Software Requirements Specification

for

Sure-Park System

Version 3.2 approved

Prepared by Team C

July 29th, 2016

Team members: **Dongho Lee (leader), Sunghoon Byun, Daesoon Kim, Woojin Han, Kyeongseok Yang, Dongjae Kim, Myoungki Hong**

Table of Contents

Table of Figures

Revision History

Person in charge	Date	Reason For Changes	Version
Dongho Lee	June 16th	This is the document which describes our planned work. It contains the initial architectural drivers and system overview in brief, to make mentor understand our work easily.	1.0
Sunghoon Byun	June 23th	We refined the initial version of this document. And we are ready to submit the document to our mentor.	1.1
Dongho Lee	June 28th	We modified the document according to mentor's feedback. However, it will need to be changed more in detail.	1.2
Dongho Lee	July 4th	After meeting with mentor, we thought we must change the document entirely. Therefore, we are going to consider the project plan, Quality Attribute scenario and test plan intensively.	1.3
Woojin Han	July 5th	System context figure was edited, and also we added descriptions. Physical, static and dynamic views are included, but it needs to be more specified.	1.4
Sunghoon Byun	July 6th	Our team tailored Open Unified Process (openUP) to meet our limited time. It is based on the iterative and incremental process, and we chose this process to be flexible with the formal process.	1.5
Dongho Lee	July 7th	This document was integrated for the purpose of meeting with mentor. This document is the midterm version.	2.0
Dongho Lee	July 9th	The context, Use case diagram and three perspective views were modified in this document. And Use case #02 was added in Use case description.	2.1
Sunghoon Byun	July 11th	Use cases are refined, and risk descriptions are modified.	2.2
Dongho Lee	July 13th	Quality attributes are refined, and Quality attribute scenarios are modified.	2.3
Dongho Lee	July 16th	Architectural & Detail designs were decided, and we modified the several views according to comments of mentor.	2.4
Sunghoon Byun	July 18th	The grammatical errors and minor inconsistency of description are being rectified.	3.0
Sunghoon Byun	July 22th	We added testing results and lessons learned.	3.1
Sunghoon Byun	July 28th	We concluded that our document is done 100%.	3.2

Terminology

Terms	Description
ADS	Architecture Drivers Specification
Car driver	Car driver reserve parking slot and cancel the reservation.
Parking slot	Available place that is reserved by car driver.
Parking facility	This is a parking facility that includes 4 parking slots.

1. Introduction

1.1 Project overview

We will design a system that enables car drivers to find reserve parking slots quickly and efficiently. Also, we will give our system functionalities of monitoring and parking facilities operations. The objective of this project is to establish the efficient and valuable parking system that supports the following functionalities:

- The system should be able to easily check available parking slots in the parking facility and reserve them.
- The system should efficiently utilize the slot in their parking facilities, thereby increasing profits.
- The system should efficiently utilize personnel and reduce the number of people required to operate any given garage, thereby reducing operating costs.

Our project will be based on an "Internet of Things" (IoT) context. We will build a smart parking facility management system that includes the virtual parking facility using Arduino 2560 processor, the client system for reservation, the administrative system for parking facility, and monitoring system for parking slots.

1.2 Problem statement

Most of drivers are prone to have problems with parking their own vehicles in parking facilities. Actually, they don't know where a vacant slot is. Also, they tend to wander on the facility to find a vacant slot. The existing parking facilities need several additional employees to monitor the parking facilities and manage the irregular situation. It takes a lot of time for employees to walk around, monitor the facility, and notice vehicles that are parked incorrectly.

In other words, the drivers who are trying to find a slot to park their vehicles can be frustrated. Most of the parking facilities are not so smart that they can provide the information of current parking facility enough for the stakeholders (car drivers, a parking attendant, and the owner). In some cases, there is no way of knowing whether there are available parking slots or not until we enter the parking facilities. If car drivers can know status of parking facility before entering, they can reduce a waste of time. The followings are the problems that we are planning to resolve:

1. The car drivers waste their time in finding the available parking slots in the parking facilities.

2. The owner of the parking facility cannot easily figure out the parking facility information.

3. The number of many employees who are responsible for monitoring the parking facilities cause high expense.

To improve these efficiencies, and reduce the cost of management, we would like to develop a smart parking system called Sure-Park System.

1.3 Scope & objectives

The smart parking system that we build will include the functionality of monitoring & management of the parking facility, the reservation of parking slot, and facility's statistical data analysis for the owner. The hardware of the facility will be constructed with Arduino 2560 processor and controlled by management system. Our monitoring and management system is based on Java and JavaScript. The parking facility sensors, actuators, and LEDs are shown below:

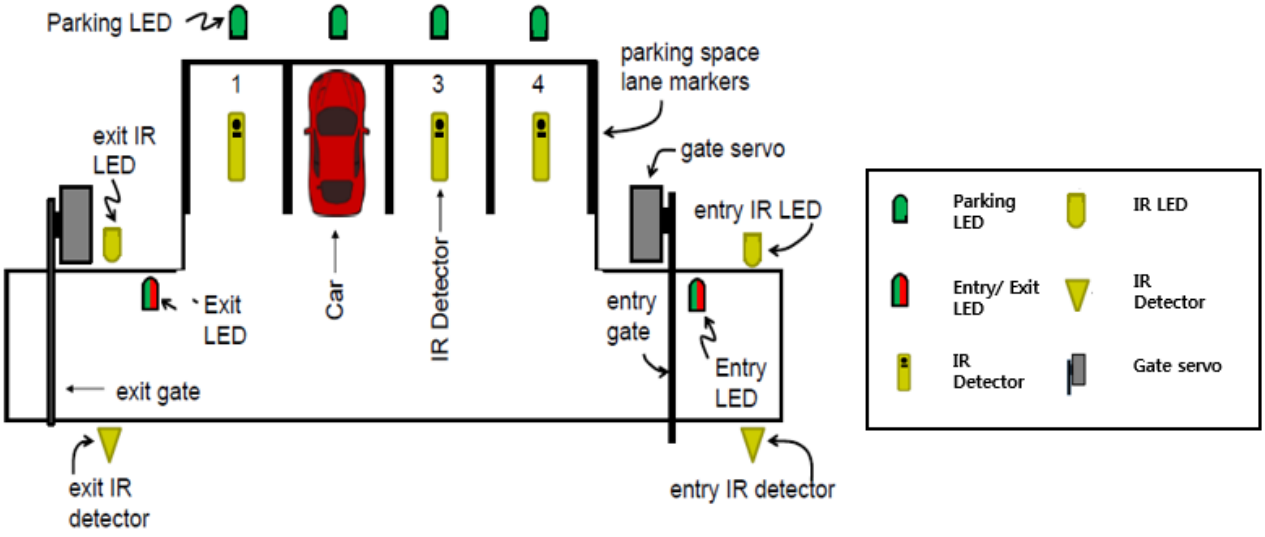

Figure 1. Blueprint of the Parking Facility

The parking facility sensors, actuators, and LEDs are monitored and controlled by an Arduino 2560 processors. This system is enabled on a Wi-Fi router, allowing Arduino to communicate with other devices that are part of the system. The basic configuration of the system is shown below:

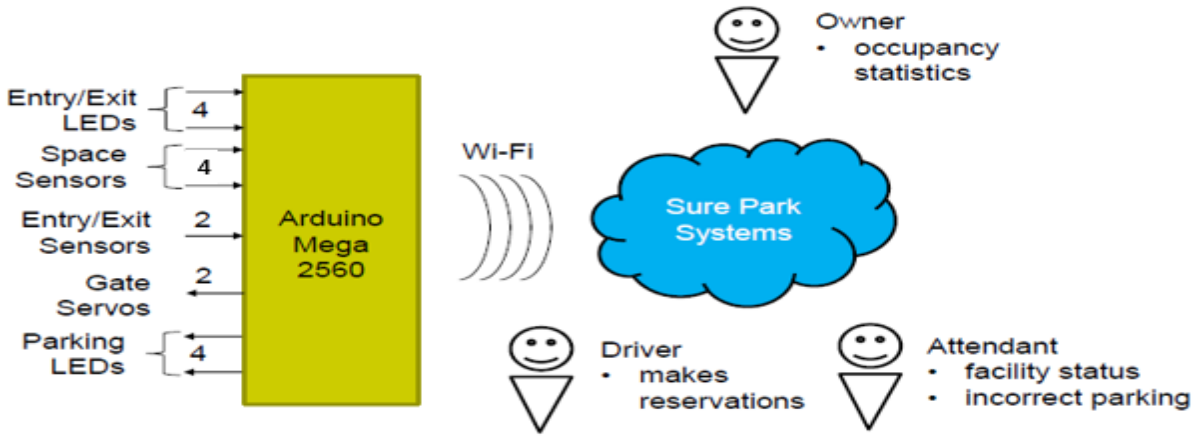

Figure 2. Configuration of the Parking System

The Arduino 2560 constitutes the virtual parking facility. In this situation, we have some technical constraints. The technical constraints of Arduino which will be used in our system are like these; four Entry/Exit LEDs, four slot sensors, Entry/Exit Sensors, and two gate servo. In other words, the number of available modules in an Arduino is limited. Also, communication will be based on Wi-Fi router. Therefore, in the case of extending the system, we may need to add more sensors, alarm LEDs, and gate servos, etc. To sum up, we will establish the IT infrastructure system that can detect the parking or leaving vehicles, and we will establish the parking facility software that can monitor the parking facility, give car drivers the reservation functionality, and provide statistical data analysis of parking facility information.

This project aims to provide the people who are related to parking facility with carious functionality (such as, reservation, cancelling, monitoring, and checking statistical data). To achieve these goals, we should implement requirement engineering, design a small parking facility using Arduino 2560, and develop an actual parking management system.

The objectives that we defined for this project are:

1. To produce a requirement specification document of the smart parking facility system.
2. To establish a smart parking system that is based on the requirement specification document.
3. To satisfy the architectural drivers of our system so that we eventually make this system valuable and competitive in the parking garage industry.

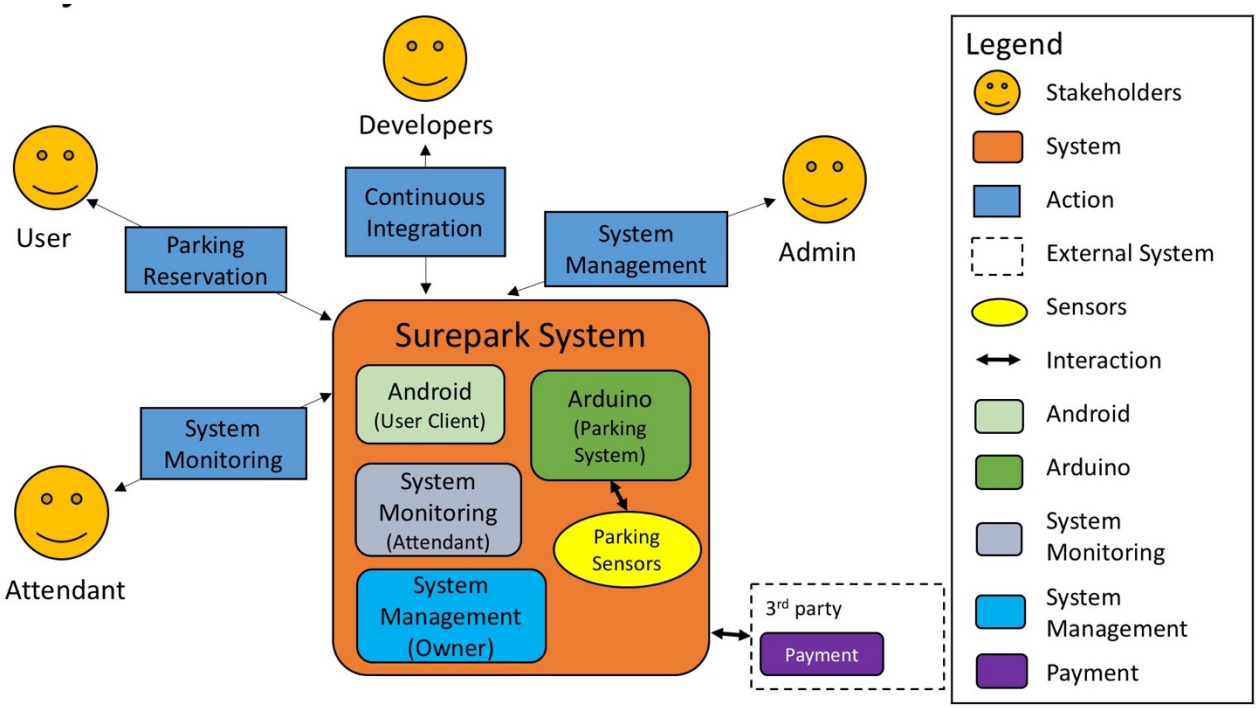

Figure 3. System context diagram

1.4 System context

This is our system context diagram that represents the overall system. Sure-park system is the server side. The client side will be Android application for reservation and web-based application for monitoring or management. Developers are our team members who try to develop the system and applications, and also try to integrate it continuously. User is a car driver who wants to reserve a parking slot by using Android application. Admin is an owner of parking facility who wants to check parking facility's statically analyzed data by using web-based management application. And also, attendant is an employee who monitors the parking facility by using web-based monitoring application. Our system includes Arduino device that assumes a virtually made parking facility, and this Arduino device is connected with various sensors by wire communication. Lastly, our system will use an external third-party system which offers a payment functionality.

2. Architectural Drivers

2.1 Functional requirements

2.1.1 Functional requirement lists

List	Description	Use case
1	Car Driver reserves a parking slot.	UC01 : Reserve parking slot
2	Car Driver checks the reservation information.	UC02 : Check reservation
3	Car Driver cancels the reservation.	UC03 : Cancel reservation
4	When cars arrive at the gate of the parking facility, Car Driver does authentication.	UC04 : Do an authentication
5	Car Driver will automatically pay his fee with credit card information based on the duration of parking.	UC05 : Pay a parking fee
6	When a Car Driver parks his car to the slot that is not his slot, system reallocates the parking slot.	UC06 : Get reallocation
7	Attendant monitors the parking facility and receives a notification.	UC07 : Monitor parking facility
8	The owner check the parking facility's statistical data.	UC08 : Check statistic data

2.1.2 Use case diagram

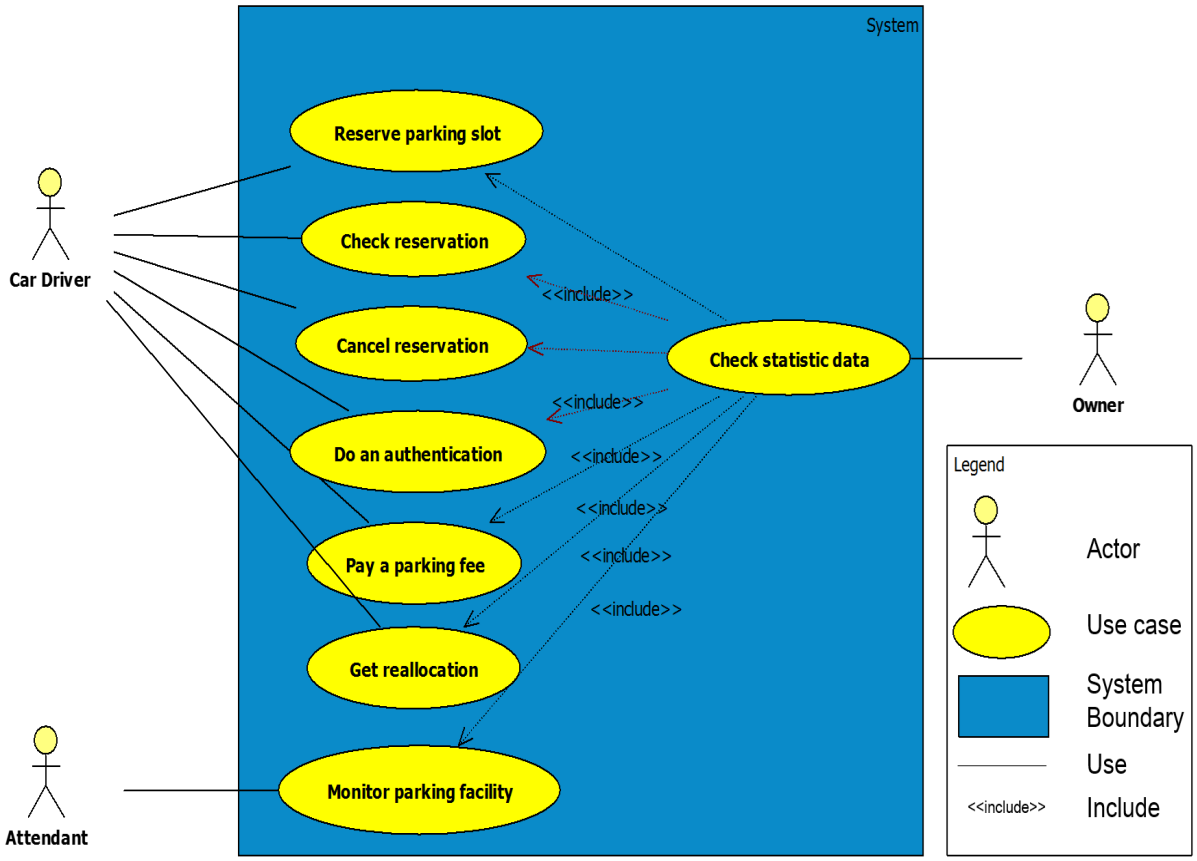

Figure 4. Use case diagram

2.1.3 Entities of the system

Entity name: Car Driver	**Entity ID:** E01
Description: The Car Driver is a human user of the system, and he uses the parking facility by reserving a parking slot.	
Provides assumptions: The Car Driver will provide: • Textual information such as phone number, credit card number and reservation time for parking reservation.	
Requires assumptions: The Car Driver requires the system to provide: • An environment to making use of the parking reservation. • Abilities to save, search, and cancel to parking reservation information. • Detailed Parking reservation Information.	

Identified use cases:
• UC01 : Reserve parking slot • UC03 : Do an authentication • UC04 : Pay a parking fee • UC05 : Get reallocation

Entity name: Attendant	**Entity ID:** E02
Description: The Attendant is a human user of the system, and he monitors parking facility.	
Provides assumptions: The Attendant will provide: • Managing the monitoring user interface.	
Requires assumptions: The Attendant requires the system to provide: • An environment to aid in the parking monitor.	
Identified use cases: • UC06 : Monitor parking facility	

Entity name: Owner	**Entity ID:** E03
Description: The Owner is a human user of the system, and he manages a parking facility and views the statistical data.	
Provides assumptions: The Owner will provide: • Textual information such as ID and Password for as a chief manager. • Controlling the manager user interface.	
Requires assumptions: The Owner will provide: • An environment to aid in the parking facility management. • Parking lot's statistical information.	
Identified use cases: • UC06 : Check statistic data	

Entity name: Parking Reservation Application	**Entity ID:** E04
Description: The Parking Reservation Application is the system that will provides reservation and confirmation of parking slot. It also supports notification.	
Provides assumptions: The Parking Reservation Application will provide: • Information for save, search, and cancel to parking reservation.	
Requires assumptions: The Parking Reservation Application requires the system to provide: • Parking reserve Information such as parking slot, authentication and etc.	

