

# **Software Design Description (SDD)**

## **COP4331 Processes for Oriented Software Development**

# **Software Design Description (SDD)**

**Employee Tracker**

**COP4331 Fall 2017**

Employee Tracker: Team 6

Team Members:

- Jordan Martin
- Navon Francis
- Brandon Bradley
- David Simoneau
- Marc Simon
- Ryan Hoeck

## **1. Introduction**

### **1.1. Project Overview**

The clients for our software solution are companies who need to track employees and potentially accompanying rolling stock. Employers want to track employees for many reasons which include monitoring deliveries, dispatching crews to off-site locations, tracking expensive rolling stock and ensuring employees are on task and punctual. Employers may range in size from small businesses to large corporations with a large variance in the number and type of employee that needs to be tracked. As we seek to design and implement a system that can serve the needs of a diverse client base, we want the scope to be limited which ensures the solution is practical and cost effective.

Current software products on the market tend to be expensive and offer niche solutions such as tracking freight and ROI where our solution aims to be affordable and easy to use. Clients are generally concerned with the complexity, security and cost of the system. The major concerns of employees who use these applications are battery life and privacy controls.

By tracking employees, clients are storing and accessing sensitive data. All of the major data breaches recently emphasize the need for encrypted storage and sound database design with sensible security precautions. These are the problems that our thoughtfully designed product can solve. By designing software that addresses the aforementioned problems we give our clients and their employees a better solution than the current products on the market.

There are three main components to our software product. A web application, a mobile application and a database. The applications will be accessible from anywhere with internet connectivity. The web application will be able to run on any modern browser that supports compiled Javascript. The mobile application will be able to run on both major operating systems; Android and iOS, and the large subset of those mobile devices with GPS capability. As a result, clients and employees will have nearly limitless options of where and when to use the software. In general, an administrator of the client will login and have permission to add, delete and monitor employee locations. An employee will usually login to the application and have their photo and location reported to the employer. The database will store the employees, their location and timestamp, providing the requested data when needed.

## 1.2. Project Scope

The most important features of the system are the mobile tracking application employees, the web application for employer management of employees and viewing of their locations, and Firebase for the backend system for authentication and data management. The mobile application will require employees to submit a username, password, and photo at login time. These get sent to Firebase for verification and logging. The mobile application then sends an update to firebase every 10 minutes with the employee's gps coordinates. On the web application side, employers will login to the system with a username and password, which again get sent to Firebase for verification and logging. The web application will present the employer with an interactive map where they can view all the employees and their locations, as well as a search box where the employer can input a employee id to look up their location history in the system.

The constraints of our system are limited, but present. For one, some areas, especially more rural ones, may not have a proper signal for a phone to communicate data to our servers, thus, there may be periods of dead-spots, or places where our app will not be able to communicate. For that period of time, we are not able to give updated data to the customer.

To name another constraint, we are developing for “most” devices. This being said, we are going to develop this system based on a high majority of carriers today, but will not be developed for non-smartphone devices or out-dated devices. This is due to the limited amount of developers as well as the cost and time it takes to develop for older technology.

The time constraint is another big thing due to the small number of developers on this team and the necessity for more keeping software and security up-to-date for the users as well as the customer.

Otherwise, we are not constrained too much to keep a good product and to keep the product cheap for the customer.

## 2. Architectural Design

### 2.1. High-level Architecture

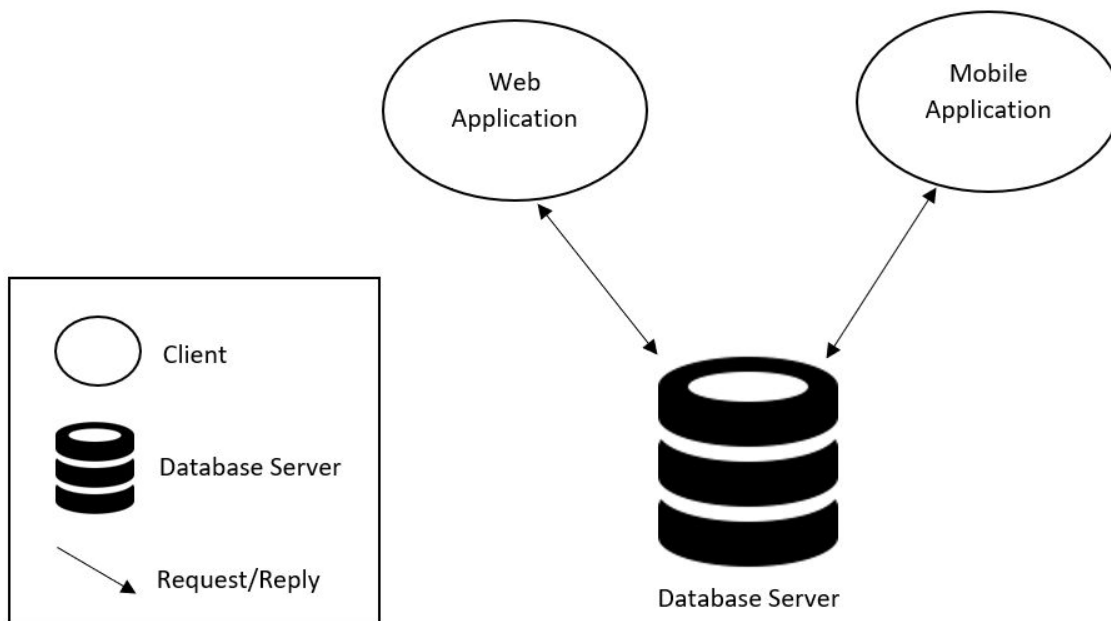
The three main components of our system are the Mobile Application, Web Application and the Database.

