Database Project Report
PicBook
(A pinterest inspired website)

PicBook is a social media website that allows its users to share interesting photos which can be something they found interesting while browsing the web or something that they clicked from their smartphone.

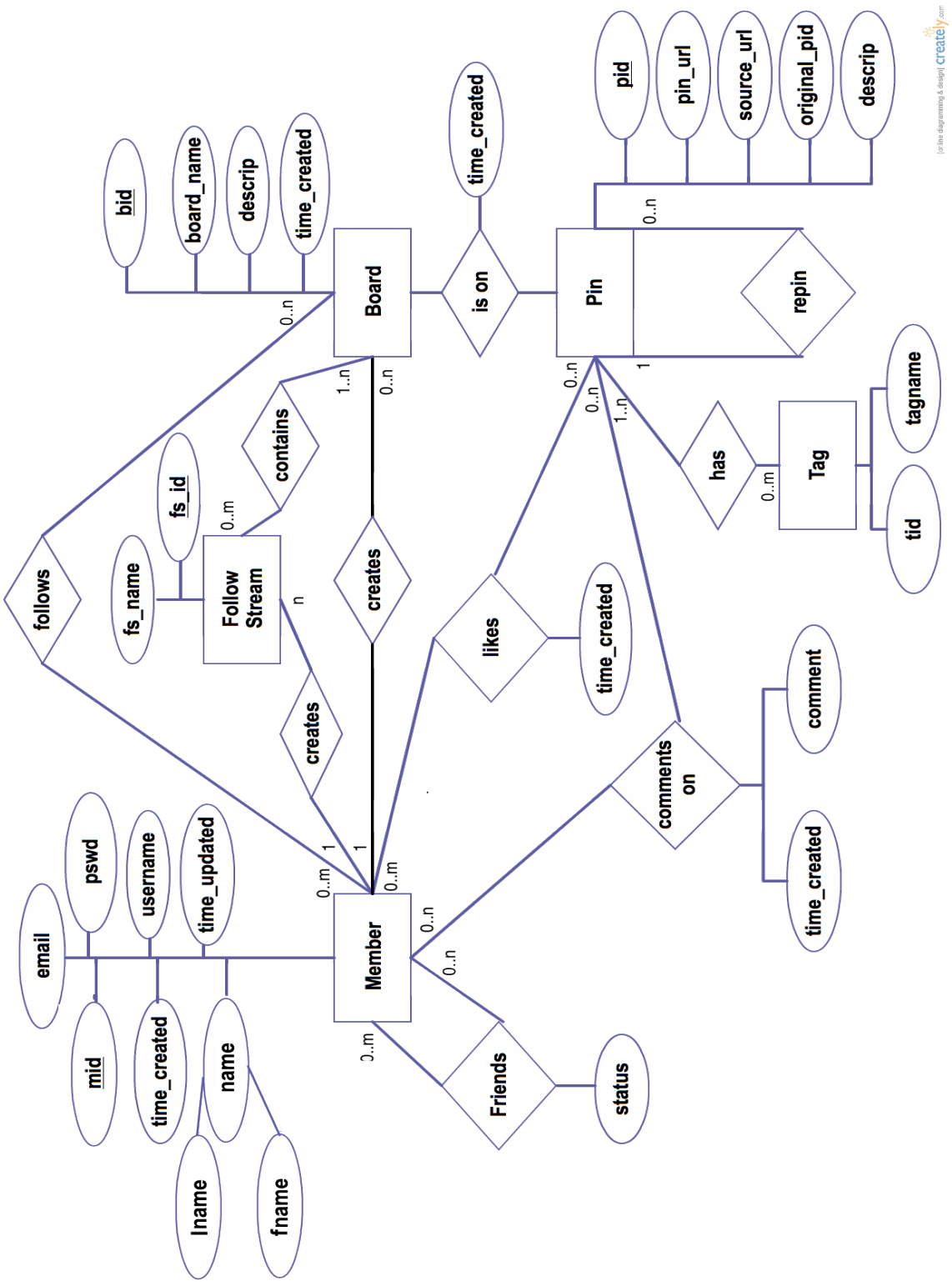
For this project we chose MS SQL Server as the backend because we plan to use ASP.NET as the frontend and it plays well with SQL Server.

The basic assumptions for the Project:

- Users signup as members for the website by giving some basic details (like: Name and Email) and get to choose a Username and Password to access their account in the future.
- Once logged in a user can:
    - View pins from all the other users,
    - Like them,
    - Comment on them (only if the owner user permits it. Details discussed later),
    - Create pins (either by copying a link from the web or uploading a picture from their PC),
    - Repin pins (copying pins created by some other user),
    - Delete pins (this includes both original pins and repins)
    - Create boards,
    - Decide who can comment on their boards (Only them, only friends or everyone. This is a setting they decide for each board separately.),
    - Follow boards (so that it is displayed in their default stream),
    - Create custom followstreams (which is a page which displays results from other's boards which have been followed and grouped by the user as followstream),
    - Send other users friend requests,
    - Approve friend requests which others have sent to them,
    - Decline friend requests which others have sent to them,
    - Tag pins
- When an original pin is deleted all the repins that originated from that pin get deleted.
- When a Repin is deleted only that Repin is deleted.
- When a pin is liked the like is attributed to the original pin.
- When a pin (Original pin or Repin) is commented on, the comment is attributed to that pin only.

