

Recipe Hub : A Simple Recipe Management System

Introduction

The Recipe Management System is web-based application designed to facilitate the user to manage and organise their recipes effectively. It allows users to add new recipes, view a list of existing recipes , and select and display details of individual recipes. Recipe Hub provides a convenient platform to store, search, and access your favorite recipes anytime, anywhere.

Technologies Used:

Frontend :

HTML for the Framing , CSS for styling , JavaScript is for functionality.

Backend :

Node.js with Express framework , MySQL for the database .

Frontend (HTML and JavaScript) :

1.Recipe Input Form :

A Recipe input form in a Recipe Management System is a user interface component where user can input and enter the details of a recipe.

2. Recipe List Section :

Dropdown menu for selecting a recipe from the list.This allows user to browse and select Recipes From a list .It is an efficient way of user to explore and choose recipies within the Recipe Management System

3. JavaScript Functions :

``fetchRecipeList``: To retrieve a list of recipies from a database or other data source

`displaySelectedRecipe`` : Displays details of the selected recipe from the list.

Backend(Node.js and Express) :

Install mysql workbench and Nodejs

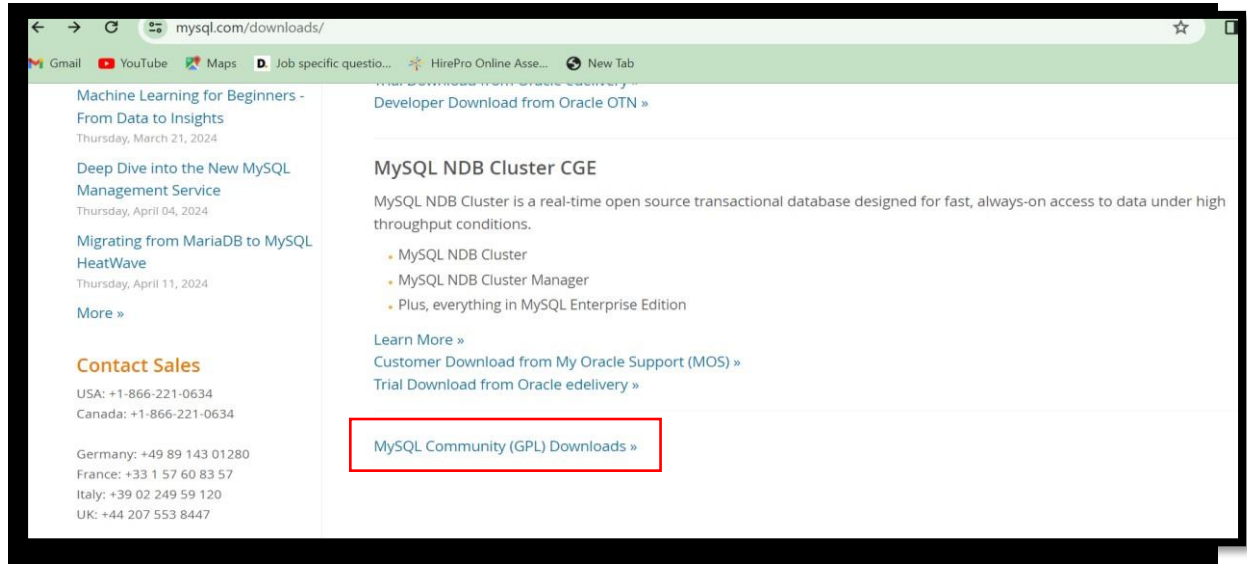
Install Mysql workbench

1.Open chrome/Google Follow

the link

<https://www.mysql.com/downloads/>

Now scroll down the page you will find



[MySQL Community \(GPL\) Downloads »](#)

Click on it

MySQL Community Downloads

- MySQL Yum Repository
- MySQL APT Repository
- MySQL SUSE Repository
- MySQL Community Server
- MySQL NDB Cluster
- MySQL Router
- MySQL Shell
- MySQL Operator
- MySQL NDB Operator
- MySQL Workbench
- MySQL Installer for Windows
- C API (libmysqlclient)
- Connector/C++
- Connector/J
- Connector/NET
- Connector/Node.js
- Connector/ODBC
- Connector/Python
- MySQL Native Driver for PHP
- MySQL Benchmark Tool
- Time zone description tables
- Download Archives

Here click on

- [MySQL Installer for Windows](#)

MySQL Installer 8.0.36

Note: MySQL 8.0 is the final series with MySQL Installer. As of MySQL 8.1, use a MySQL product's MSI or Zip archive for installation. MySQL Server 8.1 and higher also bundle MySQL Configurator, a tool that helps configure MySQL Server.

Select Version:
8.0.36

Select Operating System:
Microsoft Windows

Windows (x86, 32-bit), MSI Installer	8.0.36	2.1M	Download
<small>(mysql-installer-web-community-8.0.36.0.msi) MD5: 81061532541f716cf6c6e2c4881a154c Signature</small>			
Windows (x86, 32-bit), MSI Installer	8.0.36	285.3M	Download
<small>(mysql-installer-community-8.0.36.0.msi) MD5: d63232c190d0c9c294a2f8d776ed1c20 Signature</small>			

Click on download

MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

[Login »](#)
using my Oracle Web account

[Sign Up »](#)
for an Oracle Web account

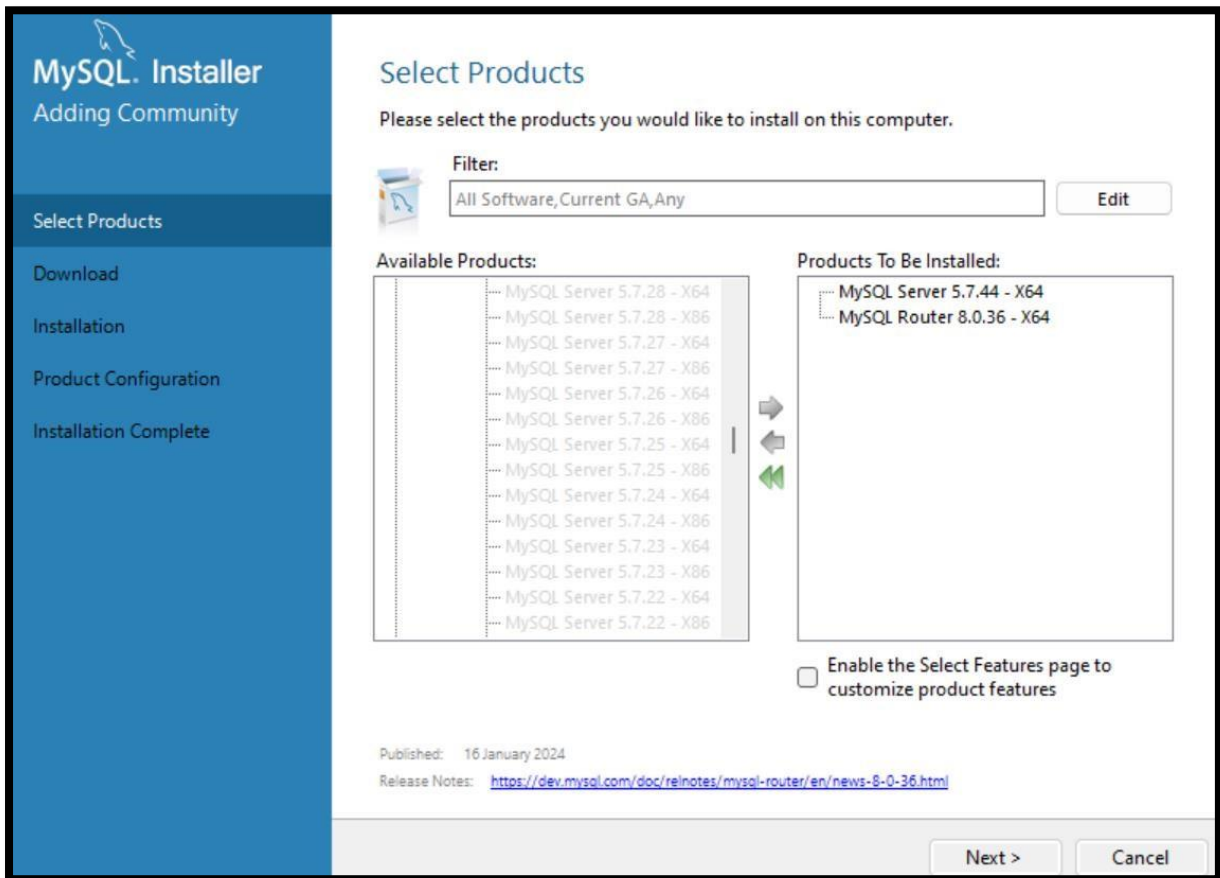
MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

[No thanks, just start my download.](#)

Click on it.