**Mobile Application:** Used mainly by Employees to clock in and out by having their timestamp recorded, picture taken, and location updated regularly. The location data will be published every 10 minutes back to the database for storage and management purposes.

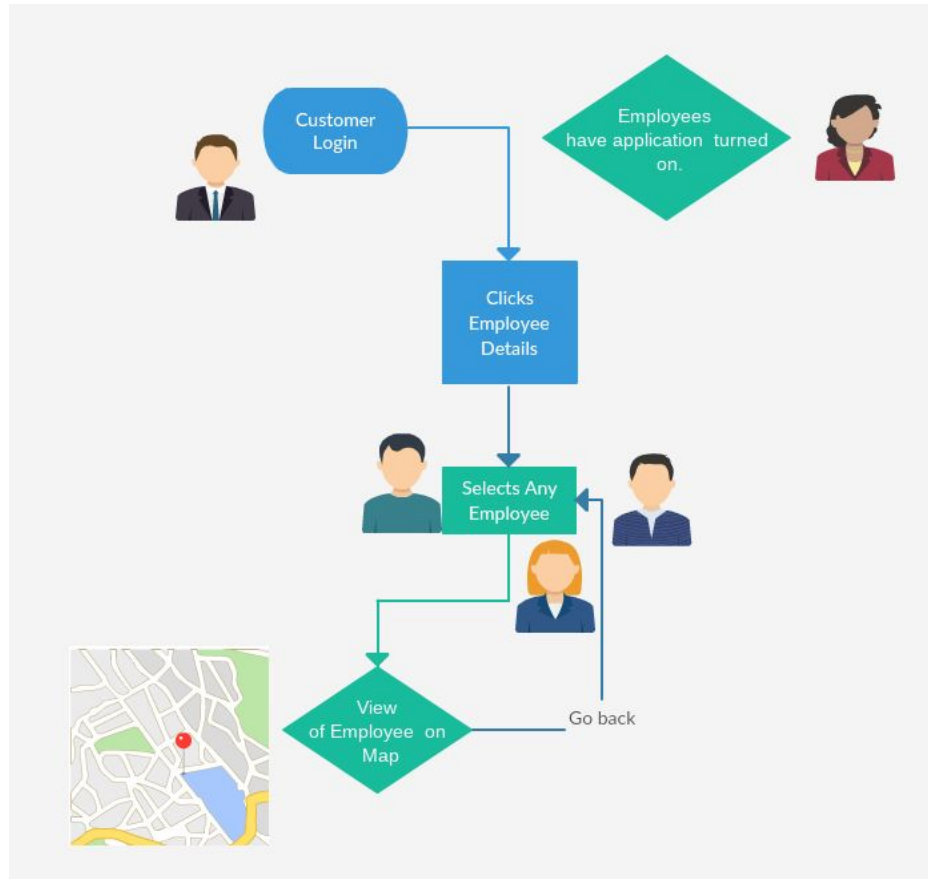
**Web Application:** Used by the Employers to add, delete, or monitor employees and to gather location and time information for management purposes.

**Database:** Built using the cloud-hosted Firebase Realtime Database. It allows for cross-platform applications which in our case include Android, iOS, and modern web browsers, to share a single database with real-time updates of common data.

We have a database that subscribes to the mobile application to receive and store updates on location using existing coordinates, store time and employee account information data, and is used across all three different platforms. Firebase also provides an application server that hosts and provides services for both the web and mobile applications. Given all of these necessary functions, the best description of our architecture is a combination of the Publish-Subscribe, Repository and Client-Server Architectures into a single custom design that fits our purpose.



