A Developer's Guide to Appgregate

Tech Stack Overview:

This program uses several languages and frameworks to build out its front and back end. The whole project is encapsulated in ASP.NET on the .NET framework. All webpages are built in ASP.NET which combines elements of both HTML and CSS in .aspx files. Underlying each page is logic written in C#. The database storing all information is built in SQLServer. Testers for this project utilize both NUnit testing, and Selenium.

Important Files Overview:

All files for this project are stored in three distinct folders: Excess Documentation, Front-End, and Selenium Tester. Anything in Excess Documentation can be ignored, as these files were only used in the development of the application, and have no practical application to any user or future modifier of this application. SeleniumTester contains all the files necessary to run the entire Selenium Tester Suite. The tester suite is stored in SeleniumTester.sln, and requires the project file to be open for Appgregate in order to run correctly. Inside the Front-End Folder, the actual application folder, Appgregate, and the Build Script folder, Build Script. The Build Script folder contains a .bat file which can be used to run the project without an IDE. The Appgregate Folder contains many important files, including all the web pages, the Appregate project file Appgregate.sln, and the folder NUnit Test which contains all the NUnit testing.

Build Script:

The build script is a very simple batch file that you can click twice to boot the program without an IDE. Instructions on modifying for personal use can be found in the .txt file in the same folder called BootProject.txt. The build script in it's unmodified state is called BuildProject.bat.

SQL Schema:

There are three tables in the set, with Schemas as follows:

CommentsTable: Used for storing and Retrieving comments.

Update | Script File: dbo.CommentsTable.sql

Name	Data Type	Allow Nulls	Default
reviewID	int	☐	
username	varchar(50)	☐	
appID	int	☐	
appRating	int	☑	((1))
commentText	varchar(MAX)	☑	
		☐	

AppTable: Where all existing apps on the AppStore can be added and recieved.

Update | Script File: dbo.AppTable.sql

Name	Data Type	Allow Nulls	Default
appID	int	☐	
Name	nvarchar(50)	☐	
Description	nvarchar(MAX)	☐	
Organization	nvarchar(50)	☐	
Platform(s)	nvarchar(50)	☐	
Version(s)	nvarchar(50)	☐	
Rating	int	☐	
View Comments	varchar(50)	☐	
		☐	

AppRequests: Where all current requests are stored until they are either Accepted into the AppTable or denied.

Name	Data Type	Allow Nulls	Default
Id	int	☐	
Name	nvarchar(50)	☑	
Description	nvarchar(MAX)	☑	
Organization	nvarchar(50)	☑	
Platform(s)	nvarchar(50)	☑	
Version(s)	nvarchar(50)	☑	
		☐	

There is also several ASP.NET user tables built in, but seeing how there is extensive ASP.NET documentation on these already, we opted not to include them.

Web Pages:

AppComments.aspx + AppComments.aspx.cs

Front-End(All ASP/HTML/CSS):

TextBox to handle comment input, set with a default string of “Feel Free to Leave a Comment” with a size of 20 characters in length by 5 characters in height.
DropDownList Contains 5 list items, designating integers from 1 to 5.
Button Simple button that contains text Submit which activates “SubmitButton_Click” on the back end.
GridView Contains multiple BoundFields, A LinkField hooked to a LinkButton for the “DenyButton_Click” on the back end. Also, the GridView has the Mygrid_RowDataBound RowDataBound event, and the GridView1_RowCommand RowCommand
SqlDataSource Activates the Default query to grab all the comments for a particular app

Back-End(All C#):

public Boolean Page_Load(object sender, EventArgs e):
Loads the page from other pages.
protected void DenyButton_Click(object sender, EventArgs e):
Is used as a dummy shell method to prevent compiler errors.
protected void Mygrid_RowDataBound(object sender, GridViewRowEventArgs e):
This method only allows the delete button to appear when Moderators or Admins are logged in.
protected void SubmitButton_Click(object sender, EventArgs e):
Gathers information from the comment fields and inserts all information gathered into a database using a SqlCommand object
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e):
Allows the delete button to delete according to row.

Default.aspx + Default.aspx.cs

Front-End(ASP/HTML/CSS):
Div Class: Contains TextBox and Button. Used for the Search Bar. On clicking the Submit button, it activates “SubmitButton_Click” on the back end.
GridView: Contains several DataFields, A template field for the View Comments button which activates “ViewComments_Click” on the back end.

SqlDataSource: This is the SQL query that activates by default and provides all the entries to the GridView from the SQL database

Back-End(C#):
public Boolean Page_Load(object sender, EventArgs e)
Loads the Default page from outside sources
protected void ViewComments_Click(object sender, EventArgs e)
Is implemented to prevent compilation errors
protected void SubmitButton_Click(object sender, EventArgs e)
This activates a query to reset the GridView with only entries that match the search.
ModerateApps.aspx + ModerateApps.aspx.cs
Front-End:
GridView Contains several DataFields, and two buttons, an AcceptButton that activates the “GridView1_Row” Command on the back end with Event Args “Accept” and a Deny Button with the same activate only with the Event Arg “Deny”
SqlDataSource Fills the GridView with all existing application requests

Back-End:
public Boolean Page_Load(object sender, EventArgs e)
Loads the AppComments page from outside sources.
protected void AcceptButton_Click(object sender, EventArgs e)
Is implemented to prevent compilation errors
protected void DenyButton_Click(object sender, EventArgs e)
Is implemented to prevent compilation errors
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
Either Accepts the request into the App Store or deletes it depending on Moderator/Admin input

Site.Master
This is provided by ASP.NET and was not modified, but it is in complete control of the theme and overlay of the HTML + CSS of this project.

Unit Tests:

All Unit tests are stored in the NUnit Test folder, they are as follows:
appCommentsExists(): Checks if the Comments page is created
defaultExists(): Checks if the Default page is created

moderateApps(): Checks if the Moderate page is create
defaultPageLoad(): Checks if Default page can load.
appCommentsPageLoad(): Checks if the App Comments page can load
moderateAppsPageLoad(): Checks if the Moderate Apps page load

Integration Tests:

All Integration tests are stored in the Selenium Tester project, they are as follows:
loginTest(): Attempts to login to the Appgregate website.
submitAppTest(): Submits an app to the Appgregate website to be approved.
denyAppTest(): Denies an app that has been submitted to Appgregate as an administrator.
acceptAppTest(): Accepts an app that has been submitted to Appgregate as an administrator.
addCommentTest(): Submits a comment to an app.
removeComment(): Removes a comment that has been submitted to an app as a moderator.