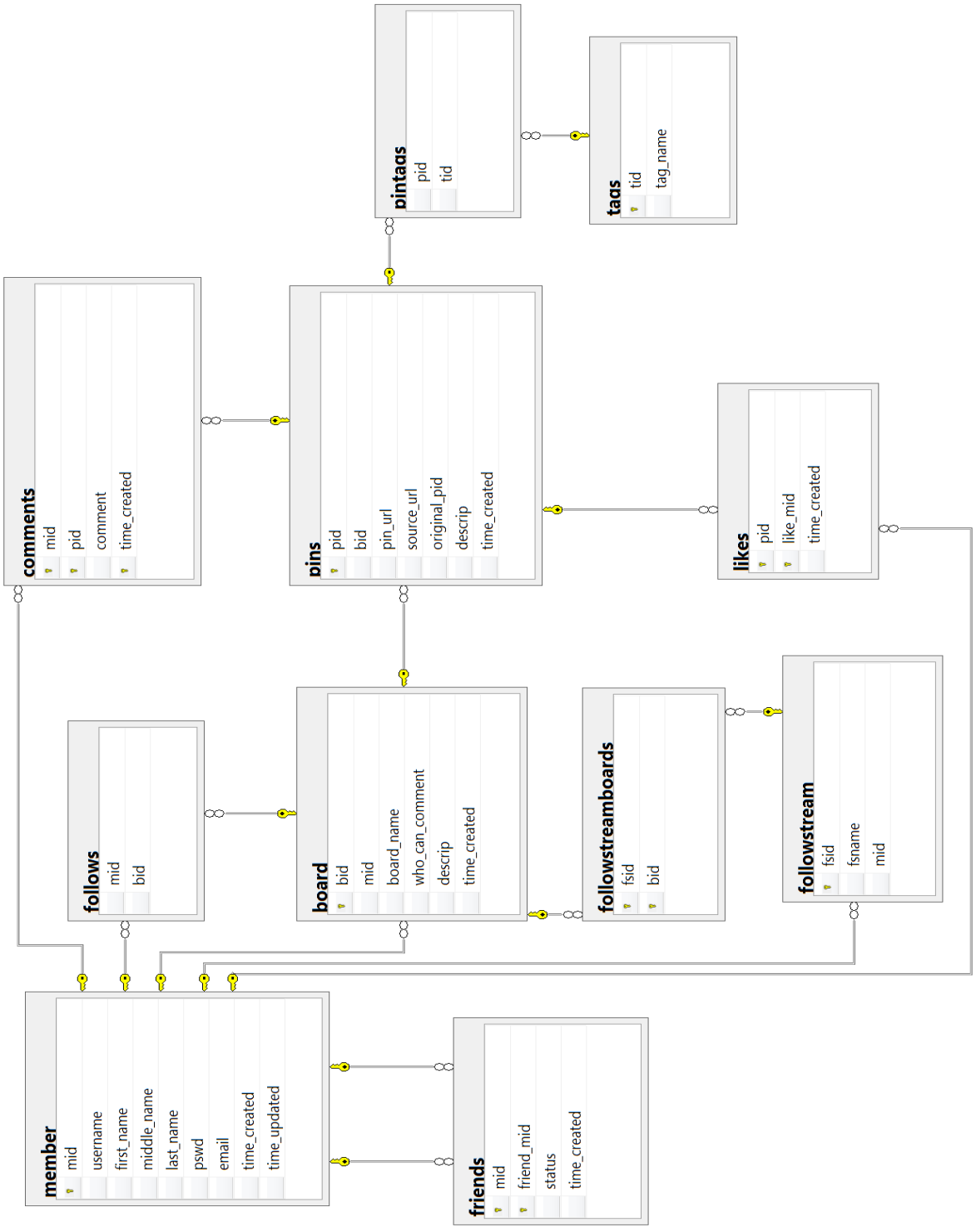
- Permission for commenting on a board is by default “Everyone”. It can be changed to “Friends only” or “Me only” by the user later on.
- ***Pin Images are not stored in the Database as BLOBs.***
- ***When an image is uploaded by the user for creating a new pin the image is stored on the file server and a url is generated to refer to the image.***
- ***When the user gives a pin url of a image on the web the image is again copied to the file server and a url is generated to refer to the image. This url is stored in the pin_url column of the Pins table. We always have the source_url field which points to the original webpage from where the pin image was copied.***
- A followstream can contain only those boards which have been followed by the user separately.
- When a user deletes a followstream, only the followstream gets deleted and he/she keeps on following the individual boards that were the part of that followstream.

ER DIAGRAM



Relational Database Schema

Database Schema



NOTE: Primary keys and Foreign keys are denoted in the diagram.

Database Schema Details

Member Table
The member table consists of the following fields:

mid	This field stores the member ID of the members. It is generated by Auto-Increment and the user has no control over what ID is assigned to him. This field is also the primary key for member table.
username	Username of users choice. Right now we are allowing users to have identical usernames. This might change if we feel we need unique username for each user.
first_name	First name of the member.
middle_name	Middle name of the member.
last_name	Last name of the member.
pswd	Stores the password set by user while creating account.
email	Email address of the member. A member uses his email address to log into the website and therefore the email address has to be unique for each member.
time_created	Timestamp for account creation.
time_updated	Timestamp for account updation.

```
create table member
(
  mid int NOT NULL identity(1,1),
  username varchar(25) NOT NULL DEFAULT '',
  first_name varchar(50) NOT NULL DEFAULT '',
  middle_name varchar(50) NOT NULL DEFAULT '',
  last_name varchar(50) NOT NULL DEFAULT '',
  pswd varchar(max) NOT NULL,
  email varchar(30) NOT NULL,
  time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
  time_updated datetime2(7) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  constraint pk_member primary key(mid),
  constraint uq_member unique(email)
)
```

Board Table
The board table consists of the following fields:

bid	This field stores the board ID for every board created. It is generated by Auto-Increment and the user has no control over what ID is assigned to a board. This field is also the primary key for board table.

