Spring 2015

Social Services Management Solution

Kurt A. Karner
Governors State University

James L. Potulny
Governors State University

Abstract

The modern privately run social services agency exists within a complex regulatory framework that requires detailed recordkeeping. In such an environment, paperwork tracking is best managed by a centralized enterprise data management system. While commercial applications that perform this function exist, they are frequently bundled with software for other business functions. Such additional software can be helpful, but it frequently duplicates the functionality of other enterprise software for these businesses, which is financially inefficient. This project developed a similar system using the ASP.NET 4.5 framework that is designed to provide a standalone solution to recordkeeping requirements. It followed a typical agile development cycle. Software requirements were gathered, design documents were drafted, and the application was developed. It was deployed to the Microsoft Azure cloud for testing. The results indicate that it could be a viable alternative to the bundled offerings available in the market with some additional development and tweaking for Microsoft Azure in future iterations.

Contents

Table of Figures

1. Introduction

Managing documents is vital for state-regulated organizations. Often, there are severe consequences if specific documents do not exist or are deficient in some way. As a result, management in organizations subject to strict regulations need to manage documents and other information in a way that is efficient, cost effective, and complete. Document management is extremely important in a heavily regulated industry, or in an organization whose funding is tied to state approval, as it is for community agencies that provide social services.

1.1 The Problem with Social Services and Information Management

In the social services industry, agencies must generate, organize, and maintain written documents that detail plans to assist the agency's clients in order for the community agency to maintain its funding. Social services caseworkers create many of these complex documents on a daily basis. These documents are created for several reasons. First, they are reviewed internally in order to ensure correctness and to ensure the best outcomes for clients. Second, caseworkers prepare them to ensure that outside parties, such as family members or guardians, can get a better understanding of how the client is progressing. Finally, regulatory requirements mandate that these documents and other records are preserved. State, and sometimes federal, entities need to review these documents in order to continue funding the community agency responsible for the client the documents concern.

The information used to create client plans is derived from many sources. Some information is derived from interactions and interviews with clients or their families, and other information is derived from legal, medical, or other official records in many different formats. Some information is taken from previous planning documents and updated to reflect changes in the client's life. All of this information needs to be confidential, but it also needs to be

accessible by caseworkers, their supervisors, and ultimately by auditors and it must be organized according to state regulations.

Based on the need to manage so much confidential information, a centralized enterprise resource planning (ERP) system would be an obvious solution. Caseworkers would benefit from a centralized and secure system because it would allow them to organize and work with this information in a logical and consistent way that is compliant with the relevant legal regulatory framework. Supervisors would benefit as well because they could assess the performance of caseworkers under them and ensure all legally required paperwork has been completed by its deadline. A system like this should lead to better productivity in community agencies because staff members would have faster access to data when compared to paper charts. Additionally, the data would be consistently organized, making staff transitions painless when caseloads change.

1.2 Review of Work on this Problem

ERP software solutions are not a novel idea, even in the not-for-profit business world. Broadly speaking, ERP software is used to solve the management of information in a business. It is "…core software used by companies and to coordinate information in every area of business" (Monk & Wagner, 2009 p. 1). Most solutions are comprehensive by necessity and generalized to support standard business functions such as maintaining inventory, bookkeeping, managing order fulfillment, and other related functions. Complete ERP solutions are generally separated into discrete components to handle the aforementioned functions and they are bundled together into a

single product. Singly packaged products exist, although they are narrowly tailored to handle problems that are atypical for ordinary businesses (e.g., Raiser's Edge) [1].

Although comprehensive and cohesive ERP solutions have long been seen as a way to improve information flows throughout a business (Davenport, 2000), they are also fraught with many issues. The return on investment for this type of software in most business cases is difficult to predict and is frequently misrepresented in order to secure additional funding to move the project forward (Ward et. al., 2009). Nevertheless, because information management in the social services industry is so complex, a system of some kind must be used. A small portion of the software industry has taken note.

Realizing the inadequacy of commercially available ERP solutions for non-profit companies generally, and for community social services agencies specifically, some software firms have developed solutions tailored to meet the needs of these and other non-profits (for example, Therap[2] or Medisked[3]). As with most other ERP solutions, these products are comprehensive and offer coverage for all aspects of the non-profit entity's business including comprehensive support for payroll, human resources, fundraising, and other business functions. These functions are generally bundled in to the base software package even if the licensing agency has no interest or need to use them.

1.3 Purpose of the Project

Mandatory bundling of additional business software into the ERP unnecessarily raises the cost of the license. After all, a community agency does not need to purchase duplicate software

[1] See http://www.blackbaud.com for additional information on the Raiser's Edge line of products. Briefly, Raiser's Edge and its derivations are a group of software fundraising tools for nonprofit organizations. These tools are designed to facilitate and manage fundraising through donations, grants, and other charitable contributions that for-profit organizations do not and, in some cases, cannot use. As such, it is offered as a standalone product instead of as a comprehensive ERP solution.

[2] See generally http://www.therapservices.net (last accessed 5/2/2015) (describing a comprehensive ERP solution)

[3] See generally http://www.medisked.com (last accessed 5/2/2015) (describing a comprehensive ERP solution)

if it is already paying for payroll software, human resources software, and other similar programs. Accordingly, a standalone software product that is designed to do a single function and do it well can be much more valuable to a community agency. This is especially true when one remembers that the budgets of many nonprofits are constrained either by limited donations or limited funding from the government.

With the foregoing in mind, this project aims to develop a system that assists caseworkers and their supervisors in non-profit human services organizations. Its scope will be initially limited to Illinois Department of Human Services cases for developmentally and intellectually disabled adults. However, the entire software system will be modular and will allow for swapping in new rules for different social services and rules for other states and agencies. The software's primary goal is to allow caseworkers to manage vital documents, case document information, and case contacts. It will also allow a supervisor to check the progress of each individual caseworker and ensure that all regulatory requirements are met.

2. Techniques, Steps, and Results

This project used an agile software development methodology in order to develop and revise the software. Due to time constraints, only a single iteration was completed. Subsequent iterations will produce a superior product.

2.1 Requirements Gathering and Initial Design

Informal inquiries with local community social service agencies have determined that they require extensive document and data tracking abilities. They would also need the ability to perform assessments and incorporate that data into a database. In the context of intellectually disabled adult clients, caseworkers would also need a means to monitor client progress as well as whether staff were trained on the learning plans for individual clients since these trainings must

be kept current. Additionally, the ability for caseworkers to take notes and document contact with clients, guardians, and state agencies is highly desired.