• Notification of parking fee and Parking slot reallocation.
Identified use cases: • UC01 : Reserve parking slot • UC03 : Pay a parking fee • UC04 : Get reallocation

Entity name: Authentication System	**Entity ID:** E05
Description: The Authentication System is the system that will provides Car Driver authentication.	
Provides assumptions: The Authentication System will provide: • Authentication environment.	
Requires assumptions: The Authentication System requires the system to provide: • Car Driver's authentication number.	
Identified use cases: • UC02 : Do an authentication	

Entity name: Parking Monitoring System	**Entity ID:** E06
Description: The Parking Monitoring System is the system that will provides the parking statistic data to owner.	
Provides assumptions: The Parking Monitoring System will provide: • Parking status information.	
Requires assumptions: The Parking Monitoring System requires the system to provide: • Attendant's manage.	
Identified use cases: • UC06 : Monitor parking facility	

Entity name: Parking Management System	**Entity ID:** E07
Description: The Parking Management System is the system that will provide functions of monitoring the parking facility and notification to Attendant.	
Provides assumptions: The Parking Management System will provide: • Parking management environment. • Parking statistical information.	
Requires assumptions: The Parking Management System requires the system to provide: • Owner's authentication information such as ID and Password. • Owner's manual control.	

Identified use cases: • UC07 : Check statistic data

Entity name: Arduino Sensor	**Entity ID:** E08
Description: The Arduino Sensors that will check the parking state, and check the car's existence in front of the access gate.	
Provides assumptions: The Arduino Sensor will provide: • Infrared Sensors.	
Requires assumptions: The Arduino Sensor requires the system to provide: • Always be available.	
Identified use cases: • UC04 : Pay a parking fee • UC05 : Get reallocation • UC06 : Monitor parking facility	

Entity name: Server	**Entity ID:** E09
Description: The Server collects data from sensors and offers authorization function to users. Also, it provide the search, store and update transactions used by Database.	
Provides assumptions: The Server will provide: • The set of authorization information collected from Car Driver and Owner. • The set of data collected from the sensors. • The set of parking reservation information.	
Requires assumptions: The Server requires the system to provide: • Sensor values. • Car Driver, Attendant and Owner's commands.	
Identified use cases: • UC01 : Reserve parking slot • UC03 : Do an authentication • UC04 : Pay a parking fee • UC05 : Get reallocation • UC06 : Monitor parking facility • UC07 : Check statistic data	

Entity name: Database	**Entity ID:** E10
Description: A Database is a collection of data that is organized, and its contents can be accessed, managed, and updated.	

Provides assumptions: The Database will provide: • Owner Data. • Parking lot Data. • Parking slot Data. • Reservation Data. • Sensor Data.
Requires assumptions: The Database requires the system to provide: • Server's queries.
Identified use cases: • UC01 : Reserve parking slot • UC03 : Do an authentication • UC04 : Pay a parking fee • UC05 : Get reallocation • UC06 : Monitor parking facility • UC07 : Check statistic data

2.1.4 Use case description

Use case title: Reserve parking slot	**Use case ID:** UC01
General use case description: If there is an empty parking slot, the user can reserve a parking slot. System allocates the parking slot to the user.	
Entities involved: • E01 – Car Driver • E04 – Parking Reservation Application • E09 – Server • E10 – Database	
Pre-conditions: The Car Driver (E01) has a telephone number and a valid credit card information.	
Primary use case flow of events:	
1.	Car Driver (E01) turns on the Parking Reservation Application (E04).
2.	Car Driver (E01) enters his telephone number, credit card number and parking reservation time into the Parking Reservation Application (E04).
3.	This information is transmitted from the Parking Reservation Application (E04) to Server (E09).
4.	The Server (E09) stores the parking reservation information in the Database (E10) and transmits an authentication number, reservation time and parking slot to the Parking Reservation Application (E04).
5.	Car Driver (E01) checks the parking reservation status in the Parking Reservation Application (E04).
Primary use case post-conditions: The Car Driver (E01) has the parking schedule information.	
Alternative use case 1's flow of events:	
1-1.	Transmission from the Parking Reservation Application (E04) to Server (E09) failed.
1-2.	Parking Reservation Application (E04) shows the transmission failure message.
Alternative use case post-conditions: Car Driver (E01) can see a transmission failure message in the Parking Reservation Application (E04).	
Alternative use case 2's flow of events:	
2-1.	If there is no empty parking slot, Server (E09) transmits the impossible reservation message to the Parking Reservation Application (E04).
2-2.	Parking Reservation Application (E04) shows the impossible parking reservations message.
Alternative use case post-conditions:	

Car Driver (E01) can see the impossible parking reservations message in the Parking Reservation Application (E04).

Use case title: Check Reservation	**Use case ID:** UC02
General use case description: The Car Driver wants to check his parking reservation information.	
Entities involved: • E01 – Car Driver • E04 – Parking Reservation Application • E09 – Server • E10 – Database	
Pre-conditions: The Car Driver (E01) must have reserved the parking slot already, and the reservation information must have been stored in local storage which means Android.	
Primary use case flow of events:	
1.	The Car Driver (E01) executes the reservation check function on the Parking Reservation Application (E04).
2.	If the reservation is already done, the Parking Reservation Application (E04) transmits the check request to the Server (E09).
3.	The Server (E09) accesses to the Database (E10) according to the check request, and transmits the result of check request to the Parking Reservation Application (E04).
4.	The Car Driver (E01) confirms the result of check request by using the Parking Reservation Application (E04).
Primary use case post-conditions: The Car Driver (E01) has succeeded in checking his parking reservation.	
Alternative use case 1's flow of events:	
1-1.	If Server (E09) cannot be connected, the Parking Reservation Application tries to access its local storage.
1-2.	The Parking Reservation Application (E04) shows the information of reservation status on the screen from its local storage.
Alternative use case post-conditions: The Car Driver can check the reservation information by utilizing the local storage of Parking Reservation Application (E04).	

Use case title: Cancel Reservation	**Use case ID:** UC03
General use case description: The Car Driver wants to cancel his parking reservation.	

Entities involved: • E01 – Car Driver • E04 – Parking Reservation Application • E09 – Server • E10 – Database	
Pre-conditions: The Car Driver (E01) should reserve the parking slot beforehand, and the information of reservation should have been stored in local storage.	
Primary use case flow of events:	
1.	The Car Driver (E01) press the cancel button on the UI of the Parking Reservation Application (E04).
2.	When the cancel button is pressed, the Parking Reservation Application (E04) transmits the cancel request to the Server (E09).
3.	The Server (E09) modifies the Database (E10) according to the cancel request, and transmits the result of cancel to the Parking Reservation Application (E04).
4.	The Car Driver (E01) confirms the result of cancel request by using the Parking Reservation Application (E04).
Primary use case post-conditions: .The Car Driver (E01) has succeed in cancelling his parking reservation.	
Alternative use case 1's flow of events:	
1-1.	Transmission from the Parking Reservation Application (E04) to Server (E09) fails
1-2.	The Parking Reservation Application (E04) show a transmission failure message on the screen.
Alternative use case post-conditions: The Car Driver checks the transmission failure message by using the Parking Reservation Application (E04).	

Use case title: Do an authentication	**Use case ID:** UC04
General use case description: When cars arrive at the gate in front of the parking facility, user does authentication. If the authentication is successful, user enters into the parking facility.	
Entities involved: • E01 – Car Driver • E05 – Authentication System • E09 – Server • E10 – Database	
Pre-conditions:	

The Car Driver (E01) has the parking schedule information (UC01). The Car Driver (E01) arrives at the parking facility.	
Primary use case flow of events:	
1.	Car Driver (E01) enters the authentication number into the Authentication System (E05).
2.	The Server (E09) checks the entered authentication number against the number in the Database (E10).
3.	Once authenticated, the Server (E09) opens the gate of parking facility.
4.	The Car Driver (E01) moves into the parking facility.
Primary use case post-conditions: The Car Driver (E01) has moved into the parking facility.	
Alternative use case 1's flow of events:	
1-1.	If the Car Driver (E01) fails authentication, Server (E09) transmits the authentication fail message to the Authentication System (E05).
1-2.	Authentication System (E05) shows the authentication fail message.
Alternative use case post-conditions: There is an authentication fail message on the Authentication System. (E05).	

Use case title: Pay a parking fee	**Use case ID:** UC05
General use case description: User will automatically be charged with their credit card information based on the duration of parking.	
Entities involved: • E01 – Car Driver • E04 – Parking Reservation Application • E08 – Arduino Sensor • E09 – Server • E10 – Database	
Pre-conditions: A Car Drivers (E01) was parked in the parking slot (UC01, UC04).	
Primary use case flow of events:	
1.	A Car Driver (E01) tries to get out of the parking facility.
2.	When the Car Driver (E01) arrives in front of the exit gate, Arduino Sensor (E08) transmits the parking status to the Server (E09).
3.	The Server (E09) calculates the parking fee by using Database (E10) and charge a parking fee by using the Car Driver (E01)'s credit card information.

4.	The Server (E09) transmits the parking fee notification to the Parking Reservation Application (E04).
5.	The Car Driver (E01) checks the parking fee notification by using the Parking Reservation Application (E04).
Primary use case post-conditions: The Car Driver (E01) has leaved the parking facility and has paid his parking fee.	
Alternative use case 1's flow of events:	
1-1.	Transmission from the Arduino Sensor (E08) to Server (E09) failed.
1-2.	Transmission of the parking status from the Arduino Sensor (E08) to Server (E09) is tried automatically.
Alternative use case post-conditions: Transmission of the parking status from the Arduino Sensor (E08) to Server (E09) succeed.	
Alternative use case 2's flow of events:	
2-1.	Transmission from Server (E09) to Parking Reservation Application (E04) failed.
2-2.	Transmission of the parking fee notification from Server (E09) to Parking Reservation Application (E04) is tried automatically.
Alternative use case post-conditions: Transmission of the parking fee notification from Server (E09) to Parking Reservation Application (E04) has succeeded.	

Use case title: Get reallocation	**Use case ID:** UC06
General use case description: If a Car Driver (E01) parks in the slot that was not supposed to be his one, system reallocates the parking slot and transmits the notification to user.	
Entities involved: • E01 – Car Driver • E04 – Parking Reservation Application • E08 – Arduino Sensor • E09 – Server • E10 – Database	
Pre-conditions: A Car Driver (E01) parks in the slot that is not for him (UC01, UC03).	
Primary use case flow of events:	
1.	Arduino Sensor (E08) checks the parking facility status and transmits this information to the Server (E09).
2.	The Server (E09) compare the real parking slot status with the Car Driver's reserved parking slot information in the Database (E10).

3.	If the parking slot status and reservation information are different each other, the Server (E09) inevitably reallocates the parking slot (which the Car Driver actually parked in) to the Car Driver (E01) and update this information in the Database (E10).
4.	The Server (E09) transmits the parking slot reallocation notification to the Parking Reservation Application (E04).
5.	The Car Driver (E01) sees the notification message which explains that 'your slot is automatically reallocated', by using the Parking Reservation Application (E04).
Primary use case post-conditions: The Car Driver (E01) received a message of the reallocation of the parking slot from the Server (E09).	
Alternative use case 1's flow of events:	
1-1.	Transmission from the Arduino Sensor (E08) to Server (E09) failed.
1-2.	Transmission of the parking status from the Arduino Sensor (E08) to the Server (E09) is tried automatically.
Alternative use case post-conditions: Transmission of the parking status from the Arduino Sensor (E08) to the Server (E09) has succeeded.	
Alternative use case 2's flow of events:	
2-1.	Transmission from the Server (E09) to the Parking Reservation Application (E04) failed.
2-2.	Transmission of the parking slot reallocation notification from the Server (E09) to the Parking Reservation Application (E04) is tried automatically.
Alternative use case post-conditions: Transmission of the parking slot reallocation notification from the Server (E09) to the Parking Reservation Application (E04) has succeeded.	

Use case title: Monitor parking facility	**Use case ID:** UC07
General use case description: The attendant monitors the parking facility and receives a notification.	
Entities involved: • E02 – Attendant • E06 – Parking Monitoring System • E08 – Arduino Sensor • E09 – Server • E10 – Database	
Preconditions:	

Arduino Sensors (E08) always check the parking facility.	
Primary use case flow of events:	
1.	The Attendant (E02) oversees the parking status by using the Parking Monitoring System (E06).
2.	Arduino Sensors (E08) detect the change of the parking status and transmit this information to the Server (E09).
3.	If the parking status is changed, the Server (E09) updates the information in the Database (E10) with the received information.
4.	The Attendant (E02) checks the change of parking status by using the Parking Monitoring System (E06).
Primary use case post-conditions: The Server (E09) updated parking information in the Database (E10). The Attendant (E02) checked the parking status by using the Parking Monitoring System (E06).	
Alternative use case 1's flow of events:	
2-1.	Transmission from the Arduino Sensor (E08) to the Server (E09) failed.
2-2.	Transmission of the parking status from the Arduino Sensor (E08) to Server (E09) is tried automatically.
Alternative use case post-conditions: Transmission of the parking status from the Arduino Sensor (E08) to Server (E09) has succeeded.	

Use case title: Check statistic data	**Use case ID:** UC08
General use case description: The owner checks statistical data of parking facility.	
Entities involved: • E03 – Owner • E07 – Parking Management System • E09 – Server • E10 – Database	
Pre-conditions: The Owner (E03) has his own account.	
Primary use case flow of events:	
1.	The Owner (E03) tries to login by entering the ID and Password into the Parking Management System (E07).
2.	The Server (E09) confirms the ID and Password by comparing with the Database (E10).

3.	If the owner is confirmed, the Server (E09) transmits the statistical data of the parking facility to the Parking Management System (E07).
4.	The Owner (E03) checks statistical data of parking facility by using the Parking Management System (E07).
Primary use case post-conditions: The Server (E09) has provided the statistically analyzed information to the Owner (E03) by displaying screens on the Parking Management System (E07).	
Alternative use case 1's flow of events:	
1-1.	Transmission (ID and password) from the Parking Management System (E07) to the Server (E09) failed.
1-2.	The Parking Management System (E07) shows a transmission failure message.
Alternative use case post-conditions: The owner (E03) sees a transmission failure message by using the Parking Management System (E07).	
Alternative use case 2's flow of events:	
2-1.	If the authentication of the owner is failed, the Server (E09) transmits an authentication failure message to the Parking Management System (E07).
2-2.	The Parking Management System (E07) shows an authentication failure message.
Alternative use case post-conditions: The owner (E03) sees an authentication failure message by using the Parking Management System (E07).	

2.2 Quality attributes

2.2.1 Quality attribute context

Code	Name	Context	Trade-off
QA1	Scalability	Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth. In our system, scalability means that the system can accept more Arduinos on condition that the system performance should not be lowered, or should be slightly lowered. In other words, our System should operate normally when it is sold to the bigger facilities. It should be scaled out to bigger facilities.	Because of our increased throughput of data, there will be a slight decrease of performance. Also, it will be less maintainable, if we scale out more.
QA2	Security	Security requirements are a class of quality attributes that relate to system confidentiality, integrity and availability (CIA). In terms of confidentiality, we should make manipulating our system possible only when the owner tries to access. In terms of integrity, it involves maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle. Data must not be changed in transit, and steps must be taken to ensure that data cannot be altered by unauthorized people (for example, in a breach of confidentiality). In terms of availability, our system should be accessible whenever the owner tries to manipulate it. However, availability is separately considered in our QA strategies, so it will be addressed in the other part. Above all, we will focus on the prevention of malicious or accidental actions, such as hacking information. (it falls within confidentiality)	Because security attributes require complex step, it will decrease usability, and it can cause slightly longer latency.
QA3	Availability	Availability is the proportion of time in which the system performs its functionality normally. Usually, people use some common way that is measuring the availability as follows: Availability = MTBF / (MTBF + MTTR) * 100. MTBF stands for Mean Time Between Failure, and MTTR stands for Mean Time To Repair. Our Team will improve availability by	Because of synchronization of backup DB, there will be the increased latency. Also, because we will store the information in

		minimizing MTTR. We tried several design decisions. However, we would not measure the availability actually. We are just planning to apply several techniques to boost our system's availability.	client side, there will be some security problems.

2.2.2 Quality attribute scenarios

Our team decided that it is enough to apply our design method with regard to security and availability. Therefore, we only described the quality attributes scenario for scalability.

Title	System scalability
Stimulus	The owner (our customer) of system decided to extend the system to accommodate far more cars.
Source(s)	The owner
Environmental Condition	The result system is very successful and it brings much incomes to the owner.
Architectural elements	The owner, the whole system
System Response	The parking facility has succeeded in enlarging the size of it. It is able to accept up to 500 cars.
Response measures	The number of Android and Arduino's requests per second should be over 500.

2.3 Constraints

2.3.1 Technical constraints

Consideration	Technical constraints
Hardware constraints	Arduino Mega 2560 Arduino Wi-Fi Shield Sensor modules - 4 IR Parking LED (Parking detector) - 4 LED (Parking led) - 2 Gate servo (Gate open/close) - 1 Entry IR LED, 1 Entry IR detector - 1 Exit IR LED, 1 Exit IR detector - 1 Entry LED, 1 Exit LED - 1 Car Pre-made parking facility model (supported by CMU)
Software constraints	Java Language (Server side) C Language (Arduino)
Environmental constraints	Local sever (laptop) Wi-Fi Router which is shared with the class No internet connection All the system elements should communicate each other through Wi-Fi connection

2.3.2 Business constraints

Consideration	Business Constraints
Stakeholders' demands	**Owner:** Owner should be able to check daily and monthly statistics of the system (such as daily peak time usage, monthly revenues, parking slot statistics, and average occupancy). **Attendant**: Attendant should be able to monitor the parking facility (occupied slots) by using our system. **Car Drivers**: Car Drivers should be able to make a parking reservation by using our system.
Available devices limitations	Our team should use the provided hardware, sensors by CMU.
Time limitation	We have only 5 weeks for 'Sure-park' project
Legal restrictions	Private information should be secured.
Policy	Arduino device cannot be taken out of the classroom. Therefore, our team is able to test this only in the classroom.

2.4 Prioritized architectural drivers

Priority	ADS	Description
1st	Time Limitation (Business Constraint)	Our team should complete the project within the limited short period. And also, we should accomplish the required quality.
2nd	Scalability (Quality Attribute)	Our System should operate well when it is sold to the bigger facilities, or it should be easily extended larger. It should be scaled out to a bigger parking lot.
3rd	Security (Quality Attribute)	We should make the information not be stolen by the malicious users. For example, we should protect our customer's credit card information from hacking.

3. Architectural & detailed design

3.1 Design decisions

3.1.1 General decisions

- We did not make login function for car drivers. It is because login function is for differentiating the car drivers, but we can do it with only phone number. Also, Tony said that we do not need to make login function for the car drivers. Additionally, our system can be accessible to a new customer because he does not need to apply for membership.

- There is a required information when a car driver tries to reservation. Our team concluded that we need reservation time, phone number, credit card information, and facility ID. Of course, reservation time is for the car driver's exact reservation. Phone number is for differentiation of who it is. Credit card information is for differentiation of who it is, and for charge. Lastly, facility ID is for scalability in the future, because user can select a facility among many facilities when the system is scaled out.

- The owner has to login the system with ID and password. It is because the only owner can have access to the statistical data, system setting configuration function.

- The owner can set the parking fee per an hour and the duration of grace period. It is because the owner needs to control fee and grace period according to market situation.

- The only one car can be entered in the parking facility at once, and we will assign car drivers parking slot sequentially.

- We have an assumption on the payment function. We have to use a Wi-Fi router in the classroom, so we cannot connect to external internet. Therefore, we assume that we are using an external third party payment module, such as Samsung-pay.

