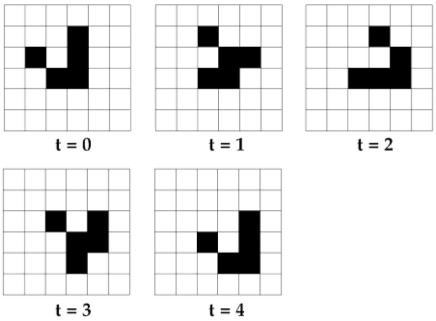
Study Period 2 2016 – Assignment 2



Game of Life Website

Due:	Sunday 21/08/2016 11:59 PM Melbourne Time
Assignment Type:	Individual or Group (of 2 or 3 members)
Total Marks:	**30 marks (+ 5 bonus marks)**

Aim

This assignment will continue the "Game of Life" theme found in Assignment 1. The aim is to develop a **Web Application** in two parts using **MVC** for the front-end, i.e., the "public" website and **Web Forms** for the internal administration website.

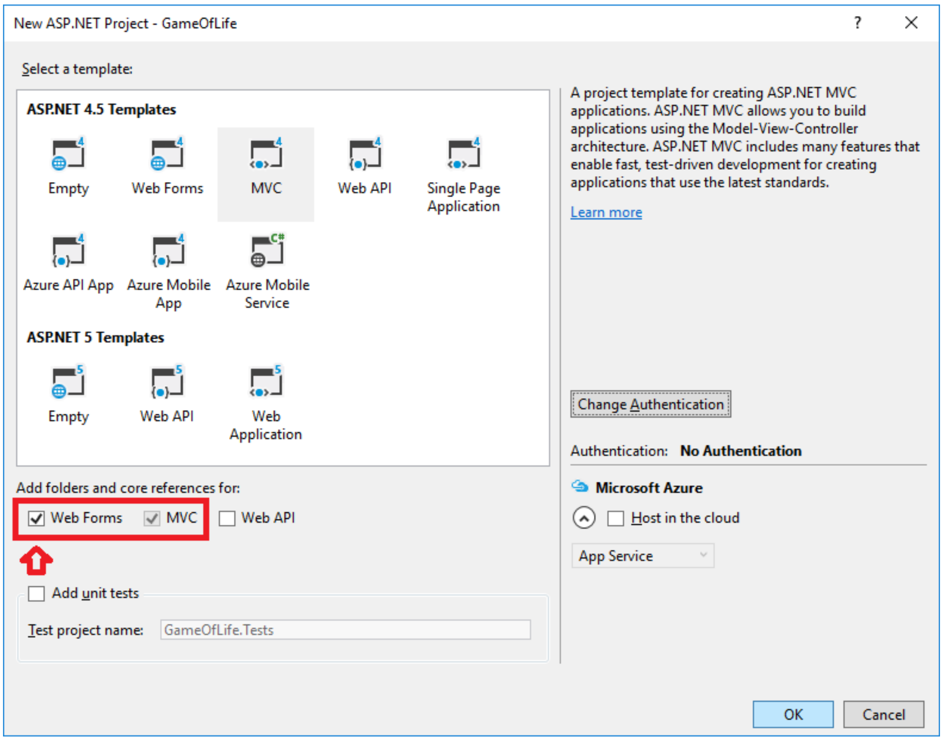
Data will be stored in a SQL Server database.

Basic requirements of the application:

1. Client website **must** be created using **MVC** and **EF**
2. Admin website **must** be created using **Web Forms** and plain **ADO.NET**

✓ **Special Note:**

If you implement both websites, MVC and Web Forms, in the **same** project you can use a LocalDB (i.e., mdf and ldf files found in App_Data). This is the **recommended** approach. To create a project that will support both frameworks you can import the references at any time or during creation, here is a screenshot:



New ASP.NET Project - GameOfLife

Select a template:

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Add folders and core references for:

☑ Web Forms ☑ MVC □ Web API

□ Add unit tests

Test project name: GameOfLife.Tests

Authentication: **No Authentication**

□ Host in the cloud

If you are going to implement the MVC and Web Forms with separate project's you will have to use a database hosted at localhost, e.g., SQL Server Express 2014 – as both of the projects will need to be referring to the same database. If you do use this approach you will also need to provide the **SQL scripts** to re-create the database/tables in your submission for marking purposes.

