

Project Documentation

Project Title - GPS based Tracking Application with Logistics Management.

|  |  |
| --- | --- |
| TEAM MEMBERS | EMPLOYEE\_ID |
| Anuj S M | 5402 |
| Dheeraj B N | 5406 |
| Harshith V | 5408 |
| Moyukh Saha | 5412 |
| Santhosh Kumar A | 5418 |
| Shruti C L | 5421 |

# TABLE OF CONTENTS

1 INTRODUCTION

* 1. [Purpose](#_bookmark5) 3
  2. [Scope](#_bookmark6) 3
     1. [Overview](#_bookmark7) 3
     2. [Benefits](#_bookmark8) 4
  3. [Structure](#_bookmark9) 4

1. [OVERALL DESCRIPTION](#_bookmark11) 5
   1. [Product Perspective](#_bookmark12) 5
      1. [User interfaces](#_bookmark14) 5
      2. [Hardware interfaces](#_bookmark15) 5
      3. [Software interfaces](#_bookmark16) 5
2. CODING STANDARDS AND GUIDELINES5
3. [REQUIREMENTS](#_bookmark25) 6
   1. [Functional Requirements](#_bookmark26) 6
      1. [Customer](#_bookmark27) 6
      2. [Seller](#_bookmark29) 7
      3. [Delivery Person](#_bookmark31) 7
   2. [Non-Functional Requirements](#_bookmark37) 8
      1. [Safety](#_bookmark38) 8
      2. [Security](#_bookmark40) 8
      3. [Human engineering](#_bookmark42) 8
      4. [Performance requirements](#_bookmark44) 9
   3. [Technical Requirements](#_bookmark37) 10
4. [UML ANALYSIS MODELS](#_bookmark46) 11
   1. [Use Cases 1](#_bookmark47)1
      1. [Use case descriptions](#_bookmark51) 12
   2. [Class Diagram 1](#_bookmark85)5
      1. [Class descriptions](#_bookmark86) 16

# INTRODUCTION

The following section provides an overview of the Project Documentation for the subject GPS based Tracking Application with Logistics Management. To begin with, the purpose of the document is presented and its intended SME’s outlined. Subsequently, the scope of the project specified by the document is given with a particular focus on what the resultant software will do and the relevant benefits associated with it. To conclude, a complete document overview is provided to facilitate increased reader comprehension and navigation.

## Purpose

The purpose of this DOCUMENTATION is to outline both the functional and non-functional requirements of the subject GPS based Tracking Application with Logistics Management. In addition to said requirements, the document also provides a detailed profile of the external interfaces, performance considerations imposed on the subsequent implementation. It is the intention that the presented set of requirements possesses the following qualities: correctness, unambiguousness, completeness, consistency, verifiability, modifiability and traceability. Consequently, the document should act as a foundation for efficient and well-managed project completion and further serve as an accurate reference in the future. It will not only provide an extensive capacity for project planning and progress assessment, but it will further assist with senior developers/SME groups interactions.

## Scope

In current formal food purchasing environments, some form of physical static menu is utilized to convey the available food item choices to customers where self-employed women are not able to reach all customers with their sales and marketing activities that are restricted to mouth publicity and experiences. This document specifies the requirements for an innovative online platform to register themselves to alleviate the problems associated with the current archaic method. Three related concepts are encompassed by the general scope of the Online food spices and snacks Ordering and Delivery System. The first pertains to the replacement of mouth publicity towards reaching a large set of consumers online, the second relates to a complementary electronic strategy for the front of house handling of a customer’s order and the third surrounds the process of transferring said electronic orders to the kitchen for preparation.

### Overview

In today's time product delivery is very important for any transport organization. Our system provides real-time asset tracking. this web application is used to track the delivery shipment using gps. The delivery truck has onboard temperature sensor through which the temperature of the organic goods can be monitored by the admin, The admin can use the alert button to send a mail and sms to inform the truck driver in case of emergencies. This web application also has supplier login along with admin and driver. the supplier can log in and check the order and truck details associated with them. In today's time product delivery is very important for any transport organization. In our system we have instant alerts on mobile SMS or Email to the truck driver.

