**Firebase Integration Guide for Request Management System**

**Overview**

This guide explains how to connect your request management components to Firebase for handling different request states, real-time updates, and data persistence.

**Architecture Overview**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Components │ │ Services │ │ Firebase │

│ │ │ │ │ │

│ ┌─────────────┐ │ │ ┌─────────────┐ │ │ ┌─────────────┐ │

│ │CreateRequest│ │◄──►│ │requestService│ │◄──►│ │ requests │ │

│ └─────────────┘ │ │ └─────────────┘ │ │ │ collection │ │

│ ┌─────────────┐ │ │ ┌─────────────┐ │ │ └─────────────┘ │

│ │MyRequests │ │◄──►│ │groupRequest │ │◄──►│ ┌─────────────┐ │

│ └─────────────┘ │ │ │ Service │ │ │ │grouprequests│ │

│ ┌─────────────┐ │ │ └─────────────┘ │ │ │ collection │ │

│ │DraftRequests│ │ │ │ │ └─────────────┘ │

│ └─────────────┘ │ │ ┌─────────────┐ │ │ ┌─────────────┐ │

│ ... │ │ │useRequests │ │ │ │requestRes- │ │

└─────────────────┘ │ │ Hook │ │ │ │ponses coll. │ │

│ └─────────────┘ │ │ └─────────────┘ │

└─────────────────┘ └─────────────────┘

**Firebase Collections Structure**

**1. requests Collection (One-to-One Requests)**

{

id: "auto-generated",

userId: "creator-user-id",

userName: "Creator Name",

userAvatar: "avatar-url",

userEmail: "creator@email.com",

// Request Details

topic: "Request Title",

title: "Request Title", // Alternative field

description: "Detailed description",

subject: "Mathematics",

// Scheduling

preferredDate: "2025-08-15",

preferredTime: "10:00",

duration: "60", // in minutes

// Status & Workflow

status: "draft|open|active|completed|archived|cancelled",

visibility: "public|private",

// Payment

paymentAmount: "200.00",

// Participation

participants: ["user-id-1", "user-id-2"],

participantCount: 0,

maxParticipants: 5,

// Metadata

tags: ["beginner", "exam-prep"],

views: 0,

likes: 0,

featured: false,

// Timestamps

createdAt: Timestamp,

updatedAt: Timestamp,

publishedAt: Timestamp,

completedAt: Timestamp,

archivedAt: Timestamp,

// Type identifier

type: "one-to-one"

}

**2. grouprequests Collection (Group Requests)**

{

id: "auto-generated",

userId: "creator-user-id",

createdBy: "creator-user-id",

createdByName: "Creator Name",

// Group Details

targetGroupId: "group-id",

groupId: "group-id", // Alternative field

groupName: "Group Name",

// Request Details

title: "Group Session Title",

description: "Detailed description",

category: "Study Session|Project Help|Exam Prep",

// Session Configuration

sessionType: "group-session|individual-help",

maxParticipants: 10,

minParticipants: 3,

// Status & Workflow

status: "draft|pending|voting\_open|accepted|active|completed|cancelled",

// Voting & Participation

votes: ["user-id-1", "user-id-2"],

voteCount: 0,

participants: ["user-id-1", "user-id-2"],

participantCount: 0,

// Payment

rate: "Rs.500/hour",

paidParticipants: ["user-id-1"],

totalPaid: 0,

// Skills & Tags

skills: ["JavaScript", "React"],

tags: ["frontend", "beginner"],

// Scheduling

deadline: "2025-08-20",

duration: "2 hours",

scheduledDateTime: "2025-08-15T10:00:00Z",

// Metadata

viewCount: 0,

responses: [],

// Timestamps

createdAt: Timestamp,

updatedAt: Timestamp,

votingOpenedAt: Timestamp,

approvedAt: Timestamp,

completedAt: Timestamp,

// Type identifier

type: "group-request"

}

**3. requestResponses Collection (Responses to Requests)**

{

id: "auto-generated",

requestId: "original-request-id",

responderId: "responder-user-id",

responderName: "Responder Name",

responderEmail: "responder@email.com",

status: "pending|accepted|declined|archived",

message: "Response message",

createdAt: Timestamp

}

**Component Integration**

**1. CreateRequest Component**

**File**: src/pages/Requests/CreateRequest.jsx

**Features**:

* Create one-to-one requests
* Save as draft functionality
* Real-time form validation
* Auto-save capabilities

**Usage**:

import { requestService } from '@/services/requestService';

// Create new request

const result = await requestService.createRequest(formData, userId, isDraft);

// Save as draft

const result = await requestService.saveDraft(formData, userId, requestId);

// Publish draft

const result = await requestService.publishDraft(requestId, userId);

**2. MyRequests Component**

**File**: src/pages/Requests/MyRequests.jsx

**Features**:

* View all user's requests (both one-to-one and group)
* Filter by type and status
* Real-time updates
* Action buttons for each request

**Usage**:

import { useMyRequests } from '@/hooks/useRequests';

const { requests, stats, loading, error } = useMyRequests();

**3. DraftRequests Component**

**File**: src/pages/Requests/draftRequest.jsx

**Features**:

* View only draft requests
* Continue editing drafts
* Publish drafts
* Delete drafts

**Usage**:

import { useDraftRequests } from '@/hooks/useRequests';

const { requests, publishDraft, deleteRequest } = useDraftRequests();

