IFT 530 – ADVANCED SQL PROGRAMMING  
DEVIL’S CALL- STUDENT PARTY E-VITE APPLICATION  
FINAL REPORT



Arizona State University

Department of Information Technology

Alok Sahu

Ramya Chandrashekar

Venkata Siva Abhishek Munukutla

Mohan Suraj Lakshmi Narayana

Aditya Kalyanaraman

Instructor: Dr Usha Jaganathan

**INTRODUCTION**:

Arizona State University is a comprehensive public research university, measured not by whom it excludes, but by whom it includes and how they succeed. ASU considers students social well-being as an important factor to their success. ASU has over 1500 student organizations and hosts a high number of events throughout the academic year to keep the students engaged and involved. This provides them with an opportunity to embrace diversity and network which is essential for students before they step into the real world. With ASU’s huge student strength, an application that can used across all organizations for invitations and guest list management would be valuable.

**OBJECTIVE**:

The objective of our project is to develop a comprehensive student event evite and management web application that can be adapted across all student organizations in ASU. We want to implement core powerful functionality with a resilient database along with a GUI that can be easily used by all individuals.

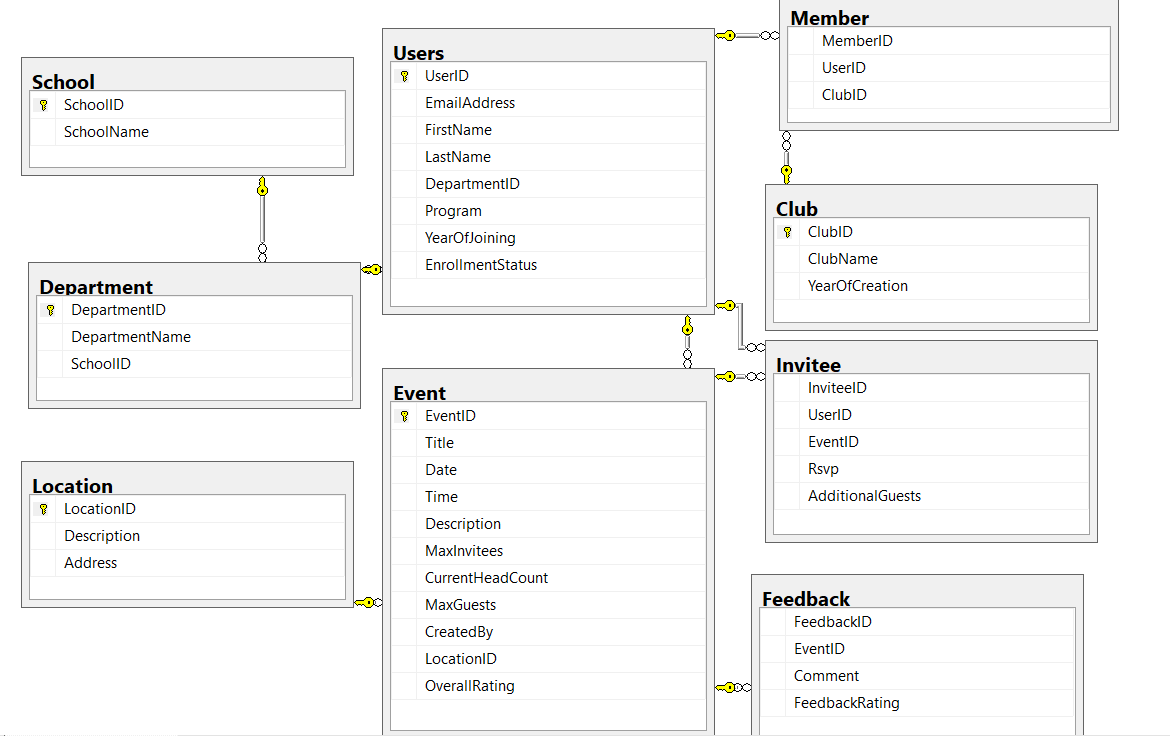
**TECHNOLOGY**:

SQL Server Management Studio, MS-SQL Server, Python Django, MySQL

**REQUIREMENTS:**

1. Login Page- Users will login using emailed and password.
2. Create E-Vite- Students can create new event with date, time, location, event description, RSVP option etc and send out invitations. This will be a form feature as part of our application.
3. RSVP option- Invitation RSVP will have option for invitees to include additional guests if they want to and if they are allowed to. The creator has control over this option and will help manage the event invitees.
4. Invitees group- Event organizer can send invitations to groups based on club (list of members) or department or school. If no option, invitations are sent to all the students.
5. Edit event details- Event organizer can modify events at a later point of time in case there is any change in the event and notify all the attendees.
6. Send Reminder- Event organizer can decide to send reminder of the event to the invitees. This will be helpful in case some guest forgets to RSVP for the event. The RSVP will close based on how the organizer sets it. (It can be one day before or one week before the party)
7. Invitee response- Attendee will be able to respond to the invitation through email. And the status is updated in the Invitee and event table.
8. When the invitees respond, the headcount data is updated in Event table.
9. Feedback option- After the party, there will be one feedback option based on which organizers can plan future events in a better way. This feedback option will be mailed to the attendees once the event is done with.
10. When the Feedback table is updated, the total rating of the event is updated.
11. List of invitees– This consists of a list of all people invited
12. Event details – This consists of all information about the event like date, venue, organizer and many more
13. List of events - This gives us the list of all planned events for the user.

**ERD DIAGRAM:**



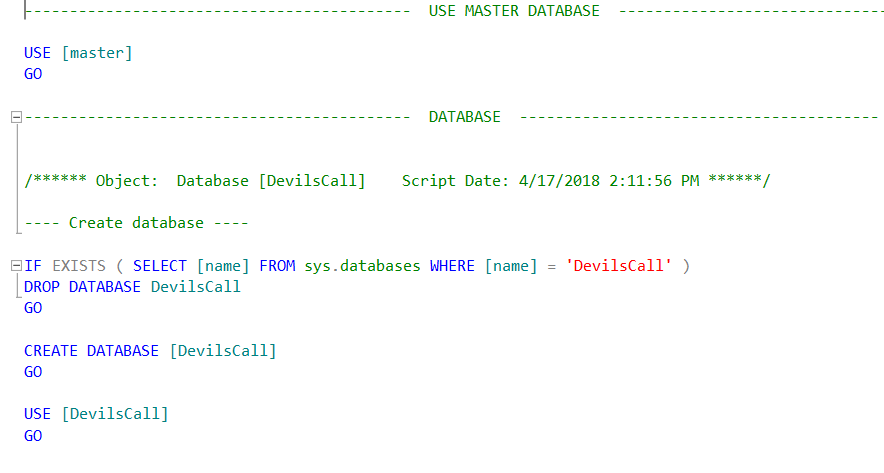
**Figure 1: Entity Relationship Diagram for the Database**

**IMPLEMENTATION:**

**DDL(Data Definition Language) :**DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

* Database Creation:

The first in our development process, after the requirements documentation phase is the creation of the database as shown below:



* Table Creation:

Next, we create tables, as mentioned in the ERD diagram, in our database. While creating the tables, we have taken care of the primary and foreign key constraints. Also, the null not null, identity and datatypes are set as required by our DB design. We start off with the base tables and then we create dependent tables.

1. Club Table- The Club table holds the details of the clubs that the students are a part of.

* ClubID – smallint
* ClubName – nvarchar
* YearOfCreation - date

---- Create Table for Clubs ----

IF OBJECT\_ID(N'DevilsCall.dbo.Club', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Club](

[ClubID] [smallint] IDENTITY(1,1) NOT NULL,

[ClubName] [nvarchar](50) NOT NULL,

[YearOfCreation] [date] NULL,

PRIMARY KEY CLUSTERED

(

[ClubID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY],

UNIQUE NONCLUSTERED

(

[ClubName] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. Department Table- The Department table holds details of the department that the students belong to.

* DepartmentID – smallint
* DepartmentName – nvarchar
* SchoolID - smallint

---- Create Table for Departments ----

IF OBJECT\_ID(N'DevilsCall.dbo.Department', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Department](

[DepartmentID] [smallint] IDENTITY(1,1) NOT NULL,

[DepartmentName] [nvarchar](50) NOT NULL,

[SchoolID] [smallint] NULL,

PRIMARY KEY CLUSTERED

(

[DepartmentID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY],

UNIQUE NONCLUSTERED

(

[DepartmentName] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. School Table- The School table holds details of the school the students belong to.

* SchoolID – smallint
* UserName - nvarchar

---- Create Table for School ----

IF OBJECT\_ID(N'DevilsCall.dbo.School', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[School](

[SchoolID] [smallint] IDENTITY(1,1) NOT NULL,

[SchoolName] [nvarchar](50) NOT NULL,

PRIMARY KEY CLUSTERED

(

[SchoolID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY],

UNIQUE NONCLUSTERED

(

[SchoolName] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. Location Table- Every location is preset in the application. Each location is available in the table Location. The Location table holds the details of the location where the event is held.

* LocationID – smallint
* Description – nvarchar
* Address – nvarchar

--- Create Table for Location ---

IF OBJECT\_ID('dbo.Location', 'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Location](

[LocationID] [smallint] NOT NULL,

[Description] [nvarchar](200) NULL,

[Address] [nvarchar](500) NOT NULL,

PRIMARY KEY CLUSTERED (

[LocationID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. User Table- The details of students along with their details are stored in this table. Any user has access to use the application. We will assume that the details are prefetched from the main university server. The invitees list is obtained from the User table.

* UserID – smallint
* EmailAddress – nvarchar
* FirstNAme – nvarchar
* LastName – nvarchar
* DepartmentID – smallint
* Program – nvarchar
* YearOfJoining – date
* EnrollmentStatus - navarchar

--- Create Table for Users ---

IF OBJECT\_ID(N'DevilsCall.dbo.Users', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Users](

[UserID] [smallint] IDENTITY(1,1) NOT NULL,

[EmailAddress] [nvarchar](50) NOT NULL,

[FirstName] [nvarchar](50) NOT NULL,

[LastName] [nvarchar](50) NULL,

[DepartmentID] [smallint] NULL,

[Program] [nvarchar](50) NULL,

[YearOfJoining] [date] NULL,

[EnrollmentStatus] [nvarchar](50) NULL,

PRIMARY KEY CLUSTERED

(

[UserID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY],

UNIQUE NONCLUSTERED

(

[EmailAddress] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. Member Table- The Member table is used to join the User table with the Clubs table. This table is used to overcome the many to many relationships. This table is the result of normalizing the relation between a user and a club.