Admin, Supplier, Truck Driver are presented with an attractive and easy-to-use surface computer GUI with the clickable’s metaphor in their menus.

**1.2.2 Benefits**

Greater flexibility in menus, an increase in GPS based Tracking Application with Logistics Management for used to track the delivery shipment using gps. The delivery truck has onboard temperature sensor through which the temperature of the organic goods can be monitored by the admin, The admin can use the alert button to send a mail and sms to inform the truck driver in case of emergencies. This web application also has supplier login along with admin and driver. the supplier can log in and check the order and truck details associated with them. In today's time product delivery is very important for any transport organization. In our system we have instant alerts on mobile SMS or Email to the truck driver.

## Structure

The structure of this Software Requirements Specification is as follows. Section 2 presents an overall description of the subject GPS based Tracking Application with Logistics Management. Attention is given to putting the resultant software product into perspective and further outlining end-user characteristics, system constraints and assumptions. Section 3 is devoted to the explicit specification of software requirements both functional and non- functional in nature. The functional requirements listed have been demarcated according to the categories of system users. For completeness, Section 4 extends upon Section 3 through the inclusion of UML analysis models and diagrams. To begin with, the identified GPS based Tracking Application with Logistics Management use cases are given. In supplement, activity diagrams for each use case are presented along with an overall class diagram and relevant ER diagram.

# OVERALL DESCRIPTION

The following section presents an overall description of the subject GPS based Tracking Application with Logistics Management. In particular, the product has been put into perspective through a detailed assessment of the user, hardware, software and communication interfaces requirements.

# Hardware Requirements

For developing the application, the following are the hardware requirements

GPS tracker, Temperature Not available.

# Software Requirements

1. Spring Tool Suite
2. Java
3. JavaScript
4. HTML
5. CSS
6. JSP
7. BOOTSTRAP
8. H2 Database
9. Maven

10.Tomcat server

1. API of third-party service for SMTP mail.
2. Jenkins for CI

# Typical customers: Our platform was designed primarily for a pre-existing logistic company(B2B).

# Platforms supported

* Web

# Support options

* 24/7 (Live rep)
* Knowledge Base
* Email/Help Desk

# Functional and Non-Functional Requirements

The following section presents the complete set of functional and non- functional requirements identified for the subject Logistics industry. Functional requirements are listed first, according to their relationship to the overall system, customers, Supplier, delivery. The non-functional requirements that pertain to safety, security, the interface, human engineering, qualification, operation, maintenance and performance are subsequently presented.

# Actors

There are Three actors in this Logistics Industry

* + Admin
  + Supplier
  + Delivery person/truck driver

# Functional Requirements

This subsection presents the identified functional requirements for the subject Logistics industry. Initially, general requirements that pertain to the whole system are given. Where possible, subsequent requirements have been demarcated based on their relevance to the users of the system, that is, Admin, deliver partnefr/truck driver , Supplier.

# Admin

[Table](#_bookmark0) presents the identified functional Admin requirements that directly relate to the customers of the subject Logistics industry.

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| 01 | There is only one admin, who has access to monitor and view all Supplier/Delivery partner details and also details of all orders. |
| 02 | Admin can approve/reject a newly registered supplier or a delivery partner. |
| 03 | Admin can place a new order. |
| **04** | **Admin can assign a particular supplier/truck driver to complete any order.** |
| **05** | **Admin will have the live temperature of the parcel, as well as live tracking facility.** |
| **06** | **Admin can alert the truck driver incase there is any fluctuations(ideally the optimum temperature required to keep the parcel in prime condition) in the temperature of the parcel** |

**Functional Admin Requirement**

# Supplier

Table presents the identified functional requirements of a Supplier.

