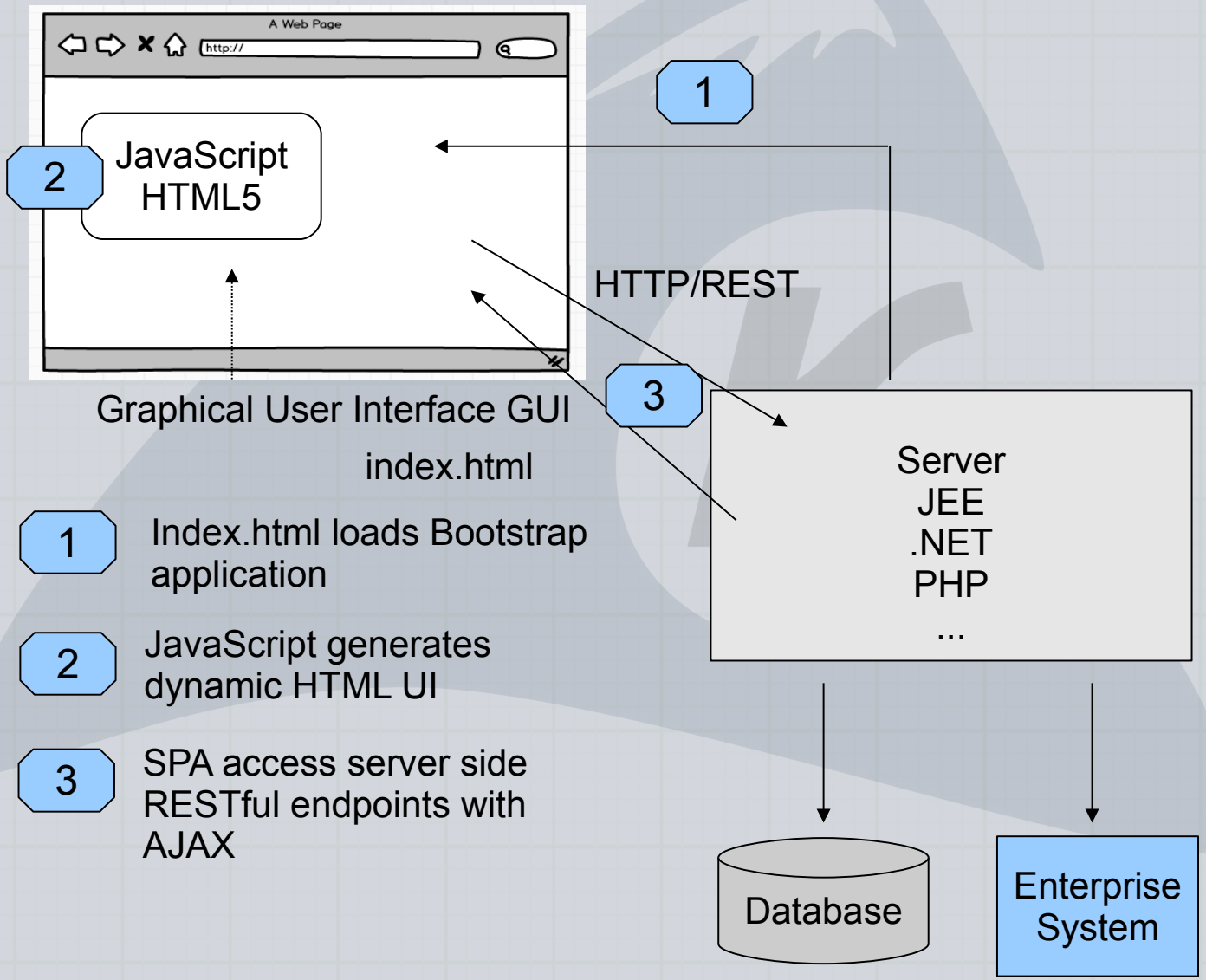
HTML5 SPA(rchitecture) Shift

Presented by: David Pitt

Keyhole Software

KC JUG Meeting: January 8, 2014

SPA (JavaScript)