- We used Spark framework in server. There are several reasons that we chose Spark. This framework is a simple and lightweight Java web framework built for rapid development. A typical Spark application is a lot less verbose than most application written in other Java web frameworks. Spark focuses on being as simple and straight-forward as possible, without the need for cumbersome (XML) configuration, to enable very fast web application development in pure Java with minimal effort. It's a totally different paradigm when compared to the overuse of annotations for accomplishing pretty trivial stuff seen in other web frameworks. These characteristics can help us build the server with relatively short time. We need to develop server quickly, and also we need to be simple with respect to the project size (it is not that huge size). It is well-known that Spark is mainly used for creating REST API's, but it also supports a multitude of template engines. Therefore, we concluded that Spark is an optimal framework for us.

- We used MySQL in our database. It is mainly because our database developers are most familiar with MySQL. Actually, due to the short development time, we thought we cannot have enough time to consider or learn other database systems. Also, MySQL is open-source system, so we can use it freely from care of license.

- We decided that car drivers will use android application. We made our client program as android application to maximize user's convenience. Because users always have their

smartphone, so users can make and cancel their parking slot reservation anywhere, anytime.

- We made attendant and owner's programs as webpage which can be accessed through desktop, and it is JavaScript-based. We thought managing the parking lot through webpage is the most efficient and convenient way in owner and attendant's view.

- We used 'push server' because it can send information to android user in real time. It makes us possible to directly deliver the message (notification) from server to Arduino & Android.

3.1.2 Security-related decisions

- We decided that we will use HTTPS, and its purpose is to strengthen security of communication. However, we could not use HTTPS, because we should pay if we use HTTPS.

- We used AES256 and SHA256. Its purpose is to protect data in preparation for hacking.
  - AES256 (Advanced Encryption Standard)
    When server and clients communicate, we transmit data which are encrypted with AES256. Therefore, we can protect these data. Also, server stores client's data encrypted with AES256. Therefore, it is less risky although this data is hacked by malicious users.
  - SHA256 (Secure Hash Algorithm)
    When the owner tries to log in the owner's parking management system, we transmits owner's data by encrypting this data with random key. In other words, we use token. Therefore, we can protect the owner's information.

- We perform an authentication step. If we perform an authentication step by using only credit card number or phone number, there may be security risks. Therefore, we give car drivers authentication number when car drivers make a reservation. Car drivers should show their given number to enter the parking facility.

3.1.3 Scalability-related decisions

- We used the server-client pattern. The client-server pattern separates client applications from the services they use. This pattern simplifies systems by factoring out common services, which are reusable. Because servers can be accessed by any number of clients, it is easy to add new clients and add data to a system. In other words, we can add multiple Arduino to the server, so it can be said that our system can be scaled out. Similarly, servers can be added to support scalability or availability.

- We used middleware. RESTful event bus is middleware of our system. This makes us convenient to manage many connections through only one component. Also, this makes communication between heterogeneous type devices easy.

3.1.4 Availability-related decisions

- We made a backup database. Because we stores information of main database into backup database periodically, we can restore fast through backup DB whenever database does not

work.

- We used ‘shared preferences’ in Android. It is a XML-based local storage. Even though Android cannot communicate with server, Android can get data from its local storage.

- We used heartbeat pattern. Whenever malfunction of Arduino occurs, we should fix it as soon as possible. Therefore, we should check periodically whether Arduino works normally or not. To do this, we used heartbeat pattern. Arduino sends its status to server every 10 minutes. The systems that are directly related to human safety should be checked every seconds, but our system has no such a safety-critical characteristics. Therefore, we decided that 10 minutes is acceptable to our system.

- We used ping & echo pattern. Its purpose is to fix server as soon as possible in case it undergoes failure. Attendant’s parking monitoring system checks the status of server every two seconds. In detail, attendant’s monitoring system tries to require server to return parking slot status every two seconds. At this time, if there is no response from server, we can decide that there some problems with our server.

3.2 Physical perspective

3.2.1 Physical view

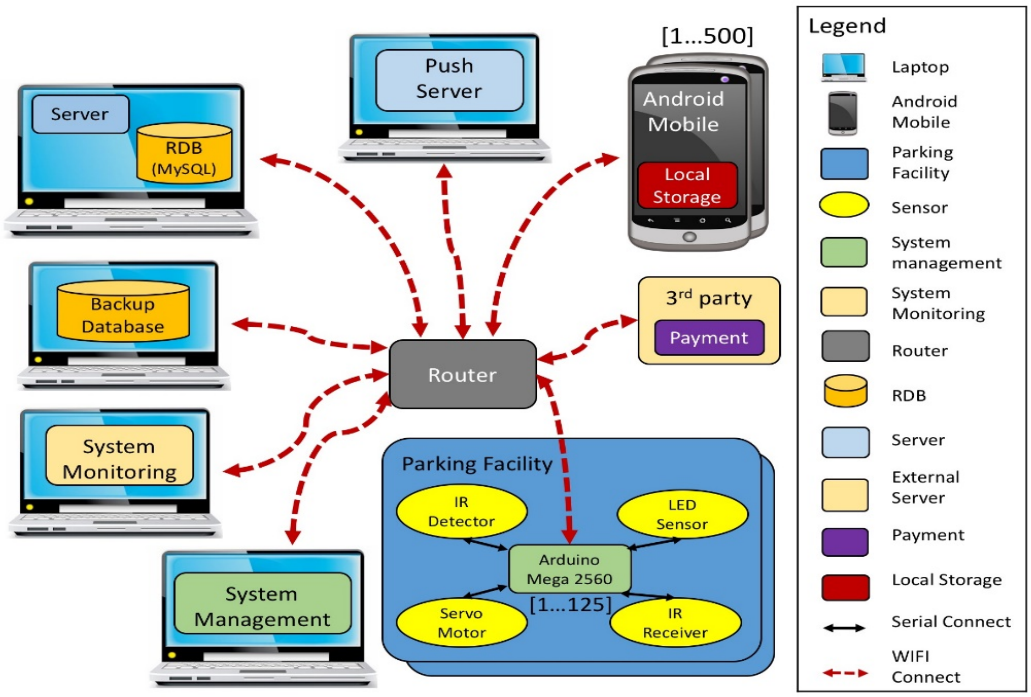

Figure 5. Physical view

This view shows the physical environments in the parking system. We use server-client pattern. Server includes relational database. There are four clients. Clients are car drivers, owner, attendant, and parking facility. Car drivers make a parking reservation by using their android devices through the Wi-Fi connection. Owner manages system by using its own web-based application. Also, Attendant monitors the parking facility by using its own web-based application. There is a backup database in a separated server, it is for system availability in case there is a system failure. We stores system data in main server into the backup database in secondary server regularly. For example, if main server's database go through a failure due to some problems, then the system will use the backup database. Push server functions as notification sender to the android device, and there are two kinds of notifications which are payment alarm and parking slot reallocation alarm. The number of Android clients which can be connected to server is up to 500 devices. The number of Arduino Mega 2560 which can be connected to server is up to 125 devices, while each Arduino contains four parking slots. This is mainly because there is a requirement that specify the minimum number of parking slots that our system should be able to be scaled out. This requirements are from our professor (Tony). For this reason, we annotated the limited number of devices in the figure. In terms of payment, we will assume that the payment of parking fee is performed by the third party external payment system, such as 'Samsung-pay', 'i-pay', and so on.

Because our system is based on the server-client pattern, we could boost scalability. For example, we can add Arduino hardware easily to server.

We can grasp overall division of system's hardware intuitively by seeing this figure. For example, we can see that there is a restriction of using only one router.

3.3 Static perspective

3.3.1 Module view

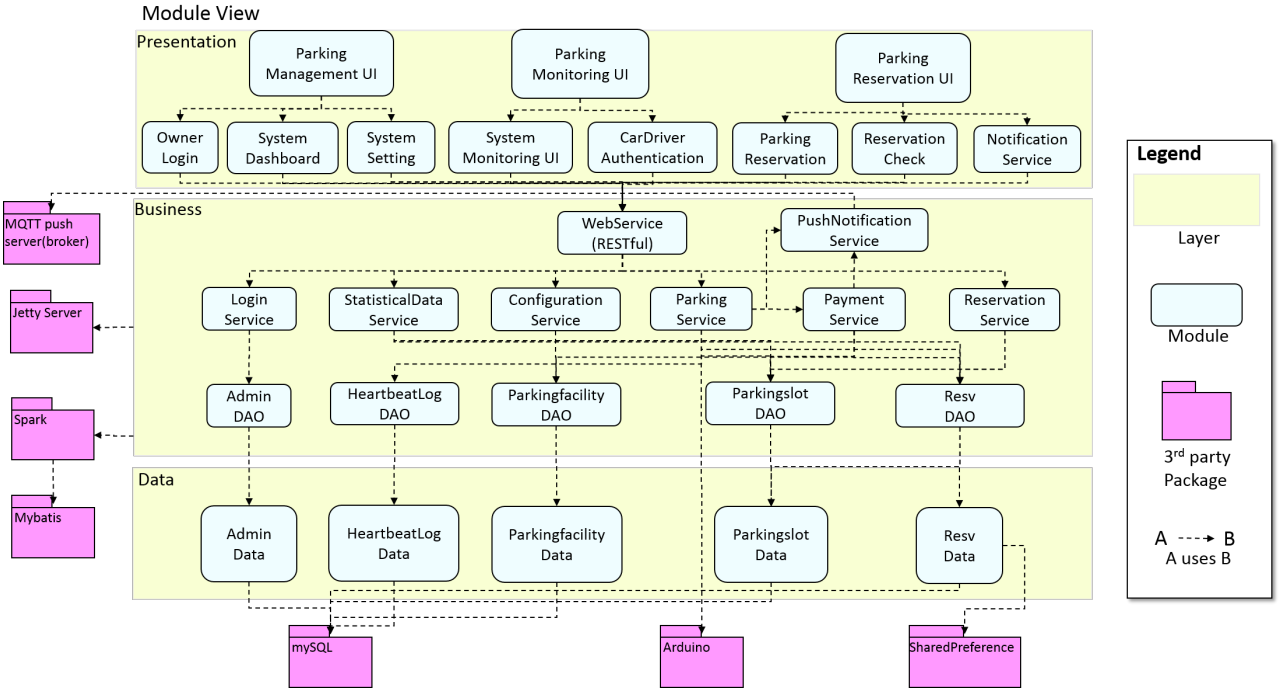

Figure 6. Module view

This figure shows our system's module view. It can be helpful to think about how we will write down the code before we start coding in the development phase.

We can see overall structure of our system intuitively. To explain the RESTful, we use REST in server to handle each service. Each service contacts Data layer via Data Access Object (DAO). The presentation layer has UI views which are displayed in the user's system. There are functional logics in business layer, so that they function as various individual functional modules. Car drivers' data and system data are stored in data layer.

For availability, we added local storage on Android (SharedPreference). It is used when the server does not work, so Android use this data. And also, we used backup DB, so it can boost our availability when main DB is under failure.

For scalability, we used MQTT push server. It used publish-subscribe pattern, so it has push-notification function. Therefore, it boosts scalability. Also, Web Service (RestFul) is our middleware, and it can communicate with heterogeneous devices by using http.

3.2.2. Class diagram

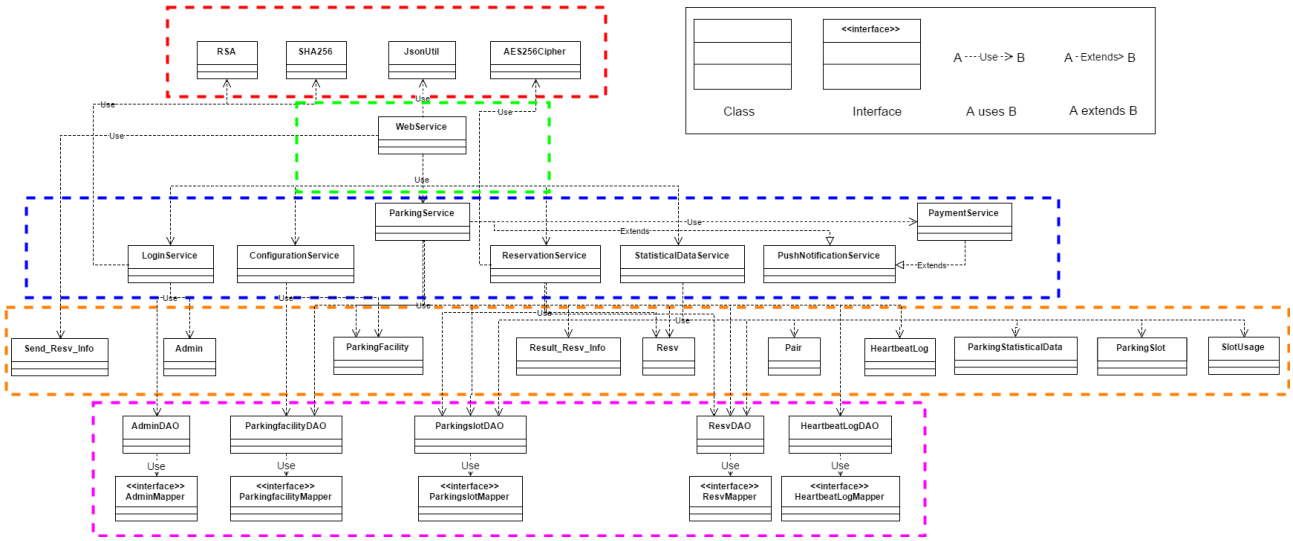

Figure 7. Server class diagram

This figure is class diagram of server. The rectangular region which is drawn with dotted line represents package. We made a package that includes classes doing similar behavior so that developers can understand structure and build easily when they implement.
This class diagram reflects quality attributes. For security, we made a SHA256 and AES256Cipher classes shown in red region to implement encryption of data which will be stored in database. For scalability, we made a WebService class shown in green region to implement RESTful service. Also, we made a PushNotificationService shown in blue region to implement push service. For availability, we made HeartbeatLogDAO class and HeartbeatLogMapper interface shown in purple region to implement heartbeat.

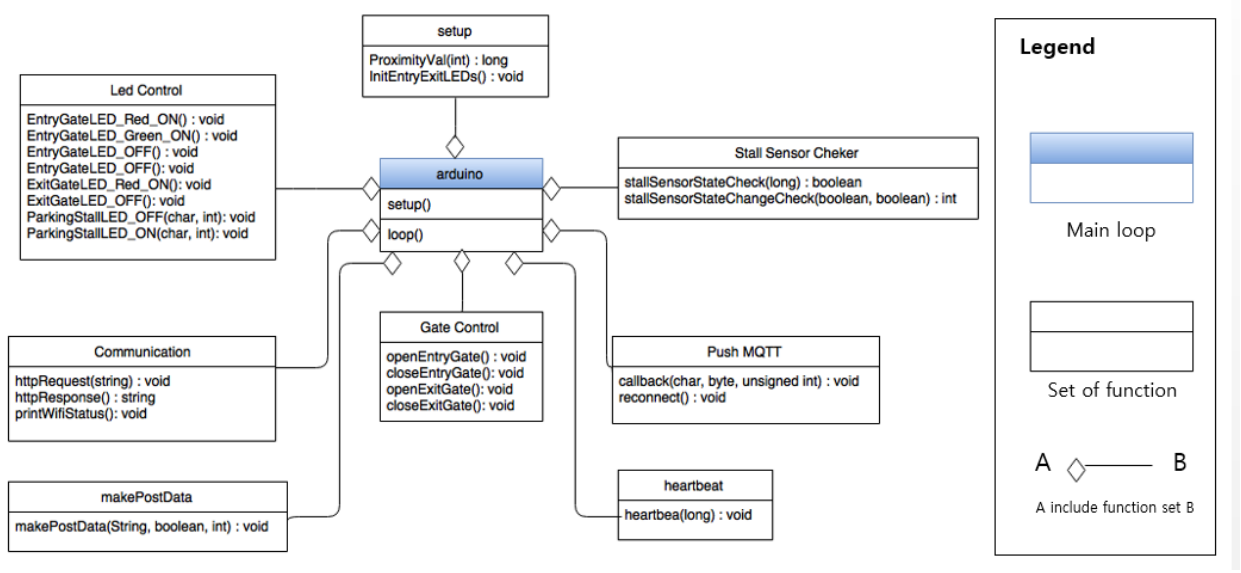

Figure 8. Arduino class diagram

This figure is class diagram of arduino. Because arduino executes the code located in main loop repeatedly, there are no classes. Rectangular which seems like a class is not an actual class, it is a set of function which do similar function. Developers can recognize easily what functions do similar things.
This class diagram reflects quality attributes. For scalability, we made a callback function shown in Push Receiver box to implement push service. For availability, we made a heartbeat function shown in http Communication box to implement heartbeat.

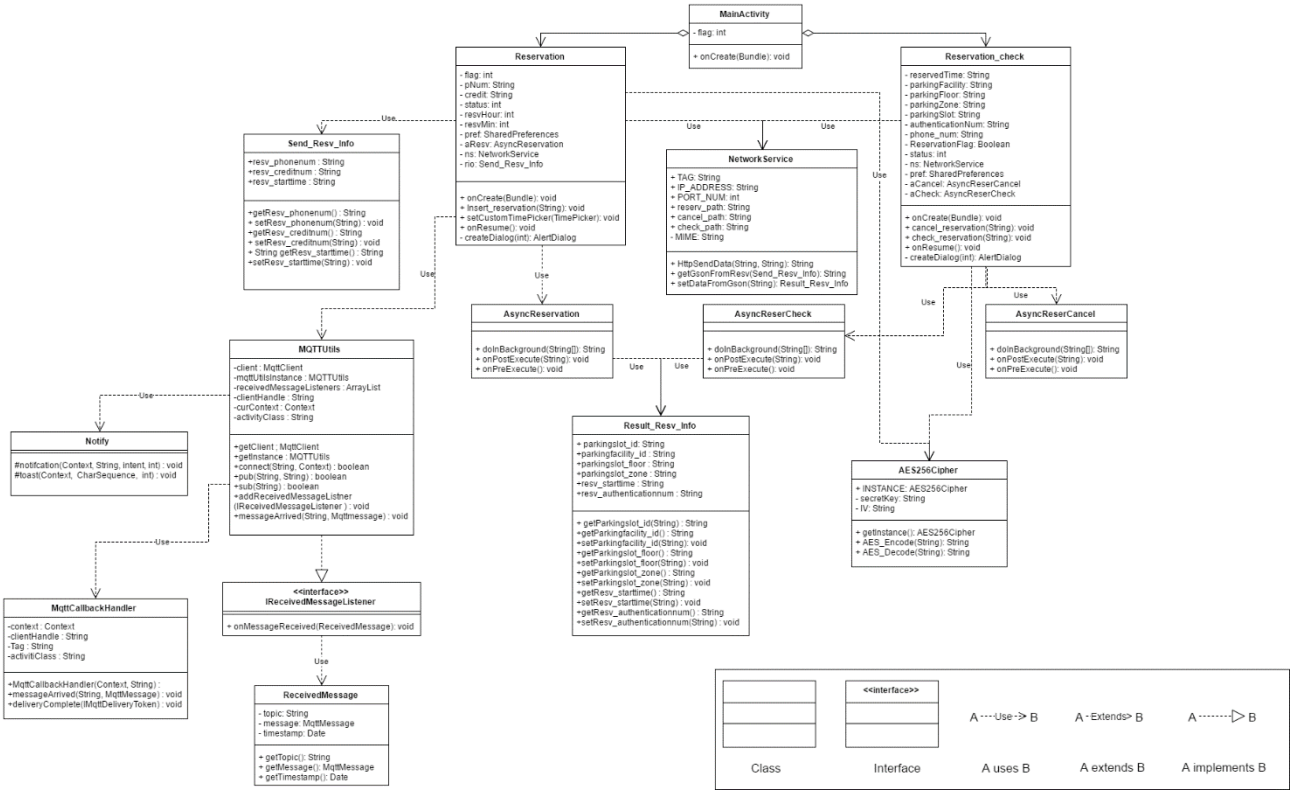

Figure 9. Android class diagram

This figure is class diagram of android application. This diagram shows relationship between classes to facilitate developers understand and build this program.
This class diagram reflects quality attributes. For security, we made AES256Cipher class to implement encryption of data which will be stored in database. For scalability, we made Notify class to implement push notification. Also, we made NetworkService class to implement RESTful service.

