DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

Campus Map Project

CSCE 361 – Software Engineer Project

Cody Bodfield, Jiachun Han, Yan Xin Lee, Toan Nguyen, Matthew Shattil

6/26/2014
Version 2.0

This document describes a system design that will allow qualified users to upload pictures they took around campus, share them with other members via place markers on Google Maps.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document		2014/06/17
2.0	Sprint 2		2014/06/27

Table of Contents

1. Introduction

This is the Software Design Description of a Campus Map Project system for the University of Nebraska – Lincoln. It will allow qualified users to upload pictures on and around city campus, and share them with other members through place markers on Google Maps. This document outlines the technical design of the application that is being developed as a small social network of City Campus photography.

1.1 Purpose of this Document

The purpose of this document is to detail each iteration of the project. This includes successes, failures, testing methodology and implementation strategies. This document also outlines the system's MySQL database, ASP.NET web application, utilized APIs and backend web hosting implementation.

1.2 Scope of the Project

This ASP.NET web application is developed for qualified users to upload photos they take around city campus. It is designed as a small social network whose functions allow photo uploading, geo-tag plotting to Google Maps, commenting on photos and filtering of other users' uploaded photos. This project integrates the Google Maps API for placing photos on a campus map, uses the Twitter API as the login manager and utilizes the Imgur API for image uploading and persistence.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

BizLogic – A sub-project that encapsulates our application's primary classes and data access classes

ASP.NET – The server-side web application framework utilized in our web application

1.3.2 Abbreviations & Acronyms

API – Application Programming Interface

AWS – Amazon Web Services

ER Diagram – Entity-Relationship Diagram

IIS – Internet Information System

SQL – Structured Query Language

UML – Unified Modeling Language Diagram

XML – Extensible Markup Language

2. Overall Design Description

In keeping with the OOP paradigm for application development, the use of unique classes is essential. Relevant data, methods and functionality are built into classes based on the idea of encapsulation. The current primary classes are **Comment**, **Picture**, **Profile** and **Results**. Its data access classes include: **CommentData**, **PictureData** and **ProfileData**.

The **Results** class is a generic class. It is designed to push success or error messages from our data operations down to the user interface. The **Results** class holds (at minimum) a boolean for success or failure and a string message.

2.1 Alternative Design Options

Alternative design options considered in the development of this application:

- Implementing a MVC architecture instead of 3-Tier Client/Server
- Building project in Java EE instead of ASP.NET
- Implementing SQL database instead of MySQL database

3. Detailed Component Description

Classes are used to represent instances of given objects. Object creation is handled by constructor methods in the respective classes via provided data. The provided data values are encapsulated and belong to the class they were used to create. By default, all member variables of classes are set to private so that variable interfacing is handled by property methods contained with the parent classes.

3.1 Database Design

The ER Diagram presented in Figure 1 represents the application's MySQL database schema. This diagram denotes the primary tables with their respective fields and relationships to other tables in the schema. The database schema is loosely based on the VB.NET class entities presented in the application. Careful planning was done to enforce separation of distinct tables and ensure encapsulation of proper information into their respective, distinct tables.

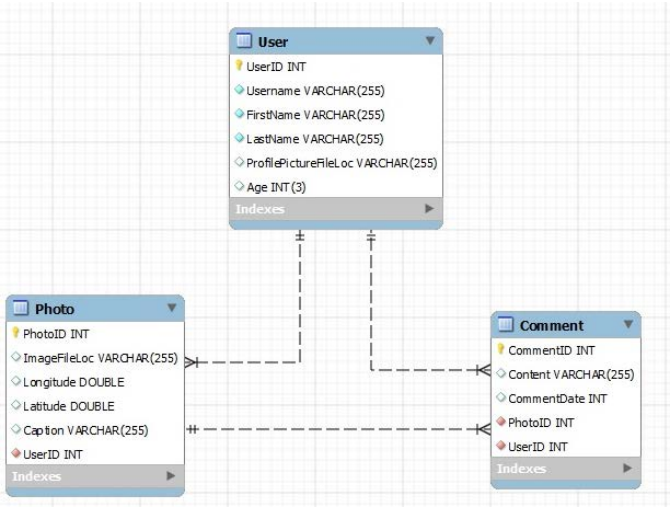

Figure 1: ER Diagram

3.1.1 Component Testing Strategy

Database testing to be implemented in future sprints.

3.2 Class/Entity Model

This part will be illustrated by a UML diagram in future sprints. Following is the current draft description.

