**Epic**

**1. User Endpoint Management**

**TODO Description**

Within this endpoint we will have the following functionalities:

* Pull User information
* Store User information
* Create User table
* Update User information

**2. Database Management**

**TODO Description**

This epic is going to be responsible on handling all database structure management and changes

* ApplicationDbContext (Separate ClassLib)
* UnitOfWork for the DbContext
* Repositories
* Wiring the AppDbContext to the startup class

**3.Controllers Optimization**

This epic will be responsible for handling all tasks for managing and optimizing controllers

**4. Authentication and Authorization**

Building the Authentication and Authorization functionalities for our web application.  
we are going to make this application completely stateless.

**5. Health Record**

Design the data structure for health inside our application and implement those changes

**6. User Profile**

Manage User Profile

- Create the update functionality for updating the user profile

- Update the user profile with more information

- Load user profile information

**7. Project Maintenance**

This epic will include all of the tickets whic will be responsible to maintaining the application in the latest framewors and all of the updates.

**Issues**

**1. Create a DbContext ClassLib**

Create a DbContext ClassLib which will be responsible on all db related tasks.

* Name: SohatNotebook.DataService
* Packages: All EfCore and Sqlite packages

Add DataService to the Api as reference

**2. Add User Entity**

- First Name

- Last Name

- Email

- Phone

- Date of Birth

- Country

**3.Create a Users Controller**

Create 3 Endpoints

**- Get : Get user information based on id of type guid**

\* Return a user dto after it has successfully got the user info

\* Return status is 200

**- Post : Add new user: it will take a dto of user**

\* Return a user dto after it has successfully got the user info

\* Return status is 201

**- Get All users: it does not take any params**

\* Return a list of user dto

\* Return status is 200

**4. Implement Unit Of Work Pattern and Repository Pattern in the Application**

1. We need to create the IRepository Interface

2. Create a Generic Repository

3. Create the IUnitOfWork Interface

4. Create the UnitOfWork Class

5. Add dependency injection in the startup class for the UoW

6. Connect the Controller to the UoW

7. Create a Repository for Users

8. Update the UsersController to use the UoW instead of direct context initialization

**5. Implement BaseController**

In the Controllers folder, we need to implement a base controller, that all of the controllers will inherit from.

This base Controller will be responsible for handling the UoW as well as any required dependency.

**6.Add Controller Versioning**

We will need to implement controller versioning which will be in the form of

domain.com/api/v1/controller

the versioning should be inside the base controller

**7.Add Authentication and Authorization to the API**

Enable JWT token Authentication into the API

Create AccountManager controller, this controller will have a login, register functionalities,

A JWT token needs to be returned to the user one they register, and a jwt token need to be sent back to the user once the login.

The Registration Form will be based on the following:

- First Name

- Last Name

- Email

- Password

The login Page should have the following

- Email

- Password

Implement an expiry date for the JWT token which is generated

**8.Add Automapper to the application**

Add automapper to the application to avoid manually mapping items per request.

**9. Send email to confirm user email address**

Send an email to the user, so they can confirm their email address,

- we need to add an endpoint, which confirm the email address,

- update the user registration and to update the verification status of emails

**10. Add Identity field to the user table**

Add a migration script to add the identity id to the user table

**11. Add Refresh token functionality**

Add the ability to have refresh tokens inside our response once the user login/register.

Update the following

\* Update the login action

\* Update the register action

\* Create a new method in the AccountsController which is responsible for the refreshing of the token without the user login in

\* Update the DTOs

\* Create the Db structure

\* Update the unit of work to give for this change

**12. Design the structure of our health record**

Create the entity structure for our health records

\* Create the entity

\* Update the database

\* Update Unit of Work

The new Class will contain the following

* Height
* Weight
* Blood Type
* Race
* Glasses

**13.Create Controller for Health record**

TODO:

**14. Update the User Profile with more information**

Add the following information to the entity:

Address

Mobile Number

Sex

Add the migration to update the database as well as need to update the unit of work, to contain a method which will be responsible for updating these information

**15. Load User profile information**

The logged in user will have the ability to load their profile information

Based on the logged-in user if we will be able to see the user information and load their profile.

Create an action inside the controller to reflect this,

we need to utilize JWT token, in order to know which user is logged in

**16. Create the update functionality for updating the user profile**

Create an action inside the controller to update the user profile and to make sure it contains all of the required information.

**17. Upgrade the application to .Net 6**

Update the main API application as well the classlib to utilise .Net 6

- Upgrading the packages

- Upgrade the references

- Upgrade the namespace

- Utilise the new all in once Program.cs

**18. Create a Unify Return for all end points – [Project Maintenance]**

Create 2 classes to cater for all the return possibilities from the Api.

\* Result class - which should be used for every single return in the Api.

\* PagedResult class - which should be used for every list return in the Api.

\* Both classes should be generic

**19. Create a Generic Error Message class which will contain all of the error messages – [Project Maintenance]**

Create a static which contain other static class.

this class will be responsible for handling error messages.

NOTE: Think about translations