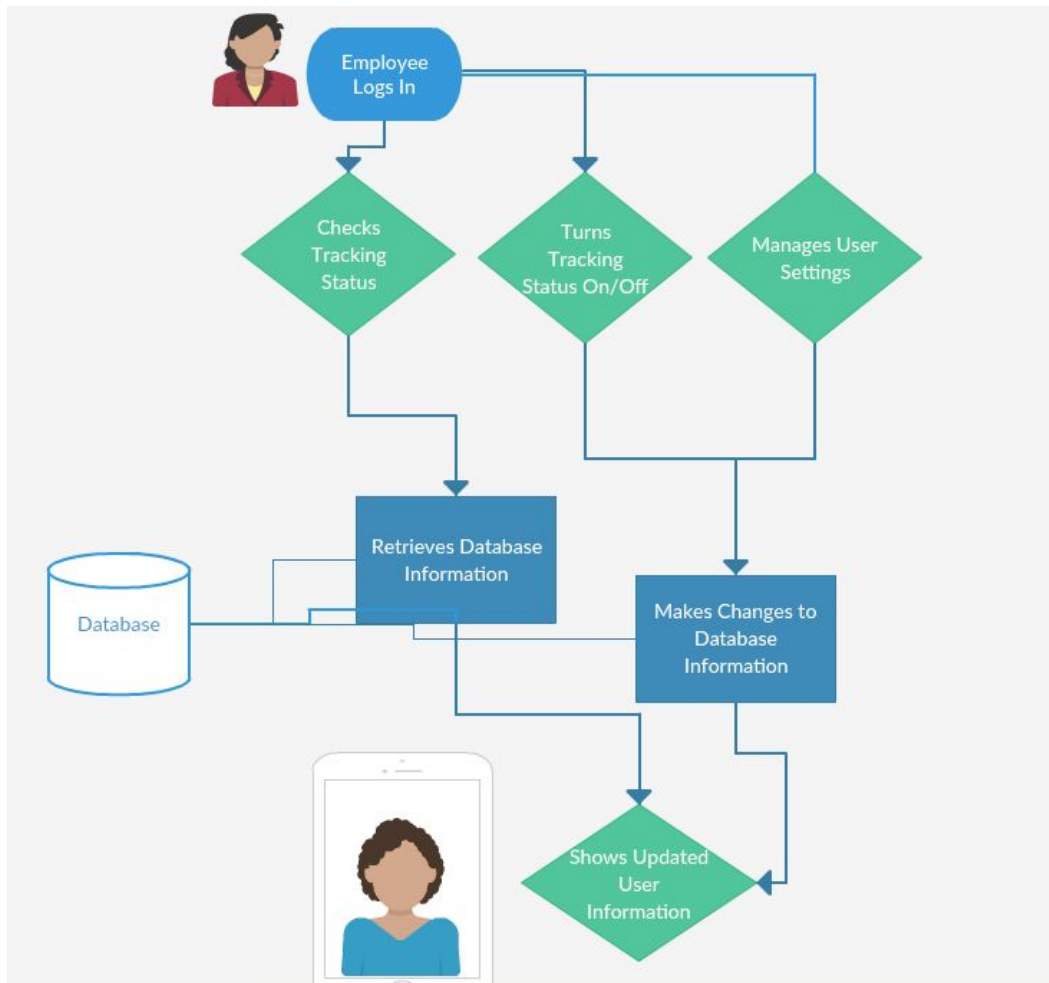
## 2.2. System-interface Architecture



*View of Customer's Flowchart*

As seen from the diagram above. This is the view from the customer's perspective. The customer will log into the application and then view employee's location details as long as the employee has their application turned on. From there, they can pull up a map of the location and go back to locate another employee.

This system itself is then connected to the database, which as seen from the architectural design image above. The database will be the source of information for the employers as well as the employees to pull information from. This will not be apparent in the front-end design, but will be happening in the back-end of things, not viewable by the user.



*View of Employees Flowchart*

Here, we can see that the employee also has some control. The employee will be able to turn on and off their tracking to keep control in their hands and from this, can make changes to their information in the database. Then, the database will present the user with up-to-date details of their information.

These easy to follow diagrams give insight to how our whole system works.

### 2.3. Data Design

The database (hosted by firebase) will be structured as a set of JSON object trees in a noSQL database environment. Each employee will be represented by a single object, with fields username, employeeID, gpsLocations, and empPicture. When an employee attempts login, the mobile application will send the device location, employee username and password, and employee picture over a secure HTTPS channel through a REST post request. The server will then attempt to verify the employee, and either allow or deny login over the same HTTPS channel. Once verified, the mobile app will store locally, then send the user's location, including a timestamp every 10 minutes until they log out of the system, again through a post request. The mobile application also has the ability to turn off the tracking (enabling privacy mode), with the ability to include an optional message, encoded as a string, that gets sent to the database.

The web application employers will utilize attempts login in the same way, except that the employer does not need to include a picture of themselves in order to log into the system. Once logged in, the web application will request a map of all active employees, ie employees currently logged into the system, and the map will display each of the employees last reported location. The request that the web application sends is again in the form of a REST get request through a HTTPS secure channel. The employer will also be able to input an employee ID into a search box, which will send a get request to the database for the entire location history of that employee. The data will be presented as a list in the mobile application. The web application will also show any messages that were sent by the mobile application, next to the user that sent the message.

Reference the Appendix, 5.1: Database Entity-Relationship Diagram.

