Chat Logs

AI models used: Chat GPT o1 and o3, Gemini

# Prompt 1:

## ChatGPT

Using the provided data dictionary, create a Firestore database with matching data types. Just create one document with test data for each collection. For any field that uses references or has objects that use references, reference a document from the appropriate collection. If any of the data types of the fields need to be changed, feel free to change them to be appropriate to the field and the collection using them. Finally, make sure that all the collections and fields are appropriate to be able to complete the use cases for this project, and create more collections or fields if the use cases cannot be completed by only using the fields described in the provided data dictionary. Make sure to use the syntax format for Google Firebase and Firestore. If this is a process that can be automated please explain how to do so as well.  
Here is the provided data dictionary:  
**Data Dictionary for MyFridge Database**

**1. User Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| user\_id | String (PK) | Unique identifier for each user. |
| email | String (Unique) | User’s email address. |
| username | String (Unique) | User’s display name. |
| bio | String | Short description about the user. |
| password | String (Hashed) | Securely hashed password. |
| pfp | String (URL) | URL of the user's profile picture. |
| followers | Array (User) | List of user IDs who follow this user. |
| following | Array (User) | List of user IDs this user follows. |
| created\_At | Timestamp | Date and time of account creation. |
| posts | Array (Post) | List of post IDs created by the user. |
| recipes | Array (Recipe) | List of recipe IDs created by the user. |

**2. Recipe Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| recipe\_id | String (PK) | Unique identifier for each recipe. |
| author | String (FK - user\_id) | User ID of the recipe's author. |
| title | String | Title of the recipe. |
| media\_url | String (URL) | URL of the recipe image. |
| ratings | Float | Average rating of the recipe. |
| isRecommended | Boolean | Indicates if the recipe is recommended. |
| ingredients | Array (Ingredient) | List of ingredients used in the recipe. |
| instructions | Array (String) | Step-by-step cooking instructions. |
| tags | Array (String) | Tags associated with the recipe (e.g., "vegan", "low-carb"). |
| comments | Array (Comment) | List of comment IDs related to this recipe. |
| created\_at | Timestamp | Date and time the recipe was created. |

**3. Ingredient Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ingredient\_id | String (PK) | Unique identifier for each ingredient. |
| name | String | Name of the ingredient. |

**4. Fridge Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| fridge\_id | String (PK) | Unique identifier for each fridge entry. |
| user\_id | String (FK - user\_id) | User ID associated with the fridge. |
| ingredients | Array (Object) | List of ingredient objects stored in the fridge. |
| shopping\_list | Array (String) | List of ingredients needed to buy. |
| updated\_at | Timestamp | Date and time when the fridge was last updated. |

**5. Post Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| post\_id | String (PK) | Unique identifier for each post. |
| user\_id | String (FK - user\_id) | User ID of the post's author. |
| title | String | Title of the post. |
| description | String | Content of the post. |
| created\_at | Timestamp | Date and time the post was created. |
| comments | Array (Comment) | List of comment IDs related to this post. |

**6. Comment Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| comment\_id | String (PK) | Unique identifier for each comment. |
| recipe\_id | String (FK - recipe\_id) | Associated recipe ID (if applicable). |
| user\_id | String (FK - user\_id) | User ID of the commenter. |
| post\_id | String (FK - post\_id) | Associated post ID (if applicable). |
| content | String | Text content of the comment. |
| created\_at | Timestamp | Date and time the comment was created. |

**7. Ratings Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| rating\_id | String (PK) | Unique identifier for each rating. |
| recipe\_id | String (FK - recipe\_id) | Recipe ID being rated. |
| user\_id | String (FK - user\_id) | User ID of the reviewer. |
| rating | Integer (1-5) | Rating value (1 to 5). |
| created\_at | Timestamp | Date and time the rating was given. |

### Result:

Below is an example of how you might seed your Firestore database with one test document per collection using the Firebase syntax. In this example, we use document references for fields that “point” to another document. You can run this code (for example, in your Node.js backend using the Firebase Admin SDK or in a client app with proper security rules) to create the documents.

Note: In a real project you might want to seed more data and have a migration/seeding script. This process can be automated by writing a script (for example, in Node.js) that reads a JSON schema (or data dictionary) and iterates over a list of collections—using Firestore’s SDK to create documents with test or seed data. You can also use the Firebase Emulator Suite to import seed data from JSON files.

