A logo with a chef hat and whisk

Description automatically generated

**Programmer Manual**

B9-Group: 19

Yotam Gilad: 315819599  
Gad Azriel: 209146927  
Tomer Ben-Lulu: 318318987  
Almog Elbaz: 213037369

Direct link to the website: <https://cook-mate-kappa.vercel.app>  
Link to the repository: <https://github.com/GadAzriel/CookMate>  
Link to MTW: <https://www.morethanwallet.com/app/862>

**Table of Contents**

Introduction…………….........................................................................................................................3

Key Features......................................................................................................................................3

System Architecture Overview.......................................................................................................4-5

Key Files and Folders......................................................................................................................6-7

Main Components …………………………………………………………………....................................................8-9

Key Functions Documentation………...........................................................................................10-13

Backend Architecture and Features................................................................................................14

Key Functions Documentation in Backend …………………………………………………..………………………15 -16

Source ………………………………………………………………………………………………………………….………………….17

**Introduction**

This manual is intended for developers working on the CookMate Interactive Cooking Assistant web application. The project is divided into a backend and a frontend, stored in separate GitHub repositories.

The manual describes the web architecture, code organization, interaction between React components in the front and Next.js API routes in the backend, and how to use and modify components with Tailwind CSS to change the user interface.

CookMate is an interactive web application designed to make cooking a seamless and enjoyable experience. Users can browse various recipes, get detailed dish preparation instructions, and engage in a step-by-step cooking guide.

**Key Features**

* **Recipe Browsing**: Users can explore a curated list of recipes on the homepage.
* **Detailed Instructions**: Each recipe comes with comprehensive preparation steps and ingredient lists.
* **Interactive Cooking Guide**: The interactive cooking feature guides users through the cooking process with a visual progress bar.
* **Responsive Design**: Optimized for various device sizes, providing a consistent user experience across desktops, tablets, and mobile devices.
* **Accessibility**: Navigation controls and UI elements are designed to be accessible and user-friendly.
* **Dark Mode and Light Mode**: Option to switch between dark and light modes.

**System Architecture Overview**

The **CookMate** application is built with a robust architecture that seamlessly integrates key technologies:

* **ReactJS** for dynamic web development using a component-based architecture.
* **Next.js** for Backend: The backend is developed using Next.js to handle API routes and server-side logic.
* **MongoDB**: A NoSQL database used for storing and retrieving recipe data, offering scalability and flexibility.
* **Tailwind CSS** for creating a responsive and aesthetically pleasing user interface.

**Key Components**

* **ReactJS**: The application is developed using React, which provides a structured, component-based architecture. This approach allows for efficient rendering and easy maintenance.
* **MongoDB**: Used as the database, storing all recipe data and enabling quick and easy access via API routes in the Next.js backend.
* **UI**: The user interface is crafted using Tailwind CSS, ensuring a modern, responsive design that works well across all devices.

**Key Libraries and Frameworks**

* **React**Purpose: JavaScript library for building dynamic user interfaces with reusable components.Key Features: Component-based architecture, Virtual DOM, and state management with hooks.
* **Tailwind CSS**Purpose: Utility-first CSS framework for styling the application directly in the HTML.Key Features: Highly customizable, responsive design, and simplified CSS management.
* **Custom CSS**Purpose: Custom styles for fine-tuning the visual appearance beyond what Tailwind provides.  
  Key Features: Allows for specific adjustments, overrides default styles, and ensures consistent branding across the application
* **React Router**Purpose: Library for routing in React, enabling navigation between different pages.Key Features: Declarative routing, dynamic routing, and URL parameter handling.
* **React Icons**Purpose: Collection of popular icons as React components.Key Features: Easy integration, customizable size and color, lightweight.
* **React Slick**Purpose: Carousel component for React, based on Slick carousel.Key Features: Responsive, customizable settings, smooth animations.
* **React Speech Recognition**Purpose: Hook for adding speech recognition to React apps.Key Features: Voice command processing, continuous listening, integration with Web Speech API.
* **Slick Carousel**Purpose: Versatile slider component for displaying dynamic content.Key Features: Fully responsive, supports swipe gestures, easy React integration.
* **GitHub Pages**  
  Purpose: A platform for hosting the website's user interface.  
  Key Features: Simple deployment, reliable hosting, and integration with GitHub for continuous deployment.

