1. Authentication APIs

1.1 Sign-Up API

Purpose: Creates a new user account with email, password, and additional user details in Firebase Authentication and Firestore.
Method: POST via Firebase SDK.
Process:

1. Input: User's email, password, name, and role (e.g., patient or dermatologist).
2. Output: A new user account is created in Firebase Authentication, and corresponding user details are stored in Firestore.

Key Code:

```
const userCredential = await createUserWithEmailAndPassword(auth, email, password);
await setDoc(doc(db, "users", userCredential.user.uid), {
 name: name,
 email: email,
 role: role,
 uid: userCredential.user.uid,
});
```

1.2 Login API

Purpose: Authenticates the user and grants access to the application.
Method: POST via Firebase SDK.
Process:

1. Input: User's email and password.
2. Output: Authenticated user session and Firebase token.

Key Code:

```
await signInWithEmailAndPassword(auth, email, password);
```

2. Image Handling APIs

2.1 Image Upload API

Purpose: Uploads the user's image to Firebase Storage and generates a unique URL for accessing it.
Method: Indirectly through Firebase Storage SDK.
Process:

1. Input: Image file selected by the user.
2. Output: URL of the uploaded image.

Key Code:

```
const storage = getStorage();
const storageRef = ref(storage, `cases/${user.uid}/${uuidv4()}/${file.name}`);
await uploadBytes(storageRef, file);
const downloadURL = await getDownloadURL(storageRef);
```

2.2 Prediction API

Purpose: Sends the uploaded image to the backend for AI-based prediction and fetches the diagnosis result.
Method: POST via Axios.
Process:

1. Input: URL of the uploaded image.
2. Output: Predicted disease name, description, and treatment recommendations.

Key Code:

```
const response = await axios.post("http://localhost:5000/predict", { imageUrl });
```

2.3 Temporary Image Deletion API

Purpose: Deletes temporary images uploaded by unauthenticated users from Firebase Storage.
Method: Indirectly through Firebase Storage SDK.
Process:

1. Input: URL of the temporary image.
2. Output: Deletes the image from Firebase Storage.

Key Code:

```
const imageRef = ref(storage, imageUrl);
await deleteObject(imageRef);
```

3. Case Management APIs

3.1 Generate Case ID API

Purpose: Generates a unique case ID for each new case.
Method: Programmatic generation using UUID and sequential ID logic.
Process:

1. Input: None.
2. Output: A unique case ID in the format `SKIN-<incremental_number>`.

Key Code:

```
let newCaseId = "SKIN-1";
const casesSnapshot = await getDocs(collection(db, "cases"));
const maxId = Math.max(0, ...casesSnapshot.docs.map(doc => parseInt(doc.data().caseId.split(
")[1])));
newCaseId = `SKIN-${maxId + 1}`;
```

3.2 Save Case Details API

Purpose: Saves case details to Firestore, including patient info, uploaded image URL, and timestamp.
Method: `POST` via Firestore SDK.
Process:

1. Input: Case details, including patient ID, email, image URL, and timestamp.
2. Output: Case document saved in Firestore.

Key Code:

```
await setDoc(doc(collection(db, "cases"), newCaseId), {
 caseId: newCaseId,
 patientId: user.uid,
 patientEmail: user.email,
 imageUrl: downloadURL,
 timestamp: new Date().toISOString(),
 status: "Open",
});
```

3.3 Fetch Case Details API

Purpose: Retrieves details of a specific case from Firestore.
Method: `GET` via Firestore SDK.
Process:

1. Input: Case ID.
2. Output: Case details, including patient info, image URL, and status.

Key Code:

```
const caseRef = doc(db, "cases", caseId);
const caseData = (await getDoc(caseRef)).data();
```

3.4 Update Case Status API

Purpose: Updates the status of a case in Firestore (e.g., from "Pending Review" to "Reviewed").
Method: `PATCH` via Firestore SDK.
Process:

1. Input: Case ID and new status.
2. Output: Updated case document in Firestore.

Key Code:

```
await updateDoc(doc(db, "cases", caseId), { status: "Reviewed" });
```

3.5 Dermatologist Comment API

Purpose: Allows dermatologists to add comments to a case.
Method: `PATCH` via Firestore SDK.
Process:

1. Input: Case ID and dermatologist comment.
2. Output: Updated case document with dermatologist's comment.

Key Code:

```
await updateDoc(doc(db, "cases", caseId), { dermDiagnosis: diagnosis });
```

3.6 Fetch Reviewed By API

Purpose: Retrieves the name of the dermatologist who reviewed a case.
Method: `GET` via Firestore SDK.
Process:

1. Input: Case ID.
2. Output: Dermatologist's name.

Key Code:

```
const reviewedBy = (await getDoc(doc(db, "cases", caseId))).data().reviewedBy;
```

4. Chat APIs

4.1 Fetch User List API

Purpose: Retrieves a list of users available for chatting based on roles.
Method: `GET` via Firestore SDK.
Process:

1. Input: Current user's role.
2. Output: List of users with the opposite role (e.g., patients for dermatologists).

Key Code:

```
const userQuery = query(collection(db, "users"), where("role", "==", roleFilter));
const usersList = (await getDocs(userQuery)).docs.map(doc => doc.data());
```

4.2 Fetch Chat Messages API

Purpose: Retrieves chat messages between two users.
Method: GET via Firestore SDK.
Process:

1. Input: Chat ID (generated by combining both user IDs).
2. Output: List of messages in chronological order.

Key Code:

```
const chatRef = doc(db, "chats", chatId);
const messages = (await getDoc(chatRef)).data()?.messages || [];
```

4.3 Send Message API

Purpose: Sends a message (text or image) to another user.
Method: POST via Firestore SDK.
Process:

1. Input: Sender ID, receiver ID, message text or image URL, and timestamp.
2. Output: Message appended to the chat document in Firestore.

