Exposing a REST Service

Antonio Goncalves
JAVA CHAMPION

@agoncal www.antoniogoncalves.org

Previous Module

- Context and Dependency Injection (CDI)
- Inject beans into others
- Loosely coupled
- Type safe
- One implementation
- Several implementations

Overview

Expose REST API

Understanding REST Services

JAX-RS

Expose and consume REST services

Testing with Arquillian

Add data to the database

What Is REST?

Representational State Transfer

Architectural style

HTTP

Very robust transport protocol

What Is JSON?

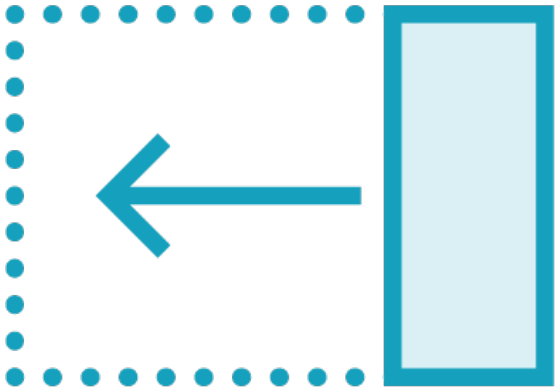
JavaScript Object Notation

Originated with JavaScript

Serializing and transmitting structured data

Web browsers have native encoding decoding

What Is JAX-RS?



- Heavily relies on HTTP
- Annotations
- URI
- Representation
- Uniform interface
- HTTP status code

What Is JAX-RS?

Resources and URIs

www.bookstore.com
/books

Representations

JSon

Uniform Interface

POST, GET, PUT,
DELETE

A Book Endpoint

```
@Path("/books")
public class BookEndpoint {

  @Inject
  private BookRepository bookRepository;

  @GET
  @Produces(APPLICATION_JSON)
  public Response getBooks() {
    List<Book> books = bookRepository.findAll();

    if (books.size() == 0)
      return Response.status(Response.Status.NO_CONTENT).build();

    return Response.ok(books).build();
  }
```

http://www.bookstore.com/**books**

Extracting Parameters

```
@Path("/books")
public class BookEndpoint {

  // ...

  @GET
  @Produces(APPLICATION_JSON)
  @Path("/{id       }")
  public Response getBook(                Long id) {
    Book book = bookRepository.find(id);

    if (book == null)
      return Response.status(Response.Status.NOT_FOUND).build();

    return Response.ok(book).build();
  }
```

```
GET    http://www.bookstore.com/books
```

HTTP Delete

```
@Path("/books")
public class BookEndpoint {

  // ...

  @DELETE
  @Path("/{id : \\d+}")
  public Response deleteBook(@PathParam("id") Long id) {
    bookRepository.delete(id);
    return Response.noContent().build();
  }
}
```

http://www.bookstore.com/**books/123**

HTTP Post

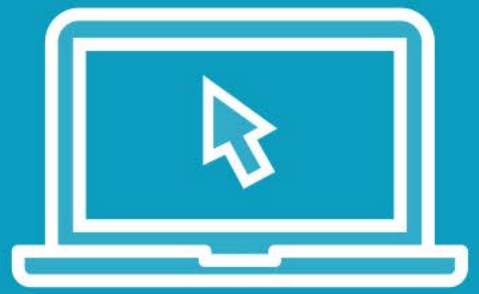
```
@Path("/books")
public class BookEndpoint {

  // ...

  @POST
  @Consumes(APPLICATION_JSON)
  public Response createBook(Book book,                                  ) {
    book = bookRepository.create(book);
    URI createdURI =
        uriInfo.getBaseUriBuilder().path(book.getId()).build();
    return Response.created(createdURI).build();
  }
}
```

http://www.bookstore.com/**books**

Demo



Expose a REST API

HTTP GET, POST and DELETE

CRUD operations

Populate the database

HTTP client

Consuming the Book Endpoint

- Automate HTTP invocations
- Client API
- HTTP requests
- Fluent building API
- Classes and interfaces