Backend (Next.js)

* Next.js  
  Purpose: A React framework for building dynamic web applications with features like server-side rendering and API routes.  
  Key Features: Server-side rendering, static site generation, API routes, and seamless integration with React.
* MongoDB  
  Purpose: A NoSQL database used for storing and managing recipe data.  
  Key Features: Flexible schema, scalability, and seamless integration with JavaScript-based applications.

**Key Files and Folders**

An overview of the essential files and directories in the CookMate project, designed to help you understand their purpose and navigate the codebase more effectively.

**1. src/** Purpose: This is the main directory where all the React components, assets, and styles are   
 stored.Key Contents: **\* App.jsx**: The root component that sets up the entire application structure and   
 manages routing. **\* index.js**: The entry point of the React application, responsible for rendering   
 App.jsx to the DOM.  
 **\* index.css:** Global CSS file applied across the entire application, providing basic   
 styles and custom overrides.  
 **\* style.js:** Centralizes Tailwind CSS class names used across components for   
 consistent styling.  
 **\*** **components/**: Contains reusable UI components.  
 \* **Header.jsx**: Manages the site’s header, including navigation and the dark   
 mode toggle.  
 \* **Footer.jsx**: Manages the footer, which includes site credits.  
 \* **Recipes.jsx**: Displays a list of recipes.  
 \* **RecipeDetail.jsx**: Provides detailed information about a selected recipe.  
 \* **Steps.jsx**: Offers an interactive cooking guide with step-by-step   
 instructions.  
 **\* pages/**: Contains components that represent full pages within the application.  
 \* **Home.jsx**: The landing page of the application.  
 \* **About.jsx**: The about page, giving details about the project and   
 contributors.  
 **\*** **assets/**: Stores static files like images, videos, and the JSON file containing recipe   
 data.  
 \* **recipes.json**: Acts as a simple database for storing all recipe data.  
 \* **Home\_images/, About\_images/, Recipes\_images/**: Folders for storing   
 images used throughout the site.  
 \* **videos/**: Contains video files associated with recipes.

**2. public/** Purpose: Contains static files that are served directly by the web server.Key Contents: **\* index.html**: The main HTML template file for the application. **\* Logo.jpg**: The logo image displayed on the website.

**3. package.json** Purpose: Manages the project’s dependencies, scripts, and metadata.Key Contents: **\*** dependencies: Lists all npm packages that the project relies on. **\*** scripts: Contains commands to run, build, and test the application.

**4. package-lock.json:** Locks the versions of dependencies to ensure consistency across different   
 environments.

**5. tailwind.config.js** Purpose: Configures the Tailwind CSS framework.Key Contents: Specifies the files Tailwind should scan for class names and allows for   
 customization of the Tailwind theme.

**6. node\_modules/** Purpose: Contains all npm packages and dependencies used by the project.

**7. README.md:** Provides an overview of the project, along with setup instructions and   
 contribution guidelines.

**Main Components**

In the CookMate project, these main components are integral to the application’s structure and functionality.   
This overview focuses on their roles from a programmer’s perspective. These components are styled using the Tailwind CSS framework, allowing for efficient and responsive design by applying CSS classes directly within the JSX code.   
Several components, such as the header, footer, RecipeDetail, and Steps, are designed as reusable components to maintain consistency and reduce redundancy across the application.

**index.js**The index.js file serves as the entry point for the React application. It’s where the React application is bootstrapped and rendered into the DOM. This file is critical because it initializes the entire application by rendering the App component inside a specified HTML element.

**App.jsx**Serves as the root component of the application. It configures routing using React Router, establishing the layout and determining which components render based on the URL. All major pages and features are initialized here.