3.4 Dynamic perspective

3.4.1 Component & connector view

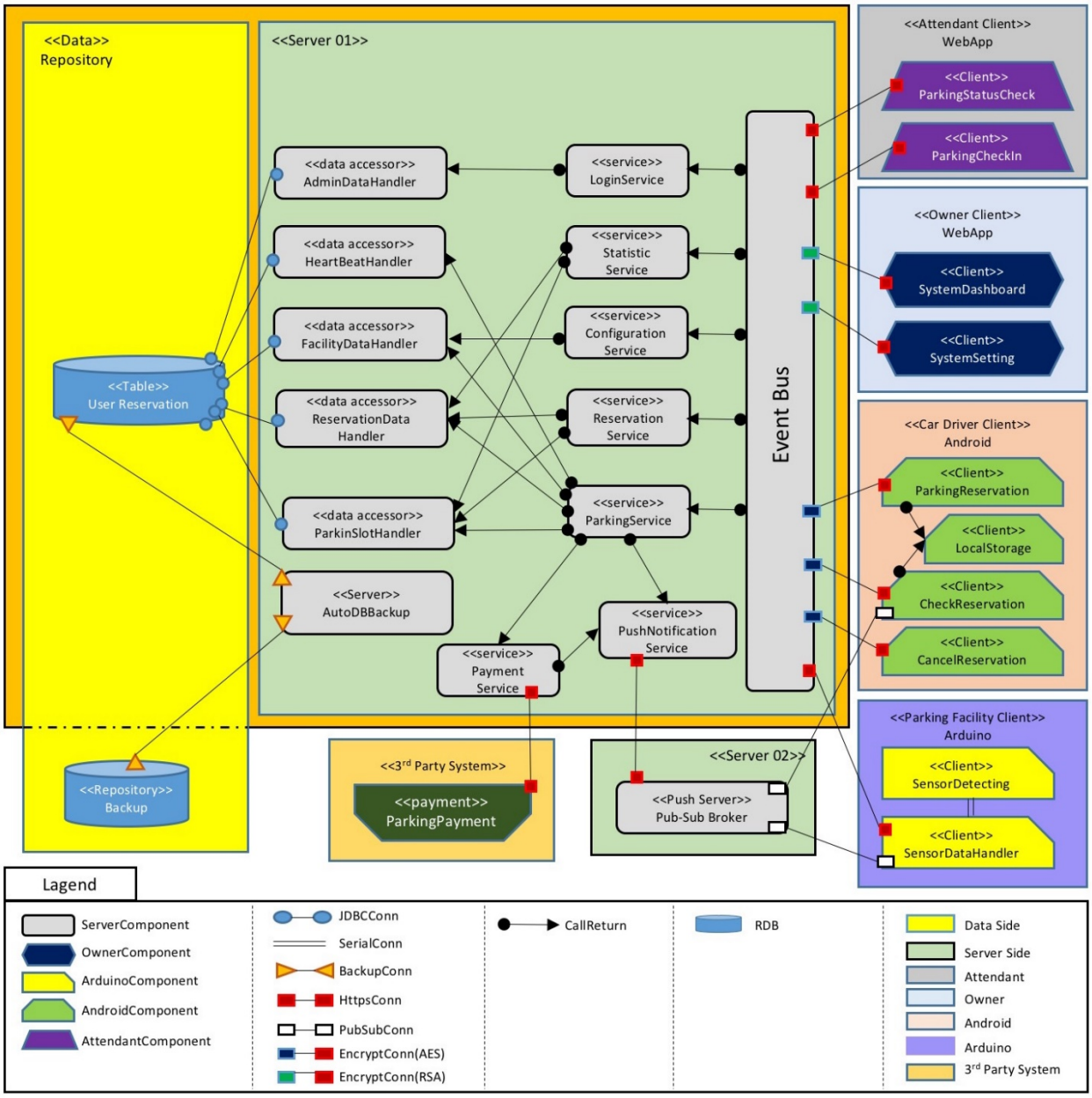

Figure 10. Component and connector view

This figure can help us understand the flow of our system. Also, it represents our "Quality Attribute" which are decided to be important in building our system.

We can find quality attributes for the system through this figure. Business logic and data repository are located in same machine. It improves a response time performance. In the availability aspect, we added local storage for saving the reservation information. It supported availability to users for checking information without internet connection. Backup database also improved availability in unexpected machine failure. For example, the hardware that includes server is broken and loss the

database, backup database helps recover the system. In the security aspect, server and client communicated each other with HTTP connection. Some connection supported AES 256 encryption for sending data. In the database, data is going to be encrypted for security. And this system also used stateless approach and token authentication for scalability, because it does not store the data in the server sessions and it can be scaled with multiple servers.

3.4.2 Sequence diagram

A. Use case 01: Reserve parking slot

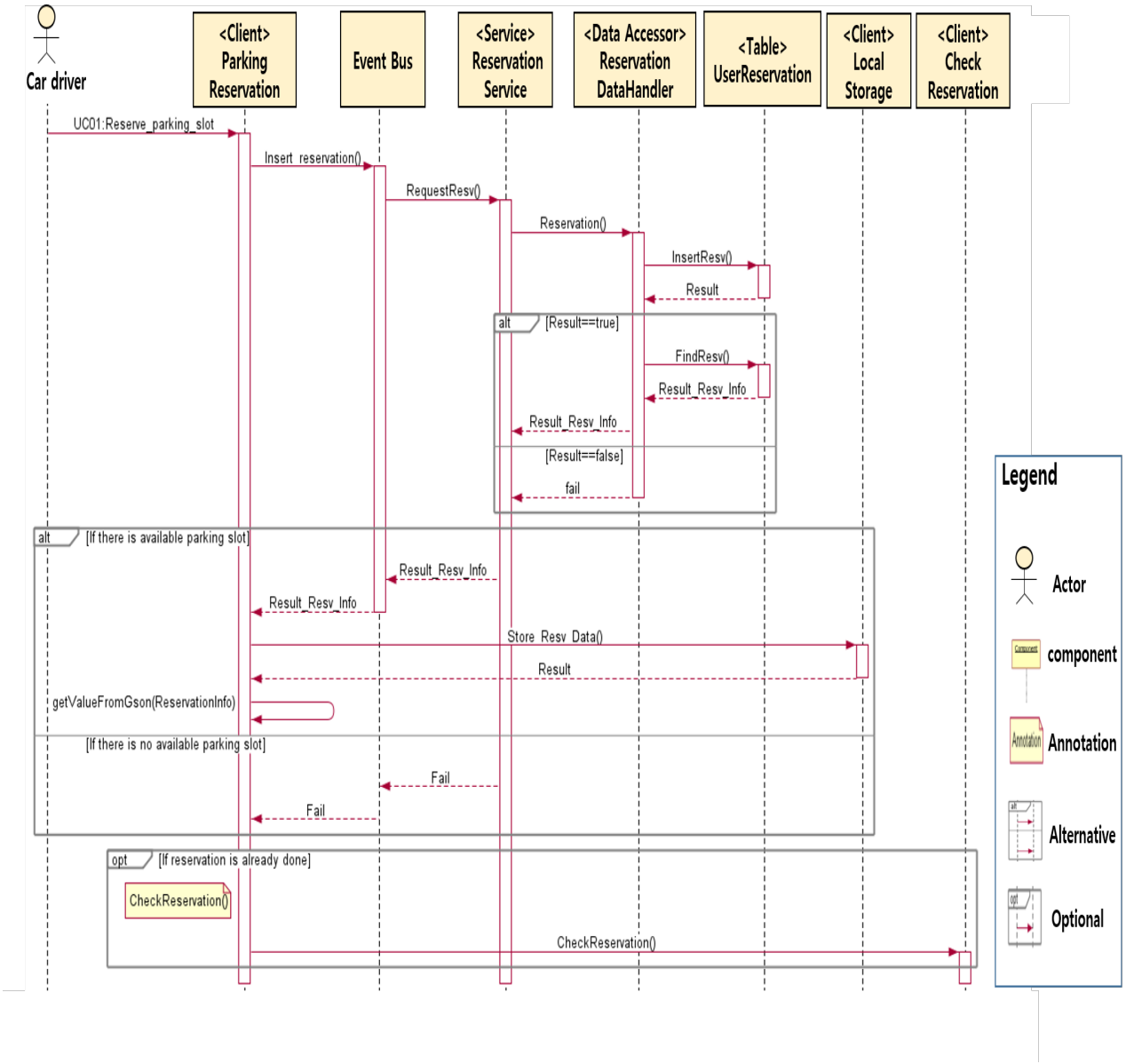

Figure 11. Sequence diagram of UC 01

This is a sequence diagram of UC01. The main actor is car driver and this sequence is executed by android app. When the car driver want to reserve the parking slot, the driver uses the android app to reserve the parking slot in the specific parking facility by encrypting the private information such as credit card number and phone number.

B. Use case 02: Check reservation

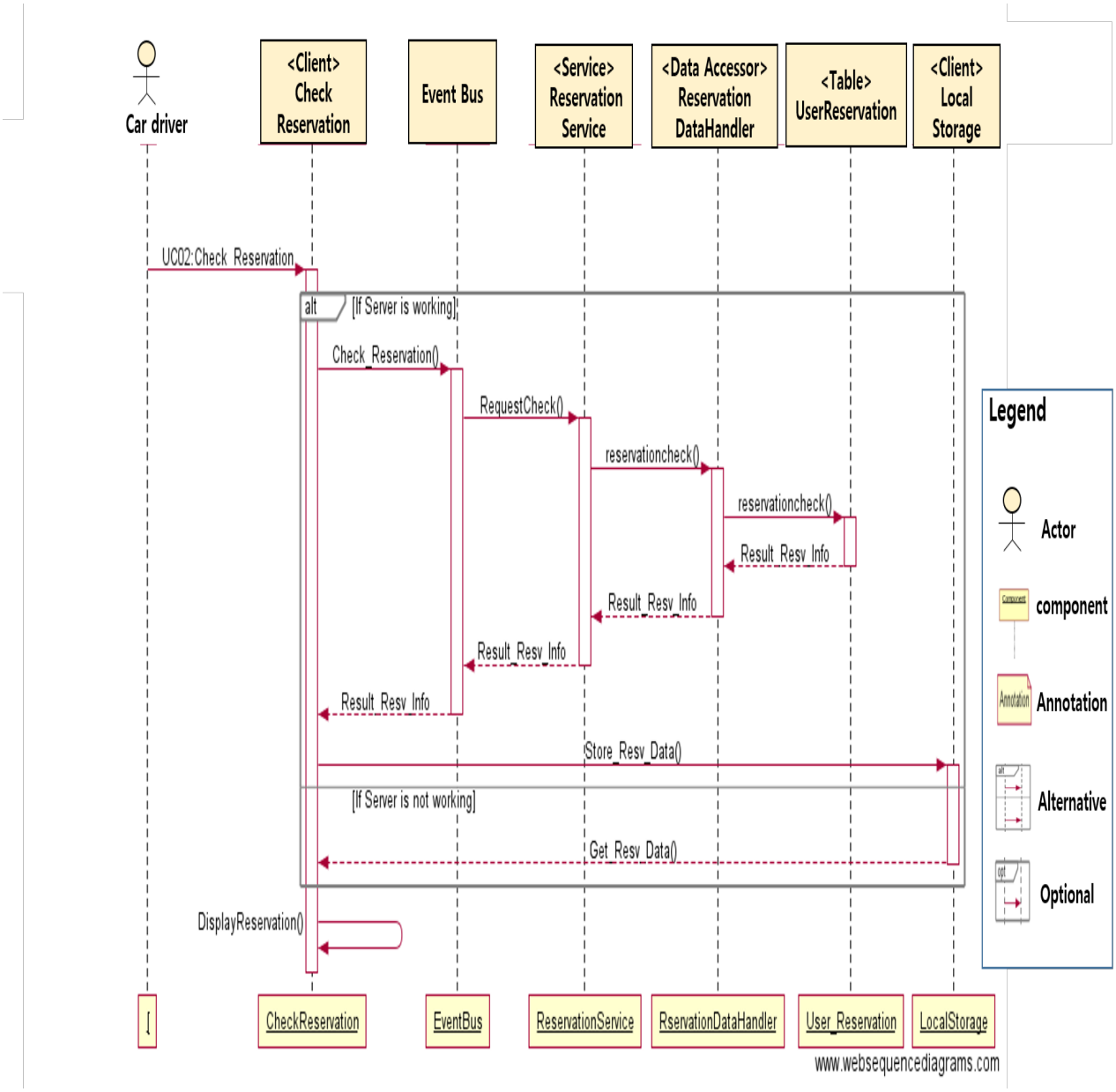

Figure 12. Sequence diagram of UC 02

This is a sequence diagram of UC02. The main actor is car driver and this sequence is executed by android app. When the car driver want to check the reservation information, the driver uses the android app to check the reservation information by requesting with phone number.

C. Use case 03: Cancel reservation

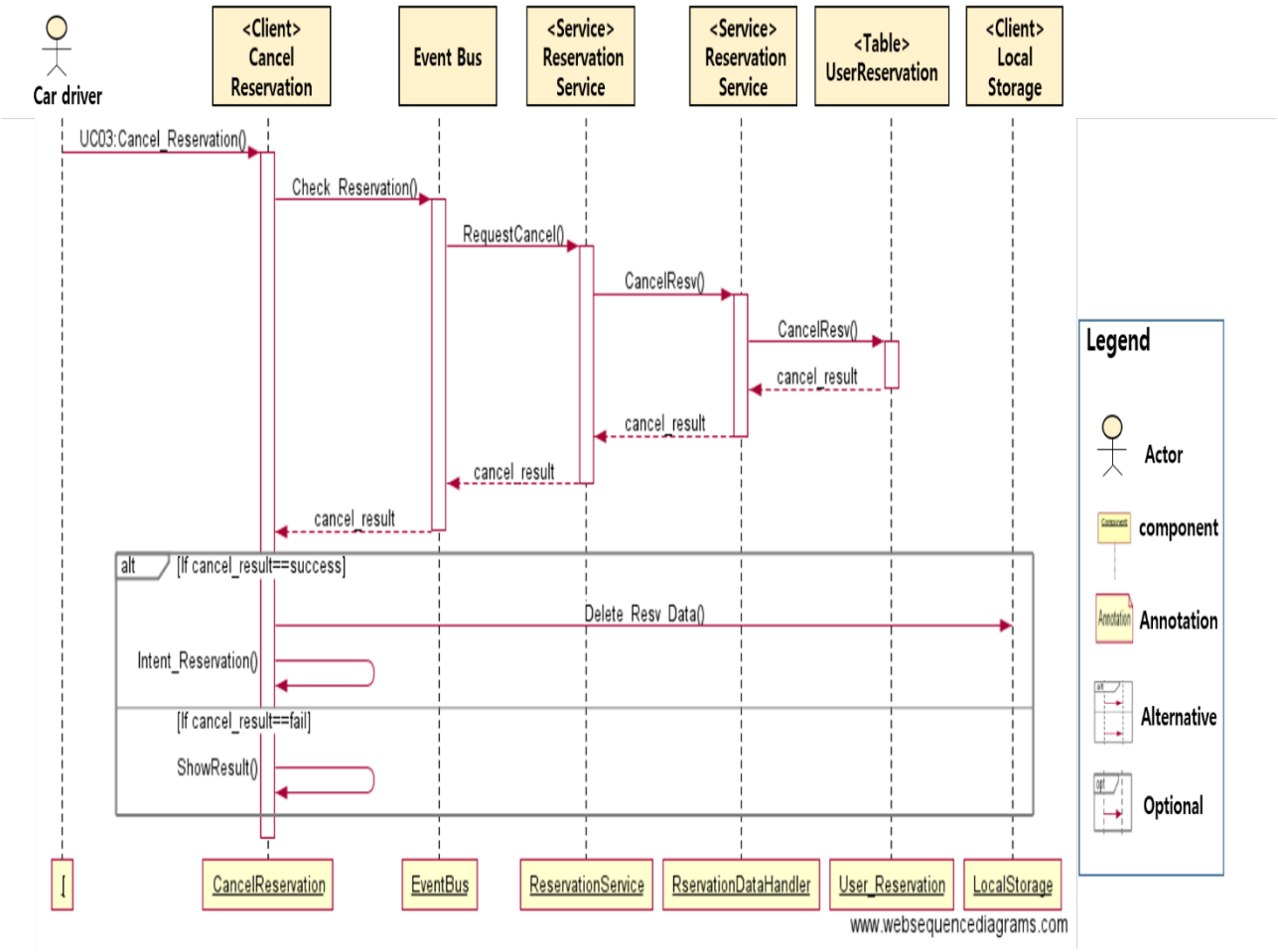

Figure 13. Sequence diagram of UC 03

This is a sequence diagram of UC03. The main actor is car driver and this sequence is executed by android app. When the car driver want to cancel the reservation, the driver uses the android app to cancel the reservation information by requesting with authentication number that car driver got in reservation process.

D. Use case 04: Do an authentication

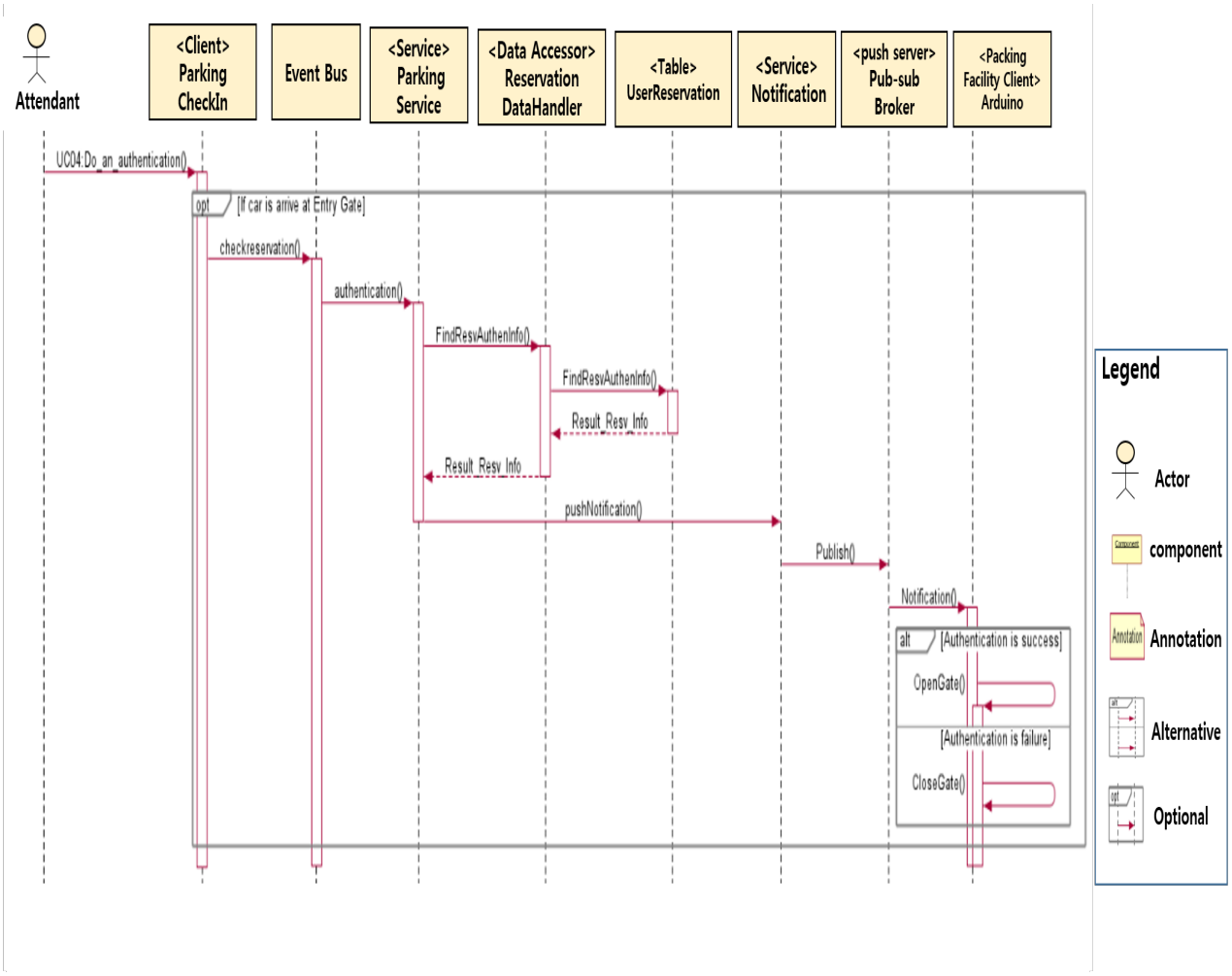

Figure 14. Sequence diagram of UC 04

This is a sequence diagram of UC04. The main actor is attendant and this sequence is executed by web-based application that is called monitoring system. Attendant can check the status of parking facility in real time. In this sequence, the attendant should input the authentication number received from car driver to enter into the parking facility. If the car driver who request to open gate is valid, the push server sends the “openGate” message to Arduino. Otherwise, the push server sends the “closeGate” message to Arduino.

