Documentation

v 1.0

CHECKARR

Contents

[1.0 Database and Data Models 1](#_Toc528442232)

[How to access these? 2](#_Toc528442233)

[1.1- Database Tables 2](#_Toc528442234)

[2.0 Controller classes in webapi: 4](#_Toc528442235)

[1. AuthController: 5](#_Toc528442236)

# Database and Data Models

MySQL is the platform used for the database with the schema named as ‘**checkarr’**



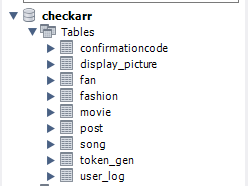
All the tables are within this schema, and all the data regarding the project gets stored here.

To link this database with code for efficiency and ease of use we have used ‘**Entity Framework’** in between database and web-api.

Entity Framework converts all these tables into classes to be used in coding by simply creating object of the required table i.e. direct mapping of tables into classes.

1. **Tables in Database:**

The following are the tables in MySQL:

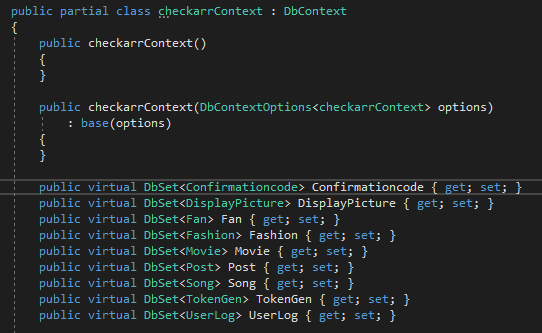


## How to access these?

The above tables and all their attributes, and values can be accessed through Entity Framework by creating a DbContext by the name checkarrcontext



This context will have all the database tables as follows with getters and setters:

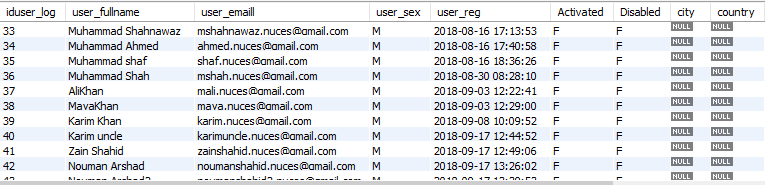


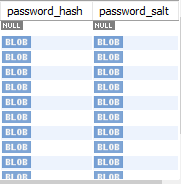
## Database Tables

The following are the database tables, all the attributes are given in the screenshots attached to each:

1. **UserLog**:

This table will have information related to user like name, fullname, etc. Password is not directly stored in this for security reasons rather a password hash and password salt are stored in binary form which are then decrypted in web-api through MACSHA12() algorithms from library files. Activated and Disabled are Booleans for User Account Status





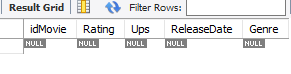
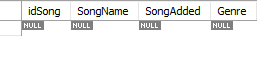
* First the original password given by the user is converted into password hash and password salt and stored in database, and later when it is required password\_hash and password\_salt are converted again into the actual password

1. **Token\_gen**:
   1. It is the JWT security token generated for each users that must be used for access or otherwise the user won’t be able to use the account for any kind of activity
   2. It also has an expiry time after which it is no longer of use



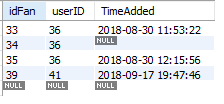
1. **Post:**
   1. This is a table for the posts that user will make
   2. It will be a post of some specific type of the available categories



1. **Movie:**
   1. This is a table for a post that is of movie type
2. **Song:**
   1. This is a table for a post that is of song type
3. **Fashion:**
   1. This is a table for a post that is of fashion type



1. **Fan:**
   1. This is a table for keeping track of the fans/followers. This table has a userID of the user which you are a follower of

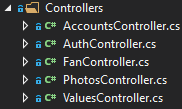


1. **Confirmation Code:**
   1. This is a table for codes that are sent right away to the user after the user registers and after confirming code the account is activated

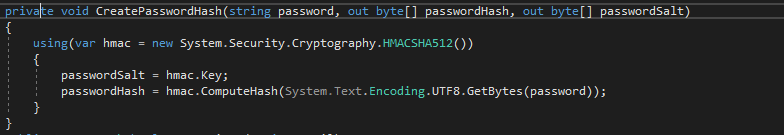


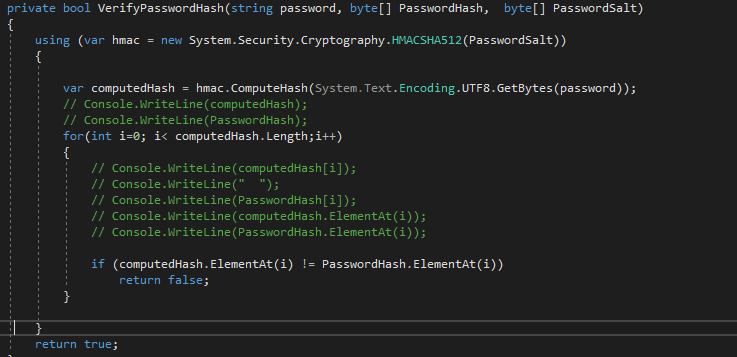
# Controller classes in webapi:

There are different controllers in api to deal with different webpages and different tasks associated with them. These are the present five controllers:



Before going to the main controllers, there are two main functions **CreatePasswordHash**() and **VerifyPasswordHash**() that are used for login and signup purposes to encode and decode password so it should be stored in database in binary format in the form of password hash and password salt so that the password cannot be seen or accessed directly through database rather it needs to be decrypted first by webapi after fetching password hash and password salts. This is done by using HMACSHA512() algorithm which is already present in math library files and there is no need to further look into it.





These two functions are going to be used as follows:

* When the user enters password for the first time during register, password is not directly stored rather **CreatePasswordHash**() is used and the password is converted into a password hash and password salt which are then stored in binary format in the database
* When the user enters the password, it is not present directly in the database instead password hash and password salt are present which are then fetched from database and **VerifyPasswordHash**() then combines hash and salt through HMACSHA512() again, and forms the original password

## AuthController:

This is a controller class for handling the following requests in form of http/get and http/post:

* Login requests
* Register requests
* Token refresh
* Token validate

1. **Register Request:**

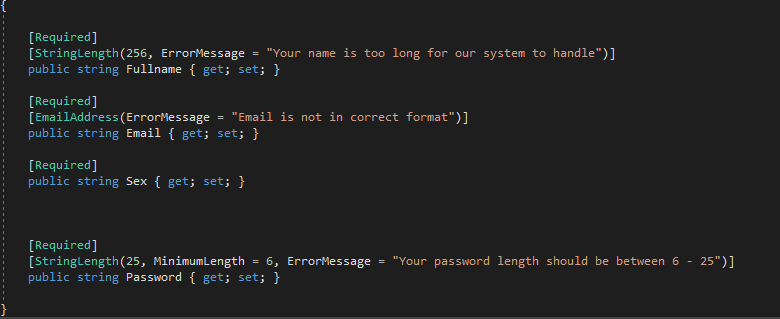


The above snap shows that it is a HttpPost request which will be generated when the user will click on Register after filling out all details.

**PASSED VALUES:**

\_userForRegisterDto in the parameter is a data transfer object that will basically have all the values that will be passed as the user entered

Fullname, Email, Sex, and Password must be sent in the JSON object sent from website or mobile application



 This shows that it is a required field and must be provided by the user

Rest are getters and setters for each of the attributes.

**WHAT SHOULD THIS Register Call RETURN?**

This will return a status code along with a message in different scenarios which are as follows:

**200, successfully created** for successful registration of user

**500, BAD REQUEST** if the api is not working or due to some other issues

Other error messages can be upon if the user already exists with the given user email.

1. **Login Request:**

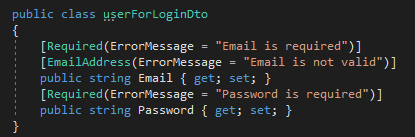


The above snap shows that it is a HttpPost request which will be generated when the user will click on Login button after filling out all details.

**PASSED VALUES:**

\_userForLoginDto in the parameter is a data transfer object that will basically have all the values that will be passed as the user entered

Email, and Password must be sent in the JSON object sent from website or mobile application



**WHAT SHOULD THIS Login Call RETURN?**

This will return a status code along with a message in different scenarios which are as follows:

**200, OK** for successful login of user

**500, BAD REQUEST** if the api is not working or due to some other issues

**401, NOT FOUND** if the user does not exist with the email entered

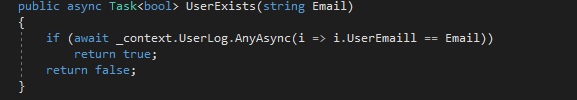
# Register (Detailed):

**Location File: Authcontroller.cs**



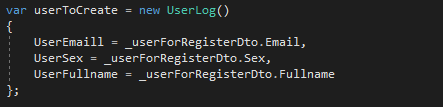
First of all, the \_userForRefisterDto.Email is one that is passed by the user on time of registration which is used and then checked whether the user already exists or not.

**Location File: Authrepository.cs**



If the user is not found then Badrequest is returned.

