**Backend service for mood sensing app**

Start Date: Nov 8, 2022

End Date: Nov 9, 2022

Author: Iltae Lee

**Requirements**

* Visual Studio 2022
* SQL server

**How to deploy it locally**

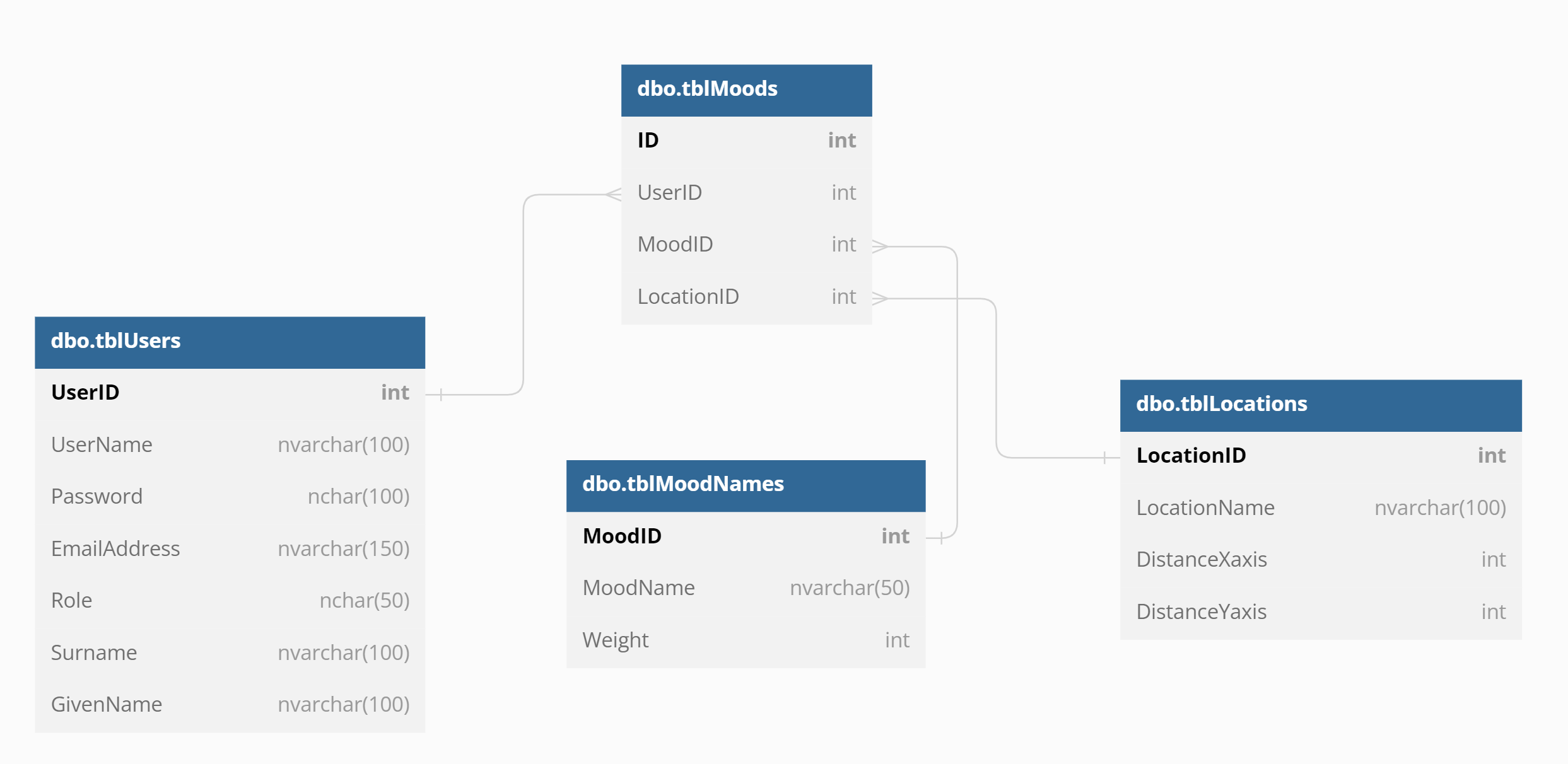
1. Run “MoodDB Schema and Data.sql” using SQL Server Management Studio. Alternatively, you can restore “MoodDB Backup” file. NUnit test cases use these data for some of the test cases.  
   
