Şur-Up

Upload Your AtaBase Projects

Project Tracking Software

By Grup Şurup, Group No: 27

Project Design Report

Muhammet Said Demir 21602021 sec-3

Zeynep Nur Öztürk 21501472 sec-3

TABLE OF CONTENTS

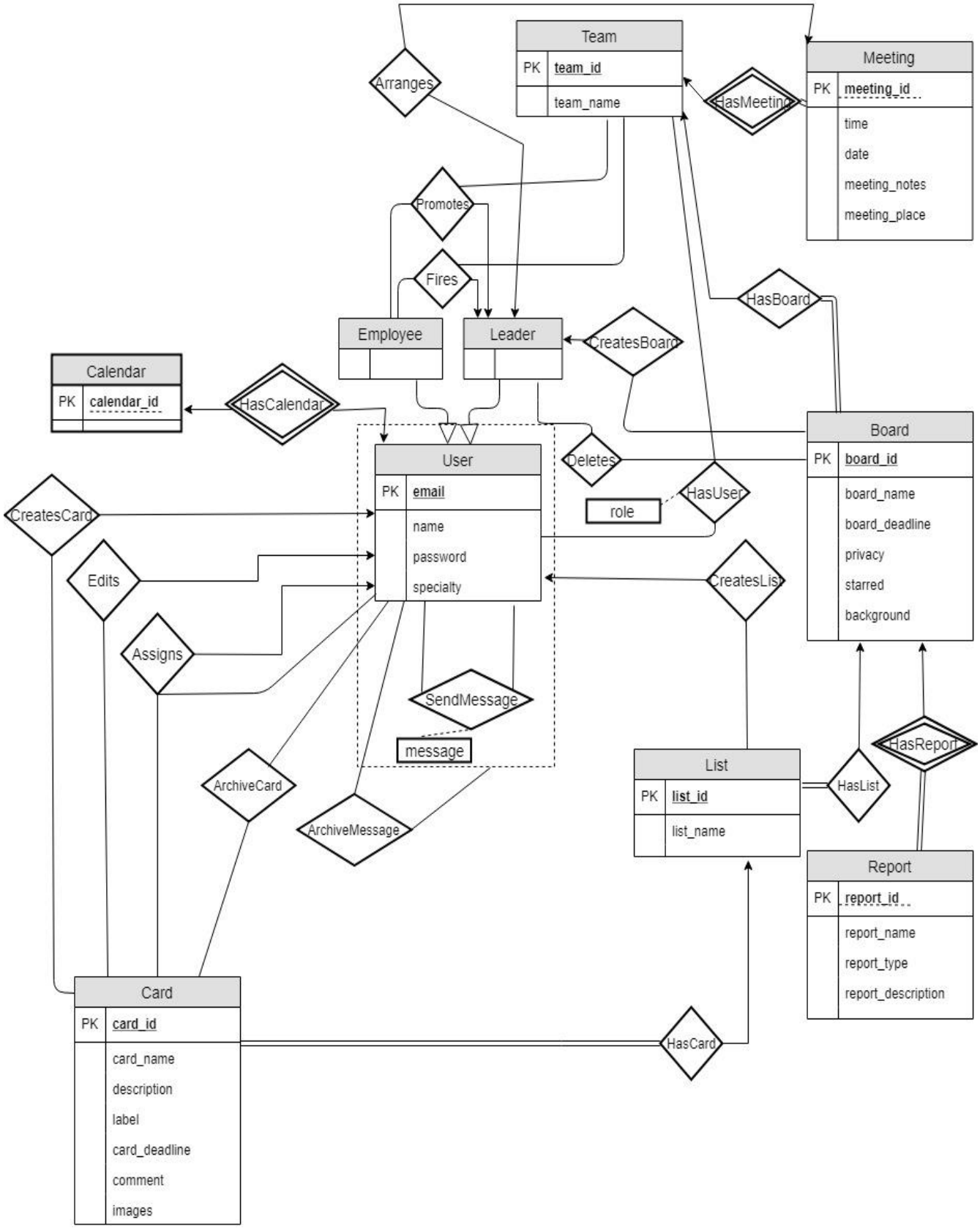
1. Revised E/R Model

According to assistant's review, we change the proposal report and revised our E/R model considering the feedback as follows:

1. We changed the cover of the project
2. We detailed the introduction part of the proposal.
3. We change the functional requirements part in the proposal as wanted from us.
4. We correct the limitation part in proposal as wanted.

We also made the following changes in our E/R model:

1. We corrected our arrow problems.
2. We keep calendar as week entity but we erase the connection between calendar and other entities.
3. We delete the archive entity. We have archive option between user and card, also between user and message.
4. We created an aggregation between user and SendMessage based on our feedback.
5. We give authority to the leader to create and delete the board.