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| 01 | Supplier can choose the supplier profile in the home page, where he will be directed to the supplier login page. A new supplier can register on the register page, whose account will only be active after being approved by the admin. |
| 02 | After logging in the supplier will be able to monitor order details and also details of truck drivers associated with each orders. |

## Delivery partner/Truck driver

[Table](#_bookmark1) presents the identified functional requirements of a truck driver.

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| 01 | Truck driver can choose the supplier profile in the home page, where he will be directed to the supplier login page. A new supplier can register on the register page, whose account will only be active after being approved by the admin. |
| 02 | Truck driver After logging in will be able to monitor order details and also details of the parcel and suppliers such as pick-up and drop locations. |
| 03 | Truck driver will receive alerts from admin incase of any temperature fluctuations. |

**Table 3.1.4 Functional Chef Requirements**

## Product Perspective

The software described in this documentation is the software for a complete GPS based Tracking Application with Logistics Management. The system merges various hardware and software elements and further interfaces with external systems. Thus, while the software covers the majority of the system's functionality, it relies on a number of external interfaces for persistence and unhandled tasks, as well as physically interfacing with humans.

### User interfaces

There is one separate user interface used by the GPS based Tracking Application with Logistics Management software, related to an interfaced physical hardware device.

#### Surface computer UI

The Surface Computer UI is the interface used by all three users i.e. Admin, Supplier, Truck Driver. This interface uses the surface computer paradigm - users interact with the system by dragging 'objects' around on the GUI display. Customers will summon their snacks and spices menu, which is combined with a system/command menu, and dismiss it with a similar gesture or by tapping a close button GUI element.

### Hardware interfaces

There is one external hardware devices used by the GPS based Tracking Application with Logistics Management, each related to a user interface. The device is the surface computers. This device must be physically robust and immune to liquid damage and stains. However, they should be fully capable computers that can use textual data from the server along with local UI/interpretation code to display UI elements and take input. All order and transaction records should be stored on the database server, not these computers. In all cases, the hardware device takes information from the GPS based Tracking Application with Logistics Management and processes the information to display. It also provides user input information to the HSOS.

### Software interfaces

The GPS based Tracking Application with Logistics Management will interface with a Database Management System (DBMS) that stores the information necessary for the GPS based Tracking Application with Logistics Management to operate. The DBMS must be able to provide, on request and with low latency, data concerning the spices and snacks menu, Additionally, it should take and archive data provided to it by the GPS based Tracking Application with Logistics Management. This data will include records of all orders and transactions (system states and state changes) executed by the HSOS. The DBMS must store all data such that it can be used for accounting, as well as accountability.

# CODING STANDARDS

# About the TEAM 2 Coding Standards

The Team 2 Coding Standards were written by Team 2. The purpose is to make the system clean, consistent, and easy to install. It focuses on programs written in JAVA, but many of the rules and principles are useful even if we write in another programming language. The rules often state reasons for writing in a certain way.

The source repository for this document can be found at [**https://savannah.gnu.org/projects/gnustandards**](https://savannah.gnu.org/projects/gnustandards)**.**These standards cover the minimum of what is important when writing a document. Likely, the need for additional standards will come up. This release of the Team 5 Coding Standards was last updated December 14,2022.

# Purpose of Coding Standards and Best Practices

To develop reliable and maintainable applications, we must follow coding standards and best practices. The naming conventions, coding standards and best practices described in this document are compiled from our own experience and by referring to various Microsoft and non Microsoft guidelines.

* 1. Architecture

1. Accessing database from the UI pages is avoided. Always having a data layer class which performs all the database related tasks is helpful. This will help us support or migrate to another database back end easily.
2. Usage of try-catch in the data layer to catch all database exceptions. This exception handler should record all exceptions from the database.
3. Separate your application into multiple assemblies. Group all independent utility classes into a separate class library. All your database related files can be in another class library.

