2011

A Dynamic Hierarchical Web-Based Portal

Matthew James Spaulding
University of South Florida, mspauldi@mail.usf.edu

A Dynamic Hierarchical Web-Based Portal

by

Matthew Spaulding

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Jay Ligatti, Ph.D.
Dewey Rundus, Ph.D.
Rafael Perez, Ph.D.

Date of Approval:
October 19, 2011

Keywords: FLEX, Actionscript, Enporion, .Net, MSSQL

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Jay Ligatti for all of his help and guidance during my thesis as well as my coprofessors, Dr. Rafael Perez and Dr. Dewey Rundus. I would also like to thank Dr. Sylvia Thomas for getting me involved in student research and Dr. Rudy Schlaf for taking me under his wing. I could not have accomplished this without the help of my parents who have supported me the whole way. I would like to thank my wonderful wife for sticking by me these past years and I would also like to thank my in-laws for birthing my wife and being awesome, in general.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A dynamic hierarchical web-based portal was created to house a suite of web-based supply chain management applications. The portal was designed in a hierarchical manner to match the structure of large companies. Administrators of the the portal have the ability to form the portal in such a way to mimic the existing structure of their company. Access rights to the applications in the portal may be granted or denied per division and the users of the portal are placed in these divisions based on their duty in the company. In addition, users are granted roles which dictate their ability to modify how the portal behaves. This thesis covers the client-side portion of the portal which includes the user interface and several client/server interfaces. The client-side was built with the Adobe Flex SDK and the Flash Builder 4 IDE, which connects to a ASP.NET MVC 3 back end and a MSSQL database. The code was internationalized to be able to plug in a localized language resource file. Unit tests were written with FlexUnit to test the code before deployment.

CHAPTER 1

INTRODUCTION

Enporion, a supply chain solutions company, delivers a suite of on-demand applications spanning the spectrum of supply chain management that includes all phases of the Forrester 8-point cycle [2], as shown in Figure 1.1. Enporion has supply chain management solutions to meet the needs that a company has for better managing its supply chain. Enporion does not require the entire suite to be purchased to implement; applications are available in an a la cart platform.

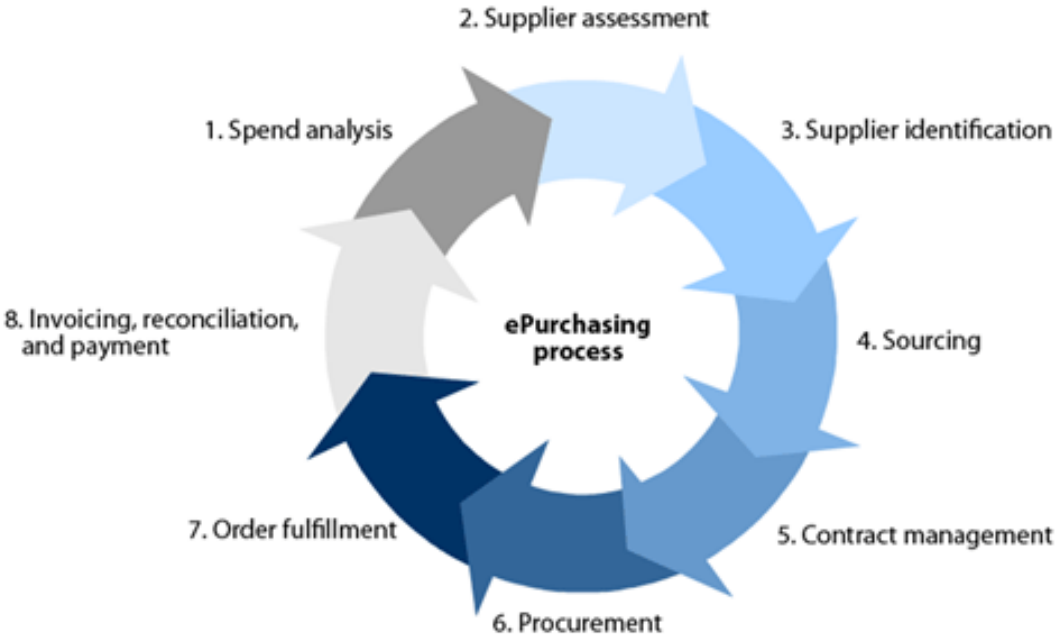

Figure 1.1. Forrester 8-point cycle [1]

Enporion was founded in 2000 by seven Fortune 500 companies to provide its customers with software solutions and professional services that streamline their sup-

ply chain and lower operational costs [3, 4]. Rather than installing large software systems into enterprise machines, Enporion offered its applications as a web-based service. These applications have adapted through the years to meet the needs of its clients. In 2009, the Company was acquired by its Chairman CEO, George Gordon, who remains in that role and is now Enporion's largest shareholder [3]. When the company started in 2000 so did the concept of the web portal [5]. Since then, many [5, 6] have predicted that the web portal would be the most important software tool in the near future. Now that Mr. Gordon is the CEO of Enporion, he has decided to update the existing HTML applications to fresh FLEX applications. Before this is done, he would like to have a portal created to house a unified suite of applications. This thesis describes my work to create such a portal.

1.1 Motivation

Enporion hosts a suite of Software as a Service (Saas) applications for business-to-business transactions. Some of these applications include Bid Manager, Procurement Manager, Catalog Manager, and several others that make up the suite. Each of the applications are sold separately to conform to the needs of the purchaser. Within each application, a company may add several users and assign roles to the users to control access of certain functionalities in the application. The problem with this current model is the lack of structure involved with maintaining a uniform user database and lack of consistency throughout the applications. Currently, in order to access these applications a user has to log into each application separately. Consequently, a company's information must be stored separately for every application the company has purchased. Furthermore, a company has to add users and assign roles to every user for each application. This could be quite a task if the company decides to purchase the entire suite of applications.

Another problem with this structure is handling session timeouts. Each application has its own session timer, and each session timer is handled differently. If a company has purchased multiple applications from the suite and would like to adjust the session timeout, the company would have to do so for each application. Localization is another issue with similar problems; the language must be set in each application.

It would be nice to have a portal where a user can access all, or some, of Enporion's applications in one place. A user can log into this portal and be able to use all of the applications that her company has purchased, with one username and password [7]. A company, on the other hand, will be able to add a new user and assign a role to the user only once for the portal. Once logged into the portal, this user will now have access to all of the applications that the company has purchased, and the user's role will be reflected throughout every application. The portal will contain its own session timer and omit the need for individual session timers for each application. Now that the user is using an application in the portal, a predetermined time of inactivity within the portal will prompt a session timeout causing the user to be logged out of the portal and, subsequently, the application. The locale of all of the applications can now be selected once in the portal and the user's language will be reflected throughout every application in the portal.

Another aspect of the portal design is its ability to scale and match the existing structure of the company using it. Most large companies are structured in a hierarchical manner and their portal should be too. When a company purchases the suite of applications, it may not want an employee in the marketing department to be able to access the contracting applications. With a hierarchical portal, users can be placed in separate divisions of the portal which will only have access to certain applications related to the department they work in. A hierarchical portal will make granting application access rights more intuitive.

1.2 An Overview of the Portal Framework

A global perspective of the portal framework can be viewed as a main application containing three sandboxed modules: the Login Module, the Portal Module, and the Register Module. Figure 1.2 depicts the global framework. In the main application there also exists the locale selector which is a global entity that can be manipulated at any time and during any state of the application. These modules are kept separate for security concerns and may only manipulate themselves, subsequent modules contained within them, and the locale. Inter-module communication is made possible by means of module communication interfaces and is otherwise prohibited. The Login and Portal Modules are linked strictly by the Login and Logout Interfaces and have no other means of communication. These intermodule-communication interfaces are described in more detail in Sections 2.2.2 and 2.3.10, respectively. The Login Module is linked to the Register Module without an interface because no information is passed in this link. Three other interfaces are the Forgot Password Interface, Invite Interface, and Register Interface. These interfaces interact with a section of the portal database that stores information that must be approved. These data-communication interfaces are not members of any module and are discussed in Sections 2.2.4, 2.3.7.2, and 2.4.1, respectively.

1.3 Thesis Organization

The rest of this thesis describes my work to address the problems listed in Section 1.1 by creating a new web portal to house Enporion's suite of supply chain management applications. Chapter 2 discusses the the design decisions that were made and walks through the portal's features. Chapter 3 overviews the Adobe Flex SDK, which was used to develop the portal. Finally, Chapter 4 concludes the thesis.