Team
PK team_id
team_name
Meeting
PK meeting_id
time
date
meeting_notes
meeting_place
Arranges
HasMeeting
Promotes
Fires
HasBoard
Employee
Leader
CreatesBoard
Calendar
PK calendar_id
HasCalendar
User
PK email
name
password
specialty
Deletes
role
HasUser
Board
PK board_id
board_name
board_deadline
privacy
starred
background
CreatesCard
Edits
Assigns
CreatesList
SendMessage
message
ArchiveCard
ArchiveMessage
List
PK list_id
list_name
HasList
HasReport
Report
PK report_id
report_name
report_type
report_description
Card
PK card_id
card_name
description
label
card_deadline
comment
images
HasCard

2. Relation Schema

2.1 User

Relational Model

User(email, name, password, specialty)

Functional Dependencies

email -> name, password, specialty

Candidate Keys

{(email)}

Normal Form

BCNF

Table Definition

CREATE TABLE User (

email VARCHAR(32) NOT NULL PRIMARY KEY,

name VARCHAR(64) NOT NULL,

password VARCHAR(32) NOT NULL,

specialty VARCHAR(128) NOT NULL) ENGINE=INNODB;

2.2 Leader

Relational Model

Leader(email, name, password, specialty)

Functional Dependencies

email -> name, password, specialty

Candidate Keys

{(email)}

Normal Form

BCNF

Table Definition

CREATE TABLE Leader (

email VARCHAR(32) NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES User) ENGINE=INNODB;

2.3 Employee

Relational Model

Employee(email, name, password, specialty)

Functional Dependencies

email -> name, password, specialty

Candidate Keys

{(email)}

Normal Form

BCNF

Table Definition

CREATE TABLE Employee (

email VARCHAR(32) NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES User) ENGINE=INNODB;

2.4 Team

Relational Model

Team(team_id, team_name)

Functional Dependencies

team_id -> team_name, email

Candidate Keys

{(team_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE User (

team_id INT NOT NULL PRIMARY KEY,

team_name VARCHAR(128) NOT NULL) ENGINE=INNODB;

2.5 Board

Relational Model

Board(board_id, board_name, board_deadline, privacy, starred, background)

