Bilkent University

Department of Computer Engineering

CS 353 - Database Systems

Term Project - Design Report

Project Name: Tasks & Managers

Group No: 26

Group Members: Cihan Erkan
Merve Sağyatanlar
Çağla Sözen
Murat Tüver

Table of Contents

1. Revised E/R Model

1.1. Modifications

For clarification of the system structure and correction of the proposed ER Diagram, following modifications were made,

- *Board, List* and *Card* became strong entities since they are the main components of he system and should be able to exist individually and uniquely in the system regardless of the existence of another system. Therefore, primary keys (ID) were added to each of them and their relations were modified accordingly preserving the total participation.
- Considering the mutual functionalities of user types, we changed the user inheritance and added a *Worker* entity to use for the common functionalities of a Standard User and a Team Leader.
- *Map* became a strong entity since it is also an entity that should be able to exist individually in the system. We changed it such that it will represent a single *Project* but a *Project* will be able to have multiple *Maps*. We preserved it's total participation since having a *Map* that doesn't represent anything does not make sense.
- We changed the relations between a *Team*, *Team Leader* and *Project* such that it will be a ternary relation having the attributes *rating* and *since* (referred as a Project Group in this report). This ternary relation represents the whole structure of a project along with its team and leader. This wholeness is represented with an aggregate structure such that a manager will manage all of this structure as a whole. Rating will be determined by the corresponding Manager and will be used to calculate each User's rating according to the average rating of all projects they are a member of.
- We have added a new entity *Issue* to provide a different functionality to cards. If the assigned user of a Card faces a difficulty for the Card, he/she will be able to create an issue and other *Worker* users in that team will be able to create *Answers* to that *Issue. Answer* is created as a weak entity within an Issue. If an *Answer* is selected to be a correct answer to that *Issue* by the user that created the *Issue* then, the *answer_ID* of the Issue will be updated with that *Answer's* ID, else it will be *NULL*.
- *Poll* was also added as a new entity to provide a new functionality for *Cards*. A Poll has an attached weak entity *Response*. *Responses* can map to a single *Poll* and a user can have a single response to a particular *Poll*. *Description* of the Response determine the answer that is an option to respond to that poll. *Description* is a partial key, and *Responses* are unique according to their *description* within the same *Poll*. *no_of_votes* determine how many votes were given to that particular *Response* to that *Poll*.
- *Rating* was changed as a normal attribute and added to *Worker* entity since only *Worker*'s can be rated by Managers.

1.2. Revised E/R Diagram

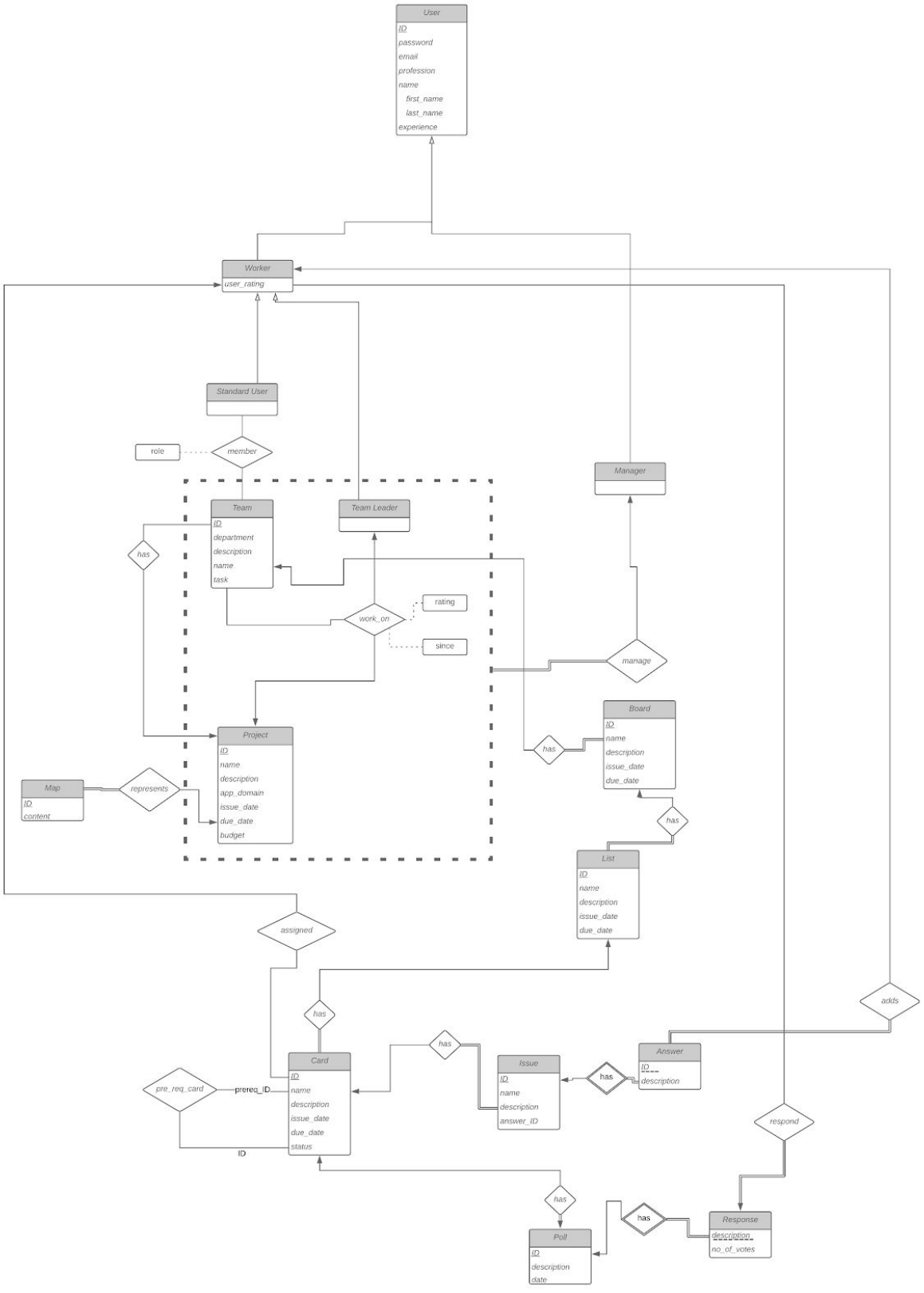

Figure 1: ER Diagram

2. Relational Schemas

2.1. User

Relational Model: User(ID,email, password, profession, first_name, last_name, experience)
Functional Dependencies: {(ID -> email, password, profession, first_name, last_name, experience), (email -> ID)}
Candidate Keys: {{ID} , {email}}
Primary Key: {ID}
Foreign Keys: {}
Normal Form: BCNF
Table Definition:

```
CREATE TABLE User(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        email VARCHAR(50) NOT NULL,
        password VARCHAR(50) NOT NULL,
        first_name VARCHAR(50),
        last_name VARCHAR(50),
        experience INTEGER
);
```

2.2. Standard User

Relational Model: StandardUser(ID)
Functional Dependencies: {}
Candidate Keys: {{ID}}
Primary Key: {ID}
Foreign Keys: {{ ID->User.ID }}
Normal Form: BCNF
Table Definition:

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

2.3. Worker

Relational Model: Worker(ID, user_rating)
Functional Dependencies: {ID -> user_rating }
Candidate Keys: {{ID}}
Primary Key: {ID}
Foreign Keys: {{ ID->User.ID }}
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Worker(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        user_rating INTEGER,
        FOREIGN KEY(ID) REFERENCES User(ID)
);
```

2.4. Team Leader

Relational Model: TeamLeader(ID, team_id, project_id)
Functional Dependencies: {}
Candidate Keys: {{ID, team_id, project_id}}
Primary Key: {ID, team_id, project_id}
Foreign Keys: {{ ID->PrivelegedUser.ID }, {team_id -> Team.ID}, {project_id -> Project.ID}}
Normal Form: BCNF
Table Definition:

```
CREATE TABLE TeamLeader(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        team_ID INTEGER NOT NULL,
        project_ID INTEGER NOT NULL,
        FOREIGN KEY (project_ID) REFERENCES Project(ID),
        FOREIGN KEY (team_ID) REFERENCES Team(ID),
        FOREIGN KEY(ID) REFERENCES PrivelegedUser(ID)
);
```

2.5. Manager

Relational Model: Manager(ID)
Functional Dependencies: {}
Candidate Keys: {{ID}}
Primary Key: {ID}
Foreign Keys: {{ ID->User.ID }}
Normal Form: BCNF

Table Definition:

```
CREATE TABLE Manager(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        FOREIGN KEY(ID) REFERENCES User(ID)
);
```

2.6. Project

Relational Model: Project(ID, app_domain, issue_date, due_date, budget, name, description, manager_ID)
Functional Dependencies: {(ID -> app_domain, issue_date, due_date, budget, name, description, manager_ID), (name, manager_ID -> ID)}
Candidate Keys: {{ID}, {name, manager_ID}}
Primary Key: {ID}
Foreign Keys: { manager_ID -> Manager.ID }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Project(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        app_domain VARCHAR(50),
        issue_date DATETIME NOT NULL,
        due_date DATETIME,
        budget INTEGER,
        name VARCHAR(50) NOT NULL,
        description VARCHAR(100),
        manager_ID INTEGER,
        FOREIGN KEY (manager_ID) REFERENCES Manager(ID)
);
```

2.7. Team

Relational Model: Team(ID, department, description, name, project_ID, leader_ID, rating, since)
Functional Dependencies: {(ID->department, description, name, project_ID, leader_ID, rating, since), (project_ID, leader_ID, name -> ID)}
Candidate Keys: {{ID}, {project_ID, leader_ID, name}}
Primary Key: {ID}
Foreign Keys: {{ project_ID -> Project.ID },{ leader_ID -> TeamLeader.ID }}
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Team(
```

```
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        department VARCHAR(50),
        description VARCHAR(100),
        name VARCHAR(50) NOT NULL,
        project_ID INTEGER,
        leader_ID INTEGER,
        rating NUMERIC(1,0) DEFAULT 1,
        since DATETIME,
        FOREIGN KEY (project_ID) REFERENCES Project(ID),
        FOREIGN KEY (leader_ID) REFERENCES TeamLeader(ID),
        CHECK( rating < 5 AND 1 < rating )
);
```

2.8. Board

Relational Model: Board(ID, name, description, issue_date, due_date, team_ID)
Functional Dependencies: {(ID -> name, description, issue_date, due_date, team_ID), (name, team_ID -> ID)}
Candidate Keys: {{ID}, {name, team_ID}}
Primary Key: {ID}
Foreign Keys: { team_ID -> Team.ID }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Board(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        description VARCHAR(100),
        issue_date DATETIME NOT NULL,
        due_date DATETIME,
        team_ID INTEGER NOT NULL,
        FOREIGN KEY (team_ID) REFERENCES Team(ID)
);
```

2.9. List

Relational Model: List(ID, name, description, issue_date, due_date, board_ID)
Functional Dependencies: { (ID -> name, description, issue_date, due_date, board_ID), (name, board_ID -> ID) }
Candidate Keys: {{ID}, {name, board_ID}}

Primary Key: {ID}
Foreign Keys: { board -> Board.ID }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE List(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        description VARCHAR(100),
        issue_date DATETIME NOT NULL,
        due_date DATETIME,
        FOREIGN KEY (board_ID) REFERENCES Board(ID)
);
```

2.10. Card

Relational Model: Card(ID, name, description, issue_date, due_date, status, prereq_ID , list_ID, assigned_ID)
Functional Dependencies: {(ID -> name, description, issue_date, due_date, status, prereq_ID, list_ID, assigned_ID),(name, list_ID, assigned_ID -> ID)}
Candidate Keys: {{ID}, {name, list_ID, assigned_ID**}}**
Primary Key: {ID}
Foreign Keys: { prereq_ID -> Card.ID, list_ID -> List.ID, assigned_ID -> Worker.ID }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Card(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        description VARCHAR(100),
        issue_date DATETIME NOT NULL,
        due_date DATETIME,
        prereq_ID INTEGER,
        list_id NOT NULL,
        assigned_ID INTEGER,
        status VARCHAR(15) CHECK IN(“DONE”, “IN PROGRESS”,
“APPROVED”, “REJECTED”, “NOT STARTED”),
        FOREIGN KEY (prereq_ID) REFERENCES Card(ID),
        FOREIGN KEY (list_ID) REFERENCES List(ID),
        FOREIGN KEY (assigned_ID) REFERENCES Worker(ID)
);
```

2.11. Map

Relational Model: Map(ID, content, project_ID)
Functional Dependencies: { (ID -> content, project_ID), (project_ID, content -> ID) }
Candidate Keys: {{ID}, {project_ID, content}}
Primary Key: {ID}
Foreign Keys: { {project_ID -> Project.ID } }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Map(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        content VARCHAR(50) NOT NULL,
        project_ID INTEGER NOT NULL,
        FOREIGN KEY (project_ID) REFERENCES Project(ID)
);
```

2.12. Poll

Relational Model: Poll(ID, description, date, card_ID)
Functional Dependencies: { (ID -> date, card_ID, description) , (card_ID -> ID)}
Candidate Keys: {{ID}, {card_ID}}
Primary Key: {ID}
Foreign Keys: { {card_ID -> Card.ID } }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Poll(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        description VARCHAR(100) NOT NULL,
        date DATETIME NOT NULL,
        card_ID INTEGER NOT NULL,
        FOREIGN KEY (card_ID) REFERENCES Card(ID)
);
```

2.13. Response

Relational Model: Response(poll_ID, description, no_of_votes)
Functional Dependencies: { (description, poll_ID -> no_of_votes) }
Candidate Keys: {{ poll_ID, description}}
Primary Key: {poll_ID, description}
Foreign Keys: { {poll_ID -> Poll.ID } }
Normal Form: BCNF

Table Definition:

```
CREATE TABLE Response(
        description VARCHAR(50) NOT NULL,
        no_of_votes INTEGER DEFAULT 0          ,
        poll_ID INTEGER NOT NULL,
        FOREIGN KEY (poll_ID) REFERENCES Poll(ID)
);
```

2.14. Issue

Relational Model: Issue(ID, name, description, answer_ID, card_ID)
Functional Dependencies: { (ID -> name ,description, answer_ID, card_ID), (card_ID, name -> ID) }
Candidate Keys: {{ID}, {card_ID, name}}
Primary Key: {ID}
Foreign Keys: { {card_ID -> Card.ID } }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Issue(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        description VARCHAR(50) NOT NULL,
        no_of_votes INTEGER DEFAULT 0          ,
        answer_ID INTEGER ,
        card_ID INTEGER NOT NULL,
        FOREIGN KEY (card_ID) REFERENCES Card(ID),
        FOREIGN KEY (answer_ID) REFERENCES Answer(ID)
);
```

2.15. Answer

Relational Model: Answer(ID, issue_ID, description, user_ID)
Functional Dependencies: { (ID, issue_ID -> description , user_ID) }
Candidate Keys: {{issue_ID, ID}}
Primary Key: {issue_ID, ID}
Foreign Keys: { {issue_ID -> Issue.ID } {user_ID -> Worker.ID} }
Normal Form: BCNF
Table Definition:

```
CREATE TABLE Answer(
        ID INTEGER PRIMARY KEY AUTO_INCREMENT,
        description VARCHAR(50) NOT NULL,
        issue_ID INTEGER NOT NULL,
```

```
        user_ID INTEGER NOT NULL,
        FOREIGN KEY (issue_ID) REFERENCES Issue(ID),
        FOREIGN KEY (user_ID) REFERENCES Worker(ID)
);
```

2.16. Member

Relational Model: Member(member_ID, role, team_ID)
Functional Dependencies: { (member_ID, team_ID -> role) }
Candidate Keys: {{member_ID, team_ID}}
Primary Key: {member_ID, team_ID}
Foreign Keys: **{{** member_ID -> StandardUser.ID **},** **{**team_ID -> Team.ID**}}**
Normal Form: BCNF
Table Definition:

```
    CREATE TABLE Member(
        member_ID INTEGER NOT NULL,
        role VARCHAR(50),
        team_ID INTEGER NOT NULL,
        FOREIGN KEY (member_ID) REFERENCES StandardUser(ID),
        FOREIGN KEY (team_ID) REFERENCES Team(ID),
        PRIMARY KEY (member_ID, team_ID)
);
```

3.Functional Dependencies and Normalization

Section 2 shows the relational design of the database with the relational schemas, functional dependencies they contain and their corresponding normal forms according to these functional dependencies. All tables are in Boyce-Codd Normal Form by design. Hence, the design did not require any decomposition or normalization.

4. Functional Components

4.1. Use Cases / Scenarios

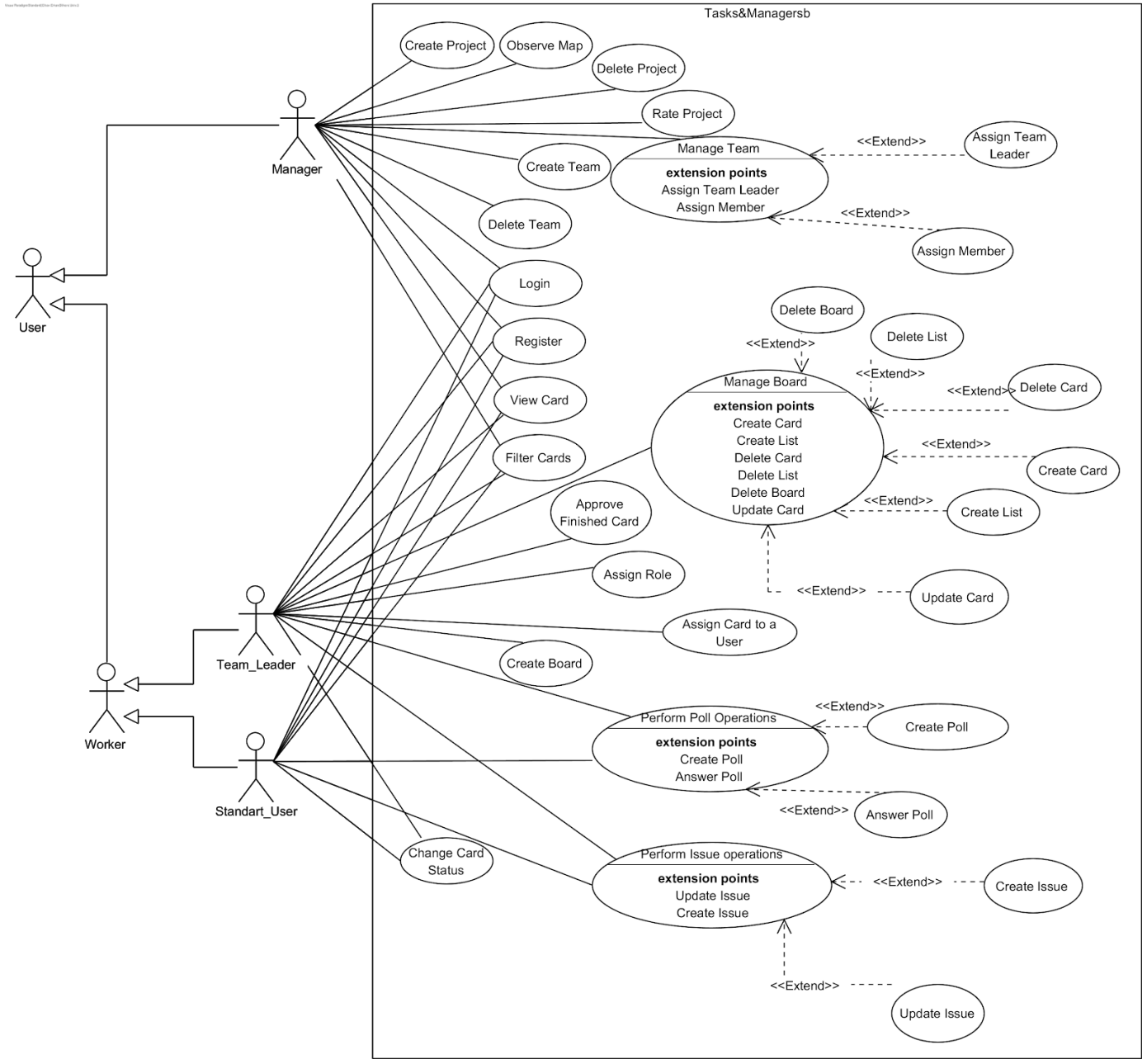

Figure 2: Use Case Diagram

4.1.1. User Account

4.1.1.1. Register

Use Case: Register to Tasks&Managers
Primary Actor: User

Stakeholders and interests:

- Newcomer user who wants to register to the system.

Pre-conditions:

- User must have an email account.
- User must be in the registration screen.

Successful Scenario Event Flow for Register:

1. User enters email address and password to register.
2. User re-types his/her password.
3. Optionally, user enters his/her name, experience and profession.
4. User successfully registers to the system.

Unsuccessful Scenario Event Flow for Register:

1. User enters email address and password to register.
2. User re-types his/her password.
3. System notifies user that registration failed.
4. System asks for information again.

Post Conditions:

- User's information is recorded to the system.

4.1.1.2. Login

Use Case: Login

Primary Actor: User

Stakeholders and interests:

- User who wants to login to an account

Pre-conditions:

- User must have an account.
- There shouldn't be any account already logged in.

Successful Scenario Event Flow for Login:

1. User enters email address and password to login.
2. System checks whether password matches.
3. User logs in.

Unsuccessful Scenario Event Flow for Login:

1. User enters email address and password to login.
2. System checks whether password matches.
3. System notifies user that password is incorrect.
4. System asks for password again.

Post Conditions:

- User granted with appropriate permissions.

4.1.1.3. View Card

Use Case: View Card

Primary Actor: User
Stakeholders and interests:
- User who wants to access information related to his/her project.

Pre-conditions:
- User must be logged in.
- User must participate in the project's team.
- User must be in project screen.

Successful Scenario Event Flow for View Card:
1. User clicks on card he/she wants to view.
2. Detailed information shown to the user.

4.1.1.4. Filter Card

Use Case: Filter Card
Primary Actor: User
Stakeholders and interests:
- User who wants filter and see cards.

Pre-conditions:
- User must be logged in.

Successful Scenario Event Flow for Filter Card:
1. User enters a board.
2. Clicks "Filter" button.
3. Selects desired filtering option.
4. Confirms selection.

Unsuccessful Scenario Event Flow for Filter Card:
1. User enters email address and password to login.
2. System checks whether password matches.
3. System notifies user that password is incorrect.
4. System asks for password again.

Post Conditions:
- Filtered cards are listed to user.

4.1.2. Team Leader

4.1.2.1. Assign Role

Use Case: Assign Role
Primary Actor: Team Leader
Stakeholders and interests:
- Team leader who wants to assign role to a team member.

Pre-conditions:
- Team leader must be logged in.
- Team leader and member must participate in the same team.

Successful Scenario Event Flow for Assign Role:

1. Team leader finds a team member from members list.
2. Team leader clicks assign role button next to user's name.
3. Team leader types the role and clicks confirm.

Post Conditions:

- New role is assigned to the team member.
- Team member is notified about the change.

4.1.2.2. Assign Card to a User

Use Case: Assign Card to a User
Primary Actor: Team leader
Stakeholders and interests:

- Team leader who wants to assign a card to a team member.

Pre-conditions:

- Team leader must be logged in.
- A card must exist in a list.
- User must be a member of team.

Successful Scenario Event Flow for Assign Card to a User:

1. Team leader selects a card to assign a team member.
2. Team leader clicks settings button of the card.
3. Team leader selects a user from the list from assign user to a card option.

Post Conditions:

- Assigned card's UI updated to show who is responsible for the task.
- Team member is notified about the assigned card.

4.1.2.3. Manage Board

Use Case: Manage Board
Primary Actor: Team Leader
Stakeholders and interests:

- Team leader who wants to manage a board.

Pre-conditions:

- Team leader must be logged in.
- A board is created.

Successful Scenario Event Flow for Delete Board:

1. Team leader selects a board.
2. Team leader clicks delete board button.
3. Team leader clicks confirm deletion.

Post Conditions for Delete Board:

- Lists and cards contained in the board deleted.
- Team members informed about deletion.

Successful Scenario Event Flow for Delete List:

1. Team leader selects a list.
2. Team leader clicks on delete icon.
3. Team leader confirms deletion.

Post Conditions for Delete List:

- Cards also deleted from the list.
- Team members informed about deletion.

Successful Scenario Event Flow for Delete Card:

1. Team leader selects a card.
2. Team leader clicks settings button of the card.
3. Team leader clicks delete.
4. Team leader confirms deletion.

Post Conditions for Delete Card:

- Card is deleted from the list.
- If card assigned to a user, that user is informed about deletion.

Successful Scenario Event Flow for Create List:

1. Team leader clicks on add list button.
2. Team leader specifies name.
3. Optionally, team leader gives a description and due date.
4. Team leader confirms creation.

Post Conditions for Create List:

- New list is created and added to the board with issue date.

Successful Scenario Event Flow for Create Card:

1. Team leader selects a list.
2. Team leader clicks add card button on top of the list.
3. Team leader specifies name of the card.
4. Optionally, team leader writes a description.
5. Team leader confirms creation.

Post Conditions for Create Card:

- New card is created and added to corresponding list.

4.1.2.4. Create Board

Use Case: Create Board

Primary Actor: Team leader

Stakeholders and interests:

- Team leader who wants to create a new board.

Pre-conditions:

- Team leader must be logged in.

Successful Scenario Event Flow for Create Board:

1. Team leader clicks on create board button.
2. Team leader enters name and description.
3. Optionally, team leader enters due date.

Post Conditions:

- New board created with issue date.

4.1.2.5. Approve Finished Card

Use Case: Approve Finished Card
Primary Actor: Team leader
Stakeholders and interests:

- Team leader who wants to approve a finished card.

Pre-conditions:

- Team leader must be logged in.
- At least a card must exist in a list.
- Card must be marked as finished by its owner.

Successful Scenario Event Flow for Approve Finished Card:

1. Team leader finds a card to approve.
2. Team leader clicks card's settings button.
3. Team leader clicks on change card status.
4. Team leader clicks "Approve" button.

Post Conditions:

- Card marked as approved.

4.1.3. Manager Account

4.1.3.1. Create Project

Use Case: Create Project
Primary Actor: Manager
Stakeholders and interests:

- Manager who wants to create a new project

Pre-conditions:

- Manager must be logged in.
- Manager must be in "Projects" screen.

Successful Scenario Event Flow for Create Project:

1. Manager clicks to "New Project" button.
2. Manager enters the name of the new project.
3. Manager clicks "Create Project" button.

Unsuccessful Scenario Event Flow for Create Project:

1. Manager clicks to "New Project" button.
2. Manager enters the name of the new project.
3. Manager clicks "Create Project" button.
4. System notifies manager that specified project name is not unique.
5. System asks manager for a new project name.

Post Conditions:

- A new project is created with specified name

4.1.3.2. Observe Map

Use Case: Create Project
Primary Actor: Manager
Stakeholders and interests:
- Manager who wants to observe the maps of projects.

Pre-conditions:
- Manager must be logged in.
- There must be at least one project.
- Manager must be in "Projects" screen.

Successful Scenario Event Flow for Observe Map:
1. Manager clicks to the the preview of the map.

Post Conditions:
- Map of the project displayed to the manager.

4.1.3.3. Rate Project

Use Case: Rate Project
Primary Actor: Manager
Stakeholders and interests:
- Manager who wants to rate a project.

Pre-conditions:
- Manager must be logged in.
- Manager must be in "Projects" screen.
- There must be at least one project to rate.

Successful Scenario Event Flow for Rate Project:
1. Manager finds the project he/she desires to rate.
2. Manager clicks to the "Rate Project" button.
3. Manager enter the rating.
4. Manager confirms rating

Post Conditions:
- System updates projects rating.

4.1.3.4. Create Team

Use Case: Create Team
Primary Actor: Manager
Stakeholders and interests:
- Manager who wants to create a new team.

Pre-conditions:
- Manager must be logged in.
- Manager must be in "Projects" screen.

Successful Scenario Event Flow for Create Team:

1. Manager finds the project he/she wants to add a team from the list of teams.
2. Manager clicks to the "Manage Teams" button.
3. Manager specifies a name for the team.
4. Manager confirms team creation.

Post Conditions:

- A new team is created.
- System adds the new team to the project.

4.1.3.5. Manage Team

Use Case: Manage Team
Primary Actor: Manager
Stakeholders and interests:

- Manager who wants to manage a team.

Pre-conditions:

- Manager must be logged in.
- There must be at least one team to manage.

Successful Scenario Event Flow for Assign Team Leader:

1. From "Project" screen manager selects a project.
2. Manager selects a team in order to assign a team leader.
3. Manager selects a user to be the team leader of team.
4. Manager checks assign as Team Leader box.

Post Conditions for Assign Team Leader:

- System updates team leader of the specified team.
- System notifies the user assigned as team leader.

Successful Scenario Event Flow for Assign Member:

1. Manager selects a project from "Project" screen.
2. Manager finds the team from list of teams.
3. Manager clicks "Add Member" icon, next to team name.
4. Manager selects a user to add to the team.
5. Manager confirms.

Post Conditions for Assign Member:

- System updates list of team members.
- System notifies assigned user.

4.1.3.6. Delete Project

Use Case: Delete Project
Primary Actor: Manager
Stakeholders and interests:

- Manager who wants to delete a project.

Pre-conditions:

- Manager must be logged in.
- There must be at least one project.

Successful Scenario Event Flow for Deleting Project:

1. Manager clicks Settings button in his/her home page.
2. Manager clicks delete project button.
3. Manager chooses a project to delete.

Post Conditions for Deleting Project:

- System deletes project and its respective components.
- System notifies the project members about deletion.

4.1.3.7. Delete Team

Use Case: Delete Team
Primary Actor: Manager
Stakeholders and interests:

- Manager who wants to delete a team.

Pre-conditions:

- Manager must be logged in.