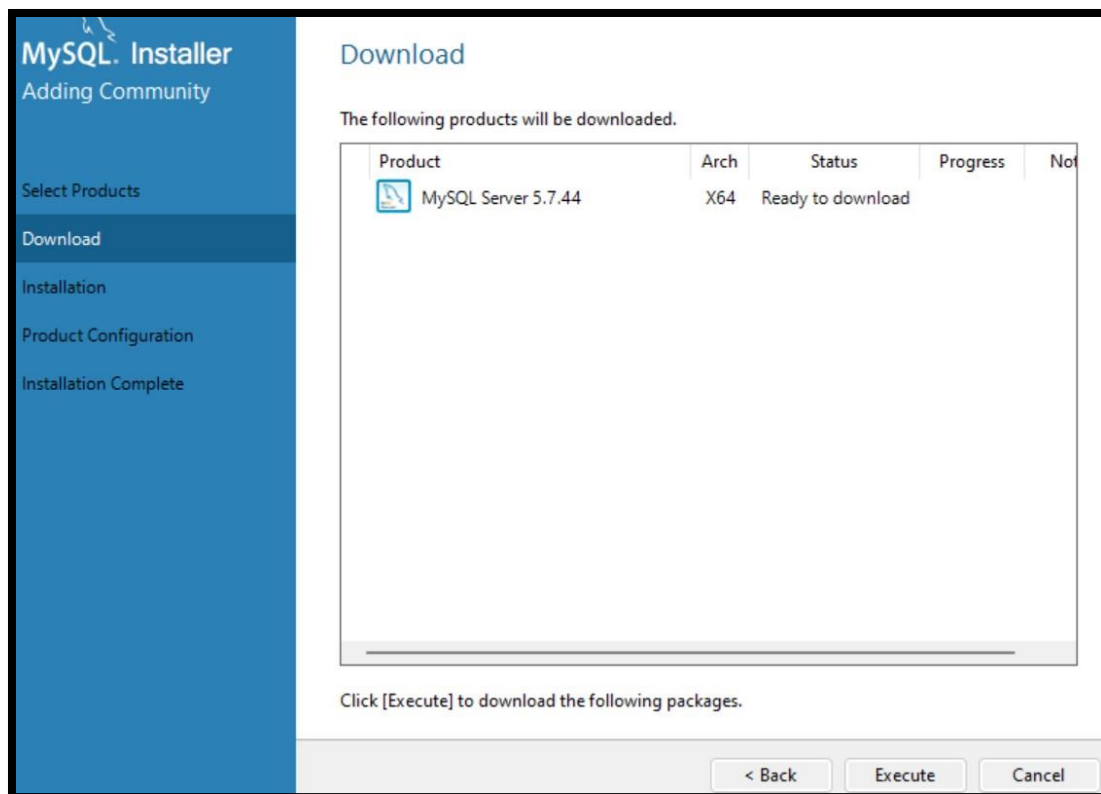
It will start downloading

Now open the Mysql installer now the setup



Click on next

The required products will be downloaded



Now after selecting click on execute.

It will start download.

It will download

mysql server

mysql workbench

mysql shell.

Router Configuration

MySQL. Installer
MySQL Router 8.0.36

MySQL Router Configuration

MySQL Router Configuration

☐ Bootstrap MySQL Router for use with InnoDB Cluster

This wizard can bootstrap MySQL Router to direct traffic between MySQL applications and InnoDB Cluster. Applications that connect to the router will be automatically directed to an available read/write or read-only member of the cluster.

The bootstrapping process requires a connection to InnoDB Cluster. In order to register the MySQL Router for monitoring, use the current Read/Write instance of the cluster.

Hostname:

Port:

Management User:

Password:

MySQL Router requires specification of a base port (between 80 and 65532). The first port is used for classic read/write connections. The other ports are computed sequentially after the first port. If any port is indicated to be in use, please change the base port.

Classic MySQL protocol connections to InnoDB Cluster:

Read/Write:

Read Only:

X Protocol connections to InnoDB Cluster:

Read/Write:

Read Only:

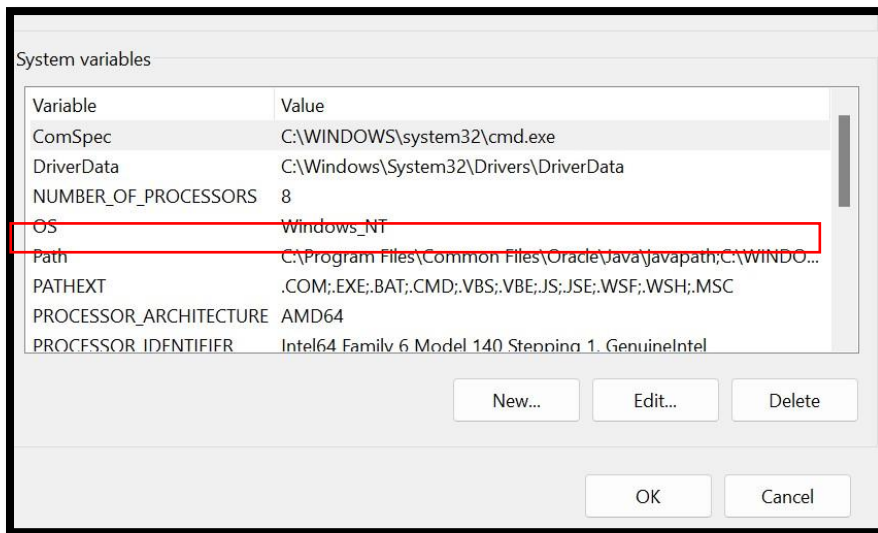
Here set the password and click on finish.

Now the Required Mysql workbench is downloaded.

Path setting:

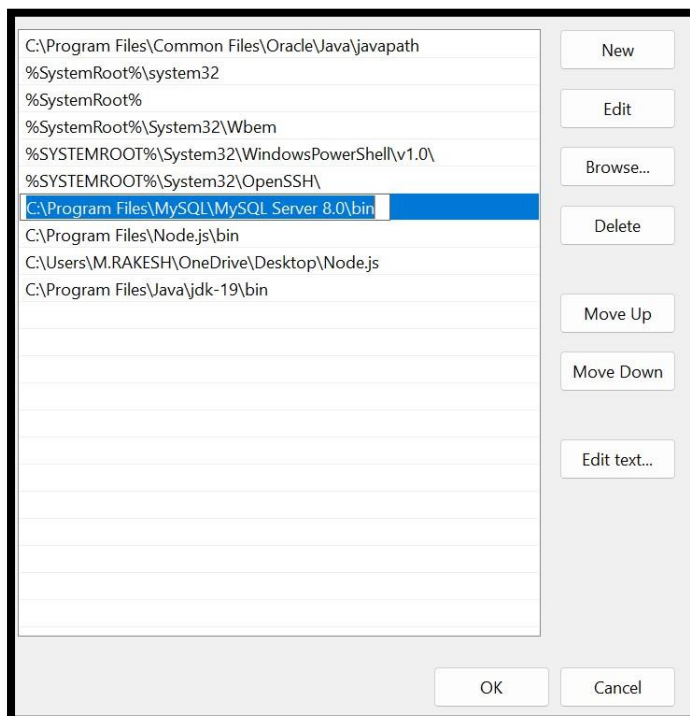
We have set the path for the Mysql workbench.

- For that open Environment Variables.
- Click on Environment Variables and select path.



Open the path and click on new.

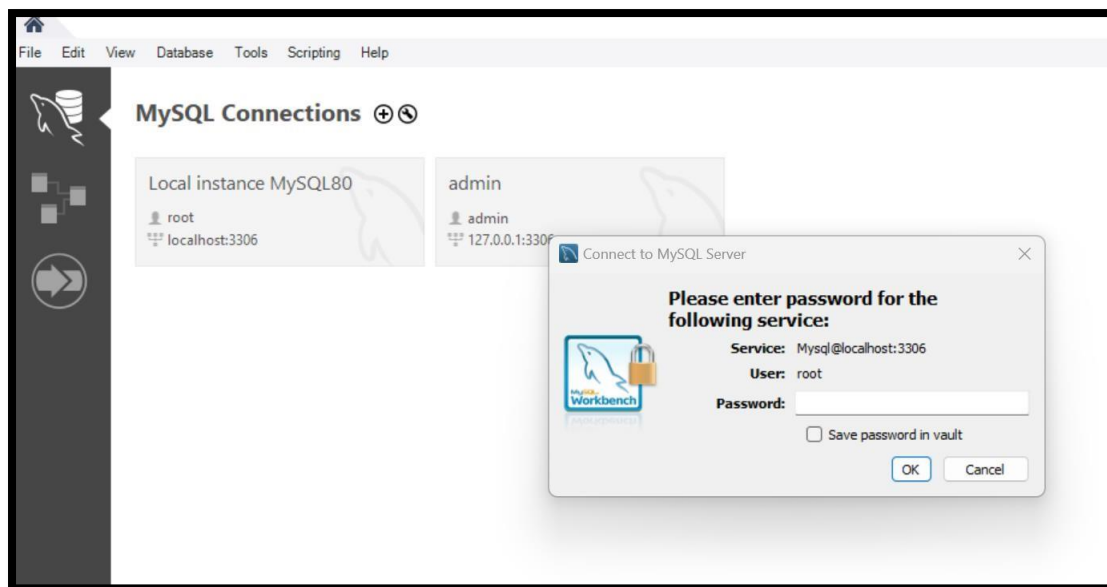
Here paste the mysql location.



Then click on ok so that path for mysql is setted.

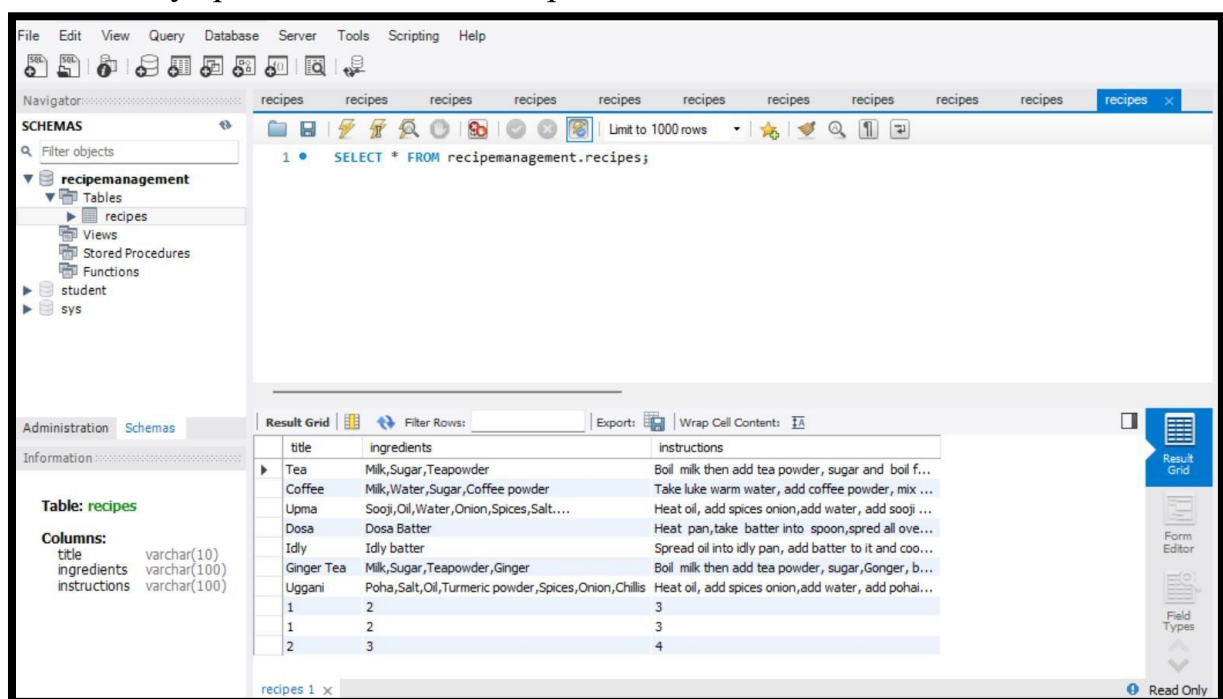
Creating a Database:

Open the Mysql workbench which is installed



Here enter the password which was created in setup.

Then the mysql workbench will be opened.

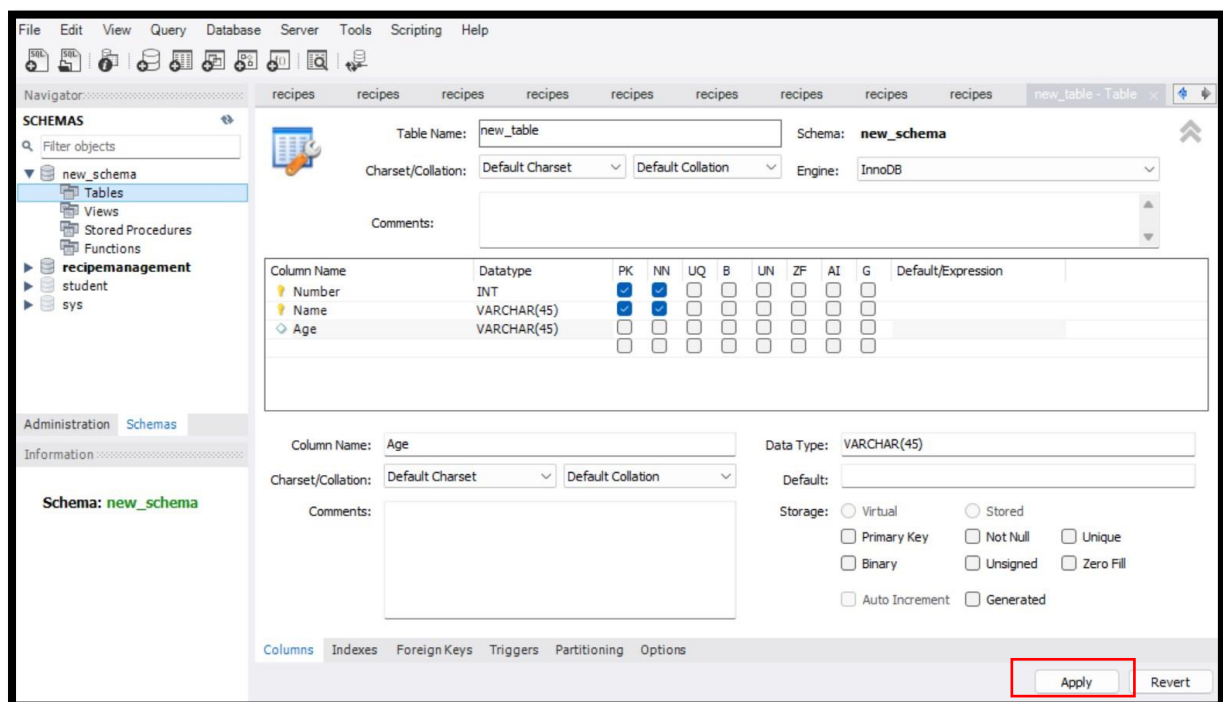


This is the demo table .

If you want to create a new database click on the highlighted bar(For creating the new schema).

Now give a name for the database.

Now we have to select the required tables with rows that should be there in database with number of values.

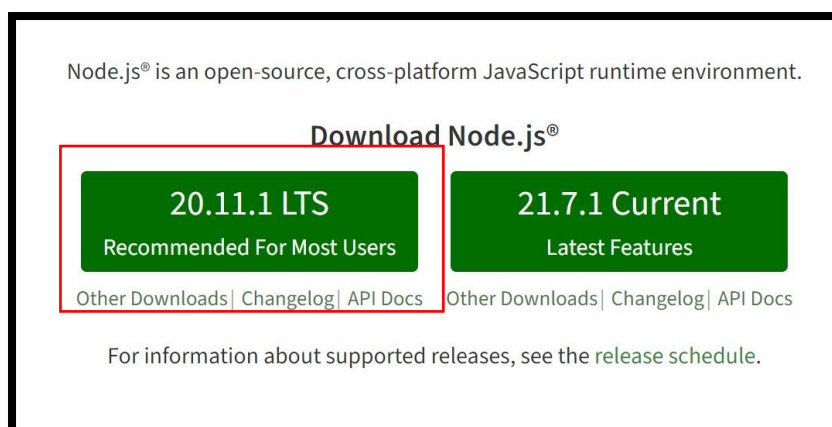


Create a database as above with the required information(For recipe Management we need title,ingredients,instructions...)