Database:

Before you begin, you will have to decide on the database structure, think of such things as:

1. How many tables do I need?
2. Which attributes/columns should appear in which table and why?
3. Which primary/foreign key relations should exist between tables?
4. Adding some test data.

Note there will be provided SQL scripts to create the tables used in the sample solution. You can use these scripts to create your database. You can also change/modify them however you want, or not use them at all.

Part A: Public MVC Website (25 marks)

Use of Web forms and/or plain ADO.NET here will fetch zero marks for the whole of Part A

Create an MVC Web Application that uses Entity Framework for all data-fetching purposes.

You **can use Visual Studio's in-built template for a MVC website**. However, **no lorem ipsum** content is allowed on the site.

A **sample** version of this site can be found here:

http://matthewrmit-001-site1.htempurl.com/

This site is only here to help you **understand** the functionality described below.

Note that for the requirements below the functionality can be integrated into the website according to your own design (or even based on the same design as the provided site). Meaning a feature can be implemented all on one page or spread out over several pages.

The following features are accessible **without logging in**:

1. **Home Page:** General information about the Game of Life and some useful links.
2. **All Templates:** Lists all templates in the system – including the template's name, height, width, cells and who created it.
3. **All Templates Search:** Allows a search for a particular template's name.
4. **Create Active Game:** Creates an active game from a selected template and saves it into the session.
5. **My Active Games:** Lists all active games stored for the current session – these games are not persisted meaning they are lost when the current session ends (e.g., restarting the Web Server, visiting the site with a different browser, etc…).
6. **Delete Active Game:** Removes an active game from the session.
7. **Play Game:** Plays an active game. The game should "tick" every second and display the updated cell's without requiring any user interaction. The game "tick" code is to be implemented *server-side.
8. **Register:** Creates a new user in the system.

The following features are accessible **after logging in**:

9. **Login/Logout:** A user can login and logout once registered.
10. **Create Template:** Creates a new template in the system – note it is the logged-in user that created this template.
11. **My Templates:** Lists all templates the logged-in user has created in the system.
12. **Delete Template:** Removes a template created by the logged-in user from the system.
13. **Save Game:** Saves an active game for the logged-in user. A saved game is persisted and is thus remembered after the user logs out and logs back in later on (even with a different browser).
14. **My Saved Games:** Lists all the saved games for the logged-in user in the system.
15. **Delete Saved Game:** Removes a saved game for the logged-in user from the system.
16. **Play Saved Game:** Plays a saved game. The game should "tick" every second and display the updated cell's without requiring any user interaction. The game "tick" code is to be implemented *server-side.
17. **Site Quality:** This represents the overall website design and style quality.
18. **Security Feature:** Add a security feature – the user password's stored in the database should be encrypted, i.e., the password should **not** be visible as plain-text.

    For example, the password "rmit" would **not** be stored as "rmit" directly, but rather it would go through an encryption and/or hashing process. As an example, using a Blowfish hash (and various other techniques) "rmit" could be stored as "$2a$06$aa3rgX0rr7TcdxFPFUHRj.8vlRzfOd52pwLkJvaCEL5ZWGT2aL/f6".

***Server-side:**

To implement the Game of Life "tick" code server-side (i.e., in C#) and have the browser show the updated cell's content without user interaction will involve either Javascript/jQuery and/or SignalR communication to the server behind the scenes in "real-time" (e.g., every 1 second) to execute a method on the server (much like a WebAPI call) and then download and display the result of that method (i.e., the updated cells).

Part B: Admin Web Forms Website (5 marks)

Use of MVC and/or EF here will fetch zero marks for the whole of Part B

Create an ASP.NET Web Forms admin website, please use ADO.NET for data-fetching purposes.

