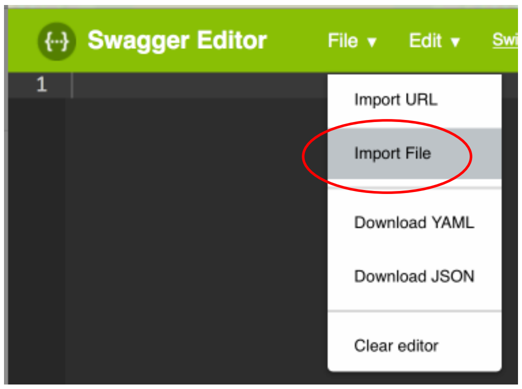
Lab 4: A Design-First Approach to Building APIs

Introduction

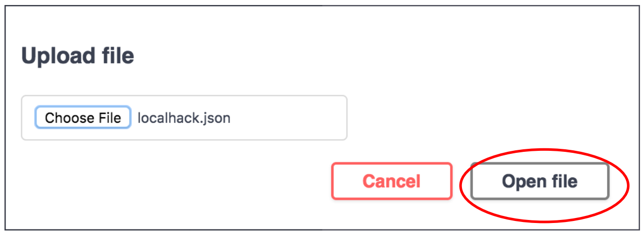
This lab will show you how follow a design-first approach to building APIs. We'll first use the open-sourced Swagger Editor to design APIs. We'll then use WebSphere Developer Tools to create a skeleton application that will contain the Swagger document we created, and finally deploy the application using Liberty server.

4.1 Design APIs using Swagger Editor

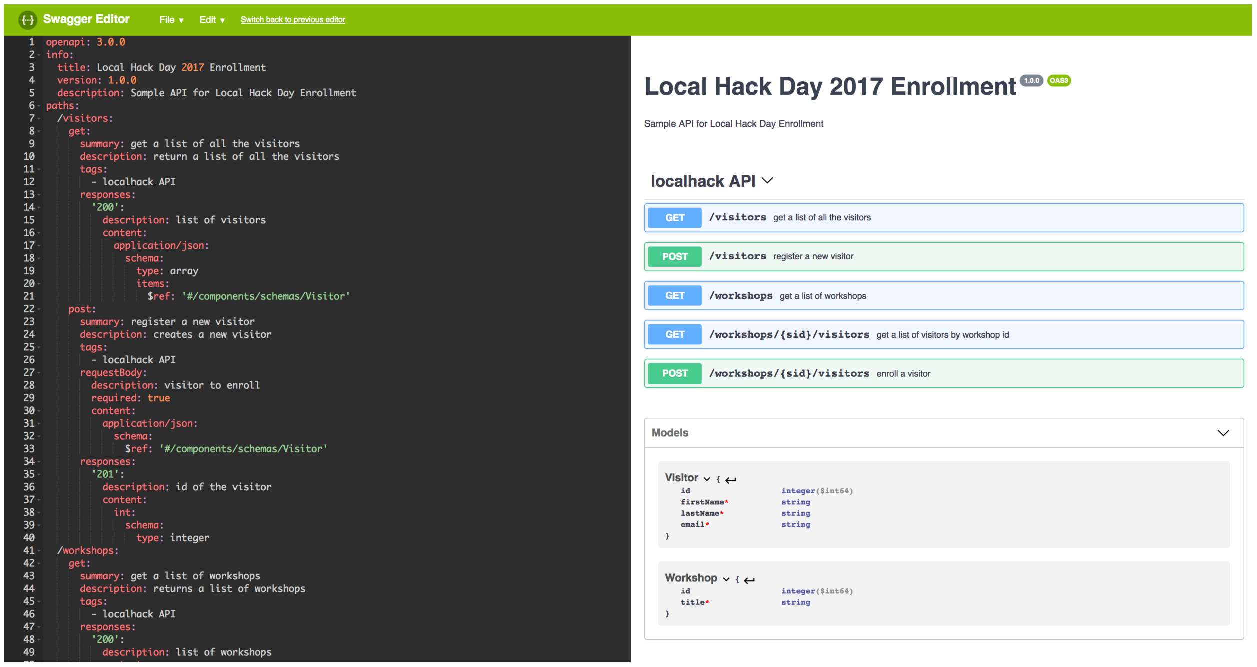
1. We will use Swagger Editor, an open-sourced browser based editor for authoring Swagger and OpenAPI definitions and to preview documentations in real time.
    a. Open a web browser and navigate to https://editor.swagger.io/
2. To get you started, we have designed a sample OpenAPI definition to manage the number of visitors at Local Hack Day 2017. Import it into the editor.
    a. Click on File -> Import File



    b. Select the `localhack.json` file from the lab artifacts and click Open File



3. The Swagger Editor will convert the document to YAML; it is similar to JSON, but instead of quotations and brace brackets, it uses indentation, so it's easier to edit. The Swagger Editor also provides a live rendering of the OpenAPI document on the right-hand side.



```yaml
openapi: 3.0.0
info:
  title: Local Hack Day 2017 Enrollment
  version: 1.0.0
  description: Sample API for Local Hack Day Enrollment
paths:
  /visitors:
    get:
      summary: get a list of all the visitors
      description: return a list of all the visitors
      tags:
        - localhack API
      responses:
        '200':
          description: list of visitors
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Visitor'
    post:
      summary: register a new visitor
      description: creates a new visitor
      tags:
        - localhack API
      requestBody:
        description: visitor to enroll
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Visitor'
      responses:
        '201':
          description: id of the visitor
          content:
            int:
              schema:
                type: integer
  /workshops:
    get:
      summary: get a list of workshops
      description: returns a list of workshops
      tags:
        - localhack API
      responses:
        '200':
          description: list of workshops
```

