



Bilkent University

Department of Computer Engineering

# CS 353 Term Project

*Project Tracking Software*

## Design Report

Bartu Atabek	21602229
Utku Görkem Ertürk	21502497
Hygerta Imeri	21603212
Deniz Yüksel	21600880

Instructor: Özgür Ulusoy

Assigned TA: Aytek Aman



April 05, 2019

This report is submitted (softcopy) to <http://track-it.cf> in partial fulfilment of the requirements of the Term Project of CS353 course.

# Table of Contents

---

<b>Table of Contents</b>	<b>1</b>
1. Revised E/R Model	3
1.1. Changes made in E/R Diagram	3
2. Relation Schemas	6
2.1 Release	6
2.2 Attachment	6
2.3 Card	7
2.4 Lists	8
2.5 CardTag	8
2.6 IssueTag	8
2.7 Schedule	9
2.8 Event	9
2.9 Team	10
2.10 Theme	10
2.11 Project	11
2.12 Actions	11
2.13 Watchlist	12
2.14 Archive	12
2.15 Board	12
2.16 AssignsCard	13
2.17 AssignsIssue	13
2.18 memberOf	14
2.19 Retains	14
2.20 User	15
2.21 PrivilegedUser	15
2.22 StandardUser	15
2.23 Issue	16
2.24 Contributes	16
2.25 AssignsIssues	17
2.26 Comment	17
3. Functional dependencies and normalization	18
4. Functional Components	18
4.1 Use Cases / Scenarios	18
4.1.1 CreateBoard	19
4.1.2 CreateTeam	19

4.1.3 InviteTeamMembers	19
4.1.4 SetPrivileges	20
4.1.5 CreateList	21
4.1.6 CreateCard	21
4.1.7 AssignUsers	22
4.1.8 RemoveCard	23
4.1.9 ArchiveCard	23
4.1.10 AddComments	24
4.1.11 MentionUsers	25
4.1.12 Release	25
4.1.13 CreateIssue	26
4.1.14 SetTheme	26
4.1.10 AddAttachment	27
4.1.11 Login	30
4.1.11 SignUp	30
5. User Interface Design and SQL Statements	32
5.1 General Pages	32
5.1.1 Home Page (without login)	32
5.1.2 Login Page (for both user type)	33
5.1.3 Sign Up Page (for both user type)	34
5.2 Project Pages	35
5.2.1 Projects Page	35
5.2.2 Create Project & Team	36
5.2.3 Boards Page	41
5.2.4 Board Content Page	42
5.2.5 Issues Page	46
5.2.6 Releases Page	48
6. Advanced Database Components	49
6.1 Reports	49
6.2 Views	50
6.3 Triggers	51
6.3.1 Schedule Reminder Trigger	51
6.3.2 Increment Version Trigger	51
6.3.3 Priority Notification Trigger	52
6.4 Constraints	52
6.5 Stored Procedure	52
6.5.1 Filtering Cards	52
6.5.2 Filtering Issues	52
7. Implementation Plan	52
8. Website	52

# 1. Revised E/R Model

---

## 1.1. Changes made in E/R Diagram

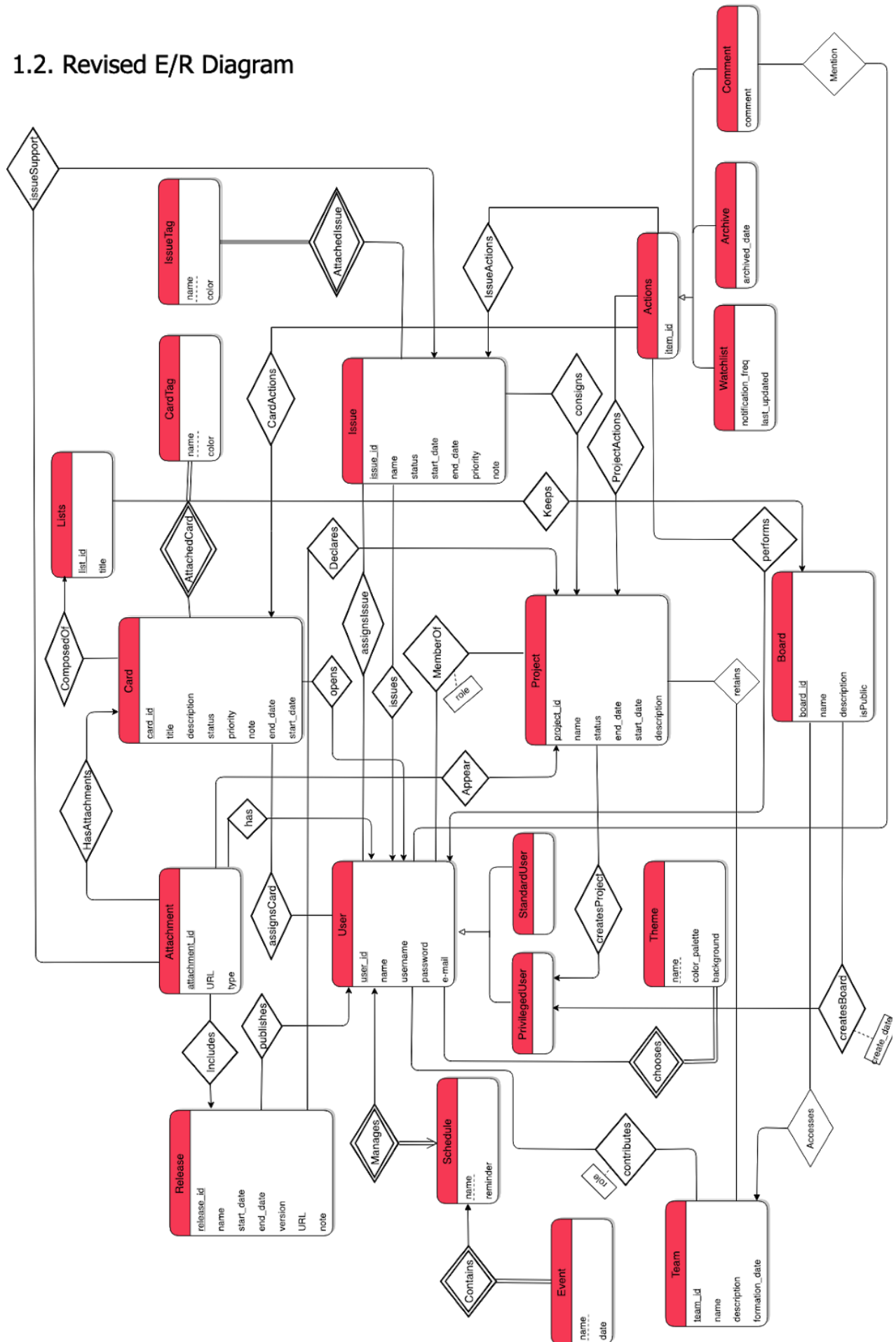
The following changes were made in our E/R diagram:

- Relation names were changed into more meaningful and unique names. There had been more than one relations with the name 'Has'. Now, every relation have an identifying name.
- The previous version of the E/R diagram did not contain any primary keys. We have added primary keys for all the existing entities along with the newly added ones.
- To be coherent with the project description specifications, we added list and board entities which keep cards inside the project in an organized manner, as well as to separate the project environment. Board has "board\_id", "name", "description", "isPublic" attributes. "Name", "description" and "board\_id" are trivial features. isPublic is used for the state of the board if the board can be seen by other users or just the members of the team that owns it. Board keeps list and list keeps cards. List has cards (previous name was Card) and list has attribute 'id' and 'title'. Privileged user can create board and teams own boards. Description attribute was added into Card entity.
- Tag entity has been deleted. CardTag and IssueTag were specialized (out of the general Tag entity) as separate weak entities.
- Attachment was a weak entity which participated in more than one relationship as a weak entity. We converted it from being a weak entity to a normal entity to conform the rules. Attachment now has its every relation as one to many, signifying that every attachment is unique.
- Attachment entity has a relation with Issue, namely "IssueSupport".
- We have added 'Actions' entity specialized into three entities which represents the actions users could make or do on on projects, cards, issues, etc. The specialized entities 'Watchlist' and 'Archive' store the information about a user's tracked items and their notification frequency which will prompt the users when an item they track has been updated or modified. The archive action serves as a storage history tool where users could archive certain items to be accessed later or reverted back. The third action added is the "Comment". A user can write a comment on a card, issue or a project. The "Actions" entity now has its relations as one to many, therefore every action is unique. Since action entity is

the one related to 'Project', 'Issue' and 'Card' entity, all of the specialized actions under 'Actions' entity can be performed on the aforementioned entities.

- Schedule entity had no relations with any other entity. An entity with the name "Event" has been created therefore, Event-Schedule' entities have a "Contains" relation. Schedule entities' date and name attributes were changed to name and reminder.
- New entities that extend the User entity were added. 'StandardUser' and 'PrivilegedUser' entities are added. Also, the User entity now has a relation, namely 'Performs', with 'Actions' entity.
- 'PrivilegedUser-Project' entities now have a 'createsProject' relation with the attribute 'createDate'.
- 'Project-Teams' entities have the relation named 'Retains'.
- 'Team-Board' entities now have the 'Access' relation.
- Theme has become a weak entity (dependent on 'User' entity).
- Mention relationship is added between 'Comment' and 'User' so that users can mention other users in comments.

## 1.2. Revised E/R Diagram



## 2. Relation Schemas

---

### 2.1 Release

**Relational Model:** Release(release\_id, name, start\_date, end\_date, version, URL, note, user\_id, project\_id)

**Functional Dependencies:** {(release\_id → name, start\_date, end\_date, version, URL, note, user\_id, project\_id)}

**Candidate Keys:** {(release\_id)}

**Primary Key:** {(release\_id)}

**Foreign Keys:** {(user\_id), (project\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Release(  
  release_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(50) NOT NULL,  
  start_date DATE NOT NULL,  
  end_date DATE,  
  version VARCHAR(20),  
  URL VARCHAR(200),  
  note VARCHAR(200),  
  user_id INTEGER,  
  project_id INTEGER,  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (project_id) REFERENCES Project(project_id),  
  CHECK (start_date <= end_date)  
);
```

### 2.2 Attachment

**Relational Model:** Attachment(attachment\_id, url, type, card\_id, project\_id, user\_id, issue\_id, release\_id)

**Functional Dependencies:** {(attachment\_id → url, type, card\_id, project\_id, user\_id, issue\_id, release\_id)}

**Candidate Keys:** {attachment\_id}

**Primary Key:** {attachment\_id}

**Foreign Keys:** { (release\_id), (project\_id), (issue\_id), (card\_id), (user\_id) }

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Attachment(  

```

```

attachment_id INTEGER PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(50) NOT NULL,
URL VARCHAR (200),
type VARCHAR (20),
note VARCHAR (200),
release_id INTEGER,
project_id INTEGER,
issue_id INTEGER,
card_id INTEGER,
user_id INTEGER,
FOREIGN KEY (user_id) REFERENCES User(user_id),
FOREIGN KEY (project_id) REFERENCES Project(project_id),
FOREIGN KEY (release_id) REFERENCES Release(release_id),
FOREIGN KEY (issue_id) REFERENCES Issue(issue_id),
FOREIGN KEY (card_id) REFERENCES Card(card_id)
);

```

## 2.3 Card

**Relational Model:** Card(card\_id, title, description, priority, status, note, start\_date, end\_date, user\_id, list\_id)

**Functional Dependencies:** {( card\_id→ title, description, priority, status, note, start\_date, end\_date, user\_id, list\_id)}

**Candidate Keys:** {(card\_id)}

**Primary Key:** {(card\_id)}

**Foreign Keys:** {(user\_id), (list\_id)}

**Normal Form:** BCNF

**Table Definition:**

```

CREATE TABLE Cards(
card_id INTEGER PRIMARY KEY AUTO_INCREMENT,
title VARCHAR(50) NOT NULL,
description VARCHAR(200),
priority INTEGER,
status VARCHAR(50) NOT NULL,
note VARCHAR(200),
start_date DATE NOT NULL,
end_date DATE,
user_id INTEGER,
list_id INTEGER,
FOREIGN KEY(user_id) REFERENCES User(user_id),
FOREIGN KEY(list_id) REFERENCES List(list_id),
CHECK (start_date <= end_date),

```



```
CHECK (priority >= 0)
);
```

## 2.4 Lists

**Relational Model:** Lists(list\_id, title, board\_id)

**Functional Dependencies:** {(list\_id → title, board\_id)}

**Candidate Keys:** {(list\_id)}

**Primary Key:** {(list\_id)}

**Foreign Keys:** {(board\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Lists(
list_id  INTEGER PRIMARY KEY AUTO_INCREMENT,
title  VARCHAR(50) NOT NULL,
board_id  INTEGER,
FOREIGN KEY (board_id) REFERENCES Board(board_id)
);
```

## 2.5 CardTag

**Relational Model:** CardTag(card\_id, name, color)

**Functional Dependencies:** {(card\_id, name → color)}

**Candidate Keys:** {(card\_id, name)}

**Primary Key:** {(card\_id, name)}

**Foreign Keys:** {(card\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE CardTag(
color  VARCHAR(50) NOT NULL,
name  VARCHAR(50) NOT NULL,
card_id  INTEGER,
PRIMARY KEY (card_id),
FOREIGN KEY (card_id) REFERENCES User(card_id)
);
```