**Header.jsx**Renders the site's header, including navigation links (Home, Recipes, About) and the dark mode toggle.   
This component handles navigation and user interface consistency across all pages. It uses React’s useNavigate for programmatic routing.   
The header is designed as a reusable component, ensuring consistent styling and functionality throughout the site.

**Footer.jsx**Provides the footer content for the application, which includes copyright information and site credits.   
It is designed to appear on all pages, ensuring consistent branding and information accessibility. Like the header, the footer is a reusable component, simplifying maintenance and updates.

**Home.jsx**Acts as the landing page for the application, introducing users to CookMate.   
It includes interactive elements such as a welcome message, a brief introduction to the features, and a call-to-action button linking to the recipes page.

**Recipes.jsx**Manages the display of all available recipes using a grid layout.   
It integrates a carousel feature for showcasing recipes and allows users to click on a recipe to view more detailed information.   
It dynamically loads data from the recipes.json file.

**RecipeDetail.jsx**Displays comprehensive information about a selected recipe, including ingredient lists, preparation instructions, and a video tutorial.   
It allows users to adjust ingredient quantities based on serving size, enhancing the cooking experience.   
This component is also designed to be reusable, ensuring that any recipe can be displayed with consistent functionality and style.

**Steps.jsx**Provides an interactive cooking guide, walking users through each step of a recipe.   
It includes features like a countdown timer, voice command integration using React Speech Recognition, and a visual progress bar to help users keep track of their cooking.   
The Steps component is reusable, enabling any recipe to be presented in an interactive format.

About.jsxContains information about the CookMate project, its mission, and the team behind it.   
This component helps users understand the purpose of the application and gives credit to the contributors.

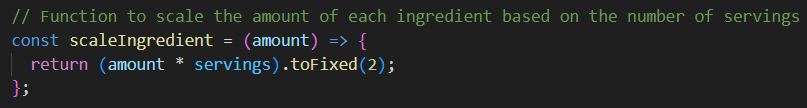
**Key Functions Documentation**

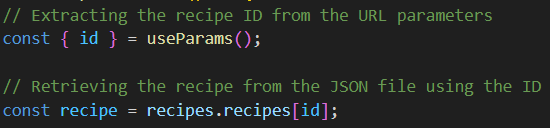
This is an overview of the key functions used in the CookMate project. The functions are central to the application's functionality, including adjusting recipe servings, navigating recipes, and implementing interactive cooking features. Each function is described with its purpose, usage context, and example code where applicable.

* **‘scaleIngredient’**Purpose: Adjust ingredient quantities based on the number of servings the user selects.

Description: This function dynamically scales the amount of each ingredient when the user changes the number of servings on the recipe detail page.

Usage Context: This is used in the RecipeDetail.jsx component to ensure that ingredient quantities are accurate for the selected servings.

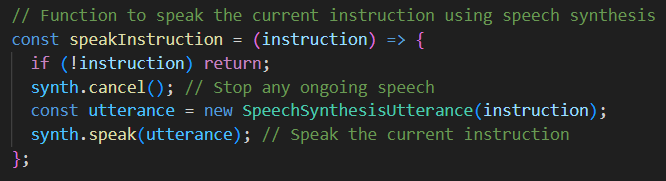


* 

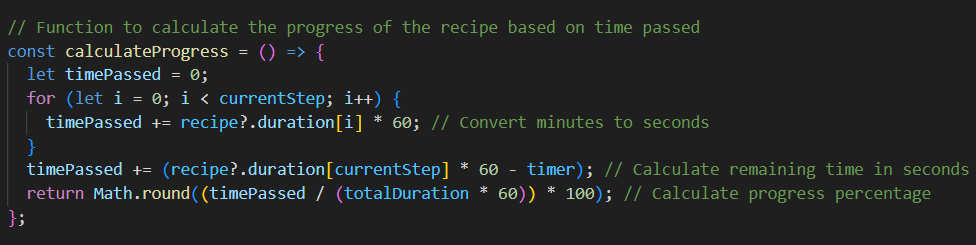
Purpose: Displays detailed information about a specific recipe.  
Description: When a user selects a recipe, this function retrieves the relevant data (ingredients, instructions, and video) from the JSON file and renders it on the RecipeDetail.jsx page.  
Usage Context: This function is central to the RecipeDetail.jsx component, providing all necessary details for the selected recipe.

