# 더치빗자루

Requirements and Specification

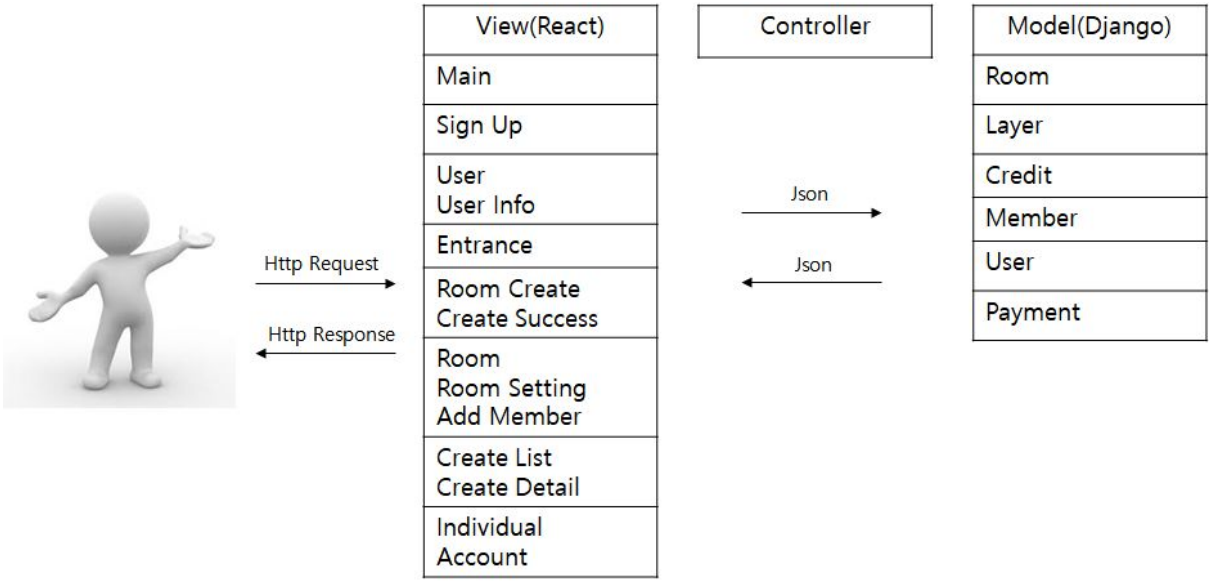
2019-04-16 - first version

## Members

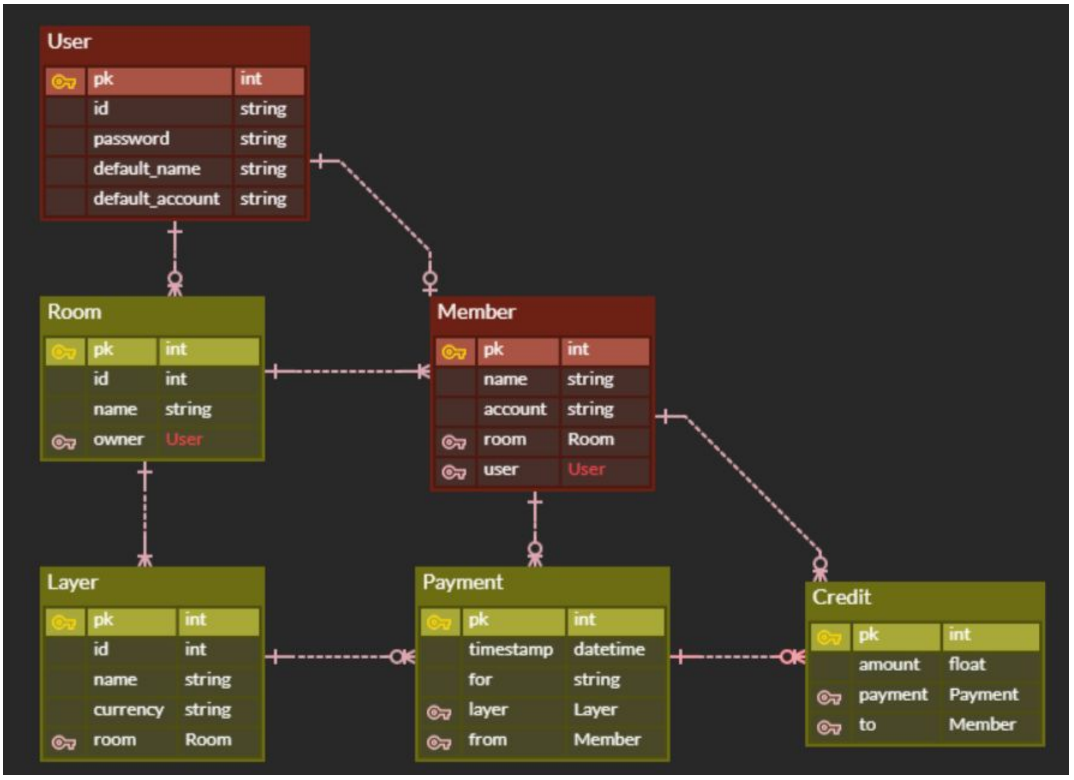
- 2017 - 10433 정유석
- 2017 - 10483 이정민
- 2017 - 17018 장태준

