Company Name: Corporate

Author(s): Brad Bergeron, Stephanie Smith, Yuan Tu

Version #: 0.3

Date created: 10/7/2015

Date updated: 10/24/2015

Software Requirements Specification

Table Of Contents

Introduction 2

Product Overview

Definitions

Nonfunctional Requirements 3

Functional Requirements

References 4

Index 4

**Audience:** Raman Aravamudhan, Aashay Borgaonkar

**Distribution List:** Corporate, Raman Aravamudhan, and Aashay Borgaonkar

**Scope of Document:** This Document will cover all of the Software requirements for the project

**Introduction**

This is all the different software we had to build to complete the user requirements for the project. The Webpage has Views that the user sees, Controllers that handle communication, Models that take care of the functionality, Tables in the server, and Unit tests to test our webpage.

**Product Overview**

**Definitions**

A **user** is anyone who would use the product. A user would either be a team administrator, leader or member.

**Nonfunctional Requirements**

**Views**

**RegistrationEntities1 Database**

This is the database being used for the application. The database consists of 2 different categories of tables: Users and Process Models. The User tables store information about the applications users. These sets of tables will be more dynamic because users will be constantly logging in/logging out and creating accounts. The second set of tables holds information about the process models. These sets of tables will be more static.

**User Tables**

There are three different types of users for this application: administrators, leaders, and members. Therefore, there are three different tables to the user information. All of these essentially hold the same information.

Here are all of the fields they share:

* UserName \*
* Email \*
* ConfirmEmail \*
* Password \*
* FirstName \*
* LastName \*
* MiddleName
* PhoneNumber
* OptionalPhoneNumber
* Address
* Gender
* Birthdate

These fields appear in the table in this order but are arranged from left to right (instead of top to bottom). It is not necessary for all of these fields to be filled out (i.e., the values can be null). Only the fields with the \* by them have to be filled out.

**member Table**

An entry in the “member” table is created everytime a user creates an member account. A user can create an member account by making sure “member” is selected in the dropdown before clicking the signup button.

The only other entry that is unique to a member is an I.D.. The I.D. is generated by the business logic, so the user doesn’t have to worry about it when filling out the form. Anyone using this application can register as a user.

**administrationV2 Table**

An entry in the “adminstrationV2” table is created everytime a user creates an admin account. A user can create an admin account by making sure “admin” is selected in the dropdown before clicking the signup button.

In addition to the to the 12 fields specified above, the admin table has two other fields:

* I.D.
* admin key

The I.D. is generated by the business logic, so the user doesn’t have to worry about it. The correct admin key MUST be provided when a user is signing up as an admin otherwise they cannot sign up.

**Leader Table**

The leader table is very similar to the admin table. An entry in the “leader” table is created everytime a user creates an leader account. A user can create an leader account by making sure “leader” is selected in the dropdown before clicking the signup button.

In addition to the to the 12 fields specified above, the admin table has two other fields:

* I.D.
* leader key

The I.D. is generated by the business logic, so the user doesn’t have to worry about it. The correct leader key MUST be provided when a user is signing up as an leader otherwise they cannot sign up.

**Process Model Tables**

**Questions’ Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Question I.D.** | **Question** | **Category** | **Question Type** | **Answer** |

The primary key for this table will be Question I.D. Question I.D. should go from 0 to 150. The Category attribute describes what the category of the question being asked. For this data set, there are a few unique subsets that all questions fall under, such as, requirements, team size, expenses, management, and process. The next attribute is Question.

Here is example of a question:

The requirements are completely understood.

or

How large is the project?

Notice for true and false and multiple choice you must add the appropriate punctuation, and on true or false you do not specify ‘true or false’ in the Question attribute. Question should be a string of the greatest length allowed. Attribute Question Type, can be of type: true or false, multiple choice, or short answer. This should also be a string. Finally, the Answer attribute is the answer to the question the user inputted. Initially, all of these should be NULL. After the user submits the form, they should be all be filled with strings. If any value in the Answer column, is still NULL after the user has submit the form, they will be redirected back to the questions page and prompted to finish answering the questions.

