CS 517/MIS 517
Database Design and Management

Fall 2015

FINAL PROJECT REPORT

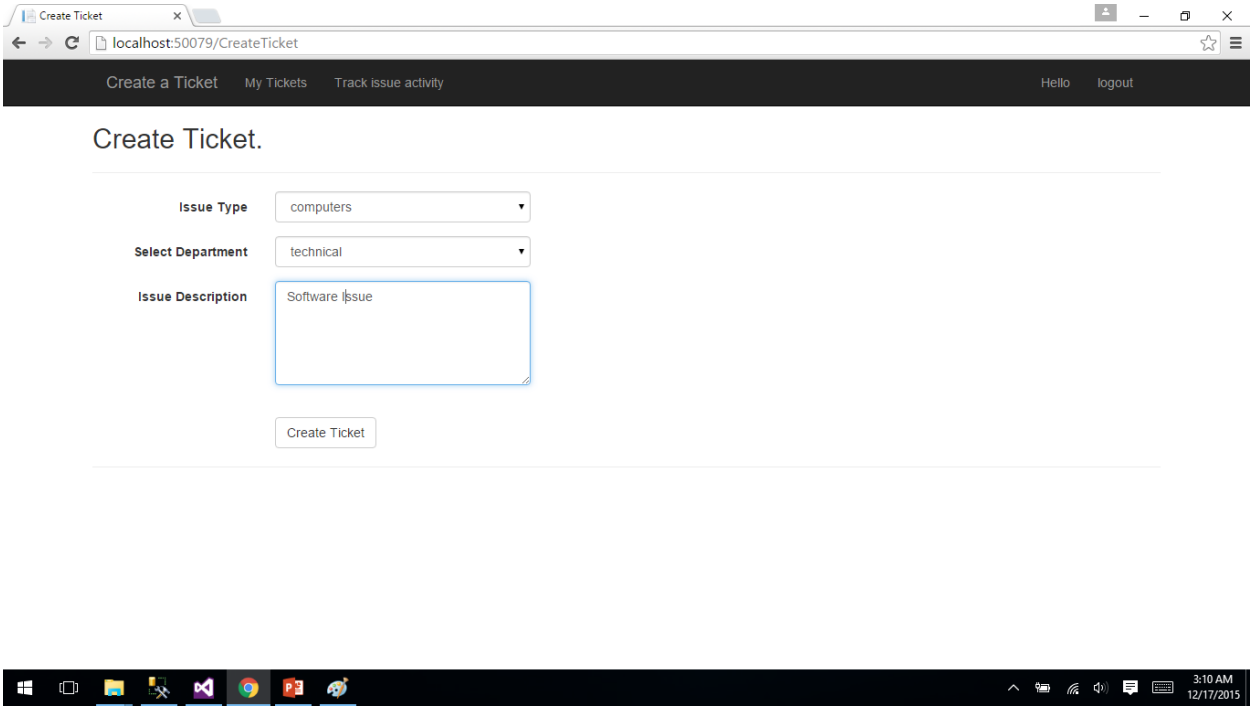
TICKETING MANAGEMENT SYSTEM

Rajeswar Reddy Veeraballi
S1018686@monmouth.edu

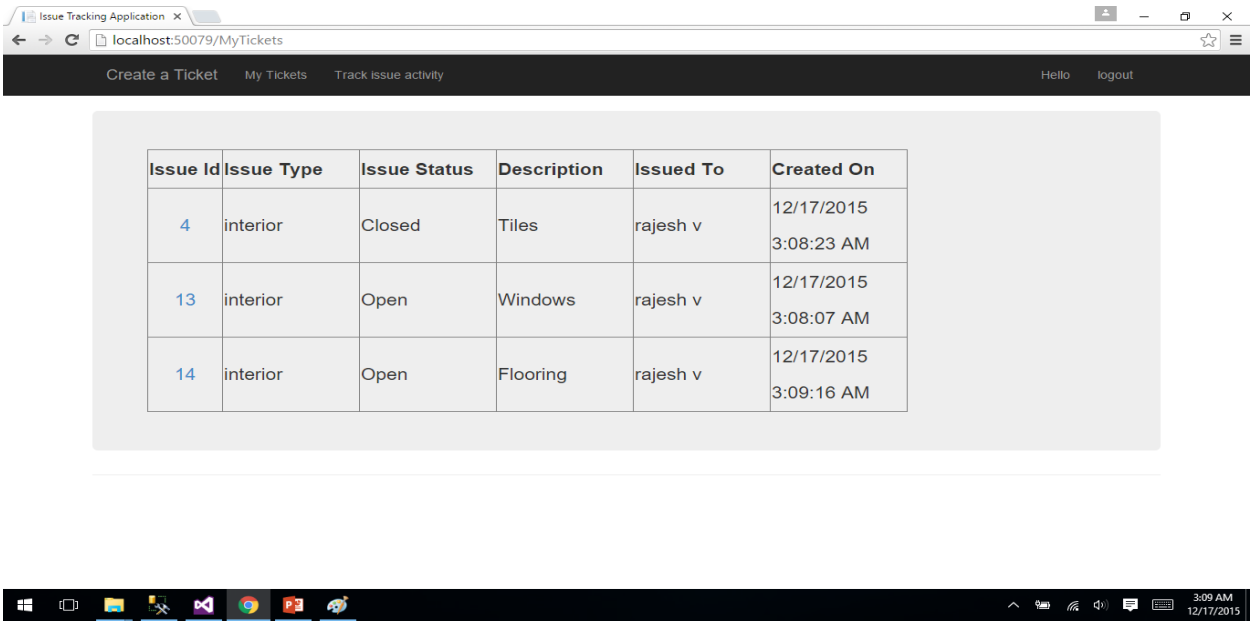
Project Description:

This is a Ticketing Management System which is used for any institution to post on campus issues to the institution management.