## 2.6 IssueTag

**Relational Model:** IssueTag(issue\_id, name, color)

**Functional Dependencies:** {(issue\_id, name → color)}

**Candidate Keys:** {(issue\_id, name)}

**Primary Key:** {(issue\_id, name)}

**Foreign Keys:** {(issue\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE IssueTag(  
  color VARCHAR(50) NOT NULL,  
  name VARCHAR(50) NOT NULL,  
  issue_id INTEGER,  
  PRIMARY KEY (issue_id),  
  FOREIGN KEY (issue_id) REFERENCES Issue(issue_id)  
);
```

## 2.7 Schedule

**Relational Model:** Schedule(user\_id, name, reminder)

**Functional Dependencies:** {(user\_id → reminder, name)}

**Candidate Keys:** {(user\_id)}

**Primary Key:** {(user\_id)}

**Foreign Keys:** {(user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Schedule(  
  name VARCHAR(50) NOT NULL,  
  reminder DATE NOT NULL,  
  user_id INTEGER,  
  PRIMARY KEY (user_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id)  
);
```

## 2.8 Event

**Relational Model:** Event(name, schedule\_name, date, user\_id)

**Functional Dependencies:** {(user\_id, name, schedule\_name → date)}

**Candidate Keys:** {(user\_id, name, schedule\_name)}

**Primary Key:** {(user\_id, name, schedule\_name)}

**Foreign Keys:** {(user\_id, schedule\_name)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Event(  
  name VARCHAR(50) NOT NULL,  
  date DATE NOT NULL,
```

```

user_id INTEGER,
schedule_name VARCHAR(50),
PRIMARY KEY (user_id,name, schedule_name),
FOREIGN KEY (user_id) REFERENCES Schedule(user_id)
FOREIGN KEY (schedule_name) REFERENCES Schedule(name)
);

```

## 2.9 Team

**Relational Model:** Team(team\_id, name, description, formation\_date)

**Functional Dependencies:** {(team\_id → name, description, formation\_date)}

**Candidate Keys:** {(team\_id)}

**Primary Key:** {(team\_id)}

**Foreign Keys:** {}

**Normal Form:** BCNF

**Table Definition:**

```

CREATE TABLE Team(
team_id INTEGER PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(50) NOT NULL,
description VARCHAR(200) NOT NULL,
formation_date DATE NOT NULL
);

```

## 2.10 Theme

**Relational Model:** Theme(name, color\_palette, background, user\_id)

**Functional Dependencies:** {(user\_id, name → color\_palette, background)}

**Candidate Keys:** {(user\_id, name)}

**Primary Key:** {(user\_id, name)}

**Foreign Keys:** {(user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```

CREATE TABLE Theme(
name VARCHAR(100),
color_palette VARCHAR(50) NOT NULL,
background VARCHAR(50) NOT NULL,
user_id INTEGER,
PRIMARY KEY (name, user_id),
FOREIGN KEY(user_id) REFERENCES User(user_id)
);

```

## 2.11 Project

**Relational Model:** Project( project\_id, name, status, end\_date, start\_date, description, user\_id)

**Functional Dependencies:** {(project\_id → name, status, end\_date, start\_date, description, user\_id)}

**Candidate Keys:** {project\_id}

**Primary Key:** {project\_id}

**Foreign Keys:** {(user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Project(  
  project_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
  status VARCHAR(50) NOT NULL,  
  name VARCHAR(50) NOT NULL,  
  start_date DATE NOT NULL,  
  end_date DATE NOT NULL,  
  description VARCHAR(200),  
  user_id INTEGER,  
  CHECK (end_date >= start_date),  
  FOREIGN KEY (user_id) REFERENCES User(user_id)  
);
```

## 2.12 Actions

**Relational Model:** Actions(item\_id, card\_id, user\_id, project\_id, issue\_id)

**Functional Dependencies:** {(item\_id → card\_id, user\_id, project\_id, issue\_id)}

**Candidate Keys:** {(item\_id)}

**Primary Key:** {(item\_id)}

**Foreign Keys:** {(user\_id), (card\_id), (project\_id), (issue\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Attachment(  
  item_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
  user_id INTEGER,  
  project_id INTEGER,  
  issue_id INTEGER,  
  card_id INTEGER,  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (project_id) REFERENCES Project(project_id),  
  FOREIGN KEY (issue_id) REFERENCES Issue(issue_id),  
  FOREIGN KEY (card_id) REFERENCES Card(card_id)
```

);

## 2.13 Watchlist

**Relational Model:** Watchlist(item\_id, notification\_freq, last\_updated)

**Functional Dependencies:** {(item\_id → notification\_freq, last\_updated)}

**Candidate Keys:** {(item\_id)}

**Primary Key:** {(item\_id)}

**Foreign Keys:** {(item\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Watchlist(  
  notification_freq INTEGER,  
  last_updated DATE,  
  item_id INTEGER,  
  PRIMARY KEY (item_id) AUTO_INCREMENT,  
  FOREIGN KEY (item_id) REFERENCES Actions(item_id)  
);
```

## 2.14 Archive

**Relational Model:** Archive(item\_id, archived\_date)

**Functional Dependencies:** {(item\_id → archived\_date)}

**Candidate Keys:** {(item\_id)}

**Primary Key:** {(item\_id)}

**Foreign Keys:** {(item\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Archive(  
  archived_date DATE NOT NULL,  
  item_id INTEGER,  
  PRIMARY KEY (item_id) AUTO_INCREMENT,  
  FOREIGN KEY (item_id) REFERENCES Actions(item_id)  
);
```

## 2.15 Board

**Relational Model:** Board(board\_id, name, description, isPublic, team\_id, user\_id, create\_date)

**Functional Dependencies:** {(board\_id → name, description, isPublic, team\_id, user\_id, create\_date)}

**Candidate Keys:** {(board\_id)}

**Primary Key:** {board\_id}

**Foreign Keys:** {(team\_id), (user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Board(  
  board_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(50) NOT NULL,  
  description VARCHAR(200) NOT NULL,  
  create_date DATE NOT NULL,  
  isPublic BOOLEAN,  
  team_id INTEGER,  
  user_id INTEGER,  
  FOREIGN KEY (team_id) REFERENCES Team(team_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id)  
);
```

## 2.16 AssignsCard

**Relational Model:** AssignsCard(card\_id, user\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(card\_id, user\_id)}

**Primary Key:** {(card\_id, user\_id)}

**Foreign Keys:** {(card\_id), (user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE AssignsCard(  
  card_id INTEGER,  
  user_id INTEGER,  
  PRIMARY KEY (card_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (card_id) REFERENCES Card(card_id)  
);
```

## 2.17 AssignsIssue

**Relational Model:** AssignsIssue(issue\_id, user\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(issue\_id, user\_id)}

**Primary Key:** {(issue\_id, user\_id)}

**Foreign Keys:** {(issue\_id), (user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE AssignsIssue(  
  issue_id INTEGER,  
  user_id INTEGER,  
  PRIMARY KEY (issue_id, user_id),  
  FOREIGN KEY (issue_id) REFERENCES Issue(issue_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id)  
);
```

## 2.18 memberOf

**Relational Model:** memberOf(user\_id, project\_id, role)

**Functional Dependencies:** {user\_id, project\_id → role}

**Candidate Keys:** {(user\_id, project\_id)}

**Primary Key:** {(user\_id, project\_id)}

**Foreign Keys:** {(user\_id), (project\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE memberOf(  
  role VARCHAR(50),  
  user_id INTEGER,  
  project_id INTEGER,  
  PRIMARY KEY (user_id, project_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (project_id) REFERENCES Project(project_id)  
);
```

## 2.19 Retains

**Relational Model:** Retains(project\_id, team\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(project\_id, team\_id)}

**Primary Key:** {(project\_id, team\_id)}

**Foreign Keys:** {(project\_id), (team\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Retains(  
  project_id INTEGER,  
  team_id INTEGER,  
  PRIMARY KEY (project_id, team_id),
```

```

FOREIGN KEY (project_id) REFERENCES Project(project_id),
FOREIGN KEY (team_id) REFERENCES Team(team_id)
);

```

## 2.20 User

**Relational Model:** User(user\_id, name, username, password, e-mail)

**Functional Dependencies:** {(user\_id → name, username, password, e-mail), (username → user\_id), (e-mail → user\_id)}

**Candidate Keys:** {user\_id}

**Primary Key:** {user\_id}

**Foreign Keys:** {}

**Normal Form:** BCNF

**Table Definition:**

```

CREATE TABLE User(
user_id INTEGER PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(50) NOT NULL,
username VARCHAR(50) NOT NULL,
password VARCHAR(50) NOT NULL,
e-mail VARCHAR(50) NOT NULL
);

```

## 2.21 PrivilegedUser

**Relational Model:** PrivilegedUser(user\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(user\_id)}

**Primary Key:** {(user\_id)}

**Foreign Keys:** {(user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```

CREATE TABLE PrivilegedUser(
user_id INTEGER,
PRIMARY KEY (user_id),
FOREIGN KEY (user_id) REFERENCES User(user_id)
);

```

## 2.22 StandardUser

**Relational Model:** StandardUser(user\_id)



**Functional Dependencies:** {}

**Candidate Keys:** {user\_id}

**Primary Key:** {user\_id}

**Foreign Keys:** {(user\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE StandardUser(  
  user_id INTEGER,  
  PRIMARY KEY (user_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id)  
);
```

## 2.23 Issue

**Relational Model:** Issue(issue\_id, name, status, start\_date, end\_date, priority, note, attachment\_id, item\_id)

**Functional Dependencies:** {issue\_id → name, status, start\_date, end\_date, priority, note}

**Candidate Keys:** {issue\_id}

**Primary Key:** {issue\_id}

**Foreign Keys:** {(attachment\_id), (item\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Issue(  
  issue_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(50) NOT NULL,  
  status VARCHAR(50) NOT NULL,  
  start_date DATE NOT NULL,  
  end_date DATE,  
  priority INTEGER,  
  note VARCHAR(250),  
  attachment_id INTEGER,  
  item_id INTEGER,  
  FOREIGN KEY (attachment_id) REFERENCES Attachment(attachment_id),  
  FOREIGN KEY (item_id) REFERENCES Actions(item_id),  
  CHECK (priority >= 0)  
);
```

## 2.24 Contributes

**Relational Model:** Contributes(user\_id, team\_id, role)

**Functional Dependencies:** {(user\_id, team\_id) → role }

**Candidate Keys:** {(user\_id)}

**Primary Key:** {(user\_id, team\_id)}

**Foreign Keys:** {(user\_id), (team\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Contributes(  
  user_id INTEGER,  
  team_id INTEGER,  
  role VARCHAR(50),  
  PRIMARY KEY (user_id, team_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (team_id) REFERENCES Team(team_id)  
);
```

## 2.25 AssignsIssues

**Relational Model:** AssignsIssue(user\_id, issue\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(user\_id, issue\_id)}

**Primary Key:** {(user\_id, issue\_id)}

**Foreign Keys:** {(user\_id), (issue\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE AssignsIssues(  
  user_id INTEGER,  
  issue_id INTEGER,  
  PRIMARY KEY (user_id, issue_id),  
  FOREIGN KEY (user_id) REFERENCES User(user_id),  
  FOREIGN KEY (issue_id) REFERENCES Team(issue_id)  
);
```

## 2.26 Comment

**Relational Model:** Comment(item\_id, comment)

**Functional Dependencies:** {(item\_id → comment)}

**Candidate Keys:** {(item\_id)}

**Primary Key:** {(item\_id)}

**Foreign Keys:** {(item\_id)}

**Normal Form:** BCNF

**Table Definition:**

```
CREATE TABLE Comment(  
  item_id INTEGER,
```

```
comment VARCHAR(250),  
PRIMARY KEY (item_id),  
FOREIGN KEY (item_id) REFERENCES Comment(item_id)  
);
```