**Multiple Choice Response Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Question I.D.** | **Response #1** | **Response #2** | **Response #3** | **Response #4** | **Response #5** | **Response #5** |

There needs to be a specific table to yield multiple choice answers. This is because with questions of type ‘True or False’ we know that there are only two possible answers, and we know what they are. For short answer, there we only need to generate a textbox. However, for multiple choice, we do not know what kind of question responses there will be. This elicits the need for such a table. There is a maximum of 6 different responses, which should be sufficient for the 6 different process models. All 6 columns do not have to be utilized, but possible answers should be begin at column Response #1 ascend by one from there. Once there are no more responses to the question, the Response #\* columns should contain a value of NULL.

**Process Models’ Table**

|  |  |  |
| --- | --- | --- |
| **Question I.D.** | **Desired Answer** | **Priority** |

For each process model, a particular question has a desired answer.

To illustrate:

Question: The requirements are completely understood (True or False)

For, for Waterfall, the desired answer to this question is true. Furthermore, a priority is associated with each desired answer. For the example above, Waterfall’s priority would be 5. A given process model can have a priority ranging from 0-5. A priority of 0 means that is does not matter at all to this particular process model, and a priority of 5 means that is critical to this process model. That is, *if a process model has a priority of 5 for a given question. The actual (user) answer must match the desired answer or else that process will be thrown out of consideration completely.*

Here is another example for Waterfall:

Question: You are a consultant company (True or False)

Desired Answer: False

Priority: 3

The desired answer is false because if your are a consultant company, you have to communicate and tend to an external client. This may mean that you have less control of the requirements. However, not catering to an external client is not critical to waterfall, so it only has a priority of 3.

There will be 6 process model tables with these columns: Waterfall, Iterative Waterfall, COTS, RAD, Agile, and ???. They can all be associated with one another and the Questions Table by the Questions I.D. attribute. The primary key for this table will be the Question I.D.

**Process Model Scores’ Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Waterfall Score** | **IterativeWaterfall Score** | **Agile Score** | **???**  **Score** | **COTS Score** | **RAD Score** |

This could have a lot of null values though...maybe do:

|  |  |
| --- | --- |
| **Score** | **ProcessModel** |

There shouldn’t be any null fields

The purpose of this table is to hold points. Each entry in database contains the a score for a time a given process model won.

Entries will be added to this table in the following way:

The developers will ‘train’ the scoring algorithm (discussed more in the functional requirements) by telling the application which process model should ‘win’ (i.e. be selected) for this survey submission. After the survey is submitted, and no blatant errors were made when filling it out, the algorithm scores the process model and stores that score in the database under the appropriate process model.

Later, when a user submits the survey, the scoring algorithm will compare the surveys score to the winning scores of all of the different process models. Whichever process model scores are closest (the criteria for closest is discussed in the functional requirements) to the actual score will be selected as the winning process model.

**Security**

The two basis types of security will be implemented in this application:

1. Password Encryption: Before storing the user’s password in the database, it will be encrypted, so it cannot be easily read.
2. Email verification: In order to verify that the user is who they say they are, they must verify their account by following a link to the application sent from the application. The application will use the email provided by the user. The user cannot login until they verify their account through email.

**Functional Requirements**

**Login Capability**

The login View is the index (initial view) that is loaded when a user navigates to the website. The name of this View is “Index”. Index is located in the Views/Accounts directory because it is apart of the account related functionality (i.e., signing in or creating an account). Index is connected with the rest of the application through AccountController, and is connected the to Account Model through model/view binding.

The login capability is a form consisting of two text boxes. When a user inputs their username and password into the text boxes,they are setting the properties “Username” and “Password”. After the user submits the form by clicking the Login button, the submission is sent to the controller method AccountController/SignIn.