Key Code:

```
const messageData = {
 senderId: currentUser.uid,
 receiverId: otherUserId,
 message: newMessage.trim(),
 imageUrl,
 timestamp: new Date().toISOString(),
 status: "sent",
};
await updateDoc(doc(db, "chats", chatId), { messages: arrayUnion(messageData) });
```

4.4 Upload Chat Image API

Purpose: Uploads an image for a chat and generates a downloadable URL.
Method: Indirectly through Firebase Storage SDK.
Process:

1. Input: Image file selected by the user.
2. Output: URL of the uploaded image.

Key Code:

```
const fileRef = ref(storage, `chat-images/${chatId}/${file.name}`);
await uploadBytes(fileRef, file);
const imageUrl = await getDownloadURL(fileRef);
```

4.5 Update Message Status API

Purpose: Updates the status of a message (e.g., from "sent" to "delivered" or "read").
Method: PATCH via Firestore SDK.
Process:

1. Input: Chat ID and message ID.
2. Output: Updated message status in Firestore.

Key Code:

```
const updatedMessages = messages.map(msg =>
 msg.receiverId === currentUser.uid && msg.status === "sent"
   ? { ...msg, status: "delivered" }
   : msg
);
await updateDoc(doc(db, "chats", chatId), { messages: updatedMessages });
```

4.6 Fetch Unread Message Count API

Purpose: Fetches the count of unread messages for each chat.
Method: GET via Firestore SDK.
Process:

1. Input: Current user ID.
2. Output: Count of unread messages for each user.

Key Code:

```
const unreadCounts = messages.filter(
 msg => msg.receiverId === currentUser.uid && msg.status !== "read"
).length;
```

5. Report APIs

5.1 View Report API

Purpose: Fetches and displays a detailed case report.
Method: `GET` via Firestore SDK.
Process:

1. Input: Case ID.
2. Output: Case details, including patient info, uploaded image, prediction, dermatologist comment, and treatment plan.

Key Code:

```
const reportData = (await getDoc(doc(db, "cases", caseId))).data();
```

5.2 Generate PDF Report API

Purpose: Generates a downloadable PDF of the case report.
Method: Client-side rendering with `html2canvas` and `jsPDF`.
Process:

1. Input: HTML element containing report details.
2. Output: PDF file downloaded to the user's device.

Key Code:

```
const canvas = await html2canvas(document.getElementById("report-content"));
const pdf = new jsPDF();
pdf.addImage(canvas.toDataURL("image/png"), "PNG", 10, 10, 190, canvas.height * 190 /
canvas.width);
pdf.save(`SkinLens_Report_${caseId}.pdf`);
```

5.3 Fetch Disease Details API

Purpose: Retrieves the description and treatment for a specific disease.
Method: GET via Firestore SDK.
Process:

1. Input: Disease name.
2. Output: Disease description and treatment details.

Key Code:

```
const diseasesRef = collection(db, "diseases");
const querySnapshot = await getDocs(query(diseasesRef, where("disease", "==", diseaseName)));
const diseaseDetails = querySnapshot.docs[0]?.data();
```

6. Dashboard APIs

6.1 Fetch Cases for Dermatologist API

Purpose: Retrieves all cases visible to the logged-in dermatologist.
Method: GET via Firestore SDK.
Process:

1. Input: None.
2. Output: List of cases assigned to or available for the dermatologist.

Key Code:

```
const casesRef = collection(db, "cases");
const casesSnapshot = await getDocs(casesRef);
const casesList = casesSnapshot.docs.map(doc => doc.data());
```

6.2 Filter and Sort Cases API

Purpose: Filters and sorts cases based on selected criteria (e.g., date, status).
Method: Client-side filtering and sorting.
Process:

1. Input: Filter attribute, search term, and sort key.
2. Output: Filtered and sorted list of cases.

Key Code:

```
const filteredCases = cases.filter(case =>
 case[filterAttribute]?.toLowerCase().includes(searchTerm.toLowerCase())
).sort((a, b) => (a[sortKey] > b[sortKey] ? 1 : -1));
```

6.3 Pagination API

Purpose: Paginates the list of cases displayed on the dashboard.
Method: Client-side pagination logic.
Process:

1. Input: Current page number and items per page.
2. Output: Cases for the current page.

Key Code:

```
const currentData = filteredCases.slice((currentPage - 1) * itemsPerPage, currentPage *
itemsPerPage);
```

7. Patient Dashboard APIs

7.1 Fetch Cases for Patients API

Purpose: Retrieves all cases for the logged-in patient.
Method: GET via Firestore SDK.
Process:

1. **Input: Patient ID.**
2. **Output: List of cases associated with the patient.**

Key Code:

```
const userCases = casesSnapshot.docs.filter(doc => doc.data().patientId === user.uid);
```

7.2 Filter and Search Cases API

Purpose: Filters and searches cases for the logged-in patient.
Method: Client-side filtering.
Process:

1. Input: Search term and status filter.
2. Output: Filtered list of cases.

Key Code:

```
7.2 Filter and Search Cases API
Purpose: Filters and searches cases for the logged-in patient.
Method: Client-side filtering.
Process:

Input: Search term and status filter.
Output: Filtered list of cases.
Key Code:
```

7.3 Fetch Logged-in User Data API

Purpose: Retrieves the details of the logged-in user (name, email, role).
Method: GET via Firestore SDK.
Process:

1. Input: User ID.
2. Output: User details from Firestore.

Key Code:

```
const userData = (await getDoc(doc(db, "users", user.uid))).data();
```