**Rest Architectural Constraints**

**Representational state transfer** (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations

Let's demystify what that means (hopefully you got the full form). REST is basically a set of rules for communication between a client and server. There are a few constraints on the definition of REST:

1. **Client-Server Architecture**: the user interface of the website/app should be separated from the data request/storage, so each part can be scaled individually.
2. **Statelessness**: the communication should have no client context stored on server. This means each request to the server should be made with all the required data and no assumptions should be made if the server has any data from previous requests.
3. **Layered system**: client should not be able to tell if it is communicating directly with the server or some intermediary. These intermediary servers (be it proxy or load balancers) allow for scalability and security of the underlying server.

**Some of the terms used in the heading:**

1. **REST Client**: code or an app that can access these REST services. You are using one right now! Yes, the browser can act as an uncontrolled REST client (the website handles the browser requests). The browser, for a long time, used an in-built function called XML Http Request for all REST requests. But, this was succeeded by FetchAPI, a modern, [promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)based approach to requests. Others examples are code libraries like axios, superagent and [got](https://github.com/sindresorhus/got) or some dedicated apps like [Postman](https://www.postman.com/).
2. **REST Service**: REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML.
3. **REST API**: this defines the endpoint and methods allowed to access/submit data to the server. We will talk about this in great detail below. Other alternatives to this are: GraphQL, JSON-Pure and Data.

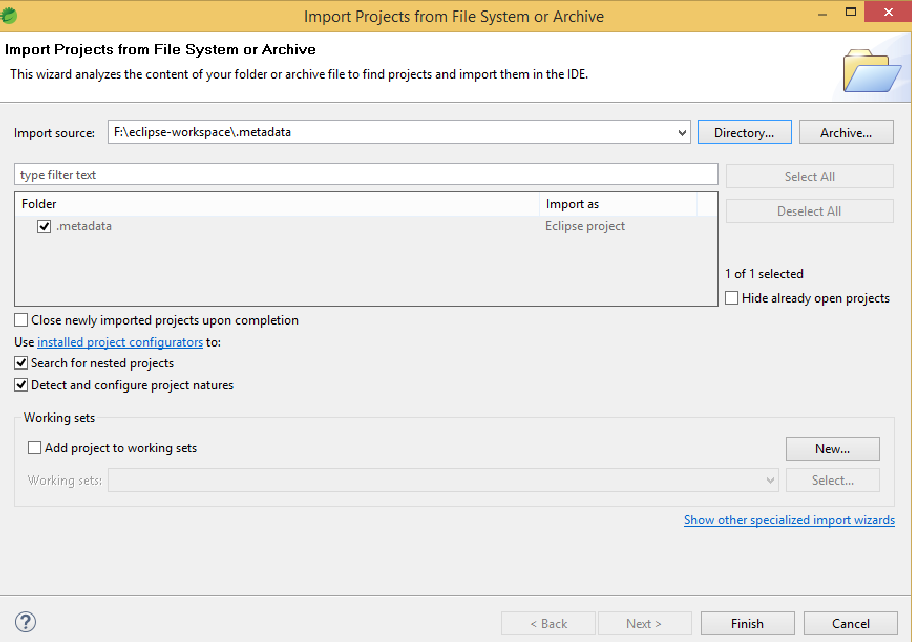
**How Rest Looks Like**  
In programming terms, there is an endpoint (a URL) that the server is waiting to get a request. We connect to that endpoint and send in some data about us (remember, REST is stateless, no data about the request is stored) and the server responds with the correct response.

