Software Design Documentation

Version 1.0

June 1, 2020

Elementary School Pickup Services

by

RoNeil Boyce

Liam Curtis

Elvis Mack

CSCI 5530 Software Engineering

**Abstract**

Elementary schools are tasked every day with the duties involved in safely and practically boarding students to their parent’s vehicles. While school staff may know the parents of students personally, introducing software to this obstacle could greatly improve the safety and efficiency of transporting students home. With a check-in application involved, the staff can identify the parents of students via their vehicle’s make, color, and model. Corresponding staff can then release the student from the classroom to board their intended, confirmed vehicle.

The security of this data is crucial, as it pertains to the safety and transportation of children. Therefore, the system’s architecture should be fixed, offering only limited flexibility from the administrator’s end to add or delete users. In recent years, schools have begun integrating mobile technologies and social media outlets to connect parents with their child’s academic activity. However, social media offers limited data protection, making a unique application with its own security entirely superior. With tailored-security, staff should confidently be able to process the students to their transports, parents will experience improved wait-times for picking up their children, and student’s will likely have more reliable, prompt rides home. This document defines all software design plans of the Elementary School Pickup Services System (ESPSS).

**Table of Contents**

[**1.0 Introduction**](#_d9bl9dlbdaee) **4**

[1.1 Purpose of System](#_h410k6jzhiwt) 4

[1.2 Functional and Non-Functional Requirements](#_m8c6q2e26j5t) 5

[1.3 Glossary](#_3bzeil60eb3p) 5

[1.4 Overview](#_r3x74kamzpn2) 6

[**2.0 Proposed Software Architecture**](#_93uof28xe5n8) **7**

[2.1 Overview](#_903a607026yi) 7

[2.2 Three Tier Architecture](#_si6r92oub622) 7

[2.3 Event Driven Architecture](#_2tdwyvy94kb0) 8

[**3.0 Object Design**](#_hk4qnaalkuvt) **10**

[3.1 Class Diagram](#_tkpi61qhhhsg) 10

[3.2 Sequence Diagram](#_ov0h6rqzk5a) 11

[3.3 State Diagram](#_td711py9nh69) 14

[3.4 Package Diagram](#_rqtea8pv4kfx) 17

[3.5 ER Diagram](#_rk08f2jh7url) 18

[**Chapter 4.0 Glossary**](#_kg0pzd7iccz1) **19**

[**Chapter 5.0 Appendix**](#_ja36jjln7lgg) **23**

[5.1 Appendix A - Use Case Diagram](#_z98iqhdzxc3k) 23

[5.2 Appendix B - Use Cases](#_qzy8poczztlr) 23

[5.3 Appendix C - Detailed Class Diagram](#_igw3rkox3do2) 34

[5.4 Appendix D - Class Interfaces](#_1jgxoytvyxmp) 35

**Elementary School Pickup Services System**

# **1.0 Introduction**

In recent years, primary schools have begun incorporating technologies and online communication tools to help support the school staff with administrative duties. One of said duties is to itinerate the student’s transportation at the end of the day. The existing procedure involves an exhausting routine of verbal confirmations. To avoid this, an elementary school has requested an application designed to efficiently and safely shuttle students to their parent’s vehicle. The software utilized in the operation must be properly considered and understood. The following chapter shall describe the new procedure involving the specified application for increased security and speed. It will provide detailed descriptions of the system’s features and uses. This chapter is intended as an overview to the system’s entire software.

In chapter two, the proposed software architecture will be presented through diagrams to elaborate on the different interfaces of the program. Chapter three will provide thorough definitions of the programmable objects associated with the software. The features and functionality of these objects will also be displayed via charts and diagrams for better conceptual visualization of the code. Finally, a glossary containing terms and their corresponding definitions will be succeeded by an Appendix chapter. In the Appendix section, use cases, classes, and packages are defined in full detail.

## **1.1 Purpose of System**

The Elementary School Pickup Services System was planned to meet the needs of a school staff while they release students to their parent’s vehicle at the end of the school day. The systems security is imperative as it will contain data associated with the children in attendance and their transportation. This program is fundamentally crucial to the client as their existing system is exhausting and involves verbal confirmation for each and every student. With the introduction of our software, the staff will be able to non-verbally confirm student’s transportation in real time. This alleviates a significant workload from the staff, who can now better focus on the safety of the students. This includes financial relief, as it will require less employees to stay late assisting with student pickup. The system’s high security will benefit the school in that the parent’s will feel more confident in a school with additional security. In short, the ESPS system will drastically simplify the coordination of student pickups while immeasurably increasing security.

## **1.2 Functional and Non-Functional Requirements**

The ESPS system shall be capable of displaying data stored from a back-end database. The UI will allow users to interact with the program at different privilege levels. Parent’s will be able to view and submit data to their personal account. Staff will be able to view and submit data to confirm and release Students at their appropriate pickup time. Administrators shall have access to a master-user account with high-level communication and security privileges.

The ESPSS will prompt the Parent to enter their account and vehicle information.