## System Architecture

### MVC



## Model




https://www.erdcloud.com/d/Gj4s8JDgumLabZQZ5

위 그림은 모델의 구조를 E-R diagram으로 표현한 것이다. Entity 사이의 관계를 표현하는 데 까마귀발(Crow-feet) 표기법을 사용하였다.

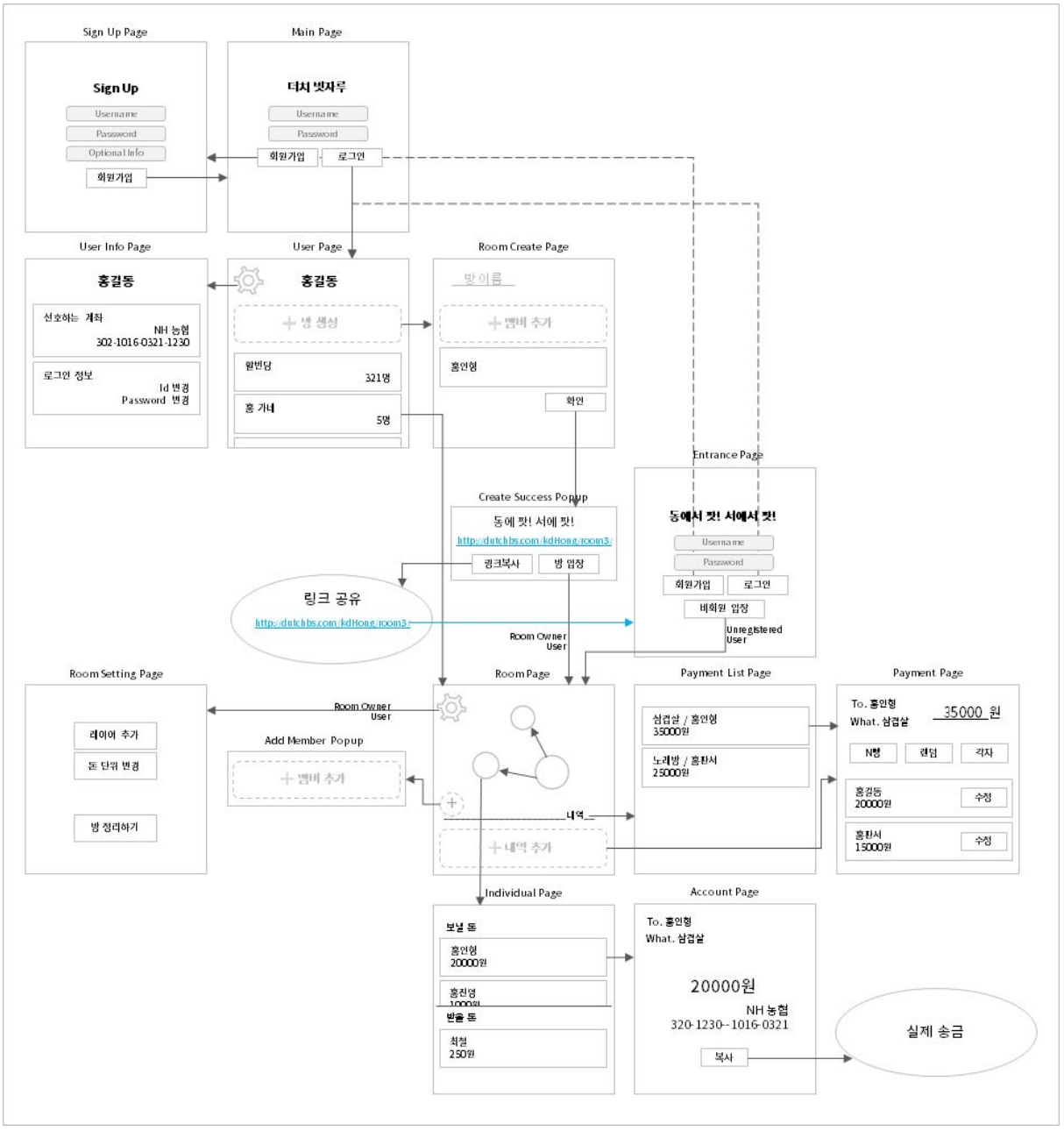
각각의 Entity는 본 서비스에서 다음 개념을 상징한다.