From a supervisory perspective, these agencies are interested in being able to ensure that all work for a case is completed on time in order to meet regulatory requirements. The supervisor must also be able quickly assess any caseworker who is behind on their work and identify any upcoming and missed deadlines. Therefore, a simple and clean user experience is a high priority for supervisors. For for agencies as a whole, and for supervisors specifically, information security is a very high priority due to the highly confidential nature of the work they do.

After the requirements were determined, a design document was drafted. This document described the major components of the program and detailed the various technology decisions that were made in response to the perceived requirements. Per industry best practices, it detailed both architectural and low-level design specifications. All functionality and data structures were documented. After the design document was completed, it served as a blueprint for the code that followed throughout the software construction process. A version control system (Git) was selected and a remote repository was generated to manage the code (Github.com). Code construction followed and the application was tested.

2.2 Technologies Used

The program relies on .NET technology and uses Microsoft Azure as a hosting provider for both the executables and for the databases. The design warehouses data in the cloud on the Azure servers, but organizations could implement a data storage solution locally in order to ensure security in a hybrid-cloud or standalone configuration. The program presents data in a tasteful manner using ASP.NET technology with a responsive design to support mobile access.

It also incorporates other modern web technologies such as AJAX and jQuery in order to make data access quicker and more seamless.

More specifically, the web pages are ASP.NET web forms. Although the ASP.NET MVC model is preferred in the industry, web forms were a better alternative given the short timeframe for the project since they allow for faster development and deployment. They are backed with code-behind files written in C# that handle the dynamic aspects of the program and serve as a data access layer to the relational database management system (RDBMS). The back end is based on Microsoft's SQL Server and uses Transactional SQL to retrieve and process data. It was tested on Microsoft SQL Server 2012 and then migrated to Microsoft Azure SQL. Much of the data manipulation is handled by the Azure SQL server and not within the code, which should address any maintenance concerns regarding the usage of ASP.NET web forms. Session is used to manage information flow within the program and to provide statefulness throughout it. Responsive design is ensured due to reliance on the Bootstrap 3.0 CSS library, which ensures a mobile first approach, and on jQuery.

For additional information, please refer to the Appendix, which provides a detailed overview of the technology and design of this particular project. The code for this paper is available in electronic format either on disk or in its most up-to-date format in a Github repository, located at https://github.com/kkarner708/Social-Services.git. Both sources contain the application source code, the ASP markup, and T-SQL scripts that define the database and its related functions, triggers, and procedures.

3. Conclusions and Avenues for Future Work

The application will simplify the process of managing caseworker documents in a a community agency. It will promote accountability by ensuring that supervisors can review the

work of their caseworkers at any time to confirm it is up to standards. It will promote efficiency, as caseworkers now have a tool that will track when their various compliance documents are due. Previously, this was a task that may have been done on pen and paper. The software can also prompt caseworkers when they are approaching due dates, provide all relevant case information with a button click, and perform client evaluations through the web interface. Finally, it serves as a tool for identifying problems with agency information in order to prepare for audits. As a result, it is capable of providing many of the benefits for the core business functions of a community agency that commercially available bundled ERP solutions can provide but without the additional (and frequently duplicative) business functionality.

While the application was well designed, conformed to the requirements of community agencies, and was constructed to meet those requirements, additional work is necessary. The software could benefit from a more modern look and feel. It is already cleanly presented and simple, but it lacks the visual polish that many modern web applications possess. Additionally, more functionality could be added to provide more value to the end users who operate in different regulatory frameworks. This would ideally come in the form of a pluggable interface that allows the end-user to deploy additional modules to enhance functionality instead of adding to the core of the application monolithically. Finally, better interoperability with existing common ERP solutions for general business functions could be a helpful extension to the project. This would allow for the program to be a true standalone extension to existing software, instead of a monolithic standalone addition to the enterprise that attempts to replace existing business software across a company.

4. References

Davenport, T. (2000). *Mission critical realizing the promise of enterprise systems*. Boston, MA: Harvard Business School Press.

Monk, E., & Wagner, B. (2009). *Concepts in enterprise resource planning* (3rd ed., p. 1). Course Technology Cengage Learning.

Ward, J., Daniel, E., & Peppard, J. (2008). Building Better Business Cases for IT Investments. *MIS Quarterly Executive*, *7*(1), 1-15.

Appendix: Software Design Documentation

Enterprise management software is a fact of life for many for-profit businesses across all sectors of the economy. In the not-for-profit sector – and particularly in social services – high quality enterprise resource planning software (ERP) is the exception and not the rule. This is due to many different factors, not the least of which is the relative low level of funding available to these types of charitable organizations versus the relatively high cost of developing a comprehensive business management solution. However, the nature of the regulatory framework surrounding the administration of social services in many states is extremely complex and dictates the use of some kind of management software. The situation leaves smaller agencies in a bind due to the aforementioned funding and software availability constraints. This project aims to resolve this problem by developing a lightweight and modular solution to manage caseloads of social service agency caseworkers.

The product created by this project is called Social Services Management Solution (SSMS). It is a lightweight web application designed for privately run state funded social services agencies. It is specifically intended to help ensure data consistency throughout an agency by acting as an organizational tool for case management. The implementation this document describes is specific to case management for cases linked to the Illinois Department of Human Services Division of Developmental Disabilities, but the application could be extended to use additional modules for other states or agencies.

It will be developed with .NET technology because that technology integrates well with Microsoft software systems, which predominate in many small to mid-sized privately run state funded social service agencies. SSMS will be suitable for on-site deployments, cloud-based deployments, or hybrid deployments. For larger agencies with a mature IT infrastructure in

place, on-site or hybrid cloud deployment makes sense and it enhances the security and level of control that the business would have over the internals of the application. For smaller companies a cloud based solution may make more sense since these organizations may lack the hardware, technical knowledge, or sophistication to deploy such a solution on their own.

This software is not intended or designed to be a comprehensive ERP solution. It will not include business management software like payroll management, human resources management, accounting, or similar multi-domain management tools. The decision for excluding those features is simple – most businesses looking for a solution to their case-management problem are looking for a narrow solution. They have probably already solved their payroll, human resources, accounting, or other business function issues so there is no need to package them with this product.[4] As a result, SSMS is leaner and more narrowly tailored to solve specific solutions an agency might face.