* **‘speakInstruction’**Purpose: Converts the current instruction into speech using the Web Speech API.

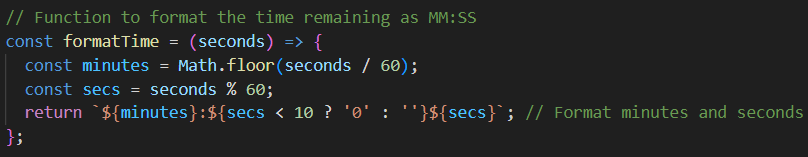
Description: This function creates a new speech synthesis utterance from the instruction and uses the browser's speech synthesis to read it aloud. It also stops any ongoing speech before starting a new one.  
Usage Context: This function is used in the Steps.jsx component to provide audio guidance during the interactive cooking session, allowing for a hands-free cooking experience.



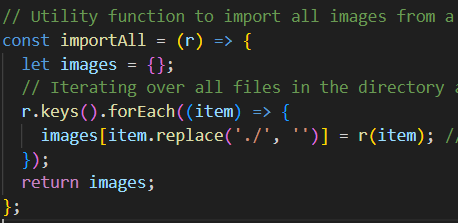
* **‘calculateProgress’**Purpose: Calculates the progress of the recipe based on the time passed for the current step.

Description: This function sums up the time spent on previous steps and the remaining time for the current step to calculate the overall progress as a percentage.  
Usage Context: This function is used in the Steps.jsx component to visually display the progress of the recipe during the interactive cooking session.

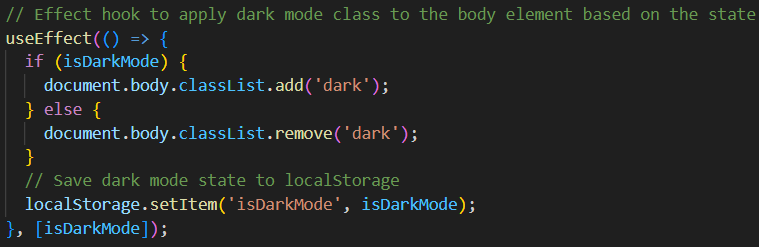
* **‘formatTime’**Purpose: Formats the time remaining as MM

Description: This function takes a time value in seconds and converts it into a readable format, displaying minutes and seconds in MM format.  
Usage Context: This function is used in the Steps.jsx component to display the remaining time for the current step in a user-friendly format.

* **‘importAll’**Purpose: Utility function to import all images from a specified directory.

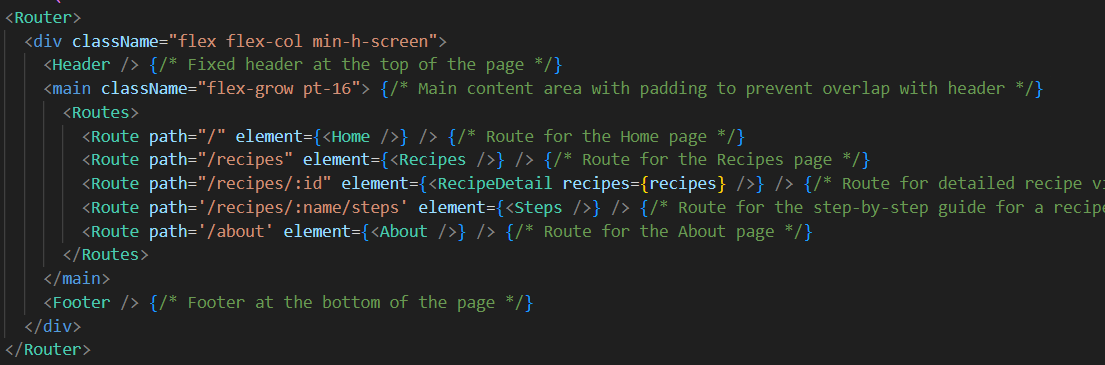
Description: This function iterates over all files in a directory and adds them to an images object. It modifies the file path by removing the leading ./ before storing the image in the object.  
Usage Context: This function is used in the Steps.jsx component to dynamically load and access images for each step in a recipe, making it easier to manage and reference images within the project.  
  