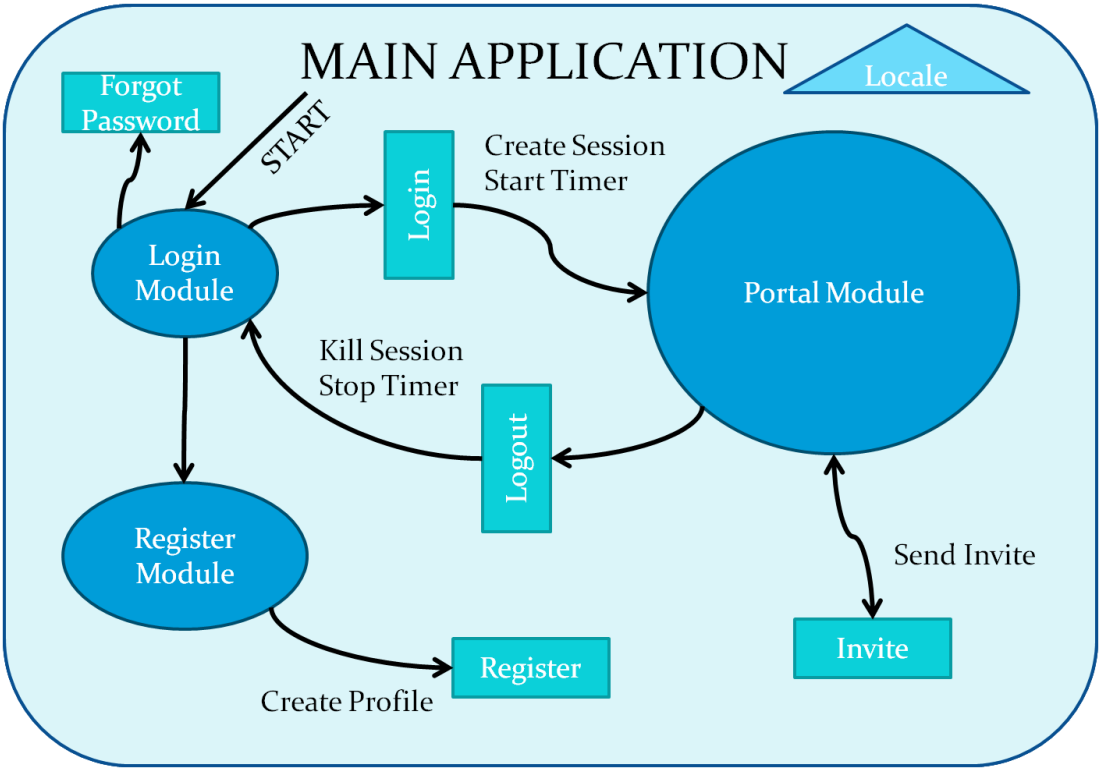

Figure 1.2. The global framework

CHAPTER 2

THE DEVELOPED WEB PORTAL INTERFACE

2.1 Main Application

The Main Application is responsible for managing all of the components and maintaining the locale. It wraps this global entity around the Login, Portal, and Registration Modules. When the Main Application is first started, the locale selector is created and the state of the application starts in the Login Module.

2.2 Login Module

The Login Module is the first state of the application at startup. This simple module contains no other submodules and two screens: the Login Screen and Forgot Password Screen. The Login module is assisted by the Login and Forgot Password Interfaces, which handle user authentication and resetting passwords, respectively.

2.2.1 Login Screen

The Login Screen is a form where the user is to enter the credentials needed to login to the portal. Figure 2.1 shows the Login Screen. The fields include email and password. When the user enters in the information and clicks the Login button, a request is sent to the Login Interface to authenticate the user with the given credentials. The login interface is described in Section 2.2.2. The Login Screen also contains the forgot password button which activates the Forgot Password Interface (Sec. 2.2.4)

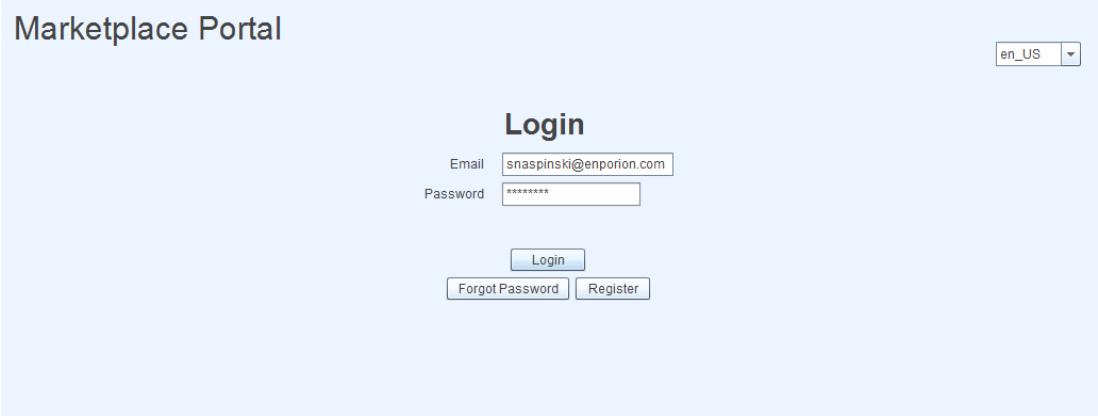

Figure 2.1. Login Screen

and the register button which changes the state of the application to the Register Module (Sec. 2.4).

2.2.2 Login Interface

The Login Interface is an inter-module communication interface and is responsible for authenticating users. It receives an email and password from the Login Screen then an HTTP POST is sent to the server where the user is authenticated. HTTP POSTs are explained in Section 3.2. The user will either pass or fail authentication. Upon success, the server returns the session information which is associated with the user. The Main Application then changes its state to the Portal Module, passes the session information, and starts the session timer. Alternatively, when the user fails authentication, the server will return a generic fail message stating the information is incorrect (Fig. 2.2). The Main Application's state remains in the Login Module and displays the fail message. The message is generic and does not tell the user which part of the credentials is incorrect.

2.2.3 Forgot Password Screen

The Forgot Password Screen is called when the user cannot remember her password and presses the forgot password button on the Login Screen. Figure 2.3 shows

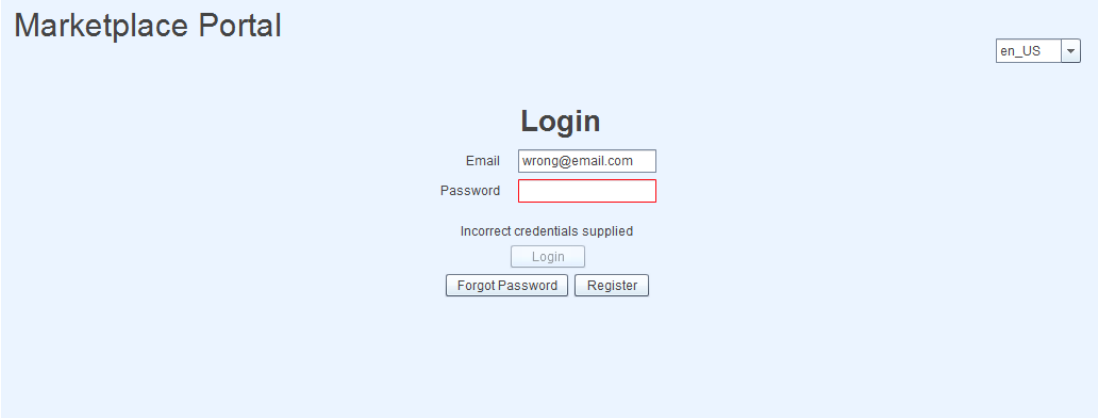

Figure 2.2. Invalid credentials error

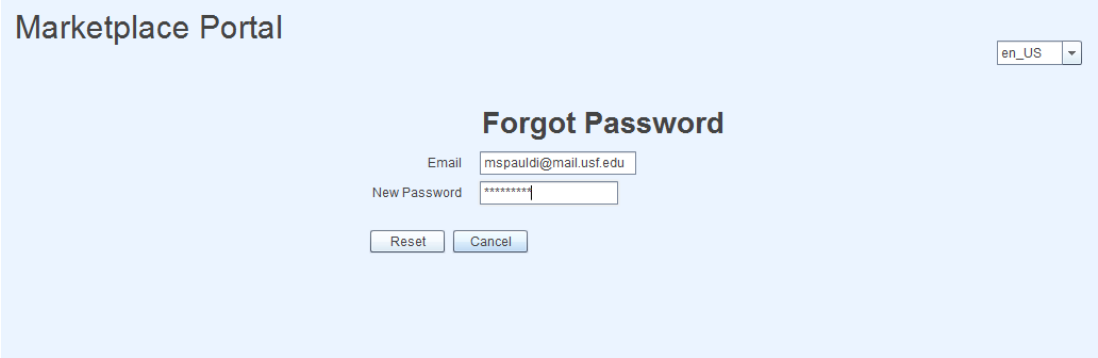

Figure 2.3. Forgot Password Screen

the Forgot Password Screen. This screen simply asks for an email and a new password which is sent to the Forgot Password Interface for processing.

