# **Inventory Management in Cook Book: Your Virtual Assistant with React.JS**

**1.Introduction**

TOPIC: COOK BOOK:YOUR VIRTUAL ASSISTENT

This document provides a detailed guide on preparing an inventory management system for a virtual assistant application named Cook Book using React.js.

* The system will allow users to:
* Manage ingredients
* Track stock levels
* Link ingredients with recipes
* Provide real-time updates for kitchen operations

TEAM MEMBERS AND THEIR ROLE:

* Abinaya.S - Documentation
* Abinaya.C - Coding
* Abarna.A - Coding
* Aglaya.S – Demo Video

2. Project Setup

# 2.1 Purpose and Features:

## Cookbook: Your Virtual Kitchen Assistant helps users cook easily by providing smart recipes, step-by-step guidance, and personalized meal planning.

* **Smart recipe suggestions based on ingredients.**
* **Step-by-step cooking guidance (voice/text).**
* **Meal planner with grocery list.**
* **Nutrition info and healthy alternatives.**
* **Save favorites and personalize recipes.**

# 2.2 Prerequisites

Before starting ,Ensure the following are installed:

* Node.js (v14 or above)
* npm or yarn
* Code editor (VS Code recommended)
* Git (optional for version control)

# 2.2 Create React App

npx create-react-app cook-book-inventory

cd cook-book-inventory

npm start

This initializes your project and launches the development server at http://localhost:3000.

3. Project Structure

**cook-book-inventory/**

**├── public/**

**├── src/**

**│ ├── components/**

**│ │ ├── IngredientList.js**

**│ │ ├── IngredientForm.js**

**│ │ └── IngredientItem.js**

**│ ├── context/**

**│ │ └── InventoryContext.js**

**│ ├── App.js**

**│ └── index.js**

**├── package.json**

**└── README.md**

4. Key Features

* Display Ingredient List
* Add New Ingredients
* Edit Existing Ingredients
* Delete Ingredients
* Track Ingredient Stock Levels
* Integrate with Recipes and Meal Plans

5. Inventory Context Setup

We will use React Context API to manage ingredient data globally.

\*\*src/context/InventoryContext.js\*\*

import React, { createContext, useState } from 'react';

export const InventoryContext = createContext();

export const InventoryProvider = ({ children }) => {

const [ingredients, setIngredients] = useState([]);

const addIngredient = (ingredient) => {

setIngredients([...ingredients, ingredient]);

};

const removeIngredient = (id) => {

setIngredients(ingredients.filter(ing => ing.id !== id));

};

const updateIngredient = (updatedIngredient) => {

setIngredients(

ingredients.map(ing => ing.id === updatedIngredient.id ? updatedIngredient : ing)

);

};

return (

<InventoryContext.Provider value={{ ingredients, addIngredient, removeIngredient, updateIngredient }}>

{children}

</InventoryContext.Provider>

);

};

6. Ingredient List Component

\*\*src/components/IngredientList.js\*\*

import React, { useContext } from 'react';

import { InventoryContext } from '../context/InventoryContext';

import IngredientItem from './IngredientItem';

const IngredientList = () => {

const { ingredients } = useContext(InventoryContext);

return (

<div>

<h2>Ingredient Inventory</h2>

{ingredients.length === 0 ? (

<p>No ingredients available.</p>

) : (

ingredients.map(ingredient => (

<IngredientItem key={ingredient.id} ingredient={ingredient} />

))

)}

</div>

);

};

export default IngredientList;

7. Ingredient Form Component

\*\*src/components/IngredientForm.js\*\*

import React, { useState, useContext } from 'react';

import { InventoryContext } from '../context/InventoryContext';

const IngredientForm = () => {

const { addIngredient } = useContext(InventoryContext);

const [name, setName] = useState('');

const [quantity, setQuantity] = useState('');

const handleSubmit = (e) => {

e.preventDefault();

addIngredient({ id: Date.now(), name, quantity: parseInt(quantity) });

setName('');

setQuantity('');

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

placeholder="Ingredient Name"

value={name}

onChange={(e) => setName(e.target.value)}

required

/>

<input

type="number"

placeholder="Quantity"

value={quantity}

onChange={(e) => setQuantity(e.target.value)}

required

/>

<button type="submit">Add Ingredient</button>

</form>

);

};

export default IngredientForm;

8. Ingredient Item Component

\*\*src/components/IngredientItem.js\*\*

import React, { useContext } from 'react';

import { InventoryContext } from '../context/InventoryContext';

const IngredientItem = ({ ingredient }) => {

const { removeIngredient } = useContext(InventoryContext);

return (

<div>

<span>{ingredient.name} - {ingredient.quantity}</span>

<button onClick={() => removeIngredient(ingredient.id)}>Delete</button>

</div>

);

};

export default IngredientItem;

9. Integrating Context in React.js

src/React.js

import React from 'react';

import { InventoryProvider } from './context/InventoryContext';

import IngredientList from './components/IngredientList';

import IngredientForm from './components/IngredientForm';

const App = () => {

return (

<InventoryProvider>

<div style={{ padding: '20px' }}>

<h1>Cook Book Virtual Assistant - Inventory</h1>

<IngredientForm />

<IngredientList />

</div>

</InventoryProvider>

);

};

export default App;

10.Screenshot or Demo

https://drive.google.com/file/d/1D0xErj7Xbs6E\_MyFSXekZcK08Qx\_z\_eF/view?usp=drivesdk

11. Testing & Deployment

* Test functionalities by adding, editing, and deleting ingredients.
* Ensure stock levels update correctly.
* Integrate with recipe management features.
* Deploy using Netlify or Vercel.

12. Future Improvements

* Add recipe-based ingredient suggestions.
* Implement user authentication.
* Integrate with APIs for nutritional information.
* Enhance UI with Tailwind CSS or Material UI.