## 2.27 Mention

**Relational Model:**Comment(item\_id, user\_id)

**Functional Dependencies:** {}

**Candidate Keys:** {(item\_id, user\_id)}

**Primary Key:** {(item\_id, user\_id)}

**Foreign Keys:** {(item\_id, user\_id)}

**Normal Form:** BCNF

**Table Definition:**

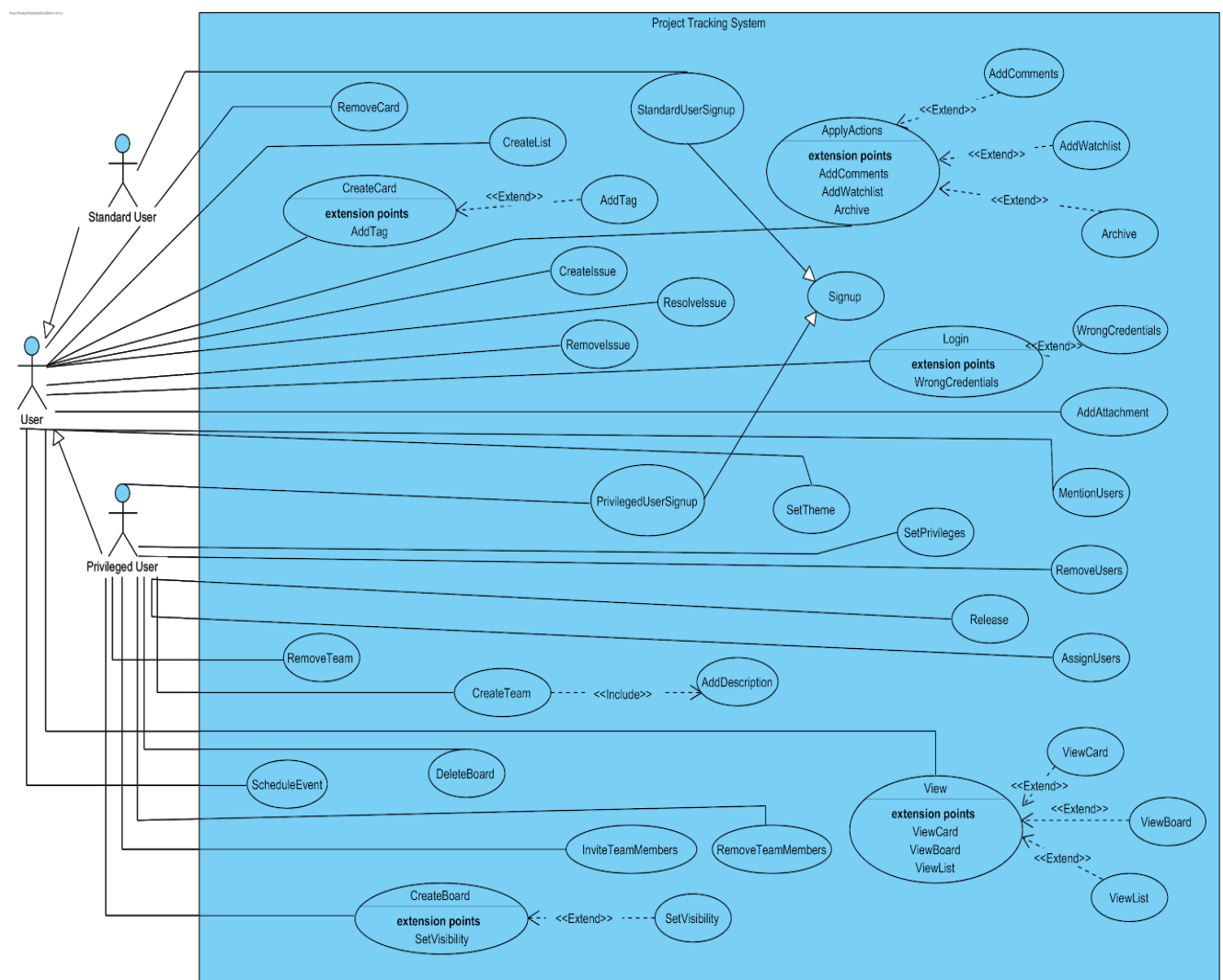
```
CREATE TABLE Mention(  
item_id INTEGER,  
user_id INTEGER,  
PRIMARY KEY (item_id, user_id),  
FOREIGN KEY (item_id) REFERENCES Comment(item_id),  
FOREIGN KEY (user_id) REFERENCES User(user_id),  
);
```

### 3. Functional dependencies and normalization

In the previous section, we have given functional dependencies of the tables. All of the tables are designed to be in BCNF. Thus, we did not perform any decomposition or normalization on the tables.

## 4. Functional Components

### 4.1 Use Cases / Scenarios



#### **4.1.1 CreateBoard**

**UseCase:** A Privileged User Creates a Board

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to create a board

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must be on the creation screen.

**Successful Scenario Event Flow for CreateBoard:**

1. A privileged user clicks the “Create Board” button, and is directed to another panel.
2. In this panel, user is prompted to set visibility, set name and type in description.
3. A board is created
4. The board is displayed in the list view of boards.

**Unsuccessful Scenario Event Flow for CreateBoard:**

1. A privileged user clicks the “Create Board” button, and is directed to another panel.
2. In this panel, user is prompted to set visibility, set name and type in description.
3. User receives an error message notifying that the board creation is unsuccessful.
4. User returns to the creation screen

#### **4.1.2 CreateTeam**

**Use Case:** A Privileged User Creates a Team

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to create a team

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must be on the creation screen.

**Successful Scenario Event Flow for CreateTeam:**

1. A privileged user clicks the “Create Team” button, and is directed to another panel.
2. In this panel, user is prompted type in the title and description.
3. A team with one member, the creator, is created.
4. The team is displayed in the list view of teams.

**Unsuccessful Scenario Event Flow for CreateTeam:**

1. A privileged user clicks the “Create Team” button, and is directed to another panel.
2. In this panel, user is prompted type in the title and description.
3. An error message is displayed to the privileged user, saying the team is unable to be created.
4. Privileged user is returned back to creation screen.

### **4.1.3 InviteTeamMembers**

**Use Case:** A Privileged User Invites Another User to a Team

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to add a team member to a team.

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must be on the creation screen.

**Successful Scenario Event Flow for InviteTeamMembers:**

1. A privileged user clicks the “Add Member” button, and a pop up appears.
2. In this pop up, the user enters the username of the user he/she is willing to add to the team.
3. A team member is added and shown in the corresponding team.

**Unsuccessful Scenario Event Flow for InviteTeamMembers:**

1. A privileged user clicks the “Add Member” button, and a pop up appears.
2. In this pop up, the user enters the username of the user he/she is willing to add to the team.
3. User receives an error message that shows the user cannot be added to the team.
4. The pop up cancels, the user is returned to the creation screen.

### **4.1.4 SetPrivileges**

**Use Case:** A Privileged User Sets Privilege of a User of a Team

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to add a team member to a team or changed the privilege of another user.

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must be on the creation screen.
- Privileged User must either be adding a user to a team or modifying the privilege of another team member that is already present in a team.

**Scenario 1**

**Successful Scenario Event Flow for SetPrivileges:**

1. A privileged user clicks the “Add Member” button, and a pop up appears.
2. In this pop up, the user enters the username of the user he/she is willing to add to the team.
3. A team member is added and shown in the corresponding team.

**Unsuccessful Scenario Event Flow for SetPrivileges:**

1. A privileged user clicks the “Add Member” button, and a pop up appears.
2. In this pop up, the user enters the username of the user he/she is willing to add to the team.
3. User receives an error message that shows the user cannot be added to the team.
4. The pop up cancels, the user is returned to the creation screen.

## Scenario 2

### Successful Scenario Event Flow for SetPrivileges:

1. A privileged user clicks to a user photo inside a team. An information panel is opened and the user is shown the information of the corresponding user.
2. The privileged user clicks to the radio button according to the privilege property of the user.
3. The privilege property of the corresponding user is changed.

### Unsuccessful Scenario Event Flow for SetPrivileges:

1. A privileged user clicks to a user photo inside a team. An information panel is opened and the user is shown the information of the corresponding user.
2. The privileged user clicks to the radio button according to the privilege property of the user.
3. An error message is displayed prompting the action is unsuccessful.
4. The user is returned to the creation screen.

## 4.1.5 CreateList

**Use Case:** A User Creates a List Inside a Board

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to create a list inside the board view screen.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be on the board view screen.

### Successful Scenario Event Flow for CreateList:

1. A user clicks the "Create List" button, and a pop up appears.
2. In this pop up, the user enters the title of the newly created list.
3. A list is added to the board screen vertical layout.

### Unsuccessful Scenario Event Flow for CreateList:

1. A user clicks the "Create List" button, and a pop up appears.
2. In this pop up, the user enters the title of the newly created list.
3. User receives an error message that shows the list cannot be created.
4. The pop up cancels, the user is returned to the board view screen.

## 4.1.6 CreateCard

**Use Case:** A User Creates a Card Inside a List

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to create a card inside a list in the board.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be on the board view screen.

- User must have at least one list to create the card inside.

#### **Successful Scenario Event Flow for CreateCard:**

1. A user clicks the “Create Card” button, and a pop up appears.
2. In this pop up, the user enters the title of the newly created card.
3. The user needs to pick a color for the card.
- 3.1. The user can provide additional information as description, priority start date and end date.
4. A card is added inside a list.

#### **Unsuccessful Scenario Event Flow for CreateCard:**

1. A user clicks the “Create Card” button, and a pop up appears.
2. In this pop up, the user enters the title of the newly created card.
3. The user needs to pick a color for the card.
- 3.1. The user can provide additional information as description, priority start date and end date.
4. User receives an error message that shows the card cannot be created.
5. The pop up cancels, the user is returned to the board view screen.

### **4.1.7 AssignUsers**

#### **Scenario 1**

**Use Case:** A Privileged User Assigns a Card to another User

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to assign another user to a card.

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must own or belong to a project.
- Privileged User must be on the board view screen.
- The will-be-assigned user must be in the same project and board with the assigning user.

#### **Successful Scenario Event Flow for AssignUsers:**

1. A privileged user clicks on a Card, then is directed to another screen.
2. The user clicks the “Assign” button on the right hand side of the screen in order to list the possible users.
3. The user clicks on the photo of the will-be-assigned user.
4. The privileged user successfully assigns a card to another user.

#### **Unsuccessful Scenario Event Flow for AssignUsers:**

1. A privileged user clicks on a Card, then is directed to another screen.
2. The user clicks the “Assign” button on the right hand side of the screen in order to list the possible users.
3. The user clicks on the photo of the will-be-assigned user.
4. The User receives an error message saying that the card cannot be assigned to the following user.
5. The user is directed back to the same screen.

#### **Scenario 2**

**Use Case:** A Privileged User Assigns an Issue to another User



**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged User who wants to assign another user to a card.

**Pre-conditions:**

- Privileged User must be logged in to the system.
- Privileged User must own or belong to a project.
- Privileged User must be on the board view screen.
- The will-be-assigned user must be in the same project and board with the assigning user.

**Successful Scenario Event Flow for AssignUsers:**

1. A privileged user clicks on “Go to Issues” button, then is directed to another screen.
2. The user clicks the corresponding issue specified with its title, and is able to see information about the issue.
3. The user clicks on the “Assigned” and a pop up to change the assigned user appears.
4. The user selects the photo of a proper user to assign the issue.
5. The privileged user successfully assigns an issue to another user.

**Unsuccessful Scenario Event Flow for AssignUsers:**

1. A privileged user clicks on “Go to Issues” button, then is directed to another screen.
2. The user clicks the corresponding issue specified with its title, and is able to see information about the issue.
3. The user clicks on the “Assigned” and a pop up to change the assigned user appears.
4. The user selects the photo of a proper user to assign the issue.
5. The User receives an error message saying that the issue cannot be assigned to the following user.
6. The user is directed back to the same screen.

#### **4.1.8 RemoveCard**

**Use Case:** A User Removes a Card from a List

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to delete a card.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be on the board view screen.

**Successful Scenario Event Flow for RemoveCard:**

1. A user clicks on a Card, then is directed to another screen.
2. The user clicks on the “Remove This Card” button to remove the card.
3. If the card state is active user will be asked to confirm their choice else no confirmation required.
4. The user successfully removes the card.

**Unsuccessful Scenario Event Flow for RemoveCard:**

1. A user clicks on a Card, then is directed to another screen.
2. The user clicks on the “Remove This Card” button to remove the card.

3. The User receives an error message saying that the card cannot be removed.
4. The user is directed back to the same screen, the card information screen.

#### **4.1.9 ArchiveCard**

**Use Case:** A User Archives a Card

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to archive a card.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be on the board view screen.

**Successful Scenario Event Flow for ArchiveCard:**

1. A user clicks on a Card, then is directed to another screen.
2. The user clicks on the “Archive This Card” button to archive the card.
3. The user successfully archives the card.

**Unsuccessful Scenario Event Flow for ArchiveCard:**

1. A user clicks on a Card, then is directed to another screen.
2. The user clicks on the “Archive This Card” button to archive the card.
3. The User receives an error message saying that the card cannot be archived.
4. The user is directed back to the same screen, the card information screen.

#### **4.1.10 AddComments**

**Scenario 1**

**Use Case:** A User Adds a Comment on a Card

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to add a comment on a Card.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be inside a specific card’s screen.

