Merchandising Department Homepage Documentation

Overview

The Merchandising Department homepage enables users to upload department-specific files, view lists of buyers and manufacturers, and manage contact-specific files. This page integrates with Firebase for file storage and Firestore for database management.

Components

1. State Management

The page uses React state to manage file selection, upload status, and department lists.

```
const [file, setFile] = useState<File | null>(null);
const [uploadStatus, setUploadStatus] = useState<string | null>(null);
const [selectedCollections, setSelectedCollections] =
useState<string[]>(['files']);
const [buyers, setBuyers] = useState<Buyer[]>([]);
const [manufacturers, setManufacturers] =
useState<Manufacturer[]>([]);
const [selectedContactId, setSelectedContactId] = useState<string |
null>(null);
const [selectedContactType, setSelectedContactType] = useState<'Buyer'
| 'Manufacturer' | null>(null);
const [contactFile, setContactFile] = useState<File | null>(null);
const [contactUploadStatus, setContactUploadStatus] = useState<string
| null>(null);
```

2. File Handling for Department Files

Function: `handleFileChange`

Updates the selected file in state when a file is chosen from the input.

```
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  if (e.target.files && e.target.files[0]) {
    setFile(e.target.files[0]);
  }
};
```

Function: `handleCheckboxChange`

Updates the list of selected collections for file upload.

```
const handleCheckboxChange = (e: React.ChangeEvent<HTMLInputElement>)
=> {
  const value = e.target.value;
  setSelectedCollections(prevSelected =>
    e.target.checked ? [...prevSelected, value] :
prevSelected.filter(item => item !== value)
  );
};
```

Function: `handleUpload`

Uploads a selected department file to the specified Firebase storage path and Firestore collections.

```
const handleUpload = async () => {

  if (!file) {
    alert('Please select a file before uploading.');
    return;
  }
  if (selectedCollections.length === 0) {
    alert('Please select at least one collection to upload the
file.');
    return;
  }

  try {
    const storagePath =
`Company/Departments/Merchandising/${file.name}`;
```

```
    const downloadURL = await uploadFileToStorage(file, storagePath);

    for (const collectionName of selectedCollections) {
      const firestorePath = {
        collectionType: 'Departments' as const,
        companyId: COMPANYID,
        departmentId: DEPARTMENTID,
        customCollectionName: collectionName,
      };
      await updateFirestore(firestorePath, downloadURL, file.name,
storagePath);
    }

    setUploadStatus('File uploaded successfully to all selected
collections!');
    setFile(null);
    setSelectedCollections([]);
  } catch (error) {
    console.error('Error uploading file:', error);
    setUploadStatus('Failed to upload file.');
  }
};
```

3. Loading Buyers and Manufacturers

Function: `loadBuyersAndManufacturers`

Fetches lists of buyers and manufacturers from Firestore on page load.

```
useEffect(() => {
  const loadBuyersAndManufacturers = async () => {
    try {
      const buyersList = await fetchContacts(COMPANYID, 'Buyer');
      const manufacturersList = await fetchContacts(COMPANYID,
'Manufacturer');
      setBuyers(buyersList as Buyer[]);
      setManufacturers(manufacturersList as Manufacturer[]);
    } catch (error) {
      console.error("Error fetching contacts:", error);
```

```
    }
  };
  loadBuyersAndManufacturers();
}, []);
```

4. Handling Contact-Specific File Upload

Function: `handleContactFileChange`

Updates the selected contact file in state when a file is chosen from the input.

```
const handleContactFileChange = (e:
React.ChangeEvent<HTMLInputElement>) => {
  if (e.target.files && e.target.files[0]) {
    setContactFile(e.target.files[0]);
  }
};
```