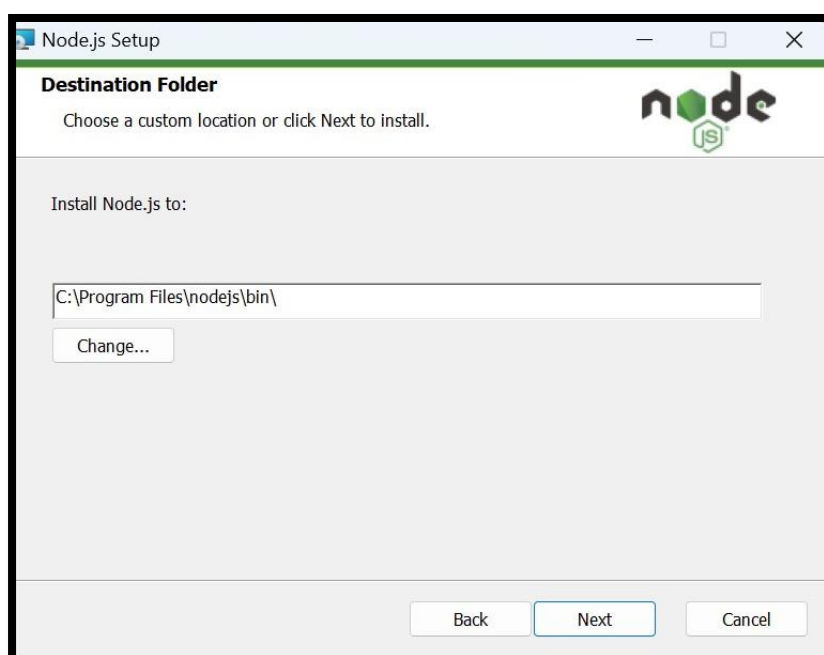
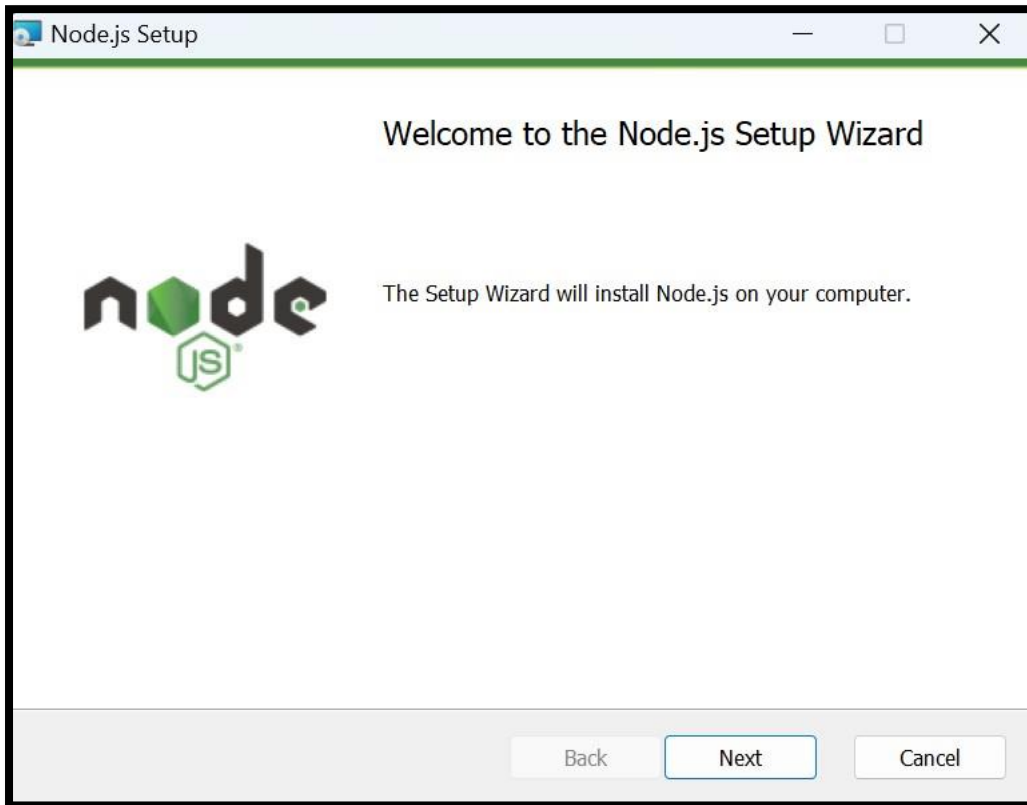
Then save the database created by clicking on apply.

Nodejs installation:

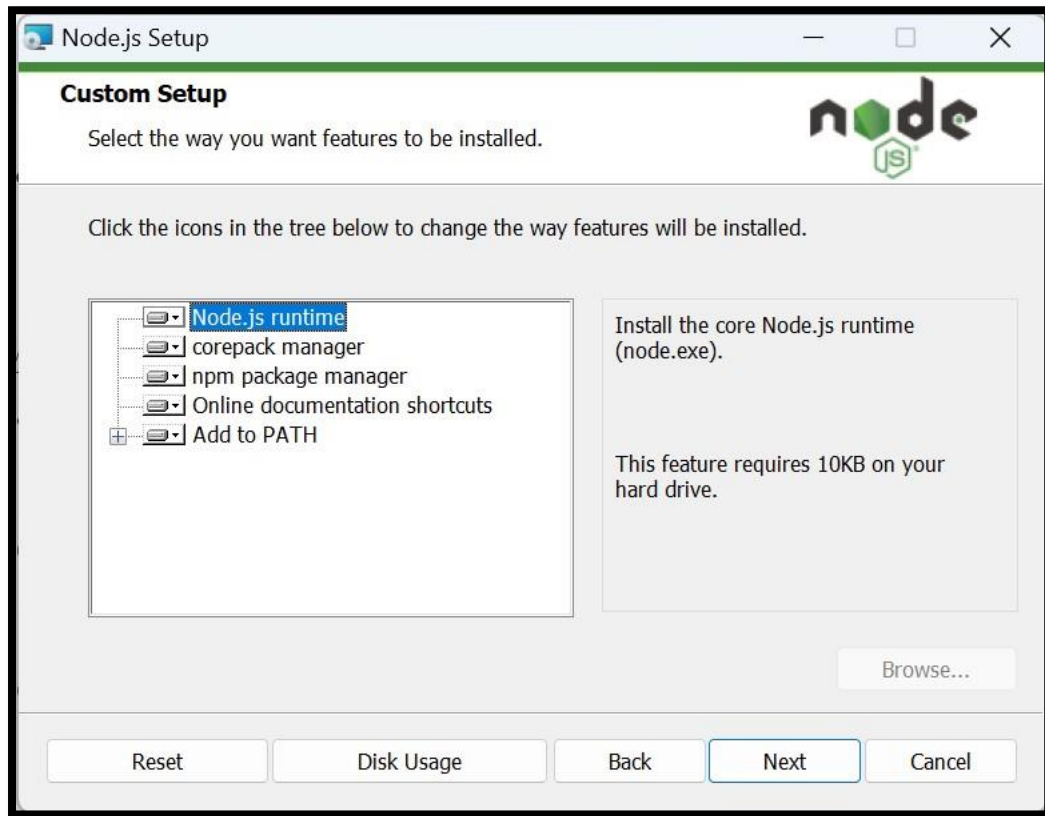
- Open chrome/Google.
- Search for Nodejs.
- <https://nodejs.org/en>



- Now click on the above LTS which supports all functions, then it will be starting the download.
- Then open the downloaded Nodejs for the setup, here is how the page will be displayed.

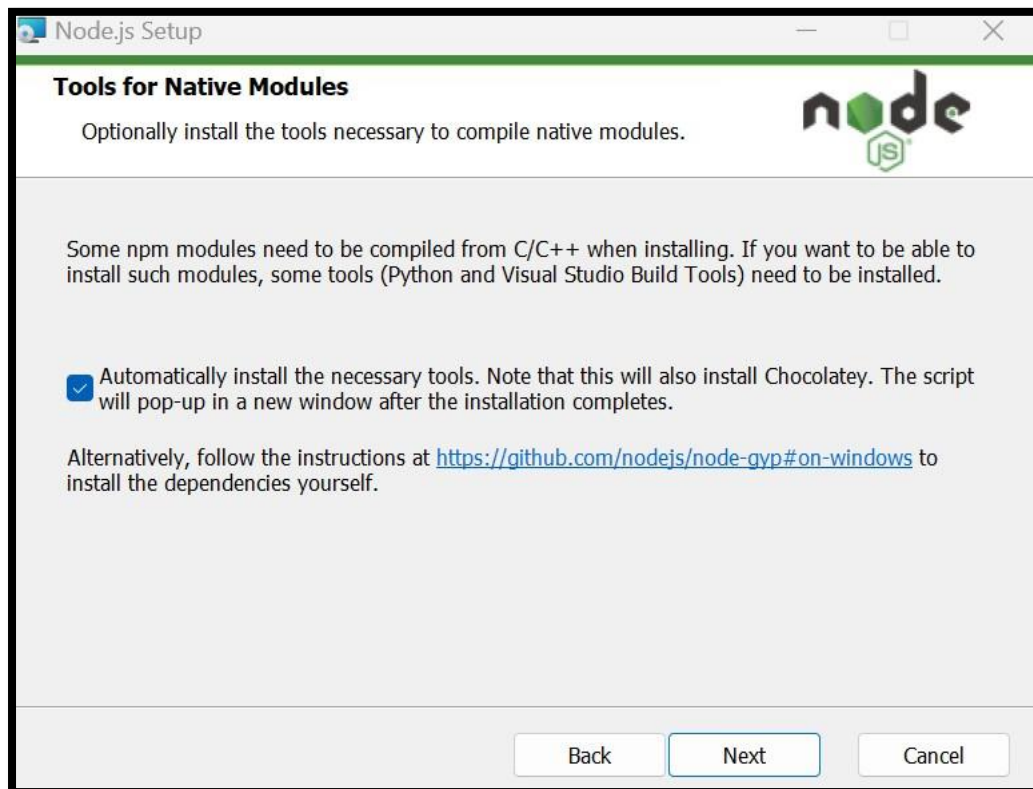


- - Then click on next for the setup.
 - Here you have to accept the all the terms in license agreement.
- Continue the process by by accepting all the terms and clicking next.