Below is a sample script:

// Assume firebase (or the Firebase Admin SDK) has already been initialized

const db = firebase.firestore();

// USERS collection

db.collection('users').doc('user\_001').set({

user\_id: 'user\_001',

email: 'testuser@example.com',

username: 'testuser',

bio: 'Hello, I am a test user.',

password: 'hashed\_password\_here',

pfp: 'https://example.com/profile.jpg',

// For references, we use document references (here we reference the same user for illustration)

followers: [ db.collection('users').doc('user\_001') ],

following: [ db.collection('users').doc('user\_001') ],

created\_at: firebase.firestore.Timestamp.now(),

posts: [ db.collection('posts').doc('post\_001') ],

recipes: [ db.collection('recipes').doc('recipe\_001') ]

});

// RECIPES collection

db.collection('recipes').doc('recipe\_001').set({

recipe\_id: 'recipe\_001',

// Reference to the user document as the author

author: db.collection('users').doc('user\_001'),

title: 'Delicious Vegan Salad',

media\_url: 'https://example.com/recipe.jpg',

ratings: 4.5,

isRecommended: true,

// Reference ingredients used in the recipe

ingredients: [ db.collection('ingredients').doc('ingredient\_001') ],

instructions: [

'Wash all vegetables thoroughly.',

'Chop vegetables and mix them in a bowl.',

'Drizzle with olive oil and lemon juice.'

],

tags: ['vegan', 'salad', 'healthy'],

comments: [ db.collection('comments').doc('comment\_001') ],

created\_at: firebase.firestore.Timestamp.now()

});

// INGREDIENTS collection

db.collection('ingredients').doc('ingredient\_001').set({

ingredient\_id: 'ingredient\_001',

name: 'Tomato'

});

// FRIDGES collection

db.collection('fridges').doc('fridge\_001').set({

fridge\_id: 'fridge\_001',

// Reference to the user who owns this fridge

user\_id: db.collection('users').doc('user\_001'),

// Each ingredient object can include additional info like quantity

ingredients: [

{ ingredient: db.collection('ingredients').doc('ingredient\_001'), quantity: 5 }

],

shopping\_list: ['Onion', 'Garlic'],

updated\_at: firebase.firestore.Timestamp.now()

});

// POSTS collection

db.collection('posts').doc('post\_001').set({

post\_id: 'post\_001',

user\_id: db.collection('users').doc('user\_001'),

title: 'My First Food Post',

description: 'Sharing my experience with a new recipe!',

created\_at: firebase.firestore.Timestamp.now(),

comments: [ db.collection('comments').doc('comment\_001') ]

});

// COMMENTS collection

db.collection('comments').doc('comment\_001').set({

comment\_id: 'comment\_001',

// For this example, we assume the comment is on a recipe

recipe\_id: db.collection('recipes').doc('recipe\_001'),

user\_id: db.collection('users').doc('user\_001'),

post\_id: null, // Not applicable when commenting on a recipe

content: 'Looks delicious!',

created\_at: firebase.firestore.Timestamp.now()

});

// RATINGS collection

db.collection('ratings').doc('rating\_001').set({

rating\_id: 'rating\_001',

recipe\_id: db.collection('recipes').doc('recipe\_001'),

user\_id: db.collection('users').doc('user\_001'),

rating: 5,

created\_at: firebase.firestore.Timestamp.now()

});

// Optional: SHOPPING LISTS collection (if you need a separate list document)

db.collection('shoppingLists').doc('list\_001').set({

list\_id: 'list\_001',

user\_id: db.collection('users').doc('user\_001'),

items: ['Cheese', 'Basil'],

created\_at: firebase.firestore.Timestamp.now()

});

// Optional: FOLLOWS collection (to represent follow relationships)

db.collection('follows').doc('follow\_001').set({

follow\_id: 'follow\_001',

follower\_id: db.collection('users').doc('user\_001'),

followed\_id: db.collection('users').doc('user\_001'),

created\_at: firebase.firestore.Timestamp.now()

});

### Explanation

* **References:**  
  In fields such as author in recipes or user\_id in posts, we store a reference to the corresponding document by using db.collection('collectionName').doc('docId'). This makes it easier to join related data when needed.
* **Timestamps:**  
  The firebase.firestore.Timestamp.now() function is used to store the current date and time.
* **Arrays of References:**  
  For fields like followers, following, posts, recipes, and comments, we store an array of document references. This ensures that related documents can be easily queried.
* **Optional Collections:**  
  Additional collections like shoppingLists and follows are added based on extended use cases (for managing shopping lists separately or for follow relationships).

