HTML5 SPA(rchitecture) Shift

Presented by: Adrienne Gessler

KCDC: May 16, 2014

Titanium Sponsors

AJi

VALOREM CONSULTING
WWW.VALOREMCONSULTING.COM
REGISTER TO WIN AN XBOX!

PAIGE TECHNOLOGIES
INTELLIGENT PAIRING. PERPETUAL SUCCESS.

AdventureTech
www.adventuretechgroup.com

Platinum Sponsors

ADAPTIVE™ SOLUTIONS GROUP

DST® SYSTEMS

Balance® Innovations

Sprint®

epiq SYSTEMS

DevExpress™

jack henry & ASSOCIATES INC.®

O'REILLY®

Kauffman LABS for Enterprise Creation

FREIGHTQUOTE

Gold Sponsors

Cerner

CENTRIQ TRAINING

Bradford & Galt CONSULTING SERVICES

ADVANTAGE TECH

KEYHOLE SOFTWARE

ComponentOne a division of GrapeCity

dsi®

LRS® Consulting Services

GARMIN®

JetBRAINS

Microsoft

MULTI SERVICE® INNOVATION WHERE IT MATTERS.

NETCHEMIA™ Transforming the way education works

OAKWOOD

stackify

KU EDWARDS CAMPUS The University of Kansas

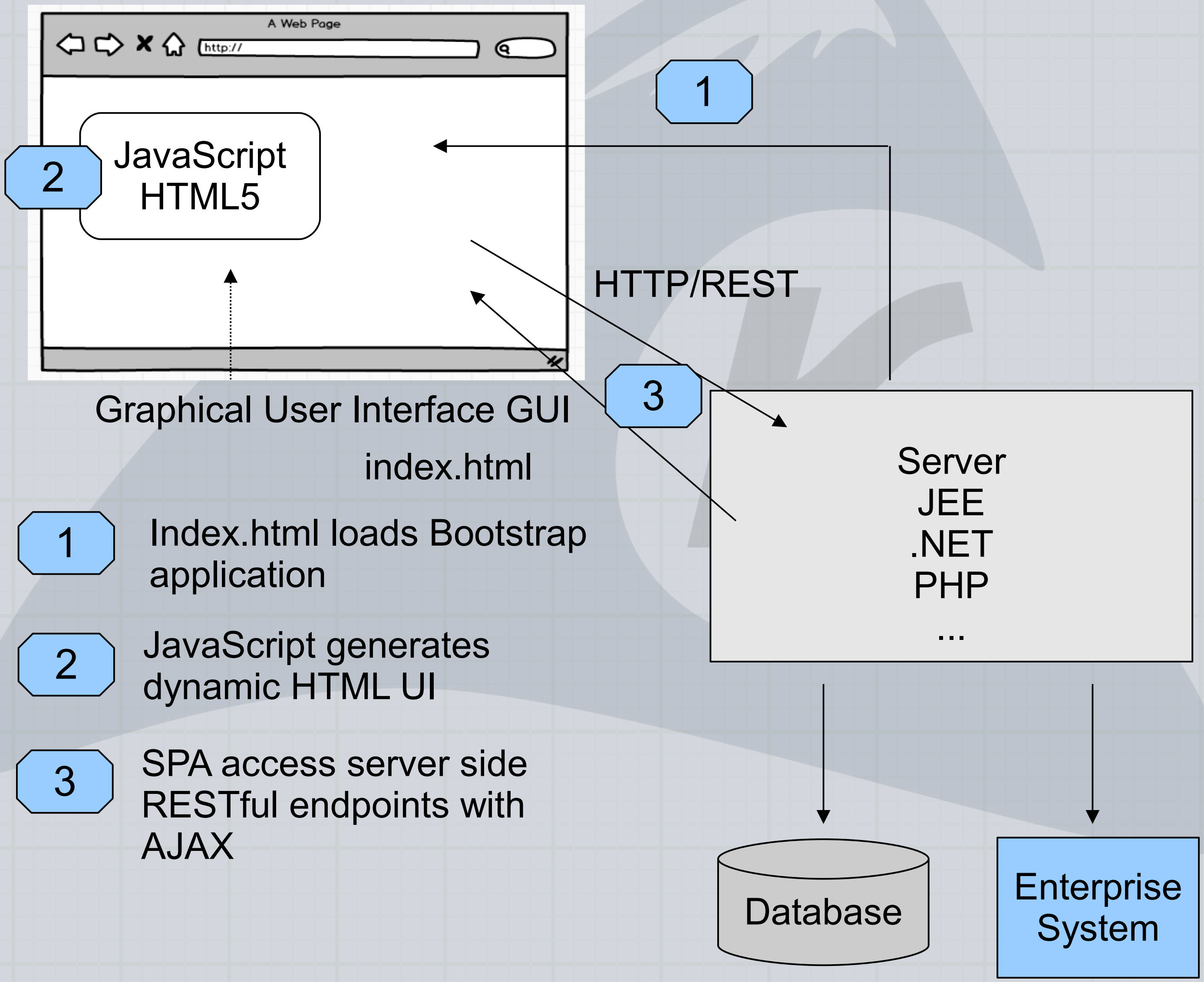
perceptivesoftware from Lexmark

TEKsystems

twilio™ CLOUD COMMUNICATIONS

UNITEDLEX

SPA (JavaScript)



1 Index.html loads Bootstrap application

2 JavaScript generates dynamic HTML UI

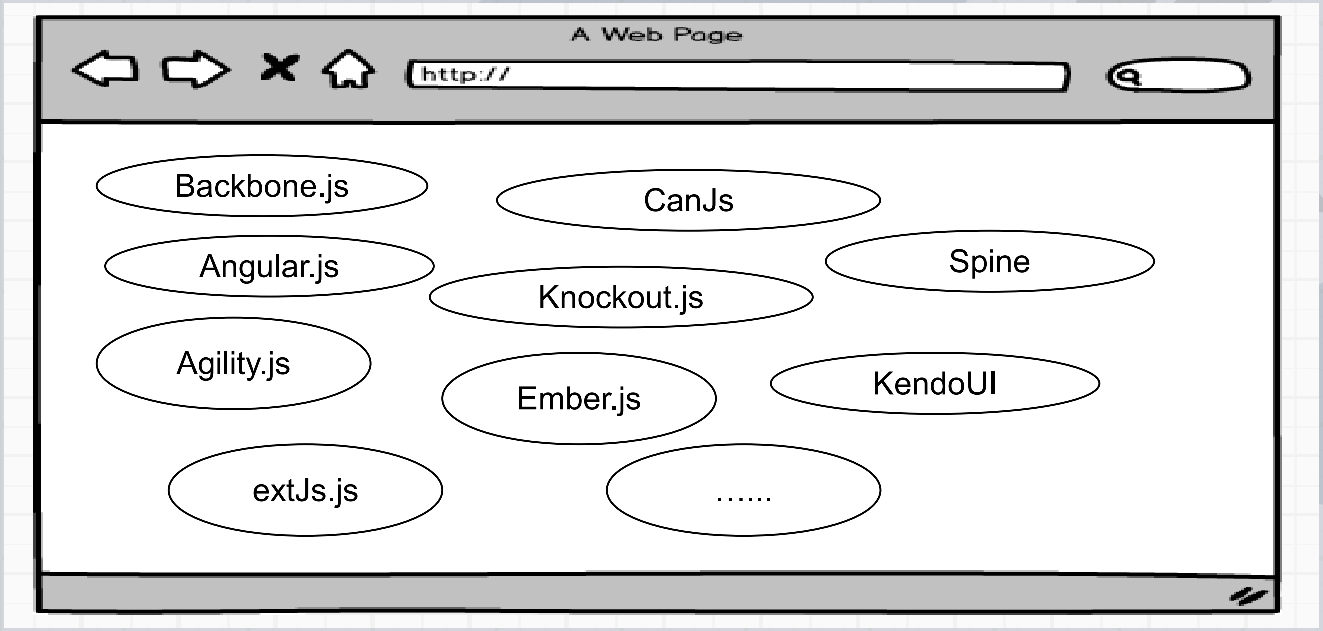
3 SPA access server side RESTful endpoints with AJAX

Why?

- Decoupled UI
- Plug-ins eliminated (Applet, Silverlight, Flex ...)
- Rich Responsive UI experience
- Exploit HTML5 Features
- Responsive to multiple devices
- Lower network bandwidth

Client Side

- JavaScript – the engine to drive the user experience
- JavaScript MVC Frameworks



UI Elements are produced by 100% client side JavaScript code...

