Sure-Park Project

Architectural Design Document

Version 1.0

|  |  |
| --- | --- |
| **Team** | **Infinite Challenge(Team 3)** |
| **Members** | Namjin Lee, Jack Oh, Charles Park, Joan Kim, Jaeheon Kim |

**History**

|  |  |  |
| --- | --- | --- |
| **Ver.** | **Date** | **History** |
| 0.1 | 19, June, 2016 | Initial |
| 0.2 | 21, June, 2016 |  |
|  |  |  |
|  |  |  |

Contents

[**1.** **System Context** 6](#_Toc454379506)

[**2.** **1st Decomposition** 7](#_Toc454379507)

[**2.1.** **Physical View of 1st Decomposition** 7](#_Toc454379508)

[**2.2.** **Dynamic View of 1st Decomposition** 7](#_Toc454379509)

[**2.3.** **Static View of 1st Decomposition** 8](#_Toc454379510)

[**2.4.** **Entity Catalog** 8](#_Toc454379511)

[**2.5.** **Rationale** 9](#_Toc454379512)

[**3.** **2nd Decomposition** 9](#_Toc454379515)

[**3.1.** **SurePark Manager** 9](#_Toc454379516)

[**3.1.1.** **Dynamic view of SurePark Manager** 9](#_Toc454379517)

[**3.1.2.** **Static view of SurePark Manager** 11](#_Toc454379518)

[**3.1.3.** **Entity Catalog** 11](#_Toc454379519)

[**3.1.4.** **Rationale** 11](#_Toc454379520)

[**3.2.** **Facility Controller** 12](#_Toc454379521)

[**3.2.1.** **Static view of Facility Controller** 12](#_Toc454379522)

[**3.2.2.** **Entity Catalog** 12](#_Toc454379523)

[**3.2.3.** **Rationale** 13](#_Toc454379524)

[**3.3.** **Web Service** 13](#_Toc454379525)

[**3.3.1.** **Dynamic view of Web Service** 13](#_Toc454379526)

[**3.3.2.** **Static view of Web Service** 14](#_Toc454379527)

[**3.3.3.** **Entity Catalog** 14](#_Toc454379528)

[**3.3.4.** **Rationale** 14](#_Toc454379529)

[**3.4.** **Facility Controller to SurePark Manager** 15](#_Toc454379530)

[**3.4.1.** **How to check if Facility Controller is alive** 15](#_Toc454379531)

[**3.5.** **SurePark Manager to Web Service** 16](#_Toc454379532)

[**3.5.1.** **How to resist attack from an attacker?** 16](#_Toc454379533)

[**4.** **Detail Design** 16](#_Toc454379534)

[**4.1.** **FC-SM packet** 16](#_Toc454379535)

[**4.1.1.** **Packet Structure** 16](#_Toc454379536)

[**4.1.2.** **Detailed Packet Scenario** 18](#_Toc454379537)

[**4.2.** **SM-WS packet** 20](#_Toc454379538)

[**4.2.1.** **Request packet structure** 20](#_Toc454379539)

[**4.2.2.** **Response packet structure** 20](#_Toc454379540)

[**4.2.3.** **Notify packet structure** 20](#_Toc454379541)

[**4.2.4.** **Commands** 20](#_Toc454379542)

[**4.3.** **SM-WS DB Schema** 24](#_Toc454379543)

[**4.4.** **SurePark Manager Detailed Design** 25](#_Toc454379544)

[**4.4.1.** **Management Service class diagram** 25](#_Toc454379545)

[**4.4.2.** **Control Service class diagram** 26](#_Toc454379546)

[**4.5.** **Facility Controller Detailed Design** 26](#_Toc454379547)

[**4.5.1.** **Facility Controller class diagram** 26](#_Toc454379548)

[**4.5.2.** **Interface design** 27](#_Toc454379549)

[**4.6.** **Web Service Detailed Design** 28](#_Toc454379550)

[**4.7.** **Sequence diagram** 28](#_Toc454379551)

[**4.7.1.** **Make a reservation (UC01)** 28](#_Toc454379552)

[**4.7.2.** **Show up & parking (UC02, UC05)** 29](#_Toc454379553)

[**4.7.3.** **Get out the Garage (UC04)** 29](#_Toc454379554)

[**5.** **Future Works** 30](#_Toc454379555)

[**6. Lesson Learned** 30](#_Toc454379556)

**FIGURE**

[Figure 1 System Context 5](#_Toc454367310)

# **System Context**

The key requisites of the project are functions that:

* drivers can reserve a parking space by using a laptop or a phone.
* parking attendants can monitor parking facilities
* the system is initially built for a small parking facility and should be able to be applied to various sized parking facilities
* the system can provide basic statistics including average occupancy, peak usage hours, parking slot statistics, and revenue, which should be extensible in order to help developers to add more analysis algorithms