### Automating the Process

To automate seeding your Firestore database:

1. **Create a JSON Schema:** Define your collections and documents in a JSON file based on your data dictionary.
2. **Write a Script:** Use Node.js with the Firebase Admin SDK to read the JSON file and iterate over each collection/document. For example, you can use a loop to call db.collection(collectionName).doc(docId).set(documentData).
3. **Use Firebase Emulator Suite:** During development, the Emulator Suite allows you to import seed data and run automated tests before deploying to production.
4. **CI/CD Integration:** Incorporate the seeding script into your continuous integration/deployment pipeline to ensure your database is seeded with test data on deployment.

This approach ensures your Firestore structure aligns with your functional requirements and use cases, and that references are properly set up for related documents.

## Gemini:

Using the provided data dictionary, create a Firestore database with matching data types. Just create one document with test data for each collection. For any field that uses references or has objects that use references, reference a document from the appropriate collection. If any of the data types of the fields need to be changed, feel free to change them to be appropriate to the field and the collection using them. Finally, make sure that all the collections and fields are appropriate to be able to complete the use cases for this project, and create more collections or fields if the use cases cannot be completed by only using the fields described in the provided data dictionary. Make sure to use the syntax format for Google Firebase and Firestore. If this is a process that can be automated please explain how to do so as well.  
Here is the provided data dictionary:  
**Data Dictionary for MyFridge Database**

**1. User Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| user\_id | String (PK) | Unique identifier for each user. |
| email | String (Unique) | User’s email address. |
| username | String (Unique) | User’s display name. |
| bio | String | Short description about the user. |
| password | String (Hashed) | Securely hashed password. |
| pfp | String (URL) | URL of the user's profile picture. |
| followers | Array (User) | List of user IDs who follow this user. |
| following | Array (User) | List of user IDs this user follows. |
| created\_At | Timestamp | Date and time of account creation. |
| posts | Array (Post) | List of post IDs created by the user. |
| recipes | Array (Recipe) | List of recipe IDs created by the user. |

**2. Recipe Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| recipe\_id | String (PK) | Unique identifier for each recipe. |
| author | String (FK - user\_id) | User ID of the recipe's author. |
| title | String | Title of the recipe. |
| media\_url | String (URL) | URL of the recipe image. |
| ratings | Float | Average rating of the recipe. |
| isRecommended | Boolean | Indicates if the recipe is recommended. |
| ingredients | Array (Ingredient) | List of ingredients used in the recipe. |
| instructions | Array (String) | Step-by-step cooking instructions. |
| tags | Array (String) | Tags associated with the recipe (e.g., "vegan", "low-carb"). |
| comments | Array (Comment) | List of comment IDs related to this recipe. |
| created\_at | Timestamp | Date and time the recipe was created. |

**3. Ingredient Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ingredient\_id | String (PK) | Unique identifier for each ingredient. |
| name | String | Name of the ingredient. |

**4. Fridge Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| fridge\_id | String (PK) | Unique identifier for each fridge entry. |
| user\_id | String (FK - user\_id) | User ID associated with the fridge. |
| ingredients | Array (Object) | List of ingredient objects stored in the fridge. |
| shopping\_list | Array (String) | List of ingredients needed to buy. |
| updated\_at | Timestamp | Date and time when the fridge was last updated. |

**5. Post Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| post\_id | String (PK) | Unique identifier for each post. |
| user\_id | String (FK - user\_id) | User ID of the post's author. |
| title | String | Title of the post. |
| description | String | Content of the post. |
| created\_at | Timestamp | Date and time the post was created. |
| comments | Array (Comment) | List of comment IDs related to this post. |

**6. Comment Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| comment\_id | String (PK) | Unique identifier for each comment. |
| recipe\_id | String (FK - recipe\_id) | Associated recipe ID (if applicable). |
| user\_id | String (FK - user\_id) | User ID of the commenter. |
| post\_id | String (FK - post\_id) | Associated post ID (if applicable). |
| content | String | Text content of the comment. |
| created\_at | Timestamp | Date and time the comment was created. |

**7. Ratings Table**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| rating\_id | String (PK) | Unique identifier for each rating. |
| recipe\_id | String (FK - recipe\_id) | Recipe ID being rated. |
| user\_id | String (FK - user\_id) | User ID of the reviewer. |
| rating | Integer (1-5) | Rating value (1 to 5). |
| created\_at | Timestamp | Date and time the rating was given. |

### Result:

This response outlines how to create the Firestore database structure based on your provided data dictionary, including test data. It also discusses automation options.

**Firestore Database Structure & Test Data**

The following structure uses the provided data dictionary as a base, making necessary adjustments for data types and relationships to ensure data integrity and usability. Note that primary keys (PKs) are automatically generated by Firestore. Foreign keys (FKs) are implemented as references to other documents.

