Appointment Scheduler - Pocket RN

Web application that allows users to schedule appointments with one another and have the ability to accept, decline, and view their appointments.

Table of Contents

1️⃣ 🏁 Quick start

Running locally for devolopment

1. Starting Firebase Emulators

```
# Run command in main directory in new terminal.
firebase emulators:start
```

2. Build Firebase Functions

```
# Run command in functions directory in new terminal.
npm run watch
```

3. Build React Application

```
# Run command in hosting directory.
npm run start
```

4. Seeding Data

If you don't want to manually add data through the use of the application. Seed creates 3 users and 5 appointments. The 5 appointments are between test123@gmail.com and a@gmail.com with a@gmail.com scheduling them.

- test123@gmail.com
- a@gmail.com
- b@gmail.com
- Password: 123456

```
# Run command in hosting directory.
npm run seed
```

2 Models/Schema

- Each document uses the auto generated id from Firestore as its unique identifier for the uid field.
- Storing the uid in the text field allows for faster operations when you return the document id for reference.
- Each user contains an array of appointment uids/IDs which allows for faster read operations when retrieving by document id rather than searching by the text field email.

Users Collection

Each document in the users collection contains:

```
// Date object for when document was created. Stored as timestamp in firestore.
creationTime: Date;

// Document uid stored in text field.
uid: string;

// User email.
email: string;

// User display name.
username: string;

// Array of appointment IDs related to user.
appointments: string[];
```

Appointments Collection

Each document in the appointments collection contains:

```
// Document uid stored in text field.
uid: string;

// Date and time the appointment is scheduled at.
scheduledTime: Date;

// User uid of sender.
userID: string;

// Sender email.
senderEmail: string;

// Person scheduled appointment with email.
receiverEmail: string;

// Description of the appointment. 1000 chars.
description: string;

// Status of appointment. "Pending" | "Accepted" | "Declined".
status: 0 | 1 | 2;

// Date object for when document was created. Stored as timestamp in firestore.
creationTime: Date;

// Date object for when document was updated. Stored as timestamp in firestore.
updatedTime: Date;

// Date object for when document was deleted. Stored as timestamp in firestore.
deletedTime: Date;
```

3 Features

Firebase User Authentication

Firebase Authentication with sign-in method Email/Password enabled.

Register / Create an account

Create an account through Firebase auth with email and password. Then create a user document in firestore.

```
/**
 * Register a user through firebase auth and then create user doc in firestore.
 * @param username string
 * @param email Email string @ domain.
 * @param password string
 * @returns
 */
export function register(
  username: string,
  email: string,
  password: string
): Promise<void> {
  return firebase
    .auth()
    .createUserWithEmailAndPassword(email, password)
    .then((userCredential) => {
      const { user } = userCredential;
      const { uid } = user!;
      const { creationTime } = user!.metadata;

      /** Update user with display name */
      user!.updateProfile({
        displayName: username,
      });

      /** Create user document */
      firebaseFunctions.httpsCallable("createUser")({
        username,
        email,
        uid,
        creationTime,
      });
    })
    .catch((error) => {
      throw new Error(error);
    });
}
```

Login

Login through Firebase auth and then return user displayname and uid from document.

```
interface LoginData {
  displayName: string;
  email: string;
  uid: string;
}

/**
 * Login user with email and password
 * @param email Email string @ domain.
 * @param password string
 * @returns User displayname, email, and uid.
 */
export function login(email: string, password: string): Promise<LoginData> {
  return firebase
    .auth()
    .signInWithEmailAndPassword(email, password)
    .then((userCredential) => {
      const { user } = userCredential;
      const displayName = user!.providerData[0]!.displayName!;
      return { displayName, email, uid: user!.uid };
    })
    .catch((error) => {
      console.log(error);
      throw new Error(error);
    });
}
```

Logout

Sign out through Firebase auth.

```
/**
 * Logout user.
 * @returns
 */
export function logout(): Promise<any> {
  return new Promise((resolve, reject) => {
    firebase
      .auth()
      .signOut()
      .then(() => {
        // Sign-out successful.
        resolve("Logged out");
      })
      .catch((error) => {
        // An error happened.
        reject(error);
      });
  });
}
```

Firebase Functions

Features for users and appointments.

🏷 User

Create User

Create user document in firestore after firebase auth.

```
interface RegisterBody {
  username: string;
  email: string;
  uid: string;
  creationTime: string;
}

/**
 * Create user doc after auth.
 */
export const createUser = functions.https.onCall(
  async (data: RegisterBody, context) => {
    const { username, email, uid, creationTime } = data;

    /** Set collections users at document uid with information */
    await firestoreDB
      .collection("users")
      .doc(uid)
      .set({
        username,
        email,
        uid,
        creationTime: new Date(creationTime),
        appointments: [],
      });

    return { text: "Registered" };
  }
);
```