Server side application data is accessed via asynchronous (AJAX) HTTP calls...

JavaScript MVC

- Emulates Server Side MVC frameworks
- JavaScript Objects (Models/server side access of JSON/API)
- UI Components (event listeners)
- HTML Templates (DOM manipulation)
- Controllers (binds models to templates/UI, handles user actions/events)
- Navigation
- Modularity and Maintainability

Architectural Shift

No more dynamic HTML on the Server Side

UI built entirely with JavaScript/HTML/CSS

Which brings us to the Server side...

RESTful Architectural Style

- Representational State Transfer
- Introduced in 2000 by Roy Fielding's doctoral dissertation
- Goals: Performance, Scalability, Simplicity, Portability, Reliability
- Stateless – the server does not know or care what state the client is in
- State is transfered to client by a set of operations and content types

RESTful Architectural Style (cont.)

- Application state and functionality represented by a uniquely addressable resource (id associated to resource)
- Resources identified by unique urls and addressed using a universal syntax for use in hypermedia links

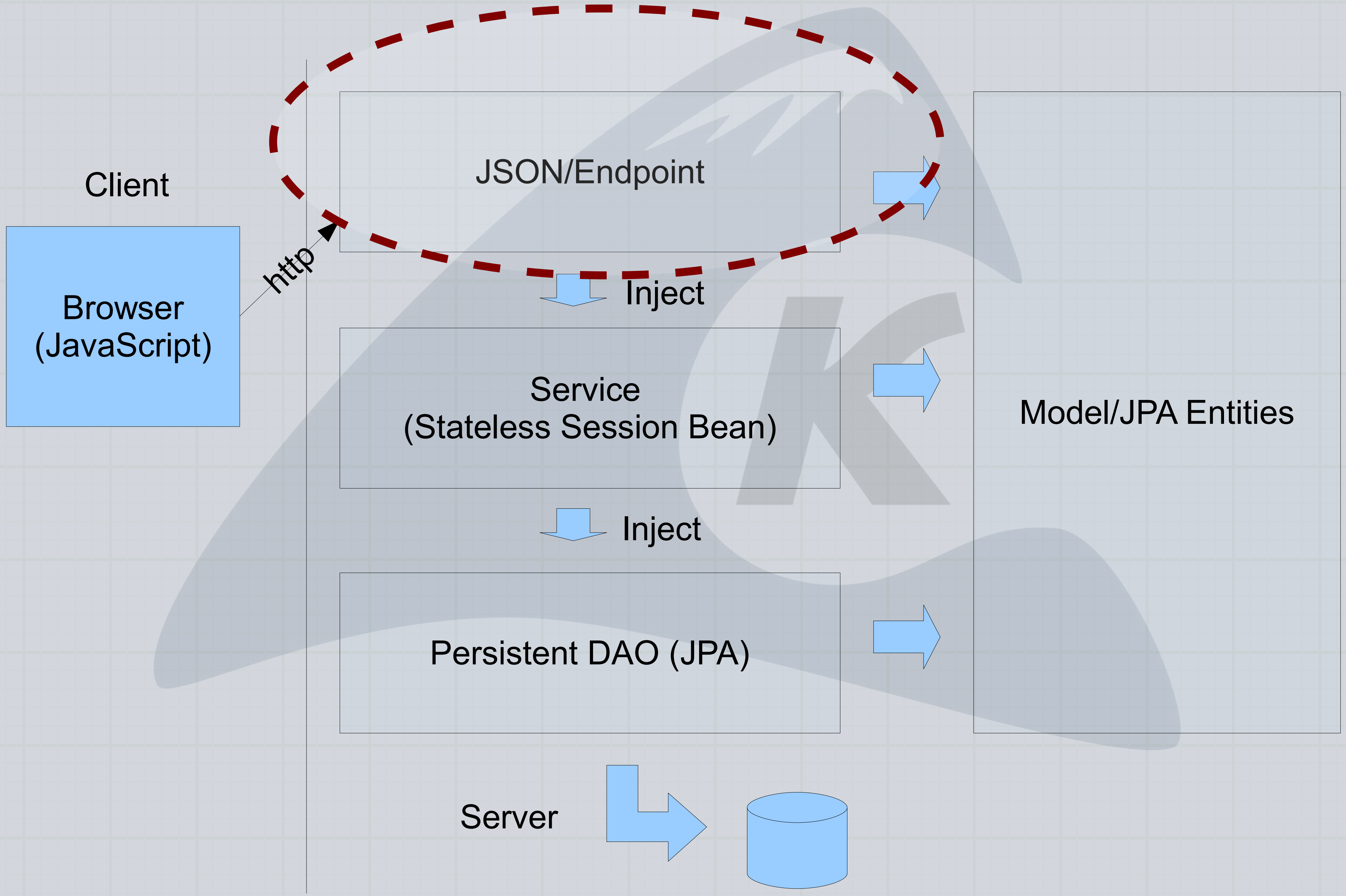
Why REST

- Server side responsibility shift to supplying data
- Complete decoupling with client
- Simple interface to and from data and business logic
- Minimal overhead of HTTP
