**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 OpenAPI 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 sessions and visitors at CASCON 2017. Import it into the editor.
   a. Click on File -> Import File
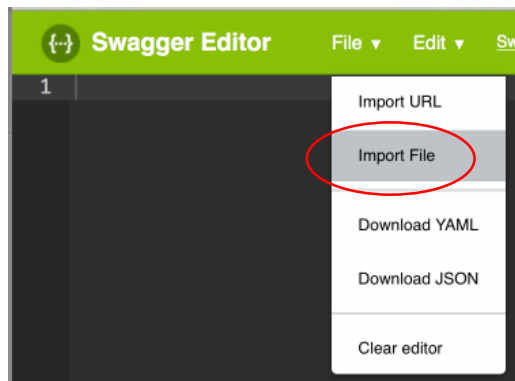
   b. Select the `cascon.json` file from the lab artifacts and click Open File
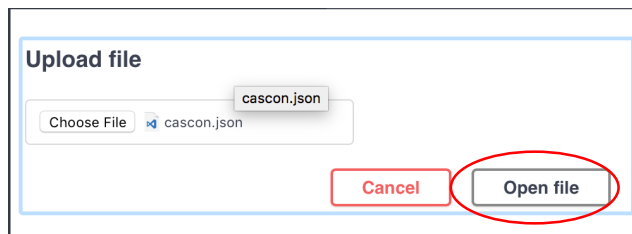
3. The Swagger Editor will convert the document to YAML; it is similar to JSON, but instead of quotations and brace brackets, it uses indentation, so it's easier to edit. The Swagger Editor also provides a live rendering of the OpenAPI document on the right-hand side.

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.

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 CASCON. 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.

```yaml
openapi: 3.0.0
info:
  title: Cascon 2017 Session Enrollment
  version: 1.0.0
  description: Sample API for Cascon session Enrollment
```

Cascon 2017 Session Enrollment `1.0.0` `OAS3`

Sample API for Cascon session 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.

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

```yaml
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 **Session**(s) a **Visitor** enrolled in.

   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: [⟳]
  /sessions:
    get: [⟳]
    post: [⟳]
  '/sessions/{sid}/visitors':
    get: [⟳]
    post: [⟳]
  '/visitors/{vid}/sessions':
    get:
      summary: get a list of sessions by visitor id
      description: returns a list of sessions a visitor is enrolled in
      tags:
      - Cascon API
      parameters:
      - name: vid
        in: path
        required: true
        schema:
          type: integer
      responses:
        200:
          description: list of sessions
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Session'
```

   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 Session model by typing '**#/components/schemas/Sessions**' (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   [Hide]

   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 OpenAPI document in JSON Format
   a. Click on File -> Download JSON and save the file.
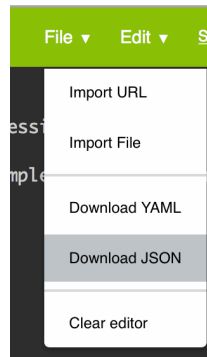
4.2 Generate JAX-RS application for the OpenAPI definition using Eclipse

1. If you have not 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 OpenAPI document
   c. Open the Servers view, and create a new server
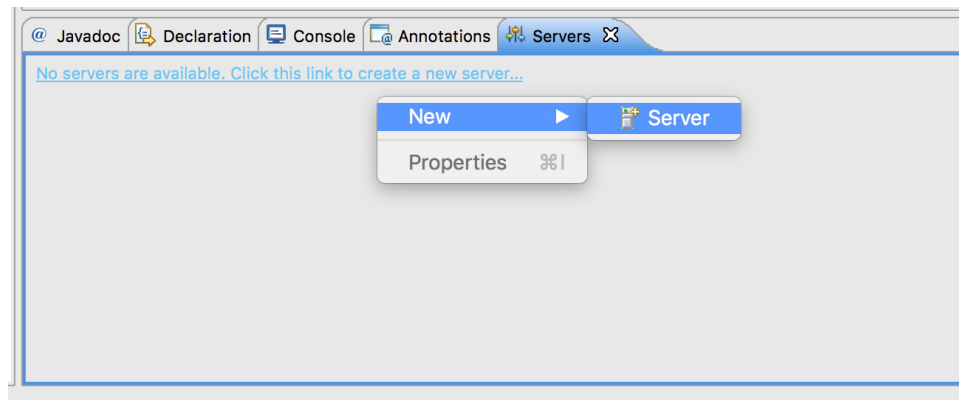
d. Create a new "**WebSphere Application Server Liberty**" server representation with the name openAPI. Hit **Next**.

e. Click **New...** to create a new Liberty server instance.
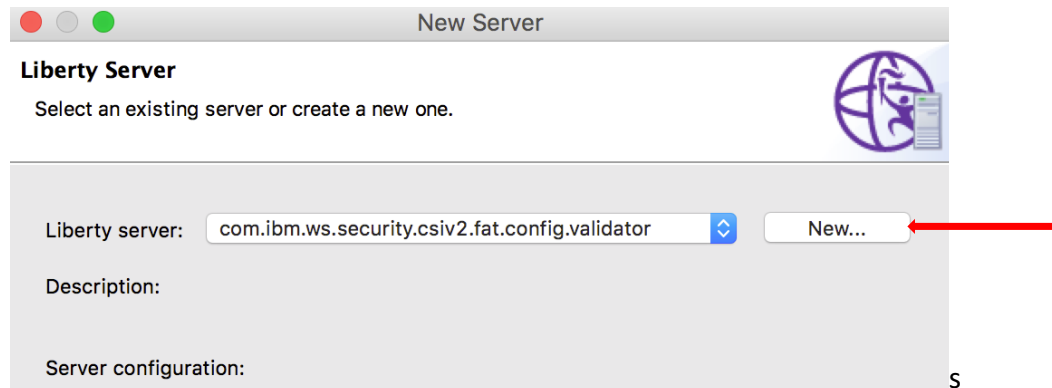
f. Specify the server name openAPI and click **Finish**. And click **Finish** again.
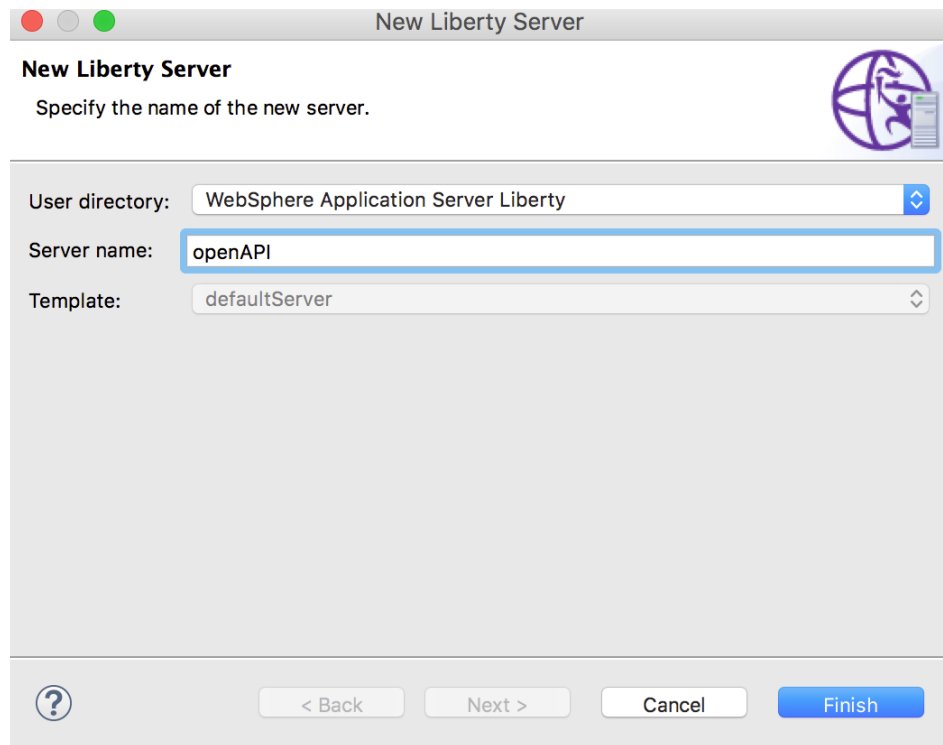
2. Copy lab artifacts to the openAPI server configuration. This lab relies on a pre-existing configuration
   a. Copy the **server.xml** from location directory to **home/student/cascon/wlp/usr/servers/openAPI** directory and *replace* the existing **server.xml**
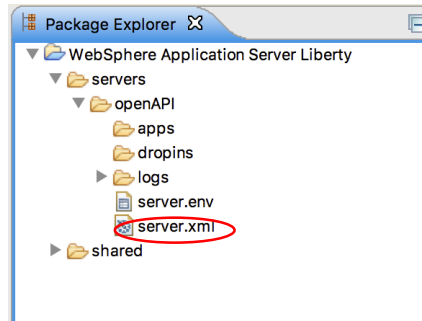
3. Create a new web project to contain the Swagger document with the Cascon APIs.
   a. In Eclipse, click File -> New -> Dynamic Web Project
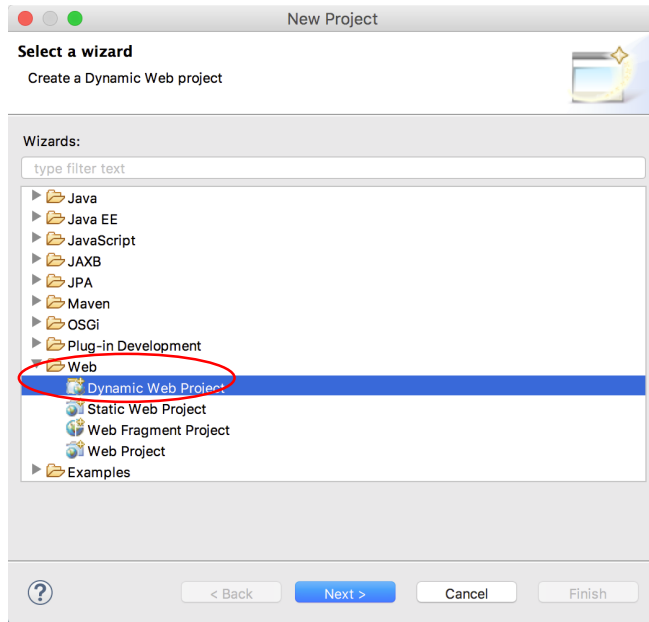
b. Enter name of the project, **Cascon_API**. And make sure to uncheck "Add project to EAR". Click Next -> Next.

c. Make sure to check the Generate web.xml deployment descriptor and click Finish
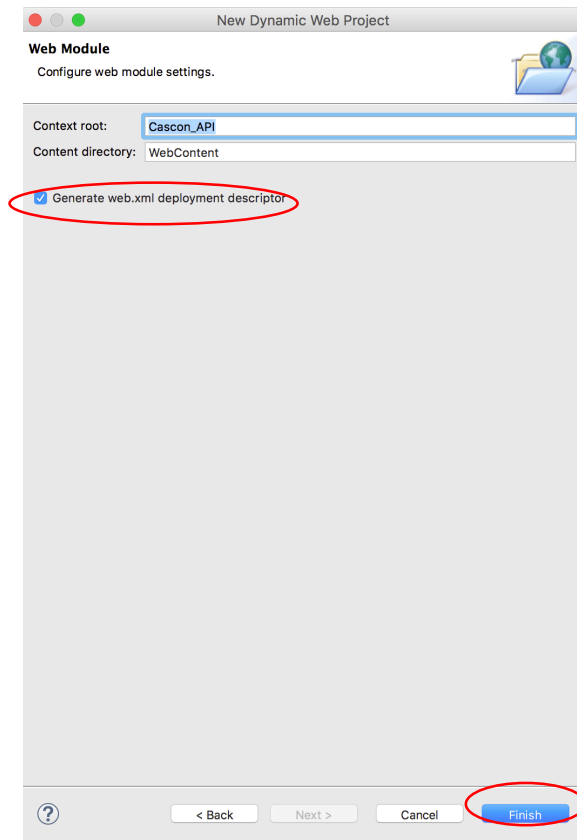
d. Copy the Swagger document that you edited in the Swagger Editor, and paste it into the **META-INF** folder of the project. The file must be renamed to `openapi.json`
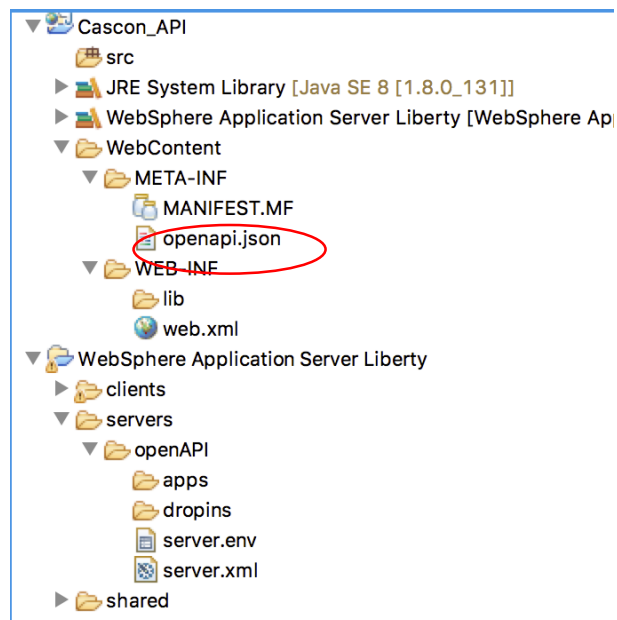
4. Deploy the **Cascon_API** application in the openAPI server
   a. In the Server view, right-click on the openAPI server and click **Add and Remove…**
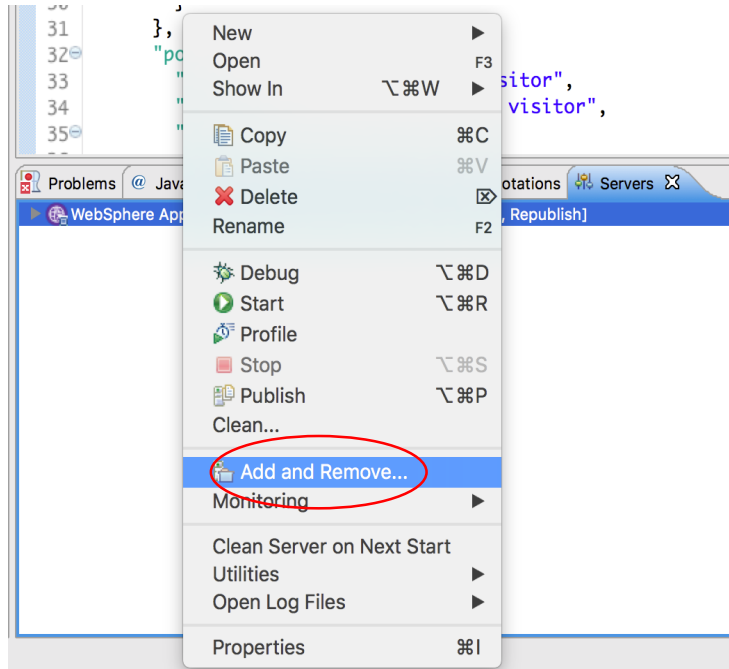
   b. Select the **Cascon_API** project from the Available section and click **Add >**, then click Finish
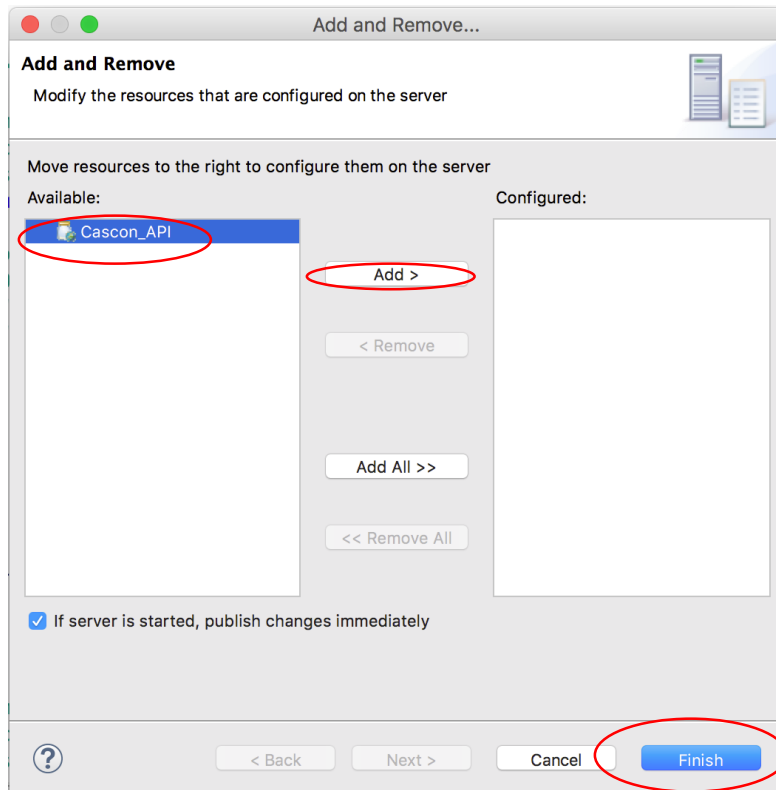
5. Start the openAPI server and use the API Explorer to access the REST API for the **Cascon_API** app
   a. In the Servers view, right-click the server in the `openAPI` server, and click **Start**
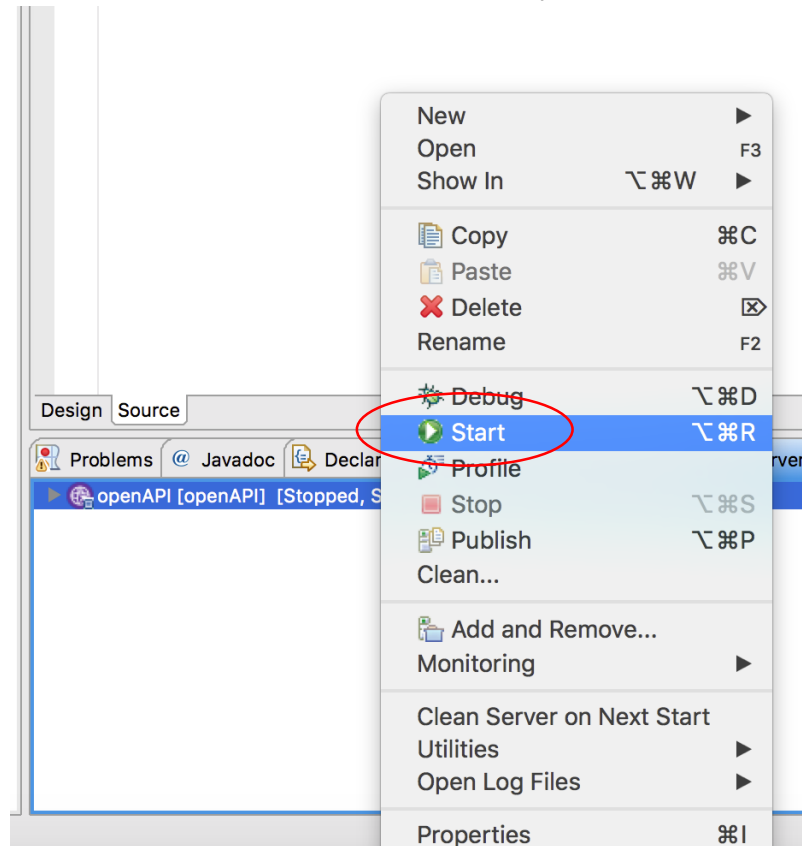
   b. To access the API Explorer, click the link in the Console view.

```
Launching openAPI (WebSphere Application Server 17.0.0.0/wlp-1.0.18.cl180 20171215-1940) on Java HotSpot(TM)...
[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 Cascon Session Enrollment App.

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.