Giving Spring some REST

Craig Walls
craig@habuma.com
Twitter: @habuma @springsocial
http://github.com/habuma

REST in One Slide

Resources (aka, the “things”)

Representations

HTTP Methods (aka, the “verbs”)

URIs and URLs

Why REST?

Key piece of the Modern Application puzzle

More APIs / Fewer “pages”

Humans and browsers consume pages

Everything can consume APIs
(incl. browsers, JS, mobile apps, other apps,etc)

Enables mobile and rich apps

Centerpiece of μService architecture

Spring's REST Story

Spring MVC 3.0+

Spring HATEOAS

Spring Security for OAuth (S2OAuth)

Spring RestTemplate

Spring Social

Spring Data REST

Spring REST Shell

WebSocket/STOMP Support (Spring 4+)

Spring Boot

Today's REST agenda

Creating REST

Securing REST

Streamlining REST

Hyperlinking REST

Consuming REST

https://github.com/habuma/SpringREST

Creating REST APIs

Getting Started

Use start.spring.io and Spring Boot!!!

Demo

There's More Than the Resource...

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(value="/{id}", method=RequestMethod.GET)
   public Book bookById(@PathVariable("id") long id) {
      return bookRepository.findOne(id);
   }

}
```

What will happen if findOne() returns null?

What *should* happen?

There's More Than the Resource...

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(method=RequestMethod.POST)
   public Book postBook(@RequestBody Book book) {
      return bookRepository.save(book);
   }

}
```

What will the HTTP status code be?

What *should* it be?

Returning a ResponseEntity

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(value="/{id}", method=RequestMethod.GET)
   public ResponseEntity<?> bookById(@PathVariable("id") long id) {
   Book book = bookRepository.findOne(id);
   if (book != null) {
     return new ResponseEntity<Book>(book, HttpStatus.OK);
   } else {
     Error error = new Error("Book with ID " + id + " not found");
          return new ResponseEntity<Error>(error, HttpStatus.NOT_FOUND);
   }
   }

}
```

Returning a ResponseEntity

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(value="/{id}", method=RequestMethod.GET)
   public ResponseEntity<Book> bookById(@PathVariable("id") long id) {
   Book book = bookRepository.findOne(id);
    if (book != null) {
      return new ResponseEntity<Book>(book, HttpStatus.OK);
    }
    throw new BookNotFoundException(id);
    }

  @ExceptionHandler(BookNotFoundException.class)
  public ResponseEntity<Error> bookNotFound(BookNotFoundException e) {
         Error error = new Error("Book with ID " + id + " not found");
         return new ResponseEntity<Error>(error, HttpStatus.NOT_FOUND);
  }
}
```

Returning a ResponseEntity

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(value="/{id}", method=RequestMethod.GET)
    public Book bookById(@PathVariable("id") long id) {
        // if findOne() were to throw BookNotFoundException
        return bookRepository.findOne(id);
    }

  @ExceptionHandler(BookNotFoundException.class)
  public ResponseEntity<Error> bookNotFound(BookNotFoundException e) {
          Error error = new Error("Book with ID " + id + " not found");
          return new ResponseEntity<Error>(error, HttpStatus.NOT_FOUND);
  }
}
```

Returning a ResponseEntity

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(method=RequestMethod.POST)
   public ResponseEntity<Book> postBook(@RequestBody Book book) {
      Book newBook = bookRepository.save(book);
      ResponseEntity<Book> bookEntity =
        new ResponseEntity<Book>(newBook, HttpStatus.CREATED);
      String locationUrl =
      ServletUriComponentsBuilder.fromCurrentContextPath().
            path("/books/" + newBook.getId()).build().toUriString();
      bookEntity.getHeaders().setLocation(URI.create(locationUrl));
      return bookEntity;
   }

}
```