- User : 서비스에 회원 가입한 사용자
- Room : 결제 내역을 공유하는 “방”
- Layer : 결제 내역을 분리해서 관리하는 한 단위
    - 해외여행 등에서 다른 통화 단위를 사용할 때 등에 유효한 개념이다.
- Payment : 한 번의 결제 행위
    - for 필드는 결제 목적을 상징한다.
    - 모임이 끝나고 돈을 정산(송금)하는 행위 또한 Payment로 취급한다.
- Credit : 한 사람에게서 한 사람으로 보내는 결제 내역
    - 하나의 Payment에 여러 개의 Credit이 발생할 수 있다.
- Member : “방”에 소속된 구성원
    - 회원가입을 하지 않은 Member가 존재할 수 있다. 따라서 user 필드는 nullable하다.

모든 Entity에는 pk 필드가 있는데 이는 Django의 Model에는 pk 필드(autoincrement int)가 자동으로 생성됨을 반영한 것이다. 일부 Entity에는 id 필드가 있는데 이는 Django Model의 pk와는 별도로 필요하여 정의하였다.

- User.id는 로그인할 때 입력하는 ID 문자열을 상징한다.
- Room.id는 외부에서 접속 가능한 공유 링크를 생성할 때 사용한다.
- Layer.id는 레이어 번호를 상징한다. (Layer.room이 다를 경우 Layer.id는 중복될 수 있음)

## View




User Interface for View Design

1. Sign Up page (`/sign_up/`)
   -새로운 user의 회원가입
   -username, password 를 input으로 받아 계정생성
   -동일한 유저가 있는지 check

2. Main page(`/`)
   -Sign_Up
   -Sign_In

3. User page(`/user/:id/`)
   -Setting : user info page로 이동
   -+방생성 버튼 으로 Create_Room page로 이동
   -user가 속해는 room의 list를 제공 (owner 인 것 표시)
    : get_room_list

4. User Info page (`/user/:id/setting/`)
   -user의 정보를 나타냄
   -선호하는 계좌 정보를 수정할 수 있음
   -로그인 정보 (username, password) 수정
    : get_user_by_id

5. Room Create page (`/user/create_room/`)
   -방이름을 input으로 받음
   -멤버 추가를 통해 graph의 node 를 설정
    : graph_add_member
   -확인을 통해 room을 생성하고 그에 따라 link 생성됨

