01 Instruction Manual for the basic structure of the application

Creating an instruction manual for a beginner to understand and replicate the Shopping List application involves breaking down each section of the code and explaining the purpose and function of each part. Let's start with the HTML.

HTML Explained

HTML (HyperText Markup Language) is the standard markup language for creating web pages. It describes the structure of a web page and is used here to create the user interface of your Shopping List application.

1. **Document Type Declaration**: `<!DOCTYPE html>` tells the web browser that this is an HTML5 document.

2. **`<html>` Element**: This is the root element of an HTML page.

3. **`<head>` Section**: Contains meta-information about the document, links to CSS files, and other resources.
   - `<meta charset="UTF-8">` ensures your document's character encoding is set to UTF-8 which includes most characters from all known languages.
   - `<meta name="viewport" content="width=device-width, initial-scale=1.0">` makes your web application responsive, meaning it will adjust to fit different device screens.
   - `<link>` tags link to external resources like icons and CSS files.

   - `<link href="https://fonts.googleapis.com/css2?family=Roboto..."></link>` is used to import the "Roboto" font from Google Fonts.

4. **`<body>` Section**: Contains the content of the web page that will be visible to the user.
   - **Input Container**: A division that contains input fields where users can enter item names and prices.
   - **List Container**: An empty division that will later be populated with the shopping list items.
   - **Script Tag**: Includes the JavaScript file (`index.js`) which contains the logic of the application.

JavaScript Explained

JavaScript is a scripting language that allows you to implement complex features on web pages.

1. **Firebase Initialization**: Sets up Firebase, a backend-as-a-service that provides features like real-time databases.
   - `initializeApp` configures the connection to your Firebase project using the provided settings.
   - `getDatabase` and other imported functions are used to interact with the Firebase Realtime Database.

2. **Database Reference**: `const dbRef = ref(database, "shoppingList");` sets up a reference to the "shoppingList" node in your Firebase database.

3. **Data Retrieval and Sorting**: The `onValue` listener reacts to changes in the Firebase data, retrieving the list, sorting it by category, and organizing it into groups.

4. **DOM Manipulation**: The `createElements` function dynamically creates HTML elements based on the data and appends them to the DOM, making them visible on the webpage.

5. **Event Listeners**: The `addEventListener` method listens for user actions (like button clicks) and executes code in response.

CSS Explained

CSS (Cascading Style Sheets) is used to style and layout web pages.

1. **General Styling**: The `*` selector is used to apply a box-sizing rule to all elements, which affects the way width and height are calculated.

2. **Responsive Design**: Media queries (`@media`) apply different styles based on the device's screen size. For instance, the `.number-of-items-input` width is adjusted for devices with a screen width of 500px or less.

3. **Styling Specific Elements**: Other CSS rules target specific classes or IDs, defining how each element should look and behave visually.

Firebase Database Structure Explained

The Firebase Realtime Database is a cloud-hosted NoSQL database. Data is stored as JSON and synchronized in real-time to every connected client. In your database:
- Each item in the shopping list is an object with keys such as `item`, `fixedPrice`, `actualPrice`, `isPurchased`, and `category`.
- The unique key for each item (e.g., `-NkwET4mO...`) is automatically generated by Firebase.

Putting It All Together

The instruction manual would walk a beginner through setting up their HTML file, writing the JavaScript logic, styling the page with CSS, and configuring Firebase. It would explain each section of the code and how it contributes to the overall functionality of the application, such as entering items, displaying the shopping list, and interacting with the Firebase database.

The manual should also provide explanations on how to set up a Firebase project and how to connect it to the web application, including detailed steps on where to find the necessary Firebase configuration and how to insert it into the code.

For each concept used (e.g., variables, functions, event listeners, etc.), the manual should provide a simple explanation and potentially link to further resources for a deeper understanding.

By following the manual, a beginner should be able to replicate the application, understand the role of each part of the code, and have a foundation for exploring more advanced topics.

02 JS Code explained step-by-step

Certainly! Here's a detailed explanation of the provided JavaScript code, broken down by blocks:

Initial Comments and Metadata

```javascript
/**!SECTION Working Schedule
 * Last working as of 12/05/23 at 11:52AM
 */

/**!SECTION
 * ## 01 Purpose of the application
 * Shopping List
 *
 * ## 02 Functionality
 * 1. The user enters an item in an input box
 * 2. The user clicks a button that dynamically creates a checkbox with a label of the item just entered and an empty input field.
 * 3. Once the shopping list is complete, the user will add the price of the item purchased and check the checkbox to indicate the item has been purchased.
 * 4. The price of each item is to be stored and added in a different variable for later use
 */
```

These are comments for developers reading the code. They describe the last time the code was confirmed to be working and outline the purpose and core functionality of the application.