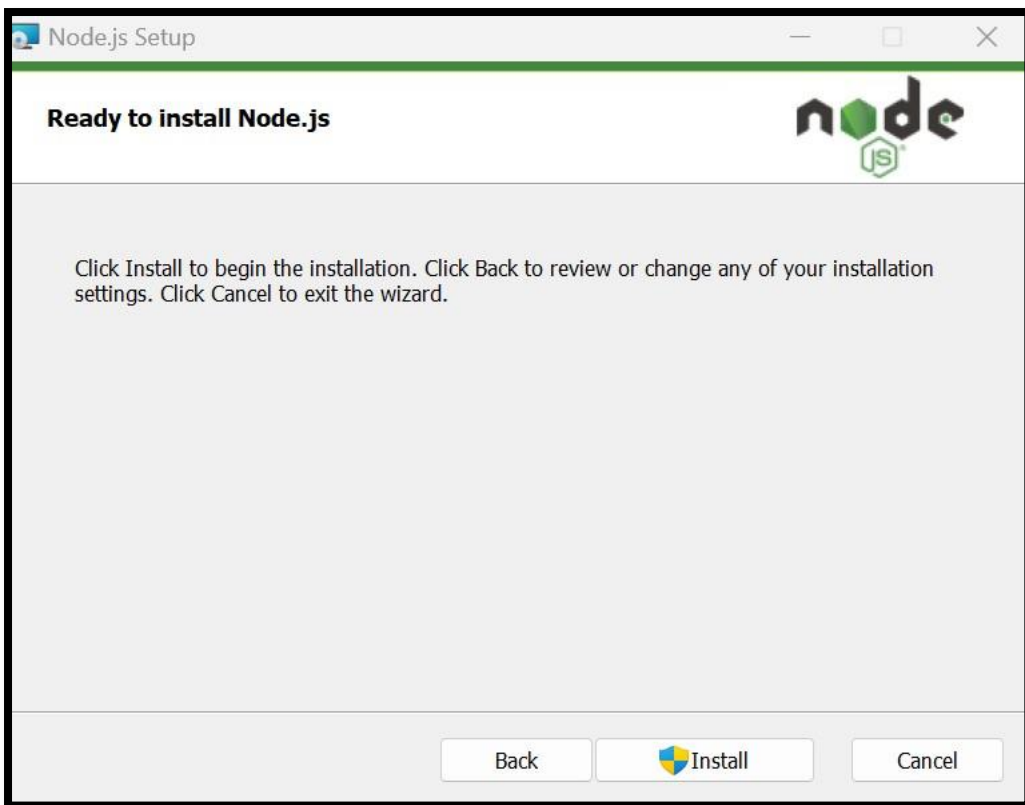
- Here click on node.js Runtime and then next.

-

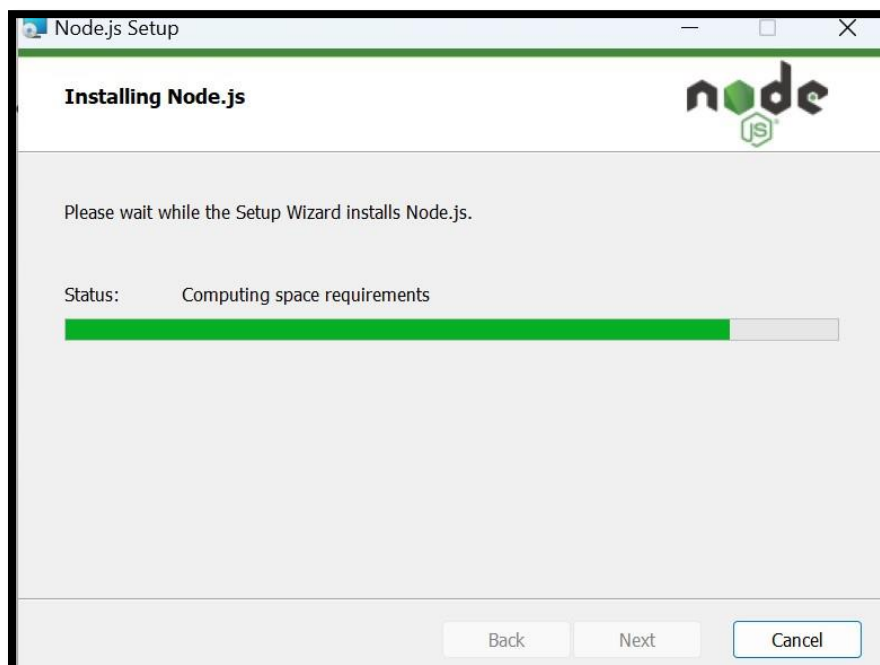


Here we have accept the terms the it will install necessary tools required and proceed to the next step.

- Click install to begin the Nodejs installation



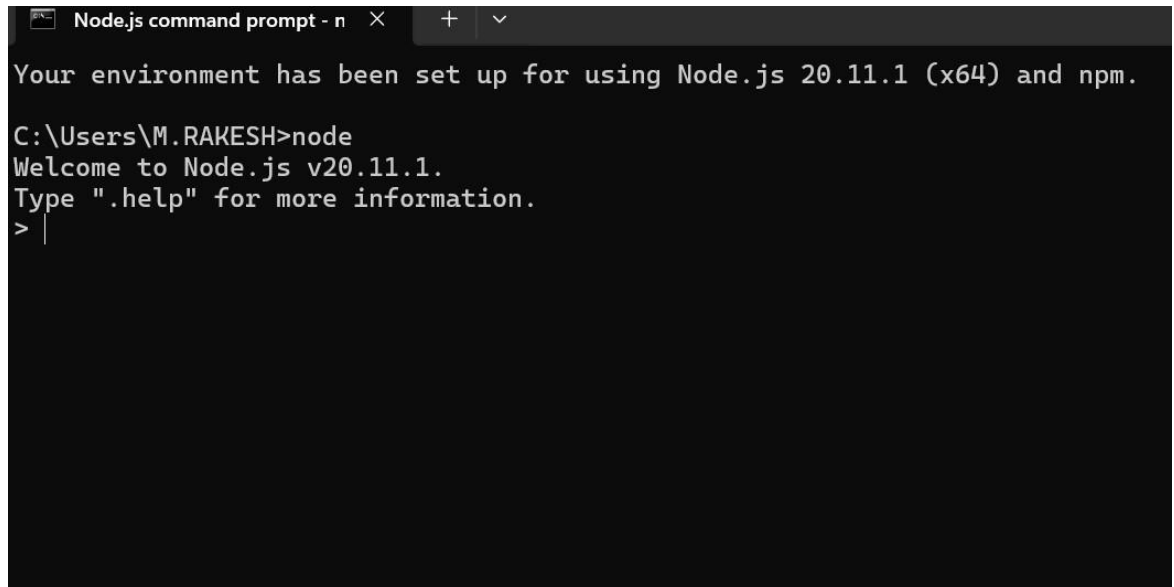
It will start installing



After installing click on finish to exit the set up process.

- Now Nodejs has been successfully installed.

-
- You can check the version of Nodejs installed by entering to Nodejs command prompt.
- Give ``node`` as command
- You are visible with the version of Nodejs.