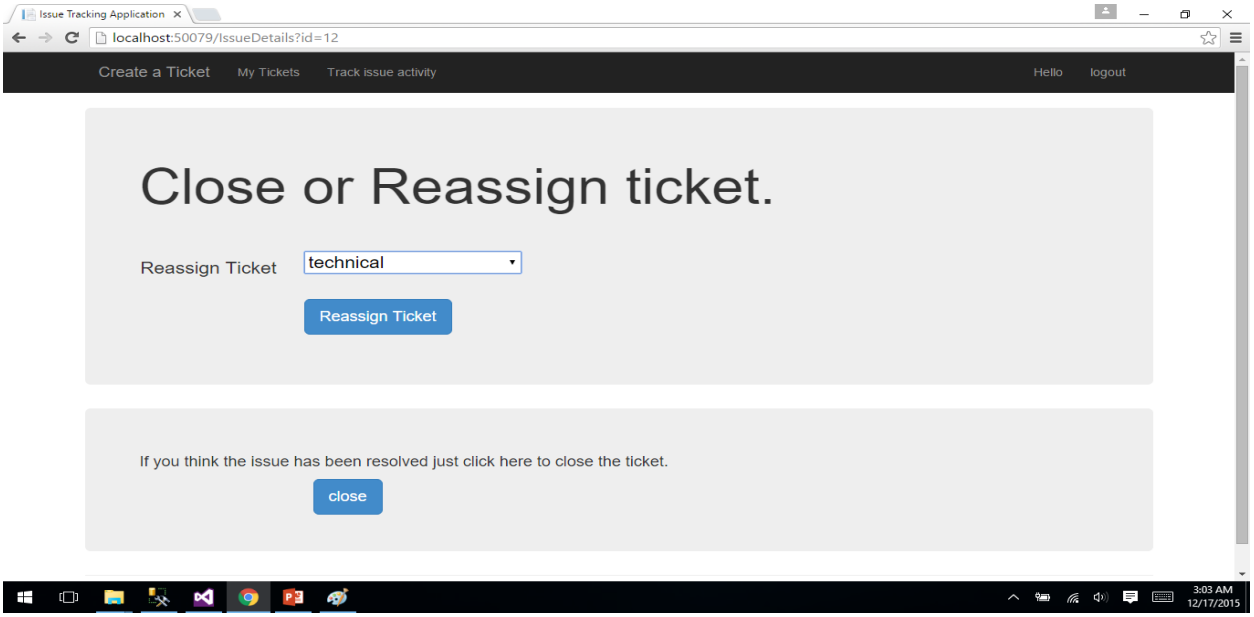
Any Person having an issue can request service by registering to the portal and once he registers he can login and can create a ticket based on the issue. The ticket can be assigned to department based on the issue. When the admin of the corresponding department can login and view all the tickets assigned to him from the 'My Tickets' page from the portal.

