2017

Conference Management System 18655

TEAM 6 - LIANGHAO GAO, MINGHAO LI, TIANYANG HU, YIMING LIU

1

Introduction&Motivation

Introduce the background and goals of this system

2

Related Work

Compare this system with existing product

3

System Design Implementation

Demonstrate system design and technologies which are used to implement this system

4

Demo

Play the demo video and show functionalities

5

Technical Experiences

Share technical experiences we have learned during the development of this system

6

Conclusions & Future Work

Summary the advantages and disadvantages .

Introduction

MOTIVATION

- “The world's largest technical professional society”
- “There is no optimal system that can support a variety of lifecycle conference needs while providing outstanding extensibility, flexibility, reusability, configurability, reliability”
- “We aim to design and develop an SOA solution toward fulfilling such requirements”

Goals

Simplicity

It seems like most of the users are so dissatisfied with the current conference system due to their complexity. We simplify the system making it easy to be used

Smooth Learning Curve

Without level by level menu, we display all functionalities in limited pages. People don’t need tutorials. Everything is natural.

Lightweight

80% functionalities can be achieved by 20% scale. We keep the system lightweight

Related Work

CONFHUB

Advantages

1.Comprehensive
2.Full-featured
3.Traditional

Disadvantages

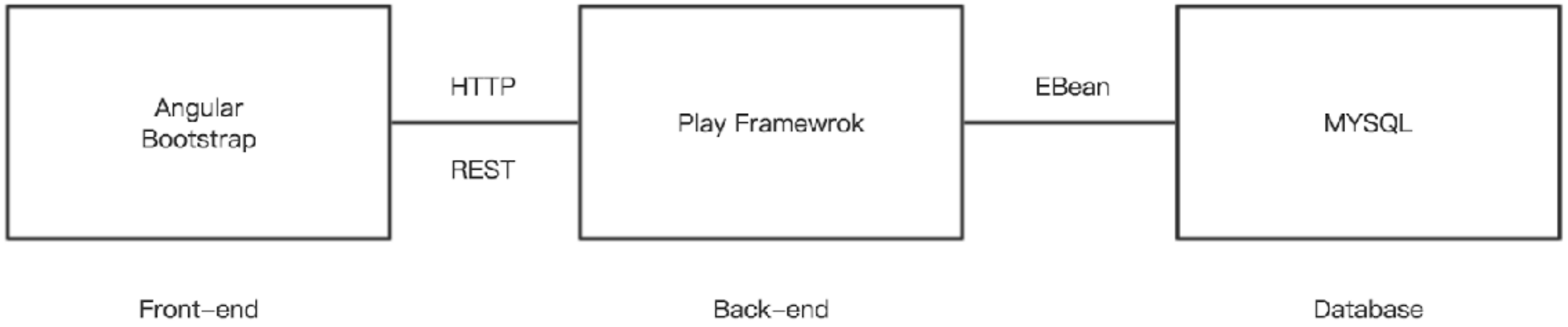
1.Complicated
2.Disordered
3.Old-fashioned

System Design

IMPLEMENTATION

MVC

Simultaneous development
High cohesion
Low coupling
Ease of modification



Angular
Bootstrap

HTTP

REST

Play Framewrok

EBean

MYSQL

Front–end

Back–end

Database

ORM

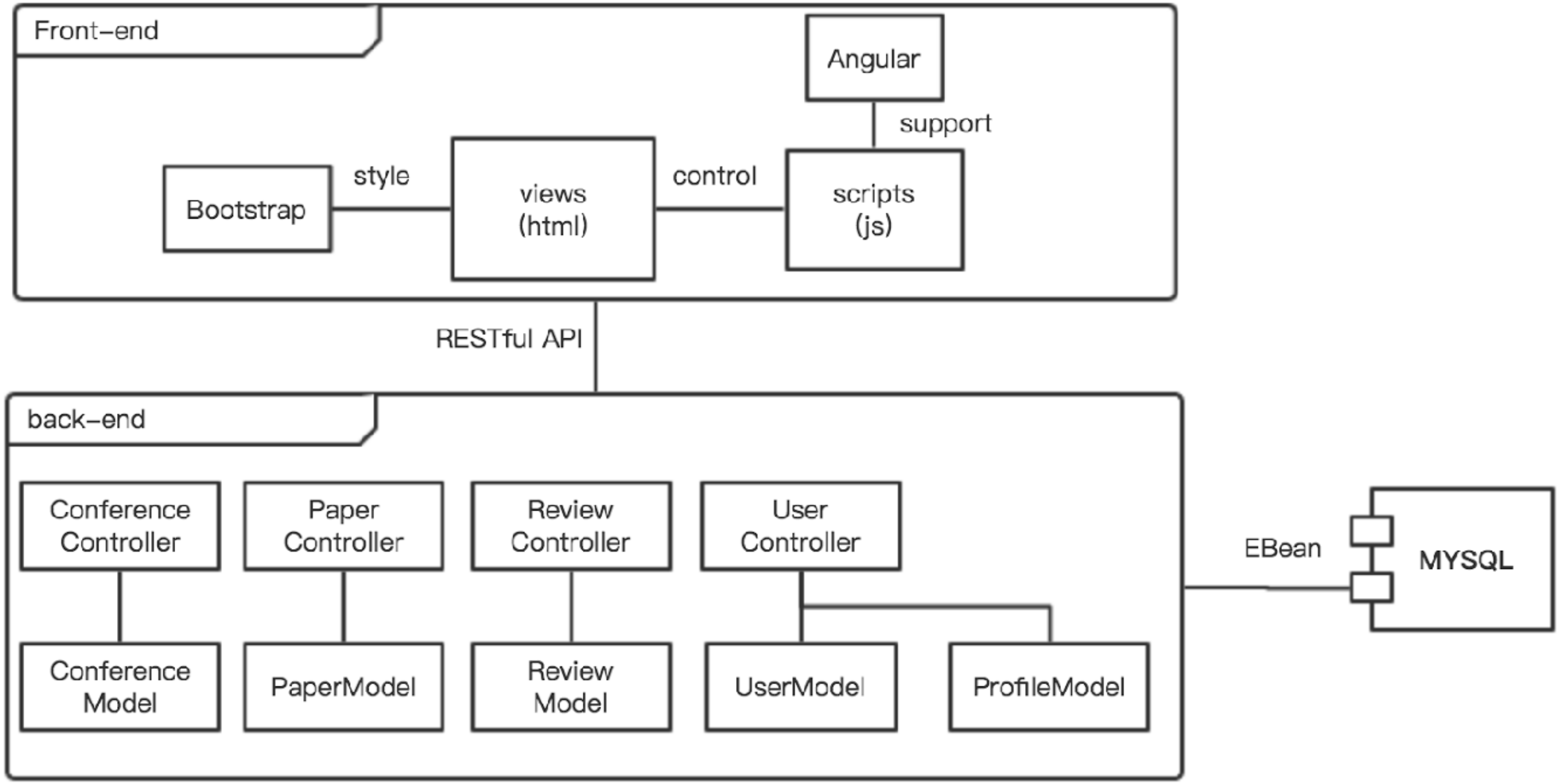
Compared to traditional techniques of exchange between an object-oriented language and a relational database, ORM often reduces the amount of code that needs to be written.

