Offline Blog Management Application

Software Requirements Specification

Ashhar Hasan (13BCS-0015)
B. Tech Computer Engineering (8th Sem)
15th March, 2017

Department of Computer Engineering
Faculty of Engineering and Technology
Jamia Millia Islamia

Contents

1. Introduction

This section gives a scope description and overview of everything included in this SRS document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided. It follows the IEEE standard for Software Requirements Specification documents.

This section ends with an overview about how the rest of the document is organised to help people reading this document have a better understanding of the document.

1.1 Purpose

The purpose of this document is to give a detailed descript ion of the requirements for the “Offline Blog Management Application” software. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications. This document is primarily intended to be proposed to a customer for its approval and a reference for developing the first version of the system for the development team.

1.2 Scope

The “Offline Blog Management Application” is a Windows desktop application which helps blog authors and owners to manage the blog content, themes, drafts, posts, media and users while offline. The application should be free to download from either the Windows Store or a direct download link from the software vendor. The software should support at least WordPress, Google Blogger and Jekyll (GitHub Pages) blogs out of the box. There should be the ability to add external plugins to provide support for other blog hosting websites and technologies. The plugin ecosystem should be open so as to allow anyone with supported tools to build an extension for the product. The software should not require administrator permissions to install or remove to allow it to be used on a per-user installation instead of a system-wide installation.

Furthermore, the software should not need either Internet or GPS connection for any operation expect first time setup, media upload and post publishing. The software should provide an environment and tools to assist in automating and bringing the publishing process offline so that all the work can be done offline and pushed online when needed

without the need for constant internet connectivity. The software shouldn't also be limited to managing a single blog at a time — a user should be able to add multiple blog accounts and switch between them seamlessly.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
Software	The Offline Blog Management Application.
Blog	An online WordPress, Google Blogger or Jekyll blog hosted on a publically accessible internet address.
Author	The person submitting an article to be either published or saved as a draft.
Reviewer	A person that examines an article and has the ability to recommend approval for publication or to request that changes be made to the article.
Stakeholder	Any person with an interest in the project who is not a developer.
User	Author or Reviewer.

1.4 References

[Committee and Board, 1998]: IEEE Software Engineering Standards Committee, "IEEE Std 830 - 1998, IEEE Recommended Practice for Software Requirements Specifications", October 20, 1998.

1.5 Overview

This document is intended for casual users, developers, testers and documentation writers, each one having their own needs and uses of the software. For a more thorough analysis on this matter, please refer to section 2.3. Section 2 provides an overall description of the software, and section 3 describes the functional requirements of the project, particularly useful to all of the aforementioned groups. Section 4 discusses the external interface requirements, and finally, in section 5, you can find the non-functional requirements of the project. Each section is divided in subsections where different matters are discussed.

2. General Description

This section will give an overview of the whole system. The system will be explained in its context to show how the system interacts with other systems and introduce the basic functionality of it. It will also describe what type of stakeholders that will use the system and what functionality is available for each type. At last, the constraints and assumptions for the system will be presented.

2.1 Product Perspective

This system will consist of two parts: one Windows desktop application and one extensibility API for writing plugins that interact with the desktop application. The desktop application will be used to manage blogs while the extensibility API will be used for writing plugins that modify behaviour of the desktop application or add additional features. The desktop application will need to communicate to a web server which hosts the blog over the internet during it's first time setup and additionally when publishing content or retreiving media files. The product will be written using C# and the .NET framework and hence expects that the machine where the software will be run is a Windows machine with .NET Framework 4.6 available or a Linux machine with Mono or dotNET Core installed.

There are similar products which allow blog management but most are limited to a single blog software like PHP WordPress Dashboard, Ghost Desktop App, Hootsuite etc. and all of them invariably need internet connectivity for proper functioning.

2.2 Product Features

A summary of product features follows:

- Connect to online blog accounts which use WordPress, Google Blogger or Jekyll.
- Create new blog posts including post text, images, categories and additional metadata.
- Create new draft posts server side which are published by the server. Only WordPress supports this.

- Create new local draft posts which can be scheduled and synced automatically provided Internet is available and the machine is turned on at the scheduled time.
- Locally preview posts and drafts using the same theme as on the remote blog.
- Manage blog comments including writing, deleting or marking a comment for review. Only Google Blogger and WordPress support this.
- Periodic sync to ensure that the data on the online blog and the local data are not inconsistent with automatic resolution of inconsistensies using heurestics.
- Addition of a secondary blog in addition to the primary blog required to use the software.
- Export of all blog posts and drafts from an existing blog to an XML based format or plaintext format depending on further needs.

