REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

UNIVERSITY OF BUEA

REPUBLIQUE DU CAMEROUN

PAIX-TRAVAIL-PATRIE

UNIVERSITE DE BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

COURSE CODE/TITLE: CEF 440 INTERNET PROGRAMMING AND MOBILE PROGRAMMING

COURSE INSTRUCTOR: DR. NKEMENI VALERY

PROJECT TITLE:

DATABASE DESIGN AND IMPLEMENTATION OF THE ROAD SIGN AND ROAD STATE MOBILE NOTIFICATION APPLICATION

GROUP 9

PRESENTED BY:

AMINATU MOHAMMED AWAL	FE22A147
DEFANG MARGARET AKUMAYUK	FE22A185
EKWOGE JUNIOR	FE22A200
ESIMO GODWILL EYABI	FE22A207
SUH AKUMAH TILUI-NTONG	FE22A299

TABLE OF CONTENTS

1. **INTRODUCTION:**

In the development of the Road Sign and Road State Mobile Notification Application, **RoadBro,** an efficient, scalable, and secure database design is critical. This system is aimed at notifying users (drivers, pedestrians, transport authorities) of road signs, hazards, and road states in real time. To achieve seamless data flow between the mobile frontend and the backend, **Firebase Firestore**—a NoSQL, real-time cloud database—was chosen for its ease of integration with React Native and robust support for both structured and semi-structured data.

2. **DATA ELEMENTS:**

This section includes the entities, attributes and the relationships that exists between them.

I. **ENTITIES AND ATTRIBUTES:**

A. **Entity: User**

This entity represents individual users of the system.

Attributes:

◦ userID: Unique identifier for each user.
◦ firstName: The user's first name.
◦ lastName The user’s last name.
◦ email: The user's email address (likely unique and used for login).
◦ password: The user's hashed or encrypted password.
◦ phoneNumber: The user’s contact details.
◦ role: User’s role in the system which can either be a road user or an admin.
◦ location: The geographical location associated with the user.
◦ languagePreference: User’s preferred language.
◦ notificationSettings: User-specific notifications settings or preferences.
◦ profilePictureURL: Image of the user.
◦ createdAt: Date of creation of the user.
◦ updatedAt: Date user’s personal information is updated.

B. **Entity: Road Sign**

This entity stores information about road signs reported or managed within the system.

Attributes:

◦ signID: Unique identifier for each road sign record.

◦ signName: Name of the road sign.

◦ signType: The type or classification of the road sign (e.g., stop, yield, warning).

◦ description: A textual description of the road sign.

◦ imageURL: The URL or path to an image of the road sign.

◦ status: Indicating if the road sign information has been verified.

◦ category: The category to which the road sign belongs(e.g., order, informational, directional, danger).

C. **Entity: Road State:**

This entity represents real-world road conditions tagged by user reports.

Attributes:

◦ stateID: Unique identifier for each road state.

◦ stateType: Type of the road state (e.g pothole, flooding, landslide).

◦ location: The geographical coordinates or description of the road section.

◦ timestamp: The time and date when the road state was reported or last updated.

◦ severity: The level of severity of the road condition (e.g., low, medium, high).

◦ reportUserID: Unique identifier of the user who submits a road state report.

D. **Entity: NOTIFICATION**

These are alerts sent to users regarding road conditions.

Attributes:

◦ notificationID: Unique identifier for each notification.

◦ notificationUserID: The ID of the User to whom the notification is sent.

◦ notificationType: The type of notification e.g (road sign alert, road state alert)

◦ message: The textual content of the notification.

◦ readStatus: A boolean flag indicating whether the user has read the notification.

◦ timestamp: The date and time when the notification was created.

E. **Entity: REPORT**

These are submitted by users to inform about road conditions..

Attributes:

◦ <u>reportID</u>: Unique identifier for each user report.

◦ <u>reportUserID</u>: The ID of the User who submitted the report.

◦ <u>reportType:</u> The classification of the report (e.g., 'Bug', 'Suggestion', 'Complaint').

◦ <u>description:</u> A detailed textual description of the report.

◦ <u>timestamp</u>: The date and time when the report was submitted.

◦ <u>createdAt:</u> Submission date of the report.

◦ <u>updatedAt:</u> Date report is updated.

F. **Entity: ADMIN**

User who manages reports and system data.

Attributes:

◦ <u>adminId:</u> Unique identifier of the admin.

◦ <u>adminCode:</u> Secret code of the admin.

◦ <u>name:</u> Name of the admin.

◦ <u>email:</u> The admin's email.

II. **Relationships:**

- **"Receives" Notification**: A User can receive multiple Notifications.
- **"Submits" Report**: A User can submit multiple Reports.
- **"Covers" RoadSign**: RoadSigns are located and mapped to physical locations.
- **"Monitors" RoadState**: RoadState information is updated and used for monitoring road conditions.

3. CONCEPTUAL DESIGN:

The conceptual design focuses on the logical structure of the data and how the entities are interrelated within the Firestore database. Given that Firestore is a NoSQL document-oriented database, we structure our data using collections and documents, where each document can contain nested fields or references.

A. **Design Highlights:**

- Each entity is modeled as a Firestore collection containing documents with attributes.
- Relationships are maintained through document references using unique IDs (e.g., userId in the Reports collection refers to a specific document in Users).
- Firebase's real-time syncing and offline capabilities are leveraged for smooth user experience.
- All data operations (e.g., read/write) are optimized through indexed fields such as userId, timestamp, and location.
- Collections are designed to be scalable with support for future expansion (e.g., addition of road quality metrics, admin-level features).

B. **Main Entities (Collections):**

- **Users**: Represents all users of the system (drivers and admins).
- **Reports**: Stores road incidents submitted by users.
- **Notifications**: Alerts sent to users, either directly or system-wide.
- **RoadSigns**: Contains static road signs and their physical locations.
- **RoadStates**: Represents the condition of roads over time.

C. **Relationships Between Entities:**

- Users **submit** reports about road conditions and hazards.
- Users **receive** specific notifications based on their preferences.
- Reports are enriched with images, locations, and timestamps for context.
- RoadSigns represent traffic signs, mapped to specific coordinates.
- RoadStates provide a real-time view of road conditions across different areas.

This conceptual model ensures a normalized and flexible structure that supports the core functionalities of the app such as dynamic alerting, location-based filtering, and user-generated content.