### Naming Conventions and Standards

|  |
| --- |
| Note :  The terms Pascal Casing and Camel Casing are used throughout this document.  **Pascal Casing** - First character of all words are Upper Case and other characters are lower case.  Example: BackColor  **Camel Casing -** First character of all words, except the first word are Upper Case and other characters are lower case.  Example: backColor |

1. Used Pascal casing for Class names

public class **User**

{

...

}

1. Used Camel casing for Method names

public Products **getProduct**(int id)

{

...

}

1. Used Camel casing for variables and method parameters

int **cartId;**

public Products getProduct(int id)

{

Loger.info(“fetching product with ID” +id);

...

}

1. Used Pascal Casing for interfaces ( Example: **ProductService** )
2. Used Meaningful, descriptive words to name variables. Do not use abbreviations.

Good:

string address

double price

1. No usage of single character variable names like i, n, s etc. Use names like index, temp

One exception in this case would be variables used for iterations in loops:

for ( int i = 0; i < count; i++ )

{

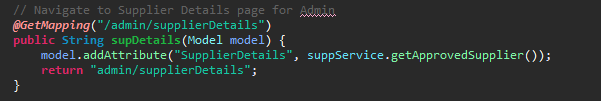
...

}

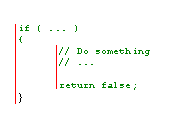
1. No usage of underscores (\_) for local variable names.
2. Do not use variable names that resemble keywords.
3. File name should match with class name.

For example, for the class HelloWorld, the file name should be helloworld.cs (or, helloworld.vb)

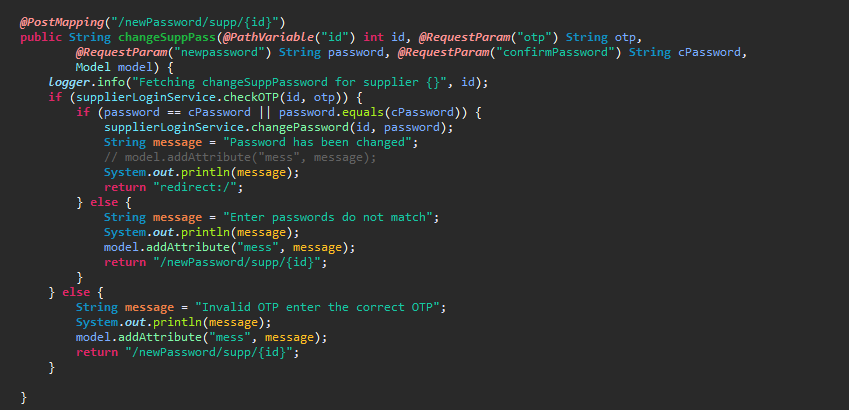
1. Used Pascal Case for file names.
   1. Indentation and Spacing
2. Used TAB for indentation. No usage of SPACES.
3. Comments should be in the same level as the code (use the same level of indentation).



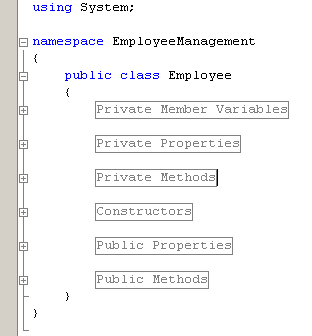
1. Curly braces ( {} ) should be in the same level as the code outside the braces.



1. Nesting the if-else blocks of code to have clear visibility of function.



1. Used #region to group related pieces of code together. If you use proper grouping using #region, the page should like this when all definitions are collapsed.



1. Kept private member variables, properties and methods in the top of the file and public members in the bottom.

* 1. Comments

Good and meaningful comments make code more maintainable. However,

1. Used **//** or **///** for comments. Avoid using /\* … \*/
2. Writing comments wherever required. Good readable code required very less comments. All variables and method names were meaningful, that would make the code very readable and will not need many comments.
   1. Exception Handling
3. In case of exceptions, used a friendly message to display it to the user, but not the actual error with all possible details about the error, including the time it occurred, method and class name etc.
4. Always catch only the specific exception, not generic exception.