If a user belongs to a club, the relation of the user as a member of the club is stored in this table.

* MemberID – smallint
* UserID – smallint
* ClubID - smallint

--- Create Table for Members ---

IF OBJECT\_ID(N'DevilsCall.dbo.Member', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Member](

[MemberID] [smallint] IDENTITY(1,1) NOT NULL,

[UserID] [smallint] NOT NULL,

[ClubID] [smallint] NOT NULL

) ON [PRIMARY]

END

GO

1. Event Table-The Event table holds details of the event created by the user

* EventID – smallint
* Title – nvarchar
* Date – date
* Time – time
* Description – varchar
* MaxInvitees – int
* CurrentHeadCount – int
* MaxGuests – int
* CreatedBy – smallint
* LocationID – smallint
* OverallRating - tinyint

---- Create Table for Events ----

IF OBJECT\_ID(N'DevilsCall.dbo.Event', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Event](

[EventID] [smallint] IDENTITY(1,1) NOT NULL,

[Title] [nvarchar](200) NOT NULL,

[Date] [date] NOT NULL,

[Time] [time](7) NULL,

[Description] [nvarchar](1000) NULL,

[MaxInvitees] [int] NULL,

[CurrentHeadCount] [int] NULL,

[MaxGuests] [int] NULL,

[CreatedBy] [smallint] NOT NULL,

[LocationID] [smallint] NULL,

[OverallRating] [tinyint] NULL,

PRIMARY KEY CLUSTERED

(

[EventID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

END

GO

1. Invitee table- The invitee table can tell us the people invited for a particular event. If a user is invited to an event, the user is considered an invitee. Details such as number of guests (for example, Plus One) and RSVP are stored in this table

* InviteeID – smallint
* UserID – smallint
* EventID – smallint
* Rsvp – nvarchar
* AdditionalGuests - int

--- Create Table for Invitee ---

IF OBJECT\_ID(N'DevilsCall.dbo.Invitee', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Invitee](

[InviteeID] [smallint] IDENTITY(1,1) NOT NULL,

[UserID] [smallint] NOT NULL,

[EventID] [smallint] NOT NULL,

[Rsvp] [nvarchar](10) NULL,

[AdditionalGuests] [int] NULL

) ON [PRIMARY]

END

GO

1. Feedback Table- The Feedback table is used to hold the feedback from the user for an event.

* FeedbackID – smallint
* EventID – smallint
* Comment – nvarchar
* FeedbackRating - tinyint

---- Create Table for Feedback from invitees ----

IF OBJECT\_ID(N'DevilsCall.dbo.Feedback', N'U') IS NULL

BEGIN

CREATE TABLE [dbo].[Feedback](

[FeedbackID] [smallint] IDENTITY(1,1) NOT NULL,

[EventID] [smallint] NOT NULL,

[Comment] [nvarchar](200) NULL,

[FeedbackRating] [tinyint] NOT NULL

) ON [PRIMARY]

END

GO

* Table Constraints:

--- Default value for RSVP is set to no ---

ALTER TABLE [dbo].[Invitee] ADD DEFAULT ('no') FOR [Rsvp]

GO

--- Default value for AdditionalGuests is set to 0 ---

ALTER TABLE [dbo].[Invitee] ADD DEFAULT ((0)) FOR [AdditionalGuests]

GO

--- SchoolID in Department table references SchoolID in School table ---

ALTER TABLE [dbo].[Department] WITH CHECK ADD CONSTRAINT [FK\_Department\_School\_SchoolID] FOREIGN KEY([SchoolID])

REFERENCES [dbo].[School] ([SchoolID])

GO

ALTER TABLE [dbo].[Department] CHECK CONSTRAINT [FK\_Department\_School\_SchoolID]

GO

--- LocationID in Event table references LocationID in Location table ---

ALTER TABLE [dbo].[Event] WITH CHECK ADD CONSTRAINT [FK\_Event\_Location\_LocationID] FOREIGN KEY([LocationID])

REFERENCES [dbo].[Location] ([LocationID])

GO

ALTER TABLE [dbo].[Event] CHECK CONSTRAINT [FK\_Event\_Location\_LocationID]

GO

--- CreatedBy in Event table references UserID in Users table ---

ALTER TABLE [dbo].[Event] WITH CHECK ADD CONSTRAINT [FK\_Event\_Users\_UserID] FOREIGN KEY([CreatedBy])

REFERENCES [dbo].[Users] ([UserID])

GO

ALTER TABLE [dbo].[Event] CHECK CONSTRAINT [FK\_Event\_Users\_UserID]

GO

--- EventID in Feedback table references EventID in Event table ---

ALTER TABLE [dbo].[Feedback] WITH CHECK ADD CONSTRAINT [FK\_Feedback\_Event\_EventID] FOREIGN KEY([EventID])

REFERENCES [dbo].[Event] ([EventID])

GO

ALTER TABLE [dbo].[Feedback] CHECK CONSTRAINT [FK\_Feedback\_Event\_EventID]

GO

--- EventID in Invitee table references EventID in Event table ---

ALTER TABLE [dbo].[Invitee] WITH CHECK ADD CONSTRAINT [FK\_Invitee\_Event\_EventID] FOREIGN KEY([EventID])

REFERENCES [dbo].[Event] ([EventID])

GO

ALTER TABLE [dbo].[Invitee] CHECK CONSTRAINT [FK\_Invitee\_Event\_EventID]

GO

--- UserID in Invitee table references UserID in Users table ---

ALTER TABLE [dbo].[Invitee] WITH CHECK ADD CONSTRAINT [FK\_Invitee\_Users\_UserID] FOREIGN KEY([UserID])

REFERENCES [dbo].[Users] ([UserID])

GO

ALTER TABLE [dbo].[Invitee] CHECK CONSTRAINT [FK\_Invitee\_Users\_UserID]

GO

--- ClubID in Member table references ClubID in Club table ---

ALTER TABLE [dbo].[Member] WITH CHECK ADD CONSTRAINT [FK\_member\_club\_clubID] FOREIGN KEY([ClubID])

REFERENCES [dbo].[Club] ([ClubID])

GO

ALTER TABLE [dbo].[Member] CHECK CONSTRAINT [FK\_member\_club\_clubID]

GO

--- UserID in Member table references UserID in Users table ---

ALTER TABLE [dbo].[Member] WITH CHECK ADD CONSTRAINT [FK\_member\_Users\_UserID] FOREIGN KEY([UserID])

REFERENCES [dbo].[Users] ([UserID])

GO

ALTER TABLE [dbo].[Member] CHECK CONSTRAINT [FK\_member\_Users\_UserID]

GO

--- DepartmentID in Users table references DepartmentID in Department table ---

ALTER TABLE [dbo].[Users] WITH CHECK ADD CONSTRAINT [FK\_Users\_Department\_DepartmentID] FOREIGN KEY([DepartmentID])

REFERENCES [dbo].[Department] ([DepartmentID])

GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [FK\_Users\_Department\_DepartmentID]

GO

--- Restrict Rsvp in Invitee table to no, yes and maybe values ---

ALTER TABLE [dbo].[Invitee] WITH CHECK ADD CHECK (([Rsvp]='no' OR [Rsvp]='maybe' OR [Rsvp]='yes'))

GO

--- Restrict enrollmentstatus in Users table to alumni, online, parttime, or fulltime values ---

ALTER TABLE [dbo].[Users] WITH CHECK ADD CONSTRAINT [ck\_enrollment] CHECK (([enrollmentstatus]='alumni' OR [enrollmentstatus]='online' OR [enrollmentstatus]='partime' OR [enrollmentstatus]='fulltime'))

GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [ck\_enrollment]

GO

--- Restrict program in Users table to postdoc, phd, graduate and undergraduate values ---

ALTER TABLE [dbo].[Users] WITH CHECK ADD CONSTRAINT [ck\_program] CHECK (([program]='postdoc' OR [program]='phd' OR [program]='graduate' OR [program]='undergrad'))

GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [ck\_program]

GO

**DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

We use select, Update and insert into statements as part of stored procedures. Below are the different Stored Procedures in the Devil’s Call Database.

* Create Stored Procedure:

1. sp\_createEvent- Used to create an event with the following details. Once the event is created, the guest type is checked namely club, department and school. Based on the guest type, the invitee details are displayed.

--- Create Stored Procedure sp\_createEvent ---

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_createEvent', N'U') IS NULL

BEGIN

CREATE PROC [dbo].[sp\_createEvent]

@title nvarchar(200)='Event\_',

@date date=null,

@time time,

@description nvarchar(1000),

@maxInvitees int,

@currentHeadCount int,

@maxGuests int,

@createdBy smallint,

@locationID smallint,

@overAllRating tinyint,

@guestType varchar(50)=null,

@guestTypeGroup varchar(50)=null

as

SET @date = ISNULL(@date, convert(date,GETDATE()))

SET @currentHeadCount = 0;

SET @overAllRating = 0;

if @title = 'Event\_'

begin

set @title=@title + convert(varchar,current\_timestamp)

end

--SET @time = ISNULL(@time, convert(time,CURRENT\_TIMESTAMP))

BEGIN TRANSACTION;

BEGIN TRY

insert into dbo.Event

values(@title,@date,@time,@description,@maxInvitees,@currentHeadCount,@maxGuests,@createdBy,@locationID,@overAllRating)

declare @eventid smallint

select @eventid=max(EventID) from dbo.Event

where CreatedBy =@createdby

if lower(@guestType)='department'

begin

insert into Invitee(UserID,EventID)

select u.UserID,@eventid from Users u

join Department d on u.DepartmentID=d.DepartmentID

where d.DepartmentName=@guestTypeGroup

end

else if lower(@guestType)='school'

begin

insert into Invitee(UserID,EventID)

select u.UserID,@eventid from Users u

join Department d on u.DepartmentID=d.DepartmentID

join School s on s.SchoolID=d.SchoolID

where s.SchoolName=@guestTypeGroup

end

else if lower(@guestType)='club'

begin

insert into Invitee(UserID,EventID)

select u.UserID,@eventid from Users u

join member m on u.userid=m.userid

join club c on c.clubid=m.clubid

where c.clubname=@guestTypeGroup

end

else if @guestType is null

begin

insert into Invitee(UserID,EventID)

select u.UserID,@eventid from Users u

end

select u.UserID,u.FirstName,u.LastName,u.EmailAddress

from users u join invitee i

on u.UserID=i.UserID

where i.EventID=@eventid

END TRY ----whole transaction

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

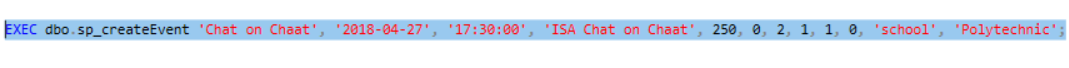
IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

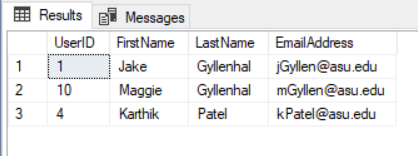
END

GO

OUTPUT:



Once data is inserted using the stored procedure, a table with invitees list is returned.