- There must be at least one team.

Successful Scenario Event Flow for Deleting Team:

1. Manager clicks manage team button in his/her home page.
2. Manager clicks minus icon next to the team.

Post Conditions for Deleting Team:

- System deletes team and its respective components.
- System notifies the team members about deletion.

4.1.4. Team Leader and Standard User Use Cases

4.1.4.1. Create Issue

Use Case: Create Issue
Primary Actors: Team leader, Standard User
Stakeholders and interests:

- Team leader or a standard user who wants to create new issue.

Pre-conditions:

- User must be logged in.

Successful Scenario Event Flow for Create Issue:

1. User selects a card.
2. User clicks on create issue button.
3. Team leader enters name and description.
4. Optionally, team leader enters due date.

Post Conditions:

- New issue created with issue date.

4.1.4.2. Change Card Status

Use Case: Change Card Status
Primary Actors: Team leader, Standard User
Stakeholders and interests:

- Team leader or a standard user who wants to change the status of the card.

Pre-conditions:

- User must be logged in.
- For standard users, card must be assigned to them.
- Team leaders can change any card in their team's board.

Successful Scenario Event Flow for Change Card Status:

1. User selects a card.
2. User clicks card's settings button.
3. User clicks change card status button.
4. User selects the status he/she desires.

Post Conditions:

- Status of the card is updated.

4.1.4.3. Perform Poll Operations

Use Case: Perform Poll Operations
Primary Actor: Team Leader, Standard User
Stakeholders and interests:

- Team Leader or standard user who wants to create a poll
- Team Leader or standard user who wants to cast a vote in an existing poll

Pre-conditions:

- User must be logged in.

Successful Scenario Event Flow for Create a Poll:

1. User finds the card he/she wants to add a poll or creates a new card.
2. User clicks settings button of the card.
3. User clicks "Add Poll" option.
4. User types the question of the poll.
5. User determines possible answers for the poll.
6. User confirms poll creation.

Post Conditions for Assign Team Leader:

- A new poll is created.

Successful Scenario Event Flow for Answer Poll:

1. User finds a card that contains a poll.
2. User checks one or more answers.
3. User submits his/her answers.

Post Conditions for Assign Member:

- System updates results for poll.

4.1.4.4. Perform Issue Operations:

Use Case: Perform Issue Operations
Primary Actor: Team Leader, Standard User
Stakeholders and interests:

- Team leader or Standard User who wants to create an issue.
- Team leader or Standard User who wants to update an issue.

Pre-conditions for Create Issue:

- User must be logged in.

Successful Scenario Event Flow for Create Issue:

1. User selects a card.
2. User clicks on settings button.
3. User clicks on create issue button.
4. User enters issue name and description.

Post Conditions for Create Issue:

- An issue is created with issue date.
- Closed envelope mark with an exclamation mark appears in card's user interface.

Pre-conditions for Update Issue:

- User must be logged in.
- User must be either the one who created the issue or a team leader.

Successful Scenario Event Flow for Update Issue:

1. User selects card.
2. User selects issue.
3. User clicks on issue's description.
4. User makes changes to the issue.
5. User sets an answer to be the best answer.
6. User confirms his/her actions.

Post Conditions for Update Issue:

- Issue's description has changed.
- Other team members notified about the change.
- Issue is marked as resolved.

4.2. Algorithms

4.2.1 Map Creation Algorithm

A manager will be able to view multiple types of maps for a particular project they are managing. Maps will represent Projects from different perspectives by having different contents. For creation of maps and showing them on Manager Screen, an algorithm to compute different statistics of a Project is needed. For this, the algorithm to be used should first identify the content of the map to be created (i.e Team Map, Progress Map, Leader Map etc), then extract the corresponding information. Then apply some statistical operations (i.e

averaging etc.) to come up with a meaningful representation of the project according to its current structure. For example, “Progress” Map of a project will first extract teams of that particular project and get the number of approved cards for each team in that project. This statistic will be visualized as a map (or graph) on the manager screen.

4.2.2 Card Filtering Algorithm

Tasks&Managers provide the functionality to filter cards according to their due dates, or status. For this we need an algorithm that filters cards according to the user input (button click). This algorithm will extract the cards according to the user input. For example when the user chooses to filter the cards with an expired due date, or rejected cards, then the algorithm will choose those cards and they will be shown on the user screen.

4.2.3 Card Sorting Algorithm

Similar to the Filtering algorithm, a user will be able to sort the cards on the screen according to their due dates and issue dates. For this, the algorithm will first get the user input (button click) and sort according to the variable the user has chosen (i.e issue date or due date). After the algorithm is applied, cards will be shown in a particular order on the screen.

NOTE: 4.2.3 and 4.2.2 are not mutually exclusive, they will be able to be applied at the same time.

4.2.4 User Rating Algorithm

An algorithm to compute the rating of a particular user according to the rating of projects he/she was in is required in order to show when required. This algorithm will first extract the teams the user is a member/leader of. After extraction, it will compute the average value of the ratings of these teams. Then, the computed rating will be shown as the user rating when the user info is shown on screen.

4.2.5 Poll Result Algorithm

Cards may contain Polls that have multiple answers and require calculation of answer ratios. For this, when a particular poll is shown on the screen, an algorithm to extract the number of votes of each answer to that particular poll should be extracted and then for each answer a percentage should be shown on screen.

4.2.6 Prerequisite Algorithm

A card may or may not have a prerequisite card. If such prerequisite card exist, when the status of a card is updated, the algorithm should check for the status of its prerequisite. If the prerequisites status is approved then, status of the card can be updated. However, if aits prerequisite is not approved then the update should not be possible.

4.3. Data Structures

Data types Integer, Numeric, Varchar and Datetime were used in the schemas as attribute domains for the following reasonings.

- **Integer:** Used for storing data in the form of numbers without a decimal point. It is used in this system for ID's, vote counts and project budget.
- **Numeric:** Used for storing data in the form of numbers with decimal points. It is used in this system for ratings.
- **Varchar:** Used for storing strings of variable length. It is used in this system to store name, description, email, password, app_domain, department, status, content and role.
- **Datetime:** Used for representing important dates for entities. It is used in this system to represent issue_date, due_date and since attributes.

Apart from the attribute domains, **Lists** will be used to represent Teams, Team Members and Projects in the Front-End side of Tasks&Managers.

5. User Interface Design and SQL Statements

For simplicity, we used dummy values that are going to be obtained from user interaction from the frontend side when the system is deployed. (For example user.ID = 1980361342, in the real system, the query will rely on a ID obtained from the system instead of 1980361342)

5.1. User Pages

5.1.1. Login Page

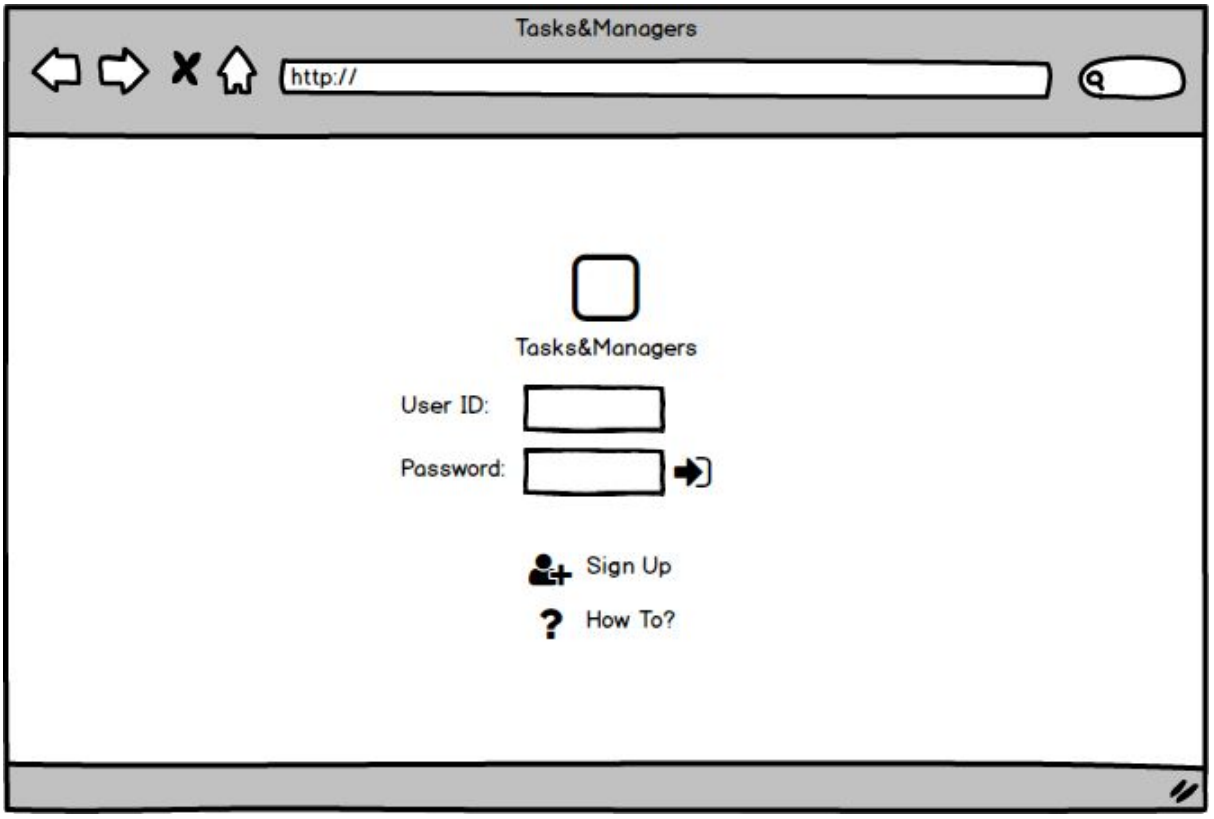

Figure 3: Login Page

SQL query which checks user credentials:
SELECT email, password
FROM User
WHERE email = 'test@mail.com' and password = '1234567890'

If user fails to enter valid credentials, following screen is displayed. After closing pop-up, user may try again to login again.

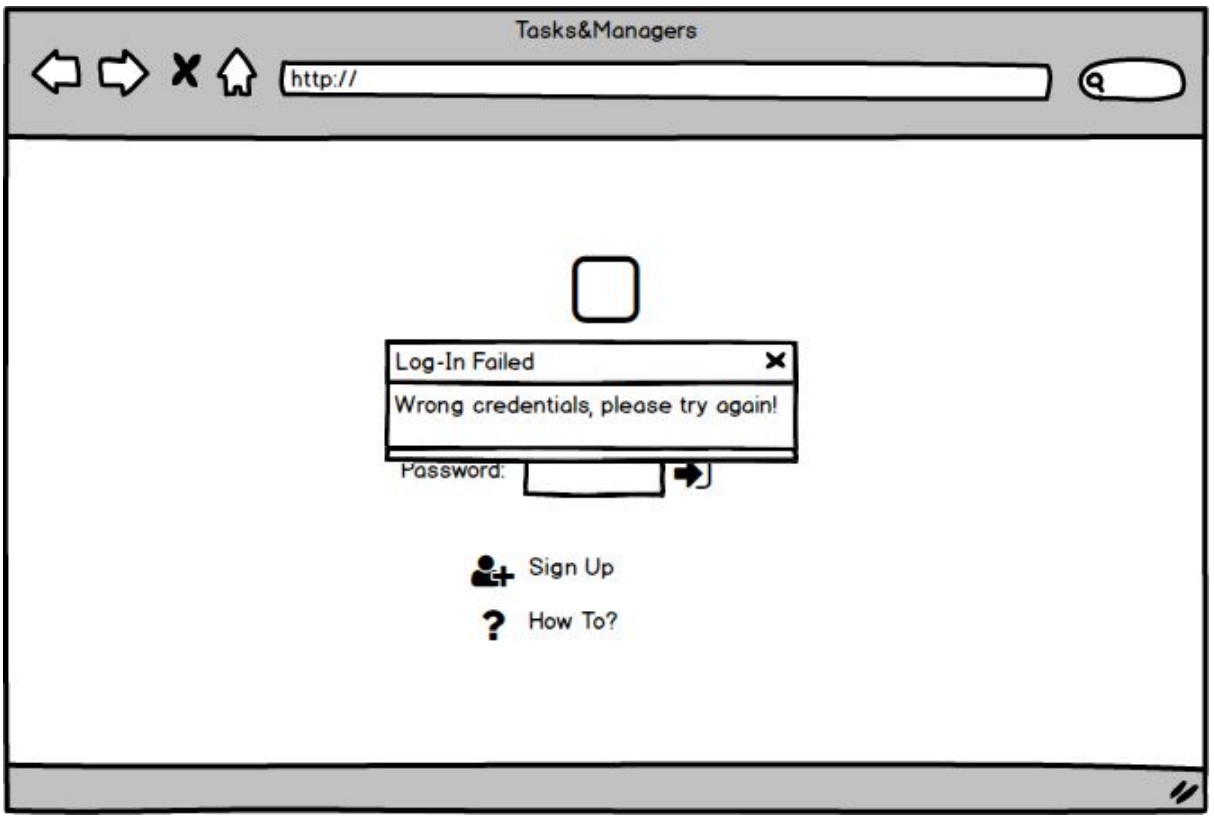

Figure 4: Login-Failed Page

5.1.2. How to Page

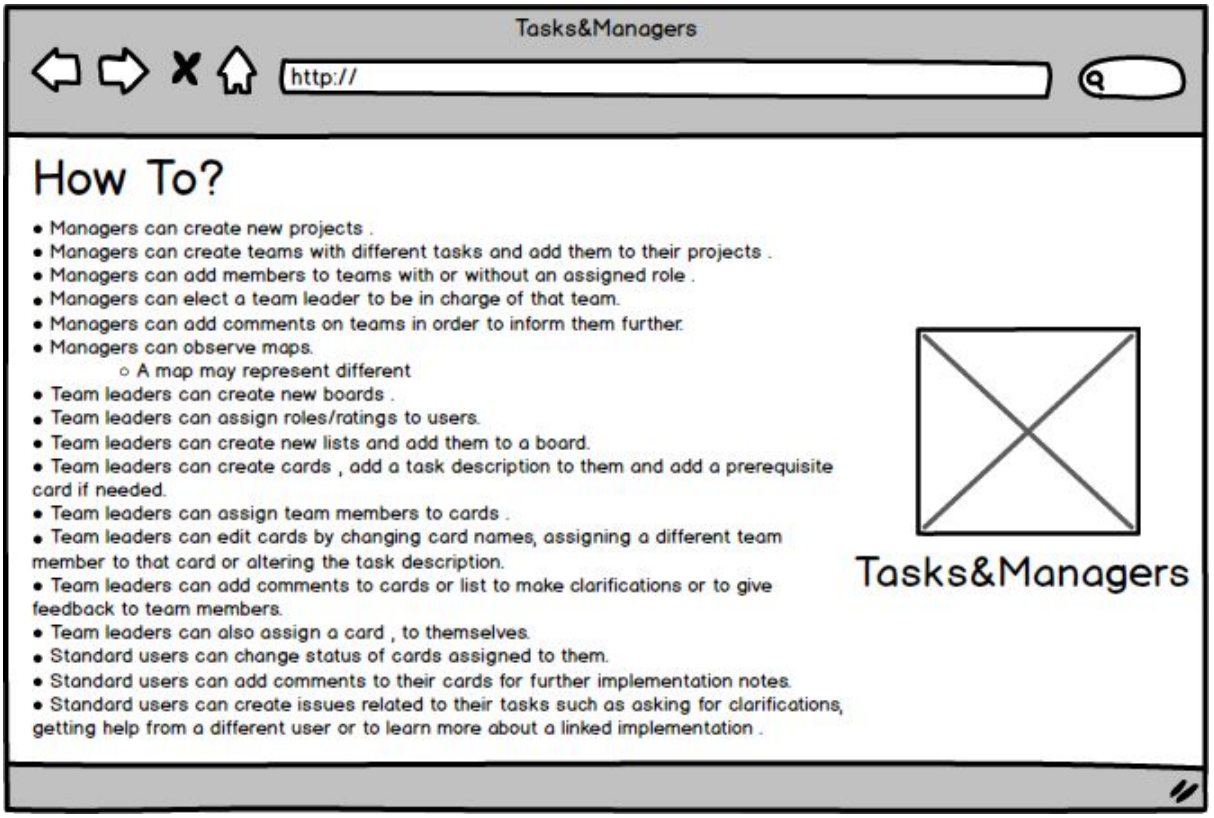

Figure 5: How To Page

This page contains simple information that can help new users to grasp Tasks&Managers. There are no queries in this page.

5.1.3. Register Page

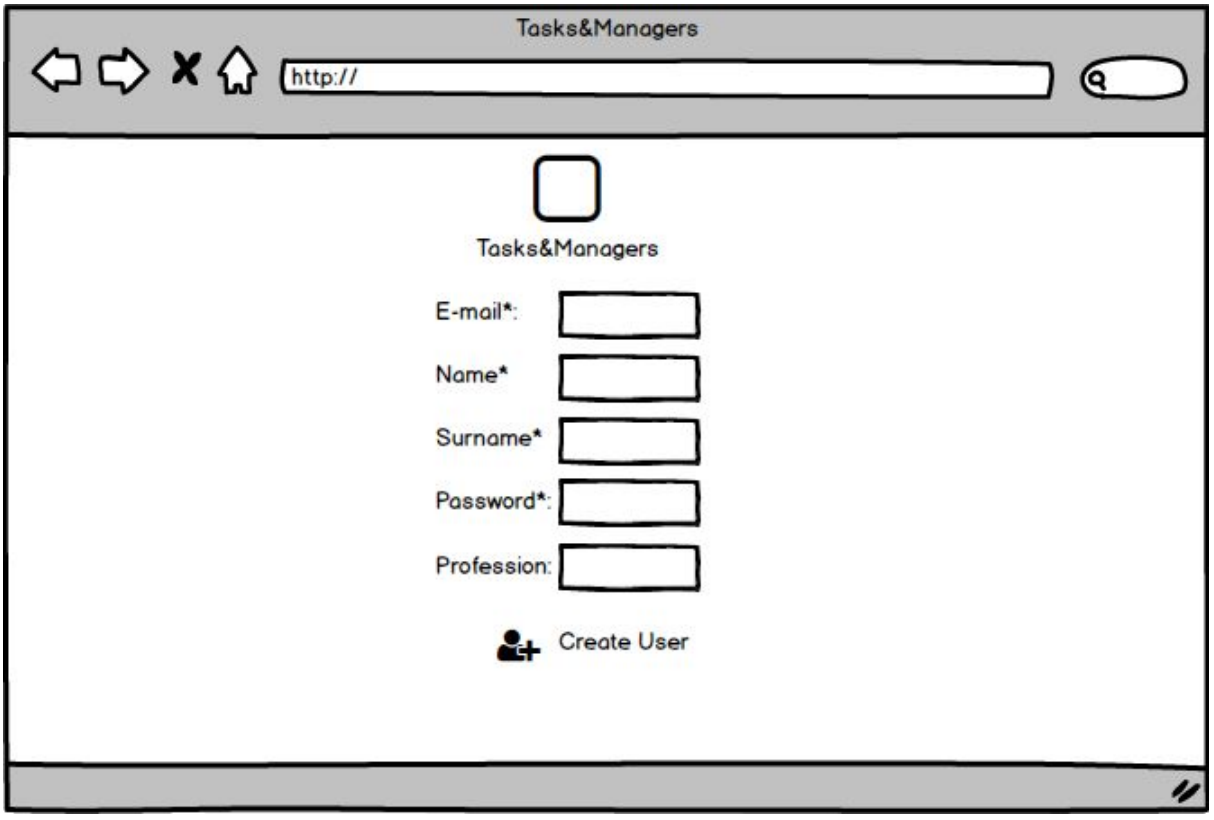

Figure 6: Register Page

SQL query for registering an account:
INSERT INTO User(email, first_name, last_name, password, profession)
VALUES (‘xxx@y.com’, ‘First’, ‘Second’, ’sEm4?l*3Fd1’, ‘Software Developer’);

5.1.4. Account Page

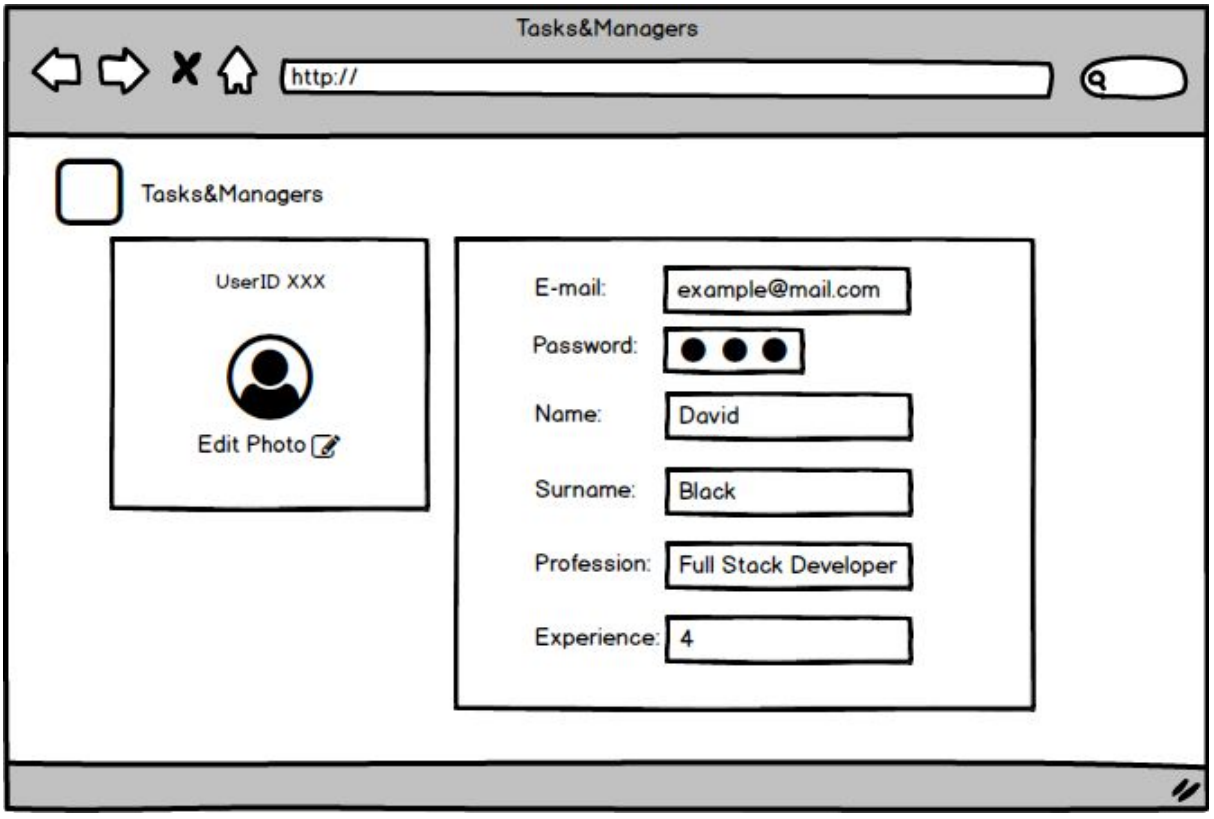

Figure 7: Account Page

SQL query to display user information:
SELECT id, email, profession, first_name, last_name, experience
FROM user
WHERE user.ID = 1980361342;

SQL query to change user email:
UPDATE user
SET user.email = 'newemail@yy.com'
WHERE user.ID = 1980361342;

SQL query to change user password:
UPDATE user
SET user.password = 'aGT4*-S2jM'
WHERE user.ID = 1980361342;

SQL query to change user profession:
UPDATE user
SET user.profession = 'Software Team Leader'
WHERE user.ID = 1980361342;

SQL query to change user's name:
UPDATE user
SET first_name 'Name', last_name, last_name = 'LastName'
WHERE ID = 1980361342;

SQL query to change user's experience:
UPDATE user
SET experience = 5
WHERE ID = 1980361342;

5.1.5. Filter Page

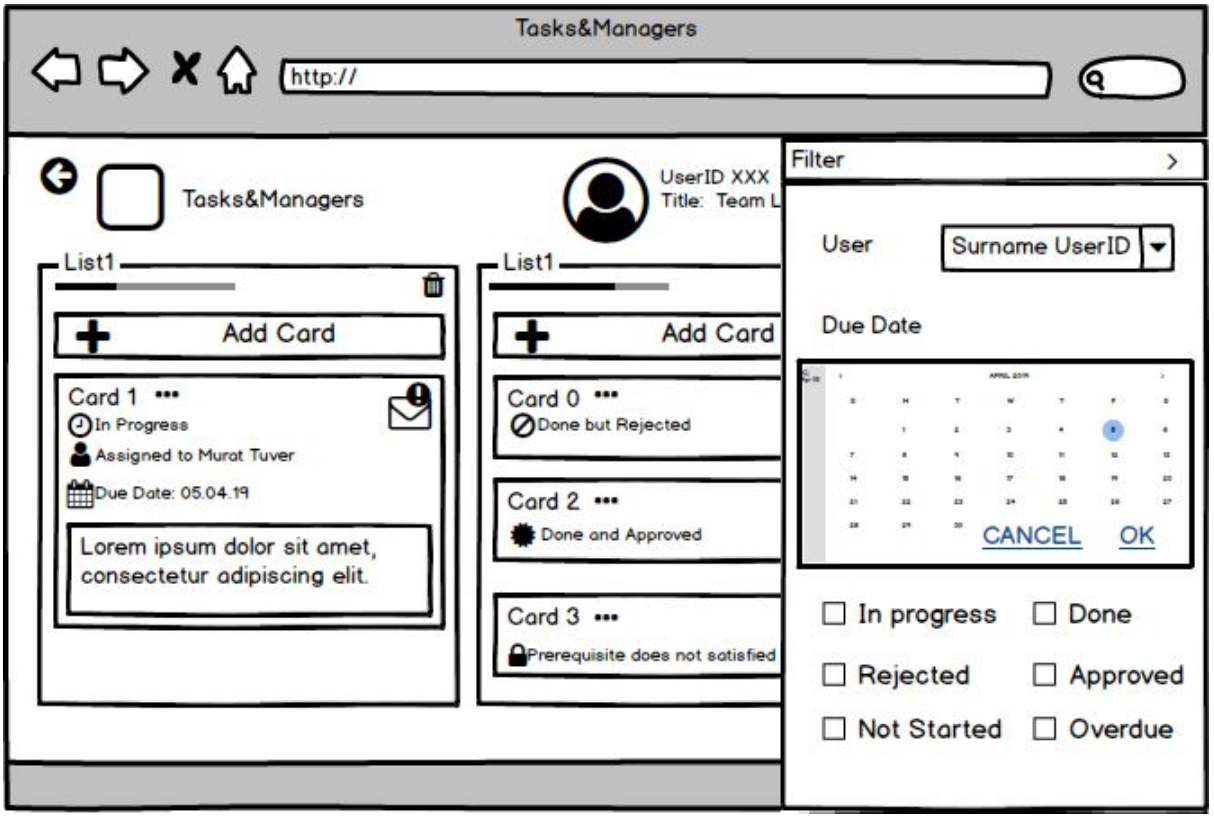

Figure 8: Filter Page

NOTE: Procedures used in following queries are implemented in the *section 6.5.*

SQL query to filter cards by status:
EXEC FilterCardStatus 'Done';

SQL query to filter cards by expired due date:
EXEC FilterCardDueExp;

SQL query to filter cards by due date not expired:

EXEC FilterCardDueOK;

SQL query to Sort cards by ascending due date:
EXEC SortCardDueAsc;

SQL query to sort cards by descending due date:
EXEC SortCardDueDes;

SQL query to sort cards by ascending issue date:
EXEC SortCardIssAsc;

SQL query to sort cards by descending issue date:
EXEC SortCardIssDesc;

SQL query to sort cards by assigned user:
SELECT name, description, issue_date, due_date, status, prereq_ID, list_ID
FROM card
WHERE assigned_ID = 23123;

SQL query to display card with desired due date:
SELECT name, description, issue_date, due_date, status, prereq_ID, list_ID
FROM card
WHERE due_date = ‘20190412’

5.1.6. New Project Page

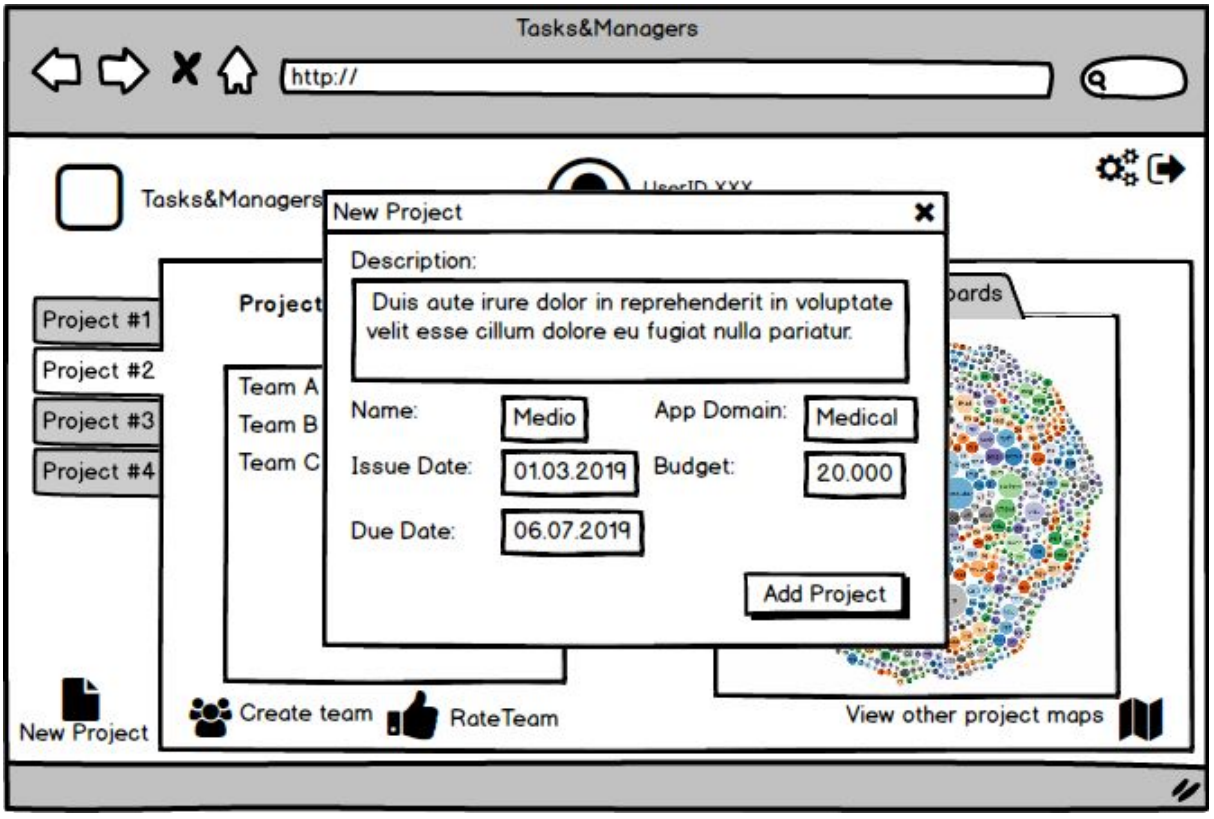

Figure 9: New Project Page

SQL query to add a new project:
INSERT INTO Manager(ID)
VALUES (1639027);

INSERT INTO Project(app_domain, issue_date, due_date, budget, name, description, manager_ID)
VALUES ('Medical', '20190103', ''20190706'', 15000, 'Project Name', 'A cool project description', 1639027);

5.2. Team Leader Pages

5.2.1. Team Leader Home Page

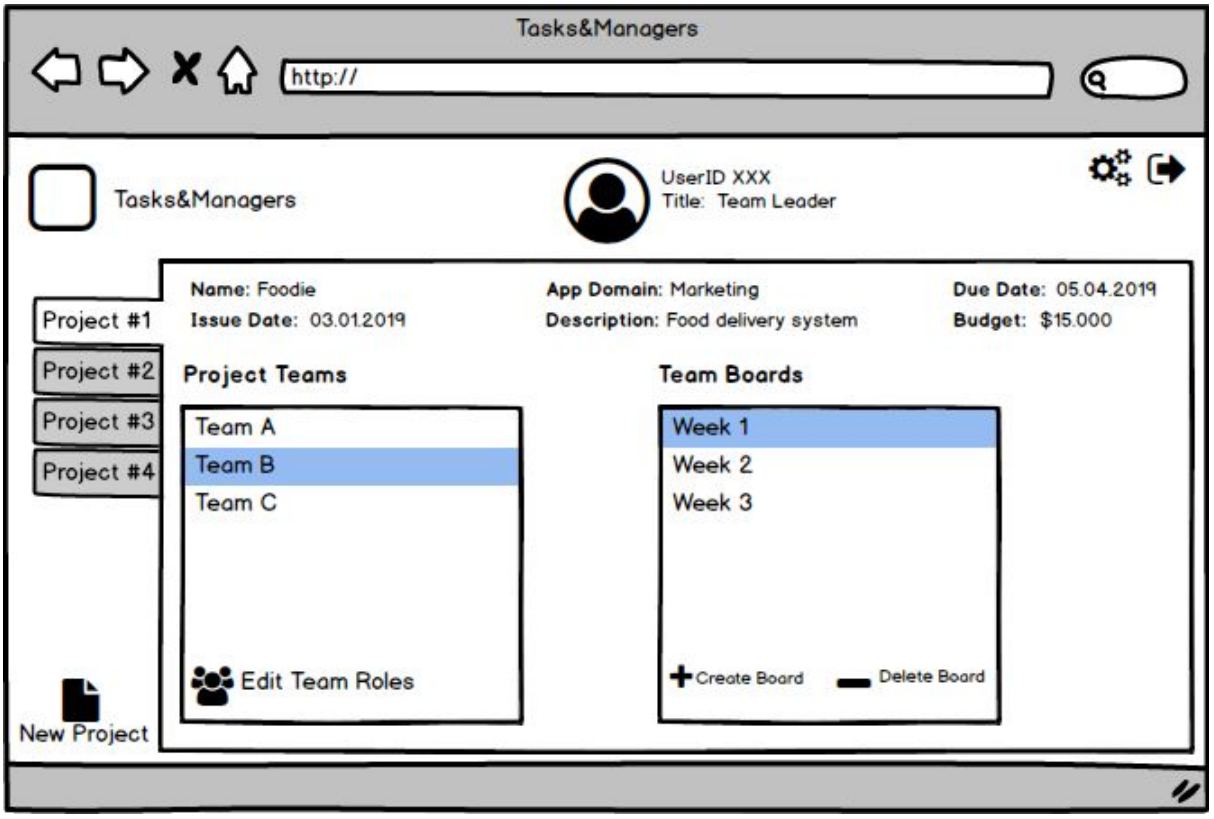

Figure 10: Team Leader Home Page

NOTE: This screen is shown to the user only if user is the team leader of clicked project's team. For example, in the screenshot, user is Project #1's Team B's team leader. Also, "Create Board" and "Delete Board" buttons only shown to user when user is the leader of that team.

Following 3 queries are used to determine user's authentication level.
SQL query to display user's title:

```
SELECT manager_ID
FROM project
WHERE manager_ID = 2376491604;

SELECT T.leader_ID
FROM project P, team T
WHERE P.ID = T.project_ID AND T.leader_ID = 2376491604;

SELECT M.member_ID
FROM team T, member M, project P
```

WHERE P.ID = T.project_ID AND M.team_ID = T.ID AND M.member_ID = 2376491604;

SQL query to display project names:

```
(SELECT name
FROM project
WHERE project.manager_ID = 2376491604)
UNION
(SELECT P.name
FROM member M, team T, project P
WHERE M.member_ID = 2376491604 AND M.team_ID = T.ID AND T.project_ID = P.ID)
UNION
(SELECT P.name
FROM team T, project P
WHERE T.leader_ID = 2376491604  AND T.project_ID = P.ID);
```

SQL query to display teams:

```
SELECT name
FROM team
WHERE project_ID = 2423;
```

SQL query to display boards:

```
SELECT name
FROM board
WHERE team_ID = 1470;
```

SQL query to create a new board:

```
INSERT INTO board(name, description, issue_date, due_date, team_ID)
VALUES ('newBoard', 'new tasks', '20190209', '20190307', 2834);
```

SQL query to delete a board:

```
DELETE FROM board
WHERE ID = 23287;
```

SQL query to display project details:

```
SELECT app_domain, issue_date, due_date, budget, name, description
FROM project
WHERE id=3746;
```

5.2.2. Team Leader Board Page

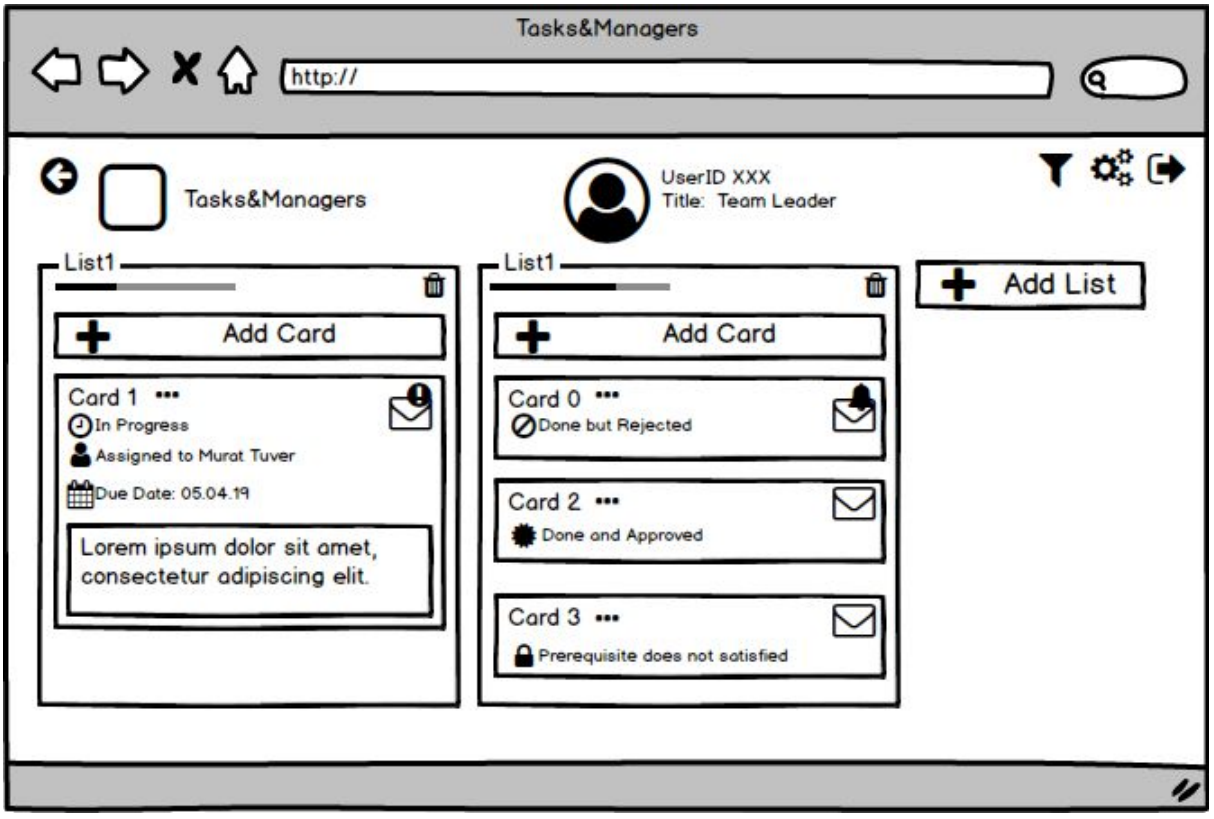

Figure 11: Team Leader Board Page

NOTE: Add card, add list buttons and delete list icons are only shown when user is a team leader.

SQL query to display lists:
SELECT name, description, issue_date, due_date
FROM list
WHERE ID = 223672;

SQL query to display a list's cards:
SELECT name, description, issue_date, due_date, status, prereq_ID, assigned_ID
FROM Card
WHERE list_ID = 155112;

SQL query to determine cards with issues:
SELECT DISTINCT card_ID
FROM issue;

SQL query to add a list:
INSERT INTO list (name, description, issue_date, due_date)

VALUES(‘a new list’, ‘side quests’, ‘20190109’, ‘20191308’);

SQL query to delete a list:
DELETE FROM list
WHERE ID = 244389;

SQL query to add card to a list:
INSERT INTO list (name, description, issue_date, due_date, prereq_ID)
VALUES(‘a new card’, ‘important task’, ‘201901017’, ‘20191308’, 238734);

5.2.3. Team Leader Card Operations

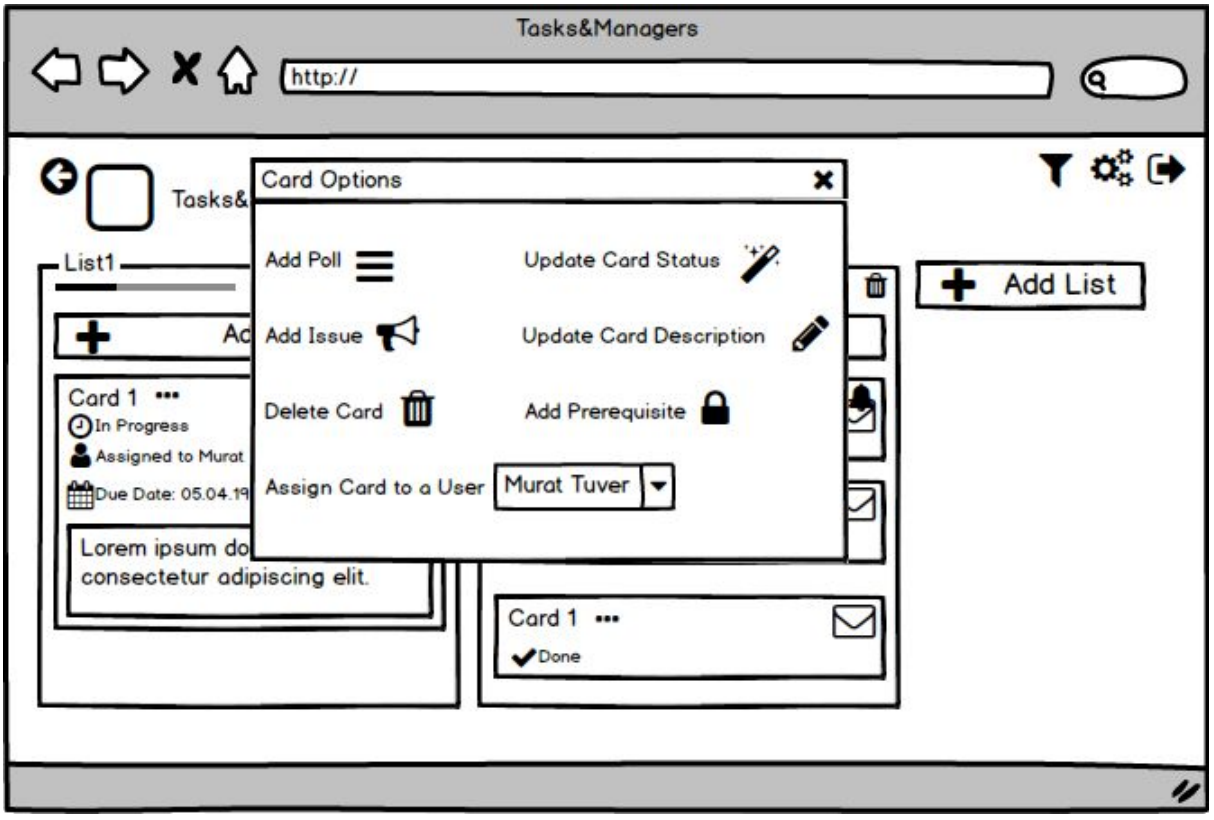

Figure 12: Team Leader Card Operations

5.2.3.1. Add Poll Page

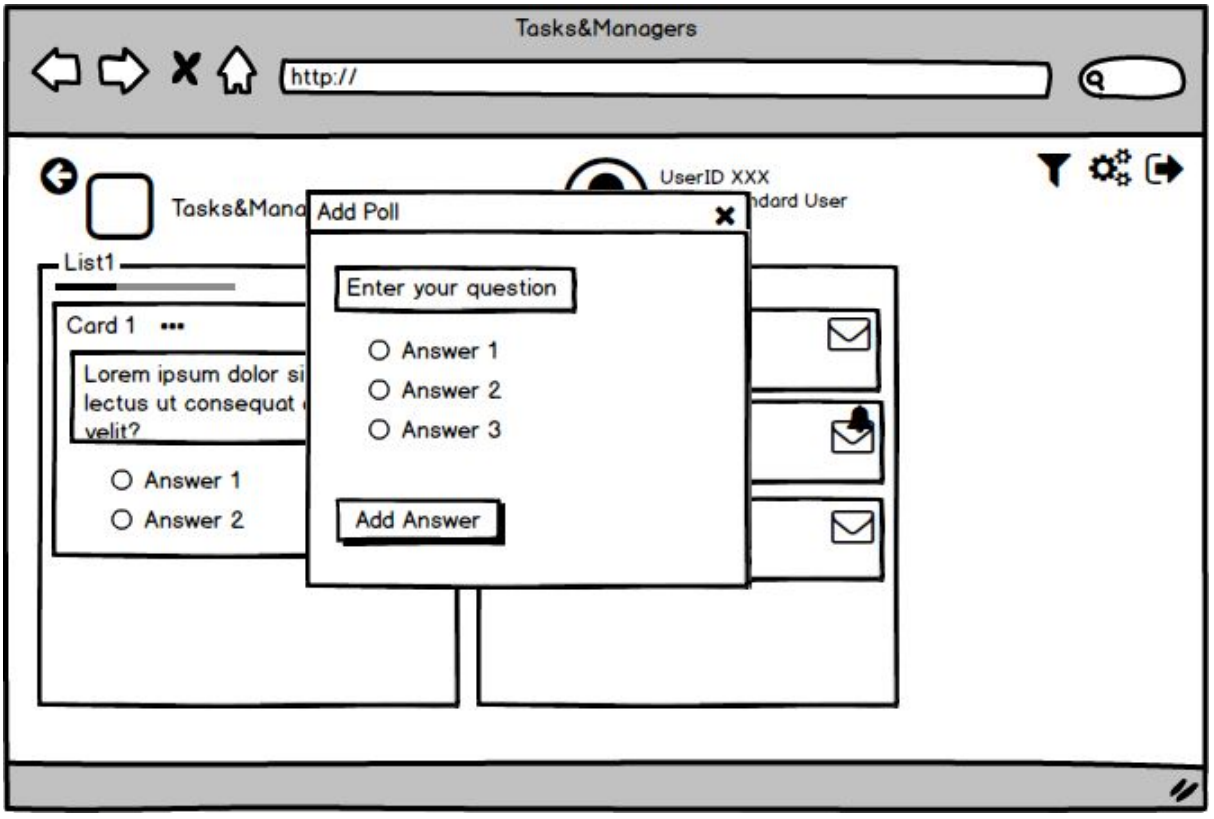

Figure 13: Add Poll Page

SQL query to add poll a card:
INSERT INTO poll (description, date, card_ID)
VALUES('Which language should we use?', '201901017', 1231);

SQL query to add answer to the poll:
INSERT INTO response(description)
VALUES('Java');

5.2.3.2. Add Issue Page

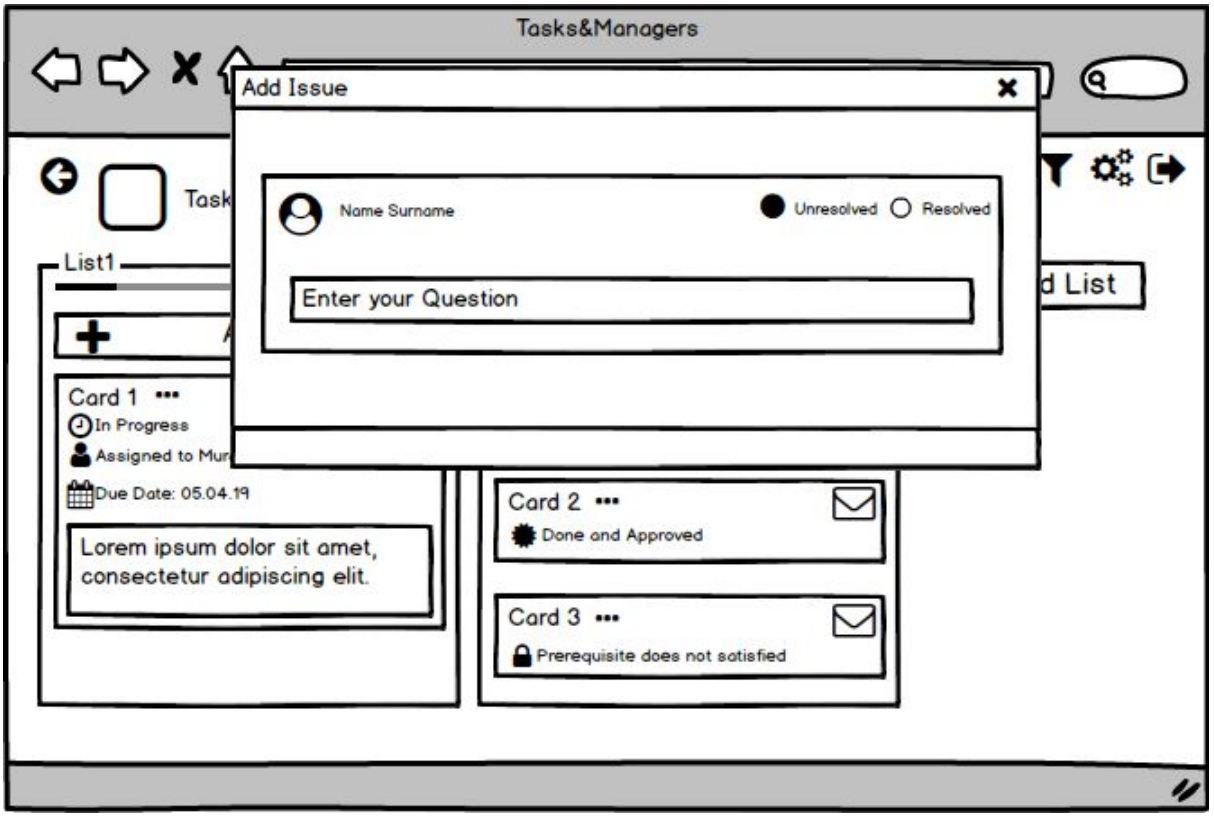

Figure 14: Add Issue Page

SQL query to add issue to a card:

INSERT INTO issue (name, description)
VALUES('Stack Overflow in recursion', 'Recursive algorithm gives overflow, need to consider another algorithm.')
WHERE card_ID = 123123;

5.2.3.3. Update Card Status

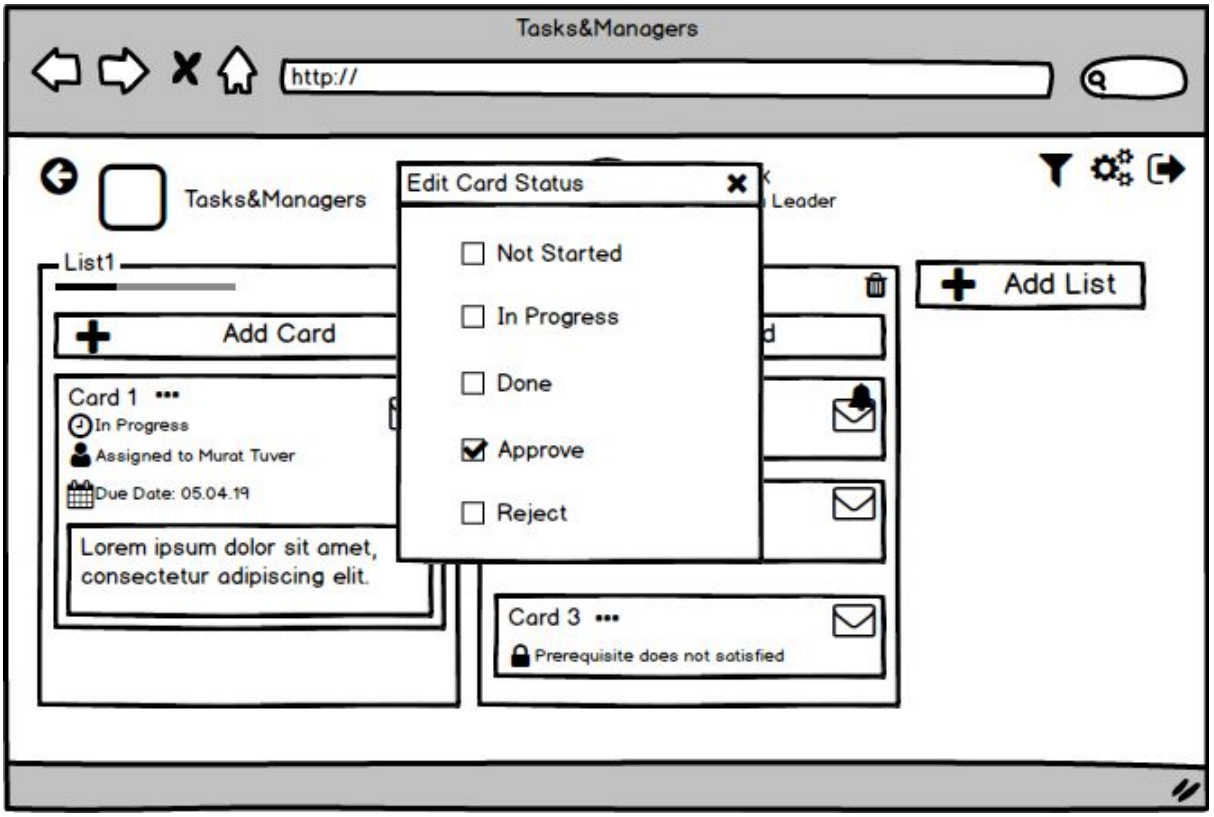

Figure 15: Update Card Status

SQL query to update card status:
UPDATE card
SET card.status = 'In Progress'
WHERE card.ID = 1980361342;

5.2.3.4. Update Card Description

SQL query to update card description:
UPDATE card
SET card.description = 'Hotfix: now implementation uses JavaScript!'
WHERE card.ID = 1980361342;

5.2.3.5. Delete Card

SQL query to delete card:
DELETE FROM card
WHERE ID = 23287;

5.2.3.6. Add Prerequisite

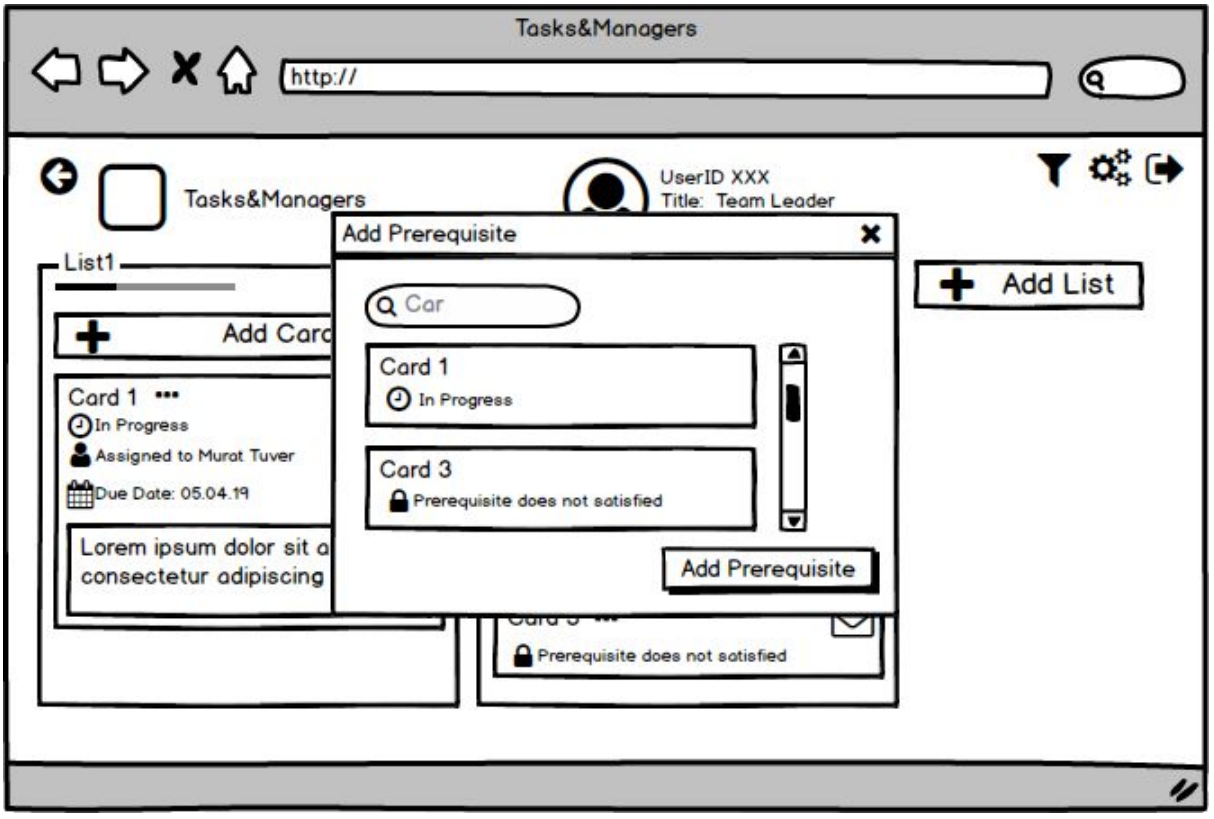

Figure 16: Add Prerequisite

SQL query to display all cards:

SELECT id, name, status, prereq_ID
FROM Card
WHERE Card.name LIKE 'Car%'
ORDER BY Card.name

SQL query to add prerequisite:

UPDATE card
SET card.prereq_ID = 198041223
WHERE card.ID = 1980361342;

5.2.3.7. Assign Card to a User

SQL query to assign card to a user:

UPDATE card
SET card.assinged_ID = 198041223
WHERE card.ID = 1980361342;

5.2.4. Edit Team Role Page

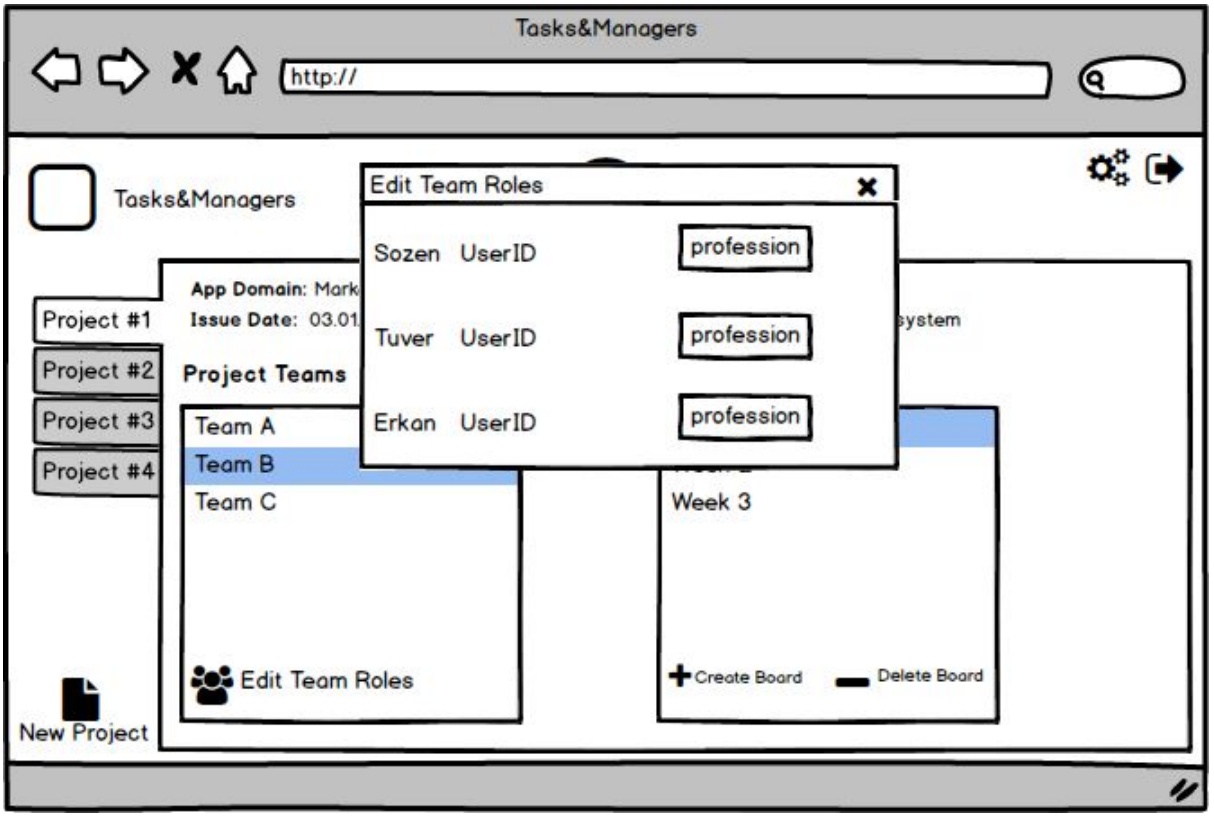

Figure 17 : Team Leader Edit Role Page

SQL query to edit role of a team member:

UPDATE member
SET member.role = ‘Database Server Developer’
WHERE member.member_ID = 1980361342;

5.3. Manager Pages

5.3.1 Manager Home Page

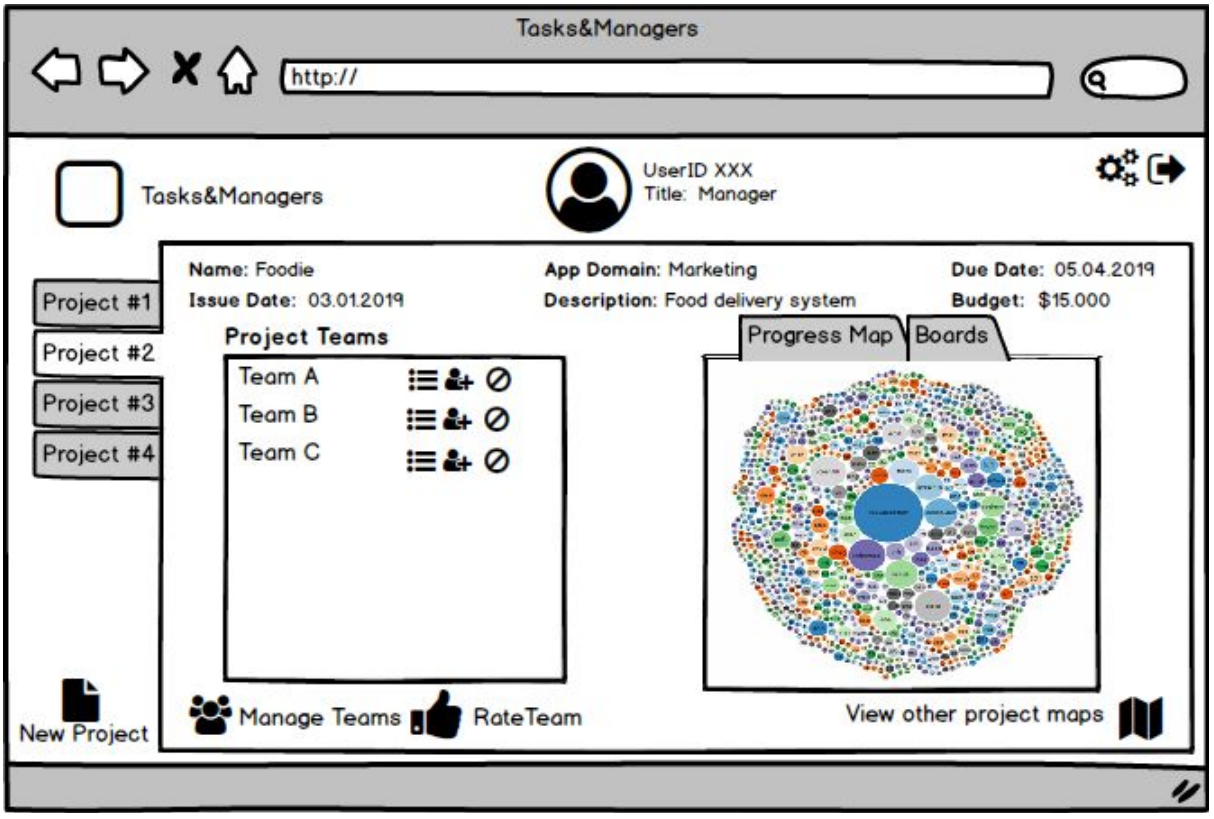

Figure 18: Manager Home Page

SQL query to display project names:

```
(SELECT name
FROM project
WHERE project.manager_ID = 2376491604)
UNION
(SELECT P.name
FROM member M, team T, project P
WHERE M.member_ID = 2376491604 AND M.team_ID = T.ID AND T.project_ID = P.ID)
UNION
(SELECT P.name
FROM team T, project P
WHERE T.leader_ID = 2376491604  AND T.project_ID = P.ID);
```

SQL query to display teams:

```
SELECT name
FROM team
WHERE project_ID = 2423;
```

SQL query to display boards:

```
SELECT name
FROM board
WHERE team_ID = 1470;
```

SQL query to create new project:

```
INSERT INTO project(app_domain, due_date, budget, name )
VALUES ('sample domain', '20200101', 20000, 'a project');
```

SQL query to view project map:

```
SELECT M.content
FROM  map M, Project P
WHERE M.project_ID = P.ID AND P.ID = 2423
```

SQL query to delete a team:

```
DELETE FROM team
WHERE team.ID = (  SELECT ID
                   FROM team
                   WHERE project_ID = 2423 AND name = 'Team B');
```

SQL query to rate a team:

```
UPDATE Worker
SET Worker.user_rating = 9
WHERE Worker.ID = 1980361342;
```

5.3.2 Manager Manage Team Pop-up

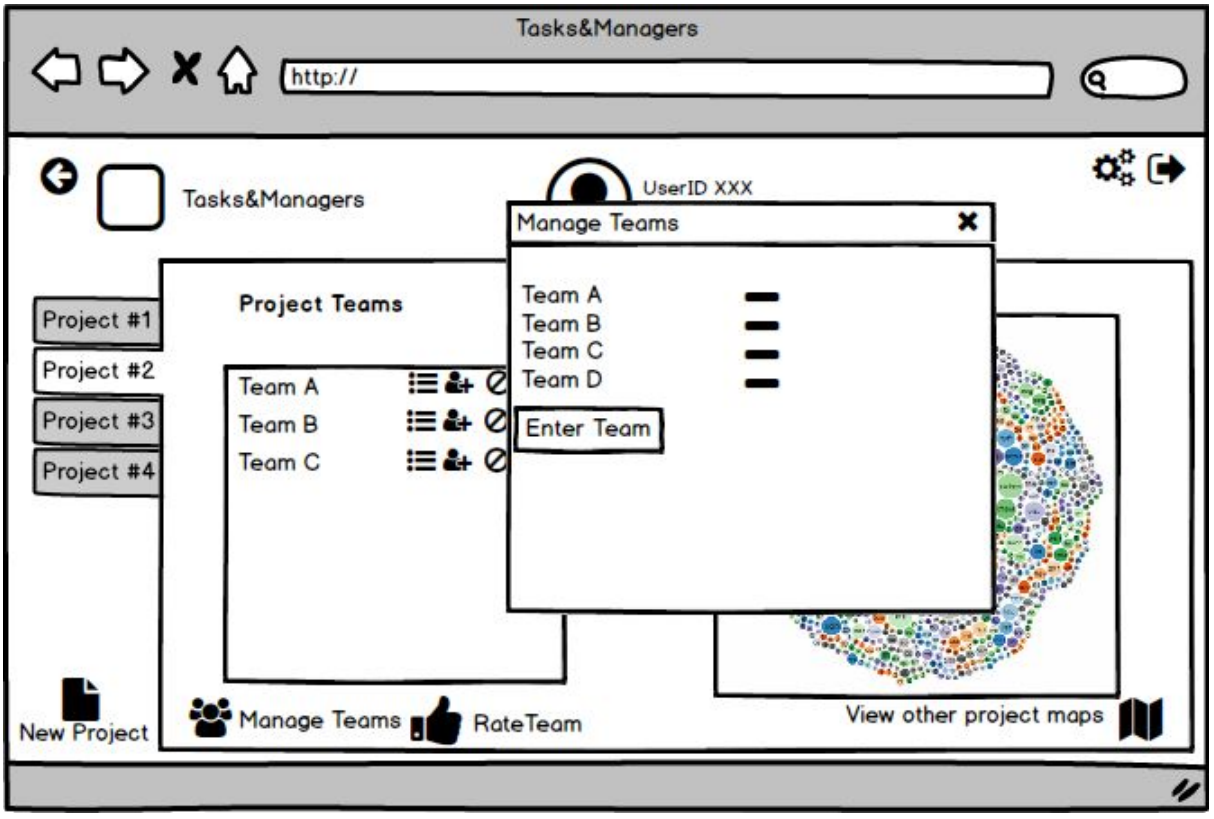

Figure 19: Manager Manage Team Pop-up

SQL query to add a team:

INSERT INTO team(department, description, name, project_ID)
VALUES ('Software', 'UI', 'UI Team', 21378243);

SQL query to assign a team leader:

DELETE FROM member
WHERE member_ID = 2372360197 AND team_ID = 23722;

UPDATE team
SET leader_ID = 2372360197
WHERE ID = 23722;

5.3.3 Manager Manage Team Members Page

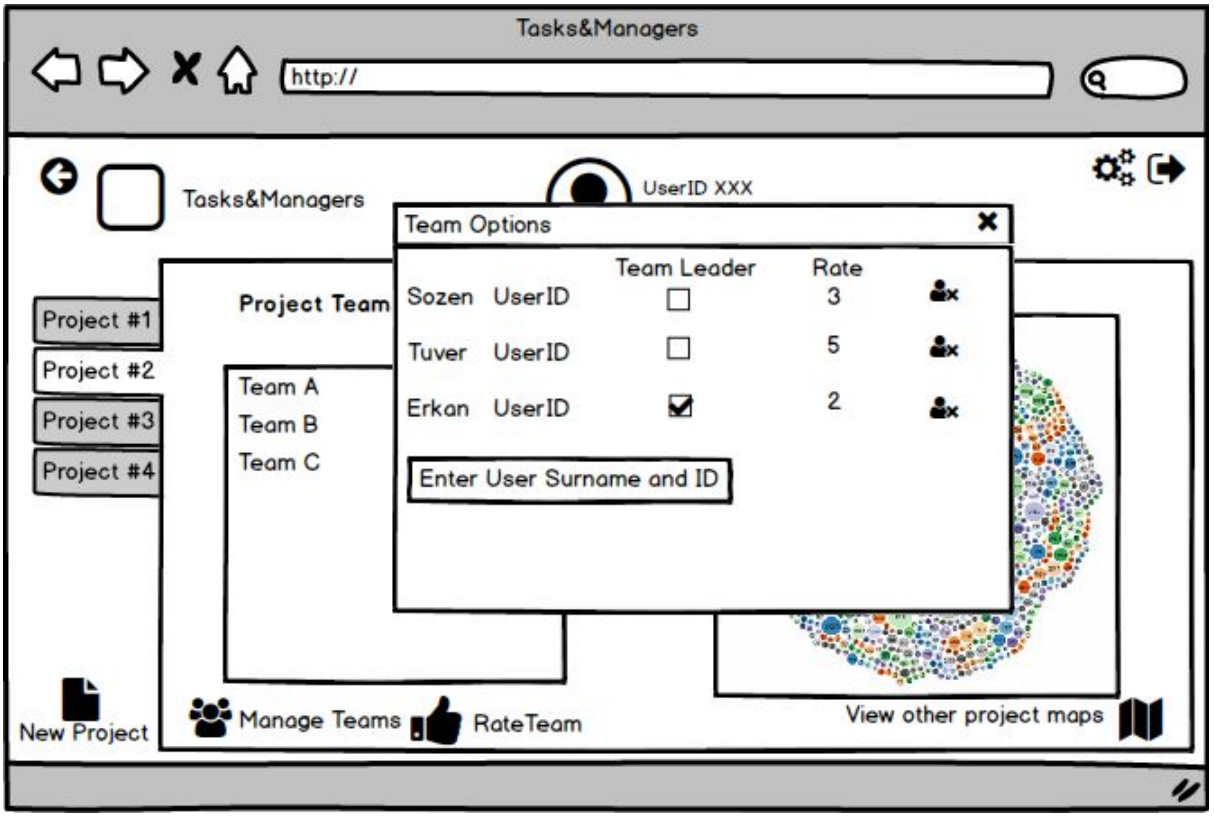

Figure 20: Manager Manage Team Members Pop-up

SQL query to display members of a team:
SELECT M.member_ID, U.last_name
FROM member M, User U
WHERE M.team_ID = 2387 AND U.ID = M.member_ID;

SQL query to add a member to team:
INSERT INTO member(member_ID, role, team_ID)
VALUES (5730792741, 'Test Engineer', 2387);

SQL query to delete a member from a team:
DELETE FROM member
WHERE member_ID = 8375791980 AND team_ID = 2387;

SQL query to display user ratings:
EXEC GetUserRating 435647

5.3.4. Manager Board Page

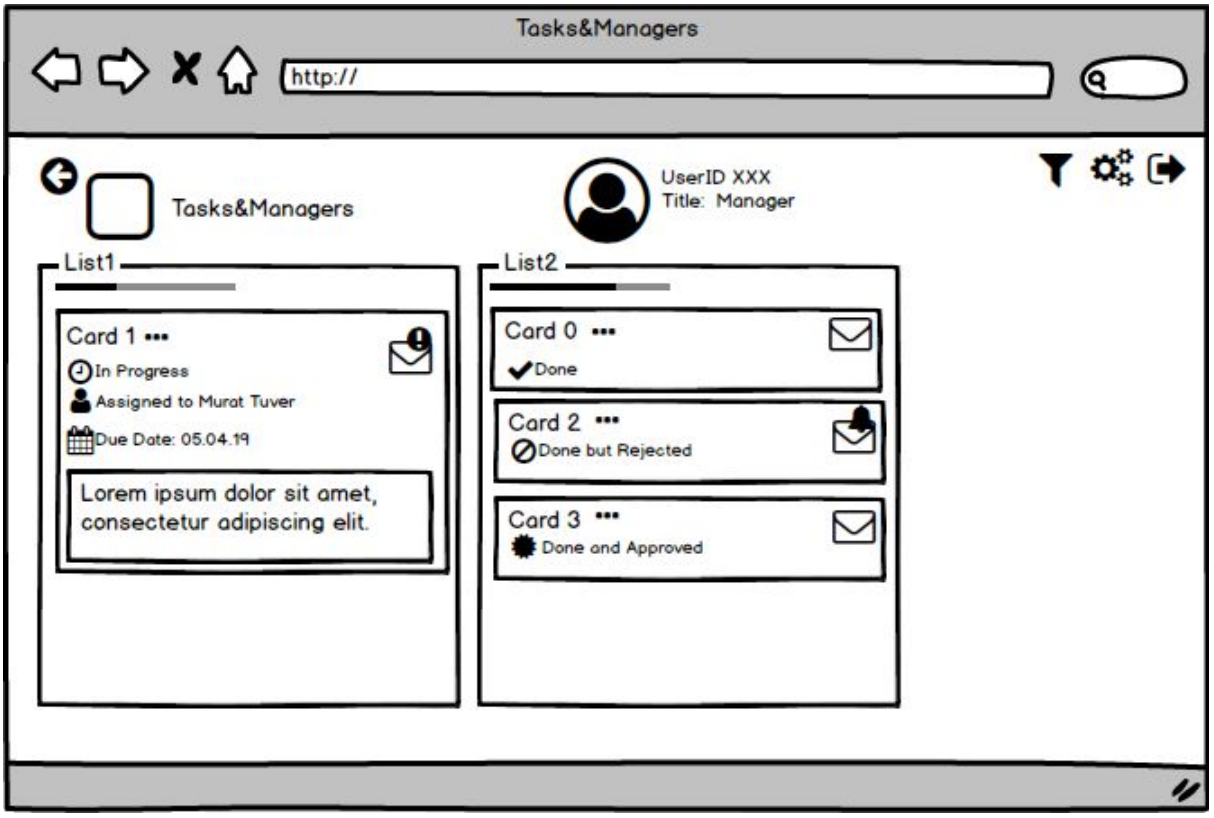

Figure 21: Manager Board Page

NOTE: Queries of the view card is same with section *5.2.2. Team Leader Board Page,* therefore they are not mentioned here again.

5.3.5. Manager Rate Teams Page

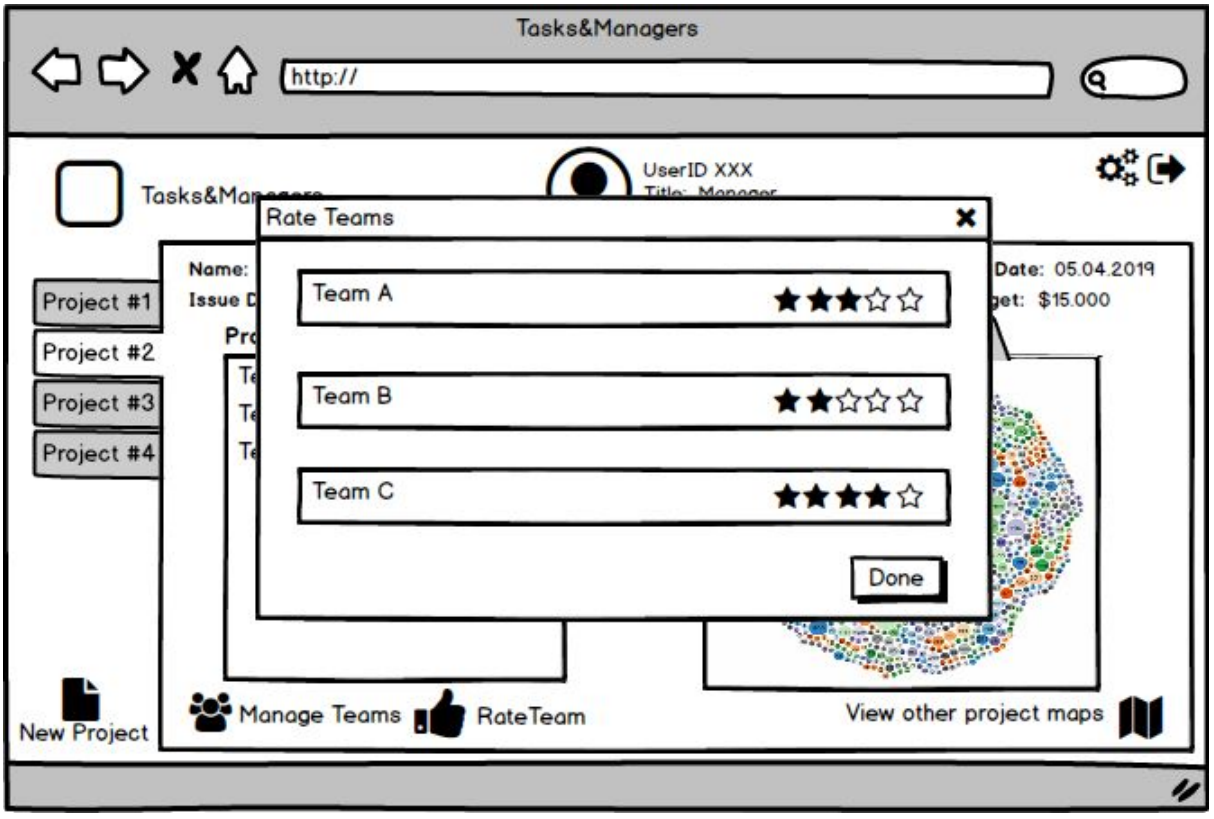

Figure 22: Manager Rate Teams Page

SQL query to give rating to a team:

```
UPDATE team
SET team.rating = 5
WHERE ID = 23722;
```

5.4. Standard User Pages

5.4.1. Standard User Home Page

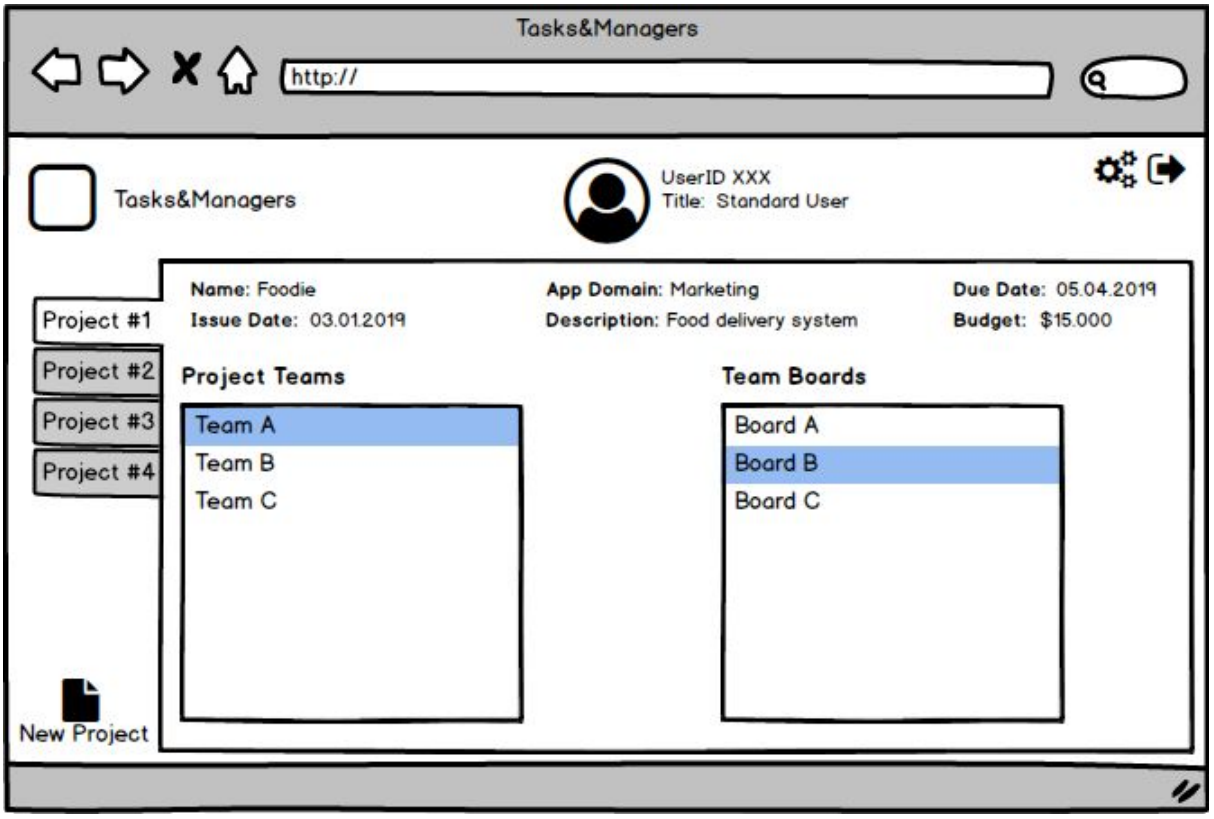

Figure 23: Standard User Home Page

NOTE: This screen is same with team leader's home page except, standard users cannot create or delete a board. Moreover, standard users can only enter their team's boards.

Following 3 queries are used to determine user's authentication level.
SQL query to display user's title:

```
SELECT manager_ID
FROM project
WHERE manager_ID = 2376491604;

SELECT T.leader_ID
FROM project P, team T
WHERE P.ID = T.project_ID AND T.leader_ID = 2376491604;

SELECT M.member_ID
FROM team T, member M, project P
WHERE P.ID = T.project_ID AND M.team_ID = T.ID AND M.member_ID = 2376491604;
```

SQL query to display project names:

```
(SELECT name
FROM project
WHERE project.manager_ID = 2376491604)
UNION
(SELECT P.name
FROM member M, team T, project P
WHERE M.member_ID = 2376491604 AND M.team_ID = T.ID AND T.project_ID = P.ID)
UNION
(SELECT P.name
FROM team T, project P
WHERE T.leader_ID = 2376491604  AND T.project_ID = P.ID);
```

SQL query to display teams:

```
SELECT name
FROM team
WHERE project_ID = 2423;
```

SQL query to display boards:

```
SELECT name
FROM board
WHERE team_ID = 1470;
```

SQL query to display project details:

```
SELECT app_domain, issue_date, due_date, budget, name, description
FROM project
WHERE id=3746;
```

5.4.2. Standard User Card Operations

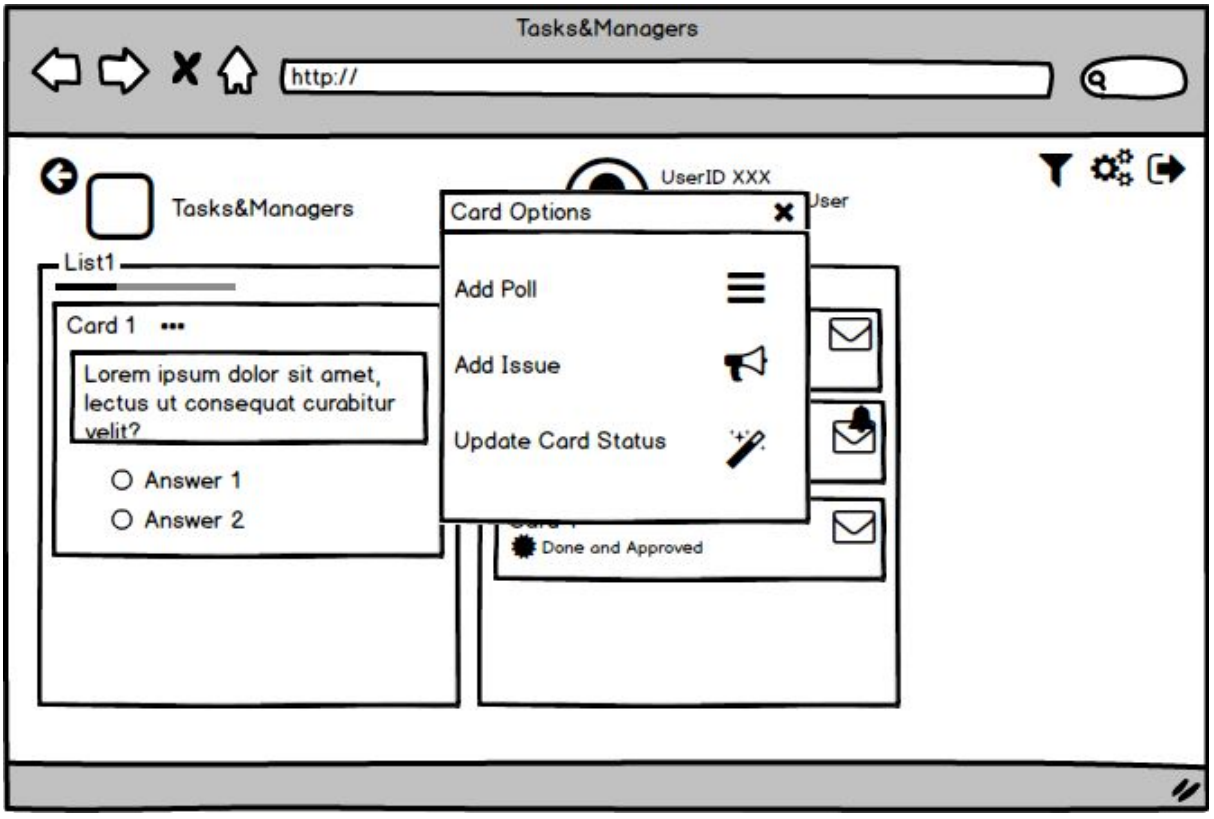

Figure 24: Standard User Card Operations

NOTE: Queries of these add poll, add issue, update card status are same with section *5.2.3. Team Leader Card Operations,* therefore they are not mentioned here again.

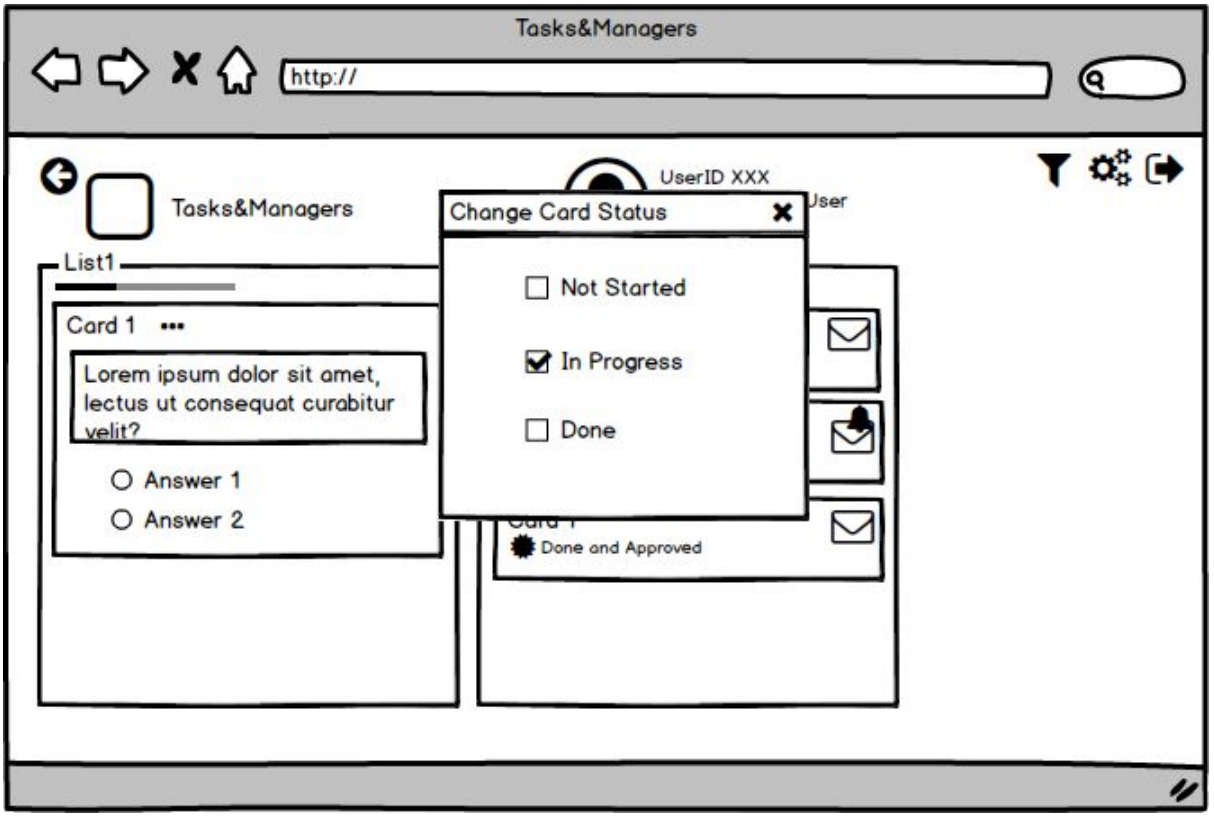

Figure 25 : Standard User Change Card Status Page

5.4.3. Standard User Board Page

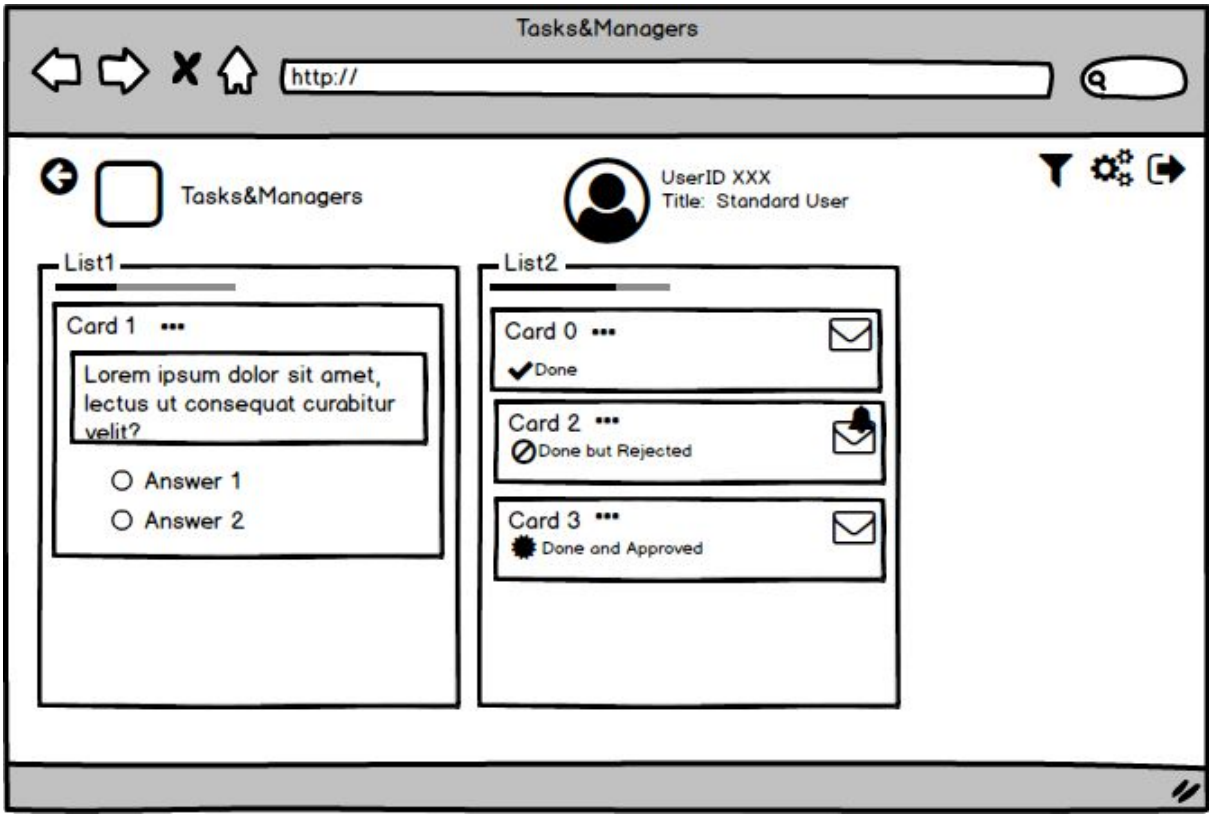

Figure 26: Standard User Board Page

SQL query to answer a poll question:

```
UPDATE response
SET no_of_votes = no_of_votes + 1
WHERE poll_ID = (SELECT ID
                 FROM poll
                 WHERE card_id = 1239524);
```

5.5. Worker Pages(Team Leaders and Standard Users)

5.5.1. Issue Page

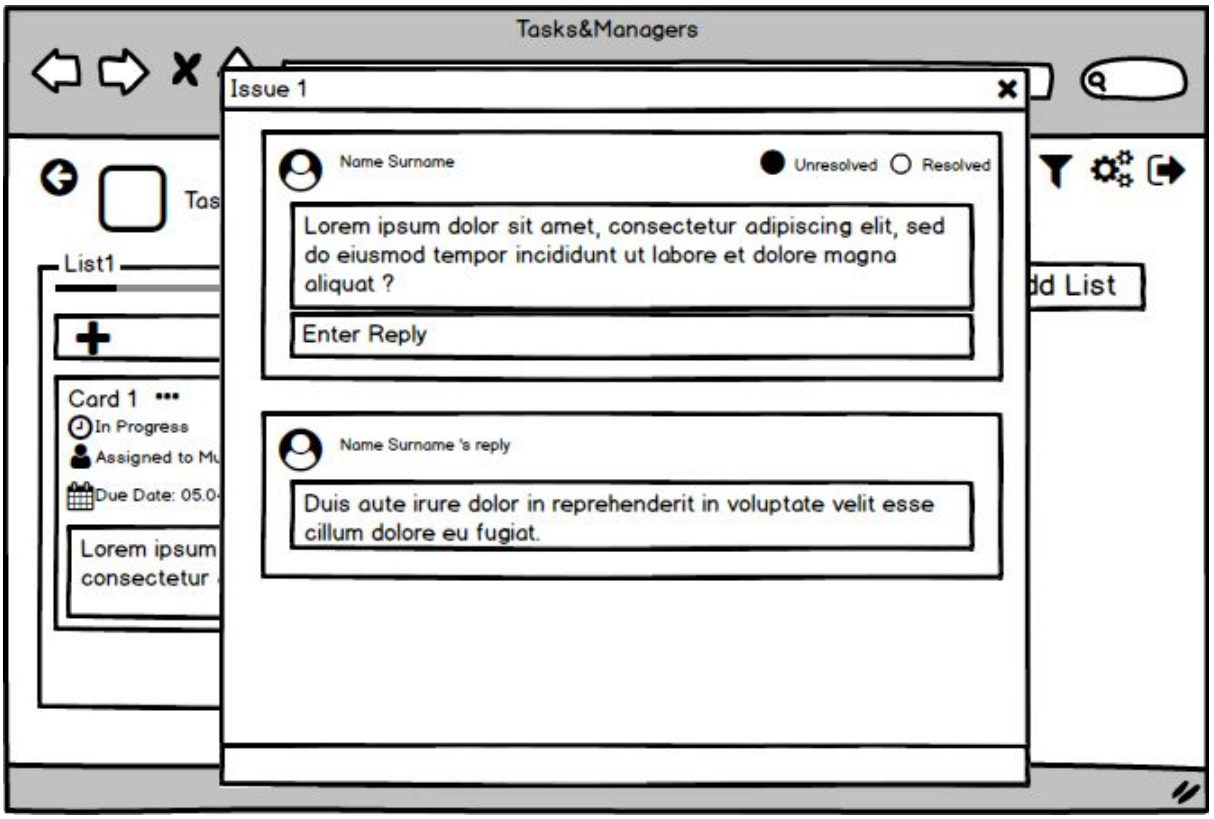

Figure 27: Issue Page

NOTE: Issue is marked as resolved when issue's creator picks the best answer.

SQL query to display issues:

```
SELECT name, description
FROM Issue
WHERE card_ID = 12863;
```

SQL query to display answers given to the issue:

```
SELECT DISTINCT Answer.ID, Answer.description, User.first_name, User.last_name
FROM Answer, User
WHERE Answer.issue_ID = 1744 AND Answer.user_ID = User.ID;
```

6. Advanced Database Components

6.1. Reports

6.1.1. Number of Completed Cards In A Project In a Week

This report will calculate the number of approved cards in each project.

SQL:

```
WITH PTeam( project_ID, team_ID ) AS
(
        SELECT project_ID, team_ID
        FROM Team
),
TBoards ( project_ID, board_ID)
(
        SELECT P.Project_ID, B.board_ID
        FROM PTeam as P, Board B
        WHERE P.team_ID = B.team_ID
),
BList ( project_ID, list_ID)
(
        SELECT P.project_ID, L.list_ID
        FROM TBoards P, List L
        WHERE P.board_ID = L.board_ID
),
LCard  ( project_ID, card_ID )
(
        SELECT P.project_ID, C.card_ID
        FROM BList P, Card C
        WHERE (P.list_ID = C.list_ID
        AND
        C.status = “APPROVED”
        AND
        C.issue_date  - CURRENT_TIME <= 7)
)
SELECT project_ID, count(card_ID)
FROM LCard
GROUP BY ProjectID;
```

6.1.2. Most Popular Application Domains

This report will generate a sorted table of Application domains and the Projects done upon them.

SQL:

```
CREATE VIEW DomainPopularity AS (
        SELECT app_domain, count(ID)
        FROM Project AS P
        GROUP BY app_domain
);
```

6.2. Views

For simplicity, we used dummy values that are going to be obtained from user interaction from the frontend side when the system is deployed. (For example M.member_ID = 2376491604 , in the real system, the query will rely on a ID obtained from the system instead of 2376491604)

6.2.1. Map Views

Map Views will be used to represent statistical data about a Project, a Team etc. to be used to create maps to be showed in Manager screens.

6.2.1.1. Number of Teams a User is In

This view will generate a sorted table of Worker ID's and the number of teams they are in.

SQL:

```
CREATE VIEW UserInTeamNo AS (
        SELECT member_ID, COUNT(team_ID) AS team_no
        FROM Member
        GROUP BY member_ID
        ORDER BY team_no DESC
);
```

6.2.1.2. Number of Projects a Manager is Managing

This view will generate a sorted table of Manager ID's and the number of projects they manage.

SQL:

```
CREATE VIEW ManageProjectNo AS (
        SELECT manager_ID, COUNT(ID) AS project_no
        FROM Project
        GROUP BY manager_ID
        ORDER BY project_no DESC
);
```

6.2.1.3. Number of Teams a Leader is Leading

This view will generate a sorted table of Team Leader ID's and the number of teams they are the Team Leader of.

SQL:

```
CREATE VIEW LeadTeamNo AS (
        SELECT leader_ID, COUNT(ID) AS team_no
        FROM Team
        GROUP BY leader_ID
        ORDER BY team_no DESC
);
```

6.2.1.4. Number of Issues in a Card

This view will generate a sorted table of Card ID's and the number of issues they have.

SQL:

```
CREATE VIEW CardIssueNo AS (
        SELECT cardID, COUNT(ID) AS issue_no
        FROM Issue
        GROUP BY card_ID
        ORDER BY issue_no DESC
);
```

6.2.1.5. Number of Approved Cards of A User

This view will generate a sorted table of Worker ID's and the number of approved Cards they have.

SQL:

```
CREATE VIEW ApprovedCardUser AS (
        SELECT assigned_ID, COUNT(ID) AS card_no
        FROM Card
```

```
        WHERE status = 'APPROVED'
        GROUP BY assigned_ID
        ORDER BY card_no DESC
);
```

6.2.1.6. Number of Approved Cards of Department

This view will generate a sorted table of Departments and the number of approved Cards they have.

SQL:

```
CREATE VIEW ApprovedCardDep AS (
        SELECT department, count(ID)
        FROM Card AS C INNER JOIN ( SELECT department, member_ID
                                    FROM Member M INNER JOIN Team T ON
                                    M.team_ID = T.ID
                                    ) AS D
        ON D.member_ID = C.assigned_ID
        GROUP BY D.department
);
```

6.2.1.7. Number of Approved Cards of Projects

This view will generate a sorted table of Projects and the number of approved Cards they have.

SQL:

```
WITH PTeam( project_ID, team_ID ) AS
(
        SELECT project_ID, team_ID
        FROM Team
),
TBoards ( project_ID, board_ID)
(
        SELECT P.Project_ID, B.board_ID
        FROM PTeam as P, Board B
        WHERE P.team_ID = B.team_ID
),
BList ( project_ID, list_ID)
(
        SELECT P.project_ID, L.list_ID
```

```
        FROM TBoards P, List L
        WHERE P.board_ID = L.board_ID
),
LCard  ( project_ID, card_ID )
(
        SELECT P.project_ID, C.card_ID
        FROM BList P, Card C
        WHERE (P.list_ID = C.list_ID
        AND
        C.status = “APPROVED” )
)
SELECT project_ID, count(card_ID)
FROM LCard
GROUP BY ProjectID;
```

6.2.2. Card Views

Card views are common in content for all users, thus a single view is adequate for the whole system.

SQL:

```
SELECT name, description, issue_date, due_date, status, prereq_ID, assigned_ID
FROM Card
WHERE list_ID = 155112;
```

6.2.3. Member Views

Member views are only viewable by a Manager, thus a single view is adequate for the whole system.

SQL:

```
SELECT M.member_ID, U.last_name
FROM member M, User U
WHERE M.team_ID = 2387 AND U.ID = M.member_ID;
```

6.2.4. Team Views

Member views are only viewable by a Manager, thus a single view is adequate for the whole system.

SQL:

```
SELECT name
FROM team
```

```
WHERE project_ID = 2423;
```

6.2.5. Project Views

Project views are viewed by each user with a common content. Any user will see all the projects they are a part of regardless of their user type in that Project.

SQL:

```
(SELECT name
FROM project
WHERE project.manager_ID = 2376491604)
UNION
(SELECT P.name
FROM member M, team T, project P
WHERE M.member_ID = 2376491604 AND M.team_ID = T.ID AND T.project_ID = P.ID)
UNION
(SELECT P.name
FROM team T, project P
WHERE T.leader_ID = 2376491604  AND T.project_ID = P.ID);
```

6.3. Triggers

6.3.1. Prerequisite Card Trigger

When a Card is updated, the system should not allow this operation if it's prerequisite cards status is not "Approved". In this case, the system should roll back.

SQL:

```
CREATE TRIGGER PrereqTrigger AFTER UPDATE OF status ON Card
REFERENCING NEW ROW AS nrow
REFERENCING OLD ROW AS orow
IF ( SELECT C.status FROM Card C WHERE  C.ID = orow.prereq_id) <> "Approved")
BEGIN
        ROLLBACK
END;
```

6.4. Constraints

- The system cannot be used without signing in with an already signed up User account.
- An Issue can have at most 1 approved Answer.

- A User can vote for a single response to a Poll.
- The status of a Card can not be changed if its Prerequisite Cards status is not “Approved”.
- A Standard User cannot create a Card, List or Board.
- A Map can only be viewed by a Manager.
- Only a Manager can rate a Team to rate a whole Project Group. (But all users can be managers of a Project they create, since they are no privilege specification during sign-up.)
- A Team can only be created by a Manager.
- A Team Leader can only be assigned by a Manager.
- A Manager and a Team Leader can add Standard Users to a Team.
- A Worker can only see the boards of the Teams they are a part of.
- A Manager can not see the Boards/Lists/Cards of a Project.
- All users will see the Project list they are a Manager/Worker of.
- A Worker will see the teams and corresponding Boards of the Teams they are a part of.
- Only the assigned User of a Card can mark an Issue as Resolved by selecting an Answer.
- A new Answer can not be added to resolved Issues.
- Only a Team Leader can delete/edit Boards, Lists and Cards.
- A Standard User can only update the status of a Card.
- There can be at most one prerequisite Card of a Card.

6.5. Stored Procedures

For functionality and reusability, Filtering of Cards and Sorting of Cards by different parameters is designed as a procedure.

6.5.1. Filtering Cards

6.5.1.1. Filtering Cards by Status according to Option Chosen

SQL:

```
 CREATE PROCEDURE FilterCardStatus @Option VARCHAR(20),
@Option VARCHAR(20)
AS
        SELECT *
        FROM Card
        WHERE Status LIKE @Option
GO;
```

6.5.1.2. Filtering Cards by Expired Due Date

SQL:

```
CREATE PROCEDURE FilterCardDueExp
AS
        SELECT *
        FROM Card
        WHERE due_date < CURRENT_DATE
GO;
```

6.5.1.3. Filtering Cards by Due Date Not Expired

SQL:

```
CREATE PROCEDURE FilterCardDueOK
AS
        SELECT *
        FROM Card
        WHERE  CURRENT_DATE < due_date
GO;
```

6.5.2. Sorting Cards

6.5.2.1. Sorting Cards by Ascending Due Date

SQL:

```
CREATE PROCEDURE SortCardDueAsc
AS
        SELECT *
        FROM Card
        ORDER BY   due_date ASC
GO;
```

6.5.2.2. Sorting Cards by Descending Due Date

SQL:

```
CREATE PROCEDURE SortCardDueDes
AS
        SELECT *
        FROM Card
        ORDER BY   due_date DESC
GO;
```

6.5.2.3. Sorting Cards by Ascending Issue Date

SQL:

```
CREATE PROCEDURE SortCardIssAsc
AS
        SELECT *
        FROM Card
        ORDER BY  issue_date ASC
GO;
```

6.5.2.4. Sorting Cards by Descending Issue Date

SQL:

```
CREATE PROCEDURE SortCardIssDes
AS
        SELECT *
        FROM Card
        ORDER BY  issue_date DESC
GO;
```

6.5.3. Calculate User Rating

SQL:

```
CREATE PROCEDURE GetUserRating @Option INTEGER,
@Option INTEGER
AS
        WITH UserRates (user_id, rate) AS
        (
                SELECT M.member_ID, T.rating
                FROM Member M , Team T
                WHERE ( M.team_id = T.ID )
                UNION ALL
                SELECT T.leader_ID, T.rating
                FROM  Team T
        )
        SELECT user_id, avg(rate)
        FROM UserRates UR
        WHERE user_id = @Option
GO;
```

7. Implementation Plan

For the backend web development of Tasks&Managers, PHP will be used along with MySQL as the database management system. For the front-end development, HTML and CSS will be used.

8. Website

Project reports will be available on:

https://caglasozen.github.io/Tasks-Managers/