Lab 1: OpenAPI in Liberty Server

Introduction

The purpose of this lab is to show you how to expose and explore REST APIs that is provided by a JAX-RS application. The sample application that will be used is a simple airlines app in which the user can create an account, book airlines tickets, view available seats and provide reviews to express their experience on any airlines.

The API Economy is the economy where companies expose their (internal) business assets or services in the form of (Web) APIs to third parties with the goal of unlocking additional business value through the creation of new asset classes.

With WebSphere Liberty's OpenAPI feature, it is very easy to document and explore your REST APIs for your Java apps; This is done using JAX-RS specification.

WebSphere Liberty now provides the way to:

- **Document Java APIs** using the new OpenAPI Specification (OAS),
- **Discover Java APIs** on a Liberty server

The following symbol appears in this document at places where additional guidance is available.

Icon	Purpose	Explanation
ℹ	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.

1.1 Deploy a JAX-RS Application and Explore APIs

ℹ

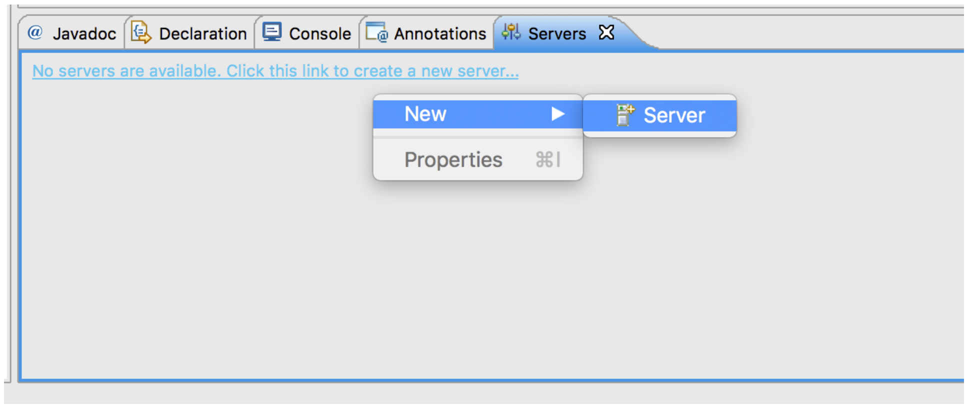
The Eclipse version installed on this CASCON workshop machine has the WebSphere Developer Tools (WDT) plugin. WAS Liberty is also installed and the path to the installation directory is already configured within Eclipse.

Instructions to install WDT and to install/configure WAS Liberty can be found on the Get started with a Hello World app on Liberty using WebSphere Developer Tools for Eclipse article on WASdev.net. WDT plugin is also available for free from the Eclipse Marketplace.

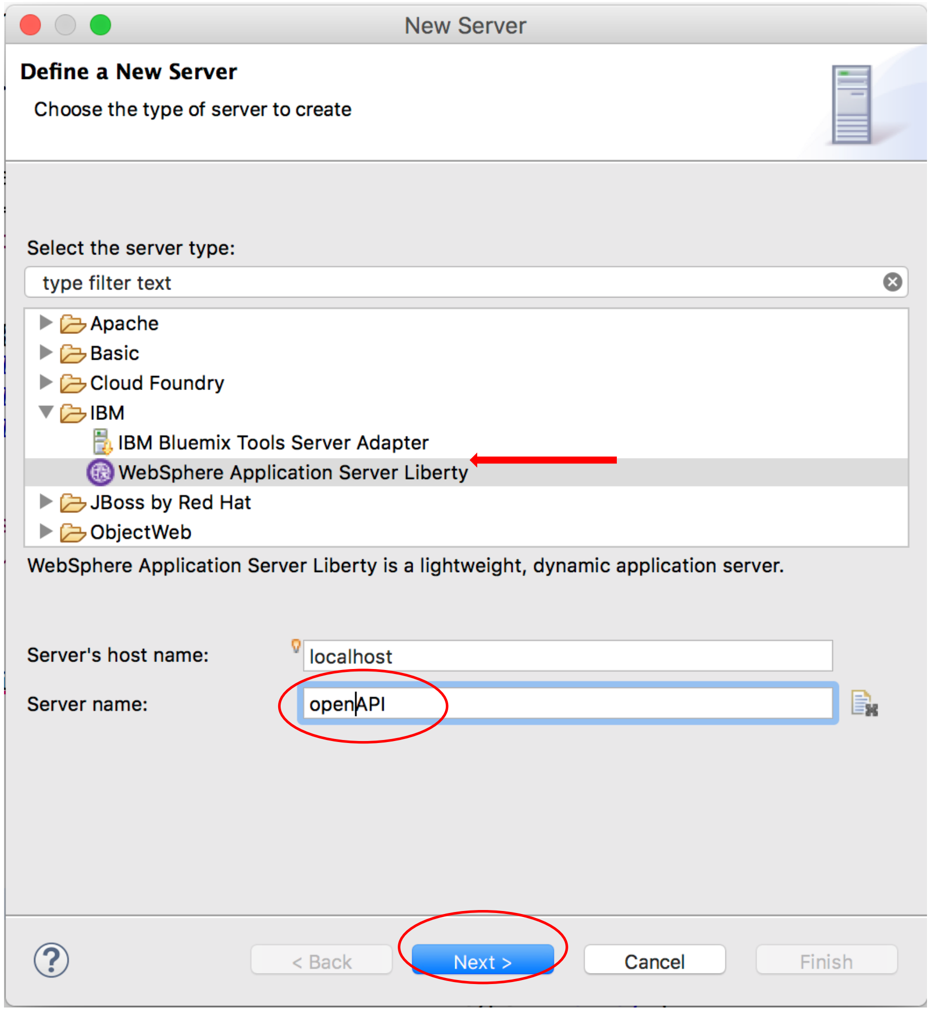
1. Start Eclipse

2. Create a new Liberty server call `openAPI` using the runtime environment provided

    a. Open the Servers view, and create a new server


Javadoc
Declaration
Console
Annotations
Servers
No servers are available. Click this link to create a new server...
New
Server
Properties ⌘I

b. Create a new "**WebSphere Application Server Liberty**" server representation with the name `openAPI`. Hit **Next**.



New Server

Define a New Server

Choose the type of server to create

Select the server type:

type filter text

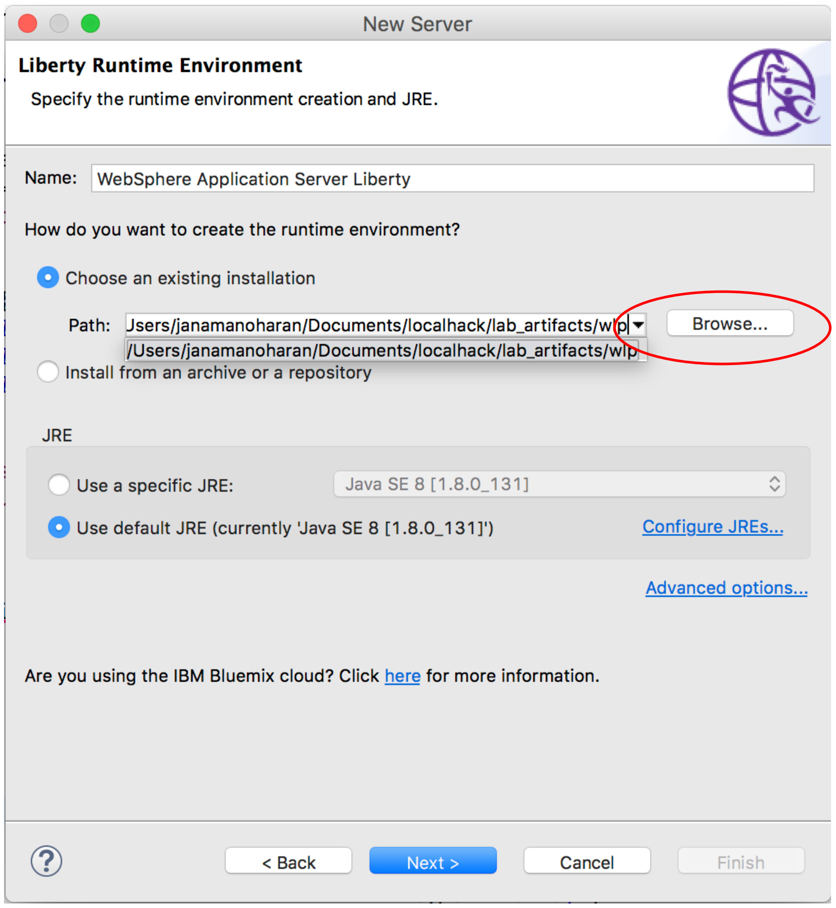
- Apache
- Basic
- Cloud Foundry
- IBM
  - IBM Bluemix Tools Server Adapter
  - WebSphere Application Server Liberty
- JBoss by Red Hat
- ObjectWeb

WebSphere Application Server Liberty is a lightweight, dynamic application server.

Server's host name: localhost

Server name: openAPI

c. For the runtime environment, choose the wlp folder that you downloaded into the `lab_artifacts` folder previously, and hit **Next**



New Server

Liberty Runtime Environment

Specify the runtime environment creation and JRE.

Name: WebSphere Application Server Liberty

How do you want to create the runtime environment?

● Choose an existing installation

Path: Jsers/janamanoharan/Documents/localhack/lab_artifacts/wlp

/Users/janamanoharan/Documents/localhack/lab_artifacts/wlp

○ Install from an archive or a repository

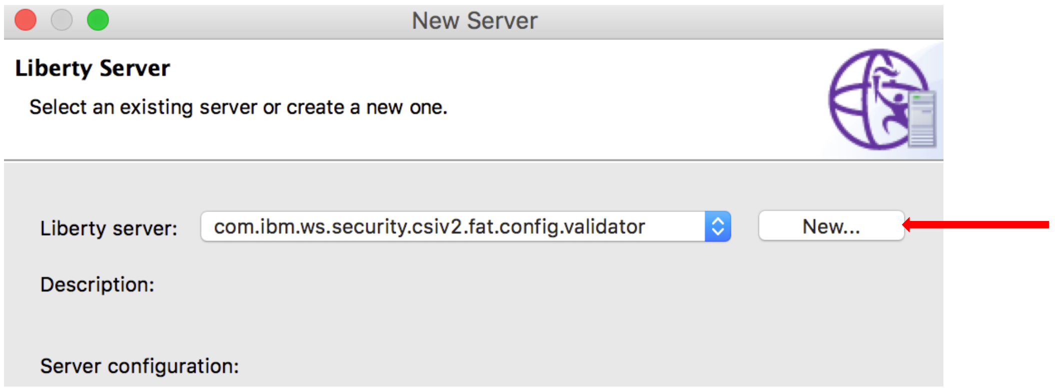
JRE

○ Use a specific JRE: Java SE 8 [1.8.0_131]

● Use default JRE (currently 'Java SE 8 [1.8.0_131]')

Are you using the IBM Bluemix cloud? Click here for more information.

d. Click **New…** to create a new Liberty server instance.



New Server

Liberty Server

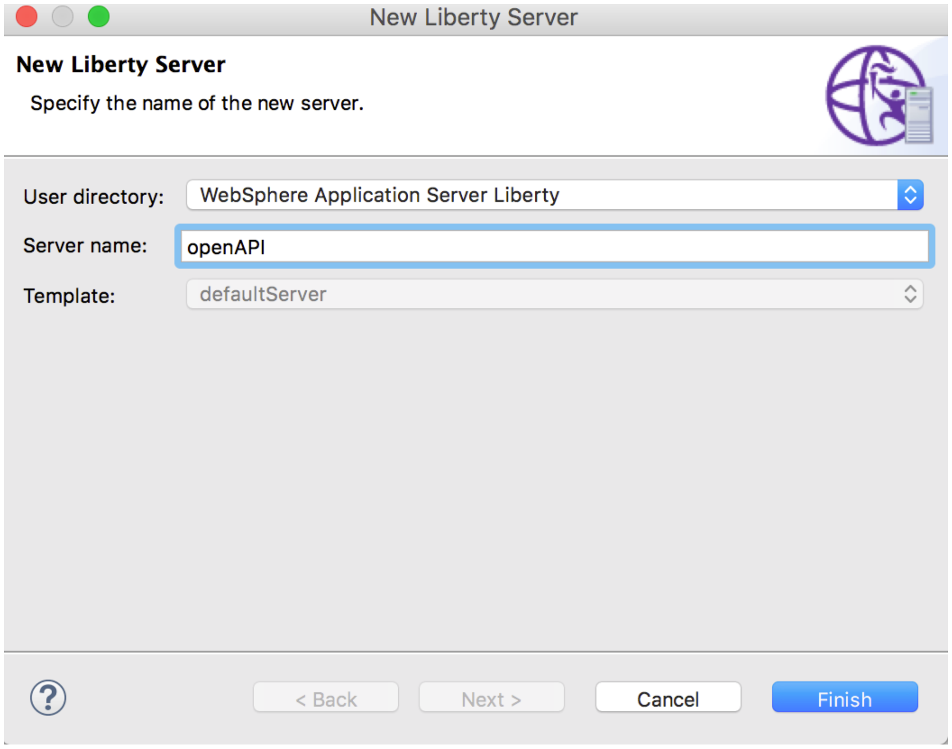
Select an existing server or create a new one.

Liberty server: com.ibm.ws.security.csiv2.fat.config.validator

Description:

Server configuration:

e. Specify the server name `openAPI` and click **Finish**. And click **Finish** again.



New Liberty Server

New Liberty Server

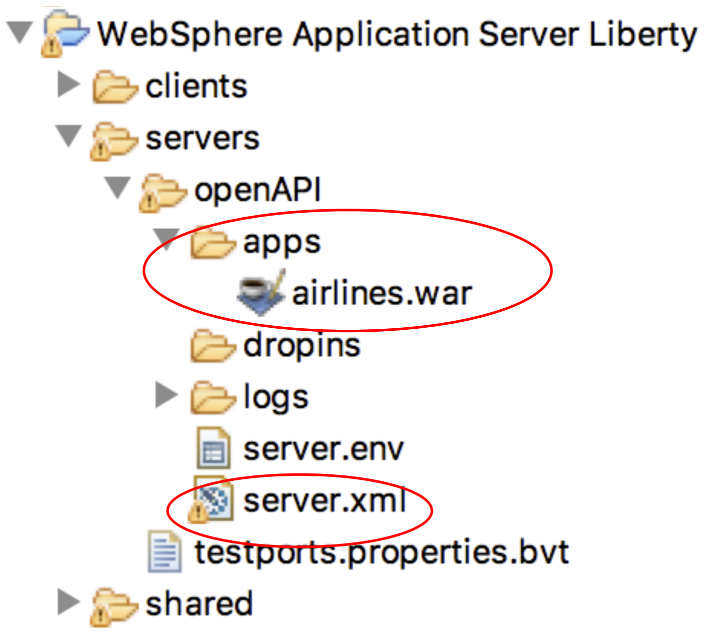
Specify the name of the new server.

User directory: WebSphere Application Server Liberty

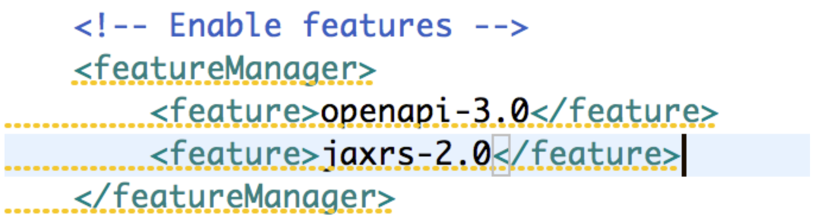
Server name: openAPI

Template: defaultServer

3. Copy lab artifacts to the `openAPI` server. This lab relies on a pre-existing application and its associated configuration. In the Project Explorer tab
   a. Copy the **airlines.war** from lab artifacts directory to the **…/wlp/usr/servers/openAPI/apps** directory
   b. Copy the **server.xml** from lab artifacts directory to **…/wlp/usr/servers/openAPI** directory and ***replace*** the existing **server.xml**



4. Understand the **server.xml** file
   a. The `jaxrs-2.0` feature allows us to support JAX-RS programming model and make http requests to our remote REST web services
   b. The `openapi-3.0` feature allows us to document our REST APIs for the airlines app in an organized manner using the OpenAPI Specification.



```
<!-- Enable features -->
<featureManager>
        <feature>openapi-3.0</feature>
        <feature>jaxrs-2.0</feature>
</featureManager>
```

   c. The port that we will be using will be 9080 and 9443

```
<!-- To access this server from a remote client add a host attribute to the following element, e.g. host="*" -->
<httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint"/>
```

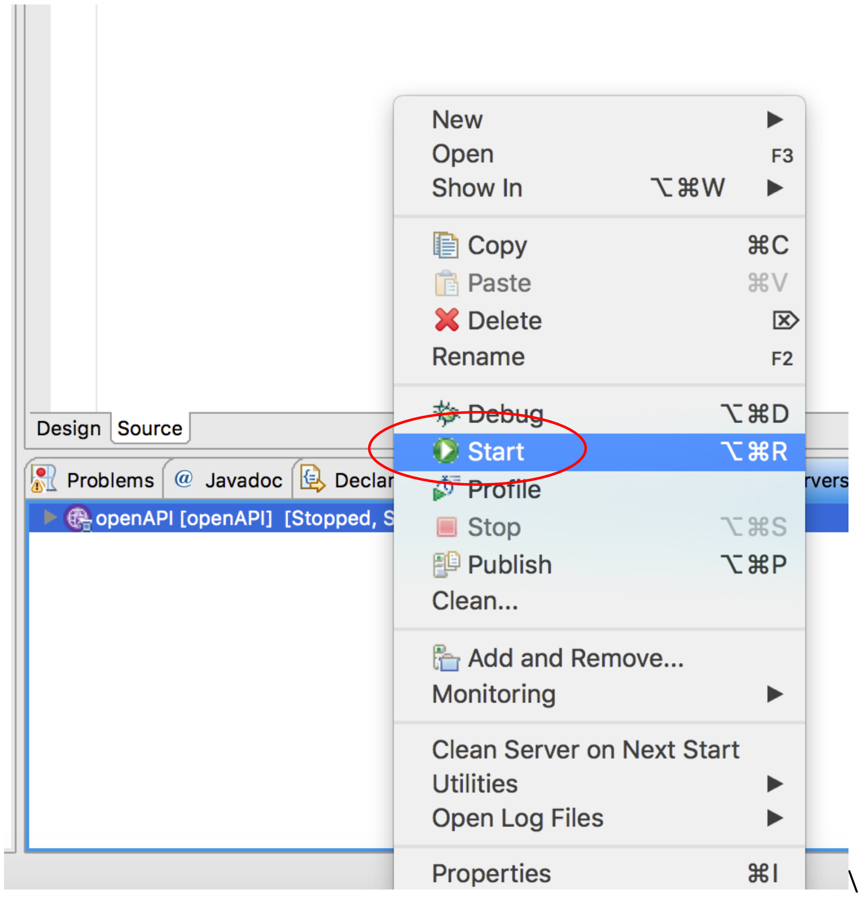
d. The `openAPI` server will be hosting the airlines application by monitoring the **airlines.war** file in the **apps** folder.

e. By including the `third-party` value in `apiTypeVisibility`, we can now use the third party OpenAPI annotations from open source to document our REST APIs

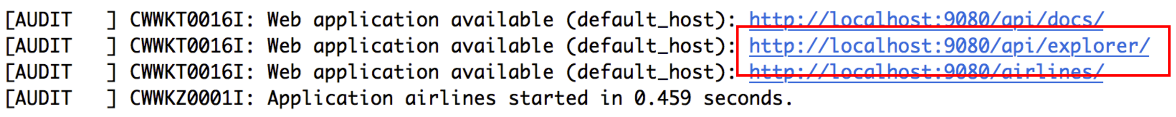
```
<webApplication id="airlines" location="airlines.war" name="airlines">
    <classloader apiTypeVisibility="spec,ibm-api,api,stable,third-party"/>
</webApplication>
```

5. Start the openAPI server and use the API Explorer to access the REST API for the airlines app

a. In the Servers view, right-click the server in the `openAPI` server, and click **Start**

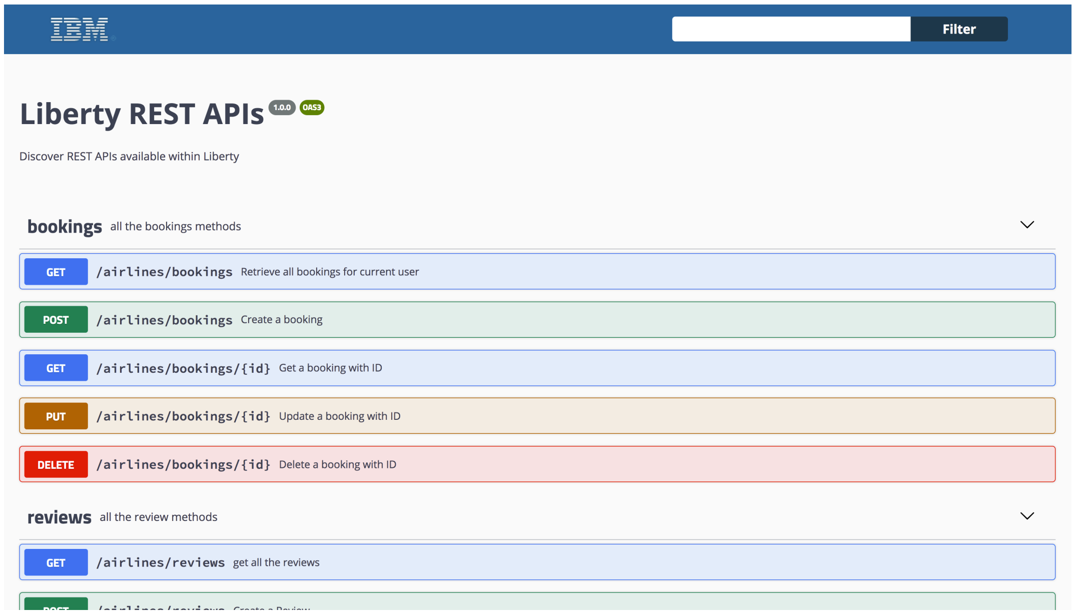


b. To access the API Explorer, click the link in the Console view.



```
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/api/docs/
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/api/explorer/
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/airlines/
[AUDIT   ] CWWKZ0001I: Application airlines started in 0.459 seconds.
```

c. In the API Explorer, all of the REST APIs provided by the airlines application will be listed.



Liberty REST APIs

Discover REST APIs available within Liberty

bookings all the bookings methods

GET /airlines/bookings Retrieve all bookings for current user

POST /airlines/bookings Create a booking

GET /airlines/bookings/{id} Get a booking with ID

PUT /airlines/bookings/{id} Update a booking with ID

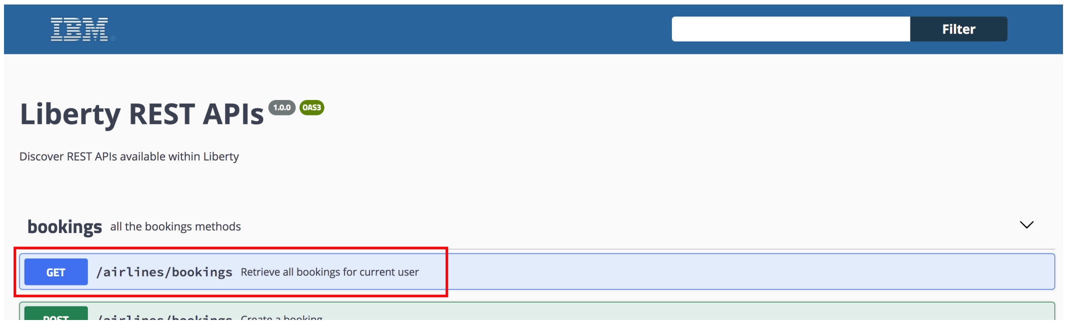
DELETE /airlines/bookings/{id} Delete a booking with ID

reviews all the review methods

GET /airlines/reviews get all the reviews

POST /airlines/reviews Create a Review

d. Click the GET /airlines/bookings resource to expand and get more details on this specific API



Liberty REST APIs

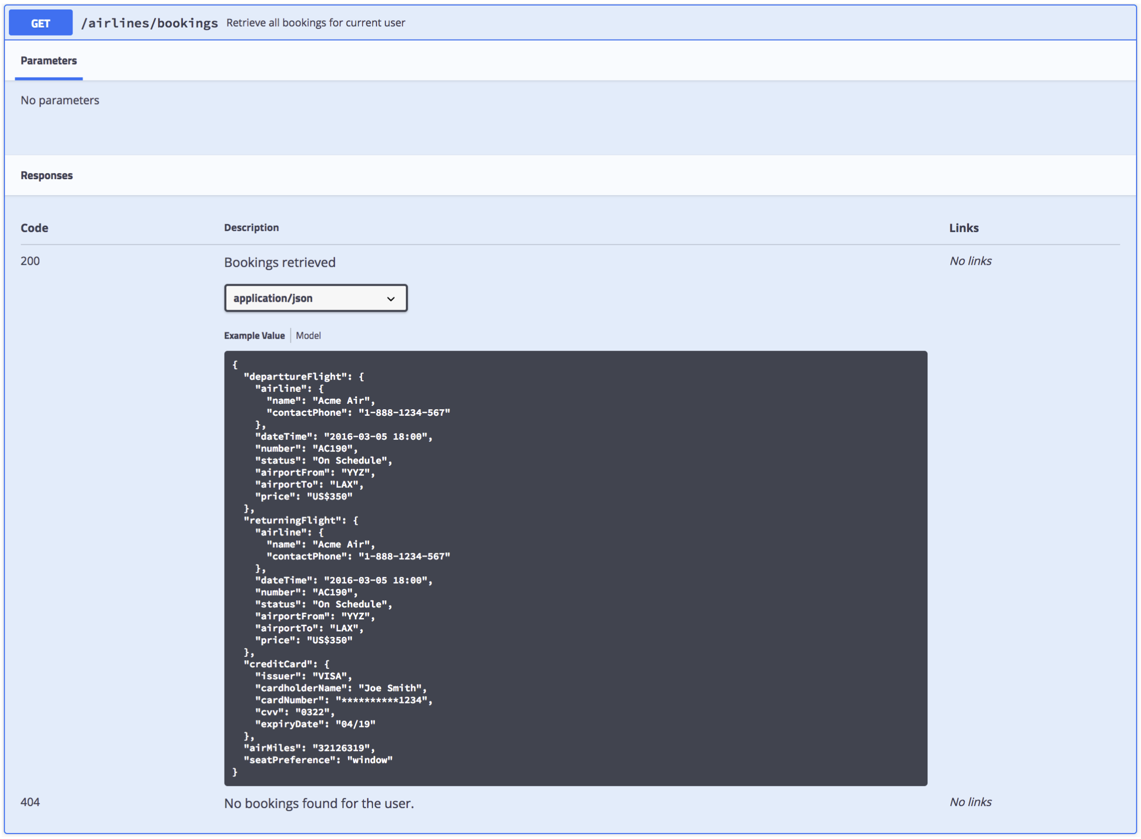
Discover REST APIs available within Liberty