2. Run “Mood.sln” solution file using Visual Studio 2022(It must be version 2022 because it uses .Net Core 6).
3. Visual Studio 2022 should be able to run the project.
4. If there is a DB connection error:
   1. Open “Mood” project, open appsettings.json file and check “ConnectionStrings”.
   2. Open “MoodNunitTest” project, open “MoodNunitTest.cs” and check DB connection string in Setup method.
5. “Mood.postman\_collection.json” is the collection I used to test the APIs.   
     
     
   Alternatively, this is the link to the collection.  
   <https://solanio.postman.co/workspace/My-Workspace~f26d2953-61d0-4cab-8d90-0e55ec17c12b/collection/18130847-d7351a6a-6c5a-4028-8284-99445f8afcb7?action=share&creator=18130847>

**Design**

• APIs

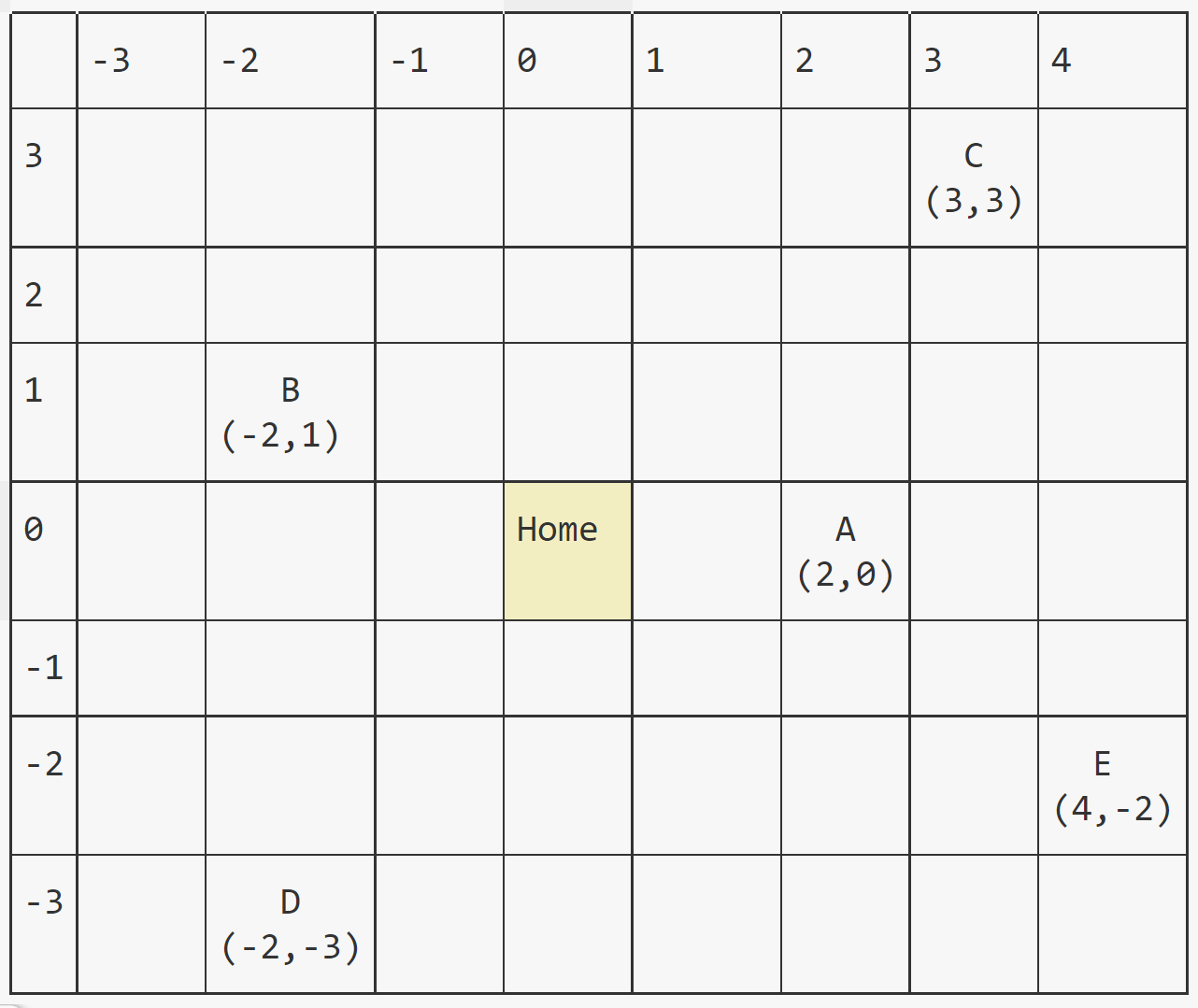
* **GET /api/Mood/HeartBeat**  
  : Returns “Service running” if the service is running.
* **POST /api/Login**  
  : Returns a JWT token if a user is successfully authenticated.
* **POST /api/Mood/PostMethod**  
  : Upload a mood capture for a given user and location.
* **GET /api/Mood/GetMoodFrequency**  
  : Return the mood frequency distribution for a given user
* **Get /api/Mood/GetClosestHappyLocation**  
  : Given the user’s current location, return the closest location where the user has been happy.