4. **ENTITY-RELATIONSHIP(ER) DIAGRAM:**

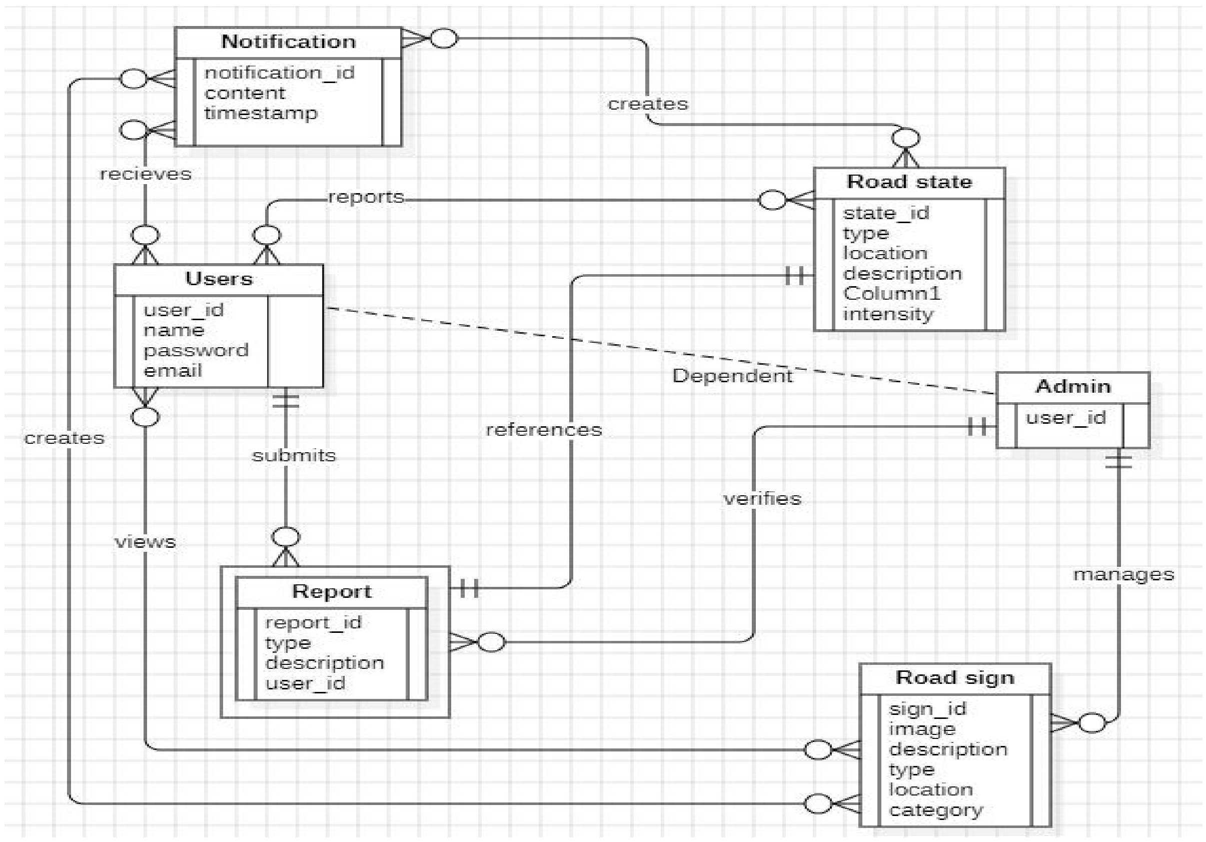
The Entity-Relationship (ER) diagram provides a high-level visual representation of how different data entities within the Road Sign and Road State Mobile Notification Application interact with each other in the database. It identifies the primary entities—**Users, Reports, Notifications, RoadSigns**, and **RoadStates**—and illustrates their inter-relationships through well-defined associations.

ER Diagram Legend:

- **Rectangle**: Entity
- **Oval**: Attribute
- **Diamond**: Relationship
- **Underlined attributes**: Primary Keys
- **Dashed oval**: Derived attributes
- **Lines with notation**: Cardinality (1:1, 1:M, M:N)

Key Relationships with Cardinality:

1. **User ——(1:M)——> Report**
    - One user can submit many reports
    - Each report belongs to exactly one user
2. **User ——(1:M)——> Notification**
    - One user can receive many notifications
    - Each notification is sent to one specific user
3. **User ——(1:M)——> RoadState**
    - One user can report many road states
    - Each road state report is submitted by one user



Detailed Entity Specifications:

1. **User Entity:**

- Primary Key: userID
- Unique Constraints: email
- Indexed Fields: email, location, role

2. **RoadSign Entity:**

- Primary Key: signID
- Indexed Fields: signType, category, location

3. **RoadState Entity:**

- Primary Key: stateID
- Foreign Key: reportUserID (references User.userID)
- Indexed Fields: location, timestamp, severity

4. **Notification Entity:**

- Primary Key: notificationID
- Foreign Key: notificationUserID (references User.userID)
- Indexed Fields: timestamp, readStatus

5. **Report Entity:**

- Primary Key: reportID
- Foreign Key: reportUserID (references User.userID)
- Indexed Fields: timestamp, reportType

5. DATABASE IMPLEMENTATION:

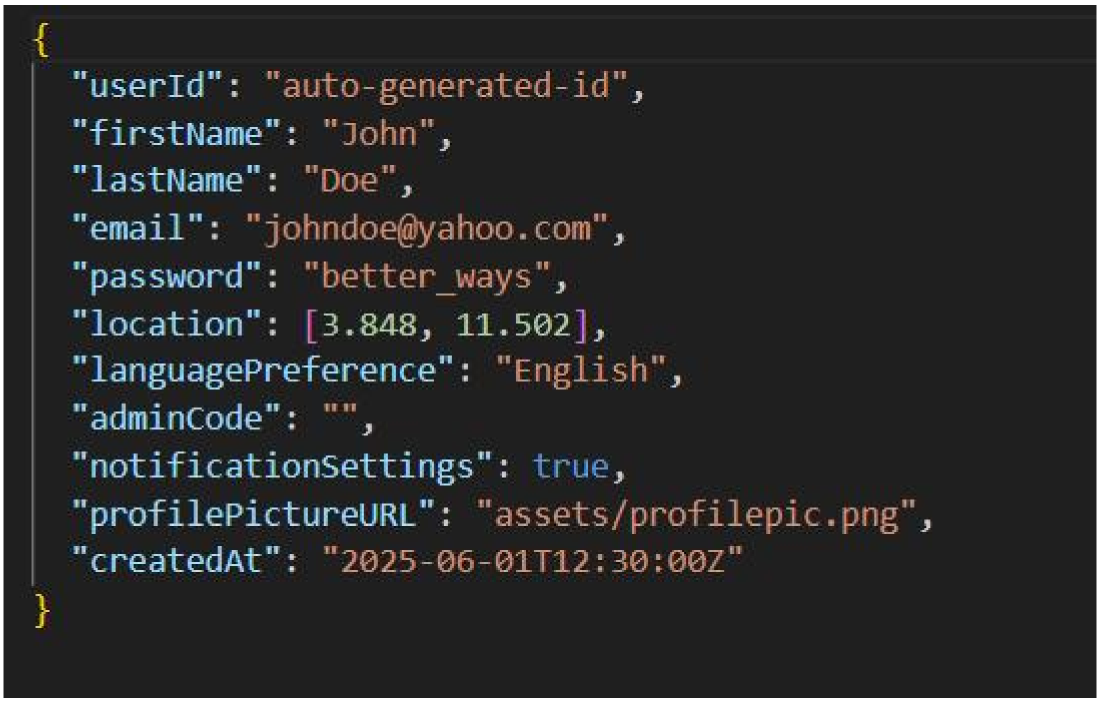
The backend database is implemented using **Firebase Firestore**. This is because it offers:

- **NoSQL Flexibility**: Data doesn't need to follow rigid schema. This suits mobile-first apps with dynamic fields like reports and alerts.
- **Real-time Updates**: Automatically syncs user-facing data such as reports, road signs, or notifications.
- **Offline Access**: Allows users to view previously loaded data even when offline.
- **Geo-queries**: Works well with location-based filtering using libraries like geofirestore.
- **Data Validation & Indexing:**

  - Firebase rules ensure only authenticated users can write to certain collections.

  - Fields like timestamp, userId, and location can be indexed for performance.