Data Name	Data Type	Maximum Length
Employee	JSON Object*	N/A
Employer	JSON Object*	N/A
Username (employee/employer)	String	256
Password (employee/employer)	String	256
gpsLocation	double tuple, in (Latitude, Longitude) format	6 Significant figures.
message	string	300
empPicture	.jpeg format.	Size: 5 MB.
timestamp	ISO date format, Int	N/A

JSON Type Specifics:

	Data Name	Data Type
Employee JSON Object	name id empPicture privacyMode company locations timestamp	String String .jpeg format boolean String double, formatted as tuple ISO date format
Employer JSON Object	name id company address	String String String String

## 2.4. Alternatives Considered

The architecture we most closely relate to is that of the following:

**Client-Server:** We are using both components in a Client-Server architecture model. Our frontend React/React-Native Client will make requests to our Firebase Database. This being our Client portion of the architecture. The Firebase database and server acts identical to the Server part of the Client-Server architecture mode.

**Repository:** We considered having the mobile application and the web application interact with the database through a repository type architecture, however the end goal of the mobile application was to have updates sent to the server, and the server would route the data. We wanted the mobile application to be simple in function, so we ultimately ruled out this architecture.

**Publish-Subscribe:** We thought publish subscribe would fit our idea of the Employee Tracker. The Firebase Server and database would subscribe to the mobile applications, which in turn would publish the GPS updates every 10 minutes. This model has flaws for our project. We wanted the server to not have unnecessary connections open to all possible mobile applications, active or inactive, as this would waste resources.

Ultimately, we decided on a combination of Client-Server and Publish-Subscribe, where the clients (the mobile applications) would publish their updates to the server (the Firebase Database), and the web application would act as a client in accordance with the Client-Server model, receiving updates from the Server whenever an employee's location gets updated.

We also considered what we needed our database and backend system to be. Originally, we considered making our own RESTful API's in Go or Flask (a Python microframework), however with discussion with the leading actors (the TA for the project) we decided such a system would slow down development given the timeframe of 10 or so weeks. We settled on Google's Firebase service as it can provide a secure login system, a database for the system, a easy way to manage the data and development as a team, and it would speed up our development time.

## 3. Detailed Design

### 3.1. Design Issues

When using the architectural style of Client-Server, we are using the built in networking methods of React to make out POST and GET requests to the backend database, Firebase. These requests are wrapped in methods such as `fetch()` that grabs API data from servers.

A Repository characteristic trait comes into play when locations of employees are then updated (analogy of students going up to a whiteboard and solving math

problems, erase, and repeat.) We chose this because having a centralized storage system (Firebase) to store coordinates that are always being updated was better practice than storing them in the frontend client.

A Publish-Subscribe characteristic train comes into play when GPS coordinates are sent to the backend. Since coordinates are being sent to the client every 10 minutes, we wanted to choose the best data handling system possible. This rapid access of coordinates is handled by our real time database. This allows for data to be updated across **all** clients easily, in turn making a super responsive experience. This also is beneficial to offline connections. When clients go offline, Firebase built in functions synchronizes local data with any remote changes that may have been made. This eliminates the risk of having old or funky data in the client that each employee be using.

By making the design decisions stated above, we reach our goal of having a super fast, efficient, and correct system with good integrity.

- specifics as to why we choose what we did, given timeframe, reqs, goal, etc.
- prototypes can be 'pure' client-server, repo, pub-sub, etc. explaining why each had pitfalls and why we choose to combine aspects.
- if you can ask a question about a reason, provide an answer to it.

### 3.2. Component *n* detailed design (include a sub-section for each component)

Please reference Appendix 5.4.1 Class Diagram for the overview of the system. Appendix 5.6 for images of the user interface at login.

#### Component: Firebase Database

- The core and the backend of the system. This holds all users, locations, and connections with the mobile and web applications.

##### Attributes:

- users: This is a reference to the list of active users in the database, both employees and employers. They are stored in a noSQL database as JSON objects.
- connection: This is a list of all active connections to the database, each one is its own Connection Object.

##### Functions:

- login(): This function verifies the user with a password username, and possibly a picture with the list of users in the database. If it returns true, it creates and opens a new Connection object with the application that sent the request.
- createUser(User): This function creates a new user, either an employer or employee in the database of possible users.
- removeUser(User): Removes a user, either an employer or employee, from the database.
- addLocation(Employee): When a location is sent from an employee, this function adds it into the database of last locations for the employee.
- getAllEmployees(): This function, when invoked, gets the list of all currently active employees and sends the list of them to the web application that requested it.



- getEmployee(Employee): This function gets a specific employee by id and returns all past locations of that employee to the web application.

#### Component: Login <<interface>>

- An interface that is utilized by both the mobile and web applications to sign into the system

##### Attributes:

- username: This is the username for the login attempt.
- password: This is the password for the login attempt.
- authorization: String that is returned to the web or mobile application if the login attempt is successful.
- timestamp: The timestamp of the login request. It is an int in ISO Date format.

