# FindAGig Final Project — Spring Boot

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

**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"
    - Entity, UserRepository & UserService — written & working
    - CRUD operations in UserController & UserService —  written & working
- **Address** — written — stored via User & Gig.  [**MANYTOONE** user to address]
-                                           [**MANYTOONE** gig to address]
- **Instrument** — written & working "/instruments"
    - Entity, InstrumentRepository & InstrumentService — written & working
    - CRUD operations in InstrumentController & InstrumentService — written & working
- **Gig** — written & working "/gigs"              [**ONETOMANY** gig to gig_status]
    - Entity, GigRepository & GigService — written & working
    - CRUD operations in GigController & GigService — written & working
    - ***DELETE OPERATION — NOT ALLOWED** By Design!*
- **Musician_instrumen**t — 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.

**Final Project Requirements:**
*Your API must meet the following, minimum requirements:*
- ***DONE**: Contain at least 5 entities:  (**User, Instrument, Gig, GigStatus, Address**)*
- ***DONE**: Contain all CRUD operations — **CRUD - Instrument, CRUD - User,** and **CRU for Gig**. Delete is only allowed on a Gig in specific circumstances.*
- ***DONE**: Contain at least 1 one-to-many relationship (**gig** to **gig_status** is a one to many relationship, **address** to **user** is a one to many relationship, and **address** to **gig** is a one to many relationship)*
- ***DONE**: Contain at least 1 many-to-many relationship (**musician_instrument** & **gig_status_instrument** join tables are many-to-many)*
- ***DONE**: Contain different application layers including at least controller, service, and repository*

**FindAGig Web API Functionality:**　(See *findAGigAPI Documentation* for more details!)

**Two types of Users — Musicians & Planners:**

- **Users** can be *CREATEd* **(POST)**, *UPDATEd* **(PUT)**, *READ* **(GET)**, & *DELETEd* **(DELETE)**).
  - In *CREATE*:　if the **instrument** doesn't exist, an **instrument will** be created.
  - In *DELETE:*　the **instrument** record is not deleted, only the connection to the user.

- **Instruments** can be *CREATEd* (POST), *UPDATEd* (PUT), *READ* (GET), & *DELETEd* (DELETE).

- **Gigs** can be c *CREATEd* (POST), *UPDATEd* (PUT), or *READ* (GET),)— **DELETE** is NOT ALLOWED
  - In *CREATE*:　if the **instrument** doesn't exist, an **instrument will** be created.

- **Addresses** are *CREATEd* within User or Gig creation — and never deleted!
  - In *USER* or *GIG CREATE* — if address doesn't exist, an **address will** be created.
  - In *DELETE:*　the **address** record is **not** deleted, only the connection to the user.
  - 

- **User(Musician)** Can *UPDATE* a Gig (must be OPEN and have matching instrument)
  - **"/gigs/{gigId}/users/{userId}/request"** — REQUEST gig
    - gig_instrument **STATUS** will be updated to **REQUESTED**
    - musician_id will be set to {userId}

- **User(Musician)** can *READ* information for — **GET**
  - **"/gigs"** — All gigs
  - **"/gigs/open"** — All available gigs by OPEN status
  - **"/gigs/state/{stateName}"** — All gigs by state
  - **"/gigs/open/state/{stateName}"** — All OPEN gigs by state
  - **"/gigs/instrument/{instrumentName}**" — All gigs by instrument
  - **"/gigs/open/instrument/{instrumentName}"** — All OPEN gigs by instrument
  - **"/gigs/genre/{genreName}"** — All gigs by genre
  - **"/gigs/open/genre/{genreName}"** — All OPEN gigs by genre
  - **"/gigs/{gigId}"**— All details this gigId:　list instruments required!
  - **"/gigs/users/{userId}"** — List all gigs assigned to this user!
  - **"/gigs/{gigId}/users"** — Lists all users assigned to this gigId!

- **User(Planner)** can do the following:
  - *CREATE* gigs & instrument requirements for a new gig — **POST "/gigs"**
  - Add (*CREATE*) instrument requirements to an existing gig — **POST "/gigs/{id}"**
  - *UPDATE* gigs: **PUT**
    - **"/gigs/{id}/users/{musicianId}/confirm/{plannerId}"**—
      - If a gig has gig_instruments that have a STATUS of **REQUESTED**, they can change status to:　**CONFIRMED**
    - **"/gigs/{gigId}/status/{statusType}"** —
      - Can change the STATUS of their gig to **PLANNED**, **OPEN**, **CANCELLED,** or **CLOSED**

　　　　　　　　　　　　　　　　　　　　　　　　　　**By:　Lisa Smith**

# Future extension (to be implemented in the future!):

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
          - Add **e-mail storage**, and ability to communicate (send e-mail or a notification) when a status changes that affects a User!

2. **_User(Planner) enhancements:_**
     - **TO DO:**
          - Send a notice to musician if they are not "CONFIRMED"! Or if they are!
          - Allow communication to all users (email via a stored e-mail address, or a notification) when a Gig is CONFIRMED, CANCELLED or CLOSED!
          - If a Planner's gig (User is the **Planner**) has **OPEN** gig_instruments
               - Send a notice to all musicians of a particular instrument — Musicians needed!

3. **_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 defined in **UserType**
     - **_TO DO: (_**Future Functionality:)
          - 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