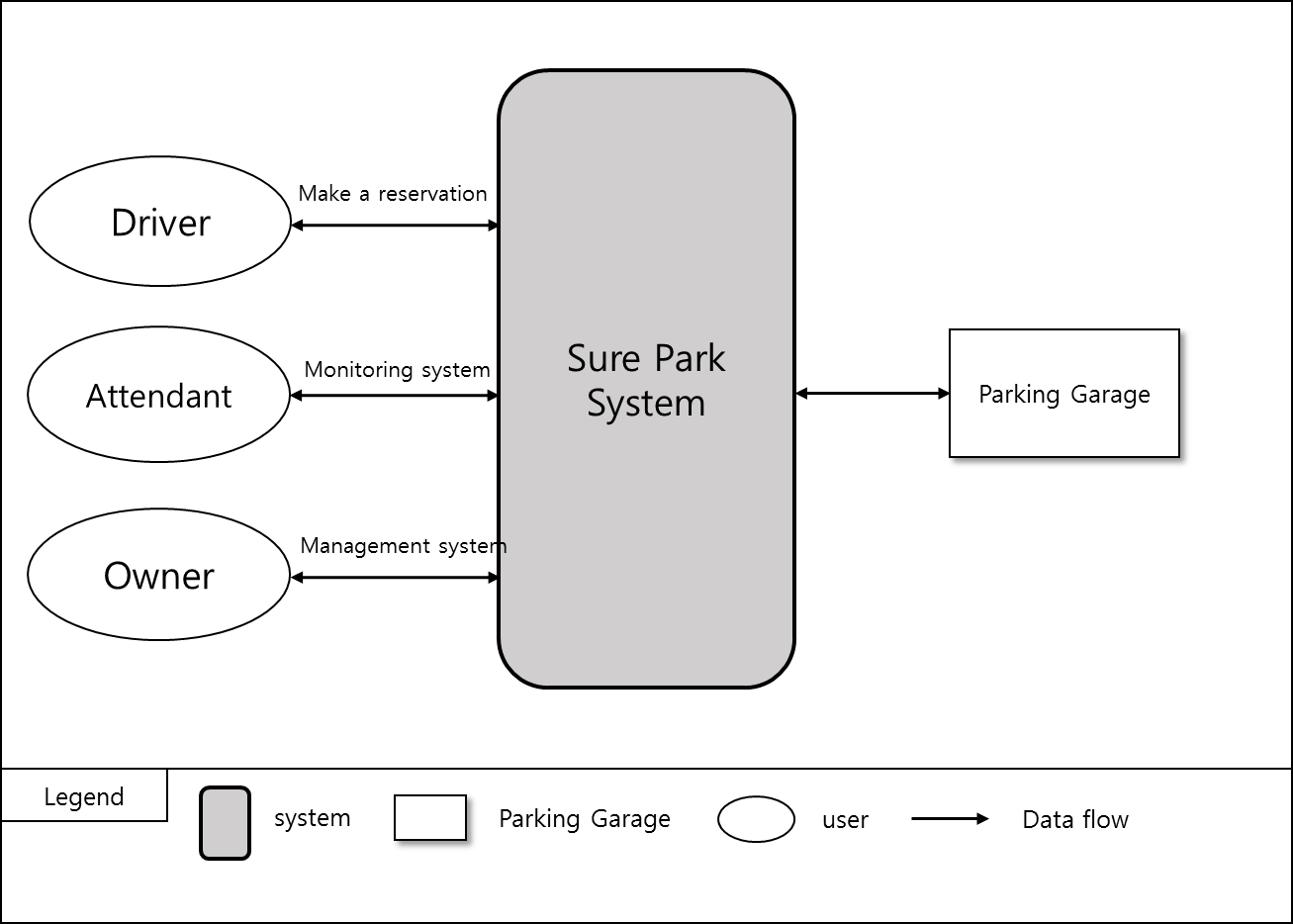


Figure 1 System Context

# **1st Decomposition**

## **Physical View of 1st Decomposition**



Figure 2 Physical view of 1st decomposition.

## **Dynamic View of 1st Decomposition**



Figure 3 Dynamic view of 1st decomposition.

## **Static View of 1st Decomposition**

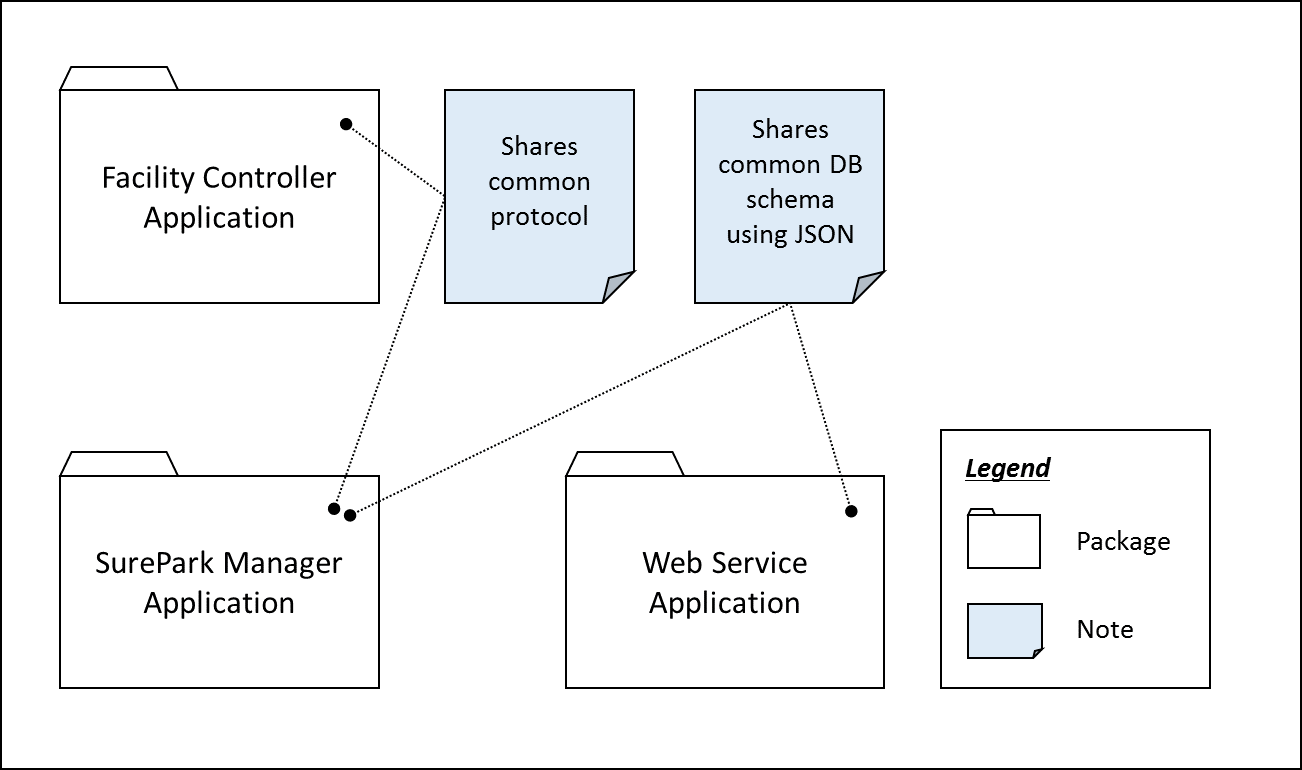


Figure 4 Static view of 1st decomposition.

## **Entity Catalog**

|  |  |
| --- | --- |
| **Entity** | **Description** |
| Web Browser | Users, attendants and owner can access their own UI through the web browser provided by the web server. |
| Web Service | Provides users with the functions of sign-up, log in, reservation, monitoring facilities and/or showing parking statistics based on data retrieved from SurePark DB.  Sends information to SurePark Manager for DB updates. |
| Facility Controller | Controls parking facilities; get the status of parking slots, turn on/off LEDs, detect a car at the gates and open/close the gates.  Receives data from SurePark Manager to control LEDs and/or gates.  Sends data to SurePark Manager to update the status of parking slots. |
| SurePark Manager | Handles show-up and no-show scenarios based on DB information.  Updates SurePark DB when a user has signed up, a reservation has been made or facility status has been changed. |
| SurePark DB | Keeps all of the data about users, garages and reservations.  Only can be updated by SurePark Manager. |

## **Rationale**

Modifiability (QA08) is one of the most important QAs of the SurePark system. An engineer needs to scale up the system within a week. We have divided the whole system into 5 parts according to their responsibilities, and applied client-server and repository pattern to connect each parts.



