Bilkent University

CS 353 Design Report

Prepared by CS353 - Section 1 - Group 15

Hüseyin Eren Çalık – 21402338

Ümitcan Hasbioğlu – 21402314

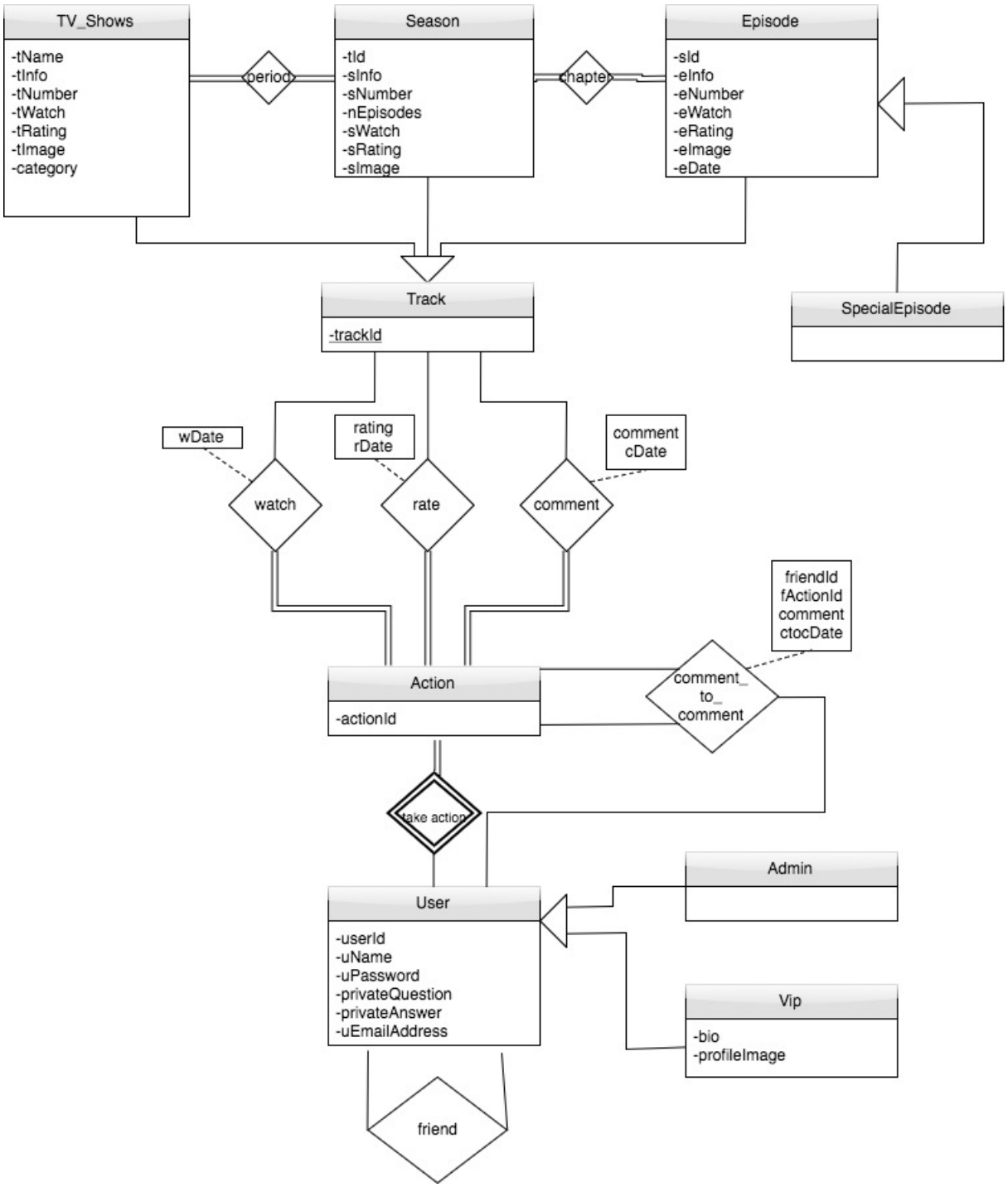
Kıvanç Gümüş – 21401767

Table Of Contents

1. Revised E/R Model

According to assistant's review, we have revised our E/R model considering the feedbacks which are the following:

1. We added special episodes which is "is-a" form of the episode. Also, we added admin and vip which are "is-a" form of the user.
2. In order to reduce the redundancy of the tv shows', seasons' and episodes' ids, we added track table which includes the unique ids of tv shows, seasons and episodes. This brought flexibility when adding new action which is related to them. In the old version we had to keep all three ids, now instead of three we keep only id which is track_id.
3. All three actions (watch, rate , comment) have become relations with the unique action ids.
4. With help of the track table, we reduce the number of relations which are related with actions. For example, we used to have three relations for each tv show, season and episode. Now, we only three relations instead of nine relations which includes watch,rate and comment with the help of action table. Also, this bring the advantage of removing extra tables such as watch, rate and comment.
5. According to assistant, actions should be weak entity since actions should be made by a user. Therefore, we added a weak relationship between user and action.
6. We changed action entity's participation in watch,rate and comment relationships into total participation.
7. To accomplish comment to a comment, we created a relationship between action and user.
8. We added category to tv shows for distinguishing the content of the shows.
9. Date is to the system for each action and medias to ordering the latest news by date.


TV_Shows
-tName
-tInfo
-tNumber
-tWatch
-tRating
-tImage
-category
period
Season
-tId
-sInfo
-sNumber
-nEpisodes
-sWatch
-sRating
-sImage
chapter
Episode
-sId
-eInfo
-eNumber
-eWatch
-eRating
-eImage
-eDate
Track
-trackId
SpecialEpisode
wDate
rating
rDate
comment
cDate
watch
rate
comment
friendId
fActionId
comment
ctocDate
Action
-actionId
comment_
to_
comment
take action
Admin
User
-userId
-uName
-uPassword
-privateQuestion
-privateAnswer
-uEmailAddress
Vip
-bio
-profileImage
friend

2. Relational Schemas

2.1. TV SHOWS

Relational Model:

Tv_Shows(trackId, tName, tInfo, tWatch, tRating, tImage, category)

Functional Dependencies:

trackId -> tName tInfo tWatch tRating tImage category

Candidate Keys:

{(trackId)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE tvshows(
        trackId     int PRIMARY KEY AUTO_INCREMENT,
        tName       varchar(100) NOT NULL,
        tInfo       varchar(500),
        tWatch      int NOT NULL,
        tRating     float NOT NULL,
        tImage      varchar(500),
        category    varchar(20);
        FOREIGN KEY(trackId) references track) ENGINE=InnoDB;
```

2.2. SEASON

Relational Model:

Season(trackId, tId, sInfo, sNumber, nEpisodes, sWatch, sRating, sImage)

Functional Dependencies:

trackId -> tId sInfo sNumber nEpisodes sWatch sRating sImage

Candidate Keys:

{(trackId}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE season(
        trackId     int PRIMARY KEY AUTO_INCREMENT,
        tId         int NOT NULL,
        sInfo       varchar(500),
        sNumber     int NOT NULL,
        nEpisodes   int NOT NULL,
        sWatch      int NOT NULL,
        sRating     float NOT NULL,
        sImage      varchar(500),
        FOREIGN KEY(trackId) references track,
        FOREIGN KEY(tId) references tv_shows(trackId)) ENGINE=InnoDB;
```

2.3. EPISODE

Relational Model:

Episode(trackId, sId ,eNumber,eInfo,eImage,eWatch,eRating,eDate)

Functional Dependencies:

trackId-> sId eNumber eInfo eImage eWatch eRating eDate

Candidate Keys:

{(trackId}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE episode(
        trackId     int PRIMARY KEY AUTO_INCREMENT,
        sId         int NOT NULL,
        eNumber     int NOT NULL,
        eInfo       varchar(500),
        eImage      varchar(500),
        eWatch      int NOT NULL,
        eRating     int NOT NULL,
        eDate       date,
        FOREIGN KEY(trackId) references track,
        FOREIGN KEY(sId) references season(trackId) ENGINE=InnoDB;
```

2.4. ACTION

Relational Model:

Action(<u>actionId</u>, <u>userId</u>)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(actionId, userId)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE action(
        actionId            int NOT NULL AUTO_INCREMENT,
        userId              int NOT NULL,
        PRIMARY KEY(actionId, userId),
        FOREIGN KEY(userId) references user) ENGINE=InnoDB;
```

2.5. USER

Relational Model:

User(<u>userId</u>, uName, uPassword, privateQuestion, privateAnswer, uEmailAddress)

Functional Dependencies:

userId -> uName uPassword uEmailAddress

uEmailAddress -> userId

Candidate Keys:

{(userId, uEmailAddress)}

Normal Form:

3NF

Table Definition

```
CREATE TABLE user(
        userId              int PRIMARY KEY AUTO_INCREMENT,
        uName               varchar(80) NOT NULL,
        uPassword           varchar(40) NOT NULL,
        privateQuestion     varchar(40) NOT NULL,
        privateAnswer       varchar(40) NOT NULL,
        uEmailAddress       varchar(254) NOT NULL) ENGINE=InnoDB;
```

