**Details of the deliverables:**

1. Github URL for the solution:

<https://github.com/vani-projects/VintriWebAPI>

1. Projects:
   1. VintriWebAPI – This is the main project with the implementations for Task 1, 2, and 3
   2. VintriWebAPI.UnitTests – This is the test project for Task 4
2. Main components of the VintriWebAPI project
   1. Models
      1. Beer.cs – It is a POCO object representing the Beer entity. It has a userRatings property in addition to id, name, and description properties which were selected based of the requirements for Task 2.
      2. BeerService.cs – This is a wrapper service to the Punk API service that provides the services as per this application’s requirements. Following 2 services are provided:
         1. BeerExists – Check if a Beer with given ID exists. Returns bool – true/false.
         2. GetBeerByName – Gets the details of a beer for a given name. If no beer with the given name exists, returns null.
      3. Database.cs – This object represents a database that will hold all the user ratings submitted as per Task 1
      4. UserRating.cs – This is a POCO object that represents a UserRating entity. User Rating will have a rating, the username of the user submitting the rating, the comments, and the Id of the beer for which the ratings
   2. Controllers
      1. BeerController.cs – This has two actions for Tasks 1 and 2
         1. AddUserRating – This performs Task 1 – Adding a new user rating for a beer and saving it in a local database.json file (Path hardcoded at this time)
         2. GetRatingsForBeer – This performs Task 2 – It finds beer by its name and also retrieves it’s user ratings from the database.json
   3. Filters
      1. ValidateUsername.cs – This performs Task 3 which is to validate that the username sent in the AddUserRating action is a valid email address using regex
3. Components of the VintriWebAPI.UnitTests project
   1. Test Classes
      1. DatabaseTest.cs – This is the corresponding unit test class for Database.cs file. The tests available include:
         1. TestAddUserRatingsFirstTime – This will test that a new user rating can be created for the very first time, when the database.json file is not existent.
         2. TestAddUserRatingSecondTime – This will test that a user rating can be appended to existing user ratings in the database.json file
         3. TestGetUserRatings – This will test if the user ratings can be read correctly from the database.json file
4. Task 1 - Add a REST API endpoint to allow a user to add a rating to a beer.

Here is the sample request for this endpoint:

curl --location --request POST 'https://localhost:5001/api/Beer/5' \

--header 'Content-Type: application/json' \

--data-raw '{

    "username":"vani@gmail.com",

    "rating":5,

    "comments":"Excellent"

}'

The service will return an HTTP 200 status to indicate the request was successful.

If the Beer Id sent is not valid, an HTTP 404 error will be sent

1. Task 2 - Add a REST API endpoint to retrieve a list of beers

Here is a sample request for this endpoint:

curl --location --request GET 'https://localhost:5001/api/Beer?beername=buzz' \

--header 'Content-Type: application/json' \

--data-raw ''

Here is the sample response for this call:

[

    {

        "id": 1,

        "name": "Buzz",

        "description": "A light, crisp and bitter IPA brewed with English and American hops. A small batch brewed only once.",

        "userRatings": [

            {

                "beerId": 1,

                "userName": "vani@gmail.com",

                "rating": 5,

                "comments": "Excellent"

            }

        ]

    }

]

If a beername that doesn’t exist is provided, we get an empty collection indicated no results were found

1. Task 3 - Add a custom Web API Action Filter Attribute

ValidateUsername.cs – An ActionFilter attribute that is decorated on the AddUserRating action on the BeerController to validate if the username is a valid email address

1. Task 4 – Add Unit Tests

DatabaseTest.cs – Created to test the functionality provided by Database.js model object