Firebase Setup

```javascript
// Install Firebase
import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.15.0/firebase-
app.js";
import {
  getDatabase,
  ref,
  push,
  set,
  onValue,
  update,
} from
"https://www.gstatic.com/firebasejs/9.15.0/firebase-
database.js";
```

These lines import functions from the Firebase JavaScript SDK, which are needed to initialize the app and interact with Firebase Realtime Database.

```javascript
const appSettings = {
  databaseURL: "https://shoppinglist-b0a3b-default-
rtdb.firebaseio.com/",
};
```

```
const app = initializeApp(appSettings);
const database = getDatabase(app);
```

Here, the `initializeApp` function is called with your Firebase project's settings, initializing the Firebase app. `getDatabase` gets a reference to the database linked with your Firebase app.

Realtime Database Interaction

```javascript
const dbRef = ref(database, "shoppingList");

onValue(dbRef, (snapshot) => {
  // Code to read data and update the DOM
});
```

`ref` creates a reference to the "shoppingList" node in your Firebase database. `onValue` sets up a real-time listener that triggers a callback function every time the "shoppingList" node changes.

Dynamic DOM Manipulation

```javascript
const listContainerDIV = document.querySelector("#list-container");
```

This selects the HTML element with the ID `list-container`, which will contain the shopping list.

```javascript
const createElements = (
  itemValue,
```

```
  fixedPriceValue,
  uniqueKey,
  actualPrice,
  isPurchased,
  numberOfItems,
  categoryContainer
) => {
  // Code to create new elements and append to the DOM
};
```

`createElements` is a function that dynamically creates HTML elements for each shopping list item, including text inputs and checkboxes, and appends them to the DOM.

Event Listener for Adding Items

```javascript
btnEl.addEventListener("click", function () {
  // Code to handle the click event
});
```

This sets up an event listener on the "Enter Items" button. When clicked, it gathers input values, creates new list elements, and stores the data in Firebase.

Firebase Data Update Function

```javascript
function updateItemInFirebase(itemId, updateObject) {
  // Code to update items in Firebase
}
```

`updateItemInFirebase` is a helper function that updates a specific item in the Firebase database with new values.

Concepts Explained
- **Variables**: Containers for storing data values. In this code, variables hold elements from the DOM, Firebase settings, and more.
- **Functions**: Blocks of code designed to perform particular tasks; `createElements` and `updateItemInFirebase` are examples.
- **Event Listeners**: They allow you to run JavaScript code in response to user events like clicks.
- **Callbacks**: Functions passed as arguments to other functions to be executed at a later time.
- **Promises**: Represent the future result of an asynchronous operation. They allow you to write code that assumes a value will be returned eventually, letting the runtime handle the details.
- **DOM (Document Object Model)**: A programming interface for web documents. It represents the page so that programs can change the document structure, style, and content.
- **Import**: The `import` statement is used to read in functions or variables from external files or modules.

The code provided is a part of a web application that interacts with Firebase to read and write data, dynamically updates the webpage based on this data, and listens for user actions to modify the shopping list accordingly.