1 Index.html loads Bootstrap application

2 JavaScript generates dynamic HTML UI

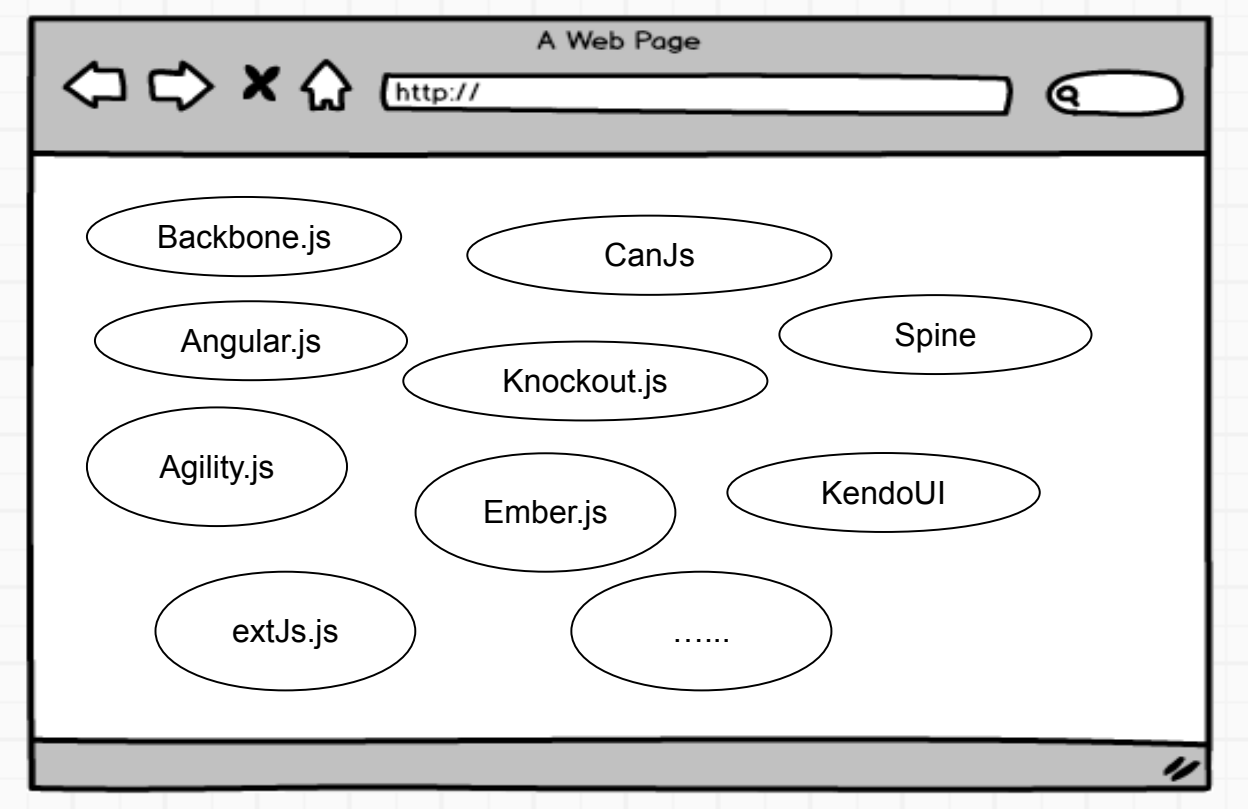
3 SPA access server side RESTful endpoints with AJAX

Why?

- Decoupled UI (UI can become throwaway)
- Plug-ins eliminated
- Rich Responsive UI experience (Fine grained DOM manipulation/AJAX asynchronous)
- Exploit HTML5 Features
- Responsive to multiple devices
- Lower network bandwidth (helps with battery life)

