CS 353 - DATABASE SYSTEMS PROJECT DESIGN REPORT

Digital Application Distribution Service

Group 5

Görkem YILMAZ - 21601927
Arda TÜRKOĞLU - 21601187
Esad Burak ALTINYAZAR - 21601771
Burak YENI - 21502761

Table of Contents

1. Revised E/R Model

After our teaching assistant reviewed our proposal report, we received feedback from the assistant and revised the E/R model according to feedback. We have corrected the mistakes on our E/R model and removed, added some attributes or tables. The changes are as follows;

Changes for entities of E/R model:

- We added Device entity.
- We added follow relation to the User.
- We added message relation to the User.
- We removed developer attribution from application.
- We removed weak entity between category and application.
- We added has relation between developer and application.
- We added Request entity.
- We added handle relation between Request and Editor.
- We removed request relation.
- We added has relation between Developer and Request.
- We added of relation between request and application.
- We removed Statistics.
- We gather up Editor,User and Developer under User with inheritance.

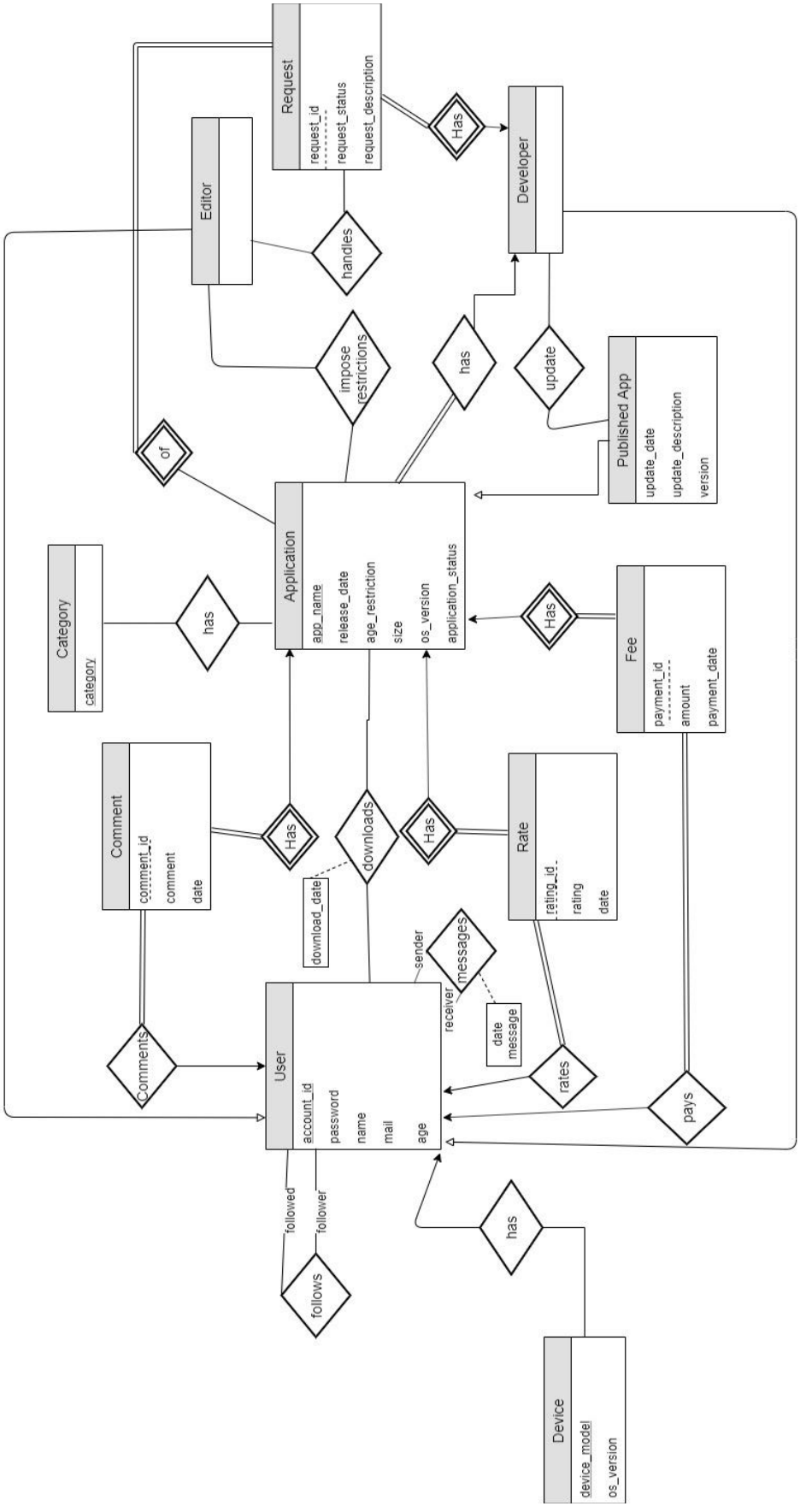


Figure 1. Revised ER Diagram

2. Relation Schemas

2.1 Editor

Relational Model:

Editor(ref_account_id, password, name, mail, age)

Functional Dependencies: {

ref_account_id ->password, name, mail, age

}

Candidate Key: {

(ref_account_id)

}

Normal Form: BCNF

Table Definition:

```
create table Editor(
        ref_account_id      INT PRIMARY KEY not null,
        password      VARCHAR(16) not null,
        name VARCHAR(16) not null,
        mail VARCHAR(50) not null,
        age numeric(2) not null,
        Foreign key (ref_account_id) References User(account_id)
);
```

2.2 Developer

Relational Model:

Developer(ref_developer_id, password, name, mail, age)

Functional Dependencies: {

ref_developer_id ->password, name, mail, age

}

Candidate Keys: {

(ref_developer_id)

}

Normal Form: BCNF

Table Definition:

```
create table Developer(
        ref_developer_id      INT PRIMARY KEY,
        password      VARCHAR(16) not null,
        name VARCHAR(16) not null,
        mail VARCHAR(50) not null,
        age numeric(2) not null,
        Foreign key (ref_developer_id) References User(account_id)
);
```

2.3 Request

Relational Model:

Request(request_id, account_id, request_status, request_description)

Functional Dependencies: {

request_id -> request_status, request_description,account_id

}

Candidate Keys: {

(request_id)

}

Normal Form: BCNF

Table Definition:

```
create table Request (
        primary key request_id          INT not null,
        request_status                  ENUM(approved, rejected,
approved_with_restrictions),
        request_description             VARCHAR(300),
        foreign key(account_id) references User(account_id)
    );
```

2.4 Published App

Relational Model:

Published App(app_name, release_date, age_restriction, size, os_version, application_status, update_date, update_description, version)

Foreign key app_name references Application

Functional Dependencies: {

app_name->release_date, age_restriction, size, os_version, application_status, update_date, update_description, version

}

Candidate Keys: {

(app_name)

Normal Form: BCNF

Table Definition:

```
create table Published App (
        app_name            VARCHAR(32) not null,
        release_date TIMESTAMP,
        age_restriction NUMERIC(2),
        size INT,
        os_version VARCHAR(20),
        application_status ENUM(approved, rejected, approved_with_restrictions),
        update_date         TIMESTAMP,
        update_description  VARCHAR(255),
        version             VARCHAR(32),
        foreign key (app_name) references Application(app_name)
    );
```

2.5 Application

Relational Model:

Application(app_name, ref_account_id, release_date, age_restriction, size, application_status, app_requirements)

Functional Dependencies: {

app_name -> ref_account_id, release_date, age_restriction, size, application_status, app_requirements

ref_account_id, release_date -> app_name , age_restriction, size, application_status, app_requirements

}

Candidate Keys: {

(app_name)

(ref_account_id,release_date)

}

Normal Form: BCNF

Table Definition:

```
create table Application (
        app_name    VARCHAR(32) PRIMARY KEY,
        ref_account_id      VARCHAR(32) PRIMARY KEY,
        foreign key (ref_account_id) References User(account_id),
```

```
    release_date VARCHAR(32),
    age_restriction VARCHAR(32),
    size INT,
    application_status VARCHAR(32),
    app_requirements VARCHAR(255)
);
```

2.6 Category

Relational Model:

Category(category)

Functional Dependencies: {

none

}

Candidate Keys: {

(category)

}

Normal Form: BCNF

Table Definition:

```
create table Category (
    category        VARCHAR(32) PRIMARY KEY,
);
```

2.7 Fee

Relational Model:

Fee(payment_id, app_name, user_id, amount, payment_date)

Functional Dependencies: {

payment_id -> app_name, user_id, amount, payment_date

}

Candidate Keys: {

(payment_id)

}

Normal Form: BCNF

Table Definition:

```
create table Fee(
        payment_id   INT PRIMARY KEY,
        foreign key (app_name) References Application(app_name),
        foreign key (user_id) References User(account_id),
        amount INT,
        payment_date VARCHAR(32),
        );
```

2.8 Comment

Relational Model:

Comment(comment_id, user_id, app_name, comment, date)

Functional Dependencies: {

comment_id -> user_id, comment, date

}

Candidate Keys: {

(comment_id)

}

Normal Form: BCNF

Table Definition:

```
create table Comment(
        comment_id  INT PRIMARY KEY,
        comment VARCHAR(255),
        foreign key (user_id) References User(account_id),
        foreign key (app_name) references Application(app_name),
        date TIMESTAMP,
        );
```

2.9 Rate

Relational Model:

Rate(rating_id, user_id, app_name, rating, date)

Functional Dependencies: {

rating_id -> user_id , app_name, rating, date

}

Candidate Keys: {

(rating_id)

}

Normal Form: BCNF

Table Definition:

```
create table Rating(
        rating_id       INT PRIMARY KEY,
        foreign key (user_id) References User(account_id),
        foreign key (app_name) References Application(app_name),
        rating NUMERIC(1),
        Date TIMESTAMP
        );
```

2.10 User

Relational Model:

User(account_id, password, name, mail, age)

Functional Dependencies: {

account_id-> password, name, mail, age

}

Candidate Keys: {

(account_id)

}

Normal Form: BCNF

Table Definition:

```
create table User(
        account_id    INT PRIMARY KEY,
        password      VARCHAR(32)
        age INT,
        name VARCHAR(32),
        mail  VARCHAR(32)
        );
```

2.11 Device

Relational Model:

Device(user_id, device_model, os_version)

Functional Dependencies: {

none

}

Candidate Keys: {

none

}

Normal Form: BCNF

Table Definition:

```
create table Device(
        user_id VARCHAR(32) PRIMARY KEY ,
        foreign key (user_id) References User(account_id),
        device_model VARCHAR(32) PRIMARY KEY,
        Os_version VARCHAR(255)
        );
```

2.12 Handles

Relational Model:

Handles(request_id, editor_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(request_id, editor_id)

}

Normal Form: BCNF

Table Definition:

```
create table Handles(
        request_id                  INT PRIMARY KEY not null,
        editor_id                   INT PRIMARY KEY not null,
        Foreign key (request_id) References Requests(request_id),
        Foreign key (editor_id) References Editor(ref_account_id),
        );
```

2.13 Request_has

Relational Model:

Request_has(request_id, ref_developer_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(request_id, ref_developer_id)

}

Normal Form: BCNF

Table Definition:

```
create table Request_has(
        request_id                  INT PRIMARY KEY not null,
        ref_developer_id            INT PRIMARY KEY not null,
        Foreign key (request_id) References Requests(request_id),
        Foreign key (ref_developer_id) References Developer(ref_developer_id),
        );
```