Local Hack Day 2017 Enrollment 1.0.0 OAS3

Sample API for Local Hack Day Enrollment

localhack API

GET /visitors get a list of all the visitors

POST /visitors register a new visitor

GET /workshops get a list of workshops

GET /workshops/{sid}/visitors get a list of visitors by workshop id

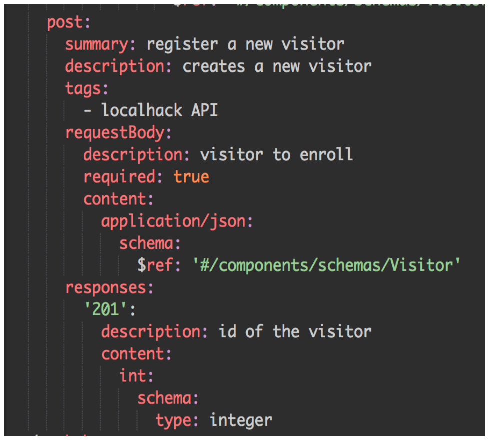
POST /workshops/{sid}/visitors enroll a visitor

Models

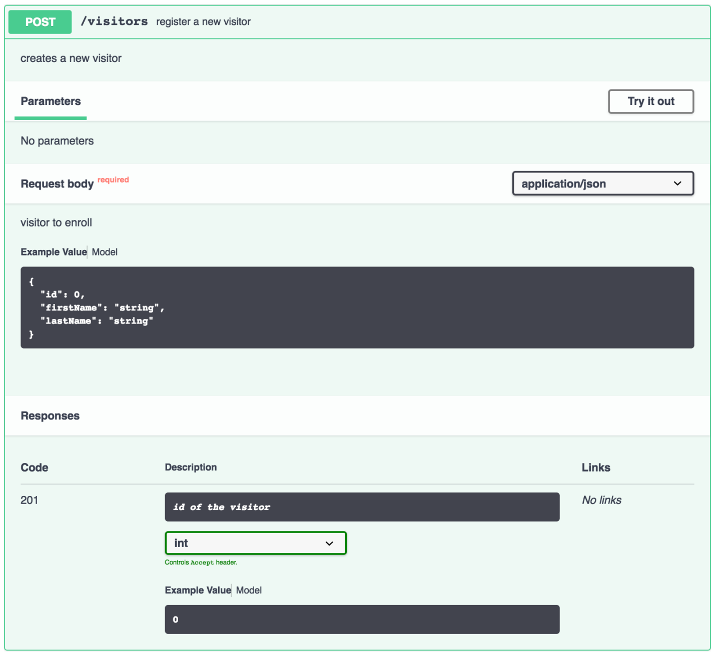
```
Visitor {
    id            integer($int64)
    firstName*    string
    lastName*     string
    email*        string
}

Workshop {
    id            integer($int64)
    title*        string
}
```

4. Spend some time exploring the endpoints and their definitions.
As you expand/collapse different sections of the rendered OpenAPI document on the right-hand side, the corresponding YAML definitions can be viewed on the left-hand side. This functionality helps to navigate the document, especially when editing large documents.



```yaml
    post:
      summary: register a new visitor
      description: creates a new visitor
      tags:
        - localhack API
      requestBody:
        description: visitor to enroll
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Visitor'
      responses:
        '201':
          description: id of the visitor
          content:
            int:
              schema:
                type: integer
```



POST /visitors register a new visitor

creates a new visitor

Parameters

No parameters

Request body required

visitor to enroll

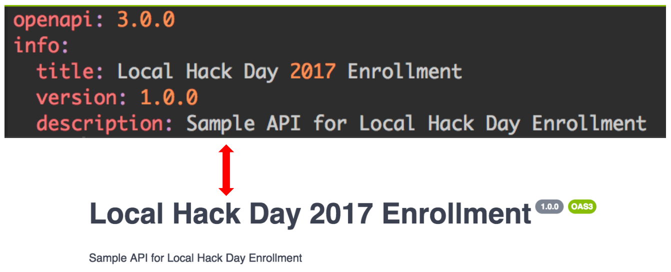
Example Value Model

```
{
  "id": 0,
  "firstName": "string",
  "lastName": "string"
}
```

Responses

Code	Description	Links
201	id of the visitor Example Value Model 0	No links

5. Let's pretend you designed this document and request feedback from your team. Using the Swagger Editor, address the review comments to improve the API.
   a. A teammate mentions that the title should also specify the year of the Local Hack Day. So let's make the change using the Swagger Editor.
      Modify the title (specified near the top of the document) to include the year. Notice that as you make changes to the document, the Swagger Editor renders it in real time.

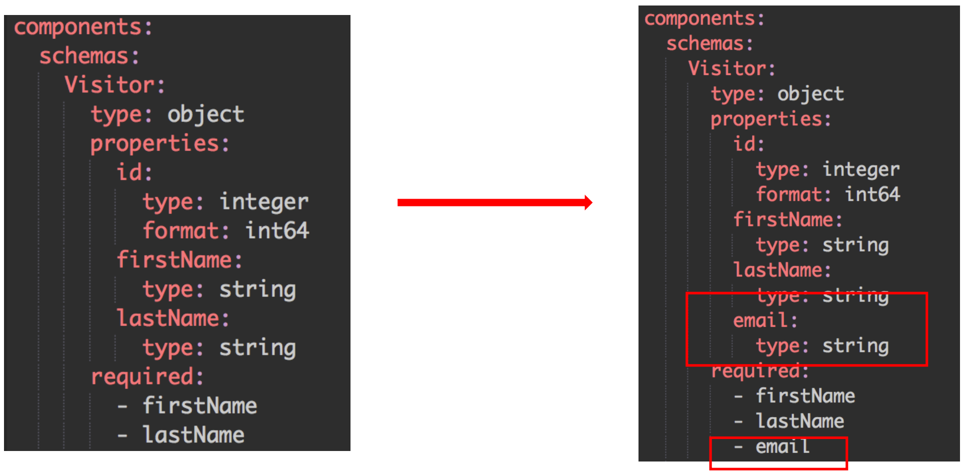


```
openapi: 3.0.0
info:
  title: Local Hack Day 2017 Enrollment
  version: 1.0.0
  description: Sample API for Local Hack Day Enrollment
```

