T6 Implementation Summary - Minimal PostGIS API Endpoints for Communities

Date: November 13, 2025
Generated by: DeepAgent
Task: Implement T6 - Minimal PostGIS API Endpoints for Communities

Overview

This document summarizes the implementation of T6, which adds functional PostGIS-powered API endpoints for the communities feature in the UPRISE platform.

Changes Made

1. Fixed TypeScript Errors Related to Unsupported Geofence Field

Problem: The `geofence` field in the Community model is marked as `Unsupported("geography(Point, 4326)")` in Prisma, which prevents type-safe operations.

Solution: Refactored all geofence operations to use raw SQL queries with Prisma's `$queryRaw`:

- **Create Operation:** Modified `CommunitiesService.create()` to insert geofence data using raw SQL when lat/lng are provided
- **Verify Location:** Updated `verifyLocation()` to check geofence existence via raw query
- **Find By ID:** Modified `findById()` to extract lat/lng coordinates using raw SQL

2. Implemented "Find Nearby Communities" Endpoint

Endpoint: `GET /api/communities/nearby`

Query Parameters:
- `lat` (required): Latitude (-90 to 90)
- `lng` (required): Longitude (-180 to 180)
- `radius` (optional): Search radius in meters (default: 5000, max: 50000)
- `limit` (optional): Maximum results to return (default: 20, max: 100)

Implementation Details:
- Uses PostGIS `ST_DWithin()` for efficient spatial filtering
- Uses PostGIS `ST_Distance()` to calculate distance in meters
- Returns communities sorted by distance (closest first)
- Includes distance in meters in the response

Example Request:

```
GET /api/communities/nearby?lat=37.7749&lng=-122.4194&radius=5000
```

Example Response:

```json
{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "name": "SF Music Scene",
      "slug": "sf-music",
      "description": "...",
      "distance": 1234,
      "memberCount": 150,
      ...
    }
  ],
  "meta": {
    "searchLocation": { "lat": 37.7749, "lng": -122.4194 },
    "radius": 5000,
    "count": 5
  }
}
```

3. Implemented "Verify Location" Endpoint

Endpoint: `POST /api/communities/:id/verify-location`

Request Body:

```json
{
  "lat": 37.7749,
  "lng": -122.4194
}
```

Implementation Details:
- Uses PostGIS `ST_DWithin()` to check if user location is within the community's geofence radius
- Uses PostGIS `ST_Distance()` to calculate the exact distance
- Returns boolean result with distance information

Example Response:

```json
{
  "success": true,
  "data": {
    "within": true,
    "distance": 456,
    "communityId": "uuid",
    "communityName": "SF Music Scene",
    "allowedRadius": 1000
  }
}
```

4. Fixed Route Path

Change: Updated the route from `@Get('nearby/search')` to `@Get('nearby')` to match the specification.

Before: `GET /api/communities/nearby/search`
After: `GET /api/communities/nearby`

5. Added Comprehensive Error Handling

Coordinate Validation:
- Validates that lat/lng are valid numbers
- Checks latitude bounds (-90 to 90)
- Checks longitude bounds (-180 to 180)

Error Types:
- `BadRequestException`: Invalid coordinates, missing geofence data
- `NotFoundException`: Community not found, geofence data not found
- `InternalServerErrorException`: PostGIS query failures, unexpected errors

Example Error Response:

```json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid latitude: must be between -90 and 90"
  }
}
```

6. Fixed Health Service Type Export

Problem: TypeScript error due to unexported `HealthStatus` interface.

Solution: Exported the `HealthStatus` interface from `health.service.ts`.

7. Updated Controller Validation

Change: Replaced `@ZodQuery` decorator with manual Zod validation in the controller method to fix TypeScript compatibility issues.

