# 

# 

# **Quest**

# **Data Tool**

# **Documentation**

# 

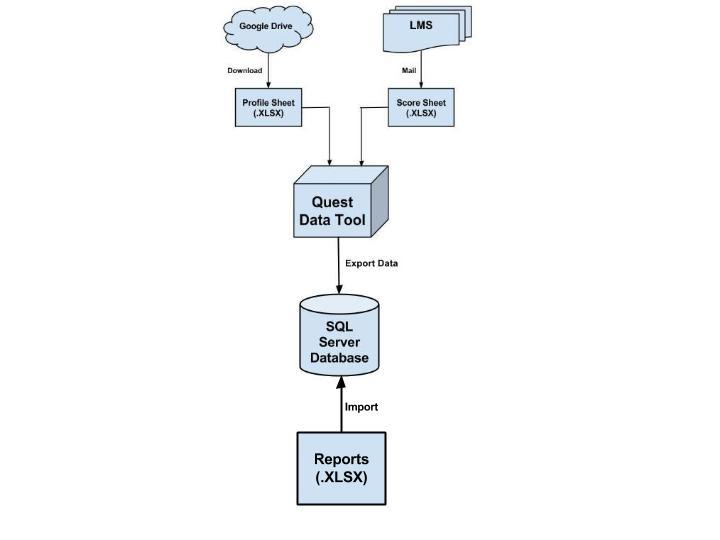
# 

# 

# 

# 

# **Overall architecture Diagram**



# **Main Components**

**Data Tool**

Quest Data Tool is C# application integrated with Entity Framework, JSON and Linq used to store data from XLSX format sheets directly to database which is hosted on Microsoft SQL Server.

The designing and querying of data is done using Entity Framework and Linq while the mapping of data from XLSX Sheets to the database is done using JSON. The reports are generated in XLSX sheets using Microsoft SQL Server database and pivot table concepts.

Entity framework code first approach has been used in designing the database.

**SQL Server**

Microsoft SQL Server is used for remote connection of database with the Quest Data Tool and hosting the database in the network so that all systems inside the network can remotely access the database easily.

This database cannot be accessed outside of Quest Network.

# 

# 

# **Where to find the code?**

Code can be found on Github.

Link:

<https://github.com/abhinavgujjar/QuestExcelReader>

# **How to generate reports?**

Step 1: Download the Report Template (.XLSX file).

Step 2: Open the Report Template using .XLSX reader.

Step 3: Next go to “Data” option and then “Refresh All”.

Step 4: If it asks for password then give Database Server user password.

Step 5: Now the latest reports are automatically updated.

**If Error**

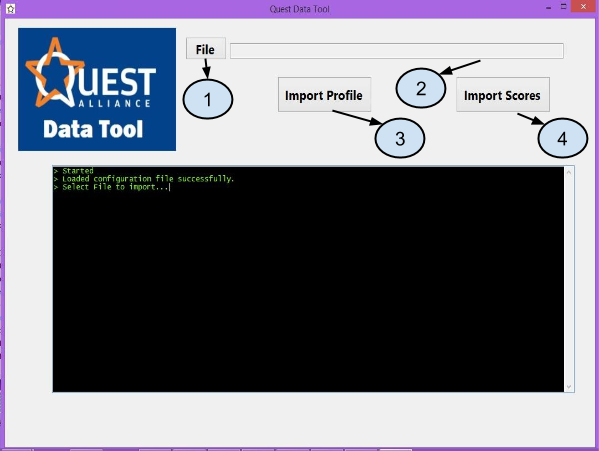
1. Check whether the system in which Database Server is stored is switched on and

connected to network or not.

2. Check whether the user system is connected to the network or not.

3. If some of the sheets are refreshed then try refreshing again.

**How to use the Data Tool?**



 **File Button**:

Function - Opens the file browser.

**File Path Display:**

Function - Displays the path of the selected file.

**Import Profile Button:**

Function-Click to import profile if profile sheet is selected.

**Import Score Button**:

Function - Click to import score if score sheet is selected.