Local Hack Day 2017 Enrollment 1.0.0 OAS3

Sample API for Local Hack Day Enrollment

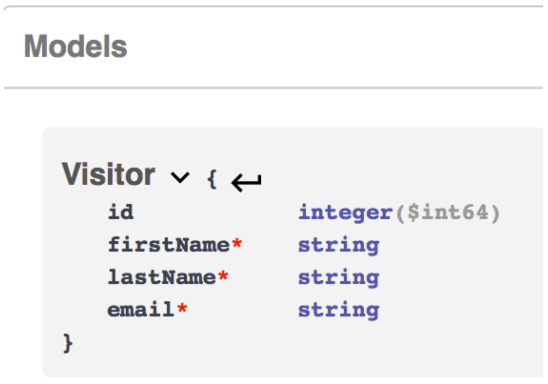
   b. Another teammate comments that the **Visitor** model should have an email address in addition to first and last names. You also agree since having an email address would be essential to communicating with them, if necessary. Let's make the changes using the Swagger Editor.
      i. Within the YAML document, the **Visitor** model is specified under the '**components'** section. It has three properties: **id**, **firstName**, and **lastName.** Add another property called **email** of type **string**. Since this is an essential property for communication, add it to the **required** section, as well.



```
components:
  schemas:
    Visitor:
      type: object
      properties:
        id:
          type: integer
          format: int64
        firstName:
          type: string
        lastName:
          type: string
      required:
        - firstName
        - lastName
```

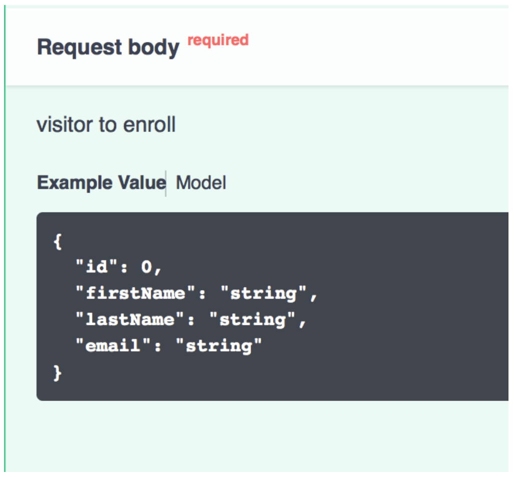
```
components:
  schemas:
    Visitor:
      type: object
      properties:
        id:
          type: integer
          format: int64
        firstName:
          type: string
        lastName:
          type: string
        email:
          type: string
      required:
        - firstName
        - lastName
        - email
```

ii. Notice that wherever the Visitor model is referenced, the changes you made in the YAML document is reflected on the Swagger UI.



Models

```
Visitor ˅ {
    id            integer($int64)
    firstName*    string
    lastName*     string
    email*        string
}
```

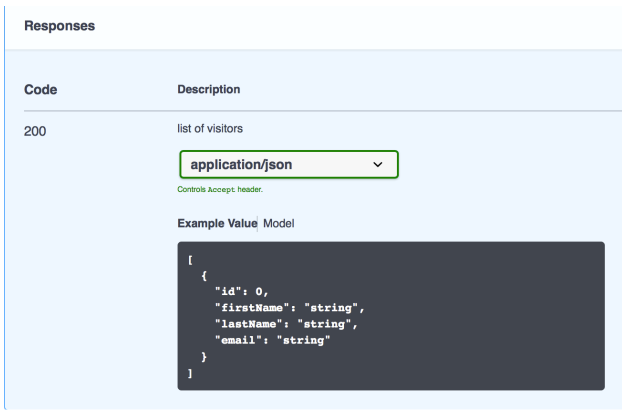


Request body required

visitor to enroll

Example Value | Model

```
{
  "id": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string"
}
```



Responses

Code	Description
200	list of visitors

application/json

Controls Accept header.

Example Value | Model

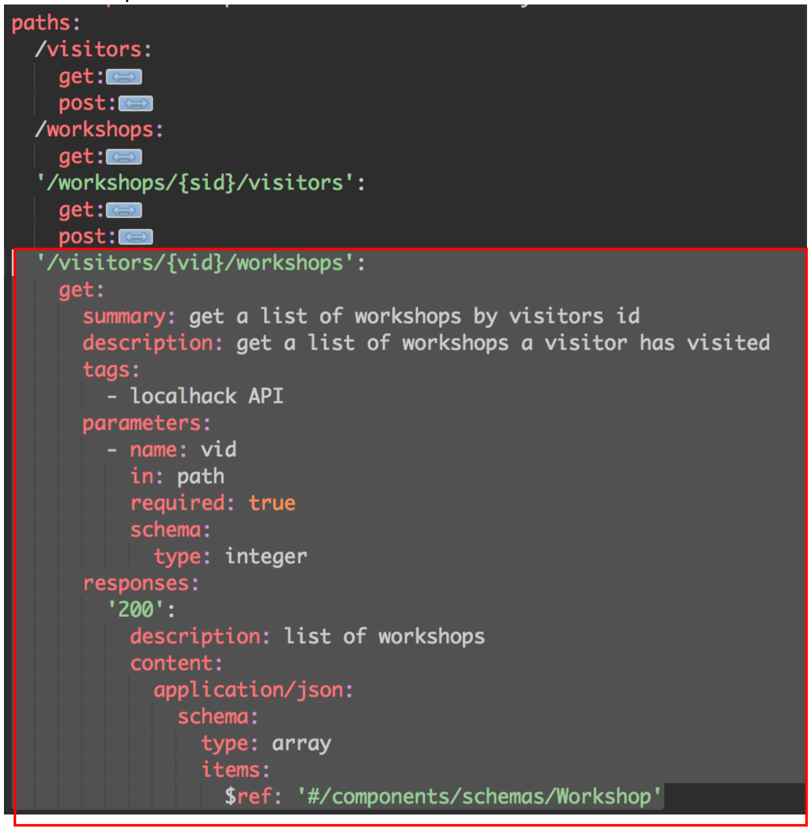
```
[
  {
    "id": 0,
    "firstName": "string",
    "lastName": "string",
    "email": "string"
  }
]
```

c. **This step is optional. If time is a constraint, then skip this and go to step 6. You would still be able to complete rest of the lab even if you skip this step.**

i. One other teammate says that there should be a way to see all the **Workshops**(s) a **Visitor** has visited.