REST

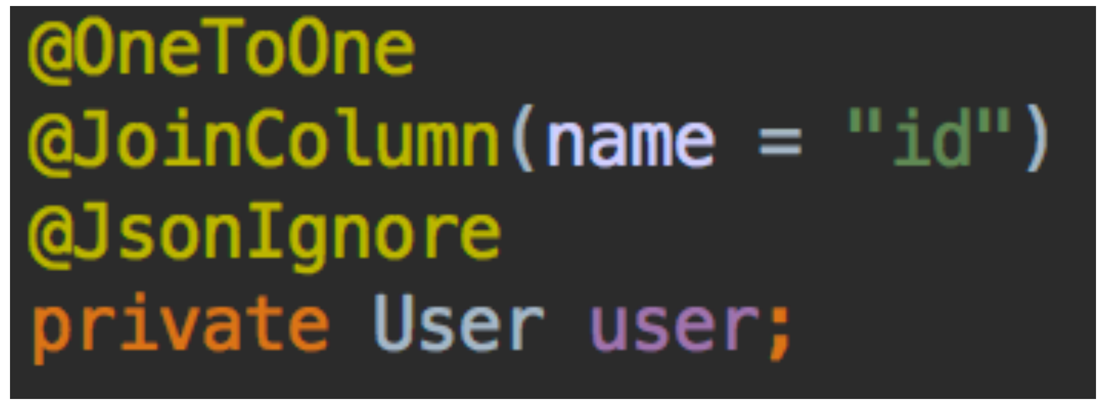
By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.

MVC IMPLEMENTATION

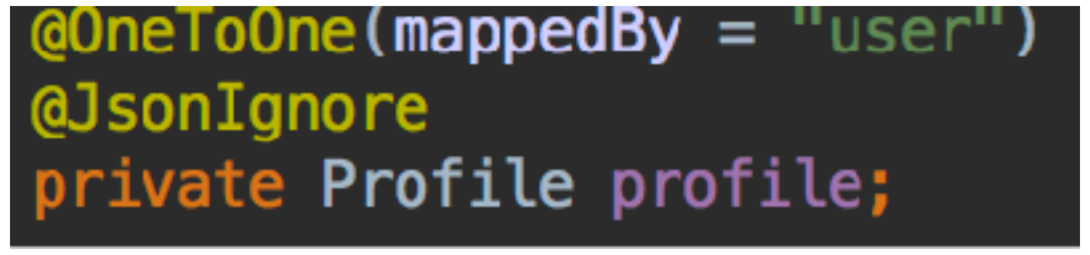
The picture below shows our MVC structure implementation.