##### Functions:

- login(username,password): attempts to log the user in with the supplies username and password.

#### Component: Location

- The object that represents a location in the system

##### Attributes:

- location: This is a location in a tuple of doubles, the latitude and longitude, both with six digits of precision.
- timestamp: The timestamp of the GPS location, this an int, formatted in ISO date format.

##### Functions:

- update(): sends the location and timestamp supplied to the database for storing.

#### Component: View

- This component updates the view of the mobile application and the web application.

##### Attributes:

- title: This is the current page title the application is on.
- map: A map with the locations of the employee marked on it.

##### Functions:

- update(): This updates the UI of the view to the current map if requested by the web application, otherwise it either displays the login page, or “Successfully Logged In” if the login was authorized.

#### Component: User

- The User class is a superclass of both the employee and employer classes.

##### Attributes:

- name: This is a string that is the name of the employee or employer that the class signifies.
- userID: the unique userID assigned to each user.
- company: A String, with the name of the company they work for.
- username: A string, the unique username that is used to sign into the system.

#### Component: Employee

- The employee class is a subclass of the User class.

##### Attributes:

- picture: a JPEG picture of the employee, stored in the database.

- `privacyMode`: a boolean that signifies if the user has enabled privacy mode. If they have, their location is not sent to the database.
- `location`: the Location that was last taken by the mobile application. This is stored locally on the device for sending to the database.
- `message`: A String that the employee puts when they enable `privacyMode`.

#### Component: Employer

The employer class is a subclass of the User class, representing an employer.

Attributes:

- `address`: The address of the employer, stored as a string.

#### Component: Connection

- This is the connection that is open between applications and the server for a maximum of 30 minutes before it is invalidated.

Attributes:

- `username`: The username that was used to verify the user
- `authorization`: The authorization token that was shared between the database and either the mobile or web application.
- `verify`: the status of the login attempt. Either true, the connection and login was verified, or false, the connection and login was not verified.
- `active`: the current state of the connection, if it is still active.
- `time`: the time since the last successful connection with the web or mobile application and the database. If this exceeds 30 minutes, the connection is invalidated.

#### Component: Mobile Application

- This is one of the core aspects of the Employee Tracker. This is a application that runs on each phone in use by the employees. The mobile application will prompt the employee with a username, password, and picture submission. These will be sent to the database for verification. When verified, a connection will be opened and a authorization token will be generated and sent to the application, allowing seamless periodical 10 minute interval GPS updates to the database.

Attributes:

- `user`: The employee that is using the application
- `location`: the location that is taken from the GPS of the phone.
- `loggedIn`: a boolean to represent if the application is logged in and authorized with the server.
- `auth`: This is the authorization string that is generated by the database on successful login and validation. It is stored locally for ease of reconnection when GPS updates get sent.
- `timer`: An integer that keeps track of the time since the last GPS location was sent.
- `message`: a message that gets sent to the database. The employee is prompted to input the message when they activate `privacyMode`.

Functions:

- `sendLocation`: This function reads the timer attribute, and when 10 minutes has passed, reads the location from the GPS and sends it to the database.
- `togglePrivacy()`: Is activated by a button press on the mobile application. It sets `privacyMode` of the employee to true, and the

employee is prompted to enter a message explaining why they enabled the mode. This message is sent to the database.

- sendMessage(): Sends the message that togglePrivacy() prompted the user for to the database.

#### Component: Web Application

- This is the interface used by the employees of the company that is using the system, another one of the core aspects of the Employee Tracker. The web application will prompt the employer with a username, and password submission. These will be sent to the database for verification. When verified, a connection will be opened and a authorization token will be generated and sent to the application, allowing seamless periodical updates from the database.

-

#### Attributes:

- user: The employer that is using the web application.
- loggedIn: a boolean to represent if the application is logged in and authorized with the server.
- auth: This is the authorization string that is generated by the database on successful login and validation.
- message: If any employees send a message as to why privacy mode is enabled, this gets sent and displayed to the employer in the web application.
- results: SearchResults: This represents the user that was searched for by the employer, as well as the list of locations that last employee was at.

#### Functions:

- getAllUsers(): This sends a get request to the getAllEmployees() of the database. It is returned and stored as a list of all active users of the system.
- getSpecificUser(): This is invoked when an employer using the web application searches for a specific user by userID. It returns SearchResults, with a User and the list of all locations reported by the User's mobile application.
- updateUser(): This is called when the employer wants to add or remove a user from the database. It will use the createUser() or removeUser() functions in the database respectively.
- updateMap(): periodically updates the map with the new locations of all employees using the View component.

#### Component: SearchResults

- This is the object returned when the web application requests a specific user from a search.

#### Attributes:

- user: The user object that was searched for by userID and returned by the database.
- locations: a list of all locations associated with the userID, returned by the database.

### 3.3. Requirements Traceability Matrix

Traceability Matrix located here:

	Status: (D) Development, (T) Testing, (L) Design, (R) Ready				
	Test Case: (U) Untested, (P) Test Passed, (F) Test Failed				
Req ID	Requirement Description	Architecture Reference	Design Reference (component/module)	Test Case Reference	Status
3.1.1	The Employee Tracker shall allow the Employee to log in, update location, and see other users using the mobile application	Employee Login, Tracking status	Login <<interface>>, Mobile Application	U	L
3.1.2	The Employee Tracker shall allow the Manager to see updates from the employee on the web application.	Employer Login, Select Any Employee	View, Web Application	U	L
3.1.3	The login username and password will be verified to be valid according to the requirements set by the company.	Employee Login, Customer Login	Connection, Firebase Database	U	L
3.1.4	The mobile application will continue running and collecting location coordinates for up to 30 minutes if loss of internet connectivity occurs. After 30 minutes, the system alerts both the user and the employer that the device has not had a connection for 30+ minutes.	Checks Tracking status, Employees Have Location turned on.	Connection, Mobile Application	U	L
3.1.5	The mobile application will notify the user and the employer if the mobile device's GPS or camera malfunctions.	Clicks Employee Details, Turns Tracking on/off, Makes changes to the database information.	Web application, Mobile application	U	L
3.1.6	The mobile application will send the locally stored locations to the server upon successful reconnection.	Makes changes to the Database Information.	Mobile Application	U	L
3.2.1	The Employee Tracker application shall check username and password for each user log in. The username and password inputs will be sent as strings to be checked and be validated with Firebase. The data must be exact as it is credentials for a user.	Employee Logs in, Customer Logs in	Connection, Firebase Database	U	L
3.2.2	The Employee Tracker application shall fetch, get (fetch method), and post requests to and from Firebase to	Retrieves Database Information, Makes changes to	Connection	U	L

	validate against the frontend that what the user input has integrity.	Database Information,			
3.2.3	Upon successful login, the mobile application shall send its location with 6 points of precision to the server immediately, and every 10 minutes thereafter, for the duration the user is logged in, or until a period of 30 minutes of no connectivity to the server passes.	Check Tracking Status, Makes Changes to database Information	Mobile Application Location	U	L
3.2.4	All data transfers between the Firebase server and the web or mobile application shall be through a RESTful API.	Database, Retrieves Database Information.	Connection	U	L
3.4.1	The Employee Tracker Application shall have these 3 user groups: The Employer, Employee, and Admin.	Employee, Customer, Admin	Web application, Mobile Application, Firebase Database	U	R
3.4.3	The Employee Tracking App shall prevent misuse by the user groups	Turns tracking on off, makes changes to database information.	Connection, Login, Mobile application	U	L
3.4.4	Users shall be provided training and copies of the Employee Tracking Application manual, thus minimal knowledge of the system is required. The only ability required is being able to follow a procedure outlined in the manual, and being able to read at smallest an 11pt font.	Database, Client-Server	Documentation	U	L
3.5.1	The documentation shall be stored with the software and updated in accordance to functional changes of the software.	Documentation	Documentation	U	L
3.5.2	The documentation will have pictures and diagrams of the software, with footnotes explaining what each button does.	Documentation	Documentation	U	L
3.5.3	The documentation should be concise and not span hundreds of pages.	Documentation	Documentation	U	L
3.5.4	The documentation shall have no usage of legalese or other complicated technical terms or jargon that would confuse a user.	Documentation	Documentation	P	R
3.5.5	The documentation should explain all parts of the software the user can	Documentation	Documentation	U	L

	interact with.				
3.6.1	The degree of GPS precision shall be 6 decimal places for latitude and longitude. For example, (28.6024 N, 81.2001 W) which are the GPS coordinates for the University of Central Florida. The accuracy of the GPS data will vary depending location, weather and the number of satellites in line of sight to the GPS hardware on the mobile device. The Google Maps API will be used to facilitate this requirement.	Database, Retrieves Database Information	Documentation	U	L
3.6.2	GPS coordinates in the form of latitude and longitude for each individual employee must be retained in the database.	Database, Retrieves Database Information	Firebase Database, Algorithm	U	L
3.6.3	The employee's name and unique ID will need to be retained together in the database.	Database, Make changes to the database.	Database	U	L
3.6.4	The date and time of the gathered GPS coordinates shall be retained alongside the rest of the gathered data in the database.	Database	Database, User,	U	L
3.6.5	Logins shall be retained in the database.	Client-Server, Repository	Database, Login	P	R
3.7.1	A database shall be required for the backend to provide long term storage for our client's data. As our client's grow in number the costs associated with the Firebase database platform we chose will grow as well.	Database	Firebase Database	U	L
3.7.2	A database administrator(s) will be needed to perform maintenance on the system as required. The system may need to be debugged at times and this action will require skilled personnel.	Database	Firebase Database	U	L
3.7.3	The Employee Tracker application shall require the client to have access to computers and the client's employees to have access to mobile devices. We will not need to provide any hardware to the clients for the application to function as intended. In addition, we will not require physical space or	Subscriber	Web Application, Database	U	L

	supporting amenities as the client's business will cover those aspects of their operation.				
3.8.1	Access to the system shall be controlled and secured by logins for each client.	Server-Client	Firebase Database	U	L
3.8.2	Client's data shall be isolated from other clients through logins.	Server-Client	Firebase Database	P	R
3.8.3	Client's data shall be encrypted with 256-bit AES standard	Database	Firebase Database	U	L
3.8.4	Database shall be tested against SQL injection	Database	Firebase Database	U	L
3.8.5	Automatic backups shall be performed once per day offsite (Firebase cost dependent). Backups are off site on Google's servers and protected from water, fire and other natural disasters. Data can be recovered if needed at any time.	Database	Firebase Database	U	L
3.8.6	Employee's shall not spoof their location.	Server-Client	Mobile Application	U	L
3.9.1	The Employee Tracker Application shall be available as a web application and mobile application.	Client-Server, Repository	Web Application, Mobile Application	U	L
3.9.2	Software shall be maintained by us, the software creators.	Repository		P	R
3.9.3	Automation of bug-detection will be in place	Client-Server, Repository, Database	Firebase Database, UI, Web/Mobile Applications	U	L
3.9.4	Downtime must be minimal (less than 30 minutes a week during working hours), Response time to issues should be within the hour of the issue arising.	Database	Connection, Firebase Database	U	L

#### 4. Conclusion

As modeled in this document our architecture will be a blend of Client-Server and Publish-Subscribe to effectively satisfy all the requirements of the Employee Tracker mobile and web application. Our data design uses a noSQL database, namely Firebase; to store encrypted location data and its derivative details, remotely serve historical location data and provide employer and employee logins. As React and Firebase are the main frameworks the web and mobile application are constructed with, the implementation will follow best practices of both frameworks, outlined in their respective documentations. Our group will hold informal code walkthroughs in

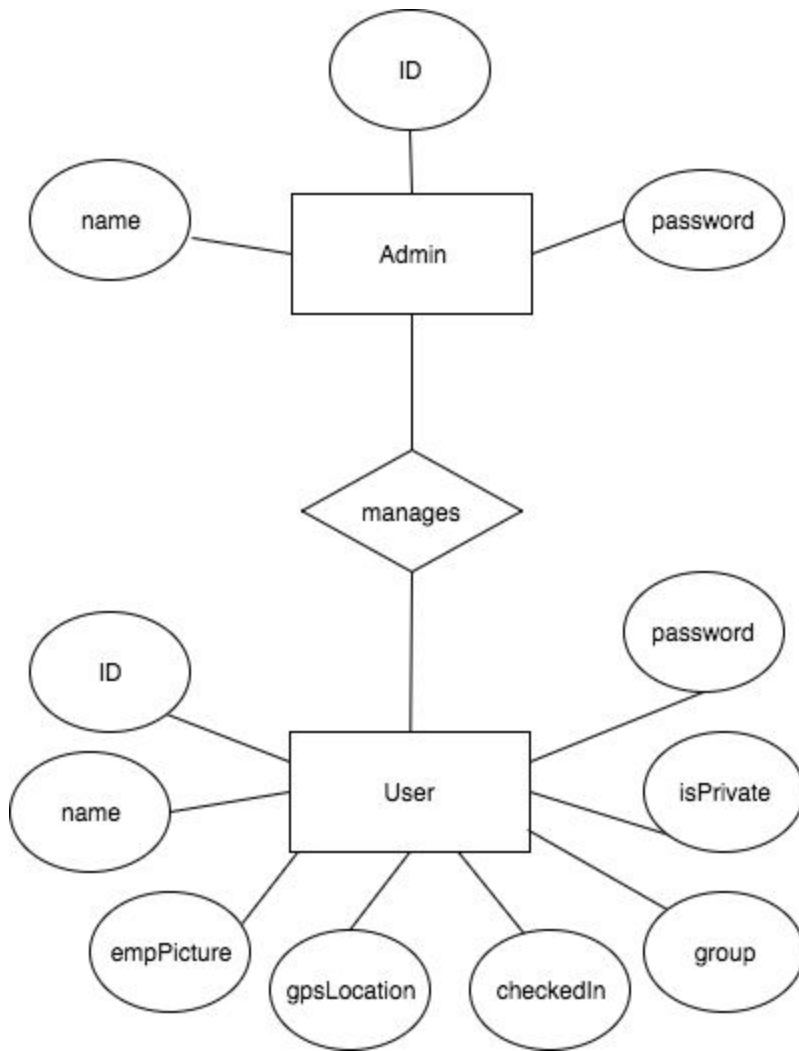
our weekly meetings and internally document our code with meaningful details to enhance the understanding of the codebase for the rest of the team that have not wrote code for a particular component. For the programming process we still intend to adopt an Agile methodology as outlined in previous deliverables. For implementation we will work in pairs, though there are no true distinct senior and junior software engineers in our team. We will rely on those with the most past experience with specific components to lead and review, loosely mimicking the pair programming structure.