- Lean and easily serialized of JSON on both server and client
- Uniform interface (standard HTTP methods and responses)
- HTTP format separates header and body
- Rich browser clients communicating via Ajax benefit from scaling principles

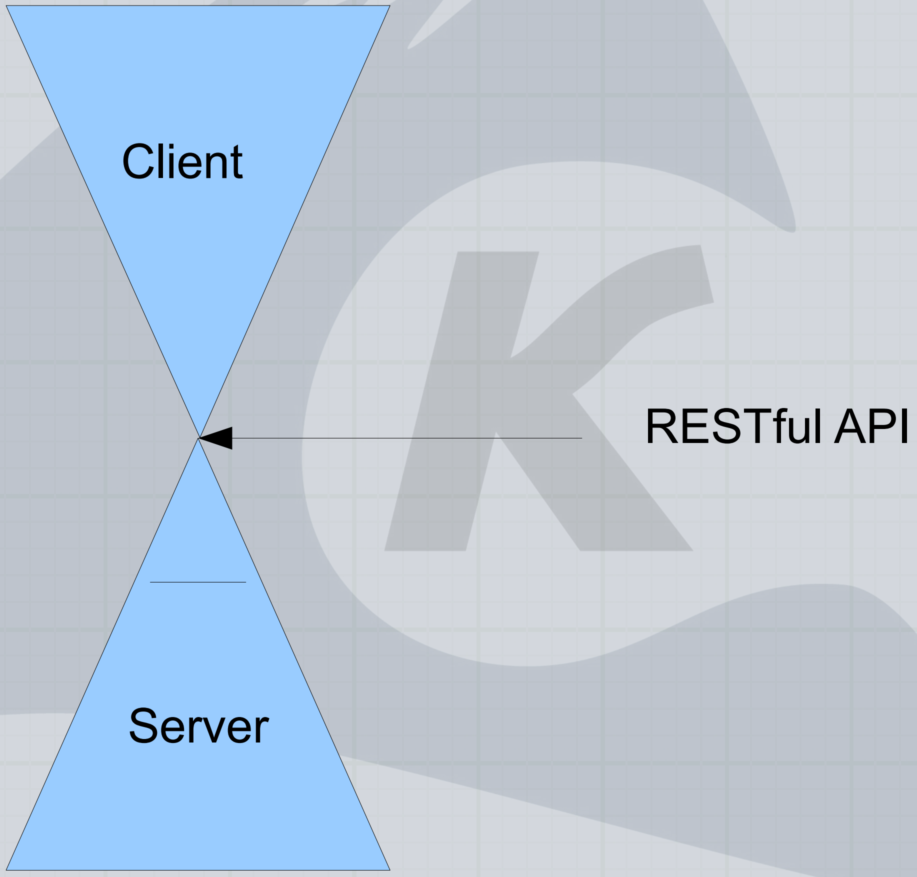
Why Java REST

- Good transition point for JEE server side developers
- Leverage existing knowledge base/code, java scalability and portability
- Integration with other Java EE APIs
- Easy way to expose existing objects to remote systems
- Other advanced features such asynchronous processing, throttling, and monitoring

JSON/API Place in EE/Spring



JEE Server Side



Restful Style API With JSON

All Categories API

http:<<server>>/api/service/categories

```
[
{"id":1,"description":"Operating System","name":"Operating System","imageUrl":""},
{"id":2,"description":"Version Control","name":"Version Control","imageUrl":""},
{"id":3,"description":"Relational Database","name":"Relational Database","imageUrl":""},
{"id":118,"description":"Language","name":"Language","imageUrl":""},
{"id":163,"description":"Testing","name":"Test Category","imageUrl":""},
{"id":168,"description":"","name":"Return Codes","imageUrl":""},
{"id":169,"description":"test 2 for request changes","name":"test 2","imageUrl":""},

  …....

]
```

JSON

RESTful API

Categories by Id...

