Software Domain Analysis & Design

-Elaboration Iteration 2-



Kick Share!

Team name: 김말국

201621028 황지훈 (goharrm@ajou.ac.kr)

201621046 용상호 (yonghole@ajou.ac.kr)

201620999 김민창 (rlarudfbf12@ajou.ac.kr)

201723305 안상희 (sanghee0610@ajou.ac.kr)

**Contents**

[1. Vision 5](#_Toc58111742)

[1.1. Introduction 5](#_Toc58111743)

[1.2. Positioning 5](#_Toc58111744)

[1.2.1. Business Opportunity 5](#_Toc58111745)

[1.2.2. Problem Statement 5](#_Toc58111746)

[1.2.3. Product Position Statement 6](#_Toc58111747)

[1.3. Stakeholder Descriptions 6](#_Toc58111748)

[1.3.1. User summary 6](#_Toc58111749)

[1.3.2. Non-User summary 6](#_Toc58111750)

[1.3.3. User-Level Goals 6](#_Toc58111751)

[1.3.4 User Environment 6](#_Toc58111752)

[1.4. Product Overview 7](#_Toc58111753)

[1.4.1. Product Perspective 7](#_Toc58111754)

[1.4.2. Assumptions and Dependencies 7](#_Toc58111755)

[1.5. Summary of System Features 8](#_Toc58111756)

[1.6. Other Requirements and Constraints 8](#_Toc58111757)

[2. Requirements 9](#_Toc58111758)

[2.1. Function Requirements 9](#_Toc58111759)

[2.1.1. Use Case Diagram 9](#_Toc58111760)

[2.1.2. Use Case Brief 9](#_Toc58111761)

[2.1.2.1. <Use Case 1> Rent Kickboard 9](#_Toc58111762)

[2.1.2.2. <Use Case 2> Return KickBoard 10](#_Toc58111763)

[2.1.2.3. <Use Case 3> Charge Point 10](#_Toc58111764)

[2.1.2.4. <Use Case 4> Manage Kickboard 10](#_Toc58111765)

[2.1.2.5. <Use Case 5> Calculate Fare 10](#_Toc58111766)

[2.1.3. Use Case Text 10](#_Toc58111767)

[2.1.3.1. <Use Case 1> Rent KickBoard 10](#_Toc58111768)

[2.1.3.2. <Use Case 2> Return Kickboard 12](#_Toc58111769)

[2.1.3.3. <Use Case 3> Charge Point 14](#_Toc58111770)

[2.1.3.4. <Use Case 4> Manage Kickboard 15](#_Toc58111771)

[2.1.3.5. <Use Case 5> Calculate Fare 16](#_Toc58111772)

[2.2. Non-Functional Requirements 18](#_Toc58111773)

[3. Analysis Modeling 18](#_Toc58111774)

[3.1. Domain Model Diagram 18](#_Toc58111775)

[3.2. System Sequence Diagram 19](#_Toc58111776)

[3.2.1. <Use Case 1> Rent Kickboard 19](#_Toc58111777)

[3.2.1.1 Domain model 19](#_Toc58111778)

[3.2.1.2. SSD 19](#_Toc58111779)

[3.2.1.3. Operation Contracts 20](#_Toc58111780)

[3.2.2. <Use Case 2> Return Kickboard 21](#_Toc58111781)

[3.2.2.1. Domain Model 22](#_Toc58111782)

[3.2.2.2. SSD 22](#_Toc58111783)

[3.2.2.3. Operation Contracts 23](#_Toc58111784)

[3.2.3. <Use Case 3> Charge Point 23](#_Toc58111785)

[3.2.3.1. Domain Model 23](#_Toc58111786)

[3.2.3.2. SSD 23](#_Toc58111787)

[3.2.3.3. Operation Contracts 24](#_Toc58111788)

[3.2.4. <Use Case 4> Manage KickBoard 24](#_Toc58111789)

[3.2.4.1. Domain Model 24](#_Toc58111790)

[3.2.4.2. SSD 25](#_Toc58111791)

[3.2.4.3. Operation Contracts 25](#_Toc58111792)

[3.2.5. <Use Case 5> Calculate Fare 26](#_Toc58111793)

[3.2.5.1. Domain Model 26](#_Toc58111794)

[3.2.5.2. SSD 26](#_Toc58111795)

[3.2.5.3. Operation Contracts 27](#_Toc58111796)

[4. Design Modeling 27](#_Toc58111797)

[4.1. <Use Case 1> Rent Kickboard Realization 28](#_Toc58111798)

[4.1.1. Design Sequence Diagrams 28](#_Toc58111799)

[4.1.1.1. System Operation 1: startRental() 28](#_Toc58111800)

[4.1.1.2. System Operation 2 : checkKickBoard() 29](#_Toc58111801)

[4.1.1.3. System Operation 3 : rentKickboard() 30](#_Toc58111802)

[4.1.1.4. Design Class Diagram 32](#_Toc58111803)

[4.2. <Use Case 2> Return Kickboard Realization 32](#_Toc58111804)

[4.2.1. Design Sequence Diagrams 32](#_Toc58111805)

[4.2.1.1. SystemOperation 1 : scanQRcode() 32](#_Toc58111806)

[4.2.1.2 Design Class Diagram 35](#_Toc58111807)

[4.3. <Use Case 3> Charge Point Realization 35](#_Toc58111808)

[4.3.1. Design Sequence Diagrams 35](#_Toc58111809)

[4.3.1.1. System Operation 1 : chargePoint() 35](#_Toc58111810)

[4.3.1.2. Design Class Diagram 37](#_Toc58111811)

[4.4. <Use Case 4> Manage Kickboard Realization 37](#_Toc58111812)

[4.4.1. Design Sequence Diagrams 37](#_Toc58111813)

[4.4.1.1. System Operation 1 : requestResultOfDistribution() 37](#_Toc58111814)

[4.4.1.2 Design Class Diagram 40](#_Toc58111815)

[4.5 <Use Case 5> Calculate Fare Realization 40](#_Toc58111816)

[4.5.1. Design Sequence Diagrams 40](#_Toc58111817)

[4.5.1.1. System Operation 1 : calculateFare(userID) 40](#_Toc58111818)

[4.5.1.2 Design Class Diagram 43](#_Toc58111819)

[5. Architecture 44](#_Toc58111820)

[5.1. Introduction 44](#_Toc58111821)

[5.2. Architectural Factors 44](#_Toc58111822)

[5.3. Logical view 48](#_Toc58111823)

[5.4. Process view 49](#_Toc58111824)

[5.5. Deployment view 50](#_Toc58111825)

[5.6. Use case view 50](#_Toc58111826)

[5.7. Data view 51](#_Toc58111827)

[5.8. Technical Memos 52](#_Toc58111828)

[6. Conclusion 55](#_Toc58111829)

[7. References 56](#_Toc58111830)

[8. Appendix 56](#_Toc58111831)

[9. Revision History 58](#_Toc58111832)

1. Vision

1.1. Introduction

기술 발전에 따라 전기 배터리의 용량이 커지고 사이즈가 작아지면서, ‘개인 이동성’이 중요시되는 시대가 도래하였다. 특히 전동 킥보드는 편리하고 가격도 저렴해 전 세계적으로 인기가 많고 유명하다. 최대 시속 30km까지 주행할 수 있으며, 집에서 편리하게 충전할 수 있다. 또한, 자전거보다 훨씬 빠르고 편리하며 대중교통으로 가기 힘든 목적지까지 더 빠르게 이동할 수 있게 해준다. 공유 또한 우리 사회의 새로운 트렌드가 되면서, 자동차와 자전거 공유 서비스가 일상화되고 있다. 전동 킥보드는 이미 편리하고 효율적이지만, 공유 서비스와 결합하면 더욱 강력하다.

보통 대학교 캠퍼스는 매우 넓고 대중교통도 원활하게 지원되고 있지 않다. 따라서 전동 킥보드 공유 서비스를 대학교에 적용시킴으로써 수업과 수업사이의 짧은 시간내에 넓은 캠퍼스를 보다 편하게 이동할 수 있을 것이며 학교 외부 인근 주변시설도 편하게 이용할 수 있을 것이다.

1.2. Positioning

1.2.1. Business Opportunity

사람들이 이동 시간의 단축과 효율성을 추구하게 되면서, 빠르고 편리한 이동수단인 전동 킥보드가 각광을 받고 있다. 버스 혹은 지하철을 타고 이동해야 했던 곳을 간편하고 저렴하게 이동할 수 있다는 점에서 사람들은 전동 킥보드를 이용한다. 다만, 목적지에 도착하거나 전동 킥보드를 보관하기 어려운 장소에 가면, 전동 킥보드는 짐이 되기 마련이다. 따라서 사람들은 언제나 편리하게 대여와 반납이 가능한 전동 킥보드 공유 서비스를 선호한다. 공유 킥보드 업체는 증가하는 사용자들의 이용 흐름에 맞춰 킥보드를 효율적으로 수거, 분배하여 이익을 극대화하여야 한다. 이에 따라 사용자들의 흐름을 유동적으로 분석하고 최적의 분배 계획을 업데이트 할 수 있는 시스템이 필요하다.

1.2.2. Problem Statement

학생들은 대부분의 시간을 이동하는데 보낸다. 대부분의 대학들은 면적이 넓기 때문에, 캠퍼스 안에서도 학생들은 수업을 듣기 위해 이동하는데 많은 시간을 보내야 한다. 수업과 수업 사이에 매점에 들리거나 프린트를 하는 것은 대부분의 경우 불가능하다. 혹여 건물과 건물 사이의 거리가 멀 경우, 이동하는 것 만으로도 충분히 지각할 가능성도 존재한다. 공강 시간에 학교 주변에서 밥을 먹거나 어딘가로 가야할 때, 학생들은 항상 서둘러야 한다.

공유 킥보드 업체는 수익을 극대화하기 위해 최대한 많은 사용자에게 킥보드를 이용할 수 있도록 해야 한다. 다만 사용자의 대부분이 불규칙한 패턴을 가지고 이동하거나, 모두 비슷한 방향으로 이동하는 경우 킥보드들은 오랜 시간 방치될 수 있다. 특히 캠퍼스의 경우, 킥보드가 인적이 드문 곳에 반납될 경우, 다음 사용자가 이용하기까지 오랜 시간이 걸려 효율이 떨어진다.

현재 존재하는 킥보드 공유 서비스 앱은 특정 지역을 대상으로 맞춤화 된 서비스를 제공하지 못한다. 특정 지역 내의 인구 이동 흐름과 무관하게 킥보드를 배치, 분배하기 때문에 킥보드의 분포가 고르지 못해 사용자들이 많이 사용할 수 없는 경우가 많으며, 킥보드 사용 효율이 떨어지는 문제가 발생한다.

1.2.3. Product Position Statement

우리의 서비스는 전동 킥보드 공유 서비스 업체와 대학생들에게 매우 유용하다. 공유 업체는 대학교를 목표로 할 수 있다. 캠퍼스는 매 학기마다 학생들이 규칙적인 시간표를 가지고 반복적으로 이동하기 때문에 학생들의 이동이 특정한 패턴을 가지게 된다. 공유 업체는 우리의 시스템을 사용하여 킥보드를 적절한 장소에 분배하고 관리하여 최대한 많은 학생들이 이용하도록 할 수 있다. 요일마다 시간대별로 학생들의 수업 일정은 반복되기 때문에, 학생 이동이 많은 건물을 기준으로 킥보드를 분배할 수 있다. 또한 학생들은 자신의 캠퍼스에 특화되어 관리되는 킥보드 서비스를 이용함으로써, 보다 더 편리하게 공유 킥보드를 사용할 수 있다. 따라서 우리의 서비스는 학교라는 특정 지역의 인구 이동 흐름을 분석하여 그에 맞는 킥보드 공유 계획을 산출해주어, 사용자는 편리한 이용을, 관리자는 효율 및 수익성의 극대화를 기대할 수 있다.

1.3. Stakeholder Descriptions

1.3.1. User summary

- 학생: 역 혹은 버스 정류장에서 학교로 접근이 힘든 사람, 수업 듣는 강의실 간의 거리가 먼 학생, 학교 주변을 이동할 일이 많은 학생.

- 킥보드 매니저: 학교 캠퍼스를 대상으로 공유 킥보드 사업을 원하는 사람, 공유 킥보드 사업의 효율성을 극대화하여 수익 구조를 개선하고 싶은 사람

1.3.2. Non-User summary

- 시스템 관리자: 프로그램의 유지 보수가 가능한 사람, 사용자 정보의 보안을 책임질 수 있는 사람, 알고리즘 설계 및 관리가 가능한 사람

1.3.3. User-Level Goals

- 학생: 면허 등록, 킥보드 대여, 킥보드 반납, 포인트 충전,

- 킥보드 매니저: 알고리즘 계산 결과 확인, 킥보드 상태 확인, 킥보드 위치 정보 접근, 사용자 정보 확인

1.3.4 User Environment

- 시스템은 사용하기 편리해야 한다.

- 모든 사용자는 킥보드의 현재 위치에 접근이 가능해야 한다.

- 사용자의 모바일 기기는 어플리케이션이 설치되기 적합한 환경이야 한다.

- 사용자는 인터넷이 연결된 환경이다.

1.4. Product Overview

1.4.1. Product Perspective

Kick Share! 시스템은 캠퍼스에 최적화된 전동 킥보드 공유 서비스를 통해 학생들이 건물 간 이동시간을 단축하고 편리하게 이동할 수 있도록 한다. 전동 킥보드는 캠퍼스 내부와 사전 협의가 완료된 캠퍼스 주변에서만 사용 가능하고, 전동 킥보드의 보관소는 각 건물의 출입구와 캠퍼스 주변에 설치된다. 학생들은 QR Code를 통해 보관소에서 전동 킥보드를 대여 또는 반납하고, Banking Program을 통한 결제로 전동 킥보드를 이용할 수 있는 포인트를 충전할 수 있다. 또한 Kick Share! 시스템을 외부 시스템과 연동하여 운전면허증의 유효성을 확인하고, 전동 킥보드의 위치를 실시간으로 확인할 수 있다.



