Parking Management System

COMP 246 (SEC.006)

Instructor: Mohammad Anwar Hossain

**Students:**

* Aesha Bhatt 301034022
* Minh Chau Hoang 300946197
* Xiaohui Hao 300878632
* Gaofeng Pan 300973143
* Carlos Zabaleta 301002235

**TABLE OF CONTENTS:**

**PART- A**

Problem Statement--------------------------------------------------------------------3

Workflows by Subsystem:------------------------------------------------------------4

Activity UML Diagrams-----------------------------------------------------------------4

a)Use Cases & Actors By Subsystem---------------------------------------------7

b) Use Case Diagram----------------------------------------------------------------10

c) Users Stories for each goal listed by subsystem---------------------------11

Domain Class Diagram------------------------------------------------------------15 ER Diagram----------------------------------------------------------------------------16

**PART -B**

Detail class diagram and Package diagrams-----------------------------------17

System Sequence diagram---------------------------------------------------------20

Sequence Diagrams------------------------------------------------------------------24

State Machine Diagrams-------------------------------------------------------------25

**PART -C**

Mock-up UI -----------------------------------------------------------------------------26

ERD model with database schema-----------------------------------------------29

Skeletal or stub code generated from Class Diagram------------------------31

Component & Deployment diagrams---------------------------------------------31

Technology stack for the Software Implementation----------------------------38

Project Plan-----------------------------------------------------------------------------39

**Vision Statement**:

College Parking Application is a mobile app which gives the facility to the students and employees to have all the information related to the College Parking as payments, real-time parking slot availability and the car’s pick up location.

**Problem Description**:

During weekdays it is more difficult to find a parking slot available in different colleges. It is a waste of time driving around to get an available slot, besides it may be difficult to remember your parking spot location, in case you were in a hurry or you forgot where you parked your car.

Hence we believe the best solution is to develop a mobile application to make different payments, it provides the current location of your car and it also provides the parking slot availability in the college.

**System Capabilities:**

* It provides students or employees the accurate maps of the college’s parking.
* It displays the current location of your vehicle.
* It displays the best route to the nearest parking slot available.
* It reminds you where you parked your car.
* It provides a method of payment for the students and the employees.
* It displays the fastest route from your location to the parking slot where your car is located.
* The system doesn’t provide the same quality of service as the students or faculty members.

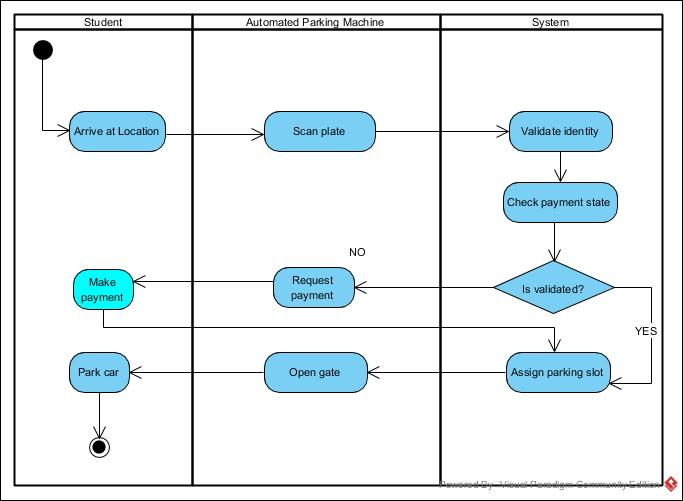
**Business Benefits:**

* Makes it easier for the customers to find a parking slot available, therefore the customer would save time finding a parking slot available and getting to the parking slot where the car is located.
* Reduce the cost of managing the system.

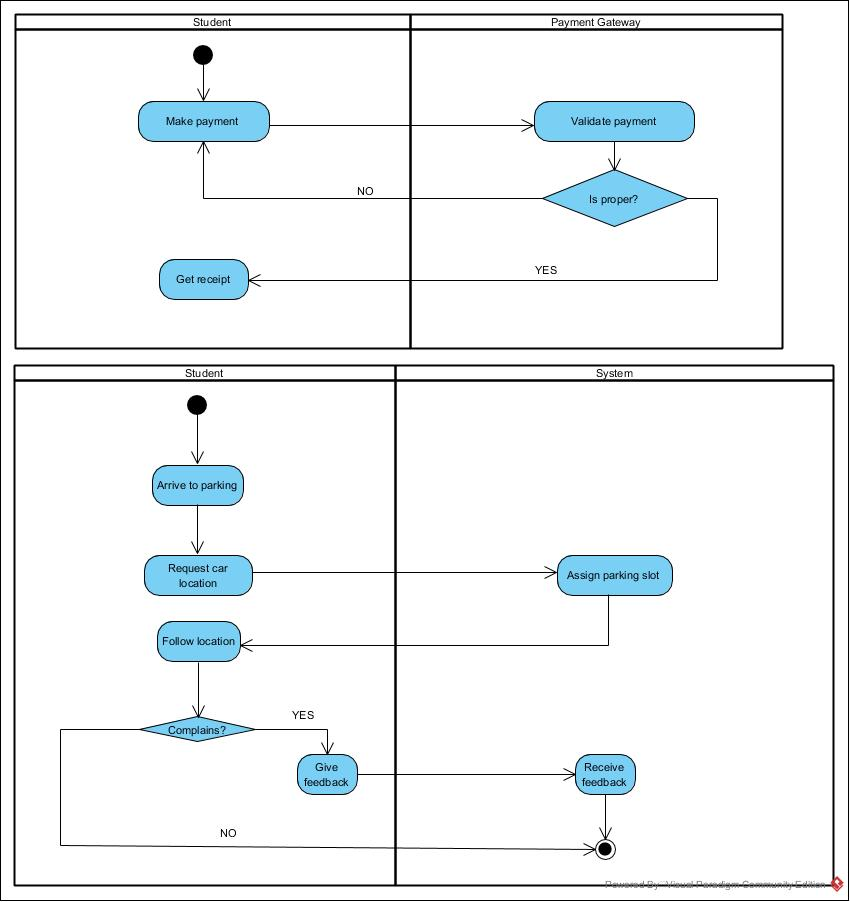
**Workflows by Subsystem:**

**Activity Diagram**:

The activity diagram stimulates the workflow of how our application will work. The diagram below shows activities of student when he enters the location and his number plate is scanned. His identity is then verified and he is assigned the parking slot if everything is completed.

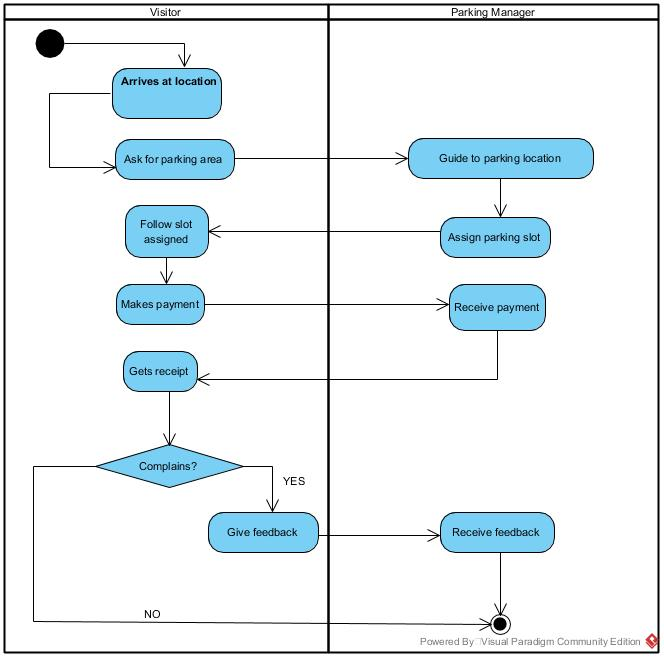


*Activity Diagram for Automated Parking Machine Subsystem*

**

*Activity Diagram for Payment Stub*

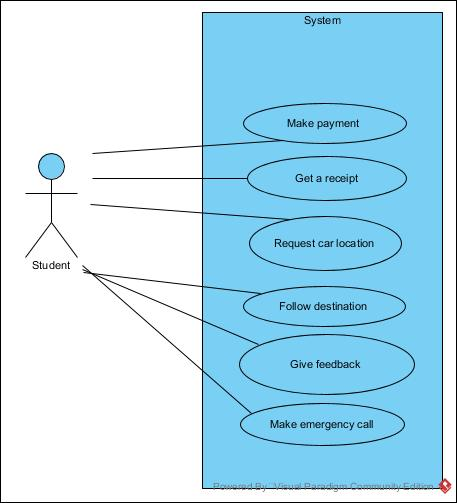
In this activity diagram, student who is a part of the system initiates the payment and gets the receipt. He then arrives to the parking and asks for the nearby available parking slot. On scanning his number plate, he is provided the location which he follows. He also can place any complain he wants to by giving the feedback which is totally optional.



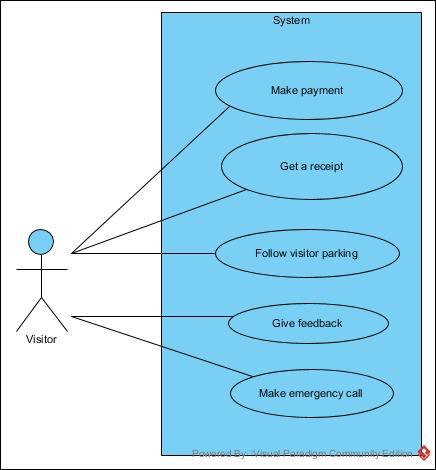
*Activity Diagram for Payment Subsystem*

This activity diagram focuses on how a visitor’s activity is being recorded once he arrives to the location. As he is just a visitor, he is not an active part of the system so he doesn’t have an access to the system. Hence, he contacts the parking manager so that he could get his parking area and can pay for it accordingly.

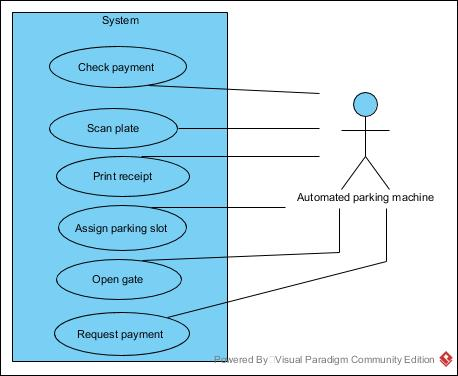
**Use Cases and Actors by Subsystem:**

****

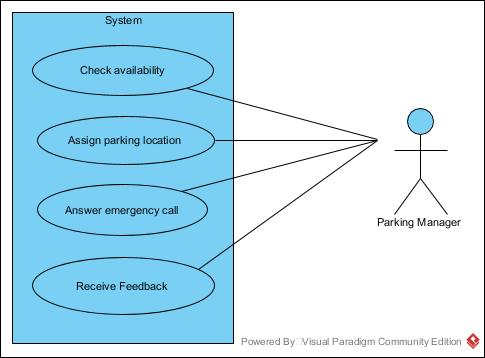
*Use Case for Student Subsystem*

****

Use Case for Visitor Subsystem



*Use Case for Machine Subsystem*



*Use Case for Parking Manager Subsystem*

**Use Case Diagram of the System:**

The Use Case Diagram familiarises with the actors and their functionality.

The actors are User (who uses the system), Mobile Phone Application,Automated Parking System and the Parking Manager.

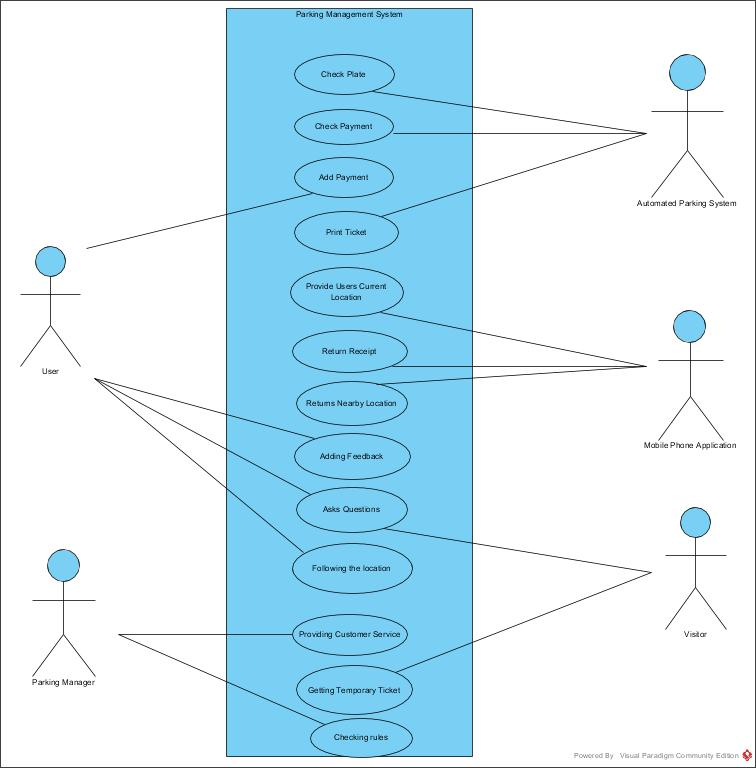


Fig 2.5- Use Case Diagram

**Users Stories:**

|  |  |  |
| --- | --- | --- |
| Use Case | User Story | Acceptance Criteria |
| Follow  Returned  Location | As a student/employee, I want to follow the navigation to an empty slot on my phone screen | 1. The APP should be installed, required rights should be given.  2. 4G Internet is required.  3. GPS chips in the phone are needed.  4. Maps of the parking lot are highly accurate.  5. Customer account information and car plate should be stored on the database.  6. Two states of slot “occupied” “empty” should be dynamic real-time updated on the database. |
| Add Payment | As a student/employee, I want to add payment automatically. | 1. Customer account information and car plate should be stored on the database.  2. Gate camera should scan plate and identify the customer.  3. Check time and cut money from prepayment account automatically. |
| Add Feedback | As a student/employee, I want to add feedback or make an emergency call. | 1. Words or pieces of voice can be recorded in APP for feedback.  2. The emergency call can be made by mobile or through phones which are installed at every gate of the parking lot. |
| Receive Feedback | As a parking manager, I want to receive feedback in APP | 1. Feedback can be a list in the database server. |
| Provide  Customer  Service | As a parking manager, I want to answer a customer’s emergency call and provide customer service. | 1. Customer’s current location should be displayed to the manager as soon as the emergency call is connected. |
| Make Payment | As a visitor, I want to make payment in payment machine. | 1. The visitor has own parking area.  2. The state of the slot in the visitor area is not controlled by the database system.  3. Visitor parking area provides payment machine and accessing card.  4. Figure out parking time by inserted accessing card picked up when entering. |

**Student**

**Make payment**

As a student/employee, I want to make a payment at the beginningof the semester so that I can park my car in the parking area.

**Get a receipt**

As a student/employee, I want to get a receipt for the payment I have made so that I can have a proof of my payment.

**Request car location**

As a student/employee, I want to ask for the location of my car so that I find my car.

**Follow destination**

As a student/employee, I want to follow the direction to an empty slot on my phone screen so that I can park my car.

**Give feedback**

As a student/employee, I want to give feedback so that I can give my opinion of the service.

**Make emergency call**

As a student/employee, I want to make an emergency call so that I can report an emergency.

**Visitor**

**Ask for parking area**

As a visitor, I want to inquire for the visitor parking area so that I can park my car.

**Follow visitor parking**

As a visitor, I want to follow the direction for the visitor parking area so that I can park my car.

**Makes payment**

As a visitor, I want to make a payment when I leave so that I can leave the parking area.

**Get a receipt**

As a visitor, I want to get a receipt for the payment I have made so that I can have a proof of my payment.

**Give feedback**

As a visitor, I want to give feedback or make an emergency call so that I can give my opinion of the service.

**Make emergency call**

As a visitor, I want to make an emergency call so that I can report an emergency.

Parking manager

Automated parking machine

**Scan plate**

As an automated parking machine, I want to scan the car’s plate so that I can check if the customer is eligible to park.

**Check payments**

As an automated parking machine, I want to check if the customer made a payment so that I can check if the customer has any previous holds.

**Assign parking slot**

As an automated parking machine, I want to assign a nearby parking slot to the customer so that I can help the customer where to park.

**Print receipt**

As an automated parking machine, I want to print the receipt for the visitor so that the visitor can complete the payment.

**Parking Manager**

**Receive feedback**

As a parking manager, I want to receive feedback so that I can help the customer.

**Check availability**

As a parking manager, I want to check the availability of a parking slot so that I can assist the visitor.

**Assign parking location**

As a parking manager, I want to assign an available parking slot to the visitor so that he can park his car.

**Answer emergency call**

As a parking manager, I want to receive feedback so that I can help the customer.

**Domain Class Diagram:**

The multiplicities of the Domain Class Parking Management System can be explained as follows:

The user has only one Parking Account but the Parking account may have zero or more Payment details. Parking Manager holds zero or more parking information, also it checks through zero or more parking availability. It can interact with zero or more users but the user can interact with only one Parking Manager. Users specialize to Students, Employees, and visitors and hence acts as a Super Class.

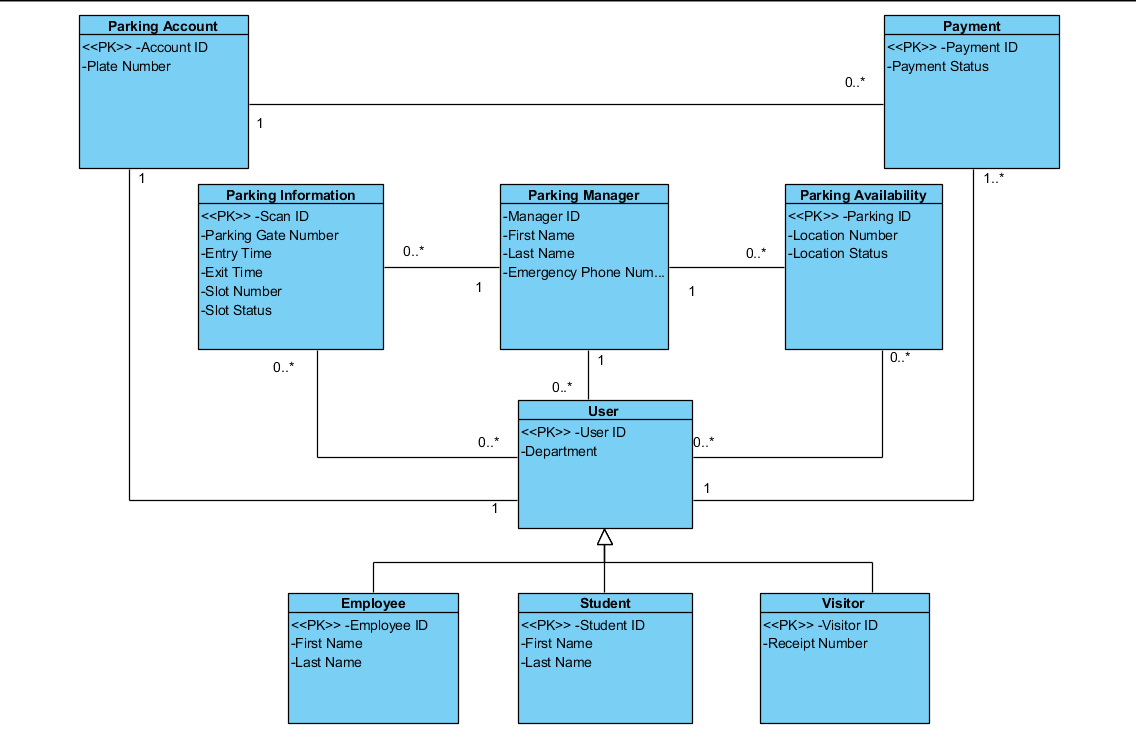
****

Fig 3.1- Domain Class Diagram

**Entity-Relationship Diagram:**

The Entity Relationship Diagram consists of four classes - User, Phone Information, Parking Data and Parking Manager with their relevant attributes.

The cardinalities for the ER Diagram can be represented as:

Student, Employee, Visitor generalizes to User who has one-one cardinality with his Phone(A user can get information through one Phone only). The parking information returns data for zero or user to zero or more devices. The Parking Manager follows only one Parking data provided but can interact with zero or more users.

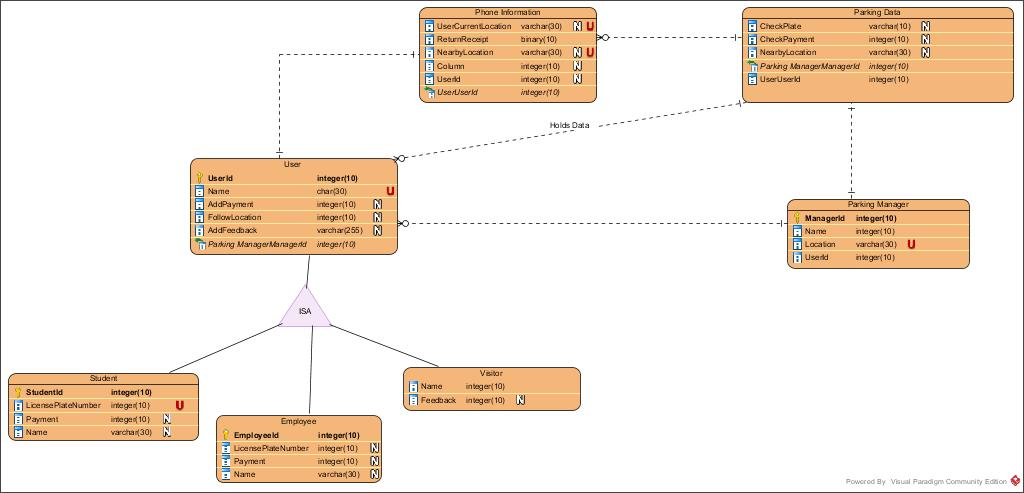
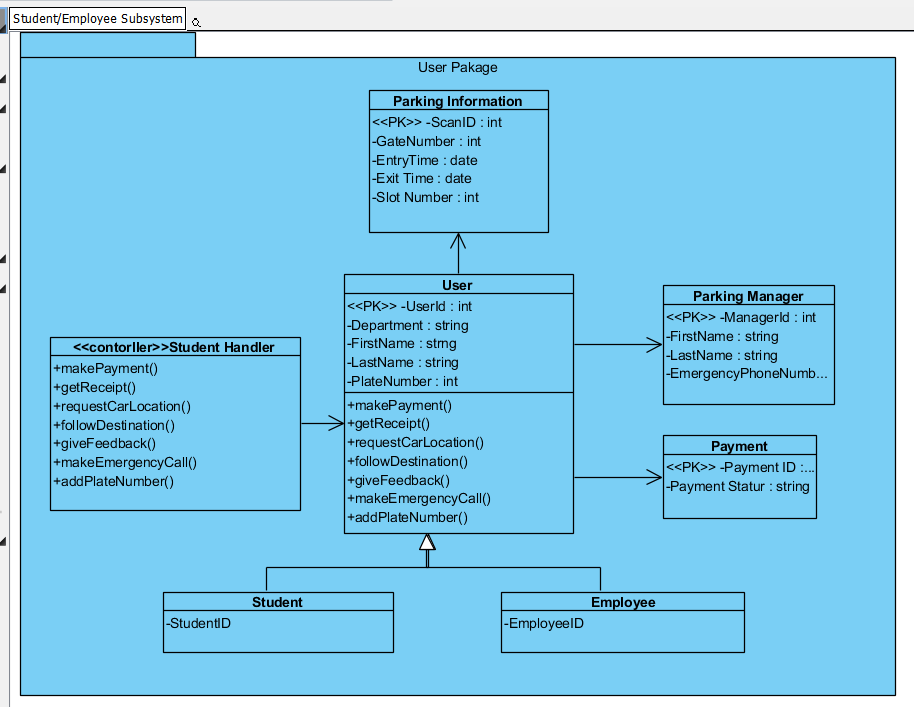


Fig 4.1- Entity Relationship Diagram

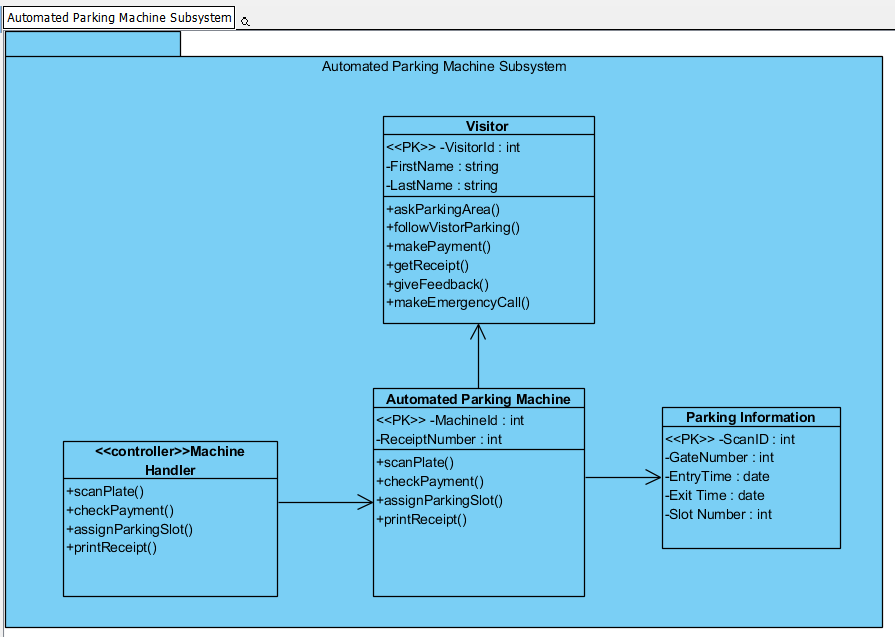
**PART B:**

**Detailed Design Class Diagram - packaged by Subsystem**

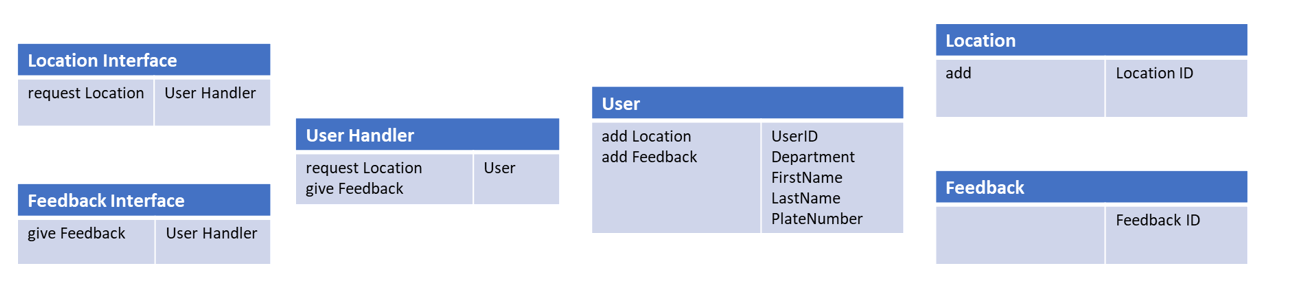
1.0 User Subsystem

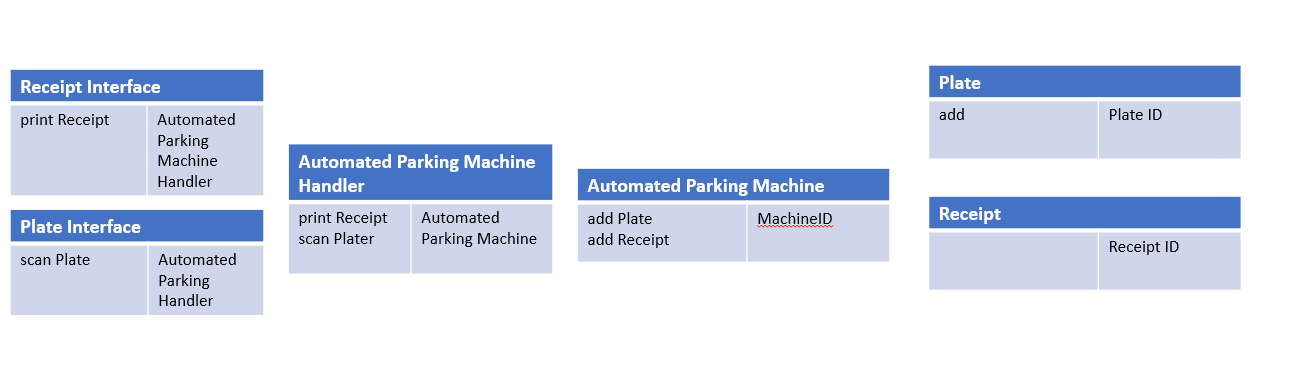


2.0 Automated Parking Machine Subsystem

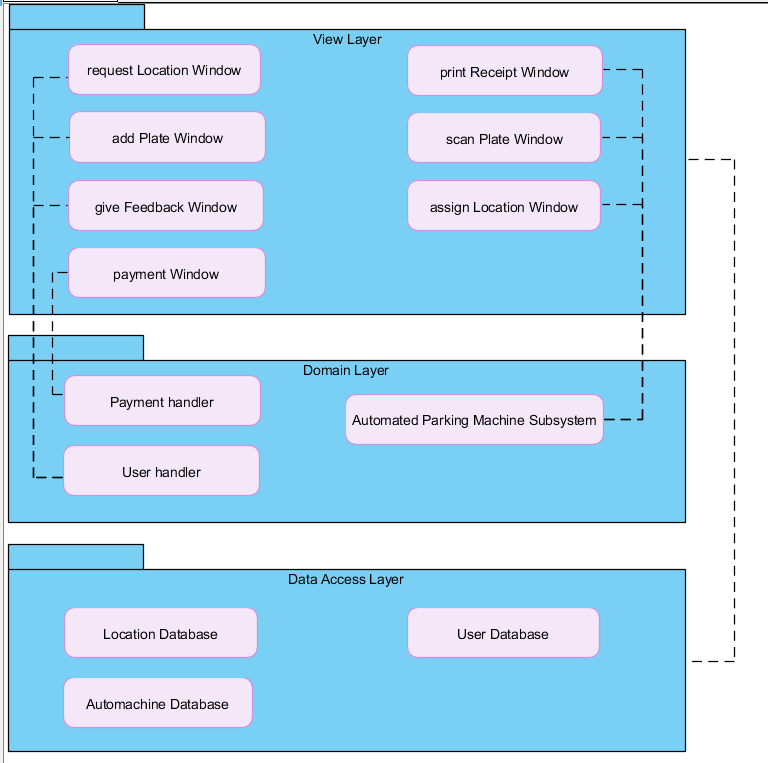


**CRC Cards**

****

****

**Package Diagram**

****

Dependencies:

View layer depends on Data Access layer

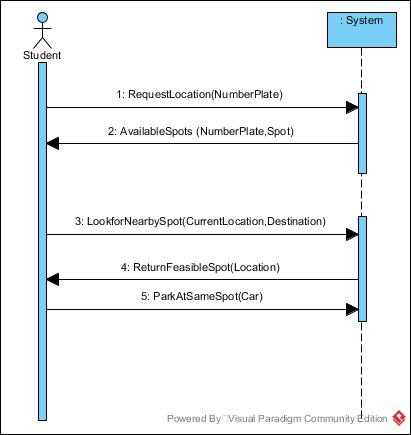
RequestLocationWindow, AddPlateWindow, GiveFeedbackWindow depends on User Handler

Payment Window depends on Payment Handler

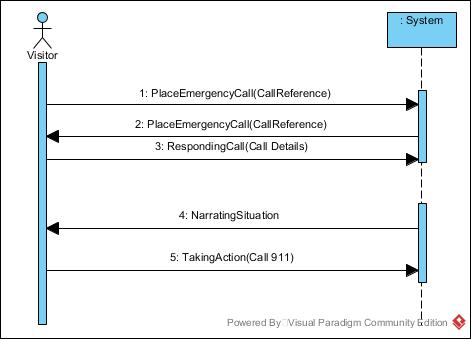
PrintReceiptWindow, ScanPlateWindow, AssignLocationWindow depends on Automated Parking Machine Subsystem

**System Sequence Diagram:**

System Sequence Diagrams shows the interaction of an object with the system. Here, we would develop 5 System Sequence Diagrams to understand the object interaction better.

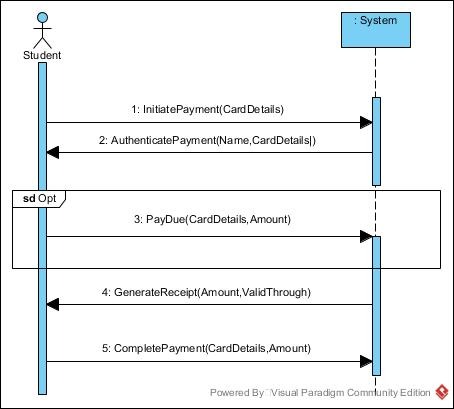
**g**

*Student System Sequence Diagram*

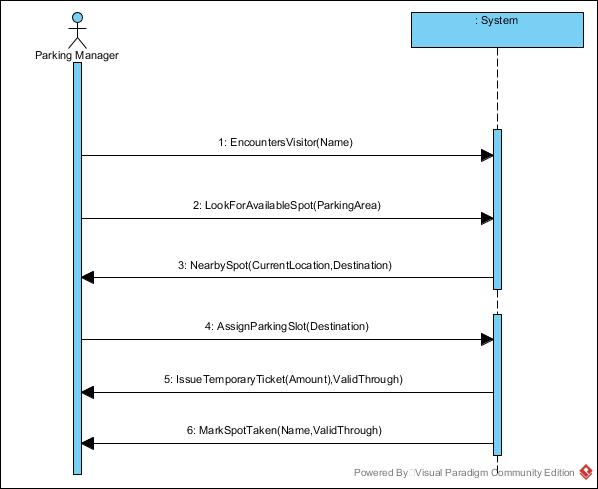
****

*Visitor System Sequence Diagram*

Here for Visitor SSD , the user places an emergency call and the system handles the situation by taking an action.

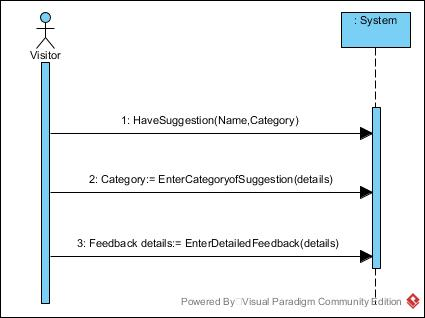
****

*Student System Sequence Diagram*

****

*Parking Manager System Sequence Diagram*

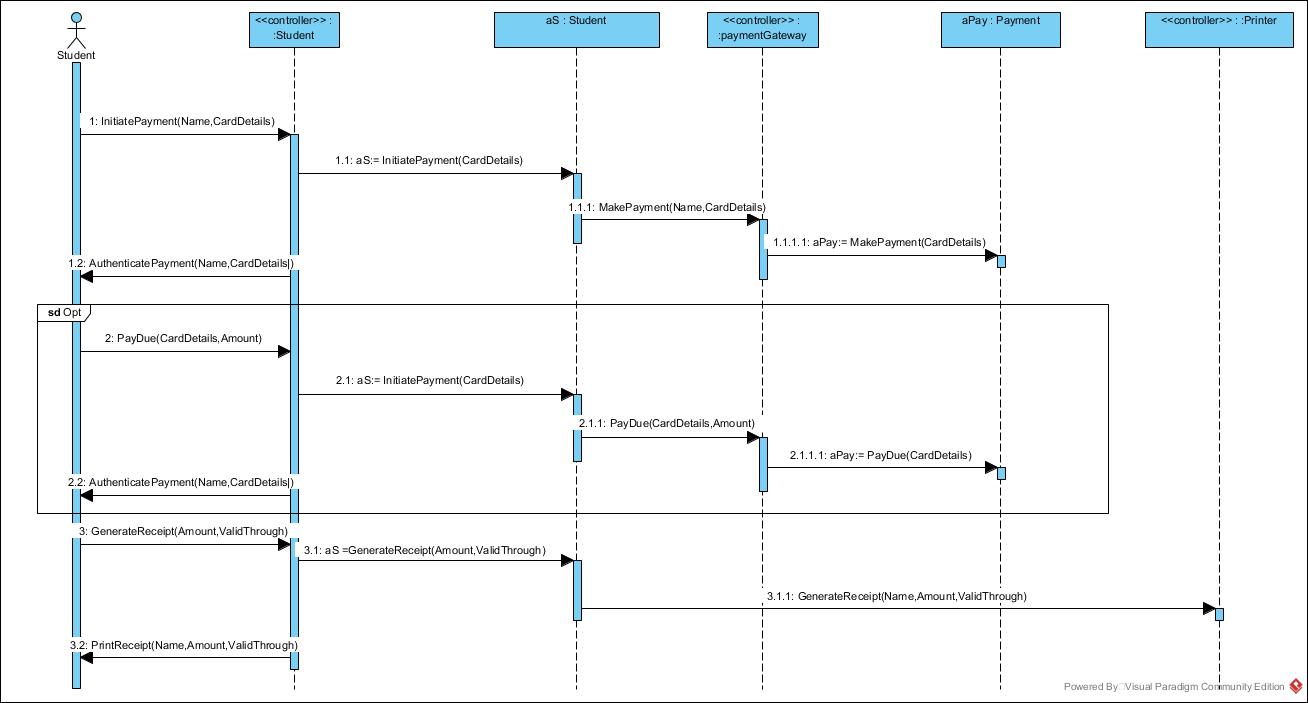
When encountering the visitor, parking manager starts looking for available spots in the visitor’s parking. As visitor is not an active part of the system , parking manager comes into the role of assigning location. It then issues temporary ticket which has time it is valid through. Then the spot finally is marked as “Taken”.

****

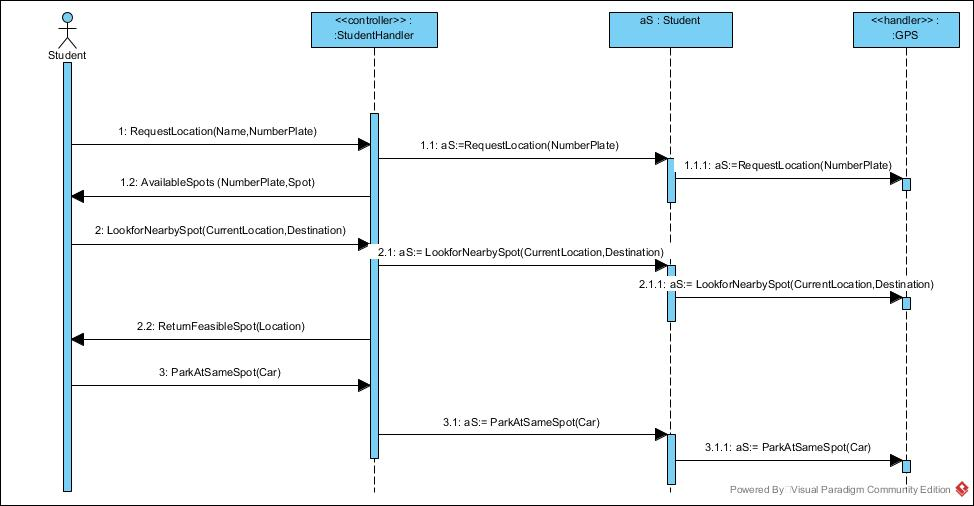
*Visitor System Sequence Diagram*

If Visitor has a suggestion, he enters the category the suggestion is for and provides details of it. Then he enters the specific comments/ feedbacks he has for the system in order to make it more robust.

**Sequence Diagram:**

****

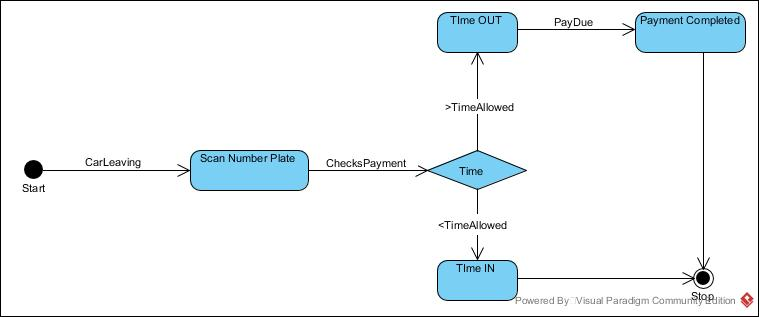
*Student Sequence Diagram for Get Receipt*

****

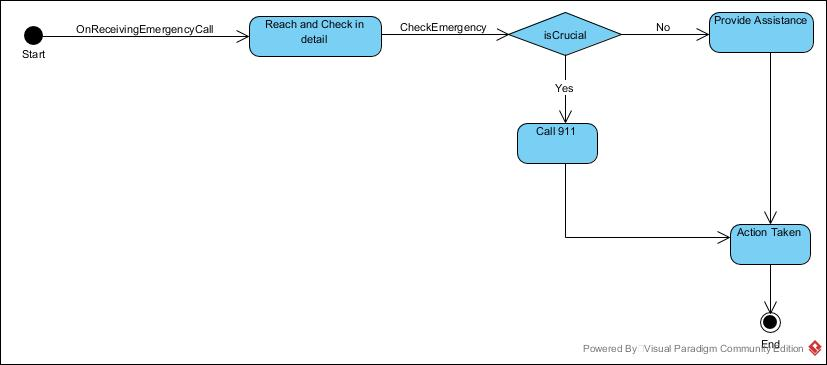
*Student Sequence Diagram for Request Car Location*

**State Machine Diagram:**

Usually, state machine diagrams are developed for knowing the events and how that events change the state of the particular object. In the first State Machine Diagram, when the visitor’s car is leaving, system checks the payment for which it has to check the time the car entered in. If it is more than the specific time , user has to pay the overdue.



*State Machine Diagram that checks Due Payment*



*State Machine Diagram Emergency Call- Parking Manager*

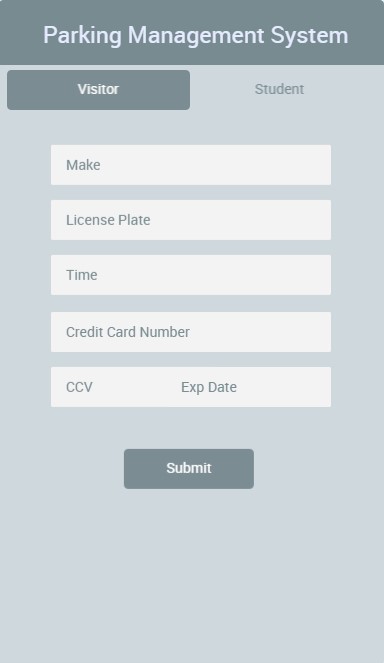
**PART C**

**Mock up UI:**

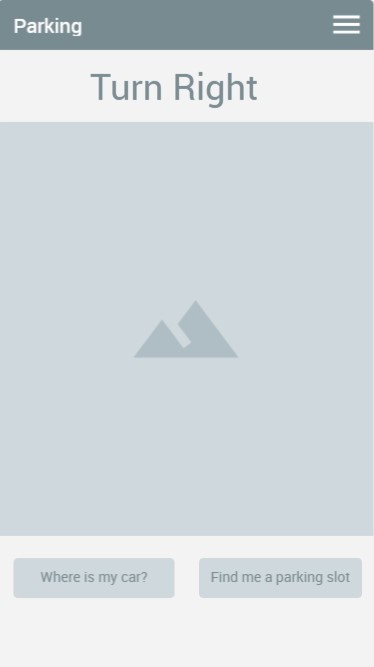
Now that we are done with the designing phase of the project, the next step is the implementation of what the system s supposed to do.

For the same, we have created some UI interfaces so that the functionality of the system can be understood clearly.

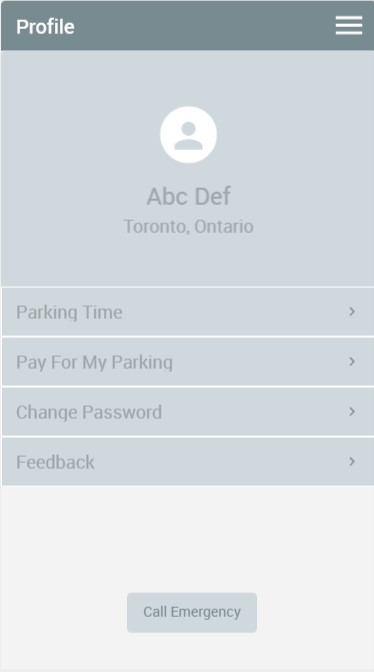
This UI refers to the Student/ Faculty login interface on the app, they can request a new password if they forget it as well.



Visitor can register and pay for their parking as well. The app will require visitor's vehicle information.



This UI interface is of navigation, i.e. user can request to get a parking slot or get them the way to their vehicle when they want to leave. Two buttons over here namely, Find me a parking slot returns the available empty slots to the user when he wants to park his car. Vice versa, when he wants to leave, he would like to find his car - he can do so using Where is my car? button which guides him back to his car.



Interface for the user profile, we can check our parking time, pay more for the parking or do monthly payment, change the password and give out feedback for the app

**ERD Model With Database Schema:**

**Functional Sub-System Chart**

|  |  |  |
| --- | --- | --- |
| Subsystem | Use Case | User |
| Authentication | Register account | Visitors |
| Authentication | Login | Student, employee, visitor |
| Payment | Make payment | Student, employee, visitor |
| Navigation | Follow destination | Student, employee, visitor |
| Navigation | Request car location | Student, employee, visitor |
| Report | Give feedback | Student, employee, visitor |
| Emergency | Make emergency call | Student, employee, visitor |

**Creating Database code:**

/\* User \*/

Create table User

(UserID Number(4) Not Null,

FirstName VARCHAR2(30) Not Null,

LastName VARCHAR2(30) Not Null,

Address VARCHAR2(30) Not Null,

AccountID Number(4) Not Null,

PaymentID Number(4) Not Null,

Constraint UserId\_PK Primary Key(UserID),

Constraint AccountID\_FK Foreign Key(AccountID) References Parking Account(AccountID),

Constraint PaymentID\_FK Foreign Key(PaymentID) References Payment(PaymentID));

/\* Parking Information \*/

Create table Parking Information

(ParkingID Number(4) Not Null,

EntryTime date Not Null,

ExitTime date Not Null,

UserID Number(4) Not Null,

Constraint ParkingID\_PK Primary Key(ParkingID),

Constraint UserID\_FK Foreign Key(UserID) References User(UserID));

/\* Parking Avaliability \*/

Create table Parking Avaliability

(AvaliabilityID Number(4) Not Null,

LocationNumber Number(4) Not Null,

LocationStatus Number(4) Not Null,

Constraint AvaliabilityID\_PK Primary Key(AvaliabilityID),

Constraint UserID\_FK Foreign Key(UserID) References User(UserID));

/\* Payment\*/

Create table Payment

(PaymentID Number(4) Not Null,

PaymentStatus VARCHAR2(30) Not Null,

Constraint PaymentID\_PK Primary Key(PaymentID));

/\* Parking Account \*/

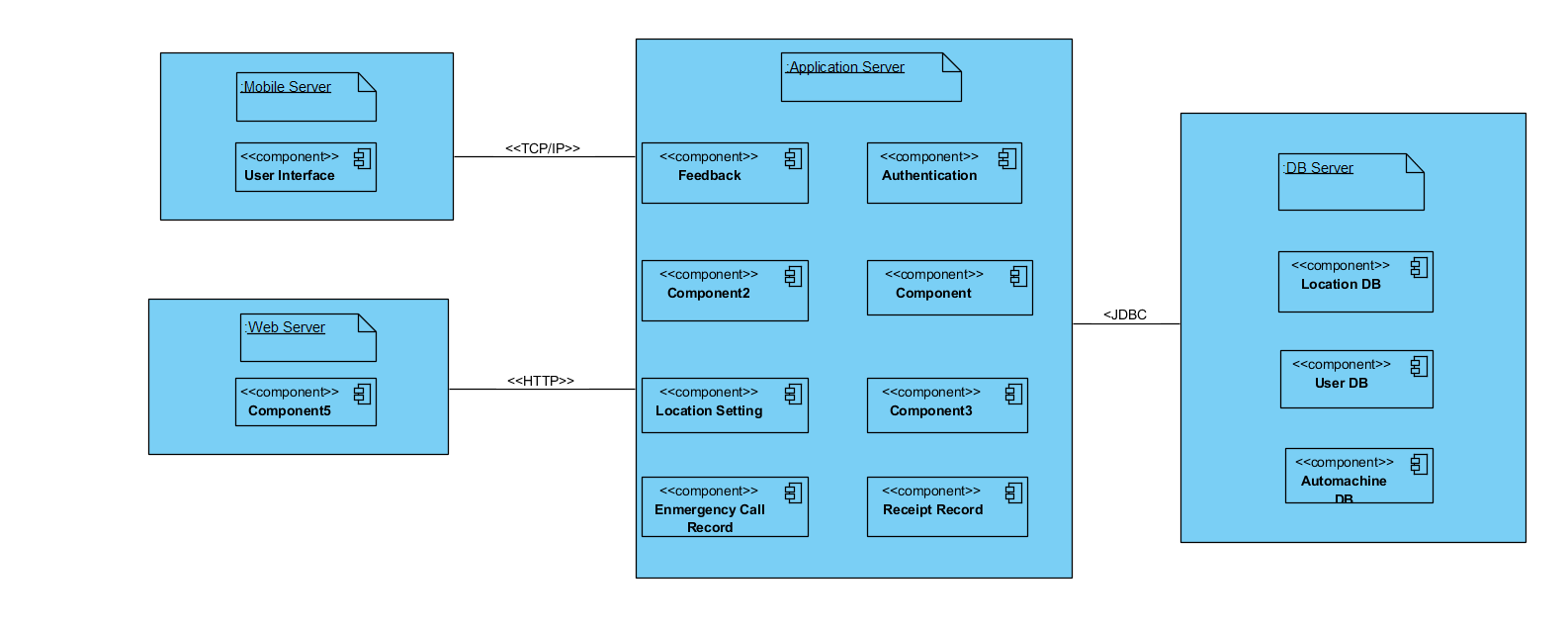
Create table Parking Account

(AccountID Number(4) Not Null,

PlateNumber VARCHAR2(30) Not Null,

Constraint AccountID\_PK Primary Key(AccountID));

**Component Diagram**

****

**Skeletal or stub code generated from Class Diagram :**

1. **Automated Parking Machine:**

#ifndef AUTOMATED PARKING MACHINE\_H

#define AUTOMATED PARKING MACHINE\_H

namespace Domain\_Class {

class Automated\_Parking\_Machine {

public:

int MachineId;

int ReceiptNumber;

void scanPlate();

void checkPayment();

void assignParkingSlot();

void printReceipt();

};

}

#endif

**Automated Parking Machine C++ file:**

#include "..\..\Users\chaus\Downloads\Automated Parking Machine.h"

void Domain\_Class::Automated\_Parking\_Machine::scanPlate() {

// TODO - implement Automated Parking Machine::scanPlate

throw "Not yet implemented";

}

void Domain\_Class::Automated\_Parking\_Machine::checkPayment() {

// TODO - implement Automated Parking Machine::checkPayment

throw "Not yet implemented";

}

void Domain\_Class::Automated\_Parking\_Machine::assignParkingSlot() {

// TODO - implement Automated Parking Machine::assignParkingSlot

throw "Not yet implemented";

}

void Domain\_Class::Automated\_Parking\_Machine::printReceipt() {

// TODO - implement Automated Parking Machine::printReceipt

throw "Not yet implemented";

}

**2) Employee:**

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

namespace Domain\_Class {

class Employee : Domain\_Class::User {

private:

int Employee\_ID;

int First\_Name;

int Last\_Name;

};

}

#endif

**3) Parking Availability:**

#ifndef PARKING AVAILABILITY\_H

#define PARKING AVAILABILITY\_H

namespace Domain\_Class {

class Parking\_Availability {

private:

int Parking\_ID;

int Location\_Number;

int Location\_Status;

};

}

#endif

**4) Parking Information**

#ifndef PARKING INFORMATION\_H

#define PARKING INFORMATION\_H

namespace Domain\_Class {

class Parking\_Information {

public:

int Scan\_ID;

int Parking\_Gate\_Number;

int Entry\_Time;

int Exit\_Time;

int Slot\_Number;

int Slot\_Status;

};

}

#endif

**5) Parking Manager:**

#ifndef PARKING MANAGER\_H

#define PARKING MANAGER\_H

namespace Domain\_Class {

class Parking\_Manager {

public:

int Manager\_ID;

string First\_Name;

string Last\_Name;

int Emergency\_Phone\_Number;

void checkAvailabillity();

void assignParkingLocation();

void answerEmergencyCal();

void recieveFeedback();

};

}

#endif

**Parking Manager- C++ file**

#include "..\..\Users\chaus\Downloads\Parking Manager.h"

void Domain\_Class::Parking\_Manager::checkAvailabillity() {

// TODO - implement Parking Manager::checkAvailabillity

throw "Not yet implemented";

}

void Domain\_Class::Parking\_Manager::assignParkingLocation() {

// TODO - implement Parking Manager::assignParkingLocation

throw "Not yet implemented";

}

void Domain\_Class::Parking\_Manager::answerEmergencyCal() {

// TODO - implement Parking Manager::answerEmergencyCal

throw "Not yet implemented";

}

void Domain\_Class::Parking\_Manager::recieveFeedback() {

// TODO - implement Parking Manager::recieveFeedback

throw "Not yet implemented";

}

**6) Payment**

#ifndef PAYMENT\_H

#define PAYMENT\_H

namespace Domain\_Class {

class Payment {

public:

int Payment\_ID;

int Payment\_Status;

};

}

#endif

**7) Student**

#ifndef STUDENT\_H

#define STUDENT\_H

namespace Domain\_Class {

class Student : Domain\_Class::User {

private:

int Student\_ID;

int First\_Name;

int Last\_Name;

};

}

#endif

**8) User**

#ifndef USER\_H

#define USER\_H

namespace Domain\_Class {

class User {

public:

int UserId;

string Department;

string FirstName;

private:

string LastName;

intt PlateNumber;

public:

void makePayment();

void getReceipt();

void requestCarLocation();

void followDestination();

};

}

#endif

**User C++ Code-**

#include "..\..\Users\chaus\Downloads\User.h"

void Domain\_Class::User::makePayment() {

// TODO - implement User::makePayment

throw "Not yet implemented";

}

void Domain\_Class::User::getReceipt() {

// TODO - implement User::getReceipt

throw "Not yet implemented";

}

void Domain\_Class::User::requestCarLocation() {

// TODO - implement User::requestCarLocation

throw "Not yet implemented";

}

void Domain\_Class::User::followDestination() {

// TODO - implement User::followDestination

throw "Not yet implemented";

}

**Lists Technology tools for Project Development to date:**

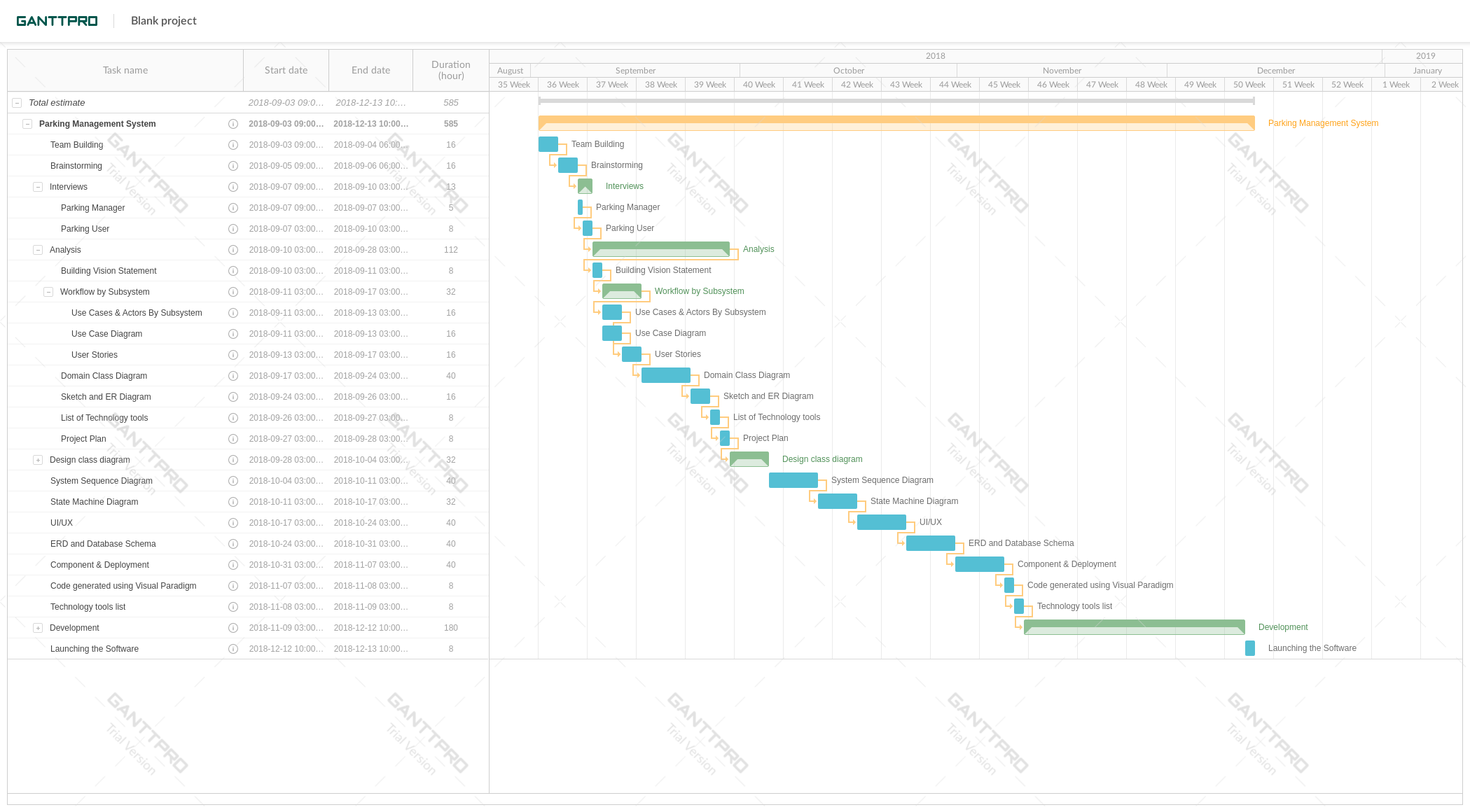
Technologies used to develop this system are:

Software

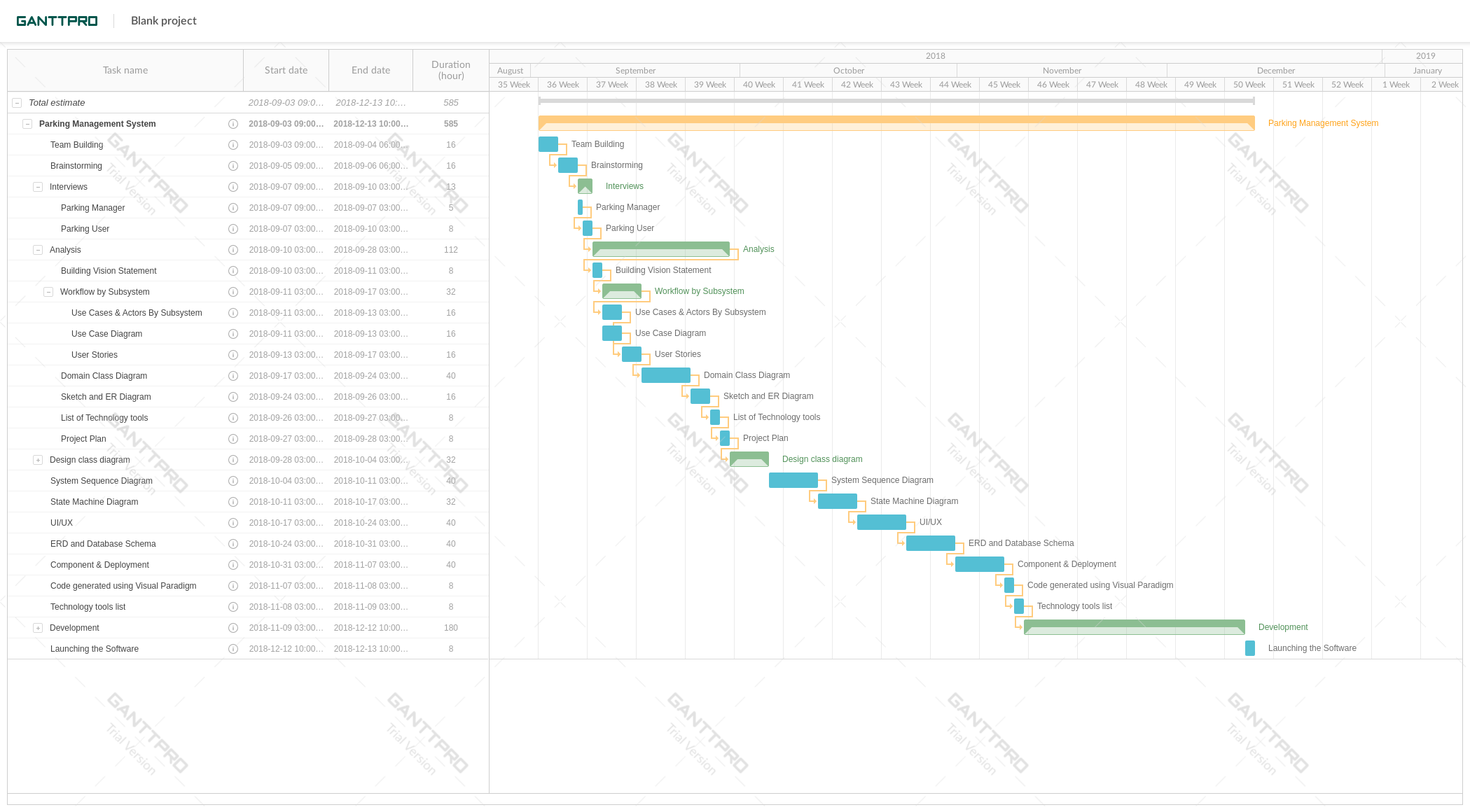
* Visual Paradigm
* Microsoft Project
* LibreProject
* Office 2016
* Photoshop
* GanttPro
* Oracle SQLDeveloper

**Final project Plan:**

In the following project plan, which was made in GanttPro, we show the different activities and times required to accomplish the main objective which is to provide the facility to the students and employees to have all the information related to the College Parking.

****

Gantt chart - activities, duration, start, and finish date



Gantt chart - duration of activities