<MentionNotifier>

System Design

<1.0>

<07/01/2018>

<Berk Ergun – 210CS2013>

Prepared for

CSE490 Thesis Project



Table of Contents

[1. Introduction 1](#_Toc503118333)

[1.1. Purpose of the System 1](#_Toc503118334)

[1.2. Design Goals 1](#_Toc503118335)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc503118336)

[1.4. References 2](#_Toc503118337)

[2. Current Software Architecture 2](#_Toc503118338)

[3. Proposed Software Architecture 2](#_Toc503118339)

[3.1. Overview 2](#_Toc503118340)

[3.2. System Decomposition 2](#_Toc503118341)

[3.3. Hardware Software Mapping 4](#_Toc503118342)

[3.4. Persistent Data Management 4](#_Toc503118343)

[3.5. Access Control and Security 5](#_Toc503118344)

[3.6. Global Software Control 5](#_Toc503118345)

[3.7. Boundary Conditions 6](#_Toc503118346)

[4. Subsystem Services 7](#_Toc503118347)

[5. References 7](#_Toc503118348)

SYSTEM DESIGN DOCUMENT[1]

# Introduction

With our software, we strive to keep it as minimal and efficient as possible. We will be using a 3-layer architecture.

## Purpose of the System

The purpose of the system is to allow for its user to keep up with things of his/her interest over the internet through a desktop application.

## Design Goals

The utmost, greatest design goal of this project is to complete this system with its main functionalities by delivery time, which is the end of this semester. There can be no extensions and thus, any decision that’ll effect the delivery schedule negatively will be forfeited.

Otherwise, the system design goals are as follows:

***Efficiency***: As a background task first and foremost, MentionNotifier should have as little memory and CPU time footprint as possible.

***Modifiability***: MentionNotifier should be modifiable by its User if they deem so.

***Portability***: While MentionNotifier is built with Windows in mind, it still should work under different operating systems.

***Availability***: By using the Google Web Search Engine, MentionNotifier should have a near 100% availability for use.

## Definitions, Acronyms, and Abbreviations

SDD: Software Design Document

Service: A set of related operations.

3-Layer Architecture: An application consisting of 3 hierarchically ordered subsystems.

Interface: A shared boundary between two or more components of a system that exchange information. [2]

Pickle: Python module implementing binary protocols for serializing and de-serializing a Python object structure. [3]

## References

See: Section 5.

# Current Software Architecture

There is no existing system that our system will be replacing. Similar software that may be comparable to it can be Google Alerts and social media. Google Alerts is a content change detection and notification service, offered by the search engine company Google. [4] When it finds new information, it sends an email to the user of the service to inform him/her. Instead of this web service approach, our system is a desktop application that directly deals with the search results provided by Google. It also is barebones in design, requiring no mail information from its User.

Social media like Facebook, twitter or Instagram allow for their users to follow ‘tags’ or ‘groups’ which they will get new information about when a post is made including that ‘tag’ or by that ‘group’. While this may be enough for smaller circles, our system strives to retrieve a much more general information about Users’ desired keywords.