There are four main classes **Results**, **Comment**, **Picture** and **Profile**.

The **Comment** class represents a user's comments on their photos and the photos of others. It consists of six variables: _sComment, _sPicture, _dTime, _sCommentID and _sUersID.

The **Picture** class is used to contain pictures users uploaded. It has variables of _sLongititude, _sLatitude, _sCaption, _sPictureID, _lComment (as list of String), _sUserID and _sImagePath.

The **Profile** class contains information about the individual user. It has variables of _sUername, _sFirstName, _sLastName, _sUersID, _nAge, _sProfilePicturePath, _lCommentList (as list of Comment), _lPictureList (as list of Picture).

The **Results** class allows us to push success or error messages from our operations down to the user.

3.2.1 Component Testing Strategy

Class component validation testing occurs during an ongoing basis during development. However, class component validation testing documentation will be implemented in future sprints.

3.3 Database Interface

Interfacing with the database is handled in our application's BizLogic sub-project. There are three data class object that correspond with their matching class object. The three data class objects are ProfileData, CommentData, and PictureData. These classes contain any database query (and helper methods) to the corresponding table in the database. By segmenting off the data access classes we are able to enforce proper class encapsulation. This allows us to better maintain like-purposed code in a single class. Calls to these data access class functions are implemented in the corresponding object class. To increase the robustness of our application, any data that is to be passed to the database is first validated. Doing so allows us to reduce errors in our system.

3.3.1 Component Testing Strategy

Database interface testing occurs during an ongoing basis during development. However database interface testing documentation will be implemented in future sprints.

4. Agile Development

Development of this web application is implemented using the Agile software development paradigm. The method of agile development chosen for this project is SCRUM. Our development timeline is divided into four sprints. The application is built incrementally, approximately 25% of project completion per sprint, with any requirements adaptions being implemented as needed between sprints. Backlog management is handled via Trello at https://trello.com/csce361groupproject . Source code management is handled via GitHub at https://github.com/CSCE361GroupProject .

4.1 Sprint 1

The initial sprint of this application development focused on the design of the project structure, division of tasks between team members, and projected backlog generation for each sprint. This sprint focused on the implementation of the project framework. This includes database design and implementation, project classes design and implementation, web page design, and backend hosting setup.

4.2 Sprint 2

The second sprint of this application development focused on adding more core functionality to the system. This included implementing data validation and database access. Additionally, logic was put in place for the correct flow of the website based on username login and whether that user exists in the system. This feature is a precursor and placeholder to the Twitter API implementation. The new login flow also includes loading user-specific data into their home profile page. Also implemented in this sprint was the Imgur API for photo upload and hosting. Implementing this API adds the core function to the application and enables work to be scheduled for completion in future sprints.

4.3 Changes & Refactoring

Sprint I – Initial design. No changes made.

Sprint II – No design changes of note.

5. Website Implementation

The backend hosting of the web application is handled on a dedicated Windows Server 2012 instance provided by Amazon Web Services. Deployment of the website application to the web server is completed via direct file transfer within Remote Desktop Connection. The dedicated Windows server's IIS manager handles server controls used within the web application. The web application can be accessed at http://54.88.28.177/Login.aspx.

6. API Implementation

6.1 Twitter API

Twitter API will be implemented in future sprints to manage application login, profile registration, and profile loading.

6.2 Google Maps API

The Google Maps API has been implemented to display and navigate a map of campus. In future sprints, work will be done to allow users to add and view pictures using this map.

6.3 Imgur API

The Imgur API will be implemented in future sprints to manage photo upload and hosting. The Imgur API was implemented in Sprint 2. The application utilizes an anonymous upload to the image hosting site. This upload implementation reduces the complexity of user authorization as it does not interface with any user accounts. The anonymous upload uploads the image and returns a url to the image. Our implementation uploads the photo via an ASP FileUpload control and saves the image url and any other requisite data to the database. To assist in our implementation of the Imgur API in ASP.NET we used an online tutorial from PC Tips as a reference.

7. Bibliography

Amazon Web Services. (2014). Retrieved June 2014, from Amazon: http://aws.amazon.com/

How to Use the Imgur API with VB.net and C#. (2014, January). Retrieved June 25, 2014, from PC Tips: http://pc-tips.net/imgur-api-vb-net/

Imgur API Version 3. (2014, June 4). Retrieved June 25, 2014, from Imgur: https://api.imgur.com/

MySQL Workbench Documentation. (2013, November 13). Retrieved October 2013, from MySQL: http://dev.mysql.com/doc/workbench/en/index.html