<그림1. Kick Share! 시스템 컨텍스트 다이어그램>

1.4.2. Assumptions and Dependencies

- Kick Share! 시스템은 학교 총장님과 시청에 승인이 되어있다.

- 전동 킥보드는 캠퍼스 내부와 사전 협의가 된 캠퍼스 인근에서만 사용 가능하다.

- 이 시스템을 사용하는 User는 모두 운전면허를 갖고 있다.

- 운전면허가 없는 사람은 회원가입을 할 수 없다.

- User는 한 번에 한 개의 전동 킥보드만 대여 가능하다.

- 전동 킥보드의 보관소 위치와 전동 킥보드와 관련된 정보는 데이터베이스에 저장되어있다.

- 모든 전동 킥보드의 보관소와 각각의 전동 킥보드에는 고유한 QR Code가 부여되어 있다.

- 전동 킥보드가 대여 중일 때에는 GPS를 이용하여 실시간으로 전동 킥보드의 위치를 업데이트한다.

- System은 항상 GPS를 이용하여 전동 킥보드의 위치를 실시간으로 추적 가능하다.

- 페널티를 받은 사람은 시스템 이용에 제한이 있다.

- 전동 킥보드는 두 가지 상태(사용 중, 대여 가능)를 가진다.

- 전동 킥보드는 항상 네트워크와 연결되어 있다.

1.5. Summary of System Features

- 회원 가입 시 운전면허증 유효성 확인

- User 정보 관리 (포인트 확인, 킥보드 대여 상태, 페널티 확인)

- Banking Program으로 포인트 충전 및 차감

- 각각의 전동 킥보드는 모두 고유한 QR Code를 가짐

- GPS를 이용하여 전동 킥보드의 위치, 킥보드 보관소의 정보 (남아있는 킥보드의 수와 배터리 상태) 실시간 업데이트

- 전동 킥보드의 사용 패턴과 이용량을 분석하는 시스템 내부의 분배 알고리즘 구현

1.6. Other Requirements and Constraints

- 애플리케이션 UI는 사용하기 쉽고 간단하며, 보기 좋아야 한다.

- 모든 학생들이 운전면허증을 갖고 있는 것은 아니다.

- 전동 킥보드 대여 중에 사고가 발생할 경우, 관련 법안이 제정되어 있지 않기 때문에 보상 기준을 정하기가 어렵다.

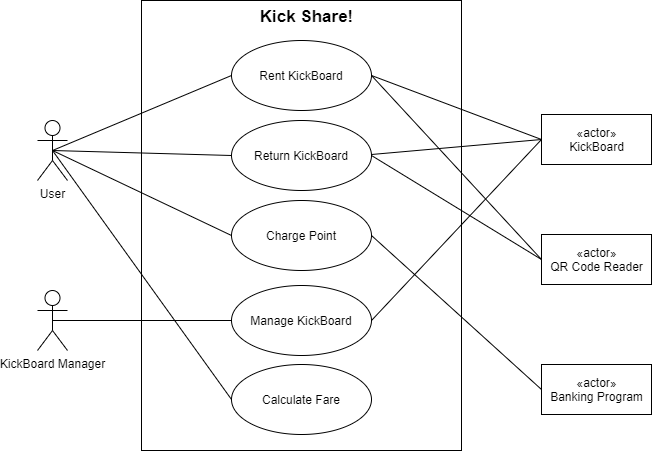
- 이 시스템에 회원 가입한 사람이 회원이 아닌 다른 사람에게 전동 킥보드를 빌려주는 경우, Kick Share! 시스템에서 확인할 방법이 없다.

- 전동 킥보드의 네트워크 통신이 끊긴 상태로 발생한 문제에 대해서는 Kick Share! 시스템에서 책임질 수 없다.

2. Requirements

2.1. Function Requirements

2.1.1. Use Case Diagram



<그림2. Use case Diagram>

2.1.2. Use Case Brief

2.1.2.1. <Use Case 1> Rent Kickboard

User가 전동 킥보드를 대여하기 위해서는, Kick Share! 시스템에 회원으로 등록되어 있어야 하고, 킥보드 보관소에는 킥보드가 존재해야 한다. 전동 킥보드 보관소는 학교 내부를 포함한 학교 주변 지역에 설치되어 있으며, User는 보관소에서 전동 킥보드를 대여할 수 있다. 시스템이 User가 사용할 전동 킥보드를 알아야 하므로 User는 전동 킥보드에 부착된 QR코드를 애플리케이션을 통해 인식해야 한다. 시스템에서는 인식된 QR코드로부터 해당 전동 킥보드가 사용 가능한 상태인지 확인한다. 킥보드가 ‘대여 가능’상태라면, 시스템은 User의 계정에 일정 금액 이상의 포인트가 존재하는지 확인하고 일정 금액조차 없다면 User에게 사용 불가 알림과 함께 포인트 충전을 안내한다. User계정에 일정 포인트 이상이 존재한다면 User는 전동 킥보드를 사용할 수 있다. 시스템은 전동 킥보드와 User의 상태를 사용 중으로 바꾸고 User의 사용 위치, 시간 등을 기록한다. 시스템은 사용자의 이용 시간을 기준으로 실시간으로 이용 요금을 계산한다. 만약 이용 요금이 사용자의 잔여 포인트를 초과한다면 시스템은 유저에게 알림을 보내고 유저의 킥보드 이용을 중단시킬 수 있다.

2.1.2.2. <Use Case 2> Return KickBoard

User는 전동 킥보드 보관소 중 아무 곳에서나 반납하여 킥보드의 사용을 종료하는 것이 가능하다. 보관소에서 User는 킥보드의 QR코드를 인식시킨다. 킥보드는 반드시 보관소에서 보관되어야 하므로 시스템에서는 해당 전동 킥보드가 실제로 보관소에 위치하고 있는지 확인한다. 킥보드가 보관소에 위치하고 있다면, 시스템은 현재 시간과 대여를 시작한 시간을 이용하여 User가 지불해야 할 요금(포인트)을 계산한다. 시스템은 User의 계정에서 포인트를 차감하고 킥보드 반납 처리를 완료한다. 반납 처리가 완료되면, 시스템은 User와 킥보드의 상태를 대여 가능으로 변경한다.

2.1.2.3. <Use Case 3> Charge Point

포인트는 현금으로 결제하는 방식으로 진행되며, 이 시스템 내부에서만 사용 가능한 화폐 개념이다. 사용자의 계정에 일정 금액 이상의 포인트가 없다면 사용자는 킥보드를 대여할 수 없다. 따라서 킥보드를 대여하기 전에 최소 금액의 포인트를 충전(결제)해야 한다. 포인트 충전은 외부 결제 대행 서비스를 이용하여 진행된다.

2.1.2.4. <Use Case 4> Manage Kickboard

시스템은 사용자들이 킥보드 사용을 시작한 위치, 사용을 끝낸 위치 등을 시간 별로 기록한다. 그리고 킥보드의 배터리 잔량을 실시간으로 모니터링 한다. 시스템은 기록된 데이터와 현재 킥보드의 위치, 배터리 상태를 고려하여 킥보드 매니저의 킥보드 수거 및 분배 계획 수립 알고리즘을 계산한다. 알고리즘의 계산 결과를 토대로 킥보드 매니저는 킥보드를 수집하며 충전이 필요한 킥보드는 충전을, 사용 가능한 킥보드는 미리 사용자들이 많은 곳으로 분배한다. 만일 사용자가 킥보드를 타고 정해진 구역을 벗어 났을 때는 시스템이 이를 킥보드 매니저에게 알리고, 킥보드 매니저는 해당 유저에게 킥보드 반환을 요청한다. 필요 시, 킥보드 매니저는 시스템을 통해 사용 지역을 벗어난 유저의 킥보드 작동을 중단시킬 수 있다.

2.1.2.5. <Use Case 5> Calculate Fare

사용자는 KickBoard를 사용중 일 때, 시스템에게 현재까지의 사용요금을 요청한다. 따라서 사용자는 point의 충전 여부를 미리 결정할 수 있다. 시스템은 사용자가 서비스 이용을 시작한 시간을 기준으로 요금 계산을 요청한 시간까지의 요금을 계산해 보여준다.

2.1.3. Use Case Text

2.1.3.1. <Use Case 1> Rent KickBoard

|  |
| --- |
| **Use case uc1:** Rent KickBoard  **Scope:** Kick Share!  **Level:** User goal  **Primary Actor:** System User  **Stakeholders and Interests:**  - User: 수업에 늦었을 경우나 다음 수업 건물이 멀리 있는 경우, 학교 주변을 이동할 일이 생긴 경우 킥보드를 이용해 편리하게 이동하고 싶다.  **Preconditions:**  - User는 Kick Share! 애플리케이션을 설치하고 회원가입을 완료한 상태이다.  - User는 로그인이 완료된 상태이다.  - 전동 킥보드 보관소에는 충분히 많은 전동 킥보드가 있다.  - User의 계정에는 최소한의 일정 포인트가 충전되어 있다.  **Postconditions:**  - 지정된 보관소에서 전동 킥보드를 반납할 수 있다.  - 유저가 킥보드를 빌리기 시작한 시간이 저장됐다.  **Main Success Scenario**  1. User는 애플리케이션을 통해 계정에 잔여 포인트가 있는지, User가 대여하고자 하는 보관소에 대여 가능한 전동 킥보드가 있는지 확인한다.  2. User는 보관소로 이동하여 QR코드를 인식한다.  3. 시스템은 사용자의 계정에 잔액이 충분하며, 해당 전동 킥보드가 사용 가능한 상태인지 확인한다.  4. 시스템은 사용자와 킥보드를 사용중인 상태로 변경한다.  5. 시스템은 이용 시작 시간 정보를 저장한다.  **Extension**  1a. 계정에 잔여 포인트가 부족함.  1. 사용자는 대여 서비스 이용을 할 수 없다.  1b. 보관소에 대여 가능한 킥보드가 없음.  1. 사용자는 그 다음으로 가까운 보관소로 이동하여 킥보드를 대여한다.  3a. 사용자의 잔액이 최소 이용 가능 금액 미만이다.  1. 시스템은 사용자의 대여 요청을 거절한다.  2. 시스템은 최소 이용 포인트가 부족하다는 메시지를 띄운다.  3b. 해당 전동 킥보드가 사용 가능한 상태가 아님.  1. User는 다른 킥보드의 QR코드를 인식시킬 수 있다.  5a. 실시간 이용 요금이 사용자의 잔여 포인트를 초과한다.  1. 시스템은 유저에게 포인트 부족 안내 메시지를 띄운다.  2. 시스템은 해당 사용자의 킥보드 이용을 중단시킨다.  **Special Requirements**  - QR코드 시스템을 사용하기 위해서는 외부 API가 필요하다.  - 포인트 결제는 외부 결제 수단을 사용한다.  - QR코드 시스템은 항상 사용 가능해야 한다.  - 킥보드의 GPS정보는 항상 사용 가능해야 한다.  **Technology and Variations list**  - 대여 완료 시 킥보드의 상태정보(사용중/대여가능)를 서버에 업데이트.  **Frequency of Occurrence**  - 시스템은 24시간 운영된다.  - 포인트 충전 과정은 외부 시스템을 사용하므로 점검 시간이 발생할 수 있다.  **Open Issue**  - 믿을 만한 User인지 어떻게 확인할 것인가? |

2.1.3.2. <Use Case 2> Return Kickboard

|  |
| --- |
| **Use case uc2:** Return KickBoard  **Scope:** Kick Share!  **Level:** User goal  **Primary Actor:** System User  **Stakeholders and Interests:**  - User: 지불해야할 킥보드 이용 요금을 알고 싶다. 킥보드 보관소의 위치를 알고 싶다.  **preconditions:**  - 킥보드 대여 과정이 성공적으로 완료되었다.  **postconditions:**  - User의 계정에서 포인트가 차감된다.  - 킥보드의 상태가 대여 가능으로 변경된다.  **Main Success Scenario**  1. User는 목적지 근처의 킥보드 보관소로 간다.  2. User는 보관소에서 킥보드에 있는 QR코드를 인식시킨다.  3. 시스템은 킥보드의 위치가 보관소 중 한 곳이 맞는지 확인한다.  4. 시스템은 QR코드를 인식한 시간 정보를 통해 현재 시간과 시스템에 기록된 대여 시작 시간을 이용하여 이용 시간과 차감 포인트를 계산한다.  5. 시스템은 User의 계정에서 계산된 포인트만큼 차감한다.  6. 시스템은 해당 킥보드를 대여 가능한 상태로 전환시킨다.  **Extension**  3a. 킥보드의 위치가 보관소가 아님.  1. 시스템은 반납 서비스를 거절한다.  2. 시스템은 User에게 반납은 보관소에서만 가능하다는 메시지를 띄움.  3. User는 보관소로 이동하여 반납 절차를 진행할 수 있다.  **Special Requirements**  - QR코드 시스템을 사용하기 위해서는 외부 API가 필요하다.  - 포인트 결제는 외부 결제 수단을 사용한다.  - QR코드 시스템은 항상 사용 가능해야 한다.  - 킥보드의 GPS 정보는 항상 사용 가능해야 한다.  **Technology and Variations list**  - 반납 완료 시 킥보드의 상태정보(사용 중/대여 가능)를 서버에 업데이트.  **Frequency of Occurrence**  - 시스템은 24시간 운영된다.  - 포인트 충전 과정은 외부 시스템을 사용하므로 점검 시간이 발생할 수 있다.  **Open Issue**  - GPS가 킥보드의 잘못된 위치 정보를 제공한다면 어떻게 반납을 진행할 것인가?  - User가 포인트를 충전할 수 없는 상황이라면 어떻게 할 것인가? |