Figure 5 Architectural patterns of SurePark system



# **2nd Decomposition**

## **SurePark Manager**

### **Dynamic view of SurePark Manager**

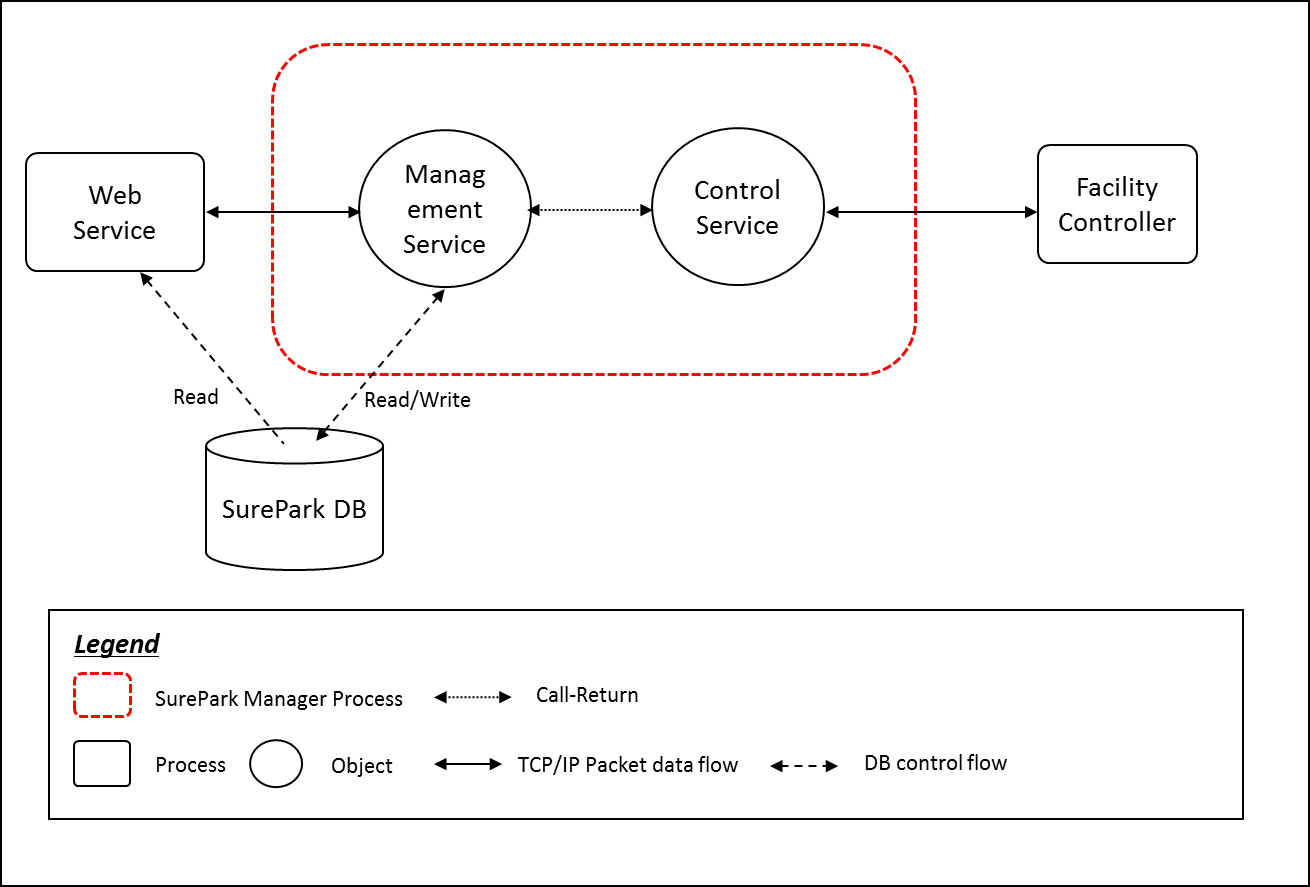


Figure 6 Dynamic view of SurePark Manager

### **Static view of SurePark Manager**

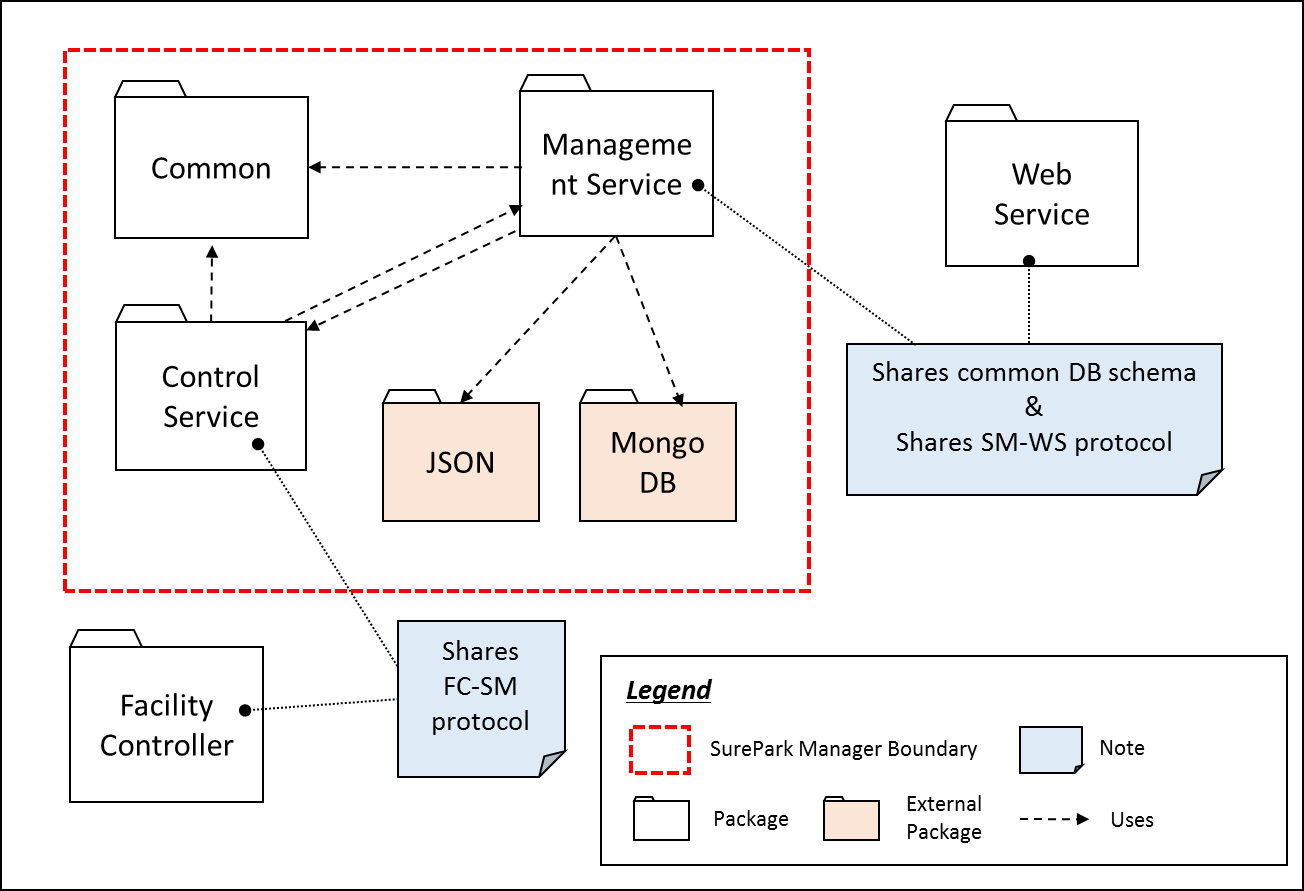


Figure 7 Static view of SurePark Manager

### **Entity Catalog**

|  |  |
| --- | --- |
| **Entity** | **Description** |
| Common | Common package includes garage and reservation information. Control service and Management service refers common parts. |
| Control Service | Control Service have a responsibility of managing facilities. It try to connect/disconnect facilities following garage information. And it send/receive packets for controlling facilities. |
| Management Service | Management Service have a responsibility of managing DB transaction and it also have a responsibility of managing Web Service program. |

### **Rationale**

The SurePark Manager has two major responsibilities. One thing is handling Facility Controllers and the other is handling Web Service and DB. We’ve divided it ‘Control service’ and ‘Management service’ for increasing cohesion. Control service and Management service call each interface class for reducing coupling (QA08).