# Proposed Software Architecture

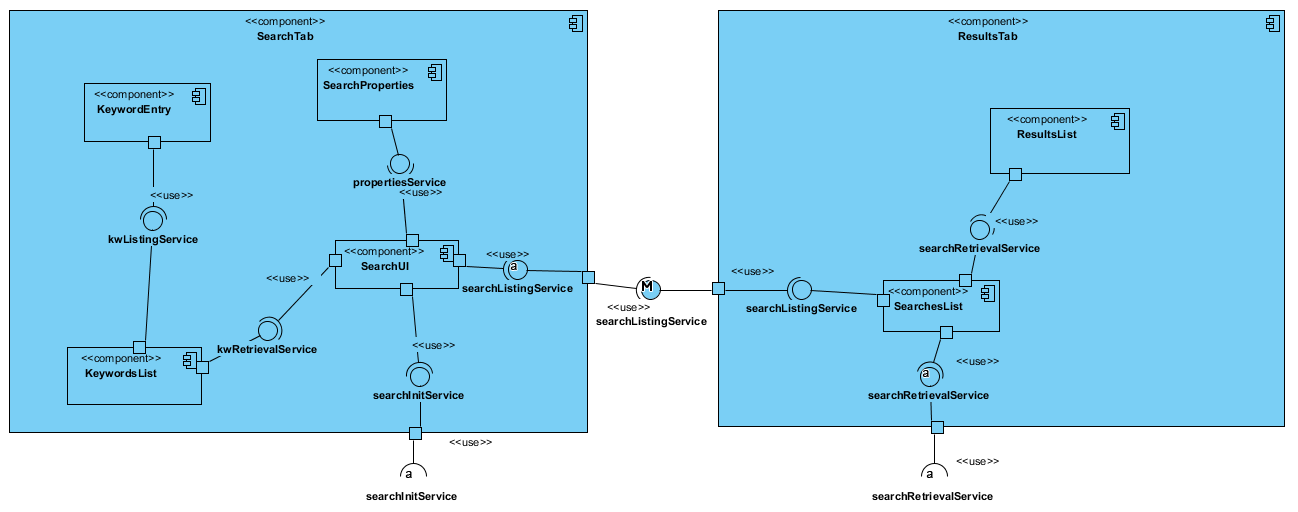
## Overview

The system will have a 3-layer design approach in its architecture. These 3 layers will be the Interface which will allow its users to interact with the system, the Application, which is the functionality of the system itself and the Storage layer where all the persistent data is kept.

## System Decomposition

The Presentation Layer holds UI elements of our system which the User can interact through with our application. It is divided in to two main subsystems, the ResultsTab and SearchTab subsystems. These represent the two tabs in our application UI which hold the other UI elements of our application.

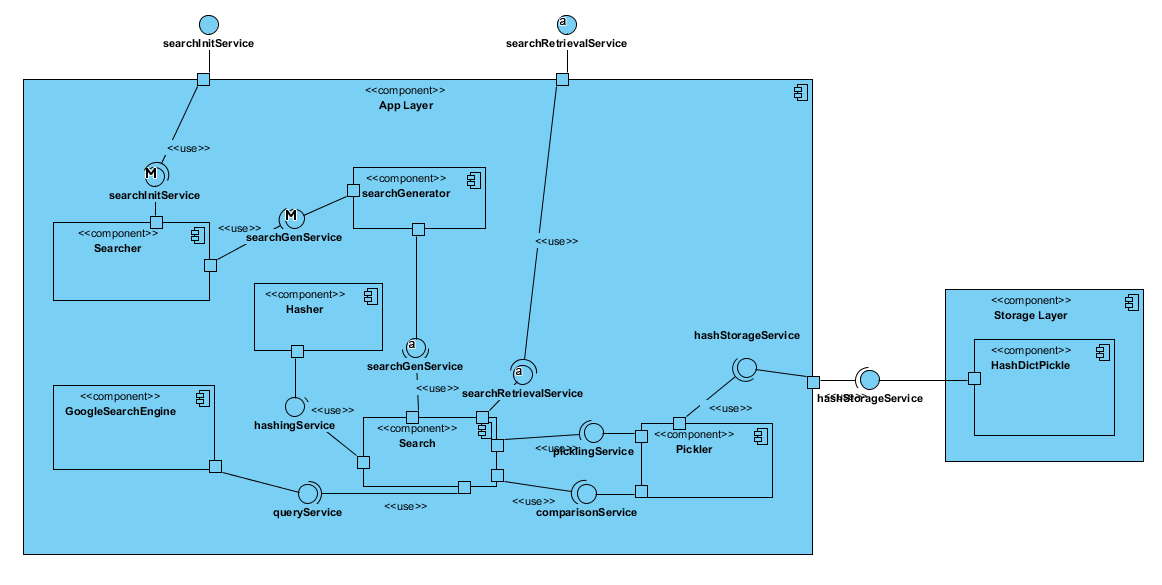
***Presentation Layer***



The SearchTab will be UI that will welcome the User and will hold KeywordList subsystem which’ll hold the keywords entered by the User and will provide the selected keywords for other subsystems if requested. KeywordEntry subsystem is responsible for keyword entry by the User. SearchProperties subsystem holds the properties of a search, the search depth and search intervals. The Search subsystem retrieves all the information entered into other subsystems by the User and calls the application searchService. It also adds the search initiated into the ResultsTab’s SearchesList subsystem.

ResultsTab will be the UI that the User will view searches and their results through. It holds the SearchesList which will represent the searches ongoing started by the User and the ResultsList subsystem which holds the list of results selected by the User through the SearchesList subsystem. SearchesList will also be responsible for the initiation of search stopping function in the application layer. Also, ResultsList will be responsible for listing updated results at top.