Let's add a new path: **/visitors/{vid}/sessions.** It contains a GET operation, which has one path parameter called "**vid**" representing the visitor ID and it returns a list of sessions that the visitor has enrolled in.

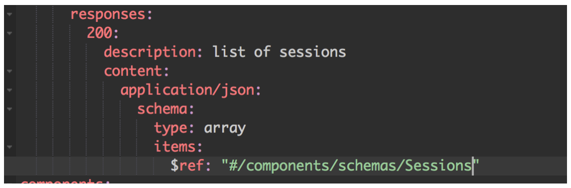
Add the new path as shown below.



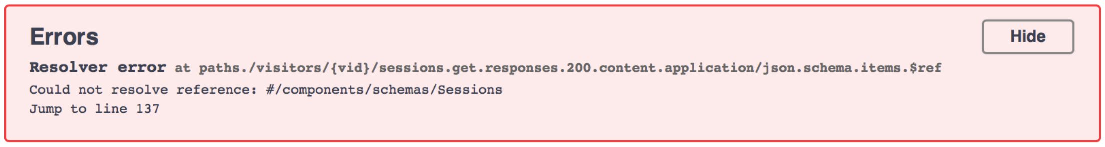
```
paths:
  /visitors:
    get:
    post:
  /workshops:
    get:
  '/workshops/{sid}/visitors':
    get:
    post:
  '/visitors/{vid}/workshops':
    get:
      summary: get a list of workshops by visitors id
      description: get a list of workshops a visitor has visited
      tags:
        - localhack API
      parameters:
        - name: vid
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: list of workshops
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Workshop'
```

ii. Swagger Editor also validates the document in real-time and flag any errors.

In above example, if you were to incorrectly type the reference to the Workshop model by typing '**#/components/schemas/Workshops**' (notice the extra 's' at the end), then the Swagger Editor will flag an error and provide some information to resolve the issue.



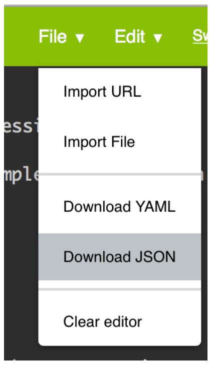
```
      responses:
        200:
          description: list of sessions
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/Sessions"
```



Errors

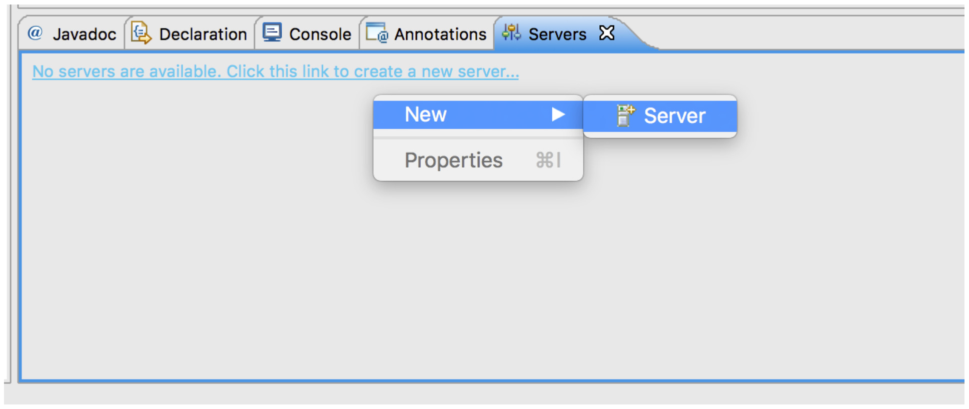
```
Resolver error at paths./visitors/{vid}/sessions.get.responses.200.content.application/json.schema.items.$ref
Could not resolve reference: #/components/schemas/Sessions
Jump to line 137
```

6. At this point in the lab, we are done designing our APIs. Let's export the Swagger document in JSON Format
    a. Click on File -> Download JSON and save the file.