2.2.4 Forgot Password Interface

The Forgot Password Interface sends the email address and new password from the form to the server. The server stores the request in the database and sends an email to the email address it received with a link to confirm the password change. Figure 2.4 shows the message that directs the user to check her email.

When the user clicks on the link, the unique identifier matches with the stored request for password change and the request is granted. A message displays that the account has been activated with the new password.

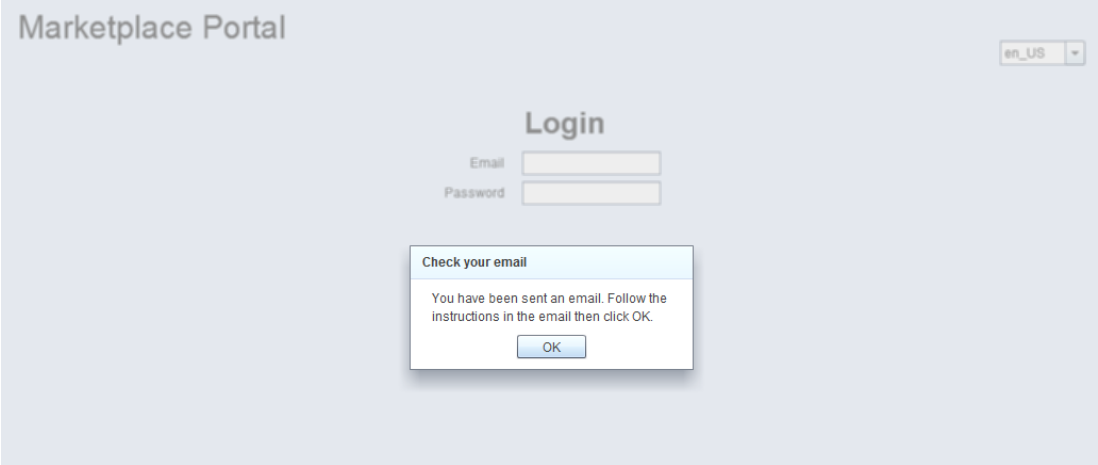

Figure 2.4. Forgot Password Instructions

This password reset process is necessary to provide security for the users. The email is used to verify that the person changing the password is actually the person who the account belongs to. Without this process anybody would be able to change a user's password with knowing only the user's email address.

2.3 Portal Module

The Portal Module is the primary work area of the application. While the Login Module is static and remains the same, the Portal Module varies depending on a user's and company's roles and privileges. Roles and privileges are explained in Section 2.3.3. This module's main responsibility is housing the applications within the portal. The Portal Module is also the place where administrators create subcompanies, add users, and assign roles.

2.3.1 Session

Every user that passes authentication and logs into the Portal is given a session, which is unique for every user. The session consists of user privileges and company settings which dictate how the portal behaves. The portal is constructed to vary depending on the session values. The portal behavior may vary by access rights to

submodules in the portal, the portal may look different, or may be set for different timeout values, etc.

2.3.2 Session Timer

The session timer is used to detect inactivity for a predetermined amount of time set by the portal administrator. The timer is indirectly manipulated by the user. That is, it is reset whenever the user 'kicks' the timer either by moving the mouse or pressing a key. When the timer is kicked, the time is set back to its preset value. In the case of long inactivity, the timer will count down to zero triggering the Session Warning Popup.

2.3.2.1 Session Warning Popup

The Session Warning Popup (Fig. 2.5) encompasses the current state of the application, forcing the application inactive, and leaving the user with only the option to continue the session. The popup creates its own timer and starts it immediately at creation; the amount of time determined by the portal administrator. The popup displays the time counting down as the user decides the action to take. Two scenarios can take place during this state: 1. the user continues the session or 2. the timer expires.

1. If the user decides to continue the session, the popup will disappear, the session timer is reset, and the application will continue in its current state.

2. If the timer were to expire, the application activates the Login Popup keeping the application in its current state.

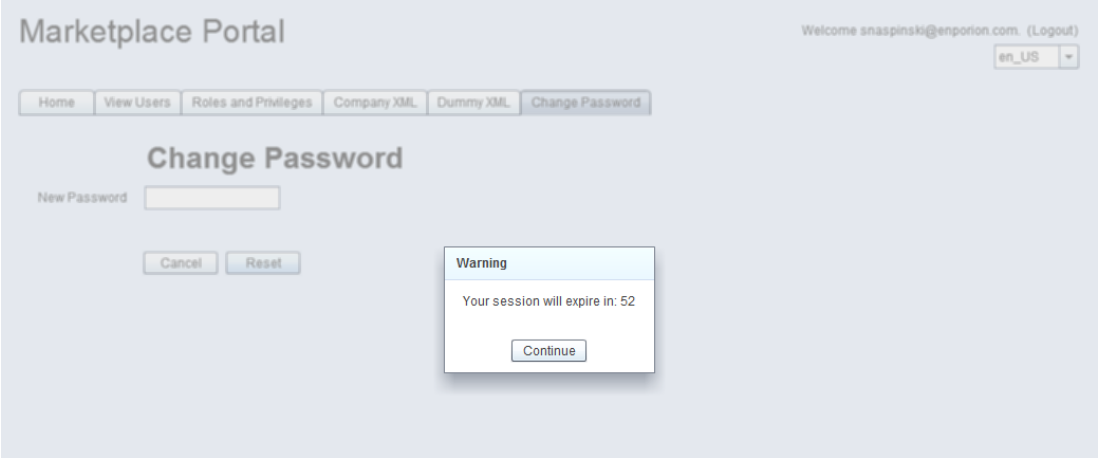

Figure 2.5. Session Warning Popup

2.3.2.2 Login Popup

The Login Popup (Fig. 2.6) exists in a state suspended between being logged in and logged out. When the session timer runs out, this popup acts as safety net to prevent a user from losing her work. The Login Popup may seem similar to the Login Module due to it asking the user for creditials, however it is different in many ways. The main difference being that the Login Popup is any entity of the Portal Module and does not even communicate with the Login Module. When the user logs in with the Login Module, credentials are sent to the Login Interface and a new session is created. Adversely, the Login Popup checks credentials and simply resumes the session that has timed out, returning the application to its previous state and rescuing any unsaved work.

2.3.3 Roles and Privileges

Roles and privileges dictate access rights to submodules within the portal. Companies are granted privileges by the portal administrator and users within a company may be granted only the privileges that were inherited by the company. Inheritance will make more sense when explained in Section 2.3.5.

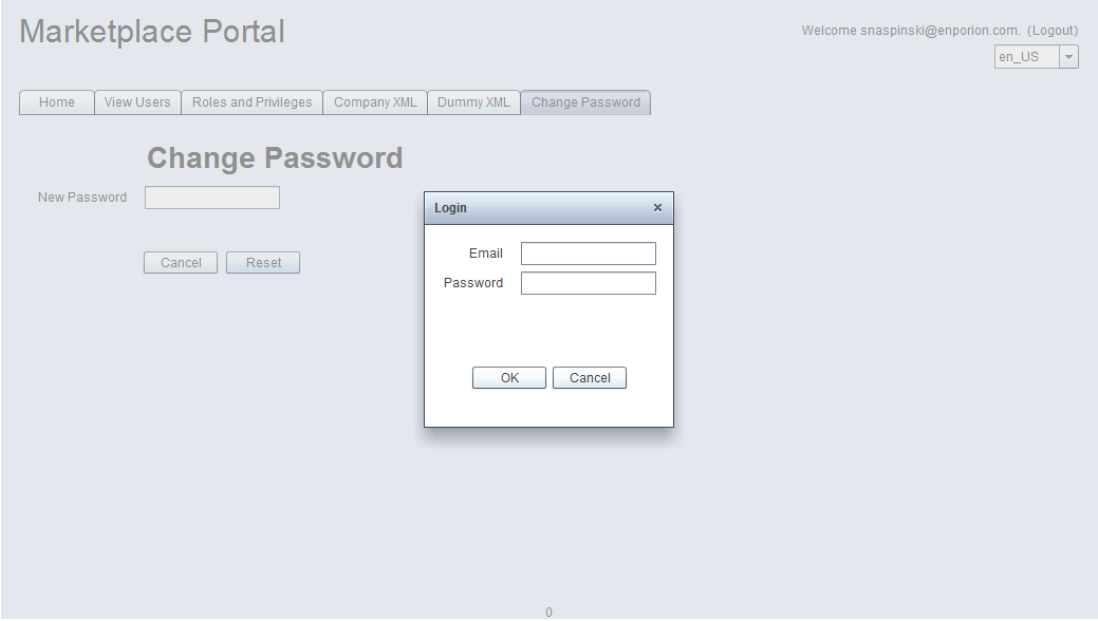

Figure 2.6. Login Popup

2.3.3.1 Privileges

The Portal Module has a hierarchical tree structure. It consists of several submodules; these submodules are made up of subsubmodules and so on. This structure continues until the last module in the hierarchy contains no submodules; we will call these last modules leaf modules. Every leaf module has a privilege tied to it. A module may be accessed only if a user has been granted the privilege tied to the module.