GET

`http:<server>/api/service/category/100`

POST ... { description: "Language",id: 118,imageUrl: ""name: "Language"}

`http:<server>/api/service/category/`

(POST non idempotent – user create when you know the location of the factory to create item or to update

PUT .... { description: "Language",imageUrl: ""name: "Language"}

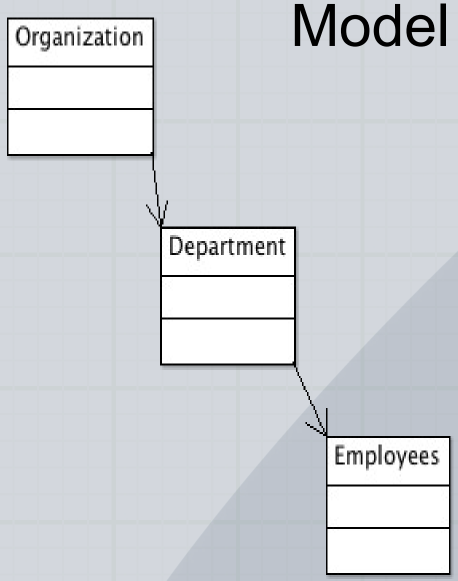
`http:<server>/api/service/category/100`

(PUT idempotent – create or replace – use when you know the URL of the item)

DELETE...

`http:<server>/api/service/category/100`

API Design



Model

- Fine Grained (keep object models succinct)
- Limited Object Navigation (1. to Many)

Model traversed through API calls

Organization 100

/api/organization/100

Departments for organization 100

/api/departments/organization/100

Employees for department 200

/api/employees/dept/200

API Design - Some Basics

- Follow standards – HTTP methods and response (status) codes – including error codes
- Avoid tunneling - using the same method on a single URI for different actions
- Every resource has its own URL
- Distinct resources should be used for each operation
- Allow application flow by providing links and reference id's in representations (Hypermedia as the Engine of Application State), but don't force it on client
- Analyze to find your simple cases (create/read/update/delete)
- Determine granularity for more complicated scenarios (consider network resources/client needs/ etc)

Authentication/Authorization

- Basic/Digest HTTP Authentication
- Public key
- TOKEN-based
- Session-based (Roll Your Own, yes another Login user story) use a servlet filter
- Container Supported JAAS (form authentication)
- Spring Security
- OAuth2

Versioning/Caching

- Versioning
- Caching

Many other advanced (ish) topics to consider, but the basics are pretty straightforward...

JAX-RS API

- JSR 339 – REST Architecture Style, Java API for RESTful Web Services
- API specification adhering to the uniform REST interface to simplify the development and deployment of web service clients and endpoints.
- From version 1.1 on, JAX-RS is an official part of Java EE 6. A notable feature of being an official part of Java EE is that no configuration is necessary to start using JAX-RS

JAX-RS Specification

- @ApplicationPath
- @Path
- @GET, @PUT, @POST, @DELETE and @HEAD, ...
- @Produces, @Consumes
- @PathParam, @QueryParam
- @MatrixParam, @HeaderParam, @CookieParam, @FormParam
- @Context

Java RESTful End Point Frameworks

- JBOSS EasyRest
- Jersey
- Apache CFX
- Spring MVC
- Provided with JEE6
- Roll own (Servlet, JSONMapper, etc.) Not recommended
- khsSherpa

Jersey

- Serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation
- Provides support for Spring in the REST layer

Client->Jersey->Web Service Component Resouce->Underlying Resource

https://jersey.java.net

Jersey cont.