2.1.3.3. <Use Case 3> Charge Point

|  |
| --- |
| **Use case uc3:** Charge Point  **Scope:** Kick Share!  **Level:** User goal  **Primary Actor:** Kickboard Manager  **Stakeholders and Interests**   * User: kickboard를 이용하기 위해 최소의 point를 충전하기를 원한다.   **Pre-conditions:**   * 외부 결제 시스템이 안정적인 상태여야 한다.   **Postconditions:**  - 사용자의 계정에 point가 충전된다.  **Main Success Scenario**   1. 사용자가 포인트 충전을 누르고 충전할 금액을 선택한다. 2. 외부 결제 시스템이 로딩되며 사용자는 결제를 진행한다. 3. 시스템은 결제한 금액 만큼 사용자 계정에 포인트를 충전시킨다.   **Extension**  \*a. 외부 결제 시스템에 문제가 생긴경우  1. 결제를 이용할 수 없거나 다른 결제 시스템을 이용한다.  **Special Requirements**  - 충전 금액은 ‘최소 포인트’이상이어야 한다.  **Technology and Variations list**  - 외부 결제 시스템을 다양하게 수용한다.  **Frequency of Occurrence**  - 사용자가 충전을 원할때, 또는 잔여 포인트가 부족할 때.  **Open Issue**  - 다른 모든 외부 결제 시스템에 오류가 생긴다면 어떻게 해야할 것인가? |

2.1.3.4. <Use Case 4> Manage Kickboard

|  |
| --- |
| **Use case uc4:** Manage Kickboard  **Scope:** Kick Share!  **Level:** User goal  **Primary Actor:** Kickboard Manager  **Stakeholders and Interests**   * User: 이 Use Case에 직접적인 영향은 주지 않지만 킥보드가 필요할 때 항상 근처에 있기를 원할 것이다. 또한 킥보드의 정확한 위치를 알고 싶어 할 것이다. * Kickboard Manager: 학생들의 이동 흐름에 맞춰 킥보드를 분배하여 최대한 많은 학생이 킥보드를 이용할 수 있도록 하고 싶다.   **Pre-conditions:**   * KickBoard Manager는 Kick Share! 시스템에 ‘관리자 권한’으로 로그인을 한 상태여야 한다.   **Postconditions:**  - 사용자의 유동이 많은 곳에 킥보드가 배치된다.  **Main Success Scenario**   1. KickBoard Manager는 주기적으로 시스템에게 킥보드의 위치 분배 요청한다. 2. 시스템은 킥보드의 위치 정보와 학생 이동 흐름 패턴을 이용해 위치 분배 알고리즘을 실행한다. 3. 시스템은 외부 시스템인 KickBoard에 요청을 보내 배터리가 부족한 킥보드를 찾아낸다. 4. 시스템은 분배 결과와 배터리가 부족한 킥보드를 KickBoard Manager에게 알린다. 5. 킥보드 매니저는 해당 알람을 받고 KickBoard를 수거/분배, 충전을 한다.   **Extension**  \*a. Sharing Kickboard의 시스템에 계속 문제가 생겼을 경우.  1. 킥보드 매니저는 시스템을 다시 시작할 수 있다.  2. 킥보드 매니저는 모든 킥보드를 수거해 올 수 있다.  2a. 특정 킥보드의 위치가 정해진 장소가 아닌 다른 장소에서 확인된다.  1. 킥보드 매니저는 시스템으로부터 알람을 받아 해당 킥보드를 회수해 올 수 있다.  2. 킥보드 매니저는 해당 킥보드를 사용하고 있는 유저에게 반환 요청을 할 수 있다.  3. 킥보드 매니저는 킥보드를 사용중인 유저의 사용을 중지시킬 수 있다.  2b. 특정 킥보드의 위치가 확인이 안될 경우.  1. 해당 킥보드가 마지막으로 확인된 위치를 확인해 킥보드를 찾는다.  2. 해당 킥보드를 마지막으로 사용한 사용자에게 알람을 전송한다.  **Special Requirements**  - 우리 시스템은 외부 시스템인 KickBoard로 킥보드의 정보를 받아 올 수 있어야 한다.  **Technology and Variations list**  - 시스템은 외부 시스템인 KickBoard와 소통하기 위해 통신 및 네트워킹이 가능해야 한다.  **Frequency of Occurrence**  - 주기적인 간격(예. 하루에 두 번)을 두고 실행한다.  **Open Issue**  - 사용자의 위치 정보를 가지고 오는 것이 문제가 되지 않는가?  - 알고리즘의 정확도는 믿을 수 있는가?  - 외부 시스템인 KickBoard와 네트워킹이 불안정 한다면 어떻게 해야 하는가? |

2.1.3.5. <Use Case 5> Calculate Fare

|  |
| --- |
| **Use case uc5: Calculate Fare**  **Scope:** Kick Share!  **Level:** System goal  **Primary Actor:** User  **Stakeholders and Interests**   * User: 이용한 요금을 확인하고 싶다.   **Pre-conditions:**   * 사용자는 성공적으로 킥보드를 대여했다. * 사용자가 킥보드를 빌리기 시작한 시간이 시스템에 저장돼 있다.   **Postconditions:**  - 사용자의 이용 요금이 계산된다.  - 사용자의 이용 요금이 보유 포인트로부터 차감된다.  - 킥보드가 성공적인 반납을 위한 조건을 갖춘다.  **Main Success Scenario**   1. 시스템이 사용자의 현재까지 사용한 요금의 계산을 요청한다. 2. 시스템이 사용자의 Time info로 부터 이용 시작 시간을 확인한다. 3. 시스템이 이용 시작 시간으로부터 요금 계산 요청 시간까지의 요금을 계산한다. 4. 계산된 요금을 반환한다.   **Extension**  -  **Special Requirements**  - 사용자의 이용 시작 시간에 접근할 수 있어야 한다.  **Technology and Variations list**  - 시스템은 현재 킥보드의 정보를 업데이트 하기 위해 주기적으로 DB에 접근한다.  **Frequency of Occurrence**  -  **Open Issue**  - 킥보드의 GPS가 정상 작동 하는가?  - 킥보드에 부착된 시스템이 정상 작동 하는가?  - 킥보드의 통신 모듈과 실시간으로 정상 연결 가능한가? |

2.2. Non-Functional Requirements

**Functionality**

- 오류가 발생할 경우 프로그램은 데이터를 저장하지 않는다.

- 모든 기능을 사용하기 이전에 사용자 인증이 필요하다.(precondition)

- 앱 사용 중에 오류가 발생하면, 이전 페이지로 다시 로드된다.

- 킥보드의 정보가 외부 데이터베이스로 네트워크를 통해 안전하게 전달된다.

- 서버는 추후의 기능추가 및 서비스 확장을 위해 client의 local데이터를 적기에 갱신할 수 있어야한다.

**Usability**

- 사용자를 위한 애플리케이션 설계는 명확하고 단순해야 한다.

**Reliability**

- 시스템과 킥보드 사이의 데이터 전달은 정확해야 한다.

- 데이터 전송을 위해 통신 환경이 원활 해야 한다.

- 데이터를 전송하는 동안 끊어짐이 없어야 한다. 데이터가 변경되거나 파괴되는 경우, 데이터 전송을 그만 두어야 한다.

- 사용자가 본인의 localDB를 조작할 수 없도록 한다.

- 서로 다른 노드에 있는 서버와 client간의 네트워크 오류에 대한 회복방안이 정의되어야 한다.

**Performance**

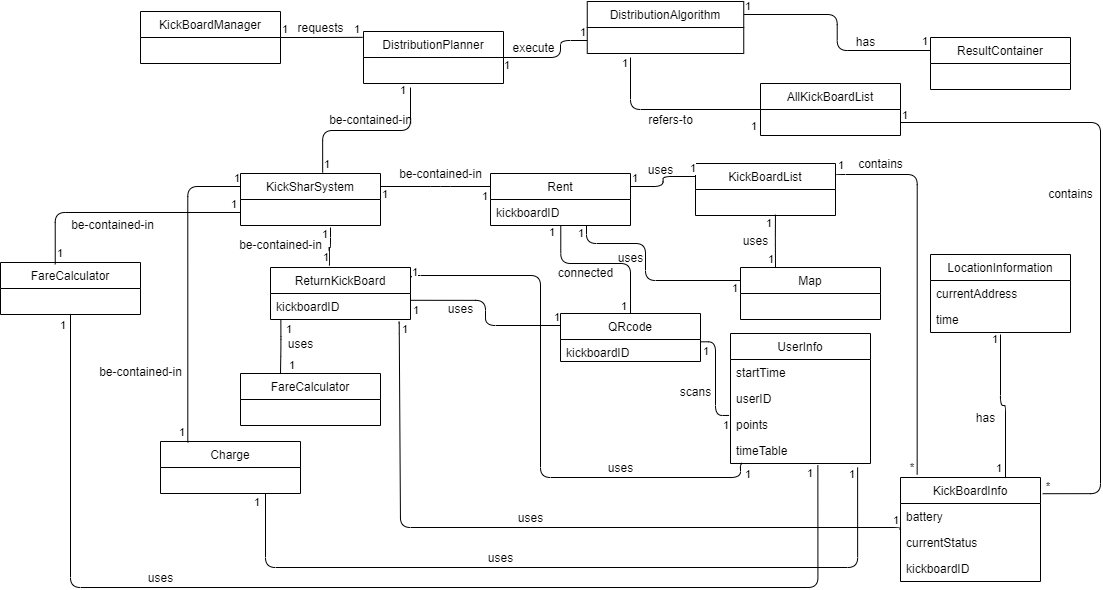
* 유저가 사용하려는 킥보드의 정보는 1초 이내에 응답해 킥보드의 사용을 알린다.
* 사용자의 정보를 서버의 데이터베이데이부터 client로 가져오는데 0.5초 내로 진행되어야 한다.

**Supportability**

* 이 애플리케이션은 여러 학교에서 사용 가능하다. 자체 학습 알고리즘이 있고, 알고리즘과 데이터베이스는 사용자의 요건에 따라 변경이 가능하다.
* IOS, Android 등과 같이 Kick Share! 애플리케이션에 대해 다양한 모바일 OS를 원하는 고객이 있다. 따라서 애플리케이션은 이러한 요구를 반영해야 하므로 다양한 구동 환경의 분석이 필요하다.

3. Analysis Modeling

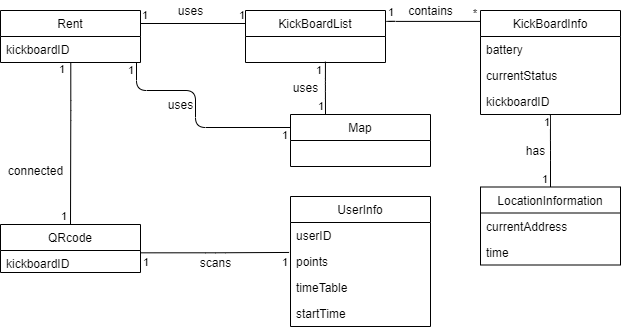
3.1. Domain Model Diagram



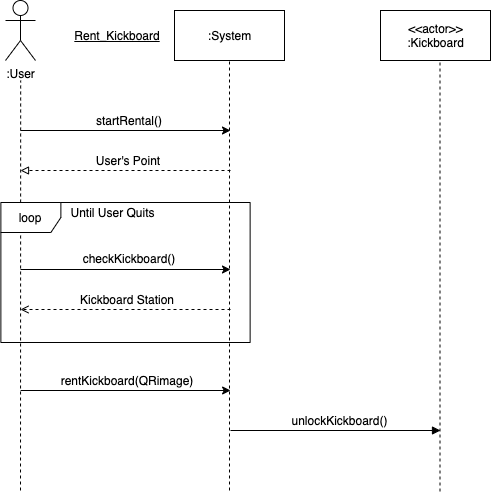
3.2. System Sequence Diagram

3.2.1. <Use Case 1> Rent Kickboard

3.2.1.1 Domain model

****

3.2.1.2. SSD

****

**System Operation List**

- startRental() : 사용자가 대여 메뉴에 들어간다. 시스템은 외부 DB에 접근하기 위한 Adapter를 Factory를 통해 받아온다. 해당 Adapter object는 이후에도 계속적으로 사용된다.

- checkKickboard() : 사용자가 Map interface를 통해 학교 부지 내에 사용 가능한 킥보드의 위치를 확인한다.

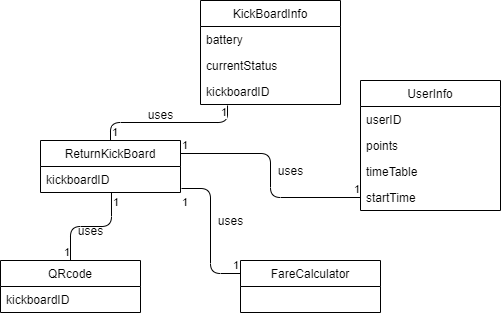
- rentKickboard(QRimage) : 사용자가 시스템에게 대여를 요청한다. 시스템은 사용자의 사용 가능한 포인트를 알려준다. 사용자가 킥보드에 부착되어 있는 QR code를 스캔한다. 시스템이 사용자의 대여 자격을 확인하고 사용자에게 킥보드를 대여해주며, 해당 킥보드는 대여 불가능 상태로 변경한다.

3.2.1.3. Operation Contracts

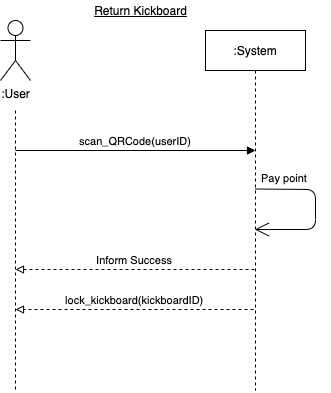
|  |
| --- |
| * Operation : startRental() * Cross References : Rent Kickboard * Precondition : None * Postcondition :   + UserInfo instance uii가 생성된다. |
| * Operation : checkKickboard() * Cross References : Rent Kickboard * Precondition :   + KickboardList가 비어있지 않다.   + KickboardInfo의 유효한 location이 존재한다. * Postcondition :   + Map instance mi가 생성된다.   + Kickboard의 location이 MapInterface와 association이 형성된다. |
| * Operation : rentKickboard(QRimage) * Cross References : Rent Kickboard * Precondition : 사용 가능한 Kickboard가 현재 station에 존재한다. * Postcondition :   + TimeInfo instance tii가 생성된다.   + tii.startTime이 현재 시간으로 초기화된다.   + 현재 UserInfo instance와 tii 간의 association이 형성된다.   + 현재 KickBoardInfo의 currentStatus가 ‘사용중’으로 변경된다.   + LocationInformation instance lii가 생성된다.   + 현재 KickBoardInfo의 currentAddress와 현재 시간이 lii의 attribute에 갱신된다. |

3.2.2. <Use Case 2> Return Kickboard

3.2.2.1. Domain Model

****

3.2.2.2. SSD



**System Operation List**

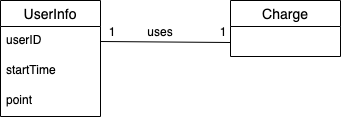
- scan\_QRcode(userID) : 사용자가 킥보드를 반납하기 위해 QR code를 스캔한다. 시스템이 대여를 요청한 사용자를 확인하고, 킥보드의 위치와 이용 시간 등을 확인하여 사용자의 계정에서 포인트를 차감한다. 그리고 사용자에게 포인트가 제대로 지불되었다고 안내하고, 해당 킥보드를 대여 가능 상태로 변경한다.