# **How to setup SQL Server?**

# Configuring MS SQL Server for Remote Access:

1. Open **cliconfg** from a RUN prompt and make sure TCP/IP is an enabled protocol.
2. **For SQL 2005/2008/2008 R2:** Check the Services tool, **Start** > **Administrative** **Tools** > **Services**, to see that the service named **SQL Server (MSSQLSERVER**) is started.
3. **For MS SQL 2012:** Use the Windows key or hover over the left lower corner of the desktop and select **Administrative Tools**, then **Services** to see that the service named **SQL Server** (**MSSQLSERVER**) is started.
4. Ensure that you are using the correct credentials to authenticate. The default SQL administrator account is named **sa** and if you built the server from one of our server images with MSSQL pre-installed, the password will be in a text file on the root of the C partition.
5. Use **netstat –an** from the command prompt to verify that the server is listening for SQL traffic on the correct ports.
6. If the server is not listening for SQL traffic on the correct ports, use SQL Server Configuration Manager to change the ports.
   * For MS SQL 2005/2008/2008 R2, go to **Start** > **All Programs** > **Microsoft SQL Server 2005** (or**2008/2008 R2**) > **Configuration Tools** > **SQL Server Configuration Manager**.
   * For MS SQL 2012: Use the Windows key or hover over the left lower corner of the desktop and select **All Programs** > **Microsoft SQL Server 2012** > **Configuration Tools** > **SQL Server Configuration Manager**.
   * Open the **+** next to SQL Server Network Configuration.
   * Right-click **TCP/IP** and select **Properties**.
   * Select **IP Addresses**.
   * All TCP ports mentioned on all interfaces should be 1433. Change this to reflect the correct port number and restart the SQL services.
7. If you are using named instances when installing SQL, giving you the ability to host multiple SQL versions or service types, you will have to specify the name of the SQL instance when connecting rather than just using the server’s name or IP. If you have created a named instance, you will need to access it by appending the name to the server’s name or IP, following a backslash (e.g. 12.34.56.78\SQLINSTANCENAME or SQLSERVERNAME\SQLINSTANCENAME).

#### Create exceptions in Windows Firewall:

1. Click **Start**, point to **Control Panel**, point to **Windows Firewall Settings**
2. Click **Change settings** link, point to **Exceptions tab**
3. Click **Add port...** button, do the following:
4. Name: 1433  
   Port number: 1433  
   Protocol: TCP

Click **OK**, and click **apply**.

# 

# **Code Documentation**

**Libraries Used:**

1. **EPPlus** - EPPlus is a .net library that reads and writes Excel 2007/2010 files using the Open Office Xml format (xlsx).

**Framework Used:**

1. **Entity Framework** - Entity Framework (EF) is an open source Object-Relational Mapping (ORM) framework for ADO.NET that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write.

**Data Models: (ExcelReader >> Models)**

1. **StudentProfile.cs** - Used for defining students profile data table.
2. **Subject.cs** - Used for defining subjects data table.
3. **SubjectScore.cs** - Used for defining subjects score data table.
4. **Placement.cs** - Used for defining students placement data table.
5. **Post\_Placement.cs** - Used for defining students post placement data table.

**Migrations: (ExcelReader >> Migrations)**

1. **Configuration.cs** - Used for migration of tables.

**Main Classes:**

**ExcelReader**

1. **Importer.cs** - Importing data from excel to the datatable.
2. **WorksheetHelper.cs** - Helper class for reading the excel file.

**Gimli**

1. **Form1.cs** - This is main class which interacts with the UI and most of the coding is done in this class only.

**Configuration Files:**

1. **legacyimport.config** - This file contains configuration for profile sheets.

Example:

'FirstName': 'B',

'LastName' : 'C',

This means profile sheet’s **B** column indicates First name of students and **C** column

indicates Last name of student.

This naming of columns like **B** column is named as “**FirstName**” will be used by

calling the worksheet helper in the **importer.cs** file to get the values of B column.

Example:

student.FirstName = \_helper.getCellValue("FirstName", rowIndex);

student.LastName = \_helper.getCellValue("LastName", rowIndex);

\*\* “**student**” is the object of profile datatable.

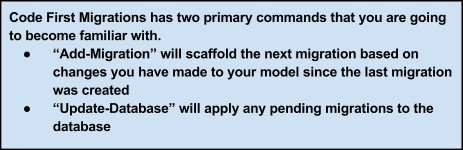
**2**. **Score Sheet** - For score sheet we are not using configuration files rather we are tracking the full sheet column wise. For more details refer to the “**buttonImportScores**” method in “**Form1.cs**” file.