2.3 User Characteristics

There are two types of users that interact with the system: users of the desktop application, and users of the extensibility APIs. Each of these types of users have different uses of the system so each of them have their own requirements. The desktop application users can only use the application to manage the blogs using functions discussed in section 2.3 above. The extension authors will not use the desktop application directly but can instead interact with it programmatically using the provided extensibility API. The extensibility API should ideally be wrapped using a library that provides binding in popular languages like C#, C++, Ruby and Python but that will be subject to feature prioritisation during the development of the product.

2.4 General Constraints

The desktop application is constrained by the system resources only. The biggest constraint is the availability of internet for the first time setup of the software and during media retreival. Another third party constraint is the availability of the web server where the user's blog is hosted. The software is also somewhat limited by the locale that the user uses their system in because there are no localizations planned for the initial version of the software meaning that the entire interface will use English US as the interface language.

2.5 Assumptions and Dependencies

One assumption about the product is that it will always be used on desktop machines that have a means of running the .NET Framework either natively or using a host system like Mono. If the desktop does not have the .NET Framework available then the software will simply refuse to install. Another assumption is the absence of any firewall restrictions for the application or the presence of a sane proxy to use the internet for all operations that require internet connectivity.

2.6 Apportioning of Requirements

In the case that the project is delayed, there are some requirements that could be transferred to the next version of the application. Those requirements are to be developed in the second release. See the revised versions of the SRS for reference.

3. Specific Requirements

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

3.1 External Interface Requirements

This section provides a detailed description of all inputs into and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface.

3.1.1 User Interface

A first-time user of the desktop application should see the first-time setup wizard which is a 7 step wizard to automate the setting up of a blog for use with the software. The first-time setup wizard will now be referred to as OOBE (for Out of Box Experience). There should be an option to allow a user to skip the OOBE if they so want but doing so should provide the user with a detailed message about how to start the OOBE again if needed.

3.1.1.1 Out of Box Experience

The user should be greeted with an OOBE when starting the software for the first time and also when explicitly invoked using a menu. The OOBE is a 7 step wizard to help connect an online blog to the software. It consists of the following steps:

1. Selection of an online blog source. The source can be a WordPress blog (either self-hosted or on wordpress.com), a Google Blogger blog or a Jekyll blog (self-hosted or on GitHub pages).
2. Providing the user credentials to the blog to allow connecting to the blog and retrieving information.
3. Customizing operating modes for the blog and giving the source a name to allow easier management of multiple sources.

4. Optional step for downloading the theme currently in use at the online blog to allow a one to one local preview even without internet connectivity.
5. Optional step for retrieval of comments left on the blog.
6. Creation of a test post on the blog to ensure that the connection works properly. The test post should be deleted as soon as the OOBE finishes.
7. Writing the OOBE settings to the Windows registry to ensure that the OOBE runs only once unless explicitly invoked.

3.1.1.2 Normal Operation Interface

During normal operation the interface should mimic that of Microsoft Office Word. It means that there should be a ribbon-interface toolbar on the top of the window to provide access to all the menus available in the software. Each section of the ribbon should be accessible using an intuitive keyboard shortcut. Furthermore, each individual option within a ribbon panel should also be accessible using nothing but the keyboard.

Below the ribbon interface should be the main text editor. The text editor should have three modes of operation which should be indicated by three buttons at the bottom of the text editor. The three modes are:

Markdown mode: The text in this mode is the Markdown equivalent of the text entered in the other modes.

HTML mode: The text in this mode is the HTML output that will be generated and used to render the post.

Rich Text mode: This is the default mode and is similar to how Microsoft Office Word works. The user can apply formatting using toolbar buttons and keyboard shortcuts and the changes will be reflected in other modes.

Preview mode: This is a read-only mode where the user can see the rendered output — how the post will look in a browser when published on the blog. The user needs to have set up the offline theme capability during the blog setup process to use this feature. This mode will notify the user to do so if they haven't already done it.