6. create success popup page (`/user/create_room/`)

7. Entrance page (`/owner/:room/entrance/`)

8. Room page (`/user/:room/`)

9. Room Setting page (`/user/:room/setting/`)

10. add member popup page (`/user/:room/add_member/`)

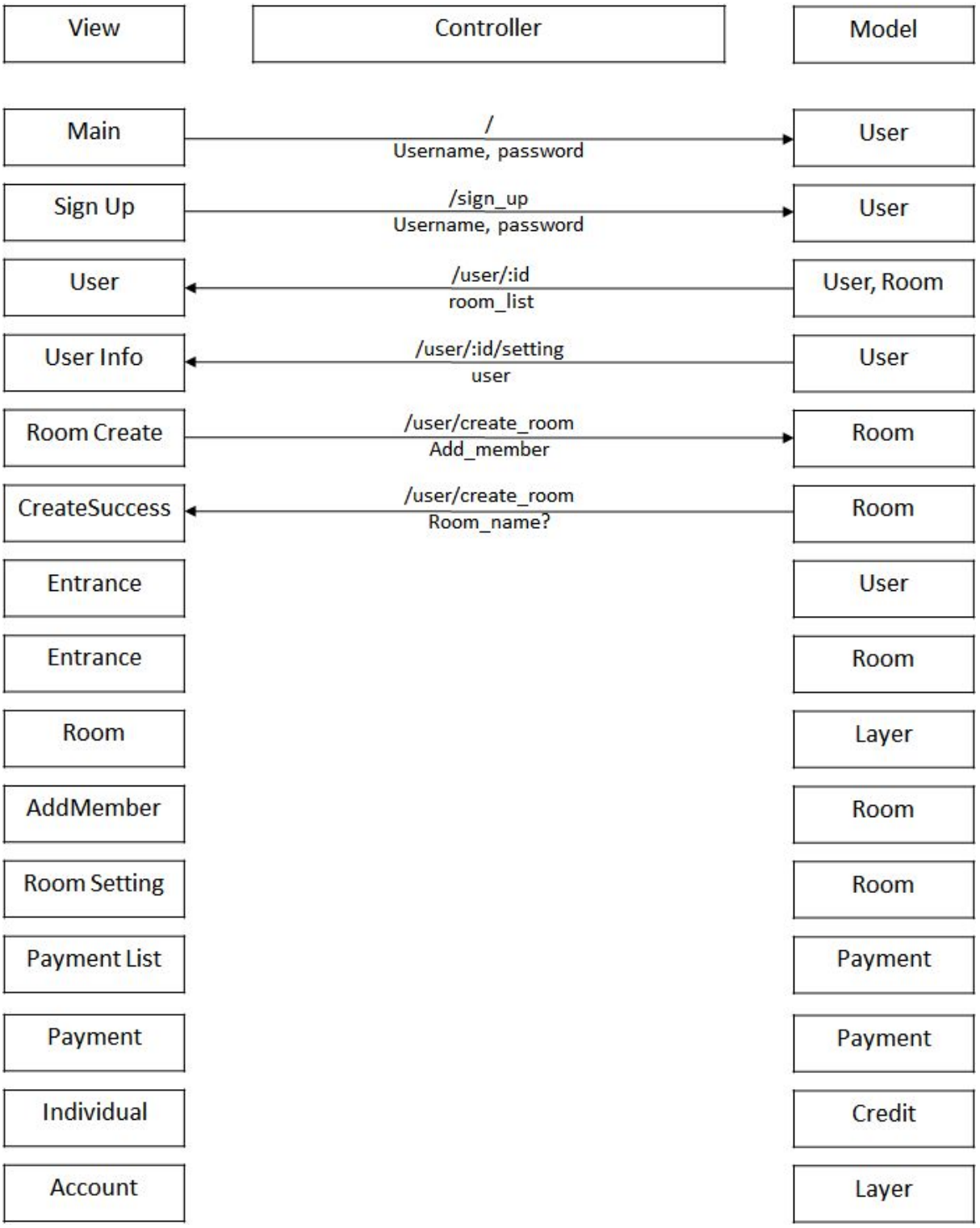
11. Payment List page (`/user/:room/payment_list/`)