2.3.3.2 Roles

A role is a set of privileges. Rather than assigning a large amount of privileges to each user created in a company, a user can simply be assigned a role. For instance, a company may have an ADMIN role which contains all of the privileges that the company has. These privileges may include a DELETE USER privilege. The company may decide that not all users should have this privilege so the company administrator may make a new role named, say, EMPLOYEE. The EMPLOYEE role would contain all of the privileges necessary to complete a general employee's tasks, but not have

Figure 2.7. Add Role Screen

the privilege to, say, delete a user. In addition to granting users roles, a company may also grant users privileges as well. Say a user is granted the EMPLOYEE role, but needs access to some extra modules not contained in the EMPLOYEE role; the privileges tied to these modules can be added on top of the role. Conversely, if a user has the EMPLOYEE role, privileges grated by this role may not be taken away from the user.

2.3.4 Roles and Privileges Module

The Roles and Privileges Module is where roles are created, edited, and assigned to users. The privileges available to be assigned to users are inherited by the privileges which were given to the company. In other words, a user in a company may only be granted a privilege that was granted to that company. The administrator for a company has the ability to create roles for the users in the company. Figure 2.7 shows how a role is created. In this case, the administrator is creating a role that grants the users privileges to add users to the company and edit their profiles, but disallows them to remove users or alter the company structure.

2.3.5 Privilege Inheritance

Users are not the only ones granted privileges. Each company is also granted privileges by the portal administrator. This enables the portal to be structured in a hierarchical fashion. This feature allows the portal to control access to applications hierarchically rather than only by user. For example, suppose a company is divided into several subcompanies, two of them being Sales and Contracting. The portal can be divided into subcompanies to match the company's structure. The portal administrator can now assign the Sales company privileges which enable sales related applications. Similarly, the Contracting company can be granted contracting application privileges. When a user, Sally, is added to the Sales company, she inherently has privileges to the sales applications. Likewise, if Connie is added to the Contracting company, she inherently can access contracting applications. In this setup, Sally and Connie are guaranteed to never be allowed access to an application which they have no business using. Now let's say Bob is an employee who works in sales as well as contracting. The portal administrator would then add Bob to both respective companies. When Bob needs to do sales work he would log into the Sales company and if he needs to do contracting work he can simply switch to the Contracting company. In order for this portal structure to be usable, users must have the ability to easily switch companies. Company switching is explained in Section 2.3.8.

Privilege inheritance is also important to matching Enporion's business model. As mentioned earlier, Enporion sells its applications separately and businesses have the ability to purchase only the applications that it needs. When the company's portal is created, Enporion grants to the portal only the privileges which are tied to the applications that the company has purchased. Inherently, this company may only grant the privileges which it has been granted. If the company decides later that it would like to add another one of Enporion's applications, Enporion would then simply grant that application's privileges to the portal.

Figure 2.8. View Companies Screen

2.3.6 View Companies Module

Companies are the names given to groups of users with similar application access rights. Companies can have subcompanies which may contain some or all of the application access rights their parent company has. In the View Companies Module, an administrator can visualize and manipulate the hierarchical structure of the portal. Figure 2.8 shows the concept of the company hierarchy.

2.3.7 View Users Module

The View Users Module (Fig. 2.9) is where a company's users are manipulated. This module is typically reserved for an administrator. In this module is a list of all of the users that belong to the company that is currently selected. There is also a list of the users who have registered to be in this company. This module is where users in the company are added, edited, assigned roles and removed.

2.3.7.1 Invite

Clicking the invite button brings up a prompt to enter a user's email address. Entering an email and pressing invite activates the Invite Interface.

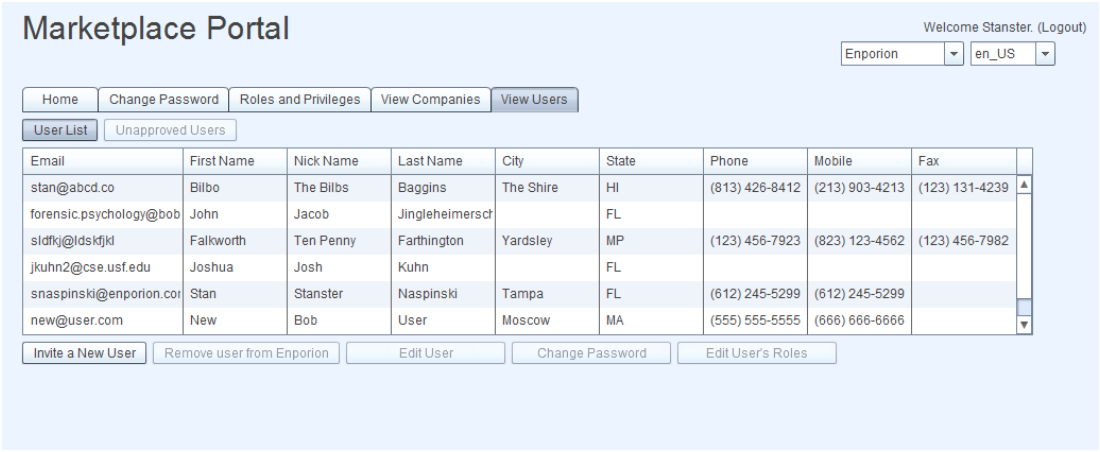

Figure 2.9. View Users Screen

2.3.7.2 Invite Interface

The Invite Interface is a data-communication interface that receives an email address from the invite user prompt and sends an email to the potential user asking her to join this company. The interface stores this invite in the database and pairs it with a unique identification string. The invitee will receive an email asking her to follow a link. This link contains the unique identification string which is paired with the invite stored in the database. The purpose of this process is to prevent a bot from creating numerous users in an attack on the portal. The user registration process is discussed in section 2.4.

2.3.7.3 Unapproved Users

Once a user has successfully completed the registration process, she is placed in the unapproved users list (Fig. 2.10). The company administrator has the option to either approve or reject the user. This approve/reject step is an added layer of protection for registration. When the admin accepts the user, she becomes a member of the company. An email is sent to the user with a notification that she has been accepted.

Figure 2.10. Unapproved Users Screen

2.3.7.4 Edit User

Administrators have the ability to edit a user's information. Figure 2.11 shows the administrator setting up the new user's profile. The profile contains information such as name, address, phone number, etc. Figure 2.12 shows the user added to the company with the profile changes just made.

2.3.7.5 Remove User

The administrator may remove users from a company. Removing a user from a company does not delete the user from the portal. A user may be a member of several companies; removal from one company does not affect a user's status in any other company.

2.3.8 Switching Companies

The portal follows a hierarchical structure and is divided into companies and these companies may be divided into subcompanies and so on. Users may belong to several companies within the portal and must have and easy way to switch between the companies they belong to. The Company Dropdown (Fig. 2.13) provides an intuitive

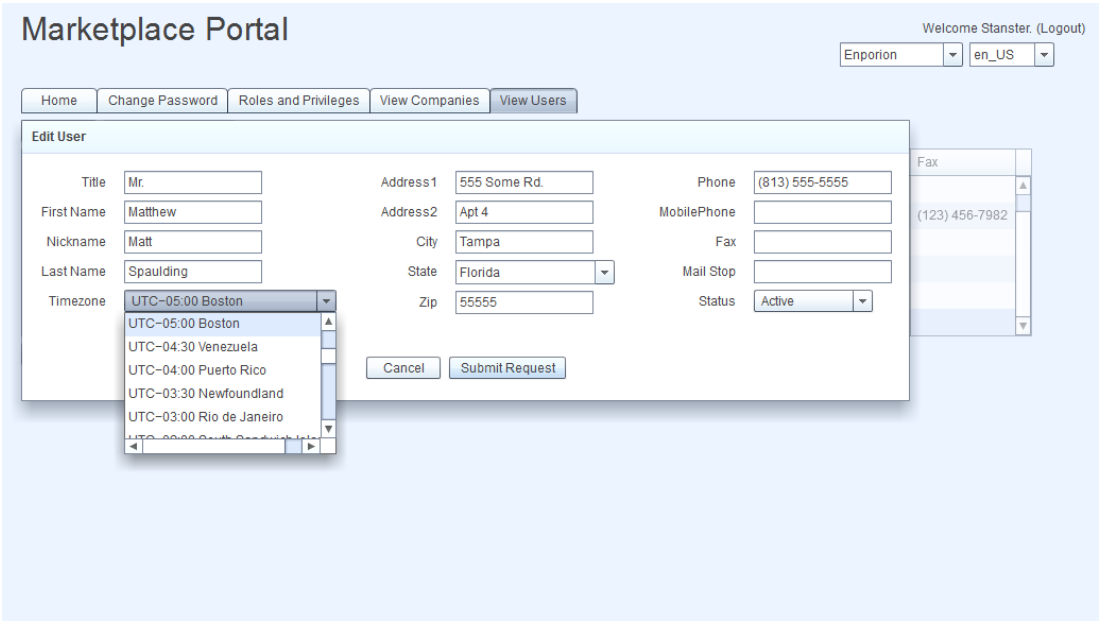

Figure 2.11. Edit User Screen

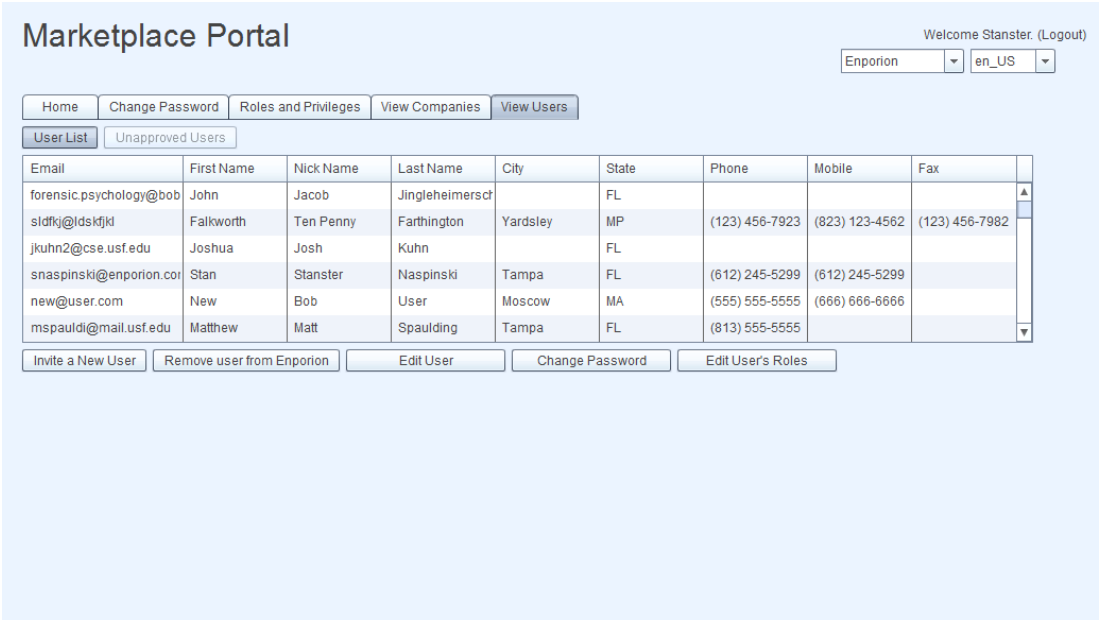

Figure 2.12. New user added

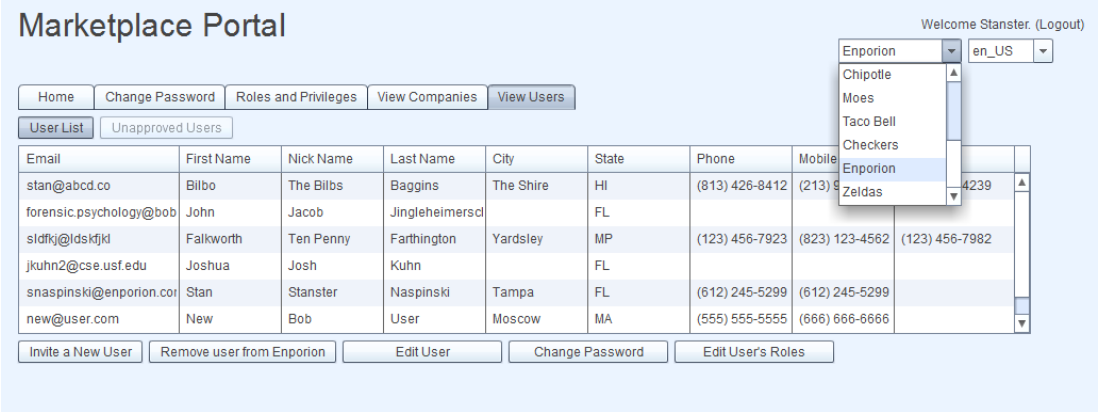

Figure 2.13. Company Dropdown

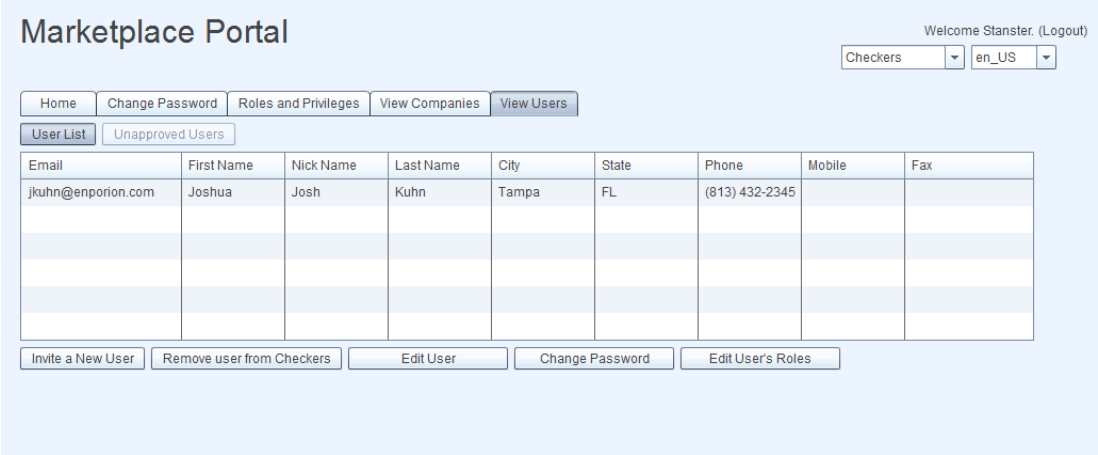

Figure 2.14. Company changed

means of switching companies. When this dropdown is clicked, a list appears showing all of the companies that the user belongs to. Clicking on a company in the list takes the user to the selected company. Figure 2.14 shows a user switching to a different company. Notice that the list of users has changed to reflect the users belonging to the selected company.

2.3.9 Change Password Module

The ability to change your own password is available to every user in the portal no matter the user's role. The process for changing a password in the Change Password Module is much different than the process used for a forgotten password. A forgotten

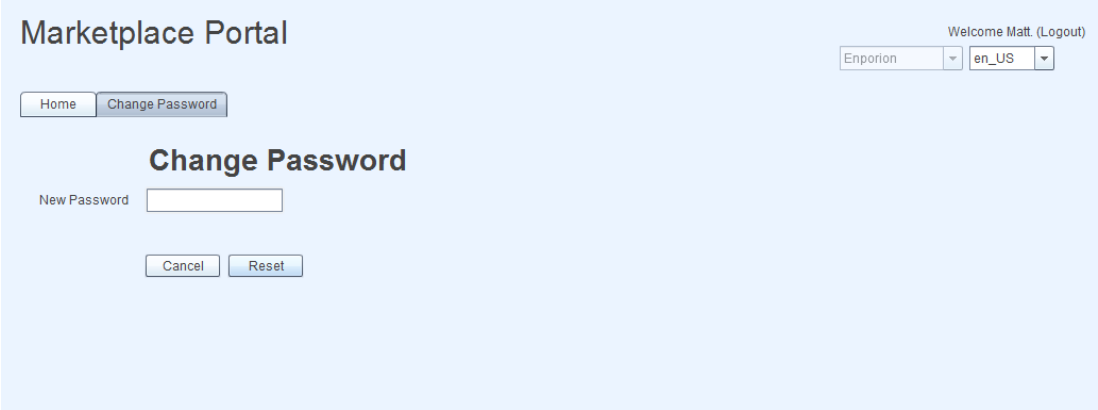

Figure 2.15. Change Password Screen

password must be changed using Forgot Password Interface, described in Section 2.2.4. In this case we are not sure if the person (or bot) attempting to change the password is actually who they claim to be and a verification process is used. On the other hand, in the Change Password Module we can assume that the person changing the password is legitimate since she must have logged in to reach this module. Without the need for verification, this process is much simpler. The user simply enters a new password and it is changed in the database.

2.3.10 Logout Interface

The Logout Interface is an inter-module communication interface used between the Portal Module and the Login Module. The Logout Interface is responsible for ending a user's session. When the logout button is clicked, this interface communicates with the back end to let it know that the session is to be killed. The back end responds once the session is terminated and the interface changes the Main Application's state to the Login Module.

2.4 Register Module

The Register Module is where potential users ask to become a member of a company in a portal. This potential user should have received an email asking her to

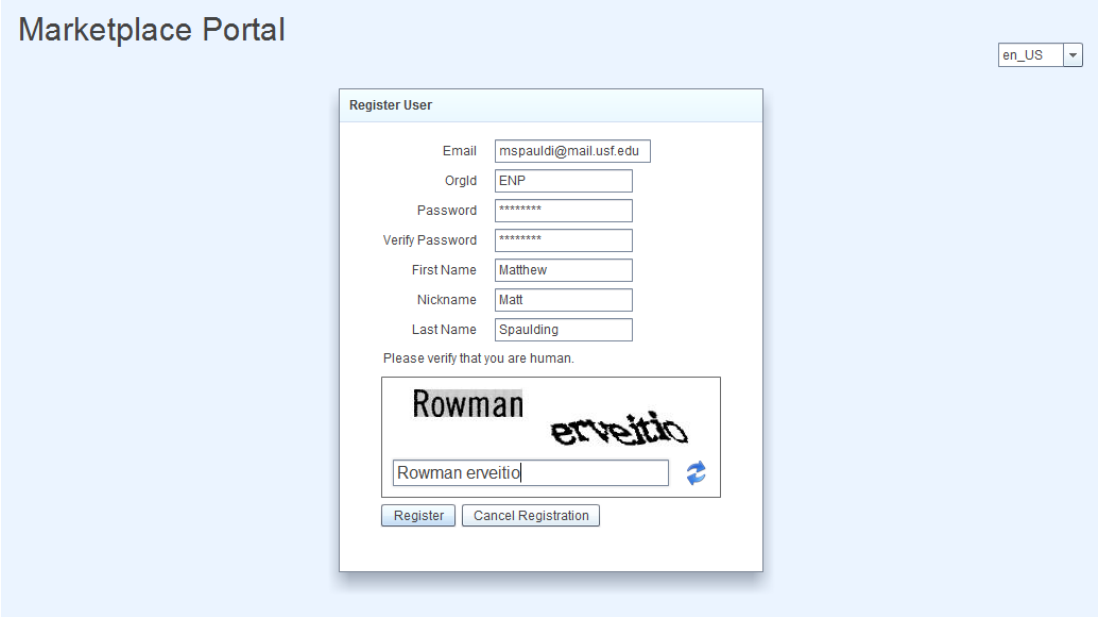

Figure 2.16. Registration Screen

register by following a link. This email was generated by the Invite Interface which was explained in Section 2.3.7.2. Following this link will take the user to the Register Module. A form is displayed asking the user for information needed to create the profile (Fig. 2.16). The email address and OrgID fields are auto-populated by the link provided in the email. The OrgID is a three-letter key that represents a company; here, ENP represents Enporion. The user creates a password and enters a name for the profile. Before registration, the user must prove that she is human by completing a Captcha verification, preventing bot attacks. This information is sent to the Register Interface.

2.4.1 Register Interface

The Register Interface is a data-communication interface that responsible for interacting between the Register Module and the portal database. Upon successful registration (Fig. 2.17), the new user request is sent to the server and the user is placed in the unapproved users database for the company the request was sent to.

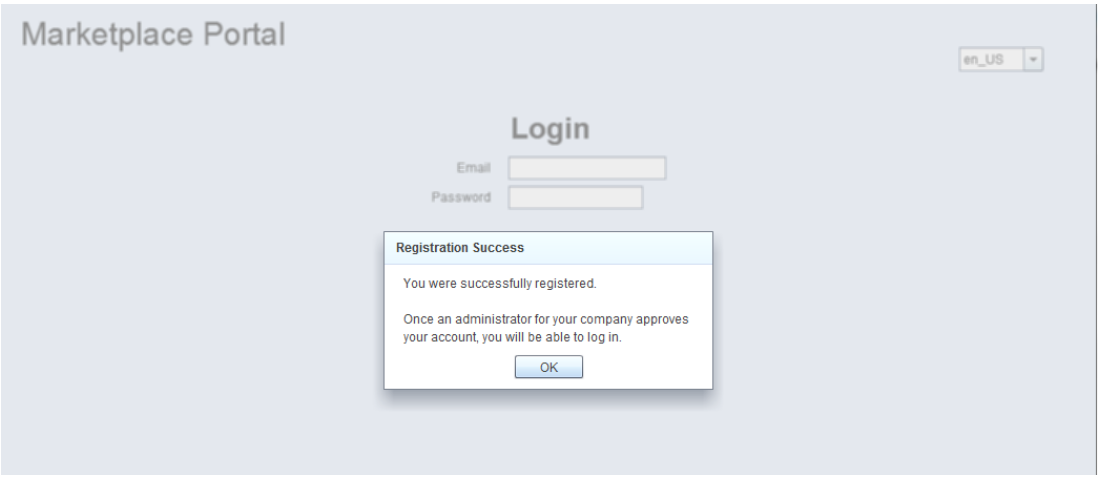

Figure 2.17. Successful registration message

This user is not yet a member of the company and must be approved by the company administrator. This interface sends an email notifying the company administrator of the request. New user approval is explained is Section 2.3.7.3.

2.4.2 User Approved

When the company administrator approves the user, she can now log into the portal. Figure 2.18 shows the portal as the new user sees it. The user can now access the applications that this company has access to, which would appear in the Home Screen. Since this user was not given an administrator role, she does not have access to the modules which are reserved for an administrator, like altering companies or users. Also, notice that the Company Dropdown is disabled. This is because the user belongs to only one company.

2.5 Locale

Localization is the process of customizing data and resources of code to fit the needs of a specific market [8]. The locale of the portal may be changed at any time by using the Locale Selector.

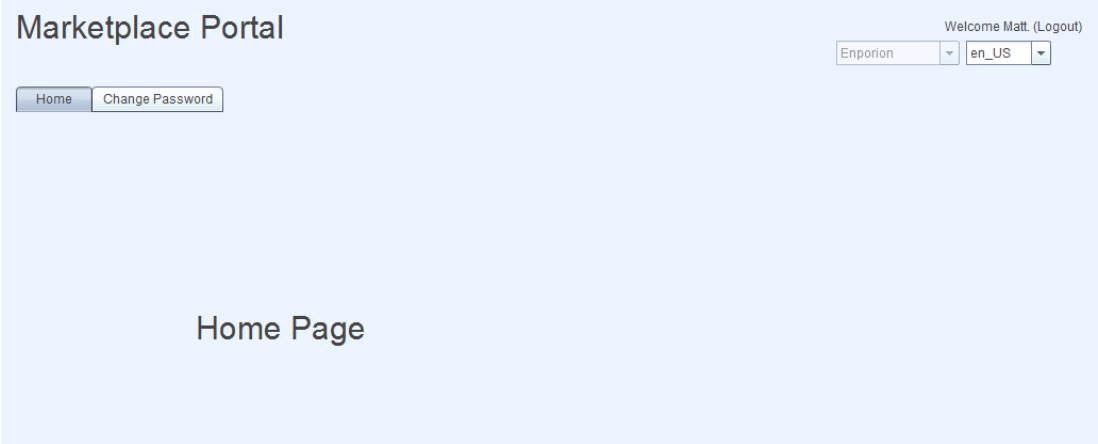

Figure 2.18. New user's portal

2.5.1 Locale Selector

The Locale Selector is a dropdown box located at the top-right of the application that allows the user to select the preferred language to use (Fig. 2.19). The default locale is set to en_US or English as spoken in the United States. This global entity can be directly manipulated within any module state. The user has the ability to change the locale of the application regardless of what state the application may be in; while logging in, anywhere within the portal, or during registration. All of the locale information is stored locally on the user's machine at compile time. Doing this has pros and cons associated with it. The advantage of this approach is that the locale change is reflected immediately using only local resources without the need to communicate with the server. More importantly, minimizing data transfer to the server from global entities minimizes security risks for the application. The disadvantage of storing the locale information on the local machine is that all of the information must be transferred from the server resulting in unnecessary data transfer at startup and a more 'bloated' application. I decided that the security and response time advantages outweigh the bloating disadvantages. Internationalization and localization are more thoroughly explained in Sections 3.3 and 3.4, respectively. Figure 2.20 shows the portal when switched to the fr_FR locale.

