Documentation for Authentication and Profile Management Code

Overview

This documentation provides an overview of key functions and code logic within the authentication module and `ProfilePage` component. The code handles user profile management, phone number verification, password reset, and name change functionalities for an application using Firebase and React.

1. Authentication Functions

1.1. `sendVerificationCode`

Purpose: Sends a phone verification code to the user's phone number.

- **Parameters**:
  - `phoneNumber` (string): The phone number to which the verification code will be sent.
- **Logic**:
  - Uses Firebase's `RecaptchaVerifier` for bot protection.
  - Uses `PhoneAuthProvider` to initiate phone number verification.
  - Returns a `verificationId` if successful.
- **Error Handling**: Logs errors to the console if the process fails.

Example Usage:

javascript
Copy code
```
const verId = await sendVerificationCode("+1234567890");
```

1.2. `verifyAndUpdatePhoneNumber`

Purpose: Verifies the received OTP and updates the user's phone number in both authentication and Firestore.

- **Parameters**:
  - `verificationCode` (string): The OTP entered by the user.
  - `verificationId` (string): The verification ID returned from `sendVerificationCode`.
- **Logic**:
  - Uses `PhoneAuthProvider.credential()` to create credentials.
  - Updates the phone number in Firebase Auth and Firestore.

- **Error Handling**: Logs errors to the console and shows an error message on failure.

Example Usage:

javascript
Copy code
```
await verifyAndUpdatePhoneNumber("123456", "verification-id-example");
```

1.3. `resetPassword`

Purpose: Sends a password reset email to the user.

- **Parameters**:
  - `email` (string): The user's email address.
- **Logic**:
  - Sends a POST request to the server endpoint `/api/resetPass`.
  - Alerts the user on success or logs errors if the request fails.
- **Error Handling**: Logs errors to the console if an exception occurs during the fetch operation.

Example Usage:

javascript
Copy code
```
await resetPassword("user@example.com");
```

1.4. `changeDisplayName`

Purpose: Updates the display name of the currently authenticated user.

- **Parameters**:
  - `newDisplayName` (string): The new display name for the user.
- **Logic**:
  - Calls `user.updateProfile()` to change the display name in Firebase Auth.
  - Updates the user's profile in Firestore with the new name.
- **Error Handling**: Logs errors to the console if updating fails.

Example Usage:

javascript
Copy code
```
await changeDisplayName("New Name");
```

1.5. `logoutUser`

Purpose: logs the current user out.

- **Parameters**:
  - None
- **Logic**:
  - Calls `signOut(auth)` to log the current user out
- **Error Handling**: Logs errors to the console if updating fails.

1.6. `handleAccountCreation`

Purpose: Handles gathering data from input field and sends `createuser` api request

- **Parameters**:
  - None (gets global parameters from the HTML page)
- **Logic**:
  - Gets the current user login information.
  - Calls the /api/createUser and jsonifys input fields
  - Sends a success or error message based on what the api route returned
- **Error Handling**: Logs errors to the console if the process fails.

Example Usage:

javascript
```
onClick={handleAccountCreation}
```

1.7. `handleSignIn`

Purpose: Handles signin in user.

- **Parameters**:
  - None (gets global parameters from the HTML page)
- **Logic**:
  - Gets the user login information.
  - Calls signInsUser
  - Get user's department
  - Automatically forwards to user's department page
- **Error Handling**: Logs errors to the console if the process fails.

Example Usage:

javascript
```
<button className={styles.submit} onClick={handleSignIn}>
```

2. ProfilePage Component (`profilePage.tsx`)

Purpose: A React component that manages the display and interaction of user profile information.

Key Features:

- **Displays user information**: Name, email, role, phone number, and departments.
- **Fetches employee data**: Uses `getEmployeeProfile` to load profile details.
- **Handles authentication state**: Checks if the user is authenticated and redirects if not.
- **Phone number update**: Sends and verifies phone numbers through `sendVerificationCode` and `verifyAndUpdatePhoneNumber`.
- **Password reset**: Triggers `resetPassword()` to send a password reset email.
- **Sign-out functionality**: Provides a button for users to sign out.

Key State Variables:

- `employeeData`: Stores the user's profile data.
- `departments`: Stores department names the user is authorized to access.
- `phoneNumber`, `verificationCode`, `verificationId`: Manage phone number verification.
- `isSignedIn`: Indicates if a user is currently authenticated.

