BILKENT UNIVERSITY

CS353 DATABASE SYSTEMS

GROUP ID: 28

UptownFANK Project Tracking Software

Design Report

Authors:	*Student IDs:*
Fatbardh Feta	21600334
Alemdar Salmoor	21500430
Naisila Puka	21600336
Kunduz Efronova	21600469

April 5, 2019

Contents

1 Introduction **4**

2 Revised ER Diagram **4**

2.1 Diagram Updates 4

2.2 ER Diagram 6

3 Relation Schemas **7**

3.1 BasicUser 7

3.2 Phone 7

3.3 SuperUser 8

3.4 Team 9

3.5 Member 9

3.6 Board 10

3.7 WorksOn 11

3.8 List 11

3.9 Card 12

3.10 PerformsTask 13

3.11 CheckList 13

3.12 Item 14

3.13 Comment 15

3.14 Replies 16

3.15 Upvotes 16

3.16 Attachment 17

3.17 Label 18

3.18 Labelling 19

4 Functional Dependencies and Normalization **19**

5 Functional Components **19**

5.1 Use Case Diagram 20

5.2 Use Case Scenarios 21

5.2.1 CreateTeam . . . 21
5.2.2 ManageTeam . . . 21
5.2.2.1 DeleteTeam . . . 22
5.2.2.2 AddMember . . . 22
5.2.2.3 RemoveMember . . . 23
5.2.3 CreateBoard . . . 23
5.2.4 AddList . . . 24
5.2.5 SetUserPrivileges . . . 24
5.2.6 ManageBoard . . . 25
5.2.6.1 ManageCards . . . 25
5.2.6.2 ManageLists . . . 26
5.2.6.3 Archive . . . 26
5.2.6.4 AssignUser . . . 27
5.2.7 AddAttachment . . . 27
5.2.8 AddLabel . . . 28
5.2.9 ManageAttachments . . . 28
5.2.10 ManageLabels . . . 29
5.2.11 AddChecklist . . . 29
5.2.11.1 CompleteItem . . . 30
5.2.12 Comment . . . 30
5.2.12.1 Reply . . . 31
5.2.13 Register . . . 31
5.2.13.1 RegisterAsSuperUser . . . 32
5.2.13.2 RegisterAsBasicUser . . . 32
5.3 Data Structures . . . 33

6 User Interface Design and Corresponding SQL Statements 34
6.1 Register and Login Functionalities . . . 35
6.1.1 Register Screen . . . 35
6.1.1.1 BasicUser . . . 35
6.1.1.2 SuperUser . . . 36
6.1.2 Login Screen . . . 37
6.2 View Main Personal Info . . . 38

6.2.1 Personal Homescreen 38
6.3 View Boards/Lists/Cards 40
6.3.1 View Board Screen 40
6.3.2 View Card Screen 41
6.4 Create Team and Board 43
6.4.1 Create Team Screen 43
6.4.2 Create Board Screen 44
6.5 Create List and Card 45
6.5.1 Create List Screen 45
6.5.2 Create Card Screen 46
6.6 Add CheckLists, Attachments, Labels 47
6.6.1 Create CheckList Screen 47
6.6.2 Create Label Screen 48

7 Advanced Database Components 48
7.1 Reports 49
7.2 Views 50
7.2.1 Owned Boards 50
7.2.2 Users Assigned to a Card 50
7.2.3 Cards a User Is Assigned to 50
7.3 Triggers 51
7.4 Constraints 51
7.5 Stored Procedures 52

8 Implementation Plan 53

9 Online Access 53

10 Conclusion 53

1 Introduction

This is the design report for our project tracking software called UptownFank. UptownFank aims to be a simple useful tool for tracking different projects, usually shared between some team members. It will pave the way for maintaining project communication and keeping track of the accomplished and to-do tasks. The design report starts the explanation of the design of the database, followed by the functional components, user interface design and lastly, advanced database components.

2 Revised ER Diagram

2.1 Diagram Updates

We made the following changes to our proposed ER diagram:

Deletions:

- Since our project tracking software will create the possibility for superusers to create different boards, we realized that having both Project and Board entities was redundant since a project turned out to be equivalent to a board. Therefore, we deleted Project entity and from this point forward we identify a project with a board.
- As a consequence of previous deletion, we deleted Contains and Owns binary relationships because they were related to the Project entity in one side.
- We have deleted the following statistical attributes:
  - cardsNo from List entity
  - size from Team entity
  - itemsNo from CheckList
  - upvotesNo and repliesNo from Comment

These attributes can be retrieved by a simple count query, so they formed redundant information in our database.

Updates:

- We changed Comment entity from strong to weak entity, since a comment depends on its parent card. In other words, a comment cannot exist without a card that it belongs to.
- Our system was previously using "maintains" relationship between a Board and a Superuser entity. However, this was ambiguous. Therefore, we changed from "maintains" to "owns". This means the a Board is owned by exactly one Superuser entity, and that specific superuser has all maintenance rights upon that board (create, delete, update, etc.).

Insertions:

- Our system was not providing a way for a BasicUser to upvote a specific comment. For this reason, we added "upvotes" relation between BasicUser and Comment entities to make this possible.
- We deleted the Project entity, but this entity had two important attributes: requirements for any specific project obligations and estimatedTime for giving an idea of the size of the project (e.g. a 3-month project is a small one which could be part of a term project for a specific university course). Since we identified a project with a board, we added these two attributes to the Board entity.
- The system was not saving any CheckList items. For this reason, we added Item entity as a weak one depending on CheckList. That is, each item exists only within a CheckList entity. We also added a "complete" relationship between Item and BasicUser entity in order to save who completed the item.
- The system now provides a proper "Reply" functionality to a comment. Not only a BasicUser can upvote, but he/she can also reply to a specific comment, provided by our additional "replies" relationship between a Comment and another Comment entity.

2.2 ER Diagram

Here we present our revised entity-relationship diagram created in Draw.io [1], based on the updates explained.

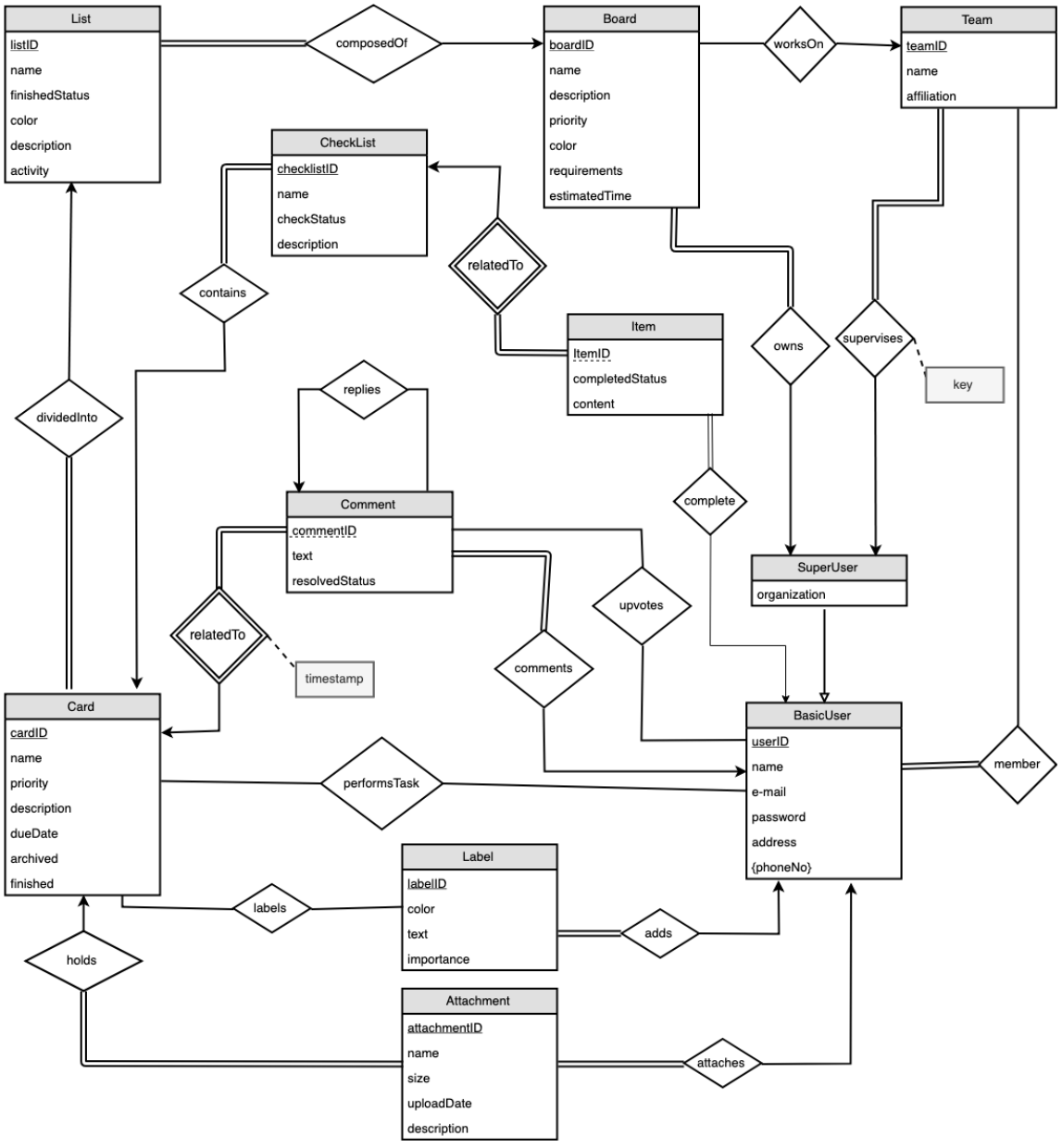

Figure 1: Entity-Relationship Diagram

3 Relation Schemas

In this section we will "convert" the system's ER diagram into a set of relations, where for each relation attribute domains, candidate keys, primary key and foreign keys are specified, as well as their normal form.

3.1 BasicUser

Relational Model: BasicUser(<u>userID</u>, name, email, password, address)

Functional Dependencies: {(userID → name, email, password, address), (email → userID)}

Primary Key: {userID}

Candidate Keys: {{userID}, {email}}

Foreign Keys: {}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE BasicUser(
    userID INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL,
    address VARCHAR(50) NOT NULL
);
```

3.2 Phone

Relational Model: Phone(<u>userID, phoneNo</u>)

Functional Dependencies: {}

Primary Key: {userID, phoneNo}

Candidate Keys: {{userID, phoneNo}}

Foreign Keys: {(userID → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Phone(
    userID INTEGER NOT NULL,
    phoneNumber VARCHAR(20) NOT NULL,
    PRIMARY KEY (userID, phoneNumber),
    FOREIGN KEY userID references BasicUser(userID)
);
```

3.3 SuperUser

Relational Model: SuperUser(<u>userID</u>, organization)

Functional Dependencies: {(userID → organization)}

Primary Key: {userID}

Candidate Keys: {{userID}}

Foreign Keys: {(userID → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE SuperUser(
    userID INTEGER PRIMARY KEY AUTO_INCREMENT,
    organization VARCHAR(200) NOT NULL,
    FOREIGN KEY (userID) REFERENCES BasicUser(userID)
);
```

3.4 Team

Relational Model: Team(teamID, name, affiliation, supervisor, key)

Functional Dependencies: {(teamID → name, affiliation, supervisor, key), (key → teamID)}

Primary Key: {userID}

Candidate Keys: {{userID}, {key}}

Foreign Keys: {(supervisor → SuperUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Team(
    teamID INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    affiliation VARCHAR(200) NOT NULL,
    supervisor INTEGER NOT NULL,
    key INTEGER UNIQUE,
    FOREIGN KEY supervisor REFERENCES SuperUser(userID)
);
```

3.5 Member

Relational Model: Member(userID, teamID)

Functional Dependencies: {}

Primary Key: {userID, teamID}

Candidate Keys: {{userID, teamID}}

Foreign Keys: {(userID → BasicUser.userID), (teamID → Team.teamID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Member(
    userID INTEGER NOT NULL,
    teamID INTEGER NOT NULL,
    PRIMARY KEY (userID, teamID),
    FOREIGN KEY userID REFERENCES BasicUser(userID),
    FOREIGN KEY teamID REFERENCES Team(teamID)
);
```

3.6 Board

Relational Model: Board(boardID, name, description, priority, color, requirements, estimatedTime, ownerID)

Functional Dependencies: {(boardID → name, description, priority, color, requirements, estimatedTime, ownerID)}

Primary Key: {boardID}

Candidate Keys: {{boardID}}

Foreign Keys: {(owner → SuperUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Board(
    boardID INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    description VARCHAR(1000) NOT NULL,
    priority INTEGER CHECK(priority >= 0 AND priority <=5),
    color VARCHAR(50) DEFAULT 'YELLOW' NOT NULL,
    requirements VARCHAR(1000) NOT NULL,
    estimatedTime DATETIME NOT NULL,
    FOREIGN KEY ownerID REFERENCES SuperUser(userID));
```

3.7 WorksOn

Relational Model: WorksOn(boardID, teamID)

Functional Dependencies: {(boardID → teamID)}

Primary Key: {boardID}

Candidate Keys: {{boardID}}

Foreign Keys: {(boardID → Board.boardID), (teamID → Team.teamID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE WorksOn(
    boardID INTEGER PRIMARY KEY NOT NULL,
    teamID INTEGER NOT NULL,
    FOREIGN KEY boardID REFERENCES Board(boardID),
    FOREIGN KEY teamID REFERENCES Team(teamID)
);
```

3.8 List

Relational Model: List(listID, name, finishedStatus, color, description, activity, boardID)

Functional Dependencies: {(listID → name, finishedStatus, color, description, activity, boardID)}

Primary Key: {listID}

Candidate Keys: {{listID}}

Foreign Keys: {(boardID → Board.boardID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE List(
    listID INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    finishedStatus VARCHAR(5),
    color VARCHAR(30) DEFAULT 'YELLOW' NOT NULL,
    description VARCHAR(1000),
    activity VARCHAR(1000),
    FOREIGN KEY boardID REFERENCES Board(boardID),
    CHECK(finishedStatus IN('True', 'False')),
);
```

3.9 Card

Relational Model: Card(cardID, name, priority, description, dueDate, listID, archived, finished)

Functional Dependencies: {(cardID → name, priority, description, dueDate, listID, archived, finished)}

Primary Key: {cardID}

Candidate Keys: {{cardID}}

Foreign Keys: {(listID → List.listID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Card(
    cardID INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    priority INTEGER CHECK (priority <= 6 AND priority >= 0),
    description VARCHAR(1000) NOT NULL,
    dueDate DATETIME NOT NULL,
    archived VARCHAR(5) CHECK(archived IN('True', 'False')),
```

```
    finished VARCHAR(5) CHECK(finished IN('True', 'False')),
    FOREIGN KEY listID REFERENCES List(listID)
);
```

3.10 PerformsTask

Relational Model: PerformsTask(cardID, userID)

Functional Dependencies: {}

Primary Key: {cardID, userID}

Candidate Keys: {{cardID, userID}}

Foreign Keys: {(cardID → Card.cardID), (userID → User.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE PerformsTask(
    cardID INTEGER,
    userID INTEGER,
    PRIMARY KEY (cardID, userID),
    FOREIGN KEY userID REFERENCES User(userID),
    FOREIGN KEY cardID REFERENCES Card(cardD)
);
```

3.11 CheckList

Relational Model: CheckList(checklistID, name, checkStatus, description, relatedCard)

Functional Dependencies: {(checklistID → name, checkStatus, description, relatedCard)}

Primary Key: {checklistID}

Candidate Keys: {{checklistID}}

Foreign Keys: {(relatedCard → Card.cardID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CheckList(
    checklistID INTEGER PRIMATY KEY,
    name VARCHAR(50) NOT NULL,
    checkStatus VARCHAR (5) CHECK(checkStatus IN('True', 'False')),
    description VARCHAR(1000) NOT NULL,
    relatedCard INTEGER,
    FOREIGN KEY relatedCard REFERENCES Card(cardID)
);
```

3.12 Item

Relational Model: Item(itemID, relatedChecklist, completedStatus, content, completor)

Functional Dependencies: {(itemID, relatedChecklist → completedStatus, content, completor)}

Primary Key: {itemID, relatedChecklist}

Candidate Keys: {{itemID, relatedChecklist}}

Foreign Keys: {(relatedChecklist → CheckList.checklistID), (completor → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Item(
    itemID INTEGER PRIMARY KEY,
    relatedCheckList INTEGER NOT NULL,
```

```
    completedStatus VARCHAR(5),
    content VARCHAR(1000) NOT NULL,
    completor INTEGER NOT NULL,
    CHECK(completedStatus IN('True', 'False')),
    FOREIGN KEY relatedCheckList REFERENCES CheckList(checklistID)
    FOREIGN KEY completor REFERENCES BasicUser(userID)
);
```

3.13 Comment

Relational Model: Comment(commentID, relatedCard, timestamp, resolvedStatus, commenter, text)

Functional Dependencies: {(commentID, relatedCard → timestamp, resolvedStatus, commenter, text)}

Primary Key: {commentID, relatedCard}

Candidate Keys: {{commentID, relatedCard}}

Foreign Keys: {(relatedCard → Card.cardID), (commenter → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Comment(
    commentID INTEGER,
    relatedCard INTEGER NOT NULL,
    timestamp TIMESTAMP NOT NULL,
    resolvedStatus VARCHAR(5),
    commenter INTEGER NOT NULL,
    text VARCHAR(1000) NOT NULL,
    PRIMARY KEY (commentID, relatedCard),
    FOREIGN KEY relatedCard REFERENCES Card(cardID),
    FOREIGN KEY commenter REFERENCES BasicUser(userID),
```

```
    CHECK(resolvedStatus IN('True', 'False'))
);
```

3.14 Replies

Relational Model: Replies(replyID, relatedCard, commentID)

Functional Dependencies: {(replyID, relatedCard → commentID)}

Primary Key: {replyID, relatedCard}

Candidate Keys: {{replyID, relatedCard}}

Foreign Keys: {(replyID, relatedCard → Comment.(commentID, relatedCard)), (commentID, relatedCard → Comment.(commentID, relatedCard))}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Replies(
    replyID INTEGER NOT NULL,
    commentID INTEGER NOT NULL,
    relatedCard INTEGER NOT NULL,
    PRIMARY KEY (replyID, relatedCard),
    FOREIGN KEY (replyID, relatedCard) REFERENCES
    Comment(commentID, relatedCard),
    FOREIGN KEY (commentID, relatedCard) REFERENCES
    Comment(commentID, relatedCard)
);
```

3.15 Upvotes

Relational Model: Upvotes(userID, commentID, relatedCard)

Functional Dependencies: {}

Primary Key: {userID, commentID, relatedCard}

Candidate Keys: {{userID, commentID, relatedCard}}

Foreign Keys: {(commentID, relatedCard → Comment.(commentID, relatedCard)), (userID → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Upvotes(
    userID INTEGER NOT NULL,
    commentID INTEGER NOT NULL,
    relatedCard INTEGER NOT NULL,
    PRIMARY KEY (userID, commentID relatedCard),
    FOREIGN KEY userID REFERENCES BasicUser(userID),
    FOREIGN KEY (commentID, relatedCard) REFERENCES
    Comment(commentID, relatedCard)
);
```

3.16 Attachment

Relational Model: Attachment(attachmentID, name, size, uploadDate, description, attacher, relatedCard)

Functional Dependencies: {(attachmentID → name, size, uploadDate, description, attacher, relatedCard)}

Primary Key: {attachmentID}

Candidate Keys: {{attachmentID}}

Foreign Keys: {(attacher → BasicUser.userID), (relatedCard → Card.cardID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Attachment(
    attachmentID INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    size INTEGER NOT NULL,
    uploadDate DATETIME NOT NULL,
    description VARCHAR(1000) NOT NULL,
    attacher INTEGER NOT NULL,
    relatedCard INTEGER NOT NULL,
    FOREIGN KEY attacher REFERENCES BasicUser(userID),
    FOREIGN KEY relatedCard REFERENCES Card(cardID)
);
```

3.17 Label

Relational Model: Label(labelID, color, text, importance, adder)

Functional Dependencies: {(labelID → color, text, importance, adder)}

Primary Key: {labelID}

Candidate Keys: {{labelID}}

Foreign Keys: {(adder → BasicUser.userID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Label(
    labelID INTEGER PRIMARY KEY,
    color VARCHAR(50) NOT NULL,
    text VARCHAR(100) NOT NULL,
    importance INTEGER CHECK(0 < importance <= 5),
    adder INTEGER NOT NULL,
    FOREIGN KEY adder REFERENCES BasicUser(userID)
);
```

3.18 Labelling

Relational Model: Label(cardID, labelID)

Functional Dependencies: {}

Primary Key: {cardID, labelID}

Candidate Keys: {{cardID, labelID}}

Foreign Keys: {(cardID → Card.cardID), (labelID → Label.labelID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Labelling(
    cardID INTEGER NOT NULL,
    labelID INTEGER NOT NULL,
    FOREIGN KEY cardID REFERENCES Card(cardID),
    FOREIGN KEY labelID REFERENCES Label(labelID)
);
```

4 Functional Dependencies and Normalization

As already shown in the previous section, all of our relation schemas are in Boyce Codd Normal Form (BCNF). This can be easily checked by the fact that all the functional dependencies contain a candidate key in their left hand side. Therefore, we do not need to perform any decomposition or normalization on our system's tables.

5 Functional Components

In this section we will introduce the Use-Case diagram of Uptown Fank System, followed by detailed scenarios and the usage of chosen data structures.

5.1 Use Case Diagram

Figure 2: Use Case Diagram

5.2 Use Case Scenarios

POTENTIAL USER GROUP: SUPERUSER

5.2.1 CreateTeam

Use case name	CreateTeam
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks the Create Team button. 2. System provides fields for team name, and team description. 3. SuperUser clicks the required fields and clicks create button. 4. System asks to invite members to the team. 5. SuperUser specifies the users to be invited to the team. 6. System creates a new team and user is taken to the menu of that team and a key to access the team is created.
Entry conditions	• The SuperUser is logged into the system.
Exit conditions	• The new team with specified fields is created. • The key to create a team is created.

5.2.2 ManageTeam

Use case name	ManageTeam
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser chooses on the specified Team 2. System opens the Team Information the user can edit 3. SuperUser modifies the team name or description and saves the state. 4. System signals the successful completion of the modifications and saves the last modified state.
Entry conditions	• The SuperUser has to be the one who created the team.
Exit conditions	• New name of the team is saved if modified. • New description of the team is saved if modified.

5.2.2.1 DeleteTeam

Use case name	DeleteTeam
Participating actors	Initiated by SuperUser during ManageTeam use case
Flow of events	1. SuperUser chooses the delete Team option in the Manage Team page. 2. System asks to confirm the delete operation. 3. SuperUser confirms the deletion 4. System acknowledges the successful deletion of the team.
Entry conditions	This use case extends the ManageTeam use case. It is initiated whenever SuperUser decides to delete a team
Exit conditions	• New name of the team is saved if modified. • New description of the team is saved if modified.

5.2.2.2 AddMember

Use case name	AddMember
Participating actors	Initiated by SuperUser during ManageTeam use case
Flow of events	1. SuperUser chooses to add Member. 2. System asks to provide the members to be added. 3. SuperUser provides the names of the required users and confirms the selections. 4. System acknowledges the successful addition of new users.
Entry conditions	This use case extends the ManageTeam use case. It is initiated whenever SuperUser decides to add new members.
Exit conditions	• New member(s) are added to the team

5.2.2.3 RemoveMember

Use case name	RemoveMember
Participating actors	Initiated by SuperUser during ManageTeam use case
Flow of events	1. SuperUser chooses a team members to remove and chooses remove member option 2. System asks to confirm the removal of selected members 3. SuperUser confirms the removal of members from the system 4. System acknowledges the successful removal of specified team members
Entry conditions	This use case extends the ManageTeam use case. It is initiated whenever SuperUser decides to remove team members
Exit conditions	• Selected member(s) are deleted from the team

5.2.3 CreateBoard

Use case name	CreateBoard
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks Create New Board button 2. System asks for name, description, priority, color, requirements, and estimated time 3. SuperUser fills out required fields and clicks create button 4. System creates new board with specified characteristics and user is taken to the created board menu
Entry conditions	• The SuperUser owns team in where board is created
Exit conditions	• The new board is created and belongs to the team in which it was created

5.2.4 AddList

Use case name	AddList
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser chooses to create list in the Card 2. System asks for name, description, color 3. SuperUser provides the name, description, color and confirms 4. System acknowledges the successful removal of team members
Entry conditions	• SuperUser has to be in the board he is a supervisor of
Exit conditions	• The new list is created with provided information

5.2.5 SetUserPrivileges

Use case name	SetUserPrivileges
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks on one of the supervised teams 2. System displays team details: boards and members 3. SuperUser clicks on members and chooses a member he/she wants to grant privileges on 4. System lists available privileges that can be granted for BasicUser 5. SuperUser chooses all the privileges he/she wants to grant 6. System grants chosen privileges on BasicUser
Entry conditions	• SuperUser supervises at least one team and has at least one member on the team
Exit conditions	• The BasicUser is granted with chosen privileges

5.2.6 ManageBoard

Use case name	ManageBoard
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks on the specified Board 2. System opens the Board Information the user can edit and provides option of deleting the board entirely 3. SuperUser modifies the board name or description and saves the state or deletes the board 4. System reports the successful completion of the modifications and save the last modified state or reports of successful deletion of board
Entry conditions	• The SuperUser has to be the one who created the team and board
Exit conditions	• New name of the board is saved if modified • New description of the board is saved if modified • Board is removed from team if deleted

5.2.6.1 ManageCards

Use case name	ManageCards
Participating actors	Initiated by SuperUser during ManageBoard use case
Flow of events	1. SuperUser clicks on the specified Card 2. System opens the Card Information the user can edit and provides option of deleting the Card entirely 3. SuperUser modifies the Card name or description and saves the state or deletes the Card 4. System reports the successful completion of the modifications and save the last modified state or reports of successful deletion of Card
Entry conditions	• The SuperUser has to be the one who created the Team and Board within Card exists
Exit conditions	• New name of the Card is saved if modified • New description of the Card is saved if modified • Card is removed from Board if deleted

5.2.6.2 ManageLists

Use case name	ManageLists
Participating actors	Initiated by SuperUser during ManageBoard use case
Flow of events	1. SuperUser clicks on the specified List 2. System opens the List Information the user can edit and provides option of deleting the List entirely 3. SuperUser modifies the List name or description and saves the state or deletes the List 4. System reports the successful completion of the modifications and save the last modified state or reports of successful deletion of List
Entry conditions	• The SuperUser has to be the one who created the Team and Board and Card within List exists
Exit conditions	• New name of the List is saved if modified • New description of the List is saved if modified • List is removed from Board if deleted

5.2.6.3 Archive

Use case name	Archive
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks on the card he/she wants to archive 2. System archives chosen card and archives it 3. System reports whether operation was successful or not
Entry conditions	• The SuperUser is supervisor of team within card exists
Exit conditions	• The archived card disappears from users view and successfully archived

5.2.6.4 AssignUser

Use case name	AssignUser
Participating actors	Initiated by SuperUser
Flow of events	1. SuperUser clicks on card he/she desires assign to BasicUser 2. System takes SuperUser to card menu 3. SuperUser clicks on "Assign User" button 4. System lists BasicUsers available for assignment 5. SuperUser chooses desired BasicUsers 6. System assigns BasicUsers to card
Entry conditions	• The SuperUser is supervisor of team within card exists
Exit conditions	• The BasicUsers are assigned to card

POTENTIAL USER GROUP: BASICUSER

5.2.7 AddAttachment

Use case name	AddAttachment
Participating actors	Initiated by BasicUser
Flow of events	9. BasicUser chooses to upload attachment in the Card 10. System asks for name, description 11. BasicUser provides the name, description and confirms 12. System records attachmentID, size, uploadDate, attacher and relatedCard and acknowledges the successful upload of attachment
Entry conditions	• BasicUser has to be a member of team that works on chosen Board within Card exists
Exit conditions	• The new attachment is uploaded with specified records

5.2.8 AddLabel

Use case name	AddLabel
Participating actors	Initiated by BasicUser
Flow of events	5. BasicUser chooses to create list in the Card 6. System asks for color, text, importance 7. BasicUser provides the color, text, importance and confirms 8. System records labelID and adder and reports the successful creation of label
Entry conditions	• BasicUser has to be a member of team that works on chosen Board within Card exists
Exit conditions	• The new label is created with provided information

5.2.9 ManageAttachments

Use case name	ManageAttachments
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser clicks on the specified Card 2. System takes BasicUser to Card menu 3. BasicUser modifies Attachment name or description and saves the state or deletes the Attachment 4. System reports the successful completion of the modifications and save the last modified state or reports of successful deletion of Attachment
Entry conditions	• BasicUser has to be a member of team that works on chosen Board and Card within Attachment exist
Exit conditions	• New name of the Attachment is saved if modified • New description of the Atatchment is saved if modified • Attachment is removed from Card if deleted

5.2.10 ManageLabels

Use case name	ManageLabels
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser clicks on the specified Card 2. System takes BasicUser to Card menu with all Labels listed there 3. BasicUser modifies Labels text or importance and saves the state or deletes the Label 4. System reports the successful completion of the modifications and save the last modified state or reports of successful deletion of Label
Entry conditions	• BasicUser has to be a member of team that works on chosen Board and Card within Labels exist
Exit conditions	• New text of the Label is saved if modified • New importance of the Label is saved if modified • Label is removed from Card if deleted

5.2.11 AddChecklist

Use case name	AddCheckList
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser selects Add CheckList from the Card 2. System asks for name and description 3. BasicUser provides name and description 4. System confirms the successful addition of a checklist to the board by displaying the checklist
Entry conditions	• BasicUser is in the required card
Exit conditions	• CheckList is created for the chosen card with provided name and description

5.2.11.1 CompleteItem

Use case name	CompleteItem
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser selects the CheckList Item to complete. 2. System acknowledges the completion of the chosen CheckList Item.
Entry conditions	• This use case extends the AddCheckList use case whenever BasicUser decides to complete a CheckList
Exit conditions	• Chosen CheckList Item is marked completed

5.2.12 Comment

Use case name	Comment
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser clicks on the specified Card 2. System takes BasicUser to Card menu 3. BasicUser leaves comment 4. SuperUser sets status of comment whether to resolved or unresolved 5. System reports the successful addition of comment
Entry conditions	• BasicUser has to be a member of team that works on chosen Board and Card within Comment created
Exit conditions	• New comment for Card is added

5.2.12.1 Reply

Use case name	Reply
Participating actors	Initiated by BasicUser during Comment use case
Flow of events	1. BasicUser clicks on the specified Card 2. System takes BasicUser to Card menu 3. BasicUser clicks on comment and chooses Reply option 4. System reports the successful addition of reply
Entry conditions	• BasicUser has to be a member of team that works on chosen Board and Card within Comment created
Exit conditions	• New Reply for Card is added

5.2.13 Register

Use case name	Register
Participating actors	Initiated by BasicUser
Flow of events	1. BasicUser either chooses to register as a BasicUser or SuperUser 2. System responds with appropriate options for each case
Entry conditions	• BasicUser should have started the system
Exit conditions	• BasicUser is registered in the system as BasicUser or SuperUser if he/she chose so.

5.2.13.1 RegisterAsSuperUser

Use case name	Register as SuperUser
Participating actors	**Inherited** from Register
Flow of events	1. BasicUser chooses to register as a SuperUser 2. System asks for name, email, password, phone numbers, and additional organization name 3. BasicUser provides name, email, password, phone numbers, and organization name and confirms 4. System notifies of successful registration of with SuperUser privileges if everything is entered correctly
Entry conditions	**Inherited** from Register
Exit conditions	**Inherited** from Register

5.2.13.2 RegisterAsBasicUser

Use case name	Register as BasicUser
Participating actors	**Inherited** from Register
Flow of events	1. BasicUser chooses to register as a BasicUser 2. System asks for name, email, password, and phone numbers 3. BasicUser provides name, email, password, and phone numbers 4. System notifies of successful registration if everything is entered correctly
Entry conditions	**Inherited** from Register
Exit conditions	**Inherited** from Register

5.3 Data Structures

In our relation schemas we used the following data types: Integer, Varchar, Date and Timestamp.

- Integer: This is used for all possible IDs belonging to different entities in our system. Integers are simple and appropriate for this purpose since we only need an identifying data type and it won't have complexities (such as decimal points etc).
- Varchar: This datatype is used for almost all possible "content" filling in the different tables that we have created. The size of the Varchar ranges based on the attribute that is going to be represented with this data type, as mainly explained in the following:
  - For names, we have used a varchar of size 50 since we account for First, Middle and Last name. We have also used the same size for emails, passwords and addresses.
  - For blocks of description (these belong to various main parts of our system, including boards, lists and cards), we have used a varchar of size 1000. The system considers the average size of a word as 5 characters. Since the length or duration of words is clearly variable, for the purpose of measurement of text entry, the definition of each "word" is often standardized to be five characters or keystrokes long in English [2]. In this way, our system accounts for a paragraph of around 200 words.
  - For attributes which will give completes status information and the like, we want the value of that column to be either "true" or "false" since these two words are universal. For this reason, size of the varchar is at most 5.
  - For affiliations and organizations, the system uses a varchar of size 200 at most.
- Date and Timestamp are used for representing specific deadlines, as well as showing the exact time when comments were posted.

In the front-end, we will be using various HTML rendered lists and tables.

6 User Interface Design and Corresponding SQL Statements

In this section we provide sketches of GUI and corresponding SQL queries for the system's major tasks for achieving the functionality requirements.

NOTE: *We have divided "view" option for a particular user into homescreen and specific components. For this reason, the functionalities are divided into 6 main groups.*

6.1 Register and Login Functionalities

6.1.1 Register Screen

Figure 3: Register Screen

6.1.1.1 BasicUser

—Registering a user

INSERT INTO BasicUser(name, email, password, address) VALUES('Naisila', 'naisila.puka@ug.bilkent.edu.tr', @password, 'Bilkent Main Campus');

—totalPhone will hold the total number of phone numbers the user enters

for(i = 1 up to totalPhone){

INSERT INTO Phone(userID, phoneNo) VALUES (SELECT userID FROM BasicUser WHERE email = 'naisila.puka@ug.bilkent.edu.tr') , @phoneNo[i]);

}

6.1.1.2 SuperUser

—Additional to the operations performed in BasicUser, we also insert into SuperUser table

INSERT INTO SuperUser(userID, organization) VALUES (SELECT userID FROM BasicUser WHERE email EQUAL 'naisila.puka@ug.bilkent.edu.tr'), 'Bilkent University');

6.1.2 Login Screen

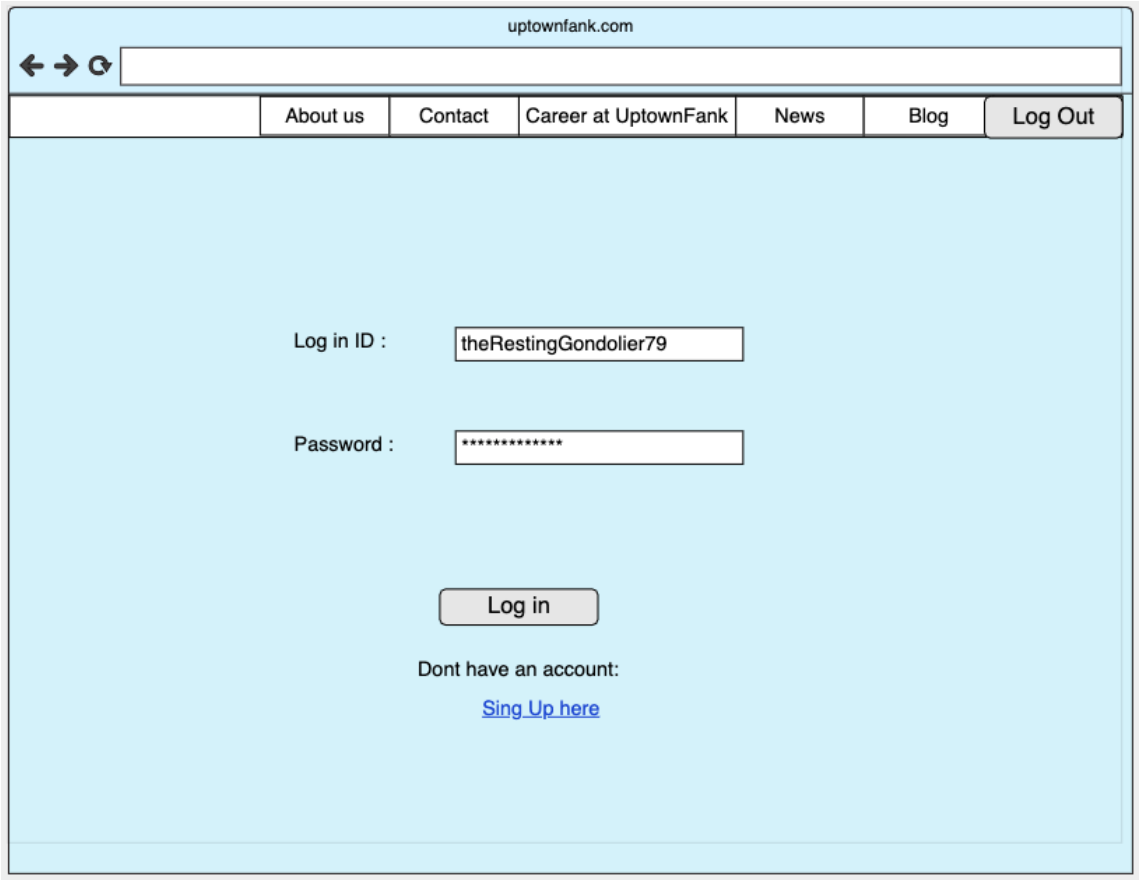

Figure 4: Login Screen

—To check whether the information entered is valid, we will use the following query

SELECT email, password FROM BasicUser WHERE email = 'theRestingGondolier' AND password = @password;

6.2 View Main Personal Info

6.2.1 Personal Homescreen

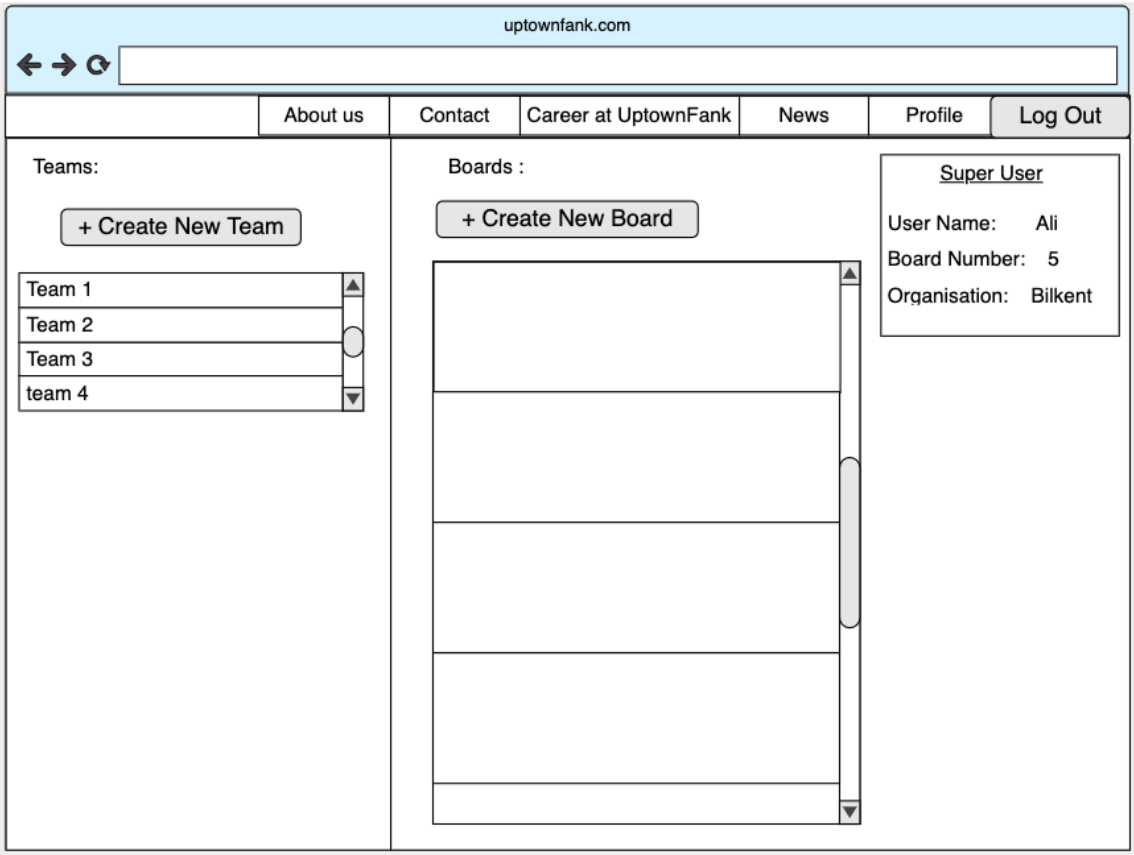

Figure 5: Homepage

—Show all teams that the Superuser Manages:

SELECT T.name FROM Team T WHERE T.supervisor = 5321;

—Show all teams that the user is part of:

SELECT T.name FROM Team T NATURAL JOIN Member M WHERE T.userID = 5321;

—Show all the boards that the user owns:

SELECT B.name FROM Board B WHERE B.ownerId = 5321;

—Show personal info of the user:

SELECT U.name FROM BasicUser U WHERE U.userID = 5321;

—Extra info for SuperUser's:

SELECT COUNT(boardID) FROM Board B JOIN SuperUser U ON U.userID = B.ownerID WHERE u.userID = 5321;

SELECT organisation FROM SuperUser WHERE userID = 5321;

6.3 View Boards/Lists/Cards

6.3.1 View Board Screen

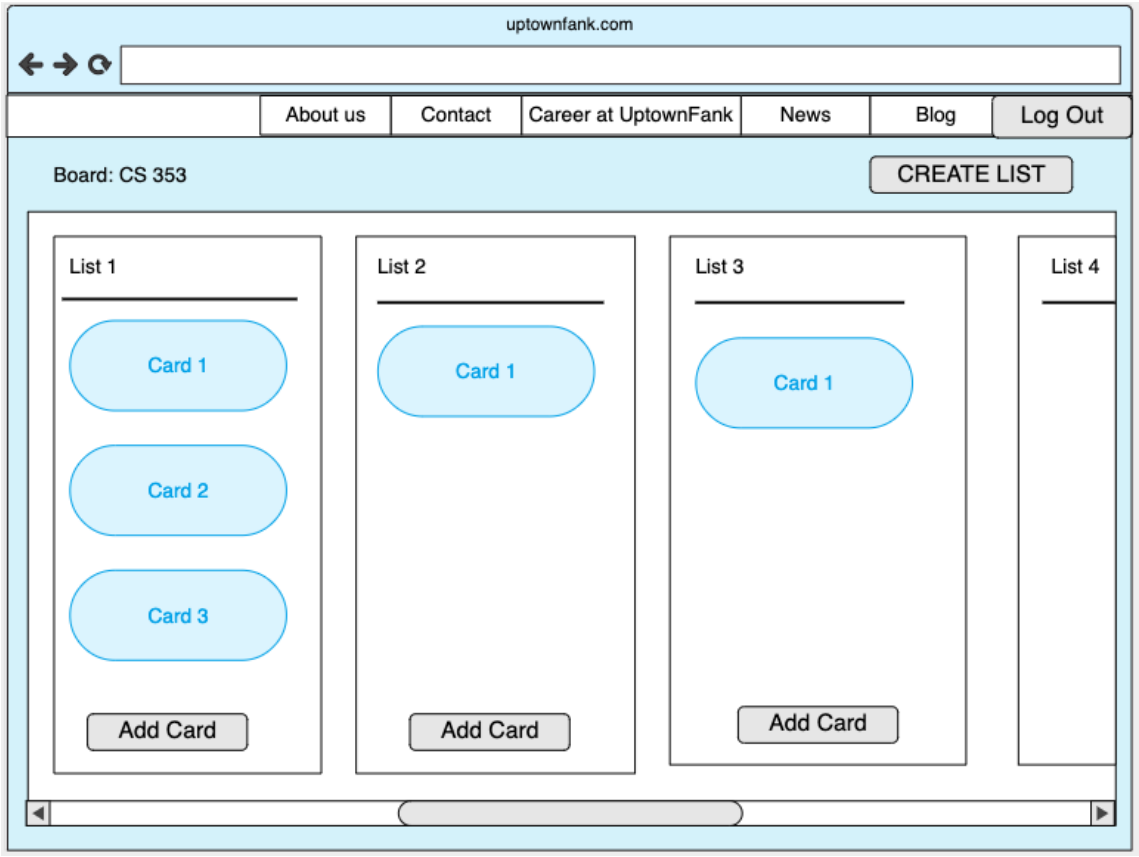

Figure 6: View Board Screen

—Create new List

INSERT INTO List(listID, name, finishedStatus, color, description, activity, boardID) VALUES (1000, 'sec2', 'False', 'Blue', 'Here we manage all reports' ,'First report finished', 1010);

—Create New Card

INSERT INTO Card(cardID, name, priority, description, dueDate, listID,archived, finished) VALUES (458, 'Design Report', 3, 'write introduction', '10/5/2019-10:00', 5610, 'False', 'False');

6.3.2 View Card Screen

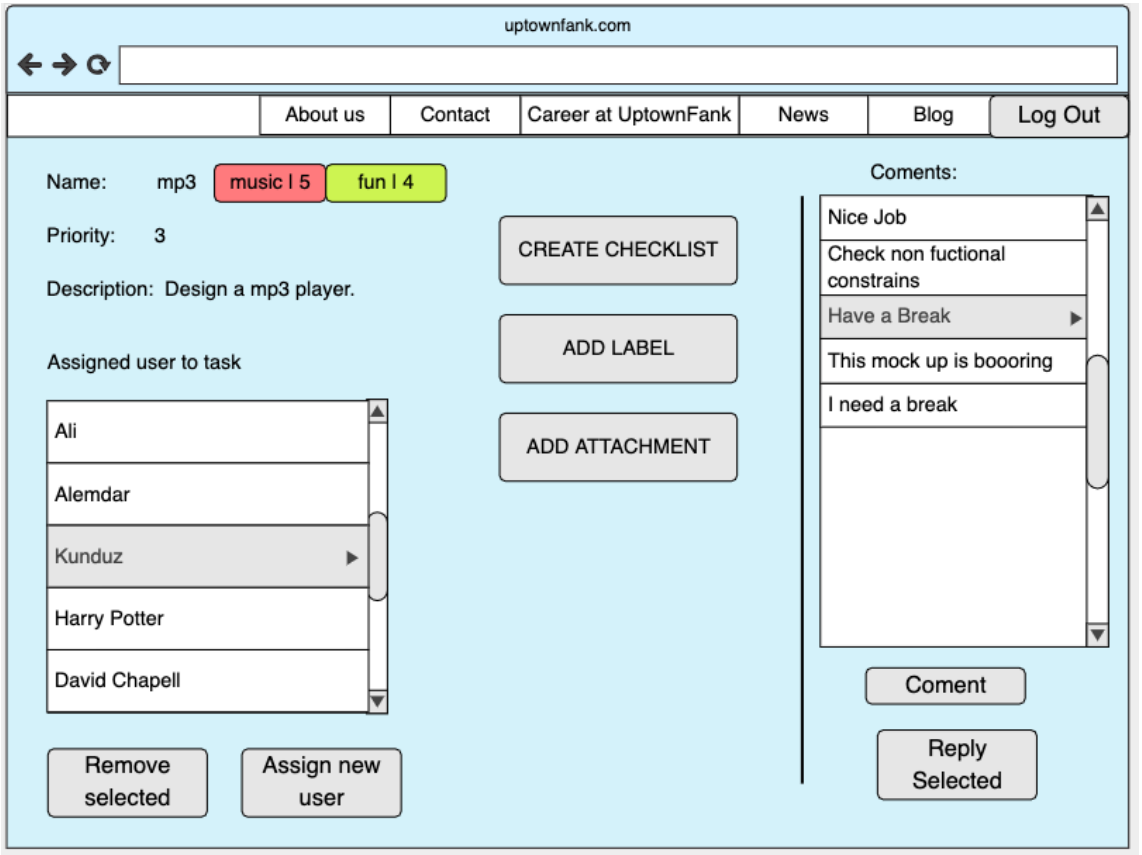

Figure 7: View Card Screen

—Show card's general info

SELECT C.name, C.priority, C.description FROM Card C WHERE C.listID = 10010;

—Show the names of the users that are assigned this card, assign a new user to this card or remove a user from the card

SELECT name FROM (SELECT userID FROM Card C NATURAL JOIN Performs P WHERE C.cardID = 7810) NATURAL JOIN User U;

INSERT INTO PerformsTask(cardID, userID) VALUES (3333, 1000);

DELETE FROM PerformsTask(cardID, userID) VALUES (3333, 1000);

—Shows all labels associated with the Card

SELECT L.text, L.color , L.priority FROM LABELLING LBS NATURAL JOIN LABEL L WHERE LBS.cardID = 10;

— Comment on a card and reply to a comment

INSERT INTO Comment(commentID, relatedCard, timestamp, resolvedStatus, commenter, text) VALUES(5450, 3333, '10/10/2019-15:03', 'False', 5555, 'How do you write a SQL query?');

INSERT INTO Comment(commentID, relatedCard, timestamp, resolvedStatus, commenter, text) VALUES(1234, 3333, '10/10/2019-15:06', 'False', 6789, 'Google it!');

INSERT INTO Replies(replyID, relatedCard, commentID) VALUES (1234, 3333, 5450);

6.4 Create Team and Board

6.4.1 Create Team Screen

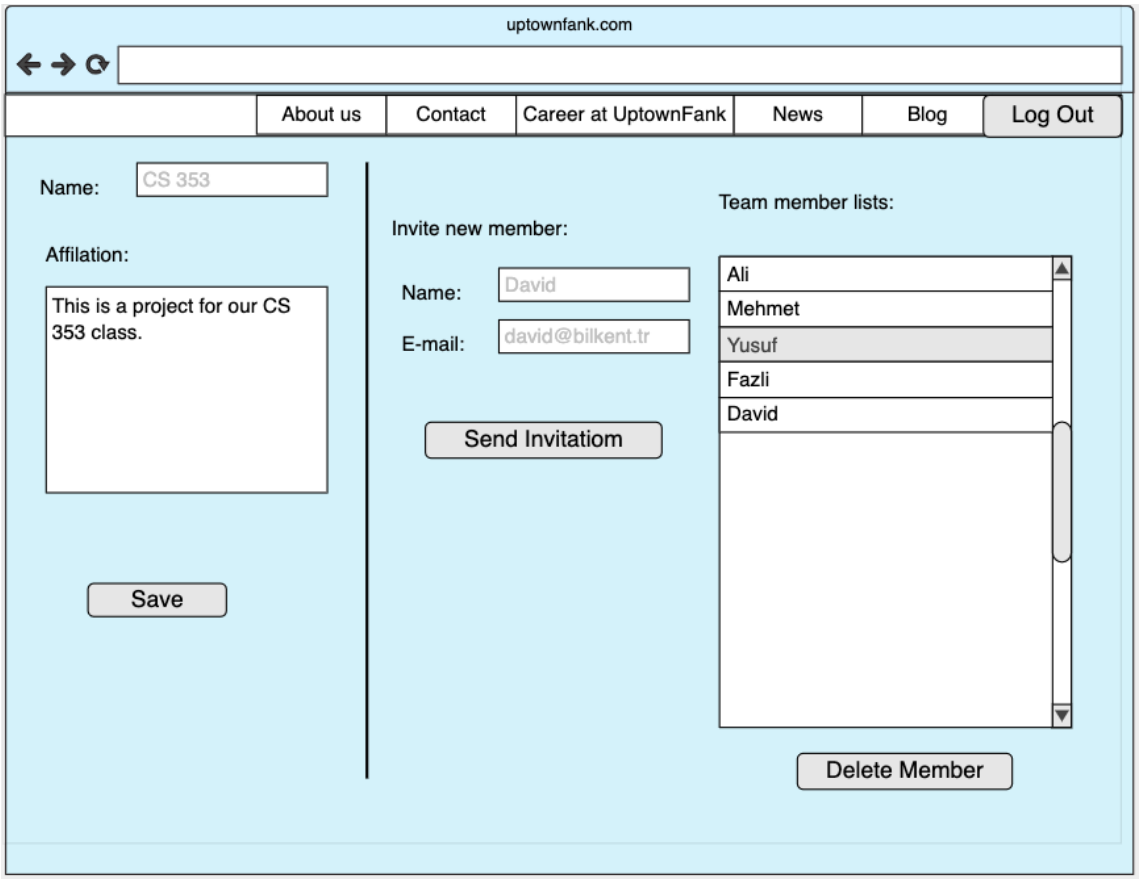

Figure 8: Create Team Screen

—*Create Team*

INSERT INTO Team (teamID, name, affiliation, supervisor, key) VALUES (3333, 'CS353', ' Hello students!', @userID, 112233);

—*Insert member into team*

INSERT INTO Member(userID, teamID) VALUES (55687, 5050);

INSERT INTO Member(userID, teamID) VALUES (33325, 500);

—Delete member from team

DELETE FROM Member WHERE teamID = 500 AND userID = 55687 ;

6.4.2 Create Board Screen

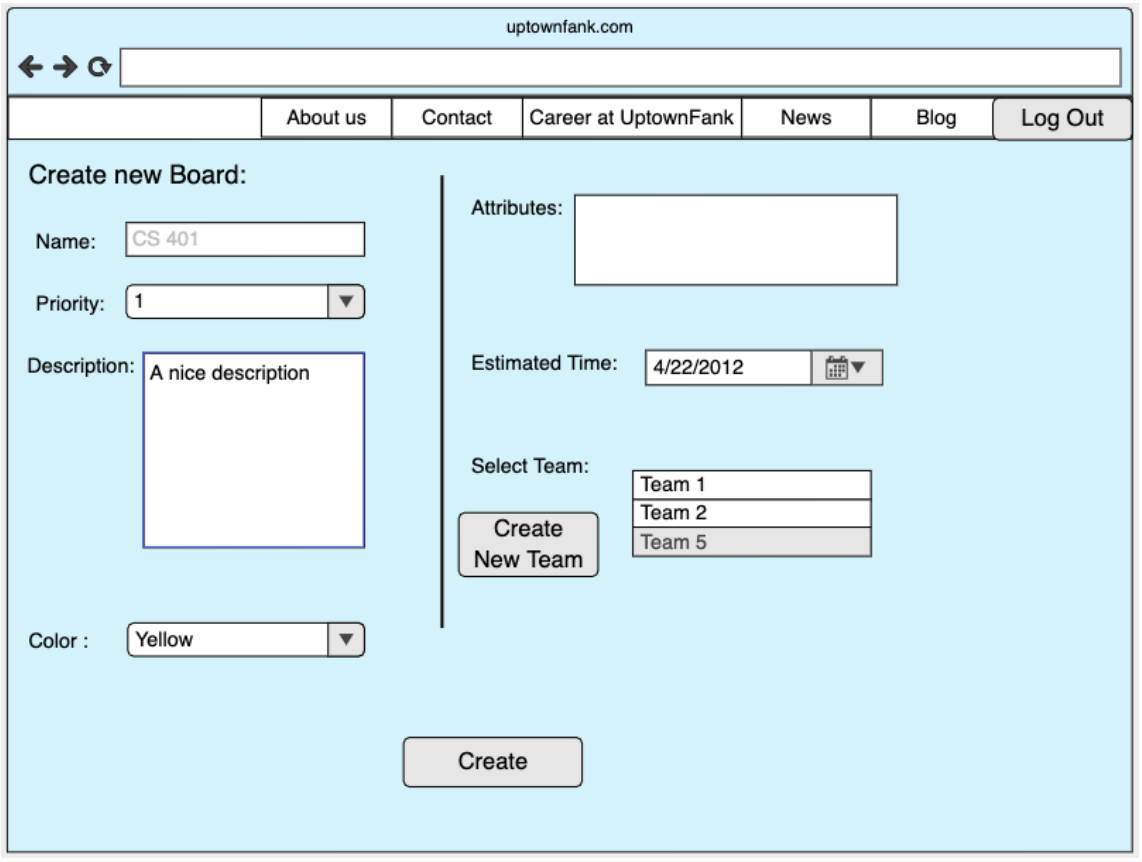

Figure 9: Create Board Screen

—Create new Board

INSERT INTO Board (boardID, name, description, priority, color , requirements, expectedTime) VALUES(0654, 'CS353Board', 'This will be used for CS353', 5, 'Yellow', 'Database project', '10/5/2019- 10:30');

INSERT INTO WorksOn(boardId, teamID) VALUES (0654, 500);

6.5 Create List and Card

6.5.1 Create List Screen

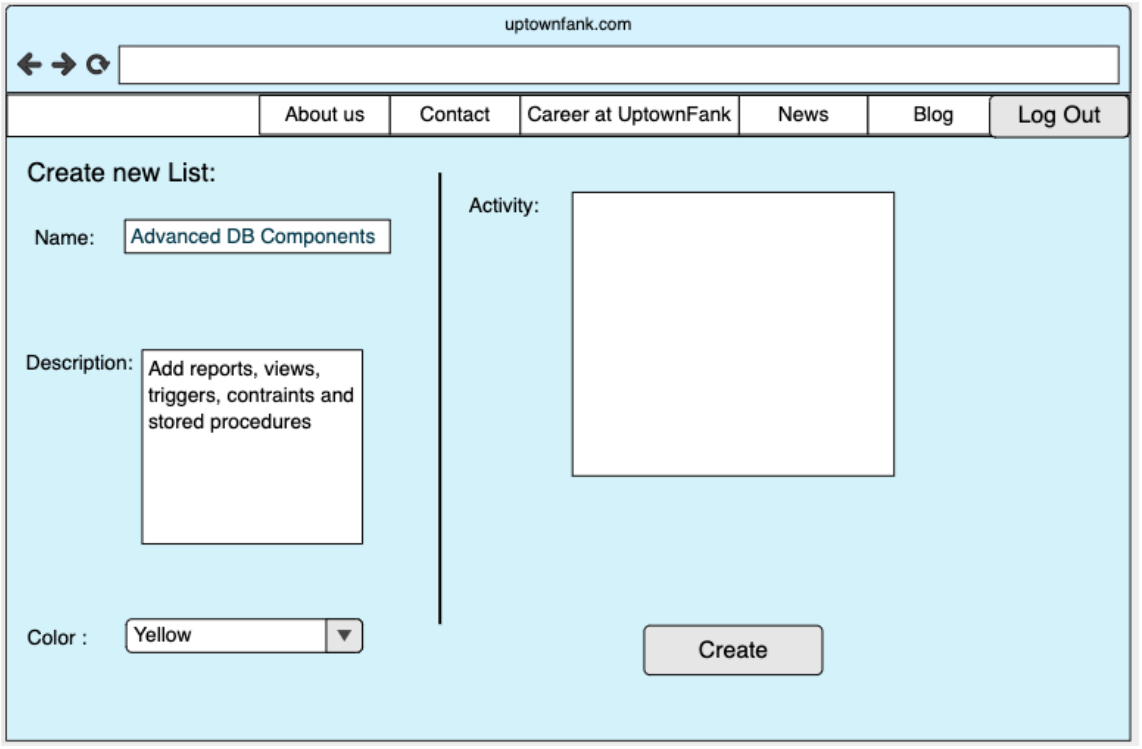

Figure 10: Create List Screen

—*Create a list:*

INSERT INTO List(listID, name, finishedStatus, color, description, activity,boardID) VALUES (1239, 'list3', 'False', 'Yellow', 'List of sec2', 'added class members to list', 445566);

6.5.2 Create Card Screen

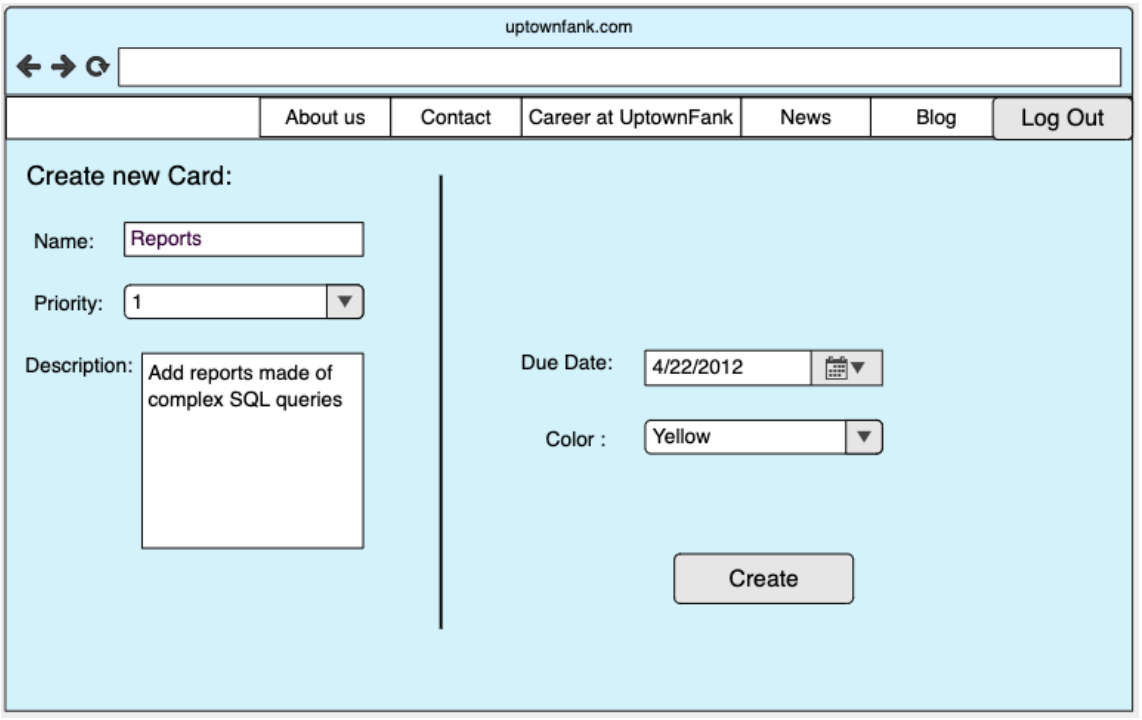

Figure 11: Create Card Screen

—*Creates a new card*

INSERT INTO Card(cardID, name, priority, description, dueDate, listID,archived, finished) VALUES(998877, 'card2', 5, 'Tasks to perform by TA', '15/09/2019-10:45', 5464, 'False', 'False');

6.6 Add CheckLists, Attachments, Labels

6.6.1 Create CheckList Screen

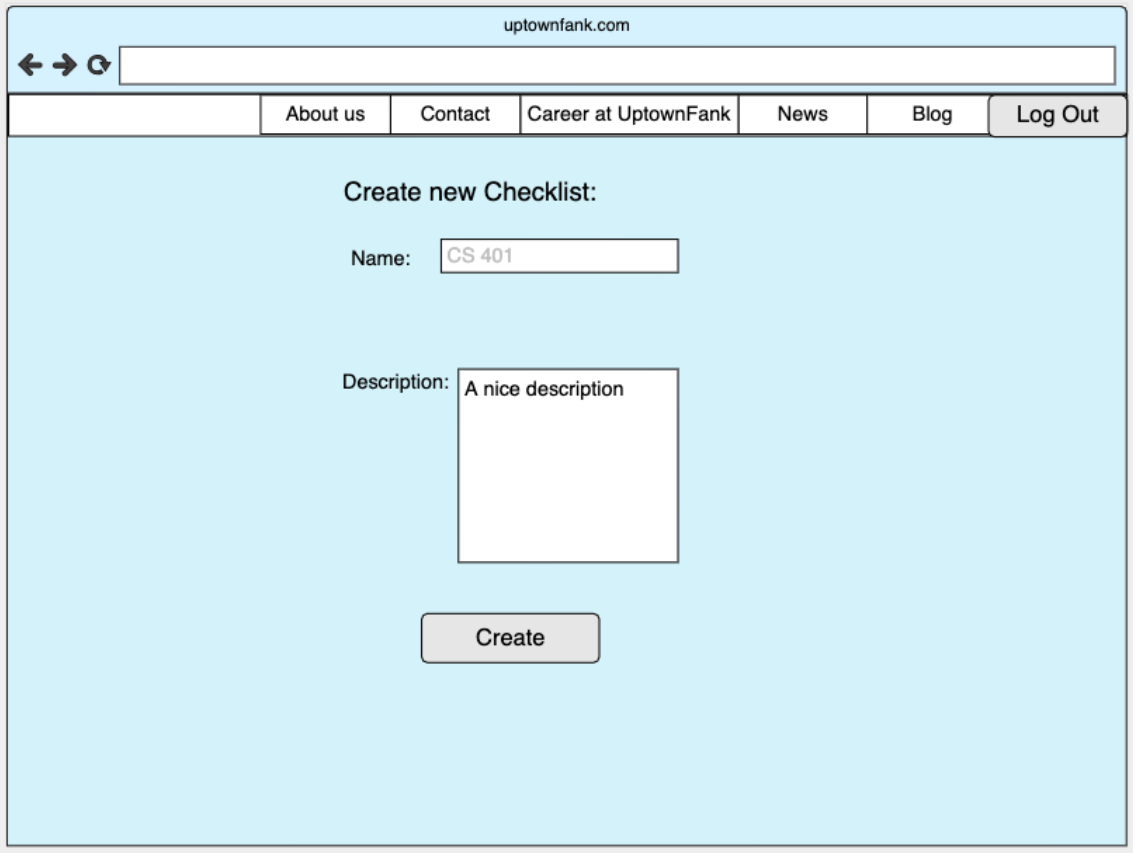

Figure 12: Create CheckList Screen

—*Creates a CheckList*

INSERT INTO CheckList(checklistID, name, checkStatus, description, relatedCard) VALUES (7897, 'project to do list:', 'False', 'A list of tasks', 456875);

6.6.2 Create Label Screen

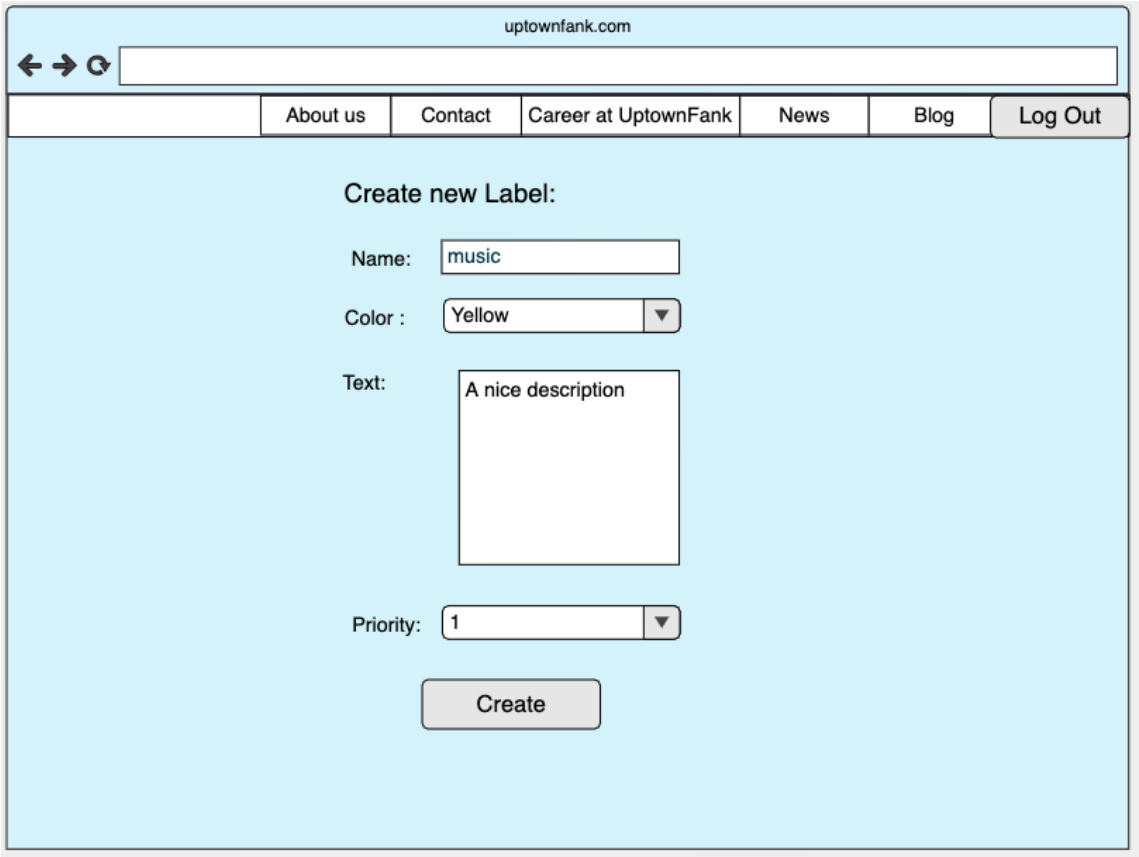

Figure 13: Create Label Screen

—Creates a Label

INSERT INTO Label(labelID, color, text, importance, adder) VALUES (455487, 'Black', 'Databaze', 6, 48795);

7 Advanced Database Components

We will consider various advanced database components for UptownFANK, as are explained in the following subsections.

7.1 Reports

In order to provide insightful analysis about our software, we have prepared the following reports:

- The team with the largest number of members, along with its supervisor and the number of members:

  WITH membersNo AS(SELECT teamID, COUNT(userID) as memberCount FROM Member GROUP BY teamID)

  SELECT t.name, t.supervisor, m.memberCount FROM Teams t NATURAL JOIN membersNo m

  WHERE m.memberCount >= ALL (SELECT memberCount FROM membersNo);

- Users who are part of all projects supervised by a particular user:

  SELECT u.name, u.email FROM BasicUser u WHERE NOT EXISTS(

  (SELECT t.teamID FROM Teams t WHERE t.supervisor = @particularID)

  EXCEPT

  (SELECT m.teamID FROM Member m WHERE m.userID = u.userID));

- 10 Most popular cards in the whole database based on their comments, along with the list and board they belong to:

  WITH commentsNo AS(SELECT relatedCard AS cardID, COUNT(commentID) AS commentsNo FROM Comment GROUP BY relatedCard)

  SELECT c.name AS CardName, c.commentsNo AS NumberOfComments, l.name AS ListName, b.name AS BoardName

  FROM commentsNo c JOIN Card cd JOIN List ls JOIN Board b

  ON(c.cardID = cd.cardID AND cd.listID = ls.listID AND ls.boardID = b.boardID)

  ORDER BY c.commentsNo DESC LIMIT 10;

7.2 Views

Our system's main views will be explained in the following sub sections.

7.2.1 Owned Boards

This view will provide the system with all the owned board names of a particular *superuser* of our software. It will be used in the personal homepage of a superuser, so that he/she can directly see what he/she is in charge of, ordered by their priorities.

CREATE VIEW OwnedBoards AS (SELECT B.name, B.estimatedTime, B.priority FROM Board B JOIN SuperUser U ON U.userID = B.ownerID WHERE U.userID = @userID ORDER BY B.priority DESC);

7.2.2 Users Assigned to a Card

This view will provide the system with all the names and emails of the assigned users to a particular card. It will be used in View Card Screen, where every person can see all the assigned users at a particular card. This could be useful in order to provide communication between users who have been assigned the same job.

CREATE VIEW AssignedUsers AS (SELECT name, email FROM (SELECT userID FROM Card C NATURAL JOIN Performs P WHERE C.cardID = @cardID) NATURAL JOIN User U);

7.2.3 Cards a User Is Assigned to

This view will provide the system with all the names, priorities, descriptions and due dates of all the cards a particular user was assigned to. The view will be used in the personal homepage of each user, so that he can easily see in which cards his work is located.

CREATE VIEW AssignedCards AS (SELECT C.name, C.priority, C.description, C.duedate FROM (SELECT cardID FROM Performs P WHERE P.userID = @userID) NATURAL JOIN Card C);

7.3 Triggers

We will implement the following triggers, which go back to back with the contraints in the other section as well:

- When a SuperUser inserts a new member to a team the system firstly checks if there are less than 30 members in the team and then allows the addition of the new member. If insertion is valid, all views and reports will be accordingly updated.
- On the other hand, if a SuperUser is trying to delete a member from a team, the system will enforce the trigger of checking whether that team has more than 1 member left after deletion. If deletion is valid, all views and reports will be accordingly updated.
- When the owner of a board tries to assign a specific card to a user, the system will check if the number of assigned users to that particular card has exceeded the maximum of 10 after assignment. If assignment is valid, all views and reports will be accordingly updated.

7.4 Constraints

Our system will impose the following constraints:

- A team can have at most 30 members and at least 2 members, that is, the supervisor and another person.
- A checklist can have at most 20 items. Note here, that in case the user is not satisfied with 20 items, he/she could rather create more than one checklist. The aim of our system is to make the project as easy to track as possible. We want to enforce Divide and Conquer approach to project management.

- A card can be assigned to at most 10 people. If more than 10 people are needed to finish the job of a card, then a list should be considered instead of a card.
- The system does not support offline access, and a user has to be logged in his/her account in order to navigate through the software.
- A user can only edit a card that the user has been assigned, not the other cards. The user can only view the other cards.
- A comment can be deleted only by the commenter or the supervisor of the board that the card is part of.

7.5 Stored Procedures

We plan to use the following stored procedures:

- **Check an Item in a Checklist**: Every time a user checks an item in the checklist, check status changes from false to true, and then further conditions are checked. First, if the checklist has no more unchecked items, we change the status of the checklist to finished. After that, checks on the whole card are made as well, because in case there is no unfinished checklist in the card, we change the status of the card to finished as well. This procedure does not vary between different item checks, therefore we chose to use a stored procedure.
- **Delete a Card**: Deletion of a card will not be straightforward. Rather, the system will go through safety check procedure. For every checklist in the card, the system will find all items that have not been checked. For every attachment in the card, the system will find them and then will present them to the user. A notification will arise indicating each unchecked item in the checklists within that card and each attachment pertaining to that card. The user will be asked whether he/she is sure for the deletion of that card. Only after indicating a positive answer, the corresponding SQL query will be executed.

8 Implementation Plan

In this project we plan to use PHP for the system's back-end web service, along with the MySQL database. In front-end, we will create a website using HTML/CSS.

9 Online Access

Our project can be accessed online through our:

Webpage: `https://uptownfank.github.io`

GitHub: `https://github.com/NaisilaPuka/uptownfank` (currently private)

10 Conclusion

With this project, we aim to get the best out a database management system in order to create a useful tool for tracking various projects divided to different teams. Having put a lot of thought in proper relation schemas and in normalizing them, we try to make our system as efficient as possible. Making use of different entities and relationships, the database system will provide full referential integrity for us, which will make UptownFANK a reliable and safe system to use for maintaining project information.

References

[1] "Draw.io: free online diagram software for making flowcharts, process diagrams, org charts, UML, ER and network diagrams," *draw.io*. [ONLINE]. Available at: `https://www.draw.io/`. [Accessed: March 4, 2019]

[2] "Words per minute," *wikipedia.org*. [ONLINE]. Available at: `https://en.wikipedia.org/wiki/Words_per_minute`. [Accessed: April 4, 2019]