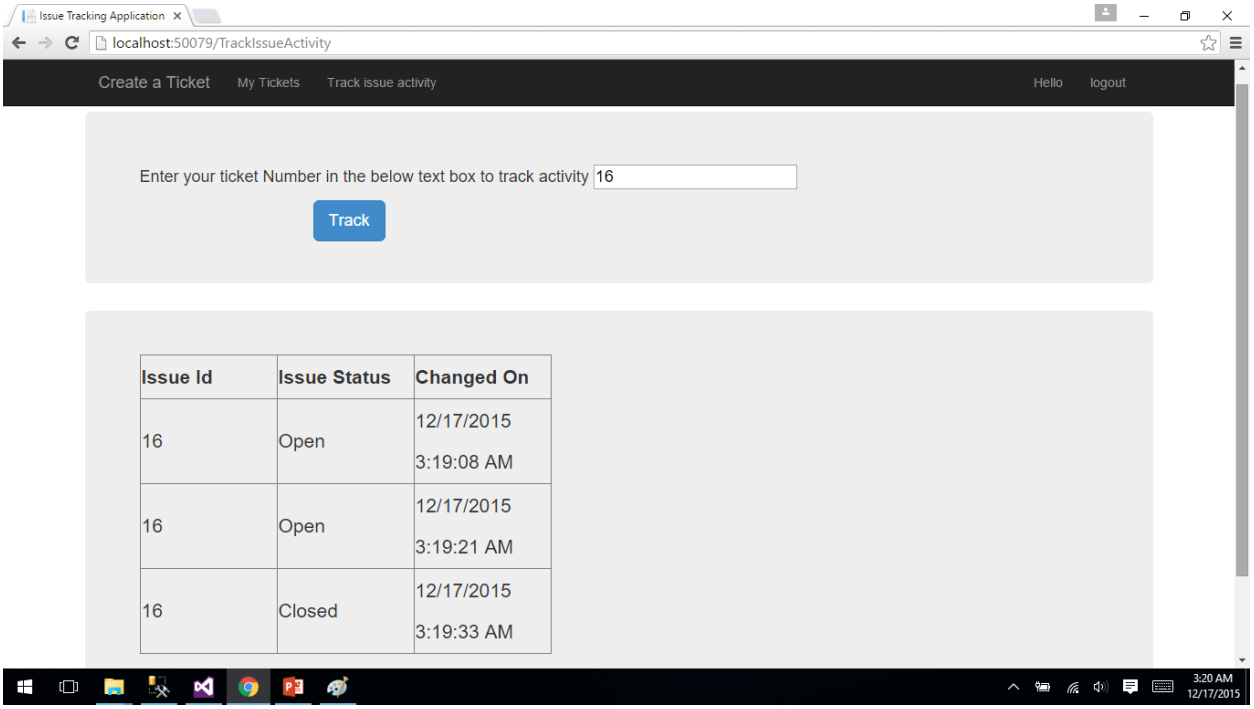


Note: I already assigned department admin for each department, for technical- paul, electrical- john, Interior/Furniture- rajesh, Gardening/Playground- ram and password as 123 for all logins. So, for example is a user posts an issue by selecting department as technical then the issue is assigned paul, and paul should login as the department admin to check the ticket.



Department admin can pick any particular ticket by clicking on issue id number and reassign the tickets to appropriate department if the ticket has been originally requested by user to wrong department. If the admin feels the issue has been resolved then he can close the ticket at any point of time. The user can type on the ticket number in the 'Track Issue Activity' and can know the status of the ticket.





Note: Please find all other portal screens in the 'Ticketing Management System portal screens.zip file.

Design Methodology:

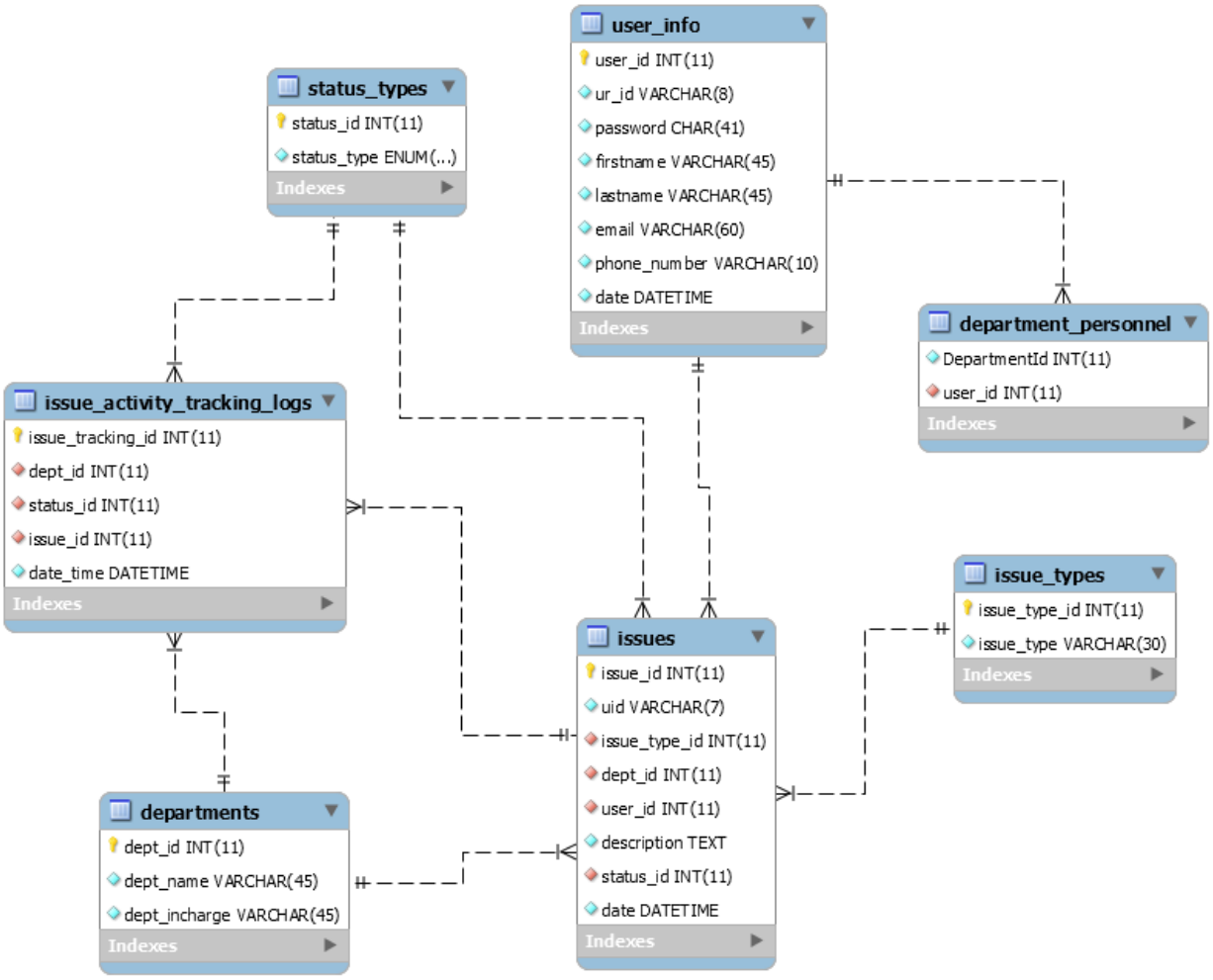
Portal Implementation: Ticketing Management System is a web and mobile based application developed by using **Asp.net, Bootstrap and Ado.net** is used for database connectivity.