Key Hooks and Logic:

- **`useEffect` for authentication state**: Redirects unauthenticated users to the login page.
- **`useEffect` for error messages**: Clears error messages after a timeout.
- **`handleVerificationCode`**: Triggers sending of the verification code for phone updates.

Component Structure:

jsx
Copy code
```
<div className="profile-container">
  <h1>{employeeData.name}</h1>
  {/* Additional user details */}
  <button onClick={() => resetPassword(employeeData.email)}>Change
Password</button>
  {/* Phone number update form */}
  {isSignedIn && <button onClick={handleSignOut}>Sign Out</button>}
</div>
```

3. Design Principles Used

- **Error Handling**: Comprehensive use of `try/catch` blocks for robust error logging.
- **Modular Design**: Functions like `sendVerificationCode` and `resetPassword` are modular, making them reusable and testable.
- **User Experience**: Interactive components provide alerts and real-time feedback, enhancing the user experience.

Conclusion

The provided code integrates Firebase authentication services seamlessly for user management, including phone verification, password resets, and profile updates. It ensures a secure and user-friendly experience through clear error messages, alerts, and validation logic.

Documentation for AIButton Component

Overview

This documentation provides an in-depth overview of the `AIButton` component's key functionalities, logic, and structure. The `AIButton` is a React component designed to enhance user interaction by offering summarization and chatbot functionalities integrated with document-based AI services.

---

1. Component Purpose

The `AIButton` allows users to:

- Interact with an AI chatbot.
- Summarize selected documents.
- Toggle between "Summarize" and "Chat" modes.
- Display an interactive card with a responsive UI.

The component supports PDF summarization using `pdfjs-dist` for text extraction and communicates with an API endpoint for generating document summaries.

---

2. Key Features and Functions

2.1. Modes

- **Summarize Mode**: Allows users to select a document and generate a summarized version.
- **Chat Mode**: Provides a text input to interact with an AI chatbot (chat functionality placeholder).

2.2. Functions

toggleCard

- **Purpose**: Toggles the visibility of the card.

- **Logic**: Flips the `isCardVisible` state.

Example Usage:
javascript
Copy code
```
<button onClick={toggleCard}>Toggle Card</button>
```

handleModeToggle

- **Purpose**: Switches between "Summarize" and "Chat" modes.
- **Parameters**:
    - `selectedMode` (string): The selected mode ('summarize' or 'chat').
- **Logic**: Updates the `mode` state and clears the `inputValue`.

Example Usage:
javascript
Copy code
```
handleModeToggle('chat');
```

handleInputChange

- **Purpose**: Updates the `inputValue` state as the user types.
- **Parameters**:
    - `e` (React.ChangeEvent<HTMLInputElement>): The event triggered by user input.

Example Usage:
javascript
Copy code
```
<input onChange={handleInputChange} />
```

extractTextFromPdf

- **Purpose**: Extracts text content from a PDF document.
- **Parameters**:
    - `arrayBuffer` (ArrayBuffer): The buffer of the PDF file.
- **Returns**: A string containing the extracted text.
- **Logic**:
    - Uses `pdfjsLib` to parse and extract text from each page.

Example Usage:
javascript
Copy code
```
const text = await extractTextFromPdf(arrayBuffer);
```

handleFileSelect

- **Purpose**: Updates the selected file for summarization.
- **Parameters**:
  - `file` (SummarySearchResult): The selected file object.
- **Logic**: Updates `fileSelectedForSummary` and resets `summaryContent`.

Example Usage:
javascript
Copy code
```
handleFileSelect(file);
```

handleSummarizeClick

- **Purpose**: Initiates the summarization process for the selected file.
- **Logic**:
  - Downloads the file.
  - Extracts text content using `extractTextFromPdf`.
  - Sends a POST request to the `/api/summarize` endpoint.
  - Displays the summary or an error message.
- **Error Handling**: Alerts users for missing files and handles fetch errors gracefully.

Example Usage:
javascript
Copy code
```
<button onClick={handleSummarizeClick}>Summarize</button>
```

-

useEffect for Escape Key

- **Purpose**: Closes the card when the Escape key is pressed.
- **Logic**:
  - Adds and removes an event listener for `keydown`.
- **Example Usage**: Automatically triggered during component lifecycle.

---

3. Component Structure

Key State Variables

- `isCardVisible`: Toggles card visibility.
- `mode`: Stores the current mode ('summarize' or 'chat').
- `inputValue`: Stores the text input value for the chat mode.
- `fileSelectedForSummary`: Stores the file selected for summarization.
- `summaryContent`: Stores the generated summary.
- `loading`: Indicates whether the summarization process is in progress.

JSX Structure

jsx
Copy code
```
<div>
  {/* Circle Button */}
  {!isCardVisible && (
    <button className="circle-button" onClick={toggleCard}>
      <img className="ai-logo" src="/images/ollamaLogo.png" />
    </button>
  )}

  {/* Card */}
  {isCardVisible && (
    <div className="card">
      <div className="tab-options">
        <button className="card-close-button"
onClick={toggleCard}>×</button>
      </div>
      <div className="mode-toggle">
        <button onClick={() =>
handleModeToggle('summarize')}>Summarize</button>
        <button onClick={() => handleModeToggle('chat')}>Chat</button>
      </div>
      <div className="content-display">
        <SpinnerDiamond enabled={loading} />
        <ReactMarkdown>{summaryContent}</ReactMarkdown>
      </div>
      <div className="input-area">
        {mode === 'summarize' ? (
          <>
```

```
            <SearchBarAI paths={paths} onFileSelect={handleFileSelect}
/>
            <button className="action-button"
onClick={handleSummarizeClick}><FaArrowCircleUp /></button>
          </>
        ) : (
          <>
            <input type="text" value={inputValue}
onChange={handleInputChange} />
            <button className="action-button">Send</button>
          </>
        )}
      </div>
    </div>
  )}
</div>
```

---

4. Design Principles

Error Handling

- Try/catch blocks provide robust error reporting.
- User-friendly alerts handle invalid states (e.g., no file selected).

Modular Design

- Functions like `extractTextFromPdf` and `handleSummarizeClick` are reusable and testable.

User Experience

- Interactive UI with real-time feedback, including spinners for loading states.
- Keyboard shortcuts enhance accessibility.

---

5. Example Usage

javascript

Copy code
```
<AiButton paths={['/path/to/files']} />
```

---

6. Conclusion

The `AIButton` component provides a modular and user-friendly interface for interacting with AI-powered summarization and chat functionalities. With clear state management and robust error handling, the component seamlessly integrates with document-based AI workflows.

Design: File List Management

Overview

This document describes the `FileList` component, a React utility for managing, displaying, and interacting with files stored in a Firebase Firestore database and Firebase Storage. It provides features such as sorting, searching, file selection, thumbnail generation, deletion, and sharing.

System Components

Firebase Setup

- Utilizes Firebase Firestore for storing metadata about files.
- Firebase Storage is used to store and retrieve file data.
- Firebase Authentication manages user authentication.

Core Functionalities

1. **File Retrieval:** Fetches file metadata and URLs from Firebase Firestore.
2. **Thumbnails:** Generates thumbnails for PDF and image files.
3. **Search & Sort:** Provides tools to search files by name and sort by metadata fields.
4. **Selection & Deletion:** Enables file selection for deletion or sharing.
5. **Role-Based Access:** Integrates admin checks for privileged actions.
6. **File Sharing:** Opens a modal to share selected files.

Detailed Code Breakdown

1. Fetching Files

Function: `fetchFiles`

- **Purpose:** Retrieves file metadata from Firestore and processes it for display.
- **Process:**
    - Queries Firestore for file documents.
    - Calls `processFiles` to generate file thumbnails and metadata.
    - Updates state variables `files` and `filteredFiles` with results.

---

2. Processing File Data

Function: `processFiles`

- **Purpose:** Processes Firestore query results into structured file data.
- **Process:**
    - Retrieves file metadata and tags from Firestore.
    - Generates thumbnails for PDFs and images.
    - Returns a structured array of file data objects.

---

3. Thumbnail Generation

Function: `generatePDFThumbnail`

- **Purpose:** Creates a thumbnail for the first page of a PDF.
- **Process:**
    - Renders the PDF's first page onto a canvas.
    - Converts the canvas content to a data URL for use as a thumbnail.