**Web application which allows users to manage your favourite recipes**

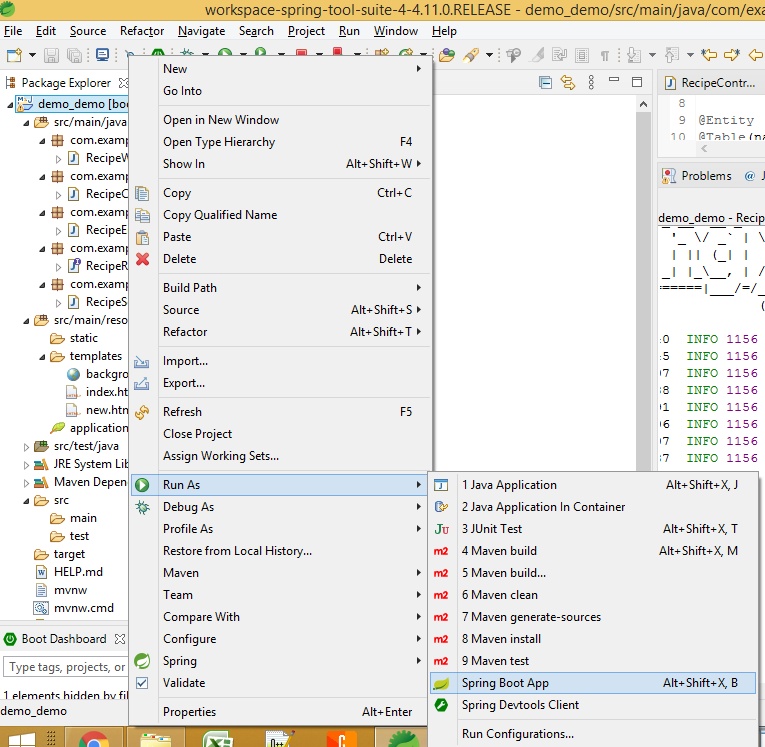
**How to run the application.**

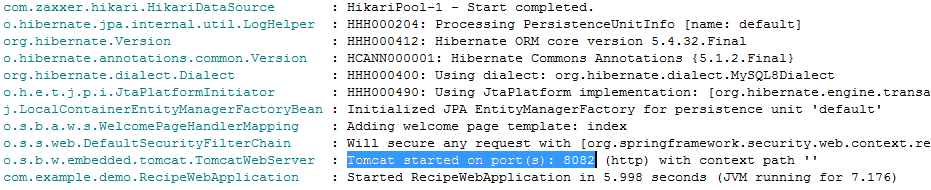
**Step 1 :** Download the code from Github and extract the zip file.

**Step 2 :** Import the project using File -> Open Project from File System and choose the project path using spring tool suite.