**1. Users Collection**

{

"user\_id": "user123", //Auto-generated by Firestore, not explicitly set

"email": "testuser@example.com",

"username": "TestUser",

"bio": "This is a test user.",

"password": "hashedPassword", // In reality, a hashed password would be used

"pfp": "https://example.com/pfp.jpg",

"followers": ["user456"],

"following": ["user789"],

"created\_at": firebase.firestore.FieldValue.serverTimestamp(),

"posts": ["post1"],

"recipes": ["recipe1"]

}

**2. Recipes Collection**

{

"recipe\_id": "recipe1", //Auto-generated by Firestore, not explicitly set

"author": {

"user\_id": "user123" //Reference to user document

},

"title": "Test Recipe",

"media\_url": "https://example.com/recipe.jpg",

"ratings": 4.5,

"isRecommended": true,

"ingredients": [

{ "ingredient\_id": "ing1", "quantity": "1 cup" },

{ "ingredient\_id": "ing2", "quantity": "2 tbsp" }

],

"instructions": [

"Step 1: Do this.",

"Step 2: Do that."

],

"tags": ["test", "recipe"],

"comments": ["comment1"],

"created\_at": firebase.firestore.FieldValue.serverTimestamp()

}

**3. Ingredients Collection**

{

"ingredient\_id": "ing1",//Auto-generated by Firestore, not explicitly set

"name": "Flour"

}

**4. Fridge Collection**

{

"fridge\_id": "fridge1", //Auto-generated by Firestore, not explicitly set

"user\_id": "user123", //Reference to user document

"ingredients": [

{ "ingredient\_id": "ing1", "quantity": "1 cup", "expiry": "2024-03-15" },

{ "ingredient\_id": "ing2", "quantity": "2 tbsp", "expiry": "2024-03-20" }

],

"shopping\_list": ["milk", "eggs"],

"updated\_at": firebase.firestore.FieldValue.serverTimestamp()

}

**5. Posts Collection**

{

"post\_id": "post1", //Auto-generated by Firestore, not explicitly set

"user\_id": {

"user\_id": "user123" //Reference to user document

},

"title": "Test Post",

"description": "This is a test post.",

"created\_at": firebase.firestore.FieldValue.serverTimestamp(),

"comments": ["comment2"]

}

**6. Comments Collection**

{

"comment\_id": "comment1", //Auto-generated by Firestore, not explicitly set

"recipe\_id": "recipe1", //Reference to recipe document

"user\_id": {

"user\_id": "user123" //Reference to user document

},

"post\_id": null,

"content": "This is a test comment.",

"created\_at": firebase.firestore.FieldValue.serverTimestamp()

}

**7. Ratings Collection**

{

"rating\_id": "rating1", //Auto-generated by Firestore, not explicitly set

"recipe\_id": "recipe1", //Reference to recipe document

"user\_id": {

"user\_id": "user123" //Reference to user document

},

"rating": 5,

"created\_at": firebase.firestore.FieldValue.serverTimestamp()

}

**Automation**

Creating this database structure can be automated using various tools and approaches:

* **Firebase Admin SDK:**You can use the Admin SDK (with appropriate authentication) to programmatically create collections and documents. The SDKs are available for many languages (Node.js, Python, Java, etc.). This is the most flexible and recommended method. You'd write a script that creates collections, and then iterates through the test data, creating documents.
* **Cloud Functions:**For a more event-driven approach, you could deploy a Cloud Function that triggers upon a specific event (e.g., creation of a new user) and automatically sets up the related data in other collections.
* **Third-party tools:**Several NoSQL database migration or management tools could potentially automate the creation of the Firestore database from a structured definition like the one above.

**Example using Firebase Admin SDK (Node.js):**

This is a simplified example; error handling and more robust data validation would be needed in a production environment.

const admin = require('firebase-admin');

// ... Initialize Firebase Admin SDK ...

const db = admin.firestore();

// Function to create a document with error handling

async function createDocument(collection, data) {

try {

await db.collection(collection).add(data);

console.log(`Document added to ${collection}`);

} catch (error) {

console.error(`Error adding document to ${collection}:`, error);

}

}

//Example Usage: Create a User

createDocument('users', {

email: 'testuser@example.com',

username: 'TestUser',

// ... other user data

});

// ... similar functions to create recipes, ingredients, etc. ...Remember to replace placeholders like // ... Initialize Firebase Admin SDK ... with your actual Firebase initialization code. This code would need to be expanded to include all your collections and test data. The key is structuring your data in a way that's easy to iterate over and create the necessary documents.