## **Facility Controller**

### **Static view of Facility Controller**

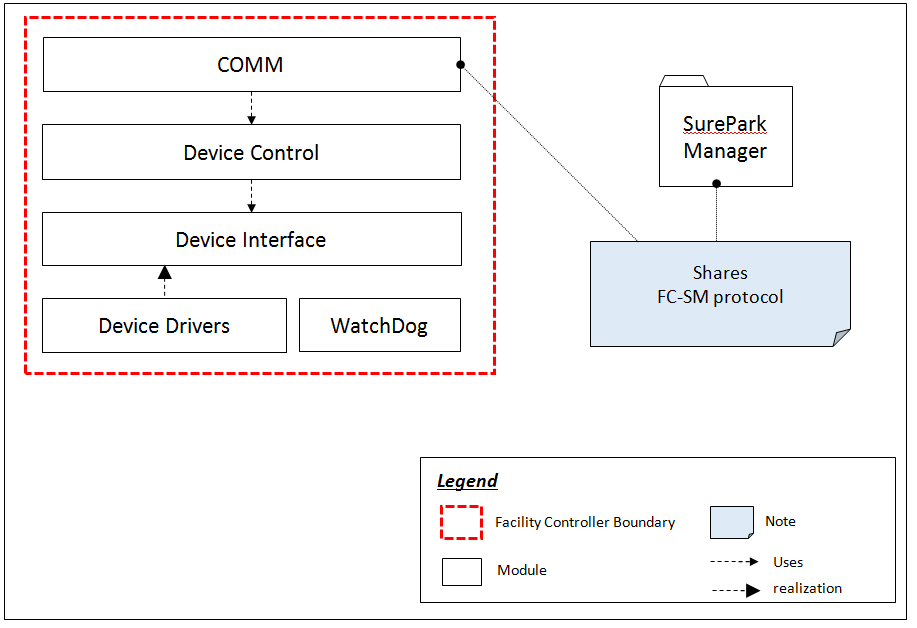


Figure 8 Static view of Facility Controller

### **Entity Catalog**

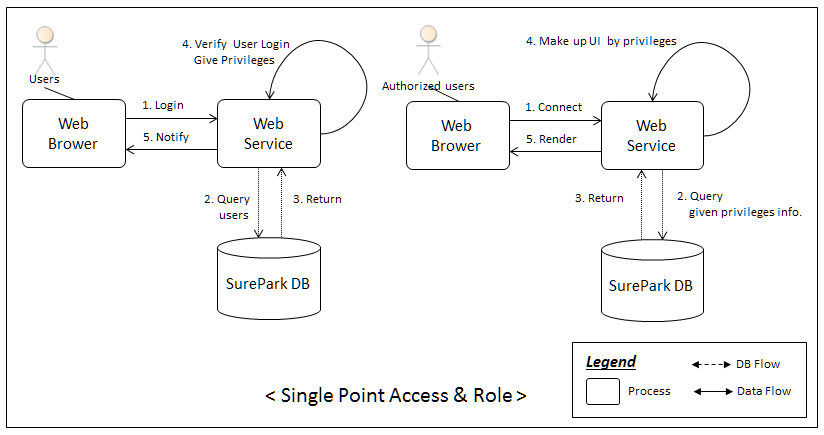
|  |  |
| --- | --- |
| **Entity** | **Description** |
| Comm | Communication module between SurePark Manager and Facility Controller |
| Device Control | Module for non-feedback device control (exit/entry gate LED, exit/entry servo on/off).  This module observes state of devices when the state changed. |
| Device Interface | Set states of exit/entry gate LED, exit/entry servo and stall LEDs by Device Control  Get states of stall sensors and exit/entry sensors when observer request |
| Device driver | Control ON/OFF or OPEN/CLOSE(exit/entry gate LED, exit/entry servo and stall LEDs)  Sensing value of stall sensors and exit/entry sensors |
| Watch Dog | A watchdog timer is used to detect and recover from MCU malfunctions. During normal operation, the MCU regularly restarts the watchdog timer to prevent it from "timing out". |

