**Software Requirements Document for Volunteer Work Schedule**

**Prepared for C-S 741: Software Engineering Principles**

**By**

**Abe Gustafson**

**Date: Oct 3, 2016**

**Version 1.0**

**Table of Contents**

1. **Introduction 3**

**1.1. Purpose 3**

**1.2. Scope 3**

**1.3. Problem Description 3**

**1.4. Assumptions 4**

**1.5. System Characteristics 5**

**1.6. User Characteristics 5**

1. **References 5**
2. **Functional Requirements 6**

**3.1. Functional requirements for all types of users 7**

**3.2. Functional requirements for users 9**

**3.3. Functional requirements for administrators 13**

1. **Non-functional requirements 14**

**1. Introduction**

**1.1. Purpose**

This document describes the requirements specification of software for a volunteer work schedule system to be used by anyone. It is based on the format from a simplified version of IEEE Standard 830-1998 for writing software requirements specification.

**1.2. Scope**

The software product described in this document is a prototype to be used by anyone who wants to volunteer for an event, or who wants to create their own events for people to volunteer for.

**1.3. Problem Description**

A number of non-profit organizations such as the Youth Hockey Association in Onalaska, Wisconsin uses a number of volunteer services for its day-to-day operations. The goal is to create a volunteer schedule site that would let users create their own events that other users could “volunteer” and sign up for. For example, on a game-day-event, the organization uses volunteers to help the game activities (scoring, refereeing, escorting players, canteen services, ticketing and so on), and all of the details are coordinated online. It will allow the users who create/customize these volunteer events to choose their own “volunteer positions” that they require for the event, along with other things like date, time, location, additional requirements, etc. This project aims at developing a web-based volunteer scheduling system. There will be different types of users and administrators. Users will sign up for different volunteer events, and also be able to create their own volunteer activities. Administrators will be able to do anything a user can do, while also being able to unblock accounts, remove people from events, remove events, and add new preset volunteer roles. There will be account registration for the users. This registration will consist of them entering their information (name, email, phone number), and then the system will generate a username for them. They will also be required to provide answers to different security questions that will later be required during login. Administrator accounts can only be created by other administrators. During login, users are prompted to enter username and password and one of their security questions. If they fail all questions they are locked out of their account. There will be a smart search function that helps users search for events they are interested based on different criteria (name, date, time, volunteer roles, location).

The product must be developed as a web-based application with a database-backend.

**1.4. Assumptions**

The following assumptions are made to support the development of the volunteer work schedule.

1. There will be two types of users in this system – users and administrators. Users can register, and once they have an account they can sign up for, and create events.
2. Users will have access to both sign up for volunteer events and create their own volunteer events.
3. The volunteer system will have its own internal database that holds all the information needed to run and maintain the site.
4. Each volunteer event has its own unique event ID attached with the user ID of the event creator. Other details like date, start/end time, location, and volunteer roles will be stored in the system.
5. The events are considered past events when the end time of the event is past the current time. Upon becoming labeled as a past event, it will no longer show up to anyone looking for it.
6. The details of a user to be stored in the system will include the first and last name, email, and phone number.
7. All the users of the system (Administrators and Users) will be required to use two pieces of information to login – username(email), and password. The username for both users and Administrators are their emails.
8. Every time a user creates an event, signs up for an event, changes an event, withdraw themselves from an event, or deletes the event, all the data about such events and users are recorded in the system database.
9. A user will be able to browse events, but will not be able to sign up for an event or be able to create an event if they are not signed in to the system.
10. Users will be able to see the events they are signed up for.
11. In the volunteer schedule system there will be no way to add other administrators other than by an administrator creating other admin accounts. This means that there must be at least one created when the system starts.
12. When an event has all its volunteer positions filled in one role, the role will close and not be available to anyone.
13. A user will not be able to sign up for more than one event that occur at the same time.
14. Reports may include number of current events, number of past events, volunteers in each event, number of events a user is signed up for, number of users, and potentially more. Details of analysis will be included during later stages of the development process.
15. Notifications between users will be internal in the case that an event is altered or deleted.