Search Users

Search users by email to schedule an appointment with.

```
interface SearchBody {
  email: string;
  uid: string;
}

/**
 * Search for users to schedule an appointment with
 */
export const searchUsers = functions.https.onCall(
  async (data: SearchBody, context) => {
    const { email } = data;
    const users = await firestoreDB.collection("users");
    const query = await users.where("email", "==", email);

    /**
     * https://firebase.google.com/docs/firestore/query-data/queries?authuser=3
     */
    const execution = await query
      .get()
      .then((querySnapshot) => {
        let results: { [key: string]: string }[] = [];
        /** Push in user information that matches email */
        querySnapshot.forEach((doc) => {
          results.push(doc.data());
        });

        return results;
      })
      .catch((error) => {
        console.log("Error getting documents: ", error);
      });

    return { users: execution };
  }
);
```

🏷 Appointment

Schedule Appointment

Create appointment document, return uid and update field with it. Update both receiver and sender appointment arrays with newly created appointment uid.

```typescript
interface ScheduleAppointmentBody {
  uid: string;
  sender: string;
  receiver: string;
  description: string;
  scheduledTime: string;
}

/**
 * Create appointment document, update document with auto generated id and return id.
 * Update both user appointment arrays with id.
 */
export const scheduleAppointment = functions.https.onCall(
  async (data: ScheduleAppointmentBody, context) => {
    const { sender, receiver, description, uid, scheduledTime } = data;

    /**
     * Returns document id after adding document into collection.
     * https://firebase.google.com/docs/firestore/manage-data/add-data
     */
    const createAppointment = await firestoreDB
      .collection("appointments")
      .add({
        userID: uid,
        senderEmail: sender,
        receiverEmail: receiver,
        description,
        scheduledTime: new Date(scheduledTime),
        creationTime: new Date(),
        status: Status.PENDING,
      })
      .then((docRef) => {
        docRef.update({
          uid: docRef.id,
        });

        return docRef.id;
      })
      .catch((error) => {
        console.log(error);
      });

    /**
     * Update both sender and receiver with the appointment.
     * https://stackoverflow.com/questions/55714423/firestore-query-then-update
     */
    await firestoreDB
      .collection("users")
      .where("email", "in", [receiver, sender])
      .get()
      .then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
          const data = doc.data();
          doc.ref.update({
            appointments: [...data.appointments, createAppointment],
          });
        });
      })
      .catch((error) => {
        console.log(error);
      });
  }
);
```

Get Appointment

Helper function retrieveAppointment to retrieve appointment with uid.

```
/**
 * Helper function to retrieve appointment data. Modify firestore timestamp to string.
 * @param id string
 * @returns appointment data {}
 */
const retrieveAppointment = async (id: string) => {
  const appointment = await firestoreDB
    .collection("appointments")
    .doc(id)
    .get()
    .then((docSnapshot) => {
      const data = docSnapshot.data();
      /** Don't return appointment if soft deleted */
      if (!data!.deletedTime) {
        /** Make copy of data and alter firestore timestamps */
        return Object.assign({}, data, {
          scheduledTime: data!.scheduledTime.toDate().toString(), // Convert firestore timestamp
          creationTime: data!.creationTime.toDate().toString(), // Convert firestore timestamp t
        });
      } else {
        return;
      }
    });

  return appointment;
};

/**
 * Retrieve array of appointments from uid.
 */
export const getAppointments = functions.https.onCall(
  async (data: { uid: string }, context) => {
    const appointmentIDs = await firestoreDB
      .collection("users")
      .doc(data.uid)
      .get()
      .then((docSnapshot) => {
        const data = docSnapshot.data();
        return data!.appointments;
      });

    let results = [];

    for (let i = 0; i < appointmentIDs.length; i++) {
      const result = await retrieveAppointment(appointmentIDs[i]);

      if (result) results.push(result);
    }

    return results;
  }
);
```

Update Appointment

Update timestamp for updatedTime and status in document.

```
interface UpdateAppointmentBody {
  uid: string;
  status: string;
}

/**
 * Update appointment document by id with new status
 */
export const updateAppointmentStatus = functions.https.onCall(
  async (data: UpdateAppointmentBody, context) => {
    const { uid, status } = data;
    let code;

    if (status === "ACCEPTED") code = Status.ACCEPTED;
    if (status === "DECLINED") code = Status.DECLINED;

    const appointment = await firestoreDB
      .collection("appointments")
      .doc(uid)
      .update({
        status: code,
        updatedTime: new Date(),
      })
      .then((result) => {
        return result;
      })
      .catch((error) => {
        console.log(error);
      });

    return appointment;
  }
);
```

Delete Appointment

Update timestamp for deletedTime in document.

```
/**
 * Soft delete appointment by adding deleted time.
 */
export const deleteAppointment = functions.https.onCall(
  async (data: { uid: string }, context) => {
    const { uid } = data;

    const appointment = await firestoreDB
      .collection("appointments")
      .doc(uid)
      .update({
        deletedTime: new Date(),
      })
      .then((result) => {
        return result;
      })
      .catch((error) => {
        console.log(error);
      });

    return appointment;
  }
);
```

4 Additional Features to Implement

- Time block
- Length of appointment
- Authentication for database operations