3.2.2.3. Operation Contracts

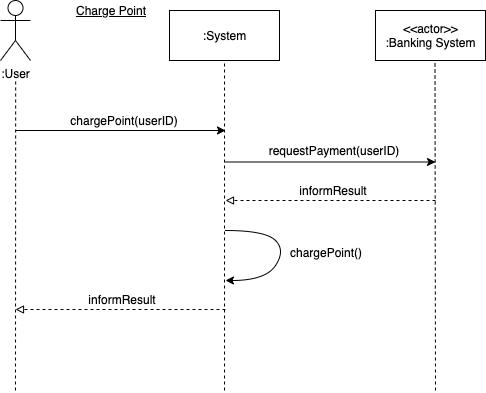
|  |
| --- |
| * Operation : scan\_QRcode() * Cross References : Return Kickboard * Precondition   + 사용자가 킥보드를 대여한 상태이다.   + 사용자는 요금을 지불하기에 충분한 포인트를 보유하고 있다.   + 사용자가 목적지 근처의 킥보드 보관소에 도착하였다. * Postcondition   + 시스템은 KickBoard의 currentStatus를 ‘대여 가능’으로 변경한다.   + User의 point attribute의 값이 갱신된다.   + GPSInformation과 KickkBoard의 association이 해제되며 GPSInformation의 정보를 시스템이 기록한다.   + KickBoardList가 갱신된 kickboard 객체로 업데이트된다.   + time info instance가 모두 삭제된다. |
| * Operation : free\_kickboard() * Cross References : Return Kickboard * Precondition : User의 point가 차감된 상태이다. * Postcondition : Kickboard Description instance가 삭제된다. |

3.2.3. <Use Case 3> Charge Point

3.2.3.1. Domain Model

****

3.2.3.2. SSD



**System Operation List**

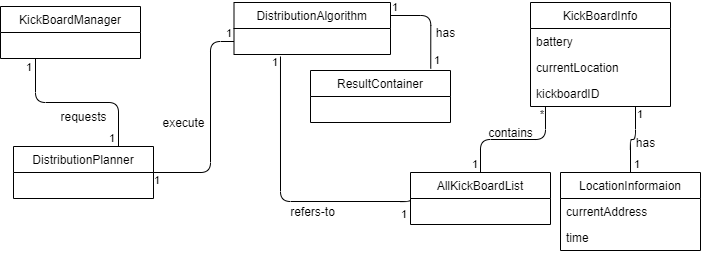
* chargePoint(userID) : 사용자가 포인트 충전을 요청한다. 시스템이 외부 결제 인터페이스 (Banking system)를 활용하여 사용자의 포인트를 충전하고 결과를 반환한다.

3.2.3.3. Operation Contracts

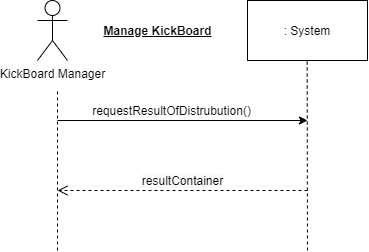
|  |
| --- |
| * Operation : chargePoint() * Cross References : Charge Point * Precondition :   + 사용자의 올바른 결제 정보가 시스템에 등록되어야 한다.   + 외부 시스템인 Banking system이 정상적으로 작동해야 한다. * Postcondition :   + PaymentInfo와 User의 association이 설정된다.   + 사용자의 point attribute가 업데이트 된다. |

3.2.4. <Use Case 4> Manage KickBoard

3.2.4.1. Domain Model

****

3.2.4.2. SSD



**System Operation List**

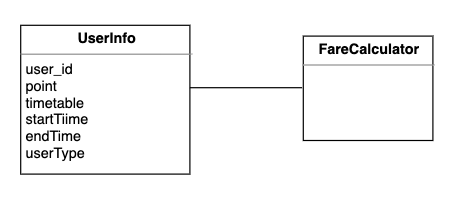
* requestResultOfDistribution() : KickBoard Manger가 System에게 킥보드 들을 배치할 새로운 정보를 요청한다. System은 분배 알고리즘을 적용하여 새로운 킥보드 배치 정보를 계산하고 KickBoard Manager가 확인할 수 있도록 한다.

3.2.4.3. Operation Contracts

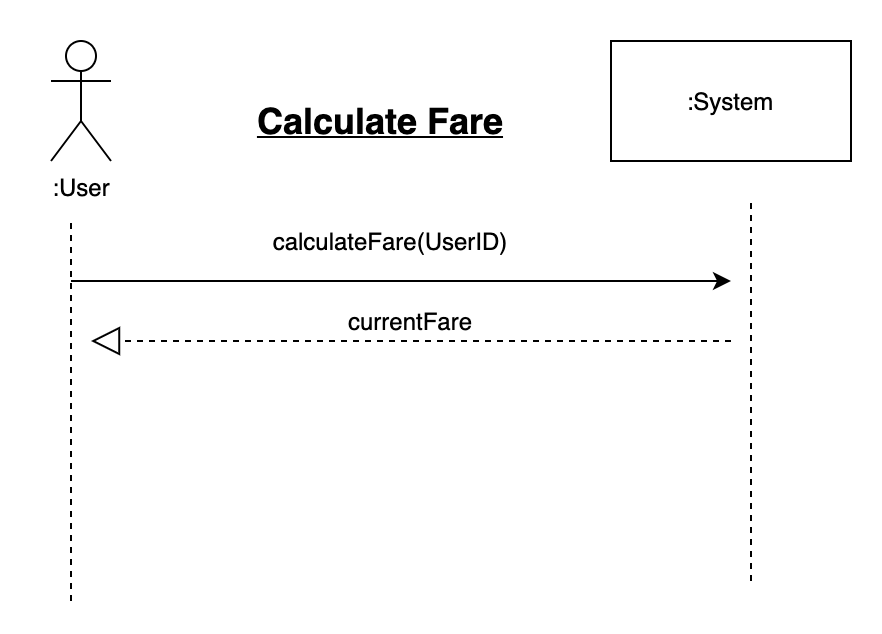
|  |
| --- |
| * Operation : requestResultOfDistribution() * Cross References : Manage Kickboard * Precondition :   + System Operation 요청자가 KickBoard Manager(권한이 있는 actor)이어야 한다.   + Kickboard Manager의 client가 네트워킹이 월할해야 한다. * Postcondition :   + DistributionPlanner instaince dpi가 생성된다.   + DistributionAlgorithm instance dai가 생성된다.   + dpi와 dai간의 association이 형성된다.   + AllKickBoardList instance aki가 생성된다.   + KickBoardInfo instance kii가 생성된다.   + aki와 kii간의 association이 형성된다.   + ResultContainer instance rci가 생성된다.   + dai와 rci간의 association이 형성된다. |

3.2.5. <Use Case 5> Calculate Fare

3.2.5.1. Domain Model

****

3.2.5.2. SSD

****

**System Operation List**

* calculateFare(userID) : 시스템이 사용자의 요청을 받아 요금을 계산한다. 시스템은 요청한 사용자의 userid를 이용해 서비스 이용 시작 시간과 user의 type에 맞는 분당 이용 요금 정보를 이용해 요금을 계산한다.

3.2.5.3. Operation Contracts

|  |
| --- |
| * Operation : calculateFare(userID) * Cross References : Calculate Fare * Precondition :   + User가 현재 킥보드를 대여한 상태이다.   + User의 startTime이 설정되어 있다.   + User는 회원 가입 시 자신이 속한 그룹을 알맞게 선택하였다. * Postcondition :   + FareCalculator instance fci의 fare attribute값이 (‘현재 시간 - startTime’ \* 분당 요금) 값으로 변경된다.   + 사용자에게 계산된 Fare 값이 반환된다. |

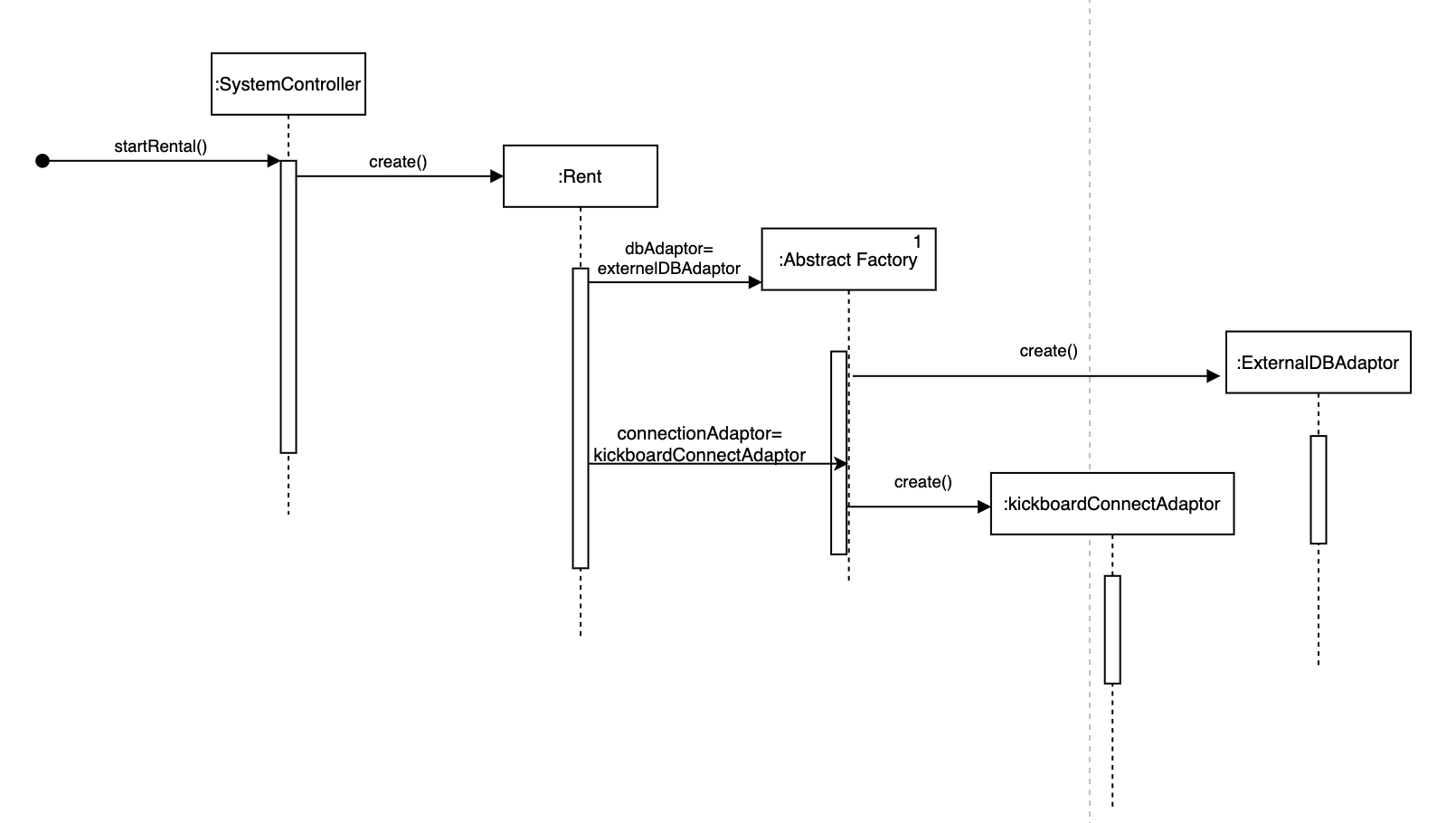
4. Design Modeling

4.1. <Use Case 1> Rent Kickboard Realization

4.1.1. Design Sequence Diagrams

4.1.1.1. System Operation 1: startRental()

* startRental() :Rental을 위한 시스템 내부 설정을 진행한다.