3.1.1.3 Management Interface

A separate menu based interface is used to manage the connected blogs. The interface should be a two-pane interface with sections on the left side and corresponding information on the right hand side. The main sections should include the following:

Blog Accounts: This section should list all the currently connected blogs along with the URLs where they are hosted and should prominently display identifiable branding so that the user can easily differentiate between different blog services. It should provide the following operations:

1. Viewing details of existing connections.
2. Removing existing connections after confirming the action. Additionally the software should prompt the user to sync information if the local changes have not been synced with the online blog to ensure that no data is lost.
3. Adding new connections. The blog provider branding should be prominently visible to make sure that the user knows which type of blog they are adding.
4. Changing blog connection credentials. It is possible that the blog's user authentication credentials have been changed after setting up the blog (due to the user changing their password). To help fix this issue the user can simply change the stored password and username from this menu to avoid the need to reconnect to the blog from scratch.
5. Explicitly starting the OOBE. This menu should provide a button to explicitly start the OOBE if the user skipped it during the first-time startup of the application.

Plugins: This section should provide a list of currently installed plugins with a button to open the plugin's settings. There should also be a button to list all available plugins. The list will be retrieved from the internet and cached until the next time internet connectivity is detected.

Text Editor: This section will list all the settings pertaining to the text editor including the following:

1. Default mode for the text editor.
2. Default font used for the Markdown mode.
3. Markdown flavour used. The software should handle CommonMark, GitHub Flavoured Markdown and both Setext and ATX style headers.
4. Enabling and disabling of syntax highlighting if code is detected.
5. Word wrap and other similar features.

Internet Settings: This section allows the configuration of the following:

1. Whether to use a proxy server to connect to the internet.
2. Proxy server to use for HTTP connections.

3. Proxy server to use for HTTPS connections.
4. Maximum bandwidth the product is allowed to use.

Update Settings: This section allows the user to check for any available updates or to switch from the regular update channel to nightly releases channel.

3.1.2 Hardware Interface

Since neither the desktop application nor the extensibility API have any designated hardware, it does not have any direct hardware interfaces.

3.1.3 Software Interface

The desktop application communicates with the web server where the blog is hosted in order to get the data from the blog including previous posts, drafts, comments and associated metadata.

3.1.4 Communication Interface

The communication between the different parts of the system is important since they depend on each other. However, in what way the communication is achieved is not important for the system and is therefore handled by the underlying operating systems for both the desktop application and the extensibility API.

3.2 Functional Requirements

This section includes the requirements that specify all the fundamental actions of the software system.

3.2.1 Download

The user should be able to download and install the desktop application from an application store or a direct download link.

3.2.2 Update delivery and notification

The software should notify the user of any available software updates and download and apply them according to the settings user has configured in the application.

3.2.3 Connecting a blog with the software

The software should allow a user to connect a publicly accessible blog to the software.

Inputs:

1. The web address of the blog.
2. The type of blog or the service that it is hosted on. Can be one of Google Blogger, WordPress, Jekyll or GitHub Pages.
3. User credentials for authenticating to the website. Google Blogger additionally requires that API access is enabled in Google Blogger settings.

Process:

1. Send an HTTP GET request to the web address to ensure that the address exists.
2. Check that the type of blog at the web address matches the type specified by the user.
3. Download the files from the blog including theme, content and metadata.
4. Create a test post on the blog using the application and check if it was created correctly to verify that the connection is working.

Error Handling:

1. In the case of absence of a website at the URL we present a message to the user to fix the URL.
2. In the case of mismatch between user specified blog type and detected blog type we present a message a message to the user with the option of automatically fixing their mistake.

3.2.4 Creating a new post

The software should allow the user to create a new blog post.

Inputs:

1. The title of the blog post.
2. The category of the blog post.

3. The published state of the post. It can be published or draft.
4. The blog connection on which the post is to be made.
5. The associated metadata specific to every blog provider.

Process:

1. Create a file on the local filesystem in the application's data directory representing the post heirarchy on the associated blog account.
2. Create a metadata file of the same name as the post with a different extension to keep track of associated post metadata.
3. Present the post file in the application's editor. The default mode should be rich text editing mode.

Error Handling:

1. If a post by the same name already exists on the same blog, provide an alert to the user and allow them to change the name without having to create another post from scratch.
2. If a post by the same name already exists on the same blog but is not synced locally, alert the user the next time the application tries to sync data from the online blog.

3.2.5 Rename an existing post

The software should allow the user to rename an existing post.

Inputs:

1. The title of the blog post to rename.
2. The category of the blog post.
3. The published state of the post. It can be published or draft.
4. The blog connection to which the post belongs.
5. The new name to give to the blog post.

Process:

1. Check if a file corresponding to the new name already exists in the application's data directory. If it does, alert the user and allow them to change the name without starting over.
2. Create a new file on the local filesystem and copy the contents of the current file over to the new file. This should include the metadata and url files as well.
3. Delete the files with the old name only after ensuring that the newly created files can be read correctly and there is no difference in the content.