2.6. SPECIAL EPISODE

Relational Model:

Special_Episode(trackId)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(trackId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE specialepisode(
        trackId                 int PRIMARY KEY AUTO_INCREMENT
        FOREIGN KEY(trackId) references episode) ENGINE=InnoDB;
```

2.7. ADMIN

Relational Model:

Admin(<u>userId</u>)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE admin(
        userId          int PRIMARY KEY AUTO_INCREMENT
        FOREIGN KEY(userId) references user) ENGINE=InnoDB;
```

2.8. VIP

Relational Model:

Vip(userId)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE vip(
        userId          int PRIMARY KEY AUTO_INCREMENT
        FOREIGN KEY(userId) references user) ENGINE=InnoDB;
```

2.9. TRACK

Relational Model:

Track(trackId)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(trackId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE track(
        trackId         int PRIMARY KEY AUTO_INCREMENT)  ENGINE=InnoDB;
```

2.10. WATCH

Relational Model:

watch(<u>actionId</u>, <u>userId</u>, trackId, wDate)

Functional Dependencies:

actionId, userId-> trackId wDate

Candidate Keys:

{(actionId, userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE watch(
        actionId        int AUTO_INCREMENT,
        userId          int NOT NULL,
        trackId         int NOT NULL,
        wDate           date NOT NULL,
        PRIMARY KEY(actionId, userId),
        FOREIGN KEY(actionId) references action,
        FOREIGN KEY(userId) references user,
        FOREIGN KEY(trackId) references track) ENGINE=InnoDB;
```

2.11. RATE

Relational Model:

rate(<u>actionId</u>, <u>userId</u>, trackId, rating, rDate)

Functional Dependencies:

actionId, userId -> trackId rating rDate

Candidate Keys:

{(actionId, userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE rate(
        actionId        int AUTO_INCREMENT,
        trackId         int NOT NULL,
        userId          int NOT NULL,
        rating          int NOT NULL,
        rDate           date NOT NULL,
        PRIMARY KEY(actionId, userId),
        FOREIGN KEY(actionId) references action,
        FOREIGN KEY(userId) references user,
        FOREIGN KEY(trackId) references track) ENGINE=InnoDB;
```

2.12. COMMENT

Relational Model:

comment(<u>actionId</u>, <u>userId</u>, trackId, comment, cDate)

Functional Dependencies:

actionId -> trackId userId comment c_date

Candidate Keys:

{(actionId, userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE comment(
        actionId        int PRIMARY KEY AUTO_INCREMENT,
        trackId         int NOT NULL,
        userId          int NOT NULL,
        comment         varchar(280) NOT NULL,
        c_date          date NOT NULL,
        PRIMARY KEY (actionId, userId),
        FOREIGN KEY(actionId) references action,
        FOREIGN KEY(userId) references user,
        FOREIGN KEY(trackId) references track) ENGINE=InnoDB;
```

2.13. COMMENTTOCOMMENT

Relational Model:

commentToComment(actionId, userId, friendId, fActionId, comment, ctocDate)

Functional Dependencies:

actionId, userId -> friendId fActionId comment

Candidate Keys:

{(actionId, userId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE commenttocomment(
        actionId        int AUTO_INCREMENT,
        userId          int NOT NULL,
        friendId        int NOT NULL,
        fActionId       int NOT NULL,
        comment         varchar(280) NOT NULL,
        PRIMARY KEY (actionId, userId),
        FOREIGN KEY(actionId) references action,
        FOREIGN KEY(userId) references user,
        FOREIGN KEY(friendId) references friend(userId),
        FOREIGN KEY(fActionId) references action(actionId) ) ENGINE=InnoDB;
```

2.14. FRIEND

Relational Model:

Friend(<u>userId</u> ,<u>friendId</u>)

Functional Dependencies:

No dependencies.

Candidate Keys:

{(userId, friendId)}

Normal Form:

BCNF

Table Definition

```
CREATE TABLE friend(
        userId          int NOT NULL,
        friendId        int NOT NULL,
        PRIMARY KEY(userId, friendId),
        FOREIGN KEY(userId) references user
        FOREIGN KEY(friendId) references user(userId)) ENGINE=InnoDB;
```

3. Functional Dependencies and Normalization of Table

All functional dependencies and normal forms are indicated in Relation Schemas in Section 2 of this Project Design Report. We checked whether majority of relations in our design are in Boyce-Code Normal Form and only one relation is in 3NF. We concluded that no decomposition is required.

4. Functional Components

4.1. Uses Cases and Scenarios

In Trip Teevee, there are three types of users which are basic users, VIPs and administrators. Basic users and VIPs have similarities and minor differences. Administrators are the users who are authorized to modify/delete/add data to the system. In order to use the system, all types of users except admins are supposed to register and login to the system. Admins accounts are inserted into the system by backend developer.

Basic User:

- Basic user can login to the system with his/her e-mail address and password.
- Basic user can access his/her activity history which includes watched marks, rates, comments and his/her friends' comments to his/her actions.
- Basic user can change his/her password before logging in on the guest page by answering the private question he/she provided while registration.
- Basic user can see TV Shows, seasons and episodes which include image, name, information, comments, rate and number of marks.
- Basic user can search for TV Shows.
- Basic user can view his/her friends' recent actions which are commenting, marking as watched and rating. Also, he/she can comment to the actions.
- Basic user can view VIP purchase page.
- Basic user can comment on Tv Shows, seasons and episodes individually.
- Basic user can mark Tv Shows, seasons and episodes as watched.
- Basic user can rate Tv Shows, seasons and episodes.
- Basic user can view top 5 most watched TV Shows.
- Basic user can see top 5 users with most actions.

VIP User:

- VIP user is a type of basic user but additionally he/she has extra features.
- VIP user can view his/her profile.
- VIP user can additionally search for seasons and episodes.
- VIP user can add biography into his/her profile.
- VIP user can add image into his/her profile.
- VIP user can view calendar.

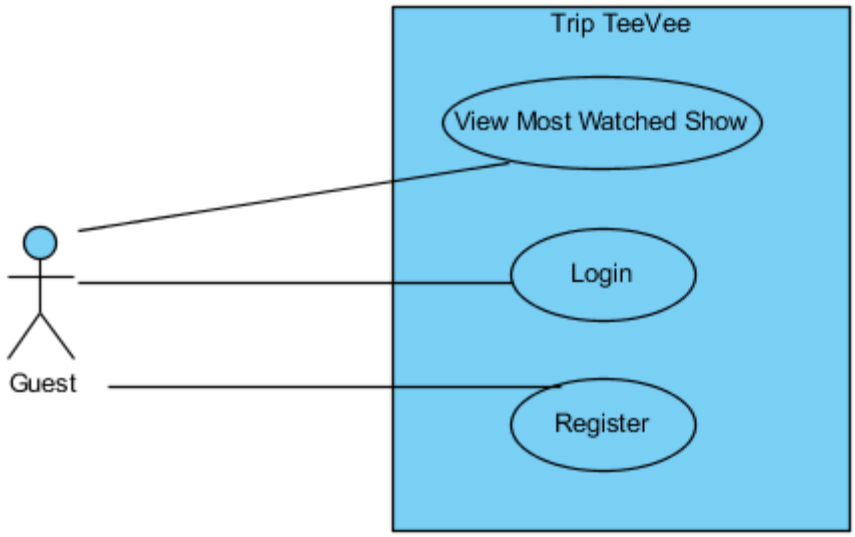

Trip TeeVee
Login
Change Password
View Tv Shows/Seasons/Episodes
View Friend Actions
View VIP Purchase Page
Comment on Tv Shows/Seasons/Episodes
Comment on Friend Actions
Mark TV Shows as Watched
Rate TV Shows
View Top 5 most watched TV Shows
View Top 5 Users with most Actions
View His/Her Action History
Search for TV Show
Add Bio
Add Profile Image
Search for Seasons and Episodes
View Calendar
Basic User
VIP

Administrator:

- Admin can view TV Shows, seasons and episodes including their all entities.
- Admin can view users with their all entities.
- Admin can view comments with their all related information.
- Admin can modify, delete and add TV Shows, seasons and episodes.
- Admin can delete users.
- Admin can delete and modify comments.