- By the use of **Bootstrap** the application is made **mobile compatible**.
- Portal has multiple screens (5 screens) Login validation, Registration, Create Ticket, My tickets, Track issue activity.
- User and admin logins.

Database Implementation:

ER Diagram: MySQLworkbench application is used to create an ER digram.

The ER diagram shows entities, relations and relationships.



- SQL Server 2012 is used as database.

Below are the 7 entities and relations were identified to track Ticketing Management System:

Entities	Primary Keys
• user_info	user_id
• issues	issue_id
• issue_types	issue_type_id
• departments	dept_id
• department_personnel	
• status_types	status_id
• issue_activity_tracking_logs	issue_tracking_id

Foreign Keys in Entities:

Entities		Foreign Keys
• issues	→	user_id, issue_type_id, dept_id, status_id
• department_personnel	→	user_id
• issue_activity_tracking_logs	→	issue_id, dept_id, status_id

Stored Procedures:

- Stored Procedures for all DB interaction
- 7 procedures – 2 with Transaction protection and Exception Handling

  Below are some stored procedures explained:

- STORED PROCEDURE with Transaction protection and Exception Handling with comments to reassign tickets:

/* this will reassign the ticket*/

```
USE [Ticketing]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[uspReassignTicket] @IssueId int,@deptId int
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION
            UPDATE [dbo].[issues] SET [dept_id]= @deptId,date =GETDATE() WHERE  [issue_id]=@IssueId
          COMMIT TRANSACTION
/* If update statement is successfull then commits transaction */
  END TRY

  BEGIN CATCH
  IF @@TRANCOUNT > 0
/* If update statement fails then the transaction will be rolled back */

    ROLLBACK TRANSACTION;

    DECLARE @ErrorNumber INT = ERROR_NUMBER();
/* This will display the error where it occured */
    DECLARE @ErrorLine INT = ERROR_LINE();
    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
```

```
   DECLARE @ErrorState INT = ERROR_STATE();

   PRINT 'Error number: ' + CAST(@ErrorNumber AS VARCHAR(10));
   PRINT 'Error line number: ' + CAST(@ErrorLine AS VARCHAR(10));

   RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH
END
```

- STORED PROCEDURE with Transaction protection and Exception Handling with comments to close ticket

```
/* this will close the ticket*/
USE [Ticketing]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[uspCloseTicket] @IssueId int
AS
BEGIN
  BEGIN TRY
   BEGIN TRANSACTION
            UPDATE [dbo].[issues] SET [status_id]= 2, date=GETDATE() WHERE  [issue_id]=@IssueId
           COMMIT TRANSACTION
           -- If update statement is successfull then commits transaction
  END TRY

  BEGIN CATCH
  IF @@TRANCOUNT > 0
  -- If update statement fails then the transaction will be rolled back

   ROLLBACK TRANSACTION;

   DECLARE @ErrorNumber INT = ERROR_NUMBER();
 --This will display the error where it occured
   DECLARE @ErrorLine INT = ERROR_LINE();
   DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
   DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
   DECLARE @ErrorState INT = ERROR_STATE();
```

```
  PRINT 'Error number: ' + CAST(@ErrorNumber AS VARCHAR(10));
  PRINT 'Error line number: ' + CAST(@ErrorLine AS VARCHAR(10));

  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
 END CATCH
        END
```

