Expeditors – Java Academy
Capstone 03 Instructions & Requirements

Enhance the API created in Capstone 01 & 02. As a reminder, capstone 02 included the addition of Spring Boot-based REST API endpoints to our application for a Music and Entertainment management system (tracks and artists). For Capstone 03, you will be refactoring your application to add database persistence in a PostgreSQL database to your previously defined REST API endpoints. You should be able to successfully exercise your API via tests and via a tool like cURL, Postman, or an equivalent (e.g., http request files in IntelliJ). Additionally, you will be adding JWT-based authentication to your API to ensure calls to data management endpoints are secured.

- Students will work in teams designing and building the application
    - Design work should be done as a team – each member of the team should be aligned on any design decision made
    - Implementation individually or as a group using paired or mob programming; if executed as a group, it is critical that every member of the team get an opportunity for "hands on keyboard" coding
    - Each team should have a short meeting at a fixed time every day to talk about status and any issues or blockers that team members are facing

Features:

- Continue to provide CRUD (Create/Retrieve/Update/Delete) functionality
- Design and create the database and tables required by the application
- Create JPA Entities with appropriate relationship mappings – you can choose to use the defined database entities directly in code or you can add a DTO layer with a tool like MapStruct (or similar approach) for mapping between the two
- Refactor the existing code as necessary to allow for persistence
- Persist the data in a database
- JWT-based Authentication/Authorization – You can use any of the following approaches discussed in our VILT sessions
    - Option 1: Implement security (including encryption of user passwords at rest) in a separate Users table in your PostgreSQL database and incorporate JWT generation and validation logic
    - Option 2: Implement security using KeyCloak
- Allow for querying across domains (tracks and artists) – enable management of each domain model type individually as well as management of the composed type (i.e., support CRUD functionality for artists and for tracks which have an associated artist)
- Take advantage of annotations to configure your application components and reduce boilerplate code

Tech Stack:

- Java
- Spring/Spring Boot
- PostgreSQL database – containerized or running locally
- JPA/Hibernate

- JWTs – using “roll your own” JWT generation/validation components or via KeyCloak

Deliverables:

- A functioning REST API built using Java and Spring/Spring Boot that correctly persists and provides access to data in a PostgreSQL database
- A suite of unit tests for all tiers of the application (including mocking where appropriate)
- “Slice” (or integration) tests for your controllers using testing facilities provided by Spring Boot
- A presentation on the completed application