### **Rationale**

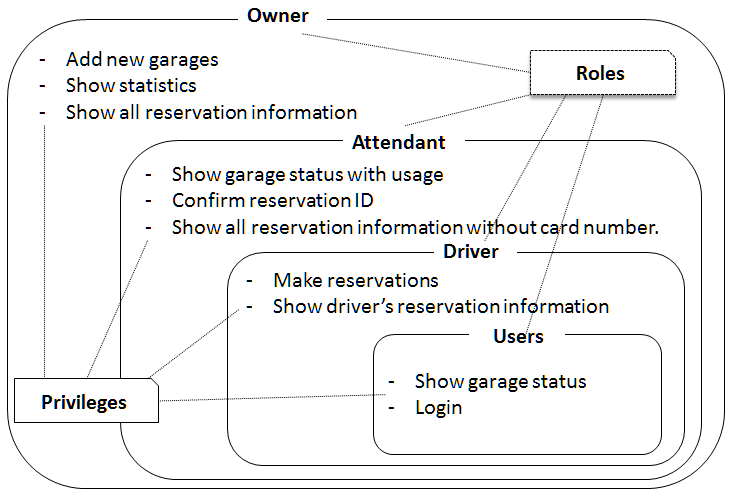
The structure is layered type. We use interfaces so that new devices is be added. thus, We can reduce dependency. Developers can design scale up/out the system easy. (QA08)

## **Web Service**

### **Dynamic view of Web Service**

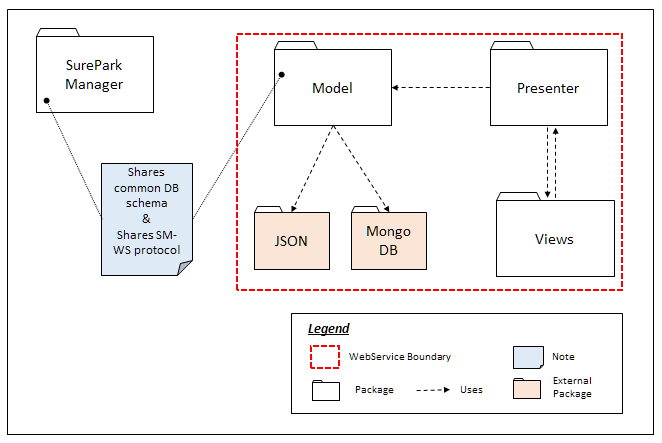


**Role & Privileges**



**Rationale**

### **Static view of Web Service**



### **Entity Catalog**

|  |  |
| --- | --- |
| **Entity** | **Description** |
| Views | The Views compoent is the representation of the user interface. The Views is implemented using html5 and css and responsible for rendering and accepting user input. |
| Presenter | The presenter component is responsible for orchestrating all the application’s use cases. The presenter involves that invoking operation and taking user input from View and settings data. |
| Model | The Model component encapsulates all business logic and data in the application. It contains the transaction with other process which are DB read/write and communication with SureParkManager process. |

### **Rationale**

MVP can isolate user interface from business tier, easily interchange user input from Views and encapsulate the model. It could enhance the modifiability (QA08). If the business tier is changed, it could easily adapt changing the Model component.

## **Facility Controller to SurePark Manager**

### **How to check if Facility Controller is alive**

#### Dynamic behavior of FC-SM packet



<figure X,

#### Rationale

We need to check the status of the Facility Controller. The SurePark Manager communicate with the Facility Controller through TCP/IP. Communication could be failed because of the many reasons. We’ve considered heartbeat tactic and ping-echo tactic to detect this kinds of faults. Ping-echo sends packets each other but heartbeat sends a packet one way. Since the SurePark Manager has to control several Facility Controllers, we’d better reduce network traffic. So we choose heartbeat tactic for availability (QA02).

## **SurePark Manager to Web Service**

### **How to resist attack from an attacker?**



#### Rationale

SurePark Manager and WebService use MongoDB for database. It supports a number of authentication mechanisms that clients can use to verify their identity. Only verified user can access database.

# **Detail Design**

## **FC-SM packet**

### **Packet Structure**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Symbol** | **Facility Id** | **Code** | **Value** | **End Symbol** |
| 1byte($) | 4byte | 1byte | Variable length | 1byte(\n) |

* Start Symbol: Start point of valid packet.
* Facility Id: Assigned the facility Id.
* Code: Indicate what kind of packet is. I means “Information”. S means “Slot Status”. E means “Entry Gate”. X means “Exit Gate”, L means “LED”.
* Value: It depends on “Code”. Please refer to “detailed packet scenario”.
* End Symbol: End point of valid packet.

### **Detailed Packet Scenario**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Information  (SM->FC) | After connection, the SurePark Manager has to send information to the Facility Controller. The facility controller has to control slot number by this information. Max slot number is 10 by technical constraint.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | $ | Facility Id | I | Slot number | \n |   Ex) $1001I4\n (Garage 1001 consists of 4 stalls.) |
| Heartbeat  (FC->SM) | Basically, Facility controller send heartbeat packet at every 10 seconds. Timer for which heartbeat restarts after sending all packet. For example, if the facility controller send a slot status packet, it reset the timer for heartbeat and recount that time.  The SurePark manager recognizes if the facility controller is alive through all received packets.   |  |  |  | | --- | --- | --- | | $ | Facility Id | \n |   Ex) $1001\n |
| Open Entry Gate  (SM->FC) | This packet request to open entry gate to facility controller.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | $ | Facility Id | E | 1 | \n |   Ex) $1001E1\n (Garage 1001 has to open the entry gate.) |
| Turn on LED  (SM->FC) | This packet request to open entry gate to facility controller. Slot index designates 0 ~ Slot Number-1.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | $ | Facility Id | L | Slot index | \n |   Ex) $1001L2\n (Garage 1001 has to turn on the LED No.2.) |
| Slot Status  (FC->SM) | Whenever a slot status has changed, facility controller send this packet to SurePark manager. Slot index depends on slot number of information packet.   * Value 0: slot is opened. * Value 1: slot is occupied. * Value 2: slot is broken.  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | $ | Facility Id | S | S0 | S1 | … | SN | \n |   Ex) $1001S1001\n (Slot 0 and 3 are occupied.) |
| Exit Gate  (FC->SM) | After a car gets out of garage, the facility controller send this packet to SurePark manager. If a car is just going out of garage, not parked, the facility controller send it with value 2.   * Value 1: A car which pared at garage went out of garage. * Value 2: A car which is not parked at garage went out.  |  |  |  |  |  | | --- | --- | --- | --- | --- | | $ | Facility Id | X | 1 | \n |   Ex) $1001X1\n (A car which pared at garage went out of garage.)  Ex) $1001X2\n (A car which is not parked at garage went out.) |

## **SM-WS packet**

Communication between SureParkManger and WebService use SM-WS packet. The packet structure is used JSON for sending, and the all packet data is encapsulated to string. The communication use request-response mechanism. WebService request services, and SureParkManager do services and response services.