Error Handling:

1. A file with the new name already exists then alert the user and allow them to change the name without having to start the rename process again.
2. If a post by the same name already exists on the same blog but is not synced locally, alert the user the next time the application tries to sync data from the online blog.
3. The rename failed due to some other reason then transparently roll-back the changes and inform the user about it. Allow them to either retry the action, abort the action or look at the logs to allow submitting them to the developers for further investigation.

3.2.6 Modify an existing post

The software should allow the user to modify an existing post including it's title, content, metadata, media, published status and publish date.

Inputs:

1. The title of the blog post to modify.
2. The category of the blog post.
3. The published state of the post. It can be published or draft.
4. The blog connection to which the post belongs.

Process:

1. Open a new window with two panes, the left one for editing the visual content (title, body, media etc.) and the right pane to edit the metadata like post category, published status, date, author etc.
2. Allow the user to edit any and all of those fields freely.
3. Create an autosave of the new information as backup every 1 minute.
4. Prompt the user to save the information when closing the window if any changes were made.
5. Saving the changed information should edit the backing files in-place instead of creating new copies to avoid overhead.

Error Handling:

1. A file with the new name already exists then alert the user and allow them to change the name without having to start the modify process again.
2. If a post by the same name already exists on the same blog but is not synced locally, alert the user the next time the application tries to sync data from the online blog.

3. The modification failed due to some other reason then transparently roll-back the changes and inform the user about it. Allow them to either retry the action, abort the action or look at the logs to allow submitting them to the developers for further investigation.

3.2.7 Add media to a post

The software should allow the user to attach media files of any type as long as they are supported by the blog provider.

Inputs:

1. The title of the blog post to add the media to.
2. Optional: Location within the document where to attach the media file.
3. The list of filetypes accepted by the blog provider.
4. The URL or local fileystem URI to the media to attach.

Process:

1. Open a two paned window with a URL bar and browse button on the top to allow for manually providing a URI to fetch the media files from.
2. The left pane shows previews of the media file currently selected.
3. The right pane allows modifying the properties of the media file before inserting it into the blog post.
4. If internet connectivity is available then the software should fetch the media files from remote URLs otherwise it should simply write the URL to the URL file associated with the blog post so that the media can be retrieved at a later time.

Error Handling:

1. If the filetype is not supported by the blog provider alert the user about the same and show them a list of accepted filetypes.
2. If the remote media could not be fetched inform the user and ask them if to retry or write the information to the associated URL file for later updatation.

3.2.8 Delete a post

The software should allow the user to delete any existing posts as long as the user has sufficient priviledges.

Inputs:

1. The title of the blog post to delete.
2. The user priviledge level on the blog.

Process:

1. Show a confirmation message asking the user to confirm the action. If the user accepts it then delete the post but keep a local copy of it for at least 30 days.
2. Move the data files, metadata and url files to a specific directory used to store deleted posts.

Error Handling:

1. The user doesn't have sufficient privileges for the action then alert the user asking them to contact an administrator or log in using the blog administrator's account.

3.2.9 Schedule a blog post

The software should allow a user to schedule a blog post to be published at a future time and date.

Inputs:

1. The blog entry to schedule.
2. The date and time on which to schedule the post.

Process:

1. Show a dialog box telling the date and time both in local timezone and UTC timezone to allow the user to confirm if the times are correct. If the user accepts schedule the post otherwise abort the action.
2. If the blog provider supports scheduled posts (WordPress and Blogger) use their API to schedule a post.
3. If the blog provider doesn't support scheduled posts then use the application's own scheduler to create a scheduled job entry and notify the user that the scheduled post will only be posted if the user has the system turned on with internet connectivity.

Error Handling:

1. If the scheduled time is missed due to the system being turned off or no internet connectivity, wait for internet connectivity and update the post after informing the user of the situation.

4. Non-Functional Requirements

4.1 Performance Requirements

The software should be able to run on any system that allows the installation of the software. That means that any system with internet connectivity (even if occasional), Windows with .NET Framework 4.6 (Windows 8 and later have it pre-installed) or Linux with Mono.

As for storage requirements the application should not take more than 50MB in a clean install. The software should be able to handle arbitrary amount of blog data provided the storage space on the user's system supports it. The software should also be mindful of the user's storage space and should leave at-least 10% of the system storage space empty and alert the user about the situation and proceed only if expressly granted permission by the user.