mid	This field tells us which member created this board. This field is a foreign key which refers to the mid field of the Member table. Also this field has a on delete cascade property which essentially means that whenever an member row in the member table (parent table) is deleted, all the boards related to that mid are are deleted from the board table (child table).
board_name	Name of the board. Different boards can have the same name but one member can only have one board of a particular name.
who_can_comment	This field is used as the permission tracker for each board. If this field says ‘e’, then anyone who is a member of the website can comment on the pins on this board. If it says ‘f’, then only the friends of the member who owns this board may comment on the pins on this board. If it says ‘m’, then only the owner may comment on the pins on this board. The default setting is ‘e’ which means any one can comment on the pins on a board by default.
descrip	A few lines to describe what the board is about.
time_created	Timestamp for board creation.

```
create table board
(
  bid int NOT NULL identity(1,1),
  mid int NOT NULL,
  board_name varchar(25) NOT NULL,
  who_can_comment varchar(1) NOT NULL DEFAULT 'e',
  descrip text NOT NULL,
  time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
  constraint pk_board primary key(bid),
  constraint fk_board foreign key(mid) references member(mid) ON DELETE CASCADE,
  constraint uq_board unique(mid, board_name),
  constraint ck_who_can_comment check (who_can_comment in ('f','e','m'))
)
```

Pins Table
The pins table consists of the following fields:

pid	This field stores the pin ID for every pin created. It is generated by Auto-Increment and the user has no control over what ID is assigned to a pin. This field is also the primary key for pin table. Every time a pin is repinned a new entry is made to the pins table and a new pid is assigned.
bid	This field tells us which which board does this pin belong to. This field is a foreign key which refers to bid in the board table. Owner of the pin is found by looking up the owner of the board it is pinned on. Also this field has a on delete cascade property which essentially means that whenever a board row in the board table (parent table) is deleted, all the pins related to that bid are are deleted from the board table (child table).
pin_url	URL containing only the image of the pin.
source_url	URL containing the webpage from where the pin was captured.

original_pid	This field tells us if a pin is original or a repin of some other pin. **If the original_pid field is the same as the pid field, this means that the pin is original. If they don't match then the original_pid refers to the original pin and this pin is a repin.**
descrip	A short description of the pin.
time_created	Timestamp for pin creation.

```
create table pins
(
pid int NOT NULL identity(1,1),
 bid int NOT NULL,
 pin_url text NOT NULL,
 source_url text,
 original_pid int NOT NULL DEFAULT Ident_current('pins'),
 descrip text NOT NULL,
 time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
 constraint pk_pins primary key(pid),
 constraint fk_p_bid foreign key(bid) references board(bid) ON DELETE CASCADE
);
```

Likes Table
The likes table consists of the following fields:

pid	This field refers to the pin which is being liked. This field is a foreign key which refers to the pid in the pins table. This field has a on delete cascade property which essentially means that whenever a pin row in the pin table (parent table) is deleted, all the likes related to that pin are are deleted from the board table (child table).
like_mid	This field refers to the member who is liking the field. This field is a foreign key which refers to the mid in the member table.
time_created	Timestamp of when the like was made.

pid, like_mid fields together form the primary key.

```
create table likes
(
 pid int NOT NULL,
 like_mid int NOT NULL,
 time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
 constraint fk_l_pid foreign key(pid) references pins(pid)  ON DELETE CASCADE,
 constraint fk_l_lm foreign key(like_mid) references member(mid),
 constraint pk_like primary key(pid, like_mid)
 );
```

Comments Table
The comments table consists of the following fields:

mid	The mid of the member who commented on the pin. This field is a foreign key which refers to the mid in the member table.
pid	The pid of the pin being commented on. This field is a foreign key which refers to the pid in the pins table. This field has a on delete cascade property which essentially means that whenever a pin row in the pin table (parent table) is deleted, all the comments related to that pin are are deleted from the board table (child table).
comment	The actual comment text.
time_created	Timestamp for comment creation.

mid, pid, time_created fields together form the primary key.

```
 create table comments
(
 mid int NOT NULL,
 pid int NOT NULL,
 comment text NOT NULL,
 time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
 constraint pk_comments primary key(mid, pid, time_created),
 constraint fk_c_mid foreign key(mid) references member(mid),
 constraint fk_c_pid foreign key(pid) references pins(pid) ON DELETE CASCADE
);
```

Friends Table
The friends table consists of the following fields:

mid	Member who receives the friend request.
friend_mid	Member who sends the friend request.
status	Status of the friend request. A friend request can either be pending (‘p’) or approved (‘y’). When a member declines the friend request the record is deleted from the table so that they can send the request again in the future.
time_created	Timestamp

mid, friend_mid fields together form the primary key.

```
create table friends
(
 mid int NOT NULL,
 friend_mid int NOT NULL,
 status varchar(1) NOT NULL,
 time_created datetime2 NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
  constraint pk_friend primary key(mid, friend_mid),
  constraint fk_fr_mid foreign key(mid) references member(mid) ON DELETE CASCADE,
  constraint fk_fr_friend_mid foreign key(friend_mid) references member(mid),
  constraint ck_status check (status in ('y','p'))
);
```

Followstream Table
The followstream table consists of the following fields:

fsid	This field stores the Followstream ID for every followstream created by every user. It is generated by Auto-Increment and the user has no control over what ID is assigned to a followstream. This field is also the primary key for Followstream table.
fsname	Name of the followstream.
mid	Member ID of the member who created this followstream. This field is a foreign key which refers to the mid field of the Member table. This field has a on delete cascade property which essentially means that whenever a member row in the member table (parent table) is deleted, all the followstream created by that member are deleted from the followstream table (child table).

```
create table followstream
(
  fsid int NOT NULL identity(1, 1),
  fsname varchar(50) NOT NULL,
  mid int NOT NULL,
  constraint pk_follow primary key(fsid),
  constraint fk_f_mid foreign key(mid) references member(mid) ON DELETE CASCADE
);
```

