Detailed Design Document

Database

1. Purpose

The TrackED backend database stores everything needed to track nursing students in clinicals: who they are, who their professors are, where they are scheduled to be, and whether any incidents occurred during an assignment. The database is implemented with Entity Framework Core and SQLite and is exposed through Web API controllers so the front-end can create, read, update, and delete records.

2. Tech stack

Entity Framework Core with code-first migrations

SQLite database file named app.db under the Data folder

ASP.NET Core Web API controllers for CRUD endpoints

No manual SQL scripts are required. The schema is created and updated through EF migrations.

3. Running the project and creating the database

1. Make sure .NET 8 SDK is installed.
2. Open a terminal in the TrackEd project folder (where TrackEd.csproj lives).
3. If you want a clean database, delete Data/app.db.
4. Run: `dotnet run`

On startup, DbSeeding/DbSeeder.cs runs Database.Migrate() and seeds the database with a small set of test data. This will create app.db if it does not exist and apply any pending migrations.

The API base URL will be printed in the console, usually something like:

https://localhost:5287

4. Core tables

The main tables used by the system are:

AppUsers

Single table that holds all user records. Students and Professors both inherit from User and are stored here using EF’s Table Per Hierarchy mapping. Shared fields: Enumber, email, name, phone, PasswordHash, and IsAdmin.

Students

Specialization of AppUser for student records. Adds Year, Major, IsLocationTrackingOn, LastLatitude, LastLongitude, and navigation property to Assignments.

Professors

Specialization of AppUser for professor records. Adds IsAdmin and navigation property to the Assignments they own.

Locations

Static or semi-static locations for clinical assignments. Each row holds LocationName, Latitude, Longitude, and RadiusMeters for future geofencing logic.

Assignments

Connects one Student and one Professor at a specific Location, Date, StartTime, and EndTime. This is the main “clinical schedule” table and is referenced by Incidents.

Incidents

Records something noteworthy that happened during an Assignment. Stores AssignmentID, Timestamp, and Reason (using the IncidentReason enum).

5. Seed data

On first run, the seeder inserts:

One Professor in AppUsers

One Student in AppUsers/Students

One Location in Locations

One Assignment tying the test student and professor to the seeded location

This gives you working data to test all GET endpoints immediately. You can create additional records with POST requests without touching the seeder.

6. Example API calls

Replace the port with whatever your app prints on startup.

Get all students:

```
curl https://localhost:5287/api/student
```

Get a student by ENumber:

```
curl http://localhost:5287/api/student/9001001
```

Update a student's location:

```
curl -X PUT "https://localhost:5287/api/student/9001001/location" \
 -H "Content-Type: application/json" \
 -d '{"latitude":36.3035,"longitude":-82.3692}'
```

Get assignments for a student:

```
curl https://localhost:5287/api/assignment/student/9001001
```

Create a new assignment:

```
curl -X POST "https://localhost:5287/api/assignment" \
 -H "Content-Type: application/json" \
 -d '{
  "studentEnumber": 9001001,
  "professorEnumber": 9000001,
  "locationID": 1,
  "date": "2025-11-30",
  "startTime": 800,
  "endTime": 1100
 }'
```

Look up a student by email:

```
curl "https://localhost:5287/api/student/by-email?email=student1@etsu.edu"
```

7. Resetting the database

If the schema or seed data gets out of sync:

1. Stop the app.
2. Delete Data/app.db.
3. Run:

```
dotnet run
```

The database will be recreated, migrations will be applied, and the seed data will be inserted again.

8. How to extend the database

If you add new fields or entities:

1. Update the entity class in Models/Entities.
2. Run:

```
dotnet ef migrations add <MeaningfulName>
dotnet ef database update
```

3. Add or update DTOs under Models/DTOs.
4. Update the matching controller to expose the new data through the API.
5. If needed, update DbSeeder so test data includes the new fields.

Keeping changes in that order helps avoid breaking migrations or existing endpoints.

Geolocation

Location Accesses

To access a student’s location, the generic Geolocation API implemented on all browsers is used. This API returns the student’s position as a longitude and latitude pair. This data is then sent up to the server using an API endpoint. The Geolocation API is run on a set interval to ensure that students are in their work locations. In its current implementation, it is set to update every 60 seconds, but as more students are added, it may be better to increase the amount of time between refreshes.

Maps API

As the student’s location is updated, the displayed Google Maps embed is also updated with the student’s updated location. The Google Maps embed is used to show a student’s location relative to their work location. If the student is out of bounds, a path is drawn to get them back in their work area.

Tracking Students

Student locations are stored along with their user information within the project database. Their coordinates are compared against a table of work assignments that define the work location, along with an acceptable range (as a radius) that the student must be in to be considered at their work location. If a student stands outside of their work location for too long, an alert would be sent to the professor (not implemented in the current release).