bookings all the bookings methods

GET /airlines/bookings Retrieve all bookings for current user

POST /airlines/bookings Create a booking

e. Upon expanding GET /airlines/booking, you can see what this API offers
   i. The name of the API and what service it provides
   ii. Any parameters that the API may take, in this case this GET method does not take any parameters
   iii. All the possible responses along with a sample response of how each would look like



`/airlines/bookings` Retrieve all bookings for current user

No parameters

Code	Description	Links
200	Bookings retrieved	No links
404	No bookings found for the user.	No links

application/json

```
{
  "departtureFlight": {
    "airline": {
      "name": "Acme Air",
      "contactPhone": "1-888-1234-567"
    },
    "dateTime": "2016-03-05 18:00",
    "number": "AC190",
    "status": "On Schedule",
    "airportFrom": "YYZ",
    "airportTo": "LAX",
    "price": "US$350"
  },
  "returningFlight": {
    "airline": {
      "name": "Acme Air",
      "contactPhone": "1-888-1234-567"
    },
    "dateTime": "2016-03-05 18:00",
    "number": "AC190",
    "status": "On Schedule",
    "airportFrom": "YYZ",
    "airportTo": "LAX",
    "price": "US$350"
  },
  "creditCard": {
    "issuer": "VISA",
    "cardholderName": "Joe Smith",
    "cardNumber": "**********1234",
    "cvv": "0322",
    "expiryDate": "04/19"
  },
  "airMiles": "32126319",
  "seatPreference": "window"
}
```

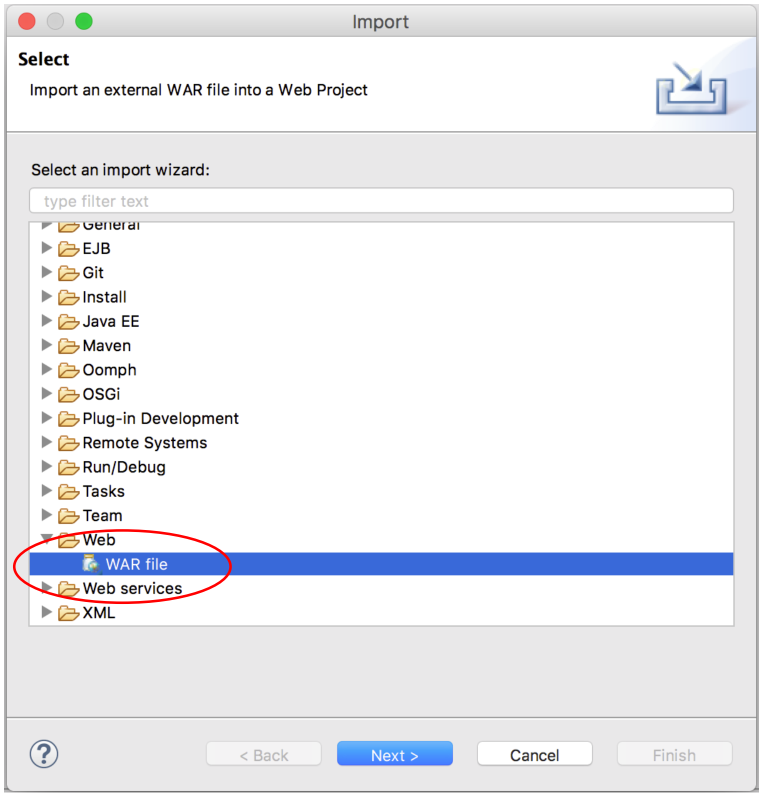
f. Take some time to play with the available REST APIs and the API Explorer. Next, we will look at the source code for the airlines app and see how the OpenAPI documentation can be defined within the application source code.

You have just experienced how easy it is to use the OpenAPI feature and the API Explorer in Liberty to discover and work with APIs running on a Liberty server.

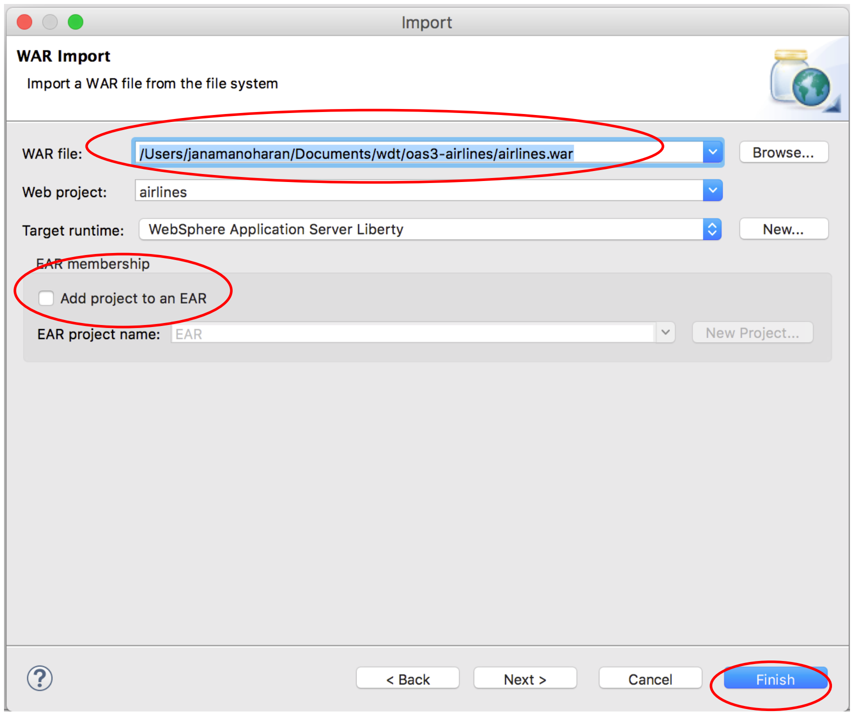
In the next section of the lab, you will explore configurations and the OpenAPI annotations of the airlines application that made the REST API exploration possible.