- STORED PROCEDURE by using JOINS

/* this SP is done using joins and gets all the issues for a user depending on the department he belongs to*/

```
USE [Ticketing]
GO
CREATE PROCEDURE [dbo].uspGetIssues @UserId int
AS

        SELECT issue_id,[issue_type],[description],[status_type],[issues].[date], [firstname]+' '+
[lastname] as CreatedBy
        FROM [dbo].[user_info]
        JOIN [dbo].[Department_Personnel] ON
[dbo].[Department_Personnel].UserId=[user_info].[user_id]
        JOIN [dbo].[Departments] ON
[Department_Personnel].DepartmentId=[dbo].[Departments].[dept_id]
        JOIN [dbo].[issues] ON [dbo].[issues].[dept_id]=[dbo].[Departments].[dept_id]
        JOIN [issue_types] ON [issues].[issue_type_id]=[issue_types].[issue_type_id]
        JOIN [dbo].[status_types] ON [issues].[status_id]=[dbo].[status_types].[status_id]

        where [user_info].[user_id]=@UserId

GO
```

Triggers:

- Triggers for inserting and updating Issues log

  Below is one of the triggers explained:

/* this trigger will insert issue into issue log whenever issue is updated */

```
CREATE TRIGGER [dbo].[Issue_UPDATE]
    ON [dbo].[issues]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @DeptId INT
        DECLARE @StatusId INT
    DECLARE @IssueId INT
        DECLARE @Date DATETIME


    SELECT @DeptId = INSERTED.dept_id, @StatusId=INSERTED.status_id, @IssueId=INSERTED.issue_id,
@Date=INSERTED.date
    FROM INSERTED
 iF ( UPDATE( dept_id) OR UPDATE(status_id))
    INSERT INTO issue_activity_tracking_logs
    VALUES(@DeptId, @StatusId,@IssueId,@Date)
END
```

VIEWS:

- View for issue activity information

Below is the explained View:

/* this view has issue activity info*/

```
CREATE VIEW V_IssueActivity_Details
AS SELECT
        issue_id,
        [status_type],
        [dept_name],
        date_time
FROM [dbo].[issue_activity_tracking_logs]
JOIN [dbo].[Department_Personnel]
        ON [dbo].[Department_Personnel].[DepartmentId] = [issue_activity_tracking_logs].[dept_id]
JOIN [dbo].[Departments]
        ON [Department_Personnel].DepartmentId = [dbo].[Departments].[dept_id]
JOIN [dbo].[status_types]
        ON [issue_activity_tracking_logs].[status_id] = [dbo].[status_types].[status_id]
```

FUNCTIONS:

- Function for checking user existence

Below is the explained Function:

/* Calls the function to check the existence of the user before logging in into the portal */

```
        CREATE FUNCTION dbo.udfCheckIfUserExists(@Name varchar, @Password varchar)
 RETURNS int
 AS
 BEGIN
 DECLARE @Exists INT
  IF EXISTS (SELECT * FROM [dbo].[user_info] WHERE ur_id= @Name and [password]=@Password)
  set @Exists= (SELECT [user_id] FROM [dbo].[user_info] WHERE ur_id= @Name and
[password]=@Password)
        else
        set @Exists= 0
        RETURN @Exists
 END
```

DataBase Query Optimization:

Most of the columns used in joins are made on primary keys which are clustered indexes and I created the non-clustered index on columns that are used in joins but are not clustered index.

- Created Non-Clustered Index on dept_id in issues table

```
CREATE INDEX INDX_Issue_DeptId ON Issues (dept_Id)
```

- Additional keys for query optimization - date, department id, issues

Supporting Documents:

- Portal Application File as Ticketing Management System Portal Application File.zip
- ER Diagram(.mwb and pdf) as Ticketing Management System ER Diagram.zip
- SQL Server DB Schema(with Stored Procedures with Exception Handling and Transactions, Views, Triggers, Functions, Indexes all with comments and Data) as Ticketing Management System.sql
- Presentation Slides as Ticketing Management System Presentation Slides.pptx
- Portal Screens as Ticketing Management System Portal Screens.zip