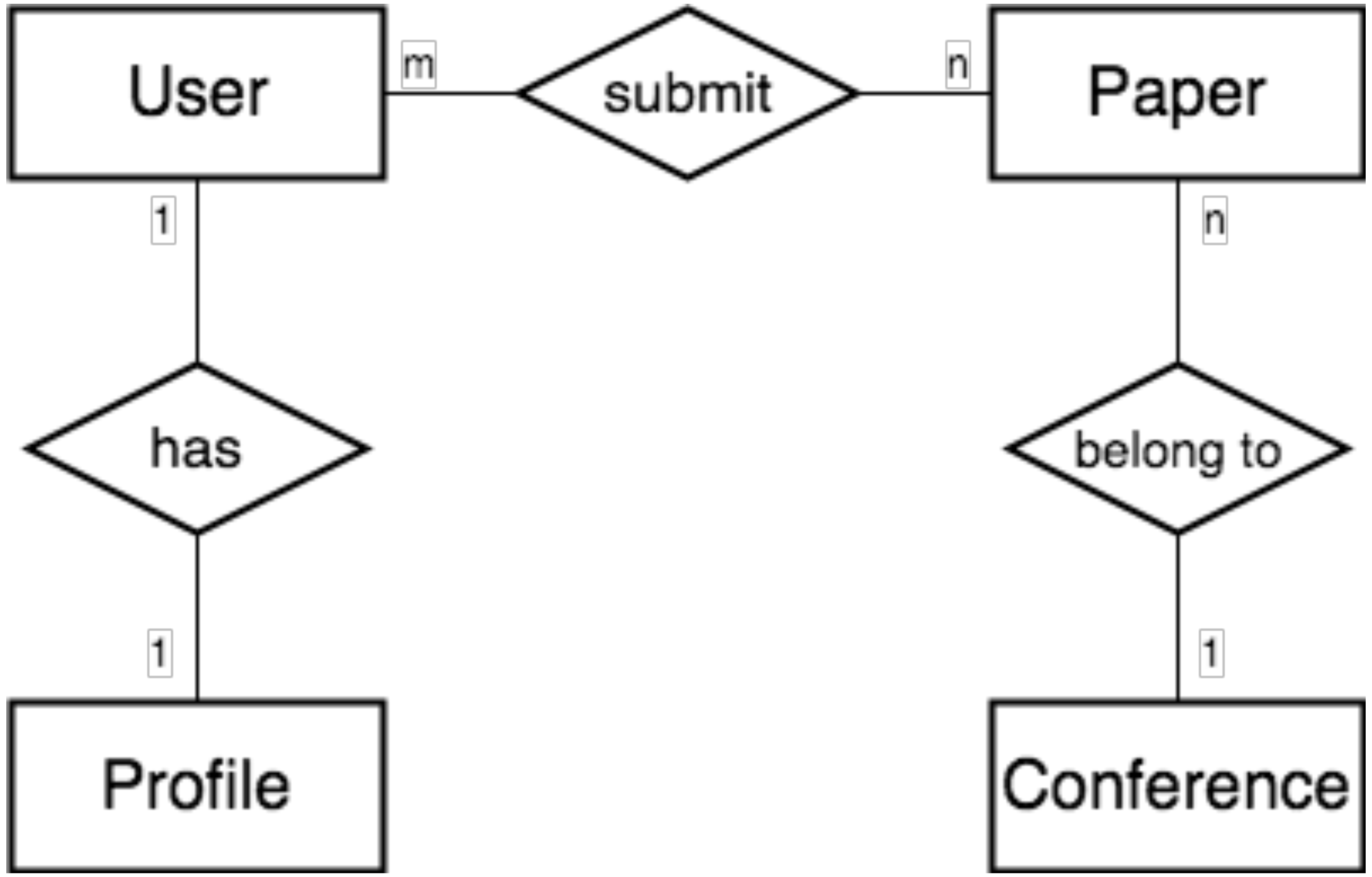
SPRINT1 E-R DIAGRAM



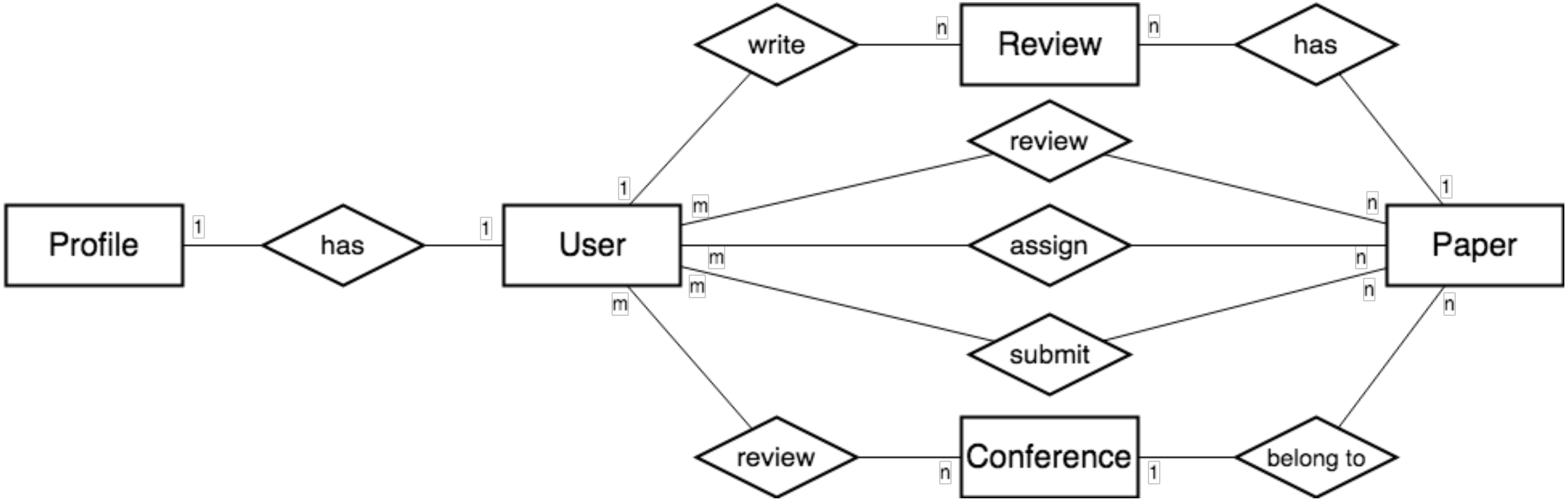
```
@OneToOne
@JoinColumn(name = "id")
@JsonIgnore
private User user;
```