1.2 Examine JAX-RS Application Source and OpenAPI Annotations

1. The **airlines.war** from the lab artifacts includes the application source. You can explore the source by importing the WAR into Eclipse.
    a. In Eclipse, click File -> Import -> Web -> WAR file



b. Choose the **airlines.war** file from the lab artifacts. Make sure the “Add project to an EAR” (Enterprise Archive) is unchecked and hit Finish.



Import

WAR Import

Import a WAR file from the file system

WAR file: /Users/janamanoharan/Documents/wdt/oas3-airlines/airlines.war

Web project: airlines

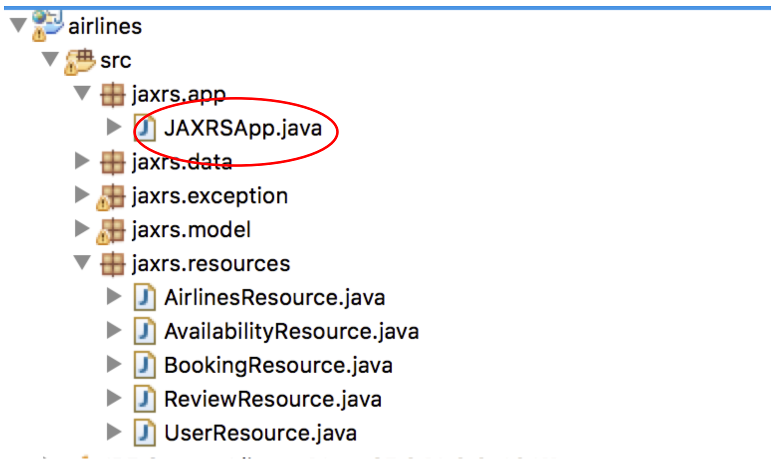
Target runtime: WebSphere Application Server Liberty

EAR membership

☐ Add project to an EAR

EAR project name: EAR

c. Open the **JAXRSApp.java** file

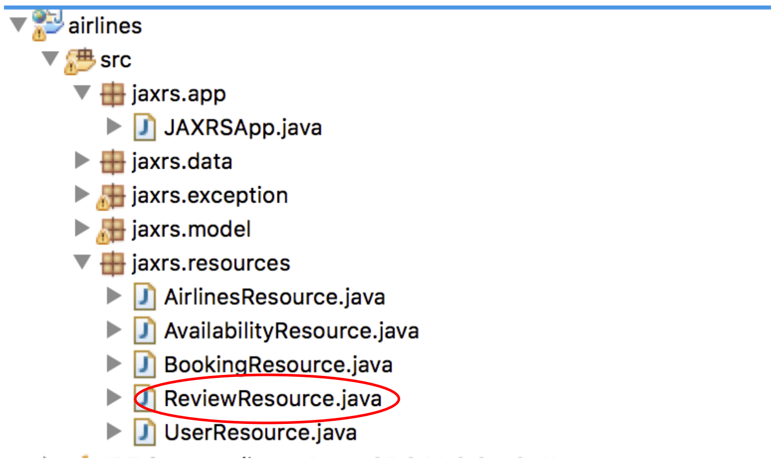


```
airlines
  src
    jaxrs.app
      JAXRSApp.java
    jaxrs.data
    jaxrs.exception
    jaxrs.model
    jaxrs.resources
      AirlinesResource.java
      AvailabilityResource.java
      BookingResource.java
      ReviewResource.java
      UserResource.java
```

d. Review the top-level annotation. The `@OpenAPIDefinition` is used to name and describe the API, this includes the name, contact information, version number and any other external documentation it may use.

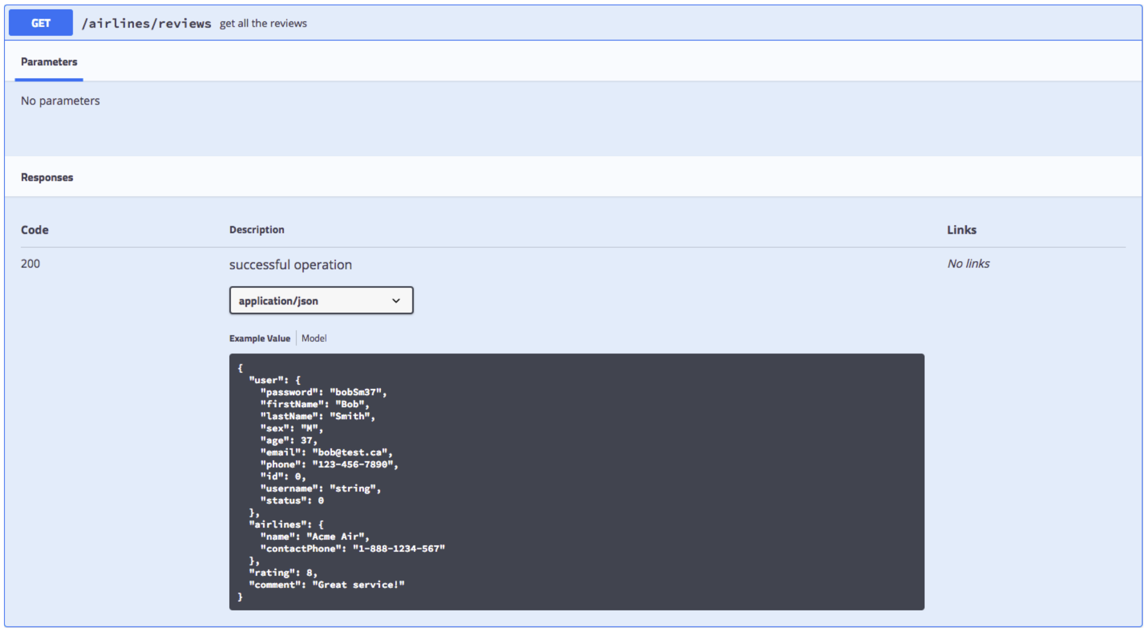
```
@ApplicationPath("/")
@OpenAPIDefinition(
        tags = @Tag(name = "Airlines", description = "airlines app"),
        externalDocs = @ExternalDocumentation(
                description = "instructions for how to deploy this app",
                url = "https://github.com/microservices-api/oas3-airlines/blob/master/README.md"),
        info = @Info(
                title="AirlinesRatingApp API",
                version = "1.0",
                contact = @Contact(
                        name = "AirlinesRatingApp API Support",
                        url = "https://github.com/microservices-api/oas3-airlines")))
```