And the communication use notify mechanism also. SureParkManager notify events to WebService, but no response for this mechanism.

### **Request packet structure**

|  |  |
| --- | --- |
| **Request command** | **Data** |
| String | JSON Object |
| JSON Object | |

### **Response packet structure**

|  |  |
| --- | --- |
| **Response command** | **Acknowledge** |
| String | String |
| JSON Object | |

### **Notify packet structure**

|  |  |
| --- | --- |
| **Request command** | **Data** |
| String | JSON Object |
| JSON Object | |

### **Commands**

|  |  |
| --- | --- |
| **Command** | **Data** |
| newGarage | When new garage is installed, Web Service add to the garage’s information to SureParkManager. SureParkManager update garages database.  - Request packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | garageName | String | Save garage name | | garageNumber | String | garage ID | | slotNum | String | Number of parking slot | | slotStatus | JSON Object | status of the each parking slot | | gracePeriod | String | gracePeriod value | | parkingFee | String | Parking fee per hour | | garageIP | String | garage IP address | | isAvailable | String | If this garage available |   Ex) {"newGarage": {"garageName":"AAAAA", "garageId":”1001”, "slotNumber":”4”, "slotStatus":["Open","Open","Open","Open"],  "gracePeriod":"90", "parkingFee":"30", "garageIP":"127.0.0.1", "isAvailable":true}}  - Response packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Data** | | Acknowledge | String | “OK” or “FAIL” |   Ex) {"newGarage": "OK"} or {"newGarage": "FAIL"} |
| newReservation | When a driver makes a reservation, WebService send new reservation to SureParkManger. SureParkManager update reservations database.  - Request packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | userID | String | user id who are made reservation | | reservationTime | String | reservation date/time | | cardInfo | String | card information | | confirmInformation | String | confirmation number | | gracePeriod | String | gracePeriod value | | parkingFee | String | parking fee per hour | | usingGarage | String | garage name to park |   {"newReservation": {"userID":"jack", “reservationTime”:”2016-06-22T14:30:00.000Z”, "cardInfo":"1111-\*\*\*\*-\*\*\*\*-4444", "confirmInformation":"A1234", "gracePeriod":"90", "parkingFee":"30", "usingGarage":"Sure-Park"}}  - Response packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Data** | | Acknowledge | String | “OK” or “FAIL” |   Ex) {" newReservation ": "OK"} or {" newReservation ":"FAIL"} |
| cancelReservation | When a driver cancels a reservation, WebService send cancel reservation to SureParkManger. SureParkManager update reservations database.  - Request packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | userID | String | user id who want cancel reservation | | confirmInformation | String | confirmation number | | usingGarage | String | garage name to park |   Ex) {"cancelReservation": {"userID":"jack", "usingGarage":"Sure-Park", "confirmInformation":"A1234"}}  - Response packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Data** | | Acknowledge | String | “OK” or “FAIL” |   Ex) {" cancelReservation ": "OK"} or {" cancelReservation ":"FAIL"} |
| newUser | When new user signed up, Web Service send new user sign up information to SureParkManager. SureParkManager update users database.  - Request packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | userID | String | user ID | | userPassword | String | user Password | | userType | String | classify driver, attentant and owner | | userName | String | user name | | userEmail | String | user email address | | displayName | String | user nick name |   Ex) {"newUser": {"userID":"myID", "userPassword":"myPassword", "userType":"user", "userName":"myName", "userEmail": “myEmail@email.com", "displayName":"MY NAME"}}  - Response packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Data** | | Acknowledge | String | “OK” or “FAIL” |   Ex) {" newUser ": "OK"} or {" newUser ":"FAIL"} |
| parkingCar | When a driver who has reservation is show up with confirm information, Web Service send information to SureParkManager. SureParkManager update reservation database, and control facilities.  - Request   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | userID | String | user id who are made reservation | | usingGarage | String | confirmation number | | confirmInformation | String | garage name to park |   Ex) {"parkingCar": {"userID":"myID", "usingGarage":"Sure-Park", "confirmInformation":"A1234"}}  - Response packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Data** | | Acknowledge | String | “OK” or “FAIL” |   Ex) {" parkingCar ": "OK"} or {" parkingCar ":"FAIL"} |
| updateSlotStatus | When slot status is changed, Web Service need to update slot status. SureParkManger notify updateSlotStatus event to WebService.  - Notify packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | garageName | String | garage name to update | | garageNumber | String | garage number to update |   Ex) {"updateSlotStatus":{"garageName":"Sure-Park", "garageNumber":"1001"}} |
| wrongParking | When facility controller detect a driver parked wrong slot, , Web Service need to update slot status. SureParkManger notify wrongParking event to WebService.  - Notify packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | garageName | String | garage name which has wrong parking | | garageNumber | String | garage number which has wrong parking |   Ex) {" wrongParking ":{"garageName":"Sure-Park", "garageNumber":"1001"}} |
| detectFailure | When SureParkManager detect facilities failure, Web Service need to notice failure to attendants. SureParkManger notify detectFailure event to WebService.  - Notify packet   |  |  |  | | --- | --- | --- | | **Data Field** | **Data type** | **Description** | | garageName | String | garage name which detect failure | | garageNumber | String | garage number which detect failure |   Ex) {" detectFailure ":{"garageName":"Sure-Park", "garageNumber":"1001"}} |

## **SM-WS DB Schema**

|  |  |  |  |
| --- | --- | --- | --- |
| **collection** | **name** | **type** | **description** |
| **users** | userID | string | Save user ID |
| userPassword | string | Save user Password |
| userType | string | classify driver, attentant and owner |
| userName | string | Save user name |
| userEmail | string | Save user email address |
| registerationTime | Date | Save user registeration date/time |
| displayName | string | Save user nick name |
| **garages** | garageName | string | Save garage name |
| garageNumber | integer | garage ID |
| slotNumber | integer | Number of parking slot |
| slotStatus | JSON Object | status of the each parking slot |
| updateTime | JSON Object | Update time of parking slot |
| gracePeriod | integer | gracePeriod value |
| parkingFee | integer | Parking fee per hour |
| garageIP | string | garage IP address |
| isAvailable | Boolean | If this garage available |
| **reservations** | userID | string | user id who are made reservation |
| reservationTime | Date | reservation date/time |
| cardInfo | string | card information |
| confirmInformation | string | confirmation number |
| gracePeriod | integer | gracePeriod value |
| reservationStatus | string | reservation status |
| parkingFee | integer | parking fee per hour |
| usingGarage | string | garage name to park |
| usingGarageNumber | integer | garage number tp park |
| usingSlot | string | slot numer to park |
| parkingTime | Date | parking date/time |
| leaveTime | Date | leave date/time |
| chargingFee | integer | charge fee |

## **SurePark Manager Detailed Design**

### **Management Service class diagram**

Define the relation among classes. To emphasis modifiability, layer pattern is used. Each layer’s interface use java interface and the reason is dependency inversion principle (DIP).



