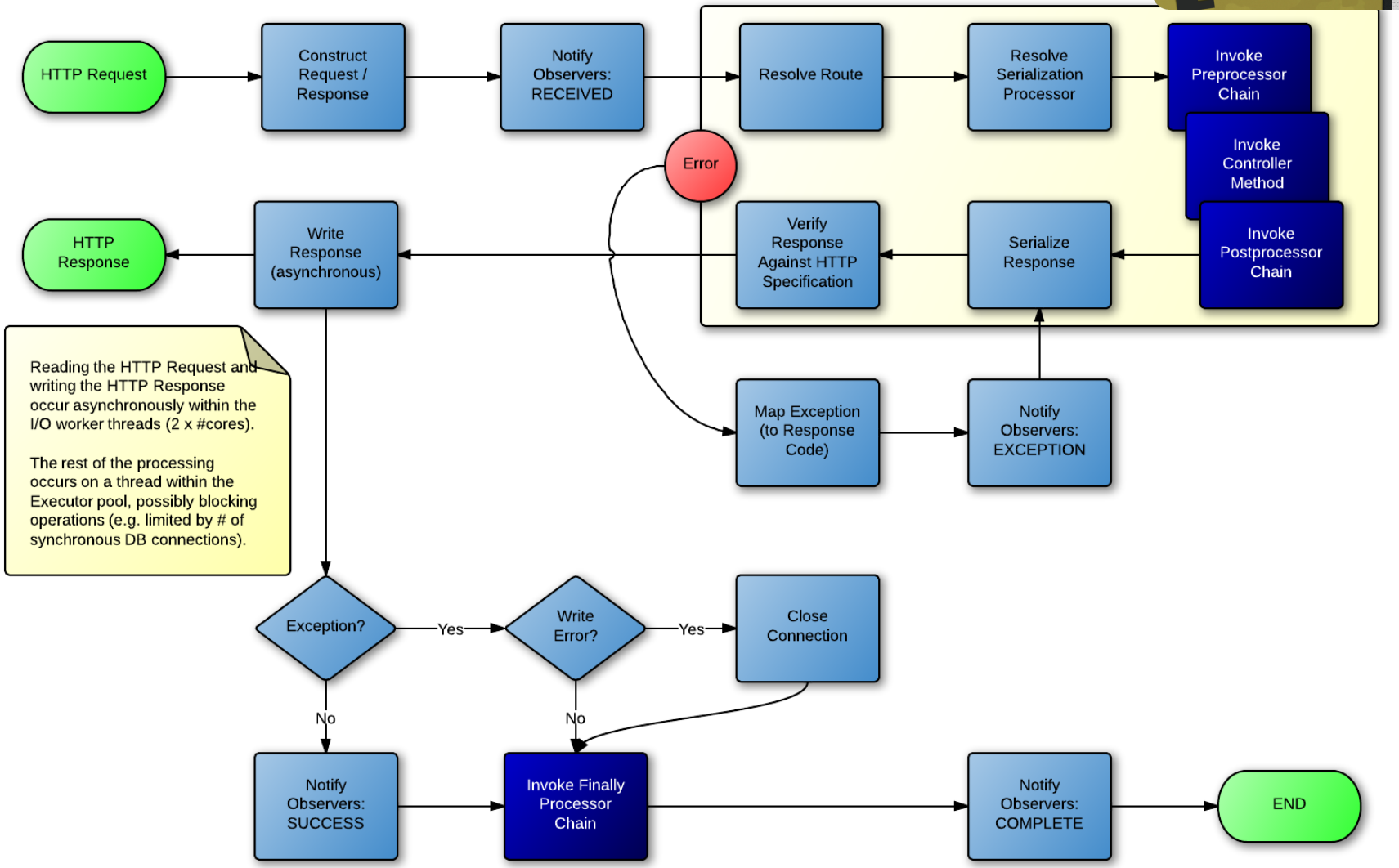
Starting a Project

- Getting Started Guide:
  - http://githubenterprise.pearson.com/openclass-gettingstarted/Getting-Started-Guide
  - Maven settings.xml file

- **Normally:** MongoDB Scaffolding Project
  - http://githubenterprise.pearson.com/openclass-gettingstarted/RestExpress-Scaffold
  - mvn archetype:generate –DarchetypeCatalog=local

Touch Points

- Create Routes (Routes.java)
- Create a Controller (e.g. CourseController.java)
- Create a Service (e.g. CourseService.java)
- Create a Repository (e.g. MongodbCourseRepository.java)
- Add Configuration.java (stitch repository, controller, service)
  - “getter” for controller


HTTP Request
Construct Request / Response
Notify Observers: RECEIVED
Resolve Route
Resolve Serialization Processor
Invoke Preprocessor Chain
Invoke Controller Method
Invoke Postprocessor Chain
Serialize Response
Verify Response Against HTTP Specification
Error
Write Response (asynchronous)
HTTP Response
Reading the HTTP Request and writing the HTTP Response occur asynchronously within the I/O worker threads (2 x #cores).
The rest of the processing occurs on a thread within the Executor pool, possibly blocking operations (e.g. limited by # of synchronous DB connections).
Map Exception (to Response Code)
Notify Observers: EXCEPTION
Exception?
Yes
Write Error?
Yes
Close Connection
No
No
Notify Observers: SUCCESS
Invoke Finally Processor Chain
Notify Observers: COMPLETE
END