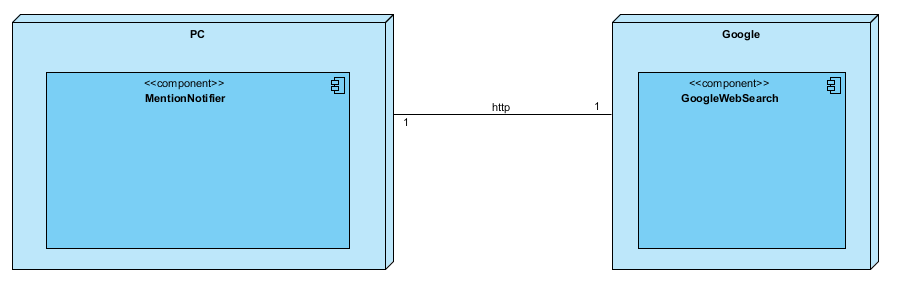
***Application and Storage Layers***



The Application Layer holds the main functionality of the system. It retrieves information entered in to the Presentation Layer and performs its operations. The Searcher subsystem is the initiator for the main functionality of our system, making searches. It retrieves the data from the UI and feeds the data into the SearchGenerator subsystem. The SearchGenerator subsystem creates the search with the fed data. The Search subsystem represent each instance of search started by the User. It first connects to the GoogleSearchEngine subsystem and retrieves the search results. Then using the Hasher subsystem, it creates a hash for each link for a result so that the system can later compare with a later search iteration to decide whether the link site was updated or not. Then, the Pickler subsystem ’pickles’ each link and it’s corresponding link into a ‘\*.pickle’ file, which is represented by the HashDictPickle subsystem, the sole subsystem of the Storage Layer. If there already exists a pickle corresponding to the keywords searched, it compares the new search results with the one kept in the pickle file and marks each result to represent whether they were updated or not. The search operation is then done; it fires a notification request to the OS with the keyword and the updated results count. From then on, it provides the searchRetrievalService for the Presentation Layer for access in to its results and waits for its next iteration of searching.

## Hardware Software Mapping

The software’s 3-layered architecture allows for great flexibility in deploying our system, but for our purposes, we will have a 2-node design where we hold our whole application in a single PC and connect to the GoogleWebSearch through http requests.



## Persistent Data Management

Our system does not use a database management system and instead makes do with ‘pickle’ files created by the Pickler subsystem. As such, it is suspect to fail if another application or the OS restricts access or read/write permissions to the system.

The files are serialized pickle files, creating using Python’s ‘pickle’ module, allowing us to persistently keep Python objects in storage. In our case, we are keeping dictionaries for each corresponding search where we hold search result links as ‘key’s and their hashes as ‘values’ within.

## Access Control and Security

As a system with no management or administration protocols, our single User actor can access all functionalities of our system provided by the UI.

The system provides no security measures as it holds no real personal information about the User other their past searches and their links as pickle files. These files can be accessed through Python – or about any text editor - and we hold no responsibility over their security or accessibility.

## Global Software Control

* KeywordEntry subsystem initiates the KeywordsList subsystem to save the keyword provided by the User.
* Search subsystem initiates the SearchProperties subsystem to retrieve the search properties set by the User.
* SearchUI subsystem initiates the KeywordsList subsystem to retrieve the selected keywords entered by the User.
* SearchUI subsystem initiates the Searches List subsystem to provide name for search initiated.
* SearchUI subsystem initiates the Searcher subsystem to initialize a search with the data it retrieved from other subsystems.
* Searcher subsystem initiates the SearchGenerator subsystem to create a search instance.
* Search subsystem initiates the GoogleSearchEngine subsystem to retrieve the search results from.
* Search subsystem initiates the Hasher subsystem to access each search result link and retrieve a hash from.
* Search subsystem initiates the Pickler subsystem to store the hashes for each link in a persistent file. If there already exists a pickle file corresponding to the search, it compares its hashes with the ones kept in the pickle file.
* Pickler subsystem initiates the HashDictPickle subsystem to write the files with.
* SearchList subsystem initiates the Search subsystem to retrieve its selected search and provide it to the ResultsList subsystem.
* SearchList subsystem initiates the ResultsList subsystem to provide its search data for viewing.

## Boundary Conditions

Startup:

* The application will start with the Search Tab showing.

Keyword entry:

* Keyword field is entered empty.
* Keyword field has invalid characters.

Pickler:

* Application has no read/write permissions.
* Search has new link results not existing in the stored dictionary.
* Search doesn’t have the link results existing in the stored dictionary.

GoogleSearchEngine:

* