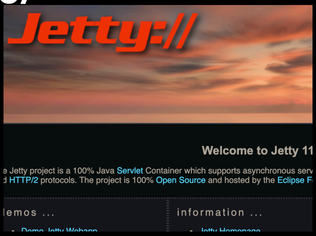
Tutorials

Jetty Web API Servlet

jetty://

API Design: Java Server and Client

- Lab: Java Server
- Note: This is not meant to be an exhaustive st by-step for configuring Jetty but may be helpful:
  - Jetty: https://www.eclipse.org/jetty/
  - Unzip
  - Set env var:
    - JETTY_HOME=/Users/bearc2020/Downloads/jetty-home-11.0.5
  - Start server:
    - java -jar start.jar
  - Load http://localhost:8080/



Welcome to Jetty 11

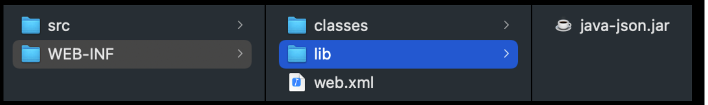
e Jetty project is a 100% Java Servlet Container which supports asynchronous serv
d HTTP/2 protocols. The project is 100% Open Source and hosted by the Eclipse F

API Design: Java Server and Client

- On same level as jetty-home-<ver>
  - Create jetty-base dir on same level as jetty-home…
  - Run:
  - java -jar start.jar jetty.base=/path/to/your/jetty-base --add-module=server,http,deploy,annotations
  - Change dir to your jetty-base and run:
  - java -jar /path/to/your/jetty-home/start.jar --add-module=server,http,deploy,annotations

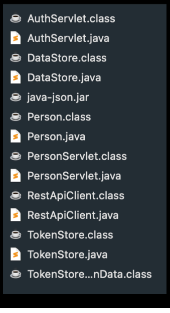
API Design: Java Server and Client

- Under the jetty-base/webapps dir
- Create a PersonServlet dir
- Inside PersonServlet create src and WEB-INF dirs
  - src is for source code but can be located anywhere
  - WEB-INF is for configuration, classes and libraries



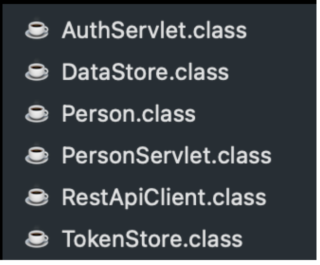
API Design: Java Server and Client

- Download java-json.jar from here:
- [java-json/java-json.jar.zip]
- Copy that to the src and WEB-INF/lib directories
- Optional: Copy various source files into the src directory


```
AuthServlet.class
AuthServlet.java
DataStore.class
DataStore.java
java-json.jar
Person.class
Person.java
PersonServlet.class
PersonServlet.java
RestApiClient.class
RestApiClient.java
TokenStore.class
TokenStore.java
TokenStore...nData.class
```

API Design: Java Server and Client

- Compile the various .java files e.g.,
- javac -cp <path to jetty home>/lib/jetty-jakarta-servlet-api-5.0.2.jar:./java-json.jar:. <filename>.java
  - NOTE: Not all java files require the serlet api and json jar files.
- Copy the class files into
  - WEB-INF/classes



```
AuthServlet.class
DataStore.class
Person.class
PersonServlet.class
RestApiClient.class
TokenStore.class
```

API Design: Java Server and Client

- Create the web.xml file under the WEB-INF dir
  - Boilerplate…

```
<web-app
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
```

API Design: Java Server and Client

• Servlet(s)…

```
<servlet>
    <servlet-name>PersonServlet</servlet-name>
    <servlet-class>PersonServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>PersonServlet</servlet-name>
    <url-pattern>/people/*</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>AuthServlet</servlet-name>
    <servlet-class>AuthServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>AuthServlet</servlet-name>
    <url-pattern>/auth</url-pattern>
</servlet-mapping>
```