# Prompt 2

## ChatGPT

Using firebase syntax, make all of the fields that are references reference a random document id of the type of document that it needs. To do this, you can:

// Add a new document with an auto-generated ID

collectionRef.add({

  name: 'My Document',

  data: 'some value'

}).then(docRef => {

  console.log('Document written with ID: ', docRef.id);

}).catch(error => {

    console.error("Error adding document: ", error);

});

Here is the provided code:

var admin = require("firebase-admin");

var serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({

  credential: admin.credential.cert(serviceAccount)

});

const db = admin.firestore();

// Function to create a document with error handling

async function createDocument(collection, data) {

    try {

      await db.collection(collection).add(data);

      console.log(`Document added to ${collection}`);

    } catch (error) {

      console.error(`Error adding document to ${collection}:`, error);

    }

  }

  createDocument('Users', {

    // "user\_id": "user123", //Auto-generated by Firestore, not explicitly set

    "email": "testuser@example.com",

    "username": "TestUser",

    "bio": "This is a test user.",

    "pfp": "https://example.com/pfp.jpg",

    "followers": ["user456"], //Reference to user document

    "following": ["user789"], //Reference to user document

    "created\_at": admin.firestore.FieldValue.serverTimestamp(),

    "updated\_at": admin.firestore.FieldValue.serverTimestamp(),

    "posts": ["post1"], //Reference to post document

    "recipes": ["recipe1"] //Reference to recipe document

  }

    );

    createDocument('Recipes', {

        // "recipe\_id": "recipe1", //Auto-generated by Firestore, not explicitly set

        "author": {

          "user\_id": "user123" //Reference to user document

        },

        "title": "Test Recipe",

        "media\_url": "https://example.com/recipe.jpg",

        "ratings": 4.5,

        "isRecommended": true,

        "ingredients": [

          { "ingredient\_id": "ing1", "quantity": "1 cup" }, //Contains reference to ingredient document

          { "ingredient\_id": "ing2", "quantity": "2 tbsp" } //Contains reference to ingredient document

        ],

        "instructions": [

          "Step 1: Do this.",

          "Step 2: Do that."

        ],

        "tags": ["test", "recipe"],

        "comments": ["comment1"], //Reference to comment document

        "created\_at": admin.firestore.FieldValue.serverTimestamp(),

        "updated\_at": admin.firestore.FieldValue.serverTimestamp()

      }

    );

    createDocument('Ingredients', {

        // "ingredient\_id": "ing1", //Auto-generated by Firestore, not explicitly set

        "name": "Tomato"

      }

    );

    createDocument('Ingredients', {

        // "ingredient\_id": "ing2",//Auto-generated by Firestore, not explicitly set

        "name": "Flour"

      }

    );

    createDocument('Fridges', {

        // "fridge\_id": "fridge1", //Auto-generated by Firestore, not explicitly set

        "user\_id": "user123", //Reference to user document

        "ingredients": [

          { "ingredient\_id": "ing1"}, //Contains reference to ingredient document

          { "ingredient\_id": "ing2"} //Contains reference to ingredient document

        ],

        "shopping\_list": ["milk", "eggs"], //Contains references to ingredient document

        "updated\_at": admin.firestore.FieldValue.serverTimestamp()

      }

    );

    createDocument('Posts', {

        // "post\_id": "post1", //Auto-generated by Firestore, not explicitly set

        "user\_id": "user123", //Reference to user document

        "title": "Test Post",

        "description": "This is a test post.",

        "created\_at": admin.firestore.FieldValue.serverTimestamp(),

        "comments": ["comment2"] //Reference to comment document

      }

    );

    createDocument('Comments', {

        // "comment\_id": "comment1", //Auto-generated by Firestore, not explicitly set

        "recipe\_id": "recipe1", //Reference to recipe document

        "user\_id": "user123", //Reference to user document

        "post\_id": null, //Reference to post document

        "content": "This is a test comment.",

        "created\_at": admin.firestore.FieldValue.serverTimestamp()

      }

      );

    createDocument('Ratings', {

        // "rating\_id": "rating1", //Auto-generated by Firestore, not explicitly set

        "recipe\_id": "recipe1", //Reference to recipe document

        "raters": [

            {

          "user\_id": "user123", //Reference to user document

          "rating": 5,

        }

    ],

        "avg\_rating": 5

      }

    );

### Result:

Below is an example of how you can seed your Firestore database so that every field meant to reference another document uses a randomly generated document ID. In this example, we add documents using the collectionRef.add() method. Once a document is created, its returned reference is used when creating related documents. For instance, instead of hardcoding "user123", we create a user document first and then use its reference when creating a recipe. You can run this as a Node.js script with the Firebase Admin SDK.