Client Side

- JavaScript MVC Frameworks



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

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

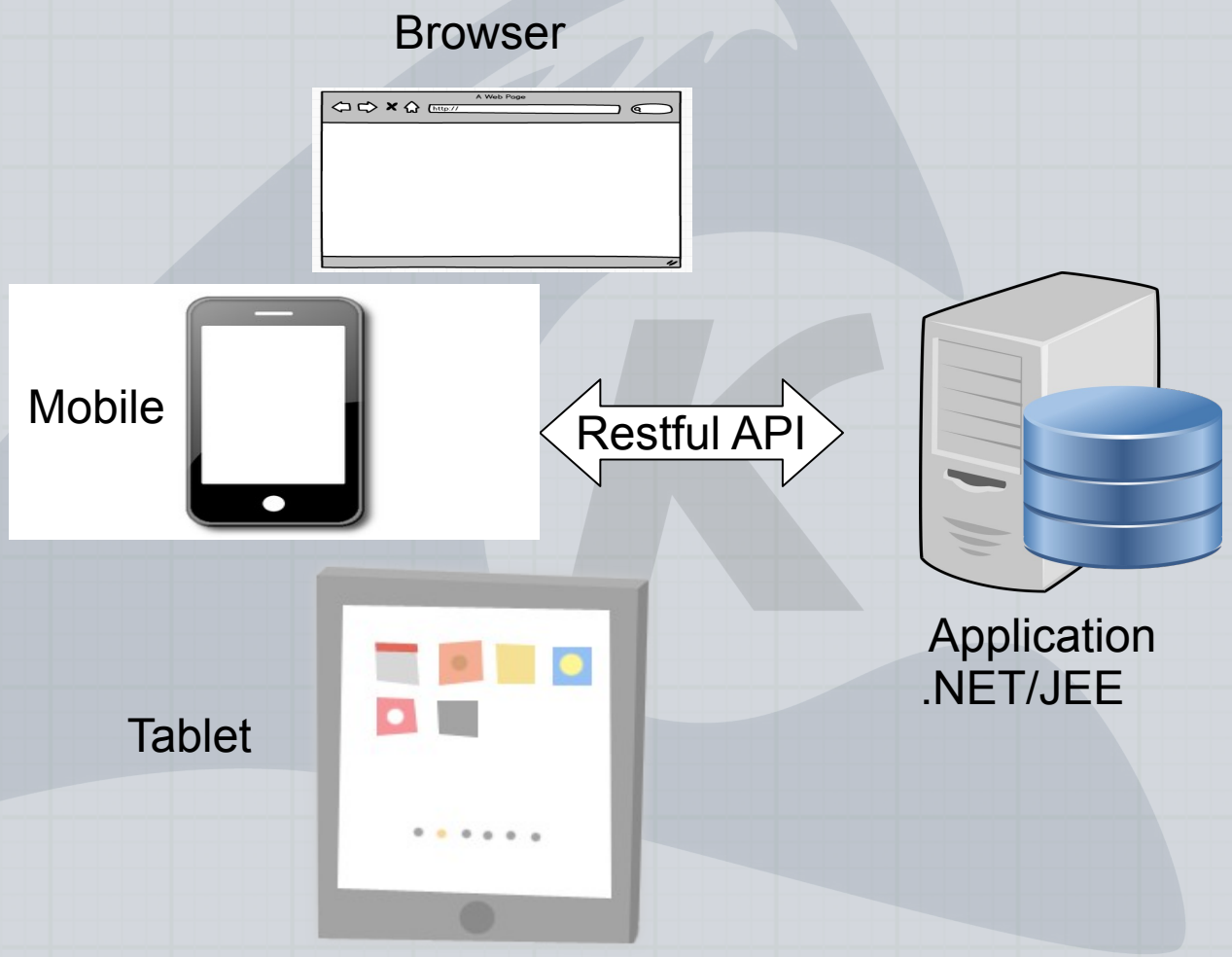
JavaScript MVC

- 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

Lots of Choices

- Essentially two types:
  - Opinionated (Angular.js, ExtJs...)
  - Non-opinionated (Backbone.js...)
- All have benefits, hard to say what is best
- Bottom line - become proficient in JavaScript programming