**1.5.**  **System Requirements**

The final product must run on a Mac or PC on either Chrome, Safari, Firefox, or Microsoft Edge.

**1.6.**  **User Characteristics**

No additional knowledge is required for users.

**2. References**

1. ISO/*IEEE Standard 29148-2011 Systems and Software Engineering – Life Cycle Processes – Requirements Engineering*, IEEE Publications, 2011.
2. Leszek A. Maciaszek, *Requirements Analysis and System Design (Third Edition),* Addison-Wesley, 2007.
3. Phillip A. Laplante, *Requirements Engineering for Software and Systems*, CRC Press, 2009.
4. K. Periyasamy, “Software Requirements Document for Transportation Service System,” *C-S 741: Software Engineering Principles*, Sep. 2016.

**3 Functional Requirements**

Each functional requirement is given in the following format:

*Index:*

*Name:*

*Purpose:*

*Input parameters:*

*Action:*

*Output parameters:*

*Exceptions:*

*Remarks:*

*Cross-references:*

*Index* refers to a unique index assigned to this functional requirement. It will be used for cross-referencing this requirement from other requirements and also from other documents such as the design document.

*Name* is a descriptive name given to this functional requirement. This name does not need to be unique.

*Purpose* is a short description (in a line or two) of the functionality. It is used to quickly understand the functionality and is also used to search the required functionality when browsed through.

*Input parameters* refers to a set of parameters that the given functionality accepts as input. These parameters are required in order to design and implement the current functionality. No type information will be included for parameters in this document.

*Action* refers to a set of activities or tasks to be completed in order to implement the given functionality. The ordering of these tasks is not specified in this document although some ordering may be explicit from the description. No implementation details must be given in this document.

*Output parameters* refers to a set of parameters that are output/exhibit by the current functionality, when implemented. No type information will be included for parameters in this document.

*Exceptions* refers to a set of conditions, each of which indicates a situation in which the implementation of the given functionality will stop. Notice that this column only lists a set of exceptions that might occur but does not suggest any action that must be taken when the exceptions occur. These actions will be included in the design document.

*Remarks* includes a set of comments that explain more about the functionality. It also describes hints to the designer and implementer that are suggested by the requirements analyst.

*Cross-references* refers to a set of other functional requirements that are related to the current functionality.

**3.1 Functional Requirements for all types of users**

*Index*: LN.1

*Name*:  Login

*Purpose*:  To let a user login into the system.

*Input parameters*: *User name(email)*, *Password*

*Action*: Verify that *User name* is valid and it exists in the system.

Verify that *User name* and *Password* match in the system data store.

Lock account if failed a third time.

Display the tasks the user can invoke.

*Output parameters*: None.

*Exceptions*: Invalid user name.

Invalid *password*.

*User name* and *Password* do not match.

*Remarks*: Format constraints for *User name* and *Password* will be dealt with during design and implementation.

*Cross-references*: None.



*Index*: LN.2

*Name*:  Logout

*Purpose*:  To logout from the system.

*Input parameters*: *None*.

*Action*: Ensure that the user has logged in.

Terminate the session of this user and display the home page.

*Output parameters*: None.

*Exceptions*: User has not logged in.

*Remarks*:  *None*

*Cross-references*: LN.1



*Index*: LN.3

*Name*:  Change Password

*Purpose*:  To change the Users Password

*Input parameters*: *Old Password, New Password, Confirmed New Password*.

*Action*: Ensure that the user has logged in.

Verify that the *Old Password* is Correct.

Verify that the *New Password* and *Confirmed Password* match the criteria, and are the same.

Update Users Password in the Database.

*Output parameters*: None.

*Exceptions*: User has not logged in.

*Remarks*:  *None*

*Cross-references*: None.