1. sp\_ createFeedback - Used to accept feedback once an event is done. Feedback rating between 1 and 5 is expected

to be given by the guest and the ratings are updated continuously in the Event table using triggers.

--- Create Stored Procedure sp\_createFeedback ---

IF OBJECT\_ID(N'DevilsCall.dbo.createFeedback', N'U') IS NULL

BEGIN

create proc [dbo].[sp\_createFeedback]

@eventID smallint,

@comment nvarchar(200),

@feedBackRating tinyint

as

BEGIN TRANSACTION;

BEGIN TRY

insert into Feedback

values(@eventID,@comment,@feedBackRating)

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

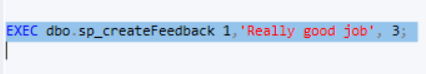
IF @@TRANCOUNT > 0

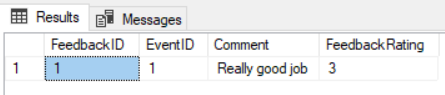
COMMIT TRANSACTION;

END

GO

OUTPUT:





1. sp\_inviteeRsvp - This stored procedure is used to update the invitee list based on the RSVP of the invitee. Using the UserID, EventID, RSVP and the number of additional guests as specified by the invitee in the invitation email, the stored procedure is executed as soon as the invitee submits the form via email. The corresponding changes are made in the Events table to keep a tab of the RSVP’s and total guest count. If an invitee specifies an additional guest number that exceeds the maximum number allowed, proper error message indicating that“Maximum guests limit exceeded” is displayed.

--- Create Stored Procedure sp\_inviteeRsvp ---

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_inviteeRsvp', N'U') IS NULL

BEGIN

CREATE proc [dbo].[sp\_inviteeRsvp]

@userID smallint,

@eventID smallint,

@rsvp nvarchar(10),

@additionalguests int

as

BEGIN

DECLARE @count int;

SELECT @count = MaxGuests FROM Event WHERE EventID = @eventID

IF @count >= @additionalguests ---- condition to check if the user has exceeded the allowable guest count

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

update invitee

set rsvp=@rsvp,AdditionalGuests=@additionalguests

where UserID=@userID and EventID=@eventID

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

END

ELSE

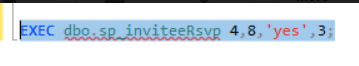
PRINT 'Max Guests exceeded. Please check!'

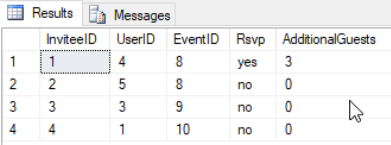
END

END

GO

OUTPUT:





1. sp\_ inviteeUpdateGuest - This stored procedure is used to update the invitee list based on the change in number of additional guests of the invitee. If an invitee specifies an additional guest number that exceeds the maximum number allowed, proper error message indicating that “Maximum guests limit exceeded” is displayed.

--- Create Stored Procedure sp\_inviteeUpdateGuest ---

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_inviteeUpdateGuest', N'U') IS NULL

BEGIN

CREATE proc [dbo].[sp\_inviteeUpdateGuest]

@userID smallint,

@eventID smallint,

@additionalguests int

as

BEGIN

DECLARE @count int;

SELECT @count = MaxGuests FROM Event WHERE EventID = @eventID

IF @count >= @additionalguests ---- condition to check if the user has exceeded the allowable guest count

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

update invitee

set AdditionalGuests=@additionalguests

where UserID=@userID and EventID=@eventID

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

END

ELSE

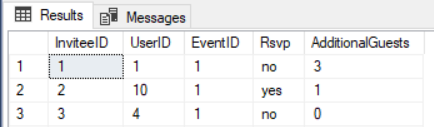
PRINT 'Max Guests exceeded. Please check!'

END

END

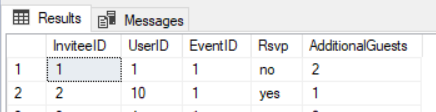
GO

OUTPUT:



Changing number of guests in row with InviteeID = 1





1. sp\_ listOfInvitees - This stored procedure is used to display all the invitees for a particular event.

--- Create Stored Procedure sp\_listOfInvitees ---

/\*

This stored procedure is used to display all the invitees for a particular event.

\*/

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_listOfInvitees', N'U') IS NULL

BEGIN

CREATE proc [dbo].[sp\_listOfInvitees]

@eventid smallint

as

select UserID,EmailAddress, FirstName,LastName, DepartmentName,SchoolName,Rsvp,AdditionalGuests

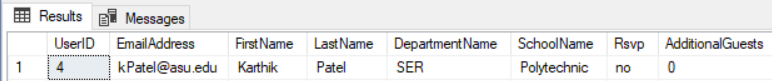
from v\_listOfInvitees where EventID =@eventid

END

GO

OUTPUT:





1. sp\_listOfEvents: This stored procedure is used to display list of all events.

--- Create Stored Procedure sp\_listOfEvents ---

/\*

This stored procedure is used to display list of all events.

\*/

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_listOfEvents', N'U') IS NULL

BEGIN

CREATE proc [dbo].[sp\_listOfEvents]

@Userid smallint

as

select EventID,Title, e.date,e.time, e.description,MaxInvitees,CurrentHeadCount,MaxGuests,l.Description,OverallRating

from Event e join users u on e.CreatedBy=u.UserID

join Location l on e.LocationID=l.LocationID where

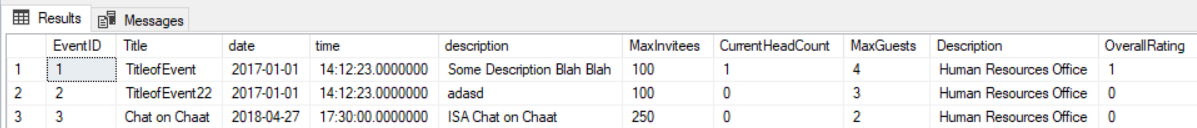
UserID = @Userid

END

GO

OUTPUT:





1. sp\_ EditEvent- This stored procedure is used to edit the event. This is done by updating the Event Table.

--- Create Stored Procedure sp\_ EditEvent ---

alter proc [dbo].sp\_EditEvent

@Eventid smallint,

@Title nvarchar(200)=null,

@Date date=null,

@Time time(7)=null,

@Description nvarchar(1000)=null,

@MaxInvitees int=null,

@LocationID smallint=null

as

if @Title is not null

begin

Update event

set Title=@Title

where EventID=@Eventid

end

if @Date is not null

begin

Update event

set Date=@Date

where EventID=@Eventid

end

if @Time is not null

begin

Update event

set Time=@Time

where EventID=@Eventid

end

if @Description is not null

begin

Update event

set Description=@Description

where EventID=@Eventid

end

if @MaxInvitees is not null

begin

Update event

set MaxInvitees=@MaxInvitees

where EventID=@Eventid

end

if @LocationID is not null

begin

Update event

set LocationID=@LocationID

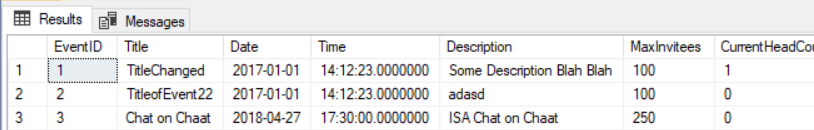
where EventID=@Eventid

end

GO

OUTPUT:





1. sp\_ InviteesReminder- This stored procedure is used to select the Invitees who did not respond to the Invitation. They have not RSVP. This can be used to get their email Id to send the Invitees an email to respond.

--- Create Stored Procedure sp\_ InviteesReminder---

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_InviteesReminder', N'U') IS NULL

BEGIN

CREATE proc [dbo].sp\_InviteesReminder

@eventid smallint

as

select UserID,EmailAddress, FirstName,LastName, DepartmentName,SchoolName,Rsvp,AdditionalGuests

from v\_listOfInvitees where EventID =@eventid and RSVP is null

END

GO

1. sp\_ EventInformation- This stored procedure is used to display all the Information for a particular event.

--- Create Stored Procedure sp\_ EventInformation'---

IF OBJECT\_ID(N'DevilsCall.dbo.sp\_EventInformation', N'U') IS NULL

BEGIN

CREATE proc [dbo].sp\_EventInformation

@Eventid smallint

as

select \*

from Event where

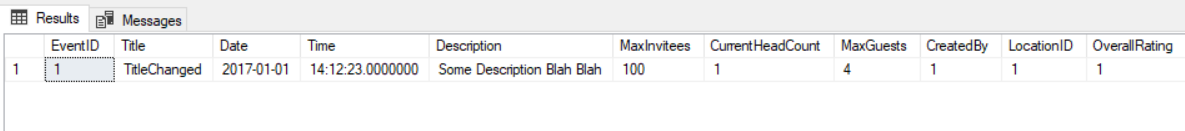
EventID = @Eventid

END

GO

OUTPUT:





* Create Triggers

1. tr\_invitee\_currenCountUpdateRsvp- This trigger is invoked when a record is updated in Invitee table. When the Rsvp of an invitee is changed, the trigger is invoked to update

the current head count for the event the invitee is going to attend. If the rsvp changes to yes or maybe from a no, then the number of guests and the invitee is added to the current head count for the event. If the rsvp changes to no, the number of guests and invitee is removed from the current head count for the event.