Securing the API

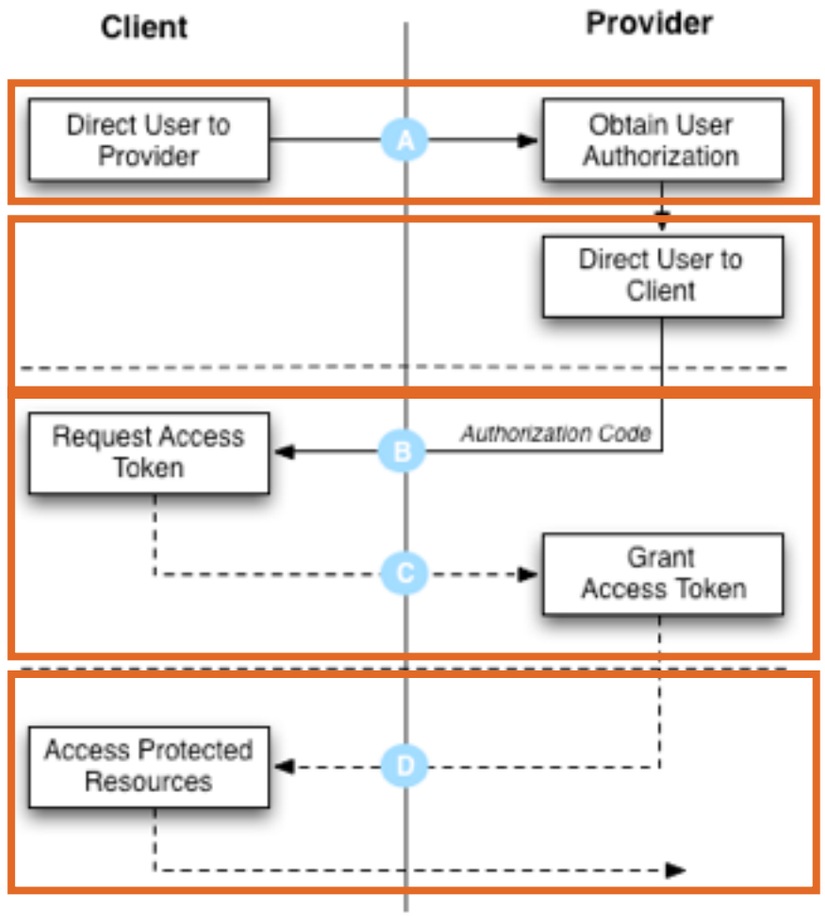
OAuth

- An open standard for authorization
- Supported by Facebook, Twitter, LinkedIn, TripIt, Salesforce, and dozens more
- Puts the user in control of what resources are shared

http://oauth.net

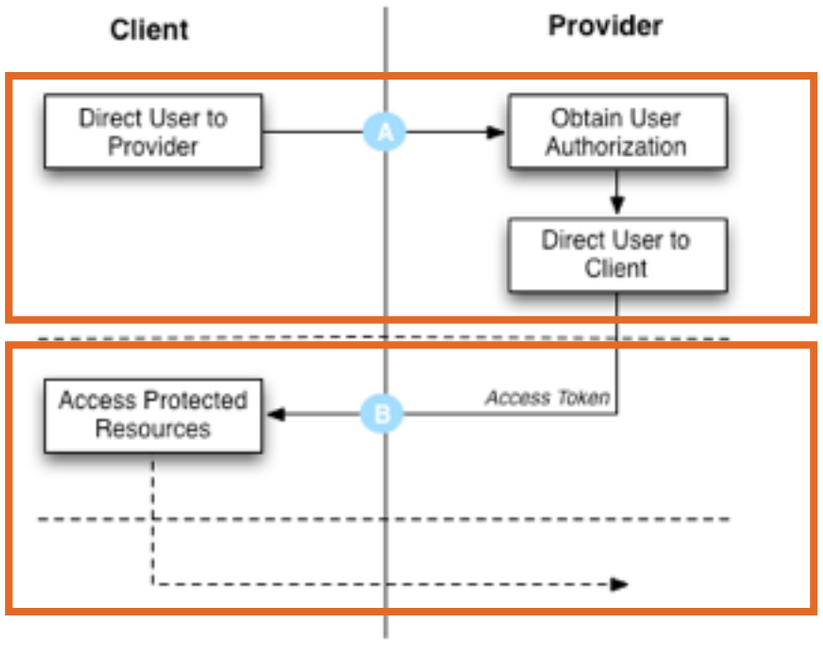
Authorization Code Grant

- Like OAuth 1.0 flow
  - Starts with redirect to provider for authorization
  - After authorization, redirects back to client with code query parameter
  - Code is exchanged for access token
- Client must be able to keep tokens confidential
- Commonly used for web apps



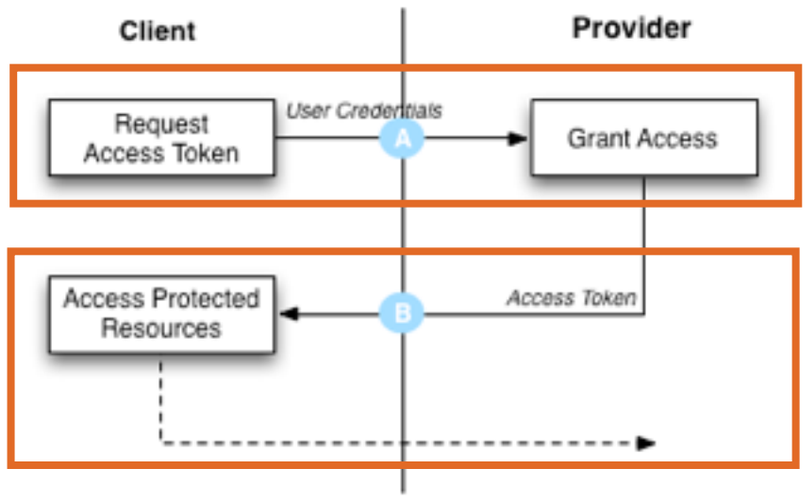
Implicit Grant

- Simplified authorization flow
  - After authorization, redirects back to client with access token in fragment parameter
- Reduced round-trips
- No refresh token support
- Commonly used by in-browser JavaScript apps or widgets



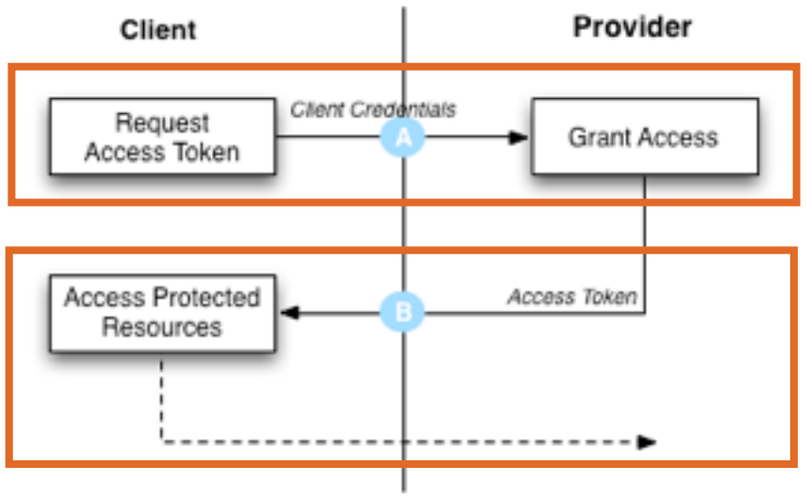
Resource Owner Credentials Grant

- Directly exchanges user's credentials for an access token
- Useful where the client is well-trusted by the user and where a browser redirect would be awkward
- Commonly used with mobile apps



Client Credentials Grant

- Directly exchanges the client's credentials for an access token
- For accessing client-owned resources (with no user involvement)



The OAuth 2 Bearer Token Header

```
Authorization: Bearer e139a950-2fc5-4822-9266-8a2b572108c5
```

OAuth Provider Responsibilities

- Authorization server
  - If supporting authorization code and/or implicit grant, must serve an authorization page
  - Support an authorization endpoint for all supported grant types
  - Not obligated to support all grant types
  - Produce and manage tokens
- Resource server
  - Validate access tokens on requests to resource endpoints

Spring Security OAuth (S2OAuth)

- Based on Spring Security
- Declarative model for OAuth
  - Provider-side support for authorization endpoints, token management, and resource-level security
  - Also offers client-side OAuth
- Implemented for both OAuth 1 and OAuth 2
- http://www.springsource.org/spring-security-oauth

Key Pieces of S2OAuth

- Authorization Server
  - Implemented as Spring MVC controller
  - Handles /oauth/authorize and /oauth/token endpoints
- Resource Server
  - Implemented as servlet filters

Demo

Streamlining the API

PATCHing a Resource (naively)

```
@RestController
@RequestMapping("/books")
public class BooksController {

...

   @RequestMapping(value="/{id}" method=RequestMethod.PATCH)
   public void updateBook(@PathVariable("id") long id,
                          @RequestBody Book bookPatch) {
      Book book = bookRepository.findOne(id);
      // patch from non-null fields
      if (bookPatch.getTitle() != null)
            book.setTitle(bookPatch.getTitle());
      …
      bookRepository.save(book);
   }

}
```

PATCH http://*host*/*app*/books/{id}

What should a PATCH look like?

The spec gives very little guidance

Just a set of instructions

Partial resources not recommended

HTTP Patch: https://tools.ietf.org/html/rfc5789

What does a PATCH look like?

Consider JSON Patch

```
PATCH /todos HTTP/1.1
Host: example.org
Content-Length: 326
Content-Type: application/json-patch+json
If-Match: "abc123"

[
  { "op": "remove", "path": "/4" },
  { "op": "add", "path": "/4", "value": { "description", "Make donuts”, “complete":false } },
  { "op": "test", "path": "/3/description",  "value": "Do TPS Reports" },
  { "op": "replace", "path": "/3/description",  "value": "Send TPS Reports" },
  { "op": "replace", "path": "/3/complete",  "value": true },
  { "op": "move", "from": "/3", "path": "/1" },
  { "op": "copy", "from": "/2/description", "path": "/5/description" }
]
```

HTTP Patch: https://tools.ietf.org/html/rfc5789
JSON Patch: https://tools.ietf.org/html/rfc6902

Benefits of JSON Patch

With typical REST

One endpoint == One resource == One entity

One request == One operation

Create 3 new resources:	3 POSTs
Modify 2 resources:	2 PUTs
Delete 1 resource:	1 DELETE
TOTAL	6 HTTP requests

Benefits of JSON Patch

With JSON Patch over HTTP PATCH

One request could target multiple resources

One request could perform many different operations

All of those operations would be atomic

Create 3 new resources:	3 POSTs
Modify 2 resources:	2 PUTs
Delete 1 resource:	1 DELETE
TOTAL	1 atomic HTTP request

Freeing up Request Threads

With Callable

```
@RequestMapping(method=RequestMethod.GET)
public Callable<List<Book>> allBooks() {
   return new Callable<List<Book>>() {
      @Override
      public List<Book> call() throws Exception {
         return bookRepository.findAll();
      }
   };
}
```

With DeferredResult

```
@RequestMapping(method=RequestMethod.GET)
public DeferredResult<List<Book>> allBooks() {
   DeferredResult<List<Book>> result = new DeferredResult<List<Book>>();

   // Deferred result is pushed into a queue and sometime later,
   // someone does:
   //       result.setResult(bookRepository.findAll());

   return result;
}
```

Hyperlinking the API

Linking Resources

HATEOAS

Hypermedia As The Engine Of Application State

aka, "Connectedness"

Responses carry links to related endpoints

API is self-descriptive

Client can "learn" about the API

Self-Describing API: Default

```
{
  "links" : [
    {
      "rel" : "self",
      "href" : "http://localhost:8080/BookApp/books/5"
    },
    {
      "rel" : "all",
      "href" : "http://localhost:8080/BookApp/books/"
    },
    {
      "rel" : "author",
      "href" : "http://localhost:8080/BookApp/authors/2"
    }
  ],

  "id" : 5,
  "title" : "Spring in Action",
  ...
}
```

Self-Describing API: HAL

```
{
  "_links" : {
    "self": {
      "href" : "http://localhost:8080/BookApp/books/5"
    },
    "all": {
      "href" : "http://localhost:8080/BookApp/books/"
    },
    "author": {
      "href" : "http://localhost:8080/BookApp/authors/2"
    }
  },

  "id" : 5,
  "title" : "Spring in Action",
  ...
}
```

Enabled with

`@EnableHypermediaSupport(type=HypermediaType.HAL)`

Spring HATEOAS

Convenient support for HATEOAS-ifying an API

https://github.com/SpringSource/spring-hateoas

Defining a Resource

```
public class BookResource extends ResourceSupport {
   ...
   public String getTitle() {…}
   public String getIsbn() {…}
}
```

Adding Links to a Resource

```
@RequestMapping(value="/{id}", method=RequestMethod.GET)
public ResponseEntity<BookResource> bookById(@PathVariable("id") long id) {
  Book book = bookRepository.findOne(id);
  if (book != null) {

    BookResource resource = bookResourceAssembler.toResource(book);
    resource.add(ControllerLinkerBuilder.linkTo(BooksController.class)
        .withRel("all"));
   resource.add(ControllerLinkerBuilder.linkTo(AuthorsController.class)
       .slash(book.getAuthor().getId());
     .withRel("author");

    return new ResponseEntity<BookResource>(resource, HttpStatus.OK);
  }

  throw new BookNotFoundException(id);
}
```

Adding Links to a Resource (2)

```
@RequestMapping(value="/{id}", method=RequestMethod.GET)
public ResponseEntity<BookResource> bookById(@PathVariable("id") long id) {
  Book book = bookRepository.findOne(id);
  if (book != null) {

    BookResource resource = bookResourceAssembler.toResource(book);
    resource.add(ControllerLinkerBuilder.linkTo(
        methodOn(BooksController.class).allBooks()).withRel("all"));
   resource.add(ControllerLinkerBuilder.linkTo(
       methodOn(AuthorsController.class)
           .authorById(book.getAuthor().getId())
       .withRel("author");

    return new ResponseEntity<BookResource>(resource, HttpStatus.OK);
  }

  throw new BookNotFoundException(id);
}
```

Assembling a Resource

```
public class BookResourceAssembler
       extends ResourceAssemblerSupport<Book, BookResource> {

  public BookResourceAssembler() {
    super(BooksController.class, BookResource.class);
  }

  public BookResource toResource(Book book) {
    return createResourceWithId(book.getId(), book);
  }

}
```

Consuming the API

RestTemplate

Handles boilerplate HTTP connection code

Keeps your focus on the resources

Using RestTemplate

```
RestTemplate restTemplate = new RestTemplate();
Book book = new Book("Spring in Action", "Gregg Walls");

Book newBook = restTemplate.postForObject(
                          "http://host/app/books", book, Book.class);

book = restTemplate.getForObject(
            "http://host/app/books/{id}", Book.class, newBook.getId());

book.setAuthor("Craig Walls");
restTemplate.put("http://host/app/books/{id}",
                 book, book.getId());

restTemplate.delete("http://host/app/books/{id}", book.getId());
```

Tweeting with RestTemplate

```
RestTemplate rest = new RestTemplate();
MultiValueMap<String, Object> params =
      new LinkedMultiValueMap<String, Object>();
params.add("status", "Hello Twitter!");
rest.postForObject("https://api.twitter.com/1/statuses/update.json",
      params, String.class);
```

Oh no!

```
WARNING: POST request for "https://api.twitter.com/1/statuses/update.json" resulted in 401 (Unauthorized); invoking error
handler
org.springframework.web.client.HttpClientErrorException: 401 Unauthorized
    at org.springframework.web.client.DefaultResponseErrorHandler.handleError(DefaultResponseErrorHandler.java:75)
    at org.springframework.web.client.RestTemplate.handleResponseError(RestTemplate.java:486)
    at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:443)
    at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:401)
    at org.springframework.web.client.RestTemplate.postForObject(RestTemplate.java:279)
```

Configuring Spring Social

```
@Configuration
@EnableSocial
public class SocialConfig implements SocialConfigurer {

  @Inject
  private DataSource dataSource;

  @Override
  public void addConnectionFactories(ConnectionFactoryConfigurer cfConfig, Environment env) {
    cfConfig.addConnectionFactory(
        new TwitterConnectionFactory(
            env.getProperty("twitter.appKey"), env.getProperty(“twitter.appSecret")));
  }

  @Override
  public UsersConnectionRepository getUsersConnectionRepository(ConnectionFactoryLocator cfl) {
    return new JdbcUsersConnectionRepository(dataSource, cfl, Encryptors.noOpText());
  }

  @Bean
  @Scope(value=“request”, proxyMode=ScopedProxyMode.INTERFACES)
  public Twitter twitter(ConnectionRepository repo) {
    Connection<Twitter> conn = repo.findPrimaryConnection(Twitter.class);
    return conn != null ? conn.getApi() : null;
  }

  @Bean
  public ConnectController connectController(
      ConnectionFactoryLocator cfl, ConnectionRepository repo) {
    return new ConnectController(cfl, repo);
  }
}
```

Inject API binding and Tweet

```
public class TwitterTimelineController {

   private final Twitter twitter;

   @Inject
   public TwitterTimelineController(Twitter twitter) {
      this.twitter = twitter;
   }

   @RequestMapping(value="/twitter/tweet", method=RequestMethod.POST)
   public String postTweet(String message) {
      twitter.timelineOperations().updateStatus(message);
      return "redirect:/twitter";
   }

}
```

Thank you!

Flights Sample (a work in progress)
https://github.com/habuma/SpringREST

Spring Boot
http://projects.spring.io/spring-boot

Spring HATEOAS
http://projects.spring.io/spring-hateoas

Spring Security for OAuth (S2OAuth)
http://projects.spring.io/spring-security-oauth

Spring Social
http://projects.spring.io/spring-social

Spring Data REST
http://projects.spring.io/spring-data-rest

Spring REST Shell
https://github.com/spring-projects/rest-shell