2.14 Application_Has

Relational Model:

Application_Has(app_name, ref_developer_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, ref_developer_id)

}

Normal Form: BCNF

Table Definition:

```
create table Application_Has(
        app_name                    VARCHAR(32) PRIMARY KEY,
        ref_developer_id            INT PRIMARY KEY not null ,
        Foreign key (app_name) References Application(app_name),
        Foreign key (ref_developer_id) References Developer(ref_developer_id),
        );
```

2.15 ImposeRestrictions

Relational Model:

ImposeRestrictions(app_name, editor_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, editor_id)

}

Normal Form: BCNF

Table Definition:

```
create table ImposeRestrictions(
        app_name                    VARCHAR(32) PRIMARY KEY,
        editor_id                   VARCHAR(32) PRIMARY KEY not null,
        Foreign key (app_name) References Application(app_name),
        Foreign key (editor_id) References Editor(account_id),
        );
```

2.16 Update

Relational Model:

Update(app_name, ref_developer_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, ref_developer_id)

}

Normal Form: BCNF

Table Definition:

```
create table Update(
        app_name                VARCHAR(32) PRIMARY KEY not null,
        ref_developer_id        INT PRIMARY KEY not null,
        Foreign key (app_name) References Published App(app_name),
        Foreign key (ref_developer_id_) References Developer(account_id_),
        );
```

2.17 Of

Relational Model:

Of(app_name, request_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, request_id)

}

Normal Form: BCNF

Table Definition:

```
create table Handles(
        app_name                VARCHAR(32) PRIMARY KEY not null,
        request_id              INT PRIMARY KEY not null,
        Foreign key (app_name) References Application(app_name),
        Foreign key (request_id) References Requests(request_id));
```

2.18 Category_has

Relational Model:

Category_has(category, app_name)

Functional Dependencies: {

none

}

Candidate Keys: {

(category, app_name)

}

Normal Form: BCNF

Table Definition:

```
create table Category_has(
        category                VARCHAR(32) PRIMARY KEY not null,
        app_name                VARCHAR(32) PRIMARY KEY not null,
        Foreign key (category) References Category(category),
        Foreign key (app_name) References Application(app_name),
        );
```

2.19 Fee_has

Relational Model:

Fee_has(app_name, payment_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, payment_id)

}

Normal Form: BCNF

Table Definition:

```
create table Fee_has(
        app_name                VARCHAR(32) PRIMARY KEY not null,
        payment_id              INT not null,
        Foreign key (app_name) References Application(app_name),
        Foreign key (payment_id) References Fee(payment_id),
        );
```

2.20 Rate_has

Relational Model:

Rate_has(app_name, rating_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(app_name, rating_id)

}

Normal Form: BCNF

Table Definition:

```
create table Rate_has(
        app_name                    VARCHAR(32) PRIMARY KEY not null,
        rating_id                   INT PRIMARY KEY not null,
        Foreign key (app_name) References Requests(app_name),
        Foreign key (rating_id) References Rate(rating_id),
        );
```

2.21 Downloads

Relational Model:

Downloads(account_id, app_name)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id, app_name)

}

Normal Form: BCNF

Table Definition:

```
create table Downloads(
        app_name                    VARCHAR(32) PRIMARY KEY not null,
        account_id                  INT PRIMARY KEY not null,
        Foreign key (account_id) References Requests(account_id),
        Foreign key (app_name) References Application(app_name),
        );
```

2.22 Comment_has

Relational Model:

Comment_has(comment_id, app_name)

Functional Dependencies: {

none

}

Candidate Keys: {

(comment_id, app_name)

}

Normal Form: BCNF

Table Definition:

```
create table Comment_has(
        app_name                VARCHAR(32) PRIMARY KEY not null,
        comment_id              INT PRIMARY KEY not null,
        Foreign key (comment_id) References Comment(comment_id),
        Foreign key (app_name) References Application(app_name),
        );
```

2.23 Comments

Relational Model:

Comments(account_id, comment_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id, comment_id)

}

Normal Form: BCNF

Table Definition:

```
create table Comments(
        account_id                  VARCHAR(32) PRIMARY KEY not null,
        comment_id                  INT PRIMARY KEY not null,
        Foreign key (account_id) References User(account_id),
        Foreign key (comment_id) References Comment(comment_id),
        );
```

2.24 Follows

Relational Model:

Follows(account_id1, account_id2)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id1, account_id2)

}

Normal Form: BCNF

Table Definition:

```
create table Follows(
        account_id1                 VARCHAR(32) PRIMARY KEY not null,
        account_id2                 VARCHAR(32) PRIMARY KEY not null,
        Foreign key (account_id1) References User(account_id),
        Foreign key (account_id2) References User(account_id),
        );
```

2.25 Device_Has

Relational Model:

Device_Has(account_id, device_model)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id)

}

Normal Form: BCNF

Table Definition:

```
create table Device_Has(
        device_model                VARCHAR(32) PRIMARY KEY not null,
        account_id                  VARCHAR(32) PRIMARY KEY not null,
        Foreign key (account_id) References User(account_id),
        );
```

2.26 Messages

Relational Model:

Messages(account_id1, account_id2, message, date)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id1, account_id2)

}

Normal Form: BCNF

Table Definition:

```
create table Messages(
        Account_id1     VARCHAR(32) PRIMARY KEY nut null,
        Account_id2     VARCHAR(32) PRIMARY KEY nut null,
        message         VARCHAR(255),
        Date            TIMESTAMP,
        Foreign key (account_id1) References User(account_id),
        Foreign key (account_id2) References User(account_id),
        );
```

2.27 Pays

Relational Model:

Pays(payment_id, account_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(payment_id, account_id)

}

Normal Form: BCNF

Table Definition:

```
create table Pays(
        payment_id                  INT PRIMARY KEY not null,
        account_id                  VARCHAR(32) PRIMARY KEY not null,
        Foreign key (account_id) References User(account_id),
        Foreign key (payment_id) References Fee(payment_id),
        );
```

2.28 Rates

Relational Model:

Rates(account_id, rating_id)

Functional Dependencies: {

none

}

Candidate Keys: {

(account_id, rating_id)

}

Normal Form: BCNF

Table Definition:

```
create table Rates(
        account_id                  VARCHAR(32) PRIMARY KEY not null,
        rating_id                   INT PRIMARY KEY not null,
        Foreign key (account_id) References User(account_id),
        Foreign key (rating_id) References Rate(rating_id),
        );
```

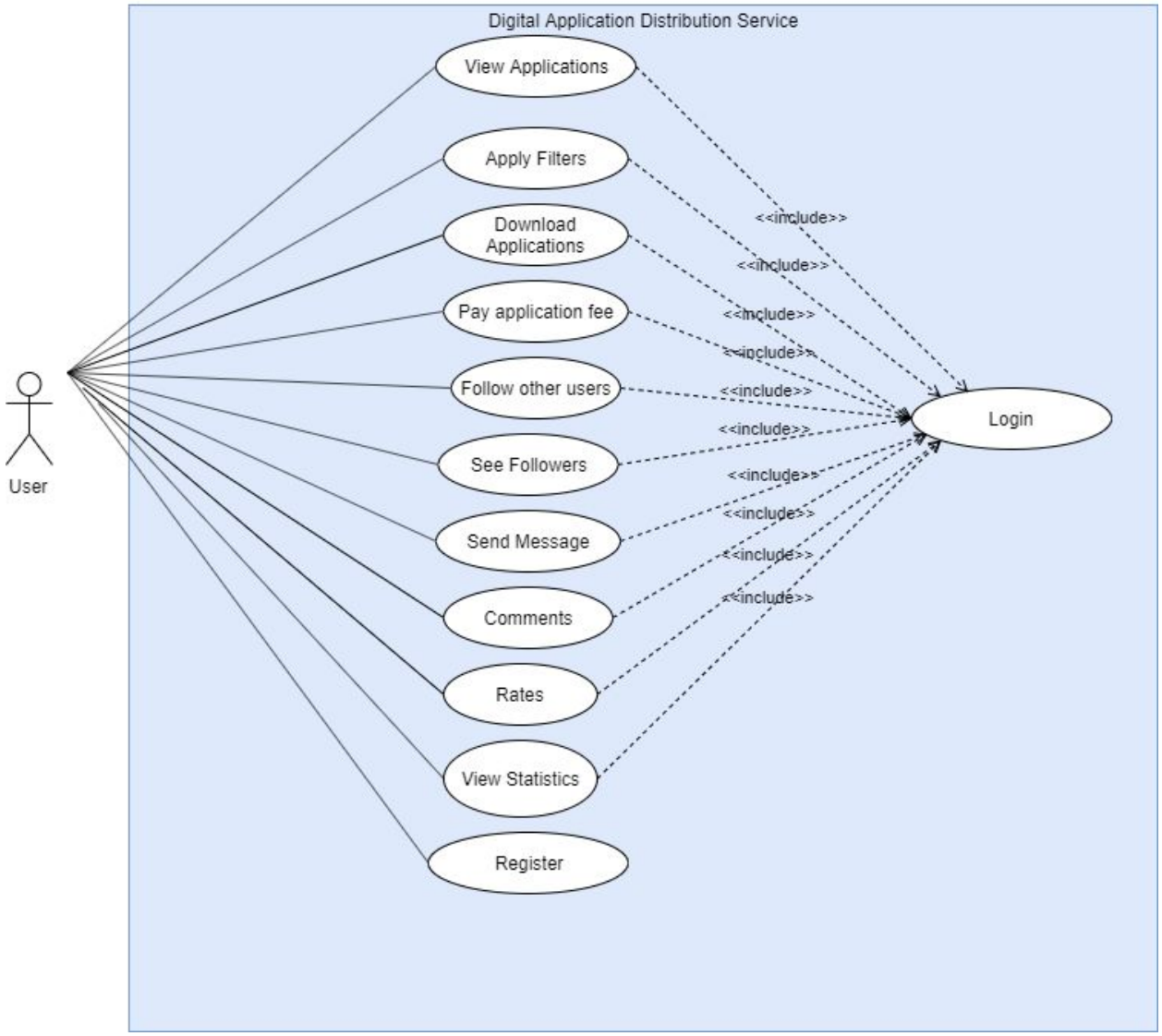
3. Functional Dependencies and Normalization of Tables

All of the schemas in our database is in Boyce Codd Normal Form (BCNF). Hence, there is no need for further normalization.