E. Use case 05: Pay a parking fee

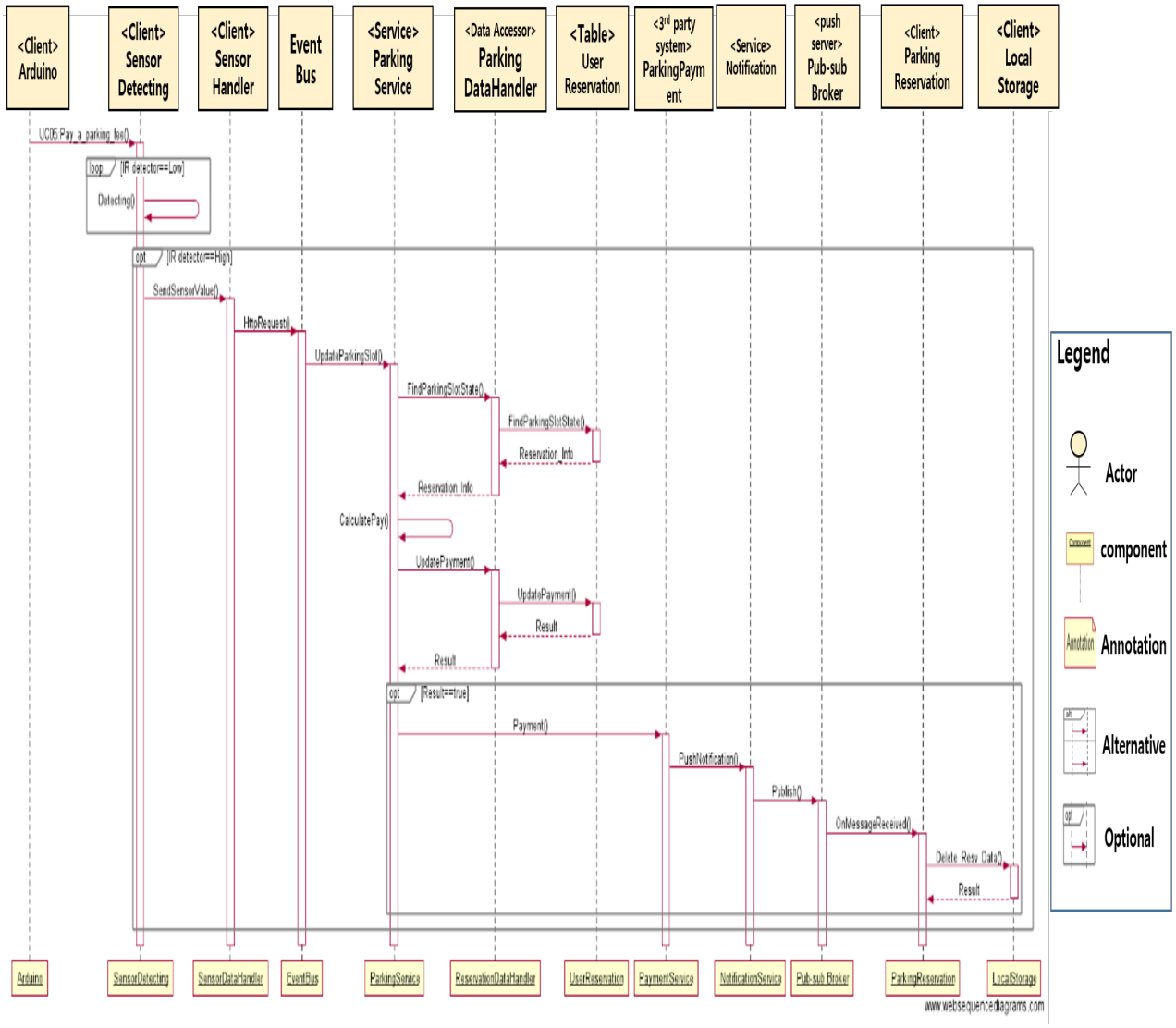

Figure 15. Sequence diagram of UC 05

This is a sequence diagram of UC05. This sequence is started from Arduino. Arduino should continuously detect whether the car come out or not because we should calculate the parking fee. If the Arduino detect the movement that the car is nearby exit gate, then the Arduino tries to communicate with server and then the server calculate the parking fee and request to send the payment information to push server. Finally if the car driver comes out to parking facility, the car driver get the message of payment information.

F. Use case 06: Get reallocation

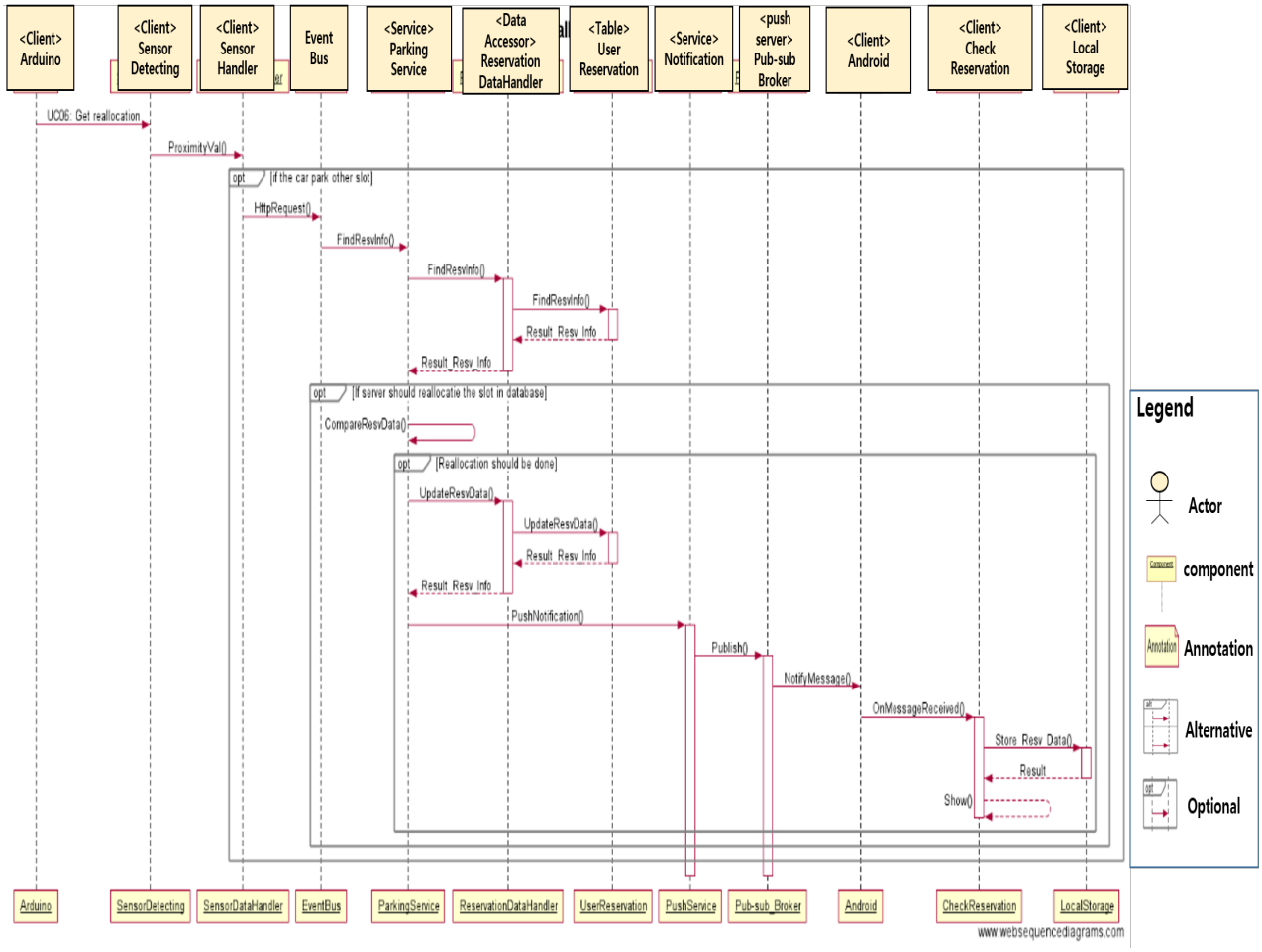

Figure 16. Sequence diagram of UC 06

This is a sequence diagram of UC06. This sequence is started from Arduino. Arduino should continuously detect whether the car come out or not because we should calculate the parking fee. If the Arduino detect the movement that the car is nearby exit gate, then the Arduino tries to communicate with server and then the server calculate the parking fee and request to send the payment information to push server. Finally if the car driver comes out to parking facility, the car driver get the message of payment information.

G. Use case 07: Monitor parking facility

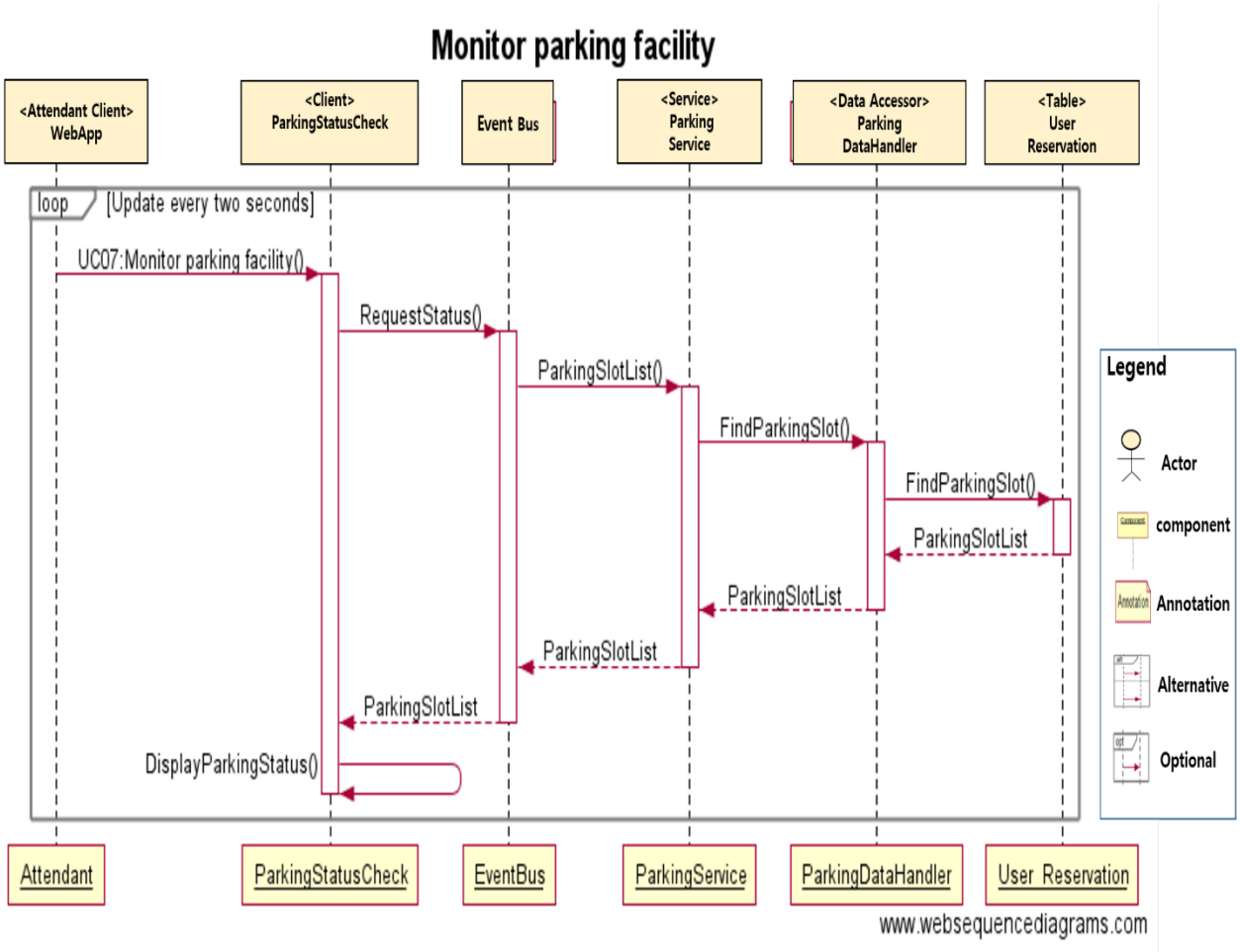

Figure 17. Sequence diagram of UC 07

This is a sequence diagram of UC07. In the description document, it is described that the attendant should monitor the parking facility. Firstly, Arduino sends the parking facility status to server and the server should store the updated information into Database. Then monitoring system can see the updated information in real time by requesting the parking status periodically to server.

F. Use case 08: Check Statistic data

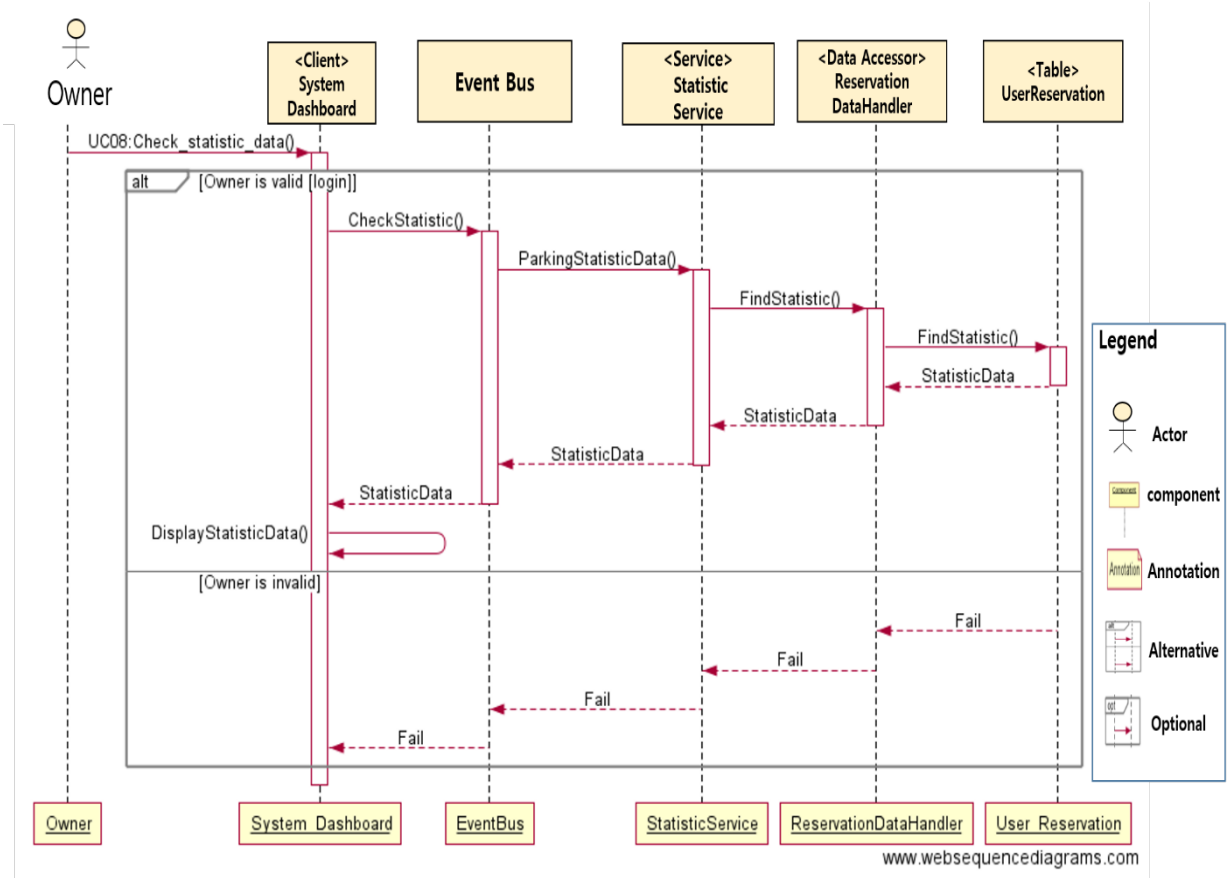

Figure 18. Sequence diagram of UC 08

This is a sequence diagram of UC08. The main actor is owner in sequence. In the description document, it is described that the only owner should check the statistic. So we implement the login function for only owner. If the owner is valid, then owner can see the statistical data of parking facility. In order to get the statistical data, the management system for owner should get the information from “userReservation” table.

4. Project plan

4.1 Development process

Basically, we set our development process with openUP (Unified Process). And also, we tailored openUP to our limited time (5 weeks). We chose this process based on the following reasons.

Firstly, we have a very limited time, and inevitably we should proceed both development and documentation with iterations (which are small-waterfalls). In every iteration, our team should develop all the artifacts (such as requirements, design, test case, code, and etc.) incrementally. Because openUP is a generic process model that is organized into phases(inception, elaboration, construction, transition) but separates activities (requirements, analysis and design, etc.) from these phases, we are able to execute all the process by small iteration, and also create each iteration's artifacts.

Secondly, openUP is basically used with tailoring it to the development team's own process. Therefore, we are able to be relatively free from the rules of the specific process. To overcome unstable agreement of requirements, and also frequent change, we need to be not restricted by formal process.

In principle, openUP is fundamentally based on the following figure.