SignIn takes a single parameter: an Account object name “User”. It then calls the Account model method “Create”. The method “Create” simply checks to see if the appropriate fields (i.e., the username and password fields) have been filled out (i.e, do not equal NULL), and then calls the another Account model method that checks the registration database to see if:

1. The username exists AND
2. if the username exists, do the passwords match up

The find method checks each table (i.e., memberTable, administrationTable, and leaderTable”, and either returns a String containing the type of user they are (i.e., Admin, Leader, Member), or it returns a String containing the word “Fail” because the Login failed.

**Sign Up Capability**

The user can navigate to the sign up view through the login (index) view. If the user clicks the ‘sign up’ button, they are redirected to a page that contains all of the information they should fill out in order to sign up. A user can sign up as an admin, leader, or member. The user can sign up as an admin or leader if they have the required keys. Any user can sign up as a member. The user selects who they want to sign up as by selecting the appropriate category from the drop down menu on the index page before hitting the signup button.

The models associated with this capability are: Member, Leader, and Admin.

The Member, Leader, and Account model hold all of the properties belonging to each of these objects. These properties are what get added to the database when the Init method is called. These models also contain the method to validate the user and to determine if the user already exists.

The controller methods associated with this capability is: AccountController: Signup, CreateMember, CreateLeader and CreateAdmin. When the user hits the signup button they are routed to the SignUp controller which loads the appropriate view (this depends on what was selected in the drop down menu). After the user submits the form, they are routed to the appropriate create controller method. These methods call a model method to verify if the user filled out the form correctly. If the form was filled out correctly the controller method calls a model method to store the user in the database. and loads the login page. If the form was not filled out correctly, controller loads the previous form and prompts the user to fill it out again.

The views associated with this capability are: SignUpAdmin, SignUpMember, and SignUpLeader. These views hold a form that is made up of 12/13 text boxes. The form is bounded to the Member/Administrator/Leader model, so when a user fills out the form, it is setting the model’s attributes. When the user hits the ‘Sign Up’ button they are directed to the appropriate create controller.

**Email Verification**

**View Projects**

**Add Projects**

**Process Model Selection**

After the user submits the answers all of the questions and submits the questions form. ProcessModel model will began analyzing the data to make an educated decision on which process model the user should select.

To achieve this goal, the software must first eliminate all Process Models that had an incorrect answer with their priority-5 questions. After this, the software will utilize an algorithm called “K-nearest neighbor” to find the closest match for any of K-remaining process models.

In order to use K-nearest neighbor, it must be trained, here is how we will go about the training process:

We will create a set of data points. Each data point is an array of total question size (150, in this case) where the index is the question I.D., and the value is the numerical representation of the question answer, that will store the answers to all of the questions. The numerical representation of the question is the priority if the question was answered correctly, otherwise, it’s the priority\*-1. The set will be of at least size 20. The goal of the training data is to form well defined clusters that new user data can then be compared too.

Add Project Capability

* leader creates project
* adds member to project
* assigns roles to members

Assign Role Capability

Admin

- admin can see calendar, team leader, and team member

- can demote leaders (promote members?)

Calendar

* anyone can see the calendar
* anyone can add/delete
* event name, start time, end time, who it’s delivered too.
* no scheduling conflicts
* team leader has to add approve/ dissaprove
* 1 calendar per process
* continuous

The numerical representation may need to be adjusted, but let’s just make it the priority for now.

|  |  |  |
| --- | --- | --- |
| Software ID | Software | Type |
| 1 | Login | View |
| 2 | Team Member Sign up | View |
| 3 | Team Leader Sign up | View |
| 4 | Admin Sign up | View |
| 5 | Team Member | Database table |
| 6 | Team Leader | Database table |
| 7 | Admin | Database table |
| 8 | Accounts | Controller |
| 9 | Insert SQL | Model |
| 10 | Select SQL | Model |
| 11 | Completeness | Unit Test |

**References**

<http://www.cse.chalmers.se/~feldt/courses/reqeng/examples/srs_example_2010_group2.pdf>

<http://www.codeproject.com/Articles/32970/K-Nearest-Neighbor-Algorithm-Implementation-and-Ov>

No References

**Index**

Author’s 1

Iteration 1 3