Example: **User Document**



```json
{
  "userId": "auto-generated-id",
  "firstName": "John",
  "lastName": "Doe",
  "email": "johndoe@yahoo.com",
  "password": "better_ways",
  "location": [3.848, 11.502],
  "languagePreference": "English",
  "adminCode": "",
  "notificationSettings": true,
  "profilePictureURL": "assets/profilepic.png",
  "createdAt": "2025-06-01T12:30:00Z"
}
```

6. BACKEND IMPLEMENTATION:

Architecture Overview

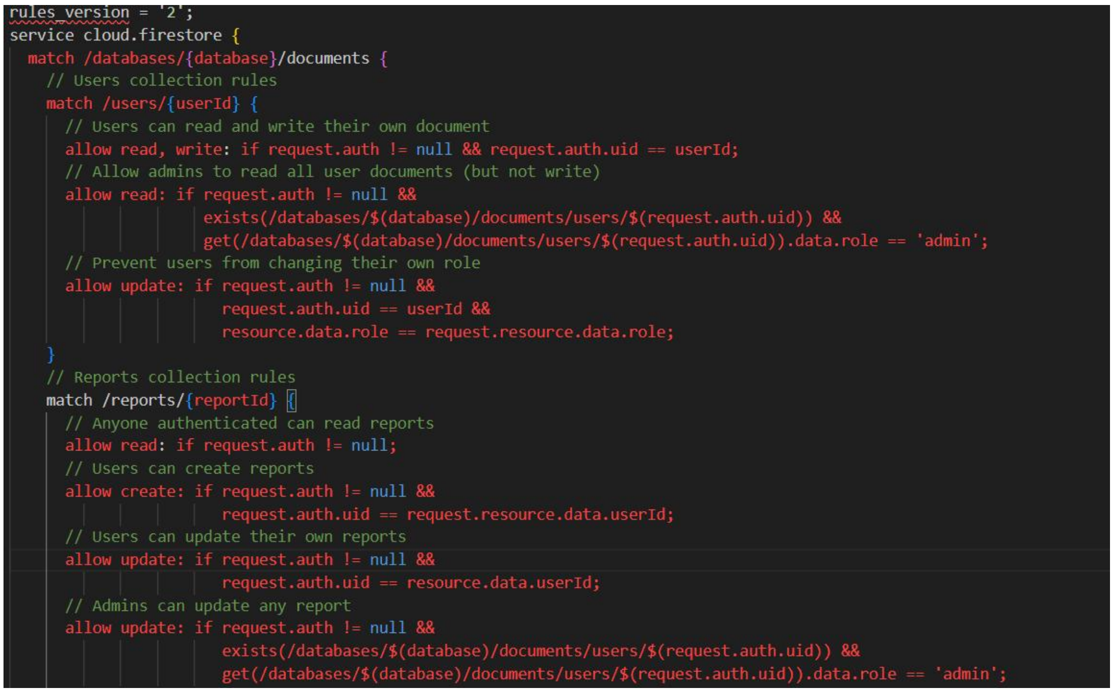
The backend follows a serverless architecture using Firebase services, providing scalability and real-time capabilities essential for a mobile notification system.

Technology Stack:

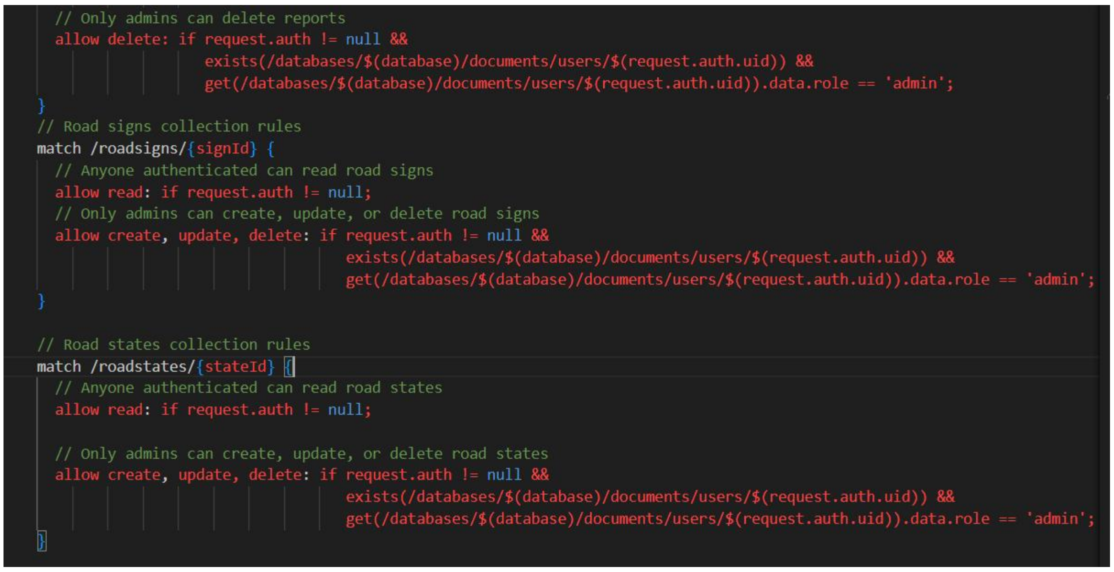
- **Firebase Authentication**: Handles user registration, login, and session management
- **Cloud Firestore**: NoSQL database for real-time data storage and synchronization
- **Cloud Functions**: Server-side logic for complex operations and triggers
- **Firebase Cloud Messaging (FCM)**: Push notifications to mobile devices
- **Firebase Storage**: Image storage for road signs and report attachments