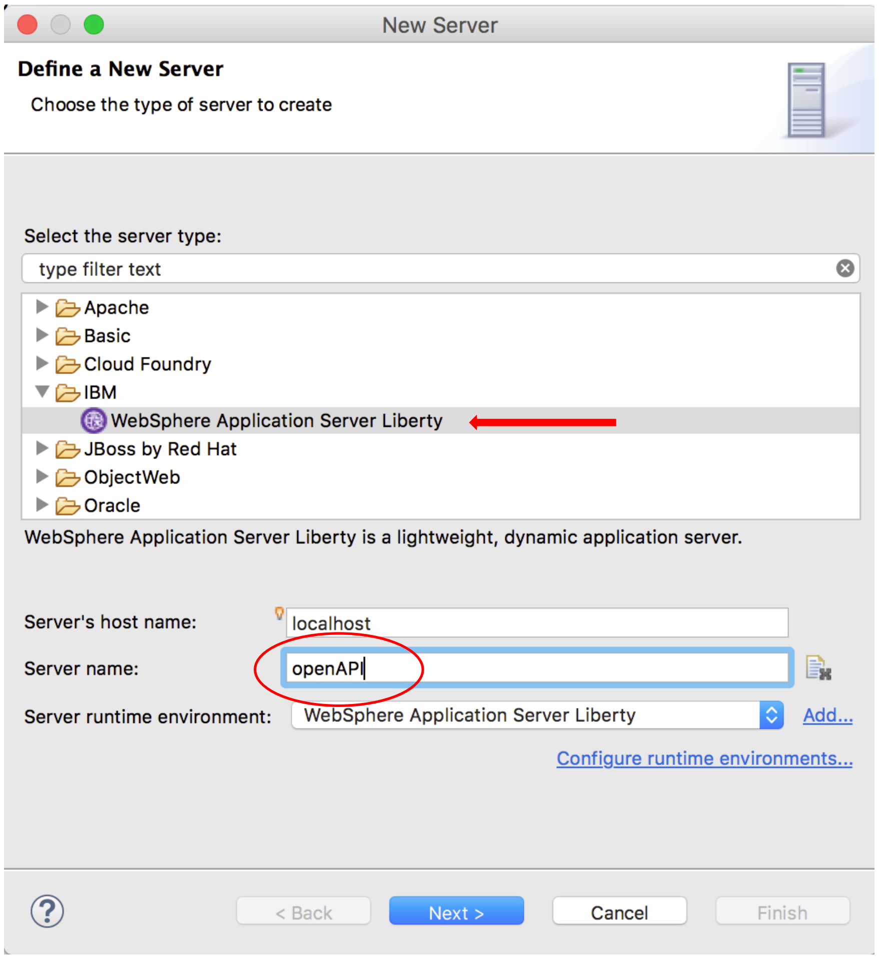


4.2 Generate JAX-RS application for the Swagger definition using Eclipse

1. If you have successfully completed Lab 1 or Lab 3, skip ahead to step **b.**
    a. In the **OpenAPI** server, delete any **.war** files and **server.xml** files from the previous labs. You can use the same server from the previous labs for this one, as well. Skip ahead to step **2.**
    b. Create a new Liberty server called `openAPI` to host your Swagger document
    c. 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
Properties ⌘I
Server

d. 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:

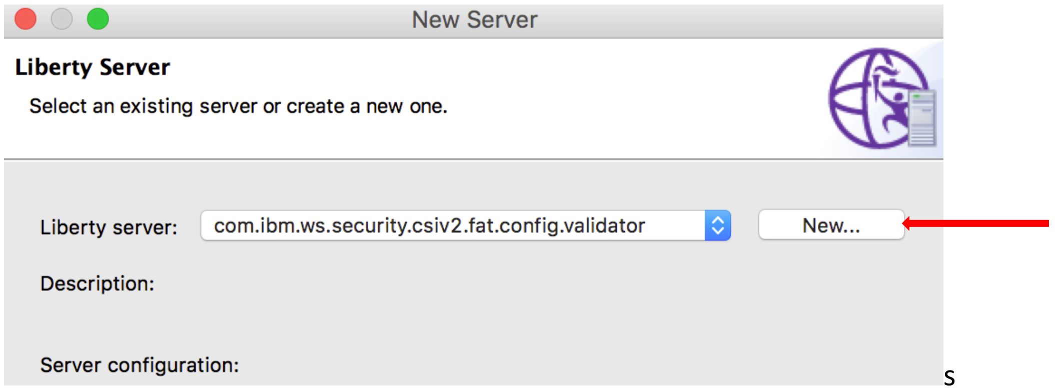
type filter text

- Apache
- Basic
- Cloud Foundry
- IBM
  - WebSphere Application Server Liberty
- JBoss by Red Hat
- ObjectWeb
- Oracle

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

Server's host name:	localhost
Server name:	openAPI
Server runtime environment:	WebSphere Application Server Liberty

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



Liberty Server

Select an existing server or create a new one.

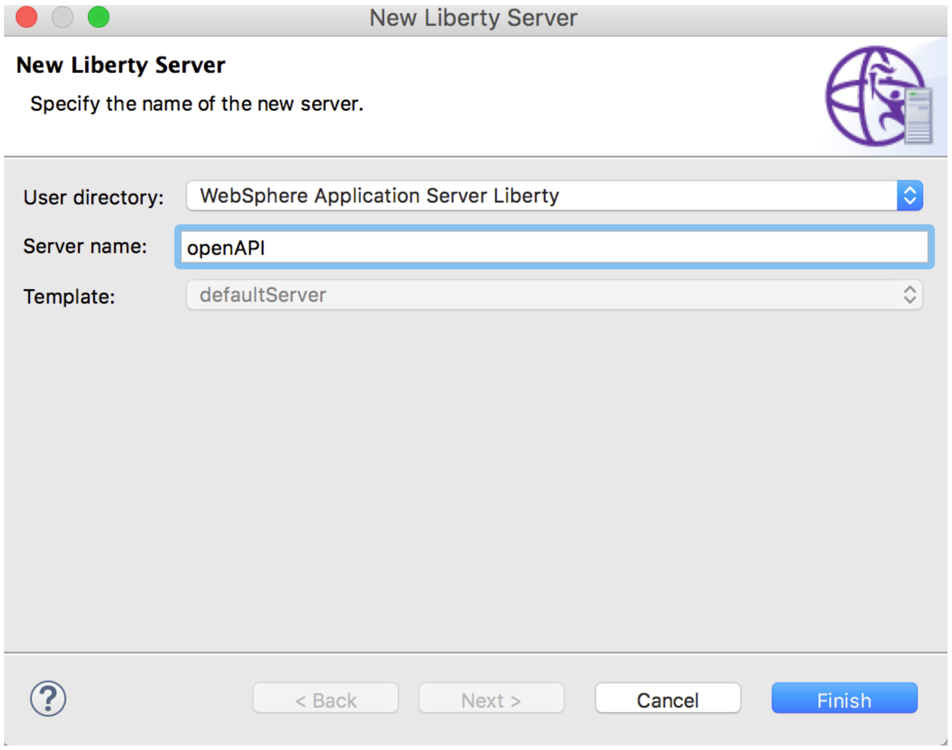
Liberty server: com.ibm.ws.security.csiv2.fat.config.validator

Description:

Server configuration:

s

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



New Liberty Server

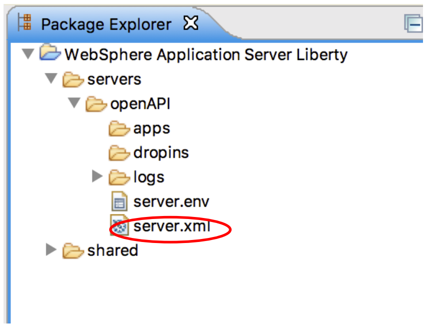
Specify the name of the new server.