Function: `handleContactFileUpload`

Uploads a file to the Firebase storage path and Firestore for a specific contact (either a Buyer or Manufacturer).

```
const handleContactFileUpload = async () => {
  if (!contactFile || !selectedContactId || !selectedContactType) {
    alert("Please select a file and a contact before uploading.");
    return;
  }

  try {
    const storagePath = `Company/${selectedContactType === 'Buyer' ?
'Buyers' : 'Manufacturers'}/${selectedContactId}/${contactFile.name}`;
    const downloadURL = await uploadFileToStorage(contactFile,
storagePath);

    const firestorePath = {
      collectionType: selectedContactType === 'Buyer' ? 'Buyers' as
const : selectedContactType === 'Manufacturer' ? 'Manufacturers' as
const : 'Departments' as const,
```

```
      companyId: COMPANYID,
      buyerId: selectedContactType === 'Buyer' ? selectedContactId :
undefined,
      manufacturerId: selectedContactType === 'Manufacturer' ?
selectedContactId : undefined,
    };

    await updateFirestore(firestorePath, downloadURL,
contactFile.name, storagePath);

    setContactUploadStatus('File uploaded successfully to contact!');
    setContactFile(null);
  } catch (error) {
    console.error('Error uploading contact file:', error);
    setContactUploadStatus('Failed to upload file to contact.');
  }
};
```

5. Listing Files for Department and Selected Contact

The `FileList` component displays a list of files in the department and for selected contacts. It takes `collectionPath` and `title` as props to customize the displayed files.

Example Usage of `FileList`

```
<FileList collectionPath={deptFilesPath} title="Department Files" />
<FileList collectionPath={selectedContactFilesPath as [string,
...string[]]} title={`${selectedContactType} Files`} />
```

Error Handling

Errors are managed with try-catch blocks in functions that interact with Firebase, and user-friendly messages are displayed to indicate the status of uploads or failures.

Conclusion

This homepage provides a well-structured and efficient interface for the Merchandising department to manage files related to both department activities and specific contacts. Firebase

Authentication and Firestore ensure that the data is securely stored and accessed across different user roles and departments.

Logistics Department Homepage Documentation

Overview

The Logistics Department homepage provides a comprehensive interface for managing department files, integrating AI features, and supporting searchable content. It enables users to upload files to various department-specific collections (e.g., transportation, customs, and financial files) using Firebase for file storage and Firestore for database management. Additionally, it includes a search bar and AI-assisted features for streamlined access to information.

Components

1. State Management

The page uses React state hooks to manage file selection, upload status, selected collections, and the state of file lists:

javascript
Copy code

```
const [file, setFile] = useState<File | null>(null);
const [uploadStatus, setUploadStatus] = useState<string | null>(null);
const [selectedCollection, setSelectedCollection] =
useState<string>('transportationFiles');
const [selectedFiles, setSelectedFiles] = useState<string[]>([]);
const [fileListUpdated, setFileListUpdated] = useState(false);
```

2. File Handling for Department Files

Function: `handleFileChange`

Updates the selected file in state when a file is chosen:

javascript
Copy code

```
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  if (e.target.files && e.target.files[0]) {
    setFile(e.target.files[0]);
```

```
  }
};
```

Function: `handleFileSelect`

Toggles file selection for further actions such as downloading or viewing details:

javascript
Copy code

```
const handleFileSelect = (fileId: string) => {
  setSelectedFiles((prevSelected) =>
    prevSelected.includes(fileId)
      ? prevSelected.filter(id => id !== fileId)
      : [...prevSelected, fileId]
  );
};
```

Function: `handleUpload`

Uploads the selected file to the Firebase storage path and updates Firestore for the chosen department collection.

javascript
Copy code

```
const handleUpload = async () => {
  if (!file) {
    alert('Please select a file before uploading.');
    return;
  }

  const storagePath = `Company/Departments/Logistics/${file.name}`;
  const downloadURL = await uploadFileToStorage(file, storagePath);
  const firestorePath = {
    collectionType: 'Departments' as const,
    companyId: COMPANYID,
    departmentId: DEPARTMENTID,
    customCollectionName: selectedCollection,
  };
```

```
  try {
    await updateFirestore(firestorePath, downloadURL, file.name,
storagePath);
    setUploadStatus('File uploaded successfully!');
    setFile(null);
  } catch (error) {
    console.error('Error uploading file:', error);
    setUploadStatus('Failed to upload file.');
  }
};
```