--- Create Trigger tr\_invitee\_currenCountUpdateRsvp ---

IF OBJECT\_ID(N'DevilsCall.dbo.tr\_invitee\_currenCountUpdateRsvp', N'U') IS NULL

BEGIN

CREATE TRIGGER dbo.tr\_invitee\_currenCountUpdateRsvp

ON dbo.Invitee FOR UPDATE

AS

BEGIN

DECLARE @eventid int, @currentcount int,@count int, @rsvpn nvarchar(10), @rsvpo nvarchar(10);

SELECT @eventid=EventID, @rsvpn=Rsvp FROM inserted;

SELECT @count = AdditionalGuests, @rsvpo = Rsvp FROM deleted;

SET NOCOUNT ON

IF UPDATE(Rsvp)

BEGIN

IF @rsvpn <> @rsvpo

BEGIN

IF (@rsvpn = 'yes' or @rsvpn = 'maybe') and @rsvpo = 'no'

BEGIN

SELECT @currentcount = CurrentHeadCount FROM Event WHERE EventID = @eventid

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Event SET CurrentHeadCount = (@currentcount + @count + 1) WHERE EventID=@eventid

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

END

IF (@rsvpn = 'no') and (@rsvpo = 'yes' or @rsvpo = 'maybe')

BEGIN

SELECT @currentcount = CurrentHeadCount FROM Event WHERE EventID = @eventid

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Event SET CurrentHeadCount = (@currentcount - @count - 1) WHERE EventID=@eventid

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

END

END

END

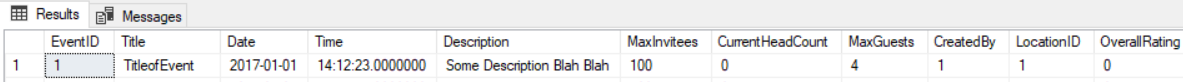
END

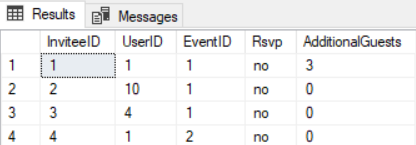
END

OUTPUT:

When changed to no:

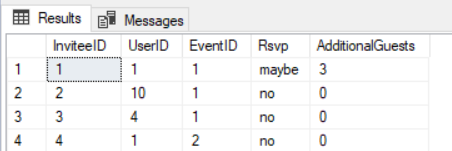


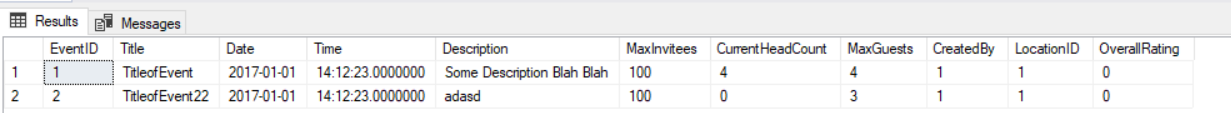




When changed to yes or maybe:







2. tr\_events\_currentCountUpdateGuests - This trigger is invoked when there is an update on Invitee table.When the additional guest count is changed for an invitee, the current head count in the events table is updated.

---- Create Trigger [dbo].[tr\_events\_currentCountUpdateGuests] ----

IF OBJECT\_ID(N'DevilsCall.dbo.tr\_events\_currentCountUpdateGuests', N'U') IS NULL

BEGIN

CREATE TRIGGER dbo.tr\_events\_currentCountUpdateGuests

ON dbo.Invitee FOR UPDATE

AS

BEGIN

DECLARE @eventid int, @currentcount int, @old int, @new int, @rsvp nvarchar(10);

SELECT @new = AdditionalGuests, @eventid=EventID, @rsvp=Rsvp FROM inserted;

SELECT @old = AdditionalGuests FROM deleted;

SET NOCOUNT ON

IF UPDATE(AdditionalGuests)

BEGIN

IF @old <> @new

BEGIN

SELECT @currentcount = CurrentHeadCount FROM Event WHERE EventID = @eventid

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Event SET CurrentHeadCount = @currentcount + @new - @old WHERE EventID=@eventid

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

END

END

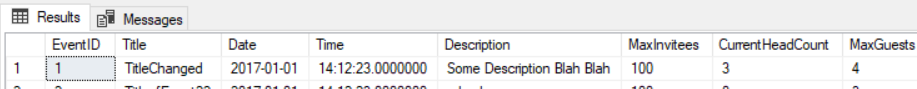
END

END

OUTPUT:



Number of guests changed to 1 from 3 for user 10.



CurrentHeadCount changes to 1 from 3.

3. tr\_events\_feedbackrating - This trigger is invoked when there is an insertion in Feedback table.

The overall rating for the corresponding event is updated in Events table.

IF OBJECT\_ID(N'DevilsCall.dbo.tr\_events\_feedbackrating', N'U') IS NULL

BEGIN

CREATE TRIGGER dbo.tr\_events\_feedbackrating

ON dbo.Feedback FOR INSERT

AS

BEGIN

DECLARE @rating tinyint, @eventid int, @overall tinyint, @count int;

SELECT @rating = FeedbackRating, @eventid = EventID from inserted;

SELECT @overall = OverallRating FROM Event WHERE EventID =@eventid;

SELECT @count = COUNT(FeedbackRating) FROM Feedback WHERE EventID =@eventid;

BEGIN TRANSACTION

BEGIN TRY

UPDATE Event SET OverallRating = ((@overall \* @count) + @rating)/(@count + 1) WHERE EventID = @eventid

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

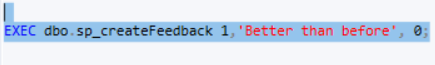
IF @@TRANCOUNT > 0

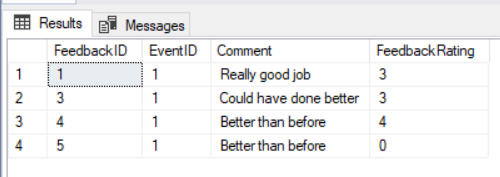
COMMIT TRANSACTION;

END

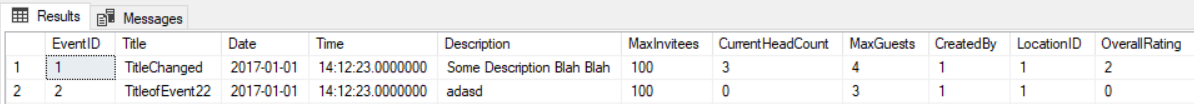
END

OUTPUT:





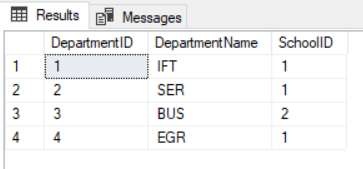
OVERALL RATING FOR EVENT 1 IS UPDATED.



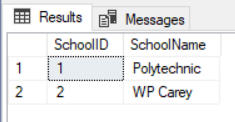
* Data Insertion:

Data in tables Department, School, Member, Club, Location, and Users are to be pre-fetched from the university’s main server. However, to demonstrate the utility of the server, we inserted dummy data into the tables.

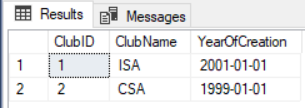
* Department table data:



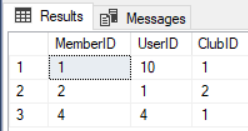
* School table data:



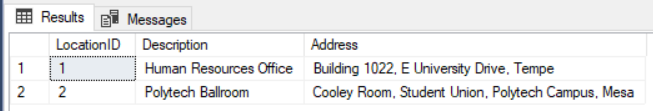
* Club table data:



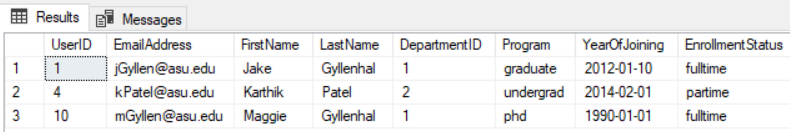
* Member table data:



* Location table data:

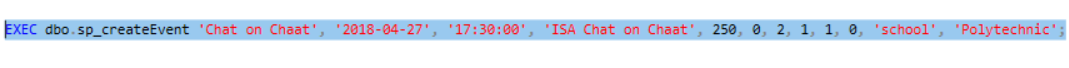


* Users table data:

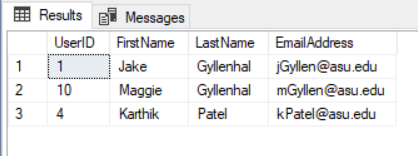


To create events and feedback, stored procedures are provided. The stored procedures provide logic for insertion.

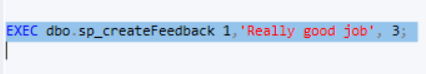
* sp\_createEvent:

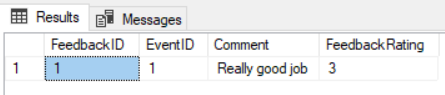


Once data is inserted using the stored procedure, a table with invitees list is returned.



* sp\_createFeedback:





* Views:
  + v\_listOfInvitees: This view is created to gather a result-set from six different tables to return list of invitees for an event.

---- Create View for list of invitees ----

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

IF OBJECT\_ID(N'DevilsCall.dbo.v\_listOfInvitees', N'U') IS NULL

BEGIN

CREATE view dbo.v\_listOfInvitees

as

select u.UserID,EmailAddress, FirstName,LastName, DepartmentName,SchoolName ,e.EventID,Title, Rsvp,AdditionalGuests

from Users u join Department d

on u.DepartmentID =d.DepartmentID

join School s on s.SchoolID=d.SchoolID

join Member m on u.UserID=m.MemberID

join Event e on e.EventID=m.MemberID

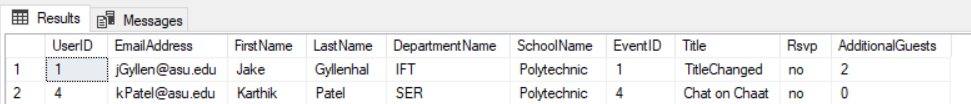
join Invitee i on i.UserID=u.UserID

END

GO

OUTPUT:

Whenever an event is created, this view is used to return a list of invitees for that event.