**Step 3 :** Run the application.



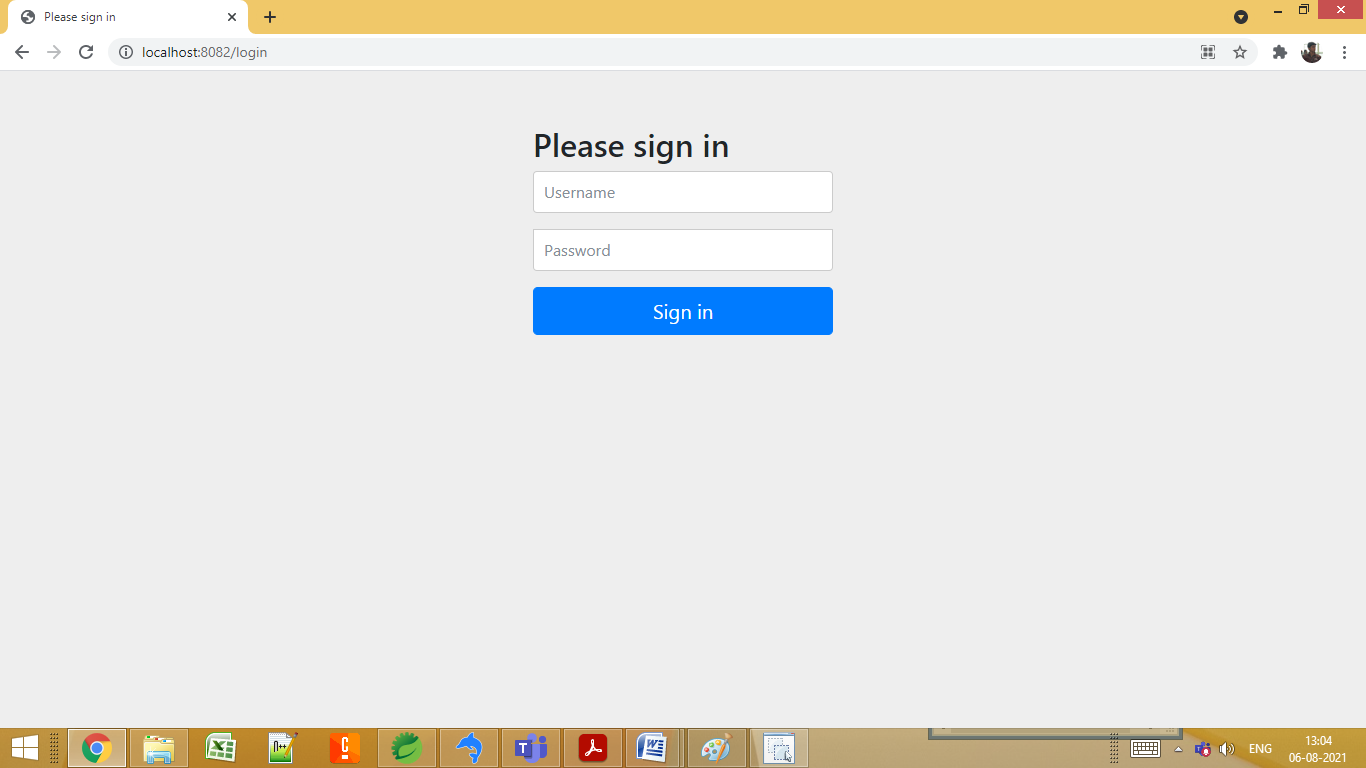


**Step 4 :** Open URL

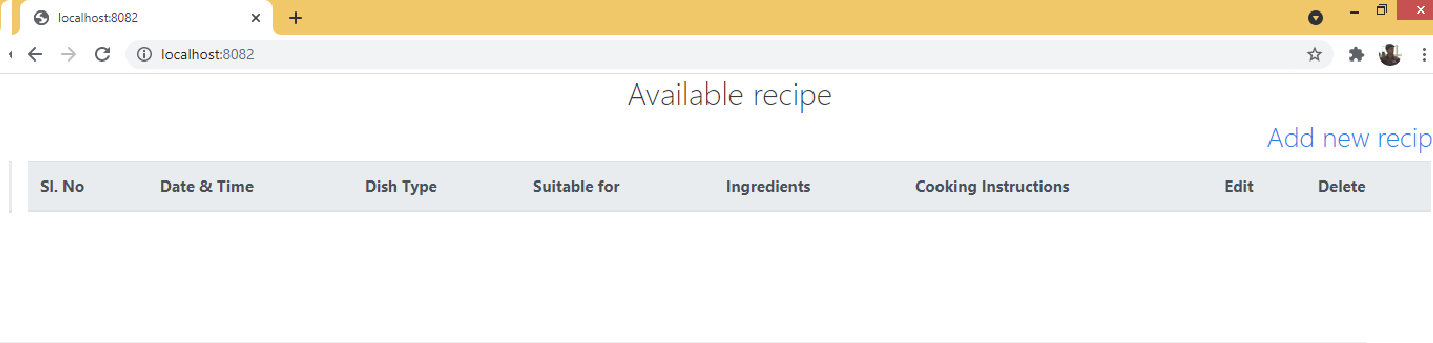