The software is also not computationally expensive and should hence run on almost any processor as long as the application can be compiled for the architecture. This generally means that all AMD64, x86-64 and Intel 32-bit processors are supported.

The software should also be modest about RAM usage and should be able to perform all functions under 200MB of RAM except for media handling or previewing of a theme.

4.2 Safety Requirements

As with any software, data loss is a possibility and hence the software should perform all data bound actions using a locking mechanism to prevent race conditions and corruption due to simultaneous access of files. The software should also perform all filesystem based transactions in an atomic manner to ensure that all actions can be rolled back as long as the backup files are available. The atomic transaction support depends on the filesystem that the software is running on and hence is not a strict guarantee.

With that said, the software comes with absolutelty no warranty and cannot be held responsible for any loss of data due to the use of the software.

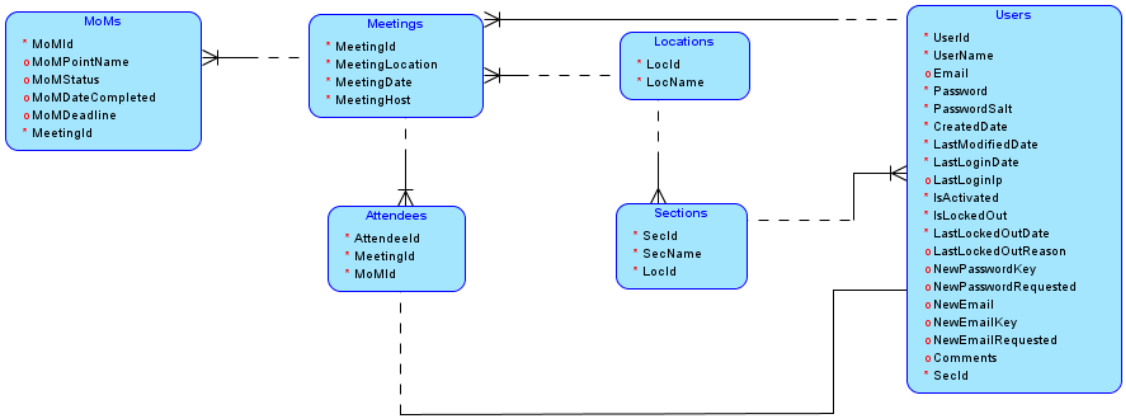
4.3 Security Requirements

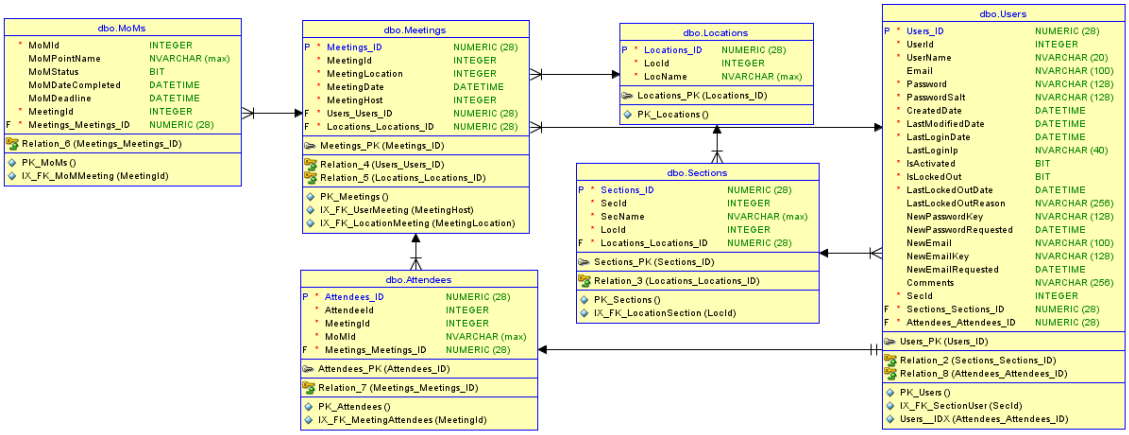
The software has mechanisms to ensure that no user is able to perform an action not allowed at their privilege level on the blog. This means that only those blog accounts that are administrators can actually delete posts, or create posts or even modify them. The capabilities of non-admin and admin users are not decided by the software but are instead decided by the blog provider they are using and the configuration settings on the blog providers.

The software is bound by the security features of the host operating system too and hence cannot modify files of another installation of the same software under a different user account.

5. Appendix

5.1 Data Models





Bibliography

[Committee and Board, 1998] Committee, I. C. S. S. E. S. and Board, I.-S. S. (1998). Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers.