Requirements

With the rising popularity of superhero movies, there has been a heightened awareness of superheroes in our midst. The frequency of superhero (and supervillain) sightings is increasing at an alarming rate. Given this development, the Hero Education and Relationship Organization (HERO) has asked our company to develop a database and data layer for their new superhero sightings web application.  
  
The system has the following requirements:

* It must keep track of all superhero/supervillain information.
  + Heroes have names, descriptions, and a superpower.
  + Heroes are affiliated with one or more superhero/supervillain organizations.
    - * Need ID
      * Many to many relationship
* It must keep track of all location information:
  + Locations have names, descriptions, address information, and latitude/longitude coordinates.
    - * ID = coordinates
* It must keep track of all superhero/supervillain organization information:
  + Organizations have names, descriptions, and address/contact information.
  + Organizations have members.
    - * NOT REQUIRED – ORGANISATION INFO NOT NECESSARILY NEEDED
      * Can do for portfolio purposes
* A user must be able to record a superhero/supervillain sighting for a particular location and date.
  + - * Front end – recording information
* The system must be able to report all of the superheroes sighted at a particular location.
  + - * Can just be postcode (unique)
      * May need info like city, etc.
* The system must be able to report all of the locations where a particular superhero has been seen.
* The system must be able to report all sightings (hero and location) for a particular date.
* The system must be able to report all of the members of a particular organization.
* The system must be able to report all of the organizations a particular superhero/villain belongs to.

Need to develop a full stack application that allows for superhero/ villain sightings

Deliverables

To complete this assignment, you must deliver the following items:

* An entity relationship diagram
  + You may use MySQL Workbench to create a diagram, or use your choice of alternative tools such as Pencil, Draw.IO, or LucidChart.
  + The database must achieve **2nd normal form** at minimum.
    - * Primary key
      * No multiple values in fields
      * Unrelated info to the pk is extracted to its own table
  + Proper naming conventions should be used.
  + The ERD should be very easy to read, with all components clearly labeled.
  + Use a common and appropriate file format, such as a png or jpg image or a PDF document.
* A database creation script
  + The script should **create the database** with all tables, columns, and relationships.
  + Make reasonable assumptions about column data types; be prepared to justify your decisions.
  + The script should be re-runnable. This means it should drop the database and all objects if they exist and recreate them. You should be able to execute the script many times in a row without error. See the scripts provided for the databases used in the Relational Database unit for examples.
* DAO Implementation and Unit Tests
  + DAO should have an interface and an implementation.
    - * Java interface
  + DAO and DTOs must fully represent all data and relationships contained in the database design.
  + Implementation must make proper use of transactions.
    - * @transaction ?
  + Unit tests must fully test all create, read, update, and delete functionality for all entities and test all many-to-many and one-to-many relationships in the database.
    - * Do simple one or two tests

Requirements

The Hero Education and Relationship Organization (HERO) is so impressed with the work we did on the Superhero Sightings Data Layer project that they have awarded us a contract to build a Spring Boot web application.  
  
The system has the following requirements:

1. It must have a screen(s) to create, view, edit, and delete superheroes/supervillains in the system.
2. It must have a screen(s) to create, view, edit, and delete superpowers in the system.

* Not required

1. It must have a screen(s) to create, view, edit, and delete locations in the system.
2. It must have a screen(s) to create, view, edit, and delete superhero/supervillain organizations in the system.

* Not required

1. It must have a screen(s) to create, view, edit, and delete superhero/supervillain sighting (superhero/supervillain, location, and time) in the system.
2. It must have a home page that displays general information about the application, navigation to all the other pages, and a newsfeed of the latest 10 sightings in the database.
3. CHALLENGE 1: Allow users to upload a picture for each superhero/supervillain and then display the picture when the sighting(s) of that superhero/supervillain is displayed.

* Challenge = Not Expected – but you can do it if you want to

1. CHALLENGE 2: Incorporate Google Maps into the home/landing page and show the location of each of the sightings in the sightings news feed (this is a stretch goal - completely on your own. We have no material about incorporating Google maps into your application).

* Challenge = Not Expected – but you can do it if you want to
* Not too bad – google maps can be embedded within a website

Deliverables

To complete this assignment, you must deliver the following items:

1. Wireframes for all pages in the web app:
   1. You may use tools such as Balsamiq, Pencil, Draw.IO, or LucidChart.
   2. Include notes on endpoints, path variables / query string / form parameters as appropriate.

* INCLUDE WIREFRAMES
* Just basic structure of the pages on the website – no formatting UI elements
* Version 1.0

1. Spring Boot Web App Implementation:
   1. The web application must be built following the MVC patterns presented in the course.
   2. Implementation must make proper use of dependency injection.
   3. The web application must fully integrate the Superhero Sighting Data Layer

# Tips

* Use the walkthrough and make notes
* Do small bits at a time
* Take your time
* Work together <3
* Don’t worry if there are parts that are left out
  + But there will always be one bad assignment
* This assignment requires a plan

|  |  |
| --- | --- |
| **ERD - Second Normal Form: The database design meets AT LEAST the requirements for second normal form.** | Done |
| **ERD - Primary Keys: Each table has an appropriate primary key named either id or entityId or an appropriate composite key with all fields named correctly.** | Done |
| **ERD - Relationships: The relationships between the tables are expressed correctly with foreign keys on entity tables for one-to-many relationships and bridge tables for many-to-many relationships.** | Dome |
| **ERD - Data Columns: Each field has reasonable types and scales.** | Done |
| **Script Re-runnable: The script used to create the database can be re-run as needed, including dropping the database, rebuilding it, and USE the database.** | Done |
| **Script - Fits ERD Specification: The script creates tables named as presented in the ERD, with columns specified according to the ERD requirements, all foreign key relationships present, and auto-incrementing where appropriate.** | Done |
| **DTOs - Full Entity Modeling: All database entities have equivalent class representations with appropriate fields, getters, and setters.** | Done |
| **DAOs - Separation of Concerns: Each entity has its own DAO.** | Done |
| **DAOs - Interface Extraction: Each DAO implements a method that defines all of its public methods.** | Done |
| **DAOs - Full CRUD Operations: Each DAO implements add, getById, getAll, edit, and delete methods.** | Done |
| **Unit Testing: Golden Path Testing Unit tests verify that all public dao methods perform their operations as expected given valid inputs, including validation persistence checks that do not trust the return value of the method being tested.** | Done |
| **Unit Testing -Bad Input Testing: Unit tests verify that all public dao methods produce the correct exception types on invalid inputs, including invalid ids and nulls.** | done |
| **Code Style: All code is written in a readable, conventions-compliant manner.** | Done |