* **Dark Mode Handling with useEffect**Purpose: Manages the application's dark mode by adding or removing a CSS class on the body element and persisting the dark mode state in localStorage.

****Description: This useEffect hook runs whenever the isDarkMode state changes. It adds or removes the dark class from the body element to toggle dark mode. It also stores the current dark mode state in localStorage to persist the user's preference across sessions**.**Usage Context: This function is used in the Header.jsx component to manage the dark mode toggle, ensuring that the user’s preference for light or dark mode is applied and remembered across different sessions.

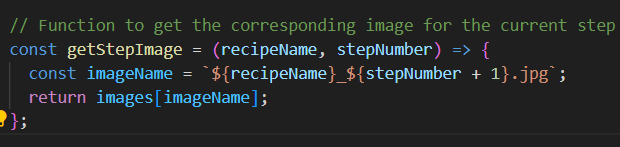
* **Routing Setup**Purpose: Defines the structure and navigation of the application using React Router.

Description: The routing setup is implemented within the App.jsx component using React Router. It includes the Header and Footer components, and dynamically loads different pages based on the URL path. This setup ensures that users can navigate between the Home, Recipes, Recipe Details, Steps, and About pages.  
Usage Context: This routing setup is a core part of the App.jsx component, responsible for rendering different components based on the URL path. It ensures smooth navigation across different sections of the CookMate application. The Header remains fixed at the top, and the Footer stays at the bottom, providing consistent navigation and branding throughout the site.

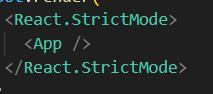
****

* **‘getStepImage’**Purpose: Retrieves the corresponding image for the current step of a recipe.

Description: This function constructs the image file name based on the recipe name and step number, then fetches and returns the image from the preloaded images object.  
Usage Context: This function is used in the Steps.jsx component to dynamically load and display the correct image associated with each step of the recipe during the interactive cooking session. It enhances the user experience by providing visual aids that correspond to the current step in the cooking process.



* **‘React.StrictMode’ Wrapper**Purpose: Provides a development tool that helps identify potential problems in an application by activating additional checks and warnings for its descendants.

Description**:** Wrapping the App component with React.StrictMode activates additional checks and warnings in development, helping to identify potential issues like deprecated APIs or unintended side effects. This is useful for enforcing best practices during the development phase.  
Usage Context: The React.StrictMode is used in the index.js file, wrapping the App component. It ensures that the entire application, including all child components, is rendered in strict mode during development. This does not affect the production build but helps maintain code quality by detecting and warning about potential issues during the development phase.

**Backend Architecture and Features**

The backend of CookMate is designed to efficiently handle recipe data and facilitate seamless interaction with the frontend through a straightforward API.

**Folder Structure Overview**

* app/api/route.js: This is where the API endpoints are defined. This file handles incoming requests and routes them to the appropriate functions.
* lib/mongodb.js: This file manages the connection to the MongoDB database, ensuring that the application can store and retrieve data as needed.

**Controllers and Database Interaction**

* API Routes: The route.js file in the app/api directory is responsible for defining the routes that the frontend will interact with. These routes handle incoming HTTP requests and communicate with the MongoDB database via the logic defined in mongodb.js.
* Database Connection: The mongodb.js file in the lib directory is responsible for establishing a connection to the MongoDB database. This setup allows the backend to perform CRUD (Create, Read, Update, Delete) operations on the recipe data.

**Security Considerations**

* Environment Configuration: The. env.local file is used to store environment-specific variables, such as database connection strings, ensuring sensitive information is kept secure and separate from the codebase.
* Basic Input Validation: Basic input validation is performed within the API routes to ensure that the data received from the frontend is properly formatted and valid before being processed or stored in the database.

**Key Functions Documentation in Backend**

This is an overview of the critical functions used in the Backend project. The functions are central to the application's functionality, including handling recipe data and facilitating seamless interaction with the front end through a straightforward API.  
Each function is described with its purpose, usage context, and example code where applicable.

* **Mongodb.js:**

Purpose:

Provides a utility for establishing and managing a connection to a MongoDB database, ensuring that the connection is efficiently reused across multiple requests.  
Description:

This module handles the connection to a MongoDB database using the MongoClient class from the MongoDB package. It first checks if the MONGODB\_URI environment variable is set; if not, it throws an error, ensuring that the application cannot run without a valid connection string. The connection to the database is managed through the clientPromise, which ensures that the MongoDB client is only instantiated once, even if multiple requests are made. The connect to database function can be called to retrieve a connected database instance, making it easy to perform database operations.  
Usage Context:

This module is used within the backend of the CookMate application to interact with the MongoDB database. The connectToDatabase function is typically called in API route handlers or other backend logic requiring a database connection. This approach optimizes resource usage and improves performance by ensuring that the connection is only established once and reused across requests. The use of logs (e.g., "Log 1", "Log 2", etc.) helps in debugging by providing clear points of reference in the connection process, making it easier to identify where potential issues might occur.

****

* **Route.js**  
  Purpose:

Handles a GET request to retrieve all recipes from the MongoDB database and returns them as a JSON response.

Description:

This function is an API route handler for a GET request within a Next.js application. It begins by establishing a connection to the MongoDB database using the connectToDatabase function. Once connected, it queries the CookMate\_recipes collection to fetch all recipes stored in the database. The retrieved recipes are then returned as a JSON response to the client. If an error occurs at any point during the process, it is caught and logged, and a JSON response with an error message and a 500 status code is returned.

Usage Context:

This code is used in the backend of the CookMate application, specifically for an API route that handles GET requests. When the frontend or another client makes a GET request to this route, this function is executed to retrieve the list of recipes from the database. The logs (e.g., "Log 5", "Log 6", etc.) provide detailed insights into the execution flow, making it easier to debug and identify where issues may arise. This function is essential for serving recipe data to the frontend, enabling users to view available recipes within the CookMate application.

****

**Sources**

**General Development and Documentation:**

* React Documentation: For comprehensive understanding and latest updates in React.  
  <https://legacy.reactjs.org/docs/getting-started.html>
* Tailwind CSS Documentation: To explore more about utility classes and custom configurations.  
  [**https://tailwindcss.com/docs/installation**](https://tailwindcss.com/docs/installation)
* GitHub: For version control and code sharing with your team.  
  [**https://github.com**](https://github.com/)
* chatGPT:  
  <https://openai.com/chatgpt>

**Database Management**

* MongoDB: Official documentation for using MongoDB, especially useful if you’re using a NoSQL database.  
  <https://www.mongodb.com/docs/>
* Mongoose: Documentation for Mongoose, which simplifies MongoDB interactions.  
  <https://mongoosejs.com/docs/>

**Learning and Community Support**

* Stack Overflow: A valuable resource for troubleshooting and community advice.  
  [**https://stackoverflow.com**](https://stackoverflow.com/)
* Reddit: Subreddits like r/reactjs and r/webdev can be great for getting help and discussing ideas.  
  [**https://www.reddit.com/r/reactjs/?rdt=48266**](https://www.reddit.com/r/reactjs/?rdt=48266)

[**https://www.reddit.com/r/webdev/**](https://www.reddit.com/r/webdev/)

**Link to DB**

* MongoDB’s: mongodb+srv://<username>:<password>@cookmate.6pib8.mongodb.net/?retryWrites=true&w=majority&appName=cookmate
* The Backend (Next.js): [**backendcookmate-5llw.vercel.app/api**](https://backendcookmate-5llw.vercel.app/api)