Final Project — Spring Boot

Week 4 URL to GitHub Repository: https://github.com/sw-dev-lisa-s-nh/SpringBoot-week4.git

Requirements:

Over the next four weeks you will be working on building your own API. The API can be anything you want, as long as it doesn't violate any school policies. Your API must meet the following, minimum requirements:

- ***DONE**: Contain at least 5 entities*
- *Contain all CRUD operations*
- *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*

DONE: *This week, please come up with an idea for your project and get it approved by your instructor as early as possible so you can start working on it. It is also recommended that you create a list of features that your API will have so you can work from that list. B*elow is an example.

Library API Example:

- *User can register and login*
- *Admin can promote user to admin*
- *Admin can create books*
- *Admin can delete books*
- *Admin can update books*
- *User can browse all books*
- *User can browse books by genre*
- *User can view all details about a specific book*
- *Users can leave reviews on a book*
- *User can check out x amount of books at a time (creates a checkout entity)*
- *When a book is checked out a due date is set for 2 weeks from checkout date*
- *User can return book*
- *When a book is returned a fee is assessed on the users account if the book is past due*

My Proposal:

Idea: My idea is to have a website that helps musicians connect with possible events and gigs & additionally have event or gig planners connect with available musicians. There are users (user & credential entity), aspect of location (address entity), musicians (musician entity), instrument (instrument entity), and events (event or gig entity).

- A **musician** would register to use my website, and then when they log in, they can search available events, request to play at a particular event or gig.
- An **event organizer**, also a user, would register & log in, and that user could create an event or gig, and create a number of OPEN positions within the event.

Database: findagigDB

Entities:

- **User** — written & working "/users"
  - UserService — written & working
  - UserController — written & working
  - DELETE OPERATION — Needs to be FIXED
- **Address** — written — stored via User & Gig. [**ONETOONE** user to address]
- ***Credentials*** — written "/register" & "/login" ***(NOT FINISHED YET)***
- **Instrument** — written & working "/instruments" [**ONETOMANY** user to instrument]
  - InstrumentService — written & working
  - InstrumentController — written & working
  - InstrumentRespository— written & working
  - DELETE OPERATION — Needs to be FIXED — doesn't cascade to the gig_status.
- **Gig** — written & working "/gigs" [**ONETOMANY** gig to gig_musician_match]
  - GigService — written & working
  - GigController — written & working
  - GigRepository — written & working
  - DELETE OPERATION — Needs to be FIXED — can't delete gig (need to write this)

- **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

Two types of Users — Musicians & Planners:

- **User(Musician or Planner)** "/users/register" can register (creates credentials with salting & hashing, and stores the password in the database, but only the hash is visible).
- **User(Musician or Planner)** "/users/login" can login (using credentials)
- **User(Musician or Planner)** can be created, updated and read — deleted needs to be fixed.
- **Instruments** can be created, deleted, updated or read (cascades — except gig_status)
- **Users** can be created or updated **with instruments**
  - If the **instrument** doesn't exist, an **instrument** entry **will** be created.
  - **BUG**: User is created BUT connection in musician_instrument is NOT!
- **Gigs** can be creates, updated or read — deleted needs to be fixed.
- **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 area*
- ***User(Musician)*** *can browse at all available gigs by instrument required*
- ***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:*
    - ***OPEN** (send a notice to musician — no thank you)*
    - ***CONFIRMED** (send a notice to musician — hired!)*
  - *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):

System Admin: (privilege to do whatever is needed in the API & database)

- *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*