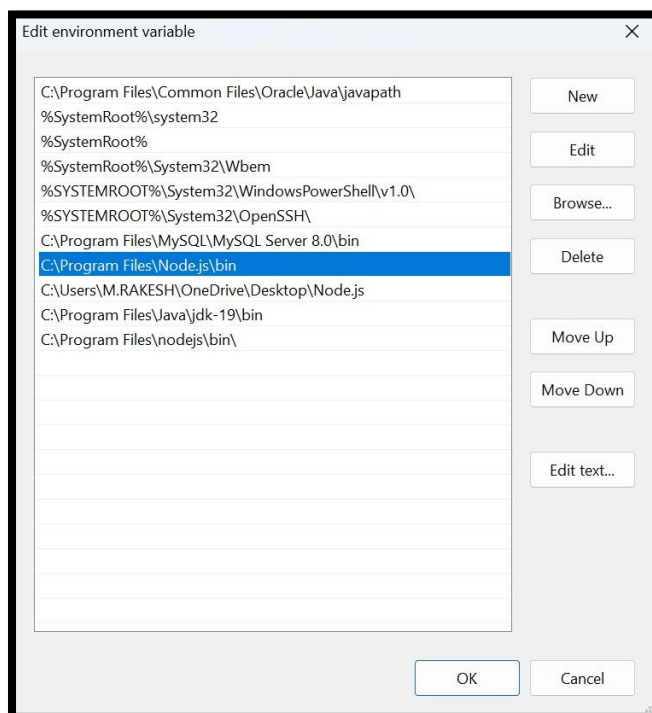
```

Node.js command prompt - n  X  +  v
Your environment has been set up for using Node.js 20.11.1 (x64) and npm.

C:\Users\M.RAKESH>node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
>

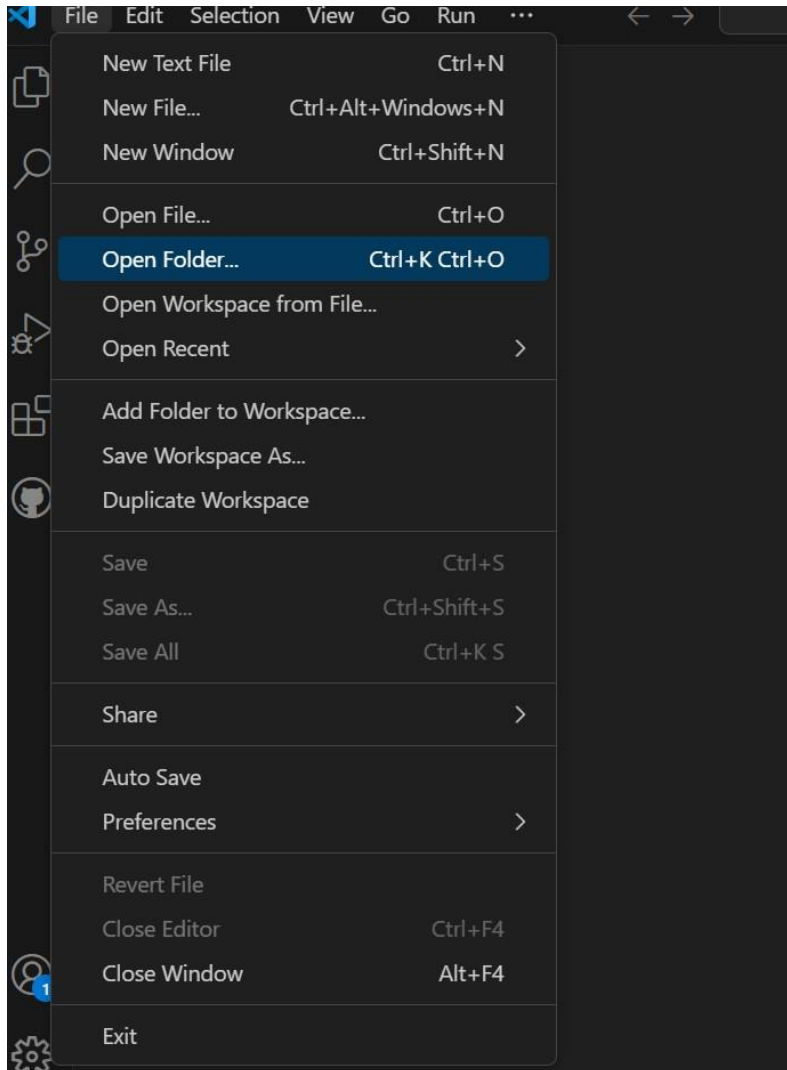
```

- Now set the path for the Nodejs installed.
- Open environment variables and set the new path.

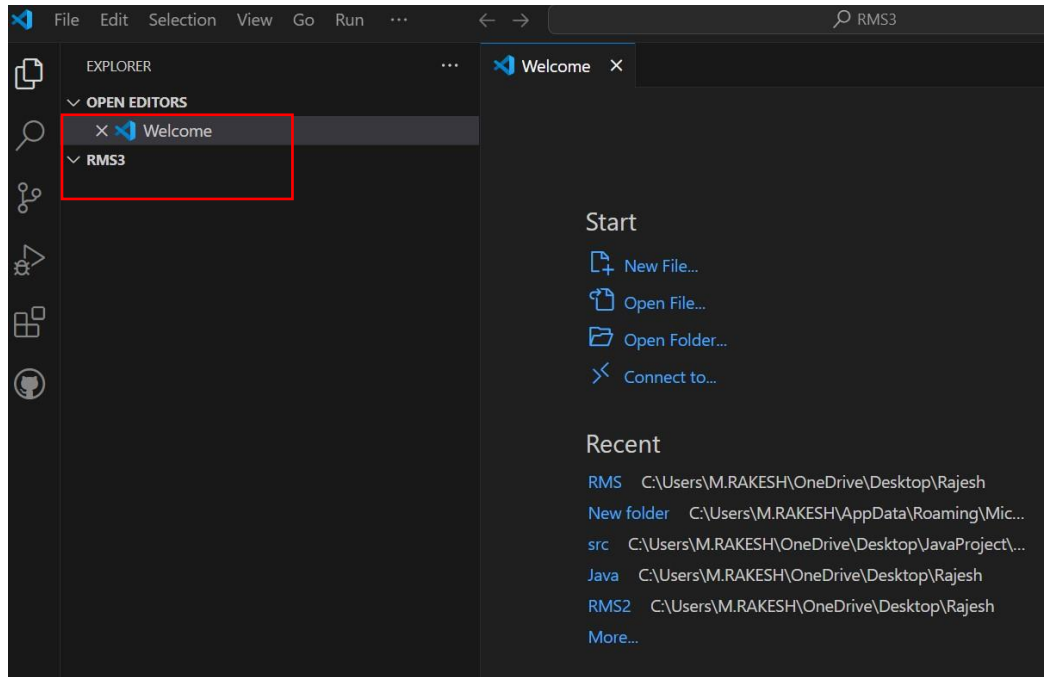


Connection of Node.js to Vs and Execution

- Open Visual studio.
- Create a folder in the desktop(RecipeManagement) .
- Then open files and click on open folder.



- Now enter new folder and enter the name or the path of the folder.
- Else create a new folder and open it in VS.
- Your folder will be visibled on the left side of the visual studio page.

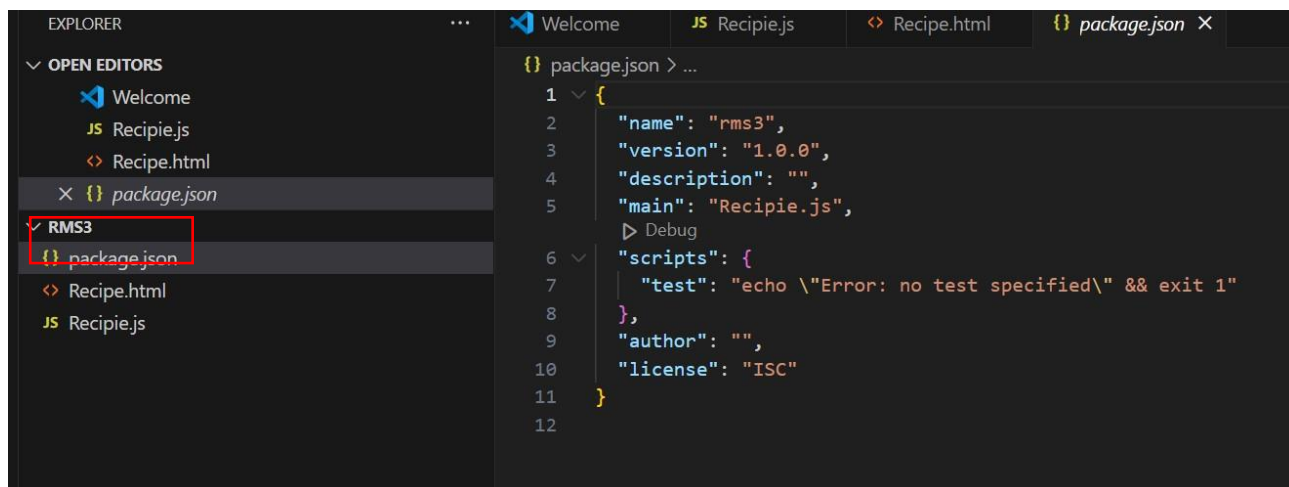


Create to files for html and nodejs.(i.e.,Recipe.html,Recipe.js)

Initialize Node.js Project:

- In the terminal or command prompt , navigate to your project folder where you want to create Node.js project.
- Now in desired directory run the following command
- **Npm init**//it prompt you to answer some questions on your project.
-

After answering ,npm will generate package.json file automatically in project



directory based on your response.

Install Necessary Packages:

Install any required Node.js packages using npm install in the terminal.

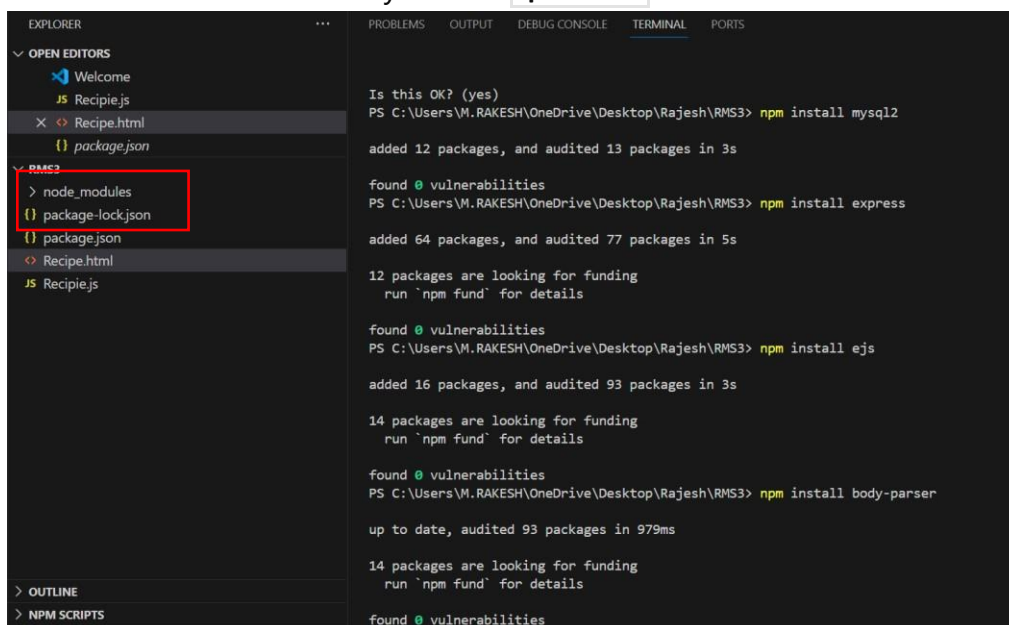
1. **npm install express** // installs the Express Framework.