4. Functional Requirements

4.1 Use Case and Scenarios

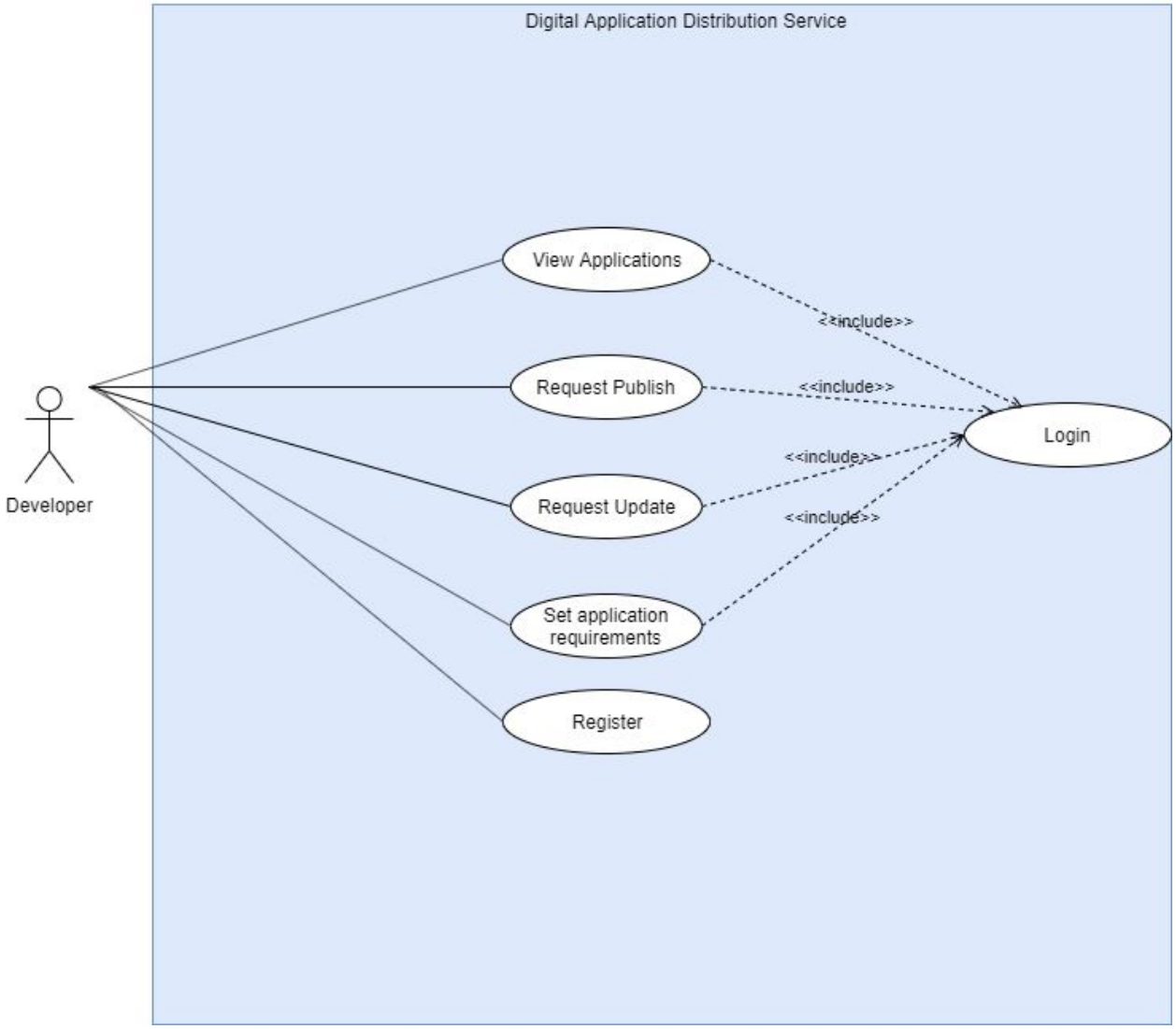
4.1.1 User



- Users can view all the applications in the system.
- Users can apply filters to view applications.
- Users can download applications.

- Users can pay for applications.
- Users can follow other users.
- Users can see their followers.
- Users can send messages to other users.
- Users can comment on applications.
- Users can rate applications.
- Users can view statistics of applications.
- Users need to login to do the thing above, otherwise they can register to the system.

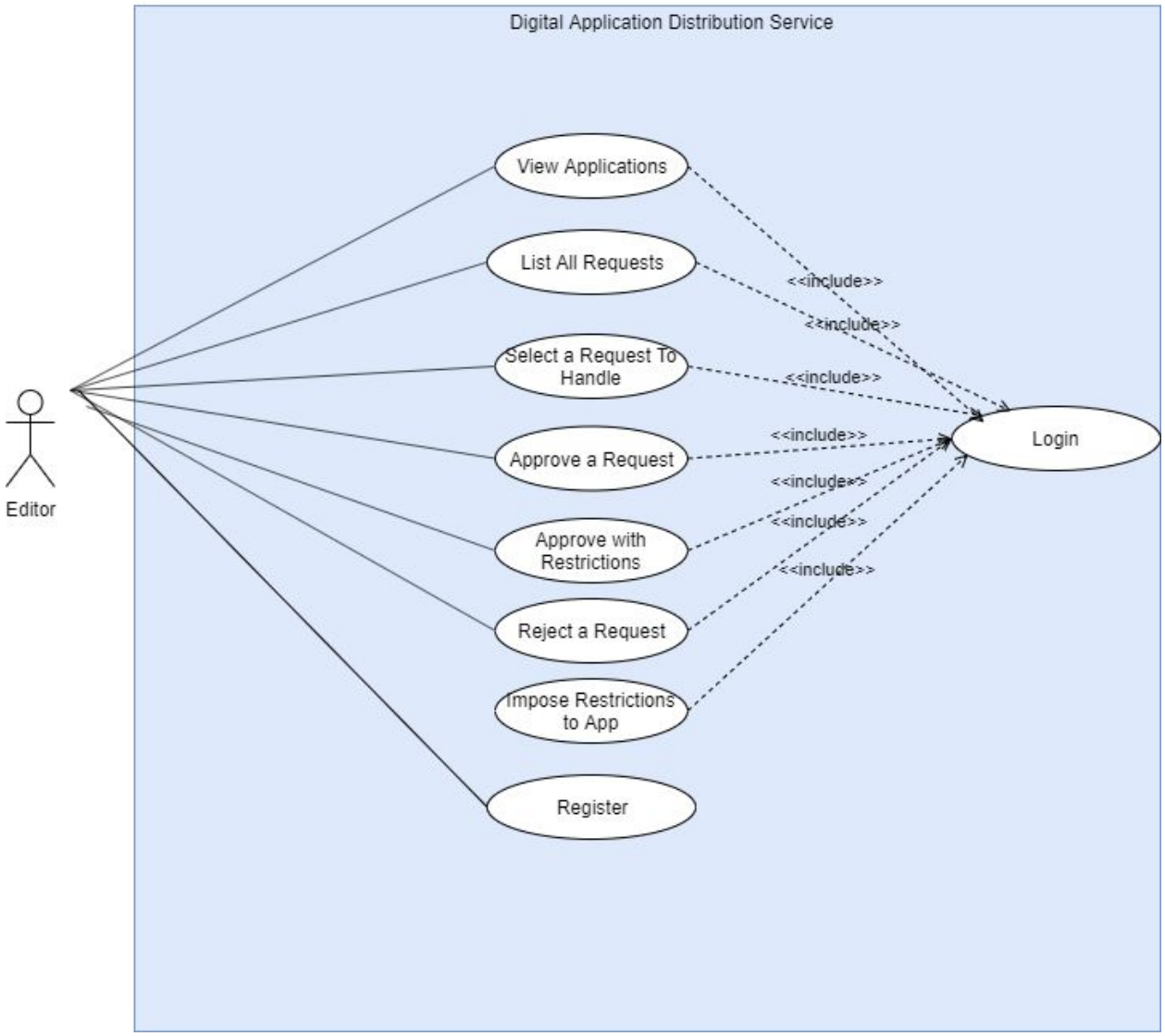
4.1.2 Developer



- Developers can view applications.
- Developers can request publishing applications.
- Developers can request updates on applications.
- Developers can set application requirements.

- Developers need to login to do things above, otherwise they can register to the system.

4.1.3 Editor



- Editors can view applications.
- Editors can list all requests.
- Editors can select a request to handle.
- Editors can approve a selected request.
- Editors can reject a selected request.
- Editors can impose restrictions to applications.
- Editors need to login to do the things above, otherwise they can register.

4.2 Algorithms

4.2.1 Download an Application(by a standard user)

Downloading an application algorithm goes as follows. First a user lists all available applications. Then selects the one he/she wants to download. Then user chooses one of his devices to download. The device that will be used needs to meet the minimum requirements. Otherwise, user cannot download the application. Total number of downloads can be seen for the application that will be downloaded. After user downloads the application, he/she can comment on or rate the application. Those comments and rates are shown in application's page. Users can also delete or edit their comments by clicking on it.

4.2.2 Publish/Update a New Application(by a developer)

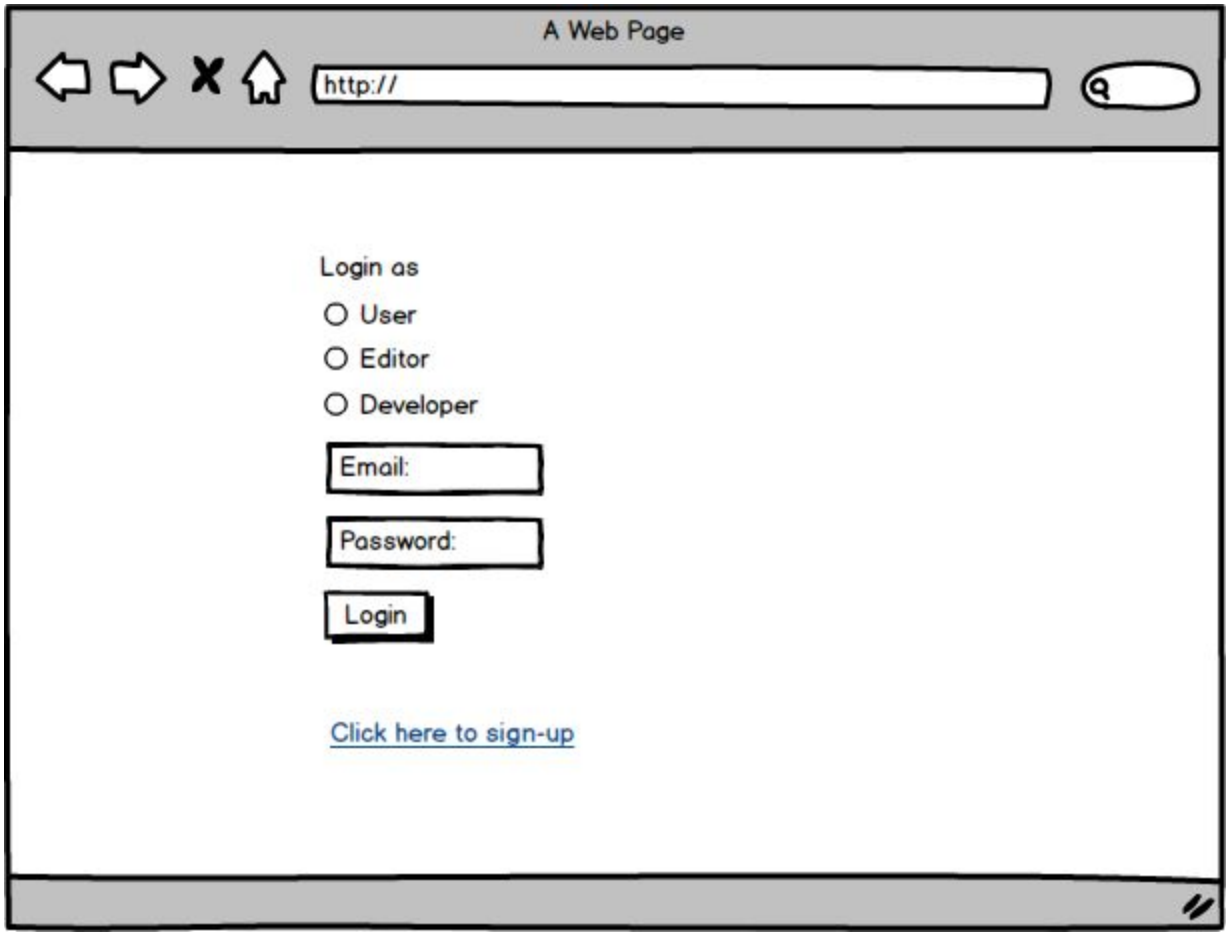
Developers can publish and update applications. In order to publish one, developers need to specify the name, description and category etc. They also need to set a minimum requirements. Lastly, they need to send a request of approval to the editor.
In order to update an application, developers need to select the application from the list of applications. Then they specify the description of update and the new version number.