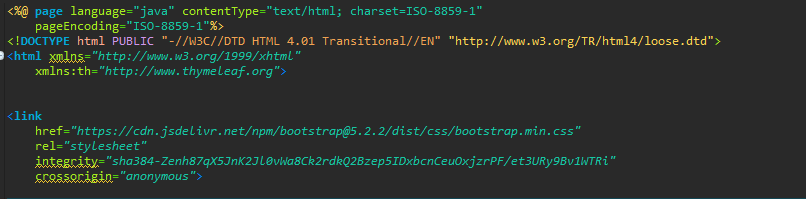


1. No need to catch the general exception in all your methods. Leave it open and let the application crash. This will help you find most of the errors during development cycle. You can have an application level (thread level) error handler where you can handle all general exceptions. In case of an 'unexpected general error', this error handler should catch the exception and should log the error in addition to giving a friendly message to the user before closing the application, or allowing the user to 'ignore and proceed'.

### Best Practices for front-end Coding

###### **1. Declaring a DOCTYPE**

The DOCTYPE declaration should be in the first line of HTML. Actually, it activates the standard mode in all browser. It is recommended that you use the HTML doctype:



###### **2. Closing the tags**

Leaving some tags open is simply a bad practice. Only self-closing tags (  
,

,,,, etc) are valid. Normal elements can never have self-closing tags.

###### **3. Explanation of div that are closing**

If we view most of the website source at the very bottom of the page an almost endless list of closing tags. Without proper code organization, it can be messy. Using indentation and comment for every div is a good practice.

Example:--<!– #header →

###### **4. Include external CSS inside the HEAD tag**

Style sheets can be placed anywhere but the HTML specification recommends that they be placed within the document HEAD tag. The primary benefit is that pages will load faster.

###### **5.Considered placing JavaScript at the bottom**

When loading a script, the browser cannot continue until the entire file has been loaded. If we have JavaScript files in order to add functionality, we placed those files at the bottom, just before the closing body tag. This is a good performance practice and the results are quite noticeable.



###### **5. Usage of lowercase in tags**

It is a good practice to keep markup lower-case. The capitalizing markup will work and will probably not affect how your web pages are rendered, but it does affect code readability. We should keep it simple.

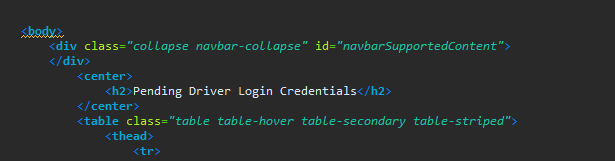
###### **6. Keeping the syntax organized**

When pages will grow, managing HTML can be hard. There are some quick rules that can help us to keep our syntax clean and organized. These include the following:

* Indent nested elements
* Use double quotes, not single or completely omitted quotes
* Omit the values on Boolean attributes

###### **7. Using practical ID and classes names and values**

We should only give elements an ID attribute if they are unique. Classes can be applied to multiple elements that share the same style properties. It is always preferable to name something ID or class, by the nature of what it is rather than by what it looks like. Class names should be all lowercase and use hyphen delimiters.

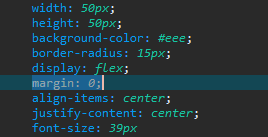


###### **8. Writing CSS using multiple lines and spaces**

It is important to place each selector and declaration on a new line. That will make the code easy to read and edit.

###### **9. Avoiding units on zero values**

One way to easily cut down on the amount of CSS we write is to remove the unit from any zero value. A zero will always be a zero.



###### **10. Modularizing styles for reuse**

CSS is built to allow styles to be reused, specifically with the use of classes. For this reason, styles assigned to a class should be modular and available to share across elements as necessary.

###### **11. Using multiple stylesheets, but be aware of them expanding beyond control**

Depending on the complexity of the design and the size of the site, sometimes it’s easier to make

###### **12. Checking in cross-browser while developing.**

One of the biggest mistake made when developing HTML, CSS, and javascript, was not to test pages on multiple browsers while developing them. Generally, we used to write all code and just view in one of the browsers to see how it was rendered.This becomes the one of best practice to follow.