2. **npm install mysql2** //installs the MySQL.

3. **npm install ejs** //installs the EJS (Embedded JavaScript) package

4. **Npm install body-parser**// This package is used to parse incoming request bodies in **Express.js**

5. **Npm install package-lock.json**// file is automatically generated and updated with node-modules when you run **npm install**.



1. MySQL Connection :

- Establishes the connection to a MySQL database named 'recipes'(user defined) on the localhost.

○ **`const mysql = require ('mysql2') ;`**: This is a built-in Node.js function used to import modules. It loads the **mysql2** module into the **mysql** variable.

○ **`const express = require('express');`** : Commonly used Node.js function to import the Express frame work .

- **`const app = express();`** : Creates an instance of the Express application (or) initializes a new Express application.
- **`const port = 3028;`** : Defines the constant declared port number of the server .
- **`const connection = mysql.createConnection({...});`** : Creates a connection to the MySQL database ('recipes/user defined') using host , user , password and database name .

```
const mysql = require('mysql2');
const express = require('express');
const app = express();
const port = 3002;

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'admin',
  database: 'recipemanagement',
});
```

2.Express Middleware :

- 'body-parser' is a Node.js middleware that extracts the entire body portion of an incoming request stream and exposes it on req.body for handling JSON and URL-encoded data.
- **`app.use(bodyParser.json());`** :It is a middleware that is commonly used with the Express framework. Configure Express to parse JSON data to request bodies.
- **`app.use(bodyParser.urlencoded({ extended : true }));`** To parse incoming request bodies encoded in URL-encoded format allowing user to access the data submitted by the client.

```
const bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

○

3.Route Handler:

- Handles GET requests to retrieve a list of all recipes or details of a specific recipe from list.
- Handles Post requests to add new recipes to the 'RECIPES' table in the database.
- Serves the HTML file on the root endpoint.
- **`app.get('/', ...)`** : Sets a Route Handler and Servers the HTML File on the root endpoint.
- **`app.get('/recipes', ...)`** : Fetches and returns a list of recipe titles .
- **`app.get('/recipes/:title?', ...)`** : Fetches details of a specific recipe or all recipes based on the title parameter.
- **`app.post('/', ...)`** : Handles POST requests to add new recipes to the database .

4.Database Interaction :

- In a recipe management system, database(RECIPES) interaction is crucial for storing, retrieving, updating, and deleting recipe data.
- Provides error handling for database interactions .

Connection Handling :

- **`connection.connect(...)`** : Connects to the MySQL database, handling connection errors .

Fetching Recipe Titles :

- **`app.get('/recipes', ...)`** : Executes a query to fetch recipe titles and handles potential errors of the database.

Fetching Recipe Details :

- `app.get('/recipes/:title?', ...)` : If a title parameter is provided in the URL (`/recipes/some-title`), the server responds with a message indicating that it is fetching recipes with the specified title.

```
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/web.html');
});

app.get('/recipes', function (req, res) {
  connection.query('SELECT title FROM RECIPES', function (error, results, fields) {
    if (error) {
      console.error('Error fetching titles from MySQL:', error);
      res.status(500).send('Internal Server Error');
      return;
    }
    res.json(results);
  });
});

app.get('/recipes/:title?', function (req, res) {
  const recipeTitle = req.params.title;
  console.log('recipeTitle ' + recipeTitle);
});
```

Inserting New Recipes :

- `app.post('/', ...)` : Inserts new recipe data into the 'recipes' tables and handles errors .

```
app.post('/', function (req, res) {
  var title = req.body.title;
  var ingredients = req.body.ingredients;
  var instructions = req.body.instructions;

  // Sample data to be inserted
  const recipeData = {
    title: title,
    ingredients: ingredients,
    instructions: instructions,
  };
});
```

○

Server Start :

- Listens for incoming connections on port number.

- `app.listen(port,...)`: Starts the server on the specified port(3028) and logs a confirmation message. You can access your application by navigating to `http://localhost:3000` in your web browser

```
app.listen(port, () => {  
  console.log(`Server is running at http://localhost:\${port}`);  
});
```

Create Node.js Files:

- Use VSCode to create your node file name.js files (e.g., web.js).
- Write your Node.js code in these files.
- In the integrated terminal, run your Node.js application.
- Node file name.js (node web.js).

Open Integrated Terminal:

- Use the `ctrl+alt+ ~` button to open the integrated terminal for running the Node.js file.

Run Node.js Application:

- In the integrated terminal, run your Node.js application.
- node file name.js (**node web.js**)

Debugging:

Set breakpoints in your code.

Create a configuration for debugging by adding a `.vscode/launch.json` file.

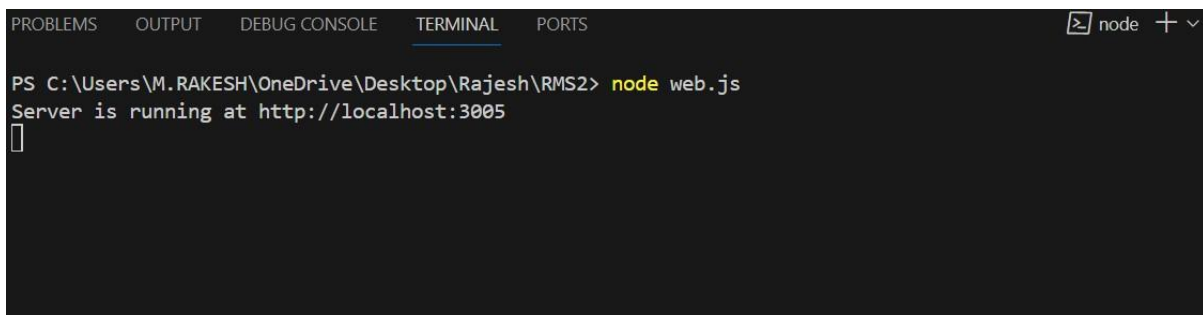
Example launch.json configuration:

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "node",  
      "request": "launch",  
      "name": "Launch Node.js",  
      "program": "${workspaceFolder}/app.js",  
      "stopOnEntry": false,
```

```
"cwd": "${workspacefolder}/app.js,  
"runtimeArgs": ["--inspect-brk"],  
"sourceMaps": true,  
"outFiles": ["${workspaceFolder}/out/**/*.js" ]}]}
```

Running the code. View Output:

- Check the output of your Node.js application in the integrated terminal by running the code as **node file name.js**.
- As my file name is web.js I am running **node web.js**. • Click on the server port link it re directs to the web page.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS node + v  
PS C:\Users\M.RAKESH\OneDrive\Desktop\Rajesh\RMS2> node web.js  
Server is running at http://localhost:3005  
█
```

By selecting any recipe from recipe list the required ingredients, instructions will be displayed.



Recipe Management System

Title:

Ingredients:

Instructions:

Recipe List

Select Recipe:

Ingredients:

Instructions:

HTML Code Explanation:

Save the file name with **File name.html(Recipe.html)**

HTML Code Explanation :

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport  
content="width=devicewidth,initialscale=1.0">
```

```
  <title>Recipie Management System</title>
```

