1. App_state of Class

This code represents a "state management" class that can be used in Flutter. State management is a method for maintaining and updating application state by monitoring data changes in an application and notifying all application parts of these changes.

The above code defines a class called `AppState`. This class extends the `ChangeNotifier` class so that when the data in it changes, it notifies listeners (for example Flutter widgets) by calling the `notifyListeners` method. This way, application screens can be updated in response to status changes.

The properties and methods in the `AppState` class are as follows:

1. `loadingSearch`: A flag (boolean value) used while performing a search, waiting for search results.

2. `searchTextFocusNode`: Focus button for a text input field.

3. `searchTextController`: Text controller for a text input field.

4. `selectedBottomNavbarIndex`: The index of the selected item in the footer navigation bar.

5. `selectedCuisines`: List of cuisine types selected by the user.

6. `selectedIngredients`: List of materials added by the user.

7. `results`: List of recipes found by the app.

8. `favorites`: List of recipes that the user has marked as favourites.

9. The `AppState` class has a constructor method. This method takes the user's favorite recipes when the application is started and loads them into the `favorites` list.

10. `addCuisine`, `removeCuisine`: Methods for adding and removing selected cuisine types.

11. `addFavorite`, `removeFavorite`: Methods to add and remove favorite recipes.

12. `updateLoading`: Method to update `loadingSearch` value.

13. `updateResults`: Method to update the `results` list.

14. `clearResult`: Method to clear the `results` list.

15. `updateSelectedNavbarIndex`: Method to update the selected footer navigation bar item.

16. `addIngredients`, `removeIngredients`, `clearIngredients`: Methods for adding, removing and cleaning selected materials.

This piece of code contains the core functionality used to manage states in the application. State management becomes an essential component for efficiently and regularly managing application state in large and complex applications. This piece of code becomes better understandable and valuable when used in a wider context in an application.

2. Recipe_model of Class

This code represents a `RecipeModel` class that can be used in Flutter applications. This class is used to represent recipe data.

Inside the `RecipeModel` class are the following properties:

1. `title`: The title (String) of the recipe.

2. `ingredients`: The list of ingredients used in the recipe (List<String>).

3. `directions`: The preparation method or instructions of the recipe (List<String>).

4. `link`: The link or source (String) of the recipe.

5. `source`: The source (String) of the recipe.

6. `ner`: List of Named Entity Recognition of the recipe (List<String>).

7. `cuisine`: The culinary type (String) of the recipe.

The class's constructor method (`RecipeModel`) provides access to these properties and creates a recipe instance. Also, the class has the ability to convert between JSON data and the class object. There are `toMap` and `fromJson` methods for this.

- `toMap`: Converts class properties to a `Map` data.

- `fromJson`: Takes JSON data and parses it as `Map`, then creates a `RecipeModel` object using this `Map` data.

Also, the `toJson` method converts the `toMap` result to JSON format.

The class also implements the `==` and `hashCode` methods. The `==` method is used when comparing the contents of two `RecipeModel` objects, while the `hashCode` method is used to get a unique hash value of the object.

This `RecipeModel` class can represent data for recipes in an orderly fashion and can help manage recipes within the application. JSON conversion makes this class easy to use in applications such as storing data or exchanging data from the server.

3. Recipe_Manager of Class

This code represents the RecipeManager class, which is a recipe data manager. This class is used to read and search recipes data in the app. Recipes data is stored in a file in JSON format and this class reads and processes data from this file.

Methods inside the RecipeManager class:

readData: It is an asynchronous method that reads recipe data in JSON format. This method takes the dataset from "assets/data/dataset.json", parses the JSON data and converts it into RecipeModel objects. As a result, all recipes are returned as a list.

searchRecipe: It is an asynchronous method that filters recipes by given ingredients (ingredients) and selected cuisine types (selectedCuisines). This method gets all recipes using the readData method. Then it filters the recipes according to the given ingredients and filters once again according to the selected cuisine types. As a result, a list of recipes that meet the appropriate criteria is returned.

