Invent an idea for a useful server (or service) that 1. maintains some relevant data persistently during its operation, 2. supports multiple clients, and 3. has no GUI.

Maintaining data persistently means that when your server terminates and starts up again some arbitrary time later, the data still exists and can be used and modified by later processing.

Multiple clients means some of the data may be associated with one client and some with another. There may also be some aggregate or shared data. Your server needs to be able to tell the clients apart and it should provide basic protection for the integrity and confidentiality of client-specific data, protecting it from other clients. You do not need to consider carefully crafted hacker exploits or protecting client data from the server-provider itself. It is strongly recommended that you use some "free" widely-used authentication package, e.g., Auth0 (Links to an external site.), not develop the security code yourself. Clients can be other servers and services with their own clients, not just end-user apps.

Instead of a GUI (graphical user interface), your server should provide an API (Application Programming Interface).

Part 1:

If your team name, team membership, planned programming language, platform and/or team GitHub repository has changed since your team formation assignment, please explain. If not, simply state that nothing has changed.

Answer:

No change to prior submission; information re-pasted for safety:

Team Name:

Grey Orange

Team Membership & UNIs:

Name: Yuhao Dong
UNI: yd2626

Name: Joe Rebagliati
UNI: jr4162

Name: Ryan Soeyadi
UNI: rs4163

Name: Madison Thantu
UNI: mgt2143

Planned Programming Language & Platform:

Docker containers will be used for Java development

GitHub repository:

https://github.com/rsoeyadi/friendly-couscous

Part 2:

Write a few paragraphs that provide an overview of the server that your team would like to develop and answers these three sets of questions:

1. What will your server do? What kind of functionality or features will it provide?
2. Who or what will be its clients? What might they use the functionality for?
3. What kind of data will your server create or accumulate? What will the data be used for? For this assignment, you do not need to present your features as user stories or use cases.

Our service is a music playlist management service. The service will process data accessed via a persistent Postgres database after which such data will be processed via Java functions. Certain functions will be publicly accessible via a RESTful API. The service will provide the ability to track popular songs in different playlists, create playlists, create groups, and modify the underlying database for the service. A client could create/participate in different groups that might like a similar type of music. A client could also create accounts, create playlists, and retrieve songs (with similar types) from our database and add them to customized playlists. A client may also post new songs to the underlying database accessible to all clients. We can also analyze the playlists for a client. For example, we can find out what are the three most popular songs in a list, and the average length of songs in a playlist.

Our clients could be any application/organization that wants to create its own music service with analytics for a user base (e.g., music lovers, etc.). They could use the functionalities stated above to create their own playlists and their own groups, and add the songs to the database.

For our service, we will have a few different data schemas. One is the user profiles (UID, Name, account name, password, etc). Another one would be the groups that could be linked to users. We would also have a data schema for playlists that are linked to individual playlists, and songs (their UID) in the playlist. We will have a separate database to store the songs. We are planning to import base song data from an open-source dataset (https://github.com/mdeff/fma).

(for songs, we can have a column listing how many people added it to the list)

Part 3:

Write a few paragraphs that describe, at a high level, how you plan to test the functionality of your server without any clients, GUI, or otherwise. (It's ok to use testing tools that have GUIs.) You will expand testing in the second iteration to include sample clients, but need to test stand-alone during the first iteration. Think in terms of testing your server as a whole, in addition to unit testing of individual subroutines, and answer these three questions:

1. How will you test that your server does what it is supposed to do and provides the intended functionality?
2. How will you check that your server does not behave badly if its clients use it in unintended ways or provide invalid inputs?
3. How will you test that your server handles its data the way it's supposed to?

First, while we are developing our service, we will use unit testing with JUnit (https://junit.org/junit5/) to ensure that we are writing functions to encapsulate the smallest possible logical groupings and that these functions work as intended. Next, we will do integration testing to test higher-level functions that are tied to routes per the RESTful API. To do this, we will use Postman (https://www.postman.com/). For example, the functionality to add a user to a playlist could be attached to a route /addUser with a POST request. Postman allows us to test such routes while developing on localhost.

In order to test that the service does not behave badly if a client uses it in an unintended way or provides invalid inputs, we will test edge cases. One edge case to consider is non-standard entry points to a function like global variables. If somehow a client accesses non-standard entry points, we want to ensure that the service does not break. We will also test a variety of invalid inputs passing them to the service entry points (routes). If we detect that the client is entering invalid inputs, we will return the client a 400-level status code and not execute its request.

Since we are doing unit testing, this should help ensure that the server handles data the way it is supposed to. For example, if we have a private function called averageSongLength(), unit testing should allow us to create assertions we know should be true or false based on the input for the aforementioned function. To be more specific, if averageSongLength(song1) should return 5, then our unit test would assert that true should be returned from averageSongLength(song1) == 5.

https://www.postman.com/

Submission instructions: One member of your team should submit a single file (preferably a pdf) presenting your preliminary proposal. Your file must contain your team name and the names/unis of all team members. You may submit repeatedly until the deadline. Note that if multiple team members submit, the most recent submission will override all previous submissions by the same or different team members.