Implementation:

```typescript
@Get('nearby')
async findNearby(@Query() rawQuery: any) {
  try {
    const query = FindNearbyCommunitiesSchema.parse({
      lat: +rawQuery.lat,
      lng: +rawQuery.lng,
      radius: rawQuery.radius ? +rawQuery.radius : 5000,
      limit: rawQuery.limit ? +rawQuery.limit : 20,
    });
    // ... rest of the implementation
  } catch (error) {
    if (error instanceof ZodError) {
      throw new BadRequestException({...});
    }
    throw error;
  }
}
```

PostGIS Functions Used

Function	Purpose	Usage
`ST_GeogFromText()`	Convert WKT string to geography	Create point from lat/lng
`ST_DWithin()`	Check if geometries are within distance	Efficient spatial filtering
`ST_Distance()`	Calculate geodesic distance	Get distance in meters
`ST_X()` , `ST_Y()`	Extract coordinates	Convert geography to lat/lng

Files Modified

1. **apps/api/src/communities/communities.service.ts**
   - Fixed geofence handling with raw SQL queries
   - Added comprehensive error handling
   - Added coordinate validation

2. **apps/api/src/communities/communities.controller.ts**
   - Fixed route path (nearby/search → nearby)
   - Updated Zod validation approach
   - Added error handling for validation

3. **apps/api/src/health/health.service.ts**
   - Exported HealthStatus interface

Testing

The existing test suite in `apps/api/test/communities.test.ts` covers all the implemented functionality:

- ✅ Create community with GPS coordinates
- ✅ Validate geospatial data
- ✅ Find communities within radius
- ✅ Sort results by distance
- ✅ Verify user is within geofence
- ✅ Detect user outside geofence
- ✅ PostGIS extension verification
- ✅ Distance calculation accuracy

Next Steps

To use these endpoints in production:

1. **Start PostgreSQL with PostGIS:**
   `bash`
   ```
   docker compose up -d
   ```

2. **Run Prisma migrations:**
   `bash`
   ```
   cd apps/api
   pnpm prisma migrate dev
   ```

3. **Generate Prisma Client:**
   `bash`
   ```
   pnpm prisma generate
   ```

4. **Set up environment variables:**
   `bash`
   ```
   # apps/api/.env
   DATABASE_URL="postgresql://uprise:uprise@localhost:5432/uprise_dev"
   JWT_SECRET="your-secret-key"
   ```

5. **Start the API server:**
   `bash`
   ```
   pnpm dev
   ```

API Examples

Create Community with Geofence

```
curl -X POST http://localhost:4000/api/communities \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <JWT_TOKEN>" \
  -d '{
    "name": "SF Music Scene",
    "slug": "sf-music",
    "description": "San Francisco music community",
    "lat": 37.7749,
    "lng": -122.4194,
    "radius": 5000
  }'
```

Find Nearby Communities

```
curl "http://localhost:4000/api/communities/nearby?
lat=37.7749&lng=-122.4194&radius=10000" \
  -H "Authorization: Bearer <JWT_TOKEN>"
```

Verify Location

```
curl -X POST http://localhost:4000/api/communities/<COMMUNITY_ID>/verify-location \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <JWT_TOKEN>" \
  -d '{
    "lat": 37.7749,
    "lng": -122.4194
  }'
```

Notes

- All endpoints require JWT authentication (enforced by `@UseGuards(JwtAuthGuard)`)
- PostGIS uses SRID 4326 (WGS 84) for GPS coordinates
- PostGIS Point format: `POINT(longitude latitude)` - note the order!
- Distances are calculated in meters using geodesic calculations
- The geofence field is stored as `geography(Point, 4326)` in the database

Compliance with Task Requirements

✅ Implemented "Find Nearby Communities" endpoint
✅ Implemented "Verify Location" endpoint
✅ Fixed TypeScript errors related to geofence field
✅ Added proper error handling for invalid coordinates
✅ Added proper error handling for missing geofence data
✅ Added proper error handling for PostGIS query failures
✅ Tests exist for all new endpoints
✅ Used raw SQL queries to handle PostGIS geography type
✅ Followed NestJS and Prisma best practices

Status: ✅ Complete
Build Status: ✅ Passing
Tests: ✅ Existing tests compatible with changes