- MessageBodyReader/MessageBodyWriter
- ExceptionMapper classes
- @Ref annotation/URI builder
- @Link
- Injection

Jersey Endpoint Example

```
@Path(“/service/category”)
public class Categories {

   private CategoryRepository categoryRepo;

   @Get
   @Produces({Mediatype.APPLICATION_JSON,”application/x-javascript})
   @Path(“/categoryId”)
   public Category getCategory(@PathParam(“categoryId”) final long
categoryId)

         return categoryRepo.findForId(categoryId);

   }
}
```

https://jersey.java.net

JBOSS RestEasy

- JBOSS project, but can run in any Servlet container, but offers tighter integration with the JBoss Application Server
- Imbedded server implementation for JUnit testing
- EJB, Seam, Guice, Spring, and Spring MVC integration
- Server in-memory cache.
- Rich set of providers for wide variety of response types

JAX-RS Compliant

http://www.jboss.org/resteasy

JBOSS RestEasy

```
@Path("/message")
public class MessageRestService {

@GET
@Path("/{param}")
public Response printMessage(@PathParam("param")
integer id) {

 Category category = repo.findForId(id);

return
Response.status(200).entity(category).build();

}

}
```

JAX-RS Compliant

http://www.jboss.org/resteasy

Sherpa Endpoint Example

```
@Endpoint(authenticated=false)
public class CategoryEndpoint {

@Autowired
CategoryService service;

@Action (mapping = "/service/categories", method = MethodRequest.GET)
public List<Category> categories() {
return service.findAll();
}

@Action (mapping = "/service/category/{categoryId}", method =
MethodRequest.GET)
    public Category getCategory(@Param("categoryId") Long categoryId) {
        Category cat =  service.findById(categoryId);
        return cat;
    }

}
```

https://github.com/organizations/in-the-keyhole

Spring MVC

- Not a REST framework in itself, but now comprehensive REST support in Spring MVC for web services after version 3

- Does not implement JAX-RS, but offers similar functionality

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Spring MVC Example

```
@RequestMapping("/category/{categoryId}",
method=RequestMethod.GET)
public String findOwner(@PathVariable long catergoryId,
Model model) {
  Category cat = catergoryService.findOwner(ownerId);
  model.addAttribute("category", owner);
  return "displayCategory";
}
```

POJO are serialized to JSON using a View Resolver

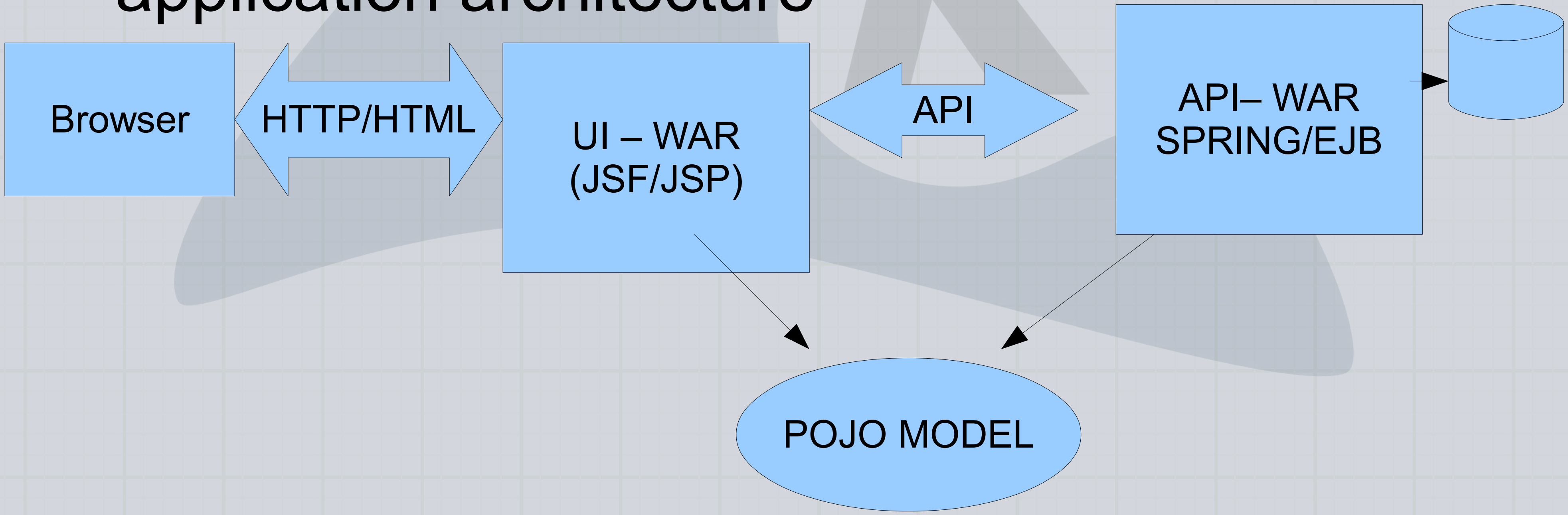
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Other Options

- Apache CFX
- RESTlets
- RESTX

Positioning for SPA

- Learn JavaScript, really learn JavaScript
- Add in REST as you build your client
- Another option for transition: Introduce API style programming to existing Dynamic Java HTML application architecture



Fin…

Any questions?