**3.2 Functional Requirements for Users**

*Index*: User.1

*Name*:  Add Event

*Purpose*:  To create a volunteer event that other users can sign up for.

*Input parameters*:  *Date, Address, City, State, Start and End time, Volunteer roles needed, Event Description, Event Name, Volunteer Role Descriptions.*

*Action*: Ensure that all of the input parameters are valid and that none of them are empty.

The system creates unique Event ID that is used to recognize the event.

Creates Role ID’s for each volunteer position.

Record the event information in the Volunteer Schedule database.

*Output parameters*: Event Created.

*Exceptions*: One or more input parameters are missing.

One or more input parameters are invalid.

*Remarks*: Event ID’s can be auto generated. Role ID’s are unique to each event, so they could simply be 1, 2, 3…etc.

*Cross-references*: None*.*



*Index*: User.2

*Name*: Browse available volunteer events

*Purpose*:  To view all open and current events. Filter between events based on

input parameters (if provided).

*Input parameters*: *Date, Address, City, State, Type of Volunteers needed, Event Name.*

*Action*: Show *Event Names, Times, Dates, Address, City, State, and type of Volunteers needed.*

Ensure that all of the input parameters are valid (if provided).

Populate the results with the most relevant event with respect to the Input Parameters given.

If no Input Parameters are given, show closest events with respect to current *Date.*

Allows users to search through all upcoming events, cycling through potentially all upcoming events.

*Output parameters*: A set of volunteer events each containing the following details: Event Name, Position, Description, Start and End date and times, Address, City, State.

*Exceptions*: One or more input parameters are invalid.

*Remarks*: Only show a certain number of events at a time, let users page through them. Make events only show the very relevant information while the rest is shown via expanding on the event. Past events and full events are not shown.

*Cross-references*: User.1



*Index*: User.3

*Name*:  Sign up for Volunteer Event

*Purpose*:  To let a User, sign up for a volunteer event.

*Input parameters*: *None*

*Action*: After browsing events and finding a position to sign up for, Pair the Users *User ID* with the *Role ID* (which is connected to the *Event ID*)

Check the database to verify that the *User ID* signing up for this event does not have conflicting with times of other Events they have signed up for by comparing the different users Role ID’s.

The parameters will be recorded and stored in the Volunteer Schedule database.

Remove the events respective Role ID (the one the user is signing up for) from available position status.

If this is the last position still open, close the events available status.

Send a confirmation notification with the event information to the user who signed up.

*Output parameters*: Sign up confirmation message

*Exceptions*: A user who is not logged in is not able to sign up.

*Remarks*: Essentially there are positions for each event that can fill up. Once someone signs up for a position, it should no longer be available to others trying to sign up.

*Cross-references*: User.1, User.2, User.8



*Index*: User.4

*Name*:  Withdraw from Volunteer Event

*Purpose*:  To let a user withdraw themselves from an event.

*Input parameters*: *Withdraw Reason*

*Action*: Ensure that the user trying to withdraw from the event is the same user that signed up for the event.

De-pair the *User ID* from the *Role ID* and set the *Role ID’s* status to “available”.

Send a notification to the owner of the Event that the current position is open again, along with the *Withdraw Reason* sent from the user.

Update the status of the position in the database.

*Output parameters*: Withdraw notification message to Event owner.

*Exceptions*: A user may only remove themselves from the event.

*Remarks*: Potentially have the input of the User ID, Event ID, and Role ID all happen behind the scenes to maximize ease of use.

*Cross-references*: User.2, User.3, User.8



*Index*: User.5

*Name*:  Edit Event

*Purpose*:  To let a user, edit the event information after they have created it.

*Input parameters*: *Changed information*

*Action*: Ensure that the event is not a past event.

Must ensure that the user with User ID is the one who created this event (cross check with currently logged in user information)

Receive event information from database using *Event ID*.

Ensure that the changes they make to the Event information are valid and not empty.

Notify all users affected by the change with the *Changed information.*

Update event information and store the changes in the database.