JS Code Below:

```
/**!SECTION Working Schedule
 * Last working as of 12/05/23 at 11:52AM
 */

/**!SECTION
 * ## 01 Purpose of the application
Shopping List

## 02 Functionality
1. The user enters an item in an input box
2. The use clicks a button that dynamically creates a
checkbox with a label of the item just entered and an
empty input field.
3. Once the shopping list is complete, the user will add
the price of the item purchased and check the checkbox
to indicate the item has been purchased.
4. The price of each item is to be stored and added in a
different variable for later use
 */
// Install Firebase
import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.15.0/firebase-
app.js";
import {
  getDatabase,
  ref,
  push,
  set,
  onValue,
  update,
```

```
} from
"https://www.gstatic.com/firebasejs/9.15.0/firebase-
database.js";

const appSettings = {
  databaseURL: "https://shoppinglist-b0a3b-default-
rtdb.firebaseio.com/",
};

const app = initializeApp(appSettings);
const database = getDatabase(app);

// Create a reference to the database
const dbRef = ref(database, "shoppingList");

onValue(dbRef, (snapshot) => {
  listContainerDIV.innerHTML = ""; // Clear existing
items
  const items = snapshot.val();

  if (items) {
    const itemsArray = Object.keys(items).map((key) =>
({
      ...items[key],
      key,
    }));
    itemsArray.sort((a, b) =>
a.category.localeCompare(b.category));

    const categoryGroups = {};

    // Group items by category
```

```
    itemsArray.forEach((item) => {
      if (!categoryGroups[item.category]) {
        categoryGroups[item.category] = [];
      }
      categoryGroups[item.category].push(item);
    });

    // Create a container for each category and append
items
    Object.keys(categoryGroups).forEach((category) => {
      const categoryDiv = document.createElement("div");
      categoryDiv.className = "category-container";
      const categoryTitle =
document.createElement("h4");
      categoryTitle.className = "category-title";
      categoryTitle.textContent = category;
      categoryDiv.appendChild(categoryTitle);

      categoryGroups[category].forEach((item) => {
        createElements(
          item.item,
          item.fixedPrice,
          item.key,
          item.actualPrice,
          item.isPurchased,
          item.numberOfItems,
          categoryDiv // Pass the category container
instead of category name
        );
      });

      listContainerDIV.appendChild(categoryDiv);
```

```
    });
  }
});

// Delcare variables for input element and button to
input intems in the empty list
const inputEl = document.querySelector("#input-el");
const inputFixedPriceEl =
document.querySelector("#input-price-el");
const btnEl = document.querySelector("#enter-items");
const inputLabel = document.querySelectorAll(".input-
label");
// Array of objects to store details about shopping list
const listItemsArray = [
  {
    id: 0,
    item: "",
    price: 0,
    fixedPrice: 0,
    isPurchased: false,
  },
];
const uniqueId = `item-${listItemsArray.length + 1}`;
const listContainerDIV = document.querySelector("#list-
container");
listContainerDIV.classList.add("list-container");

// Event listener for the add button
btnEl.addEventListener("click", function () {
  // Trim values to remove unnecessary whitespace
  const inputElValue = inputEl.value.trim();
```

```
  const inputFixedPriceElValue =
inputFixedPriceEl.value.trim();
  const selectedCategory =
document.getElementById("categoryDropdown").value; //
Get the selected category

  // Only proceed if both fields have values
  if (inputElValue && inputFixedPriceElValue) {
    createElements(inputElValue,
inputFixedPriceElValue); // Create list elements

    const newItem = {
      item: inputElValue,
      fixedPrice: inputFixedPriceElValue,
      actualPrice: 0, // Initialize with a default value
      numberOfItems: 0, // Initialize with a default
value
      isPurchased: false,
      // category: selectedCategory // Add the selected
category here
      categoryContainer, // Changed parameter
    };
    // When you create a new item and push it to the
database:
    push(dbRef, newItem).then((snapshot) => {
      const uniqueKey = snapshot.key; // This is the
unique identifier generated by Firebase
      // Store this uniqueKey in your item object or set
it as a data-id attribute on your HTML element
    });

    // Clear input fields after adding the item
```

```
    inputEl.value = "";
    inputFixedPriceEl.value = "";
  } else {
    // Alert if fields are incomplete
    alert("Please fill in both the item name and the
estimated price.");
  }
});

inputEl.value = ""; //Cledar input field after user
clicks the button
// Create a function to dunamically create and append
elements to the DOM
/**
 * Creates and appends elements to the list container.
 *
 * @param {string} itemValue - The value of the list
item.
 * @param {string} fixedPriceValue - The value of the
fixed price.
 */

// Function to dynamically create and append elements to
the DOM
/**
 * Creates and appends elements for a shopping list
item.
 *
 * @param {string} itemValue - The value of the item.
 * @param {number} fixedPriceValue - The fixed price of
the item.
```

```
 * @param {string} uniqueKey - The unique key of the
item.
 * @param {number} actualPrice - The actual price of the
item.
 * @param {boolean} isPurchased - Indicates whether the
item is purchased or not.
 * @param {HTMLElement} categoryContainer - The
container element for the category.
 */
const createElements = (
  itemValue,
  fixedPriceValue,
  uniqueKey,
  actualPrice,
  isPurchased,
  numberOfItems,
  categoryContainer // Changed parameter
) => {
  // Create a new div to hold the item details
  const itemContainer = document.createElement("div");
  itemContainer.className = "list-item-container";

  // Inside your createElements function
  const categoryDiv = document.createElement("div");
  // categoryDiv.textContent = "Category: " + category;
  categoryDiv.textContent = "";
  itemContainer.appendChild(categoryDiv);

  // Input for displaying the item name
  const listItem = document.createElement("input");
  listItem.type = "text";
  listItem.className = "list-item";
```

```
  listItem.value = itemValue;
  listItem.setAttribute("data-id", uniqueKey);
  listItem.disabled = isPurchased;

  // Ensure fixedPriceValue, actualPrice, and
numberOfItems are correctly parsed as numbers
  fixedPriceValue = parseFloat(fixedPriceValue) || 0;
  actualPrice = parseFloat(actualPrice) || 0;
  numberOfItems = parseInt(numberOfItems, 10) || 1;

  // Input for displaying the fixed price
  const fixedPriceDisplay =
document.createElement("input");
  fixedPriceDisplay.type = "number";
  fixedPriceDisplay.className = "fixed-price";
  fixedPriceDisplay.value = (
    parseFloat(fixedPriceValue) * numberOfItems
  ).toFixed(2);
  fixedPriceDisplay.disabled = true;

  // Input for entering the actual price
  const actualPriceInput =
document.createElement("input");
  actualPriceInput.type = "number";
  actualPriceInput.className = "actual-price-input";
  actualPriceInput.placeholder = "Actual Price";
  actualPriceInput.value = actualPrice || "";
  actualPriceInput.disabled = isPurchased;

  // Checkbox for marking the item as purchased
  const isPurchasedCheckbox =
document.createElement("input");
```

```
  isPurchasedCheckbox.type = "checkbox";
  isPurchasedCheckbox.className = "is-purchased-
checkbox";
  isPurchasedCheckbox.checked = isPurchased;
  isPurchasedCheckbox.setAttribute("data-id",
uniqueKey);

  // Input for entering the number of items
  const numberOfItemsInput =
document.createElement("input");
  numberOfItemsInput.type = "number";
  numberOfItemsInput.className = "number-of-items-
input";
  numberOfItemsInput.placeholder = "Enter number of
items";
  numberOfItemsInput.value = numberOfItems || 1;
  numberOfItemsInput.disabled = isPurchased;

  // Event listeners for updating Firebase on changes
  actualPriceInput.addEventListener("change", (event) =>
{
    event.stopPropagation();
    const updatedPrice = parseFloat(event.target.value)
|| 0;
    updateItemInFirebase(uniqueKey, { actualPrice:
updatedPrice });
  });

  // Event listener for when the number of items is
updated
  numberOfItemsInput.addEventListener("change", (event)
=> {
```

```
    event.stopPropagation(); // Stop the event from
bubbling up the DOM tree

    // Parse the updated number of items as an integer
    const updatedNumberOfItems =
parseInt(event.target.value, 10) || 1;

    // Calculate the new total fixed and actual prices
    const updatedTotalFixedPrice = (
      fixedPriceValue * updatedNumberOfItems
    ).toFixed(2);
    const updatedTotalActualPrice = (
      actualPrice * updatedNumberOfItems
    ).toFixed(2);

    // Update the DOM elements with the new values
    fixedPriceDisplay.value = updatedTotalFixedPrice;
    actualPriceInput.value = updatedTotalActualPrice;

    // Get the item's unique ID from the data attribute
    const itemId = listItem.getAttribute("data-id");

    // Reference the specific item in the database
    const itemRef = ref(database,
`shoppingList/${itemId}`);

    // Update Firebase with the new total fixed and
actual prices
    update(itemRef, {
      fixedPrice: updatedTotalFixedPrice,
      actualPrice: updatedTotalActualPrice,
      numberOfItems: updatedNumberOfItems,
```

```
    }).catch((error) => {
      console.error("Error updating in Firebase",
error);
    });
  });

  isPurchasedCheckbox.addEventListener("change", (event)
=> {
    event.stopPropagation();
    const isChecked = event.target.checked;
    listItem.disabled = isChecked;
    fixedPriceDisplay.disabled = isChecked;
    actualPriceInput.disabled = isChecked;
    numberOfItemsInput.disabled = isChecked;
    updateItemInFirebase(uniqueKey, { isPurchased:
isChecked });
  });

  // Append all elements to the item container div
  itemContainer.appendChild(listItem);
  itemContainer.appendChild(actualPriceInput);
  itemContainer.appendChild(numberOfItemsInput);
  itemContainer.appendChild(fixedPriceDisplay);
  itemContainer.appendChild(isPurchasedCheckbox);

  // Append the item container to the main list
container in the DOM
  listContainerDIV.appendChild(itemContainer);

  // Append the item container to the passed category
container
  categoryContainer.appendChild(itemContainer);
```

```
};

// Helper function to update an item in Firebase
function updateItemInFirebase(itemId, updateObject) {
  const itemRef = ref(database,
`shoppingList/${itemId}`);
  update(itemRef, updateObject).catch((error) => {
    console.error("Error updating item in Firebase",
error);
  });
}
```