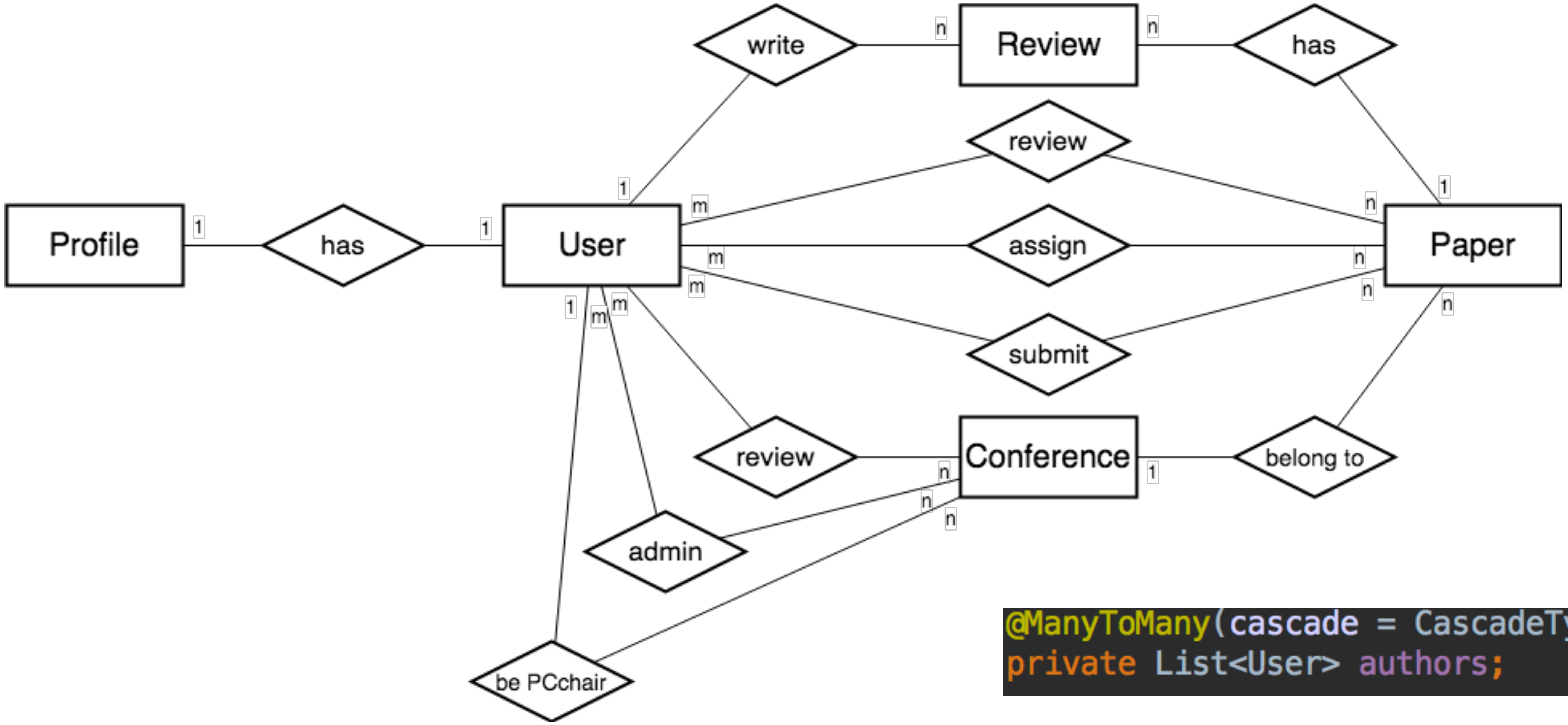
```
@OneToOne(mappedBy = "user")
@JsonIgnore
private Profile profile;
```