• DB Schema



This DB Schema design can also be found at <https://dbdiagram.io/d/636a064fc9abfc611170ff76>

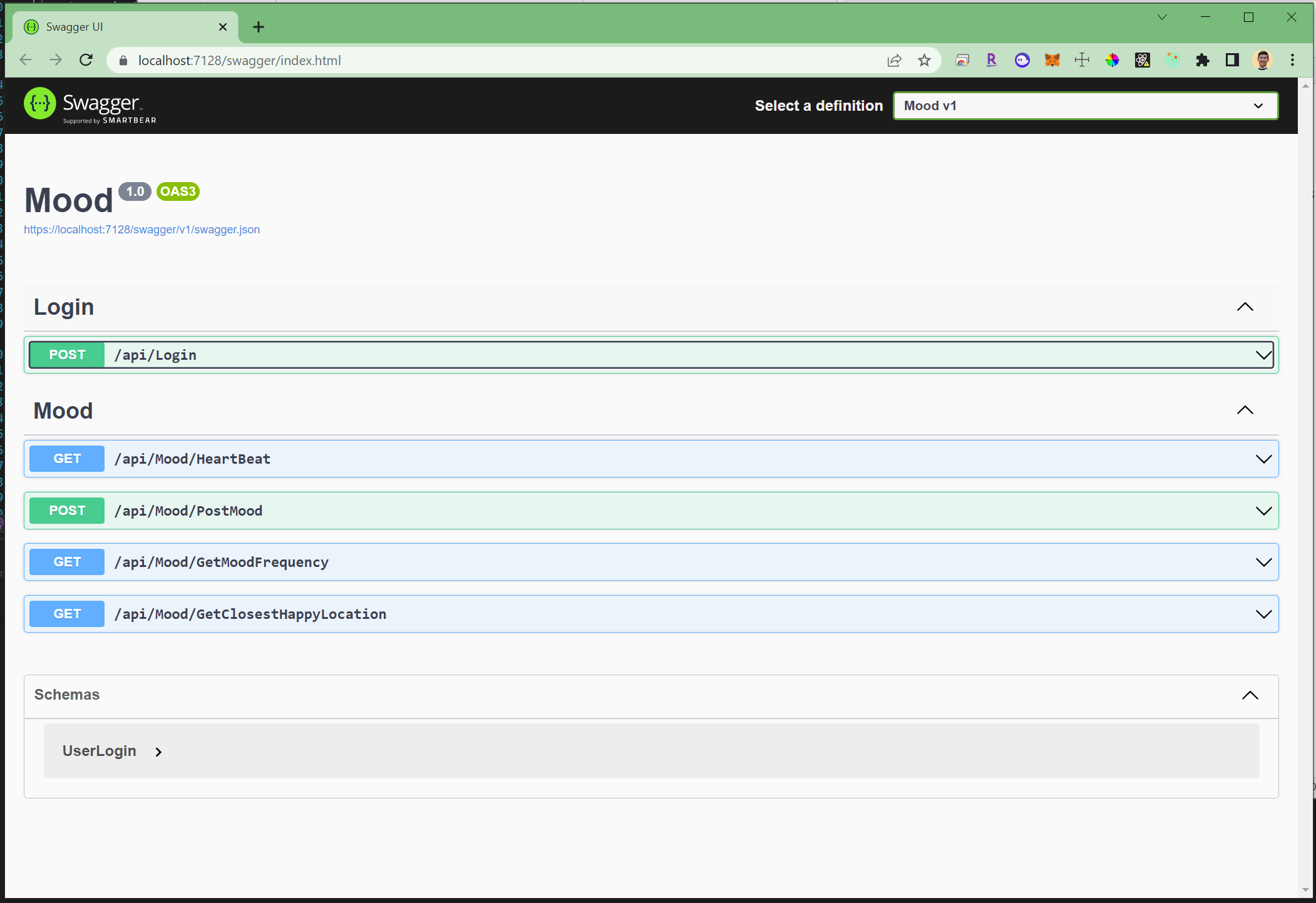
**Assumptions**

* In the following diagram example, there are 5 locations, A, B, C, D and E. First number indicates X axis value and the second number indicates the Y axis value. For example, location B is -2 away on X axis and 1 away on Y axis from home. On this 2-dimensional space, one can only move left, right, up or down. One can't move diagonally. tblLocations contains location data(DistanceXaxis and DistanceYaxis) for the 5 locations in relation to home or the starting point. This data is used to calculate the distance from any given location.   
    
  
* If distance from a location is the same for multiple locations, *GetClosestHappyLocation* returns the location with higher happy score. For example, let’s say we want to find the closest happy location from A where the user is currently at. A to C is 4 because (3-2)+(3-0). A to B is 5. A to D is 7 and A to E is 4. The distance between A to C and A to E are the same. In this case, we will calculate happy score for the user for both locations. This API returns C because the happy score for C is 4 while the happy score for E is 3 for the same user.
* *GetClosestHappyLocation* does not necessary return the location the user was most happy. For example, let’s say user 1 was most happy was at location A but was less happy at location B. Let’s say the user is at location C. Location B is closer to location C than location A. Then, the API returns location B.