12. Payment page (`/user/:room/payment_list/:payment/`)

13. Individual page (`/user/:room/member/`)

14. Account page (`/user/:room/member/account/`)

Controller

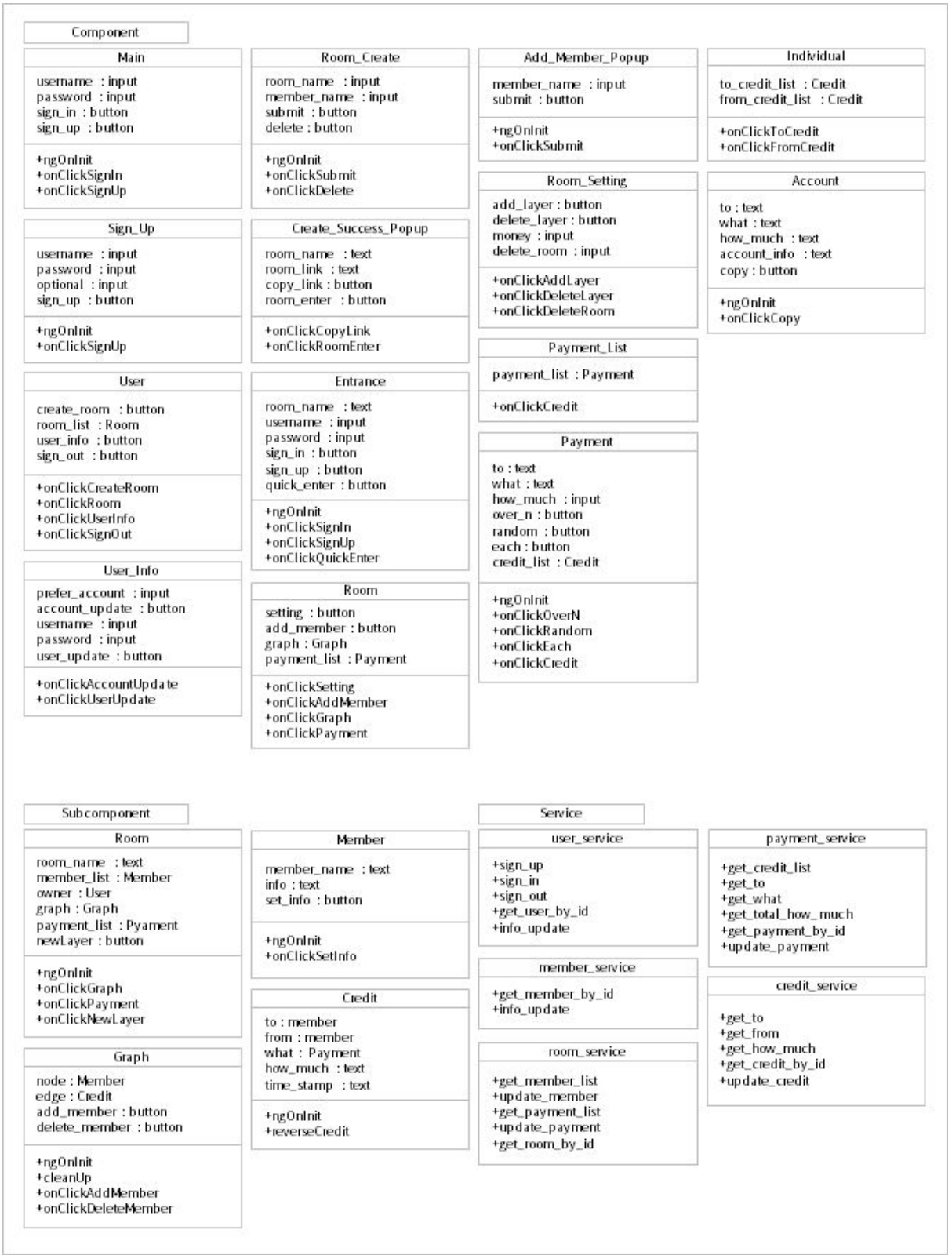

View
Controller
Model
Main
/
Username, password
User
Sign Up
/sign_up
Username, password
User
User
/user/:id
room_list
User, Room
User Info
/user/:id/setting
user
User
Room Create
/user/create_room
Add_member
Room
CreateSuccess
/user/create_room
Room_name?
Room
Entrance
User
Entrance
Room
Room
Layer
AddMember
Room
Room Setting
Room
Payment List
Payment
Payment
Payment
Individual
Credit
Account
Layer

# Design Details

## Frontend Design

### Frontend Components

frontend 의 구성은 다음과 같다. 각 component에서 수행가능한 method 들은 box 안에 있다.



## Frontend Algorithms

### Component

1) Main
2) Sign_Up
3) User
4) User_info
5) Room_Create
6) Create_Success_Popup
7) Entrance
8) Room
9) Add_Member_Popup
10) Room_Setting
11) individual
12) Account

### Subcomponent

1) Room
2) Graph
3) Member
4) Credit

### Service

1) user_service
2) member_service
3) payment_service
4) credit_Service

## Frontend Relations

frontend의 component 간 관계는 다음 그림과 같다.

## Algorithm for Simplifying Credits

송금 횟수를 최소화시키기 위해 결제 내역을 변형하는 알고리즘은 Client-side에서 JavaScript로 구현한다. 이를 위해서는 Layer 내에 존재하는 모든 Credit 정보를 백엔드로부터 읽어와야 한다. (`GET /rooms/:room_pk/layers/:layer_pk/credits`)