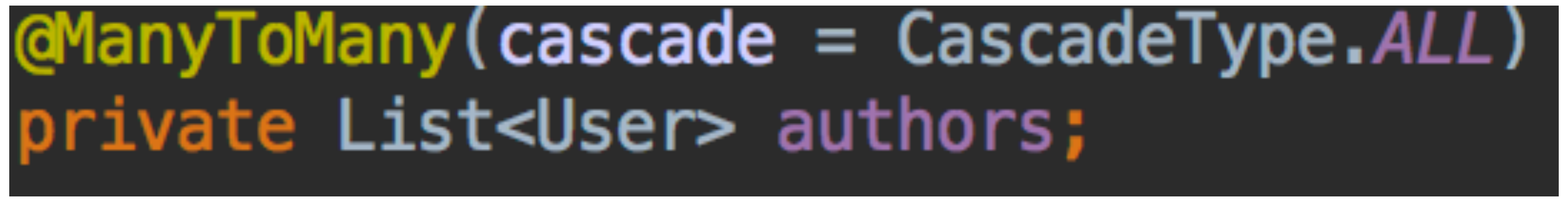


SPRINT2 E-R DIAGRAM

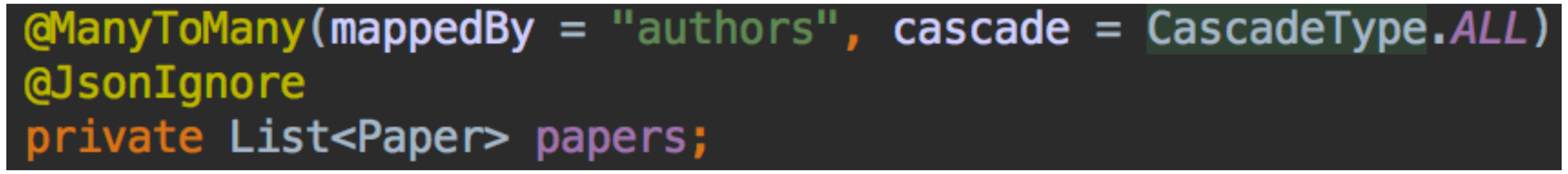


SPRINT3 E-R DIAGRAM





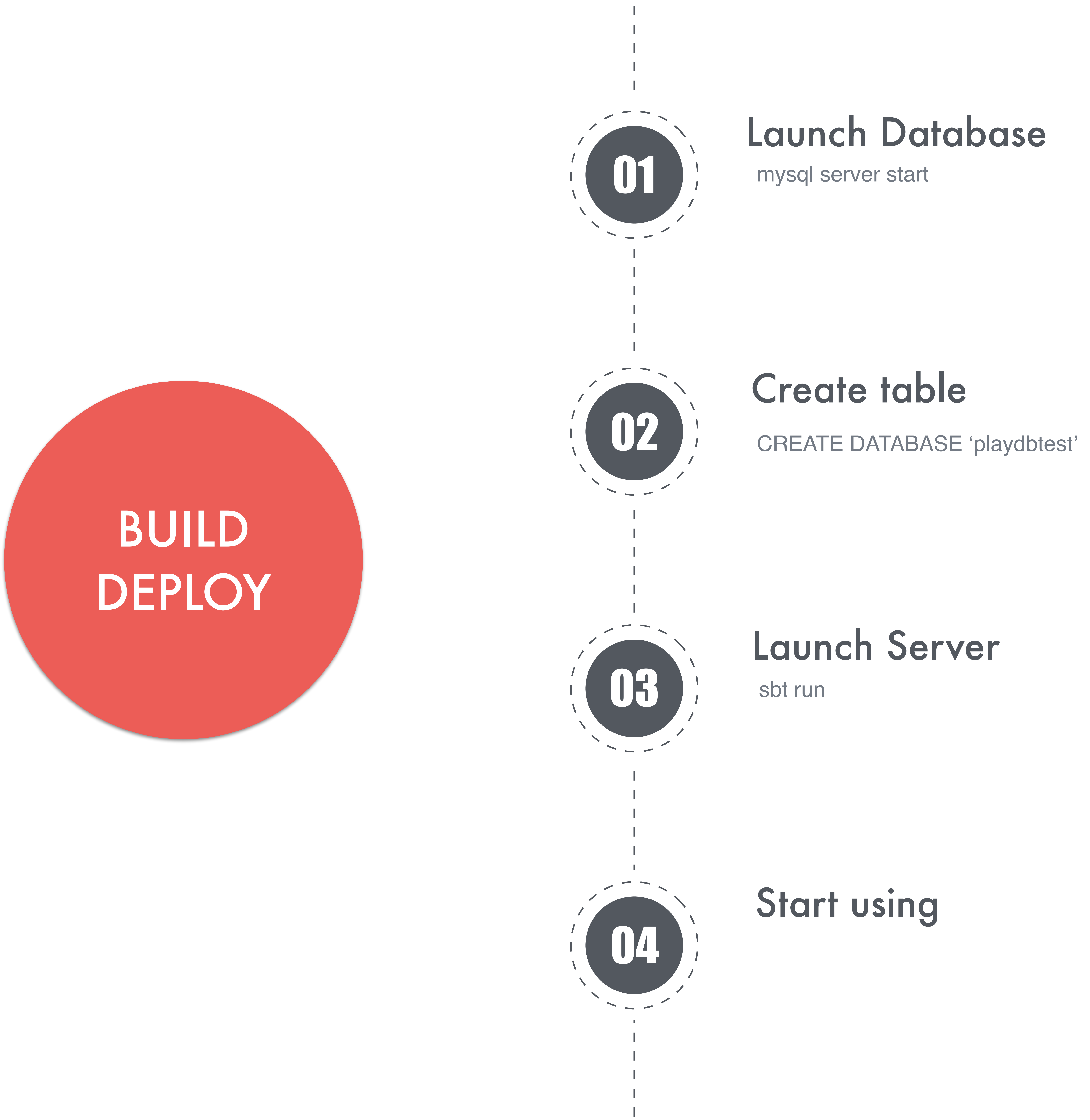
```
@ManyToMany(cascade = CascadeType.ALL)
private List<User> authors;
```



```
@ManyToMany(mappedBy = "authors", cascade = CascadeType.ALL)
@JsonIgnore
private List<Paper> papers;
```

Demo

VIDEO



BUILD
DEPLOY

01

Launch Database

mysql server start

02

Create table

CREATE DATABASE ‘playdbtest’

03

Launch Server

sbt run

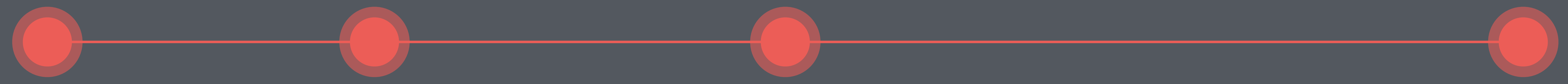
04

Start using

Technical Experiences

API DOCUMENT

We design and implement strict RESTful APIs.
During the progress, we found a good API tool
named SWAGGER hub