4.2.3 Decide on publication/update requests(by an editor)

Editors can list all the publication and update requests in the pool. Then select a request to handle. Then editor either approves, rejects or approves with restrictions an application. According to the result of the request, editor notifies the developer.

5. User Interface Design and Corresponding SQL Statements

5.1 Log in

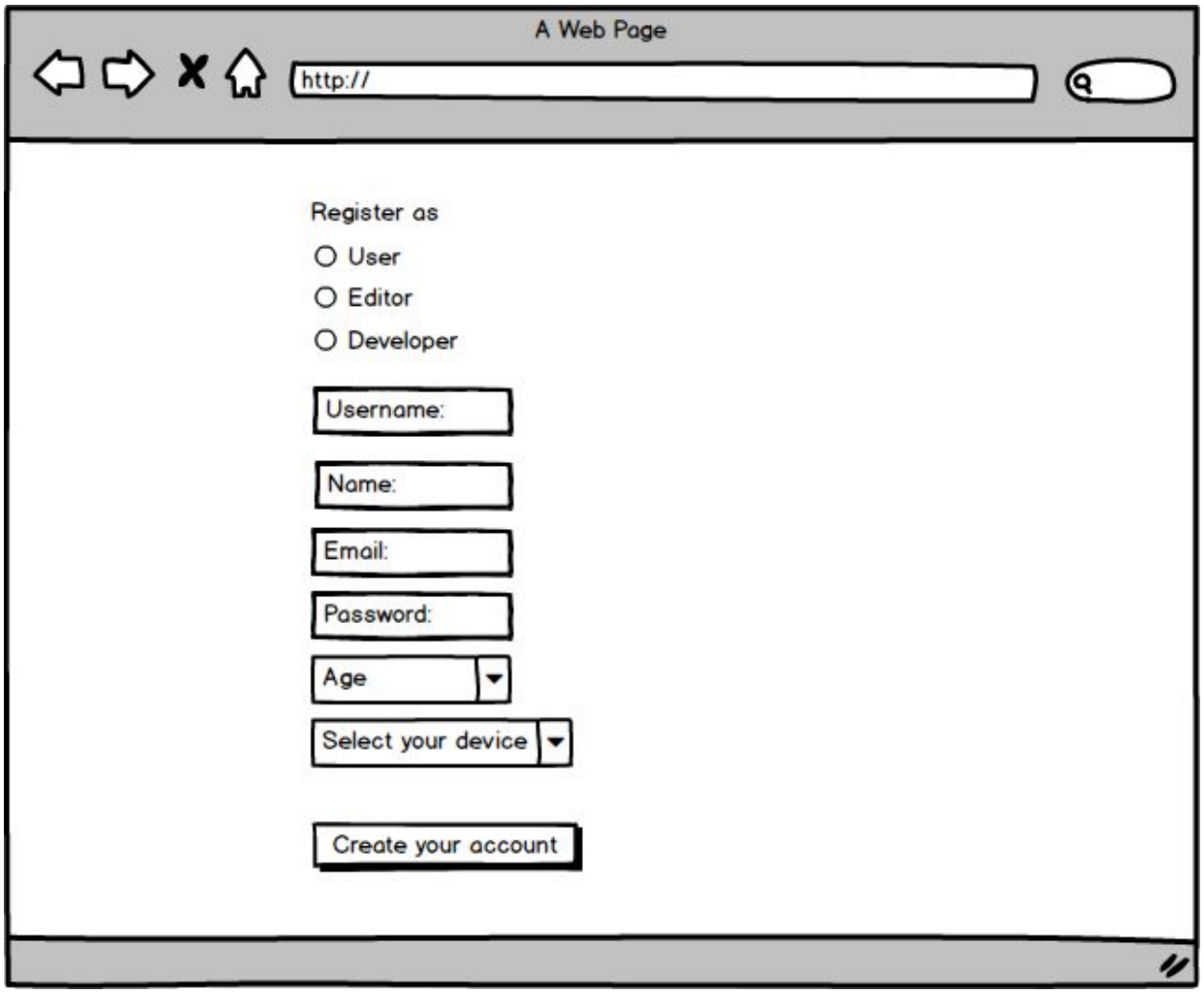


Input: @password, @account_id
Process: When a user enters the email and password, the system searches to find a tuple from User table with given email and password. If such a user exists, the system let the user to log in to the system. If email and password do not match with any of user, system gives a warning. Since there are 3 different user types, there can be need to use Editor or Developer tables in the statement below instead of User table.
Statement:

```
select case when exists(
        select *
        from User U
```

where U.account_id = @account_id and U.password = @password
) then cast (true as boolean)
else cast (false as boolean);

5.2 Sign Up

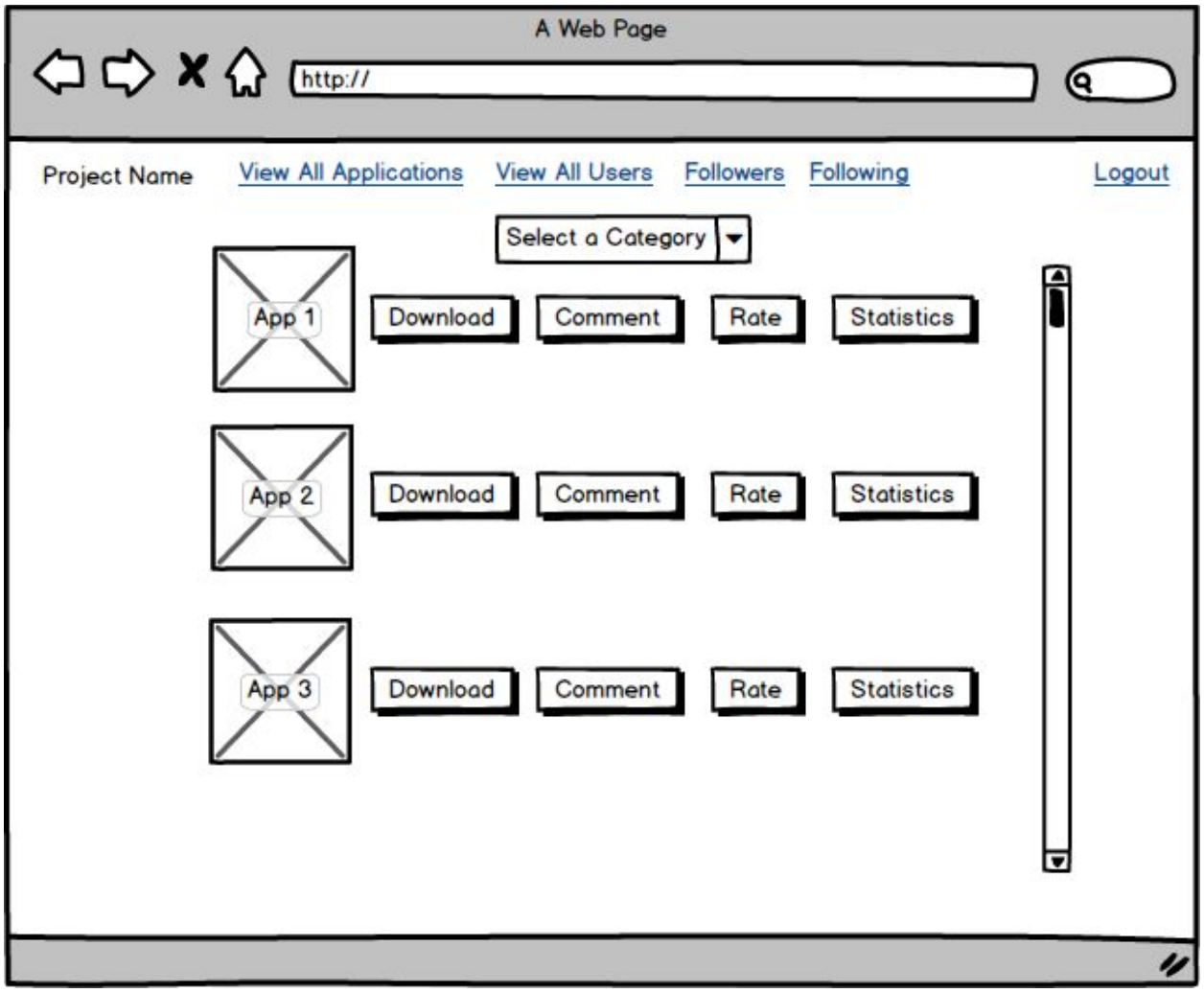


Input: @account_id @password @email, @name, @age, @device-model
Process: A person can register to the system as User, Editor or Developer by entering a valid username, name, email, password, age and their device.
Statement:

- If Editor:
  insert into Editor values (@account_id, @password, @email, @name, @age, @device-model);
  If User:

insert into User values (@account_id, @password, @email, @acc_name, @age, @device);
If Developer:
insert into Developer values (@account_id, @password, @email, @name, @age, @device-model);