This piece of code can be used to provide data management for recipes in the app. Along with the RecipeModel class, it can be used to visualize recipes in the app and allow the user to search for recipes by selecting specific ingredients and cuisine types. Storing the data in a file in JSON format makes it easy to modify and update the dataset. The RecipeManager class is a useful class for managing operations such as retrieving and filtering a dataset.

4. Favorite_Manager of Class

This code represents a class called FavoriteManager. This class contains static methods used to manage favorite recipes. SharedPreferences are used to keep and store favorite recipes within the app.

Static methods in FavoriteManager class:

favoriteKey: Represents the key value to be used to store favorite recipes. This value is used in the area where SharedPreferences and favorite recipes are stored.

getFavoriteRecipes: It is a static asynchronous method that takes data from the storage area of favorite recipes and returns it as a RecipeModel list. Using SharedPreferences, it takes JSON data of favorite recipes, analyzes this data and converts it into RecipeModel objects.

addFavorite: Static asynchronous method used to add a new favorite recipe. This method first gets the current favorite recipes using the getFavoriteRecipes method, adds the new recipe to this list, and then saves the updated favorite recipe list to SharedPreferences in JSON format.

removeFavorite: Static asynchronous method used to delete favorite recipes. This method first retrieves the current favorite recipes using the getFavoriteRecipes method, removes the specified recipe from the list and saves the updated favorite recipe list back to SharedPreferences in JSON format.

This code is used within the app to allow users to add, remove and manage their favorite recipes. SharedPreferences are an easy and common method for persistently storing small amounts of data (key-value pairs). Therefore, it is preferred as a convenient method to save the user's favorite recipes on their device.

5.Screens of Class

Sure, here are summaries of the given code fragments:

5.1 `Search` Class:

This class represents a search screen where the user can search for recipes by ingredients and cuisine types. Users can add ingredients in the search box, see ingredients with chips, and filter recipes based on selected ingredients and cuisine types. It also displays cards listing search results and provides interaction to add or remove favorite recipes.

5.2 `Recipe` (Recipe) Class:

This class represents a screen that shows the details of a selected recipe. The recipe's title, ingredients, instructions, and recipe link are visualized. It also provides functionality to copy the link of the recipe to the clipboard.

5.3 `Home` Class:

This class represents the main screen of the application. Creates a widget with top navigation bar, bottom navigation bar and content areas. Via the bottom navigation bar, users can navigate between the "Search" and "Favorite" screens. At the same time, the `Consumer` widget, which monitors the application status, listens for status changes and makes instant updates on the screen.

5.4 `Favorite` Class:

This class represents a screen where the user can view their favorite recipes. It visualizes the list of favorite recipes and users can interact on the recipes. It also represents recipe data, used in conjunction with the `RecipeModel` class.

These four classes together provide the basic features of a recipe app. Users can search for recipes within the application, see their details, mark favorite recipes and view their favorite recipes. This way, users can easily find and manage recipes in the app.

6. Main of Class

This snippet exposes the `MyApp` class, which represents the main entry point of a Flutter application. The `MyApp` class makes the application launch and display the home screen (`Home`). It also sets the theme and state management for the application.

Important elements found inside the `MyApp` class:

1. `ChangeNotifierProvider`: A `ChangeNotifierProvider` created with the `AppState` class is used to manage application state and track state changes across the entire widget tree. As part of the `ChangeNotifierProvider`, the `AppState` object represents the state of the application and passes this state to all child widgets.

2. `MaterialApp`: This widget is the core widget of a Flutter application and manages the theme and navigation throughout the application. `MaterialApp` specifies the application title (`title`), debug mode (`debugShowCheckedModeBanner`), theme settings and home screen (`home`).

3. `theme`: Sets the general theme settings for the application. In this code snippet, a light theme is used. `primaryColor` specifies the color `Colors.orange` as the primary color. Also, the `AppBar` theme is customized and set to `Colors.orange`.

4. `home: const Home()`: Specifies the `Home` widget as the home screen of the application. The `Home` widget is a widget through which the user can navigate and search on the home screen of the application, representing the `Home` class.

This code starts the Flutter application starting with the `MyApp` class, manages the application state and sets the general theme settings for the application. Via `ChangeNotifierProvider` it is ensured that changes in application state are reflected to all child widgets, thus keeping the data within the application up to date.