Functional Dependencies

board_id -> board_name, board_deadline, privacy, starred, background

Candidate Keys

{(board_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Board (
board_id INT NOT NULL PRIMARY KEY,
board_name VARCHAR(64) NOT NULL,
board_deadline DATE NOT NULL,
privacy BOOLEAN,
starred BOOLEAN,
background VARCHAR(128)) ENGINE=INNODB;
```

2.6 List

Relational Model

List(list_id, list_name)

Functional Dependencies

list_id -> list_name

Candidate Keys

{(list_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE List (

list_id INT NOT NULL PRIMARY KEY,

list_name VARCHAR(64) NOT NULL) ENGINE=INNODB;

2.7 Card

Relational Model

Card(card_id, card_name, description, label, card_deadline, comment, images)

Functional Dependencies

card_id -> card_name, description, label, card_deadline, comment, images

Candidate Keys

{(card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Card (

card_id INT NOT NULL PRIMARY KEY,

card_name VARCHAR(64) NOT NULL,

description VARCHAR(512),

label NUMERIC(1,0),

card_deadline DATE NOT NULL,

comment VARCHAR(512),

images VARCHAR(512)) ENGINE=INNODB;

2.8 Meeting

Relational Model

Meeting(team_id, meeting_id, time, date, meeting_notes, meeting_place)

Foreign Key team_id References Team

Functional Dependencies

team_id, meeting_id -> time, date, meeting_notes, meeting_place

Candidate Keys

{(team_id, card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Meeting (

team_id INT NOT NULL,

meeting_id INT NOT NULL,

time TIME NOT NULL,

date DATE NOT NULL,

meeting_notes VARCHAR(512),

meeting_place VARCHAR(64) NOT NULL,

PRIMARY KEY (team_id, comment_id),

FOREIGN KEY (team_id) REFERENCES Team) ENGINE=INNODB;

2.9 Calendar

Relational Model

Calendar(email, calendar_id)

Foreign Key email References User

Candidate Keys

{(email, calendar_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Calendar (

email VARCHAR(32) NOT NULL,

calendar_id INT NOT NULL,

PRIMARY KEY (email, calendar_id),

FOREIGN KEY (email) REFERENCES User) ENGINE=INNODB;

2.10 Report

Relational Model

Report(board_id, report_id, report_name, report_type, report_description)

Foreign Key board_id References Board

Functional Dependencies

board_id, report_id -> report_name, report_type, report_description

Candidate Keys

{(board_id, report_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Report (

board_id INT NOT NULL,

report_id INT NOT NULL,

report_name VARCHAR(32) NOT NULL,

report_type VARCHAR(16),

report_description VARCHAR(512),

PRIMARY KEY (board_id, report_id),

FOREIGN KEY (board_id) REFERENCES Board) ENGINE=INNODB;

2.11 HasBoard

Relational Model

HasBoard(board_id, team_id)

Foreign Key board_id References Board

Foreign Key team_id References Team

Functional Dependencies

board_id -> team_id

Candidate Keys

{(board_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasBoard (

board_id INT NOT NULL PRIMARY KEY,

team_id INT NOT NULL,

FOREIGN KEY (board_id) REFERENCES Board,

FOREIGN KEY (team_id) REFERENCES Team) ENGINE=INNODB;

2.12 HasUser

Relational Model

HasUser(team_id, email, role)

Foreign Key team_id References Team

Foreign Key email References User

Functional Dependencies

team_id, email -> role

Candidate Keys

{(team_id, email)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasUser (

team_id INT NOT NULL,

email VARCHAR(32) NOT NULL,

role VARCHAR(32) NOT NULL,

PRIMARY KEY (team_id, email),

FOREIGN KEY (team_id) REFERENCES Team,

FOREIGN KEY (email) REFERENCES User) ENGINE=INNODB;

2.13 CreatesBoard

Relational Model

CreatesBoard(email, board_id)

Foreign Key email References Leader(email)

Foreign Key board_id References Board

Functional Dependencies

board_id -> email

Candidate Keys

{(board_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE CretesBoard (

email VARCHAR(32) NOT NULL,

board_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES Leader(email),

FOREIGN KEY (board_id) REFERENCES Board) ENGINE=INNODB;