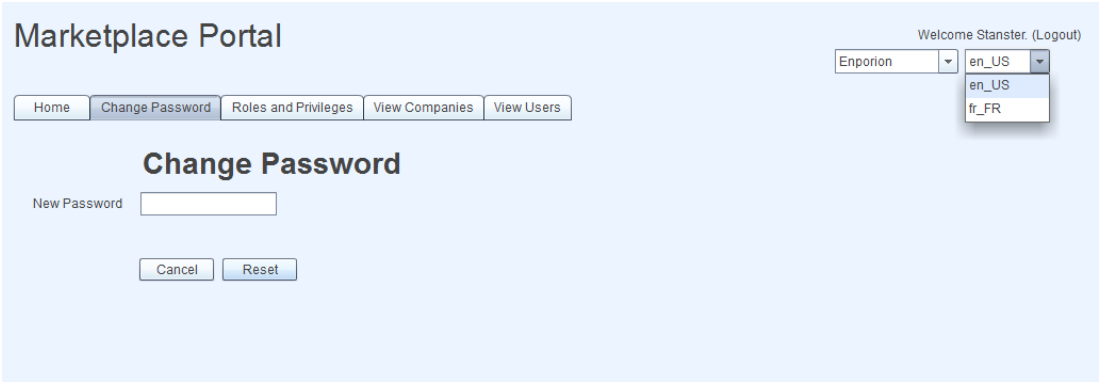

Figure 2.19. Locale Dropdown

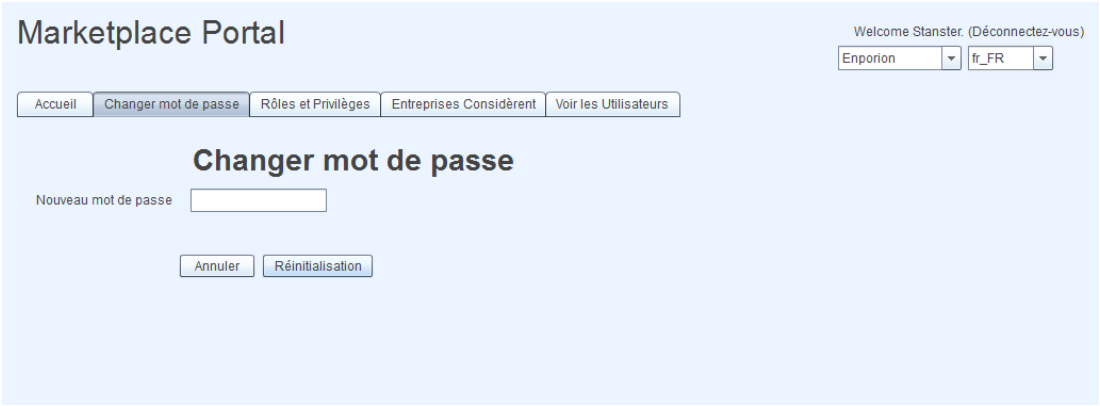

Figure 2.20. Locale changed to French as spoken in France

Figure 2.21. French Login Screen

Since the locale is a global entity that wraps all of the modules in the Main Application, the selected locale remains the same as the Main Application switches module states. Figure 2.21 shows the portal after the user has logged out. Notice that the locale remains in the French setting.

CHAPTER 3

IMPLEMENTATION

This project has been a group effort. Stan Naspinski, Nalin Saigal, Josh Kuhn, and myself make up the team. It has taken us nine months to plan, implement, and test the portal. The client-side alone took over 15,000 lines of code to complete. Much of the time, about three months, was devoted to the planning phase to ensure that the development was started correctly. In this time, we made good use of the whiteboard to draw diagrams and application flows. We surveyed technologies and completed tutorials to determine which platforms would best suit the portal while remaining relevant and upgradeable for the future. This chapter describes some of the design decisions made and the technologies used to accomplish this project.

3.1 Adobe Flex SDK

The Adobe Flex SDK provides a highly productive, open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops and operating systems [9]. The Flex SDK was used in this project for its ability to create highly interactive, expressive applications. Flex can pull data from multiple back-end sources and display it visually. Then the data can be changed and automatically updated on the back end, which is precisely what was needed for this portal. The back end of the portal was created using ASP.NET MVC 3 and a MSSQL database.

The Flex SDK is used by the Adobe Flash Builder which is a plugin to the Eclipse IDE. Adobe Flash Builder offers built-in code editors for MXML and ActionScript and a WYSIWYG editor for modifying MXML applications [10].

3.2 HTTPService

All back end communication is accomplished using Flex's built-in HTTPService class. Each HTTPService has an ID, a connection URL, a method describing what to do (either GET or POST), and a result handler to receive the response.

For instance, if the portal needs to display the list of users in a company, Flex would use the *getUsers* instance of an HTTPService to GET the list of users. This instance would contain the URL to the ASP.NET page where the list of users are queried. The back end would then structure the list of users in XML format and send it to the HTTPService result handler for processing on the front end.

If Flex needs to send information to the back end, the HTTPService instance's method would be set to POST. For example, the *login* instance would form an XML structure of an email and password and POST it to a specified URL. The back end receives this data and authenticates the user. If the user passes authentication, the back end returns a session to the result handler. If the user fails authentication, the back end returns a fail message to the result handler. The result handler will then change the application's state to the Portal Module or display the fail message depending on this response.

3.3 Internationalization

Internationalization refers to the building of software in such a way that separates localizable data and resources (i.e. language, images, schemes) from the primary functionality of the software [8].

The portal was written in such a way that it does not need to be rewritten for each local market. The Flex 4 SDK comes packaged with a method for handling internationalization of applications. The built in ResourceManager class offers a method for creating 'hooks' in the code where localized strings can be attached at runtime. The ResourceManager calls the getString method with two parameters: 1. the resource file location and 2. the name of the string to use. Figure 3.1 shows the MXML code for the Login Screen. The Login Screen contains a form with label/input pairs for email and password and some buttons on the bottom. The labels, which would normally have a text string, have been replaced by calls to the ResourceManager.

```
75    <mx:Form defaultButton="{loginButton}">
76        <mx:FormHeading label="{resourceManager.getString('Loc','LOGIN')}"/>
77        <mx:FormItem label="{resourceManager.getString('Loc','EMAIL')}">
78            <s:TextInput id="email"/>
79        </mx:FormItem>
80        <mx:FormItem label="{resourceManager.getString('Loc','PASSWORD')}" >
81            <s:TextInput id="password" displayAsPassword="true"/>
82        </mx:FormItem>
83    </mx:Form>
84    <s:Label id="errorLabel"/>
85    <s:Button label="{resourceManager.getString('Loc','LOGIN')}"
86              id="loginButton"
87              click="loginButton_clickHandler(event)"/>
88    <s:Button label="{resourceManager.getString('Loc','FORGOTPASSWORD')}"
89              id="forgotPasswordButton"
90              click="forgotPasswordButton_clickHandler(event)"/>
91    <s:Button label="{resourceManager.getString('Loc','REGISTER')}"
92              id="registerButton"
93              click="registerButton_clickHandler(event)"/>
94
```

Figure 3.1. Internationalized MXML

3.4 Localization

Localization is the process of customizing data and resources of code to fit the needs of a specific market [8]. A locale is the combination of a language and a country code; for example, en_US refers to the English language as spoken in the United States, and fr_FR refers to the French language as spoken in France. Locales can share languages. For example, en_US and en_GB (Great Britain) are different

locales and asset bundles. In this case, both locales use the English language, but there are slight differences in word spellings and phrases so they need different assets. For example, an application in the en_US locale might spell the word *color*, whereas the word would be *colour* in the en_GB locale [11].

To show proof of concept of a localized portal, two asset bundles were created, one for the en_US locale and one for the fr_FR locale (Figs. 3.2, 3.3). The files are nothing but name/value pairs where the name is the identifier that links the code to the value and the value being the string to be placed in the code. The file to be used is selected at runtime (the default being en_US) where the ResourceManager then selects which file to use by its name. For example, if the user selected fr_FR as the locale, the ResourceManager simply selects the resource file named fr_FR [12].

```
1LOGIN=Login
2EMAIL=Email
3PASSWORD=Password
4FORGOTPASSWORD=Forgot Password
5REGISTER=Register
```

Figure 3.2. en_US Resource File

```
1LOGIN=Connectez-vous
2EMAIL=E-mail
3PASSWORD=Mot de passe
4FORGOTPASSWORD=Mot de passe oublié
5REGISTER=S'enregistrer
```

Figure 3.3. fr_FR Resource File

3.5 Unit Testing

Unit tests are used to catch errors in code before deployment. The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly

as you expect [13]. Unit tests make tracking bugs easier, code easier to maintain, and code more understandable [14].