A1. Design Considerations

In order to effectively design this software, a number of very different concerns must be balanced. Although SSMS attempts to be lightweight and easily used in scope, it is built on a heavyweight back-end (.NET). Accordingly, certain assumptions are made and constraints are imposed upon the design.

A1.1 Design Assumptions

For the business that deploys this software locally or uses a hybrid-cloud based approach, this project makes a number of simplifying technical assumptions:

[4] But compare the software produced by www.therapservices.net (illustrating an alternative "heavy-weight" solution that incorporates additional business functions into management software).

- The consumer is using Microsoft products for its enterprise infrastructure solution (e.g., Active Directory, Windows workstations, Internet Information Server (IIS) or SharePoint Services) or has these products available to them.
- The consumer has ASP.NET 4.0 or higher installed on their internal or external webserver
- The end user has deployed hardware throughout their business that allows internet-based case management software to be accessed either on an intranet or over the internet.
- The business has sufficiently sophisticated IT personnel to troubleshoot some of the back end components of the system (e.g., SQL Server, IIS, RESTful web services)

Recently, Microsoft announced that the .NET framework was being released as an open source product. This may result in some variability and inconsistency in different versions of the framework in the future. As a result, this project assumes that the framework of previous releases (up to ASP.NET 5) will be used for the software, and that additional open-source variations of .NET will not significantly change the portions of ASP.NET that the project relies on. It is unclear whether this assumption is realistic, and the software will need reevaluation upon any substantial .NET upgrade by the open source community to ensure it is still correct.

A1.2 Design Constraints

Due to the short project timeframe, a number of constraints have been imposed upon the final work. This is necessary because a fifteen week semester is inadequate to develop a fully functional enterprise management tool for two programmers who have other academic and professional obligations. As a result, the application is limited to viewing and editing data with limited assessment capabilities. In the initial release, a single short assessment will be included

to demonstrate the feature; however, a large number of other assessments exist and will be incorporated if the project continues.

Additionally, some natural constraints exist. In particular, because the regulatory framework in each state is extremely complex and no two frameworks are identical, writing software for a national audience is difficult. Accordingly, this software focuses on one state's rules and its social services system (Illinois). These constraints exist because every state has different rules for case management. Indeed, different state agencies within the same state have different rules for case management. For this reason, it is limited to case management tools for social services provided to people suffering from developmental disabilities.

A1.3 System Environment

The software will run in a Microsoft enterprise environment. Specifically, the software should be running on nothing less than Windows 2008 Server, as that is still supported. It is anticipated that the software will run in a desktop environment managed by Active Directory, but this particular implementation will not support LDAP authentication for users. The software will need to interoperate with a relational database as well. This implementation uses Microsoft SQL Server, but with only slight modifications additional implementations are possible due to the design of the database access classes.

Finally, the user interface is accessible exclusively through a web browser. The system must be able to interact with all modern web browsers including, but not limited to, Internet Explorer, Mozilla Firefox, Google Chrome, and Apple's Safari browser. Primarily, this interaction will be handled through standard HTML forms and controls and some JavaScript code (primarily jQuery). AJAX usage will be limited because there are simply not many reasons to asynchronously update the UI.

A2 – Architectural Design

At the highest level, the system consists of a user interface, a database, some business logic, and data transfer classes. The user interface is based on Microsoft's ASP.NET Web Forms technology. The database can theoretically be any relational database, but this project will use Microsoft Azure SQL when deployed, and it will use SQL Server 2012 for testing. The business logic is written in C# and is mostly contained in code-behind files of the ASP pages. Some additional business logic is located in other classes as needed. The data transfer classes comprise both the abstract and concrete database access objects and use multiple model classes representing entities within the program.

The user interface is implemented through Microsoft ASP.NET and uses Web Forms. This approach was chosen over the more popular MVC model because Web Forms allow for swifter construction of content pages. While there is a tradeoff with maintainability and scalability, this is a superior approach for this project due to the constrained timeframe. Moreover, additional maintenance costs can be lessened by moving more of the application code to the SQL server, since the bulk of those issues come from the fact that code-behind files are naturally monolithic and tightly coupled to the user interface.

A2.1 Rationale for Design

The software as designed for this project will support Microsoft SQL Server 2012 and Microsoft Azure SQL. Theoretically, any relational database can be used due to the structure of the data access classes. SQL Server 2012 is an expensive solution and may not be the best solution for a cash-strapped agency looking for an on-site deployment. However Microsoft's Azure cloud supports T-SQL and provides a reasonably priced solution. Both will be supported to accommodate all types of consumers.

The business logic is programmed in C#. Since this is Microsoft's object-oriented .NET language, and because ASP.NET natively supports it, it makes the most sense to use when designing an enterprise application. The logic is stored in the code-behind files of each ASPX page, which means that it is logically linked to the page it is supporting and keeps things organized. The maintenance costs are mitigated, as stated before, by moving as much data handling code as possible to the SQL server.

Additionally, the data access classes and model classes are also programmed with C#. A data access object is easy to implement in C#, and plain old common-language-runtime objects (POCOs) are frequently used as model classes for persistence. This approach is perfectly reasonable as long as the POCO model classes are kept in separate assemblies outside of the code-behind files. Additionally, data access objects mentioned here will allow for looser coupling between the database and the business logic and code-behind files, which allows for the project to switch databases if needed relatively quickly.

A2.2 Conceptual and Logical View

SSMS will be set up like most every ASP.NET application. The web server, whether it is IIS or a cloud based server, receives a request from the end user. The server determines if it is a response for an ASP.NET web application and, if so, it forwards it on to the container. At this point, the ASP.NET security framework handles the request based on cookies and request state to determine whether the user is authenticated. If not, it redirects them to a login page. If they are authenticated, or they are seeking a non-privileged page, the ASPX file that defines the page is located and, if necessary, compiled.

The page uses the code-behind file to make database calls and other requests for system resources. If a database call is made, it retrieves the context for the database from the data access

classes. It then sends the request to run a stored procedure, and the database will respond by running it. The result is returned to the appropriate code-behind, which in turn passes the result set to the POCO model classes. The model class contains a constructor that will take the result set and populate its fields. If the database call fails, it will be handled in an exception block in a manner that is appropriate for the specific call.

Finally, after the data is retrieved (or the retrieval fails), the web forms page is returned by the ASP.NET engine and sent from the application container back to the web server. At this point, the web server sends an HTTP response to the client making the initial request and the user should have the appropriate page sent to them. See the figure below for an illustration of the entire process.