var admin = require("firebase-admin");

var serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({

credential: admin.credential.cert(serviceAccount)

});

const db = admin.firestore();

async function seedFirestore() {

try {

// 1. Create a user document

const userRef = await db.collection('Users').add({

email: "testuser@example.com",

username: "TestUser",

bio: "This is a test user.",

pfp: "https://example.com/pfp.jpg",

followers: [], // Will update later if needed

following: [],

created\_at: admin.firestore.FieldValue.serverTimestamp(),

updated\_at: admin.firestore.FieldValue.serverTimestamp(),

posts: [],

recipes: []

});

console.log('User document created with ID:', userRef.id);

// 2. Create ingredient documents

const ingredientRef1 = await db.collection('Ingredients').add({ name: "Tomato" });

const ingredientRef2 = await db.collection('Ingredients').add({ name: "Flour" });

console.log('Ingredient documents created with IDs:', ingredientRef1.id, ingredientRef2.id);

// 3. Create a recipe document referencing the user and ingredients

const recipeRef = await db.collection('Recipes').add({

author: userRef, // Using the user document reference

title: "Test Recipe",

media\_url: "https://example.com/recipe.jpg",

ratings: 4.5,

isRecommended: true,

ingredients: [

{ ingredient: ingredientRef1, quantity: "1 cup" },

{ ingredient: ingredientRef2, quantity: "2 tbsp" }

],

instructions: [

"Step 1: Do this.",

"Step 2: Do that."

],

tags: ["test", "recipe"],

comments: [], // Will update later

created\_at: admin.firestore.FieldValue.serverTimestamp(),

updated\_at: admin.firestore.FieldValue.serverTimestamp()

});

console.log('Recipe document created with ID:', recipeRef.id);

// 4. Create a fridge document referencing the user and ingredients

const fridgeRef = await db.collection('Fridges').add({

user\_id: userRef, // Reference to the user document

ingredients: [

{ ingredient: ingredientRef1 },

{ ingredient: ingredientRef2 }

],

shopping\_list: ["milk", "eggs"],

updated\_at: admin.firestore.FieldValue.serverTimestamp()

});

console.log('Fridge document created with ID:', fridgeRef.id);

// 5. Create a post document referencing the user

const postRef = await db.collection('Posts').add({

user\_id: userRef, // Reference to the user document

title: "Test Post",

description: "This is a test post.",

created\_at: admin.firestore.FieldValue.serverTimestamp(),

comments: [] // Will update later if needed

});

console.log('Post document created with ID:', postRef.id);

// 6. Create a comment document referencing the recipe and user

const commentRef = await db.collection('Comments').add({

recipe\_id: recipeRef, // Reference to the recipe document

user\_id: userRef, // Reference to the user document

post\_id: null, // Not applicable here; could be set to a postRef if needed

content: "This is a test comment.",

created\_at: admin.firestore.FieldValue.serverTimestamp()

});

console.log('Comment document created with ID:', commentRef.id);

// 7. Create a rating document referencing the recipe and user

const ratingRef = await db.collection('Ratings').add({

recipe\_id: recipeRef, // Reference to the recipe document

raters: [

{

user\_id: userRef, // Reference to the user document

rating: 5

}

],

avg\_rating: 5,

created\_at: admin.firestore.FieldValue.serverTimestamp()

});

console.log('Rating document created with ID:', ratingRef.id);

// 8. Update the user document with references to the post and recipe documents

await db.collection('Users').doc(userRef.id).update({

posts: [ postRef ],

recipes: [ recipeRef ]

});

console.log('User document updated with post and recipe references.');

// 9. Update the recipe document with a reference to the comment document

await db.collection('Recipes').doc(recipeRef.id).update({

comments: [ commentRef ]

});

console.log('Recipe document updated with comment reference.');

} catch (error) {

console.error("Error seeding Firestore:", error);

}

}

seedFirestore();

#### Explanation

* **Auto-Generated IDs:**  
  Each call to db.collection('...').add({...}) creates a document with a random, auto-generated ID. We then use the returned document reference (e.g., userRef, ingredientRef1) in our subsequent documents.
* **References as Fields:**  
  Instead of storing a simple string (like "user123"), fields such as author in the recipe or user\_id in posts store the entire document reference (e.g., userRef). This ensures that your references point directly to the correct documents.
* **Updating Documents:**  
  After creating documents, we update fields that require arrays of references. For example, once the post and recipe documents are created, we update the user document’s posts and recipes fields to include these references.