Followstreamboards table
The followstreamboards table consists of the following fields:

fsid	Followstream ID. This field is a foreign key that refers to the fsid in the followstream table. This field has a on delete cascade property which essentially means that whenever a followstream row in the followstream table (parent table) is deleted, all the followstreamboards contained in that followstream are deleted from the followstreamboards table (child table).
bid	Board ID of the boards associated with that fsid. This field is a foreign key which refers to bid in the board table.

fsid, bid fields together form the primary key.

```
create table followstreamboards
(
fsid int NOT NULL,
bid int NOT NULL,
constraint fk_fol_fsid foreign key(fsid) references followstream(fsid) ON DELETE CASCADE,
constraint fk_fol_bid foreign key(bid) references board(bid),
constraint pk_follows primary key(fsid, bid)
```

);

Follows table

The follows table consists of the following fields:

mid	Member ID who is following the board. This field is a foreign key which refers to the mid field of the Member table. This field has a on delete cascade property which essentially means that whenever a member row in the member table (parent table) is deleted, all the follow entries in the follows table associated with that member are deleted from the follows table (child table).
bid	Board ID of the board that is being followed. This field is a foreign key which refers to the bid field of the Board table.

mid, bid fields together form the primary key.

```
create table follows
(
mid int not NULL,
bid int not NULL,
constraint fk_folwg_mid foreign key(mid) references member(mid) ON DELETE CASCADE,
constraint fk_folwg_bid foreign key(bid) references board(bid),
constraint pk_folwg primary key(mid, bid)
)
```

Tags Table

The tags table consists of the following fields:

tid	This field stores the tag ID for every tag created. It is generated by Auto-Increment and the user has no control over what ID is assigned to a tag. This field is also the primary key for tags table. Every time a tag is created a new entry is made to the tags table and a new tid is assigned. If a user goes on to tag a picture with a tag that already exists the system makes sure that the existing tag is referred to and a new tag is not created.
tag_name	Name of the tag. Tag names have to be unique.

```
create table tags
(
tid int identity(1,1),
tag_name varchar(50) NOT NULL UNIQUE,
constraint pk_tags primary key(tid)
);
```

Pintags Table

The Pintags table consists of the following fields:

pid	Pin ID of the pin that is being tagged. This field is a foreign key which refers to the pid field of the Pins table.

tid	Tag ID. This field is a foreign key which refers to the tid field of the Tags table.

pid, tid fields have a unique constraint that means a particular pin can be tagged by a particular tag only once.

```
create table pintags
(
pid int NOT NULL,
tid int NOT NULL,
constraint fk_pt_pid foreign key(pid) references pins(pid),
constraint fk_pt_tid foreign key(tid) references tags(tid)
constraint uq_pt_pidtid unique(pid, tid)
);
```

(d) Write SQL queries (or sequences of SQL queries) for the following tasks.

We chose to embed each query into procedures so that it is easy to call them once we develop the frontend.

1) Signing Up, Creating Boards, and Pinning: Write queries that users need to sign up, to login, to create or edit their profile, to create pinboards, to pin a picture, and to delete a pinned picture.

Signing Up

```
CREATE PROCEDURE CreateAccount(
        @username varchar(25),
        @first_name varchar(50),
        @middle_name varchar(50),
        @last_name varchar(50),
        @pswd varchar(max),
        @email varchar(30))
AS
BEGIN
        insert into member(username,first_name,middle_name,last_name,pswd,email)
        values (@username,@first_name,@middle_name,@last_name,@pswd,@email);
END
```

Login

```
CREATE PROCEDURE CheckPassword(
   @username VARCHAR(30),
   @password varchar(max))
AS
BEGIN
        IF EXISTS(SELECT * FROM member WHERE email = @username AND pswd =
@password)
                SELECT 'true' AS UserExists
        ELSE
                SELECT 'false' AS UserExists
```

```
END
```

Update Profile

```
create procedure UpdateProfile(
        @firstname varchar(50),
        @middle_name varchar(50),
        @last_name varchar(50),
        @username varchar(30))
as
Begin
        update member
        set first_name = @firstname, middle_name = @middle_name, last_name = @last_name
        where email = @username;
end
```

Create Pinboards

```
create procedure CreateBoard(
        @board_name varchar(25),
        @discrip text,
        @mid int)
as
begin
        insert into board(mid, board_name, descrip)
        values(@mid, @board_name, @discrip);
end
```

Create Pins

```
create procedure PinPicture(
        @bid int,
        @pin_url text,
        @source_url text,
        @descrip text)
as
begin
        insert into pins(bid, pin_url, source_url, descrip)
        values(@bid, @pin_url, @source_url, @descrip);
end
```

Create Repins

```
create procedure RepinPicture(
        @bid int,
        @original_pid int,
        @descrip text)
as
begin
        declare @pin_url varchar(max);
```

```
        declare @source_url varchar(max);
        select @pin_url = pin_url, @source_url = source_url from pins where pid =
@original_pid;
        insert into pins(bid, pin_url, source_url, original_pid, descrip)
        values(@bid, @pin_url, @source_url, @original_pid, @descrip);
end
```

Delete pins and repins

```
create procedure DeletePicture(
        @pid int)
as
begin
        declare @original int;
        set @original = (select pid from pins where original_pid = @pid and pid = @pid);
        if(@original is not NULL)
                begin
                        delete from pins
                        where original_pid = @pid;
                end
        else
                begin
                        delete from pins
                        where pid = @pid;
                end
end
```

2) Friends: Write queries for asking another user to be friends, and for answering a friend requests.