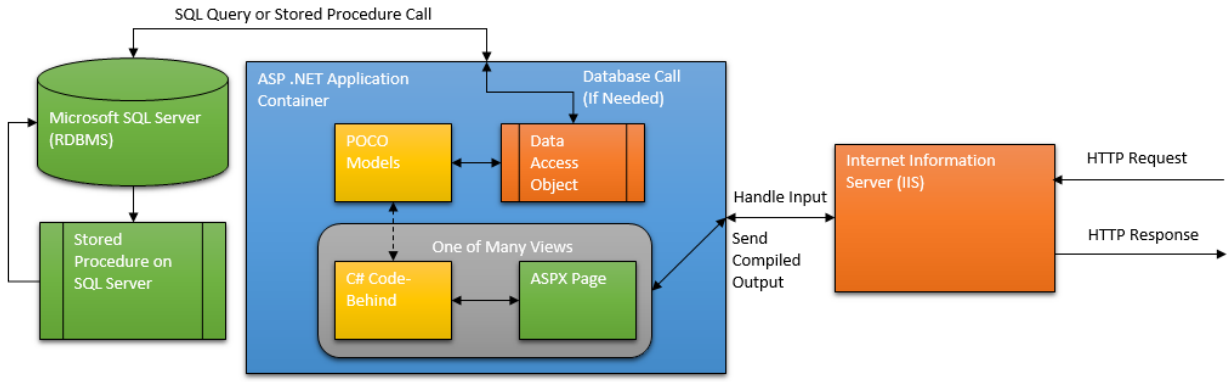

Figure A1 - conceptual diagram of the high level architecture of this project

A3 Low Level Design

The previous discussion gave a broad overview of the high level concepts involved in the design of this application. The following sections discuss each component of the application in greater detail.

A3.1 Data Access Objects and Models

The data access objects (DAO) mediate interactions between model classes, business logic, and the database itself. The DAO primarily serves as a central configuration class for database access. It contains references to the connection strings, authentication credentials, and other information needed to access the database. It also implements any special code that might be needed to ensure a stable connection. It returns a connection object to a database when called from one of the business logic classes.

Each model is linked to information in the database, but the actual calls are made from the code-behind assemblies or elsewhere in the business logic. The models are represented by POCO classes that can be initialized through their getter and setter methods. The POCO classes are defined separately from the code-behind files in order to ensure a clean separation of concerns. Below is a detailed discussion of each POCO model class.

A3.2 Models

Person. This class contains a number of fields related to the identity of a person. It is directly linked to the Person table in the database. Its fields are:

Type	Name	Description
int	id	Corresponds to database primary key
string	firstName	The Person's first name
string	lastName	The Person's last name
string	address	The Person's street address
string	address2	The second line of the person's street address
string	city	The city associated with the person
string	state	The state in which the city exists

string	zip	The zip code of the Person's residence

All fields have associated getter and setter methods. Fields are marked protected, since other classes will inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes.

Case. Case mostly contains other POCO model classes since serves mostly as a means of linking entities together. In particular it contains the following:

Type	**Name**	**Description**
int	id	Corresponds to database primary key
CaseWorker	caseWorker	Identifies the case worker
CaseWorker	supervisor	The person supervising the case
ArrayList<Person>	contactList	A list of contacts identified with the case
ArrayList<Document>	documentList	A list of documents associated with the case
Client	client	Identifies the client
Datetime	LastAnnualPhysical	The last time the subject of this case went for an annual physical examination
Datetime	LastAnnualDental	The last time the subject of this case went for an annual dental screening
Datetime	LastAnnualVision	The last time the subject of this case went for an annual vision screening
bool	isClosed	Determines if a case is open or closed
String	reason	The reason why a case was closed, if it is closed

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes.

It contains the following additional methods:

- public void CloseCase() – marks a case closed without giving a reason
- public void CloseCase(string reason) – marks a case closed with a reason given
- public bool ReopenCase() – reopens a closed case. Returns true if the operation succeeded, false if it did not

Client. Client is a POCO model class that aggregates information about a client and passes it on to the business logic of the program. It inherits from Person, and so Person's fields are omitted below. It contains the following fields:

Type	Name	Description
int	id	Corresponds to database primary key
ProgramSite	residence	Corresponds to a client's residence
ProgramSite	program	Corresponds to a client's treatment, vocational, or other program

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes.

Caseworker. CaseWorker is a POCO model class that aggregates information about a case worker and passes it on to the business logic of the program. It inherits from Person, and so Person's fields are omitted below. It contains the following fields:

Type	Name	Description
int	id	Corresponds to database primary key
string	title	The job title for this particular case worker
Department	department	Identifies the department the case worker works in

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes.

Since this is a POCO model class, there are no additional methods that encapsulate business logic. Certain methods in the code-behind assemblies contain methods that take CaseWorker as an argument in order to assign it to cases.

Document. Document is a POCO model class that describes a statutorily required document and passes it on to the business logic of the program. It contains the following fields:

Type	Name	Description
Int	id	Corresponds to database primary key
string	documentName	Identifies the document by name
String	documentPurpose	Identifies the purpose of the document
Bool	isScorable	Indicates whether a document can be scored

Int	documentScore	If isScorable is true, this field will be used as either the score or as a reference to the score if the Document class, along with the database, are extended

All fields have associated getter and setter methods. Fields are marked protected, since other classes may inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes.

Since this is a POCO model class, there are no additional methods that encapsulate business logic. Certain methods in the code-behind assemblies contain methods that take Document as an argument in order to assign it to cases.

PersonalContact. PersonalContact is a POCO model class that aggregates information about a case contact and passes it on to the business logic of the program. It inherits from Person, and so Person's fields are omitted below. It contains the following fields:

Type	**Name**	**Description**
int	id	Corresponds to database primary key
string	relationship	Defines the relationship between the contact and the case
Datetime	knownSince	Defines when the contact first became involved in the case

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes. Since this is a POCO model class, there are no additional methods that encapsulate business logic. Certain methods in the code-

behind assemblies contain methods that take PersonalContact as an argument in order to assign it to cases.

Department. Department is a POCO model class that aggregates information about the organizational unit that a case is managed by and passes it on to the business logic of the program. It contains the following fields:

Type	Name	Description
Int	Id	Corresponds to database primary key
String	Name	Gives a logical description of the department, if required
String	Scope	Describes the responsibilities of a department, if required

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class. Since this is a POCO model class, there are no additional methods that encapsulate business logic. Certain methods in the code-behind assemblies, to be discussed later, contain methods that take Department as an argument.

Please note, as discussed below, there are limited use cases for this class. It is provided mainly for a certain specific type of organization. For smaller organizations, this class can be ignored.

ProgramSite. ProgramSite is a POCO model class that aggregates information about specific residential or treatment locations where clients might spend their time and passes it on to the business logic of the program. It contains the following fields:

Type	Name	Description
Int	id	Corresponds to database primary key

	name	Logical name for the site
	progType	String value field that accepts a limited range of values, as described in section 3.1
Int	detail	Optional field. Used when progType is not "Residential" to provide additional detail about the program's location.

All fields have associated getter and setter methods. Fields are marked private, since no other classes will inherit from this class. It utilizes the DAO factory class in order to retrieve an appropriate database access class for persistence purposes. Since this is a POCO model class, there are no additional methods that encapsulate business logic. Certain methods in the code-behind assemblies, contain methods that take ProgramSite as an argument.

A4 Database Design

The database is at the core of SSMS. It is normalized in order to reduce redundancy and consists of a number of tables. The diagram below gives a general overview of the relationships in the database.

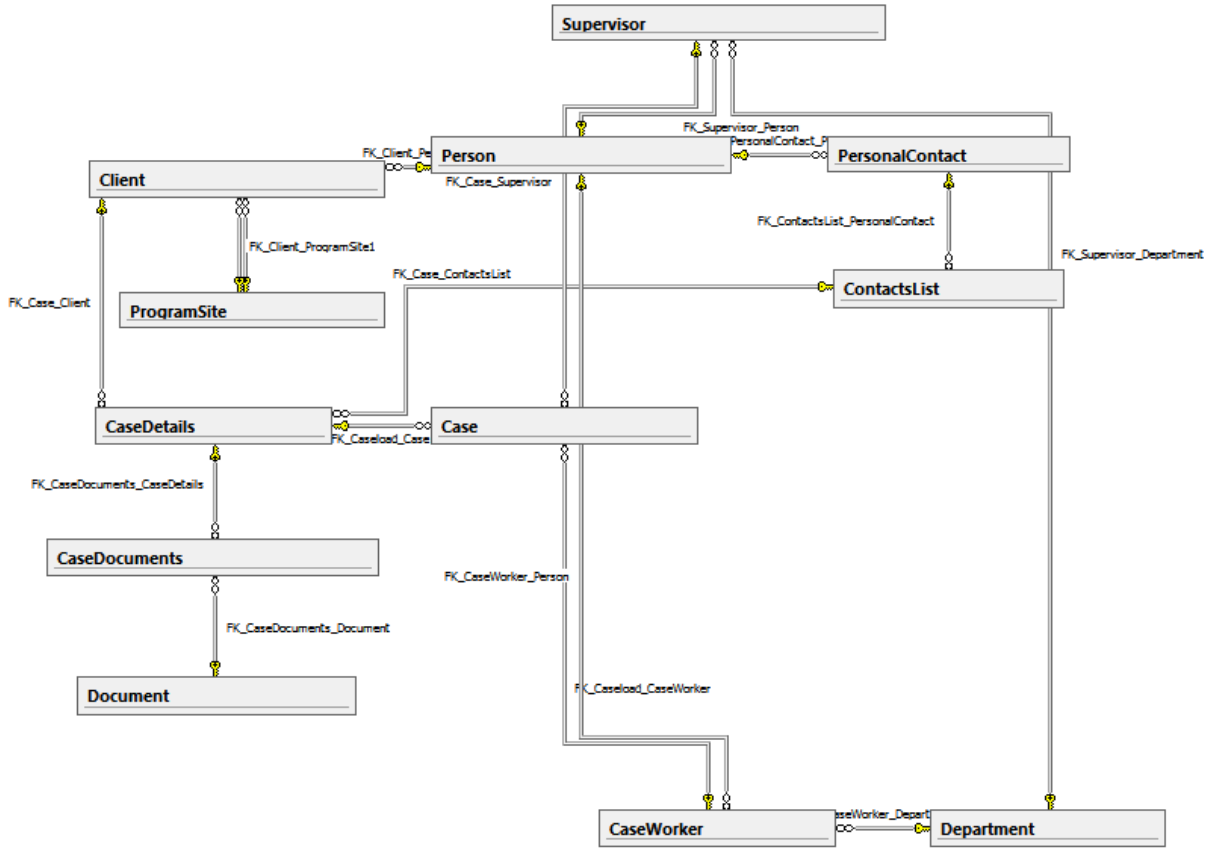

Figure A2- Database ERD for SSMS

A4.1 Table Discussion

The database contains twelve tables and man different joins. As a result it is quite complex. Below is a detailed discussion of each table in the database.

Person

Person		
Primary Key	ID	Int, Auto_Increment
	FirstName	varchar(20)
	LastName	varchar(20)
	Address	varchar(50)

	Address2	varchar(50)
	City	varchar(20)
	State	nchar(2)
	Zip	varchar(15)

The Person table is linked to by many tables. It is a generic table containing contact information for any given person within the program. As a result, case workers, supervisors, clients, and other contacts all link to this table for additional detail. Although this introduces an additional layer of complex linking, it allows for a more flexible future table design because new types of people can be introduced without significantly changing the underlying structure of the database.

The primary key is ID, which CaseWorker, Supervisor, Client, and PersonalContact all link to via foreign keys. The remaining fields are self-explanatory. FirstName and LastName are the person's name, Address is the first line of their street address, and Address2 is the rarely-used second line of their address. City, State, and Zip all relate to their concrete analogues in the world.

Case

Case		
Primary Key	ID	int, auto_increment
Foreign Key	CaseWorkerReference	int, not_null
Foreign Key	CaseReference	int, not_null
Foreign Key	CaseSupervisor	int, not_null
	isClosed	bit
	Reason	varchar(256)

This is primarily a linking table. It establishes a relation between the case worker, the case details and the case supervisor for any given case. ID is the primary key, and the remaining fields are integer-valued foreign keys that are keyed to their corresponding tables. CaseWorkerReference is keyed to CaseWorker, CaseSupervisor is keyed to Supervisor, and CaseReference is keyed to CaseDetails. This models the complex many-to-many relations that can exist in a case management system like this in the most efficient way possible.

It also contains two additional value fields. The isClosed field contains a bit-value that determines whether or not the case is open or closed based on whether the bit is marked true or false. Reason contains a short explanation of why the case was closed, if it is closed.

Client

Client		
Primary Key	ID	int, auto_increment
Foreign Key	PersonReference	not null
Foreign Key	Program	not null
Foreign Key	Residence	not null
	ClientPhoto	Image

The Client table describes a client of the privately run social services agency. It contains a number of foreign keys in order to flesh out the details of the client. Its primary key is ID and is integer-valued. The PersonReference key is a foreign key to the Person table and contains demographic information about the client in question. Program is a foreign key to ProgramSite, which is where the person receives treatment, employment services, or developmental training

during the day. Residence also links to ProgramSite and determines the identity of the person's reference.

Finally, a field called ClientPhoto exists in the database. The development team decided that it would be more efficient to use binary large objects in order to store photographs of clients instead of relying on references to locations on the server file system. This prevents accidental changes on the file system that could corrupt or otherwise damage the photos. This is largely a guard against user error.

Caseworker

CaseWorker		
Primary Key	ID	int, auto_increment
Foreign Key	PersonReference	int, not_null
	Title	varchar(15)
Foreign Key	Department	int, not_null

CaseWorker describes a person who oversees a case. Its primary key is ID, and it contains integer-valued foreign keys that link to Person (via PersonReference) and Department (via Department). It also contains a field named title which describes the formal title of the case worker. This is necessary because different types of caseworkers have different titles, and some may not be legally qualified to oversee certain types of cases. This varchar field should allow a software user – specifically a supervisor – to filter against to ensure legal compliance and a higher quality of service.

CaseDetails

CaseDetails		
Primary Key	ID	int, auto_increment

Foreign Key	ContactListIndex	int, not_null
Foreign Key	Client	int, not_null
	LastAnnualPhysical	Datetime
	LastAnnualDental	Datetime
	LastAnnualVision	Datetime
Foreign Key	DocumentListIndex	Int

CaseDetails contains specific det ails about the case that a case worker oversees. Its primary key is ID. ContactListIndex links to the list of personal contacts that may be involved in a case (e.g., relatives, guardians, doctors, state officials, etc). Client is a foreign key that links to the table Client and exists to establish a relation between the details of the case and the client. Additionally, there is a foreign key DocumentListIndex. This links to CaseDocuments, which is a linking table.

Finally, the three datetime fields correspond to common requirements in managed care settings – namely, they record the last time the person had a physical, a dental screening, and a vision screening. This data is critical in Illinois, where various state agencies can penalize a private social service agency heavily for not being within the one year period for each of these.

CaseDocuments

CaseDocuments		
Primary Key	ID	int, auto_increment
Foreign Key	DocumentReference	Int
Foreign Key	CaseReference	Int

This is a linking table. It establishes a relationship between a case and the many documents that are involved in that case. It contains a foreign key DocumentReference, which establishes a relation with an individual document that a regulatory agency requires for compliance. It also contains CaseReference, which is a foreign key to the case that the document is related to.

Document

Document		
Primary Key	ID	int, auto_increment
	DocumentName	varchar(50)
	DocumentPurpose	varchar(50)
	IsScorable	Bit
	DocumentScore	Int

This table contains information about individual statutorily required documents. It is populated on a case-by-case basis with information about the name of the required document, its purpose, and whether it can be scored. The bit field IsScorable is used to indicate whether the document can be scored and acts as a way to filter the data. The DocumentScore field contains an integer-valued score that can be used as a standalone score itself, or it can be used as a foreign key to a more complicated scoring system that would be defined by a separate module.

ContactList

ContactsList		
Primary Key	ID	int, auto_increment
Foreign Key	ContactReference	int
Foreign Key	CaseReference	int

This table is another linking table. It models the collection that exists when a case has multiple contacts, and where those contacts also appear in other cases. As such, it has a primary key ID and two foreign keys. ContactReference is a foreign key to PersonalContact. CaseReference is a foreign key to CaseDetails

PersonalContact

Personal Contact		
Primary Key	ID	int, auto_increment
	Relationship	varchar(50)
	KnownSince	datetime
Foreign Key	PersonReference	int, not_null

This table contains information about a person involved in a case. It therefore contains a foreign key reference to Person via the field PersonReference. It also contains two data fields. Relationship describes the relationship of the person to the case. KnownSince is another value field that indicates when a personal contact became involved in the case. Its primary key is ID.

Department

Department		
Primary Key	ID	int, auto_increment
	DepartmentName	varchar(50)
	Scope	varchar(50)

This table contains a number of values related to where caseworkers or supervisors work. It can also be used as a logical filter that permits a user to break up different types of case loads. The primary key is ID. DepartmentName is the logical name of the department within the organization. For small organizations, this can be safely ignored; however, more complex

organizational structures will need to rely on this field in order to ensure that everything is organized correctly. Scope is a second value field that is useful for larger organizations. It is used to define a department's area of responsibilities. Again, this is mostly useful for larger organizations who might have several different departments that do several unrelated things.

ProgramSite

ProgramSite		
Primary Key	ID	Int, Auto_Increment
	SiteName	varchar(50)
	SiteType	varchar(50)
Foreign Key	SiteDetail	Int

This table is a logical identifier for programs. They are distinguished on the SiteType field. For this particular implementation of the software, there would be only a few possible values that SiteType can contain – "Residential," "Treatment," "Employment Services," and "Developmental Training." This would be enforced by a check constraint on the table. SiteName is merely the logical name of the site.

For residential sites, it is unnecessary to introduce additional detail into the table since the person who is linked to the site will already have the address of the site associated with them. This table just provides the name and a means of organizing and filtering the data in that case. However, for other types of sites, there is no such association and so there is an optional foreign key field SiteDetail which links to Person. This muddies the entity relationship somewhat, however it is a necessary compromise in order to provide addresses and contact information for the site. Because this is optional, it is not indicated on the ERD above.

Supervisor

Supervisor		
Primary Key	ID	int, auto_increment
Foreign Key	PersonReference	Int
	Title	varchar(15)
Foreign Key	Department	Int

This table models the case worker's supervisor. Its primary key is, again, ID. Its contents are identical to caseworker. The table needs to exist however to create a clear separation between organizational management and leadership and the caseworkers below them. While this could be done by filtering on the "client" field, this would not be an optimal solution. A clear division should occur. Additionally, check constraints exist on this table to ensure that supervisors are not mixed in with case workers - their job is to supervise them, not to manage cases themselves.

A5 User Inteface Design

The user interface should be clean and simple. Rather than describe each page in a narrative, wireframe mockups are used to present what the pages should look like once constructed. These can be seen below.

A5.1 Page Views

Welcome Page.

