**Solution Document**

**Assumption/Constraint**

1. Username or email can be used to login.
2. A GPSs can store several exact locations, a location can recorded by several GPSs.
3. GPS information that stored in the database are device ID (code) and device type.
4. The username and password must at least have 6 characters
5. Token will be stored in the database, if user generates new token the existing token will be changed in the database.

**Github repository :** https://github.com/cchristian77/gps-app

**Credential**

username: administrator  
email: [administrator@example.com](mailto:administrator@example.com)  
password: administrator  
role: ADMINISTRATOR

username: justin\_beiber  
email: [justin\_bieber@example.com](mailto:justin_bieber@example.com)  
password: justin\_bieber  
role: USER

**Technology**

**Backend:**

* Framework : Express.js
* Database : PostgresSQL
* Library :

1. Prisma : ORM library to help querying and migrating tables.
2. Dotenv : Library to load environment variable from .env file.
3. Bcrypt : Library to encrypt and hash password and data.
4. Joi : Data validation library
5. node-jsonwebtoken: Library to encrypt and decrypt JWT Token for authentication.
6. Uuid: Library to generate and handle uuid.
7. Winston : Logger library to record logs.
8. Nodemon: library to automatically restart the application when application code changes

* Structure : Layered Architecture, the application consists of :

1. Controller : layer that handle and validate request, also to return the requested data
2. Service : layer that consists of main logic of the application.
3. Repository : layer that directly interact to the database.
4. DTO : to wrap the requested data

**Frontend**

* Framework : Angular
* Library :

1. PrimeNG: UI Component library for Angular.

* Structure :

1. Component : contains page layout and logic for each interface component
2. Interface : model interface for retrieved data from backend application.
3. Service : utility function that commonly used in the frontend, e.g. fetch API, handle cookie, etc.
4. Guards : middleware to control and manage routing navigation in the frontend application.

**Enhancement**

1. Implement unit testing to ensure the existing functionality if code changes.
2. Implement code linting to a analyse code and stylistic error.
3. Use latitude and longitude data to show location on the map.
4. Application containerization both for backend and frontend using tools, like Docker, Kubernetes, etc.
5. Implement more advanced authentication protocol e.g., OAuth 2.0.
6. Implement memory caching for complex calculation e.g., dashboard.
7. Implement logging system to monitor and record error and issues.
8. Implement role based middleware or authorization.

**Scalability**

1. Vertical and horizontal scaling.

Vertical and horizontal scaling is the most common approach to scale application. Vertical scaling means upgrading the server hardware, such as RAM, CPU, and storage capacity. Vertical scaling is the easiest method, but it will involve higher costs and limited resource capacity on a single machine. Whereas, horizontal scaling is conducted by adding more machines or servers to handle more requests. You can add machine as many as you can with horizontal scaling so that the load of the requests can be distributed across the machines, but it will cause an increase in the complexity and maintainability of the application.

1. Master-slave architecture

The master-slave architecture separates create/update query to the master database and read (data retrieval) query to the slave database. The changes in the master database will be synchronized and duplicated across the slaves. The pros of this architecture are the workload could be distributed across the slaves and provide backup to enhance the reliability of the application. If one of the databases is shut down, operations of the applications will not be affected because there are still other databases that can be accessed. The cons of this architecture is a synchronization issue, if there are a lot of changes in the master database, it takes time to synchronize or replicate the changes to all duplicated slaves and it may cause inconsistent data between the slaves and the master.

1. Caching.

Caching is the technique to store frequently accessed data in temporary memory. Therefore, database queries could be reduced and improve the overall performance of the system. The cons of the caching technique is memory’s capacity is limited compared to storage disk, so the caching needs to be implemented efficiently in specific scenarios. Moreover, the cache will cause stale data meaning that cache data or content is probably outdated or no longer relevant to the new situation.