**4. Active/Completed Requests**

**Files**: src/pages/Requests/activeRequest.jsx, src/pages/Requests/completedRequest.jsx

**Features**:

* Status-specific request views
* Action buttons based on status
* Statistics and analytics

**5. OneToOneRequests Component**

**File**: src/pages/Requests/OneToOneRequests.jsx

**Features**:

* Browse available requests from other users
* Respond to requests
* Filter and categorize requests

**Custom Hooks**

**useRequests Hook**

**File**: src/hooks/useRequests.js

**Purpose**: Unified hook for managing requests with different configurations

**Variants**:

// Get all user's requests

const { requests, stats } = useMyRequests();

// Get only drafts

const { requests } = useDraftRequests();

// Get only active requests

const { requests } = useActiveRequests();

// Get only completed requests

const { requests } = useCompletedRequests();

// Get available requests from others

const { requests } = useAvailableRequests();

// Get only one-to-one requests

const { requests } = useOneToOneRequests();

// Get only group requests

const { requests } = useGroupRequests();

**Services**

**requestService**

**File**: src/services/requestService.js

**Methods**:

* createRequest(requestData, userId, isDraft)
* updateRequest(requestId, updateData, userId)
* publishDraft(requestId, userId)
* saveDraft(requestData, userId, requestId)
* deleteRequest(requestId, userId)
* changeRequestStatus(requestId, newStatus, userId)
* respondToRequest(requestId, responseData, userId)

**groupRequestService**

**File**: src/services/groupRequestService.js

**Methods**:

* createGroupRequest(requestData, userId)
* updateGroupRequest(requestId, updateData, userId)
* deleteGroupRequest(requestId, userId)
* changeRequestStatus(requestId, newStatus, userId)
* getUserGroupRequests(userId, status)

**Real-time Updates**

All components use Firestore's onSnapshot for real-time updates:

// Example from requestService

const unsubscribe = onSnapshot(requestsQuery, (snapshot) => {

const requests = snapshot.docs.map(doc => ({

id: doc.id,

...doc.data(),

createdAt: doc.data().createdAt?.toDate() || new Date()

}));

callback(requests);

});

// Cleanup

return () => unsubscribe();

**Setup Instructions**

**1. Install Dependencies**

npm install firebase

**2. Firebase Configuration**

Ensure your src/config/firebase.js is properly configured:

import { initializeApp } from 'firebase/app';

import { getFirestore } from 'firebase/firestore';

import { getAuth } from 'firebase/auth';

const firebaseConfig = {

// Your config

};

const app = initializeApp(firebaseConfig);

export const db = getFirestore(app);

export const auth = getAuth(app);

**3. Firestore Security Rules**

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

// Users can read/write their own requests

match /requests/{requestId} {

allow read, write: if request.auth != null &&

(resource == null || resource.data.userId == request.auth.uid);

allow read: if request.auth != null && resource.data.visibility == 'public';

}

// Group requests - members can read, creators can write

match /grouprequests/{requestId} {

allow read: if request.auth != null;

allow write: if request.auth != null &&

(resource == null || resource.data.userId == request.auth.uid);

}

// Request responses

match /requestResponses/{responseId} {

allow read, write: if request.auth != null &&

(resource == null || resource.data.responderId == request.auth.uid);

allow read: if request.auth != null &&

resource.data.requesterId == request.auth.uid;

}

}

}

**4. Component Integration**

Replace your existing components with the updated versions:

// In your route configuration

import CreateRequest from './pages/Requests/CreateRequest';

import MyRequests from './pages/Requests/MyRequests';

import DraftRequests from './pages/Requests/draftRequest';

import ActiveRequests from './pages/Requests/activeRequest';

import CompletedRequests from './pages/Requests/completedRequest';

import OneToOneRequests from './pages/Requests/OneToOneRequests';

// Routes

<Route path="/requests/create" element={<CreateRequest />} />

<Route path="/requests/my-requests" element={<MyRequests />} />

<Route path="/requests/drafts" element={<DraftRequests />} />

<Route path="/requests/active" element={<ActiveRequests />} />

<Route path="/requests/completed" element={<CompletedRequests />} />

<Route path="/OneToOneRequests" element={<OneToOneRequests />} />

**Data Flow**

**Creating a Request**

1. User fills form in CreateRequest component
2. Click "Save Draft" → requestService.saveDraft() → Firestore with status: 'draft'
3. Click "Publish" → requestService.createRequest() → Firestore with status: 'open'

**Viewing Requests**

1. Component mounts → Custom hook (useMyRequests, etc.)
2. Hook calls service method → Sets up Firestore listener
3. Real-time updates → Component re-renders automatically

**Managing Request States**

* **Draft** → **Open/Active** (Publish)
* **Open/Active** → **Completed** (Mark as complete)
* **Open/Active** → **Archived** (Archive)
* **Any Status** → **Deleted** (Delete permanently)

**Error Handling**

All services include comprehensive error handling:

try {

const result = await requestService.createRequest(data, userId);

if (result.success) {

// Handle success

} else {

// Handle service-level error

alert(result.message);

}

} catch (error) {

// Handle unexpected errors

console.error('Error:', error);

alert('An unexpected error occurred');

}

**Performance Optimizations**

1. **Pagination**: Implement for large request lists
2. **Indexes**: Create Firestore indexes for common queries
3. **Caching**: Use React Query or SWR for client-side caching
4. **Lazy Loading**: Load request details only when needed

This integration provides a complete, real-time request management system with proper state handling, draft functionality, and seamless Firebase integration.