You **can use Visual Studio's in-built template for a Web Form website**. However, **no lorem ipsum** content is allowed on the site.

A **sample** version of this site can be found here:

http://matthewrmit-001-site1.htempurl.com/Login.aspx

This site is only here to help you **understand** the functionality described below.

The following features are accessible **after logging in**:

1. **Login/Logout:** An admin can login and logout – the admin details are to be hard-coded in advance. At a minimum (and ideally) one admin should exist and the username should be "admin@gmail.com" and the password should be "rmit".
2. **All Users:** Lists all users in the system.
3. **Delete User:** Deletes a user from the system – note this will also delete all the templates and saved games for that user as well.
4. **All Templates:** Lists all templates in the system.
5. **Delete Template:** Deletes a template from the system.

Part B: Admin Web Forms Website – Bonus (5 marks)

1. **Upload Template:** The admin can upload a template .txt file (same format as from Assignment 1) and the uploaded file is parsed and then inserted as a template into the database – note it is the logged-in user that created this template.

   Here is an example of the file content (the provided Blinker file from Assignment 1):

   ```
   5
   5
   XXXXX
   XXXXX
   XOOOX
   XXXXX
   XXXXX
   ```

2. **Stored Procedures:** All the database queries for **Part B** should be implemented with stored procedures – i.e., there should be no plain SQL statements within **Part B**. If you do implement this feature, please indicate that you've done so by mentioning it in a Readme.txt file within your submission.

Coding Standards

- Read the C# coding standard from the following website: http://msdn.microsoft.com/en-us/library/vstudio/ff926074.aspx
- Remember that there are too many to be followed, even if you can stick to 6-10 of the standards, it will be a job well done.
- Do not force yourself to implement every OO feature that you have learnt, use the features wisely and if needed.

Marking Guide

1. Part A (Public MVC Website)	25 marks
a. Home Page	(1.0)
b. All Templates	(1.0)
c. All Templates Search	(1.0)
d. Create Active Game	(2.0)
e. My Active Games	(1.0)
f. Delete Active Game	(1.0)
g. Play Game	(4.0)
h. Register	(1.0)
i. Login/Logout	(2.0)
j. Create Template	(2.0)
k. My Templates	(1.0)
l. Delete Template	(1.0)
m. Save Game	(1.0)
n. My Saved Games	(1.0)
o. Delete Saved Game	(1.0)
p. Play Saved Game	(1.0)
q. Site Quality	(2.0)
r. Security Feature	(1.0)

2. Part B (Admin Web Forms Website)	5 marks
a. Login/Logout	(1.0)
b. All Users	(1.0)
c. Delete User	(1.0)
d. All Templates	(1.0)
e. Delete Template	(1.0)

3. Part B (Admin Web Forms Website – Bonus)	5 marks
a. Upload Template	(3.0)
b. Stored Procedures	(2.0)

Late Submissions:

Late submissions are handled as per usual RMIT regulations – 10% deduction (2.5 marks) per day. You are only allowed to have a maximum of 5 late days.

Submission Instructions

You are free to refer to textbooks and notes, and discuss the design issues (and associated general solutions) with your fellow students on Blackboard; however, the assignment should be your own individual work.

Where you do make use of other references, please cite them in your work. Note that you will only be assessed on your own work.