Core Backend Services:

1. **Security Rules Implementation:**



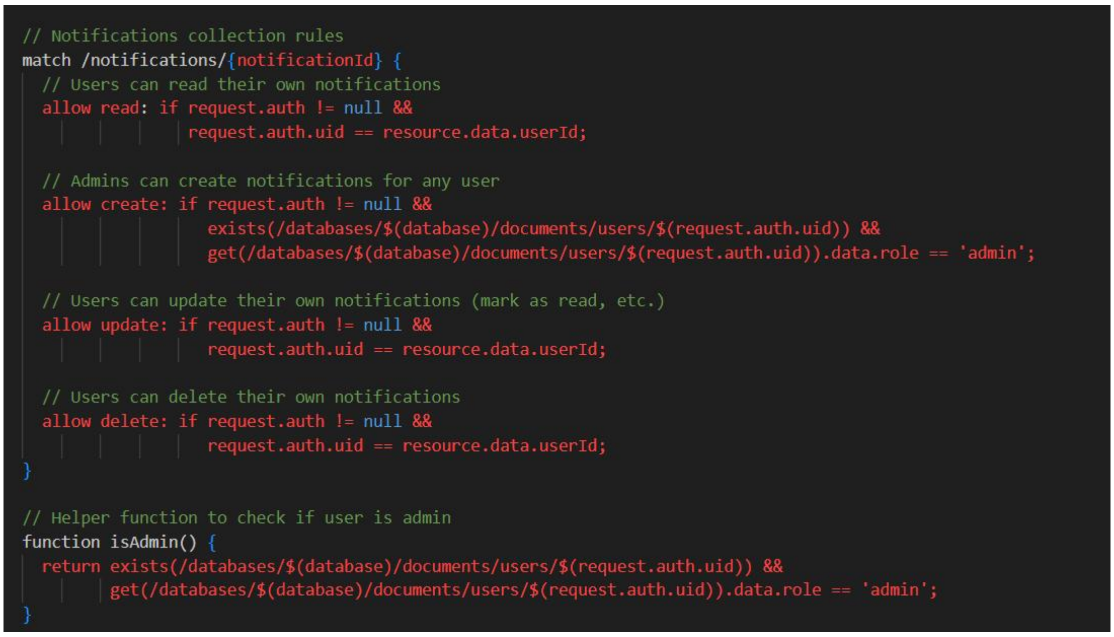
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users collection rules
    match /users/{userId} {
      // Users can read and write their own document
      allow read, write: if request.auth != null && request.auth.uid == userId;
      // Allow admins to read all user documents (but not write)
      allow read: if request.auth != null &&
                     exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                     get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
      // Prevent users from changing their own role
      allow update: if request.auth != null &&
                       request.auth.uid == userId &&
                       resource.data.role == request.resource.data.role;
    }
    // Reports collection rules
    match /reports/{reportId} {
      // Anyone authenticated can read reports
      allow read: if request.auth != null;
      // Users can create reports
      allow create: if request.auth != null &&
                       request.auth.uid == request.resource.data.userId;
      // Users can update their own reports
      allow update: if request.auth != null &&
                       request.auth.uid == resource.data.userId;
      // Admins can update any report
      allow update: if request.auth != null &&
                       exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                       get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
```