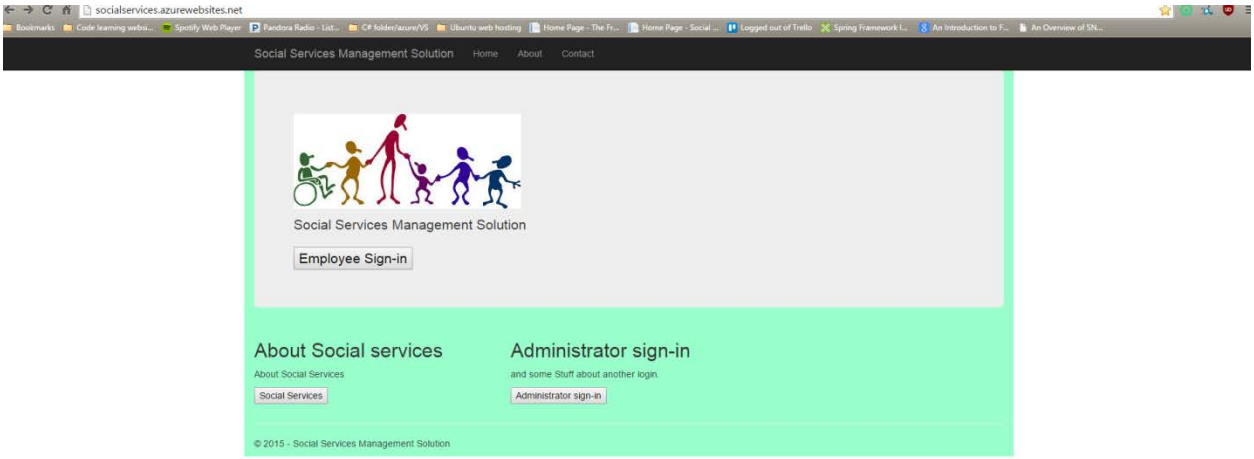

Figure A3 – The Welcome page

About Page.

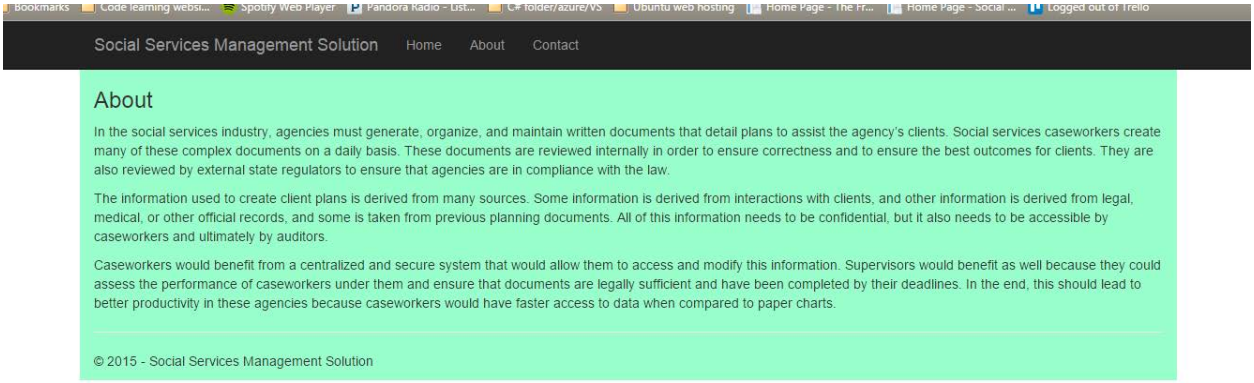

Figure A4 – The About page

Contents Page.

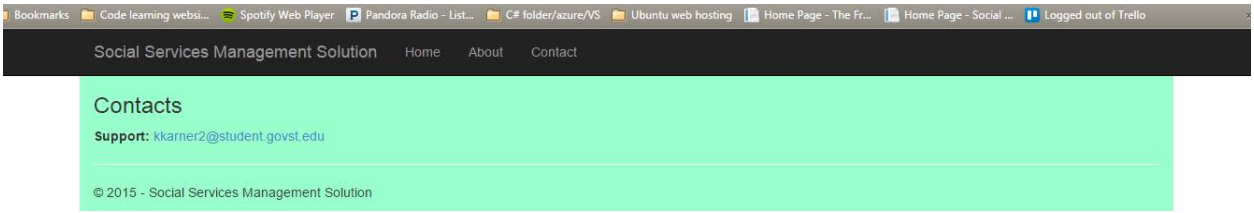

Figure A5—The Contacts page

Employee Login.

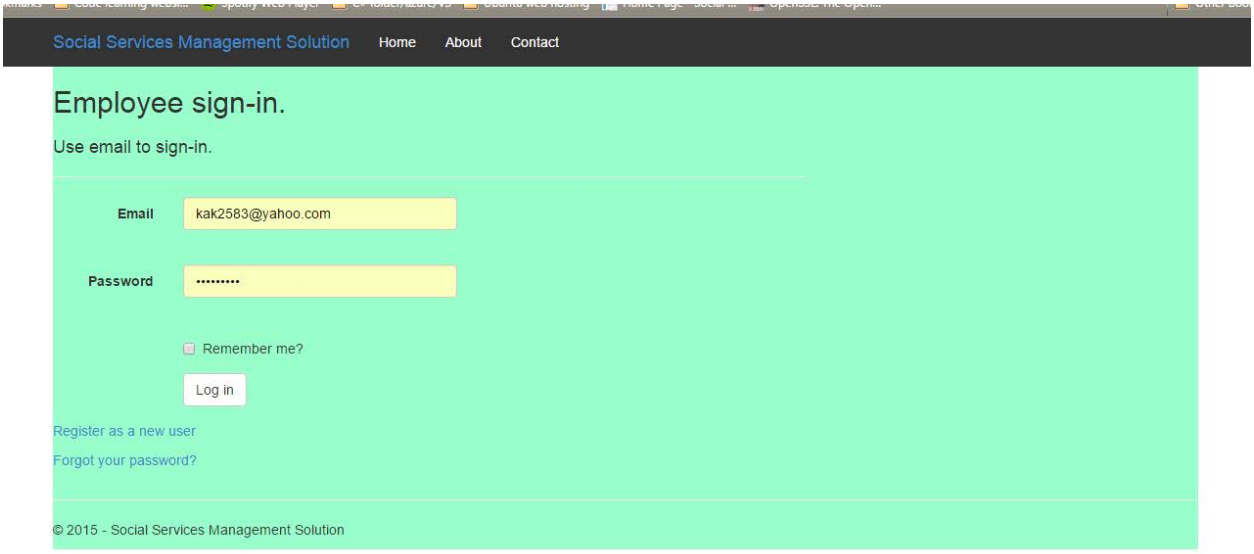

Figure A6 -- Employee Login page

Administrator Login.