Sending Friend Request

```
create procedure SendFriendRequest(
        @mid int,
        @friend_mid int,
        @status varchar(1))
as
begin
        insert into friends(mid,friend_mid,status)
        values(@mid,@friend_mid,@status);
end
```

Updating(Approving/Declining) Friend Request

```
create procedure UpdateFriendRequest(
        @mid int,
        @friend_mid int,
        @status varchar(1))
as
```

```
begin
	if(@status='y')
		begin
			update friends
			set status='y' where mid=@mid and friend_mid=@friend_mid;
		end
	else
		begin
			delete from friends where mid=@mid and friend_mid=@friend_mid
		end

end
```

3) Repinning and Following: Write queries for repinning a picture and for creating a follow stream. Also, write a query that given a follow stream, displays all pictures belonging to that follow stream in reverse chronological order.

Repinning Pin

```
create procedure RepinPicture(
	@bid int,
	@original_pid int,
	@descrip text)
as
begin
	declare @pin_url varchar(max);
	declare @source_url varchar(max);
	select @pin_url = pin_url, @source_url = source_url from pins where pid =
@original_pid;
	insert into pins(bid, pin_url, source_url, original_pid, descrip)
	values(@bid, @pin_url, @source_url, @original_pid, @descrip);
end
```

Creating a Followstream

```
create procedure CreateFollowStream(
	@mid int,
	@fsname varchar(50)
)
as
begin
	insert into followstream(fsname,mid)
	values(@fsname,@mid);
end
```

Adding to Followstream

```
create procedure AddToFollowStream(
	@fs_id int,
```

```
        @bid int
)
as
begin
        insert into followstreamboards(fsid,bid)
        values(@fs_id,@bid);
end
```

Display pins belonging to follow stream in reverse chronological order

```
select pin_url from pins
where bid in (select bid from followstreamboards
                where fsid in(select fsid from followstream
                                where mid=@mid))
order by time_created desc
```

4) Liking and Commenting: Write queries to like a picture, and to add a comment to a picture (while making sure the user is allowed to comment on this picture).

Liking a pin

```
create procedure LikePin(
        @pid int,
        @like_mid int)
as
begin
        insert into likes(pid,like_mid)
        values(@pid,@like_mid);
end
```

Commenting on a pin

```
create procedure CommentPin(
        @mid int,
        @pid int,
        @comment text)
as
begin
        declare @bid int;
        set @bid = (select bid from pins where pid=@pid);
        declare @permission varchar(1)
        set @permission = [dbo].GetPermission(@bid);
        if(@permission = 'f')
                begin
                        declare @friend_mid int;
                        set @friend_mid = (select mid from board where bid = @bid);
                        declare @isFriend varchar(1);
                        set @isFriend = [dbo].IsFriend(@mid, @friend_mid);
```

```
                        if(@isFriend ='y')
                                begin
                                        insert into comments(mid,pid,comment)
                                        values(@mid,@pid,@comment);
                                end
                        else
                                return;
                end
        else if(@permission = 'e')
                begin
                        insert into comments(mid,pid,comment)
                        values(@mid,@pid,@comment);
                end
        else
                begin
                        if( @mid = (select mid from board where bid = @bid))
                                insert into comments(mid,pid,comment)
                                values(@mid,@pid,@comment);
                        else
                                return;
                end
end
```

GetPermission() returns the permission set for the board on which the pin is pinned.

```
create function GetPermission(
        @bid int
)
returns varchar(1)
as
begin
        declare @permission varchar(1);
        select @permission = who_can_comment from board where bid = @bid;
        return @permission;
end
```

IsFriend(mid1, mid2) returns ‘y’ or ’n’ depending if two mid1 and mid2 are friends or not respectively.

```
create function IsFriend(
        @mid int,
        @friend_mid int)
returns varchar(1)
as
begin
        declare @isfriend int;
        declare @retval varchar(1);
        select @isfriend = mid from friends where ((mid=@mid and friend_mid=@friend_mid) or
(mid=@friend_mid and friend_mid=@mid)) and status = 'y';
```

```
        if(@isfriend is not NULL)
                begin
                        set @retval = 'y';
                end
        else
                begin
                        set @retval = 'n';
                end
                return @retval;
end
```

5) Keyword Search: Write a query to perform a keyword search for pictures whose tags match the keywords. Use the contain operator to do so.

```
declare @tagname varchar(50);
set @tagname='nice';
select pid,pin_url from pins
where pid in(select pid from pintags
                where tid in (select tid from tags
                                where contains(tag_name,@tagname)))
```

CS6083 Project Part 2

Implementation

Technologies Used

For the web framework that interacts with the database, we chose to use the ASP.NET web-forms Framework with C#. For the database server, we chose SQL Server 2012 Express. In the frontend, we chose to use the Twitter's Bootstrap 3.0, jQuery 1.10.3 and other jQuery/JS plugins.

While there were many options for frontend design, we chose Bootstrap for its simplicity and overall ease of use.

Project Management

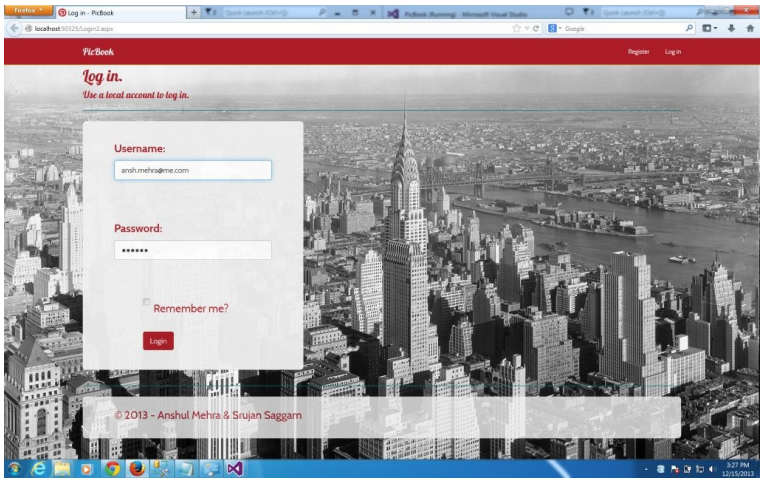
For collaborating on this project, we followed an Agile Software Development approach. We created sprints with lists of features to be done by a certain date, and we stuck to them. We had virtual stand-ups on a bi-daily basis to see how things were going, and whether there were any roadblocks.