```
      // Only admins can delete reports
      allow delete: if request.auth != null &&
                    exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                    get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
    }
    // Road signs collection rules
    match /roadsigns/{signId} {
      // Anyone authenticated can read road signs
      allow read: if request.auth != null;
      // Only admins can create, update, or delete road signs
      allow create, update, delete: if request.auth != null &&
                                    exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                                    get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
    }

    // Road states collection rules
    match /roadstates/{stateId} {
      // Anyone authenticated can read road states
      allow read: if request.auth != null;

      // Only admins can create, update, or delete road states
      allow create, update, delete: if request.auth != null &&
                                    exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                                    get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
    }
```



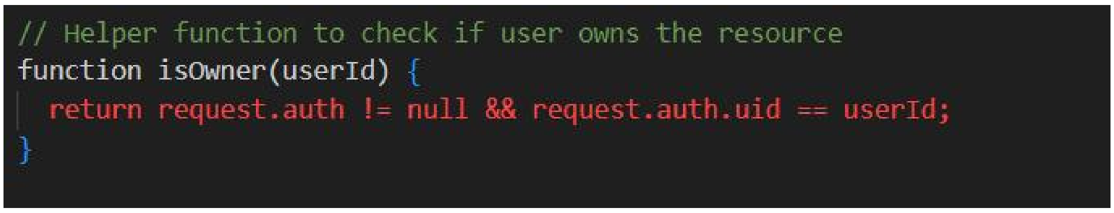
```
// Notifications collection rules
match /notifications/{notificationId} {
  // Users can read their own notifications
  allow read: if request.auth != null &&
              request.auth.uid == resource.data.userId;

  // Admins can create notifications for any user
  allow create: if request.auth != null &&
                exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
                get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';

  // Users can update their own notifications (mark as read, etc.)
  allow update: if request.auth != null &&
                request.auth.uid == resource.data.userId;

  // Users can delete their own notifications
  allow delete: if request.auth != null &&
                request.auth.uid == resource.data.userId;
}

