FindAGig Final Project — Spring Boot

Week5 & 6 URL to GitHub Repository: https://github.com/sw-dev-lisa-s-nh/SpringBoot-Final-Project.git

Requirements:

Your API must meet the following, minimum requirements:

- ***DONE**: Contain at least 5 entities. (User, instrument, Gig, GigStatus, Address)*
- *Contain all CRUD operations — the framework is there — but need to be tested!*
- ***DONE**: Contain at least 1 one-to-many relationship*
- ***DONE**: Contain at least 1 many-to-many relationship*
- ***DONE**: Contain different application layers including at least controller, service, and repository*

Final Project

My Final Project is a backend WebAPI that helps musicians connect with possible events and gigs & additionally have event or gig planners connect with available musicians. There are users (musicians & planners), aspect of location (address entity), instruments (instrument entity), and events (gig entity).

- A **musician,** once created (*FUTURE: registered and logged in),* would be able to search available events, request to play at a particular event or gig.
- An **event organizer**, once created (*FUTURE: registered and logged in),* would be able to create an event or gig, with a specific number of OPEN positions, each OPEN position will be tied to an instrument. They can look at their gigs, and if a gig has been REQUESTED by a musician, they can set that spot to CONFIRMED. They can also set a gig to CANCELLED (e.g., if a wedding has been called off).

Database: findagigDB

Entities:

- **User** — written & working "/users"
  - UserService — written & working
  - UserController — written & working
  - DELETE OPERATION — Needs to be FIXED — only allowed by the User for their own account, and only allowed if they want to delete their account entirely!
- **Address** — written — stored via User & Gig. [**MANYTOONE** user to address]
- **Instrument** — written & working "/instruments" [**ONETOMANY** user to instrument]
  - InstrumentService — written & working
  - InstrumentController — written & working
  - InstrumentRespository— written & working
  - DELETE OPERATION — NOT ALLOWED!
- **Gig** — written & working "/gigs" [**ONETOMANY** gig to gig_musician_match]
  - GigService — written & working
  - GigController — written & working
  - GigRepository — written & working
  - DELETE OPERATION —NOT ALLOWED!
- **Musician_instrument** — Join Table. — DONE!
  - [**MANYTOMANY**] user.id to musician_instruments.userId
  - [**MANYTOMANY**[instruments.instrumentId to musician_instruments.instrumentId
- **GigStatus_instrument** — Join Table. — DONE!
  - [**MANYTOMANY**] gig.instrumentId to gig_status.instrumentId
  - [**MANYTOMANY**[gig_status.instrumentId to instruments.instrumentId
- **Gig_Status** — Connects Gigs to Instruments — One record for each instrument per gig, and contains salary for a musician, as well as a status.

Example Application (API) Functionality: **IN PROGRESS:** DONE & **NOT FINISHED YET**

 By: Lisa Smith

Two types of Users — Musicians & Planners:

- **User(Musician or Planner)** can be created, updated and read — (deleted is not allowed).
- **Instruments** can be created,u pdated or read (delete is not allowed)
- **Users** can be created or updated **with instruments**
  - If the **instrument** doesn't exist, an **instrument** entry **will** be created.
- **Gigs** can be creates, updated or read — **delete** is NOT ALLOWED
- **User(Musician)** "/gigs/{gigId}/users/{userId}/request" can REQUEST gig with **OPEN** status & matching instruments
  - gig_instrument **STATUS** will be updated to **REQUESTED**
  - musician_id will be set to {userId}

Extensions to the functionality for the next two weeks:

- **User(Musician)** can browse all available gigs in their state
- **User(Musician)** can browse at all available gigs by instrument
- **User(Musician)** can view all details of a specific gig,
  - Will list all instruments required
  - Will list datetime, city, state, location
- ***User(Planner)*** *can update gigs*
  - If a gig has gig_instruments that have a **STATUS** of **REQUESTED**, they can change status back to:
    - **CONFIRMED** (send a notice to musician — hired!)
    - ***OPEN*** *(send a notice to musician — no thank you)*
  - *Can set a gig to* ***CANCELLED*** *or* ***CLOSED*** *— add functionality*
  - *If a gig has* ***OPEN*** *gig_instruments*
    - *Send a notice to all musicians of a particular instrument — Musicians needed!*

Future extension (NOT PLANNED FOR THIS PROJECT):

1. ***Entity: Credentials*** — "/register" & "/login"
   - **What is already done:**
     - *Entity Exists —* ***Credentials***
     - *Controller and Service exist for* ***/users/register*** *&* ***/users/login***
     - *Password & Username are stored in* ***Credentials***
     - *Password is salted & hashed & stored in hash in entity* ***User***
   - ***TO DO:*** *When a User registers, actually call createUser from userService.createUser(); Currently, only the username is stored*

2. *New Type of User: System Admin: (will have privilege to do whatever is needed in the API & database)*
   - **What is already done:**
     - **ADMIN** is already an option in **UserType**
   - ***TO DO:*** *Future Functionality: (Nothing here is implemented)*
     - *Admin can create instruments & musicians*
     - *Admin can read instruments & musicians*
     - *Admin can update instruments & musicians*
     - *Admin can delete instruments & musicians*
     - *Admin can create gigs*
     - *Admin can read gigs*
     - *Admin can update gigs*
     - *Admin can delete gigs*

By: Lisa Smith