3. Firestore Paths for Department Files

These paths specify Firebase Storage and Firestore locations for various collections within the Logistics Department.

javascript
Copy code

```
const customsFilesPath = [
  'Company',
  COMPANYID,
  'Departments',
  DEPARTMENTID,
  'customsFiles',
] as [string, ...string[]];

const financialFilesPath = [
  'Company',
  COMPANYID,
  'Departments',
  DEPARTMENTID,
  'financialFiles',
] as [string, ...string[]];

const transportationFilesPath = [
  'Company',
  COMPANYID,
  'Departments',
  DEPARTMENTID,
```

```
  'transportationFiles',
] as [string, ...string[]];
```

4. File Display Components

The `FileList` component is used to render files in each department collection. Each instance is initialized with the appropriate Firestore path and allows for file selection and horizontal display.

javascript
Copy code

```
<FileList
  collectionPath={transportationFilesPath}
  title="Transportation Files"
  onSearch={() => {}}
  onFileSelect={handleFileSelect}
  horizontal
  refreshTrigger={fileListUpdated}
/>
<FileList
  collectionPath={customsFilesPath}
  title="Customs Files"
  onSearch={() => {}}
  onFileSelect={handleFileSelect}
  horizontal
  refreshTrigger={fileListUpdated}
/>
<FileList
  collectionPath={financialFilesPath}
  title="Financial Files"
  onSearch={() => {}}
  onFileSelect={handleFileSelect}
  horizontal
  refreshTrigger={fileListUpdated}
/>
```

5. Search and AI Integration

Search Bar

The `SearchBar` component allows users to search files by department ID, supporting a streamlined retrieval experience.

javascript
Copy code
```
<SearchBar paths={["KZm56fUOuTobsTRCfknJ"]} />
```

AI Button

The `AIButton` component provides AI features, such as summarizing documents and answering queries related to the files in the department.

javascript
Copy code
```
<AIButton paths={['KZm56fUOuTobsTRCfknJ']}/>
```

7. API Endpoints

`/api/createUser`

Purpose: creates a user with the specified details

- **Method**: POST

Body:
json
Copy code
```
{

  "email": "Email to create the user for",

  "displayName": "Name for the user"

  “phoneNumber”: “Phone number for the user”

  “departmentId”: “The department to add the user to”

  “Role”: “Role name for the user

}
```

Response:
json

```
{

  "message": "Return error or message"

}
```

`/api/resetPass`

Purpose: Sends a reset password email to the email

- **Method**: POST

Body:
json

```
{

  "email": "User's email"

}
```

Response:
json
Copy code

```
{

  "message": "Error or success message"

}
```

Error Handling

All file upload and Firestore update operations are enclosed in try-catch blocks, with user-friendly messages displayed to indicate successful operations or errors.

Conclusion

The Logistics Department homepage provides a well-organized interface for file management, AI-enabled features, and advanced search options. With Firebase Authentication and Firestore,

this page ensures secure and efficient access to department-specific files across user roles, supporting the department's operational needs effectively.

Merchandising Department Homepage Documentation

Overview

The Merchandising Department homepage provides functionalities for uploading and managing department-specific and contact-specific files. It uses Firebase for file storage and Firestore for database management. The page also supports listing buyers and manufacturers and managing their files.

Components

1. State Management

The page utilizes React state hooks to handle the file upload process, department lists, and contact-specific files.

typescript
Copy code

```
const [file, setFile] = useState<File | null>(null);
const [uploadStatus, setUploadStatus] = useState<string | null>(null);
const [selectedCollections, setSelectedCollections] =
useState<string[]>(['files']);
const [buyers, setBuyers] = useState<Buyer[]>([]);
const [manufacturers, setManufacturers] =
useState<Manufacturer[]>([]);
```

```
const [selectedContactId, setSelectedContactId] = useState<string |
null>(null);
const [selectedContactType, setSelectedContactType] = useState<'Buyer'
| 'Manufacturer' | null>(null);
const [contactFile, setContactFile] = useState<File | null>(null);
const [contactUploadStatus, setContactUploadStatus] = useState<string
| null>(null);
```