**4.CODING STANDARDS AND GUIDELINES**

* **Naming Conventions**

Naming conventions are set for declaring any sort of entity or variable within the software. This provides a regularity in the structure of the names defined. For example, ‘every button should have its name in all capitals’ can be a naming convention.

* **Rules For Global Variables Declaration**

There are rules which define what type of variables can be declared as global and what cannot. This helps to maintain the confidentiality of the data and is also helpful in cases where the access rights are to be provided only to the selected group of users.

* **Contents Of Headers**

The data that must be there in the header of each module has a pre-defined format. This usually includes the name of the module, data of creation, date of approval, developer’s or author’s name, synopsis, description of the module, description of the variables and functions defined in the module, etc.

* **Error Return Conventions**

There are certain conditions in which the input data is bounded within some range. If in such cases, the data entered by the user exceeds the particular range, then an error message should be displayed. The range and criteria for which the error should pop up is defined in the error return conventions.

* **Use Coding Style That Is Easy To Understand And Is Globally Accepted**

It is recommended that the coding style that is being followed must be easy to understand and should be similar to that which is globally accepted or else it would be difficult for the user to understand it and also it would take time for the developers to develop a habit of following a completely new coding format.

* **The Length Of Functions Or Methods Should Be Small**

It is a good practice to keep the body of the functions small, nearly about 10-12 code lines. This makes the code readable and easily modifiable.

* **Do Not Unnecessarily Use Statements That Break The Control Flow Of Statements**

It is advised not to use jump, goto and function calls unnecessarily as it takes extra time for the processor to break the going flow, search for the mentioned statement, load those statements, execute them and then return to the previous calling function. Therefore, try to avoid such situations as much as possible.

* **Choose Identifier Names Uniquely Within The Module**

Every identifier at least within the module must have a unique name to avoid ambiguity.

* **Indentation**

Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly.

Some of the spacing conventions are given below:

* There must be a space after giving a comma between two function arguments.
* Each nested block should be properly indented and spaced.
* Proper Indentation should be there at the beginning and at the end of each block in the program.
* All braces should start from a new line and the code following the end of braces also start from a new line.
* **Try not to Use GOTO Statement**

GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

**5.UI STANDARD**

* 1. **PAGE LAYOUTS**
  2. **LABEL**
  3. **NOTIFICATION**
  4. **TEXT BOX**
  5. **BUTTON**
  6. **ERROR MESSAGE**
  7. **INPUT TEXT**
  8. **ICONS**
  9. **LIST BOX**
  10. **INFORMATION DIALOG**
  11. **CONFIRMATION DIALOG**
  12. **INPUT TEXT DATA**

**6.JAVA CODING STANDARDS**

* 1. **MEANINGFUL NAMES**
* As an illustration, naming conventions deal with everything related to appropriately naming a variable, constant, method, class, and interface, among other things.
* When you name identifiers (methods, classes, and variables), it is essential to do so in a way that is self-explanatory, distinct, and easy to read.
* Most importantly, you must always ensure to write readable codes easily understood by humans (and not just write codes to satisfy the compiler). It is a standard Java coding practice used by millions of developers.
  1. **CLASSES**
* Another critical coding practice in Java you should always attempt to implement when writing is organizing member variables of a class through their scopes.
* In other words, it would be ideal if you, as a Java developer, sorted each member according to the visibility of the access modifiers: private, default (package), protected, and public. A blank line must separate each group
  1. **FUNCTIONS**
* There must be a space after giving a comma between two function arguments.
* Each nested block should be properly indented and spaced.
* All braces should start from a new line and the code following the end of braces also start from a new line.
  1. **COMMENTS STANDARDS**
* All Classess must have comments – Explaining the responsibility of the class
* All methods must have the method level comments – explaining what it is doing
* All complex logic must have inline comments – Explaining both what and why it is done
* One line or end line comments begins //
* Standard method level comment /\*
* Place the Comments appropriately so that Javadoc can be generated easily
* Clean code needs less comments and comments should be informative
  1. **FORMATING STANDARDS AND GUIDELINES**
* All files must be formatted
* All files must not have unused imports
* All files must have relevant imports
* No wildcard imports
* Braces is mandatory for all if, else, for even it is one line
* No line break between if and else
* No Empty Blocks
  1. **ERROR HANDLING**
* Use Checked Exception for Recoverable error and Unchecked Exception for programming error
* Always provide meaning full message on Exception
* Remember Exceptions are costly in terms of performance
  1. **PROGRAMMING PRACTICES**
* Don’t make any instance or class variable as public
* Use class to access the static variable or method
* Create or Initialize objects only if necessary
* Remove all system out and use Logger wherever applicable
* Use JPA for simple and single table insert and rest use jdbc template

