CCP service specification

Webservice provider on Linux

1 Introduction

The CCP Service covers functionality for remote document management. The specification for covers basic use-cases and has the purpose of a proof-of-concept prototype. In the following we will describe some system basics as well as web services to provide by the offered prototype.

2 Software Architecture

The CCP Service for Linux will have the following software architecture. The software will be running based on a node.js webserver installed on a Linux Debian Wheezy system. User interface functionality should be implemented with react.js. For the purpose of testing it is possible to set up a Linux stack inside the Google compute environment.

3 Software Requirements and Stories

3.1 Requirements for provided webservices

This requirement list contains descriptions for requirements on a file basis. All the functionalities should be implemented for a single file, multiple files as well as folders. All webservices should return message in a simple JSON format. The detailed description of the proposd (simple) JSON requests should be part of the documentation.

3.1.1 Providing a file upload webservice

The running web server should allow and process file upload request. A file upload request can contain parameters. For the purpose of this prototype it can be assumed, that a filename, a username and and the file's timestamp is submitted as parameters as well as the files content. The webservice should emit a message to the client when the upload (3.1.2 , 3.1.3, 3.1.4) is finished.

3.1.2 Handling a file upload

When a file upload happens, the web server looks up and compares the user name against known usernames. For the purpose of simplicity for this prototype one can assume 3 usernames (Test1, Test2, Test3). When uploading a file to the server it will first be checked which user uploaded the file and the file will be forwarded to another server by an HTTP request. The usernames have an associated webserver url for their repositories where the files should be stored.

3.1.3 Handling and providing a folder upload

Similar to 3.1.3. it is possible to upload folder structures. This case should be handled in two different version. First, there should be a web services request preparing the webserver for several file upload requests. This causes the CCP Service with redirecting the folder and its content (3.1.4) until all files were received. One can assume that this request contains an additional "filelist" parameter containing the content of this folder. This content can also have subfolder structures.

The second way for handling folders is to provide a webservice where the folders content is part of the request. It be assumed that this has certain limitations in file size and that there can be an upper maximum to upload.

3.1.4 Redirecting a file

For the purpose of this prototype one can assume two kind of file storages. First, a Google Cloud Storage Bucket (https://cloud.google.com/storage/docs/concepts-techniques) and an Amazon S3 Storage service (http://aws.amazon.com/de/s3/).
Based on the username's property the storage system should be chosen and the incoming file should be redirected to this storage service. Folders should be kept in the same structure as the incoming folder structure.

For testing and implementation of this prototype software the test accounts for those storage services should be enough. Notice that the user accounts for this storage services are not dependent on the received usernames described in 3.1.2.

3.1.5 Providing a webservice for file download

The CCP Service provides an interface for downloading one or more files by redirecting these files from a storage. The webservice request should provide filename, username, file creation date and data of last changes in the file. The username is the user which uploaded this file beforehand.

3.1.6 Handling a file download

When receiving a file download request the CCP Service should be able to broadcast a search request to all storage services to find the file. If a storage service has the file with the requested filename, then the file should be downloaded from the storage service if the "last changes date" attribute is newer than the date sent provided by the requester. If the file

on the server is newer than the date in the request, the file will be downloaded from the storage and forwarded to the client which requested the download. If the date of the file is the same or older than the request parameter the webservice should return a message service. When no file with this filename is found the webservice should return an error message.

3.1.7 Webservice to delete a file

The CCP Service offers the ability to delete a file from one of the storage services. Similar to the download request there should be a query to each storage service to find the associated storage and delete it from the storage. This webservice contains a filename as well as a username. When no files with this filename is found, the webservice should return an error message.

3.1.8 Webservice to request an update on a file

The CCP Service offers the ability to delete a file from one of the

3.2 Requirements for webserver interaction

3.2.1 User Interface with react

For handling the username / storage service interaction there should be a simple user interface built with react.js to associate a username with a storage service. This can be a collection of combobox objects and buttons without, but it should provide the necessary functionality as described in 3.2.2 and 3.2.3.

3.2.2 Changing a username/storage service connection

The CCP Service interface makes it possible to associate a username and a storage service. This means, for a specific username the storage service can be changed while runtime of the CCP Service. When changing this association the redirection for file uploads (3.1.2.) is changed and therefore the files are stored at another storage service.

3.2.3 Shut down CCP Service

The interface should provide the functionality to shutdown the CCP Service.

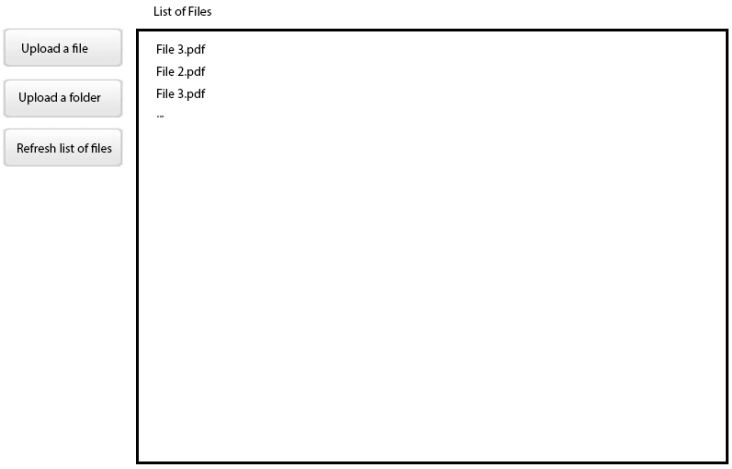
3.3 Testing

The tested data files should can be of the format .pdf or .psd. It can be assumed, that these files can have a size up to 1GB. Testing the CCP Service can be done by simple Javascript scripts running in an external webbrowser and showing the results of the tasks.

4 UI Sketches

4.1.1 UI CCP Server side

4.1.2 UI CCP Webclient for tests



5 Artifact delivery and Installation

The artifact delivery should contain all the source codes as well as an installer set up . The project source code should be organized according to the node.js organization. However it would be preferable to have a smaller installer package

The deliveries should be the source code, as well as a small Doxygen documentation and the installer package as .zip file. The source code should be documented as far as possible accordingly to the Doxygen patterns. If necessary, the documentation can be done after the final delivery is approved. Furthermore, it is recommended to keep the Google Style Guide in mind when writing the source code.