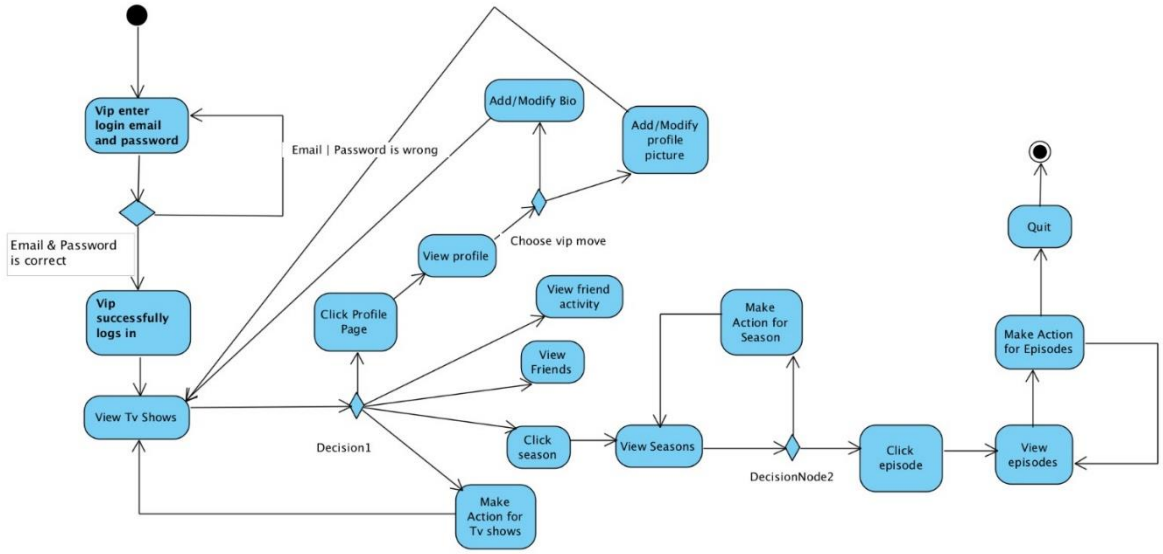
Guest:
Guests are the users who has not logged in.

- Guest can login.
- Guest can register.
- Guest can view most watched TV Show.



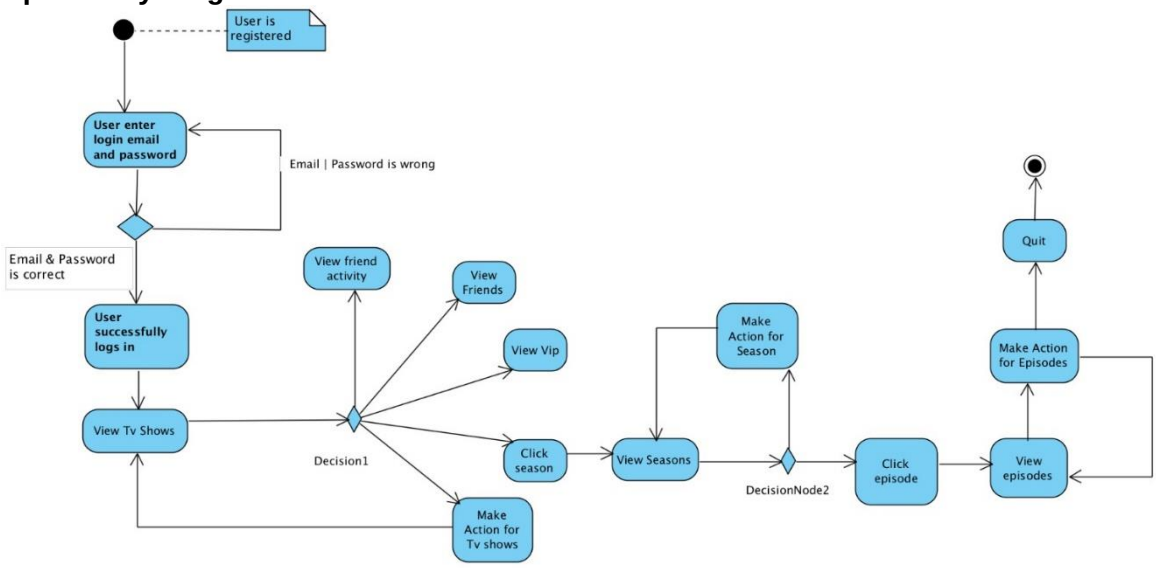
4.2. Activity Diagrams

User Activity Diagram



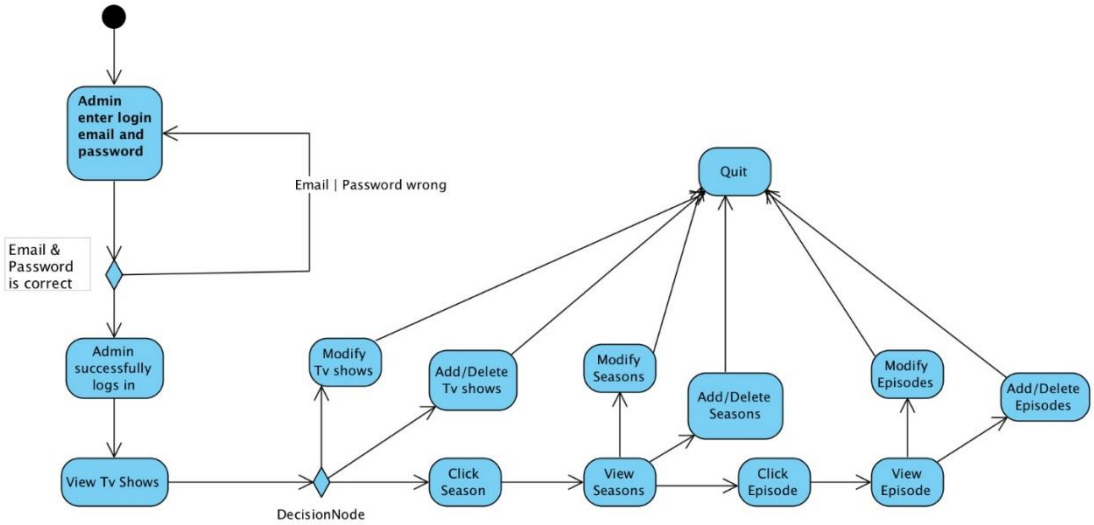
Someone should register the system to see medias otherwise he/she can only see the most watched tv show. Assuming a user was registered and is trying to logging in to system. After filling mail address and password, the system checks whether it is valid or not. If it is not valid, user has to try again. If it is valid, user successfully logs into the system and sees the tv show page. In that page, user can see his friends and their activities, view vip page, take an action or go to season page for seasons. Making an action means that commenting,marking as watched or rating. If the user clicks the season, than he can see the season page. In this page, user can take an action or go to episode page. If the user goes to episode page, he can take action and quit.

Vip Activity Diagram



Assuming a user was vip and is trying to logging in to system. After filling mail address and password, the system checks whether it is valid or not. If it is not valid, user has to try again. If it is valid, vip successfully logs into the system and sees the tv show page. In that page, vip can see his friends and their activities, view his/her profile,add biography and profile picture or take an action or go to season page for seasons. Making an action means that commenting,marking as watched or rating. If the vip clicks the season, than he can see the season page. In this page, vip can take an action or go to episode page. If the vip goes to episode page, he can take action and quit.

Admin



Assuming an admin is trying to logging in to system. After filling mail address and password, the system checks whether it is valid or not. If it is not valid, the admin has to try again. If it is valid, vip successfully logs into the system and sees the tv show page. In this page, the admin can add or delete tv show and modify them or go to season page. If he goes to season page, he can add or delete season and modify them or go to episode page. If he goes to episode page, he can add or delete episode and modify them or go to episode page. Besides, he can quit from the system anytime he wants.

4.3. Algorithms

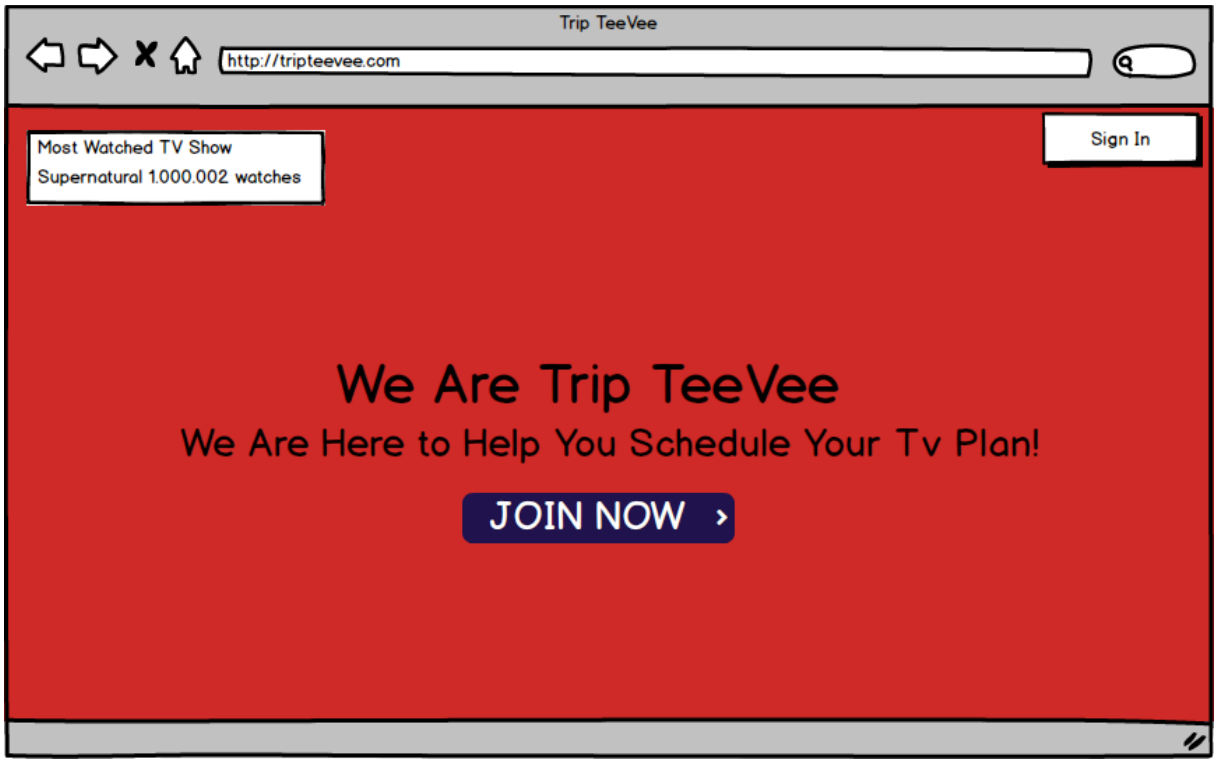
Ordering comments to a tv show, a season or an episode is going to be crucial since some comments are more reliable and correct than the others. To accomplish that situation, we develop algorithm. It takes two parameters which are rating and mark as watched. If the user is marked an episode as watch and not rate the episode the possibility of less reliable comment(vice versa). If the user, mark the episode as watch and rate it, the comment becomes the most reliable one and it appears than the others' comment who does not mark as watched and rated.