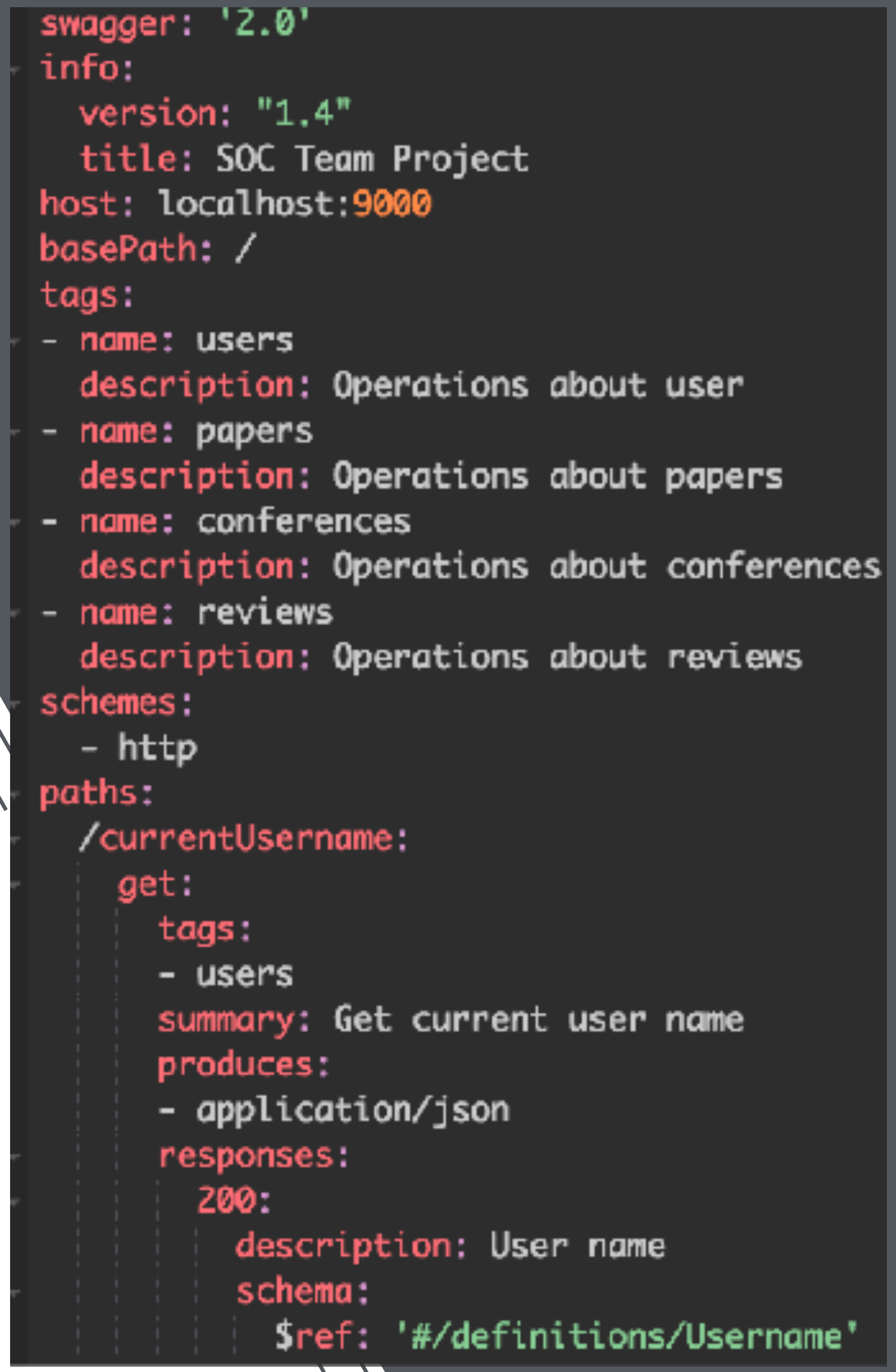


CREATE

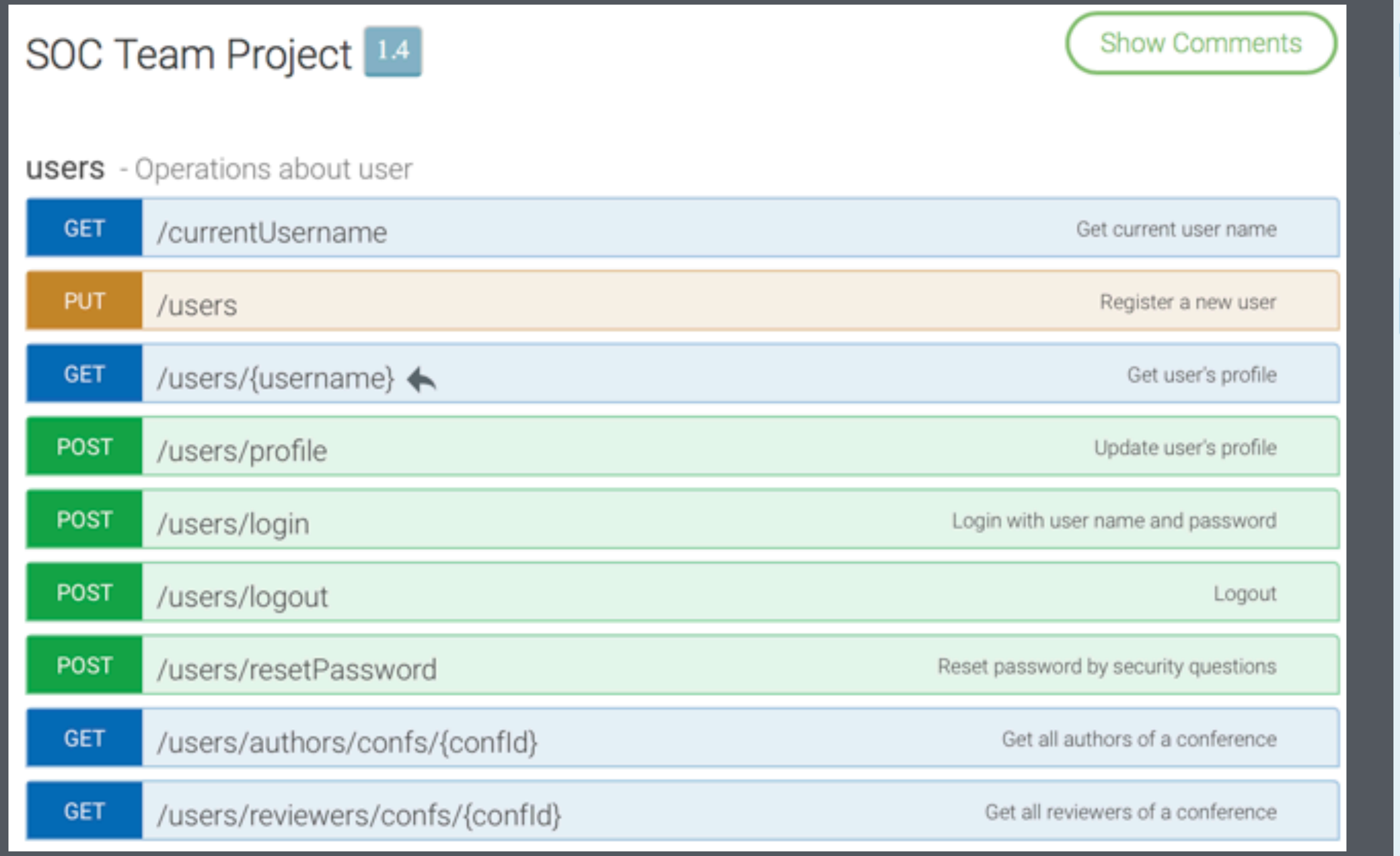
EDIT

SHARE

COOPARATION



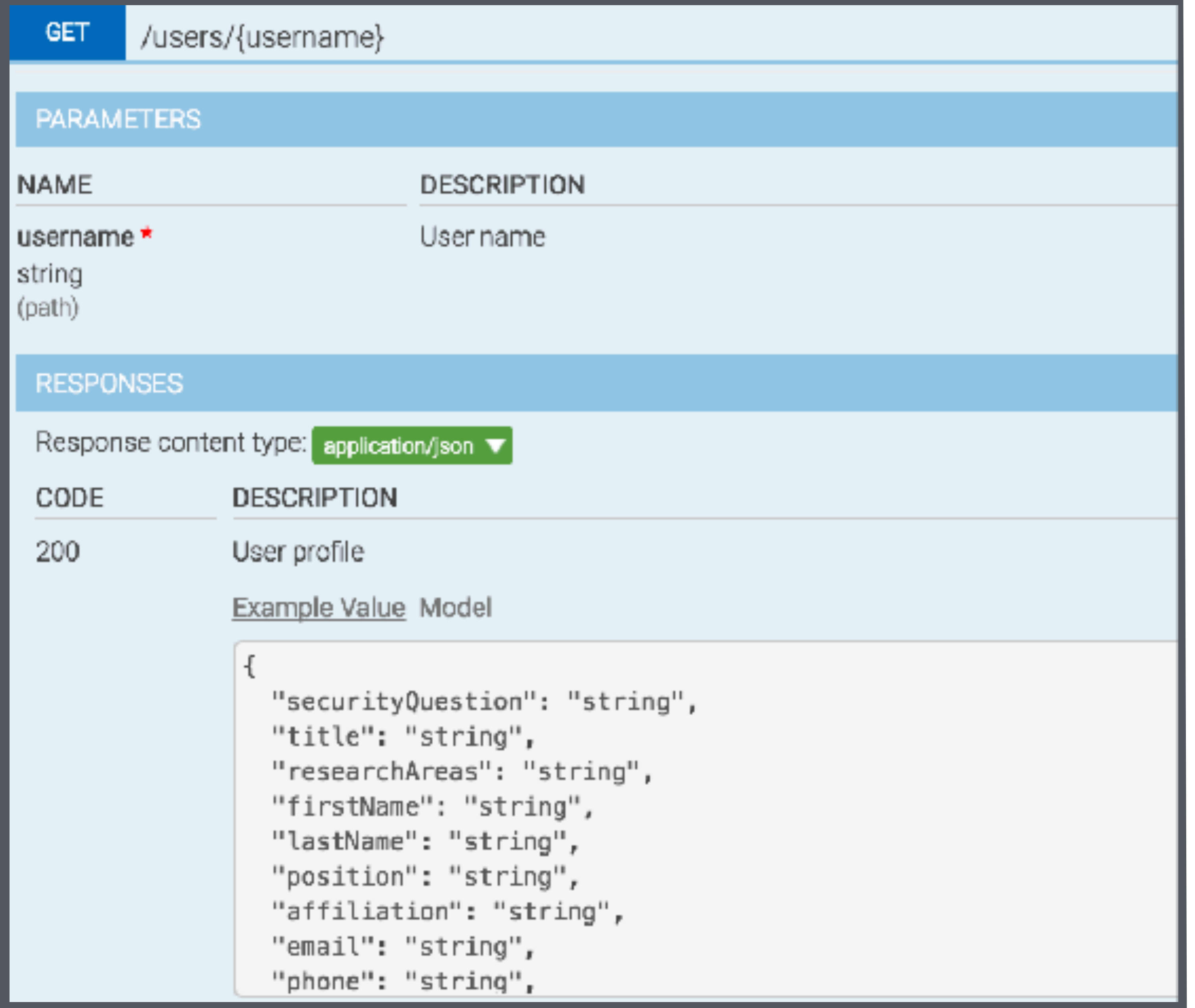
```
swagger: '2.0'
info:
  version: "1.4"
  title: SOC Team Project
host: localhost:9000
basePath: /
tags:
- name: users
  description: Operations about user
- name: papers
  description: Operations about papers
- name: conferences
  description: Operations about conferences
- name: reviews
  description: Operations about reviews
schemes:
  - http
paths:
  /currentUsername:
    get:
      tags:
      - users
      summary: Get current user name
      produces:
      - application/json
      responses:
        200:
          description: User name
          schema:
            $ref: '#/definitions/Username'
```



SOC Team Project 1.4

users - Operations about user

GET	/currentUsername	Get current user name