****

|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | Abstract Factory가 Adapter를 생성한다. |
| Information Expert | None |
| Low Coupling | Abstract Factory가 Adapter를 생성하는 책임을 가져 Rent에 대한 Low coupling을 실현한다. |
| High Cohesion | - SystemController는 system operation을 받는 responsibility를 갖는다. |
| Controller | SystemController는 모든 System input를 받는다. |
| Polymorphism | None |
| Pure Fabrication | Abstract Factory가 중간에서 DB와 Kickboard 와 소통하는 어댑터를 생성한다. |
| Indirection | None |
| Protected Variations | None |

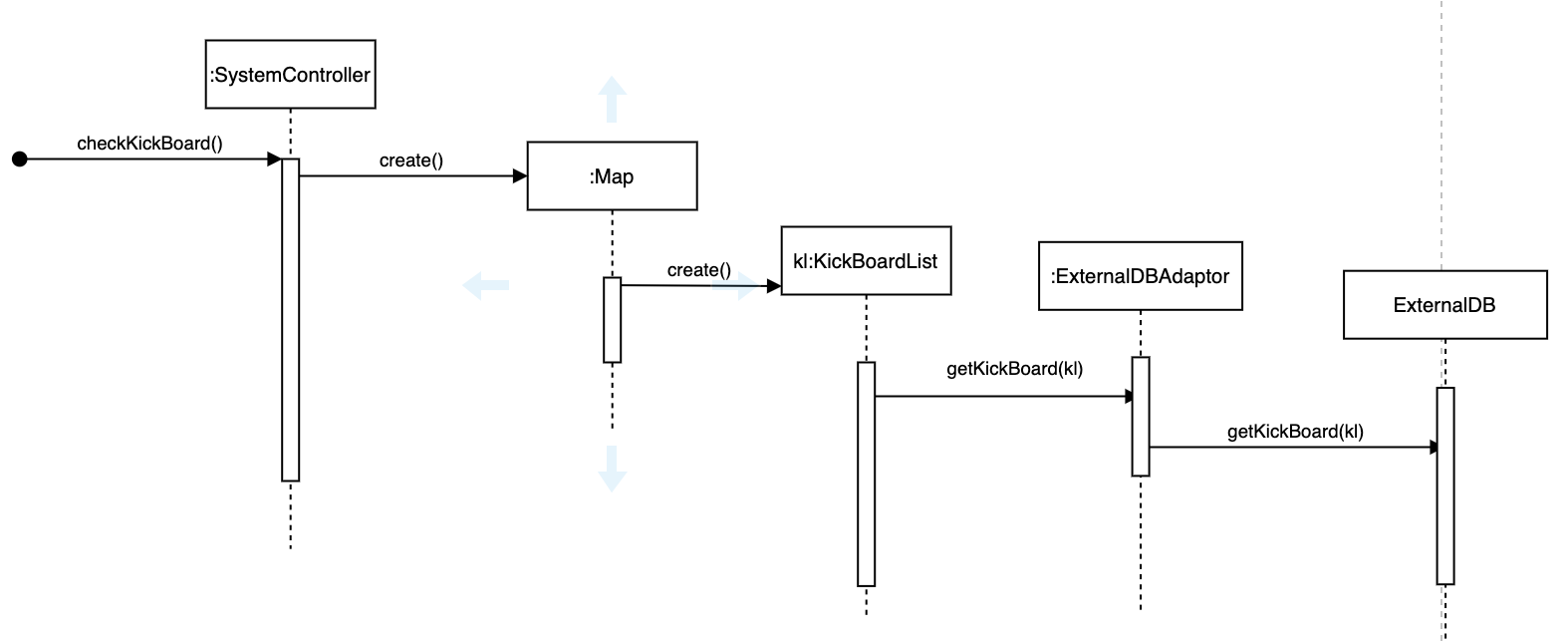
**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Factory | Factory를 이용해 어댑터를 생성하는 responsibility를 대신하게 한다. |

**<Gof Pattern>**

4.1.1.2. System Operation 2 : checkKickBoard()

* checkKickBoard() : 현재 사용자의 위치에서 사용 가능한 KickBoard의 위치를 조회한다.

****

**<checkKickBoard Interaction Diagram>**

|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | - 사용 가능한 KickBoard들의 정보를 사용자의 지도에 표시할 책임을 갖는 Map object가 KickBoardList instance를 생성한다. |
| Information Expert | KickBoardInfo의 정보를 모두 알고 있는 KickBoardList가 Map에 List를 보낸다. |
| Low Coupling | Map은 KickBoardList를 갖음으로써 여러개의 KickBoardInfo에 대한 coupling을 줄인다. |
| High Cohesion | - SystemController는 system operation을 받는 responsibility를 갖는다.  - KickBoardList는 KickBoardInfo를 갖는 책임이 있다. |
| Controller | SystemController는 모든 System input를 받는다. |
| Polymorphism | None |
| Pure Fabrication | DBHandler라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility만을 갖는다. |
| Indirection | Map은 KickBoardInfo에 대한 리스트를 직접 갖지 않으며 KickBoardList 라는 intermediate object가 존재한다. |
| Protected Variations | None |

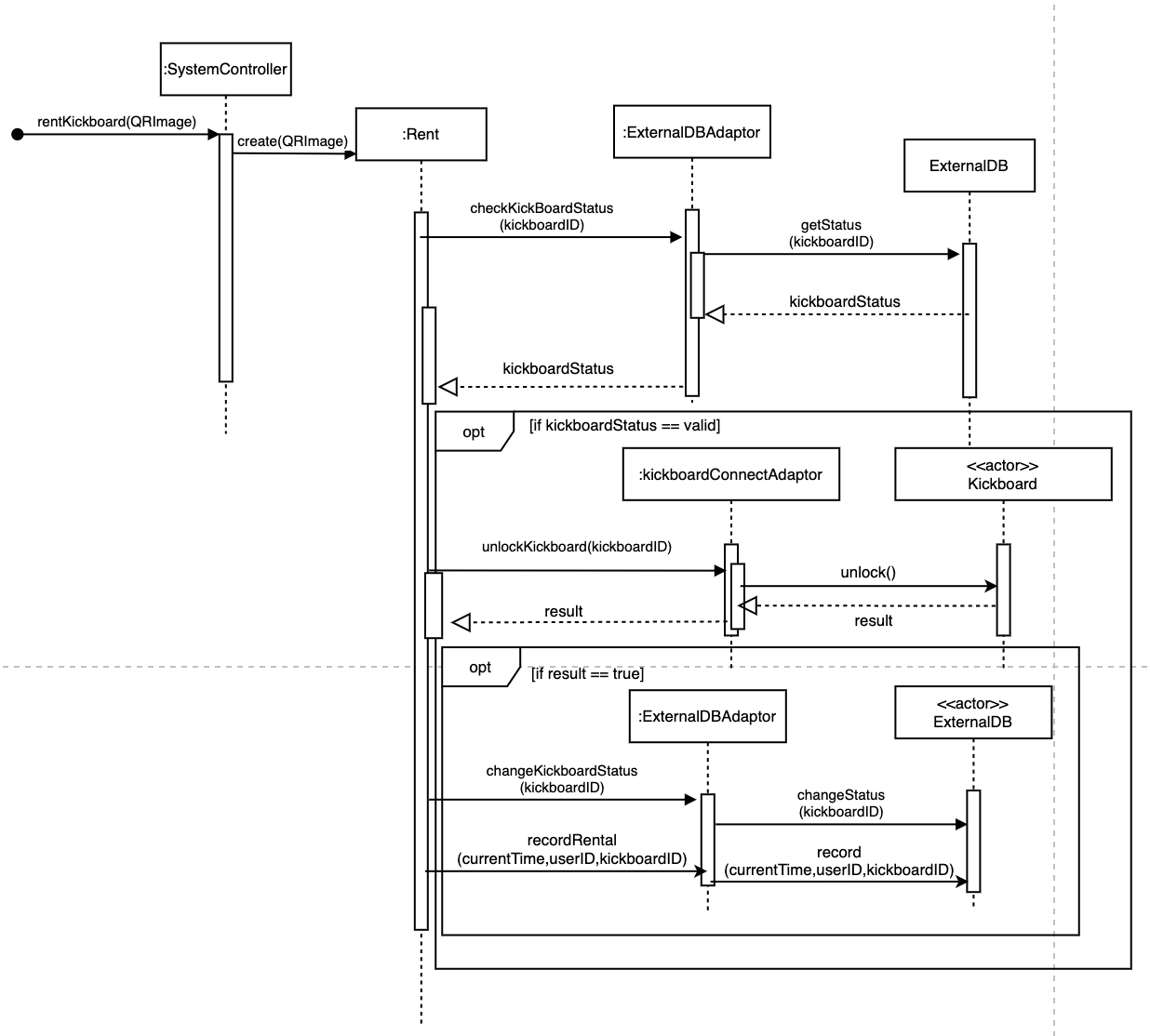
**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 DBHandler라는 Adapter을 가지고 있다. DBHandler는 actor의 역할을 하는 external database의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |

**<Gof Pattern>**

4.1.1.3. System Operation 3 : rentKickboard()

* rentKickboard() : 사용자가 KickBoard에 부착된 QRcode를 인식시키면 시스템에게 메시지가 전달된다.



**<rentKickboard Interaction Diagram>**

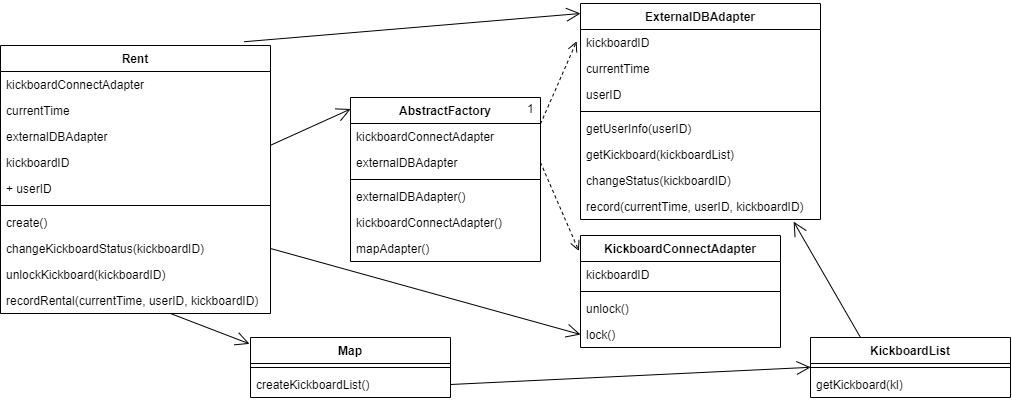
|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | - |
| Information Expert | QRCode를 통해 KickboardID를 알고 있는 Rent 객체가 ExternalDBHandler에게 Message를 보낸다. |
| Low Coupling | Rent가 직접 External DB에 메세지를 보내지 않고 DB Adapter를 이용한다. |
| High Cohesion | SystemController는 system input만을 받는 responsibility만을 갖는다. |
| Controller | SystemController는 system input을 받는다. |
| Polymorphism | None |
| Pure Fabrication | ExternDBAdapter라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility만을 갖는다.  ExternDBAdapter라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility만을 갖는다. |
| Indirection | Rent 객체와 ExternDB 사이의 직접적인 Coupling을 피하기 위해 Di를 사용한다. |
| Protected Variations |  |

**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 DBHandler라는 Adapter을 가지고 있다. DBHandler는 actor의 역할을 하는 external database의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |

**<GoF Pattern>**

4.1.1.4. Design Class Diagram

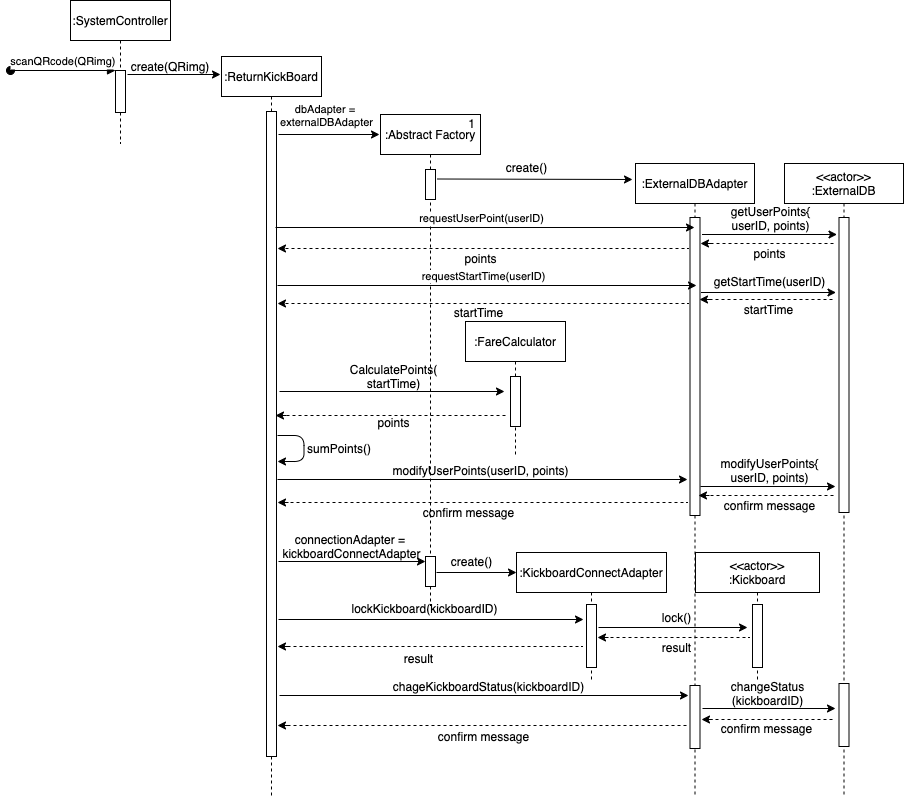
****

4.2. <Use Case 2> Return Kickboard Realization

4.2.1. Design Sequence Diagrams

4.2.1.1. SystemOperation 1 : scanQRcode()

* scanQRCode(userID) : Use Case 2에서 가장 중요한 System Operation이다. 사용자가 QR code를 스캔하면 사용자의 포인트를 차감하고 해당 Kickboard를 반납시킨다.