5.3 User View All Applications



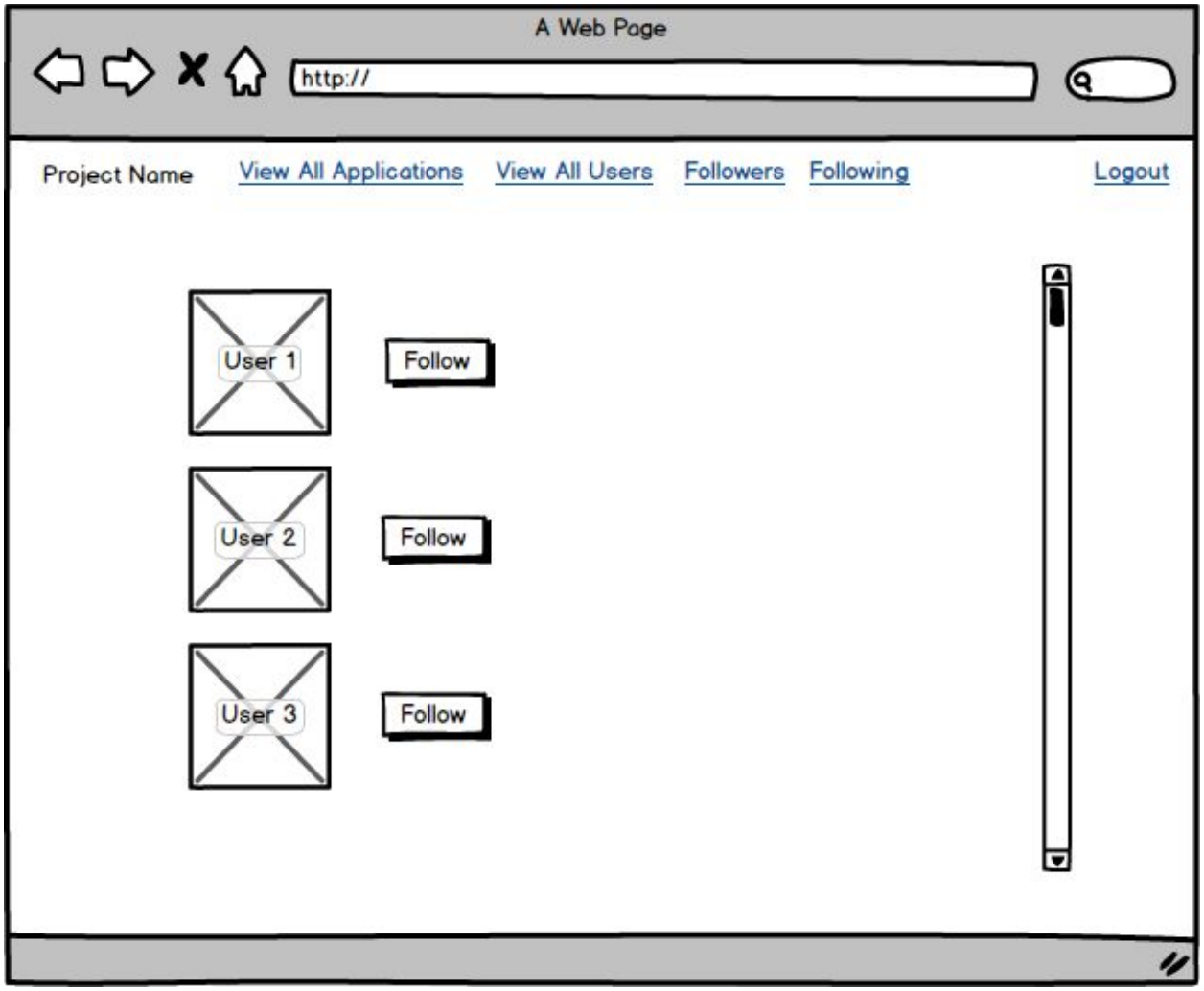
Input: @category

Process: User clicks on view all application and sees all the application on the system. User can also apply a filter to see applications in different categories.

Statement:

```
Select *
from Application A,Category C
where  C.category = @category
```

5.4 User View All Users



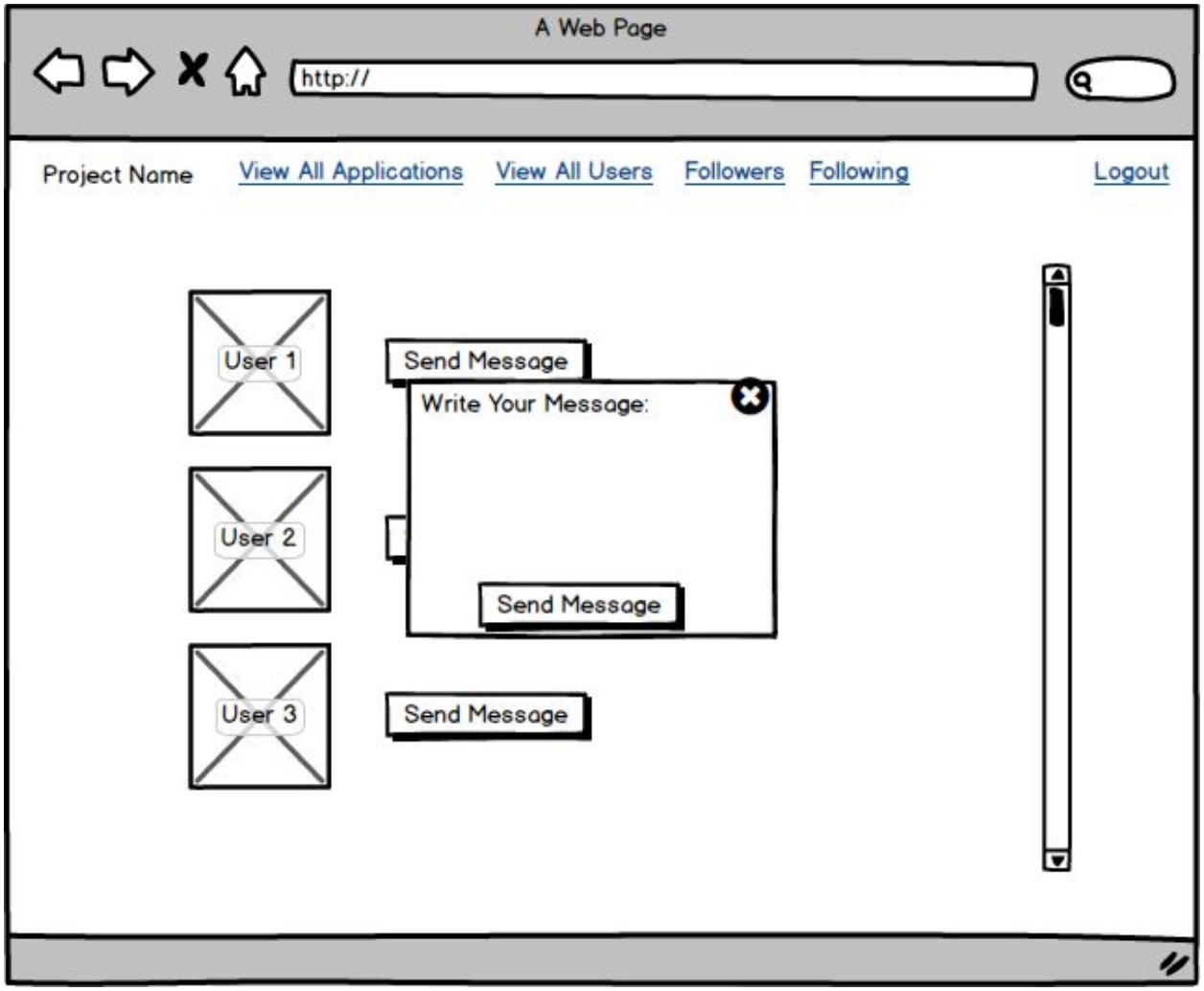
Input: @account_id

Process: An individual user can see all other users by clicking on view all users. @acc-id is necessary to eliminate the user itself from the user list.

Statement:

```
select  *
from User U
Where U.account_id <> @account_id
```

5.5 User Send Message



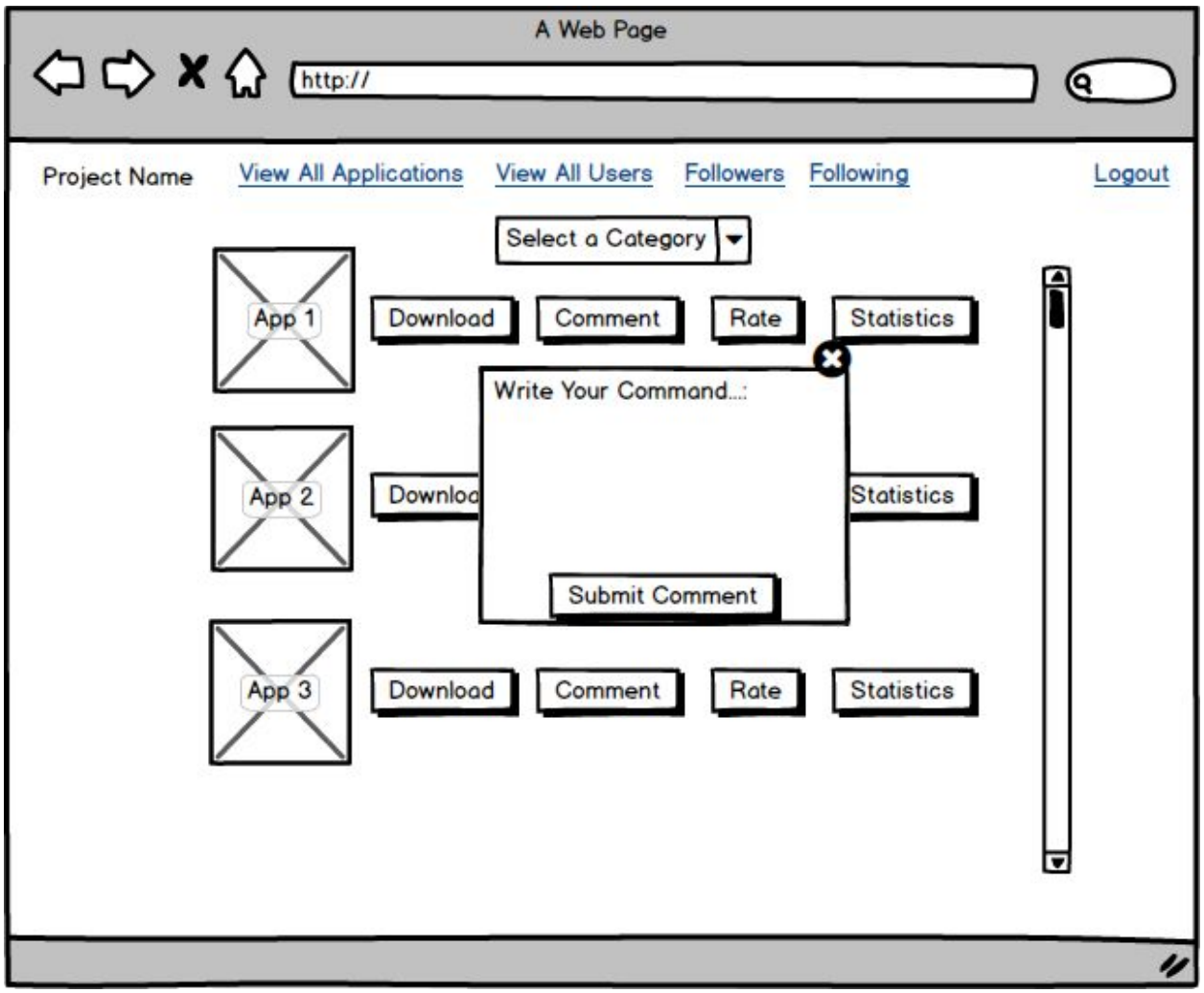
Input: @account_id, @receiver_id, @date, @message

Process: A user can send a message to other users if that users follows the other user which he wants to send the message to. Sender id is the @acc-id, and the user that will receive the message is represented by the @receiver-id.

