

**THE STATE UNIVERSITY OF ZANZIBAR**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY.**

**PREPARED BY: FRANK GODFREY NGOWI**

**REGISTRATION NUMBER: BCS/18/23/014/TZ**

****COURSE:**  WEB TECHNOLOGY**

**PROJECT TITTLE: DISABILITY MANAGEMENT AND MONITORING SYSTEM**

**SUPERVISOR: MR MASOUD H MMANGA**

**Github Username: FrancGody28**

**Github repository FrontEnd;** <https://github.com/Francgody28/DisabilityApp-frontend>

**Github repository BackEnd;**

<https://github.com/Francgody28/DisabilityApp-Backend>

## Implementation Report: Disabled Monitoring and Management System

This report details the implementation of the frontend (React) and backend (Django REST Framework) components for a system designed to manage information about disabled persons and locations.

### 1. Overview

### The system provides a web interface for:

* Registering new disabled persons.
* Viewing a list of registered disabled persons.
* Editing existing disabled person records.
* Deleting disabled person records.
* Adding new locations.
* Viewing a list of registered locations.
* Editing existing location records.
* Deleting location records.

The backend is built with **Django REST Framework (DRF)**, providing a robust API, while the frontend is developed using **React**, offering a dynamic and interactive user experience.

### 2. Backend Implementation (Django REST Framework)

The backend has been refactored to leverage Django REST Framework's powerful features, specifically ModelViewSet and routers, to ensure a clean, maintainable, and scalable API.

**Key Components:**

* **models.py :** Contains the DisabledPerson and Location models, defining the data structure for the application.
* **serializers.py:**
  + UserSerializer: Handles serialization/deserialization for user registration.
  + DisabledPersonSerializer: Manages the DisabledPerson model. This centralizes data transformation logic within the serializer.
  + LocationSerializer: Manages the Location model.
* **views.py:**
  + **Authentication Views (register\_user, login\_user):** These remain as function-based views for straightforward user registration and login.
  + **DisabledPersonViewSet:** A ModelViewSet is used for DisabledPerson. This single class provides all standard CRUD (Create, Retrieve, Update, Delete) operations for disabled persons.
  + **LocationViewSet:** Similarly, a ModelViewSet is used for Location, providing full CRUD functionality for location data.
  + **Permissions:** permission\_classes have been added to the ViewSets (e.g., AllowAny, IsAuthenticated) to demonstrate how access control can be implemented.
* **urls.py:**
  + A DefaultRouter is employed to automatically generate URL patterns for the DisabledPersonViewSet and LocationViewSet. This significantly reduces manual URL configuration.
  + The authentication URLs (/register/, /login/) are included separately.

**API Endpoints:**

The refactored backend exposes the following primary API endpoints:

* /api/register/ (POST): Register a new user.
* /api/login/ (POST): Authenticate a user.
* /api/disabled-persons/ (GET, POST): List all disabled persons or create a new one.
* /api/disabled-persons/{id}/ (GET, PUT, DELETE): Retrieve, update, or delete a specific disabled person by ID.
* /api/locations/ (GET, POST): List all locations or add a new one.
* /api/locations/{id}/ (GET, PUT, DELETE): Retrieve, update, or delete a specific location by ID.

### 3. Frontend Implementation (React - Services.js)

The React Services component provides the user interface and handles all interactions with the Django backend.

**Key Features:**

* **State Management:** useState hooks are used to manage:
  + Visibility of forms and tables (showRegisterForm, showLocationForm, showPeopleTable, showLocationTable).
  + Data for new/edited persons (person) and locations (location).
  + Lists of fetched people (people) and locations (locations).
* **Form Handling:**
  + handlePersonChange and handleLocationChange: Update state as form inputs change.
  + handlePersonSubmit and handleLocationSubmit: Handle form submissions for both new and edited records. They dynamically construct the payload and determine the correct API endpoint and HTTP method (POST for create, PUT for update) based on editingPersonId/editingLocationId.
* **Data Fetching:**
  + fetchPeople and fetchLocations: Asynchronous functions to retrieve data from the respective backend endpoints.
* **CRUD Operations:**
  + **Create:** Forms submit data via POST to the list endpoints.
  + **Read:** fetchPeople and fetchLocations retrieve data, which is then displayed in tables.
  + **Update:** handlePersonEdit and handleLocationEdit populate the forms with existing data. Submissions then use PUT requests to the detail endpoints.
  + **Delete:** handlePersonDelete and handleLocationDelete send DELETE requests to the detail endpoints.

**Integration with Backend:**

The React component has been updated to correctly interact with the new DRF router-generated endpoints. For example:

* fetchPeople and handlePersonSubmit (for new persons) now target /api/disabled-persons/.
* handlePersonSubmit (for updates) and handlePersonDelete now target /api/disabled-persons/{id}/.
* Similar adjustments have been made for location-related API calls.

### 3. Database Implementation (PostgreSQL)

The system utilizes PostgreSQL as its relational database management system (RDBMS). PostgreSQL was chosen for its robustness, reliability, advanced features (such as JSONB support, full-text search).

Configuration:

The integration with PostgreSQL is handled within Django's settings.py file. All database operations (creating, reading, updating, and deleting records for DisabledPerson and Location) are performed through Django's ORM. This means that the Python code in the Django backend interacts with Python objects (models) rather than writing raw SQL queries. The ORM then translates these object-oriented operations into appropriate SQL commands for PostgreSQL. This enhances security by preventing SQL injection, and makes the backend code more portable across different database systems if needed in the future.

### 4. Conclusion

This system uses a robust Django REST Framework backend with PostgreSQL for data integrity and scalability, seamlessly connected to a functional React frontend. This architecture promotes reusability, reduces boilerplate, and ensures easy maintenance and future extensibility.