e. Open the **ReviewResource.java** file



f. This class defines a GET /airlines/reviews API. The getAllReviews method is annotated with an @Operation annotation, which defines the operation's description, what types of responses it will return, along with the object model it will return. In this case, it is an array of Review.class. You can see how the information documented in the annotations are rendered in the UI.

```java
@GET
@Operation(

        operationId = "getAllReviews",
        summary = "get all the reviews",
        method = "GET",
        responses = @ApiResponse(
                responseCode = "200",
                description = "successful operation",
                content = @Content(
                        mediaType = "application/json",
                        schema = @Schema(
                                type = "array",
                                implementation = Review.class
                                )
                        )
                )
        )
@Produces("application/json")
public Response getAllReviews(){
```


GET /airlines/reviews get all the reviews

Parameters

No parameters

Responses

Code	Description	Links
200	successful operation	*No links*

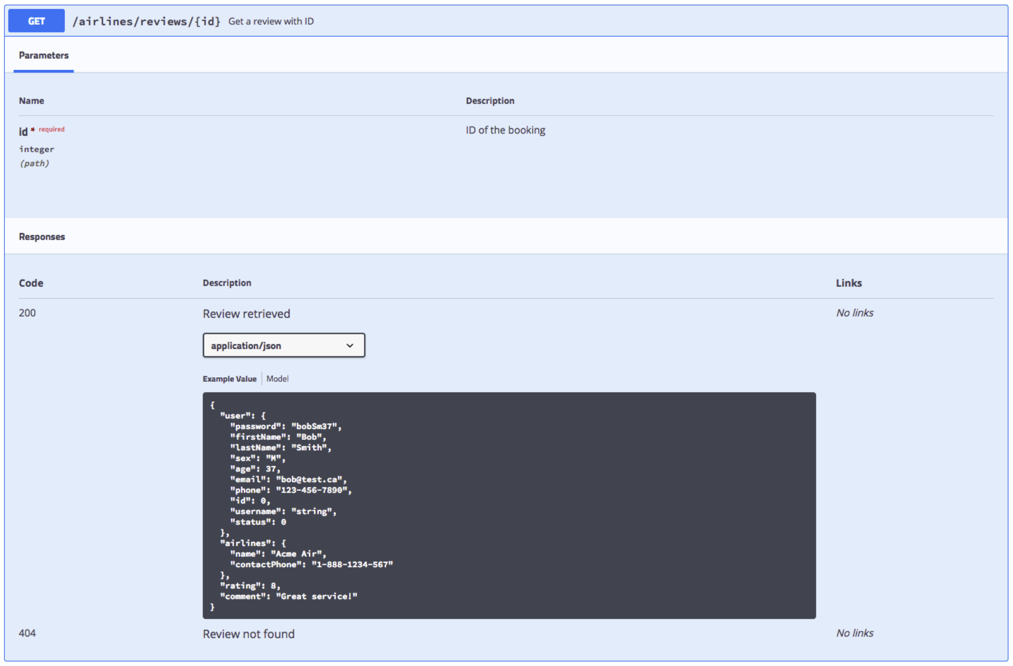
application/json

Example Value | Model

```json
{
  "user": {
    "password": "bobSm37",
    "firstName": "Bob",
    "lastName": "Smith",
    "sex": "M",
    "age": 37,
    "email": "bob@test.ca",
    "phone": "123-456-7890",
    "id": 0,
    "username": "string",
    "status": 0
  },
  "airlines": {
    "name": "Acme Air",
    "contactPhone": "1-888-1234-567"
  },
  "rating": 8,
  "comment": "Great service!"
}
```

g. Let’s look at another example; this method, getReviewsById, returns a single review given an id as a parameter. The @Parameter annotation defines the type of the parameter, description, where it belongs in the request. In this case, the parameter goes in the path.

```java
@GET
@Path("{id}")
@Operation(
        operationId = "getReviewById",

        summary="Get a review with ID",
        responses={
                @ApiResponse(
                        responseCode="200",
                        description="Review retrieved",
                        content=@Content(
                                schema=@Schema(
                                        implementation=Review.class))),
                @ApiResponse(
                        responseCode="404",
                        description="Review not found")
        })
@Produces("application/json")
public Response getReviewById(
        @Parameter(
                name = "id",
                description = "ID of the booking",
                required = true,
                in = "path",
                content = @Content(
                        examples = @ExampleObject(
                                   value = "1")))
        @PathParam("id") int id){
```