2.14 Delete

Relational Model

Delete(email, board_id)

Foreign Key email References Leader(email)

Foreign Key board_id References Board

Candidate Keys

{(email, board_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Delete (

email VARCHAR(32) NOT NULL,

board_id INT NOT NULL,

PRIMARY KEY (email, board_id),

FOREIGN KEY (email) REFERENCES Leader(email),

FOREIGN KEY (board_id) REFERENCES Board) ENGINE=INNODB;

2.15 HasList

Relational Model

HasList(board_id, list_id)

Foreign Key board_id References Board

Foreign Key list_id References List

Functional Dependencies

list_id -> board_id

Candidate Keys

{(list_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasList (

board_id INT NOT NULL,

list_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (board_id) REFERENCES Board,

FOREIGN KEY (list_id) REFERENCES List) ENGINE=INNODB;

2.16 CreatesList

Relational Model

CreatesList(email, list_id)

Foreign Key email References User

Foreign Key list_id References List

Functional Dependencies

list_id -> email

Candidate Keys

{(list_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE CreatesList (

email VARCHAR(32) NOT NULL,

list_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES User,

FOREIGN KEY (list_id) REFERENCES List) ENGINE=INNODB;

2.17 HasCard

Relational Model

HasCard(list_id, card_id)

Foreign Key list_id References List

Foreign Key card_id References Card

Functional Dependencies

card_id -> list_id

Candidate Keys

{(card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasCard (

list_id INT NOT NULL,

card_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (list_id) REFERENCES List,

FOREIGN KEY (card_id) REFERENCES Card) ENGINE=INNODB;

2.18 HasCalendar

Relational Model

HasCalendar(email, calendar_id)

Foreign Key email References User

Foreign Key calendar_id References Calendar

Candidate Keys

{(email, calendar_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasCalendar (

email VARCHAR(32) NOT NULL,

calendar_id INT NOT NULL,

PRIMARY KEY (email, calendar_id),

FOREIGN KEY (email) REFERENCES User,

FOREIGN KEY (calendar_id) REFERENCES Calendar) ENGINE=INNODB;

2.19 CreatesCard

Relational Model

CreatesCard(email, card_id)

Foreign Key email References User

Foreign Key card_id References Card

Functional Dependencies

card_id -> email

Candidate Keys

{(card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE CreatesCard (

email VARCHAR(32) NOT NULL,

card_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES User,

FOREIGN KEY (card_id) REFERENCES Card) ENGINE=INNODB;

2.20 Edits

Relational Model

Edits(email, card_id)

Foreign Key email References User

Foreign Key card_id References Card

Functional Dependencies

card_id -> email

Candidate Keys

{(card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Edits (

email VARCHAR(32) NOT NULL,

card_id INT NOT NULL PRIMARY KEY,

FOREIGN KEY (email) REFERENCES User,

FOREIGN KEY (card_id) REFERENCES Card) ENGINE=INNODB;

2.21 Assigns

Relational Model

Assigns(email1, email2, card_id)

Foreign Key email1 References User(email)

Foreign Key email2 References User(email)

Foreign Key card_id References Card

Functional Dependencies

email2, card_id -> email1

Candidate Keys

{(email2, card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Assigns (

email1 VARCHAR(32) NOT NULL,

email2 VARCHAR(32) NOT NULL,

card_id INT NOT NULL,

PRIMARY KEY (email2, card_id),

FOREIGN KEY (email1) REFERENCES User(email),

FOREIGN KEY (email2) REFERENCES User(email),

FOREIGN KEY (card_id) REFERENCES Card) ENGINE=INNODB;

2.22 HasMeeting

Relational Model

HasMeeting(team_id, meeting_id)

Foreign Key team_id References Team

Foreign Key meeting_id References Meeting

Candidate Keys

{(team_id, meeting_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasMeeting (

team_id INT NOT NULL,

meeting_id INT NOT NULL,

PRIMARY KEY (team_id, meeting_id),

FOREIGN KEY (team_id) REFERENCES Team,