#### 4.4.1.1. Entity Catalog

|  |  |
| --- | --- |
| **Layer name** | **Description** |
| Interface | This layer communicate with external thread, package and process.  Delivers request from Web service to Facility controller, and notify from Facility controller.  Delivers information to business logic. |
| Business Logic | This layer analysis request, and delivers information to Data Access.  Check reserved driver is show up or not. If driver no show, this layer update database using Data Access. |
| Data Access | This layer access database with authentication.  Information from Business Logic transact to database. |

### **Control Service class diagram**



#### Entity Catalog

|  |  |
| --- | --- |
| **Interface** | **Responsibility** |
| IControlService | IControlService interface is for management service. It provides addFacility and openEntryGate functions. |

## **Facility Controller Detailed Design**

### **Facility Controller class diagram**

For easy adding or removing devices, we use observer pattern. Device driver is sensing every 0.5sec and Observer is polling state of sensor. When state of sensor changed, Observer notify comm. Finally, Comm. send message to surepark manager.



### **Interface design**

1->3step: generalization to on/off

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 step (device name : count) | 2 step | 3 step |
| non feed back device (OUTPUT) | Stall LED : 4 | ON/OFF | ON/OFF |
| Extit LED : 1 | ON/OFF | ON/OFF |
| Entry LED : 1 | ON/OFF | ON/OFF |
| Exit Servor : 1 | OPEN/CLOSE | ON/OFF |

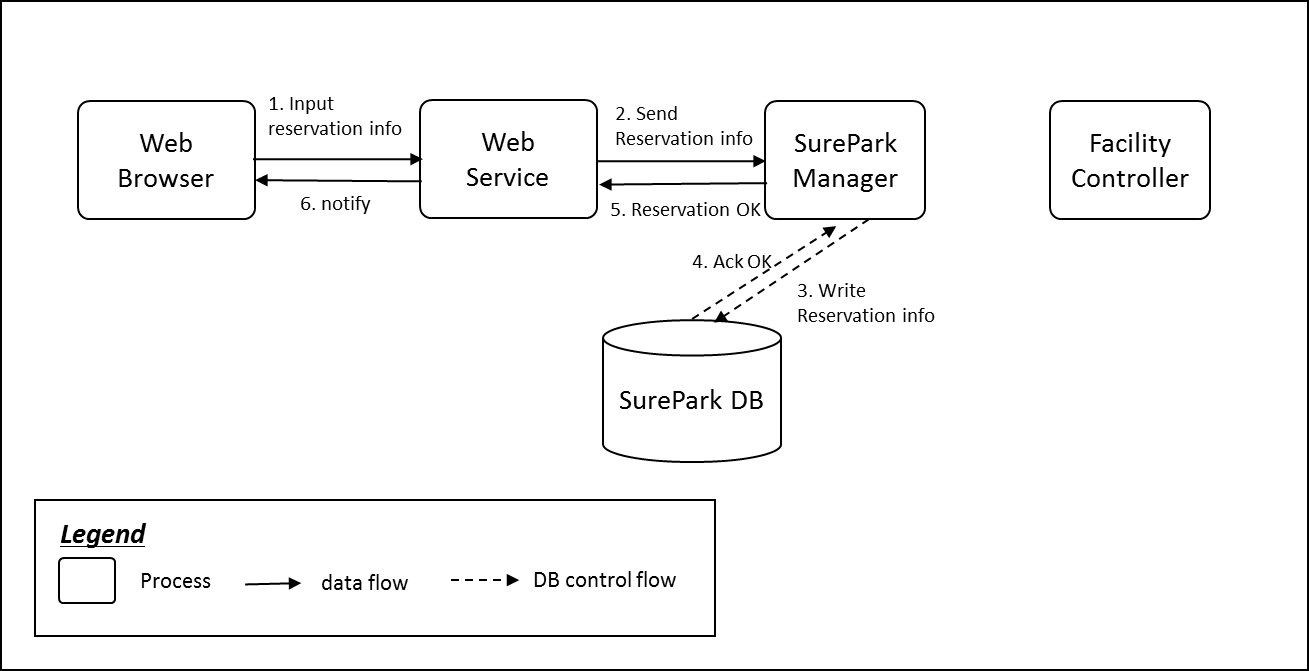
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 step (device name : count) | 2 step | 3 step | 4 step | 5 step |
| feed back device (INPUT) | Stall Sensor : 4 | value : 0~256 | IF value\_average - offset value >= 50  then no car, else car is | Parked/No car | HIGH/LOW |
| Entry Gate Sensor : 1 | HIGH/LOW | HIGH/LOW | HIGH/LOW | HIGH/LOW |
| Exit Gate Sensor : 1 | HIGH/LOW | HIGH/LOW | HIGH/LOW | HIGH/LOW |