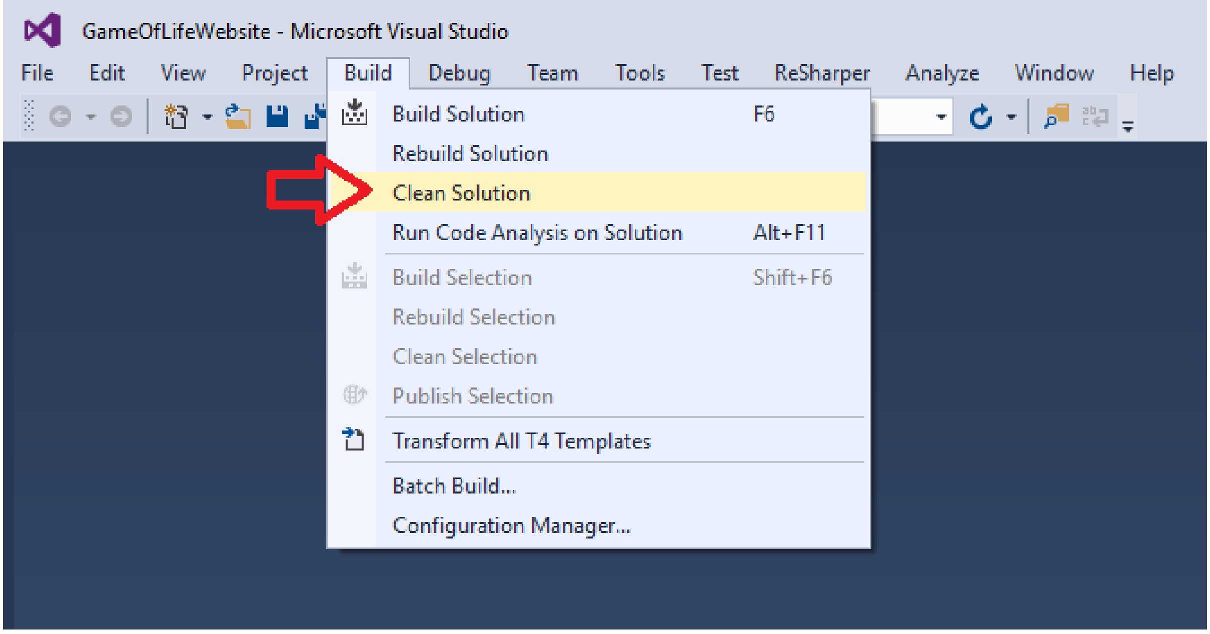
The source code for this assignment (i.e., the complete Visual Studio solution folder) should be submitted as a .zip file by the due date to the Blackboard system.

