



Bilkent University

Department of Computer Engineering

CS 353 - Database Systems Project

“Track by Me”

Final Report

Project Group Number: 18

Project Group Members:

Gökcan Değirmenci, Onur Sönmez, Fatih Taş, Orçun Yalçın

Supervisor & Course Instructor: Asst. Prof. Dr. Hamdi Dibekliolu

Submitted at November 20, 2017

Description	2
Final E/R Diagram	3
Relation Schemas	4
3.1. User	4
3.2. Friendship	4
3.3. TV Show	4
3.4. Season	4
3.5. Episode	4
3.6. Comment	4
3.7. Watch	4
3.8. Show Comment	5
3.9. Season Comment	5
3.10. Episode Comment	5
3.11. Show Watch	5
3.12. Season Watch	5
3.13. Episode Watch	5
3.14. Rate	5
3.15. Show Rate	6
3.16. Season Rate	6
3.17. Episode Rate	6
3.18. Subscribe	6
3.19. Reaction	6
3.20. Comment Reaction	6
3.21. Watch Reaction	6
3.22. Notification	6
3.23. Movie	7
3.23. Similar Movie	7
3.24. Movie Comment	7
3.25. Movie Rate	7
3.26. Cast	7
3.27. Act	7
3.28. Show Act	7
3.29. Movie Act	7
3.30. Medium	7
3.31. Scheduled Episode	8
3.32. Movie Watch	8
Implementation details	8

Advanced DB features	9
Constraints	9
Views	10
Connection Pooling	10
Secondary Indexes	10
Triggers and Stored Procedures	11
Reports	13
User's Manual	15
Regular User Guide	17
Admin Guide	19

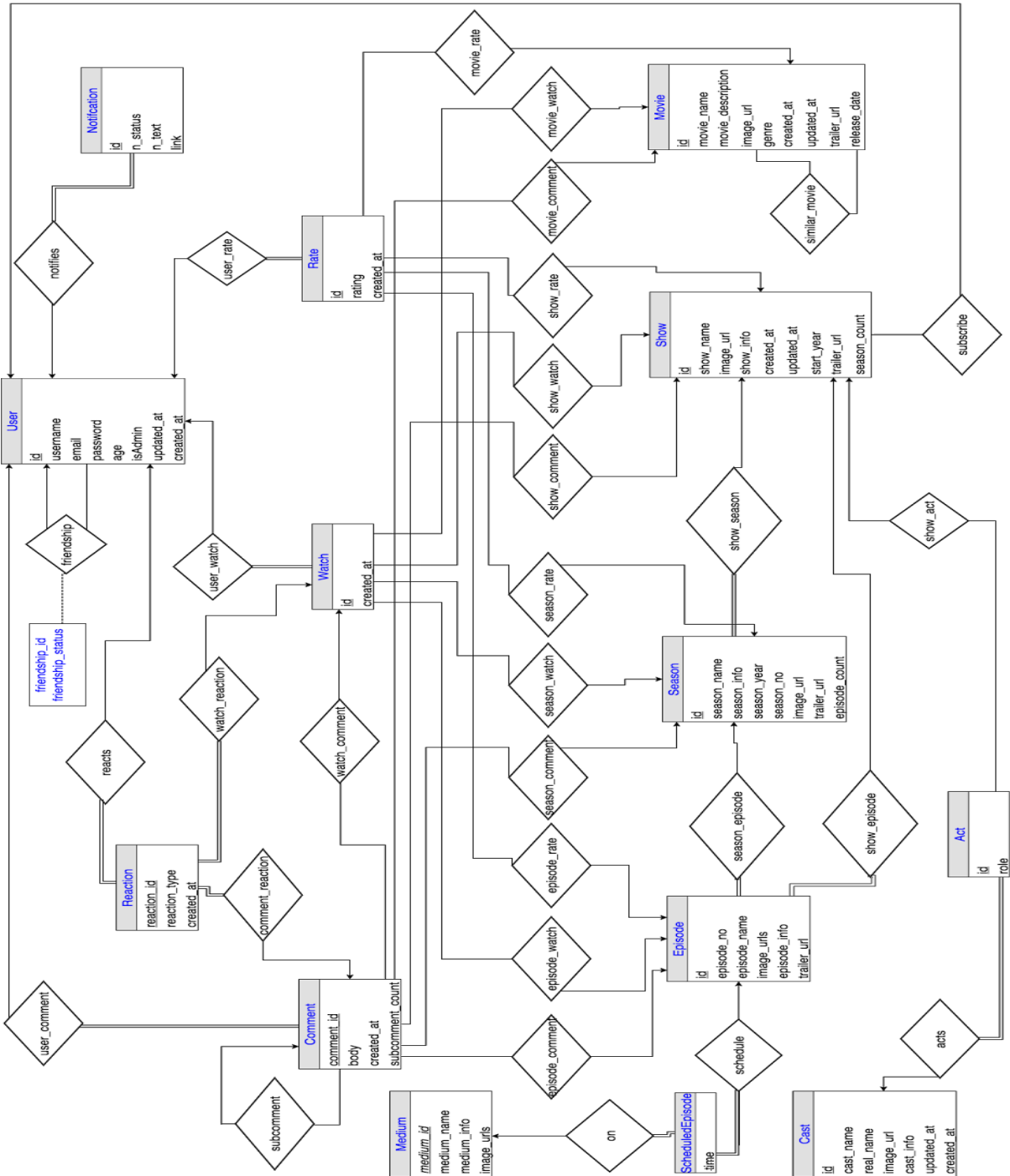
1. Description

Trackby.me is a comprehensive web-based application for tracking TV shows and movies. The system is designed to show all details of tv shows, movies, cast, medium and more. As well as monitoring all of these, also it provides much more features like rating, watching, subscribing, commenting on movies and tv shows. Additionally, it supports several social media capabilities for users like becoming friends with other users and evaluating their activities by commenting, putting reaction on them.

In general, two types of users are included. One of them is the admin who may modify almost all parts of system other than user private credentials. Admin may add, delete, update tv shows, movies, cast and medium. Nevertheless, non-admin users cannot make any of these actions. They may request a friendship, mark movies, episodes, seasons or tv shows as watched, rate them, and put comment on their pages. Remember that users must register to the system to be able to make all of these actions. After creating their profiles, they may update their profiles in any time.

Also all the actions of users are recorded and other users will be able to see their activities. Thus, communication between users are enabled through adding comment on their displayed actions and reacting them by using emojis which are similar to Facebook emojis.

2. Final E/R Diagram



2. Relation Schemas

3.1. User

users (id, username, password, email, age, isAdmin)

3.2. Friendship

friendship (first_user_id FK(users(id)), second_user_id FK(users(id)), status, action_user_id)

3.3. TV Show

tvshow (id, info, show_name, image_url, trailer_url, season_count)

3.4. Season

season(id, show_id FK(show_id)), season_no, info, season_year, image_url, trailer_url, episode_count)

3.5. Episode

episode(id, show_id FK(tvshow(id)), season_id FK(season(id)), episode_no, episode_info, image_url, trailer_url)

3.6. Comment

comment(id, parent_id FK(comment(id)), user_id FK(users(id)), body, subcomment_count)

3.7. Watch

watch(id, user_id FK(users(id)))

3.8. Show Comment

show_comment(show_id FK(tvshow(id)), comment_id FK(comment(id)))

3.9. Season Comment

season_comment(season_id FK(season(id)), comment_id FK(comment(id)))

3.10. Episode Comment

episode_comment(episode_id FK(episode(id)), comment_id FK(comment(id)))

3.11. Show Watch

show_watch(show_id FK(tvshow(id)), watch_id FK(watch(id)))

3.12. Season Watch

season_watch(season_id FK(season(id)), watch_id FK(watch(id)))

3.13. Episode Watch

episode_watch(episode_id FK(episode(id)), watch_id FK(watch(id)))

3.14. Rate

rate(id, user_id FK(users(id)), rating)

3.15. Show Rate

show_rate(show_id FK(tvshow(id)), rate_id FK(rate(id)))

3.16. Season Rate

season_rate(season_id FK(season(id)), rate_id FK(rate(id)))

3.17. Episode Rate

episode_rate(episode_id FK(episode(id)), rate_id FK(rate(id)))

3.18. Subscribe

subscribe(show_id FK(tvshow(id)), user_id(users(id)))

3.19. Reaction

reaction(id, user_id FK(users(id)), reaction_type)

3.20. Comment Reaction

comment_reaction(comment_id FK(comment(id)), reaction_id FK(reaction(id)))

3.21. Watch Reaction

watch_reaction(watch_id FK(watch(id)), reaction_id FK(watch(id)))

3.22. Notification

notification(id, user_id FK(users(id)), n_status, n_text, link)

3.23. Movie

movie(id, movie_name, movie_description, genre, imdb_rating, trailer_url, release_date)

3.23. Similar Movie

similar_movie(first_movie_id FK(movie(id)), second_movie_id FK(movie(id)))

3.24. Movie Comment

movie_comment(movie_id FK(movie(id)), comment_id FK(comment(id)))

3.25. Movie Rate

movie_rate(movie_id FK(movie(id)), rate_id FK(rate(id)))

3.26. Cast

cast(id, cast_name, real_name, image_url, cast_info)

3.27. Act

act(id, cast_id FK(cast(id)), role)

3.28. Show Act

show_act(act_id FK(act(id)), show_id FK(movie(id)))

3.29. Movie Act

movie_act(act_id FK(act(id)), movie_id FK(movie(id)))

3.30. Medium

medium(id, medium_name, info, image_url)

3.31. Scheduled Episode

scheduled_episode(medium_id FK(medium(id)), episode_id FK(episode(id)), time)

3.32. Movie Watch

movie_watch(movie_id FK(movie(id)), watch_id FK(watch(id)))

3. Implementation details

As we proposed in our initial report and further developed the application and database design in *design report*, our project consists of three main parts:

- Frontend (Client)

Client side of *TrackBy* is built upon **React.js** JavaScript library and **Semantic UI** framework. It uses next generation JavaScript, modern UI development techniques such as state-based rendering and individual custom components for achieving reliability, scalability and maintainability.

- Backend (Server)

Server side of our application is built upon **Node.js** runtime and **Express.js** web framework. All of the server code is written in TypeScript. On the other hand, our server act as a **REST API**, interacts with the persistence layer, make **CRUD** operations and returns specific resources as a response for the requests coming from client side of the application. It executes respective sql statements with the help of **node-postgres** module.

In application and data security part, we try to implement best practices in *TrackBy*. For instance, we encrypt the plain passwords coming from the client-side with bcrypt algorithm with randomly generated salt and finally store the fuzzy-string password to our database. If any case of breach occurs in our database, user credentials stay secure.

We also use **token-based authentication (JWT)** to identify the request senders and appropriately give access to protected resources.

- Persistence Layer (Database)

We choose to use PostgreSQL as a relational database. Because it has advanced

technical features, broader range of data types (CITEXT, CIRCLE, etc.) and finally it is fully open source software.

- Tools & Misc

We use **git** as version control and collaboration tool. We serve our source code in GitHub as a public repository. Also we use task-management tool such as **Trello** for better productivity.

- Problems & Challenges

To achieve smooth development experience, we need to define and also tweak some config files such as tsconfig, tslint, .env, package.json. At first, it was trouble for us. However, we did solve those problem after doing some research on internet. Another challenge was to adapt and use the PostgreSQL's own syntax (generally same as the MySQL). So again we did read the documentation, took some tutorials and voila!

4. Advanced DB features

- Constraints

Note: CITEXT is case insensitive character string type for postgres.

Friendship:

```
status CITEXT NOT NULL CHECK (  
    status IN ('PENDING', 'APPROVED', 'REJECTED')  
),  
CHECK (first_user_id < second_user_id)
```

Notification:

```
n_status CITEXT NOT NULL CHECK (  
    n_status IN ('WAITING', 'SEEN')  
)
```

Act:

```
role CITEXT NOT NULL CHECK (  
    role IN ('ACTOR', 'DIRECTOR', 'WRITER')  
)
```

Reaction:

```
reaction_type CITEXT NOT NULL CHECK (  
    reaction_type IN ('NONE', 'LIKE', 'LOVE', 'WOW', 'HAHA',  
    'SAD', 'ANGRY', 'THANKFUL')  
)
```

- Views

```
CREATE VIEW public_user AS  
    SELECT id, username, email FROM users WHERE isAdmin = False
```

- Connection Pooling

It allows us to reduce database-related overhead when it's the sheer number of physical connections dragging performance down. We use pooling method in all of the server connections to our persistence layer, PostgreSQL database.

- Secondary Indexes

```
CREATE UNIQUE INDEX age_x ON users (age);  
CREATE INDEX username_index ON users USING hash (username,  
password);
```

username_index allows to perform login operation faster. Since the login query is based on equality and uses the structure *WHERE username=\$1 AND password=\$2*, hash based indexing is the best choice.

```
CREATE INDEX showname_index ON tvshow (show_name)  
CREATE INDEX moviename_index ON movie (movie_name)  
CREATE INDEX episode_no_index ON episode USING btree (episode_no)  
CREATE INDEX season_no_index ON season USING btree (season_no)
```

Note: Some dbs provide default indexing for foreign keys but postgres does not do this. And also this will help to speed up JOIN conditions.

```

CREATE INDEX show_comment_index ON show_comment(comment_id)
CREATE INDEX movie_comment_index ON movie_comment(comment_id)
CREATE INDEX season_comment_index ON season_comment(comment_id)
CREATE INDEX episode_comment_index ON episode_comment(comment_id)
CREATE INDEX watch_index ON show_watch(watch_id)
CREATE INDEX movie_watch_index ON movie_watch(watch_id)
CREATE INDEX season_watch_index ON season_watch(watch_id)
CREATE INDEX episode_watch_index ON episode_watch(watch_id)
CREATE INDEX movie_rate_index ON movie_rate(rate_id)
CREATE INDEX show_rate_index ON show_rate(rate_id)
CREATE INDEX season_rate_index ON season_rate(rate_id)
CREATE INDEX episode_rate_index ON episode_rate(rate_id)
CREATE INDEX comment_reaction_index ON
comment_reaction(reaction_id)
CREATE INDEX watch_reaction_index ON watch_reaction(reaction_id)

```

- Triggers and Stored Procedures

```

-- SEASON COUNT UPDATE
CREATE OR REPLACE FUNCTION increase_season_count()
  RETURNS trigger AS
$$
BEGIN
  UPDATE tvshow SET season_count = season_count + 1
  WHERE tvshow.id = NEW.show_id;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

```

```

CREATE OR REPLACE FUNCTION decrease_season_count()
  RETURNS trigger AS
$$
BEGIN
  UPDATE tvshow SET season_count = season_count - 1
  WHERE tvshow.id = OLD.show_id;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

```

```

CREATE TRIGGER inc_season_count_trigger
  AFTER INSERT

```

```

ON season
FOR EACH ROW
EXECUTE PROCEDURE increase_season_count();

CREATE TRIGGER dec_season_count_trigger
AFTER DELETE
ON season
FOR EACH ROW
EXECUTE PROCEDURE decrease_season_count();

-- EPISODE COUNT UPDATE
CREATE OR REPLACE FUNCTION increase_episode_count()
RETURNS trigger AS
$$
BEGIN
    UPDATE season SET episode_count = episode_count + 1
    WHERE season.id = NEW.season_id;
    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE OR REPLACE FUNCTION decrease_episode_count()
RETURNS trigger AS
$$
BEGIN
    UPDATE season SET episode_count = episode_count - 1
    WHERE season.id = OLD.season_id;
    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER inc_episode_count_trigger
AFTER INSERT
ON episode
FOR EACH ROW
EXECUTE PROCEDURE increase_episode_count();

CREATE TRIGGER dec_episode_count_trigger
AFTER DELETE
ON episode
FOR EACH ROW