API Design: Java Server and Client

• CORS (Cross-Origin Resource Sharing) and end…

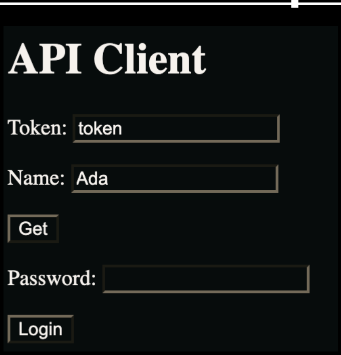
```
    <filter>
        <filter-name>cross-origin</filter-name>
        <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
        <init-param>
            <param-name>allowedOrigins</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>allowedMethods</param-name>
            <param-value>GET,POST</param-value>
        </init-param>
        <init-param>
            <param-name>allowedHeaders</param-name>
            <param-value>X-Requested-With,Content-Type,Accept,Origin,Authorization</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>cross-origin</filter-name>
        <filter-pattern>/*</filter-pattern>
    </filter-mapping>

</web-app>
```

API Design: Java Server and Client

- With the env var, you can start jetty from other dirs.
- Restart jetty from the jetty-base dir:
  - java -jar $JETTY_HOME/start.jar
  - Load: http://localhost:8080/
- Load the web app at:
  - http://localhost:8080/PersonServlet/people/Kevin

API Design: Java Server and Client

- Create HTML client and place in webapps/root directory
- Load with:
  - http://localhost:8080/ApiClient.html



API Client

Token: token

Name: Ada

Get

Password:

Login

API Design: Java Server and Client

- **DataStore is initialized with each start with preconfigured data:**

```
personMap.put("Ada", new Person("Ada", "Ada Lovelace was the first programmer.", 1815, "pwa"));
personMap.put("Kevin", new Person("Kevin", "Kevin is the author of HappyCoding.io.", 1986, "pwk"));
personMap.put("Stanley", new Person("Stanley", "Stanley is Kevin's cat.", 2007, "pws"));    }
```

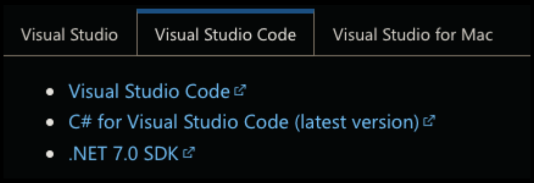
- **Users can be fetched with GET without login**
- **Logged in users can Save changes to own account**
- **Logged in users can create new users**

C# .NET Server

Microsoft .NET

API Design: C#.NET Server and Client

- **C# .NET tutorial**
- **https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio-code**


Visual Studio | Visual Studio Code | Visual Studio for Mac

- Visual Studio Code
- C# for Visual Studio Code (latest version)
- .NET 7.0 SDK

Copyright Brainwash Inc. 2022-2023

Java Standalone Server Using Spark

Java Standalone Server

- Java JDK
- https://docs.oracle.com/en/java/javase/
  - Verify JAVA_HOME is set in env

- Maven
- https://maven.apache.org/download.cgi
- https://maven.apache.org/install.html
  - Verify maven bin dir is in PATH in env

Java Standalone Server

- Create Java Maven Project
  - Create pom.xml
    - mvn archetype:generate -DgroupId=com.brainwashinc.demos.hellorestjava -DartifactId=hello-rest -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
  - Build Package
    - cd hello-rest; mvn clean package
  - Run
    - java -cp target/hello-rest-1.0-SNAPSHOT.jar com.brainwashinc.app.App

```
Bear-MBP:hello-rest bearc2020$ java -cp target/hello-rest-1.0-SNAPSHOT.jar com.brainwashinc.app.App
Hello World!
```

Java Standalone Server

- Spark
  - "A micro framework for create web applications"
  - Add to project: http://sparkjava.com/download
  - Add to pom.xml

```
<dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-core</artifactId>
    <version>2.9.4</version>
</dependency>
```

Java Standalone Server

- Spark
  - Update App class
    - Imports

```
import static spark.Spark.*;
import spark.Request;
import spark.Response;
import spark.Route;
```

    - Route

```
public class App
{
    public static void main( String[] args )
    {
        get("/hello", (req, res) -> "Hello World");
        // System.out.println( "Hello World!" );
    }
}
```

    - Build: mvn clean package (error: next slide)

Java Standalone Server

- **Spark**
  - **Error**

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-c
ompile) on project hello-rest: Compilation failure
[ERROR] /Users/Shared/Downloads/pomgen/hello-rest/src/main/java/com/brainwashinc/app/App.java:[16,34]
lambda expressions are not supported in -source 7
[ERROR]   (use -source 8 or higher to enable lambda expressions)
```

  - **Update pom.xml maven to 1.8**

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
```

Java Standalone Server

- **Spark**
  - **Run: e.g.,**
  - **mvn exec:java -Dexec.mainClass="com.brainwashinc.app.App"**