MongoDB and RestExpress

Using RepoExpress

RestExpress is DB Agnostic

- Can use any persistence technology
- RepoExpress implements Repository pattern

```
public interface Repository<T extends Identifiable>
{
	public T create(T object);
	public void delete(String id);
	public void delete(T object);
	public boolean exists(String id);
	public T read(String id);
	public List<T> readList(Collection<String> ids);
	public T update(T object);
}
```

RepoExpress Queryable Interface

- Some repositories are 'enhanced' (MongoDB in particular)

```
public interface Queryable<T extends Identifiable>
{
        public long count(QueryFilter filter);
        public List<T> readAll(QueryFilter filter, QueryRange range,
                QueryOrder order);
}
```

ObservableRepository

- An observer lets us "get in the game" as persistence operations occur…

```
public interface RepositoryObserver<T extends Identifiable>
{
        public void afterCreate(T object);
        public void afterDelete(T object);
        public void afterRead(T object);
        public void afterUpdate(T object);

        public void beforeCreate(T object);
        public void beforeDelete(T object);
        public void beforeRead(String id);
        public void beforeUpdate(T object);
}
```

Two Types of MongoDB Repositories

- MongodbEntityRepository
  - Uses Mongo's own ObjectId type as ID
  - Persistence operations on AbstractMongodbEntity instances
  - Sets 'createdAt' and 'updatedAt'
  - Observable
- MongodbRepository
  - Uses object-supplied ID
  - Supply your own ID adapter (if other than String)
  - Persistence operations on Identifiable instances (recommend extending AbstractTimestampedIdentifiable)
  - Observable

Phase 2: Students

- Domain object: Student
  - ID (supplied by AbstractLinkableEntity)
  - Name, required
  - Major
- Routes:
  - POST /students.{format}
  - GET, PUT, DELETE /students/{studentId}.{format}

Let’s Do It!

Touch Points

- Create Routes (Routes.java)
- Create a Controller (e.g. StudentController.java)
- Create a Service (e.g. StudentService.java)
- Create a Repository (e.g. MongodbStudentRepository.java)
- Add Configuration.java (stitch repository, controller, service)
  - “getter” for controller

Domain Validation

Using Syntaxe

Syntaxe is an Annotations-Based Validation Framework

- Annotate Fields
  - @StringValidation(name,required,minLength,maxLength)
  - @IntegerValidation(name,min,max)
  - @RegexValidation(name,pattern,nullable,message)
  - @FieldValidation(Validator extender class)
- Annotate Classes
  - @ObjectValidation(Validitator extender class)

- Class Inheritence
  - Validatable interface
  - validate() throws ValidationException(List<String) on error.
- ValidationEngine.validate(Object)
- ValidationEngine.validateAndThrow(Object)

SubPub

What is SubPub?

- Messaging Infrastructure
- Based on RabbitMQ
- HTTP interface (POST)
- "At Least Once" guarantee semantics
- Supports:
  - Pub/sub
  - Request/reply
- Fine and course-grained subscription options
  - Based on 'context tags'
- Ignores payload

- See: http://code.pearson.com (search for 'prospero')
  - Prospero Integration Guide

Publishing via SubPub

- SubPub-java (driver)
- http://githubenterprise.pearson.com/FREDTA2/subpub-java
- SubPubMessageBus.publish(Message);

- Implementation additions:
  - Add Configuration.getMessageBus();
  - MessageFactory.getInstance().toMessage(Object)
  - Taggable interface

- See: SubPub-Publisher
- http://githubenterprise.pearson.com/openclass-gettingstarted/RestExpress-Publisher

Subscribing to SubPub Messages

- SubPub-java (driver)

- RestExpress Plugin:
  - SubPubSubscriptionPlugin
  - http://githubenterprise.pearson.com/FREDTA2/RestExpress-SubPub-Plugin

- Extend SubPubDeliveryHandler
  - Implement *getMessageType()*
  - [optional] Implement *getContextTags()*
  - Implement *handle(Delivery)*

- See: RestExpress-Subscriber
- http://githubenterprise.pearson.com/openclass-gettingstarted/RestExpress-Subscriber

Advanced RestExpress

Pagination, Filtering, Sorting and Other Goodies

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

QueryRanges and QueryRange classes

- QueryRanges.parseFrom(Request r)
- QueryRanges.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

QueryFilters and QueryFilter Classes

- QueryFilters.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)

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?