```

```

EXECUTE PROCEDURE decrease_episode_count();

-- SUBCOMMENT COUNT UPDATE
CREATE OR REPLACE FUNCTION increase_subcomment_count()
  RETURNS trigger AS
$$
BEGIN
  UPDATE comment SET subcomment_count = subcomment_count + 1
    WHERE comment.id = NEW.parent_id;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE OR REPLACE FUNCTION decrease_comment_count()
  RETURNS trigger AS
$$
BEGIN
  UPDATE comment SET subcomment_count = subcomment_count - 1
    WHERE comment.id = OLD.parent_id;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER inc_subcomment_count_trigger
  AFTER INSERT
  ON comment
  FOR EACH ROW
  EXECUTE PROCEDURE increase_subcomment_count();

CREATE TRIGGER dec_subcomment_count_trigger
  AFTER DELETE
  ON comment
  FOR EACH ROW
  EXECUTE PROCEDURE decrease_subcomment_count();

```

- Reports

Rating of shows with highest number of rates

Query:

```
SELECT show_name, show_id, AVG(rating) as average_rating FROM rate
```

```

INNER JOIN show_rate ON show_rate.rate_id = rate.id
INNER JOIN tvshow ON tvshow.id = show_rate.show_id
WHERE show_rate.show_id
IN (
    SELECT show_id FROM show_rate
    GROUP BY show_id
    ORDER BY COUNT(*) DESC LIMIT 3
)
GROUP BY show_id

```

Sample Result:

show_name	show_id	average_rating
Mr. Robot	5	9.5
Vikings	3	9.0
Gossip Girl	2	4.6

Rating Statistics of Users Who Rated Highest Number of Rates

Query:

```

SELECT
    username,
    AVG(rating) as average_rating,
    stddev(rating) as rating_deviation,
    COUNT(*) as rate_count
FROM users
INNER JOIN rate ON rate.user_id = users.id
GROUP BY users.id
ORDER BY rate_count DESC
LIMIT 4

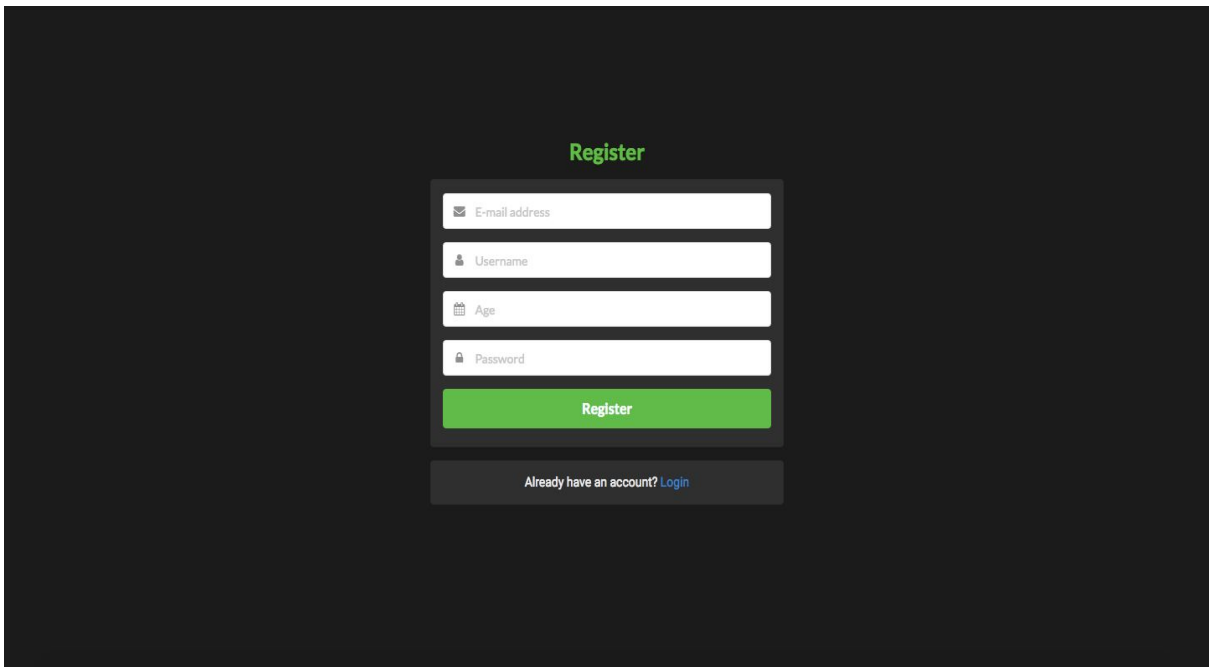
```

Sample Result:

username	average_rating	rating_deviation	rate_count
fatih	8.1	2.3	40

gokcan	7.2	3.6	37
onur	8.5	0.4	28
orcun	4.6	3.8	19

5. User's Manual



The image shows a 'Register' screen with a dark background. At the top center, the word 'Register' is written in green. Below it is a form with four input fields: 'E-mail address' (with an envelope icon), 'Username' (with a person icon), 'Age' (with a calendar icon), and 'Password' (with a lock icon). A green 'Register' button is positioned below the fields. At the bottom of the form, there is a link that says 'Already have an account? Login'.

Figure 1 - Register Screen

As it is shown in Figure 1, a user can create an account in the system by filling all the fields in register screen and clicking Register button. If an error occurs while creating the account, it will be displayed to user. If there is no error, then user will be authenticated and redirected to homepage.

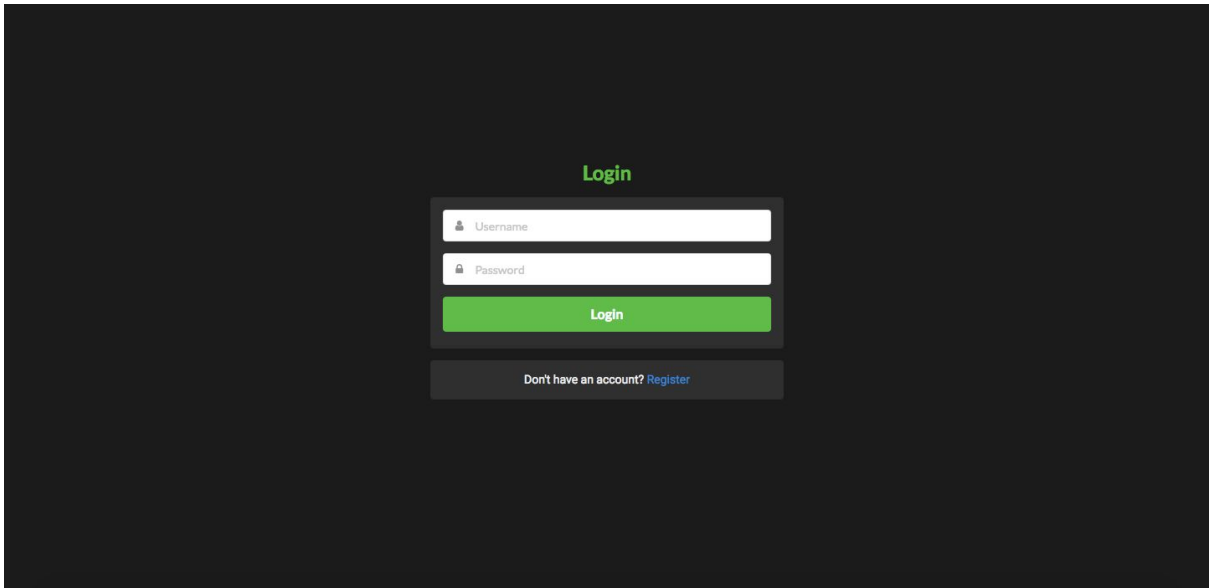


Figure 2- Login Screen

Figure 2 shows the login screen of the application. User can login to the system by filling those fields and login. If an error occurs during the login process it will be displayed to user and if there is no error then user will be authenticated and redirected to homepage of users' type e.g. Admin homepage if user is admin and regular homepage otherwise.

5.1. Regular User Guide

MR. Robot

Mr. Robot follows Elliot Alderson, a young computer programmer with an anxiety disorder, who is recruited by Mr Robot and his anarchist team of hackers 'fsociety'.

Directed by
Niels Arden Oplev

Written by
Sam Esmail

★ 9.5 Rate ☆ Mark As Watched Subscribe 3 Seasons

Seasons

Season	Year	Country	Episodes	Rating	Status
Mr. Robot: Season 3	2017	USA	9 episodes	★ 9.5	Mark as Watched
Mr. Robot: Season 2	2016	USA	12 episodes	★ 9.5	Watched
Mr. Robot: Season 1	2015	USA	10 episodes	★ 9.5	Mark as Watched

Cast

Actor	Character
Christian Slater	Mr. Robot
Rami Malek	Eliot
Carly Chaikin	Darlene
Martin Wallström	Tyrell
Portia Doubleday	Angela

Comments

38 Comments Sort By

U Write a comment

D Donnie
Great show!!! Great cast!!!
Reply 3 0 0 0 0 0
View All 5 Replies

K Katie
I love this show
Reply 3 0 0 0 0 0

Show More Comments...

Figure 3 - Example TV Show Page

Figure 3 shows an example TV Show page. Although Episode page, Season page, Movie page shows different content, structure of their user interface is similar. Therefore TV Show page is used as a sample to describe the actions that a user can do on those pages.

At top side of the page there are watch and rate buttons. User can click

mark as watched button to mark this tv show as watched and rate button to rate this tv show. At the bottom comments to the show is displayed. User can create a new comment by using the text input above listed comments. User can also reply to other comments by clicking the reply button on comment and filling the text input that appears. Icons next to the reply button are reaction buttons, using those buttons user can add a reaction to a comment and remove it by clicking the button again.

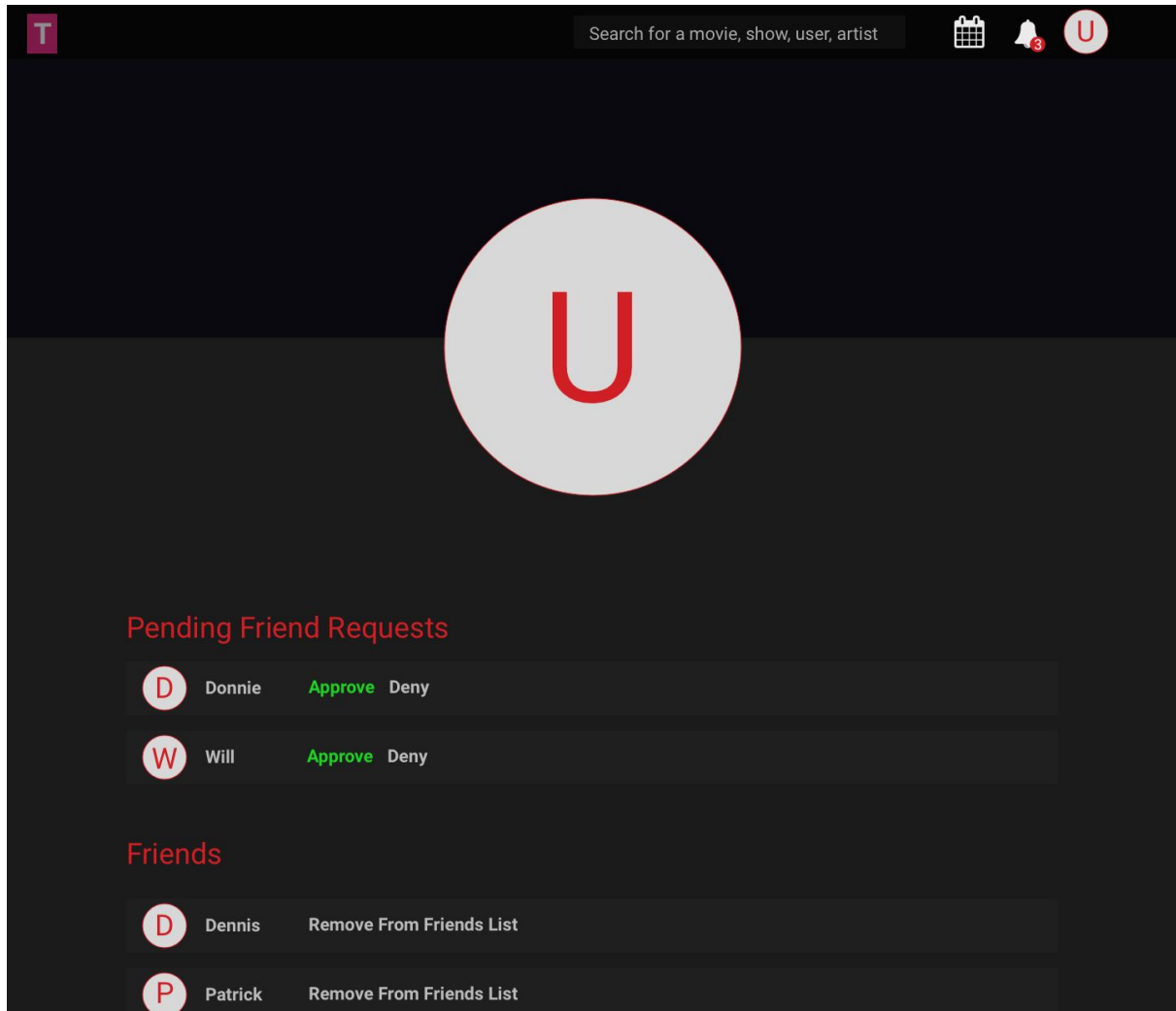


Figure 4 - Friends page

Figure 4 show the friend page. User can manage their friends from this page. Using the pending friend requests section user can approve or deny a friend request. Also user can remove anyone from friend list by using the Remove from friends list button.

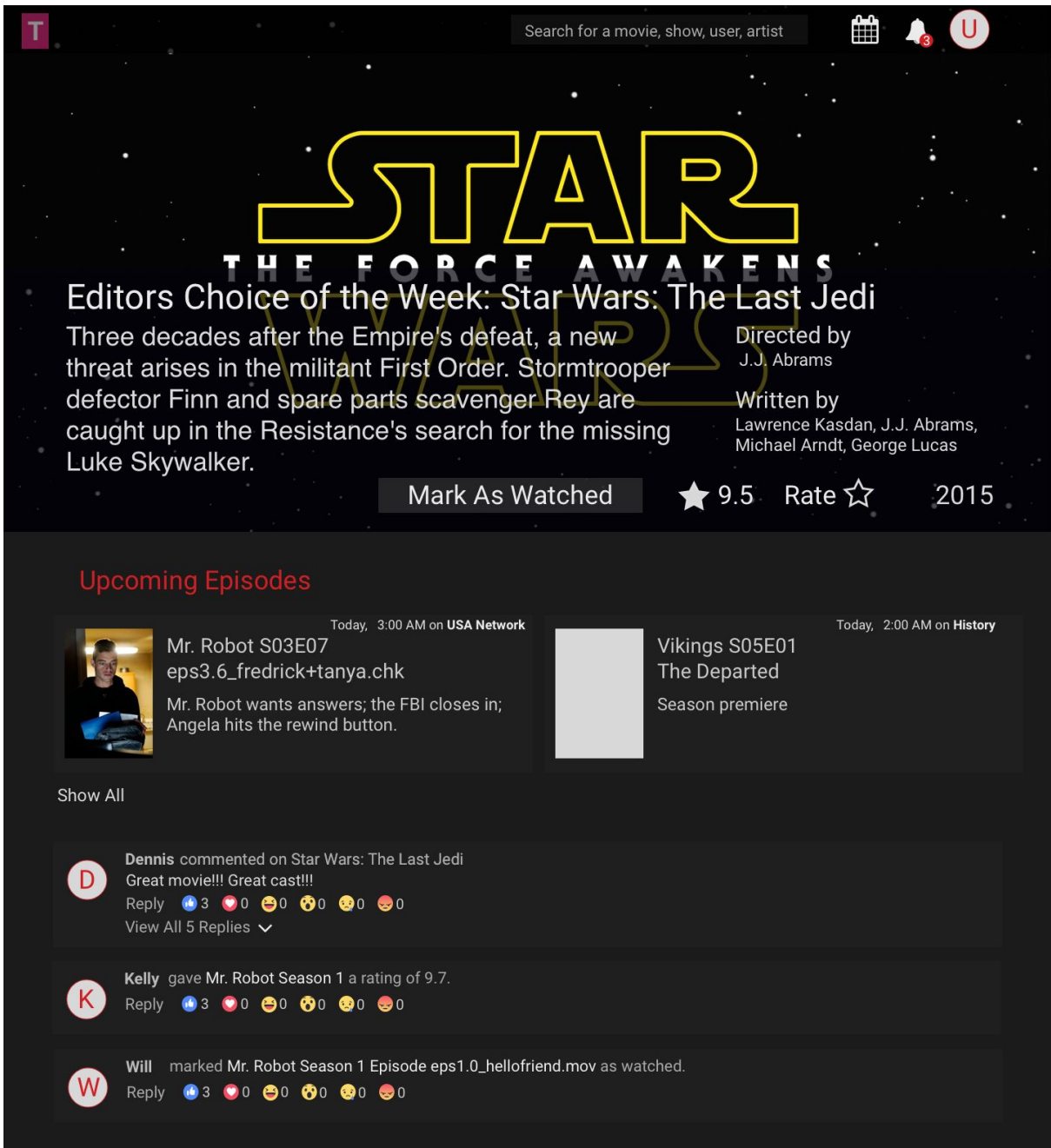


Figure 5 - Homepage

Figure 5 shows the homepage. On bottom part of the page users can see activities of their friends and can comment on or send reaction to them. User can search for tv shows, episodes, movies and users by using the search field at navigation bar.

5.2. Admin Guide

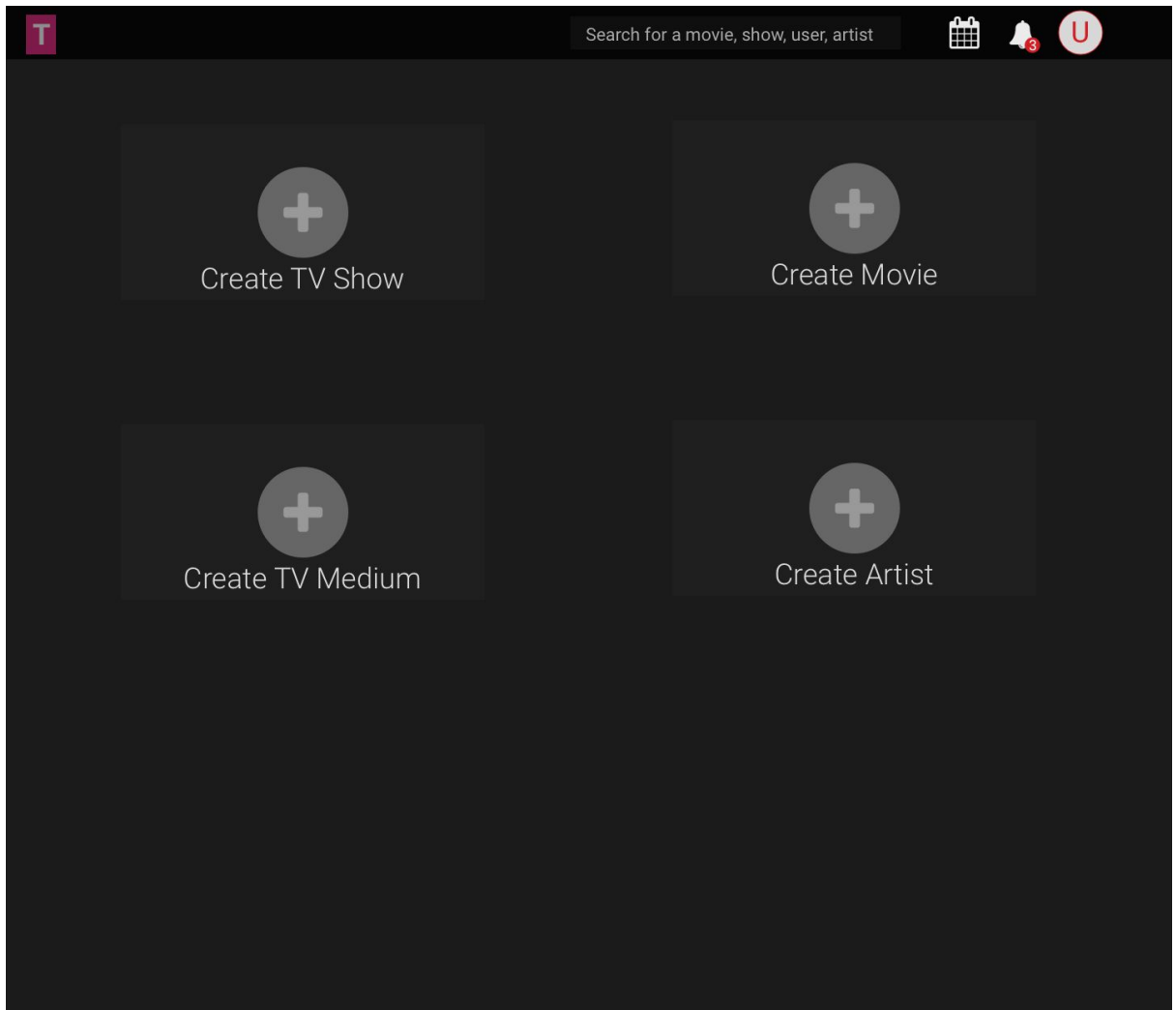


Figure 6 - Admin homepage

Besides the functionalities for regular users, admins can perform extra operations on the system. Figure 6 shows the admin homepage. By using these admin can create a new entity. Clicking one of these buttons shows a modal with necessary input fields for entering the details of the entity.

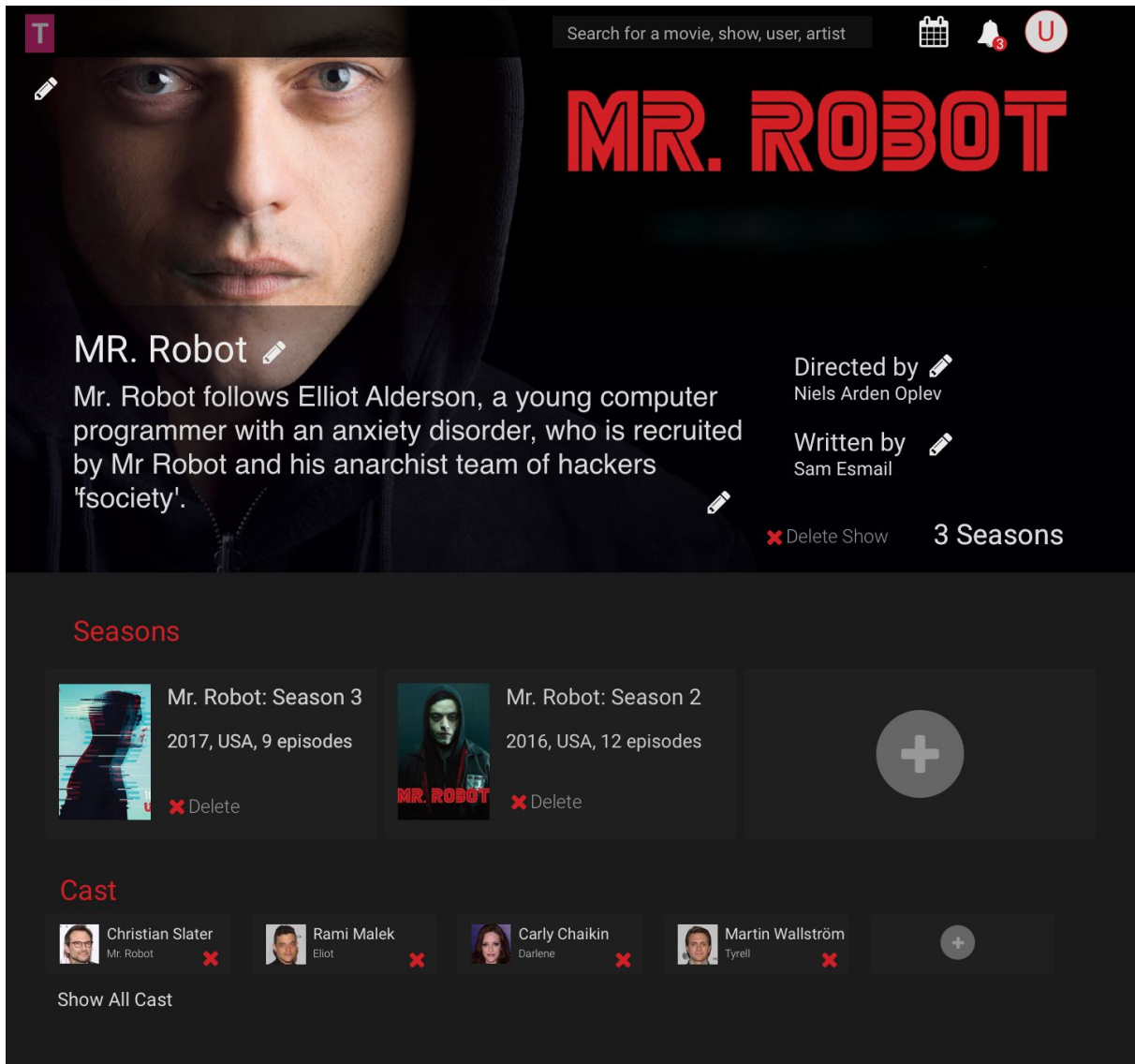


Figure 7 - Admin Tv Show Page

Figure 7 shows how tv show page is displayed to an admin user. In terms of user interface structure is similar to movie, season, episode pages. Thus Tv show page is used as a sample for describing the admin operations on these pages. There are several pencil icons on header part. Clicking these pencil icons make the text next to them editable and by editing these texts admin can modify the attributes of show. The plus icons in seasons section and in cast section allows admin to add season and new artist to show. Clicking these buttons will also display a modal similar to the ones on homepage. Red cross icons on seasons and artists allows admin to remove them. Also by clicking the delete show button at the header part , admin can delete the show.