Criteria	Ratings		Pts
Part 1 1 point each for 1. team name, 2. names of all team members, 3. unis of all team members, 3. programming language 4. platform 5.GitHub repo	**5 to >0.0 pts Full Marks**	**0ts No Marks**	5 pts
Part 2 2 points each for answering the 3 questions: 1. What will your service do? 2. Who or what will be its users? 3. What kind of data will your service create or accumulate? What will the data be used for?	**6 to >0.0 pts Full Marks**	**0 pts No Marks**	6 pts
Part 3 2 points each for answering 3 questions: 1. How will you test that your service does what it is supposed to do and provides the intended functionality? 2. How will you check that your service does not behave badly if its clients use it in unintended ways or provide invalid inputs? 3. How will you test that your service handles its data the way it's supposed to?	**6 to >0.0 pts Full Marks**	**0 pts No Marks**	6 pts
Overview Project ideas meet the 3 requirements: 1. Maintain relevant data persistently 2. Supports multiple clients 3. No GUI	**3 to >0.0 pts Full Marks**	**0 pts No Marks**	3 pts

Total Points: 20

QUESTION FOR EDSTEM LIKE MAYBE SORT OF - HOPEFULLY NOT:

Hello, we are working on a music server that provides the ability to create users, associate users with playlists, create playlists, add new songs to the server's database, etc. For the source song data that will be used, we are thinking to use the Spotify API (and/or other similar server APIs) to pull data and then save the information in a CSV file to be loaded as part of our server on start up. In other words, the pull of API data is out of scope of the server ; we will pretend we already had it for purposes of the server. Is this (1) an acceptable approach for a server (i.e., not too specific to be consider an application / client with a command line GUI) and (2) is our approach regarding source data okay (i.e., we are allowed to pull source data however we want and treat it as a given). Regarding our server being required to "do something different" from the list of public APIs provided, would this functionality suffice, or should we implement more unique functionality?

```
USER GROUP ->
        {Unique ID, Name, List<Users>}
                USER ->
                        {Name, Unique ID, NickName}
                        {Playlist1, Playlist2, Playlist3, …}
                                Playlist -> {UID, List<Songs>, Name}
```

- Functionalities:
    - Collect and aggregate music tracks from a variety of music streaming platforms E.g.,
        - Spotify
        - SoundCloud
        - Apple Music
        - Tidal
    - Create unique playlists based on this aggregated set of music
    - Merge existing playlists
- Song data to capture:
    - Track name
    - Artist
    - Album
- Here's a server that can track these things as you choose to create them
- With our service, the client is able to create their own playlists
- Key functionalities
    - The ability to create a playlist from songs

- Given a user, who is associated with 1 or more playlists, a name, a unique ID, a nickname, etc.
- Creating a user group
    - A user group contains 1+ users, maintains associations with each of those users
- We are providing the ability to construct a graph of all of these different relations and all of the objects within these relations at whatever the client chooses to do

NOTES

- Integration testing
    - Spin up our service in the cloud; define a set of actions that will ping your API and these actions will have a set of expected outcomes
    - Testing the API itself
- Unit testing
    - Running individual functions on the back-end

FEEDBACK

- Specify how we're going to conduct integration testing
- Need to define who our users will be
- Can streamline our proposal to bullet points
- Remove redundancy
- Writing a back-end in Java. The back-end will have a RESTful architecture
- We describe in general all of the actions that a generic client could take. In the rubric they're asking about specific groups of users. There are the listeners who will be making playlists and liking songs and playing songs back. There are the artists who will be uploading songs and organizing them. There will be admins who perform things like content moderation.
    - We also talked about exposing data analytics metrics.
    - You have everyday users like listeners, artists, admins. Then we have the non-daily users like data scientists who
- Group functionality by user type/user persona
- Client in this case means software. Users will access our API via the client.
    - Example: Youtube
        - There is one content view for viewers. There is another software for creators. Both of those users (viewers and creators) interact with youtube via the software.
- We are implementing a route that a version of spotify could hit
    - Lets say there are two versions of spotify - one for ar
    - We have one backend that both of those clients are going to query. So our job is to think of all of the routes and functionality that we would need to allow all types of users to utilize this
    - You have groups of users. You want to ask the question which client will they be using to get to our backend. Once we figure out all of the clients that our users will be using, we can then determine what routes we need
    - We aren't implementing the client. We are just imagining what could be implemented. And then we are supplying the routes
- Need to capture our back-end more explicitly
- Make first paragraph in part 3 less verbose
- Write subsection for edge cases
- Authentication versus authorization
    - Authentication - User says who they are
    - Authorization - Based on who they are, what is this user able to d

        - Non-standard entry point - listener tries to delete somebody else's songs, which don't belong to them
- delete/cut down last paragraph of part 3
- Since we're using postman - Make a diagram
    - Want one database and inside that database make tables to categorize our data
- Draft out our schema
    - Write out all of our tables and think about what data each of the tables has
    - If you have a song, which artist doesxt the song belong it
    - If you have a playlist, what user does that playlist belong to
    - Who owns what, what belongs to who