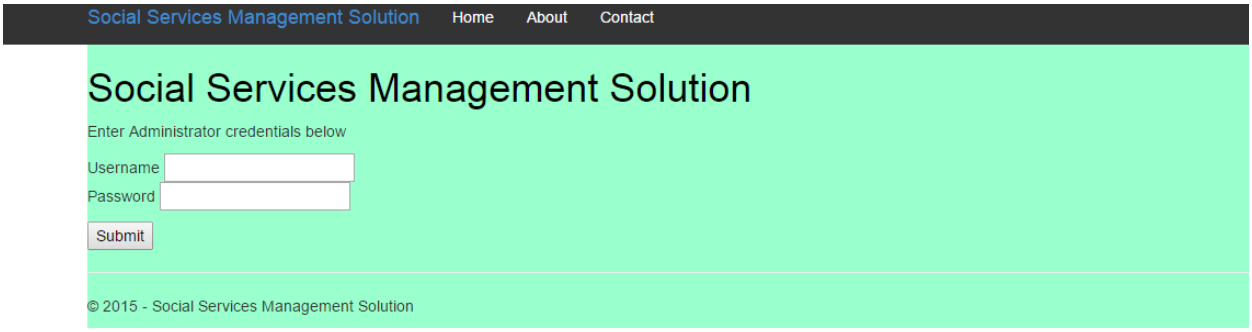

Figure A7 Administrator Login page

Employee Register.

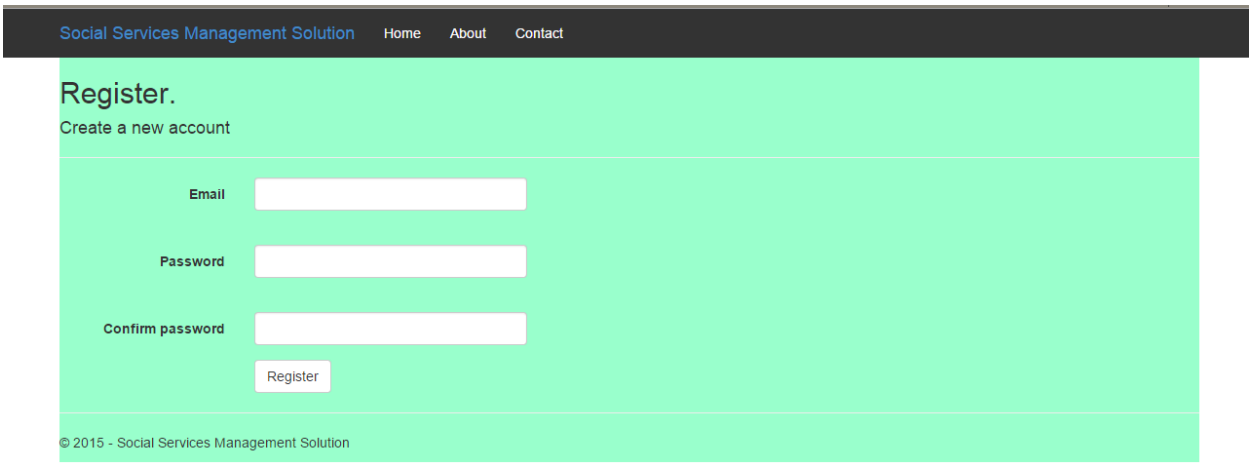

Figure A8—Employee Register

Login Success.

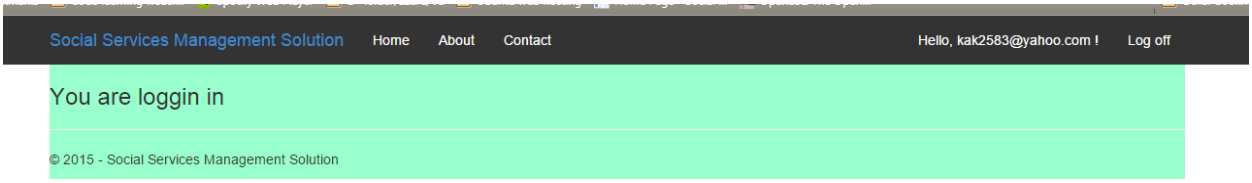

Figure A9—Login Success

Account Management.

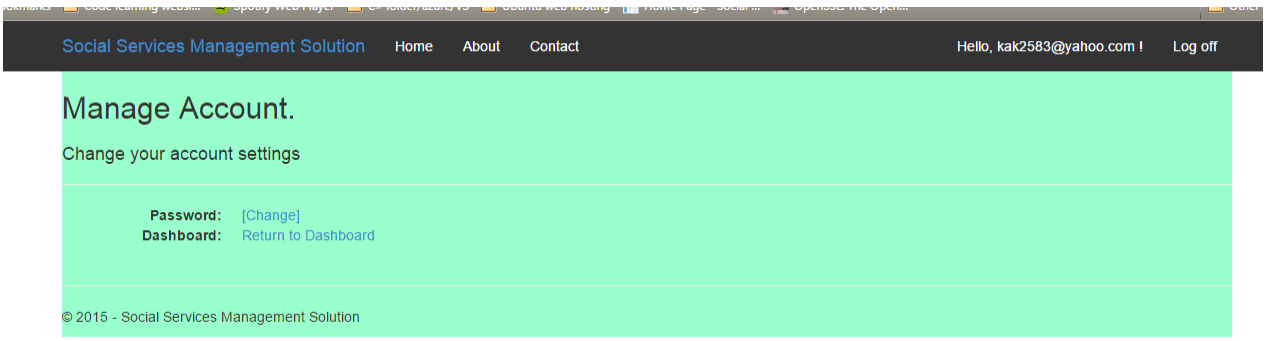

Figure A10—Account Management

A5.2 Page Code Behind

The C# code used in this program drives each individual page. They are used to manage state by storing values in session variables. They also connect to the DAO and utilize the POCO model classes listed above. A detailed discussion of routines is not necessary here, since all the routines in the code-behind files facilitate Create-Read-Update-Delete (CRUD) operations. These have extensive documentation throughout academic literature and industry periodicals, and the code here is doing nothing novel.