Statement:

insert into Message values(@account_id, @receiver_id, @date, @message);

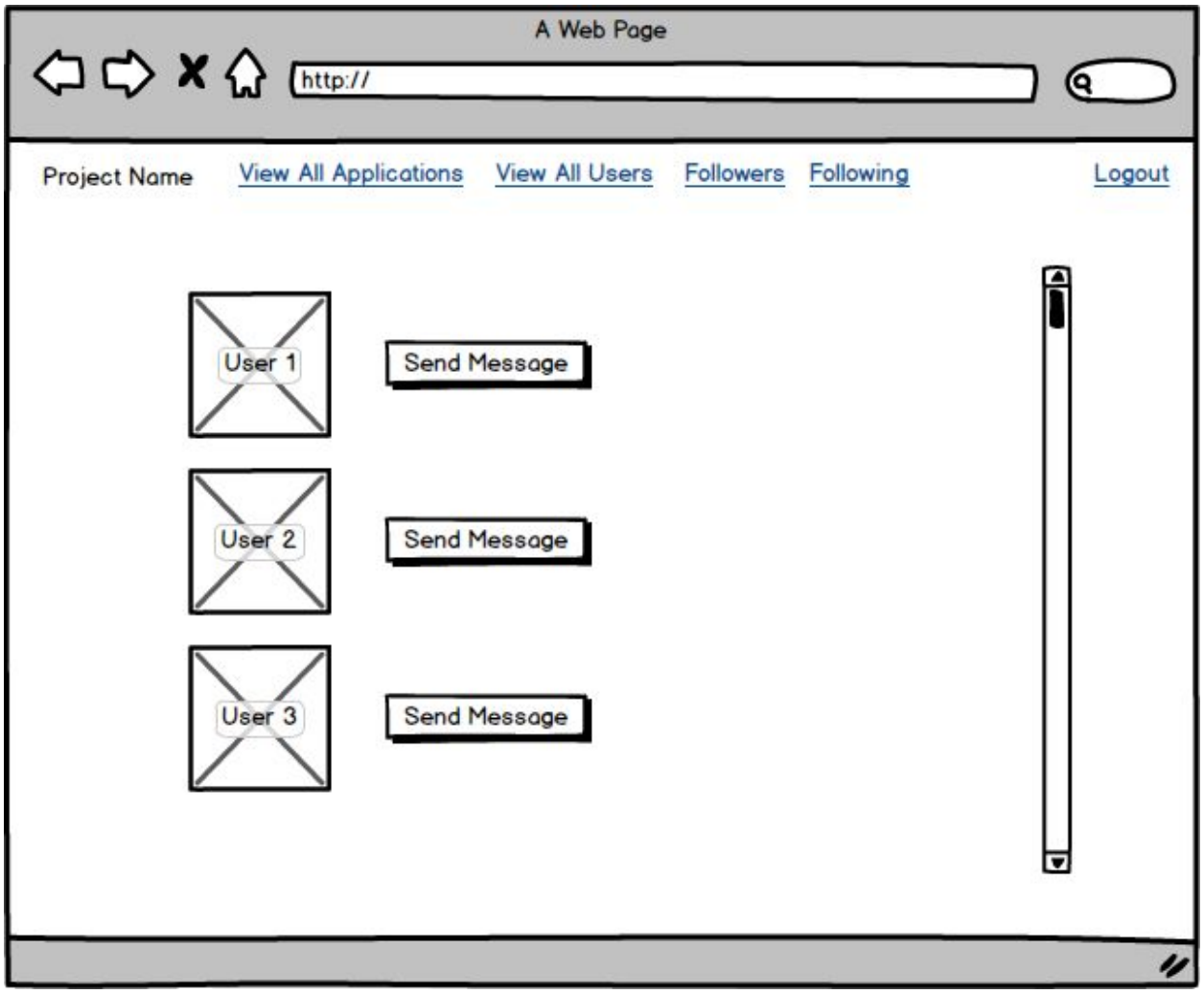
5.6 User Comment



Input: @account_id,@app_name, @comment_id, @comment, @date
Process: User can comment on an application if he downloaded the app.
Statement:

```
Select case when exists(
        Select *
        From User U, Application A
        Where U.account_id = A.account_id
) then cast (Insert into Comment values(@account_id,@app_name, @comment_id,
@comment, @date));
```

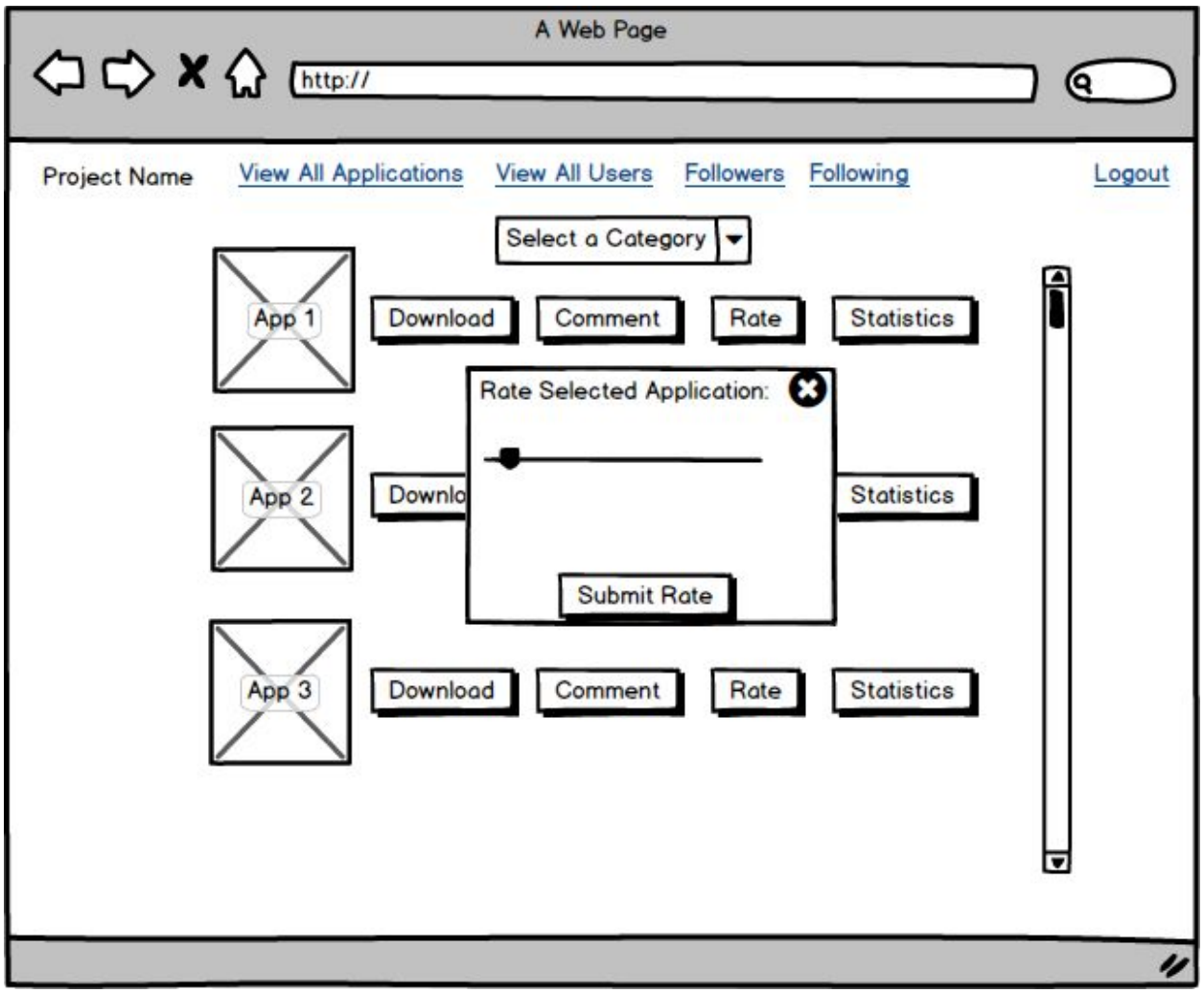
5.7 User Following



Input: @account_id
Process: User can see the list of other users which he is following. Others users are listed by their unique usernames.
Statement:

```
select U.account_id
from User U, Follows F
where F.account_id = account_id
```

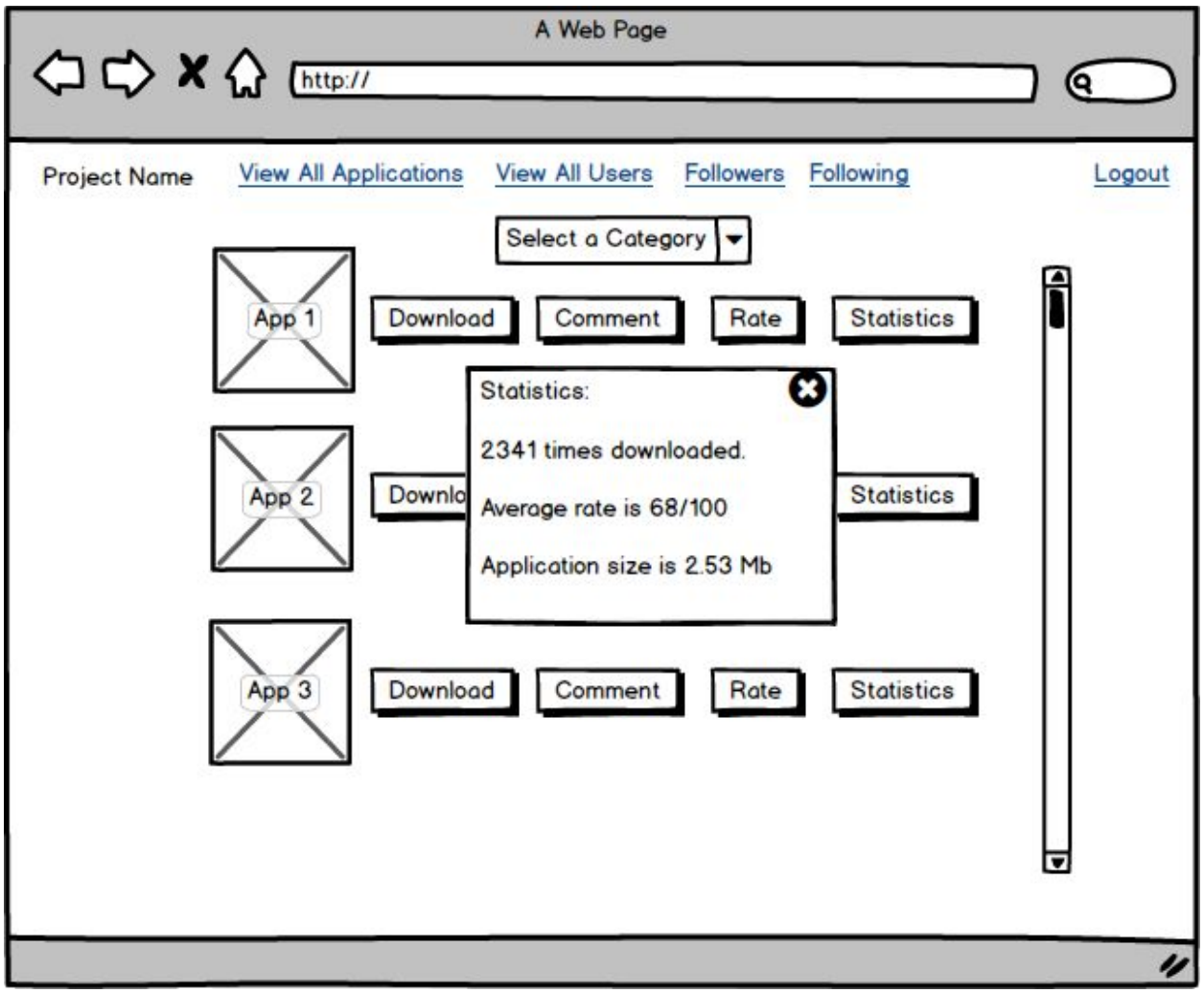
5.8 User Rate



Input: @account_id,@app_name, @rate_id, @rate, @date
Process: A user can rate an application if he downloaded that application beforehand.
Statement:

```
Select case when exists(
        Select *
        From User U, Application A
        Where U.account_id = A.account_id
) then cast (Insert into Comment values(@account_id,@app_name, @comment_id,
@comment, @date));
```

5.9 Users See Statistics



Input: @app_name

Process: A user can see the statistics of an application by clicking the statistics button next to that application.

Statement:

Download Count:

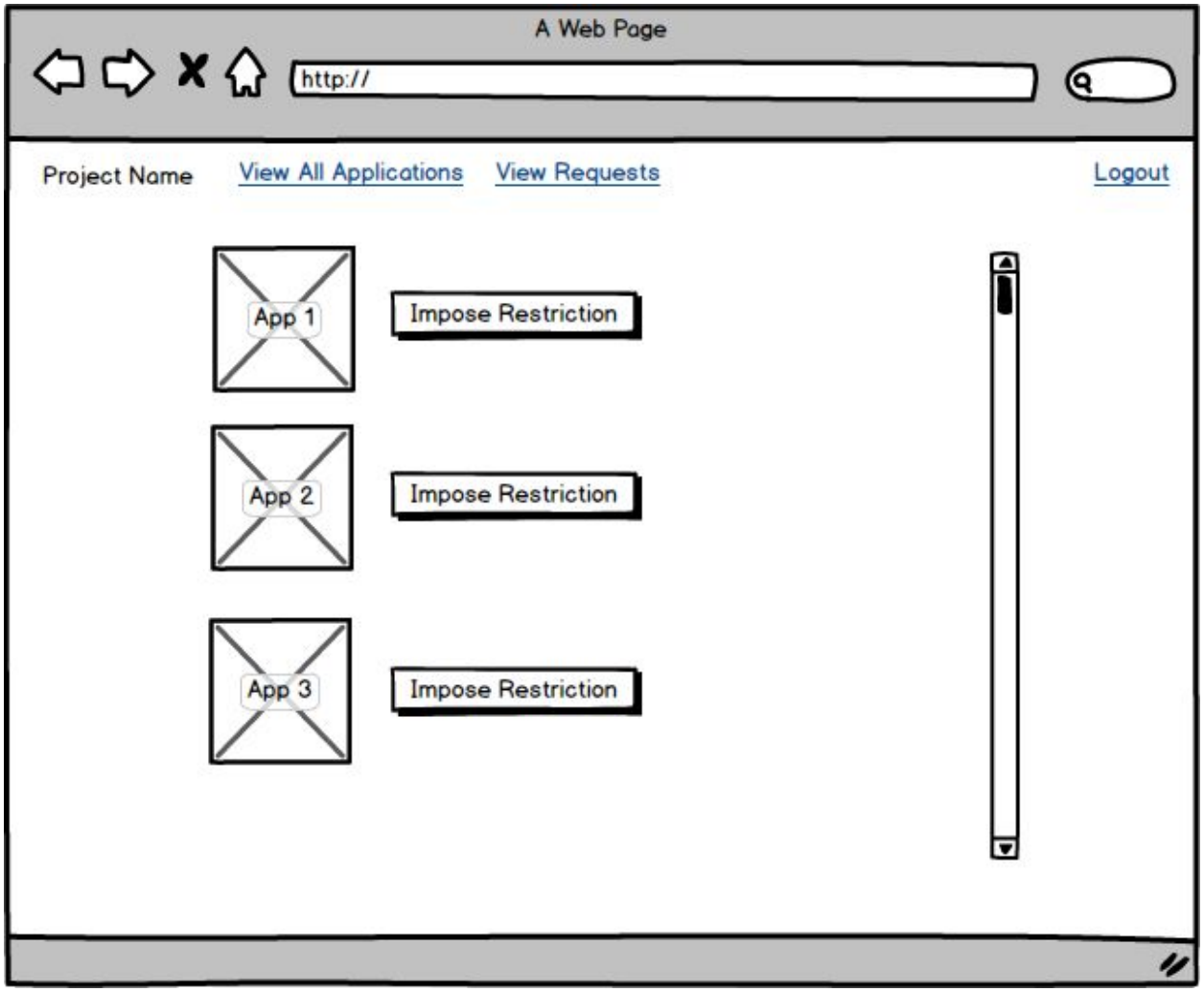
Select count(account_id) as download-count,
from Downloads D
Groupby @app_name, download-count;
Having download-count > 0 and D.app_name = @app_name

Average Rate:

select avg(rating) as rating_avg,
from Application A,Rate R
groupby rating_avg,@app_name
having A.app_name = @app_name

Size:
select size
from Application A,
where A.app_name = app_name

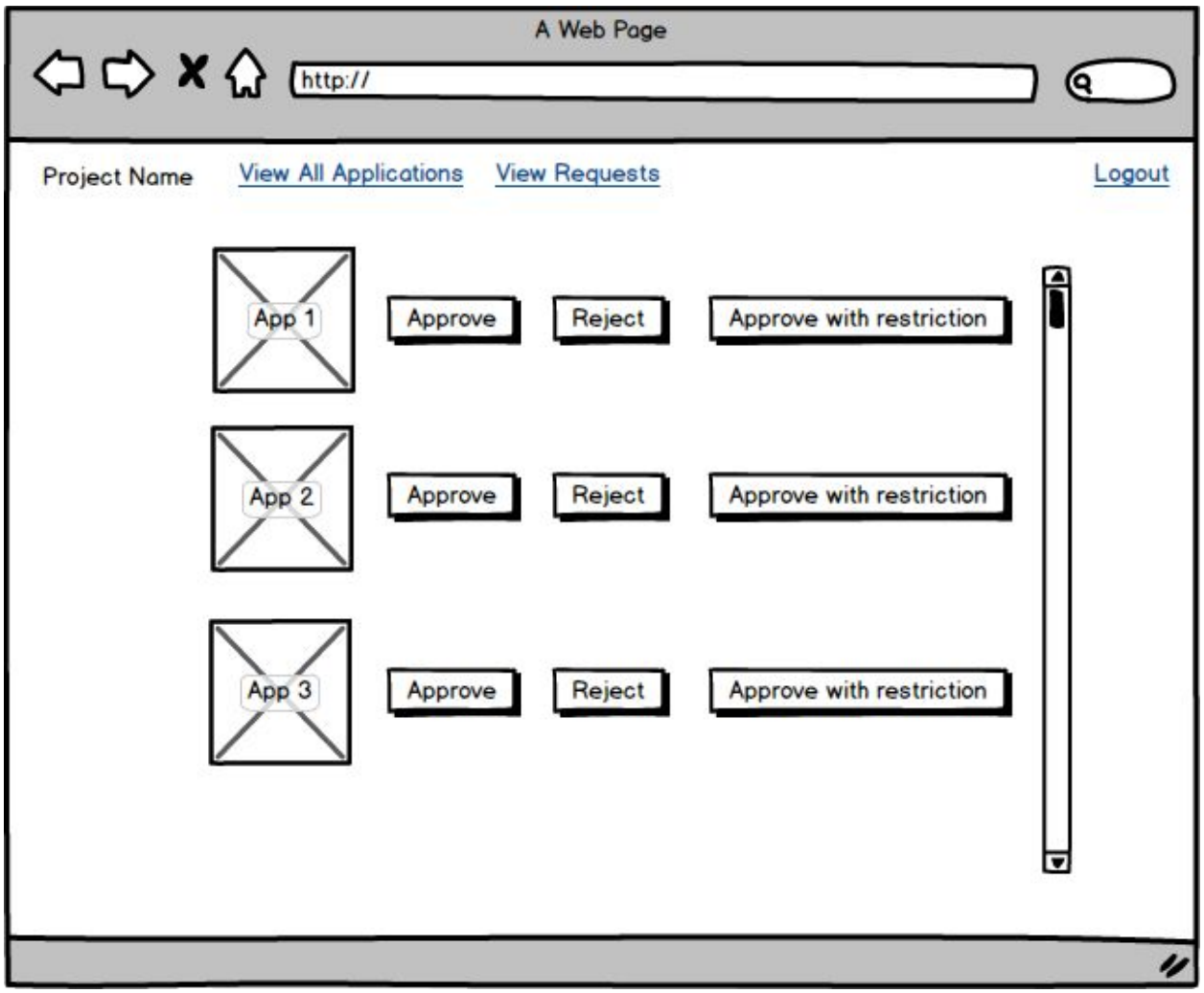
5.10 Editor View All Applications



Input: None
Process: Similar to the users, an editor can also view all the applications and he can impose restrictions by clicking the button next to each app that is listed.
Statement:

select *
from Application A

5.11 Editor View Requests



Input: @request_id

Process: An editor sees the requests by clicking on the view requests button. Editors can either approve those requests, reject them or approve them with some restrictions by clicking on the buttons next to those requests accordingly.

Statement:

```
Select *
From Requests
```

Approve:

```
Update Requests
set request_status = “approved”
Where request_id = @request_id
```

Reject:

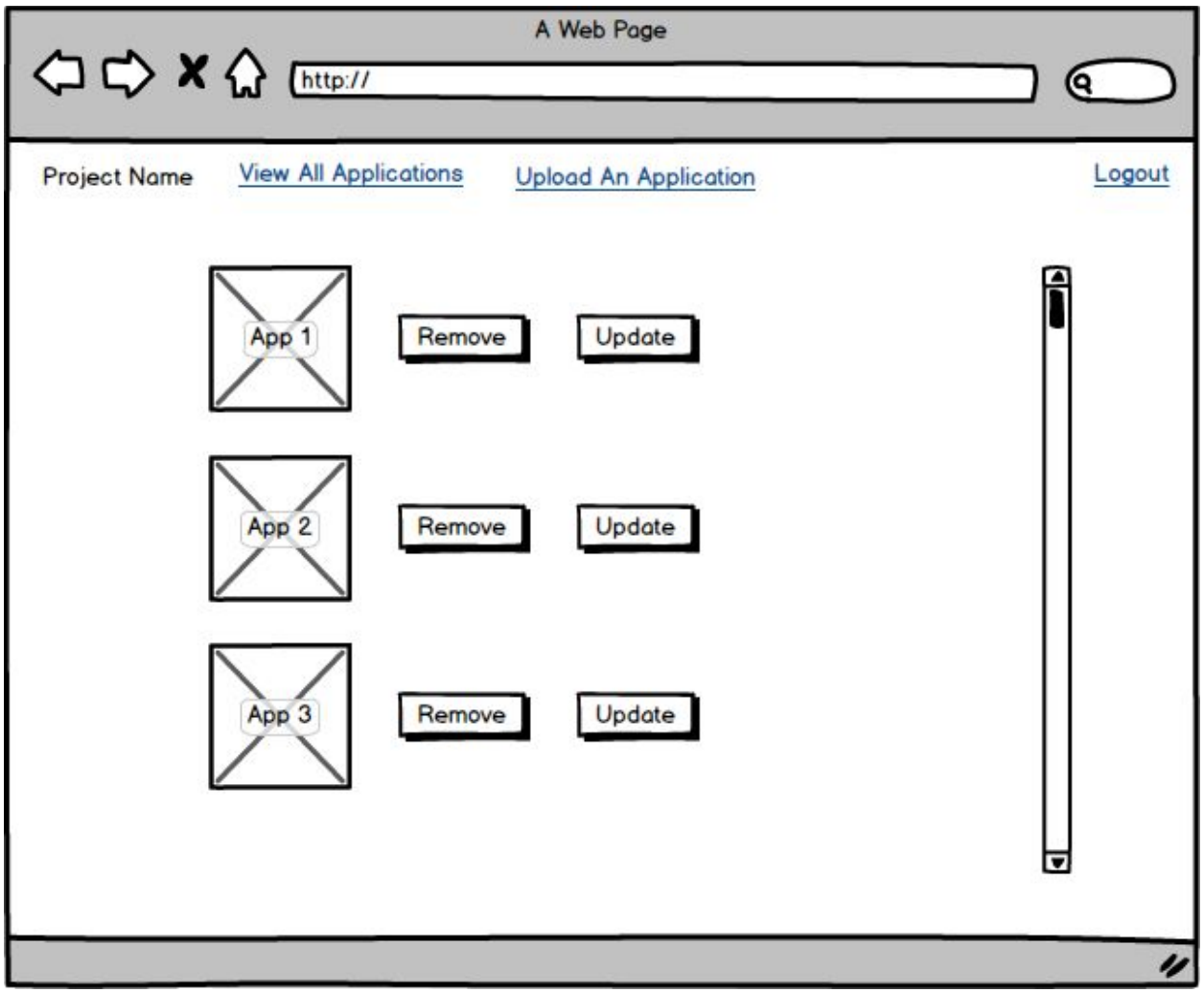
```
Update Requests
set request_status = “rejected”
```

Where request_id = @request_id

Approve with restrictions:

Update Requests
set request_status = “approved with restrictions”
Where request_id = @request_id

5.12 Developer Views His/Her Own Applications



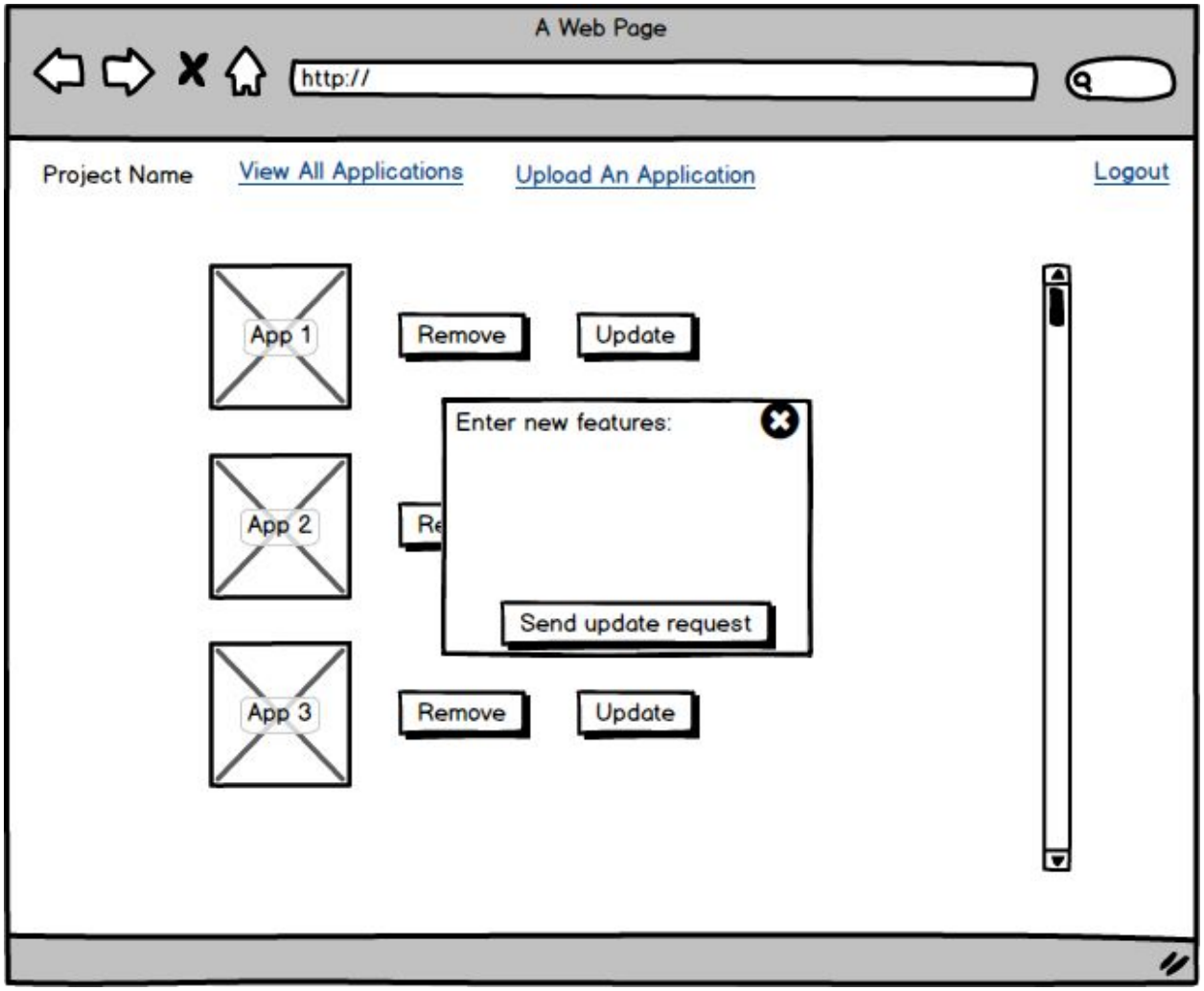
Input: @account_id

Process: A developer can also view all applications and can remove or update those applications that he developed. Removing is trivial because all the developer has to do is to click on the remove button.

Statement:

```
select *
from Application A
where A.account_id = @account_id
```

5.13 Developer Update Application



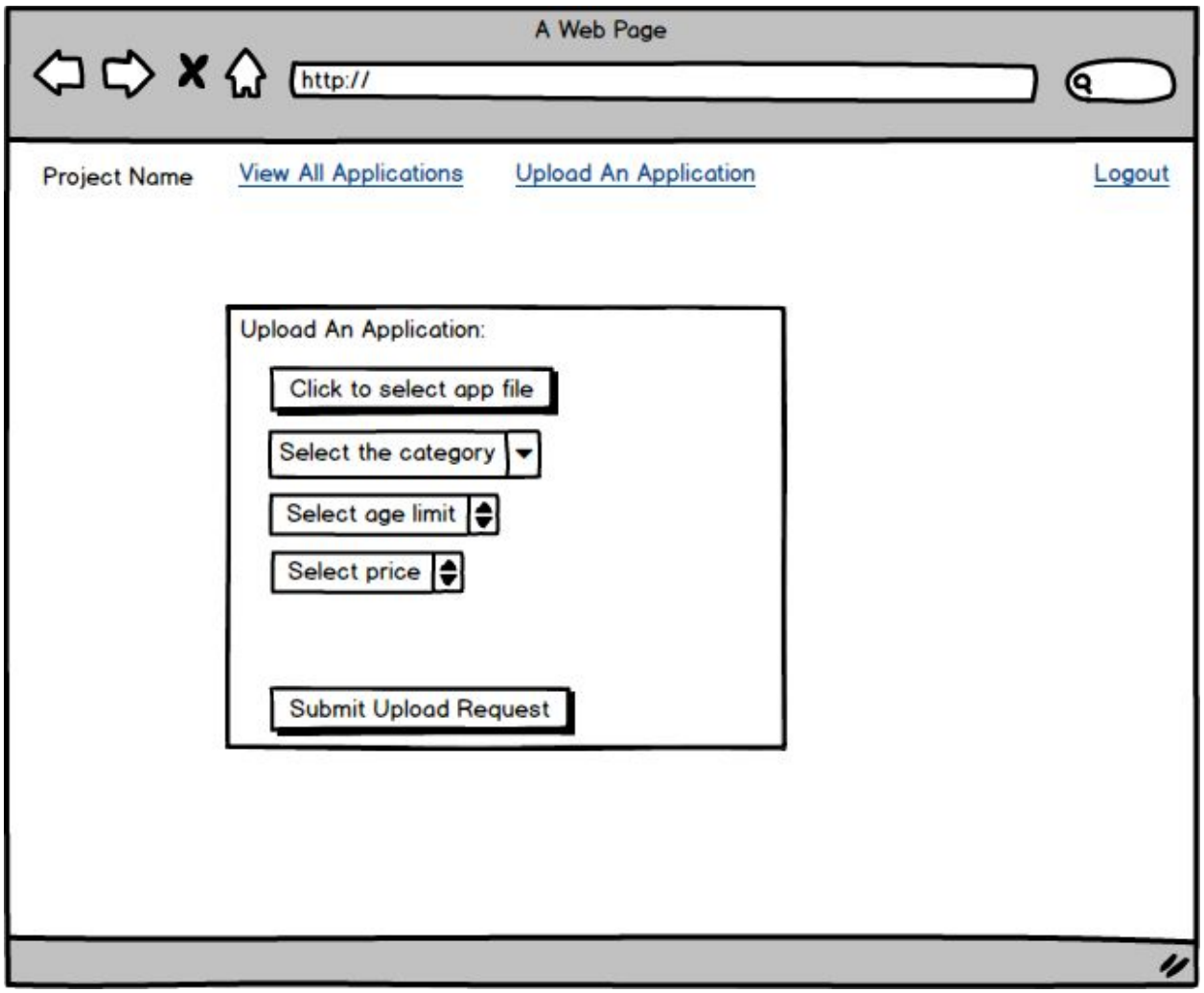
Input: @request_description

Process: Developers can update their applications the by clicking on the update button and applying the update then sending an update request to an editor.

Statement:

```
insert into Requests values(@request_description, NULL);
```

5.14 Developer Upload an Application



Input: @category, @age, @payment_amount

Process: Developer can upload an application by selecting the app file, specifying category, age limit and price of the applications. After all these are established, developer sends a submit upload request to an editor.

Statement:

Insert into Application values(@app_name, @release_date, @age, @size, @os_version, @application_status, @category, @payment_amount);

6. Advanced Database Components

6.1. Views

6.1.1 Show Downloadable Applications for this Device

```
create view [Show Applications] as
        select *
        from Applications A,Device D
        where D.os_version== A.app_requirements
```

6.1.2 Show filtered applications by age

```
create view [FilteredApplications] as
        select *
        from Applications A,User U
        where U.age>= A.age_restriction;
```

6.1.3 Show requests

```
create view [Requests] as
        select *
        from Requests
```

6.2. Reports

6.2.1. Most Downloaded Application in last month

```
        Select max ( downloaders) as most_download
        From (select count( account_id) as downloaders
                from Download D
                groupby app_name,downloaders
                having @date - 30 >= D.download_date)
        groupby app_name,most_download
```

6.2.2. Most earned application in last month

```
select max(total_fee) as most_earned
from (select total_fee as count( account_id) * F.amount
        from Download D, Fee F
        groupby app_name, total_fee
        having @date - 30 >= D.download_date),
groupby app_name, most_earned
```

6.3. Triggers

Almost all of the dynamic parts of our database are not held in the database but are calculated as required. Hence, there is not a need for many triggers.

- When an editor approves a request, the application's name will be added to the list of allowed applications.
- When a developer is deleted all of its applications and requests are deleted.
- When an editor rejects a request, application will not be shown in the list and developer receive a notification.
- When a user pays a fee for an application, developers and editors will be notificated.
- When a publish request accepted by user, all members of the database system will be notified.

6.4. Constraints

- When a user filters the apps,user should not see any apps except filtered apps.
- Only editors can see all requests.
- Developers should see only their requests.
- User should not see editor accounts, developers and editors can view.
- Every registered person in the database system, should login to do actions on system.
- Sizes of messages and pictures will be limited for storage efficiency.

6.5. Stored Procedures

In our project we will very often retrieve dynamically calculated data from the database. Thus, we will use stored procedures in order to speed up the process of development. Following are the places we will use stored procedures.

- Showing the rate of an application.
- Showing the comment of an application.
- Showing the followers of a user.
- Showing the users followed by a user.
- Showing the applications by categories.
- Showing the statistics for an application.

7. Implementation Plan

For implementing a database system in our project, MySQL will be used. Also, in order to maintain user interface and fundamental system functionalities in our project, we are going to use HTML, JavaScript, CSS and Python's Flask Library.

8. Website

Main webpage of the project is : https://github.com/gorkemyllmaz/CS-353-Fall-2019