The system shall display a pickup line for the staff to organize and view. The staff will use this list for confirming student’s vehicles. Associated with this, a counter will be displayed to the unique staff’s classroom depicting how many/which students still need rides.

The system shall grant Admins the ability to add, delete, and edit any accounts at any time, however data will be stored permanently in the database.

The system should only be accessible on-campus via the wireless internet provided by the school.

The system shall operate all year long for the administrators to make changes at any time.

## **1.3 Glossary**

|  |  |  |
| --- | --- | --- |
| **Term** | **Abbreviation** | **Description** |
| Elementary School Pickup Services System | ESPSS | Title of the given software |
| Elementary School Pickup Services | ESPS | Project name |
| Parent |  | User that provides data to Admin/Staff. |
| Pickup Time | PT | Time of student departure and event-based program. |
| Software Requirement Specification | SRS | Document to describe and visualize the entire functionality of a proposed system. |
| Staff |  | User with access to the Parent/Student data. Medium-level privileges granted. |
| User |  | Admin, Staff, or Parent |

## **1.4 Overview**

In the following chapters, the ESPSS will be explained step-by-step from a design analysis perspective. Chapter 2 shall provide detailed information corresponding to the software architecture and architecture types: three-tier architecture and message driven architecture. Chapter 3 contains the Object Design protocols including every implementable class and attribute. It displays the class diagrams and their related relations and sequences. Chapter 4 shall be an elaborate Glossary containing all terms that may be deemed to obscure for the client. Finally, chapter 5 will be the Appendix, including the majority of the use case diagrams, code samples, and a log of contributions and meeting data. The purpose of this manual is to fully elaborate the capabilities and intentions of the ESPSS software to the client.

# **2.0 Proposed Software Architecture**

The software architecture should be the cornerstone of any development process. It will present the development team with insight into the feasibility, time constraints, and various challenges that may hinder the development. The architecture is also the main design component that will communicate how the system will interact within itself, its user(s), development team, and the client(s). Because of this, the architecture must be appropriately designed and documented. Failure to accomplish this could result in hindrances not planned for, with fear of even system failure later in the development cycle.

## **2.1 Overview**

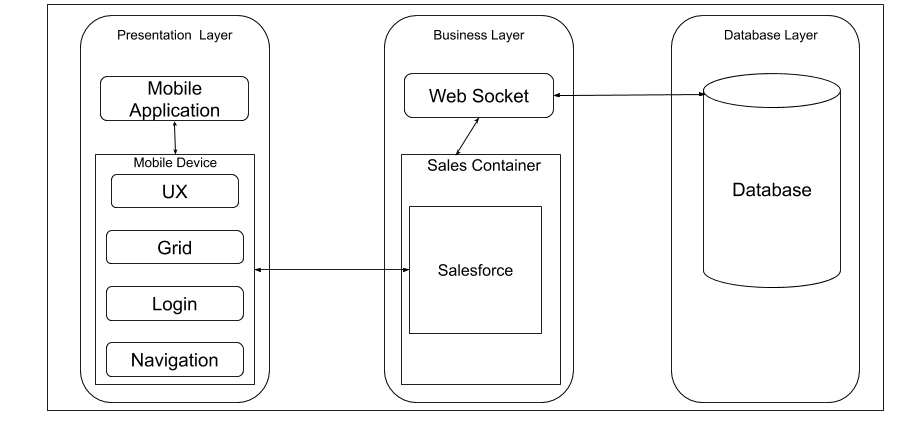
The Elementary School Pickup Services can be broken down into five areas of functionality which have been made into various packages. These packages are User, Data, Security, Exception, and UI. The way these packages interact with themselves and each other follow the three tier architecture. The data package will mainly communicate with the database which will compile the data that it gains from the User packages. Initially, the user packages will obtain no type of data than required information used for access, but the user will have the ability to input more data into their own package. The UI package will obtain and relay any additional data that is inputted by the user packages as well as formatting and displaying that information to the user. Finally, the Security and Exception packages will ensure that problems in the data manipulation and computation are handled with care and users can only have privileges fitting their respective role.

## **2.2 Three Tier Architecture**

The Elementary School Pickup Services is being built using a 3-tier architecture, consisting of a presentation layer, a business layer, and a database layer. The presentation layer has the responsibility of presenting the look and design for the user to handle when using the application. The business layer has the responsibility of executing the algorithms like pick up time and alerts that will be transferred between the presentation layer and the database layer. The database layer will store all the data used in the system which includes: users, vehicles, roles and status.

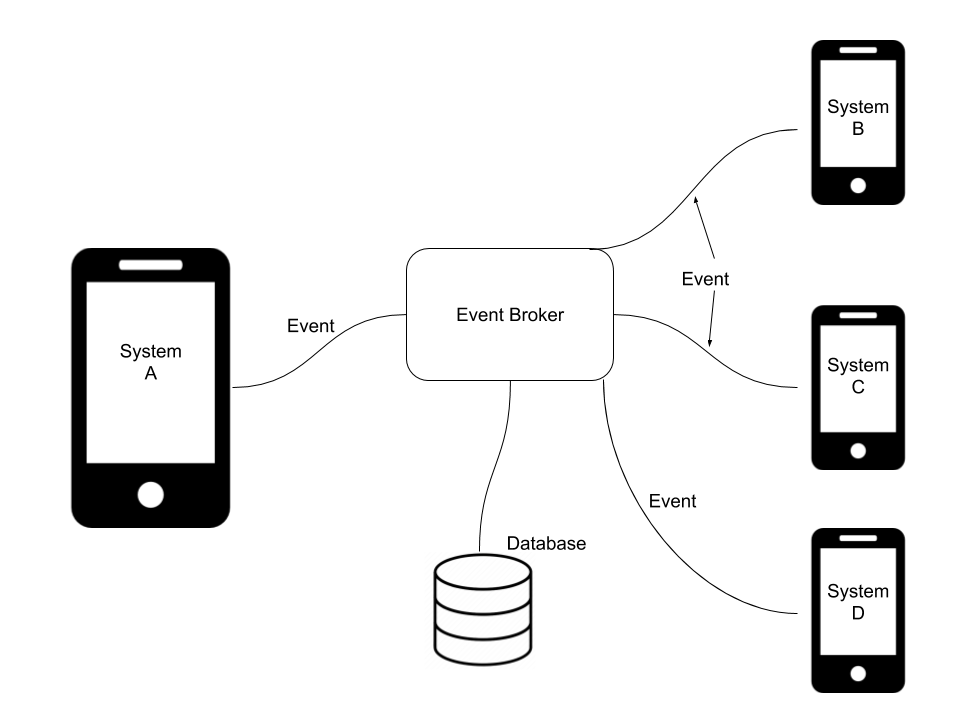


The Elementary School Pickup Services will be a Javascript-based phone application which will run on a mobile device. The implementation will follow Alpha Software application standards. The presentation layer will consist of UX,Grid,Login and Navigation components. The user will interact with these components within a mobile application. The business layer will be made up of a web socket server and (Salesforce) (please look at this). The Salesforce are primarily responsible for using business logic on the system data, while the web socket is directly persistent with the database layer that will consist of a relational database.



## **2.3 Event Driven Architecture**

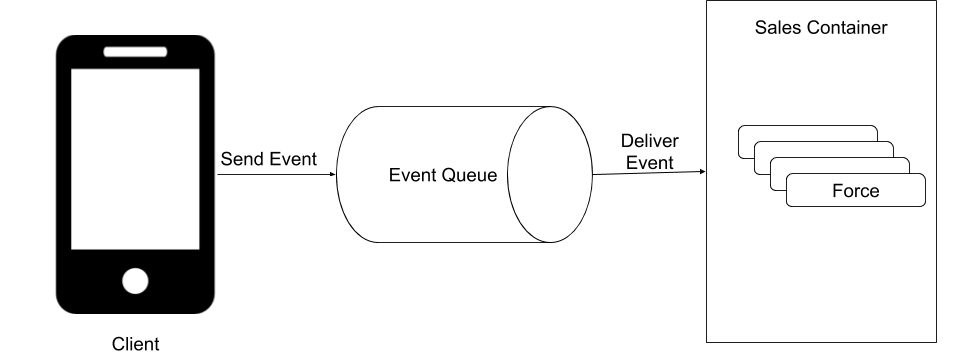
The Elementary School Pickup Services will be built using event driven architecture. The service will be composed of individual components which will communicate between each other by using various changes in states to reflect changes. Event Driven Architecture provides loose coupling and high cohesion to maintain a distributed environment. An event will be either an update, which shows the database will receive an update of new information, or an alert, in which a message will tell the receiver that something has happened.



The Elementary School Pickup Services will be implemented using Javascript Object Notation (JSON) to communicate the process events asynchronously. Updates will be sent by most components within the system, including clients and other forces. Features of the event driven forces include:

* Not maintaining states per client
* Concurrent event updates
* Updates from multiple clients can be processed by a single force

All event driven forces will involve connections to the database, which inherently will be a product of maintaining some state for updates. Event driven forces and event driven architecture importantly allows for asynchronous communication between components, namely server-side components and the database.

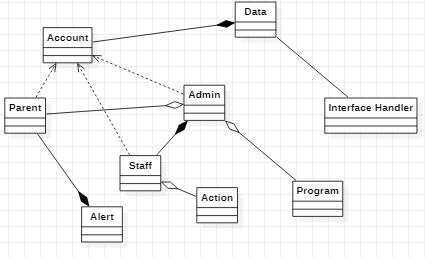


# 

# **3.0 Object Design**

The ESPSS is composed of a three-tier architecture built using a event driven architecture. The Interface Handler class will detect changes made by the user to the system and operate based off of events in the Data class. The following figures will depict the systems class UML, sequence diagram, state diagrams for each user, package diagram, and ER diagram.

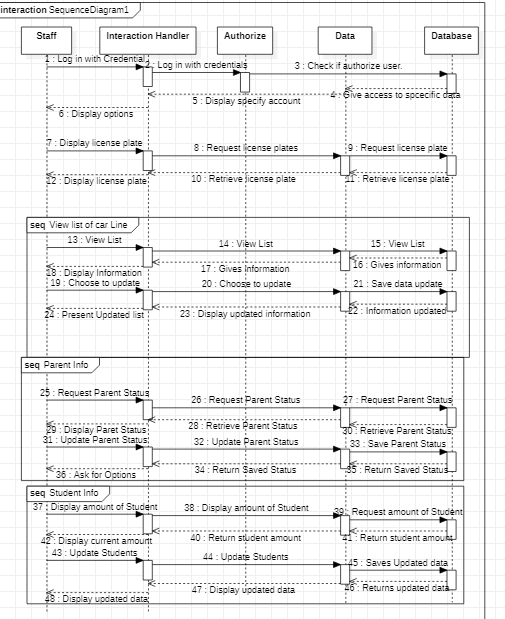
## **3.1 Class Diagram**



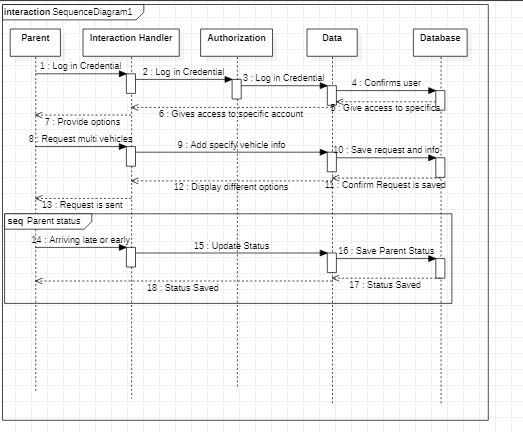
The diagram above displays the classes of ESPSS in UML format. The interface consists of three unique instances of the Account class: Parent, Staff, and Admin. The Data class manages events involving data stored in the database. The interface handler will check the system for updates to trigger events in the Data class. The Action and Program classes contain methods for users to update the system via input. The Alert class corresponds to Parent user’s alert messages distributed by the Admin.

## **3.2 Sequence Diagram**

The process starts when the admin enters their credential to the interface handler. The interfaceHandler will carry the information being delivered to the authorize for security purposes. The authorization will send the information to the data. Once the data has the information then the database will either confirm whether or not this is the right account information. Once the account has been confirmed, the Admin will have the opportunity to choose from one of three options: Add programs, Delete programs, and Update Alerts. The process for all of three options are primarily the same. First the Admin chooses one of options, which will send his request to the database from the interface event Handler. Once the request is logged the database will return the requested information. After the Admin is finished with their task they will log out from the system.

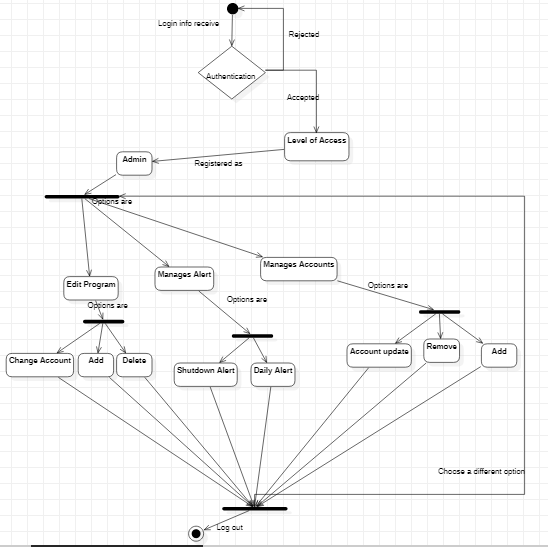


First the Staff begins by logging in through the controller. The controller will send the information to the Authenticator which will pass through a verification check. Once the database confirms the account then the interface will present the Staff three options: Parent info,View /list of car line. Within the Parent info the staff will request the parent status from the database. Once the Staff receives the Parent status, the Staff is able to update the information which will be sent back to the database. This process is the same for Student info and for the View Car Ride Line.

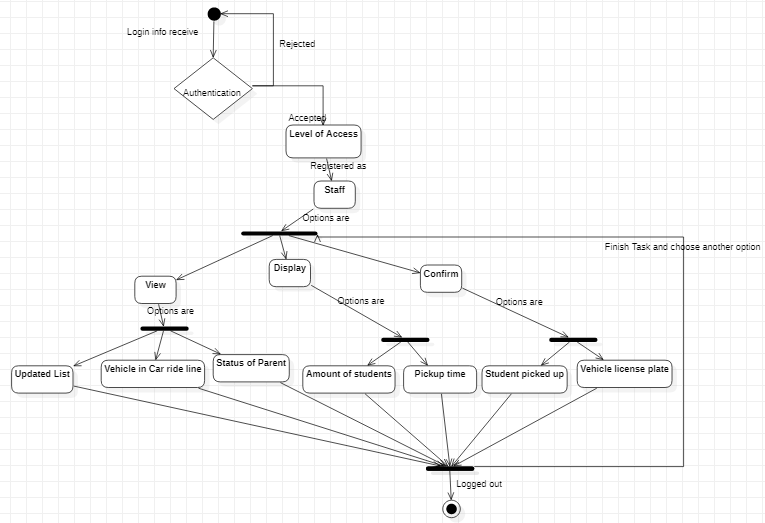


First the Parent interacts with the Handler by entering their credentials. The information they typed in will be transferred to Authorization which will then be transferred to the Data. The data will communicate to pull the specific account information and transfer the information to the Authorization form, which will give the Parent access to only the parental level. Within the parental level, the Parent will make a choice of accepting the Status update of their arrival and requesting to add multiple vehicles.

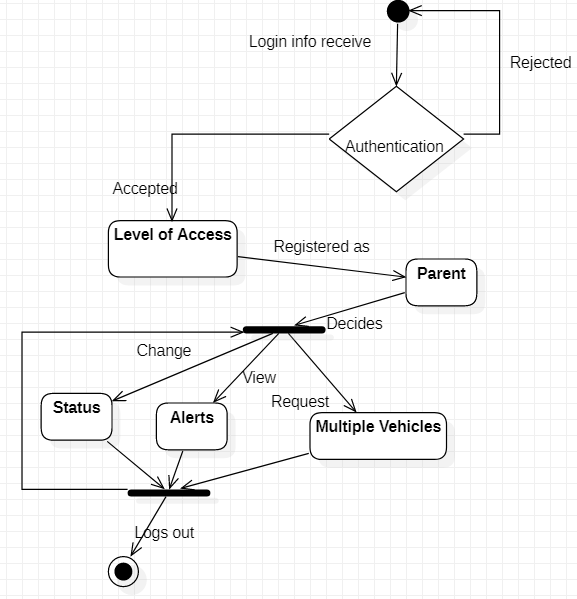
## **3.3 State Diagram**



The state diagram shows the process of an Admin completing a certain task. The first state requires that the Admin log in with their credentials and will be registered as an Admin. Once registered as an Admin you will have three options to interact with different Admin functionalities. If Admin selects Edit Program then the Admin will be given another set of options. Once the Admin completes a task you will have the option to return back to the first set of options or the Admin will be logged out.

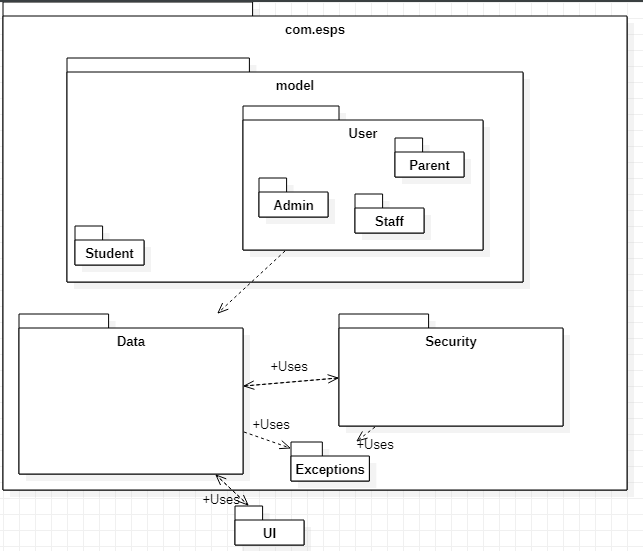


The state diagram for Staff starts at the authenticating login information. If the login information is wrong then the Staff member must reinput their information. If the login information is right then the Staff will receive the staff level of access. Within the staff level of access you will have three options: View, Display, and Confirm. If the Staff member chooses one of the options then they will have another set of options to choose from. After completing the given path the Staff member will be given a chance to return to the main menu or log out.



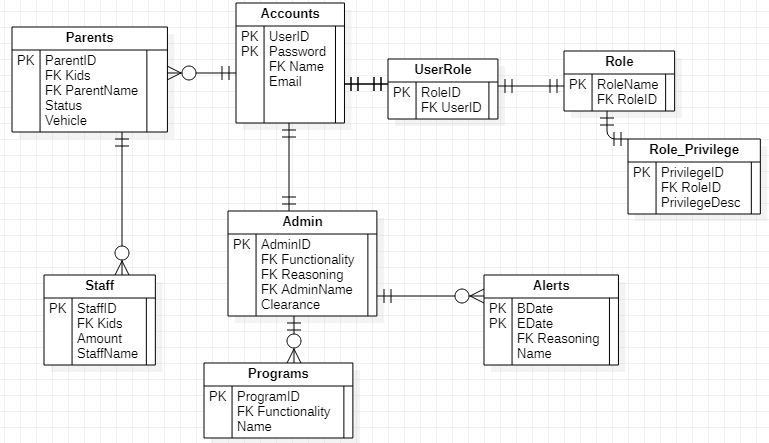
The State diagram begins with the Parent entering their credentials. If the Parent credentials is incorrect then the Parent must re-input their information. If the credential is correct then the Parent will gain access to parental level information. The parent will have three options: to change Status, to view Alerts, and to Request Vehicles. If the Parent enters one of the options then the Parent must complete the following path to be given the opportunity to either return to the first fork or to log out.

## **3.4 Package Diagram**



The ESPSS will be broken down into nine distinct packages: com.esps, model, student, user, admin, data, security, exception, and UI. The com.esps will be the primary package consuming all implementation (excluding the UI design). The model package will contain our student and user packages, with User containing the admin package inside it. The data package will be involved with DB interaction, storage, and computation. The security aspect will, in turn, hold all of the security details. The exception package relieves all packages from their inevitable handling issues. Finally, the UI shall be the only package external of com.esps. It will contain the interface implementation for user experience.

## **3.5 ER Diagram**



The Elementary School Car Ride Pickup Line will contain nine tables. Each table will contain a PK for Primary Key and FK for Foreign Key to connect each table from one another. The Accounts table has a one to one relationship with UserRole and Admin, but Accounts have a one to many relationship with Parents. The reason for the One to Many relationship is so that we won’t have duplicate accounts.The Foreign Key that connects Parents to Account would be the name. Parents have a one to many relationship with the Staff as well. For the Car Ride Pickup Line there will be two staff that communicate with the Parent by using the Foreign Key kids. Since the inside Staff must send the kid to the outside to be sent home with their Parent, this makes the reasoning behind the one to many relationship between Parents and Staff. Admin has a one to many relationship between Programs and Alerts, since Admin can initially add as many programs as needed, and assign as many alerts as needed. These Alerts have a beginning date and an end date which means there can be multiple dates that need an Alert. The UserRole, Role and Role\_Privilege will all have one to one relationships since only one user can have one role and only one Role has only one Privilege.

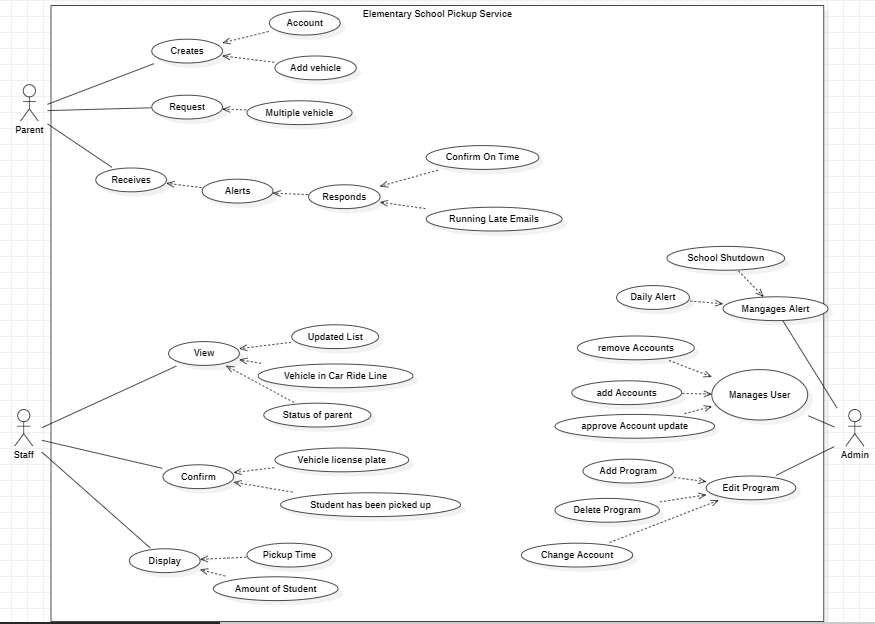
# Chapter 4.0 Glossary

|  |  |  |
| --- | --- | --- |
| **Term** | **Abbreviation** | **Description** |
| Elementary School Pickup Services System | ESPSS | Title of the given software |
| Elementary School Pickup Services | ESPS | Project name |
| Parent |  | User that provides data to Admin/Staff. |
| Pickup Time | PT | Time of student departure and event-based program. |
| Software Requirement Specification | SRS | Document to describe and visualize the entire functionality of a proposed system. |
| Staff |  | User with access to the Parent/Student data. Medium-level privileges granted. |
| User |  | Admin, Staff, or Parent |
| Administrator | Admin | The boss that has the highest privilege. Must be the principal or whoever owns the application. |
| Application | App | A program that can be used on a mobile device and/or the computer. |
| Database | DB | Collection of all information monitored by the system. |
| Functional Requirement | FR | Areas of functionality that the system must have. |
| Model |  | An abstraction of a complex entity or process. |
| Nonfunctional Requirement | NFR | Constraints on the system. |
| Password |  | A secret phrase or word only known by the person who created it and the person who manages the database. |
| Software Requirements Specification | SRS | A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. |
| Stakeholder |  | Any person with an interest in the project who is not a developer. |
| Unified Modeling Language | UML | A general purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. |
| User |  | Admin, Parent, Staff. Those who use the system. |
| Role |  | The responsibility of the given User. |
| Interface Event Handler |  | The connection between User interaction to the system. |
| Authentication |  | The correct identification of a user. |
| Feature |  | Functionality within the software that can be added. |
| Data |  | Information that is being collected and stored within the database. |
| Limitation |  | A defect that stop the creation of a feature or an understanding of the design. |
| Problem |  | A situation preventing something from being achieved. |
| Event Driven Architecture |  | Provides loose coupling and high cohesion to maintain a distributed environment. |
| Three Tier Architecture |  | Consisting of a presentation layer, a business layer, and a database layer. |
| Presentation Layer |  | Responsible for presenting the look and design for the user to handle when using the application. |
| Business Layer |  | The responsibility of executing the algorithms and connecting the presentation and Database layer. |
| Database Layer |  | The database layer will store all the data used in the system. |
| Coupling |  | Strength of connection between modules. |
| Cohesion |  | The glue that keeps modules together. |

# Chapter 5.0 Appendix

## 5.1 Appendix A - Use Case Diagram

This section contains use case diagrams that depict the relationships visually between various use cases.



## 5.2 Appendix B - Use Cases

**Use Case ID:** UC1 – Add or Remove Users From Elementary School Pickup Service

**Scenario:**

**Actor:** Admin, System

**Pre-conditions:**

Admin already has accessed the system and has been authenticated as an Admin.

**Description:**

1. The Admin chooses to assign the user’s role

2. The system displays the screen to edit roles and privileges.

3. The Admin selects a user and assigned a role and privileges.

4. The system asks the Admin to confirm role and privilege changes.

5. The Admin confirms role and privilege changes.

**Post-conditions:** The specified users are added or removed from their respective Elementary School Pickup Service.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None.

**Criticality:** High. Usefulness of application is dependent on the ability to execute instance models.

**Risk:** Low

**Constraints:**

1. The user interface should explicitly present the user only the available users that could be added to or removed from the Elementary School Pickup Service.

**Use Case ID:** UC2 – Add or Remove Programs From Elementary School Pickup Service

**Scenario:**

**Actor:** Admin, System

**Pre-conditions:**

Admin already has accessed the system and has been authenticated as an Admin.

**Description:**

1. The Admin chooses to add a program.

2. The system displays the screen to assign a functionality.

3. The Admin selects a functionality of the program.

4. The system asks the Admin to confirm program.

5. The Admin confirms program.

**Post-conditions:** The specified programs are added or removed from their respective Elementary School Pickup Service.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None.

**Criticality:** High. Usefulness of application is dependent on the ability to execute instance models.

**Risk:** Low

**Constraints:**

1. The user interface should explicitly present the user only the available users that could be added to or removed from the Elementary School Pickup Service.

**Use Case ID:** UC3 – Manages Alert for Elementary School Pickup Service.

**Scenario:**

**Actor:** Admin, System

**Pre-conditions:**

Admin already has accessed the system and has been authenticated as a Admin.

**Description:**

1. The Admin chooses when to declare an alert, and the reasoning.

2. The system displays the date, time, and reasoning.

3. The Admin selects the time and date and the reasoning for the alert.

4. The system asks the Admin to confirm the alert.

5. The Admin confirms the alert.

**Post-conditions:** All users will receive an alert from their Elementary School Pickup Service.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None.

**Criticality:** High. Usefulness of application is dependent on the ability to execute instance models.

**Risk:** Low

**Constraints:**

1. The user interface should display only the upcoming alert within ten seconds once issued by the Admin..

**Use Case ID:** UC4 – View/Edit Car Rider Line

**Scenario:**

**Actor:** Staff, System

**Pre-conditions:**

Staff has already accessed the system and has been authenticated as an Staff.

**Description:**

1. The Staff can open “Pick-up Line” page to view the current list of cars waiting.

2. The system displays a Confirm/ Delete option when a Staff selects a vehicle from the list.

3. The system updates by changing the status/color of confirmed cars from red (unconfirmed) to green (confirmed).

**Post-conditions:** The system updates the list in real time allowing for other Staff to input data accordingly.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** UC5 - Edit/Display remaining Student

**Criticality:** Medium. Staff should be able to edit the list in real time, while the system will still function if an error occurs.

**Risk:** Low

**Constraints:**

1. The system will only show the staff the vehicles remaining in the pickup line.

**Use Case ID:** UC5- Edit/Display remaining Student

**Scenario:**

**Actor:** Staff, System

**Pre-conditions:**

Staff has already accessed the system and has been authenticated as an Staff. Students who have been marked “Absent” have been updated in the system by Admin.

**Description:**

1. The Staff can open “My Classroom” page to see the list of students linked to their classroom that have not been picked up yet.

2. The staff can select a Student from the list, highlighting their information.

3. The Staff can select Release, to release the corresponding student to their vehicle after that vehicle has been confirmed by the other staff.

4. A counter displaying the number of students remaining will count down as students are released.

**Post-conditions:** The system will update the Admin whenever the list is complete or if there are outlying students still awaiting rides.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None

**Criticality:** High, as this is one of the primary features of the application.

**Risk:** Low

**Constraints:**

1. The system will only show the Staff information about the students in their own classrooms.

**Use Case ID:** UC6- Parent Create Account

**Scenario:**

**Actor:** Parent, System

**Pre-conditions:**

Admin already has accessed the system and has been authenticated as an Admin.

**Description:**

1. The Parent can open a “Create Account” Window

2. The System will prompt the user for specific information for creating an account

3. The Parent will be prompted to submit their information (Name, Email, Password, Number of Children, Children’s names, Vehicle Description)

4. The System will ask if the Parent is sure that the information is correct.

5. The Parent can click “Submit”, sending the information to Admin for verification.

**Post-conditions:** The system will update the Admin whenever the submission is complete.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None

**Criticality:** Low, the functionality of the app relies on some Parent data.

**Risk:** Low

**Constraints:** None

**Use Case ID:** UC7- Request Multiple Vehicles

**Scenario:**

**Actor:** Parent, System

**Pre-conditions:**

Parent has already accessed the system and has been authenticated as a Parent.

**Description:**

1. The Parent can open a “Add Vehicle” Window

2. The System presents the Add vehicle menu and request the specific information

3. The Parent will be prompted to submit their information (Name, Email, Password, Vehicle Description)

4. The System will verify that this is the correct information

5. The Parent will click “Yes” or “No” to verify that they did not enter any mistakes.

6. The System will send the request to the Admin for approval

**Post-conditions:** The system will update the Admin whenever the submission is complete.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None

**Criticality:** Low

**Risk:** Low

**Constraints:**

1. The system will recognize a maximum of 3 vehicles per account.

**Use Case ID:** UC8- Parent receives Alert

**Scenario:**

**Actor:** Parent, System

**Pre-conditions:**

Parent has already accessed the system and has been authenticated as a Parent.

**Description:**

1. The Parent will receive a scheduled alert 30 minutes before PT to notify them of their appointment.

2. The system will send a message to the Parent verifying their status.

3. The Parent’s receive the alert and confirm if they are “Running Late” or “Confirmed” for on time pickup.

4. The system will update the Parent’s status.

**Post-conditions:** The system will update the Staff whenever the Parent responds with their status.

**Alternative Courses of Action:**

1. The user may choose to close the application at any time.

**Concurrent Uses:** None.

**Related Use Cases:** None

**Criticality:** Low

**Risk:** Low

**Constraints:**

1. The system will only send these alerts to Parents of Students who have not been labeled “Absent”.

## 

## 5.3 Appendix C - Detailed Class Diagram

This section contains detailed class diagrams that depict the relationships visually between various classes in the corresponding package.

## 

## **5.4 Appendix D - Class Interfaces**

This section is intended to describe all of the class interfaces involved in ESPSS.

**5.4.1 User Package**

public interface User {

String getEmail();

String getPassword();

String getRole();

String getFirstName();

String getLastName();

List <String> getPrivileges();

void setEmail(String Email);

void setPassword(String Password);

void setRole(String Role);

void setFirstName(String firstName);

void setLastName(String lastName);

void setPrivileges(List <String> privileges);

}

**5.4.2 Admin Package**

public interface Admin extends User {

void addAccount(Account account);

void deleteAccount(Account account);

void accountApprove(Account account);

void sendAlert(String alert);

}

**5.4.3 Staff Package**

public interface Staff extends User{

int getRemainingStudent();

void viewCarLine();

bool confirmPickup(String vehicleInfo);

void displayStudentData(Student student);

}

**5.4.4 Parent Package**

public interface Parent extends User{

void requestVehicleApprove(String vehicleInfo);

bool emailConfirm(String Email);

void viewPersonalData();

}

**5.4.5 Student Package**

public interface Student extends Parent{

void viewStudentData();

}

**5.4.6 Data Package**

public interface Data extends ESPSException {

void authorizeUser(User user, String authorize);

void manageUser(User user);

void manageStudent(Student student);

void manageVehicle(Parent parent);

void manageStaff(Staff staff);

}

**5.4.7 Security Package**

public interface Authorization extends ESPSException{

String getUser(User user);

String getPassword(String password);

void authorizeUser(User user, String authorization);

void authorizeNetwork(boolean WifiNetwork);

void setUserInfo(User user, String password);

String getUserInfo(User user);

}

**5.4.8 Exceptions Package**

public interface ESPSException{

void InvalidCredentialsException();

void AlreadyInUseException();

void DatabaseAccessException();

void PrivilegesMismatchException();

}

**5.4.9 UI Package**

public interface InterfaceHandler {

void createGraph();

}