FOREIGN KEY (meeting_id) REFERENCES Meeting) ENGINE=INNODB;

2.23 Arranges

Relational Model

Arranges(email, team_id, meeting_id)

Foreign Key email References Leader

Foreign Key team_id References Team

Foreign Key meeting_id References Meeting

Candidate Keys

{(email, team_id, meeting_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Arranges (

email VARCHAR(32) NOT NULL,

team_id INT NOT NULL,

meeting_id INT NOT NULL,

PRIMARY KEY (email, team_id, meeting_id),

FOREIGN KEY (email) REFERENCES Leader(email),

FOREIGN KEY (team_id) REFERENCES Team),

FOREIGN KEY (meeting_id) REFERENCES Meeting) ENGINE=INNODB;

2.24 HasReport

Relational Model

HasReport(board_id, report_id)

Foreign Key board_id References Board

Foreign Key report_id References Report

Candidate Keys

{(board_id, report_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE HasReport (

board_id INT NOT NULL,

report_id INT NOT NULL,

PRIMARY KEY (board_id, report_id),

FOREIGN KEY (board_id) REFERENCES Board,

FOREIGN KEY (report_id) REFERENCES Report) ENGINE=INNODB;

2.25 SendMessage

Relational Model

SendMessage(email1, email2, message)

Foreign Key email1 References User(email)

Foreign Key email2 References User(email)

Candidate Keys

{(email1, email2, message)}

Normal Form

BCNF

Table Definition

CREATE TABLE SendMessage (

email1 VARCHAR(32) NOT NULL,

email2 VARCHAR(32) NOT NULL,

message VARCHAR(512) NOT NULL,

PRIMARY KEY (email1, email2, message),

FOREIGN KEY (email1) REFERENCES User(email),

FOREIGN KEY (email2) REFERENCES User(email)) ENGINE=INNODB;

2.26 ArchiveMessage

Relational Model

ArchiveMessage(email, message)

Foreign Key email References User

Foreign Key message References SendMessage

Candidate Keys

{(email, message)}

Normal Form

BCNF

Table Definition

CREATE TABLE ArchiveMessage (

email VARCHAR(32) NOT NULL,

message VARCHAR(512) NOT NULL,

PRIMARY KEY (email, message),

FOREIGN KEY (email) REFERENCES User) ENGINE=INNODB;

2.27 ArchiveCard

Relational Model

ArchiveCard(email, card_id)

Foreign Key email References User

Foreign Key card_id References Card

Candidate Keys

{(email, card_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE ArchiveCard (

email VARCHAR(32) NOT NULL,

card_id INT NOT NULL,

PRIMARY KEY (email, card_id),

FOREIGN KEY (email) REFERENCES User,

FOREIGN KEY (card_id) REFERENCES Card) ENGINE=INNODB;

2.28 Promotes

Relational Model

Promotes(email_l, email_e, team_id)

Foreign Key email_l References Leader(email)

Foreign Key email_e References Employee(email)

Foreign Key team_id References Team

Candidate Keys

{(email_l, email_e, team_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Promotes (

email_l VARCHAR(32) NOT NULL,

email_e VARCHAR(32) NOT NULL,

team_id INT NOT NULL,

PRIMARY KEY (email_l, email_e, team_id),

FOREIGN KEY (email_l) REFERENCES Leader(email),

FOREIGN KEY (email_e) REFERENCES Employee(email),

FOREIGN KEY (team_id) REFERENCES Team) ENGINE=INNODB;

2.29 Fires

Relational Model

Fires(email_l, email_e, team_id)

Foreign Key email_l References Leader(email)

Foreign Key email_e References Employee(email)

Foreign Key team_id References Team

Candidate Keys

{(email_l, email_e, team_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE Fires (

email_l VARCHAR(32) NOT NULL,

email_e VARCHAR(32) NOT NULL,

team_id INT NOT NULL,

PRIMARY KEY (email_l, email_e, team_id),

FOREIGN KEY (email_l) REFERENCES Leader(email),

FOREIGN KEY (email_e) REFERENCES Employee(email),

FOREIGN KEY (team_id) REFERENCES Team) ENGINE=INNODB;

3. Functional Dependencies and Normalization of Tables

All functional dependencies and normal forms are indicated in Relational Schema in section 2 of this Project Design Report. We checked whether all relations in our design are in Boyce-Codd normal form. We concluded that no decomposition is required.

4. Functional Components

4.1 Use Cases /Scenarios

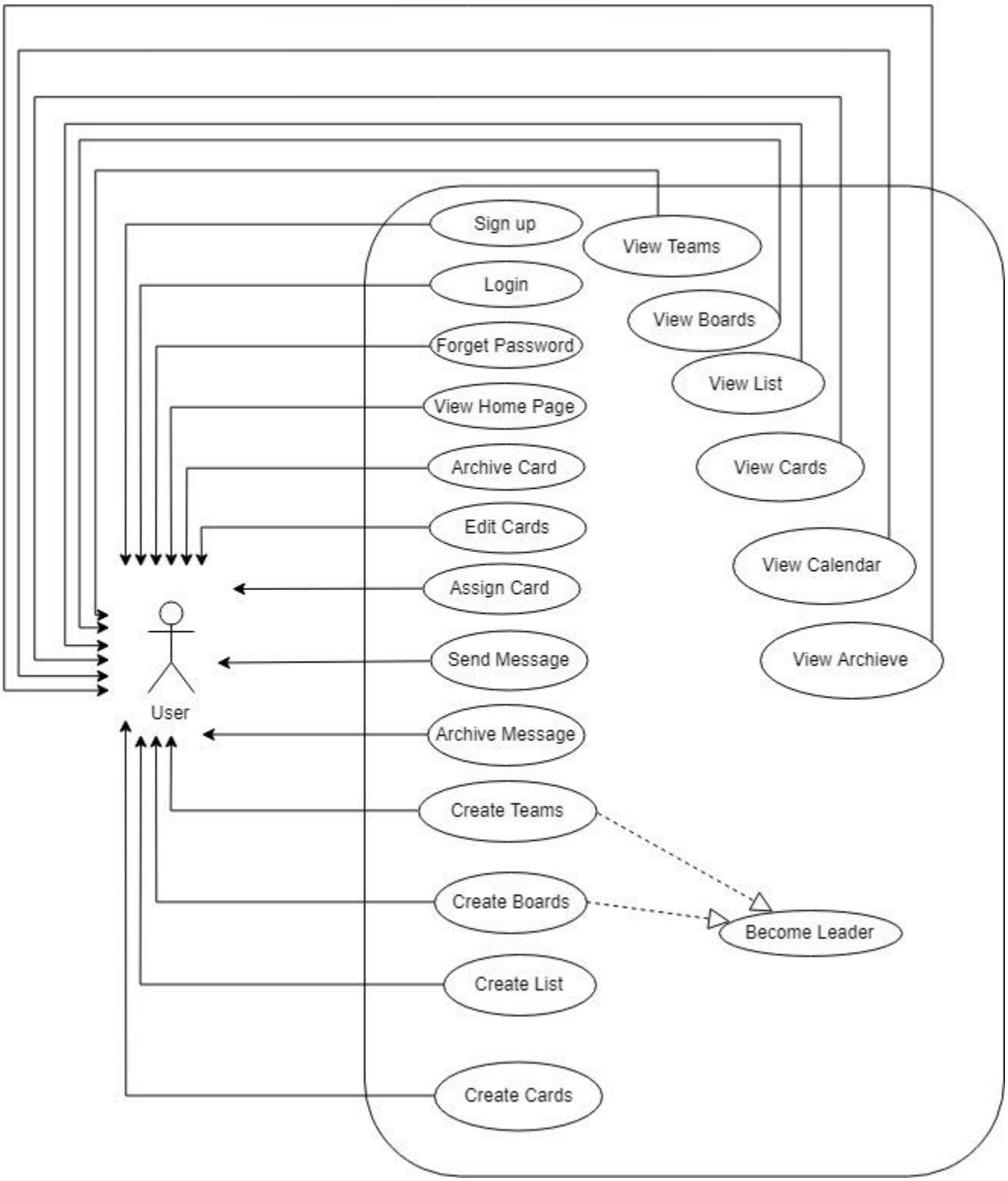