PUT	/users	Register a new user
GET	/users/{username}	Get user's profile
POST	/users/profile	Update user's profile
POST	/users/login	Login with user name and password
POST	/users/logout	Logout
POST	/users/resetPassword	Reset password by security questions
GET	/users/authors/confs/{confId}	Get all authors of a conference
GET	/users/reviewers/confs/{confId}	Get all reviewers of a conference



GET /users/{username}

PARAMETERS

NAME	DESCRIPTION
username * string (path)	User name

RESPONSES

Response content type: application/json

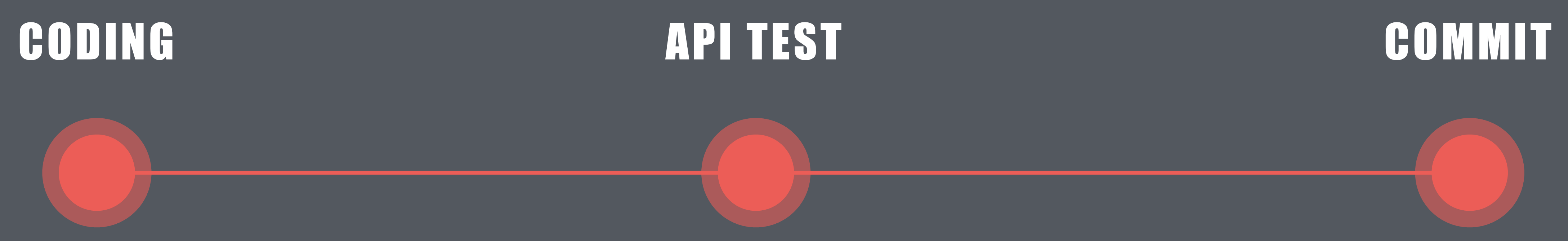
CODE	DESCRIPTION
200	User profile

```
{
  "securityQuestion": "string",
  "title": "string",
  "researchAreas": "string",
  "firstName": "string",
  "lastName": "string",
  "position": "string",
  "affiliation": "string",
  "email": "string",
  "phone": "string",
```