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

  - **Update pom.xml:**

```
     <dependency>
          <groupId>org.slf4j</groupId>
          <artifactId>slf4j-simple</artifactId>
          <version>1.7.21</version>
     </dependency>
```

  - **Rebuild, Rerun**

Java Standalone Server

- **Output with port: 4567**

```
[Thread-1] INFO org.eclipse.jetty.util.log - Logging initialized @1265ms to org.eclipse.jetty.util.log.Slf4jLog
[Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - == Spark has ignited
[Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - >> Listening on 0.0.0.0:4567
[Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.4.31.v20200723; built: 2020-07-23T17:57:36.812Z; git: 450ba27947e13e66baa8c
d1ce7e85a4461cacc1d; jvm 17+35-LTS-2724
[Thread-1] INFO org.eclipse.jetty.server.session - DefaultSessionIdManager workerName=node0
[Thread-1] INFO org.eclipse.jetty.server.session - No SessionScavenger set, using defaults
[Thread-1] INFO org.eclipse.jetty.server.session - node0 Scavenging every 600000ms
[Thread-1] INFO org.eclipse.jetty.server.AbstractConnector - Started ServerConnector@6becc6b5{HTTP/1.1, (http/1.1)}{0.0.0.0:4567}
[Thread-1] INFO org.eclipse.jetty.server.Server - Started @1354ms
```

- **Hit with browser or curl:**
  - **curl http://localhost:4567/hello**

```
Hello World
```

Java Standalone Server

- **Change the main**
  - **Add a path for /user**
    - **Takes an id and returns the parameters**

```
path("/user", () -> {
    get("/:id", (req, res) -> req.params());
});
```

- **Rebuild; rerun**
- **Hit with browser or curl:**
  - **curl http://localhost:4567/user/123**

```
{:id=123}
```

Java Standalone Server

- Change the main
  - Prefix that path with a v1 path

```
path("/v1", () -> {
    path("/user", () -> {
        get("/:id", (req, res) -> req.params());
    });
});
```

  - Rebuild; rerun
  - Hit http://localhost:4567/v1/user/123

```
{:id=123}
```

Java Standalone Server

- Add a before and after

```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"));
    path("/user", () -> {
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```

  - Rebuild; rerun
- Hit http://localhost:4567/v1/user/123
- See log output

```
Received api call
Received api done
```

Java Standalone Server

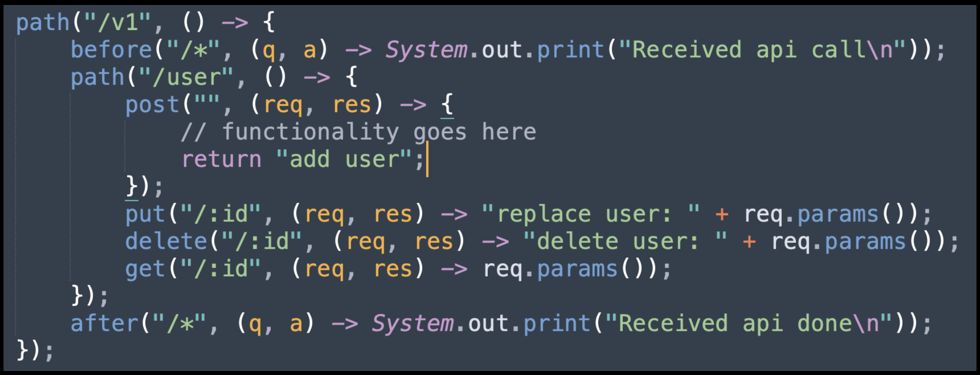
- **Add other methods**

```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"))
    path("/user", () -> {
        post("", (req, res) -> "add user");
        put("/:id", (req, res) -> "replace user: " + req.params());
        delete("/:id", (req, res) -> "delete user: " + req.params()
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```