4.4. Data Structures

For the attribute domains we use Numeric, Date and String data types of MySQL will be used.

5. User Interface Design and Corresponding SQL Statements

5.1. Main Page



Process: The homepage of the Trip TeeVee is displayed above. Non-registered users see the most watched TV Show and the watch number of that particular show. Also, they can login or register.

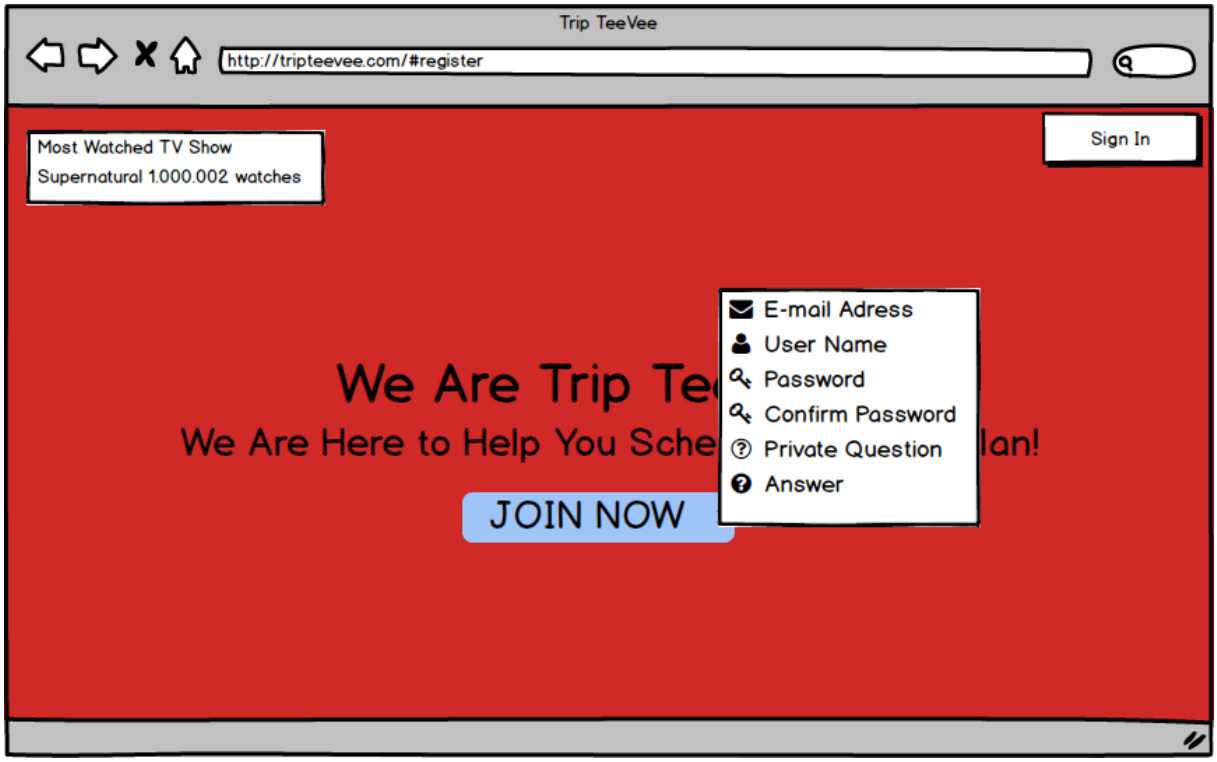
SQL Statements:

Displaying the Most Watched Show:

```
SELECT tName, MAX(watchCount)

FROM tvshows
```

5.2. Registering



Process: After clicking on Join Now button, non-registered users can create an account by filling out the necessary information.

Inputs: @email, @userName, @password, @confirmPassword, @privateQuestion, @privateAnswer

SQL Statements:

If @password=@confirmPassword:

INSERT INTO user(uName, uPassword, privateQuestion, privateAnswer, uEmailAddress)

VALUES(@userName, @password, @privateQuestion, @privateAnswer)

5.3. Logging In

Process: Registered users can login providing e-mail address and password.

Inputs: @email, @password

SQL Statements:

SELECT userID

FROM user

WHERE uEmailAddress = @

email AND uPassword=@password

5.4. Change Password

Process: A registered user can change their password by providing their email and answering their previously chosen private question.

Inputs: @email, @answer, @newpassword

SQL Statement:

SELECT privateAnswer

FROM user

WHERE uEmailAddress = @email

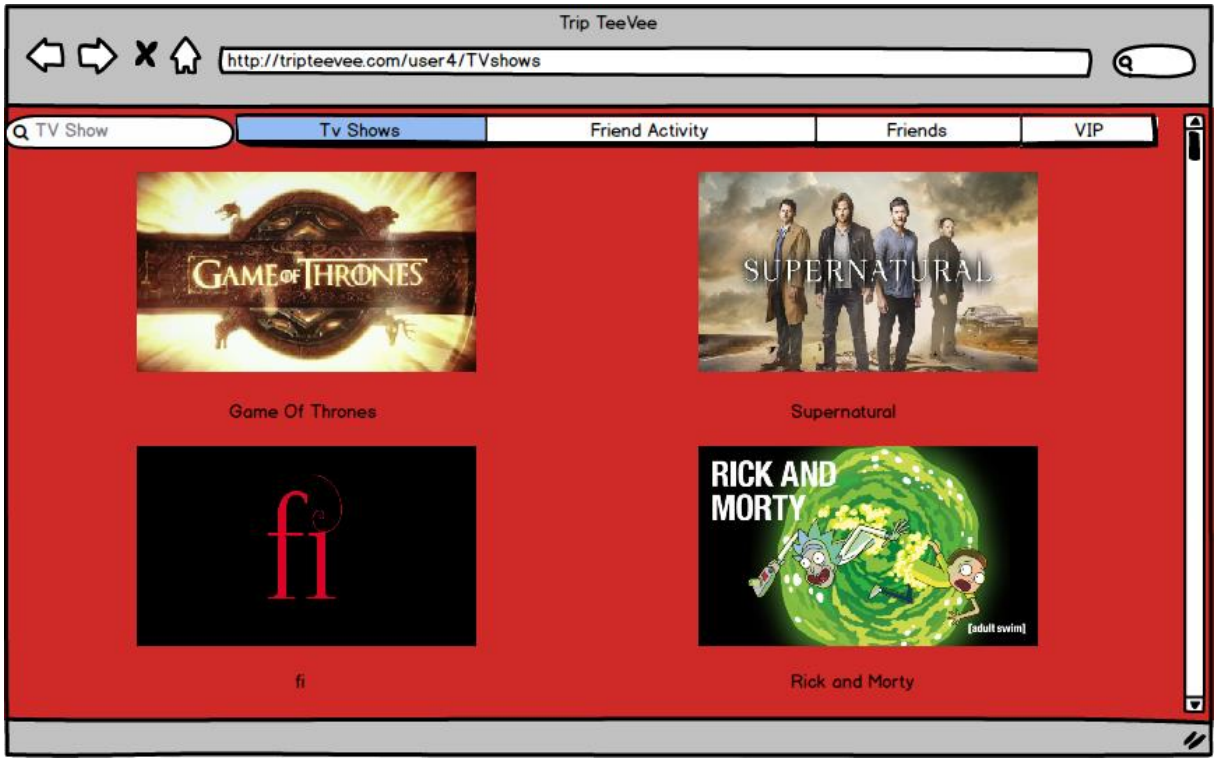
If @answer = privateAnswer

UPDATE user

SET uPassword = @newpassword

WHERE uEmailAddress = @email

5.5. TV Show Page



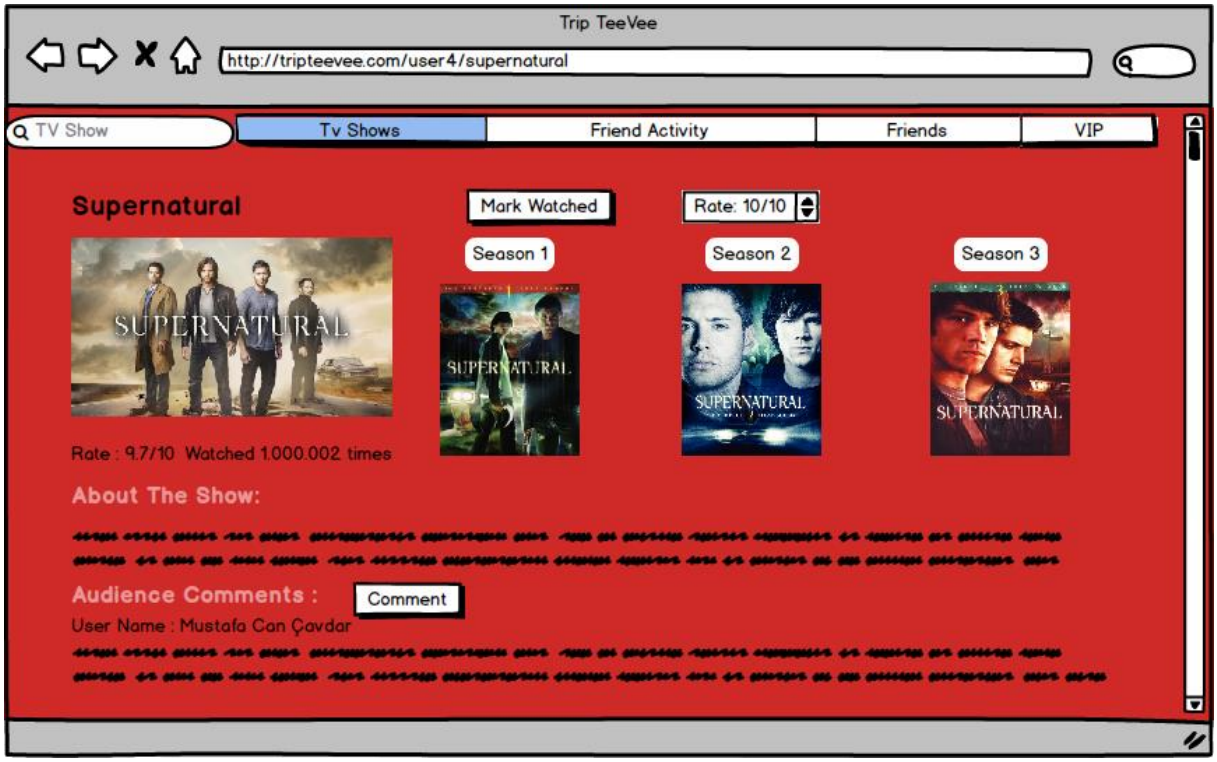
Process: A basic user can view all TV shows.

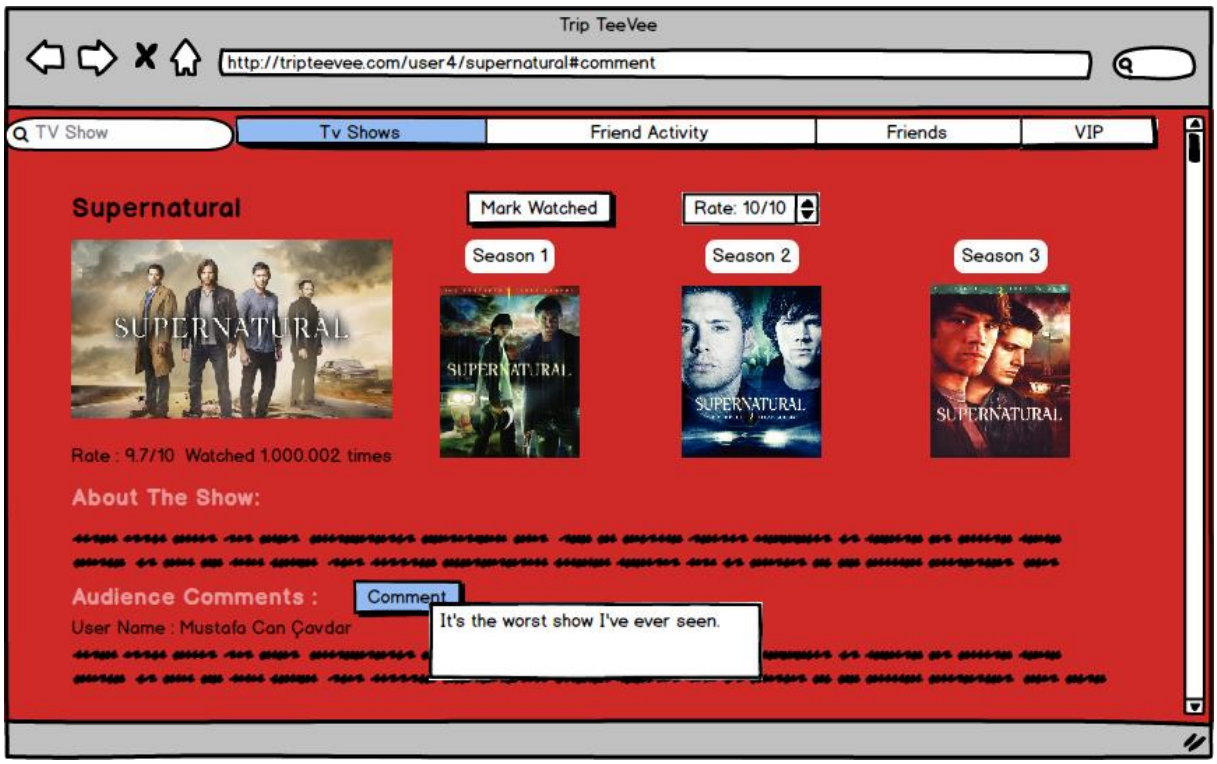
SQL Statement:

SELECT tName, tImage, trackId

FROM tvshows

5.6. Page of A TV Show





Process: A basic user can view comments, rate, watch times and seasons of a particular TV Show. They can also rate on, comment on and mark it.

Inputs: @TvShowID, @UserID, @UserComment, @UserRating, @WatchedInput, @Date

SQL Statement:

Finding Comments:

SELECT userName, comment

FROM comments NATURAL JOIN user

WHERE trackId = @TvShowID

Finding Ratings, Information, Watch Count and the corresponding Tv Show Image:

SELECT tRating, tInfo, tWatch, tImage

FROM tvshows

WHERE trackId = @TvShowID

Finding Season Numbers, Images and Season trackIds

SELECT sNumber, sImage, trackId

FROM season

WHERE tId = @TvShowID

Commenting on the Tv Show:

INSERT INTO comment(userId, trackId, comment, cDate)

VALUES(@UserID, @TvShowID, @UserComment, @Date)

After first query we update the action table

WITH actionTemp(acId,uId) AS (SELECT actionId FROM comment WHERE trackId = @TvShowID AND userId = @UserID AND cDate = @Date AND comment=@UserComment)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp

Rating the Tv Show:

INSERT INTO rate(userId, trackId, rating, rDate)

VALUES(@UserID, @TvShowID, @UserRating, @Date)

After first query we update the action table

```
WITH actionTemp(acId) AS (SELECT actionId FROM rate WHERE trackId =
@TvShowID AND userId = @UserID AND cDate = @Date AND
rating=@UserRating)

INSERT INTO action(actionId, userId)

VALUES(acId, @UserID)
```

After the action table update, we update the rating of the tv show

```
UPDATE tvshows

SET tRating=newRate

FROM (

SELECT avg(rating)

FROM rate

WHERE trackId = @TvShowID

)

WHERE trackId = @TvShowID
```

Marking the Tv Show as Watched:

```
INSERT INTO watch(userId, trackId, wDate)

VALUES(@UserID, @TvShowID, @Date)
```

After first query we update the action table

```
WITH actionTemp(acId,uId) AS (SELECT actionId FROM watch WHERE trackId =
@TvShowID AND userId = @UserID AND cDate = @Date)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp
```

After the action table update, we update the watch number of the tv show

```
UPDATE tvshows
SET tWatch=newWatch
FROM (
SELECT count(*)
FROM watch
WHERE trackId = @TvShowID
)
WHERE trackId = @TvShowID
```

5.7. Season Page of a TV Show

Process: A basic user can view comments, rate, watch times and episodes of a particular season of a TV Show. They can also rate on, comment on and mark it.

