Technical Documentation for the Implementation of the Green IT Website

1. Project Overview

The Green IT website is designed to help users measure and reduce their digital carbon footprint.
It includes features such as user registration, login, a digital footprint calculator, and an admin dashboard for managing users.
The project is implemented with a **Node.js backend**, **SQLite database**, and a **Vanilla JavaScript frontend**.

2. Backend Implementation

2.1 Framework and Tools

- **Node.js**: Backend runtime environment.
- **Express.js**: Web framework for handling HTTP requests and routing.
- **SQLite**: Lightweight database for storing user and score data.
- **jsonwebtoken**: For generating and verifying JSON Web Tokens (JWT) for authentication.
- **crypto**: For hashing passwords securely.
- **cors**: To enable cross-origin requests.

2.2 Backend Structure

The backend is located in the `api` folder and includes the following files:

- `index.js` : Main server file that defines routes and handles API logic.
- `auth.js` : Handles database operations such as user creation, updates, and score management.
- `users.db` : SQLite database file for storing user and score data.
- `vercel.json` : Configuration file for deploying the backend on Vercel.

2.3 Key Features

User Registration

- **Endpoint**: `POST /api/user/create`
- Hashes the password using SHA-256 and stores it in the database.
- Generates a JWT upon successful registration.

User Login

- **Endpoint**: `POST /api/login`
- Verifies the username and hashed password.
- Returns a JWT for authenticated sessions.

Score Management

- **Endpoint**: `POST /api/score`
  - Adds a new digital footprint score for the logged-in user.
- **Endpoint**: `GET /api/:id/score`
  - Retrieves all scores for a specific user.

Admin Features

- **Endpoint**: `GET /api/users`
  - Retrieves a list of all users.
- **Endpoint**: `POST /api/user/update`
  - Updates user information (e.g., username or password).
- **Endpoint**: `POST /api/user/delete`
  - Deletes a user from the database.

2.4 Authentication

- JWTs are used for secure authentication.
- Tokens are generated using `jsonwebtoken` and include user details like `user_id`, `username`, and `role`.
- Tokens are verified for protected routes.

2.5 Deployment

- Backend deployed on **Vercel**.
- `vercel.json` ensures that all API routes are handled by `index.js`.

3. Frontend Implementation

3.1 Framework and Tools

- **HTML/CSS**: For structuring and styling the web pages.
- **Vanilla JavaScript**: For implementing client-side logic and API interactions.
- **Axios**: For making HTTP requests to the backend.

3.2 Frontend Structure

The frontend is located in the `public` folder and includes:

HTML Files

- `index.html`: Home page.
- `main_login.html`: Login page.
- `register.html`: Registration page.
- `questions.html`: Digital footprint calculator.
- `results.html`: Displays user scores.
- `admin_dashboard.html`: Admin panel for managing users.

CSS Files

- `style.css`: Styles for all pages.

JavaScript Files

- `api.js`: Contains functions for interacting with the backend API.

3.3 Key Features

User Registration

- **Page**: `register.html`
- Sends a POST request to `/api/user/create`.
- Redirects to login page upon successful registration.

User Login

- **Page**: `main_login.html`

- Sends a POST request to `/api/login`.
- Stores the JWT in `localStorage` for session management.

Digital Footprint Calculator

- **Page**: `questions.html`
- Collects user input (hours spent on devices, streaming, etc.).
- Calculates carbon footprint based on predefined coefficients.
- Sends the calculated score to `/api/score` and redirects to the results page.

Results Display

- **Page**: `results.html`
- Fetches and displays user past scores from `/api/:id/score`.
- Shows the most recent score stored in `localStorage`.

Admin Dashboard

- **Page**: `admin_dashboard.html`
- Allows the admin to:
  - View all users.
  - Create new users.
  - Update user passwords.
  - Delete users.

3.4 Deployment

- Frontend deployed on **Vercel**.
- Static files (HTML, CSS, JS) are served from the `public` folder.

4. Database Implementation

4.1 Database Schema

The database uses **SQLite** and contains two tables:

`users` Table

- `id` (INTEGER): Primary key.
- `username` (TEXT): Unique username.
- `password` (TEXT): Hashed password.
- `role` (TEXT): User role (user or admin).

`scores` Table

- `id` (INTEGER): Primary key.
- `user_id` (INTEGER): Foreign key referencing `users` table.
- `score` (DECIMAL): Carbon footprint score.
- `date_taken` (DATETIME): Timestamp of when the score was recorded.

4.2 Database Operations

User Management

- Add, update, and delete users.
- Retrieve user details for authentication.

Score Management

- Add new scores.
- Retrieve scores for a specific user.

5. Authentication and Security

5.1 Password Hashing

- Passwords are hashed using **SHA-256** before being stored.

5.2 JWT Authentication

- JWTs are used to authenticate API requests.
- Tokens include user details and are signed with a secret key (`ILoveGreenIT<3`).

5.3 Role-Based Access Control

- Admin features (like user management) are restricted to users with the `admin` role.

6. Deployment

6.1 Backend Deployment

- Deployed on **Vercel**.
- API requests routed to `index.js` using `vercel.json`.

6.2 Frontend Deployment

- Deployed on **Vercel**.
- All static files served from the `public` folder.

7. Technical Challenges and Solutions

7.1 Mixed Content Errors

- Fixed by ensuring all API requests use HTTPS in production.

7.2 SQLite in Serverless Environment

- SQLite is simple but not ideal for serverless (Vercel).
- Future improvement: migrate to a cloud database (e.g., PostgreSQL).

7.3 Error Handling

- Robust error handling implemented in both frontend and backend to give meaningful user feedback.

8. Future Improvements

Database Migration

- Move from SQLite to a scalable cloud database (e.g., PostgreSQL).

Enhanced Security

- Use environment variables to protect sensitive data (like JWT secret).

Performance Optimization

- Minify CSS and files.
- Optimize images for faster loading.

Eco-Testing

- Use tools like **Website Carbon Calculator** to measure and reduce the website's carbon footprint.
- Use tools like **EcoPing** to measure the consumption of energy of the website