ListenUp

In this exercise, you will build a new web service that combines information from existing separate web APIs into a single resource.
The web service is based on a theoretical internet radio project called “ListenUp”.

Your solution will be evaluated on code quality, clarity and development best practices.

Goal

Your web service will need to listen on port 8005 and implement the following HTTP resources:

1. GET http://localhost:8005/users

```
Return a list of existing users


{
   “users” [
      {
         "username": "joe_example",
          "plays": 178,
          "friends": 7,
          "uri": "/users/joe_example"
      },
      ... snip additional users ...
   ],
   “uri”: “/users”
}
```

2. GET http://localhost:8005/users/joe_example

```
Return detailed information about a user


{
    "username": "joe_example",
    "plays": 178,
    "friends": 7,
    "tracks": 23,
    "uri": "/users/joe_example"
}
```

Note: “tracks” represents the number of unique tracks that this user has played.

Choice of technology

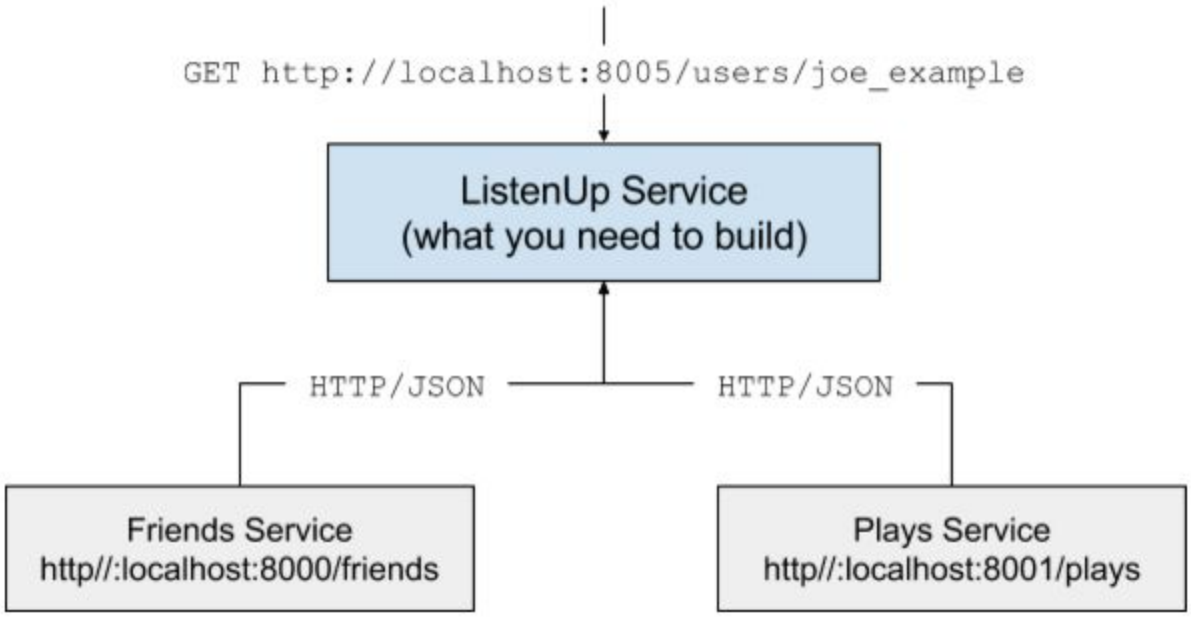
The exercise can be completed in any language/framework, but please take guidance from the recruiter as to which technology to use. A skeleton Java application is provided (java directory) to get you started if you decide to use Java. This uses the Spark framework which requires Java 8. A skeleton Python application is also provided (python directory).

Suggested resources for other languages/frameworks are available at the end of this document.

Available Services

Overview

Your web service will need to **retrieve information from two separate existing web services** to build the response detailed above. These web services already exist and **do not need to be created**, and have to be run locally.



Install & Run services locally

A makefile is available to run the existing services (Friends and Plays) written in Python.
To run these services, you need install command:

```
make install
```

and this command will start the services:

```
make services
```

These command will start the ‘Friends’ and ‘Plays’ services, detailed hereafter:

Friends Service

Available at http://localhost:8000

The friends service lists all friends associated with a given user. The following API exists on the friend service:

```
GET http://localhost:8000/friends
Returns a list of all users that the service has friend information for, example:

{
    "friends": [
        {
            "username": "ray_benigno",
            "uri": "/friends/ray_benigno"
        },

        ... snip other users ...
    ],
    "uri": "/friends"
}
```

```
GET http://localhost:8000/ray_benigno
Returns the list of friends for the specified user

{
    "friends": [
        "maranda_kjos",
        "jacquetta_hoelscher",
        "garth_coto",
        "leonor_mattis"
    ],
    "uri": "/friends/ray_benigno"
}
```

Plays Service

Available at http://localhost:8001

The play service lists all records of a user ever playing specific tracks.

```
GET http://localhost:8001/plays
Returns a list of all users that the play service has information on, example:

{
    "users": [
        {
            "username": "loris_stuckey",
            "uri": "/plays/loris_stuckey"
        }
        ... snip other users ...
    ],
    "uri": "/plays"
}
```

```
GET http://localhost:8001/plays/loris_stuckey
Returns a given user's play history

{
    "plays": [
        ... snip other plays ...
        "E75C38C1-E2BB-BAF6-620B-9255D035B693",
        "E75C38C1-E2BB-BAF6-620B-9255D035B693",
        "68B4D809-4B4F-F735-EB92-E5B17C99220B"
    ],
    "uri": "/plays/loris_stuckey"
}
```

Suggested Resources

Lightweight Web Frameworks:

- Java: Spark, JAX-RS, DropWizard
- Scala: Spray, Scalatra
- C#: Nancy
- Python: Flask
- Ruby: Sinatra