****

**<scanQRcode Interaction Diagram>**

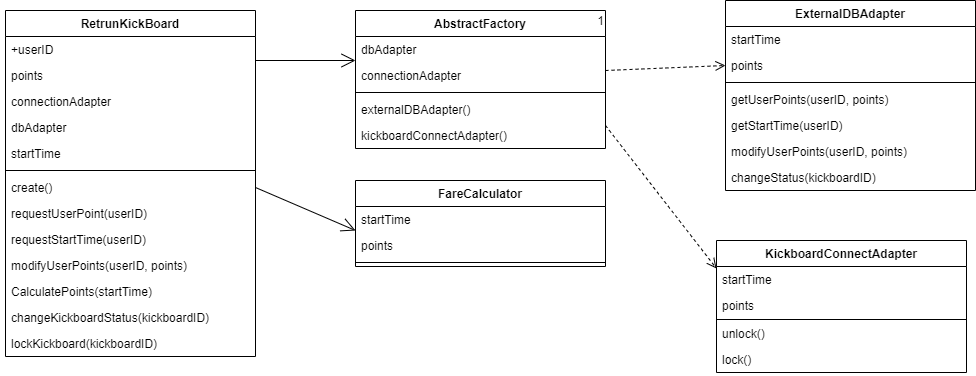
|  |  |
| --- | --- |
| **Pattern** | **Description** |
| Creator | - QRcode가 ReturnKickBoard객체를 초기화 하기 위한 데이터를 가지고 있으므로 QRcode가 Rent instance를 생성한다.  - ReturnKickBoard FareCalculator객체를 초기화 하기 위한 데이터를 가지고 있으므로 ReturnKickBoard FareCalculator객체를 instance를 생성한다. |
| Information Expert | - KickBoard의 정보를 알고 있는 ReturnKickBoard 객체가 ExternalDBAdapter에게 Message를 보낸다.  - KickBoard의 정보를 알고 있는 ReturnKickBoard 객체가 ExternalKBAdapter에게 Message를 보낸다. |
| Low Coupling | SystemController가 직접 ReturnKickBoard를 생성하지 않고 ReturnKickBoard에 대해 가장 잘 알고 있는 QRcode가 ReturnKickBoard를 생성한다. |
| High Cohesion | SystemController는 system input만을 받는 responsibility만을 갖는다. |
| Controller | SystemController는 system input을 받는다. |
| Polymorphism | None |
| Pure Fabrication | ExternalDBAdapter를 사용해 User 정보나 KickBoard 정보를 삽입, 변경한다. ExternalDBAdapter에 external datebase에 접근하는 responsibility을 할당해 low coupling과 high cohesion 유지한다. |
| Indirection | RetrunKickboard 객체와 Extern database 사이의 직접적인 Coupling을 피하기 위해 ExternalDBAdapter를 사용한다. |
| Protected Variations | External database에 접근하는 공통적인 기능을 제공하는 ExternalDBAdapter를 만들어 Low coupling을 유지한다. |

**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 ExternalDBAdapter, KickboardConnectAdapter 가지고 있다. ExternalKBAdapter, KickboardConnectAdapter을 actor의 역할을 하는 external database, external kickboard의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |
| Factory | Abstract Factory를 통해 Adapter 객체를 생성한다. 직접 객체를 생성하지 않고 Abstract Factory를 통해 생성하기 때문에 low coupling과 reusability가 보장된다. |
| Singleton | AbstractFactory는 singleton으로 선언되어 어디에서 접근하던 간에 오직 하나의 Factory 객체를 통해 특정 Object를 생성받는다. 또한 Factory를 singleton으로 선언하면 외부에서 Factory 객체를 생성할 필요도 없다. |

**<GoF Pattern>**

4.2.1.2 Design Class Diagram

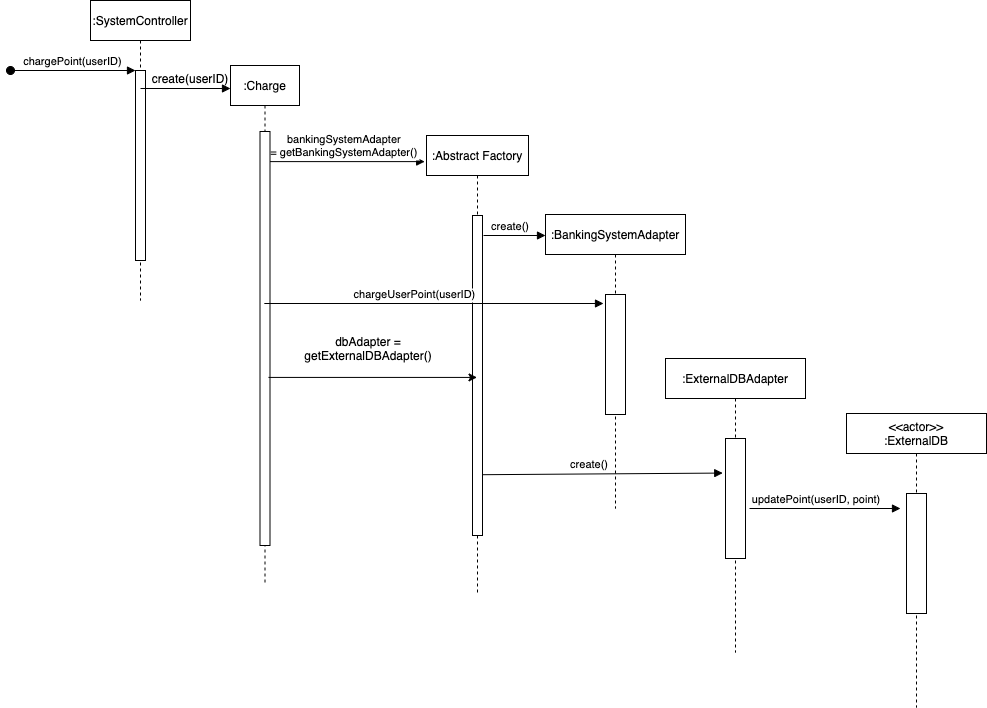


4.3. <Use Case 3> Charge Point Realization

4.3.1. Design Sequence Diagrams

4.3.1.1. System Operation 1 : chargePoint()

* chargePoint() : 사용자가 포인트 충전을 요청하면, 시스템에서 외부 결제 인터페이스 (Banking System)를 이용하여 포인트를 충전한다. 그리고 User의 충전된 포인트를 외부 DB에 업데이트시켜준다.



**<chargePoint Interaction Diagram>**

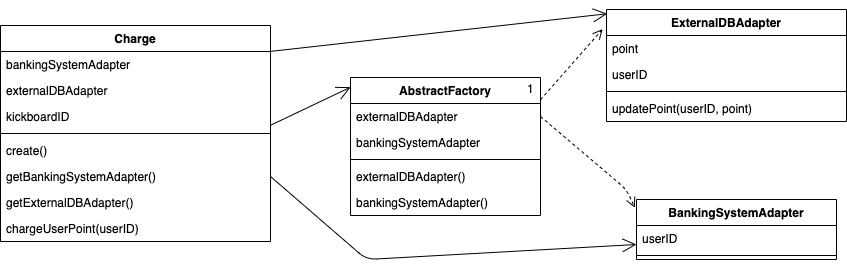
|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | - SystemController가 Charge 객체를 생성한다.  - Abstract Factory가 Adapter를 생성한다. |
| Information Expert | - User의 정보를 알고 있는 Charge 객체가 BankingSystemAdapter에게 Message를 보낸다.  - User의 정보를 알고 있는 Charge 객체가 dbAdapter에게 Message를 보낸다. |
| Low Coupling | - ExternalDBAdapter를 이용하여 ExternalDB에 접근한다. |
| High Cohesion | - SystemController는 system input만을 받는 responsibility를 갖는다. |
| Controller | - SystemController는 System input을 받는다. |
| Polymorphism | None |
| Pure Fabrication | - ExternalDBAdapter라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility를 갖는다. |
| Indirection | - Charge가 직접 Adapter를 생성하지 않고 Factory가 대신 Adapter를 생성한다. |
| Protected Variations | - ExternalDBAdapter가 ExternalDB에 접근하여 데이터를 업데이트해주기 때문에, Charge 객체는 External DB에 직접적인 concern을 하지 않아도 된다. |

**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 ExternalDBAdapter라는 Adapter을 가지고 있다. ExternalDBAdapter는 actor의 역할을 하는 external database의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |
| Factory | ExternalDBAdapter는 시스템 어디에서든지 접근을 필요로 한다. 따라서 복잡한 생성로직을 갖는 객체를 Factory가 대신 생성하는 역할을 함으로써 responsibility를 분리할 수 있으므로 high cohesion, reusability가 증가된다. |
| Singleton | AbstractFactory는 singleton으로 선언되어 어디에서 접근하던 간에 오직 하나의 Factory 객체를 통해 특정 Object를 생성받는다. 또한 Factory를 singleton으로 선언하면 외부에서 Factory 객체를 생성할 필요도 없다. |

**<GoF Pattern>**

4.3.1.2. Design Class Diagram

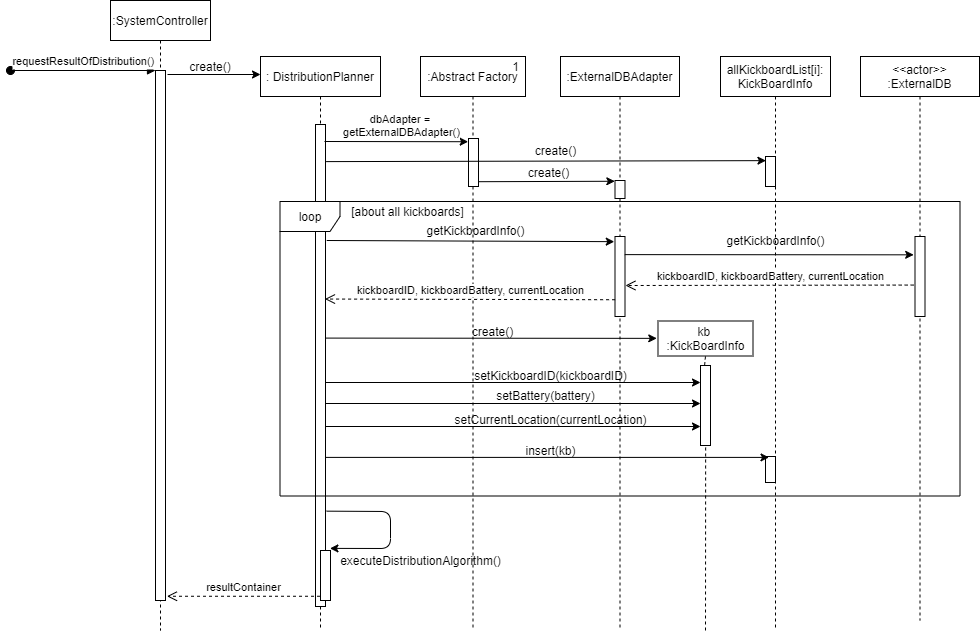
****

4.4. <Use Case 4> Manage Kickboard Realization

4.4.1. Design Sequence Diagrams

4.4.1.1. System Operation 1 : requestResultOfDistribution()

* requestResultOfDistribution() : KickBoard Manager가 System에게 킥보드 들을 배치할 새로운 정보를 요청한다. System에서는 분배 알고리즘을 적용할 데이터(전체 킥보드 정보)를 받아오기 위해 외부 DB로 부터 요청을 보낸다. 이후에 시스템은, 받아온 킥보드 데이터들로 리스트를 만들고 분배 알고리즘을 적용시킨다. KickBoard Manager에게 알고리즘 계산 결과를 반환하여 KickBoard Manager가 KickBoard를 새롭게 분배할 수 있도록 한다.



**<requestResultOfDistribution() Interaction Diagram>**

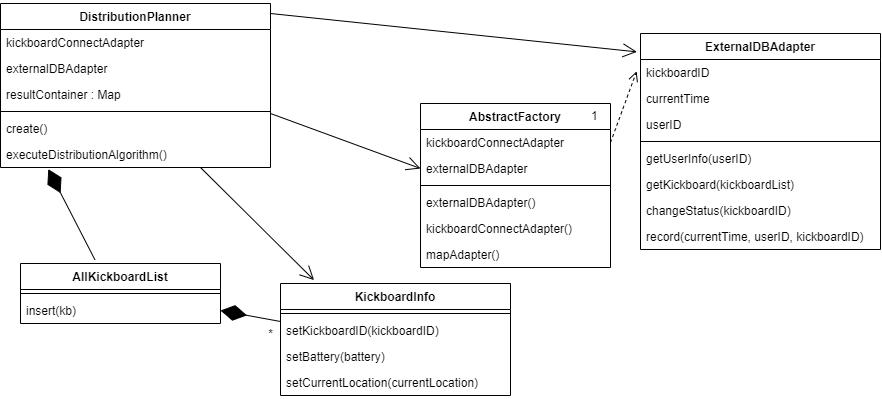
|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | - Distribution을 시작하기 위한 객체를 생성할 책임을 갖는 SystemController가 DistributionPlanner object를 생성한다.  - Adapter를 생성할 책임이 있는 AbstractFactory가 AbstractAdapter를 생성한다.  - 각 킥보드에 대한 정보를 가장 잘 알고있는 DistributionPlanner가 KickBoardInfo 객체를 만들 책임이 있다. |
| Information Expert | - 킥보드를 분배하기위해 필요한 킥보드 정보를 가장 잘 알고 있는 DistributionPlanner가 DistributionAlgorithm을 실행한다.  - 각 킥보드에 대한 정보를 가장 잘 알고있는 DistributionPlanner가 allKickboardList에 각 킥보드에 대한 정보를 삽입한다. |
| Low Coupling | - ExternalDBAdapter가 DistributionPlanner대신에 External DB에 접근함으로써 coupling이 줄어든다. |
| High Cohesion | - SystemController는 system operation을 받는 responsibility를 갖는다.  - KickBoardList는 KickBoardInfo만을 갖는다.  - AbstractFactory는 Adapter를 생성하는 cohesion을 갖는다.  - AbstractAdapter는 External DB에 접근하는 cohesion을 갖는다.  - DistributionPlanner는 분배 알고리즘을 실행시키기 위한 데이터를 얻고 분배 알고리즘을 실행시키는 cohesion을 갖는다. |
| Controller | SystemController는 모든 System input를 받는다. |
| Polymorphism | None |
| Pure Fabrication | ExternalDBAdapter라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility만을 갖는다. |
| Indirection | DistributionPlanner가 직접 Adapter를 생성하지 않고 Factory가 대신 Adapter를 생성한다. |
| Protected Variations | ExternalDBAdapter가 대신하여 External DB에 접근해서 데이터를 가져오기 때문에, DistributionPlanner object는 External DB에 대하여 직접적인 concern을 하지 않아도 된다. |

**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 ExternalDBAdapter라는 Adapter을 가지고 있다. ExternalDBAdapter는 actor의 역할을 하는 external database의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |
| Factory | 시스템 내에서 ExternalDBAdapter는 어디에서든지 접근을 필요로 한다. 따라서 복잡한 생성로직을 갖는 객체를 Factory가 대신 생성하는 역할을 함으로써 responsibility를 분리할 수 있으므로 high cohesion, reusability 증가된다. |
| Singleton | AbstractFactory는 singleton으로 선언되어 어디에서 접근하던지 간에 오직 하나의 Factory 객체를 통해 특정 Object를 생성받는다. 또한 Factory를 singleton으로 선언하면 외부에서 Factory 객체를 생성할 필요도 없다. |

**<Gof Pattern>**

4.4.1.2 Design Class Diagram

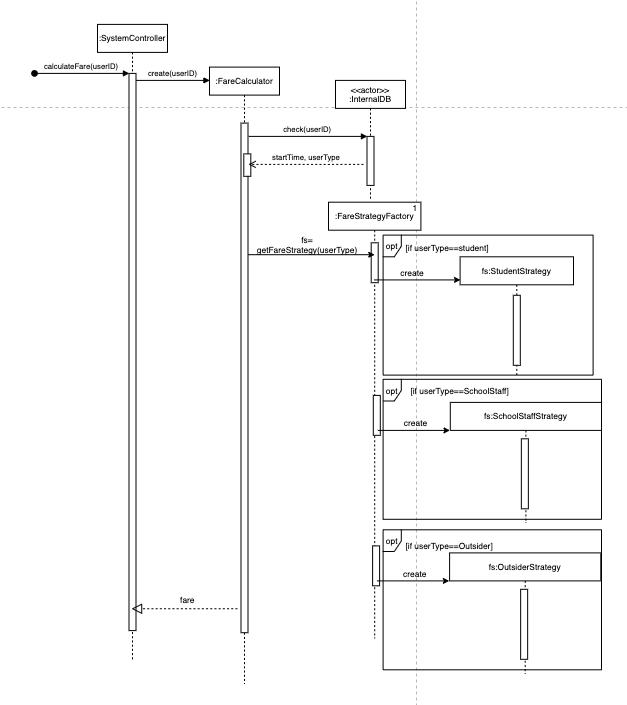
****

4.5 <Use Case 5> Calculate Fare Realization

4.5.1. Design Sequence Diagrams

4.5.1.1. System Operation 1 : calculateFare(userID)

* calculateFare(userID) : 시스템이 요금 계산을 요청한 사용자의 ID를 이용해 start\_time과 type 정보를 바탕으로 사용자의 요금을 계산한다.



**<calculateFare Interaction Diagram>**

|  |  |
| --- | --- |
| **GRASP** | **Description** |
| Creator | - SystemController가 Calculate 객체를 생성한다.  - Abstract Factory가 Adapter를 생성한다. |
| Information Expert | - User의 정보를 알고 있는 Calculate 객체가 InternalDB에 Message를 보낸다.  - User의 정보를 알고 있는 Calculate 객체가 dbAdapter에게 Message를 보낸다. |
| Low Coupling | - ExternalDBAdapter를 이용하여 ExternalDB에 접근한다.  - Calculate 객체를 통해 InternalDB와 ExternalDB에 접근함으로써 Coupling을 줄인다. |
| High Cohesion | None |
| Controller | - SystemController는 System input을 받고, Calculate 객체로부터 UserFare를 받는다. |
| Polymorphism | None |
| Pure Fabrication | - ExternalDBAdapter라는 Pure Fabrication object를 생성하여 DB를 접근하는 cohesive responsibility를 갖는다. |
| Indirection | - Calculate가 직접 Adapter를 생성하지 않고 Factory가 대신 Adapter를 생성한다. |
| Protected Variations | - ExternalDBAdapter가 ExternalDB에 접근하여 데이터를 업데이트해주기 때문에, Calculate 객체는 External DB에 직접적인 concern을 하지 않아도 된다. |

**<GRASP Pattern>**

|  |  |
| --- | --- |
| **GoF** | **Description** |
| Adapter | Pure Fabrication을 위해 ExternalDBAdapter라는 Adapter을 가지고 있다. ExternalDBAdapter는 actor의 역할을 하는 external database의 중개자 역할을 한다. 따라서 low coupling과 high cohesion이 지원되고 reusability가 증가한다. |
| Factory | ExternalDBAdapter는 시스템 어디에서든지 접근을 필요로 한다. 따라서 복잡한 생성로직을 갖는 객체를 Factory가 대신 생성하는 역할을 함으로써 responsibility를 분리할 수 있으므로 high cohesion, reusablity가 증가된다. |
| Singleton | AbstractFactory는 singleton으로 선언되어 어디에서 접근하던 간에 오직 하나의 Factory 객체를 통해 특정 Object를 생성받는다. 또한 Factory를 singleton으로 선언하면 외부에서 Factory 객체를 생성할 필요도 없다. |

**<GoF Pattern>**

4.5.1.2 Design Class Diagram

**텍스트이(가) 표시된 사진

자동 생성된 설명**

5. Architecture

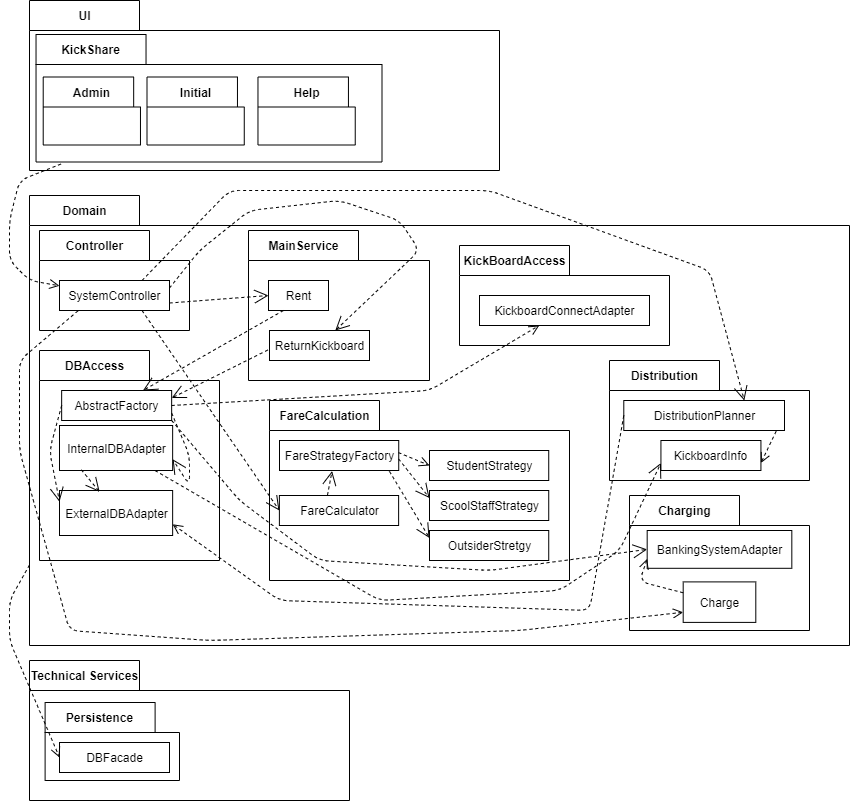
5.1. Introduction

시스템을 위한 구체적인 Architecture View들을 만들었다. 각각의 Architecture View에서는 이 시스템이 왜 이러한 Architecture를 Design 하였는지 설명한다. 여기에서는 Logical, Deployment, Use case view 등을 보이겠다. 또한 Architectural Factor에서는 Use Case의 Special Requirements, Technoloy Vatiations, Open Issues가 재검토 되었으며, Supplementary Specification에 있는 내용들을 정리하였다.

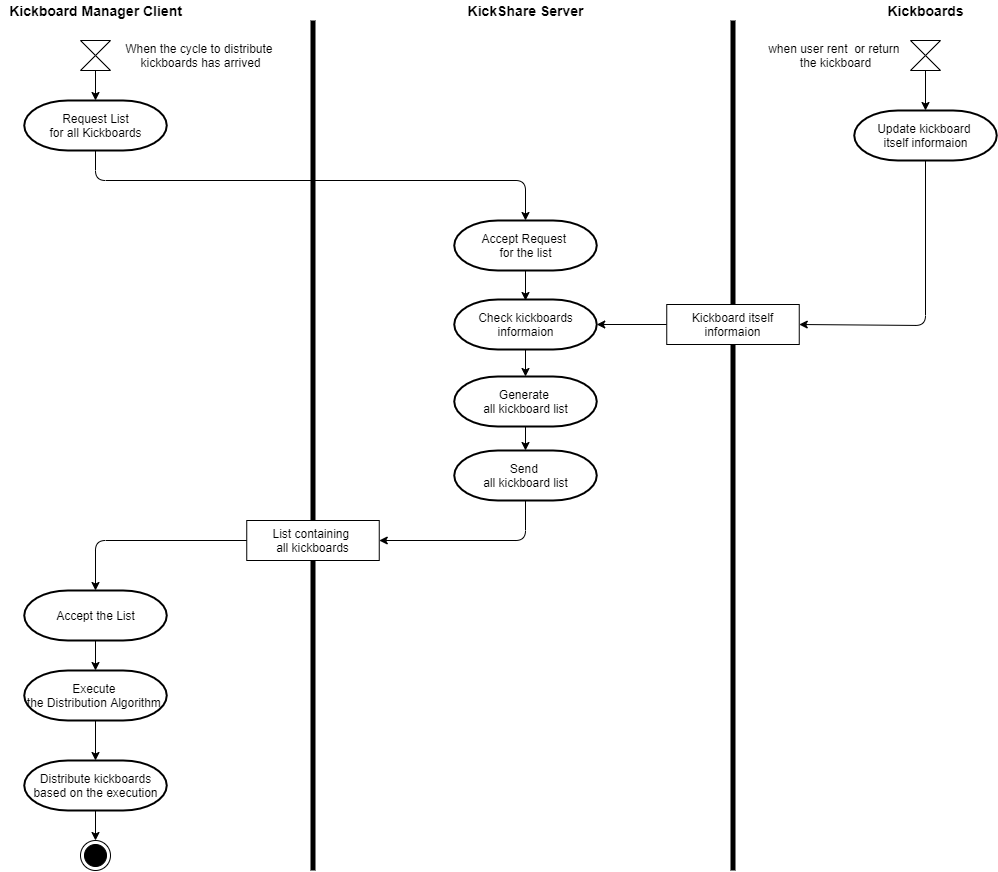
5.2. Architectural Factors

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Factor** | **Measures**  **and quality scenarios** | **Variability**  **(current flexibility and future**  **evolution)** | **Impact of factor**  **(and its variability)**  **on stakeholders,**  **architecture and**  **other factors** | | **Priority**  **for**  **Success** | **Difficulty**  **or Risk** |
| **Functionality – security** | | | | | | |
| **킥보드의 정보가 외부 데이터베이스로 네트워크를 통해 안전하게 전달된다.** | **킥보드가 외부 데이터베이스와 물리적인 거리가 멀지 않으며 안전한 프로토콜(TLS)을 사용하여 데이터를 전송한다.** | **Current flexibility -**  **외부 데이터베이스를 관리하는 서버에서 사용이 시작된 킥보드에 대해 일정 시간내로 프로토콜 메시지가 오지 않는다면 서버가 먼저 메시지를 보내서 확인한다.** | **-서버에서는 킥보드의 고유 정보를 알고 있으므로 누가(킥보드인지 아니면 stranger인지) 프로토콜 메시지를 보냈는지 보장할 수 있다.**  **- 킥보드의 파손으로 인한 위치정보 미확인인 경우에는 킥보드의 정보를 보장할 수 없다.** | **M** | | **M** |
| **Performance** | | | | | | |
| **사용자의 정보를 서버의 데이터베이스로부터 client로 가져오는데 적은 시간이 할애되어야 한다.** | **사용자 정보를 가져오는데 0.5초 이내로 완료된다.** | **-Current flexibility: client마다 local 데이터베이스를 두어 각 사용자에 대한 정보는 사용자 고유의 local 데이터베이스에 저장한다.** | **- Local 데이버에스의 사용자 정보를 서버가 필요로 한 상황에서는 서버가 갖고 있는 사용자 정보가 최신의 것인지 보장하지 못함**  **- 사용자가 고의로 자신의 local 데이터베이스를 manipulate할 경우에는 사용자의 정보를 신뢰** | | **H** | **M** |
| **Functionality** | | | | | | |
| **Clients의 local 데이터를 서버가 사용하기 위한 확장성** | **사용자의 정보가 local 데이터베이스에서 갱신될 때마다 서버로 알린다.** | **-Current flexibility: local데이터베이스의 갱신을 알린다면 서버가 추가적으로 사용자의 정보가 필요할 때(flexible) 사용할 수 있다.**  **-Evolution: 추후에 시스템의 새로운 기능추가 및 서비스 확장이 있을 경우 서버는 사용자의 정보를 믿고 사용할 수 있다.** | **-clent와 서버간의 네트워킹 overhead가 증가함.**  **-Management: local데이터베이스의 정보가 갱신될때보다 서버가 주기적으로, 또는 필요할 때에만 client와 상호작용함.** | | **M** | **L** |
| **Reliability – Recoverability** | | | | | | |
| **서로 다른 노드에 있는 서버와 client간의 네트워크 오류에 대한 회복** | **각 원격 노드간 네트워크 오류시 3초 내로 재연결을 시도한다.** | **-Current flexibility:**  **client쪽에서 서버로부터 응답이 3초내로 오지 않는다면 reconnection을 시도함.** | **서버로부터 응답이 오지 않는다면 매우 큰 결함이 발생할 것.**  **Kickboard Manager는 서버로부터 분배할 킥보드 정보를 받지 못한다면 킥보드의 재배치가 진행되기 어렵다.**  **사용자는 사용가능한 킥보드를 확인할 수 없어 rent가 원활하지 못하게 된다.** | | **H** | **H** |

5.3. Logical view

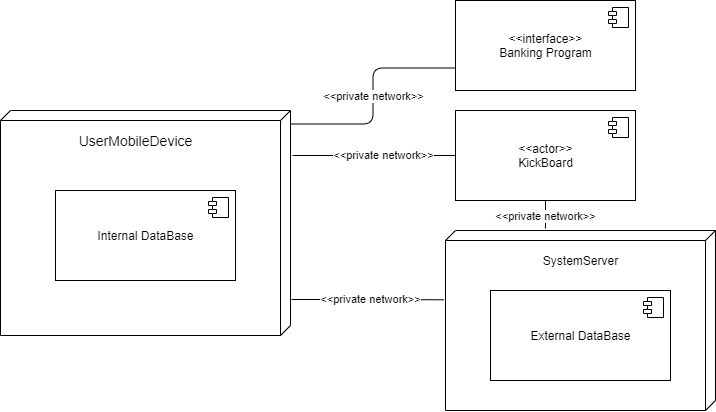
****

5.4. Process view



* Kickboard Manager와 Kickboard가 External DataBase에 동시에 접근하며 Synchronizaion을 맞춰주어야 한다.