Download This Presentation

Presentation materials available on the Keyhole GitHub:

bit.ly/keyholekcdc14

Other Keyhole Presentations Available:

- *Advanced JavaScript*
- *Debugging Techniques for Agile Teams*
- *Grunt 101*
- *JSF In The Modern Age*

Stop by Keyhole Software’s booth to talk about career opportunities with my team & get entered to win an HP Chromebook 14.

Role-based Access Permissions

- Spring Security
- JEE Container Supported (JAAS)
- Applied at API layer

```
@RolesAllowed({"admin"})
public Department create(@Param("number") int number,@Param("name")String name) {
    Department dept = new Department();
    dept.number = number;
    dept.name = name;
    return dept;
}
```

Applied to endpoint implementation...

Versioning

- Lots and lots of debate...

Version number in the URL.....

/api/v1/categories

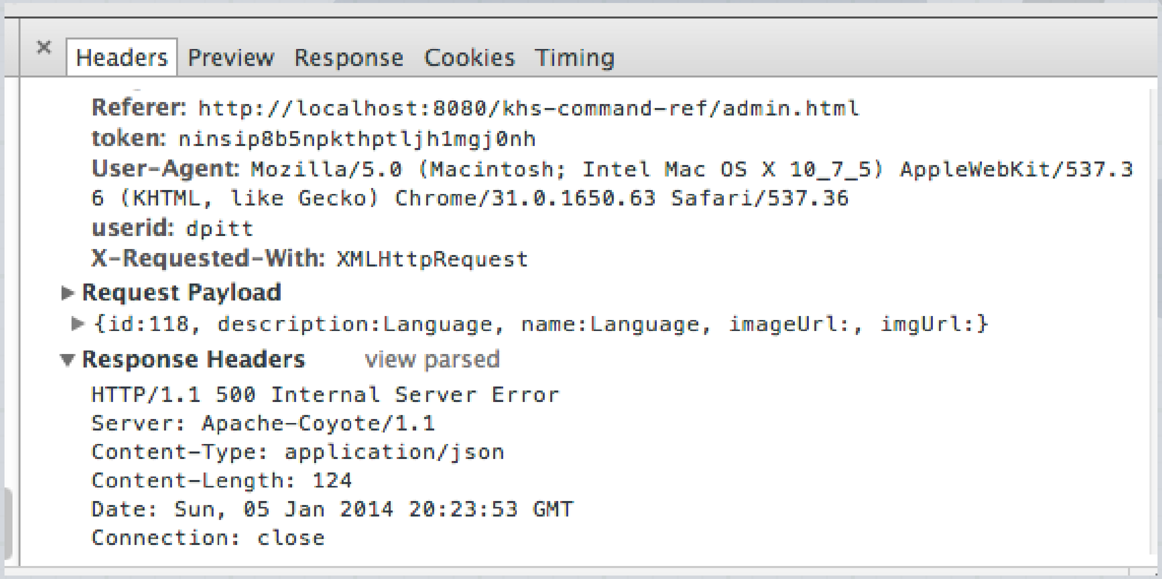
/api/categories/v1/categories

Version number in the Header using Accept Header....

Debate revolves around what is really RESTful (ie. HATEOAS would use header information), Version number in URL is easier to use.

Exceptions/Errors

- Server returns HTTP Error Codes
- Response contains exception information (Stack trace, etc.)
- Stick with standard errors



```
Referer: http://localhost:8080/khs-command-ref/admin.html
token: ninsip8b5npkthptljh1mgj0nh
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36
userid: dpitt
X-Requested-With: XMLHttpRequest
▶ Request Payload
 ▶ {id:118, description:Language, name:Language, imageUrl:, imgUrl:}
▼ Response Headers      view parsed
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
Content-Type: application/json
Content-Length: 124
Date: Sun, 05 Jan 2014 20:23:53 GMT
Connection: close
```