GET /airlines/reviews/{id} Get a review with ID

Parameters

Name	Description
id * required integer (path)	ID of the booking

Responses

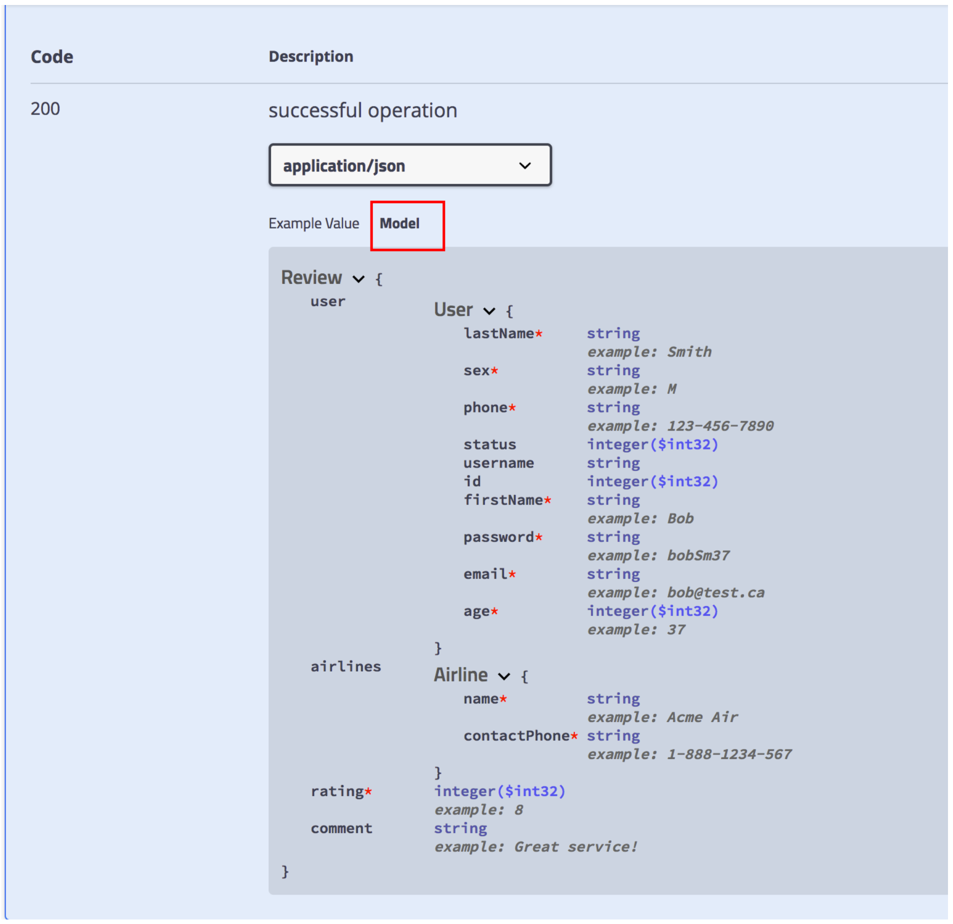
Code	Description	Links
200	Review retrieved	No links
404	Review not found	No links

application/json

Example Value | Model

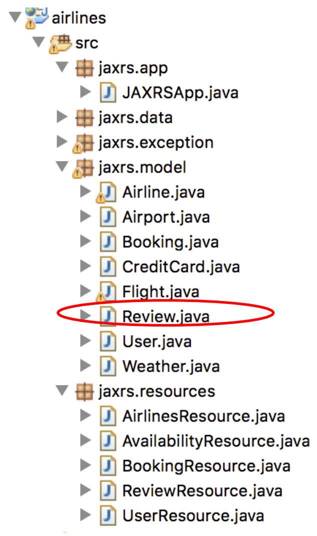
```json
{
  "user": {
    "password": "bobSm37",
    "firstName": "Bob",
    "lastName": "Smith",
    "sex": "M",
    "age": 37,
    "email": "bob@test.ca",
    "phone": "123-456-7890",
    "id": 0,
    "username": "string",
    "status": 0
  },
  "airlines": {
    "name": "Acme Air",
    "contactPhone": "1-888-1234-567"
  },
  "rating": 8,
  "comment": "Great service!"
}
```

h. The **Review.class** is referenced by both example operations. This is also exposed through the API Explorer when the operation's model is viewed.



```
Review {
   user            User {
                      lastName*      string
                                     example: Smith
                      sex*           string
                                     example: M
                      phone*         string
                                     example: 123-456-7890
                      status         integer($int32)
                      username       string
                      id             integer($int32)
                      firstName*     string
                                     example: Bob
                      password*      string
                                     example: bobSm37
                      email*         string
                                     example: bob@test.ca
                      age*           integer($int32)
                                     example: 37
                   }
   airlines        Airline {
                      name*          string
                                     example: Acme Air
                      contactPhone*  string
                                     example: 1-888-1234-567
                   }
   rating*         integer($int32)
                   example: 8
   comment         string
                   example: Great service!
}
```

i. Open the **Review.java** file



```
▼ airlines
  ▼ src
    ▼ jaxrs.app
      ▶ JAXRSApp.java
    ▶ jaxrs.data
    ▶ jaxrs.exception
    ▼ jaxrs.model
      ▶ Airline.java
      ▶ Airport.java
      ▶ Booking.java
      ▶ CreditCard.java
      ▶ Flight.java
      ▶ Review.java
      ▶ User.java
      ▶ Weather.java
    ▼ jaxrs.resources
      ▶ AirlinesResource.java
      ▶ AvailabilityResource.java
      ▶ BookingResource.java
      ▶ ReviewResource.java
      ▶ UserResource.java
```

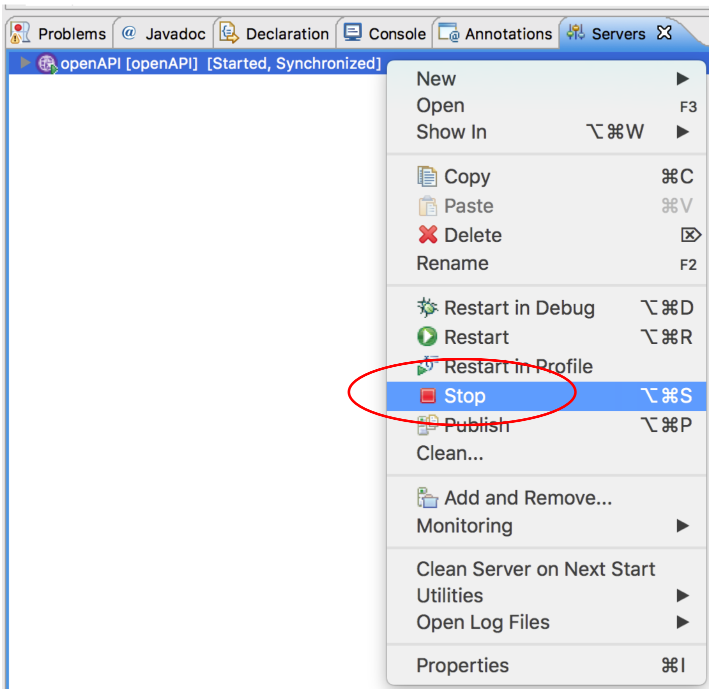
j. This class defines the Review object which used by the /airlines/reviews API. The fields of the class are annotated with the @Schema annotation to define its type, and a possible example for the API.

```
@Schema(required = true)
private User user;

@Schema(required = true)
private Airline airlines;

@Schema(example = "8", required = true)
private int rating;

@Schema(example = "Great service!")
private String comment;
```

k. Play with the other resources and its corresponding models to see how the OpenAPI annotations are used to render the UI you see in the API Explorer.

1.3 Clean up after the lab

1. Stop the `openAPI` server by right-clicking on `openAPI` in the Server view and select **Stop**



Congratulations! You have successfully completed this lab. If you would like to further play around with the Airlines application and experiment with the annotations, the app can be cloned at https://github.com/microservices-api/oas3-airlines

In the next lab, you will learn how to push the Liberty server package you created in this lab to IBM Cloud.