#### Running the Code

1. **Set Up:**  
   Follow the steps to install Node.js, set up Firebase Admin SDK, and place your serviceAccountKey.json in your project folder.
2. **Run the Script:**  
   Save the code above in a file (e.g., seed.js) and run:
3. node seed.js

You should see console logs with the generated document IDs and status messages.

This approach automates creating random document IDs and linking related documents with proper references using Firebase Firestore's syntax.

# Prompt 3

## ChatGPT

Using that as a base create 5 more sample documents with different references and sample data for each collection. Make sure if there are comments on a post the comments reference that post and have null for the recipe data and vice versa.

### Result

Below is an extended Node.js script that builds on the previous example. In this script, we create five additional sample documents per collection—with different sample data and random references. In the Comments collection, comments made on posts have their recipe\_id set to null, and comments made on recipes have their post\_id set to null.

var admin = require("firebase-admin");

var serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({

credential: admin.credential.cert(serviceAccount)

});

const db = admin.firestore();

async function seedFirestore() {

try {

// ---------------------------------

// Create 5 sample Users

// ---------------------------------

let userRefs = [];

for (let i = 0; i < 5; i++) {

const userData = {

email: `user${i}@example.com`,

username: `User${i}`,

bio: `This is user ${i}'s bio.`,

pfp: `https://example.com/user${i}.jpg`,

followers: [],

following: [],

created\_at: admin.firestore.FieldValue.serverTimestamp(),

updated\_at: admin.firestore.FieldValue.serverTimestamp(),

posts: [],

recipes: []

};

let userRef = await db.collection('Users').add(userData);

console.log(`User ${i} created with ID: ${userRef.id}`);

userRefs.push(userRef);

}

// ---------------------------------

// Create 5 sample Ingredients

// ---------------------------------

const ingredientNames = ["Cheese", "Basil", "Chicken", "Rice", "Pepper"];

let ingredientRefs = [];

for (let i = 0; i < 5; i++) {

const ingredientData = {

name: ingredientNames[i]

};

let ingRef = await db.collection('Ingredients').add(ingredientData);

console.log(`Ingredient created with ID: ${ingRef.id} (${ingredientNames[i]})`);

ingredientRefs.push(ingRef);

}

// ---------------------------------

// Create 5 sample Recipes

// ---------------------------------

let recipeRefs = [];

for (let i = 0; i < 5; i++) {

// Select a random user as the author.

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

// Pick 2 distinct random ingredients.

let indices = [];

while (indices.length < 2) {

let idx = Math.floor(Math.random() \* ingredientRefs.length);

if (!indices.includes(idx)) indices.push(idx);

}

let randomIngredients = indices.map(idx => {

return { ingredient: ingredientRefs[idx], quantity: `${Math.floor(Math.random() \* 3) + 1} unit(s)` };

});

const recipeData = {

author: randomUserRef, // reference to a user document

title: `Sample Recipe ${i}`,

media\_url: `https://example.com/recipe${i}.jpg`,

ratings: parseFloat((Math.random() \* 5).toFixed(1)),

isRecommended: Math.random() < 0.5,

ingredients: randomIngredients,

instructions: [`Step 1 for recipe ${i}`, `Step 2 for recipe ${i}`],

tags: [`tag${i}`, "sample"],

comments: [],

created\_at: admin.firestore.FieldValue.serverTimestamp(),

updated\_at: admin.firestore.FieldValue.serverTimestamp()

};

let recipeRef = await db.collection('Recipes').add(recipeData);

console.log(`Recipe ${i} created with ID: ${recipeRef.id}`);

recipeRefs.push(recipeRef);

}

// ---------------------------------

// Create 5 sample Fridges

// ---------------------------------

let fridgeRefs = [];

for (let i = 0; i < 5; i++) {

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

// Pick 2 distinct random ingredients for the fridge.

let indices = [];

while (indices.length < 2) {

let idx = Math.floor(Math.random() \* ingredientRefs.length);

if (!indices.includes(idx)) indices.push(idx);

}

let fridgeIngredients = indices.map(idx => {

return { ingredient: ingredientRefs[idx] };

});

const fridgeData = {

user\_id: randomUserRef, // reference to a user document

ingredients: fridgeIngredients,

shopping\_list: ["milk", "eggs", "butter"],

updated\_at: admin.firestore.FieldValue.serverTimestamp()

};

let fridgeRef = await db.collection('Fridges').add(fridgeData);

console.log(`Fridge ${i} created with ID: ${fridgeRef.id}`);

fridgeRefs.push(fridgeRef);