Inputs: @SeasonID, @UserID, @UserComment, @UserRating, @WatchedInput, @Date

SQL Statement:

Finding Comments:

SELECT userName, comment

FROM comments NATURAL JOIN user

WHERE trackId = @SeasonID

Finding Ratings, Information, Watch Count and the corresponding Season Image :

SELECT sRating, sInfo, sWatch, sImage

FROM season

WHERE trackId = @SeasonID

Finding Episode Numbers, Images and Episode trackIds

SELECT sNumber, sImage, trackId

FROM episode

WHERE sId = @SeasonID

Commenting on the Season:

INSERT INTO comment(userId, trackId, comment, sDate)

VALUES(@UserID, @Season, @UserComment, @Date)

After first query we update the action table

WITH actionTemp(acId,uId) AS (SELECT actionId FROM action WHERE trackId = @SeasonID AND userId = @UserID AND sDate = @Date)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp

Rating the Season:

INSERT INTO rate(userId, trackId, rating, sDate)

VALUES(@UserID, @SeasonID, @UserRating, @Date)

After first query we update the action table

WITH actionTemp(acId,uId) AS (SELECT actionId FROM rate WHERE trackId = @SeasonID AND userId = @UserID AND cDate = @Date AND rating=@UserRating)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp

After the action table update, we update the rating of the season

UPDATE season

SET sRating=newRate

FROM (

SELECT avg(rating)

```
FROM rate
WHERE trackId = @SeasonID
)
WHERE trackId = @SeasonID
```

Marking the Season as Watched:

```
INSERT INTO watch(userId, trackId, wDate)
VALUES(@UserID, @SeasonID, @Date)
```

After first query we update the action table

```
WITH actionTemp(acId,uId) AS (SELECT actionId FROM watch WHERE trackId =
@SeasonID AND userId = @UserID AND cDate = @Date)
INSERT INTO action(actionId, userId)
SELECT acId,uId
FROM actionTemp
```

After the action table update, we update the watch number of the tv show

```
UPDATE season
SET sWatch=newWatch
FROM (
SELECT count(*)
FROM watch
WHERE trackId = @SeasonID
)
WHERE trackId = @SeasonID
```

5.8. Episode Page of a TV Show

Process: A basic user can view comments, rate, watch times of episodes of a particular season of a TV Show. They can also rate on, comment on and mark it.

Inputs: @EpisodeID, @UserID, @UserComment, @UserRating, @WatchedInput, @Date

SQL Statement:

Finding Comments:

SELECT userName, comment

FROM comments NATURAL JOIN user

WHERE trackId = @EpisodeID

Finding Ratings, Information, Watch Count and the corresponding Episode Image :

SELECT eRating, eInfo, eWatch, eImage

FROM episode

WHERE trackId = @EpisodeID

Commenting on the Episode:

INSERT INTO comment(userId, trackId, comment, eDate)

VALUES(@UserID, @Episode, @UserComment, @Date)

After first query we update the action table

WITH actionTemp(acId,uId) AS (SELECT actionId FROM action WHERE trackId = @EpisodeID AND userId = @UserID AND eDate = @Date)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp

Rating the Episode:

INSERT INTO rate(userId, trackId, rating, eDate)

VALUES(@UserID, @EpisodeID, @UserRating, @Date)

After first query we update the action table

WITH actionTemp(acId,uId) AS (SELECT actionId FROM rate WHERE trackId = @EpisodeID AND userId = @UserID AND cDate = @Date AND rating=@UserRating)

INSERT INTO action(actionId, userId)

SELECT acId,uId

FROM actionTemp

After the action table update, we update the rating of the episode

UPDATE episode

SET eRating=newRate

FROM (

SELECT avg(rating)

FROM rate

WHERE trackId = @EpisodeID

)

WHERE trackId = @EpisodeID

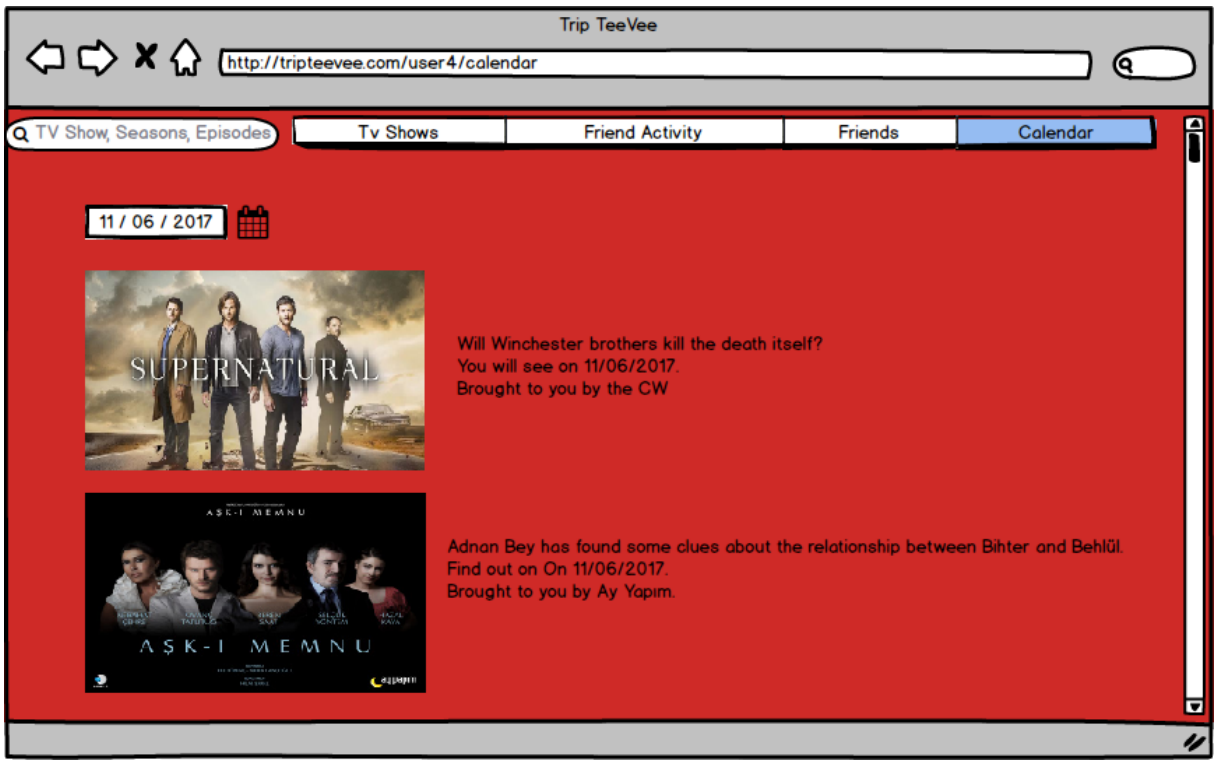
Marking the Season as Watched:

```
INSERT INTO watch(userId, trackId, wDate)
VALUES(@UserID, @EpisodeID, @Date)
WITH actionTemp(acId,uId) AS (SELECT actionId FROM watch WHERE trackId = @EpisodeID AND userId = @UserID AND cDate = @Date)
INSERT INTO action(actionId, userId)
SELECT acId,uId
FROM actionTemp
```

After the action table update, we update the watch number of the tv show

```
UPDATE episode
SET eWatch=newWatch
FROM (
SELECT count(*)
FROM watch
WHERE trackId = @EpisodeID
)
WHERE trackId = @EpisodeID
```

5.9. Calendar Page



Process: A VIP user can view the episodes aired on a particular date.

Inputs: @Date

SQL Statement:

Finding the Episodes on the Day User Choses:

SELECT eInfo, eImage

FROM episode

WHERE eDate = @Date

5.10. Friends Page

Process: Users can view their friends listed.

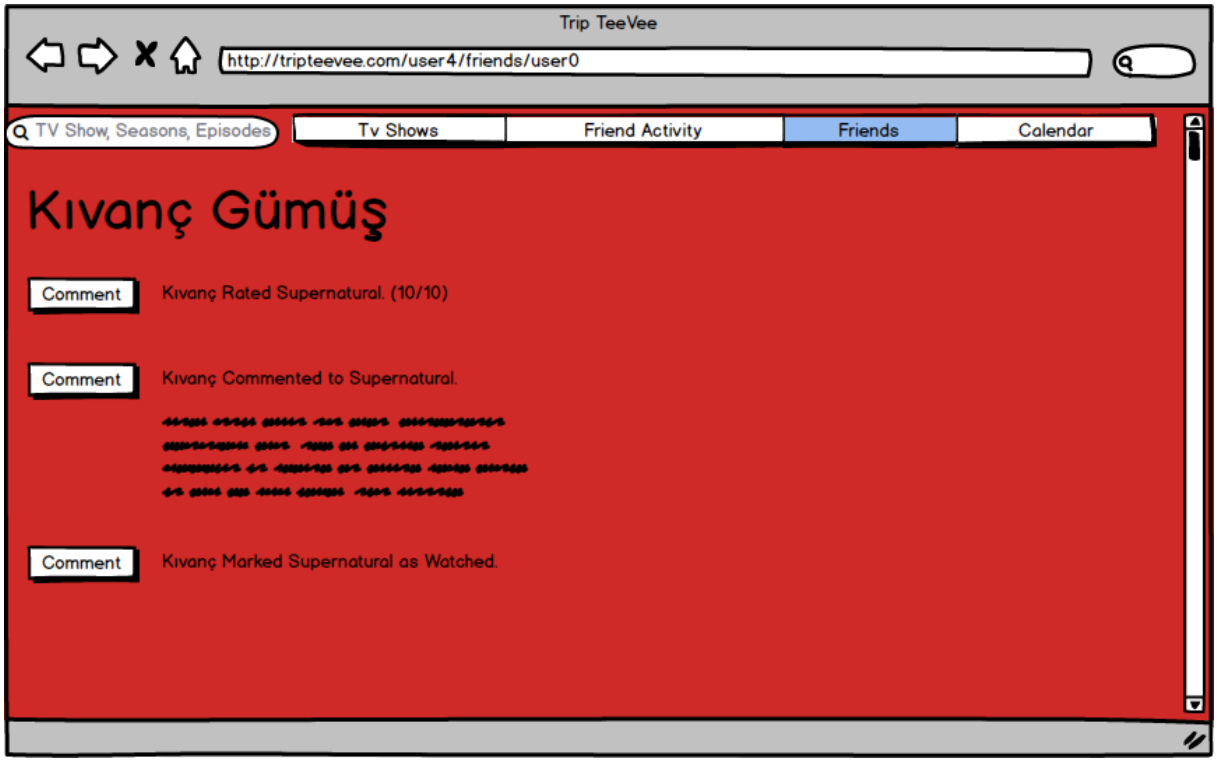
Inputs: @UserID

SQL Statement:

Finding the User's Friends' Name and IDs:

```
WITH friendIds(userId) as (SELECT friendId FROM friend WHERE userId=@UserID)
SELECT userId,uName
FROM friendIds NATURAL  JOIN user
```

5.11. Page of a Particular Friend



Process: A user can view the actions of a desired friend of theirs listed and ordered by their date of order. They can also comment to these actions.

Inputs: @FriendID

SQL Statement:

Finding a Particular User's Ratings:

for episode:

SELECT userId,uName,rating, eNumber,sNumber, tName

FROM action NATURAL JOIN rate, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND rate.trackId=episode.trackId AND sId= season.trackId AND tId = tvshows.trackId AND userId=@FriendID

for season:

SELECT userId,uName,rating, sNumber ,tName, sNumber

FROM action NATURAL JOIN rate, user, season, tvshows

WHERE user.userId=action.userId AND rate.trackId=season.trackId AND tId = tvshows.trackId AND userId=@FriendID

for tv show:

SELECT userId,uName,rating, tName

FROM action NATURAL JOIN rate, user, tvshows

WHERE user.userId=action.userId AND rate.trackId=season.trackId AND
userId=@FriendID

Finding a Particular User's Marks:

for episode:

SELECT userId,uName, eNumber,sNumber, tName

FROM action NATURAL JOIN watch, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND watch.trackId=episode.trackId AND sId=
season.trackId AND tId = tvshows.trackId AND userId=@FriendID

for season:

SELECT userId,uName, sNumber ,tName, sNumber

FROM action NATURAL JOIN watch, user, season, tvshows

WHERE user.userId=action.userId AND watch.trackId=season.trackId AND tId =
tvshows.trackId AND userId=@FriendID

for tv show:

SELECT userId,uName, tName

FROM action NATURAL JOIN watch, user, tvshows

WHERE user.userId=action.userId AND watch.trackId=season.trackId AND
userId=@FriendID

Finding a Particular User's Comments:

for episode:

SELECT userId,uName,comment, eNumber,sNumber, tName

FROM action NATURAL JOIN comments, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND comments.trackId=episode.trackId AND
sId= season.trackId AND tId = tvshows.trackId AND userId=@FriendID

for season:

SELECT userId,uName,comment, sNumber ,tName, sNumber

FROM action NATURAL JOIN comments, user, season, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId AND tId =
tvshows.trackId AND userId=@FriendID

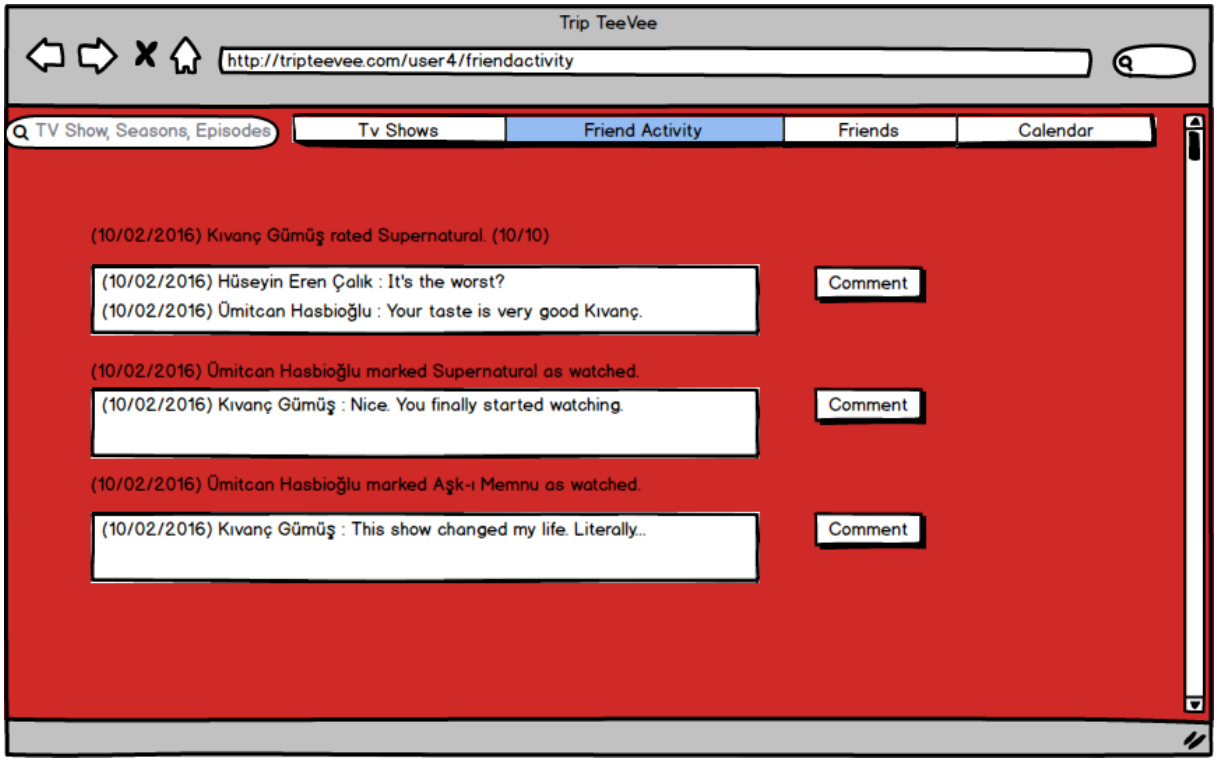
for tv show:

SELECT userId,uName,comment, tName

FROM action NATURAL JOIN comments, user, tvshows

```
WHERE user.userId=action.userId AND comments.trackId=season.trackId AND
      userId=@FriendID
```

5.12. Friend Actions Page



Process: A basic user can view and comment on their friends' actions. These will be listed and ordered by their dates.

Inputs: @UserID

SQL Statement:

Finding a Particular User's Ratings:

for episode:

SELECT userId,uName,rating, eNumber,sNumber, tName ,rDate

FROM action NATURAL JOIN rate, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND rate.trackId=episode.trackId AND sId= season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

for season:

SELECT userId,uName,rating, sNumber ,tName, sNumber,rDate

FROM action NATURAL JOIN rate, user, season, tvshows

WHERE user.userId=action.userId AND rate.trackId=season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

for tv show:

SELECT userId,uName,rating, tName,rDate

FROM action NATURAL JOIN rate, user, tvshows

WHERE user.userId=action.userId AND rate.trackId=season.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

Finding a Particular User's Marks:

for episode:

SELECT userId,uName, eNumber,sNumber, tName, wDate

FROM action NATURAL JOIN watch, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND watch.trackId=episode.trackId AND sId= season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

for season:

SELECT userId,uName, sNumber ,tName, sNumber,wDate

FROM action NATURAL JOIN watch, user, season, tvshows

WHERE user.userId=action.userId AND watch.trackId=season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

for tv show:

SELECT userId,uName, tName,wDate

FROM action NATURAL JOIN watch, user, tvshows

WHERE user.userId=action.userId AND watch.trackId=season.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

Finding a Particular User's Comments:

for episode:

SELECT userId,uName,comment, eNumber,sNumber, tName,cDate

FROM action NATURAL JOIN comments, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND comments.trackId=episode.trackId AND sId= season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

for season:

SELECT userId,uName,comment, sNumber ,tName, sNumber,cDate

FROM action NATURAL JOIN comments, user, season, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId AND tId = tvshows.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

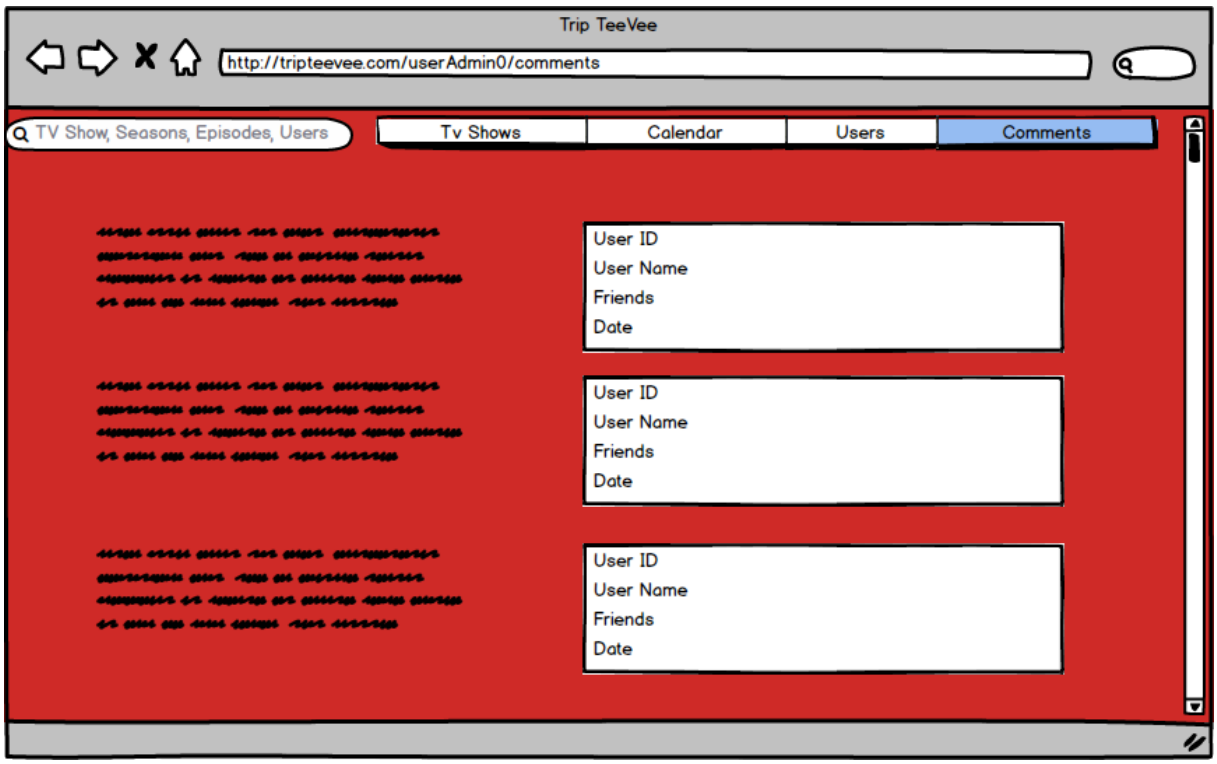
for tv show:

SELECT userId,uName,comment, tName,cDate

FROM action NATURAL JOIN comments, user, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId AND userId in (SELECT friendId FROM friend WHERE userId=@UserID)

5.13. Comments Page of Admin



Process: An Admin can view all the comments and user information of the commenters. They can also delete and modify the comments.

SQL Statement:

Finding Users' Comments, User IDs, Dates:

for episode:

SELECT userId,uName,comment, eNumber,sNumber, tName, cDate

FROM action NATURAL JOIN comments, user, episode, tvshows, seasons

WHERE user.userId=action.userId AND comments.trackId=episode.trackId AND sId= season.trackId AND tId = tvshows.trackId

for season:

SELECT userId,uName,comment, sNumber ,tName, sNumber, cDate

FROM action NATURAL JOIN comments, user, season, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId AND tId = tvshows.trackId

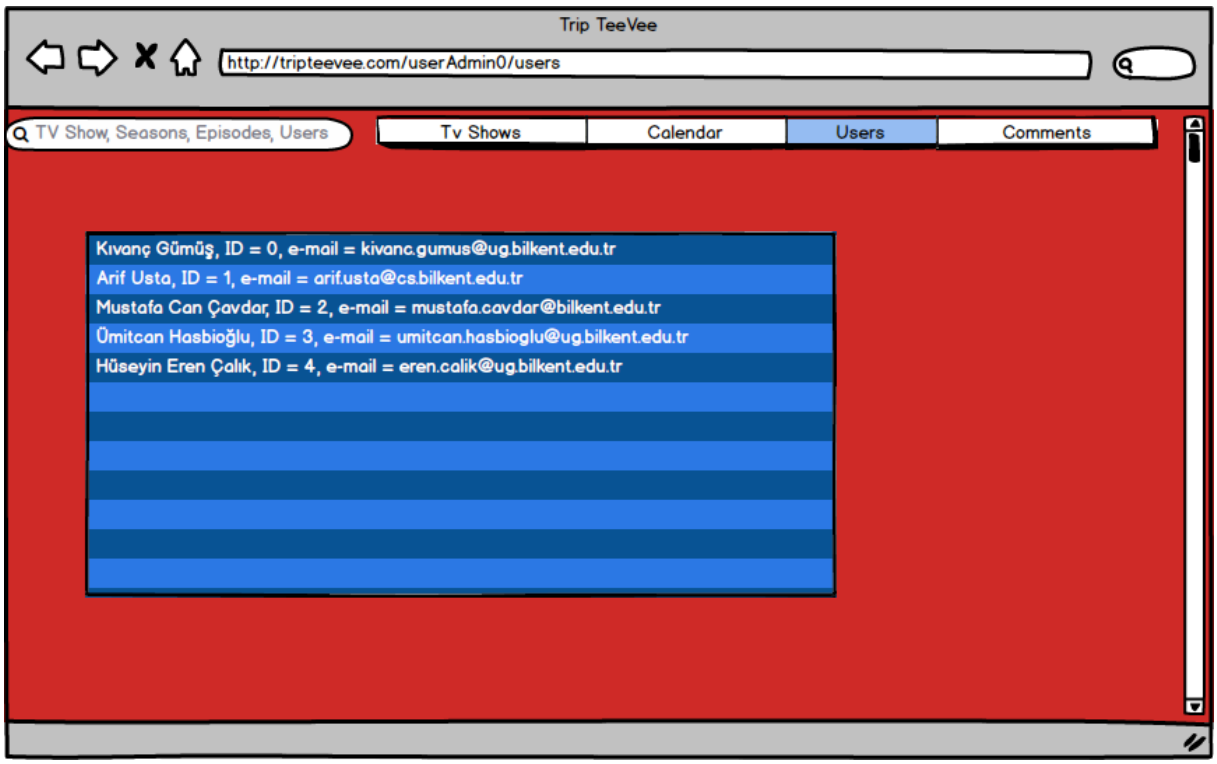
for tv show:

SELECT userId,uName,comment, tName, cDate

FROM action NATURAL JOIN comments, user, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId

5.14. Users Page of Admin



Process: An admin can view all the users and delete them.

SQL Statements:

Finding All Users' Information:

```
SELECT *
FROM user
```

ADVANCED:

Views:

Comment Views:

```
Create VIEW UserComments AS
SELECT userId,uName,comment, eNumber,sNumber, tName, cDate
FROM action NATURAL JOIN comments, user, episode, tvshows, seasons
WHERE user.userId=action.userId AND comments.trackId=episode.trackId AND
      sId= season.trackId AND tId = tvshows.trackId
UNION
SELECT userId,uName,comment, sNumber ,tName, sNumber, cDate, NULL AS
      eNumber
FROM action NATURAL JOIN comments, user, season, tvshows
```

```
WHERE user.userId=action.userId AND comments.trackId=season.trackId AND tId =
    tvshows.trackId

UNION

SELECT userId,uName,comment, tName, cDate, NULL AS eNumber, NULL AS
    sNumber

FROM action NATURAL JOIN comments, user, tvshows

WHERE user.userId=action.userId AND comments.trackId=season.trackId
```

6. Advanced Database Components

6.1. User Views

CREATE VIEW Users AS

SELECT *

FROM user

6.2. Stored Procedures

Searching: Users can provide a part of the TV Show name, friend name and the system will show the results that are similar to the search input. Searching will be handled using PHP later throughout the implementation.

6.3. Reports

Most Watched Show:

SELECT tName, MAX(watchCount)

FROM tvshows

6.4. Triggers

- When an action is taken by the user, the related tv show, season or episode's related part is updated by the system.
- When a show, season or episode is deleted, all actions related to them are deleted.
- When a user is deleted, all actions related to them are deleted.

6.5. Constraints

Guests must register and login before using the Trip TeeVee.

Guests cannot register using the same email of another account.

Users and admins must login before using their features.

Only Admins can modify,delete,add TV shows, seasons, episodes.

Only Admins can modify and delete comments.

Only Admins can delete users.

Only VIP users and Admins can use calendar feature and search for seasons, friends and episodes.

Any comments to actions must be deleted if the related action is deleted.

Users cannot mark tv shows, seasons, episodes as watched or rate them before their air date.

Users cannot mark a tv show watched or rate the tv show more than once.

Users cannot comment to the comments to the actions.

7. Implementation Plan:

Frontend: PHP, HTML

Backend: Java

Storage: Mysql