백엔드 데이터를 다음과 같이 가공하면, 후술할 알고리즘을 통해 송금 횟수를 최적화 시킬 수 있다.

```
[ { from: 1, to: 2, amount: 1000.0 },
  { from: 2, to: 3, amount: 2000.0 }, ...]
```

다음 알고리즘은 n명의 사람이 있을 때 최대 n-1번의 결제로 정산을 끝낼 수 있다. 시간복잡도는 M이 총 결제 횟수, N이 사람일 때 O(M+N)이다. '최소'는 보장할 수 없지만 충분히 효율적인 휴리스틱이다.

1. define total[M]  // total[i] : i가 받아야할 돈
2. let total[i] = sum(c.**amount** for c in credits if c.**from** **===** **i**)
   + **(-1)** ***** sum(c.**amount** for c in credits if c.**to** **===** **i**)
3. j가 j+1에게 total[j+1]-total[j]를 보내주면 된다.

## Backend Design

### Restful API

각 URL 밑에 필요한 parameters를 명시해두었다. (Params)
Optional한 필드의 경우 대괄호로 감싸 표시하였고, URL에 명시된 params는 제외하였다.

**Users**

- POST /users
  - Params: id, password, default_name, [default_account]
  - 회원가입 수행
- POST /users/:user_pk/login
  - Params: password
  - Returns: JWT
- PUT /users/:user_pk
  - Params: [password], [default_name], [default_account]
  - 회원정보 수정
- DELETE /users/:user_pk

**Room**

- POST /users/:user_pk/rooms
  - Params: id, name
  - 새로운 방 생성 (기본 Layer와 Member(owner)도 생성)
- GET /users/:user_pk/rooms
- PUT /rooms/:room_pk
  - Parmas: [id]
- DELETE /rooms/:room_pk

**Member**

- POST /rooms/:room_pk/members
  - Params: name, account
- GET /rooms/:room_pk/members
- GET /rooms/:room_pk/members/:member_pk
- PUT /rooms/:room_pk/members/:member_pk
  - Params: [name], [account], [user]
- DELETE /rooms/:room_pk/members/:member_pk