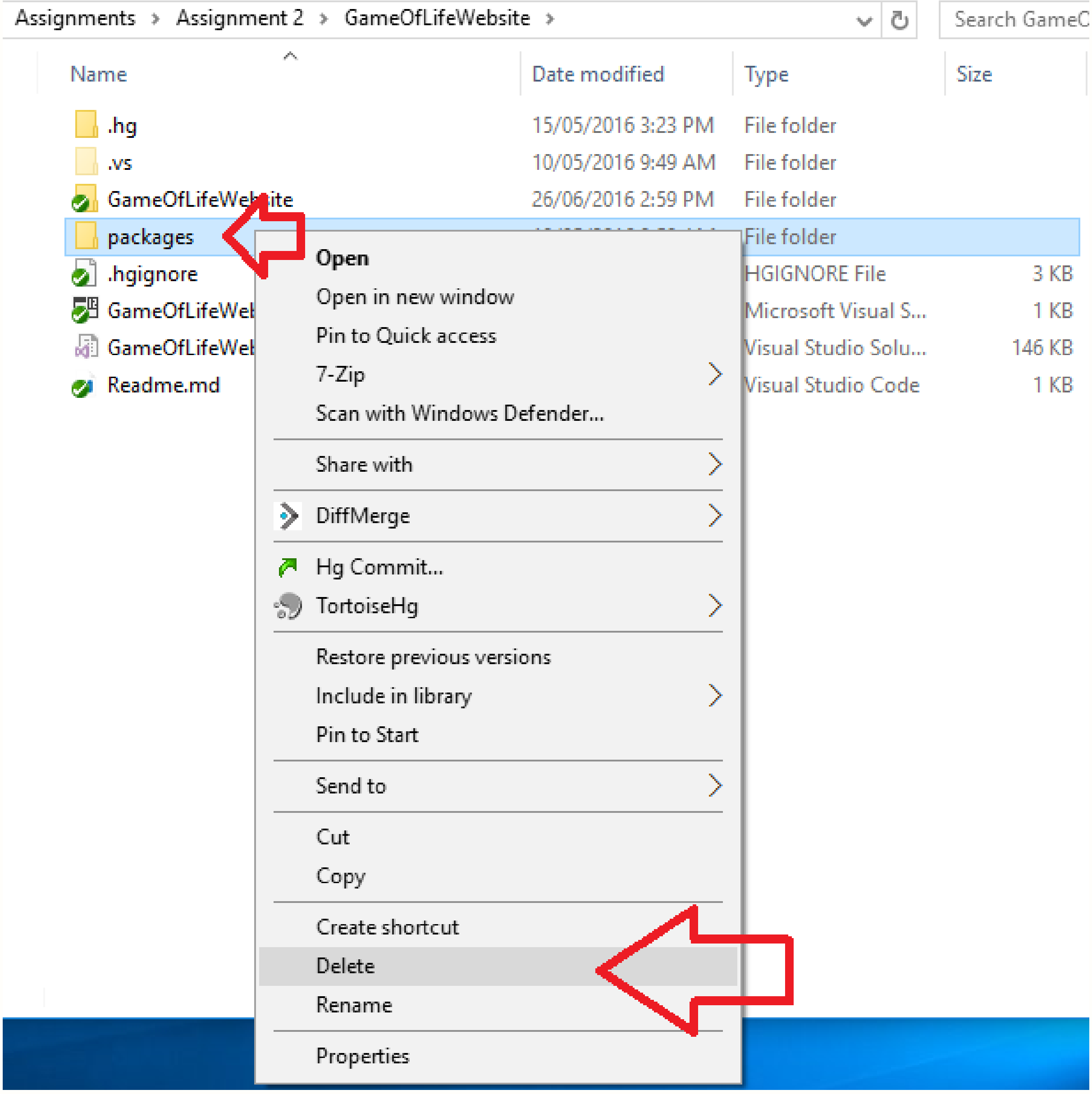
You may submit your assignment more than once; the last submission will be the one used for assessment review.

You should submit your assignment one week before the deadline as practise to make sure that you are familiar with the process and don't have any last minute issues with your final submission.

Submission Instructions

1. Before submitting please Clean your project first and then close Visual Studio and delete the packages folder. Here are some screenshots:

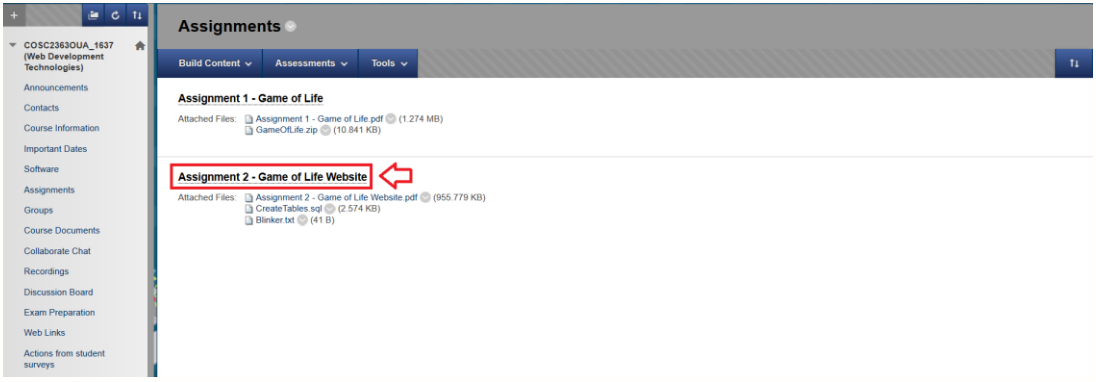




Name	Date modified	Type	Size
.hg	15/05/2016 3:23 PM	File folder	
.vs	10/05/2016 9:49 AM	File folder	
GameOfLifeWe ite	26/06/2016 2:59 PM	File folder	
packages		File folder	
.hgignore		HGIGNORE File	3 KB
GameOfLifeWel		Microsoft Visual S...	1 KB
GameOfLifeWel		Visual Studio Solu...	146 KB
Readme.md		Visual Studio Code	1 KB

Note this is only done to greatly reduce the size of your submission. As an example my project went from ~17.2 MB zip to ~1.8 MB zip.