Client API

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://www.bookstore.com/books/123");
Invocation invocation = target.request(MediaType.APPLICATION_JSON).buildGet();
Response response = invocation.invoke();
```

http://www.bookstore.com/**books/123**

Client API

```
Response response = ClientBuilder.newClient()
                        .target("http://www.bookstore.com/books/123")
                        .request(MediaType.APPLICATION_JSON)
                        .get();
```

```
GET        http://www.bookstore.com/books/123
```

Client API

```
Response response = ClientBuilder.newClient()
                        .target("http://www.bookstore.com/books/123")
                        .request(MediaType.APPLICATION_JSON)
                        .get();
```

DELETE http://www.bookstore.com/**books/123**

Client API

```
Response response = ClientBuilder.newClient()
                        .target("http://www.bookstore.com/books/123")
                        .request(MediaType.APPLICATION_JSON)
                        .delete();
```

DELETE http://www.bookstore.com/**books/123**

Response

```
Response response = ClientBuilder.newClient()
                        .target("http://www.bookstore.com/books/123")
                        .request(MediaType.APPLICATION_JSON)
                        .delete();

assertTrue(response.getStatusInfo() == Response.Status.OK);
assertTrue(response.getLength() == 4);
assertTrue(response.getDate() != null);
assertTrue(response.getHeaderString("Content-type").equals("application/json"));
```

```
DELETE    http://www.bookstore.com/books/123
```

Response

```
Response response = ClientBuilder.newClient()
                        .target("http://www.bookstore.com/books/123")
                        .request(MediaType.APPLICATION_JSON)
                        .get();

assertTrue(response.getStatusInfo() == Response.Status.OK);
assertTrue(response.getLength() == 4);
assertTrue(response.getDate() != null);
assertTrue(response.getHeaderString("Content-type").equals("application/json"));

String body = response.readEntity(String.class);
Book book = response.readEntity(Book.class);
```

```
GET        http://www.bookstore.com/books/123
```

Arquillian Client Tests

Package the business code

~~Package test classes~~

Into an archive

Deploy to WildFly

~~Execute tests inside the container~~

GET Method

```
GET    http://www.bookstore.com/books/123
```

```java
@Path("/books")
public class BookEndpoint {

  @GET
  @Produces(APPLICATION_JSON)
  @Path("/{id : \\d+}")
  public Response getBook(@PathParam("id") Long id) {
    Book book = bookRepository.find(id);

    if (book == null)
      return Response.status(Response.Status.NOT_FOUND).build();

    return Response.ok(book).build();
  }
}
```

Testing GET Method

```
GET    http://www.bookstore.com/books/123
```

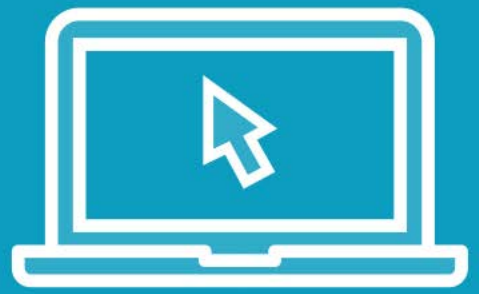
```java
@RunWith(Arquillian.class)
@RunAsClient
public class BookEndpointTest {

  @Deployment
  public static Archive<?> createDeploymentPackage() {
    return ShrinkWrap.create(WebArchive.class).addClass(...);
  }

  @Test
  public void shouldFindTheCreatedBook(
        @ArquillianResteasyResource("books") WebTarget webTarget) {