URL : http://localhost:8082/login

Authentication - Username - username

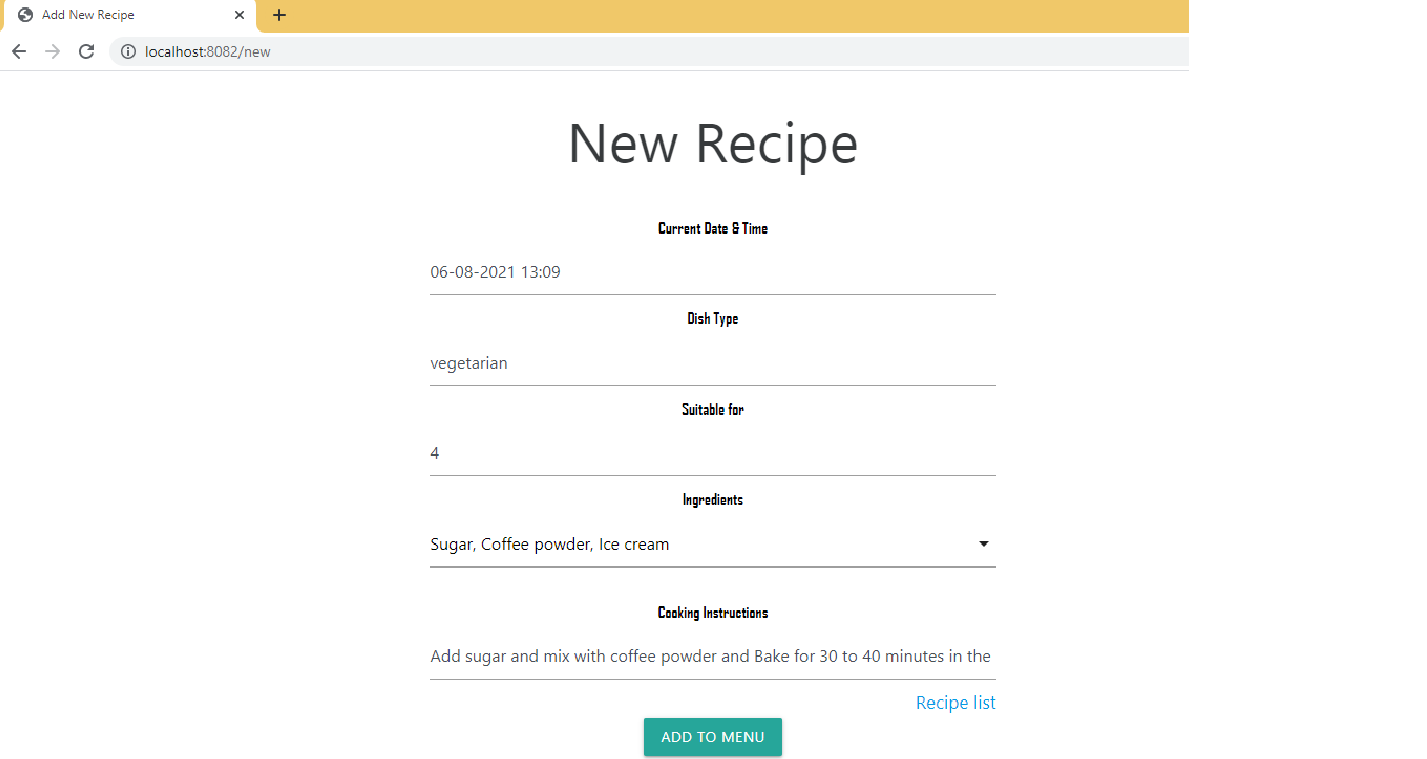
