Homework 4

Requirements

Functional Requirements:

Following are the six functional requirements:

Below requirements are implemented in homework 2.

1) Add data into the table

Description: user should be able to add the data to the table.

Acceptance Criteria:

Actor: Users

Action: Add data into table.

Method: Save.

As a user I should be able to add menu data to the table.

URL: http://localhost:8089/Restaurant

Sample Request: <http://localhost:8089/Restaurant>

{

    "name":"hummus",

    "size":"M",

    "type":"APPITIZER",

    "price":15,

    "chefFavorite": true

}

Sample Response: Successfully added the record.

2)Update data in the table

Description: Users should be able to update the new information.

Acceptance Criteria:

Actor: Users

Action: Update the data in the table.

Method: Update.

As a user I should be able to update menu data to the table.

URL: http://localhost:8089/Restaurant/1

Sample Request: <http://localhost:8089/Restaurant/1>

{

        "name": "HummusWithChickPeas",

        "size": "M",

        "type": "SALAD",

        "price": 25

    }

3)Filter by Type

Description: users should be able to see the list of menus, which matches the user given type.

Acceptance Criteria:

Actor: Users

Action: Get the list of menus which matches the user given type.

Method: filterByType.

URL: http://localhost:8089/Restaurant/type

Sample Request: <http://localhost:8089/Restaurant/type>

Type:DESSERT

Sample Response:

successfully updated the record

Below requirements are implemented in current sprint.

4) Filter Menu by name

Description: users should be able to see the list of menus, which matches the user given name.

Acceptance Criteria:

Actor: Users

Action: Get the list of menus which matches the user given name.

Method: filterByMenu

As User I should be able to get the menu which matches the name given in input.

URL: http://localhost:8089/Restaurant/getByName/{name}

Sample Request: http://localhost:8089/Restaurant/getByName/hummus

Sample Response: [

    {

        "id": 1,

        "name": "hummus",

        "size": "M",

        "type": "APPITIZER",

        "price": 20,

        "chefFavorite": false

    },

    {

        "id": 18,

        "name": "hummus",

        "size": "S",

        "type": "APPITIZER",

        "price": 13,

        "chefFavorite": true

}

 ]

5) Filter Menu by price

Description: users should be able to get the list of menus, which has less or equal price mentioned.

Acceptance Criteria: As a User I should be able to get the menu which has less, or equal price mentioned.

Actor: Users

Action: Get the list of menus which are less than or equal to price given in the input.

Method: filterByPrice

URL: http://localhost:8089/Restaurant/filter/{price}

Sample Request: <http://localhost:8089/Restaurant/filter/20>

Sample Response:

[

    {

        "id": 1,

        "name": "hummus",

        "size": "M",

        "type": "APPITIZER",

        "price": 20,

        "chefFavorite": false

    },

    {

        "id": 2,

        "name": "zalok",

        "size": "S",

        "type": "APPITIZER",

        "price": 15,

        "chefFavorite": false

    },

    {

        "id": 4,

        "name": "burger",

        "size": "M",

        "type": "APPITIZER",

        "price": 20,

        "chefFavorite": false

}

]

6)Filter Menu By Chef’s favourite

Description: users should be able to see the list of menus, which is chef’s favourite.

Acceptance Criteria: As a User I should be able to get the menu which has Chef’s favourite menu.

Actor: Users

Action: Get the list of menus which are chef’s favourite.

Method: filterByChefFavorite

URL: http://localhost:8089/Restaurant/filter/chefFavorite

Sample Request: <http://localhost:8089/Restaurant/filter/chefFavorite>

Sample Respone: [

    {

        "id": 1,

        "name": "hummus",

        "size": "L",

        "type": "APPITIZER",

        "price": 25,

        "chefFavorite": true

    },

    {

        "id": 2,

        "name": "hummus",

        "size": "S",

        "type": "APPITIZER",

        "price": 13,

        "chefFavorite": true

    },

    {

        "id": 3,

        "name": "hummus",

        "size": "M",

        "type": "APPITIZER",

        "price": 15,

        "chefFavorite": true

    }

]

Non-Functional Requirements:

1)The API should implement authentication mechanisms (e.g., OAuth, API tokens) to ensure that only authorized users can perform CRUD operations.

Description: users should be able to give OAuth or API tokens while using postman, this gives access only to validated users to perform CRUD operations as a result giving security to the project.

2) Comprehensive testing, including unit tests and integration tests, should be part of the development process and up-to-date API documentation should be included.

Description: users should be able to write and test JUnit test cases. And up-to-date API documentation should be included in the code. This requirement makes user to analyse the correct functionality of methods as a result improving code quality.

3) Implementation of SQL Database.

Description: users should be able to perform crud operation and analysis like how they are doing on current h2 database in MySQL. By this we can achieve data persistence and advanced data capabilities. By implementing this functionality user data will be saved in database even after stopping the server, this makes maintaining the data easier.

Screenshots:

Eclipse Console output:

A computer screen shot of a computer screen

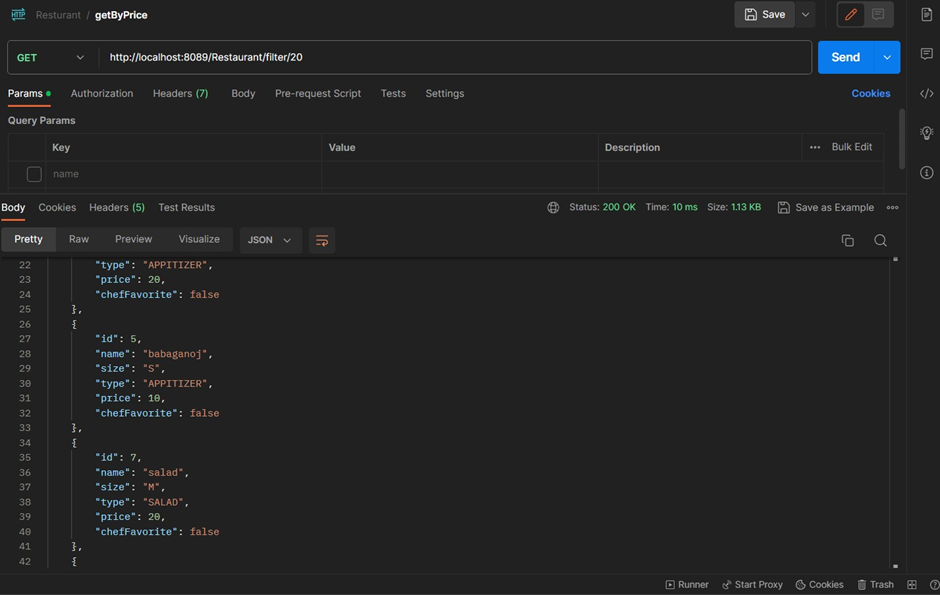
Description automatically generated

Postman Response for getByName:

A screenshot of a computer

Description automatically generated

Postman Response for filterByPrice:



Postman Response for filterByChefFavorite:

Success Response

A screenshot of a computer

Description automatically generated

Failure Response