- **Rebuild; rerun**

Java Standalone Server

- **Of course you need actually functionality for the methods…**



```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"));
    path("/user", () -> {
        post("", (req, res) -> {
            // functionality goes here
            return "add user";
        });
        put("/:id", (req, res) -> "replace user: " + req.params());
        delete("/:id", (req, res) -> "delete user: " + req.params());
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```

Java Standalone Server

- Query those methods
  - curl -X POST http://localhost:4567/v1/user
  - curl -X PUT http://localhost:4567/v1/user/1
  - curl -X GET http://localhost:4567/v1/user/1
  - curl -X DELETE http://localhost:4567/v1/user/1
- Spark documentation:
  - http://sparkjava.com/documentation

Python Standalone Server Using Flask

Python-Flask Standalone Server

- Install python
  - https://www.python.org/downloads/

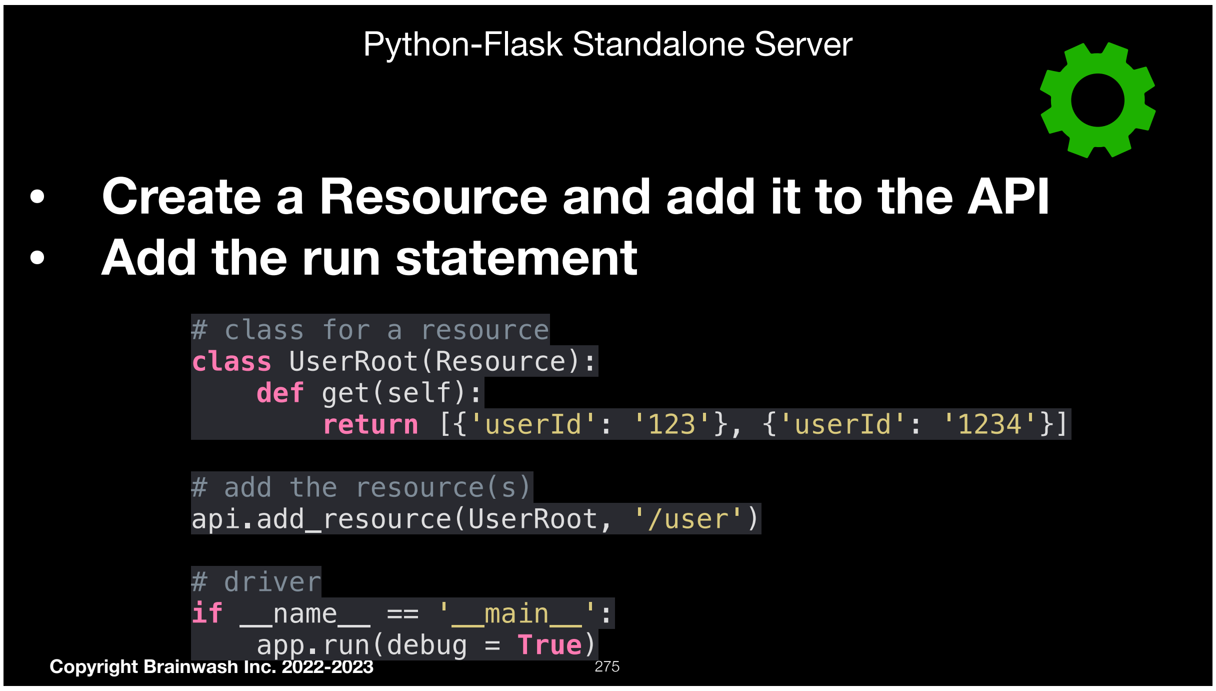
- Make a project directory (e.g., pyflaskrestful)
- Create and active an environment
  - python3 -m venv .venvrest
  - source .venvrest/bin/activate

- Install flash and flask-restful
  - pip3 install flask
  - pip3 install flask-restful

Python-Flask Standalone Server

- Create an app.py file
  - Add imports
  - Create API object

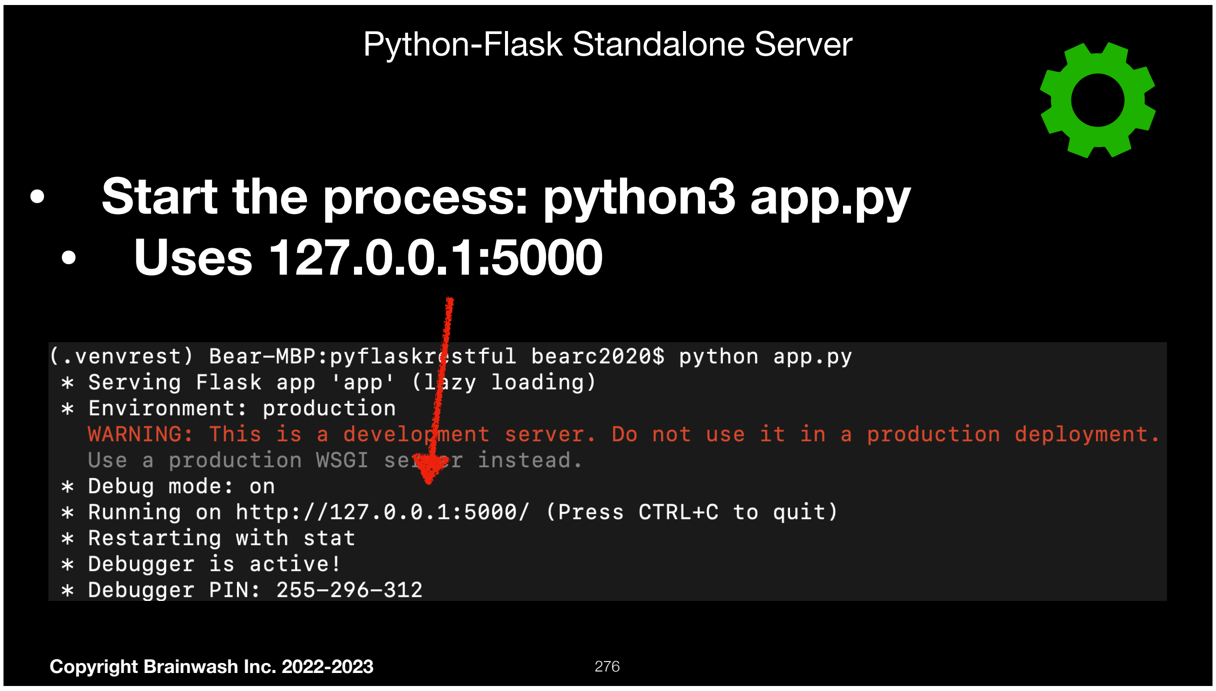
```
# using flask_restful
from flask import Flask, jsonify, request
from flask_restful import Resource, Api