3.5.1 FlexUnit

The Flex SDK offers a built-in tool for testing the correct functionality of the code. FlexUnit is a unit testing framework for Flex and ActionScript 3.0 applications and libraries. It mimics the functionality of JUnit, a Java unit testing framework, and comes with a graphical test runner [15]. Figure 3.4 shows an example of a unit test. In this test, the functionality of the *Marshal* method is tested. The *Marshal* method introspects a value object and mashals its properties into an XML structure, used to pass to the back end. Here, we test the *Marshal* method on a *Company* value object. The *Company* value object is constructed with dummy data then the *Marshal* method is performed on this object. The test asserts that the results of the marshal are equal to an XML structure that is expected. When FlexUnit runs this test, it will either pass or fail the assertion.

FlexUnit tests can only be used on testable code. Much of the application is graphical and written in MXML, which is untestable code as a test case can not be written for code that does not produce a testable result.

FlexUnit comes with a graphical interface to display the test results (Fig. 3.5). In this example, two tests were ran. One of the tests here were purposely altered to fail to show that the test actually will catch an error.

```
34          [Test]
35          public function testMarshal():void
36          {
37              var testObject:Company = new Company("9b43a416-81ee-411a-94e0-cd6cd7138e29","Enporion","1",
38              "ENP","c9aebb93-4b2d-491d-8947-65ccb1b0abbb","true","2011","false","477-98-0000","party","ninjatown",
39              "5","Lightning","false","false");
40              Assert.assertEquals(classToTestRef.marshal(testObject),
41                  <Company>
42                      <Id>9b43a416-81ee-411a-94e0-cd6cd7138e29</Id>
43                      <Name>Enporion</Name>
44                      <CompanyStatus>1</CompanyStatus>
45                      <OrgId>ENP</OrgId>
46                      <MPID>c9aebb93-4b2d-491d-8947-65ccb1b0abbb</MPID>
47                      <Active>true</Active>
48                      <YearFounded>2011</YearFounded>
49                      <IsPublicTraded>false</IsPublicTraded>
50                      <TaxId>477-98-0000</TaxId>
51                      <Duns>party</Duns>
52                      <WhereTraded>ninjatown</WhereTraded>
53                      <SalesVolume>5</SalesVolume>
54                      <MbeWbeType>Lightning</MbeWbeType>
55                      <ServiceSupplier>false</ServiceSupplier>
56                      <MaterialsSupplier>false</MaterialsSupplier>
57                  </Company>
58              );
59          }
```

Figure 3.4. Unit test

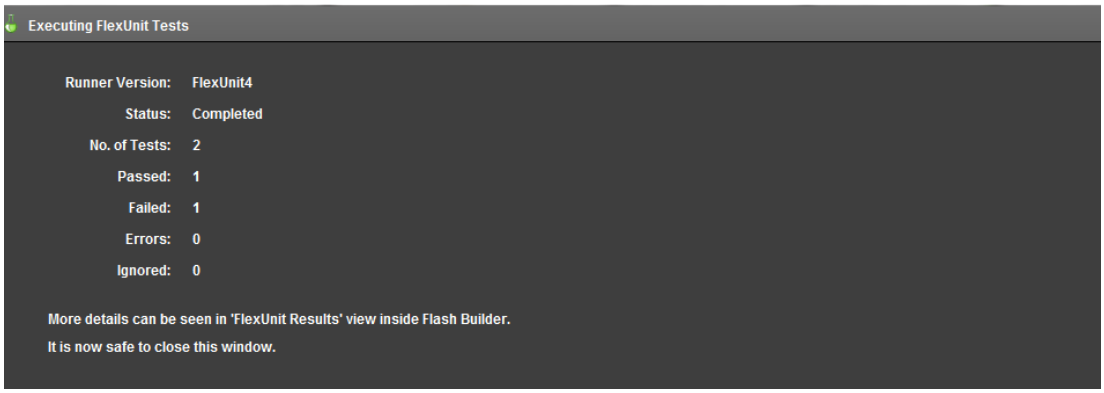

Figure 3.5. FlexUnit results

CHAPTER 4

CONCLUSIONS

A dynamic hierarchical web-based portal was created to host Enporion's suite of supply chain management applications. This portal was designed to conform to Enporion's a la cart business model. The previous method of application delivery was outdated and cumbersome. It required each application to be set up separately, each user would have to be added to every application individually, and they each had their own database. By housing the applications in a portal, the administrator need only set up the portal where the applications sit, users are only added once to the portal and are given access to only the applications which they need, and one central database now stores all of the information for the entire suite of applications.

One of the main design decisions was that the portal should be very customizable by the administrator. The administrator of the portal has the ability to structure the portal in a hierarchical manner that is consistent with the structure of the company using the portal. The portal can be divided into subdivisions which control the access of applications the portal possesses. This feature allows the portal administrator to have complete control over users of the portal while making it very easy to manage a large user base. Administrators can also create roles and assign these roles to users in the portal to dictate how the portal behaves.

About three months was taken to carefully plan out how the portal was to be created and which technologies we would use to carry out these plans. We decided to construct the portal using the Adobe Flex SDK and the Adobe Flash Builder IDE. The Flex SDK uses the HTTPService class to connect to a ASP.NET MVC 3 back

end with a MSSQL database. Over a nine month period, 15,000 lines of code were written to build the Flex portion of the application.

Enporion wanted to expand its reach to an international market. This means the portal had to be available in multiple languages without having to create a new portal for each language. The code was internationalized by means of the ResourceManager class and localized with language resource files. Now, when Enporion wants to make the portal a new language, making a new language resource file is all that needs to be done. A user can now change the language of the portal at any time by simply choosing her locale from a dropdown box. All of the applications available in the portal will also be internationalized and will be able to use a common set of resource files.

We decided to spend some time to write tests for the code to catch bugs before deployment. FlexUnit came packaged with the Adobe Flex SDK, which was used to create a unit test suite. These unit tests were created by making a test case for every piece of testable code. The unit test case will either pass or fail depending on whether the code being tested is behaving as it should. If a test were to fail, we would know precisely which part of the application is not working properly and be able to fix the problem before it reached the customer. When all test cases pass inspection, we can deploy the application with a high degree of confidence that it is functioning properly.

4.1 Reflections on the Software Engineering Experience

I have learned a plethora of knowledge from this experience. Before I started this project, I had never even heard of the Adobe Flex SDK or Flash Builder. Now, I feel very confident in my ability to use these technologies. There are many aspects of software development that I learned from this project that could not have been taught in a classroom. My team had to execute the software development from start to finish. We had to research technologies to best fulfill the requirements. We had

to learn new programming languages and understand new programming concepts to meet these requirements. We made a time-line and set milestones to keep us on track. I have heard of the software development lifecycle in theory, but could never fully understand it until I experienced it firsthand.

This project has opened my eyes to the big picture of software development. In industry, large software project are created by teams that must plan to divide work in a reasonable fashion and collaborate often to remain on the same page. Before this, I had never created a real, usable software application. I now feel confident in my ability to take a list of requirements and be able to develop a software application from beginning to end.

LIST OF REFERENCES

[1] Enporion solutions. http://www.enporion.com/solutions/solutions.html.

[2] Andrew Bartels. *ePurchasing Software Market*. 2007.

[3] Enporion history. http://www.enporion.com/about/history.html.

[4] Enporion mission. http://www.enporion.com/about/mission.html.

[5] Brian and Detlor. The corporate portal as information infrastructure: towards a framework for portal design. *International Journal of Information Management*, 20(2):91 – 101, 2000.

[6] S. L. Roberts-Witt. Making sense of portal pandemonium. *Knowledge Management*, 1999.

[7] Abdullah S. Al-Mudimigh and Zahid Ullah. Effective implementation of portals: Best practice model. *International Journal of Business and Management*, 6(2), 2011.

[8] Andrew Sears and Julie A. Jacko. *Human-Computer Interaction: Design Issues, Solutions, and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.

[9] Adobe Open Source. About flex sdk. http://opensource.adobe.com/wiki/display/flexsdk/About.

[10] Adobe. Adobe flash builder family. http://www.adobe.com/products/flashbuilder.html.

[11] Flex Team. Introduction to localization. http://livedocs.adobe.com/flex/3/html/help.html?content=l10n_2.html.

[12] Building multilanguage flex rias. http://www.gridlinked.info/amazing-i18n-solutions-for-flex-3-4/.

[13] Microsoft. Unit testing. http://msdn.microsoft.com/en-us/library/aa292197

[14] Roy Osherove. Write maintainable unit tests that will save you time and tears. http://msdn.microsoft.com/en-us/magazine/cc163665.aspx.

[15] Adobe Open Source. Flexunit. http://opensource.adobe.com/wiki/display/flexunit/FlexUnit.