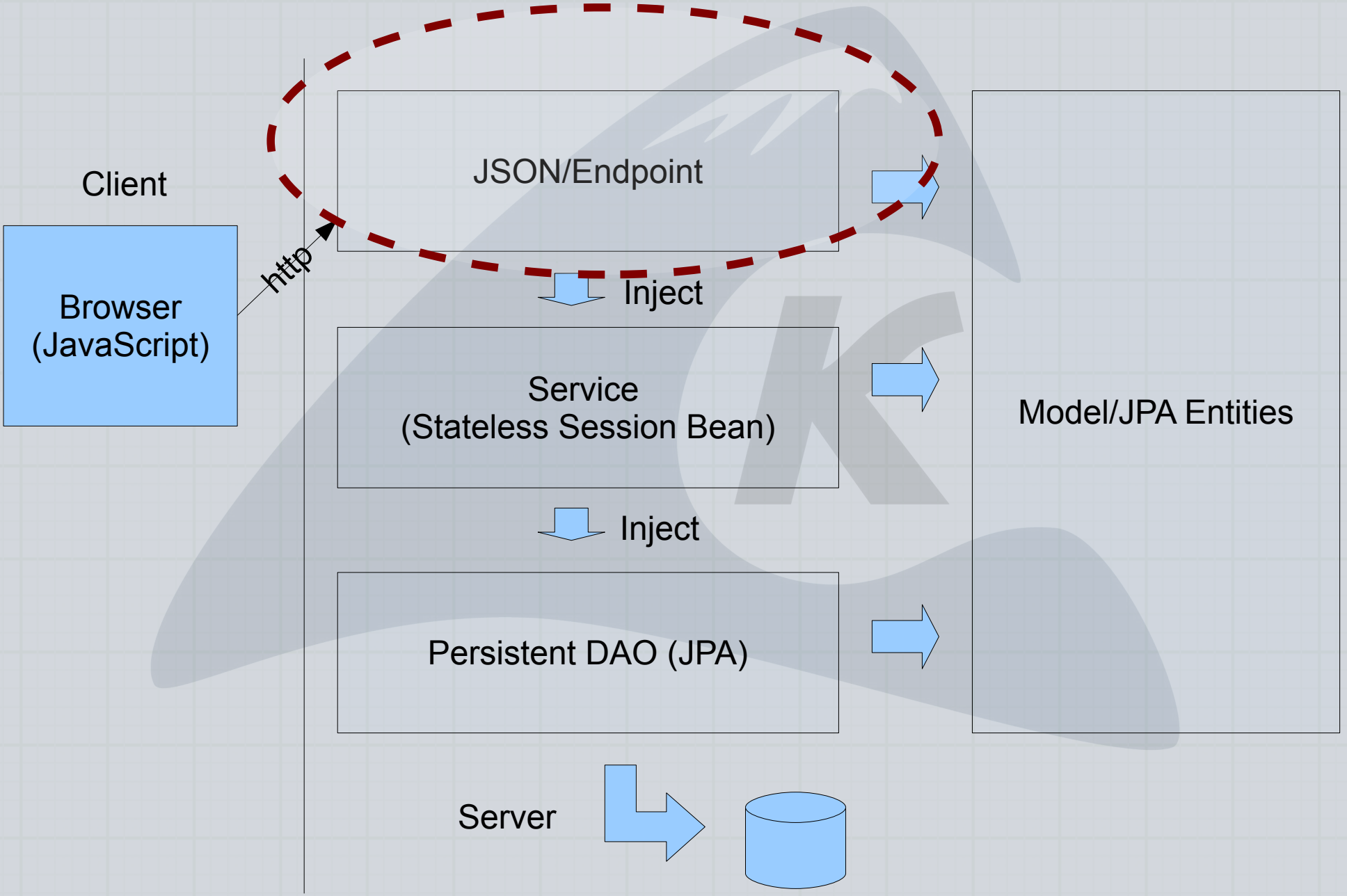
JEE Server Side



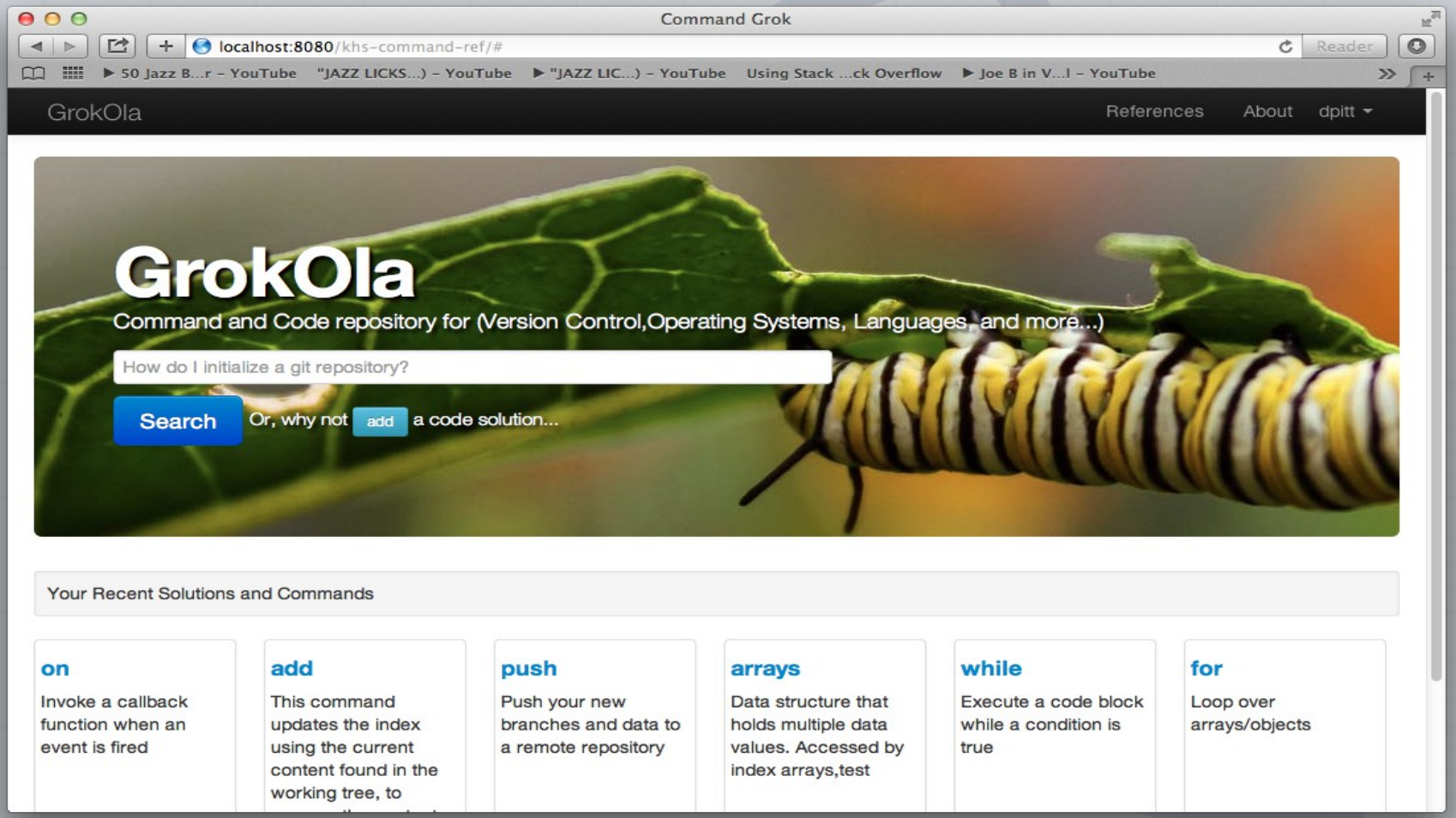
Java RESTful End Point Frameworks

- JBOSS EasyRest
- Jersey
- Apache CFX
- Spring MVC
- Provided with JEE7
- Roll own (Servlet, JSONMapper, etc.) Not recommended
- khsSherpa