---

4. Searching Files

State: `searchQuery`

- **Purpose:** Filters files based on the search query.
- **Process:** Matches the file name with the lowercased search query and updates the `filteredFiles` state.

---

5. Sorting Files

Functions: `handleSortFieldClick`, `sortFiles`

- **Purpose:** Sorts files by selected metadata fields.
- **Process:**
    - Updates the sorting field and direction.
    - Sorts files by timestamp, user, or other metadata fields.

---

6. File Selection

Function: `handleFileSelect`

- **Purpose:** Toggles the selection of files by their IDs.
- **Process:** Updates the `selectedFiles` state by adding or removing file IDs.

---

7. File Deletion

Function: `handleDelete`

- **Purpose:** Deletes selected files from Firestore and Firebase Storage.
- **Process:**
    - Confirms the action with the user.
    - Deletes selected files and updates the file list.

---

8. File Sharing

Function: `openShareModal`

- **Purpose:** Opens a modal to share selected files.
- **Process:** Ensures at least one file is selected before opening the sharing modal.

UI Integration

File Display Modes:

- `list`: Displays files in a list format.
- `grid`: Displays files in a grid layout.
- `horizontal`: Horizontal layout for files.

Sort Options:

- Fields include upload timestamp, user display name, etc.

Search Bar:

- Allows users to filter files by name.

Share Modal:

- Triggers a modal for sharing selected files.

Overview of ShareFileModal Component

The `ShareFileModal` component provides a user interface for selecting and sharing files to different destinations within a company's structure. It allows users to choose a destination type (Departments, Buyers, or Manufacturers) and perform either a move or copy operation for selected files.

Key Features

1. **File Sharing Options**:
   - The modal allows users to share files with different destination types: Departments, Buyers, or Manufacturers.
   - For Departments, users can select specific collections within the department.
   - The user can choose between moving or copying files to the selected destination.
2. **Dynamic Data Fetching**:
   - Upon opening the modal, the component fetches relevant data based on the company ID from Firestore:
     - Departments: Lists the departments associated with the company.
     - Buyers: Retrieves a list of buyers.
     - Manufacturers: Retrieves a list of manufacturers.
   - The modal updates dynamically to reflect available destinations for each selected type (Departments, Buyers, or Manufacturers).
3. **Modal Positioning**:
   - The modal position adjusts dynamically to ensure it appears near the triggering button. This takes into account the page's scroll and any parent element's scroll offsets.
4. **Error Handling**:
   - The component includes error handling for failed move or copy operations. Alerts notify the user of any issues, such as when copying files within the same department is not permitted.
5. **User Feedback**:
   - After the operation is completed, users are alerted whether the move or copy was successful or if there were any errors.
6. **Interaction and State Management**:
   - The component uses React's state and effect hooks to manage interactions and data flow:
     - `selectedCollectionType`: Tracks the destination type.
     - `destinationId`: Holds the ID of the selected destination (department, buyer, manufacturer).
     - `departments`, `buyers`, `manufacturers`: Store the fetched lists of available destinations.
     - `isCopy`: A boolean state that determines whether files are moved or copied.

7. **Responsive UI**:
    ○ The modal is responsive to user interaction, resizing, and scrolling, ensuring a smooth experience across various devices.
8. **Closing the Modal**:
    ○ Users can close the modal at any time by clicking the close button, which triggers the `onClose` function.

Components and Functions

- **useState** and **useEffect**: Used for managing component states and side effects like fetching data and handling modal positioning.
- **moveDocument**: A utility function responsible for moving or copying files based on the selected destination.
- **fetchContacts**: A function that fetches buyers and manufacturers associated with a given company.
- **getDocs** and **collection**: Firebase functions for retrieving data from Firestore collections.
- **departmentCollectionsMap**: A mapping of departments to their associated collections.

User Interface

- **File Sharing UI**: Displays the number of selected files and allows users to select a destination and collection.
- **Checkbox for Copying**: A checkbox allows users to toggle between moving and copying the selected files.
- **Dynamic Selection**: Based on the selected destination type, corresponding options for departments, buyers, or manufacturers are displayed.
- **Close Button**: A close button is available to exit the modal at any time.

Error Handling

- If an error occurs during the move/copy operation, users are alerted with a message indicating what went wrong (e.g., copying to the same department is not allowed).