For version control and sharing source code, we used Git and GitHub. Version control was important in keeping stable code separate from features we were working on. We followed the git-flow methodology (https://github.com/nvie/gitflow) to ensure a clean and organized repository. Whenever we start work on a new feature, we create a new branch. Once the feature is completed and tested, we merge it back into the development branch. Once the development branch is stable, we merge it back into the master branch for deployment.

User Flow

Sign in / Sign up

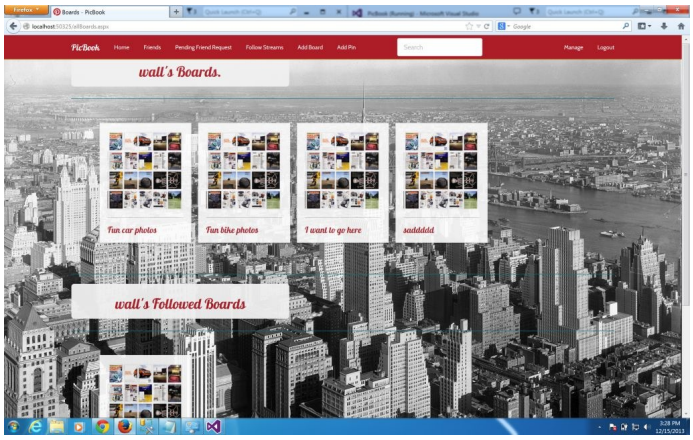
When a user first arrives at the website, he/she is presented with a sign in page. There is a link for users without accounts to sign up. The sign in form simply asks for email and password. The sign up form asks for a username, password, confirm password, first name,middle name, last name and email. Once the user signs in or signs up, they are brought to their home page. Sign up has two constrains, password length should be more than 6



characters and email should be unique. These constrains are implements in frontend and DB.

Navigation Bar

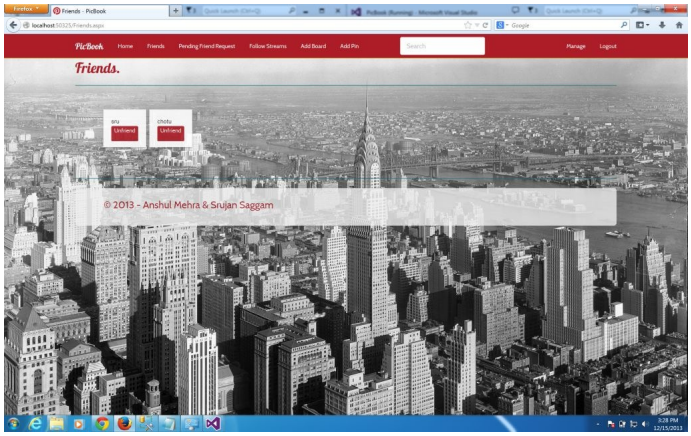
Once the user arrives at their home page, they are presented with notifications and details about their account. At the top of the page is a navigation bar that is there no matter what page the user is on. This navigation bar contains a link to return to their home page, friends page, pending friend request page, follow streams page, add board page, add pin page, search bar for unified search, edit account information page and a logout button.



Boards

On the home page, there is a section that lists some of the user's boards as tiles. The tiles contains a generic board image. Clicking any of those images will bring you to the corresponding board. Inside the board page, the user can browse all the pins on that board.

Also all the boards that are followed by the user are shown on this page in separate section.

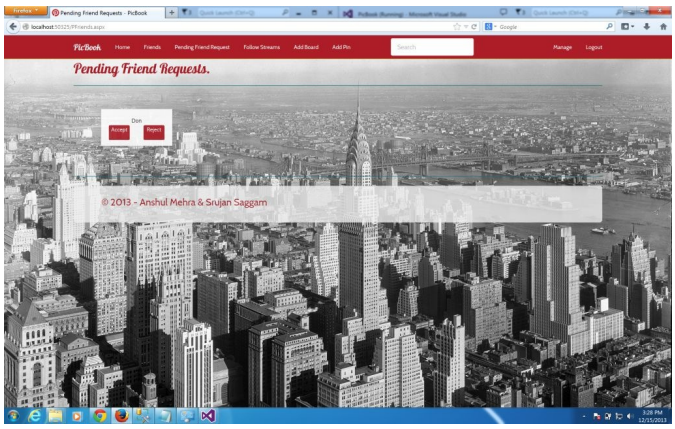


Friends

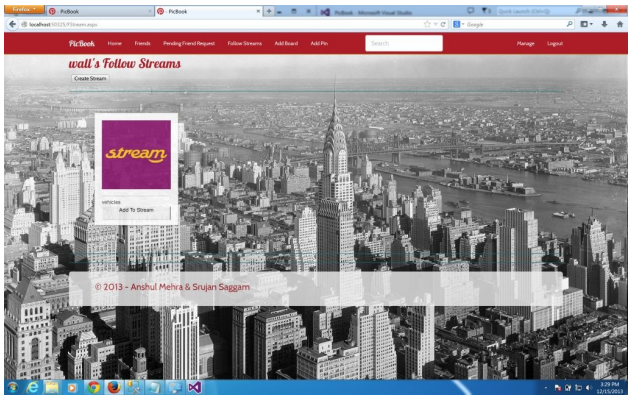
On this page, the user can see all his friends. Each friend tile has a link to the friends profile and the user can click on it to visit that profile and see his friend's boards/pins. Also, there is a unfriend button on each tile that unfriends the said user with the current user.