app = Flask(__name__)
# API object
api = Api(app)
```



Python-Flask Standalone Server

- Create a Resource and add it to the API
- Add the run statement

```python
# class for a resource
class UserRoot(Resource):
    def get(self):
        return [{'userId': '123'}, {'userId': '1234'}]

# add the resource(s)
api.add_resource(UserRoot, '/user')

# driver
if __name__ == '__main__':
    app.run(debug = True)
```



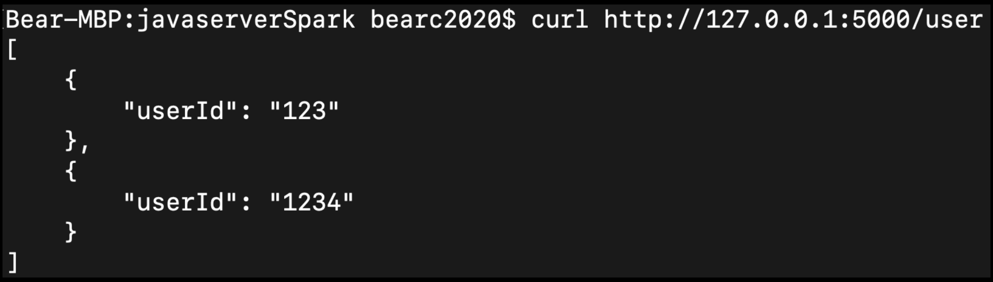
Python-Flask Standalone Server

- Start the process: python3 app.py
  - Uses 127.0.0.1:5000

```
(.venvrest) Bear-MBP:pyflaskrestful bearc2020$ python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 255-296-312
```

Python-Flask Standalone Server

- **Hit the endpoint**
  - **curl http://127.0.0.1:5000/user**



```
Bear-MBP:javaserverSpark bearc2020$ curl http://127.0.0.1:5000/user
[
    {
        "userId": "123"
    },
    {
        "userId": "1234"
    }
]
```

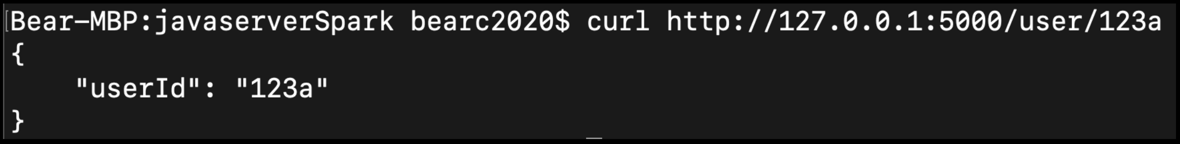
Python-Flask Standalone Server

- **Create a User level resource**
  - **Takes userId from path**
- **Add it below the existing UserRoot**

```
class User(Resource):
    def get(self, userId):
        return {'userId': userId}

# add the resource(s)
api.add_resource(UserRoot, '/user')
api.add_resource(User, '/user/<userId>')
```

Python-Flask Standalone Server

- Hit the new endpoint
  - curl http://127.0.0.1:5000/user/123a



```
Bear-MBP:javaserverSpark bearc2020$ curl http://127.0.0.1:5000/user/123a
{
    "userId": "123a"
}
```

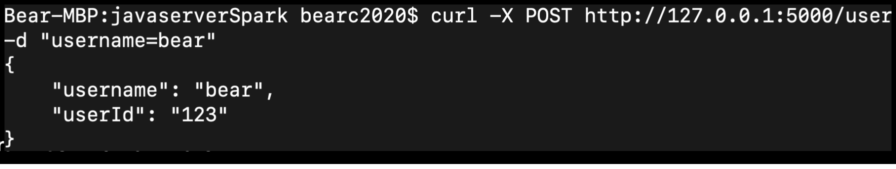
Python-Flask Standalone Server

- Create post method to UserRoot
  - Get the form data from the request

```
class UserRoot(Resource):
    def post(self):
        data = request.form['username']
        return {'username':data, 'userId':'123'}, 201
```

  - Hit the endpoint with data

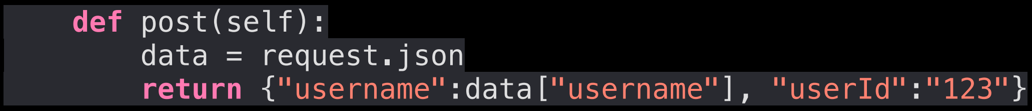
```
curl -X POST http://127.0.0.1:5000/user -d "username=bear"
```



```
Bear-MBP:javaserverSpark bearc2020$ curl -X POST http://127.0.0.1:5000/user
-d "username=bear"
{
    "username": "bear",
    "userId": "123"
}
```

Python-Flask Standalone Server

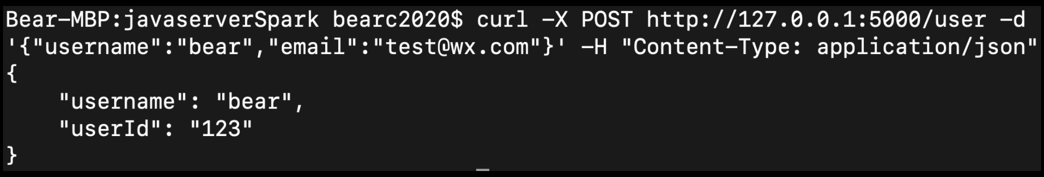
- **Change the post endpoint to expect a JSON body**
  - **Return data including body data**



```
    def post(self):
        data = request.json
        return {"username":data["username"], "userId":"123"}
```

Python-Flask Standalone Server

- **Hit the endpoint with JSON data and content-type header**
- **curl -X POST http://127.0.0.1:5000/user -d '{"username":"bear","email":"test@wx.com"}' -H "Content-Type: application/json"**



```
Bear-MBP:javaserverSpark bearc2020$ curl -X POST http://127.0.0.1:5000/user -d
'{"username":"bear","email":"test@wx.com"}' -H "Content-Type: application/json"
{
    "username": "bear",
    "userId": "123"
}
```

Python-Flask Standalone Server

Final Full Code

```
# using flask_restful
from flask import Flask, jsonify, request
from flask_restful import Resource, Api

app = Flask(__name__)
# API object
api = Api(app)

# class for a resource
class UserRoot(Resource):
    def post(self):
        data = request.json
#        return data
        return {"username":data["username"], "userId":"123"}
#        data = request.form['username']
#        return {'username':data, 'userId':'123'}, 201

    def get(self):
        return [{'userId': '123'}, {'userId': '1234'}]

class User(Resource):
    def get(self, userId):
        return {'userId': userId}

# add the resource(s)
api.add_resource(UserRoot, '/user')
api.add_resource(User, '/user/<userId>')

# driver
if __name__ == '__main__':
    app.run(debug = True)
```

RESTful API Design and Development

The End