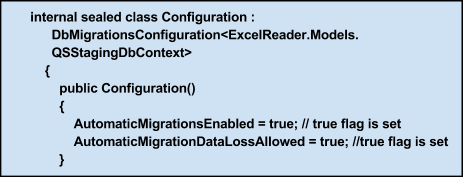
**Overall functioning of code**

* **C# - ExcelReader**
  + **References :** References contains the reference of the frameworks and libraries used in the data tool. All frameworks and libraries are added using **Nuget Package Manager**.
  + **Migrations** 
    - **Configuration.cs :** It contains the migration class which is required for database migration.

Go to **Package Manager Console** and then execute the following

commands for migrations.



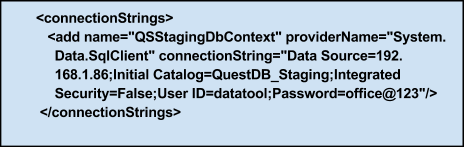


* + **Models**
    - **StudentProfile.cs** **:** The attributes of Student Profile table i.e, “**StudentProfiles**” is declared in this class.
    - **Placement.cs :** The attributes of Student Placement table i.e, “**Placements**” is declared in this class.
    - **Post\_Placement :** The attributes of Student Post Placement table i.e, “**Post\_Placement**” is declared in this class.
    - **Subject.cs :** The attributes of Subjects table i.e, “**Subjects**” is declared in this class.
    - SubjectScore.cs : The attributes of Student Subject Scores table i.e, “**SubjectScores**” is declared in this class.

* + **App.config :** The app.config is an XML file with built-in configuration sections and the ability to add your own custom settings. Custom settings can be added by using the built-in configuration sections (such as connectionStrings) or adding your own custom configuration sections.

We have used app.config to specify connection string for the server.

For example the connection string used for our server is,



* + **Importer.cs :** This imports data from .XLSX sheet to the data table using worksheetHelper class which is then inserted in the database. The method of import is already discussed above.
  + WorksheetHelper.cs : This is a helper class which helps in importing the data from .XLSX sheet to the database.
* **Gimli :** This is the main project which will interact directly with the user.
  + **References :** Same as ExcelReader.
  + **App.config** **:** Same as Excel Reader.
  + Form1.cs : This is main backend of the UI which will interact with the user it contains events like button press etc. The datatable imported by importer class is passed in this class and then the data is inserted in the database.

**Overall Flow:**

The database is created using the Entity Framework and the structure of the tables is defined in the models folder inside the respective classes. The importer class in the ExcelReader reads the data from .XLSX sheet into the datatable and then sends the datatable to the form1.cs class where it is validated and added to the database.

**Tables and columns**

**Table Name : StudentProfiles**

**Columns :**

* **int Id**
* **string FirstName**
* **string LastName**
* **string OrganisationName**
* **string TrainingCenter**
* **string BatchNumber**
* **string Prefix**
* **string Uid**
* **string Gender**
* **string Mobile**
* **String Education**
* **string MaritalStatus**
* **string Email**
* **int Age**
* **string WorkExperience**
* **string ParentName**
* **string ParentContact**
* **string PermanentAddress**
* **string FamilyMonthlyIncome**
* **string ParentOccupation**
* **string State**
* **string BatchStart**
* **string BatchEnd**
* **string EmploymentStatus**

**Table Name : Placements**

**Columns :**

* **int Id**
* **string StudentUid**
* **string EmploymentStatus**
* **string Company**
* **string Position**
* **string NatureOfJob**
* **string IfContEduReason**
* **string IfDropOutReason**
* **string UpdatedContact**
* **string Salary**
* **string Location**

**Table Name : Post\_Placement**

**Columns :**

* **int Id**
* **string StudentUid**
* **string ContinueJob**
* **string Company**
* **string Position**
* **string UpdatedContact**
* **string Salary**

**Table Name : Subjects**

**Columns :**

* **int Id**
* **string SubjectName**
* **string TotalMarks**

**Table Name : SubjectScores**

**Columns :**

* **int Id**
* **string StudentUID**
* **string Subject**
* **string Lessons**
* **int Score**

**Troubleshooting**

**What if the file does not load due to data discrepancy?**

* If there is any data discrepancy error message will be data tool console and the data will not be imported. Correct the data and try to import again.

**What kind of file formats are supported for the file import?**

* .XLSX formats are supported for the import.

**What is the structure for the import files?**

* There is standard structure for import files. Profile sheet is downloaded from the drive and score sheet is generated from the LMS so the format is standard.

**Testing**

The testing import files are uploaded along with github repository mentioned above where the project resides.