Pending Friend Request

On this page the user can see all his pending friend requests and can either accept the friend request at which point they start appearing on the friends page or decline it. Accepted friends can also comment on the pins that are on boards marked with friends

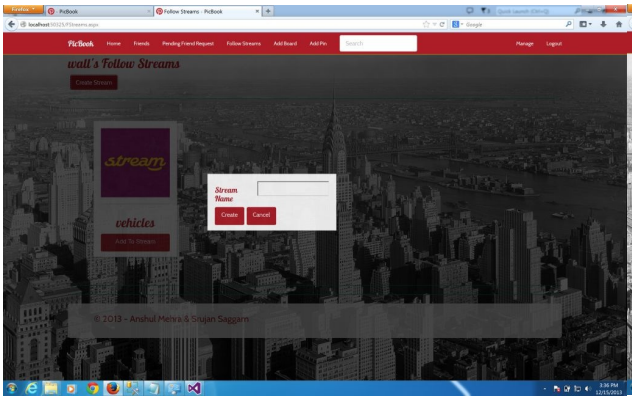


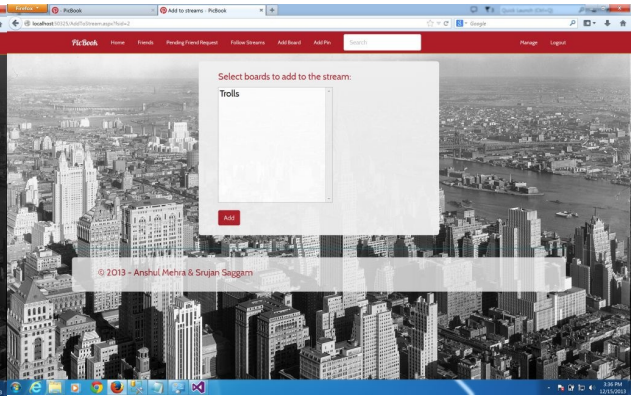
only comment policy.



Follow Streams

Another section of the nav bar is for follow streams. This page contains all the follows stream that a user has created. A user can click on Add Stream to create a new follow stream. Only boards that are being followed by the user can be added to a follow stream.





Once a follow stream is created user can click on it and see all the pins that are now a part of it. A user can delete a follow stream by clicking delete stream button on each Follow Stream page.

Pins / Repins

To pin an image to a board, goto add pin tab on the nav-bar. Once you're there, the section on the top will ask for the image URL or a file upload and some tags that go along with this new pin. Pins are identified by the board ID and picture ID that way the same picture cannot be pinned to the same board twice. While this is easy to enforce when users pin with image URLs, this is a little more difficult to enforce when users choose to upload a file.

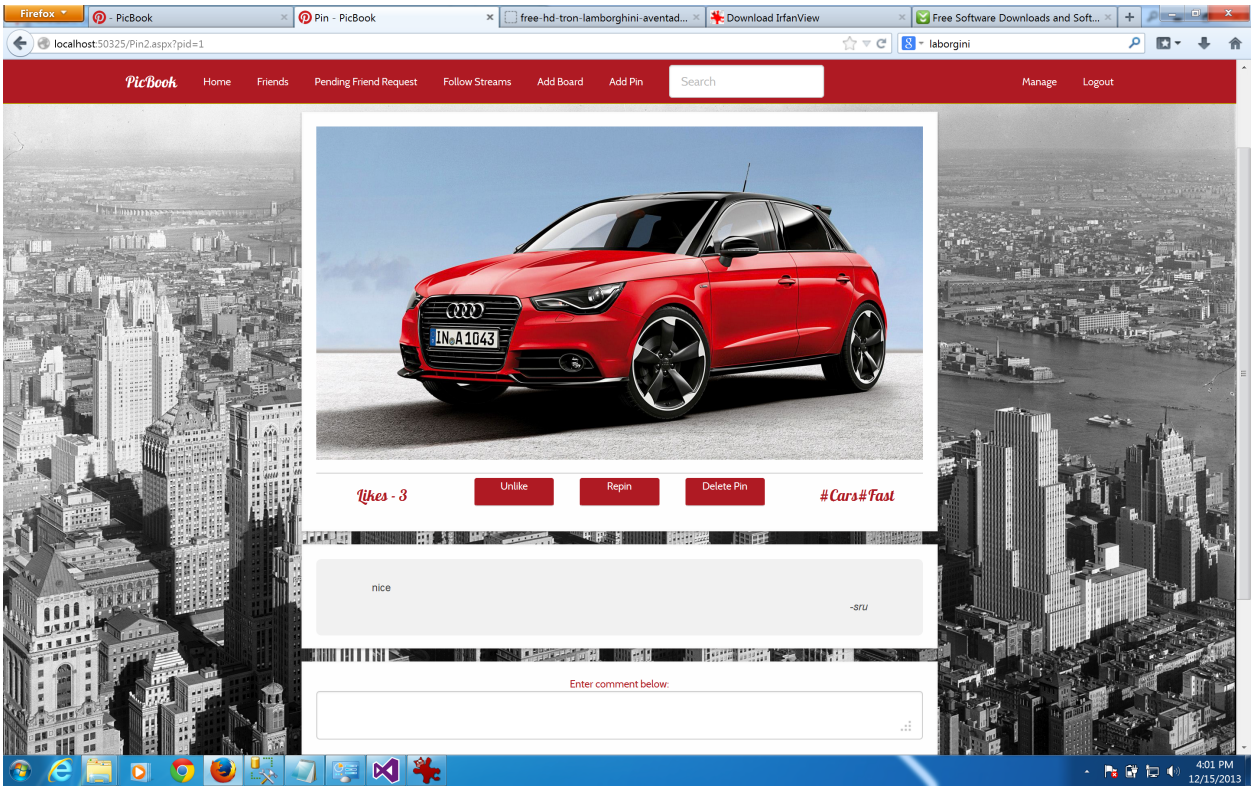
To repin an image from another board, a user can visit any pin page and click the repin button. After clicking the repin button, the user will be asked to select the board he/she wishes to put the repin in or create a new board for the repin. The repin will have a different picture ID from the original pin, but a similar original_pin ID.