# JENKINS PRACTICES

# It is an open-source tool with great community support

# Jenkins it is a Continuous Integration and Continuous delivery tool.

# Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project

# Jenkins achieves Continuous Integration with the help of plugins

# It is built with Java and hence, it is portable to all the major platforms

1. **SQL PRACTICES**

# UML ANALYSIS MODELS

## Use Cases

This subsection extends upon the functional requirements given in Section [3.1](#_bookmark26) through the presentation of detailed use cases. To facilitate an unambiguous and clear view of how the end- users interact with the subject HSOS, the actors (end-users) involved in the use cases, a use case diagram and detailed use case descriptions are provided. The use cases that find representation are Log In, Log Out, Accept Order, Deliver Item, Process Bankcard Payment, Process Cash Payment, Place Order,, Pay Bill, Accept/Reject Item and Indicate Item Ready.

### Use Case Descriptions

Table presents the Log In use case description to show the interaction of Actors when logging into the system.

|  |  |
| --- | --- |
| **Use Case** | Log In |
| **Primary Actor** | Admin, Supplier and Truck Driver |
| **Goal In Context** | Enable Actors access to the system through a app |
| **Preconditions** | The Actors has a valid username and password and is not already logged in |
| **Scenario** | 1. The Actors selects ‘Log In’ from the app menu 2. The app prompts the user for their username and password 3. The Actors enters their username and password 4. The app enables access to the system according to access control |
| **Exceptions** | The Actors enters an invalid username or password |

**Table : Log In Use Case Description**

[Table](#_bookmark2) presents the Log Out use case description to show the interaction between a Admin and a Navbar when logging out of the system.

|  |  |
| --- | --- |
| **Use Case** | Log Out |
| **Primary Actor** | Admin, Supplier and Truck Driver |
| **Goal In Context** | Disable Actors access to the system through a app |
| **Preconditions** | The actors is already logged in |
| **Scenario** | 1. The actors selects ‘Log Out’ from the Navbar 2. The Navbar disables access to the system |

**Table: Log Out Use Case Description**

Table presents the Place Order use case description to show the interaction between a customer and a surface computer when placing an order.

|  |  |
| --- | --- |
| **Use Case** | Place Order |
| **Primary Actor** | Admin |
| **Goal In Context** | Place an order for menu items from the Admin |
| **Preconditions** | The admin has been set up the applications |
| **Trigger** | The admin wants to place the order |
| **Scenario** | 1. The Admin selects 'New Order' from the menu and can able to make the new orders |

**Table : Place Order Use Case Description**

Table presents the View Details use case description to show the interaction between a Actors and a surface computer when viewing orders.

|  |  |
| --- | --- |
| **Use Case** | View Order/Track orders |
| **Primary Actor** | Admin, Supplier, Truck drivers |
| **Goal In Context** | Place an order for menu items from the Admin |
| **Preconditions** | The actors has been set up the applications |
| **Trigger** | The actors should view order details which includes (Pick-up, drop, supplier details, Truck driver details, Temperature details) |
| **Scenario** | 1. The Admin selects 'New Order' from the menu and can able to make the new orders. |

## Class Diagram

### Class descriptions

The following subsection presents descriptions for the classes identified for the subject GPS based Tracking Application with Logistics Management.