Password - password



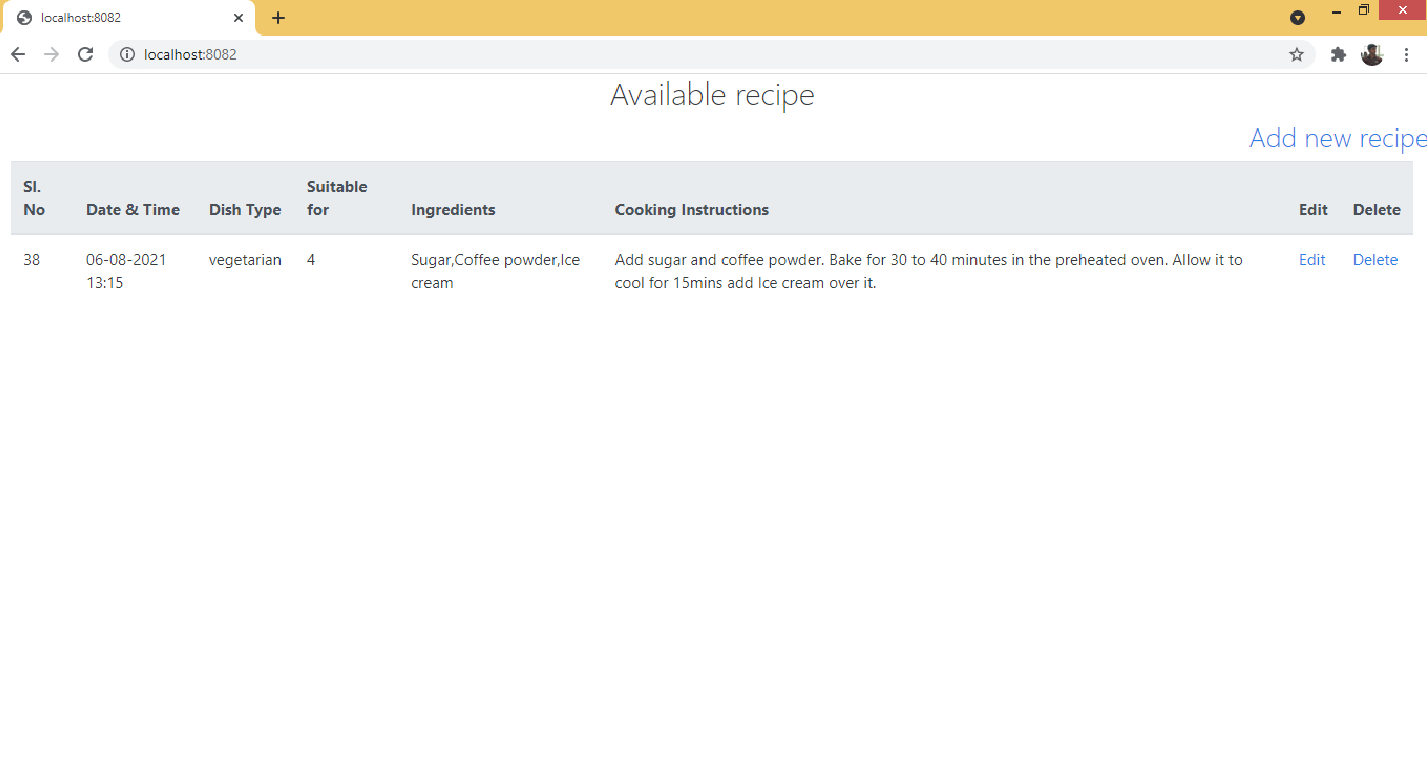
**Step 5 :** To add new recipe click "Add new recipe"



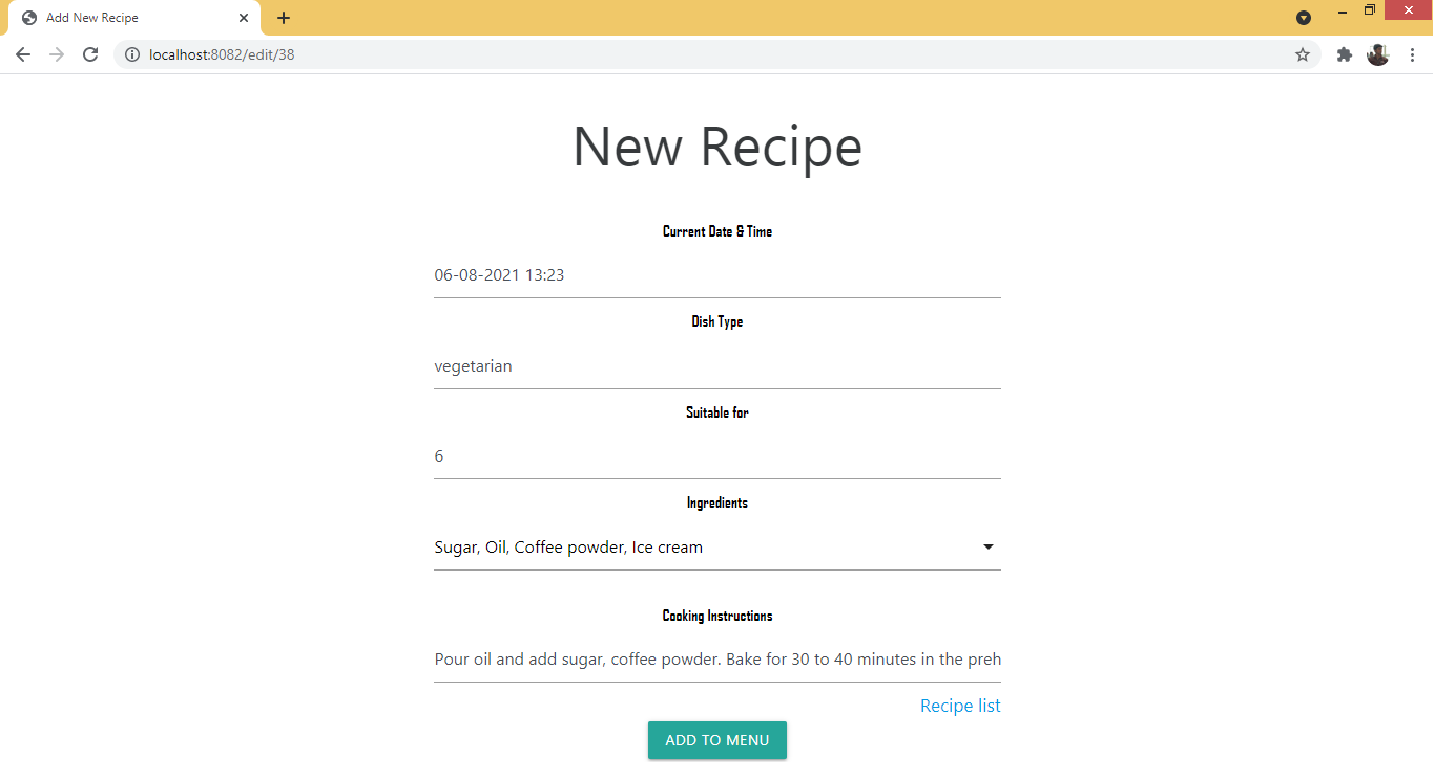
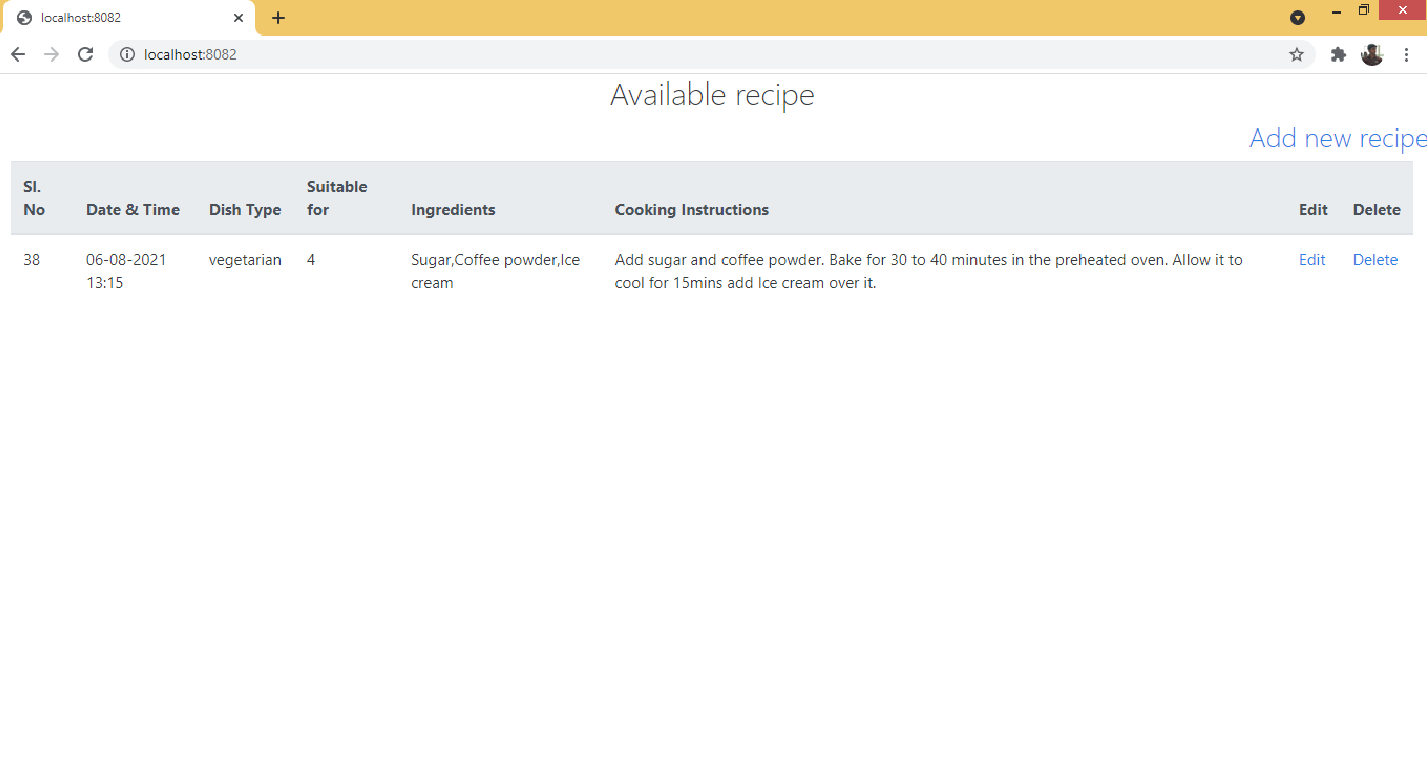
**Step 6 :** Adding a new recipe to menu.