Comments

To comment on a pin, the user simply visits a board page, clicks into a pin, then writes a comment. If the board owner created that board as a private board, the user would have to be friends with the owner in order to view or write comments on that pin. Comments are associated with individual pins, and not the original pin.

Likes

To like a pin, the user visits a board page, clicks into a pin, then clicks the like button. Likes are associated with the original pin, so whenever a user likes a pin, we have to first find the original pin ID via the Repin table. Once we find the original pin ID, we create a new entry in the Like table with the user's ID and the original pin's ID. There is a unique constraint on these two fields, so that the same user cannot like the same pin or any repins of it twice.



Tags

When a user creates a new pin, he/she has the option to assign tags to the pin. The tags are stored in the Tag table with a tag id and tag name. If the tag already exists it will be referred and if it does not exist a new tag will be created.

Once the pin is created, entries will be inserted into the PinTags table with the pin ID and tag ID of each tag the user assigned it. This table is crucial for the search by tag feature. Whenever a user searches for pins by tag names, we first find the tag IDs of those tag names, then query the PinTags table for all pins that are associated with those tag IDs.

Search Results

The nab-bar contains a unified search bar that can be used to search for users, boards and pins having matching tags.

Data Access Layer

The following methods were used to interact with the database:

Method	Purpose
IEnumerable<Pin> GetPins(string bid)	Gets all the pins pinned on a given board(bid)
IEnumerable<Board> GetBoards(string mid)	Gets all the boards owned by a given member(mid)
Pin GetPin(string pid)	Gets information about a given pin(pid)
IEnumerable<Comment> GetComments(string pid)	Gets all the comments on a given pin(pid)
Like GetLikes(string pid)	Gets the number of like on the original pin of a given pin(pid)
bool HasLiked(string mid, string pid)	Returns true if given member(mid) has liked a given pin(pid)
bool IsOwnerOfBoard(string mid, string bid)	Returns true if a given member(mid) is the owner of a given board(bid)
bool IsFollowingBoard(string mid, string bid)	Returns true if a given member(mid) is following a given board(bid)
bool ModifyLike(string mid, string pid, string flag)	Used to update like to unlike and unlike to like.
bool AddComment(string mid, string pid, string comment)	Adds comments in text box to a given pid
bool AddBoard(string mid, string bname, string descrip, string permission)	Creates a new board which is owned by member(mid), has name(bname), has description(descrip) and permission(permision)
bool AddToFollowStream(string fs_id, string bid)	Adds boards to a follow stream
bool PinTag(string pid, string tagname)	Adds a given tag name-pin combo to the PinTags table
AddPin(string mid,string bid,string pin_url,string source_url,string descrip, HttpServerUtility obj)	Creates a new pin which is owned by member(mid), is on board(bid) has url(pin_url) and description(descrip)
CreateUser(string username, string first_name, string middle_name, string last_name, string pswd, string email)	Creates a new user with all the information provided as parameters.
bool SendFriendRequest(string mid, string friend_mid, string status)	Sends friend request from member(mid) to member(friend_mid)
bool UpdateFriendRequest(string mid, string friend_mid, string status)	Updates the friend status.

bool RepinPicture(string bid, string pid, string descrip)	Repins pin(pid) to the given board(bid) and with new description(descrip)
bool CreateFollowStreamBoard(string mid, string fs_name)	Creates a new follow stream for the member(mid) with name(fs_name)
IEnumerable<Friend> GetFriends(string mid)	Returns list of friends for the given member(mid)
IEnumerable<Friend> GetPFriends(string mid)	Returns the list of pending friend requests for a given member(mid)
string GetUserName(string mid)	Returns the username for a given member(mid)
string GetBoardName(string bid)	Returns the board name for a given Board(bid)
List<string> GetHashtag(string pid)	Returns a list of hashtags associated with the given pin(pid)
Member GetUserProfile(string mid)	Returns info for a given member(mid)
bool UpdateProfile(string mid, string fname, string mname, string lname, string pswd)	Updates user profile with the given parameters
IEnumerable<Board> GetFollowingBoards(string mid)	Returns a list of boards being followed by the given member(mid)
bool FollowBoard(string mid, string bid , string commandname)	Returns true if a given member(mid) is following a given board(bid)
IEnumerable<FwStream> GetFollowStreams(string mid)	Returns a list of follow streams owned by a given member(mid)
IEnumerable<Pin> GetFSPins(string fsid)	Returns a list of Pins that belong to the given follow stream (fsid)
string GetFSBoardName(string fsid)	Returns name of the given followstream(fsid)
IEnumerable<Board> GetFBoards(string mid)	Returns the list of boards in follow streams of given member(mid)
IEnumerable<PicBook.Models.Friend> GetMemberSearchResult(string searchStr)	Searches members matching searchStr
IEnumerable<PicBook.Models.Pin> GetPinSearchResult(string searchStr)	Searches pins with tags matching searchStr
IEnumerable<PicBook.Models.Board> GetBoardSearchResult(string searchStr)	Searches boards with name matching searchStr
static bool DeleteBoard(string p, string bid)	Deletes a given board(bid)
static bool DeleteFriend(string mid, string friend_mid)	Deletes a given friendship(mid, friend_mid)

bool IsOwnerOfPin(string mid, string pid)	Returns true if a given member(mid) is owner of given pin(pid)
bool DeletePin(string p, string pid)	Deletes a given pin(pid)