1->5step : generalization to HIGH/LOW

## **Web Service Detailed Design**

## **Sequence diagram**

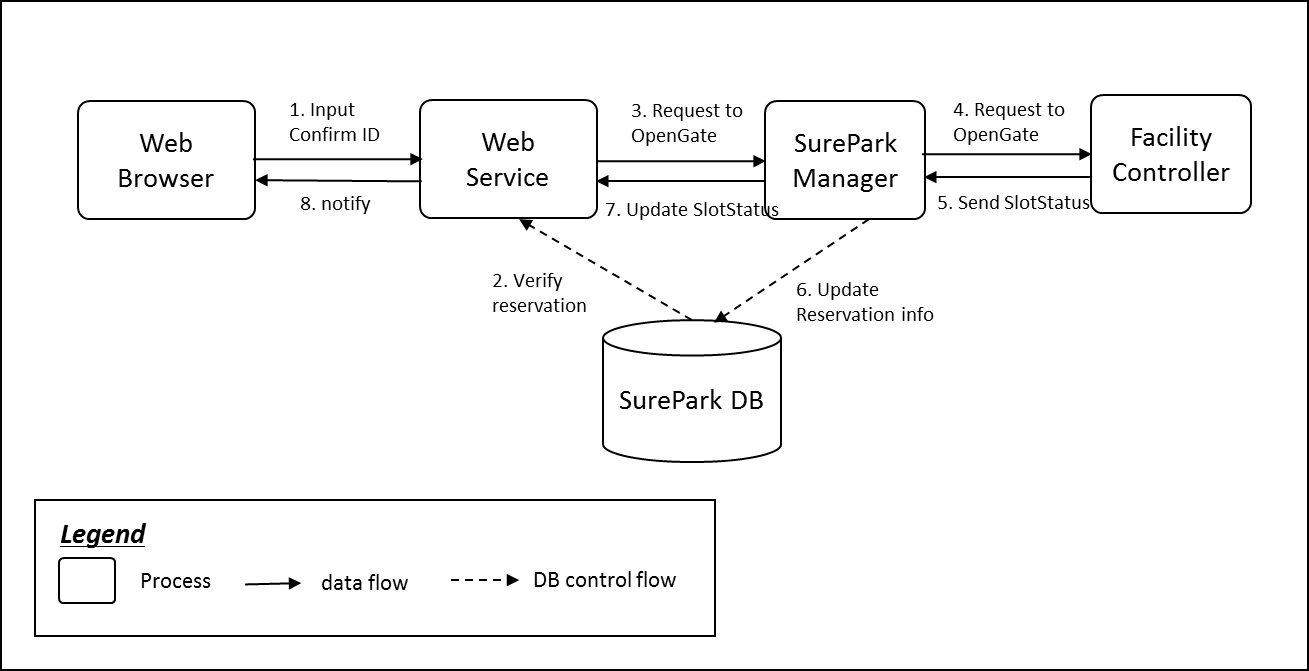
### **Make a reservation (UC01)**



#### Web Service

#### SurePark Manager

### **Show up & parking (UC02, UC05)**

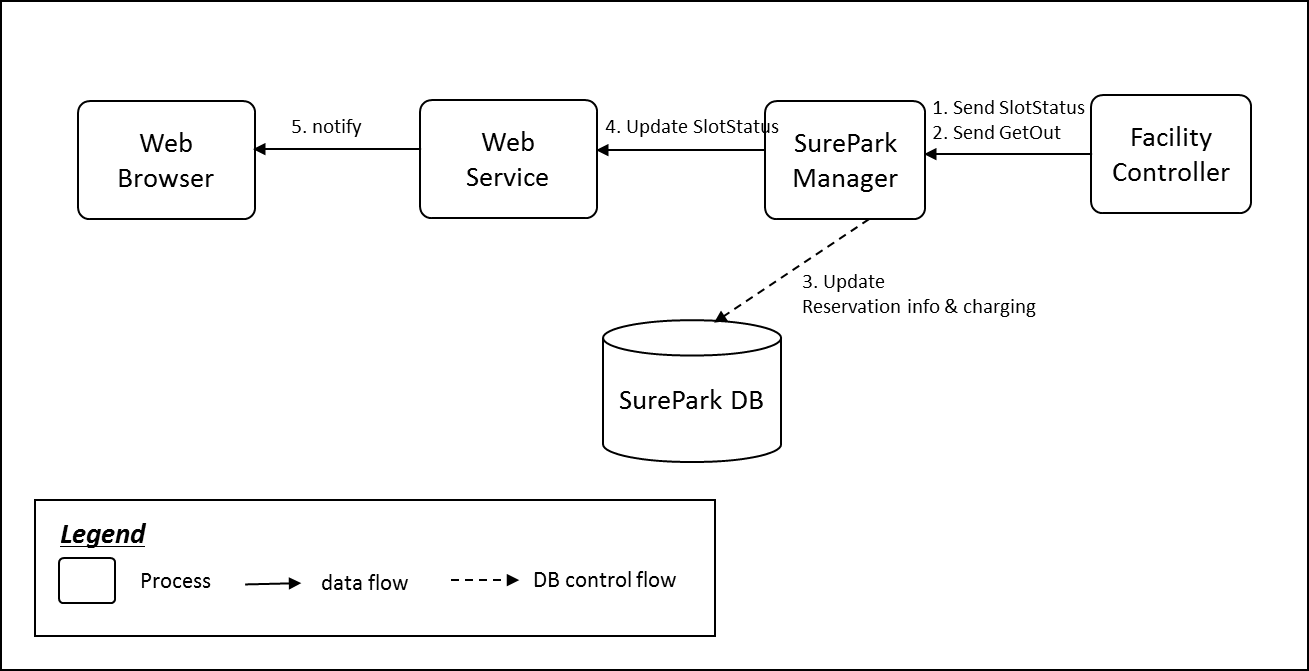


#### Web Service

#### SurePark Manager

#### Facility Controller

### **Get out the Garage (UC04)**



#### Facility Controller

#### SurePark Manager

#### Web Service

# **Future Works**

**- Implement extensibility for new algorithm for statistics (plugin type)**

The system should be extensible to enable developers to add more analysis algorithms or analysis applications without disrupting operations to add the new features. But this requirement is not implemented in the initial version of Sure Park system software because it has lower priority than other functions.

**- Enhance interoperability between Facility Controller and SurePark Manager.**

At this time, if a car tries to park at a stall, the SurePark Manager doesn’t remember a designated stall number after the connection is reconstruction. Finally, the SurePark Manager can misjudge whether a car parked at designated stall correctly or not. So we need to refactor our design in the future.

**- Mongo DB replication for availability**

There can be lost connection with DB. So we need to care of it for enhanced availability. Since Mongo DB provides replication of data, we will set it in the future.

# **6. Lesson Learned**

**- Using COTS**

We uses COTS to implement some parts of the SurePark system: MongDB, Google Chart and Java script libraries. We thought that we can save resources by using COTS, but it is half right and half wrong. Google Chart, for example, cannot be used in the demonstration because the router is not connected to the internet. We realized it too late and could not fix it. Some of Java script libraries caused network problems and it takes some time to find the root causes and fix them. Using COTS is not free, it costs both time and money.

**- Development Tools**

During the studio project, we’ve heard about lots of tools supporting project management and tries to use some of them. For example, we’ve used a time tracking application to record our time log and it makes easier for us to manage time.

**- Perspective**

Before the CMU course, we didn’t really look into the perspectives. In most cases, we just think about the static view which shows the relationship of functions, classes and/or modules because it can be mapped to the source code directly. Sometimes, we have used mixed perspectives. In the CMU course, we have learned why the perspectives are important and how to use them for the architectural design.

**- Interface**

Once the interface has been fixed, it becomes a constraint of the successive design and implementation. During the development, we have experienced this kind of situation and had to spend our resources because of changing some protocols. We have learned that the interface is one of the most important parts of the architectural design and needs to be fixed at the early stage of the development.