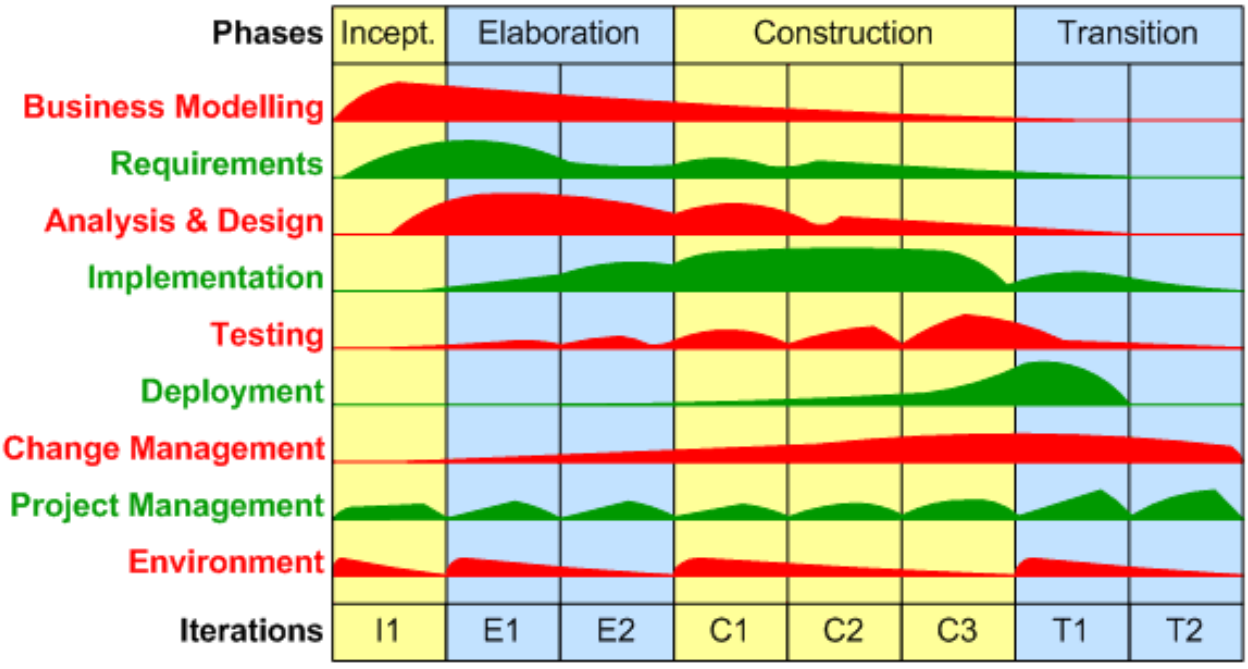

Figure 19. The general openUP process

Our team should tailor this process to our project. Horizontal axis means the flow of time. Basically, it concludes inception phase, elaboration phase, construction phase, and transition phase. Also, there are iterations which are smaller than phase. In other words, within each phase are a

number of iterations. Each iteration follows small-waterfall. An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in some artifacts.

Vertical axis means workflows of each iteration. Workflow is the sequence of activities performed in a team that produces a result of observable value to the customers (hence it is incremental). In an iteration, a team walk through the selected workflows. We are going to trim these workflows to do only really needed ones. The early iterations focus on modeling, requirements elicitation, or analysis & design. The late iterations focus on testing or deployment. Also, project management and change management always happen.

The followings are concise explanation of each phase.

- Inception phase: A team establishes the business case for the system.
- Elaboration phase: A team develops an understanding of the problem domain and the system architecture.
- Construction phase: A team does system design, programming and testing.
- Transition phase: A team deploys the system in its operating environment.

To sum up, in the early time of project, we can focus on requirements engineering or system design. However, we can proceed coding concurrently. In the late time of project, we can focus on implementing and testing, but we can also do requirement refinement (maybe mainly due to change request or unclear requirements).

The following figure is the tailored process of openUP to our team.

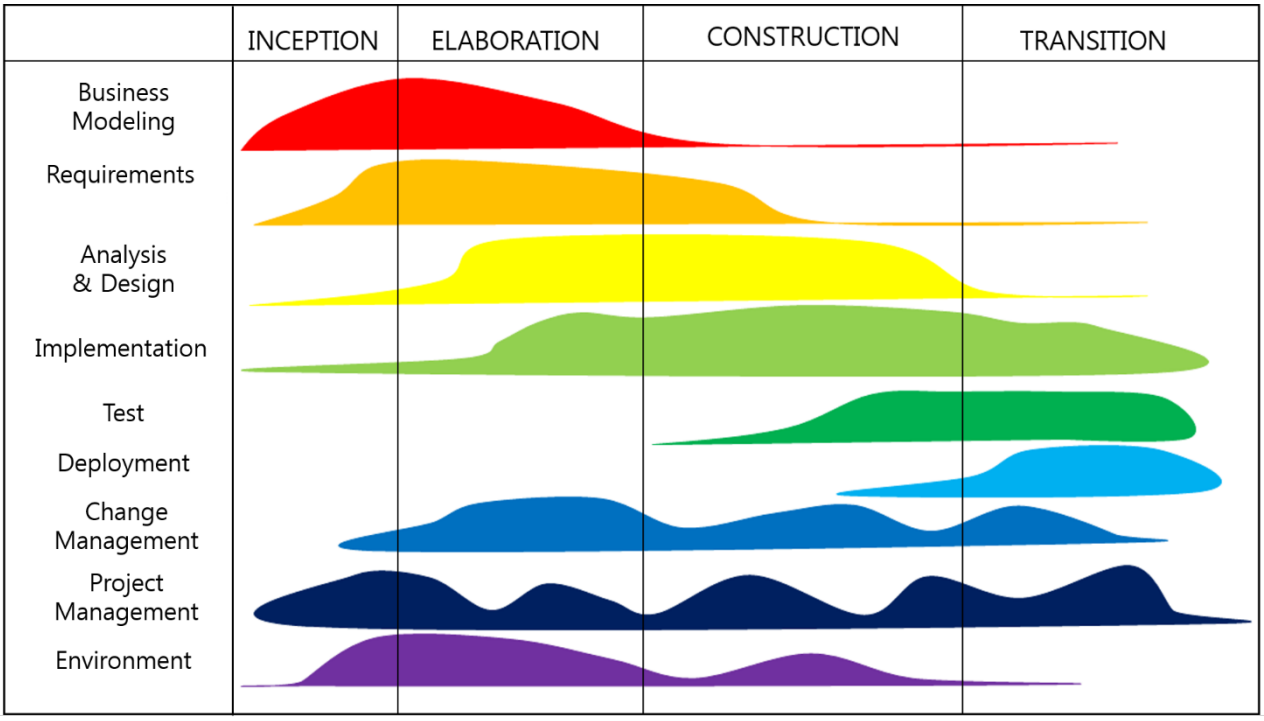

Figure 20. Tailored openUP process of our team

The above figure shows the weighted amount of each workflows (vertical axis). Of course, we are going to analyze project description and stakeholder (Tony)'s needs firstly. And also, we will firmly

set our development environment and project environment initially. Inevitably, project management works will bother us throughout the process. There will be also needs of change after the setting of requirements (mainly due to unclear requirements). Implementation and development will last until the close of project, because we do not have enough time. We would struggle with the error and bugs until the last moment. Testing will be started in relatively late phase.

And also, we can get back to any phase of this whole process, iteratively. It means there is no deterministic sequence of development. Hence, we can be relatively free from process, and we can revise all the documents and codes flexibly. It is represented in the following figure.

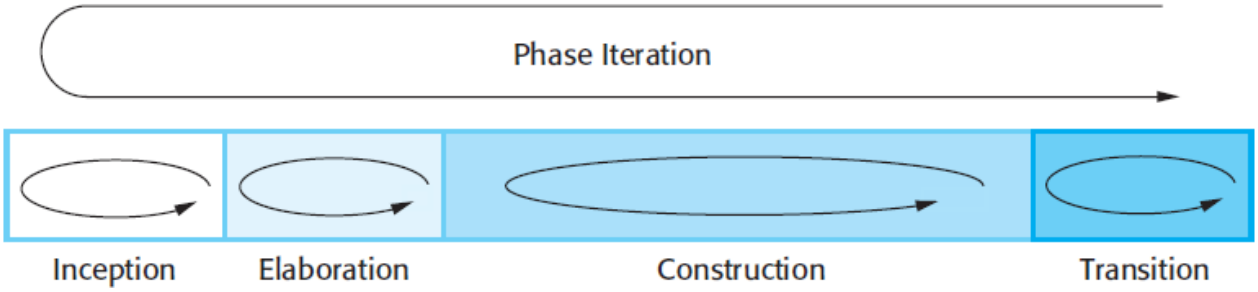

Figure 21. The concept of Iteration

All the phase can be reverted, and it counterbalances unstableness of the project.

As for tailoring, the concept of tailoring is shown below. It shows the big picture of tailoring.

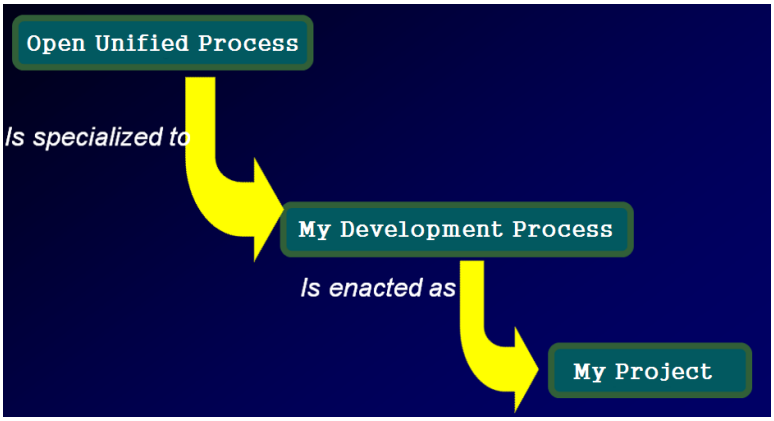

Figure 22. Tailoring openUP to our process

This figure shows the concept of what our team have tried to tailor openUP. Basically, our project is based on openUP (which is iterative, incremental).

And also, there are some key milestones (which are the related artifacts) between the phases in this process. We set our key artifacts between the phase like the below.

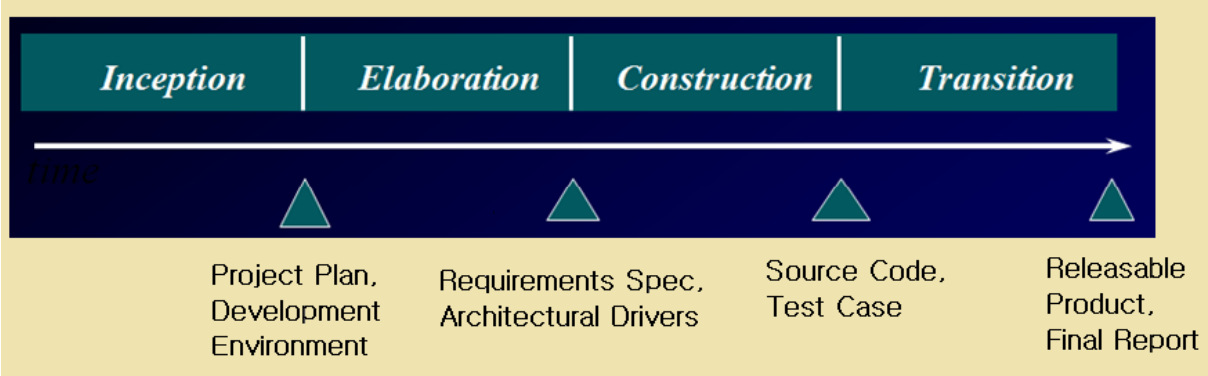

Figure 23. Key milestones between phases

Hereafter, we concisely enumerate some software artifacts, and these are not unchangeable milestones. However, we can modify these artifacts in the late phase. This is because we are on iterative and incremental process. After the inception phase, we need to have a whole and rough plan. Hence, we decided that project planning document and development environment configuration is to be made. And also after the elaboration phase, we would have requirements specification document and crucial architectural drivers. It means that we should identify, refine, and document requirements. Also, we have to prioritize requirements and constraints, so that key architectural drivers should be detected. However, we can modify our requirements slightly later. Of course after the construction phase, we should have source code, and initial test case. Although we should have executable source code and testing plan, it does not mean the close of development. And also, source code can be revised in transition phase. When the project is coming to an end, we are going to be with the perfect product and final report document & PowerPoint slides.

4.2 Project plan

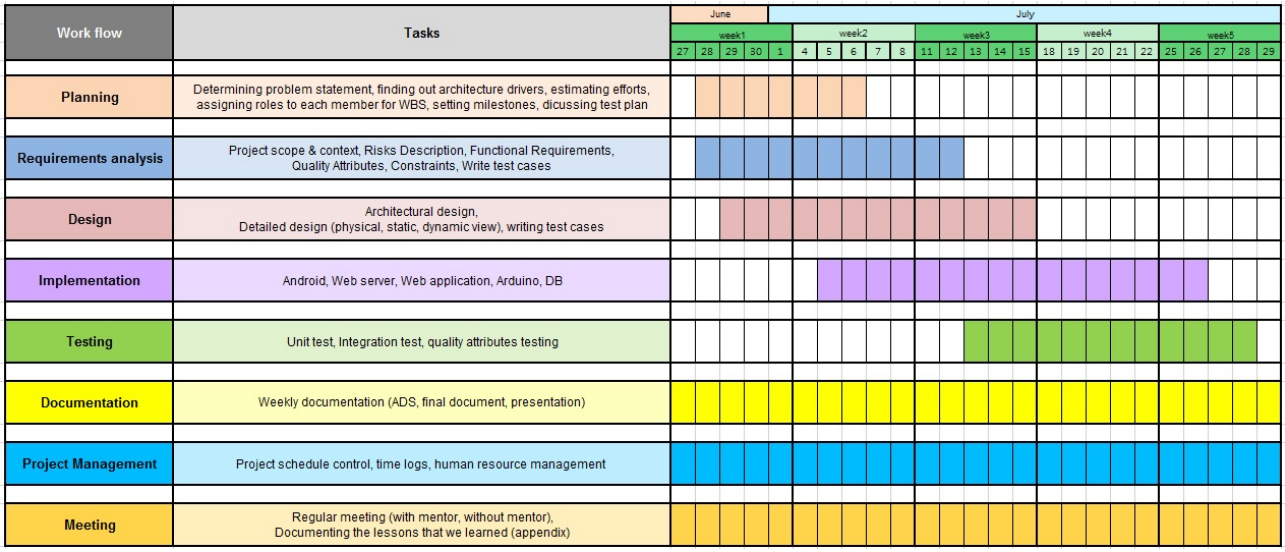

Figure 24. Project schedule

We have planned our overall schedule that is harmonized with our tailored openUP process. Although all the plan is set beforehand of actual design & implementation, our schedule is flexible and can be modified. And also, we will iterate all of the workflows. Therefore, at a specific time, we will focus on the designated workflow, but we can perform other workflows too. To sum up, in the early time of project, we can focus on requirements engineering or system design. However, we can proceed coding concurrently. In the late time of project, we can focus on implementing and testing, but we can also do requirement refinement (maybe mainly due to change request or unclear requirements). In other words, we can go back to the former works. If there are a lot of change requests, we regularly refine our all the related software artifacts, such as requirements documents, design artifacts, source code, etc.

This is mainly because we have an extremely limited time, and inevitably we should proceed both development and documentation with iterations (which are small-waterfalls). Our team should develop all the artifacts (such as requirements, design, test case, code, and etc.) incrementally. Our team is trying to be relatively free from the formal rules of the specific process.

Details of project plan are attached in the excel file. It includes:

- Project wiki
- The estimates of the efforts (PV)
- Separate views of Actual Cost (AC) according to roles, processes
- Plan table decomposed by WBS (Work Breakdown Structure)
- EVM chart that includes PV, AC, EV

4.3 Role assignment

Roles		
Member ID	Name	ACDM Role
DH	Dongho Lee	Managing engineer
DS	Daesoon Kim	Quality process engineer
MK	Myoungki Hong	Production engineers
SH	Sunghoon Byun	Requirements engineer
KS	Kyeongseok Yang	Production engineers
WJ	Woojin Han	Chief architect
DJ	Dongjae Kim	Production engineers

In this part, we are going to mention the roles of team members. Every member has his role in the project. First of all, we would like to describe what each role actually means briefly.

- Managing engineer: The managing engineer is responsible for coordinating the overall system design and system development effort.
- Quality process engineer: The quality process engineer ensures that ACDM and other defined processes are followed as prescribed to ascertain project quality goals.
- Requirements engineer: The requirements engineer leads the effort to gather and document the architectural drivers.
- Chief architect: The chief architect is responsible for overall system design.
- Production engineer: There are team members whose focus is on detailed design implementation of the architectural elements, and integration of the elements to compose the system.

We have three production engineers, who are Myoungki Hong, Kyeongseok Yang, and Dongjae Kim. This is because we are not all familiar with this project environments, so we need to focus on learning the related techniques. Dongho Lee is a leader of our team, and he will play a role of overall project coordinator. Daesoon Kim, who is a quality process engineer, will work with Sunghoon Byun, who is a requirements engineer, to coordinate the architecture design reviews and in planning product or system tests. And also, Sunghoon Byun will manage the change and evolution of the architectural drivers throughout the system or product life cycle. Lastly, Woojin Han, who is a chief architect, will work with all of the other members of the design team to coordinate the system design, beginning with gathering the architectural drivers, designing the architecture, reviewing it, refining it, and documenting it until production and deployment throughout the system or product life cycle.

4.4 Project risks

4.4.1 Risks identification

Title of risk : Learning new techniques causes project delay and cost overrun.	**ID** : R-01
Context	While progressing project, we are likely to face some techniques that we are not familiar with. This situation may lead to delays in progressing project
Condition	We face the situation that we should apply techniques that we are not familiar.
Consequence	Our project get delayed because we waste our time on learning new techniques.
Mitigation	We should study necessary techniques in advance.

Title of risk : Inefficiency of communication and project management.	**ID** : R-02
Context	If there is not an efficient way & tool of management, inefficient communication and time loss can arise.
Condition	While doing a project, we undergo inefficient management of overall project and wasteful communication time between team members, without the formal way or rules of those things
Consequence	Because we are on inefficient management, we might waste time that is related to communication and works.
Mitigation	We should be equipped with good rules or method (we can use some Project Management tools, such as Redmine).

Title of risk : Unfamiliar environment.	**ID** : R-03
Context	Because we are supposed to perform our project in strange environments, we would have trouble in adjusting to the new environments.
Condition	We have arrived in Pittsburgh from Korea.
Consequence	We have some troubles, such as the time difference or all the other living problems.
Mitigation	We think the problems will be resolved by themselves in the course of project.

Title of risk : Difficulties in English.	**ID** : R-04
Context	In the Pittsburgh, we should talk to professors in English.
Condition	We take classes in English, or we listen to what our mentor says in English.

Consequence	Classes in CMU and advices of mentor are too difficult to understand for us.
Mitigation	We need to ask professors and mentor to speak slowly, or we need to practice English conversation skills.

Title of risk : Lack of change management system.	**ID** : R-05
Context	When change request occurs, or needs to change the direction of the project occur, those may delay our project schedule.
Condition	When we are doing project according schedule, change requests arise.
Consequence	Time is wasted, and schedules are delayed.
Mitigation	We should recognize that change requests inevitably occur, so we should plan change management strategies.

/

Title of risk : Architecture lacks flexibility.	**ID** : R-06
Context	If we have to modify or change some requirements, it could be quite hard to reflect changes in architecture, because of lack of architecture flexibility.
Condition	We recognize that architecture is lacking in flexibility.
Consequence	It is very difficult to reflect change requirement in our architecture, so we should put a lot of effort and spend additional time.
Mitigation	In the stage of design, we should endeavor to design flexible architecture by referring to the lecture and reading materials.

Title of risk : Architecture is not fit for purpose.	**ID** : R-07
Context	Hard to satisfy quality attribute with our architecture which is not fit for purpose, because of our inexperience.
Condition	We are having a hard time in making right architecture.
Consequence	It is hard to satisfy quality attribute, such as scalability and security with our architecture.
Mitigation	We should consult with mentor or Anthony to suit architecture to the goal of our project.

Title of risk : Legal & regulatory limitations impacts the project.	**ID** : R-08
Context	Because we did not know well open source license or law of parking facility, our project will not be able to be acceptable.
Condition	The issues of the rule in open source license or law of parking facility occur.

Consequence	Our established system is banned or not accepted.
Mitigation	We should consider the emerging issue inevitably, and it will cause some business problems.

Title of risk : Lack of project management knowledge.	**ID** : R-09
Context	Because of the lack of project management knowledge, project schedule is delayed or product quality decreases.
Condition	When doing a project, there is shortage of project management knowledge.
Consequence	Product quality decreases, and we waste more time on doing work.
Mitigation	We need to adopt a project management tool, and we need to follow the project management process.

Title of risk : Uncertainty of development process.	**ID** : R-10
Context	Because we have no individuals who has the knowledge of development process, we could have a difficulty in creating suitable product with right process. What was worse, it will be able to conclude in bad product.
Condition	No one of our members has knowledge of what process we should adopt, and we are not sure of our process.
Consequence	Inevitably, we have to spend more time in studying development process. Therefore, there is not enough time to progress project.
Mitigation	One of the team members should study development process hard, so he can lead team into the right process.

Title of risk : Team misunderstands requirements.	**ID** : R-11
Context	Because of misunderstanding requirements, we have a wrong direction toward the project goal.
Condition	Incorrect requirements are applied to the system.
Consequence	We have to spend more time in correcting project, or our implementation should be overturned.
Mitigation	In stage of requirements analysis, we should analyze and examine the customer's requirements perfectly.

Title of risk : Scope is ill-defined.	**ID** : R-12
Context	Because we have defined system scope too widely or narrowly or ambiguously, we should re-define scope suitably.
Condition	Project scope is ill-defined. It is too wide or too narrow or too ambiguous.

Consequence	We should spend additional time in re-defining project scope. It causes delay or additional cost.
Mitigation	In the stage of requirements analysis, we should analyze requirements exactly. In the stage of design, we should accurately define the scope with exact design artifacts based on the analyzed requirements.

Title of risk : Too much unplanned activities come about.	**ID** : R-13
Context	Due to many reasons, we encounter a lot of unplanned activities, and we are not capable to handle those things thoroughly. This situation leads to an unpredicted delay during project.
Condition	There are a lot of tasks that we did not plan beforehand.
Consequence	Unplanned activities lead us to delay or cost overrun.
Mitigation	We should assign one member a project manager, and he can control, manage, and plan our schedule. Manager help us to increase productivity by scheduling and prioritizing tasks. Also, he inserts time buffers of project beforehand.

4.4.2 Qualitative risk analysis

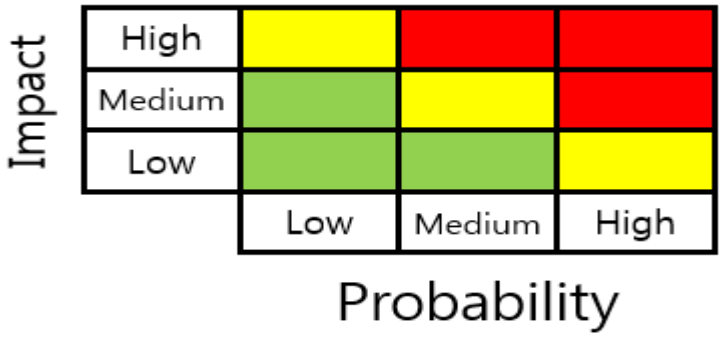

Figure 25. Risk matrix

Probability

- High – Grater than 70% probability of occurrence
- Medium – Between 40% and 70% probability of occurrence
- Low – Below 40% probability of occurrence

Impact

- High – Risk that has potential for greatly impacting project cost and project schedule
- Medium – Risk that has potential for moderately impacting project cost and project schedule
- Low – Risk that has potential for slightly impacting on project cost and project schedule

The probability of occurrence and impact of occurrence for each identified risk will be assessed by the members of team. It is based on discussion of the team members. To reach an agreement, we performed a wide-band Delphi method.

Risks that fall within the RED zone must be prevented, because risks of red zone can tremendously give damage to the project. The below table shows results.

Possible Risks	probability	Effects	Risk Level
Learning new techniques causes project delay and cost overrun	Low	Medium	
Inefficiency of communication and project management	Medium	Medium	
Unfamiliar environment	Medium	Low	
Difficulties in English	Medium	Medium	
Lack of change management system	Low	Medium	
Architecture lacks flexibility	Medium	High	
Architecture is not fit for purpose	Low	High	
Legal & regulatory limitations impacts the project	Low	High	
Lack of project management knowledge	Medium	Medium	
Uncertainty of development process	Medium	Low	
Team misunderstands requirements.	Medium	High	
Scope is ill-defined	Low	Medium	
Too much unplanned activities come about	Medium	High	

4.4.3 Risks management

Risks that fall within Green and Yellow zones are acceptable in our project. However, risks of red zone need management strategies to avoid serious damages. We established strategies and managed risks by applying management strategies.

Title of risk : Architecture lacks flexibility	**ID : R-06**
Reason of decision	While working on the project, change of requirements can happen frequently. If architecture is not flexible, we would have a big problem of adjusting our project to this situation.
Mitigation Strategy	In the stage of design, we should endeavor to design flexible architecture by referring to the lecture and reading materials.
The person in charge	Dongho Lee
Management results	Through the study on architecture, we can design the enough flexible architecture.

Title of risk : Team misunderstands requirements	**ID : R-11**
Reason of decision	The objective of project is to develop a system that customer wants. Misunderstanding customer's requirements cause a thoroughly invalid system.
Mitigation Strategy	In stage of requirements analysis, we should analyze and examine the customer's requirements perfectly.
The person in charge	Sunghoon Byun
Management results	Our team's requirement engineer did his works well, so we could have analyzed various requirements thoroughly.

Title of risk : Too much unplanned activities come about	**ID : R-13**
Reason of decision	We have limited time to develop system. Therefore, a slight delay in project can be fatal. At worst, we might not be able to complete our project. This is a very important matter.
Mitigation Strategy	We should assign one member a project manager, and he can control, manage, and plan our schedule. Manager help us to increase productivity by scheduling and prioritizing tasks. Also, he inserts time buffers of project beforehand.
The person in charge	Kyeongseok Yang
Management results	Suitable time buffer helped us to overcome unplanned activities.

5. Testing strategy and results

5.1 Testing objectives

Testing part is for assessing the value of our 'Sure-park' system. We are trying to mention the following objectives of testing:

1) To detect & modify defects in the software
2) To check whether the software meets the specified requirements or not
3) To lower the risks of project failure

5.2 Testing scope

Hereafter, we introduce an outline that describes the testing approach of the software development cycle. We are planning to perform code review, code coverage testing, use case testing, and quality attributes testing. And also, unit testing will be performed by developers on their own judgement. It is based on the developers' experiences and abilities. We think that code review will be effective in finding mistakes. And also, we will perform black-box testing which includes use case testing and quality attributes testing. After all the implementation is completed, we will perform use case testing based on our use cases. Lastly, we are going to perform quality attributes testing under the supervision of all the team members for making consensus. Although the testing scope is not broad, we think it is adequate for our project because we have limited time (5 weeks). We decided to perform not that much tasks but only essential testing. In other words, we thought it is wise to concentrate on use case testing and quality attributes testing rather than unit testing. The followings are the detailed plan of each testing:

- Code review: Code review is systematic examination (sometimes referred to as peer review) of source code. It is intended to find mistakes overlooked in the initial development phase, improving the overall quality of software. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections. Although we assign separate developers separate parts of system, we are planning to discuss code together.
- Code coverage testing: Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage, measured as a percentage, has had more of its source code executed during testing which suggests it has a lower chance of containing undetected software bugs compared to a program with low code coverage. We will use EclEmma, and it is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. It brings code coverage analysis directly into the Eclipse workbench.
- Use case testing: Use case testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after static testing and before quality attributes testing. Use case testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an use case test plan to those aggregates, and delivers as its output the integrated system ready for system

testing. We will perform this testing based on our use case, and it will be performed with all the team members' presence.

- Quality attributes testing: Quality attributes testing is conducted on a complete and integrated system to evaluate the system's compliance with its specified requirements. It is almost same with user acceptance testing, because we will finally determine our system's value. We will validate our system's acceptance based on quality attributes.

5.3 Testing elements lists

We determined what we are going to actually perform in the testing part. The lists below enumerates those items (use cases, quality attributes) that are meaningful to assure value of our system. Details on each testing element will be articulated in the testing results part.

5.3.1 Use case testing elements

Number	Elements
1	Reservation function: Use Case 01
2	Check Reservation with Server: Use Case 02
3	Check Reservation without Server: Use Case 02
4	Cancel Reservation: Use Case 03
5	Do an Authentication: Use Case 04
6	Pay a parking fee: Use Case 05
7	Get reallocation: Use Case 06
8	Monitor parking facility: Use Case 07
9	Check a statistic data: Use Case 08

5.3.2 Quality attributes testing elements

Number	Elements
1	Scalability

5.4 Testing results

5.4.1 Code coverage testing

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
spark_web	77.7 %	3,634	1,045	4,679
src/main/java	77.7 %	3,634	1,045	4,679
com.spark_web.dao	55.9 %	964	759	1,723
ResvDAO.java	61.4 %	602	379	981
ParkingslotDAO.java	47.9 %	135	147	282
AdminDAO.java	48.1 %	103	111	214
ParkingfacilityDAO.java	51.4 %	75	71	146
HeartbeatLogDAO.java	49.0 %	49	51	100
com.spark_web.service	92.6 %	1,622	130	1,752
ParkingSerivice.java	91.0 %	666	66	732
PushNotificationService.java	64.1 %	93	52	145
LoginService.java	94.0 %	94	6	100
ReservationService.java	95.2 %	119	6	125
ConfigurationService.java	100.0 %	49	0	49
PaymentService.java	100.0 %	13	0	13
StaticalDataService.java	100.0 %	588	0	588
com.spark_web.util	73.8 %	237	84	321
AesUtil.java	53.6 %	81	70	151
SHA256.java	84.6 %	44	8	52
AES256Cipher.java	97.0 %	97	3	100
JsonUtil.java	83.3 %	15	3	18
com.spark_web.domain	84.9 %	298	53	351
ParkingSlot.java	66.7 %	30	15	45
Send_Resv_Info.java	50.0 %	12	12	24
Admin.java	70.8 %	17	7	24
Resv.java	90.4 %	66	7	73
Pair.java	73.9 %	17	6	23
Result_Resv_Info.java	86.7 %	39	6	45
HeartbeatLog.java	100.0 %	31	0	31
ParkingFacility.java	100.0 %	31	0	31
ParkingStaticalData.java	100.0 %	38	0	38
SlotUsage.java	100.0 %	17	0	17
spark_web	96.4 %	513	19	532
WebService.java	96.4 %	513	19	532

Figure 26. Code coverage tool

We performed code coverage, and it showed 77.7% of total coverage. This is because we implemented exception handling, but we could not make exception situation code totally. Therefore, this tool decided that exception handling code is not covered by code coverage. And so, about 20% is shown to be ‘not covered’. We think it was inevitable, because we did not have enough time.

5.4.2 Use case testing

In this section, we enumerated the actual functionality test cases which is based on our use cases, as shown below:

TEST INFORMATION					
Test Case ID:	TC_FR_01		**Related Use Case**	**UC 01**	
Used entities	Car driver, Android, Server, Parking reservation application (Android), and Database,				
Test Description	Test the reservation function in Android application				
Test Priority	High				
Tester's name	Dongho Lee	**Date Tested**	July 22th	**Test result**	**pass**
Test Scenario	Car driver enters the valid reservation time, phone number, credit card number, parking facility ID, and then the driver can reserve the parking slot and check whether the reservation trial is successful or not.				
#	**Pre-conditions:**	**#**	**Test Data**		
1	A car driver entered the reservation time, credit card number (16 ciphers) and phone number (9~12 ciphers).	**1**	Reservation_time = 22:50		
2	Server should be connected to Android.	**2**	Credit_num = 1234567812345678		
3	Both Android and server must use the same Wi-Fi router.	**3**	Phone_num = 4124252315		

TEST STEPS				
#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	A car driver enters reservation time, phone number, credit card number, parking facility ID. Also, he Clicks reservation button.	Reservation screen on Android should show the dialog with result, such as "reservation is available".	The dialog of reservation result was shown with result, such as "reservation is available".	**pass**
2	The car driver clicks "OK" button in the dialog screen.	Android start communicating request of reservation with server.	Android communicated request of reservation with the serve by using HTTP.	**pass**
3	Server receives the request of reservation and if the reservation is available, then server return this information to Android.	Result of reservation should be received, and reservation information should be stored in local storage (shared preference).	Android received the reservation information, and stored reservation information in local storage (shared preference).	**pass**
4	Android screen is switched to reservation check screen.	Reservation check screen should show reservation information.	The car driver checked the reservation information on reservation check screen.	**pass**
Post-Condition:	The car driver received success information of reservation from sever, and store this information in Android local storage (shared preference). Also, he checked his reservation information on Android reservation check screen.			

TEST INFORMATION					
Test Case ID:	TC_FR_02			**Related Use Case**	**UC 02**
Used Entities	Car driver, Parking reservation application (Android), Server, and Database				
Test Description	Test the checking reservation function in Android application				
Test Priority	High				
Tester's name	Dongho Lee	**Date Tested**	July 22th	**Test result**	**pass**
Test Scenario	Android requests server to check the reservation, and gets the reservation information, and show them on reservation check screen.				

#	**Pre-conditions:**	**#**	**Test Data**
1	Reservation information should be stored in server DB.	**1**	Phone_num = 4124252315
2	Server should be connected with android.	**2**	CheckFlag= true
3	Both android and server should use the same Wi-Fi router.		

TEST STEPS

#	Step	Expected Results	Actual Result	Status
1	Android requests server to check the reservation with phone number information.	Server receives the request, and returns the reservation information to Android.	Android received the reservation information.	**pass**
2	Server receives the request, and then sends reservation information which is got from server's database.	Android receives reservation information from server, and stores this information in local storage (shared preference).	Android received reservation information from server, and stored this information in local storage (shared preference).	**pass**
3	Reservation check screen on Android shows the reservation information.	Reservation information are shown in reservation check screen.	Reservation information were shown in reservation check screen.	**pass**
4	On the reservation check screen, we need to check whether the switch to from reservation check screen to the reservation screen is possible or not.	It should not be possible to switch from reservation check screen to the reservation screen is possible on Android.	It was not be possible to switch from reservation check screen to the reservation screen is possible on Android.	**pass**
5	When the car driver re-try to execute Android application, Android should server to check reservation information, and should show its information.	Android application should show the reservation information on the reservation check screen.	Android application showed the reservation information on the reservation check screen.	**pass**
Post-Condition:	The car driver checked his reservation information on Android reservation check screen.			

TEST INFORMATION					
Test Case ID:		TC_FR_03	**Related Use Case**	**UC 02**	
Used Entities		Car driver, Parking reservation application (Android), Server, and Database			
Test Description		Test the check reservation function in Android Application without server			
Test Priority		High			
Tester's name	Dongho Lee	**Date Tested**	July 22th	**Test result**	**pass**
Test Scenario		Android requests server to check his reservation, but it fails because server is not working. Therefore, Android gets the reservation information from local storage (shared preference) and shows this information on the screen.			

S#	**Pre-conditions:**	**S#**	**Test Data**
1	Reservation is already finished.	**1**	Phone_num = 4124252315
2	Reservation information should be stored in the local storage (shared preference).	**2**	CheckFlag= true
3	Server is not connected with Android.	**3**	Result= fail

TEST STEPS

#	Step	Expected Results	Actual Result	Status
1	Android requests server to check his reservation information by sending his phone number.	Android cannot be connected with server.	Android was not connected with server.	**pass**
2	Android gets the reservation information from its local storage (shared preference).	Reservation information which is from local storage should be allocated to Android text views on the screen.	Reservation information which is from local storage was allocated to Android text views on the screen.	**pass**
3	Android shows the reservation information on reservation check screen.	Reservation information is shown on the screen.	Reservation information was shown on the screen.	**pass**
4	On the reservation check screen, we need to check whether the switch to from reservation check screen to the reservation screen is possible or not.	It should not be possible to switch from reservation check screen to the reservation screen is possible on Android.	It was not be possible to switch from reservation check screen to the reservation screen is possible on Android.	**pass**
5	When the car driver re-try to execute Android application, Android should server to check reservation information, and should show its information.	Android application should show the reservation information on the reservation check screen.	Android application showed the reservation information on the reservation check screen.	**pass**

Post-Condition:	Trial for communication between server and Android fails. Android gets the reservation information from its local storage, and Android shows the reservation information on reservation check screen.

TEST INFORMATION					
Test Case ID:	TC_FR_04		**Related Use Case**	**UC 03**	
Used Entities	Parking reservation application (Android), Car Driver, Server, and Database				
Test Description	Test the cancel reservation function in Android application by communicating with server				
Test Priority	High				
Tester's name	Dongho Lee	**Date Tested**	July 22th	**Test result**	**pass**
Test Scenario	A car driver clicks the cancel button, and get the response of server.				

#	**Pre-conditions:**	**#**	**Test Data**
1	Reservation information should be stored in server database.	**1**	Authen_num= "value from server"
2	Server should be connected with Android.		
3	Both Android and server should use the same Wi-Fi router.		
4	The parking status is "Reserved" in database table		

TEST STEPS

#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	The car driver clicks cancel button.	The reservation cancel process should be executed.	Reservation cancel function was called.	**pass**
2	Server receives the request of reservation cancel, and the server apply this information to DB table.	In the reservation table, the status of reservation is changed to "cancel", and the server should send the result of cancel, such as "success".	In the reservation table, the status of reservation was changed to "cancel", and the server sent the result of cancel to Android. Android received the "success" from server.	**pass**
3	Reservation information stored in local storage is deleted.	The reservation information that is stored in local storage should be deleted.	The reservation information that was stored in local storage was deleted.	**pass**
4	Android shows the dialog of cancel result on the screen.	Android shows the result ("success") of reservation cancel.	Android showed the result ("success") of reservation cancel.	**pass**
5	When the car driver re-try to execute Android application, Android should server to check reservation information, and should show its information.	Android application should show the reservation information on the reservation check screen.	Android application showed the reservation information on the reservation check screen.	**pass**

Post-Condition:	After pressing the reservation cancel button, the car driver should get the result of reservation cancel from server. Also, Android creates the dialog of this information on the screen.

TEST INFORMATION					
Test Case ID:	TC_FR_05		**Related Use Case**	**UC 04**	
Used Entities	Arduino, Server, Database, Attendant, and Authentication system (attendant's web), Car driver				
Test Description	Test the authenticating the car driver function using authentication number				
Test Priority	High				
Tester's name	Woojin Han	**Date Tested**	July 22th	**Test result**	**pass**
Test Scenario	The attendant authenticates the car driver by entering authentication number in parking management system				

#	**Pre-conditions:**	**#**	**Test Data**
1	The car (driver) approaches to the parking facility.	**1**	EntryBeamState=low
2	Server receives the information of car approach from Arduino.	**2**	StallSensorValue<15
3	The attendant should enter the authentication number into authentication system (attendant's web).	**3**	authen_num= "value from server"
4	Authentication number is valid.	**4**	

TEST STEPS				
#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	Arduino recognizes if the car approaches to the entry gate, and notify the information to server.	The server should receive the information that the car has approached to the entry gate.	The server received the information that the car had approached to the entry gate.	**pass**
2	Server requests the attendant to enter the authentication number into the authentication system.	The attendant should enter the authentication number into the authentication system correctly.	The attendant entered the authentication number into the authentication system correctly.	**pass**
3	The authentication system sends the authentication number to the server, and the server checks whether it is valid or not.	The server should receive and check the authentication number.	The server received the authentication number, and verified whether it is valid or not.	**pass**
4	Server requires 'Push Server' to require Arduino to open the entry gate.	'Push Server' should receive the request of opening the gate from the server, and send this request to Arduino.	'Push Server' received the request of opening the gate from the server, and sent this request to Arduino.	**pass**
5	Arduino open the entry Gate	The gate entry should be opened.	The gate entry was opened.	**pass**
Post-Condition:	Arduino recognized the approach of the car, and opened the entry gate.			

TEST INFORMATION					
Test Case ID:	TC_FR_06		**Related Use Case**	**UC 05**	
Used Entities	Arduino, Parking reservation application (Android), Server, Car driver, and Database,				
Test Description	Test the payment of parking pee function				
Test Priority	High				
Tester's name	Myoungki Hong	**Date Tested**	July 23th	**Test result**	**pass**
Test Scenario	When a car driver gets out of the parking facility, he will be automatically charged with their credit card information that was inputted during reservation				

#	**Pre-conditions:**	**#**	**Test Data**
1	The car should be parked in a specific parking slot.	**1**	StallSensorVal<15
2	Both Arduino and Android should be connected with server.	**2**	StallSensorState=1
3	The car should approach in front of the exit gate.	**3**	ExitBeamState=Low
4	The exit gate should be opened.	**4**	

TEST STEPS

#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	The car leave the his parking slot.	Parking slot LED should be changed to "Red".	Parking slot LED was "Red".	**pass**
2	The exit gate sensor detects the car's approach.	Exit gate LED should be changed to "Green, and it is opened.	Exit gate LED was "Green", and it was opened, so the car can get out of the parking facility.	**pass**
3	Arduino requests the server to calculate the parking fee.	The server should calculate the parking fee from 'start time' to 'exit time'.	The parking fee was calculated.	**pass**
4	The server requests 'Push Server' to publish notification message.	The 'Push Server' should send the notification to the car driver (android).	Android received the notification about the parking fee.	**pass**
Post-Condition:	The car driver has been got out of the parking facility. The android got the notification message about the payment from the server.			

TEST INFORMATION					
Test Case ID:	TC_FR_07			**Related Use Case**	**UC 06**
Used Entities	Arduino, Parking reservation application (Android), Server, Database, and Car Driver				
Test Description	Test whether the notification of reallocation operates well or not				
Test Priority	Medium				
Tester's name	Dongjae Kim	**Date Tested**	July 23th	**Test result**	**pass**
Test Scenario	When a car driver parks his car into the other driver's slot, system reallocates the slots.				

#	**Pre-conditions:**	**#**	**Test Data**
1	The driver has done a parking reservation through Parking reservation application (Android).	**1**	Phone number, credit card information
2	Server already stored the reservation data in the database.	**2**	Reservation data
3	More than two slots in the parking facility should be reserved	**3**	Number of reserved slots

TEST STEPS				
#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	The car driver enters into the parking facility, and he parks his car into the other driver's parking slot.	Arduino sends the incorrectly parked slot information to the server.	Arduino sent the incorrectly parked slot information to the server.	**pass**
4	Server tries to reallocate the parking slots.	Server reallocates the parking slots. The parked slot is allocated to the present driver, and the original slot of the present driver is allocated to the not arrived car driver.	Server reallocated the parking slots. The parked slot is allocated to the present driver, and the original slot of the present driver was allocated to the not arrived car driver.	**pass**
5	Server requests 'Push Server' to send the notification to the car drivers.	The current driver in the parking facility and the not arrived driver receive notification of reallocation.	The current driver in the parking facility and the not arrived driver received notification of reallocation.	**pass**
Post-Condition:	The car drivers checked the reallocation notification on parking reservation application (Android),			

TEST INFORMATION					
Test Case ID:	TC_FR_08			**Related Use Case**	**UC 07**
Used Entities	Parking monitoring system (attendant web), Server, Database, and Attendant				
Test Description	Test the monitoring parking facility status function				
Test Priority **(Low/Medium/High)**	Medium				
Tester's name	Woojin Han	**Date Tested**	July 24th	**Test result**	**pass**
Test Scenario	Attendant monitors the parking facility reservation status.				

#	**Pre-conditions:**	**#**	**Test Data**
1	Parking monitoring system is able to request the facility status data at every 2 seconds.	**1**	Parking facility status data
2	Parking monitoring sends signal to the server, and checks whether the server is alive or not.	**2**	Server ping data

TEST STEPS				
#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	The attendant accesses the system.	The attendant can check the parking facility status data on the parking monitoring system (attendant web).	The attendant checked the parking facility status data on the parking monitoring system (attendant web).	**pass**
2	Parking monitoring system requests the facility status data at every 2 seconds.	Parking monitoring system receives facility status data at every 2 seconds.	Parking monitoring system received facility status data at every 2 seconds.	**pass**
3	The attendant checks the reservation data.	The attendant checks the reservation data.	The attendant checked the reservation data.	**pass**
Post-Conditions:	The attendant checked the facility reservation data by using parking monitoring system.			

TEST INFORMATION					
Test Case ID:	TC_FR_09			**Related Use Case**	**UC 08**
Used Entities	Owner, Server, Database, and Parking management system				
Test Description	Test whether the owner can check the statistical data or not				
Test Priority (Low/Medium/High)	Medium				
Tester's name	Woojin Han	**Date Tested**	July 24th	**Test result**	**pass**
Test Scenario	Owner monitors the parking facility statistical data by using the parking management system.				

S#	**Pre-conditions:**	**S#**	**Test Data**
2	Owner knows his ID and password for login to the parking management system.	**2**	Owner id, Owner password

TEST STEPS				
#	**Step**	**Expected Results**	**Actual Result**	**Status**
1	Owner tries to access the parking management system	Owner succeed in login to the parking management system.	Owner succeeded in login to the parking management system.	**pass**
2	Owner click the statistical tab on the parking management system screen.	The server returns the statistical data to the owner. Parking management system shows the facility statistics, such as peak time, revenue, slot usage, user average usage information on the screen.	Parking management system showed the facility statistics, such as peak time, revenue, slot usage, user average usage information on the screen.	**pass**
Post-Condition:	Owner was able to monitor the parking facility statistical data by using the parking management system.			

5.4.3 Quality attributes testing

Our team decided that it is enough to apply our design method with regard to security and availability. Therefore, we only described the testing results for scalability. Also, we made a list of our design decisions of security and availability.

[Availability]

Name	Design decisions description
Backup DB	We made a backup database. Because we stores information of main database into backup database periodically, we can restore fast through backup DB whenever database does not work.
Shared preferences	We used 'shared preferences' in Android. It is a XML-based local storage. Even though Android cannot communicate with server, Android can get data from its local storage
Heartbeat pattern	We used heartbeat pattern. Whenever malfunction of Arduino occurs, we should fix it as soon as possible. Therefore, we should check periodically whether Arduino works normally or not. To do this, we used heartbeat pattern. Arduino sends its status to server every 30 minutes. The systems that are directly related to human safety should be checked every seconds, but our system has no such a safety-critical characteristics. Therefore, we decided that 30 minutes is acceptable to our system.
Ping & echo pattern	We used ping & echo pattern. Its purpose is to fix server as soon as possible in case it undergoes failure. Attendant's parking monitoring system checks the status of server every two seconds. In detail, attendant tries to require server to return parking slot status every two seconds. At this time, if there is no response from server, we can decide that there some problems with our server

[Security]

Name	Design decisions description
Authentication number	We perform an authentication step. If we perform an authentication step by using only credit card number or phone number, there may be security risks. Therefore, we give car drivers authentication number when car drivers make a reservation. Car drivers should show their given number to enter the parking facility.
AES256	When server and clients communicate, we transmit data which are encrypted with AES256. Therefore, we can protect these data. Also, server stores client's data encrypted with AES256. Therefore, it is less risky although this data is hacked by malicious users.
SHA256	When the owner tries to log in the owner's parking management system, we transmits owner's data by encrypting this data with random key. In other words, we use token. Therefore, we can protect the owner's information.

[Scalability]

In scalability testing, we will show performance with the number of requests by Arduino and Android.

In the Arduino performance test, our server handled Arduino's 75,000 requests in 130 seconds. Throughput was 547 per second, and average response time was 16 milliseconds.
In the Android performance test, our server handled Android's 300,000 requests in 9 minutes 30 seconds. Throughput was 561 per second, and average response time was 17 milliseconds.
It shows that this system supports more than 500 parking slots.

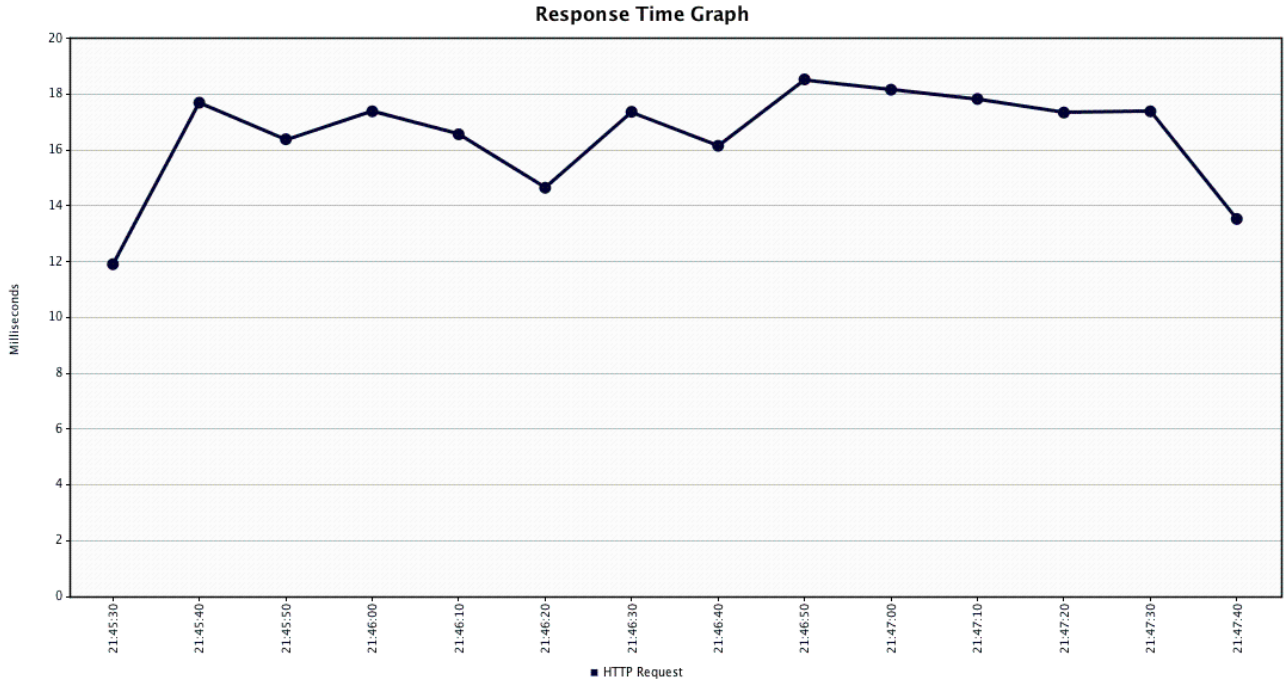

Figure 27. Arduino's average response time in 130 seconds

The above figure shows Arduino's average response time in 130 seconds.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	75000	16	4	3097	33.81	0.00%	547.7/sec	92.00	172.0
TOTAL	75000	16	4	3097	33.81	0.00%	547.7/sec	92.00	172.0

Figure 28. Arduino's throughput per second

The above figure show Arduino's throughput per second.

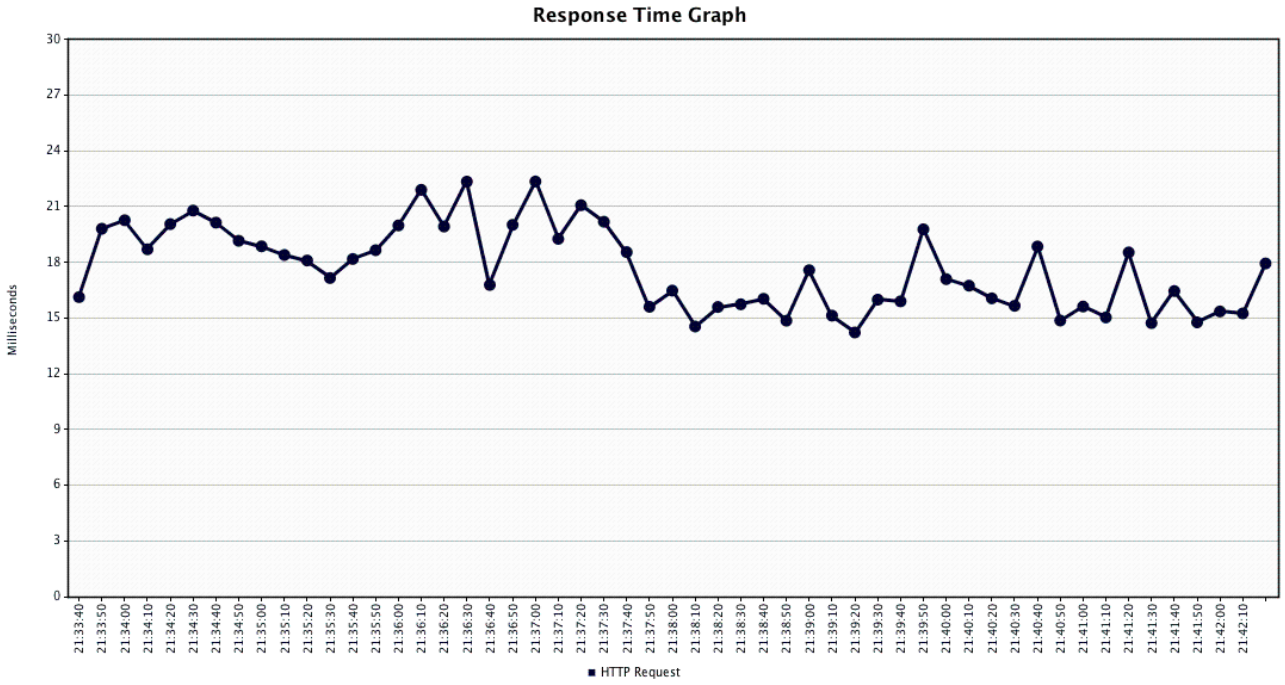

Figure 29. Android's average response time in 9min 30sec

The above figure shows Android's average response time in 9 minutes and 30 seconds.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	300000	17	3	3029	32.48	0.00%	561.3/sec	198.97	363.0
TOTAL	300000	17	3	3029	32.48	0.00%	561.3/sec	198.97	363.0

Figure 30. Android's throughput per second

The above figure show Android's throughput per second.

We could conclude that our system supports more than 500 parking slots, because server can handle more than 500 clients at once (specifically, we measure throughput per second).

Appendix A: Lessons Learned

We thought "Lessons learned" should include the following things:

- What went well?
- We could have planned to do something, what is the part we did not do?
- We could have done something, what is the part we did not do?
- What have we learned?
- What are the different things in implementation in comparison with our design?

There are three perspectives by which we classified our learned lessons. "Team", "Technical" and "Architecture".

[Team perspective]

- Time management problems
  - When we made a plan by considering limited time, we failed to distribute time appropriately. We could not spend much time in testing. Consequently, EV did not reach to PV in EVM Chart. Therefore, we learned that time planning is very difficult.

- Preliminary knowledge & Availability of team members
  - Because we did not have background knowledge of quality attributes, we had a hard time in eliciting and applying quality attributes. Also, we think we needed to know each team member's ability exactly. We had a hard time in assigning each member his tasks.

- Unexpected risks
  - We learned that there are some personal problems that we cannot predict. It means that an individual can suffer from illness or emotional affairs. For example, one of our team member had a stomach ache.

[Technical perspective]

- Limited network
  - We tried to use Google cloud message to send notification, but we could not. It was because there was a limitation that we cannot connect to external network. We resolved this problem by using MQTT. We learned that we need to make sure of technical limitation.

- Environment configuration problems
  - We tried to send query periodically to implement 'grace period'. However, we had some problems due to different MySQL versions. From this aspect, we could learn that, before implementation, we should get right versions of all the programs that we are going to use.

- Development
  - We completed 90% of our design. However, during implementation phase, we experienced that some functions or qualities are impossible to implement as we

designed. Finally, we found out alternative methods, and we learned it is common to experience technical difficulties.

[Architecture perspective]

- Importance of requirements analysis
  - We could learn the importance of requirements analysis. By doing requirements analysis, we could understand the stakeholders well, and it made us not to waste our time in refining requirements later.

- Architecture should be designed with considering quality attributes
  - We learned that we should design architecture and consider quality attributes at the same time. Because we did not consider quality attributes when we designed the architecture, we had to apply quality attributes while we were implementing, and this was difficult.

- Design decisions should not be done arbitrarily
  - When we determine design decisions by considering stakeholders' requirements, developers are the people who implement most of the functions and quality attributes. Therefore, we should determine design decisions by considering developers' ability. We learned that it needs to be very careful.

Appendix B: Strategies to satisfy the future needs

[Functional requirements]

- Statistic Data Extension
  Parking facility statistics can be extended with additional SQL query. System stores the user data with reservation data such as date, time, check in/out, payment amount, facility status and sensor data. These data can be supported to owner various data statistics.

- Car detecting in the facility
  If a facility can detect the car number, it would support more accurate information for car authentication. However, it need more special hardware and effort. In this system, it doesn't support car detecting feature.

- Unmanned facility
  In this system, there is an attendant for car driver authentication and monitoring the facility status. If owner wants to reduce the employee in the facility as unmanned facility, this system was designed for that purpose. Car driver can do self-authentication for the parking, and check the parking facility status with screen. Also, parking payment is automatic process. There is no reason attendant should be in the facility.

[Quality Attribute]

- Scalability
  Sure-park system is designed for supporting increasing parking slots and user clients. If it needs more parking space for the facility then Arduino can be added with a hardware configurations. And also, parking facility can be add with facility id. If server needs more capacity, it can be added with load balancer. In this project, we didn't design the architect with load balancer, but we applied stateless pattern for the system for scalability in the future.

- Availability
  In this system, there is backup database for recovering in the system failure. It will back up the data at every 3 am to the separated data repository. Android save the reservation information in the storage, it can support availability

- Security
  This system can be supported by HTTPS connection with the Certificate Authority (CA) and fixed IP address. In this system, we applied AES for data encryption and decryption during the data transition.

References

http://sparkjava.com/

http://www.mybatis.org/mybatis-3/

http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

http://mqtt.org/

https://en.wikipedia.org/wiki/Push_technology

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

https://en.wikipedia.org/wiki/Multitier_architecture

https://en.wikipedia.org/wiki/Representational_state_transfer

http://www.tutorialspoint.com/restful/

https://en.wikipedia.org/wiki/Middleware

https://www.instantssl.com/ssl-certificate-products/https.html

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

https://en.wikipedia.org/wiki/SHA-2

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

https://dev.mysql.com/doc/refman/5.7/en/event-scheduler.html

http://dev.mysql.com/doc/refman/5.7/en/backup-methods.html

https://www.polymer-project.org/1.0/

https://developer.android.com/reference/android/os/AsyncTask.html

https://github.com/ysyun/jmqtt_client

http://www.eclipse.org/paho/clients/android/sample/

https://www.arduino.cc/