User directory: WebSphere Application Server Liberty

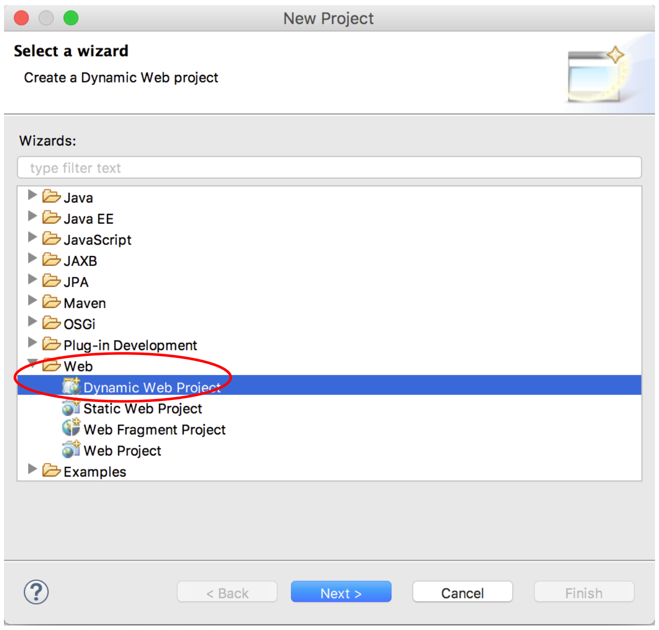
Server name: openAPI

Template: defaultServer

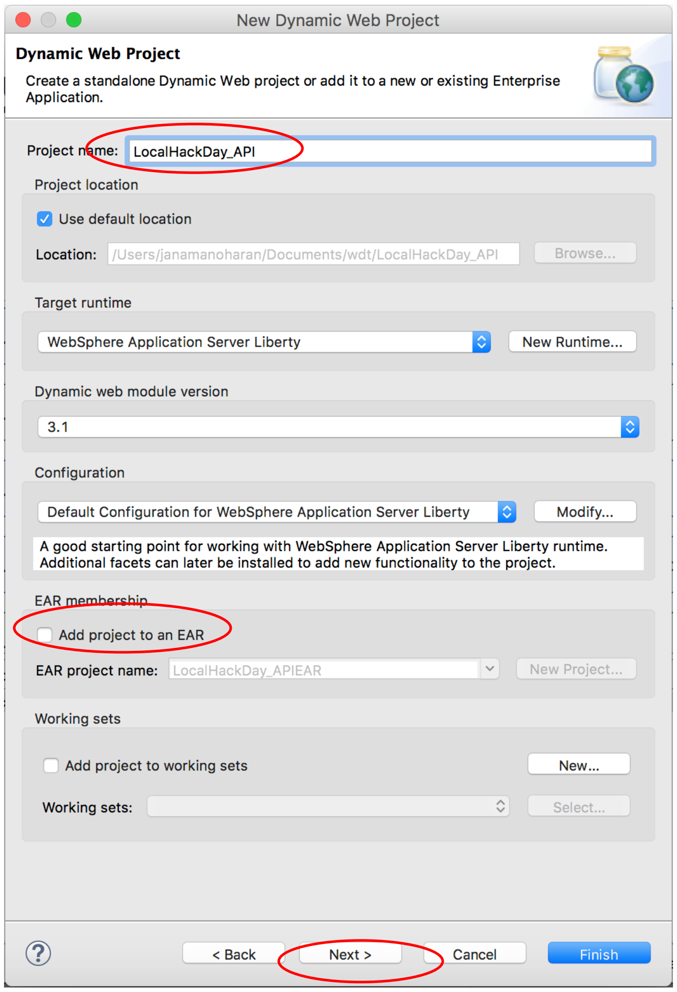
2. Copy lab artifacts to the openAPI server configuration. This lab relies on a pre-existing configuration
   a. Copy the **server.xml** from location directory to **…/wlp/usr/servers/openAPI** directory and ***replace*** the existing **server.xml**



3. Create a new web project to contain the Swagger document with the Local Hack APIs.
   a. In Eclipse, click File -> New -> Dynamic Web Project



b. Enter name of the project, **LocalHack_API**. And make sure to uncheck “Add project to EAR”. Click Next -> Next.


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: LocalHackDay_API

Project location

☑ Use default location

Location: /Users/janamanoharan/Documents/wdt/LocalHackDay_API

Target runtime: WebSphere Application Server Liberty

Dynamic web module version: 3.1

Configuration: Default Configuration for WebSphere Application Server Liberty

A good starting point for working with WebSphere Application Server Liberty runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

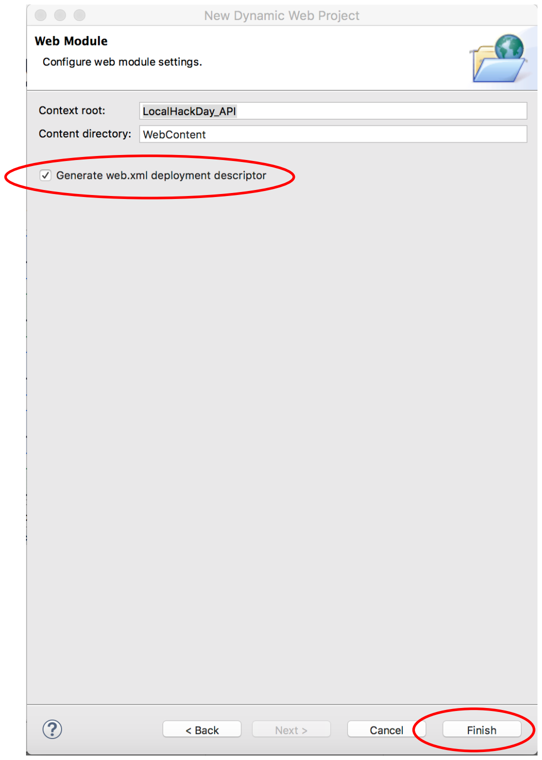
EAR project name: LocalHackDay_APIEAR

Working sets

☐ Add project to working sets

Working sets:

c. Make sure to check the Generate web.xml deployment descriptor and click Finish



Web Module

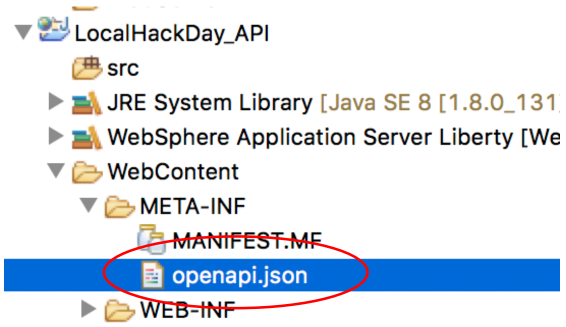
Configure web module settings.

Context root: LocalHackDay_API

Content directory: WebContent

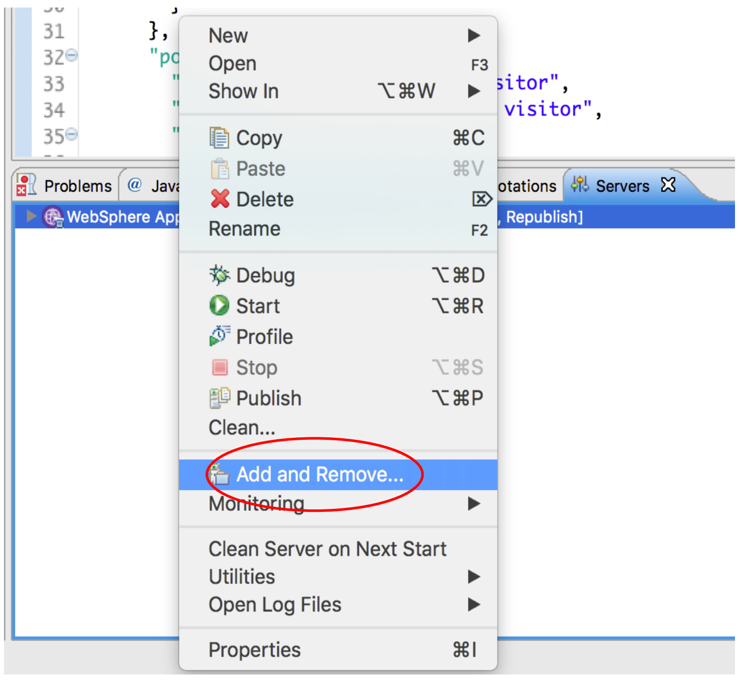
[✓] Generate web.xml deployment descriptor

d. Copy the Swagger document that you edited in the Swagger Editor, and paste it into the **META-INF** folder of the project. The file must be renamed to `openapi.json`

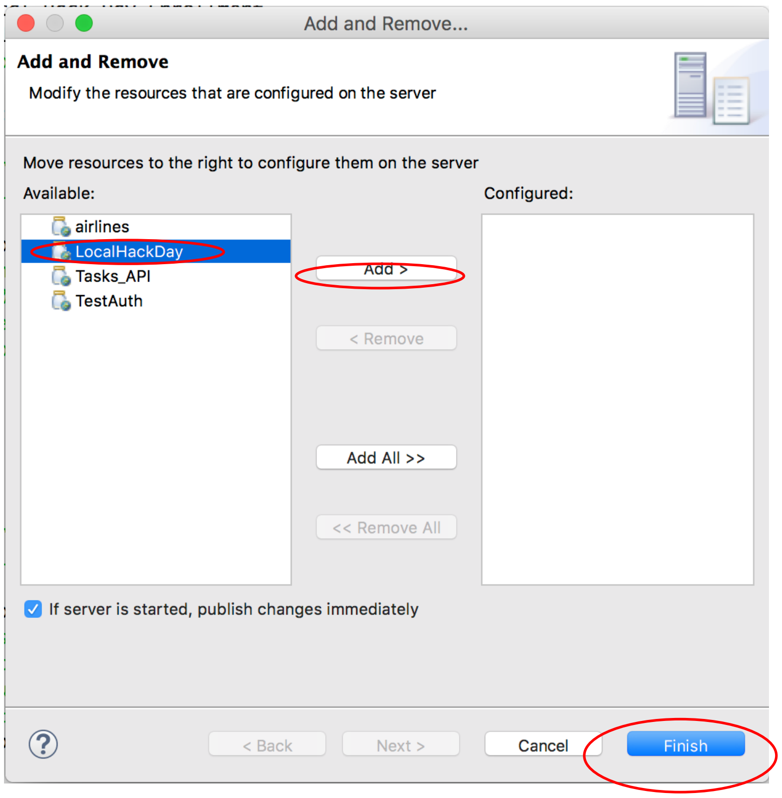


```
LocalHackDay_API
    src
  JRE System Library [Java SE 8 [1.8.0_131
  WebSphere Application Server Liberty [We
  WebContent
    META-INF
      MANIFEST.MF
      openapi.json
    WEB-INF
```