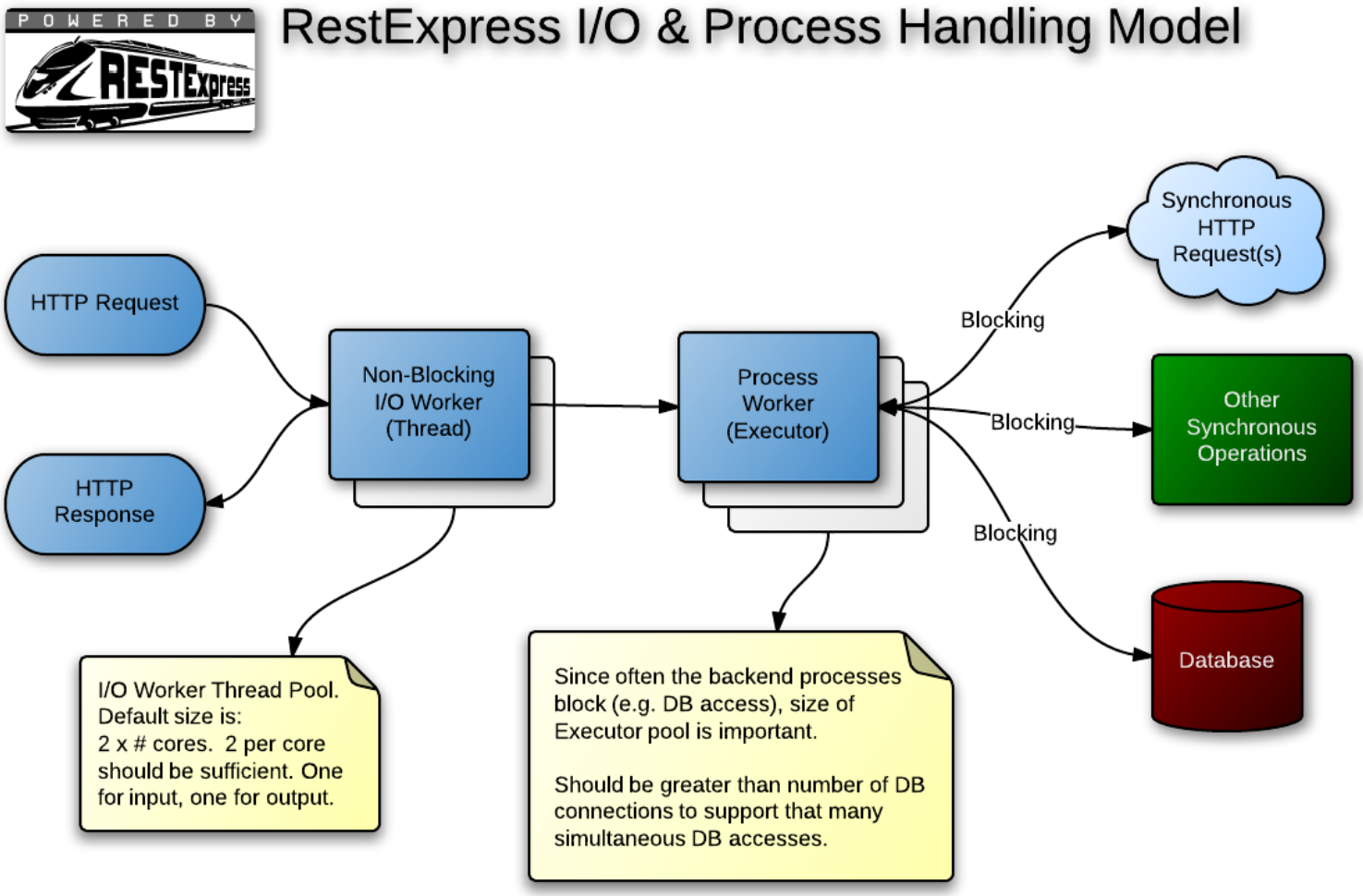
- AJAX (browser) clients
- Error conditions

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 I/O & Process Handling Model
HTTP Request
HTTP Response
Non-Blocking I/O Worker (Thread)
Process Worker (Executor)
Synchronous HTTP Request(s)
Other Synchronous Operations
Database
Blocking
Blocking
Blocking
I/O Worker Thread Pool. Default size is: 2 x # cores. 2 per core should be sufficient. One for input, one for output.
Since often the backend processes block (e.g. DB access), size of Executor pool is important.
Should be greater than number of DB connections to support that many simultaneous DB accesses.

Phase 3: Enrollments

- Domain object: Enrollment
  - ID, required (MongoDB ObjectId)
  - StudentId, required, must exist
  - CourseId, required, must exist
- Routes:
  - GET /students/{studentId}/enrollments.{format}
  - POST, DELETE /students/{studentId}/enrollments/{courseId}.{format}
  - GET /courses/{courseId}/enrollments.{format}
  - POST, DELETE /courses/{courseId}/enrollments/{studentId}.{format}

Let’s Do It!

Touch Points

- Create Routes (Routes.java)
- Create a Controller (e.g. EnrollmentController.java)
- Create a Service (e.g. EnrollmentService.java)
- Create a Repository (e.g. MongodbEnrollmentRepository.java)
- Add Configuration.java (stitch repository, controller, service)
  - “getter” for controller

HATEOAS

With HyperExpress

Hypermedia Linking (HyperExpress)

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

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

- Helpers
  - LinkUtils
  - MapStringFormat

Let’s Do It!

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/RestExpress/wiki

Title

- Bullet