**DCL(Data Control Language) :**DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

------------------------------- USER ROLES and LOGIN ----------------------------

USE master;

CREATE LOGIN DC\_admin WITH PASSWORD = 'password',

DEFAULT\_DATABASE = [DevilsCall]

GO

USE DevilsCall;

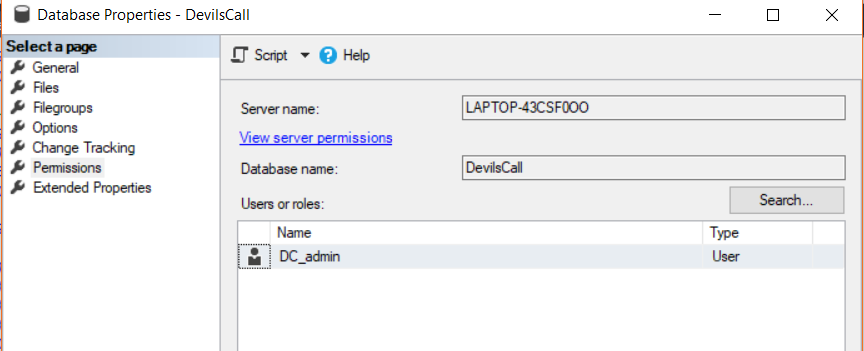
CREATE USER DC\_admin FOR LOGIN DC\_admin;

EXEC sp\_addrolemember 'db\_owner', 'DC\_admin';

EXEC sp\_addrolemember 'db\_datareader', 'DC\_admin'

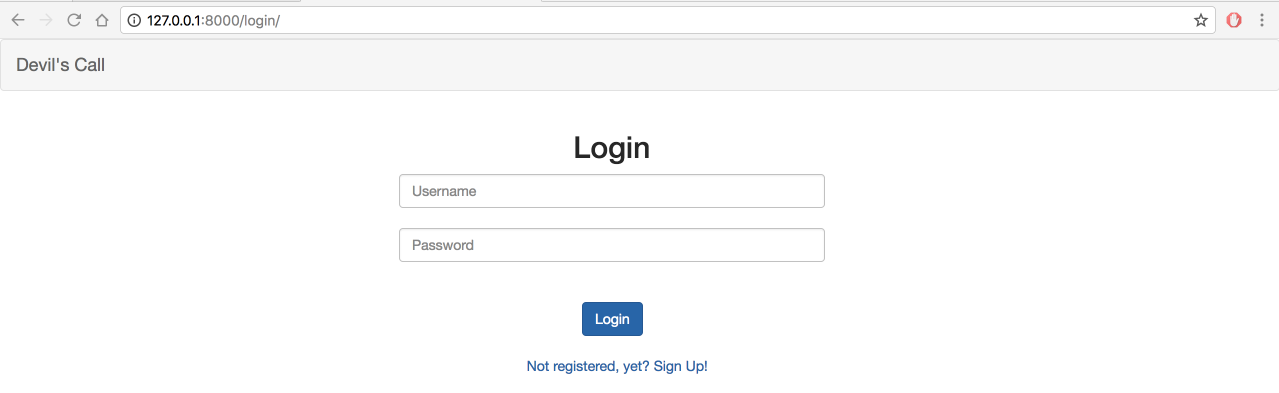
EXEC sp\_addrolemember 'db\_datawriter', 'DC\_admin'

GO

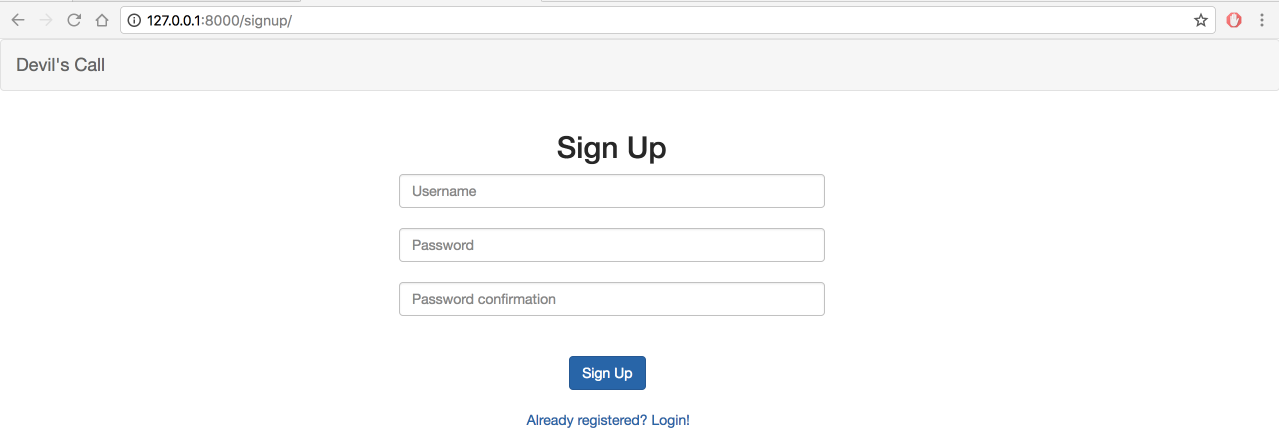


# USER INTERFACE:

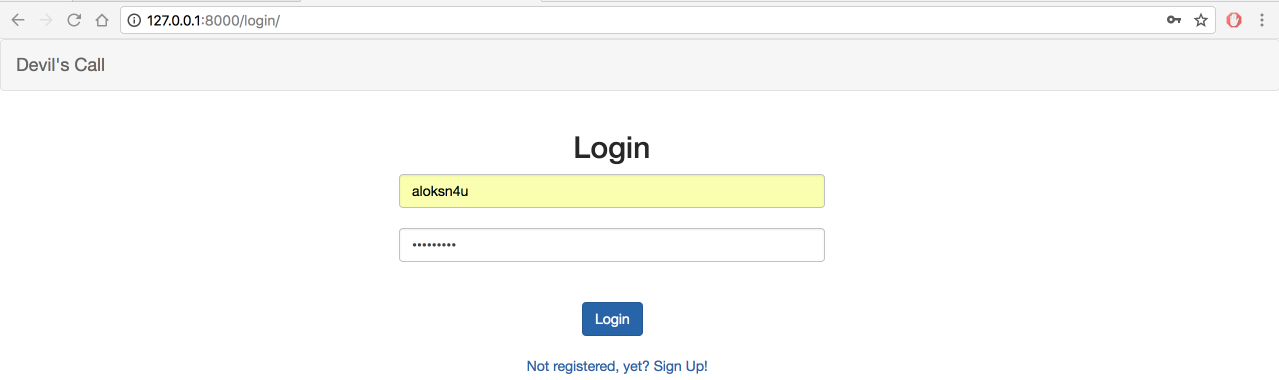
Login Page:



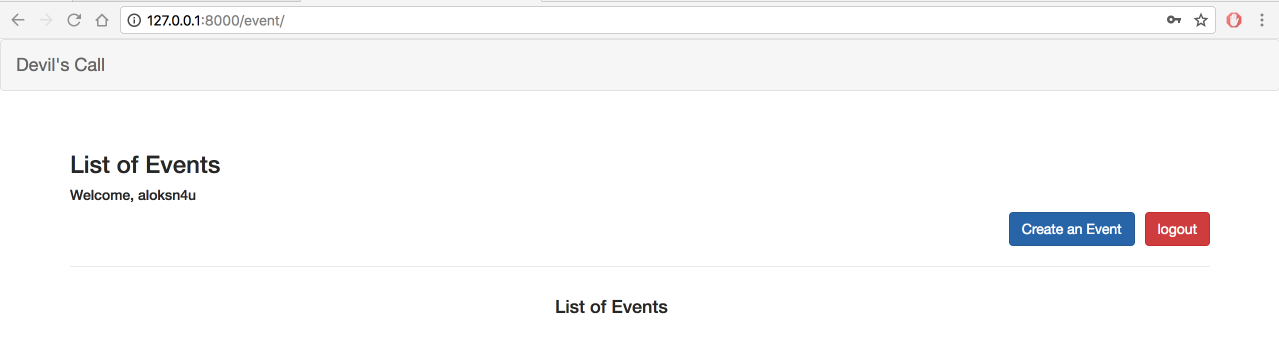
Signup Page:



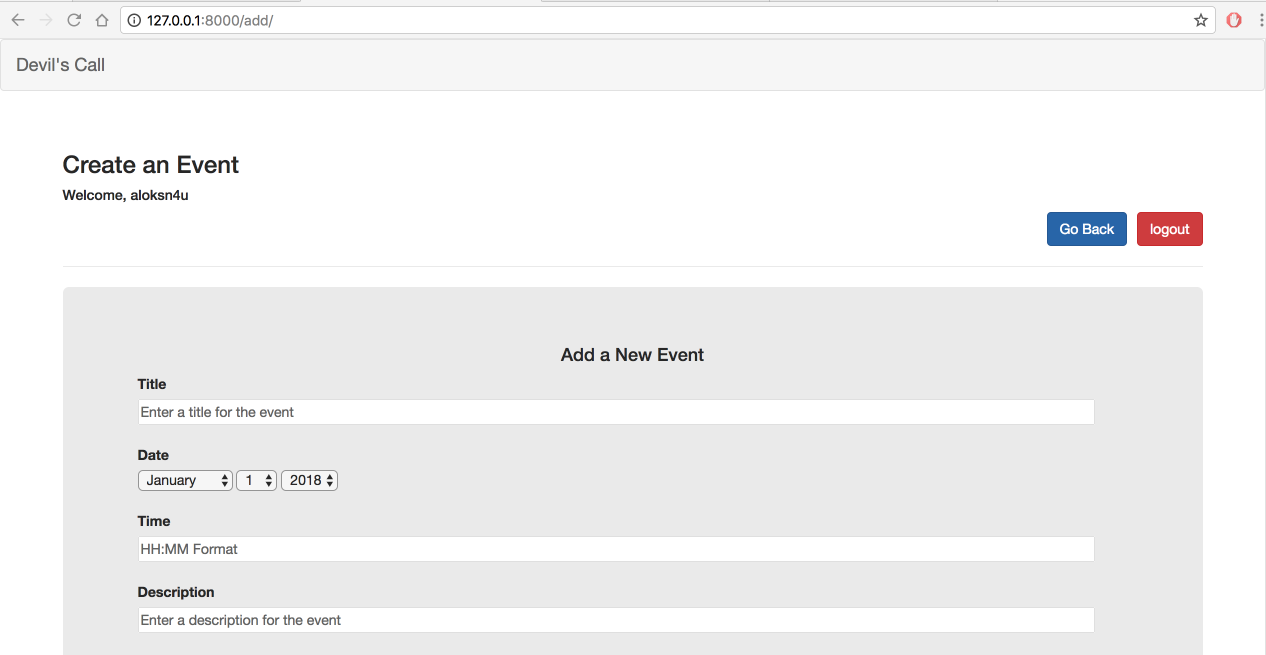
Login:



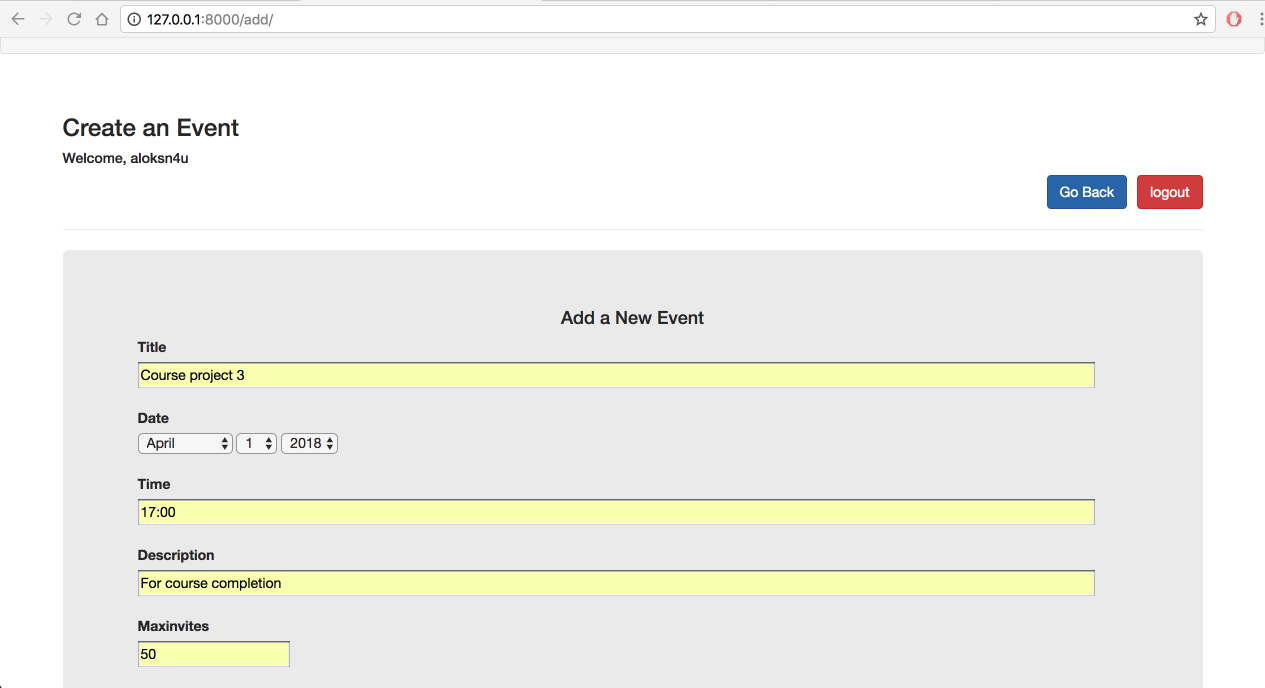
As the user has not created any events, event list is empty.

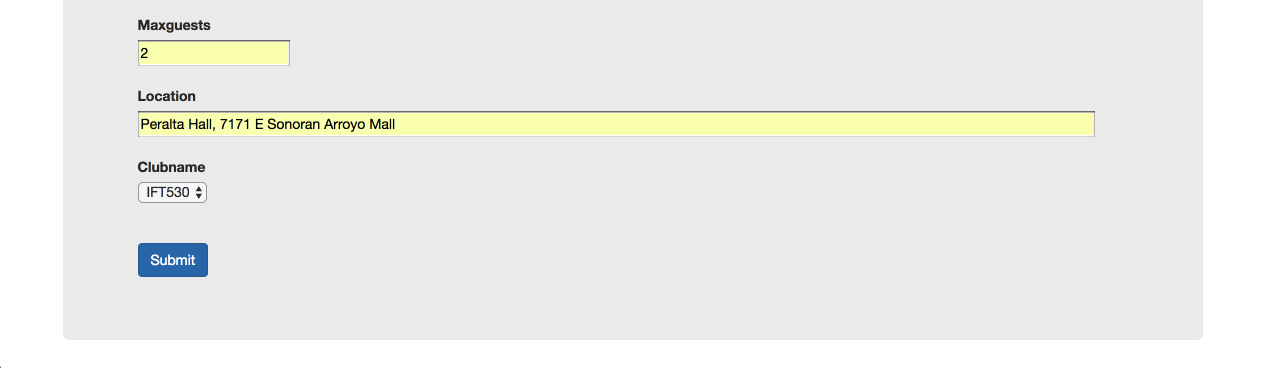


From for creating a new Event:

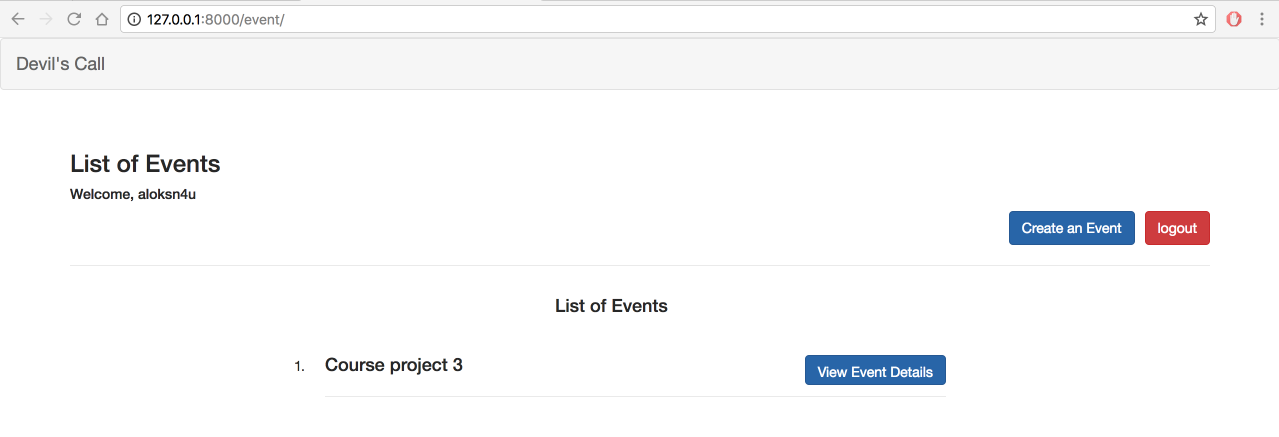


After filling out the event details:

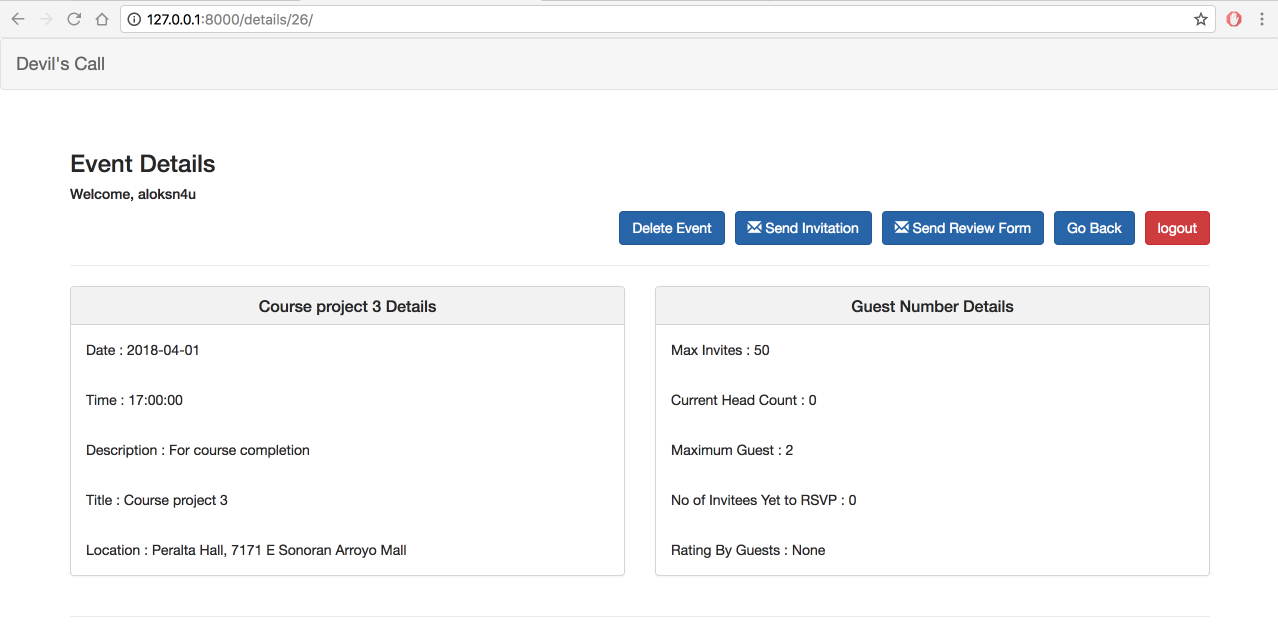




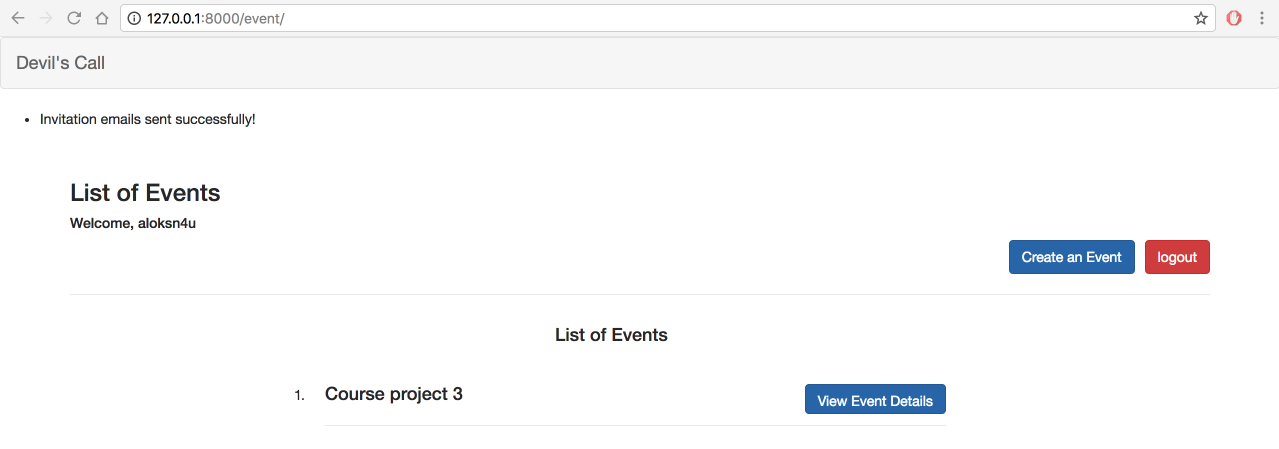
After clicking submit button:



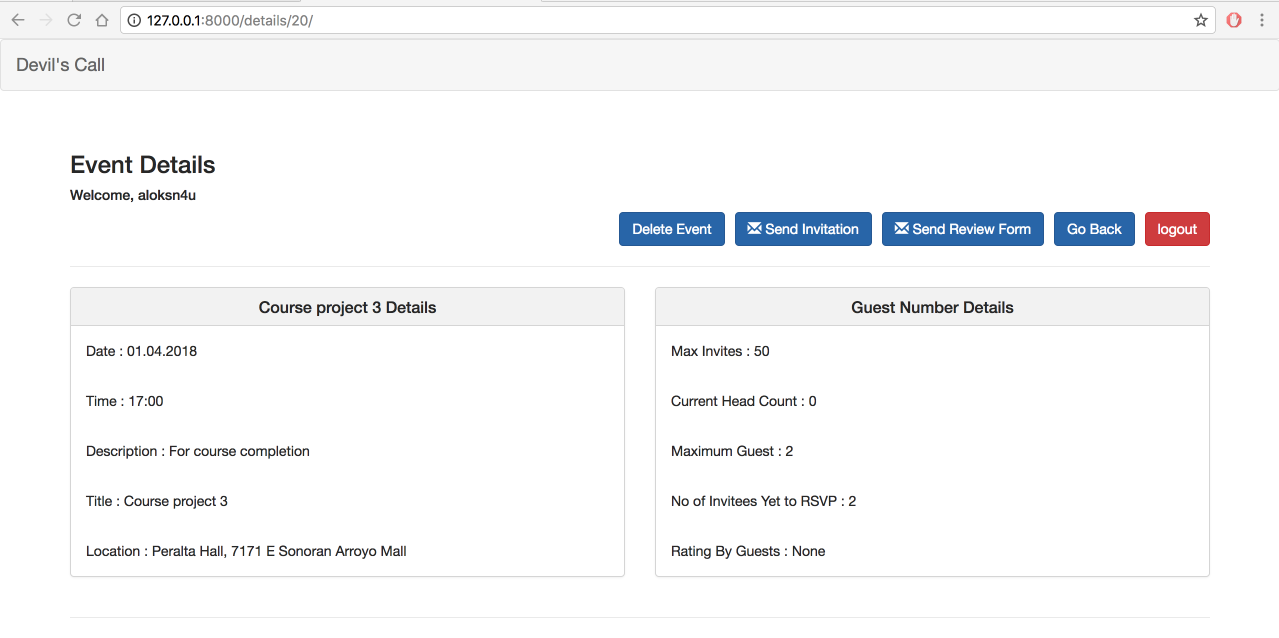
After clicking on view event details button:



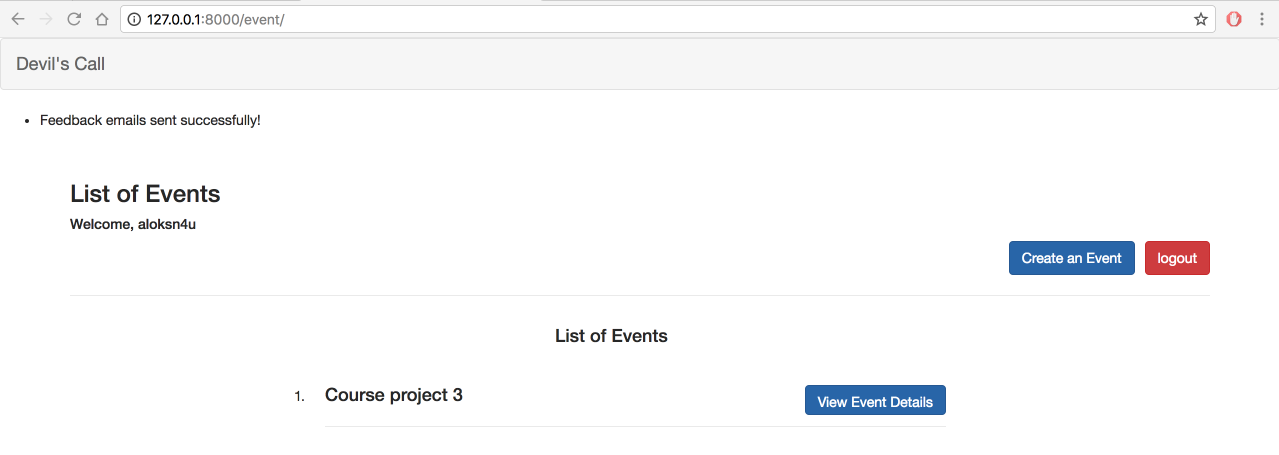
After clicking on send Invitation button:



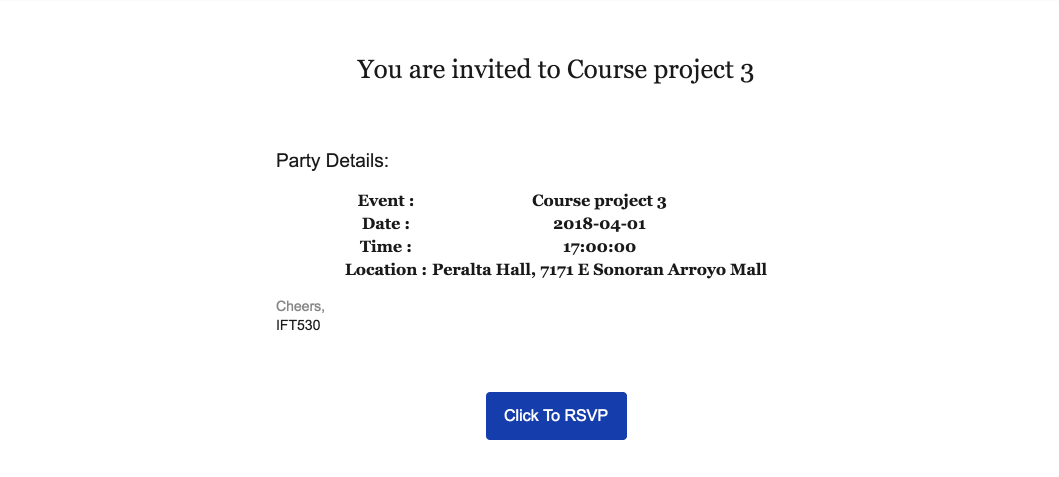
After ending invites when clicking view details:



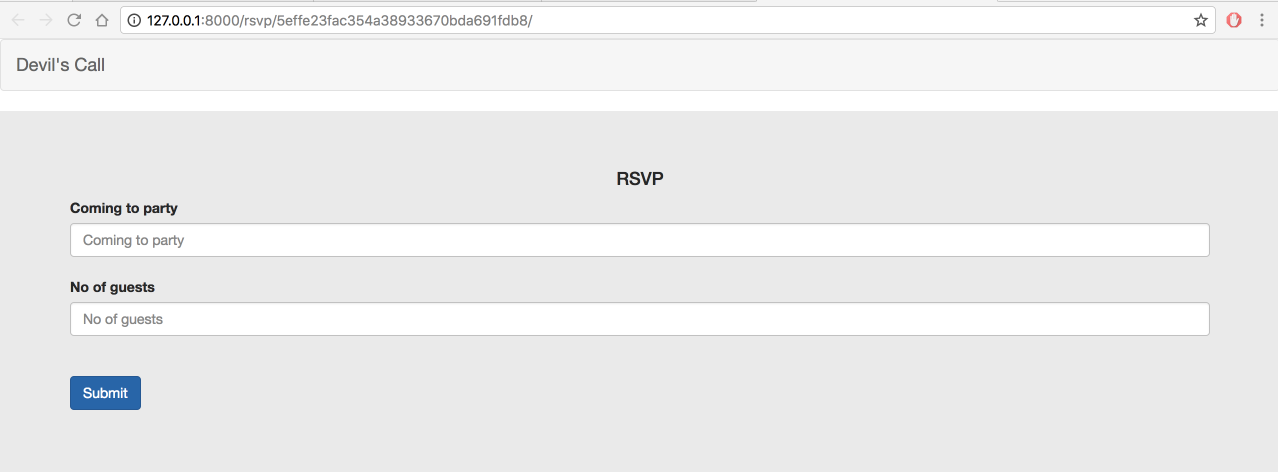
After clicking on send review form button:



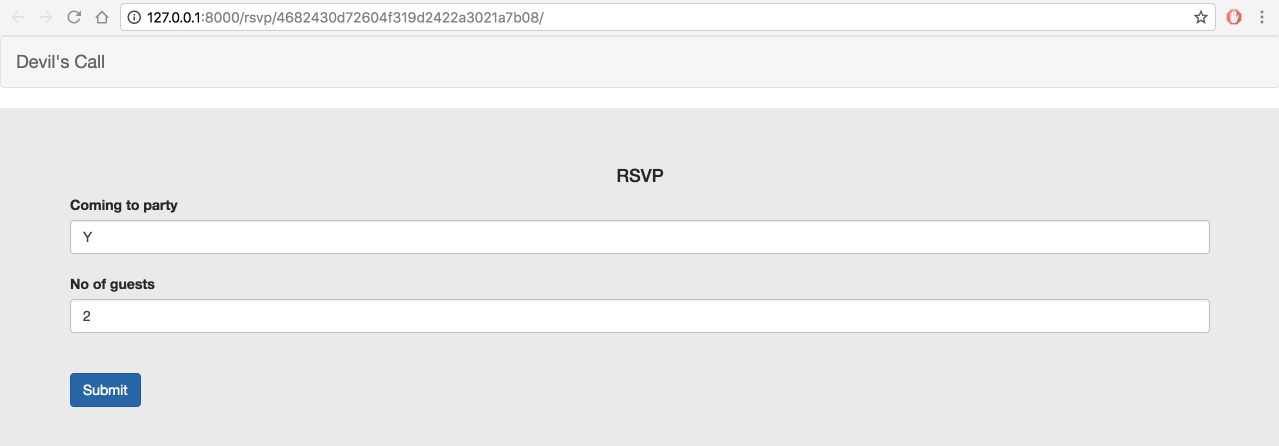
Invitation email:



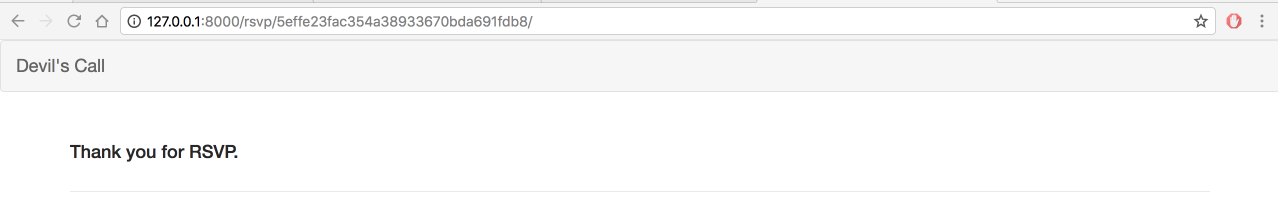
When you click on RSVP button:



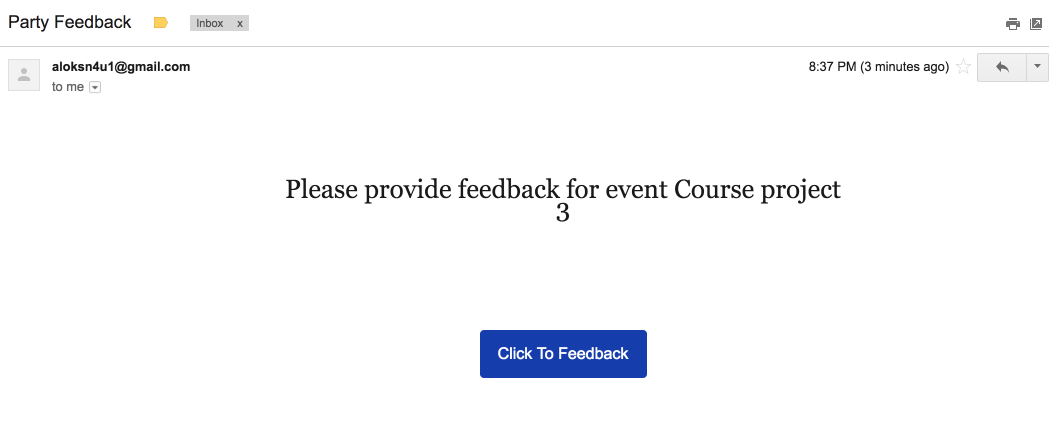
After Entering RSVP details:



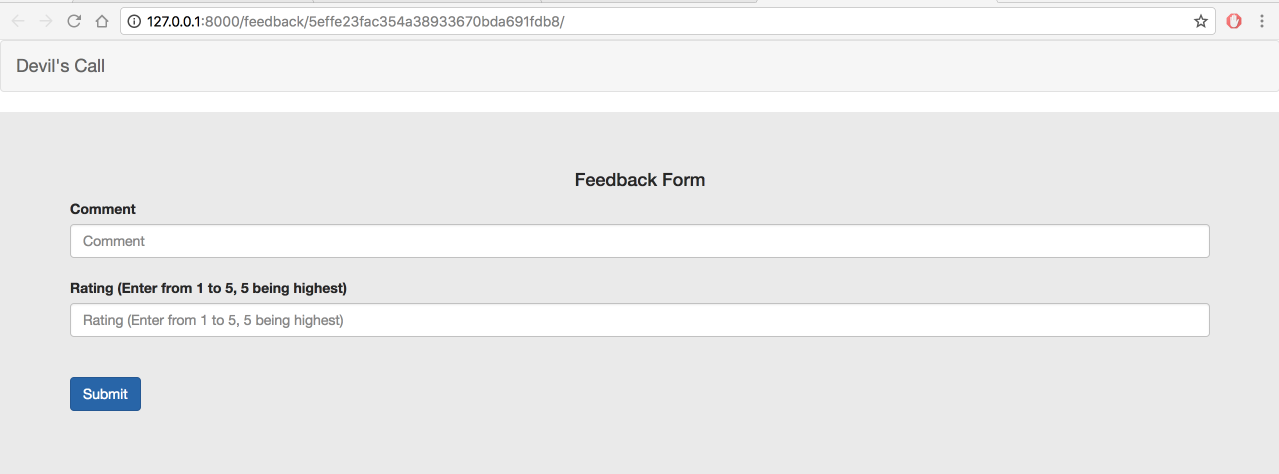
After giving Y and entering 2 in no of guests and clicking submit button:



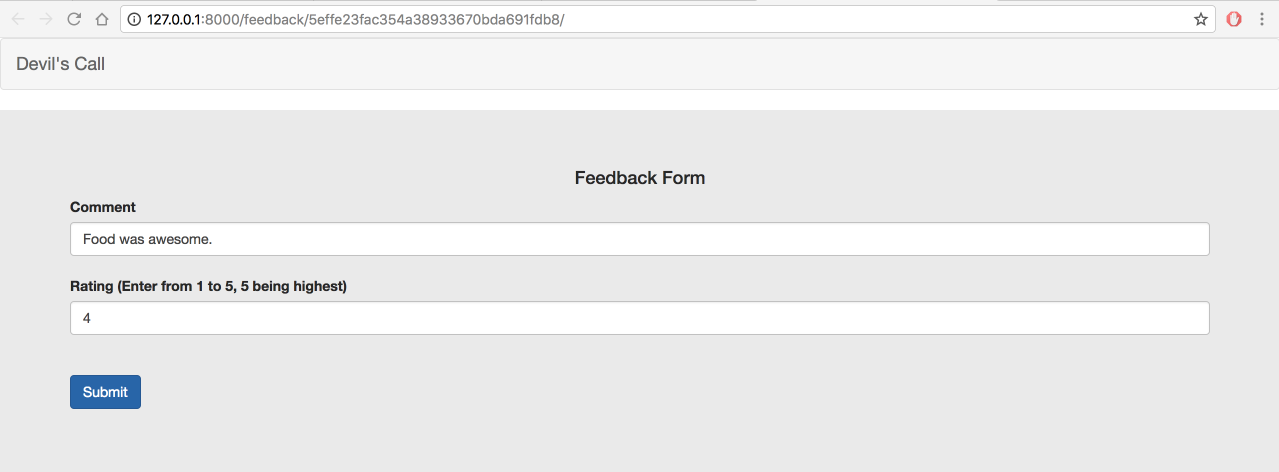
Feedback email:



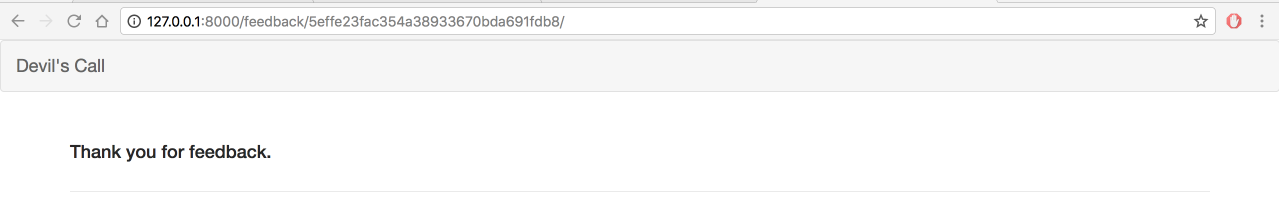
After clicking on Click to feedback button:



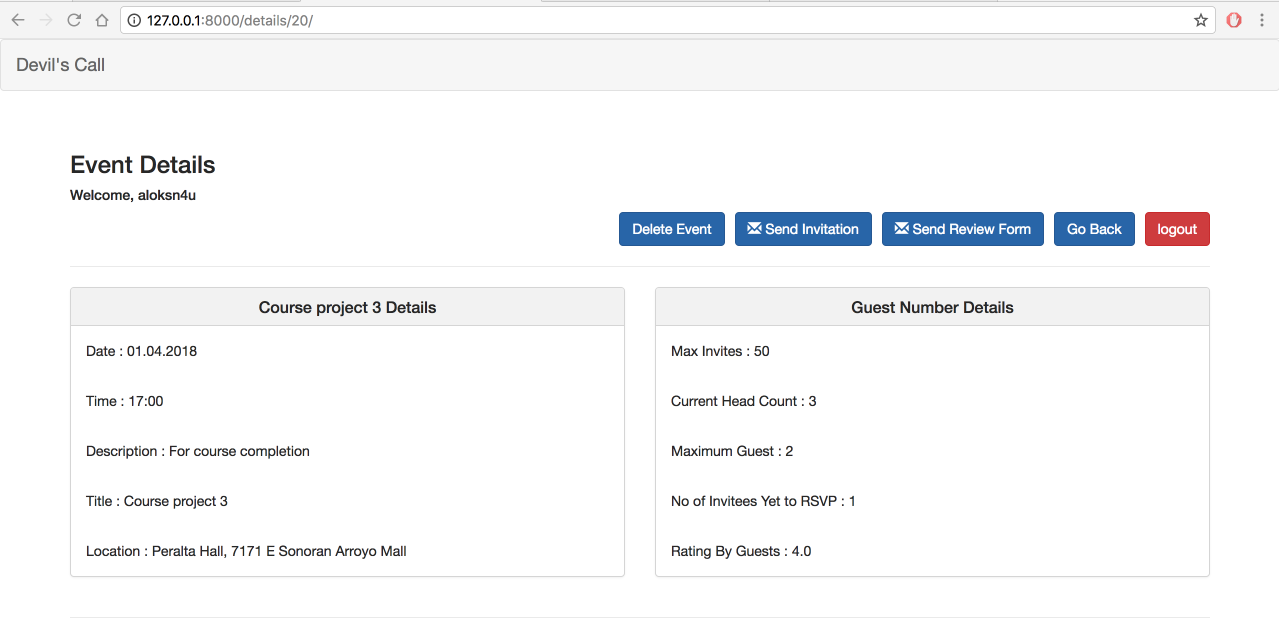
After entering feedback details:



After clicking on Submit button:



After one feedback and one RSVP, clicking on event view details:



# SCOPE AND ASSUMPTIONS:

ASU maintains a standard Data Warehouse of details like student details linked to asurite, organizations, departments and programs. As this is a project that can be potentially integrated with the ASU services like OrgSync, which has the existing data, we will be setting up the database with student and organization details. Registration of Clubs and members are out of the Scope of this project. We will be creating an evite and invitee’s management application for students and organizations who are part of the existing ASU system.

As we may not have access to ASU student clubs during the development phase of project, we will be creating database for storing Club and their member information. Hence, we will manage the data manually as we need these details for sending invitations.

# CONCLUSION:

**REFERENCES:**