#### Products

This class represents the items from seller. A Product that is part of an order may contain a requirement. The product contains the menu product's name, price, description, status and an image or model to depict the product on a surface computer or display.

#### Cart

This class represents a collection of products. An Order can be in either placed, cancelled or the approved state. An Order will typically be deleted shortly after it has been approved.

#### Customer

This class represents the user who buys products and places orders.

#### Seller

This class represents the user who displays the products and adds into the list of products

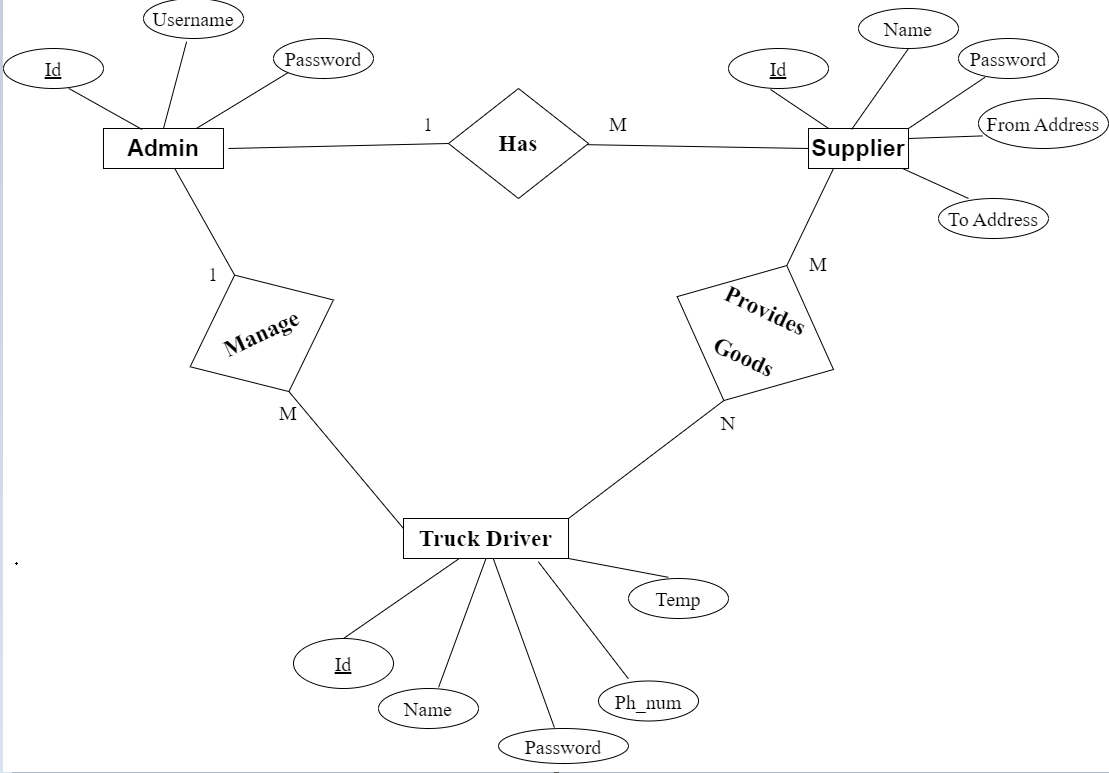
#### Orders

This class represents the status of selected items and proceeds towards the payment page leading towards successful delivery of products to customers

Diagram

Description automatically generated

Figure 4.2.1 GPS based Tracking Application with Logistics Management Class Diagram



**Figure 4.2.2 Entity-Relationship Diagram**

# Diagram, schematic Description automatically generated

**Figure 4.3 Use-case Diagram**

# Flow Chart for Admin, Supplier and Truck Driver

# 

# 

# 