---

2. File Handling for Department Files

Function: `handleFileChange`

- Updates the selected file in the state when a file is chosen.

typescript
Copy code

```
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  if (e.target.files && e.target.files[0]) {
    setFile(e.target.files[0]);
  }
};
```

Function: `handleCheckboxChange`

- Manages the selected collections for file uploads.

typescript
Copy code

```
const handleCheckboxChange = (e: React.ChangeEvent<HTMLInputElement>)
=> {
  const value = e.target.value;
  setSelectedCollections(prevSelected =>
    e.target.checked ? [...prevSelected, value] :
prevSelected.filter(item => item !== value)
  );
};
```

Function: `handleUpload`

- Handles the upload of department files to Firebase Storage and updates Firestore.

---

3. Loading Buyers and Manufacturers

Function: `loadBuyersAndManufacturers`

- Fetches and sets lists of buyers and manufacturers from Firestore on page load.

typescript
Copy code

```
useEffect(() => {
  const loadBuyersAndManufacturers = async () => {
    try {
      const buyersList = await fetchContacts(COMPANYID, 'Buyer');
      const manufacturersList = await fetchContacts(COMPANYID,
'Manufacturer');
      setBuyers(buyersList as Buyer[]);
      setManufacturers(manufacturersList as Manufacturer[]);
    } catch (error) {
      console.error("Error fetching contacts:", error);
    }
  };
  loadBuyersAndManufacturers();
}, []);
```

---

4. Handling Contact-Specific File Uploads

Function: `handleContactFileChange`

- Updates the selected contact file in the state.

typescript
Copy code

```
const handleContactFileChange = (e:
React.ChangeEvent<HTMLInputElement>) => {
  if (e.target.files && e.target.files[0]) {
    setContactFile(e.target.files[0]);
  }
};
```

Function: `handleContactFileUpload`

- Uploads a file to the Firebase storage path and Firestore for a specific contact.

---

5. Listing Files

The `FileList` component dynamically displays files for the department or a specific contact.

Example Usage:

typescript
Copy code

```
<FileList collectionPath={deptFilesPath} title="Department Files" />
<FileList collectionPath={selectedContactFilesPath as [string,
...string[]]} title={`${selectedContactType} Files`} />
```

Error Handling

- All interactions with Firebase are wrapped in `try-catch` blocks to handle potential errors.
- User-friendly error messages are displayed in case of failures during upload or file handling.

Code Highlights

Uploading Files to Firebase Storage

The `uploadFileToStorage` function handles file uploads with the following features:

- Checks for existing files and prompts for confirmation before replacement.
- Updates the upload progress in real-time.
- Returns the download URL of the uploaded file.

typescript
Copy code

```
export const uploadFileToStorage = async (file: File, storagePath:
string): Promise<string> => { ... }
```

Updating Firestore

The `updateFirestore` function adds metadata for uploaded files to the appropriate Firestore collection.

typescript
Copy code
```
export const updateFirestore = async (firestorePath: FirestorePath,
downloadURL: string, fileName: string, storagePath: string) => { ... }
```

Deleting Files

The `handleFileDelete` function removes files from Firebase Storage and Firestore.

typescript
Copy code
```
export const handleFileDelete = async (fileFullPath: string,
firestorePath: FirestorePath): Promise<void> => { ... }
```

Moving or Copying Documents

The `moveDocument` function allows moving or copying files between different Firestore locations while preserving metadata.

typescript
Copy code
```
export const moveDocument = async (
  sourcePath: FirestorePath,
  destinationPath: FirestorePath,
  documentId: string,
  copy = false,
): Promise<void> => { ... }
```

Conclusion

The Merchandising Department homepage provides a user-friendly interface for file management. It efficiently integrates Firebase and Firestore to handle department-level and

contact-specific files while ensuring security and reliability. The page is structured to facilitate scalability and maintainability.