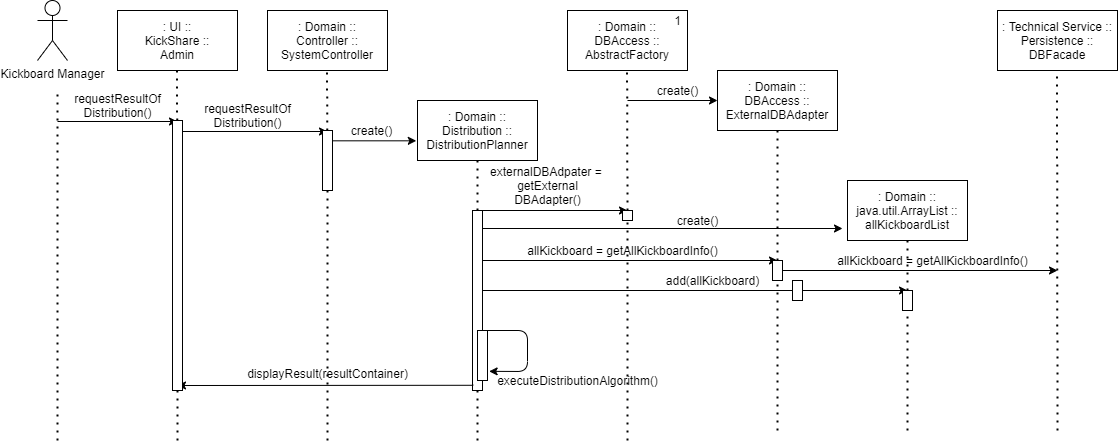
5.5. Deployment view

****

* **Discussion and motivation:**

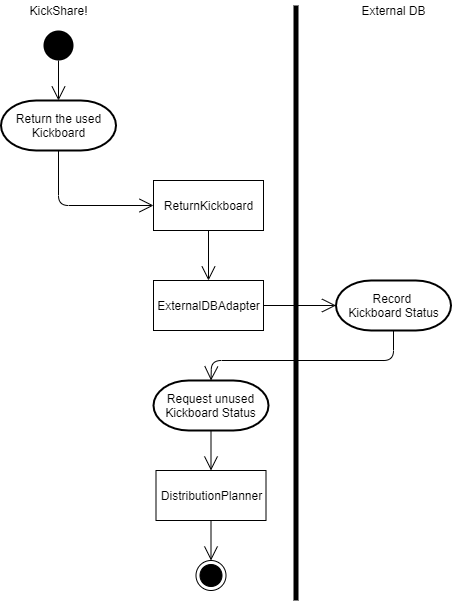
이 시스템의 서버는 사용자의 device와 다른 노드에 있으며 서버에서는 주로 External DataBase를 관리하는 역할을 하도록 구성하였다. 또한 이 시스템은 Thick client이기 때문에 UserMobileDevice에서 대부분의 일이 발생한다. 따라서 UserMobileDevice가 중심에 위치한다.

5.6. Use case view

****

**<Use Case4 : Manage Kickboard View>**

5.7. Data view

****

**<Data Flow Scenario for Kickboard’s Status>**

5.8. Technical Memos

|  |
| --- |
| **Issue :** Performance - 사용자의 정보를 서버의 데이터베이스부터 client로 가져오는데 적은 시간이 할애되어야 한다.    **Solution Summary :**  - client마다 local DataBase를 두고 사용자에 관련된 데이터를 저장한다.    **Factors :**  - 사용자의 정보를 서버에 있는 External DataBase에 접근하고 탐색하여 찾아오는데 overhead가 크다.    **Solution :**  - 사용자 정보 개개인에 대한 정보는 client마다 local DataBase(Internal DB)에서 저장하고 관리하도록 한다. 하지만 사용자의 정보가 갱신되는 경우에 대해서는 서버에 있는 External DataBase에서도 갱신을 해 주어야한다.    **Motivation :**  - 각 client마다 사용자의 정보를 필요로 할 때, 사용자정보를 저장하는 데이터베이스가 서버에만 존재한다면 서버로의 트래픽이 많이 모이게 될 것이고 성능저하 및 사용자에게 present되는 시간이 오래걸리게 될 것이다. 따라서 사용자마다 독립적으로 갖는 사용자 정보를 Local DataBase와 서버의 External DataBase에 모두 두고, client는 사용자 정보가 필요할 때 Local DataBase에 접근하여 사용하도록 한다.    **Unsolved Issues :**  - Unsolved Issue 1 : 서버에서 기능 추가 및 서비스 확장, 통계 등을 위해 사용자 정보가 필요할 수 있다. 이 경우 최신의 사용자 정보를 필요로 할 것인데, Local DataBase에서만 사용자 정보를 갱신해 준다면 문제가 될 수 있다.  - Brief Solution : Local DataBase에서 사용자 정보가 갱신될 때, 서버의 External DataBase에서도 갱신을 시켜줄 수 있다. 하지만 이러한 갱신 주기를 사용자 정보가 바뀔 때마다 한다면 역시 시스템 내부적인 Overhead가 클 것이다. 따라서 서버가 사용자 정보를 필요로 할 때에만 client들에게 요청하여 최신의 사용자 정보들을 받아올 수 있다.    **Alternatives Consideration :**  - None |

6. Conclusion

현재 이 시스템은 주요 requirement를 충족하는 use-case를 가지고 있으며, 각 use-case에 대한 도메인 모델링은 모두 완료되었다. 또한, 도메인 모델을 토대로 한 디자인 모델링도 완료된 상태이다. 다만, 유저의 실제 피드백을 받아 수정하는 부분은 아직 반영되지 않았다. 구현은 현재 약 40% 이상 진행된 상태이며, UI는 사용자가 시스템을 편리하게 사용할 수 있도록 구현되었다. 다만 결제와 포인트의 충전 등 외부 시스템을 사용하는 부분은 아직 UI 상에서 구현이 되지 않은 상태이다. 도메인 부분은 킥보드를 rent 하는 use case와 요금을 calculate하는 use case가 구현되어 있으며, 다른 use case들은 Iteration을 반복하면서 추가적으로 구현이 될 예정이다. 앞으로 더 진행해야 하는 부분으로는, 사용자의 피드백을 받아 수정 혹은 개선해야 하는 부분들이 도메인 모델에 빠르게 반영되어야 하고, 이를 기반으로 디자인과 실제 구현까지 빠르게 수정되어야 한다. 디자인 모델링에서도 계속해서 다른 디자인 패턴을 적용하는 등 더 나은 방향으로 가기 위해 개선되어야 한다. 또한, 킥보드와 통신하는 모듈이나 외부 결제 시스템 등을 사용하기 때문에, 기존의 시스템과, 혹은 다른 외부 시스템과 호환성이 좋은 모델을 찾아 적용해야 한다. 현재 단계에서는 어떤 시스템을 사용할 지는 결정되지 않은 상태이다. UI 부분 또한 사용자의 의견을 반영하여 사용하기 편리하게 개선되어야 한다. 예를 들어, 대여하기 메뉴에 들어간 뒤에 맵을 보여주는 방식과, 사용자는 어차피 대여를 위해 들어오는 것이므로 앱을 실행하면 바로 대여하기 메뉴를 보여주는 방식 중 어떤 방식을 채택할 지 등은 선 구현 후 사용자의 피드백을 받아 변경하여야 하는 문제이다.

이번 과제를 통해, 우리가 평소에 사용하는 시스템들이 기능의 단순한 구현으로 이루어진 것이 아니라, 사용자 혹은 stakeholder들의 요구사항 분석부터 도메인 모델, 디자인 모델 그리고 다양한 디자인 패턴들의 적용으로 체계적으로 만들어졌다는 사실을 알게 되었다. 또한, 어떤 시스템이던 간에 체계적인 설계와 구현이 얼마나 중요한지도 알게 되었다. 시스템을 여러 부분으로 나누고, 각 부분들이 어떤 방식으로 소통하고 상호작용 하는지를 계획하고 설계하는 것은 매우 복잡하고 어려운 일이지만, 이런 방식으로 설계된 시스템이야 말로 안정적이고 효율적으로 작동할 것임을 느낄 수 있었다. 또한 실제 기업에서는 여러 사업부가 파트를 나눠 설계를 진행하기에, UML을 이용한 도메인 모델링의 효율성이 매우 높을 것임을 예상할 수 있었다. 무엇보다, 이번 팀 프로젝트에서 팀원들 간에 시스템에 대해 말로는 똑같이 표현하고 있었지만 실제로는 다른 것을 생각하고 있는 경우가 매우 많았다. 이런 경우에 다이어그램을 통해 소통하면 훨씬 더 명확하고 빠르게 서로의 의견을 맞춰갈 수 있다는 사실을 알게 되었다. 또, 의욕이 넘쳐 미리미리 해놔도 앞의 부분에서 수정할 부분이 생기면 뒤의 부분을 전부 수정해야 하기 때문에, 한 번에 많이 진행하는 것이 아니라, iteration을 여러 번 반복하면 진행하는 방식이 더 효율적이라는 것을 느끼게 되었다.

7. References

- 플라워로드 사이트, <http://www.flowerroad.ai/>

- 운전면허정보 검증 사이트, <https://dlv.koroad.or.kr/>

- 4+1 view 위키피디아, <https://en.wikipedia.org/wiki/4%2B1_architectural_view_model>

- thick client vs thin client, <https://simplicable.com/new/thin-client-vs-thick-client>

- 전송 계층 보안, 위키피디아, [https://en.wikipedia.org/wiki](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model)/전송\_계층\_보안

- sequence diagram 구성 및 작성법, <https://sswpgm.tistory.com/29>

8. Appendix

**1) Glossary**

|  |  |  |  |
| --- | --- | --- | --- |
| **용어** | **정의 및 정보** | **형식** | **검증 규칙** |
| User | 전동 킥보드를 사용하는 고객으로, Kick Share! 시스템에 가입이 완료된 사람 |  | 운전 면허증의 유효성 체크 |
| KickBoard Manager | 킥보드와 관련된 모든 것을 처리하는 사람 (킥보드 배치, 킥보드 상태 체크) |  |  |
| QR Code | 킥보드 보관소마다 킥보드 보관소에 관한 정보를 갖고 있는 QR Code가 존재하고, 모든 전동 킥보드는 각자 고유한 QR Code를 갖고 있다. | 흑백 격자 무늬 패턴 |  |
| Banking Program | 외부 결제수단을 이용하여 포인트를 충전할 수 있도록 하는 프로그램 |  |  |
| Checking License | 외부 시스템을 이용하여 운전 면허의 유효성을 체크한다. |  |  |
| Identity Verification | 회원 가입 시, 본인 인증 확인을 위해 필요하다. |  |  |
| 포인트 | Kick Share! 시스템 이용 시, 포인트가 없다면 전동 킥보드를 사용할 수 없다. 포인트는 Banking Program을 통해 충전 가능하다. | 숫자로 구성되어있으며, 한국 돈으로 100원은 100포인트와 같다. |  |
| 분배 알고리즘 | 전동 킥보드의 사용량을 시간, 위치별로 계산하여 전동 킥보드를 적절히 분배할 수 있도록 만드는 알고리즘 |  |  |
| 전동 킥보드 보관소 | 전동 킥보드가 보관되는 장소로, 고유한 QR Code를 가지며 캠퍼스 출입구와 건물 출입구에 설치되어있다. |  |  |
| GPS | 전동 킥보드의 위치를 실시간으로 업데이트 |  |  |
| 대여 가능 | 킥보드의 상태 정보로써 킥보드의 대여가 가능함을 의미한다. |  | QR Code로 대여 |
| 사용 중 | 킥보드의 상태 정보로써 킥보드의 대여가 불가능하며, 사용자가 해당 킥보드를 사용 중임을 나타낸다. |  |  |

9. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description** | **Author** |
| Inception Draft | 2020.09.19 | First Meeting  Brain Storming | Team 김말국 |
| Inception Draft | 2020.09.20 | Second Draft | Team 김말국 |
| Inception Draft | 2020.09.22 | 주제 변경 | Team 김말국 |
| Inception Draft | 2020.09.24 | Inception Report 완성 | Team 김말국 |
| Elaboration 1 Draft | 2020.10.19 | Domain Modeling  First Draft | Team 김말국 |
| Elaboration 1 ver1.0 | 2020.10.25 | Add Mini SSD, OC | Team 김말국 |
| Elaboration 1 ver1.1 | 2020.10.26 | Update SSD, OC | Team 김말국 |
| Elaboration 1 ver1.2 | 2020.10.29 | Modify Domain Model | Team 김말국 |
| Elaboration 1 ver1.3 | 2020.10.31 | Modify SSD, OC | Team 김말국 |
| Elaboration 1 ver1.4 | 2020.11.01 | Update use cases  Update SSD, OC  Add DCD | Team 김말국 |
| Elaboration 1 ver1.6 | 2020.11.05 | Modify DCD  Add interaction diagram | Team 김말국 |
| Elaboration 1 ver1.7 | 2020.11.10 | Update interaction diagram  Add pattern | Team 김말국 |
| Elaboration 2  version 1 | 2020.11.29 | Modify DSD  Add GRASP Pattern  Add GoF Pattern | Team 김말국 |
| Elaboration 2  ver 1.1 | 2020.12.02 | Update GRASP Pattern  Update GoF Pattern  Update DSD  Modify DCD | Team 김말국 |
| Elaboration 2  ver 1.2 | 2020.12.04 | Add architecture  Add conclusion  Modify Domain model | Team 김말국 |
| Elaboration 2  ver 1.3 | 2020.12.05 | Update architecture  Modify conclusion | Team 김말국 |
| Elaboration 2  ver 1.4 | 2020.12.06 | Complete writin github  url : https://github.com/Domain-Analysis-And-SW-Design-kimmalguk/Implementation | Team 김말국 |