As our implementation of components nears completion we will develop test cases to identify faults which can lead to failures. We will track faults and correct them according to severity. As Firebase is well defined, we will use the internal tools to check for performance and coordination faults which are integral to the syncing of data between the web and mobile applications. We will test on our individual mobile devices, which include Android and iOS, as well as on Mac and Windows operating systems with regards to native Safari, Firefox and Chrome functionality. We expect to select specific testing techniques as our general outlined components become clearly defined through code and we subsequently develop a test plan. In general, we foresee following a standard testing sequence beginning with unit testing and concluding with installation testing. As we have a small workforce consisting solely of our group members and limited resources, we can not form an independent test team and will work with those limitations as best we can as a team.

## 5. **Appendices** (a list of possibilities, these items can go into an Appendices section)

### 5.1. Database Entity-Relationship Diagram

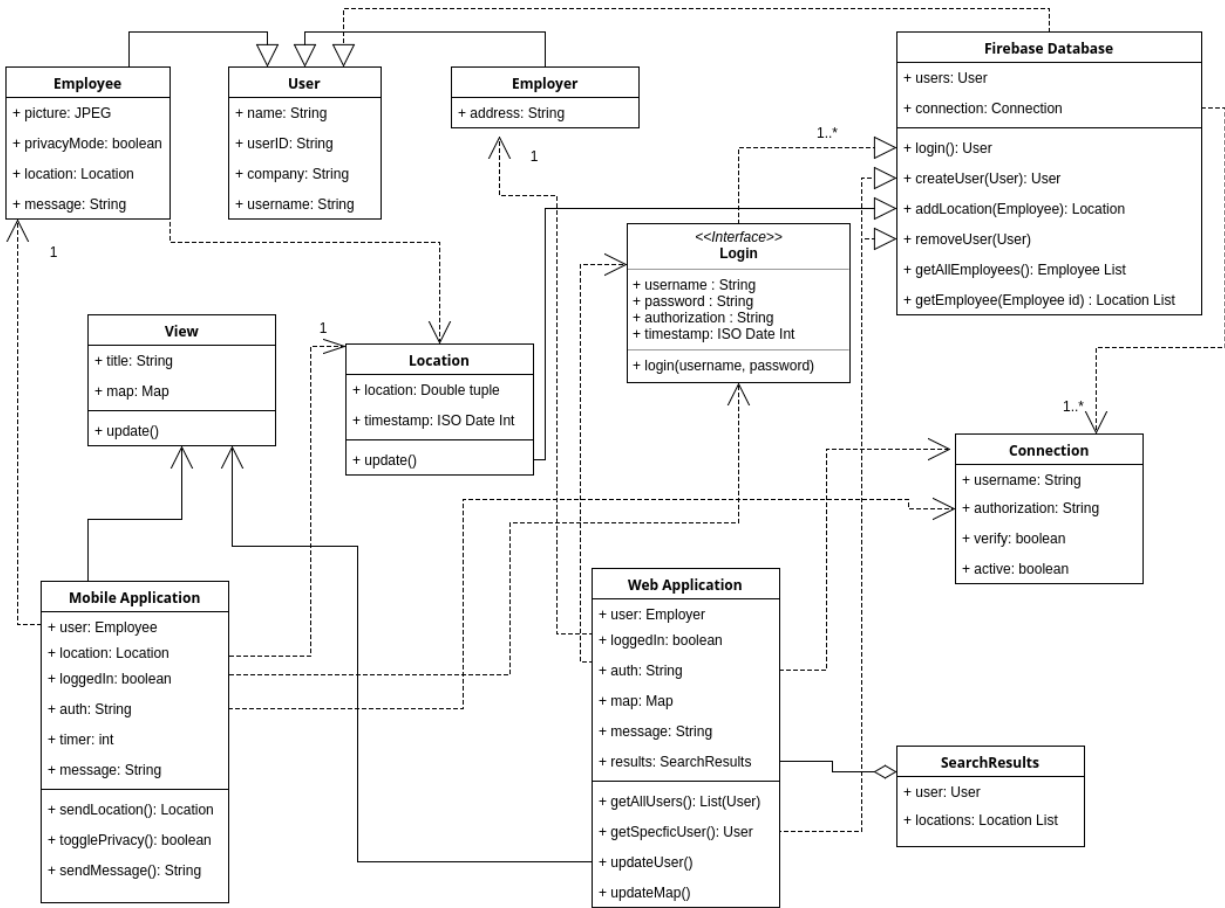




5.2. Architectural Design Block Diagram(s)

5.3. Activity diagrams(s)


5.4. Class Diagram(s)



### 5.5. Class Sequence Diagram(s)

## 5.6. User Interface Screen Snapshots

Web app Login:




Employee Tracker

Username

Password

SUBMIT

Mobile app login:

 Employee Tracker

Username

Password

SUBMIT