*Output parameters*: Notification to all users affected by the change.

*Exceptions*: User with user ID is not the owner of this event

A user cannot edit the date to a past date.

*Remarks*: *None*

*Cross-references*: User.1, User.8



*Index*: User.6

*Name*:  Delete Event

*Purpose*:  To let a user, delete an event they have created.

*Input parameters*: *None*

*Action*: Ensure that the event exists and that it is not a past event.

Ensure that the user with the User ID is the owner of the event or the currently logged in user is an administrator.

Notify all users affected by the deletion.

Remove all event information, and *Event ID,* And Volunteer positions from the database.

*Output parameters*: Notification to all users affected by the change.

*Exceptions*: Event has already been completed

A user may only delete an event that they have created

*Remarks*: *None*

*Cross-references*: User.1, User.8



*Index*: User.7

*Name*:  User Registration

*Purpose*: To register for an account

*Input parameters*: *First Name, Last Name, Phone number, Email.*

*Action*: Verify that all input is in a valid format

Create randomized password and email that password to the *Email* address provided.

Create new User ID and enter the input information attached with the User ID into the Volunteer System Database.

Redirect user to Login Page.

*Output parameters*: User Account.

*Exceptions*: One or more input parameters are of invalid format.

*Remarks*: None.

*Cross-references*: LN.1



*Index*: User.8

*Name*:  Send Notification

*Purpose*: To send a notification to another user either by the user or by the system.

*Input parameters*: *Destination Username, Message*

*Action*: Verify that the destination user account exists

Send *Message* to destination user

*Output parameters*: Notification

*Exceptions*: Cannot send a message if a username is invalid

Message must not be empty

*Remarks*: None.

*Cross-references*: User.6, User.5, User.4



*Index*: User.9

*Name*:  Receive Notification

*Purpose*: To receive a notification message from another user.

*Input parameters*: *Notification*

*Action*: Show message to user.

After the user has read the message, they can delete it.

*Output parameters*: None.

*Exceptions*: None.

*Remarks*: None.

*Cross-references*: User.8



**3.3 Functional Requirements for Administrators**

*Index*: Admin.1

*Name*:  Generate Report

*Purpose*:  To generate reports from the Volunteer Schedule system.

*Input parameters*: *Depends on the type of report*

*Action*: Depending on what is expected in the report, the report structure may vary. For all reports, the corresponding input parameters will be validated and then the report will be generated.

*Output parameters*: *Depends on the type of the report*

*Exceptions*: *Depends on the type of input parameters*

*Remarks*: Details will be determined by implementer.

*Cross-references*: None



*Index*: Admin.2

*Name*:  Unblock Account

*Purpose*:  To Unblock User Account after a User has been locked out for failing to login 3 times.

*Input parameters*: *None,*

*Action*: Sends the Locked User their password using the *User ID* to find their email address, and sends them the email.

*Output parameters*: *None.*

*Exceptions*: *None*.

*Remarks*: Could implement more security if necessary.

*Cross-references*: LN.1



*Index*: Admin.3

*Name*:  Create New Admin Account

*Purpose*: To create an account for a new Administrator.

*Input parameters*: *Email, First Name, Last Name, Phone*

*Action*: The Admin trying to create a new account for another Administrator will enter the *Input Parameters* for the new Admin.

The system will create a randomized password for the account that can be changed upon login.

The system sends an email to the user with their password.

The account will have admin privileges

*Output parameters*: New Admin Account*.*

*Exceptions*: Invalid or empty inputs

*Remarks*: None.

*Cross-references*: User.7

**4. Non-Functional Requirements**

**Security**

The system must include at least the standard security measures such as encrypted data transmission between the server and clients, and protection against SQL injection.

**Extensibility**

The system will be revised quite frequently and hence it must be flexible so that more functionalities can be added with minimal changes.

**Availability**

The system must be available most of the time so that there is no interruption in service. Since the system will be used by travelers, it should be accessible from anywhere.