// **<!-- CSS Styles -->**

```
<style>
```

```
  /* Define a class for common styling */
```

```
    body{      background-
```

```
image:
```

```
url("https://bimpos.com/sites/default/files/images/posts/412bbac9b0db-41fe-  
85a3-015e4945df69.jpeg");
```

```
background-size: cover;
```

```
background-repeat: no-repeat;
```

```
}
```

```
  centered-label {
```

```
text-align: center;
```

```
}
```

```
.container {
```

```
  justify-content: center;
```

```
  text-align: center;
```

```
}
```

```
.content {
```

```
max-width: 500px;
```

```
margin: auto;
```

```
border:3px solid white;
```

```
padding:10px;
```

```
margin:0 300px;
```

```
background-color: white;
```

```
}
```

```
.a {  
width: 100%;  
height: 25px;  
}
```

```
/* Specific styling for textareas */  
#ingredientsInput,  
#instructionsInput  
{ height:  
40px;  
}
```

```
/* Styling for the 'Add Recipe' button */  
#b {  
background-color: rgb(27, 226, 27);  
}
```

```
.content { max-  
width: 500px;  
margin: auto;  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<!-- Recipe Input Form -->
```

```
<form action="/" method="post">
```

```
    <!-- Heading for the Recipe Management System -->
```

```
<h1>Recipe Management System</h1>
```

```
    <div class="content">
```

```
    <!-- Input field for the recipe title -->
```

```
    <label for="title">Title:</label>
```

```
    <br>
```

```
    <input type="text" id="title" class="a" name="title">
```

```
    <br><br>
```

```
    <!-- Textarea for entering recipe ingredients -->
```

```
    <label for="ingredients">Ingredients:</label>
```

```
    <br>
```



```
<textarea id="ingredientsInput" class="a" name="ingredients"></textarea>
<br><br>
```

```
<!-- Textarea for entering recipe instructions -->
```

```
<label for="instructions">Instructions:</label>
```

```
<br>
```

```
<textarea id="instructionsInput" class="a" name="instructions"></textarea>
```

```
<br><br>
```

```
<!-- Button to submit the form and add a new recipe -->
```

```
<button id="b" type="submit">Add Recipe</button>
```

```
</form>
```

```
</div>
```

```
<br>
```

```
<!-- Recipe List Section -->
```

```
<div class="container">
```

```
<h2>Recipe List</h2>
```

```
</div>
```

```
<!-- Dropdown for selecting a recipe -->
```

```
<div class="content">
```

```
<label for="recipeList">Select Recipe:</label><br><br>
```

```
<select id="recipeList" onchange="displaySelectedRecipe()">
```

```
  <option value="">Select a Recipe</option>
```

```
</select><br><br>
```

```
<!-- Display section for selected recipe details -->
```

```
<label>Ingredients:</label>
```

```
<br>
```

```
<textarea id="selectedIngredients" class="a" name="selectedIngredients"
readonly></textarea>
```

```
<br><br>
```

```
<label>Instructions:</label>
```

```
<br>
```

```
<textarea id="selectedInstructions" class="a" name="selectedInstructions"
readonly></textarea>
```

```
<br><br>
```

```
</div>
```

<!-- JavaScript Section -->

```
<script>
```

```
// Event listener to ensure the DOM is fully loaded before executing  
JavaScript
```

```
document.addEventListener('DOMContentLoaded', () => {
```

```
// Fetch and populate the recipe list on page load fetchRecipeList();  
});
```

```
// Function to fetch and display the list of recipes async function
```

```
fetchRecipeList() {
```

```
  console.log('Fetching recipe list...');
```

```
// Make a GET request to the '/recipes' endpoint on the server const  
response = await fetch('/recipes');
```

```
// Parse the JSON response
```

```
const recipes = await response.json();
```

```
// Get the dropdown element
```

```
const recipeList = document.getElementById('recipeList');
```

```
// Clear existing options and add a default one
```

```
recipeList.innerHTML = '<option value="">Select a Recipe</option>';
```

```
// Iterate through the fetched recipes and add them as options to the  
dropdown
```

```
recipes.forEach(recipe => {
```

```
  const option = document.createElement('option'); option.value
```

```
  = recipe.title; option.textContent
```

```
  = recipe.title; recipeList.appendChild(option);
```

```
});
```

```
}
```

```
// Function to display details of the selected recipe async function
```

```
displaySelectedRecipe() {
```

```
// Get the selected recipe title from the dropdown var
```

```
select = document.getElementById('recipeList'); var
```

```
selectedTitle = select.options[select.selectedIndex].value;
```

```
console.log('Selected Title:', selectedTitle);
```

```
// Check if a recipe is selected
```

```
if (selectedTitle) { try {
```

```

// Update the URL to fetch details of the individual recipe using its title
const url = `/recipes/${encodeURIComponent(selectedTitle)}`;
console.log('Fetching recipe from URL:', url);

// Make a GET request to the updated URL
const response = await fetch(url);
console.log('Response status:', response.status);

// Check if the request was successful
if (!response.ok) {
  throw new Error(`Failed to fetch recipe. Status: ${response.status}`);
}

// Parse the JSON response
const recipe = await response.json();
console.log('Fetched recipe:', recipe);

// Display the ingredients and instructions of the selected recipe
document.getElementById('selectedIngredients').value = recipe.ingredients;
document.getElementById('selectedInstructions').value = recipe.instructions;
} catch (error) {

// Log and handle any errors that occur during the fetch
console.error('Error fetching recipe:', error);
}
} else {

// Clear the displayed details if no recipe is selected

document.getElementById('selectedIngredients').value = "";
document.getElementById('selectedInstructions').value = "";
}
}
</script>
</body>

</html>

```

Connection to MySQL and HTML Page Code Explanation :

Save the file name with **file name.js(web.js)**

1. Importing Dependencies

```

const mysql = require('mysql2');
const express = require('express');

```

```
const app = express();  
const port = 3063;
```

2. Creating a MySQL Connection

```
const connection = mysql.createConnection({ host: 'localhost', user: 'root',  
password: 'root', database: 'recipes',});
```

This block establishes a connection to a MySQL database named 'recipes' running on the local machine

3. Setting Up Express Middleware

```
const bodyParser = require('body-parser');  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: true }));
```

Configure Express to parse JSON and URL-encoded data for handling POST requests.

4. Serving HTML File on Root EndPoint

```
app.get('/', function (req, res) { res.sendFile(__dirname + '/web.html'); });
```

This route sends the 'web.html' file when users access the root endpoint.

5. Fetching Recipe Titles

```
app.get('/recipes', function (req, res) { // ... });
```

This endpoint retrieves a list of recipe titles from the 'RECIPES' table in the database.

6. Fetching Individual Recipe Details

```
app.get('/recipes/:title?', function (req, res) { // ... });
```

This endpoint fetches either all recipes or details for a specific recipe based on the provided title parameter.

7. Connecting to the Database

```
connection.connect(function (err) { // ... });
```

This block establishes a connection to the MySQL database and handles any connection errors .

8. Handling POST Requests to Add Recipes

```
app.post('/', function (req, res) { // ... });
```

This route handles POST requests to add new recipes to the 'RECIPES' tables in the database.

9. Starting the Express Server

```
app.listen(port, () => { console.log(`Server is running at  
http://localhost:${port}`); });
```

This line starts the Express server , making it accessible.

File name.js(Web.js)

```
const mysql = require('mysql2');  
const express = require('express');  
const app = express();  
const port = 3003;  
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  password: 'admin',  
  database: 'recipemanagement',  
});  
  
const bodyParser = require('body-parser');  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: true }));  
  
app.get('/', function (req, res) {  
  res.sendFile(__dirname + '/web.html');  
});  
  
app.get('/recipes', function (req, res) {
```

```
connection.query('SELECT title FROM RECIPES', function (error, results,
fields) {
  if (error) {
    console.error('Error fetching titles from MySQL:', error);
    res.status(500).send('Internal Server Error');
    return;
  }
  res.json(results);
});
});
```

```
app.get('/:recipe/:title?', function (req, res) {
  const recipeTitle = req.params.title;
  console.log('recipeTitle ' + recipeTitle);

  if (recipeTitle) {
    // Fetch details for a specific recipe
    connection.query('SELECT * FROM RECIPES WHERE title = ?',
[recipeTitle], function (error, results, fields) {
      if (error) {
        console.error('Error fetching recipe details from MySQL:', error);
        res.status(500).json({ error: 'Internal Server Error' });
        return;
      }

      if (results.length === 0) {
        res.status(404).json({ error: 'Recipe not found' });
        return;
      }

      res.json(results[0]);
    });
  } else {
    // Fetch all recipes
    connection.query('SELECT title FROM RECIPES', function (error, results,
fields) {
      if (error) {
```

```

        console.error('Error fetching titles from MySQL:', error);
        res.status(500).json({ error: 'Internal Server Error' });
        return;
    }

    res.json(results);
});
}
});

connection.connect(function (err) {
    if (err) {
        console.error('Error connecting: ' + err.stack);
        return;
    }
    //console.log('Connected as id ' + connection.threadId);
});

app.post('/', function (req, res) {
    var title = req.body.title;
    var ingredients = req.body.ingredients;
    var instructions = req.body.instructions;

    // Sample data to be inserted
    const recipeData = {
        title: title,
        ingredients: ingredients,
        instructions: instructions,
    };

    // Insert data into the "recipes" table
    connection.query('INSERT INTO RECIPES SET ?', recipeData, function
(error, results, fields) {
        if (error) {
            console.error('Error inserting data into MySQL:', error);
            res.status(500).send('Internal Server Error');

```

```
        return;
    }

    console.log('Data inserted successfully.');
```

```
    res.send('Data received and inserted successfully!');
  });

});

app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Conclusion:

Recipe Hub offers a comprehensive solution for managing and organizing recipes efficiently. With its user-friendly interface, robust features, and focus on usability, Recipe Hub simplifies the process of storing, searching, and for accessing the recipes for users of all levels. Through its structured data model, Recipe Hub allows users to input detailed recipe information, including title, ingredients, instructions, and more.