* *GetMoodFrequency* returns frequency(count) data for all locations. If this user has never visited the location, the count will be 0.
* *tblMoods.Weight* value for Happy is 1. -1 for Sad and 0 for Neutral.
* *PostMood* API accepts UserID, MoodID, LocationID.

**Implementations**

* MS .NET Core 6 API project was used to implement these APIs in C#.
* MS SQL, version Microsoft SQL Server 2019 (RTM) - 15.0.2000.5 was used to store initial sample data.
* JWT generated using Hmac-Sha256 is used to authorize a request.
* NUnit was used to test business logic.
* NUnit test cases call the actual API’s method in the controller. Some NUnit test cases connect to SQL and use the data in SQL to validate test cases.
* “Administrator” role is required to invoke PostMood, GetMoodFrequency and GetClosestHappyLocation API. “UserA” in tblUsers is the only user with the “Administrator” role. Therefore, only UserA’s JWT can be used to invoke those APIs. Any other user’s JWT will throw 403 Forbidden error. Any invalid JWT will throw 401 Unauthorized error.

**Screenshot**



**Check List**

• Define API: Done

• Create dev project: Done

• Layout code structure: Done

• Design data model and key data structures: Done

• Define data persistence using any data store of your choice.: Done using MS SQL

• Define Implementation of operations: Done

• Input validation: Done

• Authentication: Done

• Authorization: Done using JWT.

• Unit test: Done using NUnit.

• Last but not least, provide a README for your design, implementation and assumptions.: Done. This file.