52

API TESTING

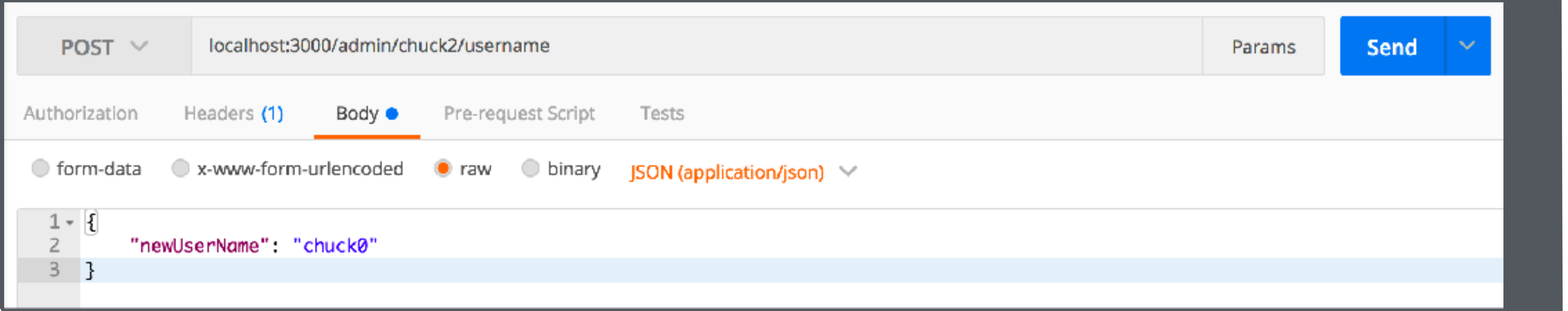
Test RESTful API by Postman
Commit codes only if they pass API testing
Minimize integration testing cost



CODING

API TEST

COMMIT



POST localhost:3000/admin/chuck2/username

○ form-data ○ x-www-form-urlencoded ● raw ○ binary JSON (application/json)

```
{
    "newUserName": "chuck0"
}
```

DATABASE INITIALIZATION

01

Set scripts

Place SQL scripts under conf/evolutions/default

02

Start server

Initialize data automatically instead of creating test data manually

03

Start Testing

Conclusion

SUMMARY

2017.03

SPRINT 1

Get familiar with play framework and ORM technology

2017.04

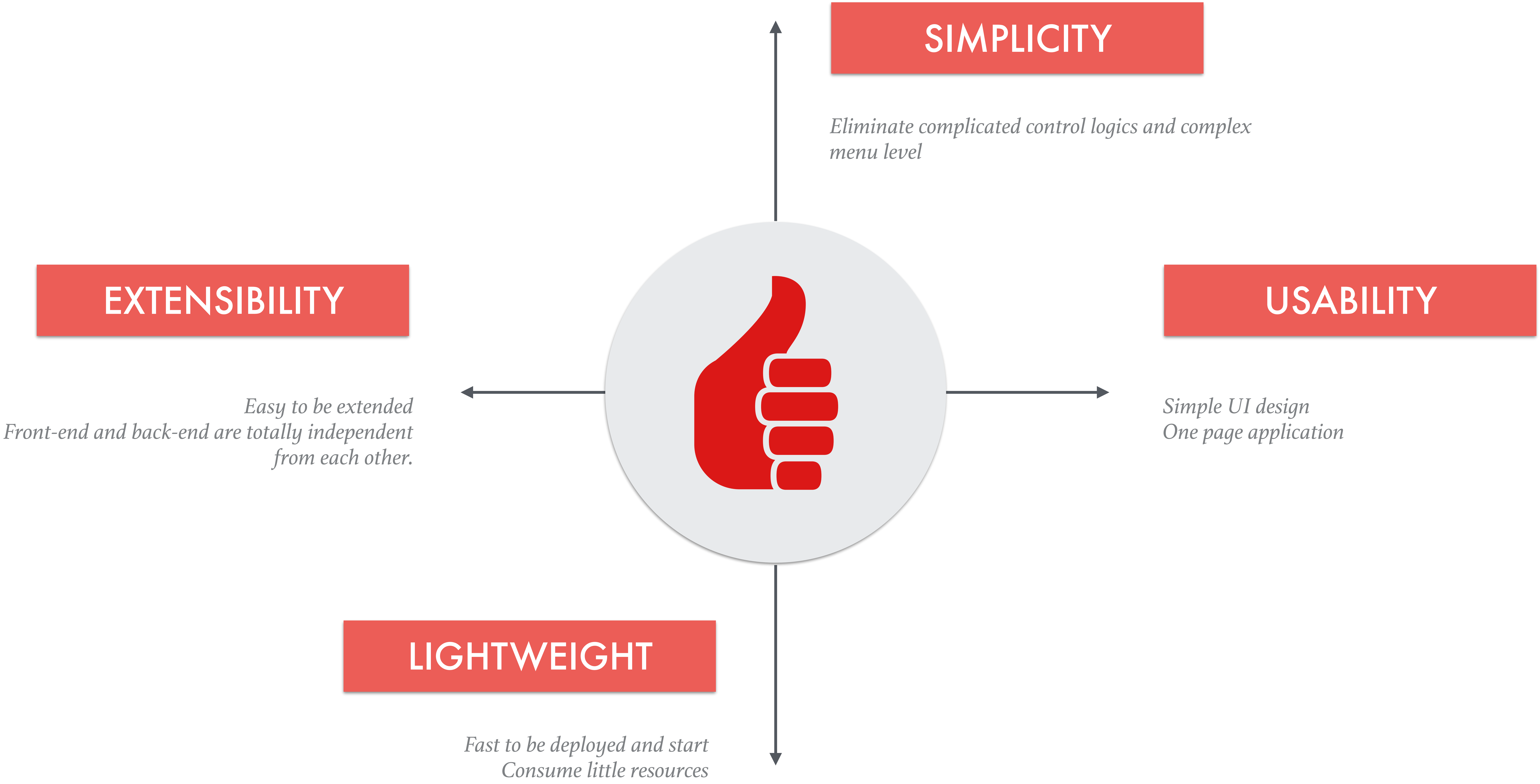
SPRINT 2

Extent models and design tons of new RESTful API.
Implement new features

2017.05

SPRINT 3

Finish all features and make interfaces user friendly



SIMPLICITY

Eliminate complicated control logics and complex menu level

USABILITY

Simple UI design
One page application

LIGHTWEIGHT

Fast to be deployed and start
Consume little resources

EXTENSIBILITY

Easy to be extended
Front-end and back-end are totally independent from each other.

NEED TO BE IMPROVED

01

ROBUSTNESS

INSUFFICIENT FRONT-END AND BACK-END ERROR HANDLING

02

SECURITY

POTENTIAL SECURITY THREATS SUCH LIKE SQL INJECTION

03

TEST

INSUFFICIENT UNIT AND UI TEST

THANK YOU