// Helper function to check if user is admin
function isAdmin() {
  return exists(/databases/$(database)/documents/users/$(request.auth.uid)) &&
      get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
}
```



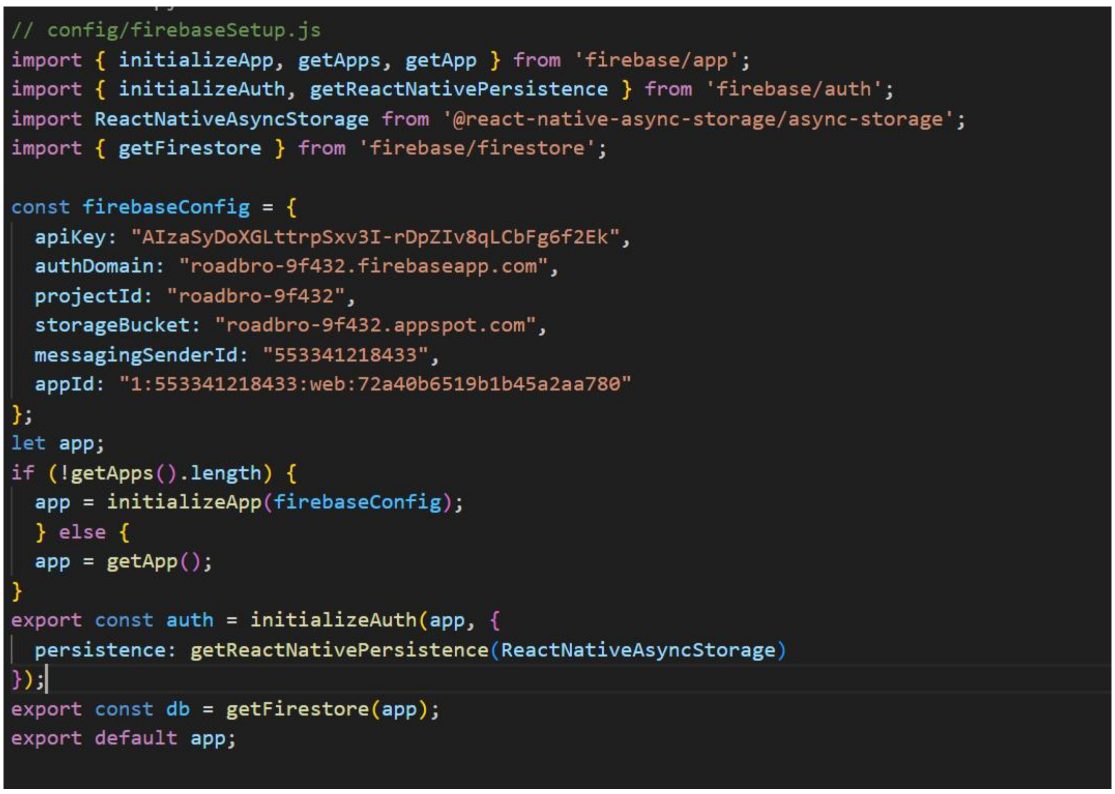
```
// Helper function to check if user owns the resource
function isOwner(userId) {
  return request.auth != null && request.auth.uid == userId;
}
```

7. CONNECTING DATABASE TO BACKEND:

Frontend-Backend Integration Architecture

The connection between the React Native frontend and Firebase backend follows a service-oriented architecture pattern, ensuring clean separation of concerns and maintainable code.

1. **Firebase Configuration and Initialization**



```
// config/firebaseSetup.js
import { initializeApp, getApps, getApp } from 'firebase/app';
import { initializeAuth, getReactNativePersistence } from 'firebase/auth';
import ReactNativeAsyncStorage from '@react-native-async-storage/async-storage';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: "AIzaSyDoXGLttrpSxv3I-rDpZIv8qLCbFg6f2Ek",
  authDomain: "roadbro-9f432.firebaseapp.com",
  projectId: "roadbro-9f432",
  storageBucket: "roadbro-9f432.appspot.com",
  messagingSenderId: "553341218433",
  appId: "1:553341218433:web:72a40b6519b1b45a2aa780"
};
let app;
if (!getApps().length) {
  app = initializeApp(firebaseConfig);
  } else {
  app = getApp();
}
export const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(ReactNativeAsyncStorage)
});
export const db = getFirestore(app);
export default app;
```

8. **CHALLENGES AND SOLUTIONS:**

Challenge	**Solution**
Handling different roles	Used Firestore role field + onAuthStateChanged with conditional navigation
Avoiding duplicate alerts	Added Cloud Function to check for existing active alerts in the same area.
Ensuring scalability	Chose NoSQL design with normalized but flexible document structure

9. **CONCLUSION:**

The database design and implementation for the **Road Sign and Road State Notification App** offers a robust, scalable, and secure backbone for all app functionalities. Using Firebase Firestore allows real-time updates and seamless integration with the React Native frontend. The use of role-based access, clear entity relationships, and modular backend functions ensures that both users and administrators can interact with the system efficiently. This well-structured approach guarantees a reliable platform for improving road safety awareness in Cameroon and beyond.