**Layer**

- POST /rooms/:room_pk/layers
  - Params: name, currency
- GET /rooms/:room_pk/layers
- GET /rooms/:room_pk/layers/:layer_pk
- PUT /rooms/:room_pk/layers/:layer_pk
  - Params: [name], [currency]
- DELETE /rooms/:room_pk/layers/:layer_pk

**Payment**

- POST /rooms/:room_pk/layers/:layer_pk/payments
    - Params: from, for, to_array, amount_array
    - 연관된 Credit까지 자동으로 생성됨
- GET /rooms/:room_pk/layers/:layer_pk/payments
- GET /rooms/:room_pk/layers/:layer_pk/payments/:payment_pk

**Credit**

- GET /rooms/:room_pk/layers/:layer_pk/credits
    - Parmams: member
    - member 필드를 지정하면 특정 멤버와 관련된 Credit만 필터링 됨
- GET /rooms/:room_pk/layers/:layer_pk/credits/:credit_pk

## Authentication

User가 관여되는 행동(방 생성 등)에 대해서는 Authentication이 필요하다.
Django REST Framework에서 제공하는 Token-based Authentication를 사용할 계획이다.

https://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication

# Implementation Plan

## Plan for Sprint 2 (23 Apr ~ 6 May)

회원가입 등 User 모델과 관련된 기능을 개발하는 것을 Sprint 2의 주요 목표로 둔다.
Sprint 2에서는 아래 다섯개의 User story를 구현할 예정이다.

- #1 Sign Up Page [장태준]
- #2 Main Page (#1 required) [장태준]
- #3 Login / User Page (#2 required) [이정민]
    - 회원가입 기능 구현 (POST user)
- #4 User Page (#3 required) [이정민]
    - 로그인 기능 구현 (POST user login)
- #7 User Info Page (#4 required) [정유석]
    - 유저 정보 수정 (PUT user login)

위 계획의 주요 리스크로는 사용자 인증(Authentication) 방식을 어떻게 할 것인지 결정하는 것에 있다. HW 2에서는 Basic Header을 이용한 Authentication을 이용하였으나 password를 Redux state로써 저장하는 것이 보안 상 우려되며 구조도 비효율적이라는 생각이 들어 앞서 Backend Design에서 명시한 대로 DRF 프레임워크에서 제공하는 Authentication 기능을 사용할 계획이다.

## Plan for Sprint 3

Room, Layer 모델과 관련한 기능을 구현하는 것을 주요 목표로 두되, Sprint 2의 계획을 얼마나 이행했는지에 따라 추후 구체화한다.

## Plan for Sprint 4

Member, Payment, Credit 모델과 관련한 기능을 구현하는 것을 주요 목표로 두되, Sprint 2의 계획을 얼마나 이행했는지에 따라 추후 구체화한다.

## Plan for Sprint 5

Frontend에 기술한 결제 상황 간소화 알고리즘 구현을 완료하고 정산 기능을 추가하는 것을 주요 목표로 두되, Sprint 2의 계획을 얼마나 이행했는지에 따라 추후 구체화한다.

# Testing Plan

Sprint 1 기준 현재 프로젝트에서 Boilerplating이 갓 끝난 때문에 구체적인 Testing Plan을 기술하기에는 어려움이 있어, 어떤 프레임워크를 사용할 것인지만 간단히 명시하였다.

## Backend

Django에서는 Python unittest 기반의 테스트 프레임워크를 제공하므로 이를 활용하고자 한다.

## Frontend

React/Redux 기반의 프론트엔드 소스 코드는 Jest라는 테스트 프레임워크를 활용하여 Unit Test를 수행한다. 현재 프로젝트에서 사용하고 있는 Atomic React boilerplate는 Jest를 이용한 unit test 예시가 준비되어 있어 이를 참고하여 테스트를 설계하고자 한다.

## Continuous Integration

지속적인 통합(Continuous Integration)을 실현하기 위해 Travis CI라는 도구를 사용한다.