2. Click on the **Assignment 2** title in Blackboard where you downloaded the assignment.



Assignment 1 - Game of Life

Attached Files: Assignment 1 - Game of Life.pdf (1.274 MB)
GameOfLife.zip (10.841 KB)

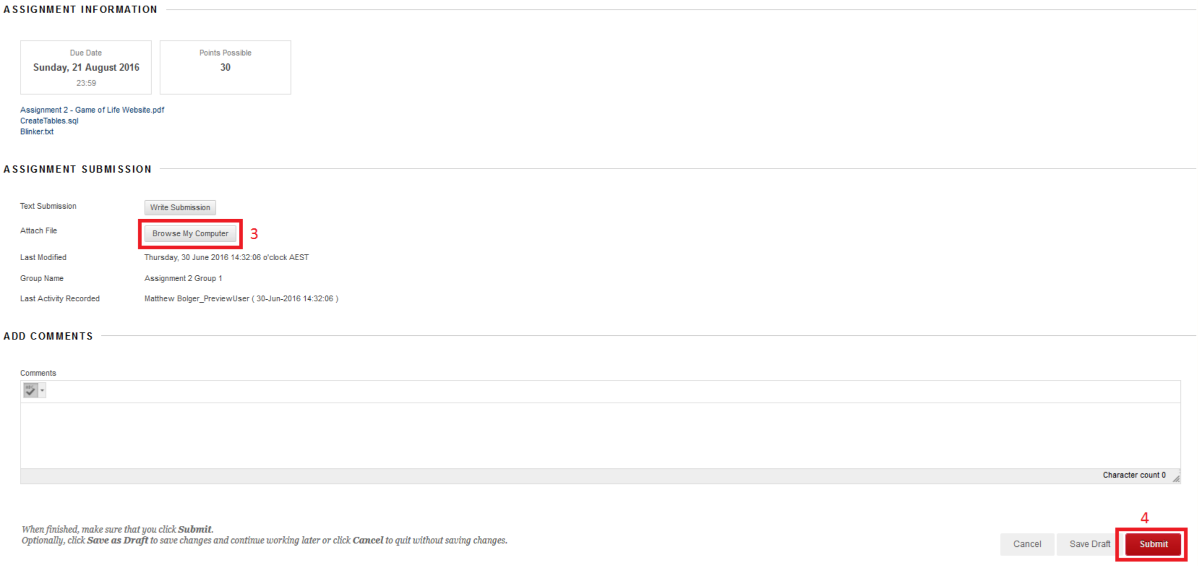
Assignment 2 - Game of Life Website

Attached Files: Assignment 2 - Game of Life Website.pdf (955.779 KB)
CreateTables.sql (2.574 KB)
Blinker.txt (41 B)

Note if you see a screen that shows the Submission History just press "Continue".

3. Zip your **Solution Directory** and attach zip file.

4. Click on the submit button.


ASSIGNMENT INFORMATION

Due Date	Points Possible
Sunday, 21 August 2016 23:59	30

Assignment 2 - Game of Life Website.pdf
CreateTables.sql
Blinker.txt

ASSIGNMENT SUBMISSION

Text Submission	Write Submission
Attach File	Browse My Computer 3
Last Modified	Thursday, 30 June 2016 14:32:06 o'clock AEST
Group Name	Assignment 2 Group 1
Last Activity Recorded	Matthew Bolger_PreviewUser (30-Jun-2016 14:32:06)

ADD COMMENTS

Comments

Character count 0

*When finished, make sure that you click **Submit**.*
*Optionally, click **Save as Draft** to save changes and continue working later or click **Cancel** to quit without saving changes.*