    response = webTarget.path("123").request(APPLICATION_JSON).get();
    assertEquals(OK.getStatusCode(), response.getStatus());
  }
}
```

Demo



Arquillian test

BookEndpointTest

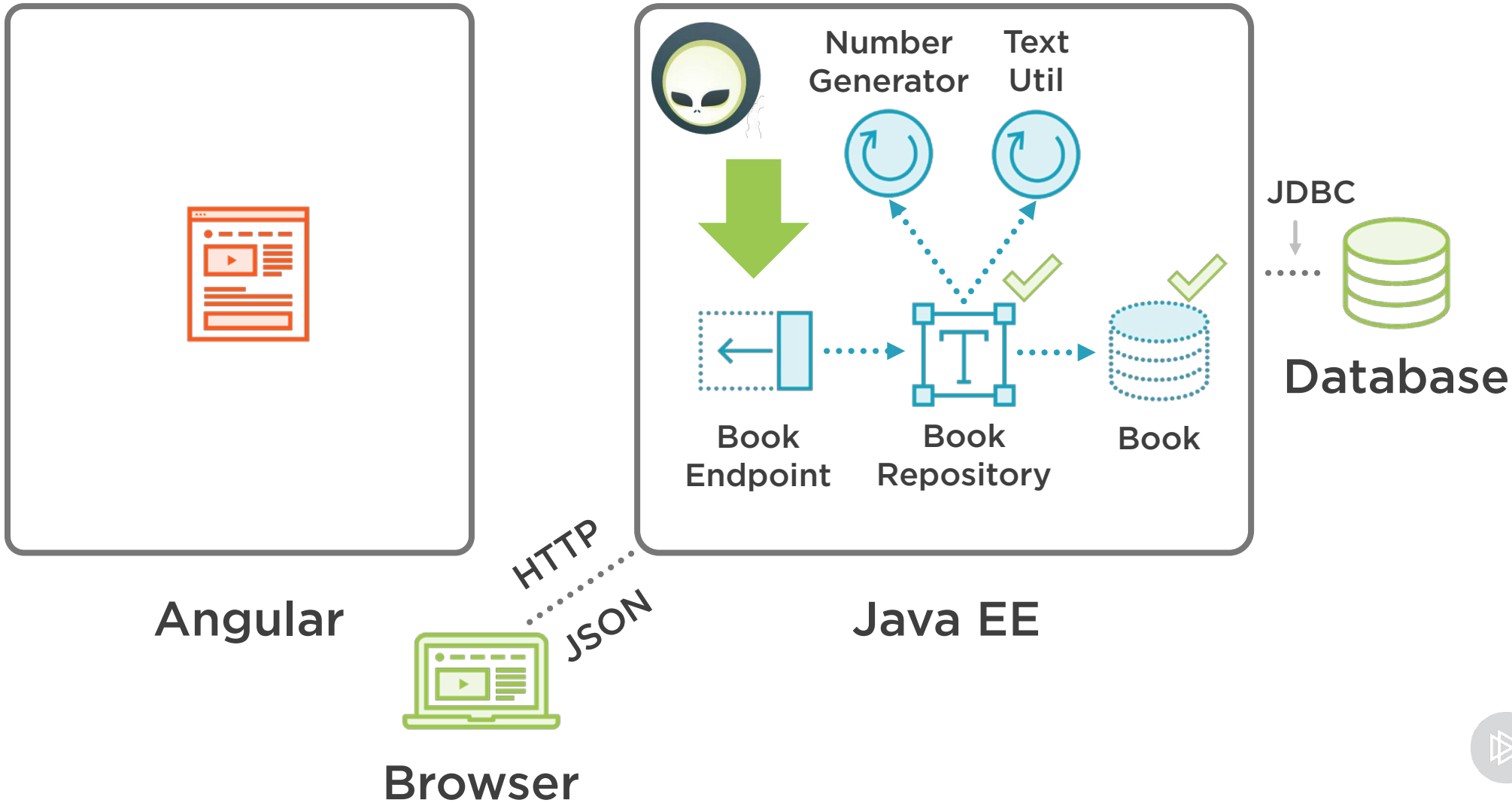
Client test

JAX-RS Client API

Access remotely

Black box testing

Anatomy of the BookStore Application



Summary

- Exposing a REST API using annotations
- Consume it with a client API
- HTTP endpoint
- Entry point for Angular
- Added data to the database
- Client Arquillian test

Next Module

Document the BookEndpoint API

REST contract

- HTTP methods
- Method parameters
- Returned response

Generate Angular services

Access our Java EE back-end