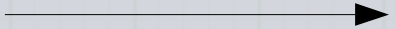
JSON/API Place in EE/Spring



Reference Application

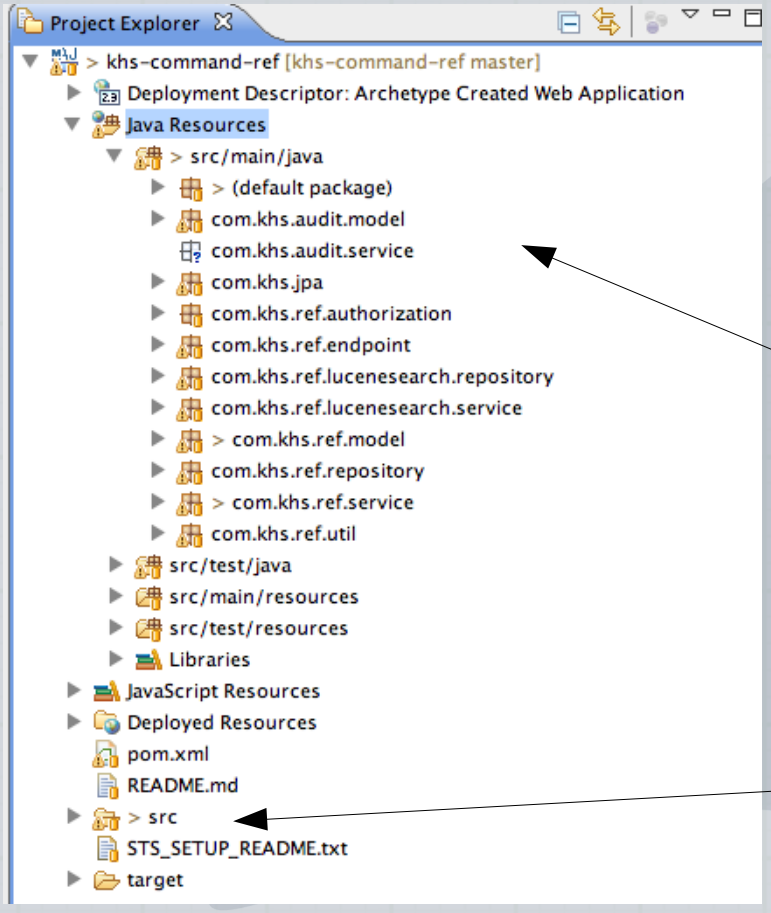


JavaScript CLIENT
(Backbone.js, Bootstrap)



SERVER -
Java/Spring

Project Structure



Maven WAR/WebApp Archetype

Server Side Java Elements

Client side JavaScript resources, CSS etc... define in src/main/webapp

RESTful Architectural Style

- Representational State Transfer
- Introduced in 2000 by Roy Fielding doctoral dissertation
- Goals:
  - Performance
  - Scalability
  - Simplicity
  - Portability
  - Reliability
- Common data representations (HTML, XML, JSON), also text , image, or ??

Restful Style API With JSON

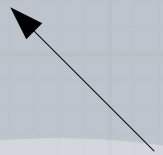
All Categories API

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...

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

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

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

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

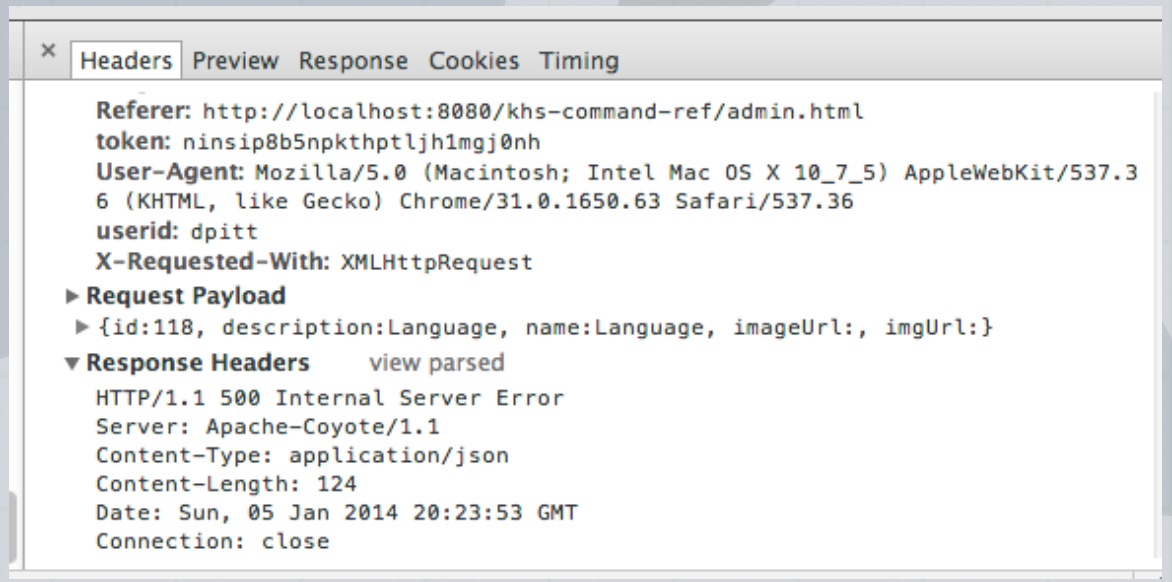
```
http:<server>/api/service/category
```

DELETE...

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

Exceptions/Errors

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



Authentication/Authorization

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

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.

Only matters if you plan on decoupling UI from server side API...

JAX-RS API

- JSR 339 – REST Architecture Style, Java API for RESTful Web Services

Packages

Package	Description
javax.ws.rs	High-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.client	The JAX-RS client API
javax.ws.rs.container	Container-specific JAX-RS API.
javax.ws.rs.core	Low-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.ext	APIs that provide extensions to the types supported by the JAX-RS API.

POJO-Based – Annotation Driven

Currently Version 2.0

Jersey Endpoint Example

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

  private CategoryRepository categoryRepo;

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

        return categoryRepo.findForId(categoryId);

  }
}
```

JAX-RS Compliant

https://jersey.java.net

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 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";
}
```

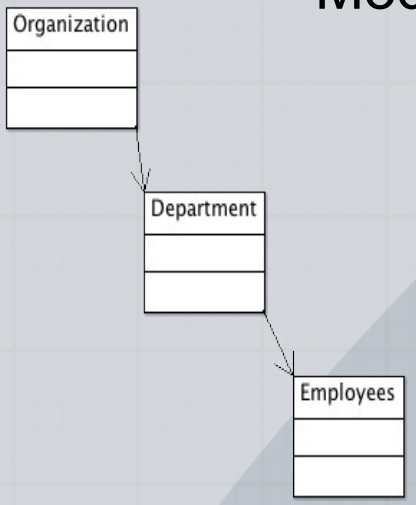
POJO are serialized to JSON using a View Resolver

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

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

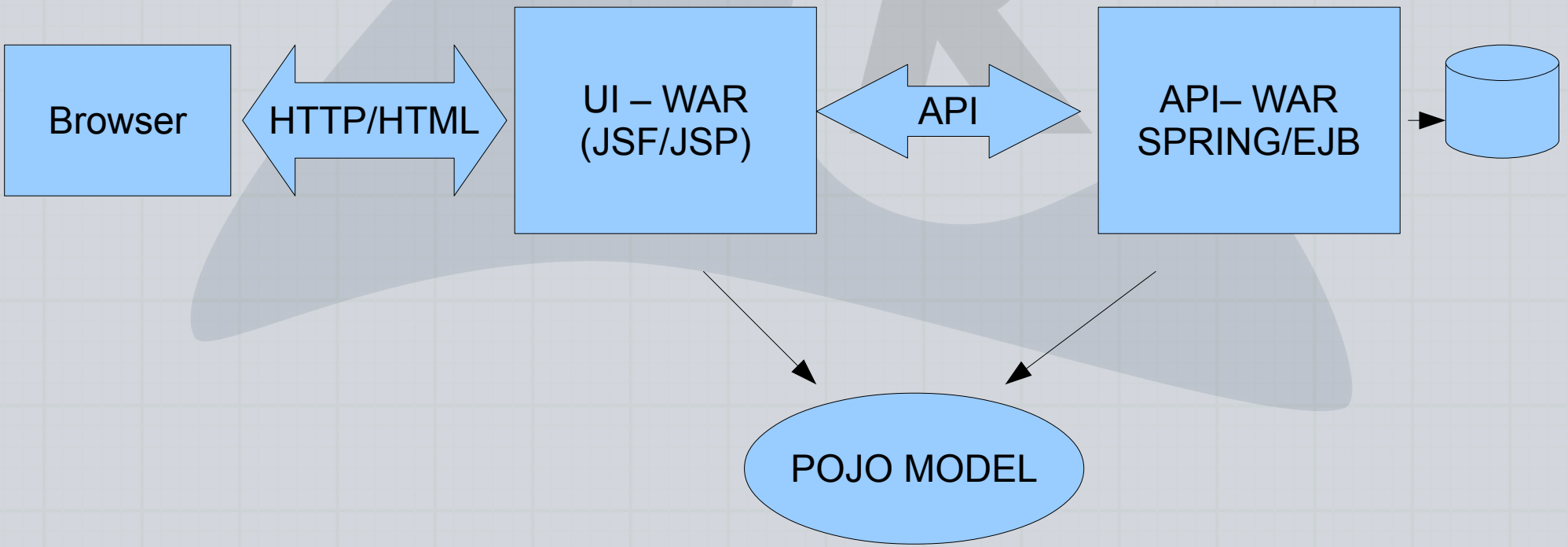
/api/departments/organization/100

Employees for department 200

/api/employees/dept/200

Positioning for SPA

- Learn JavaScript, really learn JavaScript
- Introduce API style programming to existing Dynamic Java HTML application architecture



Fin...

Any questions?