**Successful Scenario Event Flow for AddComments:**

1. The user clicks on the comment box and types in the will-be-added comment.
2. The user clicks on the “Add Comment” button.
3. The user successfully adds the comment to the card.
4. The user is directed back to the board view.

**Unsuccessful Scenario Event Flow for AddComments:**

1. The user clicks on the comment box and types in the will-be-added comment.
2. The user clicks on the “Add Comment” button.
3. The User receives an error message saying that the comment cannot be added.
4. The user is directed back to the same screen, the card information screen.

## **Scenario 2**

**Use Case:** A User Adds a Comment on an Issue

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to add a comment on an Issue.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be inside a specific issue screen.

**Successful Scenario Event Flow for AddComments:**

1. The user clicks on the comment box and types in the will-be-added comment.
2. The user clicks on the “Add Comment” button.
3. The user successfully adds the comment to the issue.
4. The user is directed back to the board view.

**Unsuccessful Scenario Event Flow for AddComments:**

1. The user clicks on the comment box and types in the will-be-added comment.
2. The user clicks on the “Add Comment” button.
3. The User receives an error message saying that the comment cannot be added.
4. The user is directed back to the same screen, the issue information screen.

## **4.1.11 MentionUsers**

### **Scenario 1**

**Use Case:** A User Adds a Mention on a Card

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to add a mention on a Card, mentioning another user.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be inside a specific card’s screen.

**Successful Scenario Event Flow for MentionUsers:**

1. The user clicks on the comment box and types the “@” character and a username nearby to mention a user.
2. The user successfully mentions another user on the card.

**Unsuccessful Scenario Event Flow for MentionUsers:**

1. The user clicks on the comment box and types the “@” character and a username nearby to mention a user.
2. The name of the will-be-mentioned user does not appear.

#### **4.1.12 Release**

##### **Scenario 1**

**Use Case:** A Privileged User Adds a Release

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- Privileged user who wants to request a release.

**Pre-conditions:**

- Privileged user must be logged in to the system.
- Privileged user must own or belong to a project.
- Privileged user must be inside a specific project's release list screen.

**Successful Scenario Event Flow for Release:**

1. The user clicks on the "Add Release" button to start the process.
2. The user enters the title, start date and end date of the release.
  - 2.1. The user can also enter version number, URL, and a note.
3. The user clicks on the "Save" button to add the release with the specified information.
4. The user successfully adds a release.

**Unsuccessful Scenario Event Flow for Release:**

1. The user clicks on the "Add Release" button to start the process.
2. The user enters the title, start date and end date of the release.
  - 2.1. The user can also enter version number, URL, and a note.
3. The user clicks on the "Save" button to add the release with the specified information.
4. The user receives a pop up message saying the release cannot be added. The pop up cancels and the user is returned back to the same screen.

#### **4.1.13 CreateIssue**

**Use Case:** A User Creates an Issue

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to request a release.

**Pre-conditions:**

- User must be logged in to the system.
- User must own or belong to a project.
- User must be inside a specific project's board view screen.

**Successful Scenario Event Flow for CreateIssue:**

1. The user clicks on the "Add Issue" button to start the process.
2. The user enters the title, start date, end date and status of the issue.
  - 2.1. The user can also enter a priority index and add a note.
3. The user clicks on the "Save" button to add the create the issue with the specified information.
4. The user successfully creates an issue.

**Unsuccessful Scenario Event Flow for CreateIssue:**

1. The user clicks on the "Add Issue" button to start the process.

2. The user enters the title, start date, end date and status of the issue.
- 2.1. The user can also enter a priority index and add a note.
3. The user clicks on the “Save” button to add the create the issue with the specified information.
4. The user receives a pop up message saying the issue cannot be added. The pop up cancels and the user is returned back to the same screen.

#### **4.1.14 SetTheme**

**Use Case:** A User Sets the Theme for the Application

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants to change the theme of the application.

**Pre-conditions:**

- User must be logged in to the system.

**Successful Scenario Event Flow for SetTheme:**

1. The user clicks on the “Settings” button that is available after the login screen.
2. The settings page is opened and the user attempts to change the theme.
3. The user successfully changes the theme of the application.

**Unsuccessful Scenario Event Flow for SetTheme:**

1. The user clicks on the “Settings” button that is available after the login screen.
2. The settings page is opened and the user attempts to change the theme.
3. The user receives an error pop up message prompting that theme change operation is unsuccessful.
4. Pop up cancels and user is directed back to the settings screen.

#### **4.1.10 AddAttachment**

**Scenario 1**

**Use Case:** A User Adds an Attachment to an Card

**Primary Actor:** User

**Stakeholders and interests:**

- Users (admins,team members) who want to add a file as an attachment to the card

**Pre-conditions:**

- User should be logged in.
- User should be a team member in the project
- User should be in the Board view

**Successful Scenario Event Flow for AddAttachment :**

1. The user clicks on the “attach file” icon
2. The user chooses where to attach the file from (PC (local files))
3. The user is shown a warning about the file size and the file type supported by the system.
4. The user selects the file to be added.
5. The user clicks on the “Add attachment” button to complete the action.

**Unsuccessful Scenario Event Flow for AddAttachment :**

1. The user clicks on the “attach file” icon
2. The user chooses where to attach the file from (PC (local files))

3. The user is shown a warning about the file size and the file type supported by the system.
4. The user selects an unsupported file type with a size greater than the one allowed by the system.
5. The user receives a 'rejection' message.
6. The user is directed to step 2 again to repeat the process.

## **Scenario 2**

**Use Case:** A User Adds an Attachment to a Project

**Primary Actor:** Privileged User

**Stakeholders and interests:**

- The privileged user who wants to add an attachment to the project

**Pre-conditions:**

- User should be logged in.
- User should be a privileged member in the project.
- User should be in the Projects view.

**Successful Scenario Event Flow for AddAttachment :**

1. The user who is in the 'Projects' view clicks on the three dots (...) on the project of interest.
2. The user is shown some action icons for all actions he can perform (delete, add member, comment, add attachment).
3. The user clicks on the '+' icon to attach a file
4. The user chooses where to attach the file from (PC (local files)).
5. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
6. The user selects the file to be added.
7. The user clicks on the "Add attachment" button to complete the action.

**Unsuccessful Scenario Event Flow for AddAttachment :**

1. The user who is in the 'Projects' view clicks on the three dots (...) on the project of interest.
2. The user is shown some action icons for all actions he can perform (delete, add member, comment, add attachment).
3. The user clicks on the '+' icon to attach a file
4. The user chooses where to attach the file from (PC (local files)).
5. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
6. The user selects an unsupported file type with a size greater than the one allowed by the system.
7. The user receives a 'rejection' message.
8. The user is directed to step 4 again to repeat the process.

## **Scenario 3**

**Use Case:** A User Adds an Attachment to an Issue

**Primary Actor:** User

**Stakeholders and interests:**

- The user who wants to add a file to support the issue statement

**Pre-conditions:**

- User should be logged in.
- User should be a team member in the project
- User should be in the Board and then Issues view

**Successful Scenario Event Flow for AddAttachment :**

1. The user clicks on the 'Go To Issues' button from the 'Board view'
2. The user clicks on one of the issues from the issue list or clicks on the 'Add issue' button on the left sidebar of the view
3. The user is shown the right sidebar with all the empty/filled fields of the issue (new issue or a previously created one).
4. The user clicks on the attachment icon
5. The user chooses where to attach the file from (PC (local files)).
6. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
7. The user selects the file to be added.
8. The user clicks on the "Add attachment" button to complete the action.

**Unsuccessful Scenario Event Flow for AddAttachment :**

1. The user clicks on the 'Go To Issues' button from the 'Board view'
2. The user clicks on the 'Add issue' button on the left sidebar of the view
3. The user is shown the right sidebar with all the empty fields to be filled in for the issue.
4. The user clicks on the attachment icon
5. The user chooses where to attach the file from (PC (local files)).
6. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
7. The user selects an unsupported file type with a size greater than the one allowed by the system.
8. The user receives a 'rejection' message.
9. The user is directed to step 5 again to repeat the process.

**Scenario 4**

**Use Case:** A User Adds an Attachment to a Release

**Primary Actor:** User

**Stakeholders and interests:**

- The user who wants to add an attachment to a release

**Pre-conditions:**

- The user should be logged in.
- The user should be/ should have been a member in a project (i.e: the user should see some projects on his/her projects page).
- The user should be privileged to upload an attachment when a release is created.
- The user should be in the Boards view.

**Successful Scenario Event Flow for AddAttachment :**

1. The user clicks on the 'Go to Releases' in the 'Boards view'
2. The user clicks on one of the releases from the listed releases or clicks on the 'Add Release' button on the left sidebar of the view

3. The user is shown the right sidebar with all the empty/filled fields of the release (new release or a previously created one).
4. The user clicks on 'Edit' button to be able to make changes (add an attachment) to a previously created release.
5. The user clicks on the attachment icon
6. The user chooses where to attach the file from (PC (local files))
7. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
8. The user selects the file to be added.
9. The user clicks on the "Add attachment" button to complete the action.

#### **Unsuccessful Scenario Event Flow for AddAttachment :**

1. The user clicks on the 'Go to Releases' in the 'Boards view'
2. The user clicks on one of the releases from the listed releases or clicks on the 'Add Release' button on the left sidebar of the view
3. The user is shown the right sidebar with all the empty/filled fields of the release (new release or a previously created one).
4. The user clicks on 'Edit' button to be able to make changes (add an attachment) to a previously created release.
5. The user clicks on the attachment icon
6. The user chooses where to attach the file from (PC (local files)).
7. The user is shown a dialog box and a warning about the file size and the file type supported by the system.
8. The user selects an unsupported file type with a size greater than the one allowed by the system.
9. The user receives a 'rejection' message.
10. The user is directed to step 6 again to repeat the process.

### **4.1.11 Login**

**Use Case:** User logs into the system

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants use the project tracking software.

**Pre-conditions:**

- User must have an account and registered to the system.

#### **Successful Scenario Event Flow for Login:**

1. The user chooses the login option from the homepage.
2. The user enters their credentials consisting of their username and password.
3. The user clicks on the login button and wait for the system to verify their credentials.
4. After the verification the user is directed to the projects page.

#### **Unsuccessful Scenario Event Flow for Login:**

1. The user chooses the login option from the homepage.
2. The user enters their credentials consisting of their username and password.
3. The user clicks on the login button and wait for the system to verify their credentials.
4. The user receives an error message saying that their credentials are invalid.



5. The user is directed back to the same screen.

#### **4.1.11 SignUp**

**Use Case:** User registers into the system

**Primary Actor:** User

**Stakeholders and interests:**

- User who wants use the project tracking software.

**Pre-conditions:**

- User must not have an account and not registered to the system.

**Successful Scenario Event Flow for Login:**

1. The user chooses the sign up option from the homepage.
2. The user enters their credentials consisting of their username, name, email, and password.
3. The user selects a theme from the given options in the screen.
4. The user chooses their type privileged or standard.
5. The user clicks on the sign up button and wait for the system to verify their credentials.
6. After the verification the user is directed to the projects page.

**Unsuccessful Scenario Event Flow for Login:**

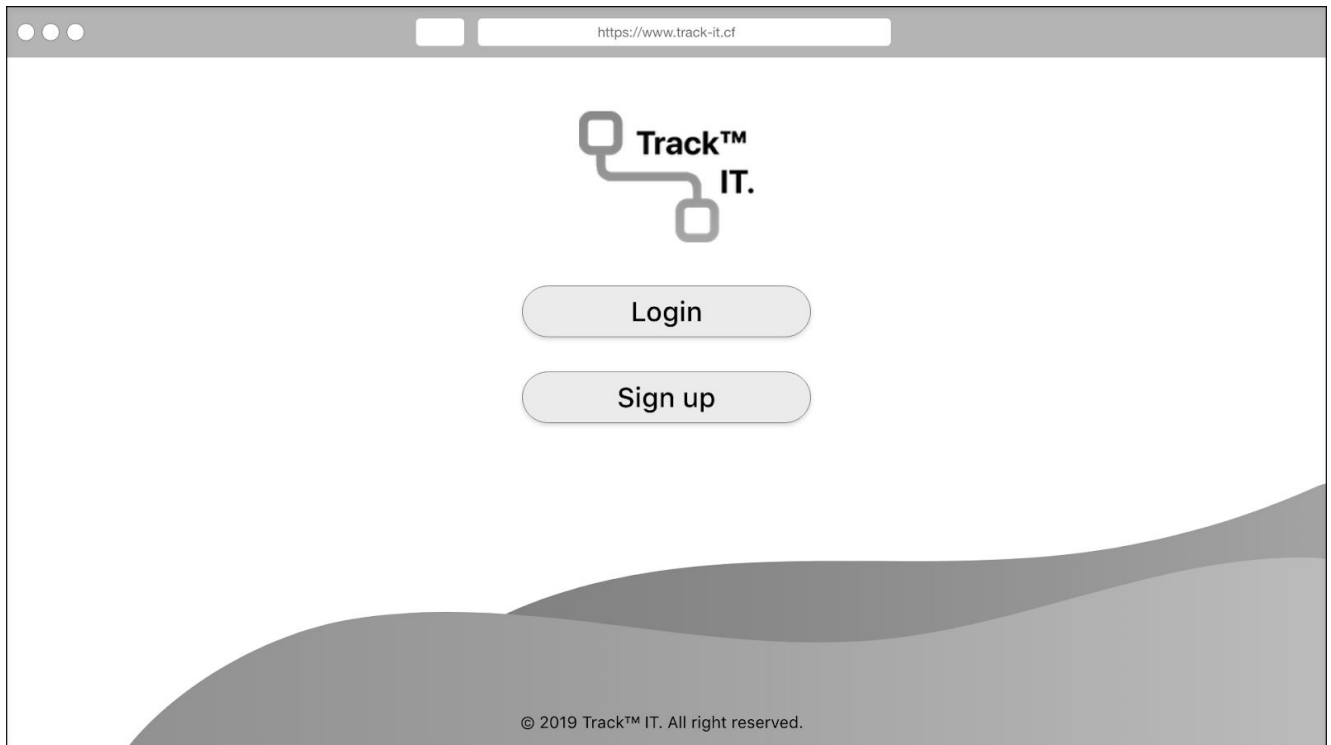
1. The user chooses the sign up option from the homepage.
2. The user enters their credentials consisting of their username, name, email, and password.
3. The user selects a theme from the given options in the screen.
4. The user chooses their type privileged or standard.
5. The user clicks on the sign up button and wait for the system to verify their credentials.
6. The user receives an error message saying that their credentials are invalid.
7. The user is directed back to the same screen.

## 5. User Interface Design and SQL Statements

---

### 5.1 General Pages

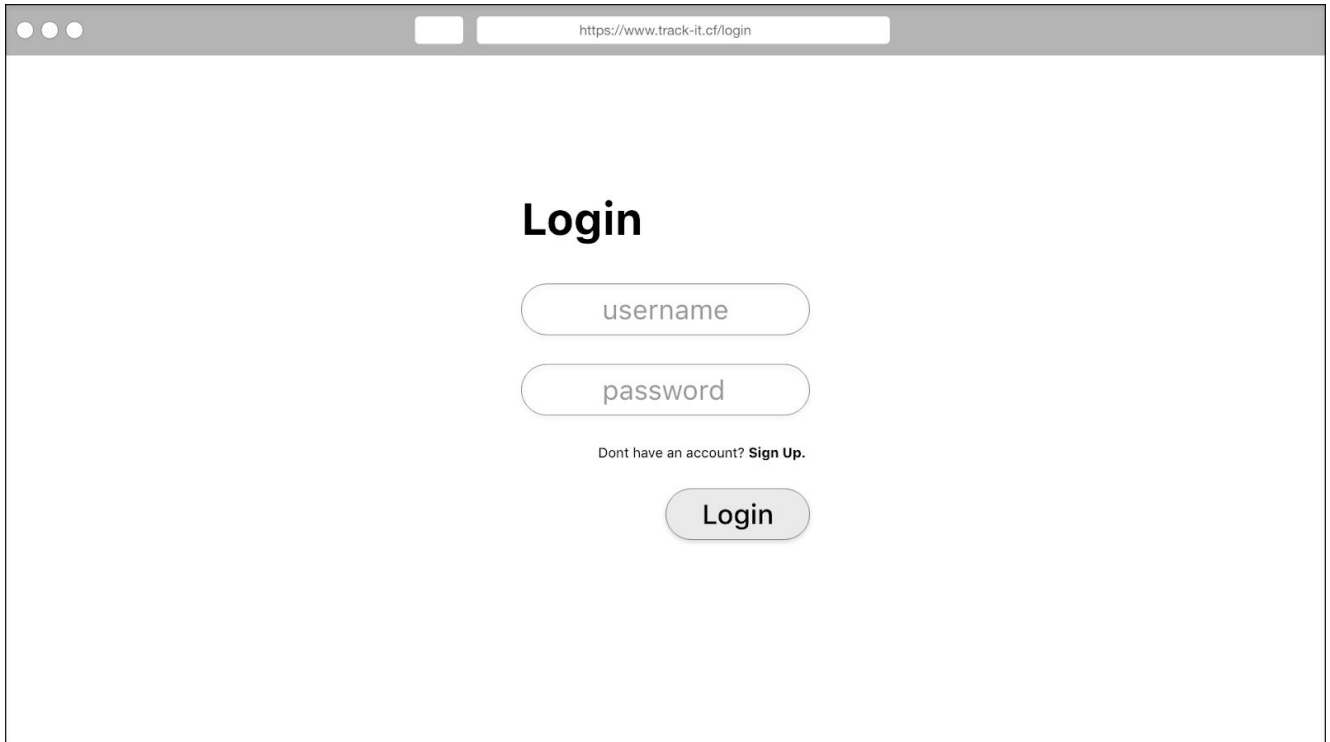
#### 5.1.1 Home Page (without login)



**Figure 1:** Home Page

In this page there is no database query needed because all needed information is embedded into html.

### 5.1.2 Login Page (for both user type)



https://www.track-it.cf/login

## Login

username

password

Dont have an account? [Sign Up.](#)

Login

**Figure 2:** Login Page

#### Login with credentials

```
SELECT username, password FROM User
WHERE username = "johnappleseed" AND password = "123456";
```

### 5.1.3 Sign Up Page (for both user type)

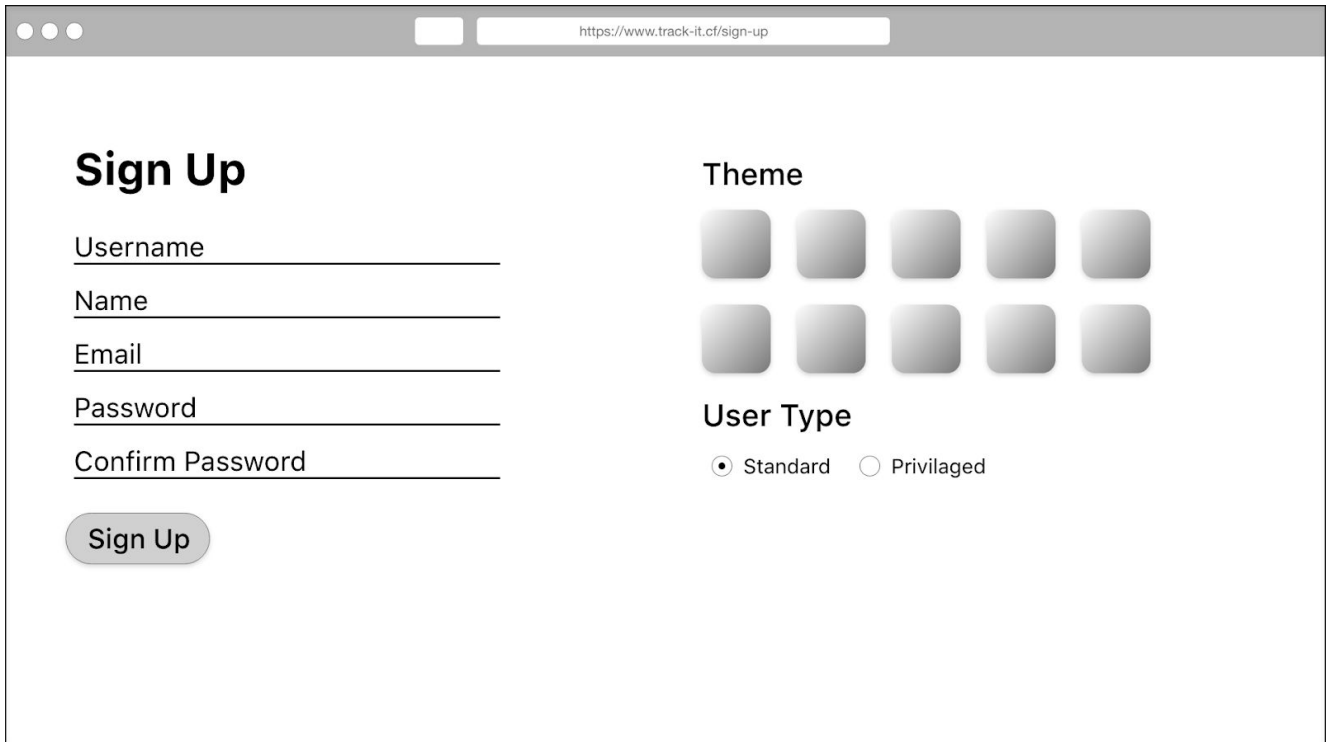


Figure 3: Sign Up Page

#### Sign up for the application

```
INSERT INTO User(name, username, password, e-mail) VALUES ('trackit_name',  
'trackit', 'trackit123', 'mail@trackit.com');
```

```
INSERT INTO Theme(user_id, name, color_palette, background) VALUES (1,  
'trackit', 'blue', 'http://track-it.cf/assets/img/logo.png');
```

#### If Privileged is selected:

```
INSERT INTO PrivilegedUser(user_id) VALUES (1);
```

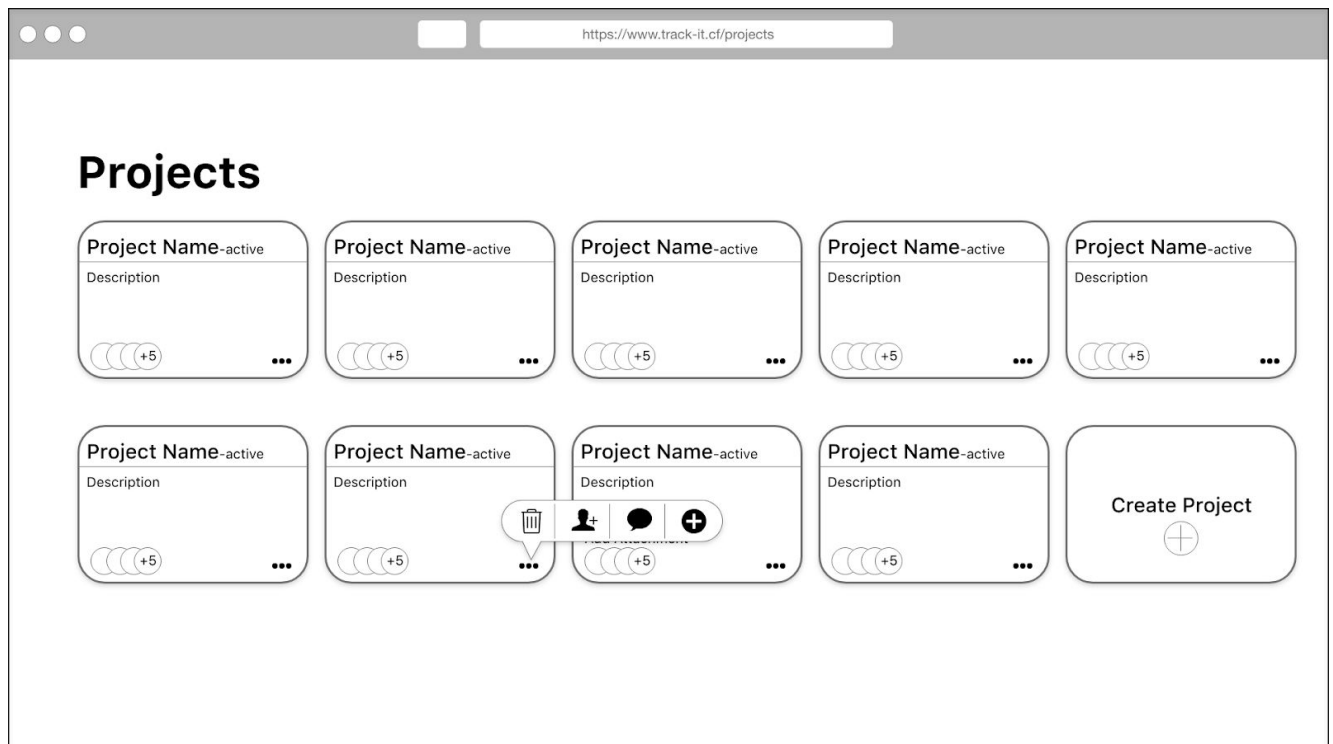
#### If Standard is selected:

```
INSERT INTO StandardUser(user_id) VALUES (1);
```

**Note:** User types are checked beforehand for these screens, so that some queries are special for specified user.

## 5.2 Project Pages

### 5.2.1 Projects Page



**Figure 4:** Projects Page

## 5.2.2 Create Project & Team

https://www.track-it.cf/create-project

Continue

### Create Project

**Project name:** Project 1

**Status:** Active

**Description:** My project

**Start Date:** 5.4.19

#### Users

Search user

Username Invite

Username Invite

Username Invite

### Team

**Team name:** Engineering Team

**Description:** Description

**Members:**

Standard Privileged Standard Privileged Standard Privileged

Standard Privileged Standard Privileged Standard Privileged

Standard Privileged Standard Privileged Standard Privileged

**My Team** ▼

**Team 2** ▼

**Team 3** ▼

Figure 5: Create Project & Team Page

To get already created projects (privileged user):

```
SELECT * FROM Project;
```

Create new project (privileged user):

```
INSERT INTO Project (name, status, end_date, start_date, description) VALUES  
( 'Project 1', '20191201', 'Sprint 3', GETDATE(), 'description123', 1);
```

Delete project (privileged user):

```
DELETE FROM Attachment AS A1 WHERE A1.release_id IN (SELECT release_id FROM  
Release AS R1 WHERE R1.project_id = 1);
```

```
DELETE FROM Release WHERE project_id = 1;
```

```
DELETE FROM Attachment WHERE project_id = 1;
```

```
DELETE FROM MemberOf WHERE project_id = 1;
```

```
DELETE FROM assignIssue WHERE issue_id IN (SELECT issue_id FROM Issue WHERE project_id = 1);
```

```
DELETE FROM IssueTag WHERE issue_id IN (SELECT issue_id FROM Issue WHERE project_id = 1);
```

```
DELETE FROM Attachment AS A1 WHERE A1.issue_id IN (SELECT issue_id FROM Issue WHERE project_id = 1);
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Issue NATURAL JOIN Watchlist WHERE project_id = 1);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Issue NATURAL JOIN Archive WHERE project_id = 1);
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Issue NATURAL JOIN Comment WHERE project_id = 1);
```

```
DELETE FROM Actions WHERE item_id IN (SELECT item_id FROM Issue NATURAL JOIN Actions WHERE project_id = 1);
```

```
DELETE FROM Issue WHERE project_id = 1;
```

```
DELETE FROM retains WHERE project_id = 1;
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Actions WHERE project_id = 1);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Actions WHERE project_id = 1);
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Actions WHERE project_id = 1);
```

```
DELETE FROM Actions WHERE project_id = 1;
```

```
DELETE FROM Board WHERE team_id IN (SELECT team_id FROM Team WHERE project_id = 1);
```

```
DELETE FROM List WHERE board_id IN (SELECT board_id FROM Board WHERE team_id IN (SELECT team_id FROM Team WHERE project_id = 1));
```

```
DELETE FROM assignsCard AS A WHERE card_id IN (SELECT card_id FROM Card
NATURAL JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN
assignsCard WHERE project_id = 1);
```

```
DELETE FROM Attachment AS A WHERE attachment_id IN (SELECT attachment_id FROM
Card NATURAL JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN
Attachment WHERE project_id = 1);
```

```
DELETE FROM CardTag AS C WHERE card_id IN (SELECT card_id FROM Card NATURAL
JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN CardTag WHERE
project_id = 1);
```

```
DELETE FROM Comment AS C WHERE card_id IN (SELECT card_id FROM Card NATURAL
JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN Action NATURAL
JOIN Comment WHERE project_id = 1);
```

```
DELETE FROM Archive AS A WHERE card_id IN (SELECT card_id FROM Card NATURAL
JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN Action NATURAL
JOIN Archive WHERE project_id = 1);
```

```
DELETE FROM Watchlist AS W WHERE card_id IN (SELECT card_id FROM Card
NATURAL JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN Action
NATURAL JOIN Watchlist WHERE project_id = 1);
```

```
DELETE FROM Action AS A WHERE card_id IN (SELECT card_id FROM Card NATURAL
JOIN List NATURAL JOIN Board NATURAL JOIN Team NATURAL JOIN Action WHERE
project_id = 1);
```

```
DELETE FROM Card AS C WHERE card_id IN (SELECT card_id FROM Card NATURAL
JOIN List NATURAL JOIN Board NATURAL JOIN Team WHERE project_id = 1);
```

```
DELETE FROM Project WHERE project_id = 1;
```

### **To get already created teams:**

```
SELECT * FROM Team WHERE team_id IN (SELECT team_id FROM contributes WHERE
user_id = 4) OR EXISTS(SELECT * FROM PrivilegedUser WHERE user_id = 4);
```

### **Create new team (privileged user):**



```
INSERT INTO Team (name, description, formation_date) VALUES ('Team 1',  
'description1234', GETDATE());
```

```
INSERT INTO contributes(user_id, team_id, role) VALUES (1, 2,  
'GrandeAuthorite');
```

```
INSERT INTO retains(project_id,team_id) VALUES(1, 2);
```

### Delete team (privileged user):

```
DELETE FROM contributes WHERE team_id = 2;
```

```
DELETE FROM retains WHERE team_id = 2;
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Comment WHERE team_id  
= 2);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Archive WHERE team_id  
= 2);
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Watchlist WHERE  
team_id = 2);
```

```
DELETE FROM Actions WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN Actions WHERE team_id = 2);
```

```
DELETE FROM Actions WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN Actions WHERE team_id = 2);
```

```
DELETE FROM Attachment WHERE attachment_id IN (SELECT attachment_id FROM Board  
NATURAL JOIN List NATURAL JOIN Card NATURAL JOIN Attachment WHERE team_id =  
2);
```

```
DELETE FROM assignsCard WHERE card_id IN (SELECT card_id FROM Board NATURAL  
JOIN List NATURAL JOIN Card NATURAL JOIN assignsCard WHERE team_id = 2);
```

```
DELETE FROM CardTag WHERE card_id IN (SELECT card_id FROM Board NATURAL JOIN  
List NATURAL JOIN Card NATURAL JOIN CardTag WHERE team_id = 2);
```

```
DELETE FROM Card WHERE card_id IN (SELECT card_id FROM Board NATURAL JOIN List
```

```
NATURAL JOIN Card WHERE team_id = 2);
```

```
DELETE FROM List WHERE list_id IN (SELECT list_id FROM Board NATURAL JOIN  
List WHERE team_id = 2);
```

```
DELETE FROM Board WHERE team_id = 2;
```

```
DELETE FROM Team WHERE team_id = 2;
```

**Show users who potentially will be assigned**(user suggestion when 3 character appears):

```
SELECT username FROM User WHERE username = 'tra';
```

**Assign user to team**(privileged user):

```
INSERT INTO contributes(user_id, team_id, role) VALUES (2, 2, 'user');
```

**If Privileged is selected:**

```
INSERT INTO PrivilegedUser(user_id) VALUES (2);
```

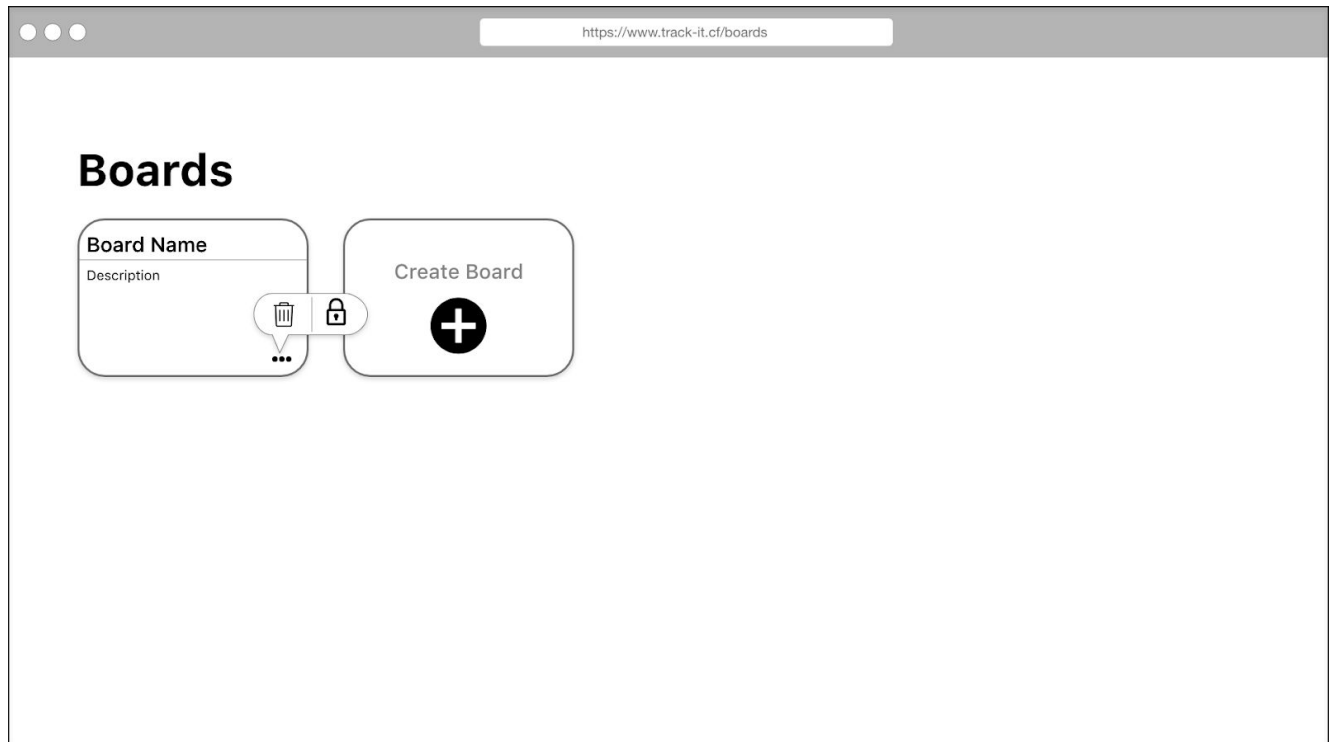
**If Standard is selected:**

```
INSERT INTO StandardUser(user_id) VALUES (2);
```

**To get already created boards:**

```
SELECT * FROM Board WHERE team_id IN (SELECT team_id FROM contributes WHERE  
user_id = 4) OR EXISTS(SELECT * FROM PrivilegedUser WHERE user_id = 4);
```

### 5.2.3 Boards Page



**Figure 6:** Boards Page

**Create board**(privileged user):

```
INSERT INTO Board(name, description, isPublic, team_id,user_id, create_date)
VALUES ('board 1', 'description123', FALSE, 2, GETDATE());
```

**Delete board**(privileged user):

```
DELETE FROM Mentioned WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Comment WHERE
board_id = 1);
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Comment WHERE
board_id = 1);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Archive WHERE
board_id = 1);
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN Actions NATURAL JOIN Watchlist WHERE
board_id = 1);
```

```
DELETE FROM Actions WHERE item_id IN (SELECT item_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN Actions WHERE board_id = 1);
```

```
DELETE FROM Attachment WHERE attachment_id IN (SELECT attachment_id FROM Board
NATURAL JOIN List NATURAL JOIN Card NATURAL JOIN Attachment WHERE board_id =
1);
```

```
DELETE FROM assignsCard WHERE card_id IN (SELECT card_id FROM Board NATURAL
JOIN List NATURAL JOIN Card NATURAL JOIN assignsCard WHERE board_id = 1);
```

```
DELETE FROM CardTag WHERE card_id IN (SELECT card_id FROM Board NATURAL JOIN
List NATURAL JOIN Card NATURAL JOIN CardTag WHERE board_id = 1);
```

```
DELETE FROM Card WHERE card_id IN (SELECT card_id FROM Board NATURAL JOIN List
NATURAL JOIN Card WHERE board_id = 1);
```

```
DELETE FROM List WHERE list_id IN (SELECT list_id FROM Board NATURAL JOIN
List WHERE board_id = 1);
```

```
DELETE FROM Board WHERE board_id = 1;
```

## 5.2.4 Board Content Page

**Note:** Boards, Issues and Releases are concurrently next to each other and could be scrolled horizontally from one page to another.

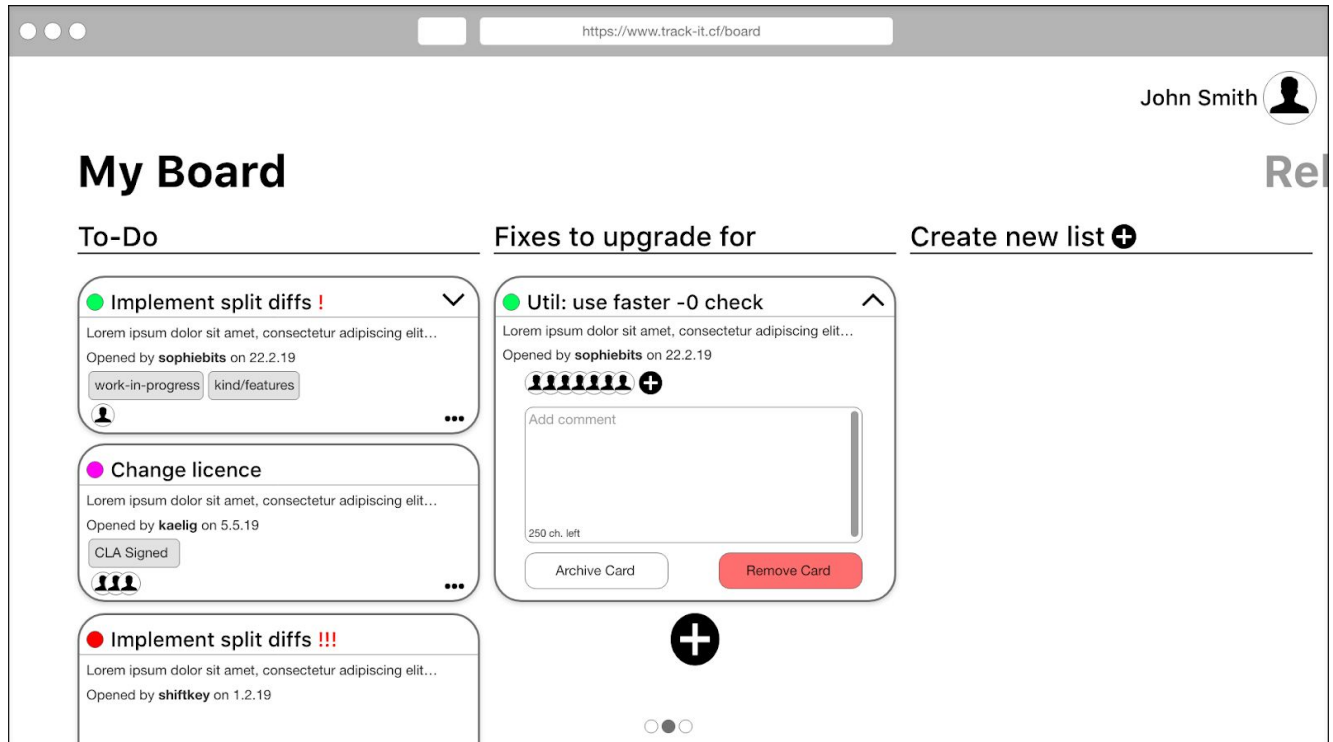


Figure 7: Board View Page Showing Lists and Cards

### Create Lists into boards:

```
INSERT INTO Lists (list_id, title, board_id) VALUES ('1234', 'To-Do', '1357');
```

### Add Cards to a list:

```
INSERT INTO Card (card_id, title, description, priority, status, note,
start_date, end_date, user_id, list_id) VALUES ('1234', 'Task One', 'This is a
task', 3, 'Active', 'Remind me', GETDATE(), NULL, '1234', 1);
```

### Add title, description and additional information to a card (check if it is you card or you are a privileged user):

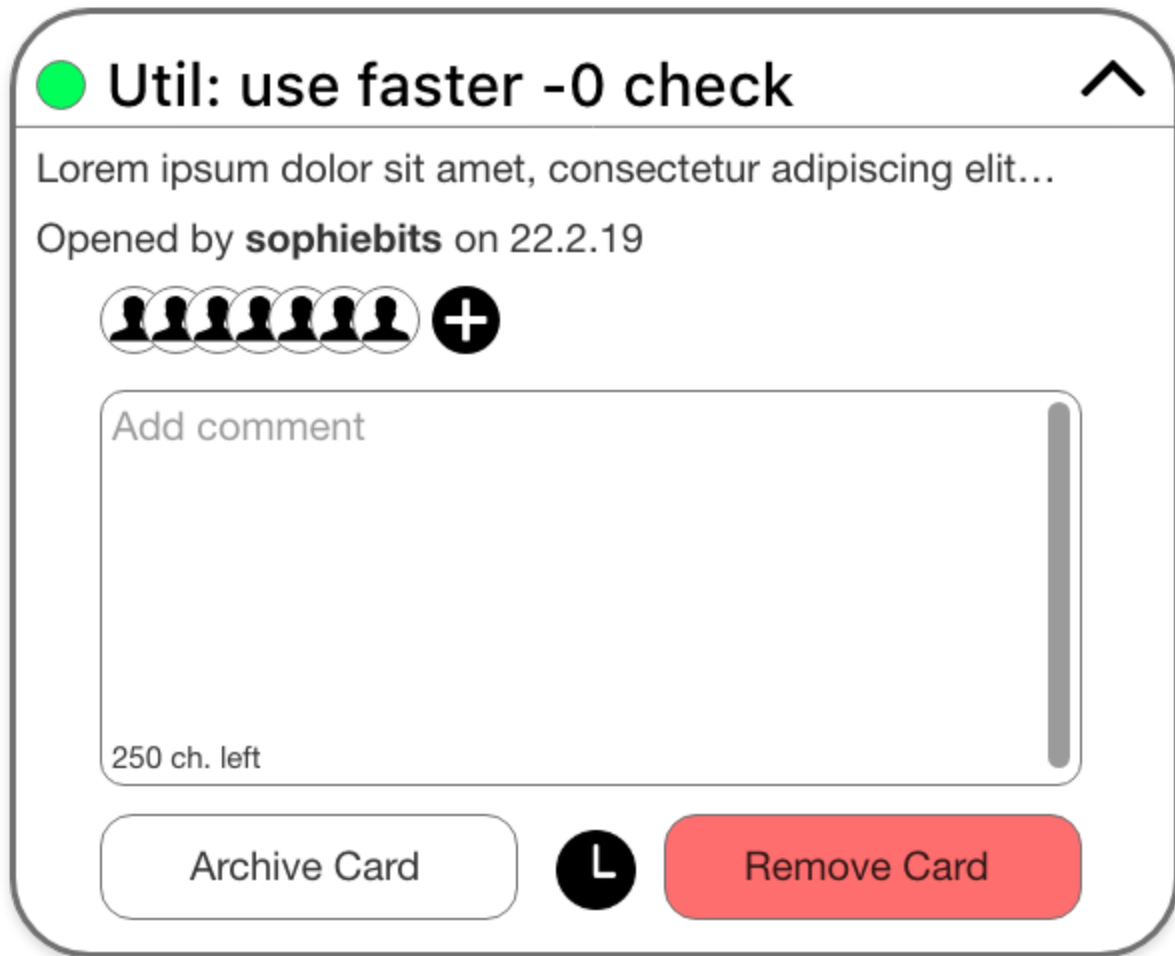
```
UPDATE Card
SET title = 'Card Name', description = 'Card Description', priority = 1,
status = 'Terminated', note = 'Card Note', end_date = GETDATE()
WHERE (card_id = 1234 AND user_id = 1234)
OR (card_id = 1234 AND EXISTS (SELECT user_id
FROM privilegedUser
WHERE Cards.user_ID = privilegedUser.user_id));
```

**Assigns cards to some people** (privileged user):

```
INSERT INTO AssignsCard(card_id, user_id) VALUES ('1234', '1357'));
```

**To get already created cards:**

```
SELECT * FROM Card WHERE list_id = 1;
```



**Figure 8:** Card Detail View

**Add comment:**

```
INSERT INTO Actions(card_id, user_id, project_id, issue_id) VALUES('2','1',  
NULL, NULL);
```

```
INSERT INTO Comment VALUES('1', 'Boards will be added to GUI');
```

**Remove card** (privileged user can delete all cards, other users can only delete their cards. This check will be done beforehand):

```
DELETE FROM Mentioned WHERE item_id IN (SELECT item_id FROM Actions WHERE card_id = 1);
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Actions WHERE card_id = 1);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Actions WHERE card_id = 1);
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Actions WHERE card_id = 1);
```

```
DELETE FROM Actions WHERE card_id = 1;
```

```
DELETE FROM Attachment WHERE card_id = 1;
```

```
DELETE FROM assignsCard WHERE card_id = 1;
```

```
DELETE FROM CardTag WHERE card_id = 1;
```

```
DELETE FROM Card WHERE card_id = 1;
```

**Archive card:**

```
INSERT INTO Actions(card_id, user_id, project_id, issue_id) VALUES('2', '1', NULL, NULL);
```

```
INSERT INTO Archive(item_id, archived_date) VALUES('2', GETDATE());
```

**Assign user to Card**(privileged user):

```
INSERT INTO assignsCard(card_id, user_id) VALUES('1', '2');
```

**Mention user:**

```
INSERT INTO Mention(item_id, mentioned_id) VALUES('1', '1');
```

**Mention user** (autocomplete for users while mentioning team id is already retrieved through the process and taken as 1):

```
SELECT username FROM Team NATURAL JOIN User WHERE team_id = '1' AND user_name = 'tra';
```

### Add Watchlist:

```
INSERT INTO Actions(card_id, user_id, project_id, issue_id) VALUES('2','1', NULL, NULL);
```

```
INSERT INTO Watchlist(item_id, notification_freq, last_updated) VALUES('3', 10, GETDATE());
```

## 5.2.5 Issues Page

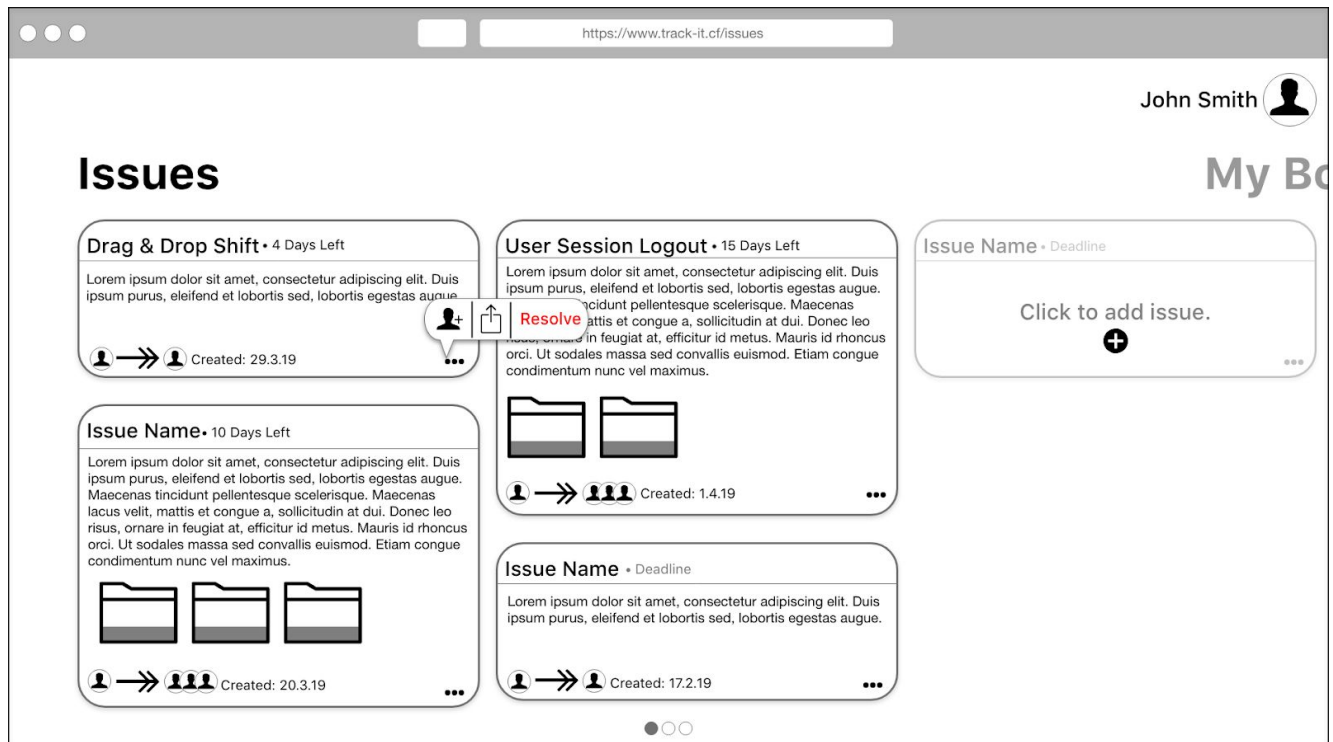


Figure 9: Issues Page



### List created Issues (assumed user in project 1):

```
SELECT name, status, start_date, end_date, priority, note FROM Issue WHERE  
project_id = '1';
```

### Create Issue:

```
INSERT INTO Issue(name, status, start_date, end_date, priority, note,  
attachment_id, item_id) VALUES ('Bord UI bug', 'active', GETDATE(), NULL,  
2, 'description about the board bug', NULL, NULL);
```

```
INSERT INTO Attachment(url, type, card_id, project_id, user_id, issue_id,  
release_id) VALUES ('attachmentUrl.com/jpeg.png', 'image', NULL, NULL, NULL,  
1, NULL);
```

```
UPDATE Issue SET attachment_id = 1;
```

### Delete Issue:

```
DELETE FROM assignIssue WHERE issue_id = 1;
```

```
DELETE FROM IssueTag WHERE issue_id = 1;
```

```
DELETE FROM Attachment WHERE issue_id = 1
```

```
DELETE FROM Watchlist WHERE item_id IN (SELECT item_id FROM Action WHERE  
issue_id = 1);
```

```
DELETE FROM Archive WHERE item_id IN (SELECT item_id FROM Action WHERE  
issue_id = 1);
```

```
DELETE FROM Comment WHERE item_id IN (SELECT item_id FROM Action WHERE  
issue_id = 1);
```

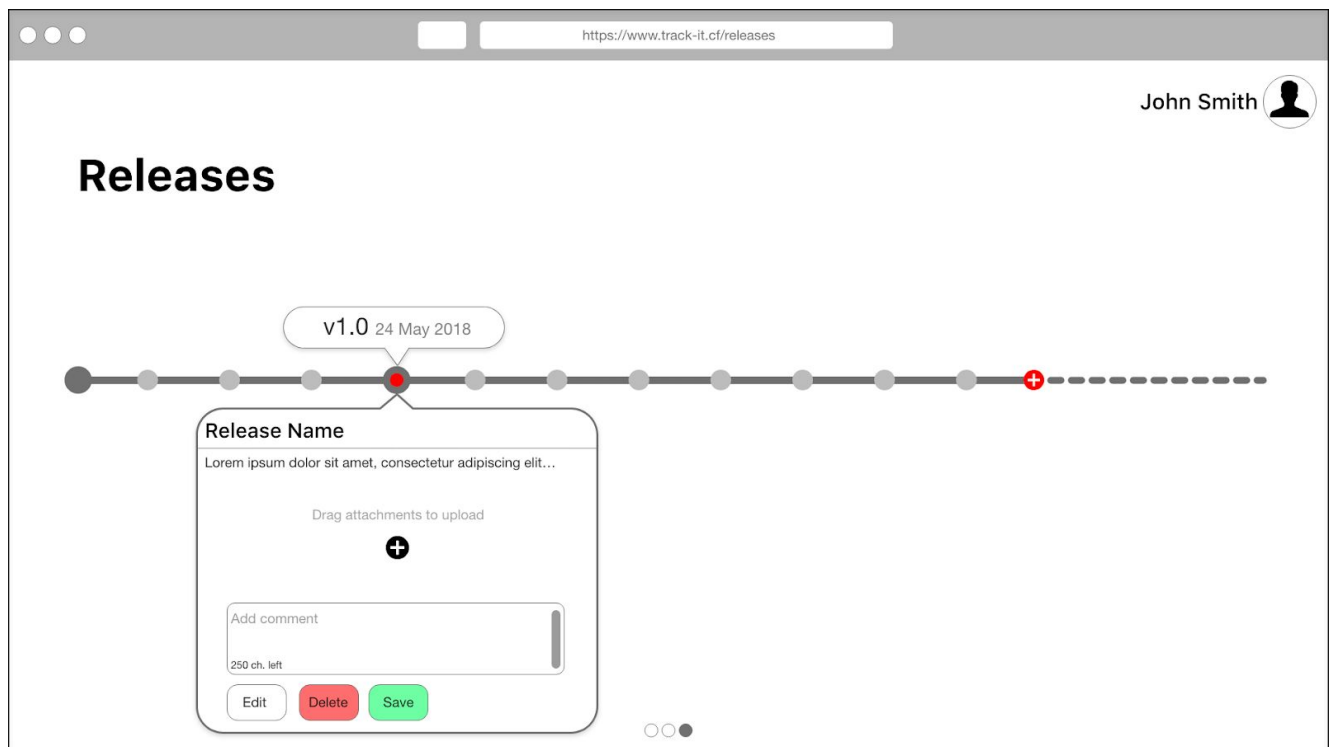
```
DELETE FROM Action WHERE issue_id = 1;
```

```
DELETE FROM Issue WHERE issue_id = 1;
```

### Resolve Issue:

```
UPDATE Issue SET status = 'resolved';
```

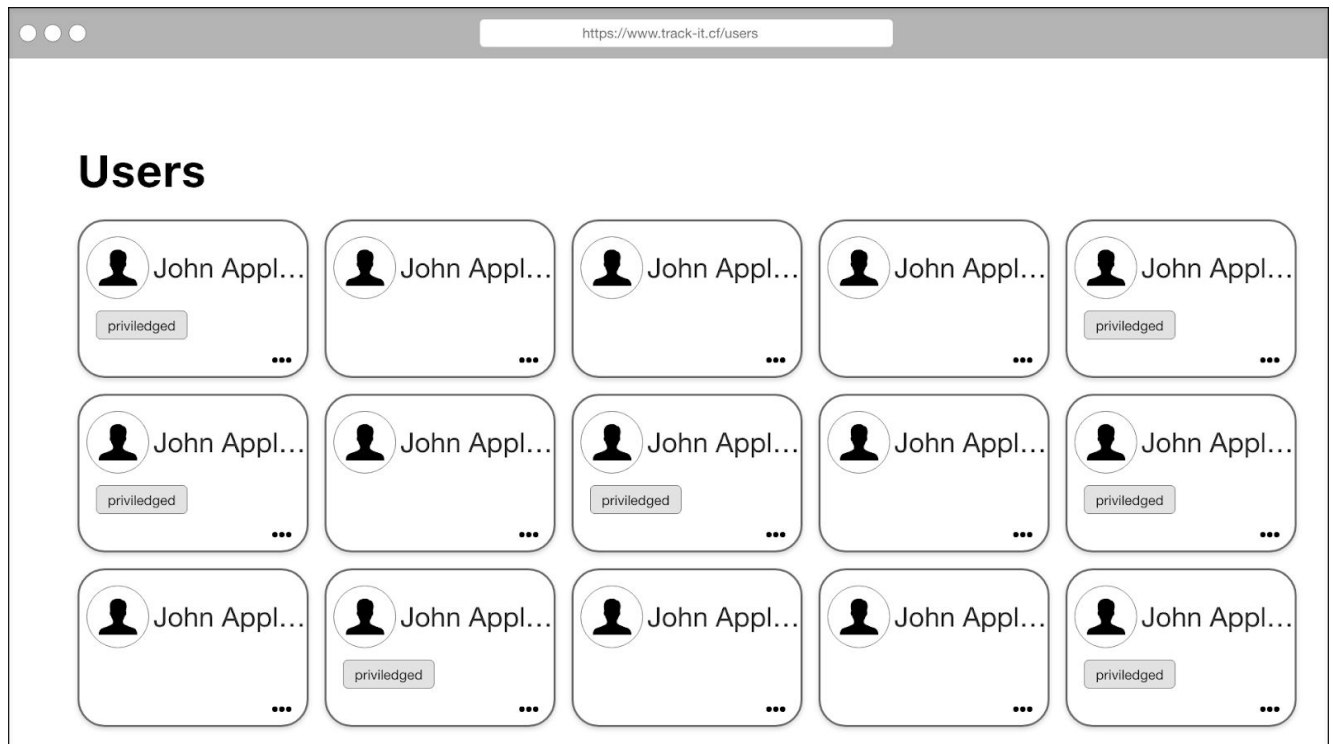
## 5.2.6 Releases Page



**Figure 10:** Releases Page

**List created releases** (assumed user in project 1):

```
SELECT *  
FROM Release  
WHERE project_id = '1';
```



**Figure 11: Users Page**

**List all users:**

```
SELECT *  
FROM User;
```

## 6. Advanced Database Components

---

### 6.1 Reports

#### 6.1.1 The team with the most issues for last month

This report generates the information about the team with the most issues for the last month.

```
WITH ISSUECOUNTS AS (  
  SELECT team_id, count(*) AS issueCount  
  FROM Issue NATURAL JOIN contributes  
  WHERE DATEDIFF(CURDATE(), start_date) <= 30  
  GROUP BY team_id  
);  
  
SELECT team_id, name, description, formation_date  
FROM ISSUECOUNTS NATURAL JOIN Team  
WHERE issueCount = (SELECT max(issueCount) FROM ISSUECOUNTS);
```

#### 6.1.2 The team with the greatest number of solved issues in last month

This report generates the information about the team with the greatest number of solved issues in last month.

```
WITH ISSUECOUNTS AS (  
  SELECT team_id, count(*) AS issueCount  
  FROM Issue NATURAL JOIN contributes  
  WHERE DATEDIFF(CURDATE(), start_date) <= 30 AND status = 'resolved'  
  GROUP BY team_id  
);  
  
SELECT team_id, name, description, formation_date  
FROM ISSUECOUNTS NATURAL JOIN Team  
WHERE issueCount = (SELECT max(issueCount) FROM ISSUECOUNTS);
```

#### 6.1.3 The average release number of the ongoing projects

This report generates the information about the average release number of the ongoing project. The result represents a parameter which can be used by the admins/managers to gain a general knowledge and evaluate the performance of the company.

```

SELECT avg(release_count)
FROM(
SELECT count(*) AS release_count
FROM Project NATURAL JOIN Release
WHERE status = 'active'
GROUP BY project_id
);

```

#### 6.1.4 The number of releases of the project with the most number of teams

This report generates the information about the number of releases of the project with the most number of teams.

```

WITH teamNums AS(SELECT team-id, count(*) AS team-cnt
                  FROM contributes
                  GROUP BY team_id);

SELECT project-id, count(*) AS release-no
FROM (
  (SELECT project-id, team-cnt
   FROM teamNums NATURAL JOIN retains
   WHERE team-cnt = (SELECT max(team-cnt) FROM teamNums)
  ) NATURAL JOIN Release
GROUP BY project_id;

```

## 6.2 Views

- Used to get profile photos of the team members in **Figure 5**

```
CREATE VIEW UsersOfTeams AS ( SELECT team_id, URL FROM User NATURAL JOIN
contributes NATURAL JOIN Attachment );
```

- Get active issues of the board, this is used in **Figure 9**

```
CREATE VIEW ActiveIssues AS ( SELECT board_id, name, start_date, end_date,
priority, note FROM Issues AS I NATURAL JOIN contributes NATURAL JOIN Board
WHERE I.status = 'active');
```

- Get mention notification. This notification shows the user that mentioned you and the place(in card, issue or project) All screens after login

```
CREATE VIEW MentionNotification AS ( SELECT user_id,comment, card_id,user_id,
project_id,issue_id FROM Comment NATURAL JOIN Mention NATURAL JOIN Actions);
```

- Get Watchlist from all of the watched items. This is used to combine and present the all the watched items (cards, projects, etc.) in the user profile.

```
CREATE VIEW AllWatchlist AS ( SELECT
user_id,notification_frequency,last_updated, card_id,user_id,project_id,issue_id
FROM User NATURAL JOIN Action NATURAL JOIN WatchList);
```

- This view is used direct access for color and desired card to group them in terms of their color. This view is used in **Figure 7**.

```
CREATE VIEW ColorTagGroup AS ( SELECT * FROM Card NATURAL JOIN CardTag);
```

- It gives profile photos of the users this view is used in **Figure 11**.

```
CREATE VIEW UserProfilePhotos AS (SELECT username,URL FROM User NATURAL JOIN
Attachment WHERE type = 'image')
```

- This is shown to privileged users in **Figure 4** using filtering feature.

```
CREATE VIEW OnGoingProjects AS (SELECT * FROM Project WHERE status = 'active')
```

## 6.3 Triggers

### 6.3.1 Schedule Reminder Trigger

When a scheduled date has reached it's time to notify the assigned user after the notification alert prompted to the user the scheduled date value should be set to null.

### 6.3.2 Increment Version Trigger

When a privileged user schedules a release the version number is automatically incremented to the next version.

### 6.3.3 Priority Notification Trigger

When an item such as an issue or a card's priority value have been modified, the users assigned to that specific item are will be automatically mentioned in the comment section of the item which will cause the scheduling of a reminder to notify the users.

## 6.4 Constraints

- A standard user can create at most 20 cards.
- Privileged users can not change other privileged users status.
- Uploaded attachments could be at most 150 mb in size.
- Only media and document types such as .pdf, .txt, .jpg, etc. can be uploaded as attachments.
- A board cannot have more than one team assigned to it.
- Simultaneous releases of same project are not allowed.

## 6.5 Stored Procedure

### 6.5.1 Filtering Cards

We have decided to design a procedure for filtering cards according to the given tag name. See Figure 7.

### 6.5.2 Filtering Issues

We have decided to design a procedure for filtering issues according to the given tag name. See Figure 9.

### 6.5.3 Filtering Users

We have decided to design a procedure for filtering users according to their user type. This is used in Figure 11 to distinguish between standard and privileged users. Also, this way their user type can be easily changed.

## 7. Implementation Plan

---

In this project we will be using PHP for our back-end web service, along with the MySQL database. In front-end, we will create a website with HTML and make use of Bootstrap CSS library to obtain a modern website view.

## 8. Website

---

This report is submitted (softcopy) to <http://track-it.cf> in partial fulfillment of the requirements of the Term Project of CS353 course.