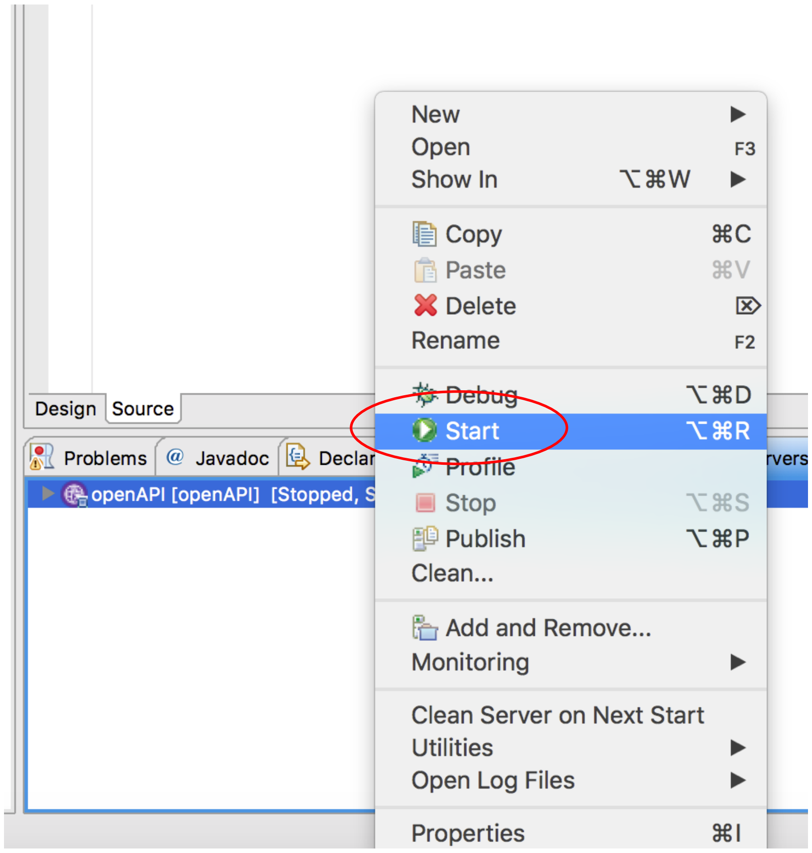
4. Deploy the **LocalHack_API** application in the openAPI server
   a. In the Server view, right-click on the openAPI server and click **Add and Remove…**



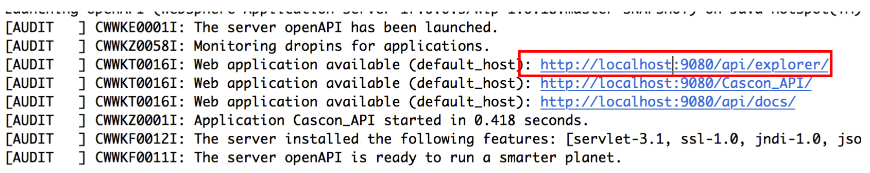
   b. Select the **LocalHack_API** project from the Available section and click **Add >**, then click Finish


Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
airlines
LocalHackDay
Tasks_API
TestAuth
Configured:
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Cancel
Finish

5. Start the openAPI server and use the API Explorer to access the REST API for the **LocalHack_API** app
    a. In the Servers view, right-click the server in the openAPI server, and click **Start**

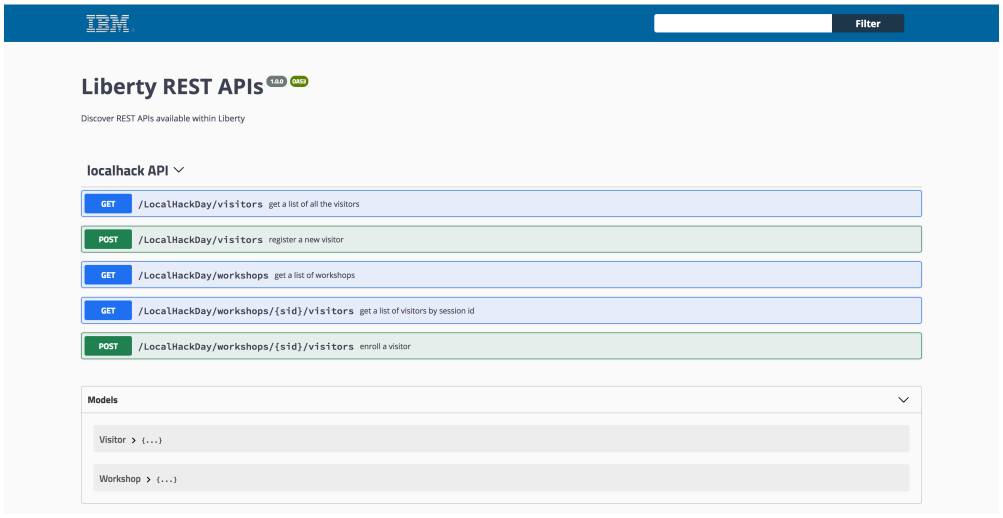


    b. To access the API Explorer, click the link in the Console view.



```
[AUDIT   ] CWWKE0001I: The server openAPI has been launched.
[AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/api/explorer/
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/Cascon_API/
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/api/docs/
[AUDIT   ] CWWKZ0001I: Application Cascon_API started in 0.418 seconds.
[AUDIT   ] CWWKF0012I: The server installed the following features: [servlet-3.1, ssl-1.0, jndi-1.0, jso
[AUDIT   ] CWWKF0011I: The server openAPI is ready to run a smarter planet.
```

6. You will see that the Swagger API you edited using the Swagger editor is now being hosted by the openAPI Liberty Server.
    a. Take some to play with the available REST APIs for the Local Hack Day Workshop Enrollment App.



Liberty REST APIs

Discover REST APIs available within Liberty

localhack API

GET /LocalHackDay/visitors get a list of all the visitors

POST /LocalHackDay/visitors register a new visitor

GET /LocalHackDay/workshops get a list of workshops

GET /LocalHackDay/workshops/{sid}/visitors get a list of visitors by session id

POST /LocalHackDay/workshops/{sid}/visitors enroll a visitor

You have just experienced how simple it is to design and deploy your APIs with a simple JSON file in Liberty using the OpenAPI feature.

Congratulations! You have successfully completed this lab.