**Location File: Authcontroller.cs**

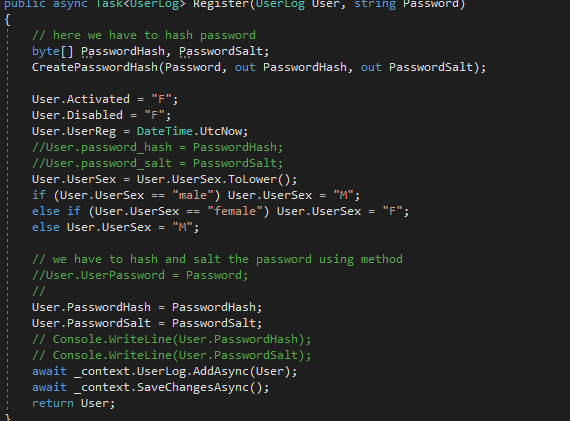


Next, all the passed values in Dto are stored in an object of type UserLog for further use



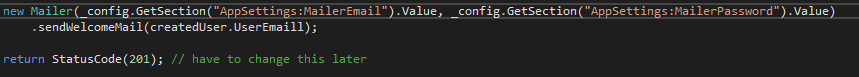
Here, Register() function is called which is present in Authrepository.cs file

**Location File: Authrepository.cs**



Here, password hash and password salt are computed as told above, and different attributes of the newly created user are set, and finally the user with all its attributes is saved in the database, and the same user is returned to the previous call.

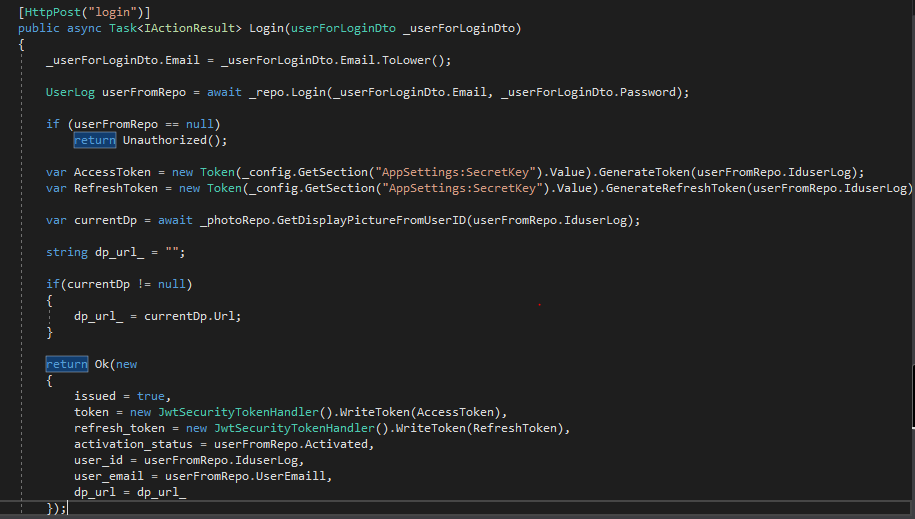
**Location File: Authcontroller.cs**



Finally, in the end a welcome email is sent to the email of new user, and a success status code of 201 is returned to http request.

# Login (Detailed):

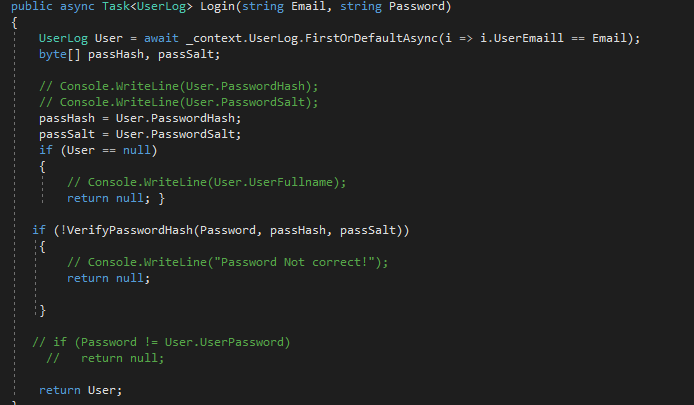
**Location File: Authcontroller.cs**





First, Login() function is called from Authrepository.cs

**Location File: Authrepository.cs**



Here, in this function first the database is searched for the given email, and the matched user is then picked, and the password hash and password salt of that user is fetched, and then using the VerifyPasswordHash() function it is matched with the password that the user entered, and if it matches then that user from database is returned.

After that in the main function of Login in Authcontroller.cs, following are done:

* Refresh token is generated
* Access token is generated
* Display picture is fetched for the found user