}

// ---------------------------------

// Create 5 sample Posts

// ---------------------------------

let postRefs = [];

for (let i = 0; i < 5; i++) {

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

const postData = {

user\_id: randomUserRef, // reference to a user document

title: `Sample Post ${i}`,

description: `This is sample post number ${i}.`,

created\_at: admin.firestore.FieldValue.serverTimestamp(),

comments: [] // will be updated later with comment references

};

let postRef = await db.collection('Posts').add(postData);

console.log(`Post ${i} created with ID: ${postRef.id}`);

postRefs.push(postRef);

}

// ---------------------------------

// Create Comments

// ---------------------------------

// Create 3 comments on posts (set recipe\_id: null)

for (let i = 0; i < 3; i++) {

const randomPostRef = postRefs[Math.floor(Math.random() \* postRefs.length)];

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

const commentData = {

recipe\_id: null, // Comment on a post

post\_id: randomPostRef, // reference to a post document

user\_id: randomUserRef, // reference to a user document

content: `This is a comment on post number ${i}.`,

created\_at: admin.firestore.FieldValue.serverTimestamp()

};

let commentRef = await db.collection('Comments').add(commentData);

console.log(`Post Comment ${i} created with ID: ${commentRef.id}`);

// Update the post document to include this comment

await db.collection('Posts').doc(randomPostRef.id).update({

comments: admin.firestore.FieldValue.arrayUnion(commentRef)

});

}

// Create 3 comments on recipes (set post\_id: null)

for (let i = 0; i < 3; i++) {

const randomRecipeRef = recipeRefs[Math.floor(Math.random() \* recipeRefs.length)];

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

const commentData = {

recipe\_id: randomRecipeRef, // reference to a recipe document

post\_id: null, // Comment on a recipe

user\_id: randomUserRef, // reference to a user document

content: `This is a comment on recipe number ${i}.`,

created\_at: admin.firestore.FieldValue.serverTimestamp()

};

let commentRef = await db.collection('Comments').add(commentData);

console.log(`Recipe Comment ${i} created with ID: ${commentRef.id}`);

// Update the recipe document to include this comment

await db.collection('Recipes').doc(randomRecipeRef.id).update({

comments: admin.firestore.FieldValue.arrayUnion(commentRef)

});

}

// ---------------------------------

// Create 5 sample Ratings

// ---------------------------------

for (let i = 0; i < 5; i++) {

const randomRecipeRef = recipeRefs[Math.floor(Math.random() \* recipeRefs.length)];

const randomUserRef = userRefs[Math.floor(Math.random() \* userRefs.length)];

const ratingData = {

recipe\_id: randomRecipeRef, // reference to a recipe document

raters: [

{

user\_id: randomUserRef, // reference to a user document

rating: Math.floor(Math.random() \* 5) + 1

}

],

avg\_rating: parseFloat((Math.random() \* 5).toFixed(1)),

created\_at: admin.firestore.FieldValue.serverTimestamp()

};

let ratingRef = await db.collection('Ratings').add(ratingData);

console.log(`Rating ${i} created with ID: ${ratingRef.id}`);

}

console.log("Firestore seeding complete with additional sample documents.");

} catch (error) {

console.error("Error seeding additional sample data:", error);

}

}

seedFirestore();

#### Explanation

1. **Users:**  
   Five users are created with unique emails, usernames, and profile pictures. Their document references are stored in an array for later use.
2. **Ingredients:**  
   Five ingredients are added (e.g., "Cheese", "Basil", etc.). Their references are saved for linking with recipes and fridges.
3. **Recipes:**  
   Five recipes are created. Each recipe randomly picks a user as its author and randomly selects two distinct ingredients. Their ingredients are stored as objects that include a reference to the ingredient and a sample quantity.
4. **Fridges:**  
   Five fridges are created. Each fridge is linked to a random user and includes two random ingredients along with a shopping list.
5. **Posts:**  
   Five posts are created, each linked to a random user.
6. **Comments:**
   * Three comments are created for posts (with the recipe reference set to null). After each comment is created, the corresponding post is updated to include the comment reference.
   * Three comments are created for recipes (with the post reference set to null). The corresponding recipe document is updated with the comment reference.
7. **Ratings:**  
   Five ratings are created where each rating references a random recipe and a random user, with a random rating value.

#### Running the Code

1. Ensure you have Node.js installed and the Firebase Admin SDK installed (via npm install firebase-admin).
2. Place your service account JSON (e.g., serviceAccountKey.json) in your project folder.
3. Save this script in a file (e.g., seed.js).
4. Run the script in your terminal:
5. node seed.js
6. Check your Firestore database (or Emulator) to see the newly created documents and their references.

This script demonstrates how to create multiple sample documents for each collection with proper document references and appropriate null values for cross-referenced fields in comments.