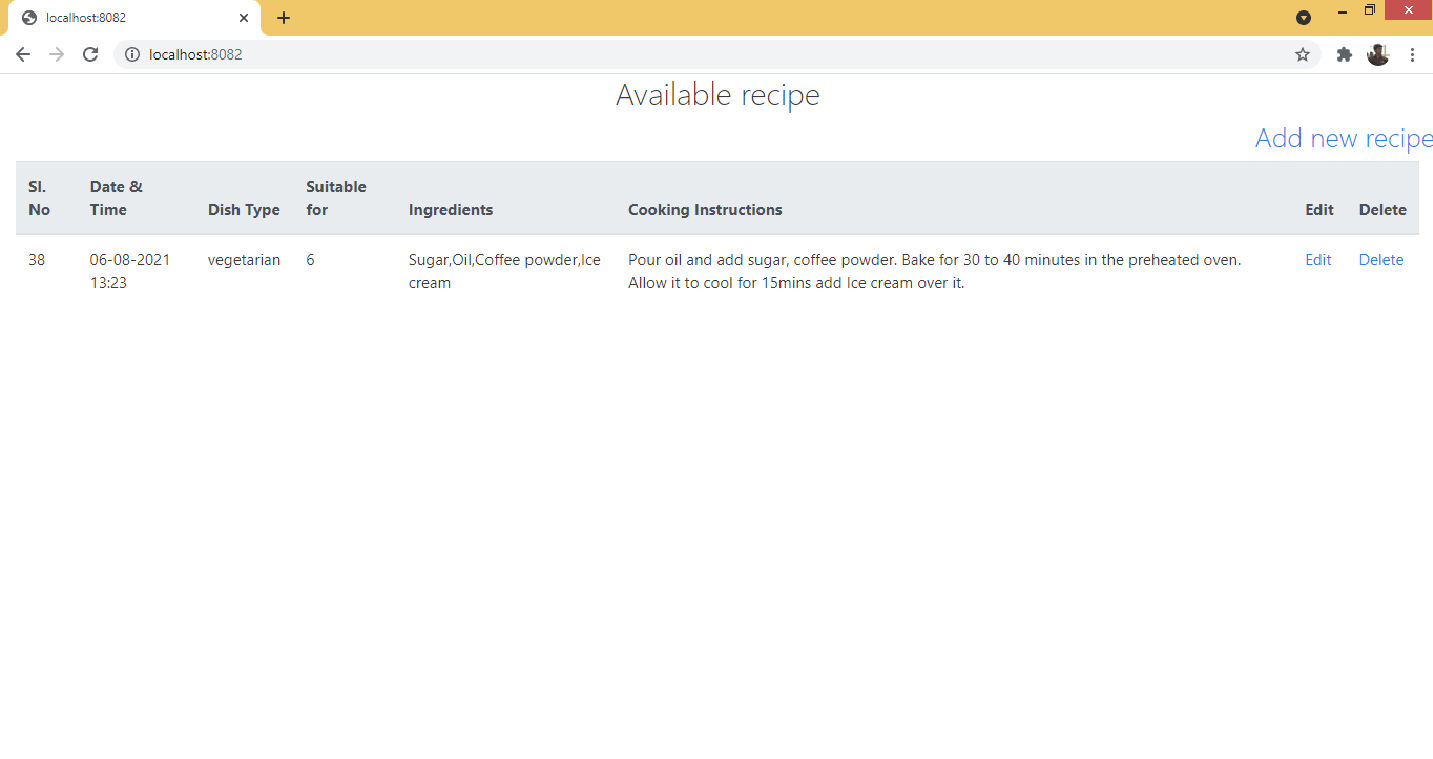


Click "Add to Menu" to add a new recipe. Once, new recipe is added it will redirect to "available recipe" page (shown below).



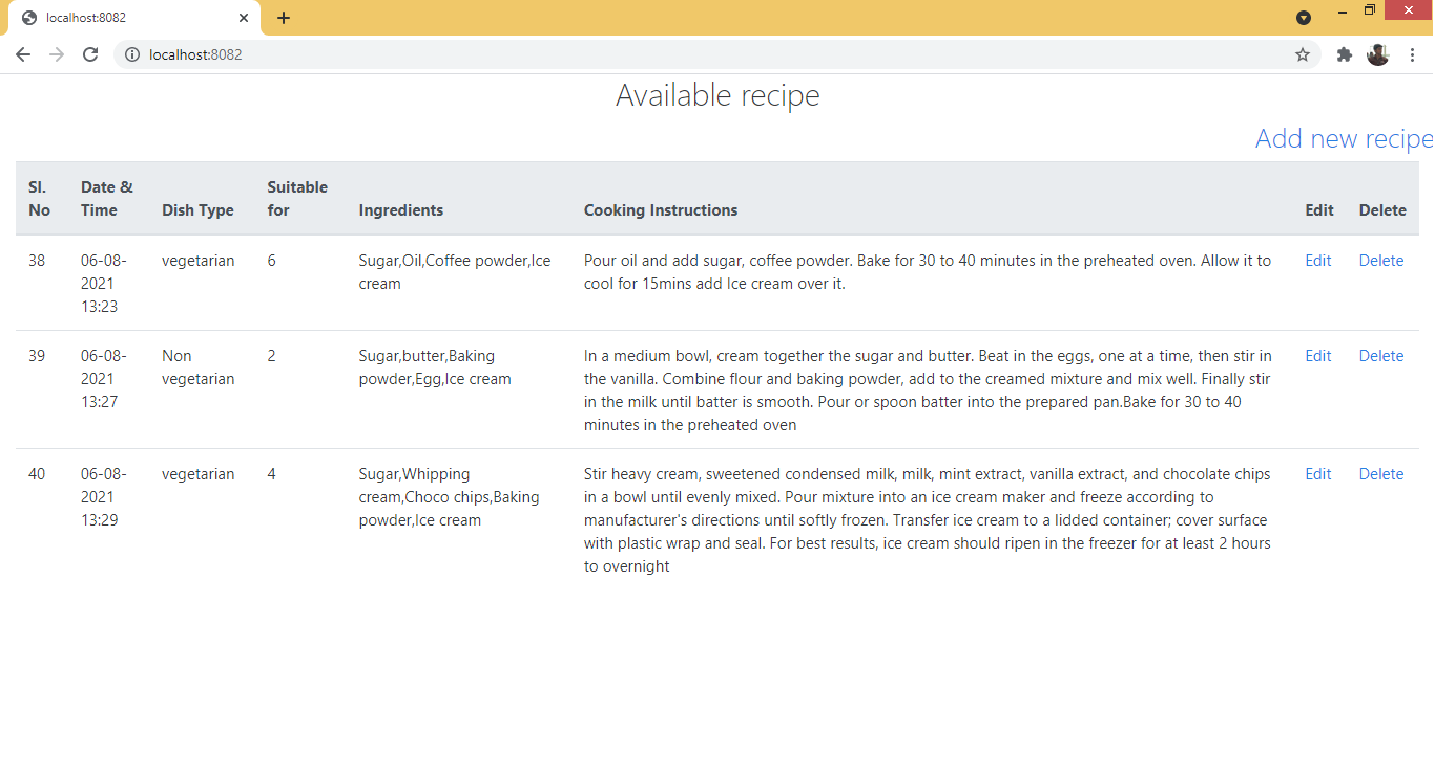
**Step 7 :** Edit a recipe. Click on "Edit" option on the right. and update the recipe.





**Step 7 :** Delete a recipe. Click on "Delete" option on the right.

Before deleting



After deleting. (Sl. No. 39 deleted)

