Software Design Document for SCRUM Project Management System

Andrew Johnson and Max Warren

October 17, 2011

Contents

Revision History

Name	Date	Reason for Changes	Version
Max Warren	10/10/11	initial draft	1.0 draft 1
Max Warren	10/15/11	added sample screenshots	1.0 draft 2
Andrew Johnson and Max Warren	10/17/11	changes after inspection with team	1.0 approved

1 Introduction

The purpose of this document is to provide those persons tasks with developing, maintaining, or supporting the SCRUM Project Management System with detailed information concerning the setup, design, and deployment of the SPMS.

This document includes a high-level architectural overview of the SPMS, followed by a more detailed description of each component. Instructions for deploying the SPMS are also included.

1.1 References

1. Johnson, Andrew and Max Warren. *Software Requirements Specification for SCRUM Project Management System*
2. Microsoft. *WPF Apps with the Model-View-ViewModel Design Pattern*. `http://msdn.microsoft.com/en-us/magazine/dd419663.aspx`.
3. Schwaber, Ken and Mike Beedle. *Agile Software Development with Scrum*. ISBN 0130676349.

2 Architectural Overview

2.1 Client-Server Architecture

The SCRUM Project Management System is designed as a client-server application. The SPMS client shall be a Windows desktop application. Each user of the SPMS will have a copy of the client application. The SPMS server will be a database server. Each company deploying the SPMS will have one SPMS database server on its corporate Intranet. The server shall support access by multiple clients concurrently:

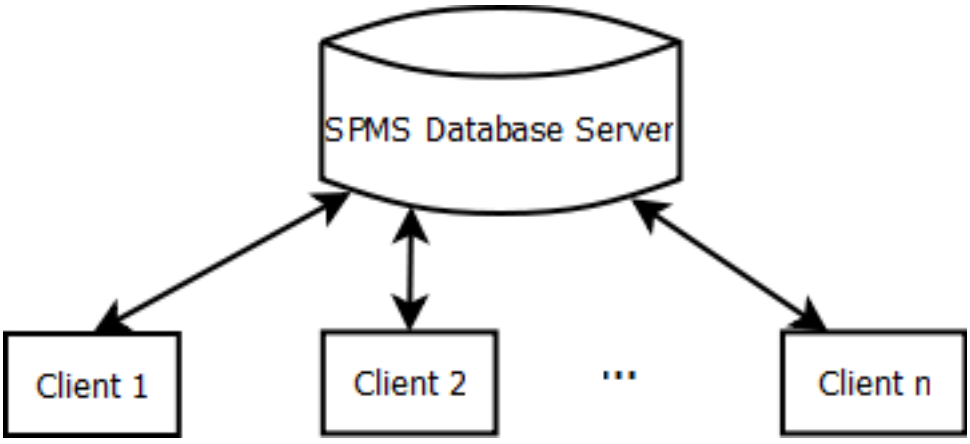

Figure 1: Client-Server Model

Before a user can access any of the data in the database, the client shall authenticate the user with the server. The client shall first attempt to authenticate the user with the AD domain controller. If this is unavailable or fails for any reason, the user will be asked to provide a

username and password. The client will then attempt to authenticate with the SPMS database using the provided credentials.

This sequence diagram shows a user successfully authenticating with AD:

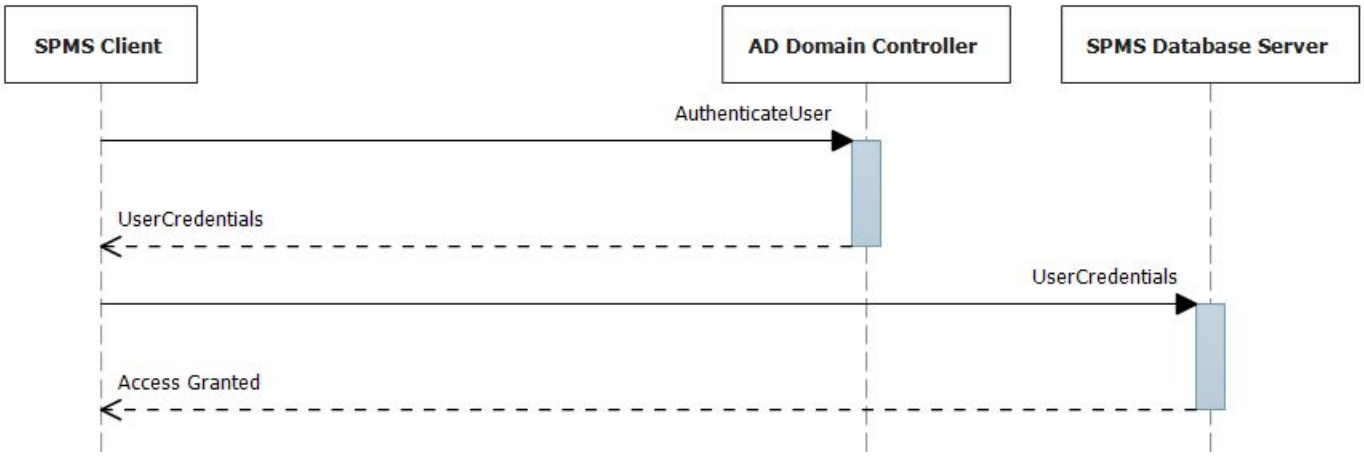

Figure 2: AD Authentication

This sequence diagram shows AD authentication failing, requiring the user to provide a username and password:

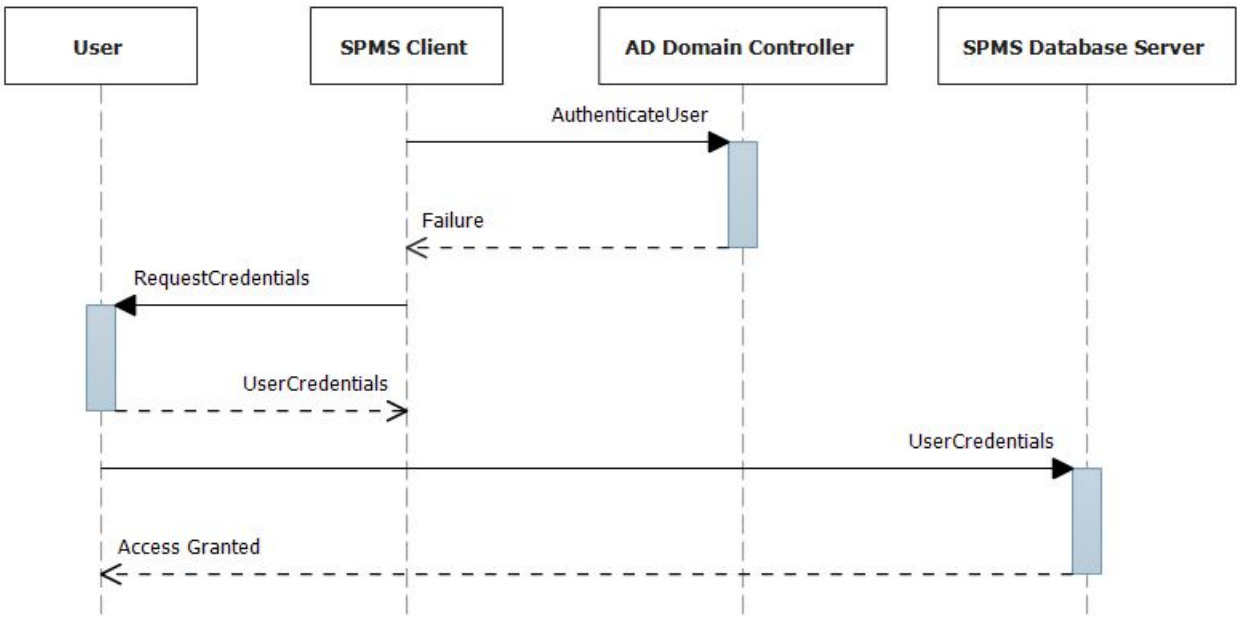

Figure 3: SPMS Database Authentication

The client-server model best represents how users of the SPMS interact with the data. Each company has some set of projects managed by the SPMS. This information is the same for all

employees of the company, but they may interact with it in different ways.

2.2 Model-View-ViewModel Pattern

The MVVM design pattern was developed by John Gossman at Microsoft specifically for use with WPF (Windows Presentation Foundation) applications[2]. An application designed using this pattern consists of three components: the model, the view model, and the view. The SPMS client shall be designed using this pattern.

The model is the component that represents the state of the application. In the SMPS, the model is a data access layer, as the state of the application is contained entirely within the database. This data access layer is responsible for requesting information from the database and for making changes to this data. The model is not aware of the other two components.

The view is the user interface of the application. The view is entirely unaware of the existence of the model. Moreover, the view has no functionality other than displaying the GUI.

The view model is responsible for translating application state as represented by the model into a form that can be displayed by the view. Any interactions a user has with program state are carried out through the view model.

This diagram shows the interactions between these components:

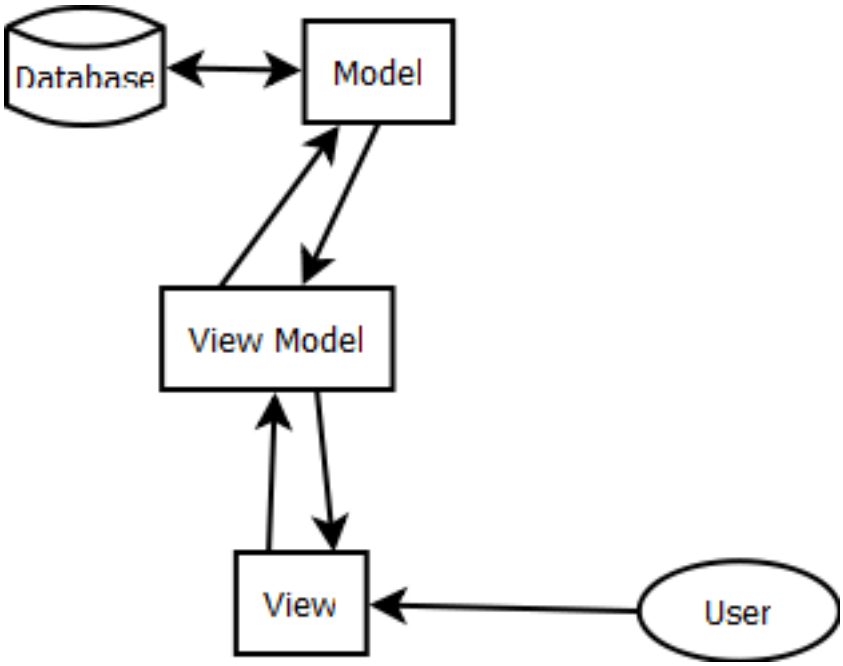

Figure 4: MVVM Component Interactions

The design of each component is discussed in greater detail in the rest of this document.

2.3 Technical Platform

The SPMS database shall be deployed on a server running Microsoft SQL Server 2008. The SPMS client shall be developed in C# for the .NET Framework 4. The user interface for the client shall be created using WPF.

3 Data Model

3.1 Overview

The database for the SPMS contains data on all the projects managed by the system. Since all of these projects are using the SCRUM development methodology, the projects must be represented in a specific form[3]. Each project managed by the system is broken down into a series of sprints and a backlog. Each sprint (including the backlog) contain some number of user stories. Each user story is then broken down into a series of tasks.

Furthermore, the database must model the users and teams responsible for developing these projects. Any employee who will use the SPMS client must have an entry in the database so that they can be assigned tasks and be authenticated by the database if necessary. All project teams must also be represented in the database.

3.2 Tables

3.2.1 Project

The `Project` table will represent projects managed by the system. The table shall have the following columns:

- project_id: A unique integer identifying the project. This column shall be the primary key for this table.
- project_name: A string giving the name of the project.
- start_date: The date on which the project entered development.
- end_date: The date on which the project was completed. This column may be null.
- owner: The ID of the user who owns the project. This is a foreign key relationship with the `User` table.
- team_id: The ID of the team responsible for developing the project. This is a foreign key relationship with the `Team` table.

3.2.2 Sprint

The `Sprint` table will represent sprints making up project managed by the system. The table shall have the following columns:

- sprint_id: A unique integer identifying this sprint. This column shall be the primary key for this table.
- sprint_name: A string giving the name of the sprint.
- start_date: The date on which this sprint begins.
- end_date: The date on which this sprint ends. This column may be null.
- project_id: The ID of the project to which this sprint belongs. This is a foreign key relationship with the `Project` table.

3.2.3 Story

The `Story` table will represent user stories that belong to a sprint or to the project backlog. The table shall have the following columns:

- story_id: A unique integer identifying this user story. This column shall be the primary key for this table.
- priority_num: An integer representing the priority of this user story. Smaller numbers have higher priority.
- sprint_id: The ID of the sprint to which this user story belongs. This is a foreign key relationship with the `Sprint` table.
- text: The text of this user story.

3.2.4 Task

The `Task` table will represent tasks that make up a user story. The table shall have the following columns:

- task_id: A unique integer identifying this task. This column shall be the primary key for this table.
- story_id: The ID of the user story to which this task belongs. This is a foreign key relationship with the `Story` table.
- text: The text of this task.
- owner: The ID of the user who is responsible for completing this task. This is a foreign key relationship with the `User` table and may be null.
- type: A set of flags representing the type of this task, such as development, QA, or documentation.
- size_complexity: An integer representing the size complexity of this task.
- business_value: An integer representing the business value of this task.

- completion_date: The date on which this task was completed. This column may be null.
- state: A set of flags representing the state of this task, such as blocked, unassigned, in progress, or completed.

3.2.5 Team

The `Team` table will represent a project team. The table shall have the following columns:

- team_id: A unique integer identifying this team. This column shall be the primary key for this table.
- team_lead: The ID of the lead for this team. This is a foreign key relationship with the `User` table.
- manager: The ID of the manager of this time. This is a foreign key relationship with the `User` table.
- team_name: A string giving the name of this team.

3.2.6 User

The `User` table will represent a user of the SPMS. The table shall have the following columns:

- user_id: A unique integer identifying this user. This column shall be the primary key for this table.
- password: The SHA1 hash of the user's password.
- team_id: The ID of the team to which this user belongs. This is a foreign key relationship with the `Team` table.
- role: A set of flags representing the user's role, such as development, QA, documentation, or management.
- name: A string giving the full name of the user.

3.3 ER Diagram

This ER diagram shows the relationships between the entities represented by our database:

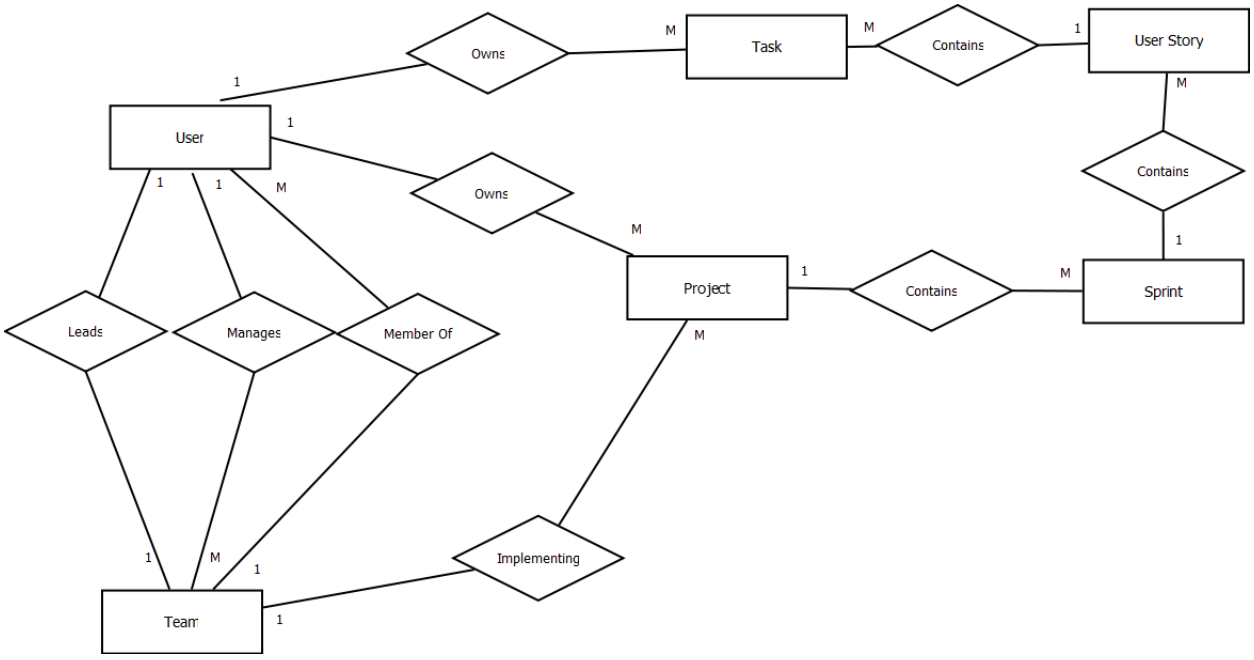

Figure 5: ER Diagram

4 Data Access Layer

4.1 Overview

The data access layer is the "model" component of the the SPMS client. This component is reponsible for getting the data requested by the view model from the database and updating the data in the database when requested to do so. This component has no knowledge of any components other than the database server.

4.2 Design

This component shall be a single static class, with a set of static methods to access and update the data in the database. Using a static class ensures that each client will only open one connection to the database, which will reduce the load on the database sever. An alternative design here would have been to make the data access layer a singleton class, but the static class implementation will be simpler and easier to understand.

For accessing the database LINQ-to-SQL shall be used. LINQ-to-SQL removes the need to embed SQL statements in the program code and provides some protection against SQL injection. Furthermore, it automatically translates the database types into the object model used by the system.

4.3 Interactions with Other Components

The data access layer interacts with the database through LINQ-to-SQL, as described above. LINQ-to-SQL generates a object model in C# to correspond to the data model in the database. This diagram shows how the relationship between the data access layer and this schema:

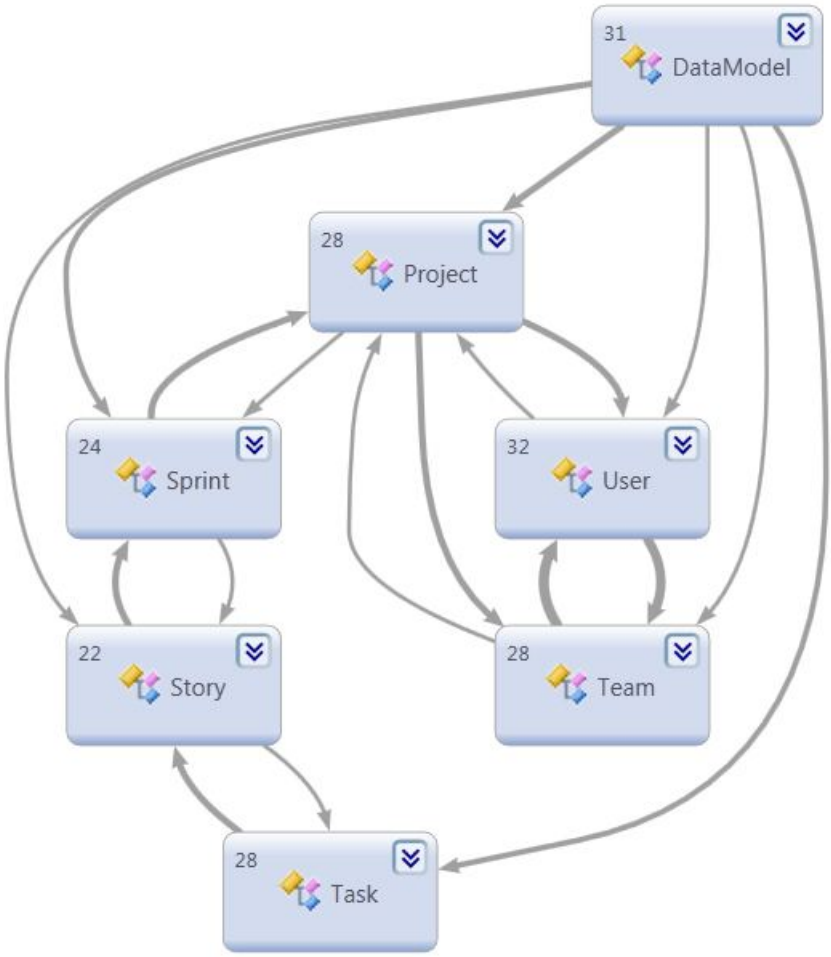

Figure 6: Data Access Layer and Schema

The data access layer is unaware of the existence of any other components. It exposes static methods to view and update data from the database that can be used by other components.

4.4 Error Handling

If an error occurs during some operation, the data access layer will inform the caller of this. The data access layer will not make any attempt to retry the operation or recover the data.

5 View Model

5.1 Overview

The view model component is responsible for translating the data produced by the data model into a form that can be displayed by the view and for translating requests from the user to change data into calls to the data access layer. The view model has knowledge of the data access layer, but is independent of the view itself.

5.2 Design

The view model is designed to be instantiated by the view. The functionality of the view model has two components. First, the view model provides data to which the view will bind. The view model will expose properties to this end. Binding to these properties will not necessarily cause the view model to fetch the newest data from the data access layer. Since the data is not automatically updating, the user can examine the data without the distraction of seeing changes from other clients. If the view wishes to update the data at a certain point, it must call an update method in the view model.

The view model also allows the view to request that data be changed. The view model will provide methods for performing the updates described in [1].

5.3 Interactions with Other Components

This diagram shows the interactions between the view model and the data access layer:

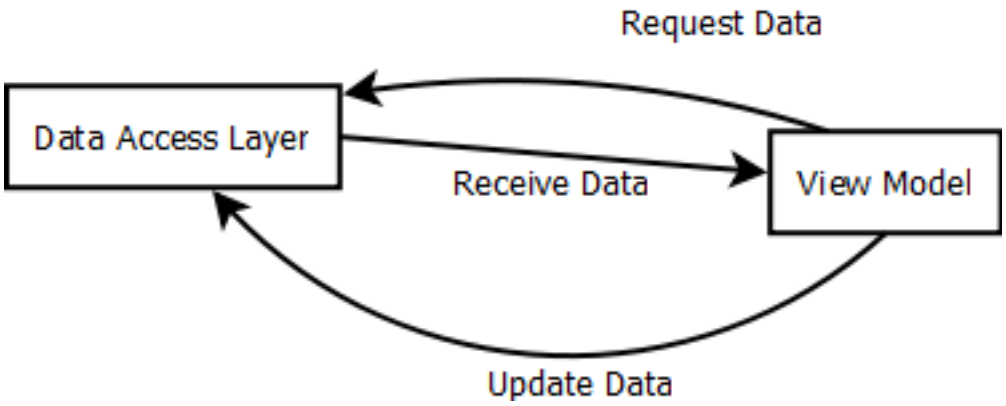

Figure 7: Interaction Between View Model and Data Access Layer

5.4 Error Handling

There are two kinds of errors that can occur in the view model. First, an error could occur in the data access layer. If such an error occurs, the view model will save the data the user was working on to a local text file. The user can continue to use the client. The next time the user starts the client, they will be prompted to recover the data from that local file.

An error would also occur if another client had modified the same data while this client was processing it. In this case the view model will display the new data to the user and ask them to retry their changes. This sequence diagram illustrates this sequence of events:

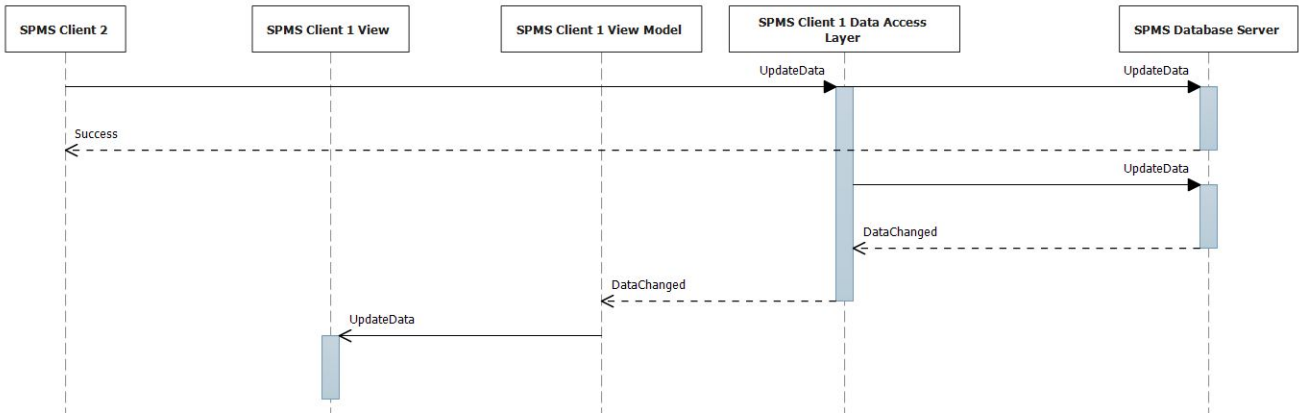

Figure 8: Concurrent Data Modification Error

6 User Interface

6.1 Overview

The user interface is the "view" component of the client. This component is responsible for drawing the client's graphical interface and responding to user interaction.

6.2 Design

The user interface is built around a two-panel design. When the user starts the client, they will be presented with a list of projects that their team is working on and a list of tasks they are responsible for:

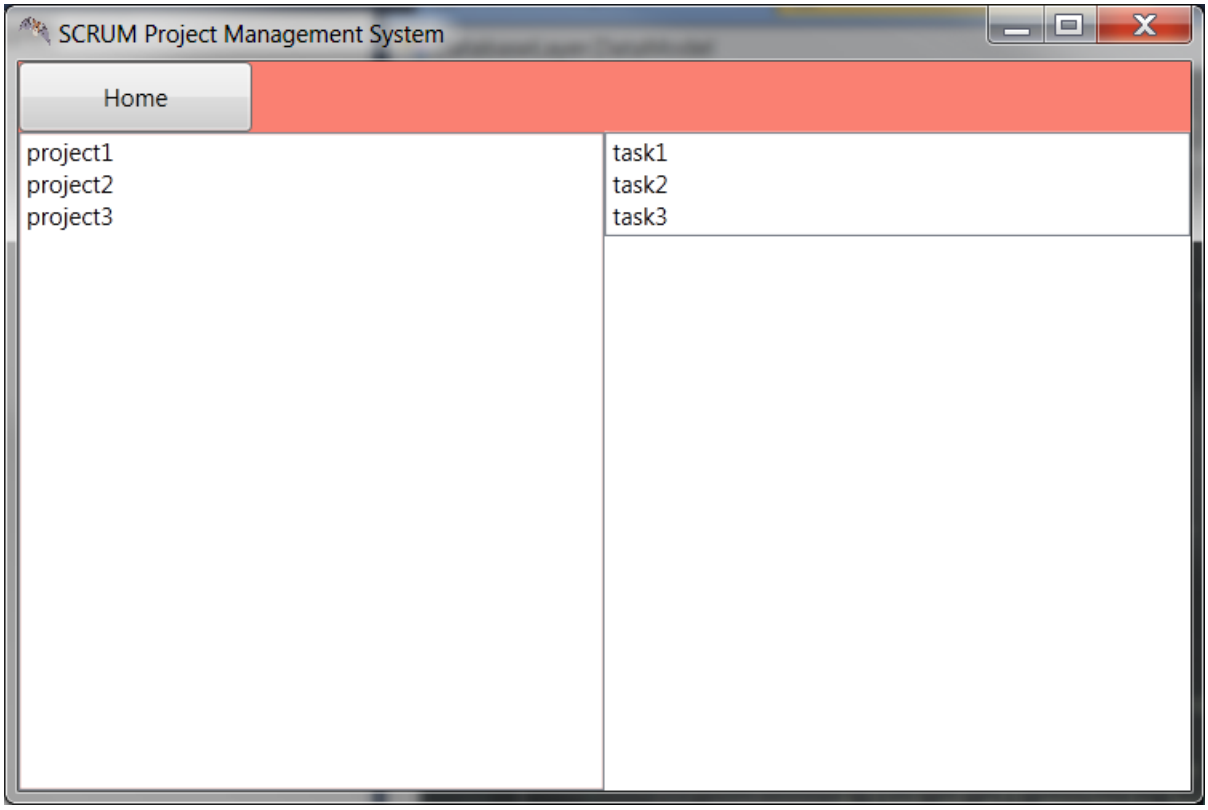

Figure 9: Home Screen

From this point they can view the data on these projects or tasks. At any point they can be viewing a list of projects, sprints, user stories, or tasks. In each case, the left pane will be a list of the sub-elements of the current object and the right pane will have the information on the current object. For example, here the user is looking at a project:

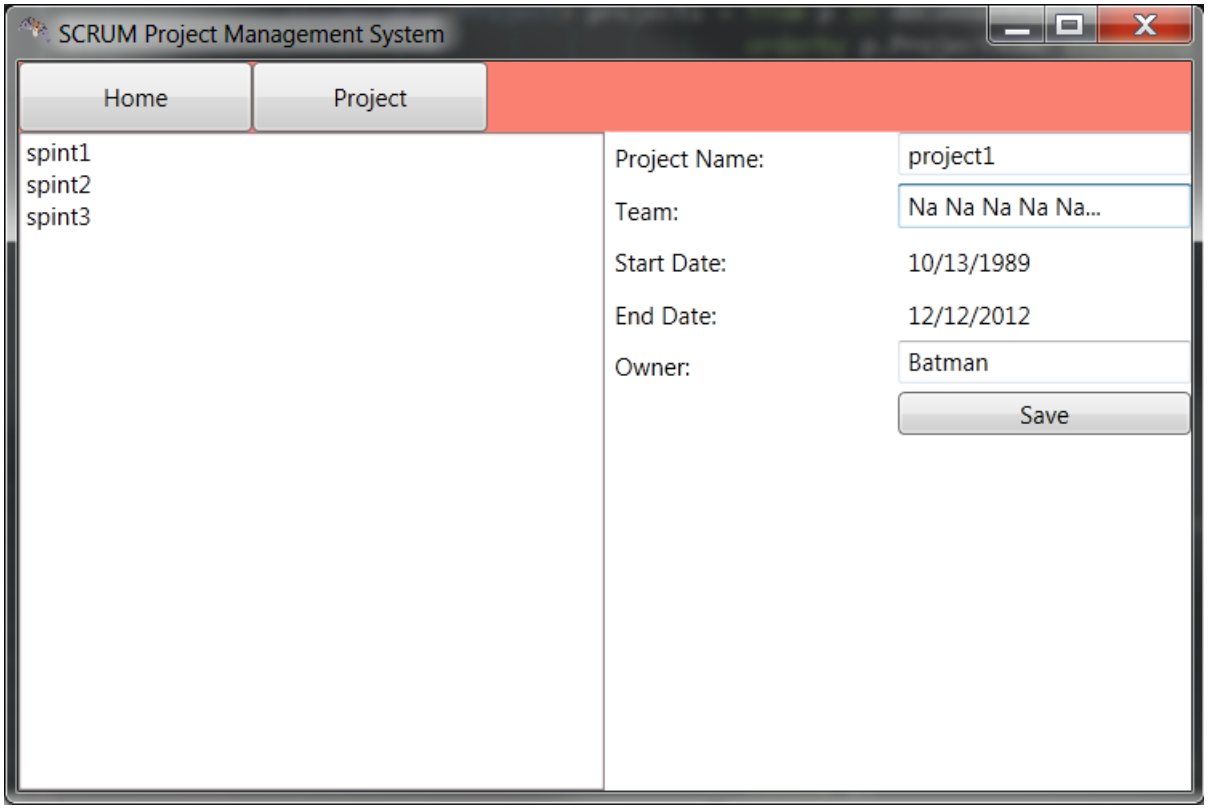

Figure 10: Two Panel Example

The user will be able to add a new instance of a type from the appropriate left pane.

"Breadcrumbs" will be placed at the top of the window to allow the user to navigate through the project hierarchy. Clicking on a breadcrumb will display to the user the information about the most recently viewed object of that type. For example, here you can see that the user has navigated to the tasks for a particular story. All of the breadcrumbs are visible:

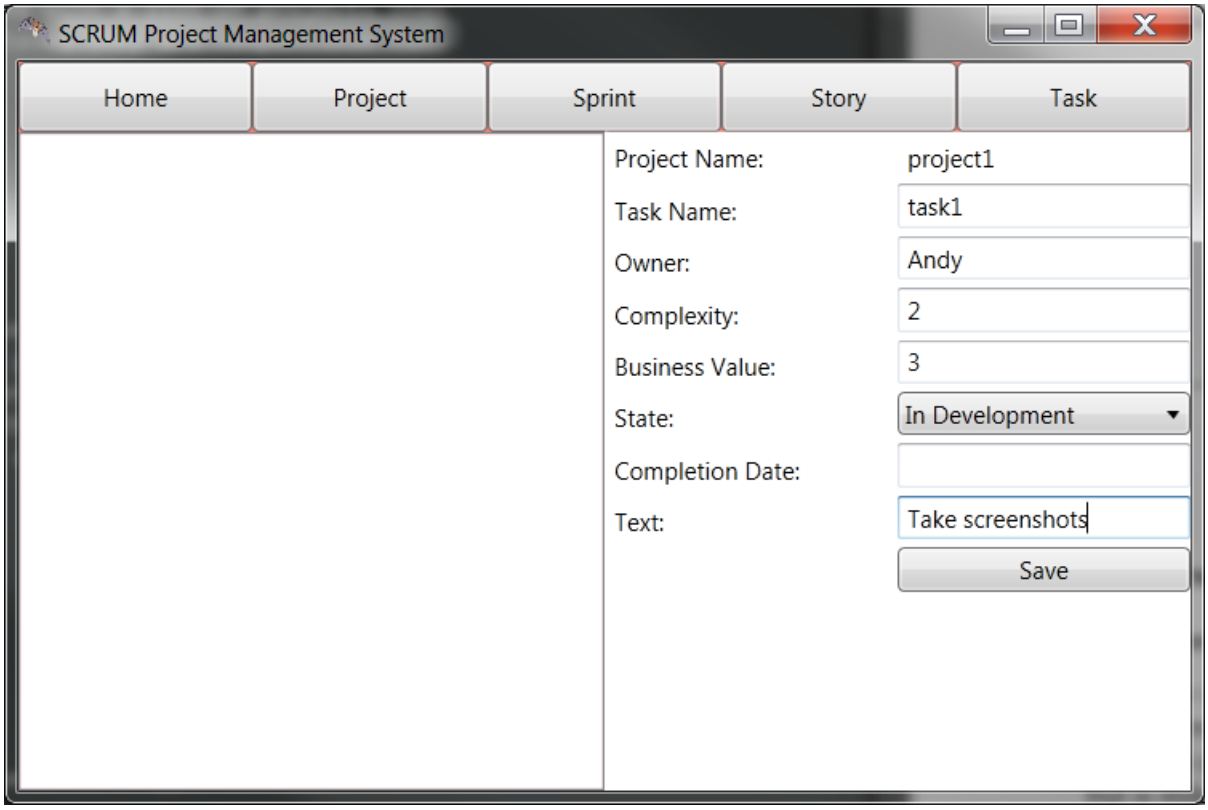

Figure 11: Breadcrumbs Example

If the user is a manager or otherwise has the necessary permissions, a menu will be visible at the top of the window allowing the user to add or change teams or team members.

6.3 Interactions with Other Components

The user interface will instantiate the view model and use it as its `DataContext`. It will then bind to the properties of the view model to display data and call the methods of the view model to update or change the data. This diagram shows the basic interaction between the user interface and the view model:

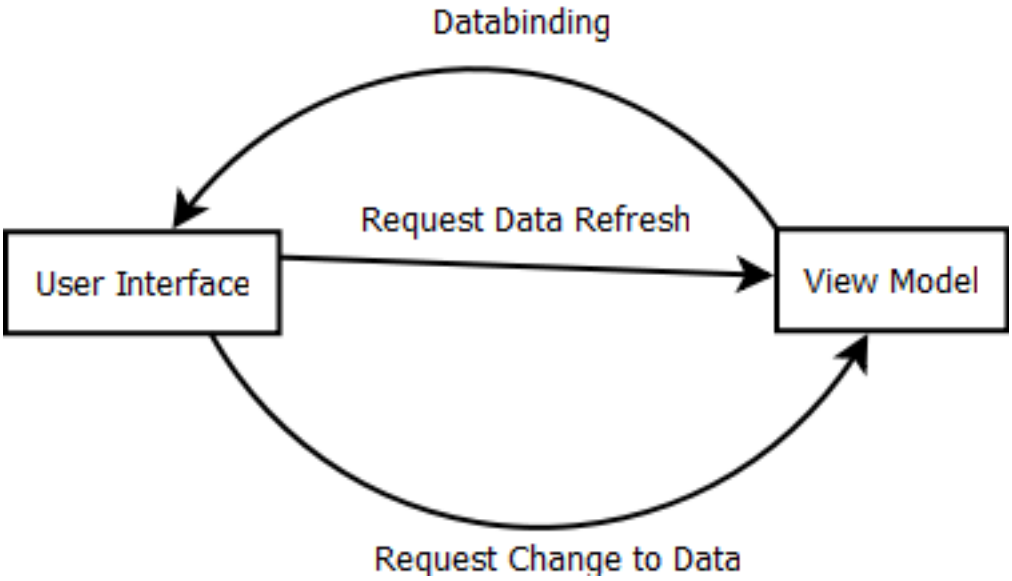

Figure 12: Interaction Between the User Interface and the View Model

The use of databinding ensures that the view will automatically update as the view model changes the data. This avoids errors where the data displayed in the UI does not reflect the actual state of the program.

7 Deployment

7.1 SPMS Server

The SPMS server should be deployed before any instances of the SPMS client are deployed. Once the server is provisioned, install and set up Microsoft SQL Server 2008. Microsoft SQL Server scripts shall be provided to create the tables described in Section 3.

7.2 SPMS Client

The SPMS client can be installed with the provided Windows installer. If required, the client can also be installed using Microsoft ClickOnce deployment. During installation the address of the database server must be provided.