In AtaBase Project Tracking System, there are two types of user: employee and leader. Both of them called user but leader has more power than employees. Therefore, we only show the user's use case that also involves employee, and leader.

User:

- **User can sign up to the system with email address and a password.**
- **User can login to the system with email address and a password.**
- **By clicking Forget Password option user can receive his password to his email.**
- **By clicking home icon user can see the home page. Home page includes the board and teams' section in it.**
- **By clicking archive card user can sent the card to his archive with its own card id, card name, description, label, card deadline, comments and images.**
- **By clicking to the card user can edit the card's attributes except for its card id**
- **By clicking card and choose assign option the user can assign card to other users. This assign operation will affect other users and their calendars.**
- **User can send message to other users.**
- **User can archive the messages by clicking archive button; message's sender, receiver and the message will be archived.**
- **User can create teams and this creation automatically makes him as the leader of the team.**

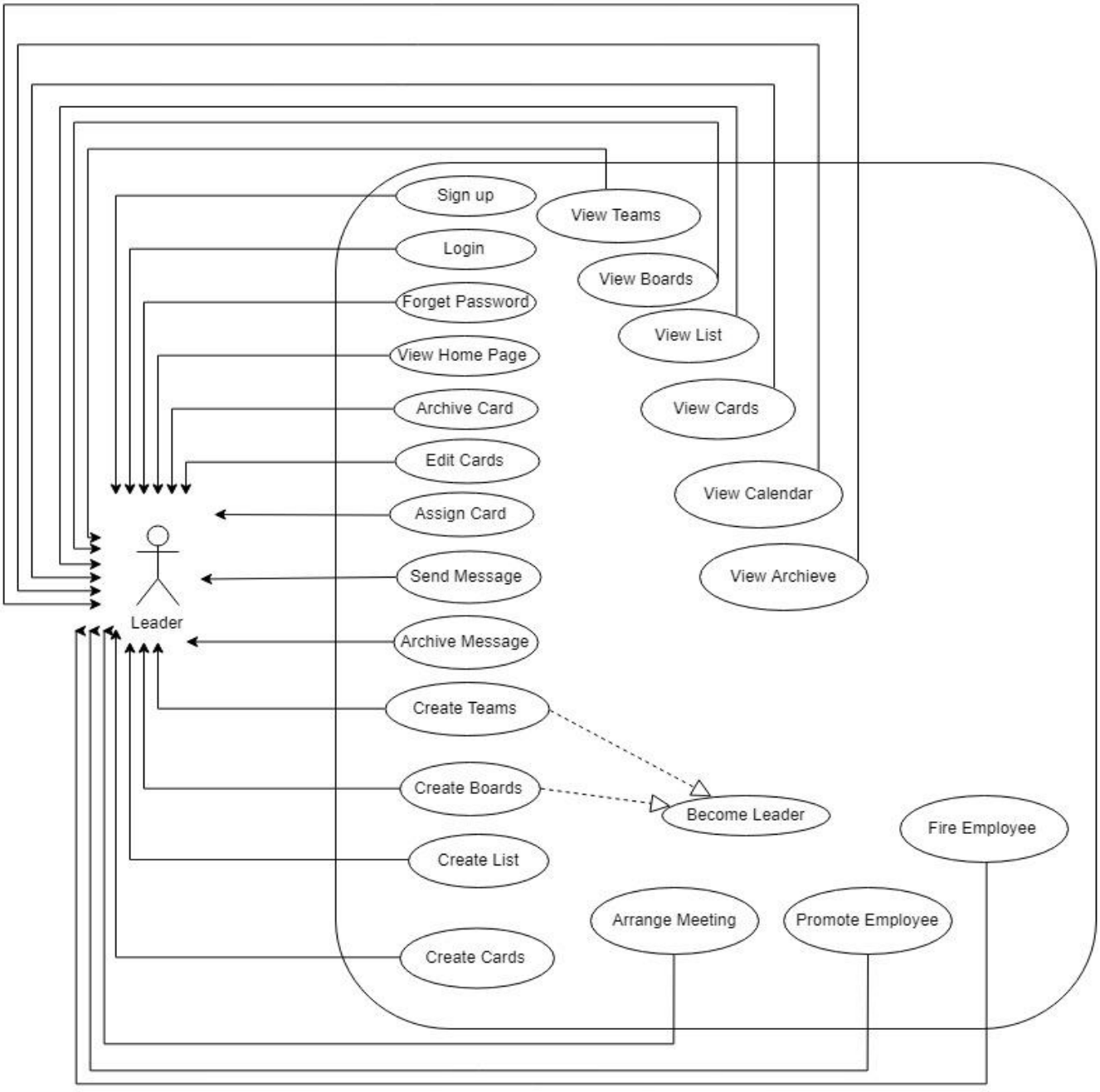
- **User can create boards and this makes him automatically leader of the team. By creating the board, board id is automatically created and user can give name deadline to board. He can use privacy and star and change the background.**
- **User can create Lists by clicking the add list button and change the name by clicking on it. List id is automatically created.**
- **User can create cards by clicking create cards button. Card id is automatically created. Card name, description, label colors, deadline, comments, and images can be added whenever a user wants.**
- **User can see the Teams and its attributes on teams' section in the home page.**
- **User can view the Board and its attributes on boards section in home page.**
- **User can see the lists and its attributes in the board.**
- **User can view the cards and its attributes in lists.**
- **User can user can see his own calendar.**

- **User can see the archive which includes messages or cards in it. Leader:**



Leader can do all the attributes above and he has some extra authority like:

- Leader can arrange meetings in a team. That meetings meeting id is created by the system but time date and place needed to be explained by leader and also notes of the meeting can be added.
- Leader can promote an employee to a leader in a team.
- Leader can fire an employee in a team.



4.2 Algorithms

4.2.1 Search Related Algorithms

Users can search through our system by using a word to search for a user.

4.2.2 Logical Requirements

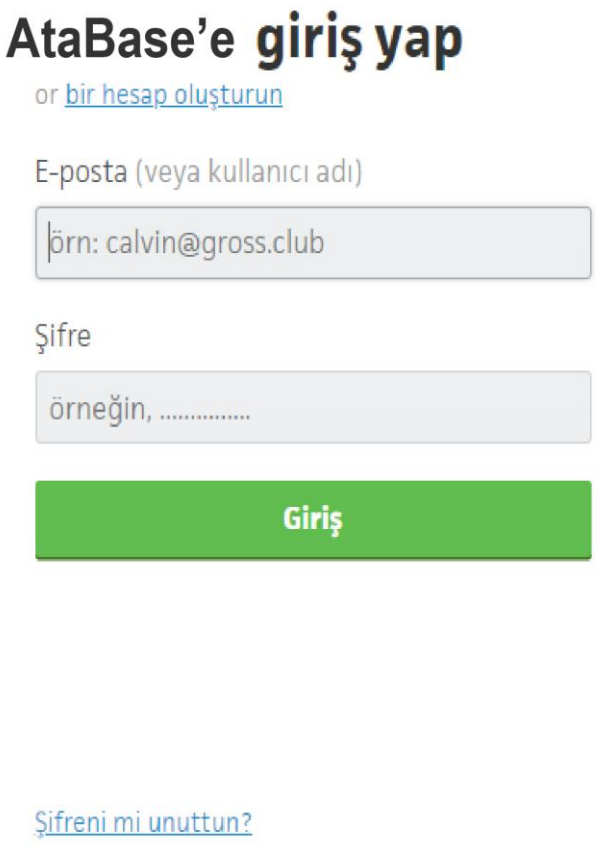
Our application will have many logical requirements. For example, a user cannot make an empty comment, sign up with an empty name, username or password. We will look up for that kind of requirements in our system for preventing future errors.

4.3 Data Structures

For the attribute domains we use Numeric type, int, Date and Time and String Type data types of MySQL.

5. User Interface Design and Corresponding SQL Statements

5.1 Login



Inputs: @email, @password

Process: When user enters the system, this page shows up. Here, user can login the system. If user forgot password; he/she can go to change password page.

SQL statements:

Enter the system

Select *

From User

Where email = @email and password = @password

5.2 Register

Input: @email @name @password @specialty

Process: When user enters the system, this page shows up. Here, user can register the system by using name, password, email, and specialty.

SQL Statement:

INSERT INTO User

VALUES (@email, @name, @password, @ specialty);

5.3 Home Page

Inputs: @email, @team_id, board_id

Process: The teams and boards of the user shown as only their names to the user.

SQL Statements:

Showing team names:

```
SELECT team_name
        FROM HasUser, Team
        WHERE email = @email
```

Showing Board Names:

```
CREATE VIEW teams_of_user AS (SELECT team_id_1
        FROM HasUser
        WHERE email = @email)
SELECT board_name
FROM HasBoard ,teams_of_user
WHERE team_id = team_id_1
```

5.4 Team Page

Inputs: @email

Process: When User enters the home page, s/he can choose teams to see all the teams s/he involved.

SQL Statements:

```
SELECT team_name
        FROM HasUser, Team
        WHERE email = @email
```

5.5 Board Page

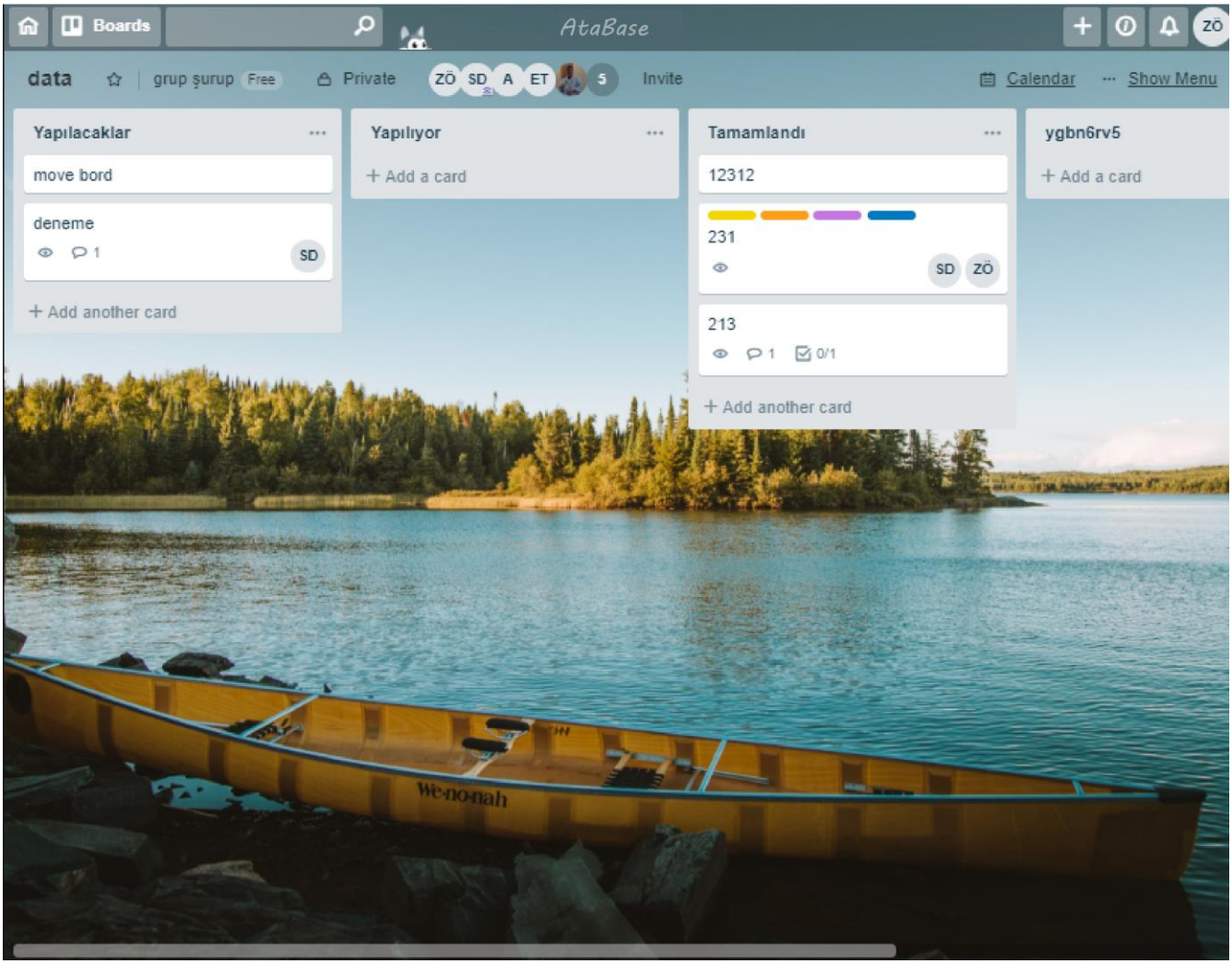
Inputs: @email, @team_id, board_id

Process: When user enters the home page s/he can choose boards to see all the board s/he has.

SQL Statements:

```
CREATE VIEW teams_of_user AS (SELECT team_id_1
        FROM HasUser
        WHERE email = @email)
SELECT board_name
FROM HasBoard ,teams_of_user
WHERE team_id = team_id_1
```

5.6 List Page



Inputs: @email @list_id @list_name @card_name @board_id

Process: when user enters the board sees the lists that are created by user and sees cards in it

See the list its cards

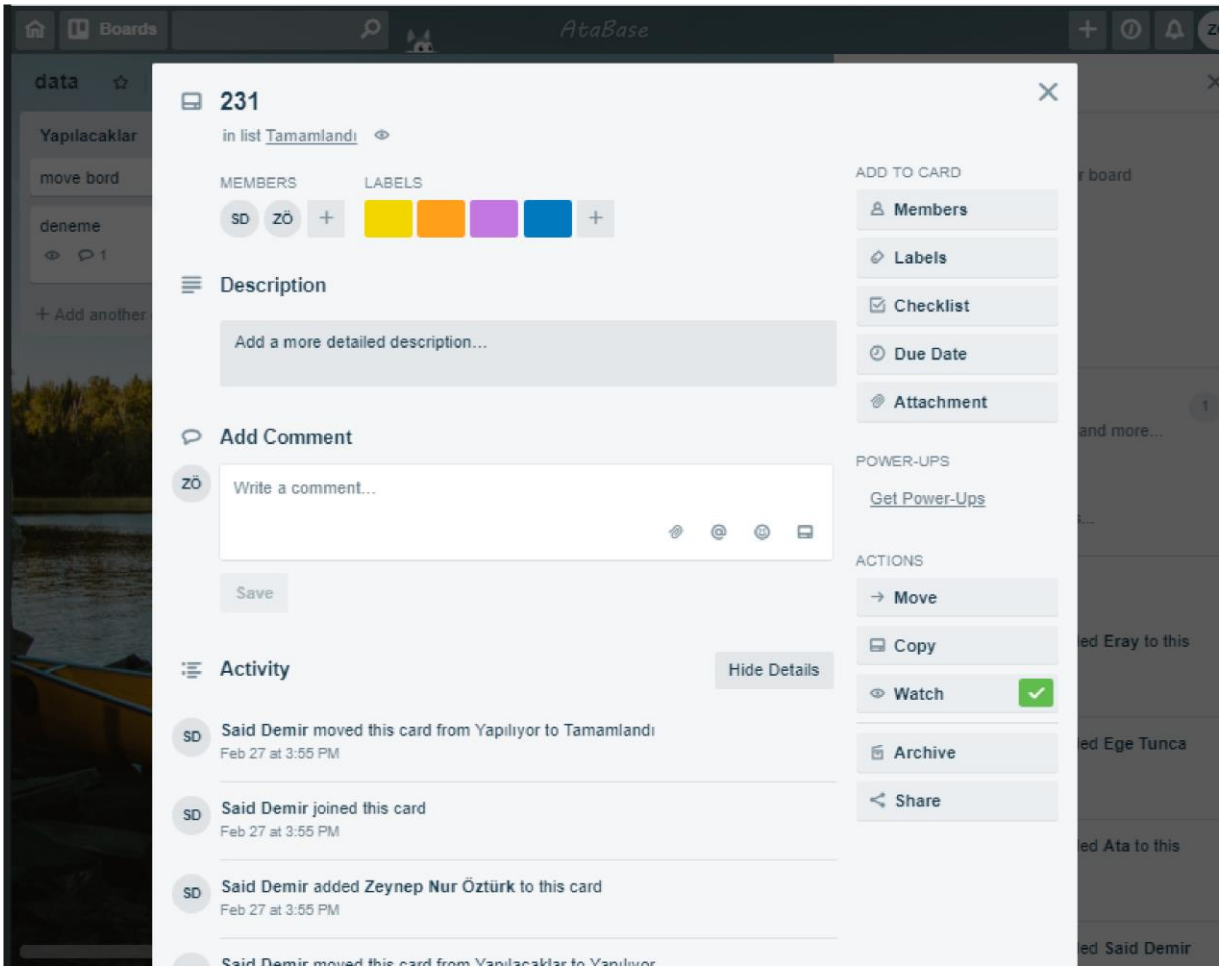
SQL Statements:

SELECT list_name, card_name

FROM HasList NATURAL JOIN List, HasCard NATURAL JOIN Card,

WHERE board_id = @board_id AND list_id = @list_id

5.7 Card Page



Inputs: @clicked_card_id, @card_name, @description, @label, @card_deadline, @comment, @images

Process: When user clicks on the Card the detailed card page will be shown like in the picture and s/he can edit or assign card to users

SQL Statements:

Show whole card attributes on page

SELECT card_name, description, label, card_deadline, comment, images

FROM Card

WHERE card_id = @clicked_card_id

Edit Card Attributes:

Inputs: @clicked_card_id, @card_name_new, @description_new, @label_new, @card_deadline_new,

@comment_new, @images_new

SQL Statements:

UPDATE Card

SET card_name = @card_name_new, description = @description_new, label = @label_new,

card_deadline = @card_deadline_new, comment = @comment_new, images = @images_new

WHERE card_id = @clicked_card_id

Assign Cards to user

Inputs: @word, @clicked_card_id @email

SQL Statements:

CREATE VIEW assign_search AS (

    SELECT email AS email2

    FROM User

    WHERE name like '%@word%')

INSERT INTO Assigns

VALUES (@email, email2, @clicked_card_id)

5.8 Calendar Page

Inputs: @email

Process: When User clicks on the calendar he sees all the deadlines that are assigned to him.

SQL Statements:

```
CREATE VIEW assigned AS (SELECT card_name, card_deadline
        FROM Card, Assigns
        WHERE email = @email)
CREATE VIEW meeting_of_user AS (SELECT team_id, date, time, meeting_id
        FROM HasUser, Team NATURAL JOIN Meeting
        WHERE email = @email
SELECT *
FROM meeting_of_user, assigned
```

5.9 User Search

Inputs: @word

Process: User enters a key word to search for a User

SQL Statements:

```
SELECT email, name
FROM User
WHERE name like '%@word%'
```

5.10 Message Page

Inputs: @email_1, @email_2, @message

Process: User clicks on message and can select the user he want to text and send the message

SQL Statements:

Showing All Users:

```
SELECT email, name
FROM User
```

Showing old messages of that user

SELECT message

From SendMessage

WHERE (email1= @email_1 AND email2= @email_2) OR (email1 = @email_2 AND email2 = @email_1)

Sending Message:

INSERT INTO SendMessage

VALUES (@email_1, @email_2, @message)

5.11 Change Password

Inputs: @email, @password, @repassword

Process: user enters the email and decides a new password.

SQL Statements:

UPDATE User

SET password = @password

WHERE email = @email AND @password = @repassword

6. Advance Database Components

6.1 View

We will have to use different views in the project, one for home page, one for card page one for board page, one for calendar page.

6.1.1 Home Page View

User will see the boards and teams of himself. Therefore, to see the boards and teams and we need to use views. We already show the code of the home page view in section 5.3

6.1.2 Board Page View

User will be able to see the boards that he involves in. Therefore, we need to use views. We already show the code in the section 5.5

6.1.3 Card Page View

User will be able to see the cards in lists that he has. Therefore, we need to use views while assigning card to any user. We already show the code in the section 5.7

6.1.4 Calendar Page View

User will be able to see the calendar. Therefore to see all the meetings and cards deadline here, we needed to use views as we already shown in section 5.8

6.2 Triggers

- When a User fired from system all connections will be deleted
- When a User promoted from system the role needed to updated
- When a board is deleted by leader all the lists and cards reports needed to be deleted
- When a meeting is deleted the calendar needed to be update
- When a meeting is assigned the calendar needed to be updated

6.3 Constraints

- If someone needs to use this system he should login with a unique email
- If he wants to see the board or team page he should enter the corresponding pages
- User need to search other users if he wants to assign a work to someone

7. Implementation Plan

At data layer we will use MySQL Server in our project as database management system. For application logic and user interface we will code in PHP and JavaScript.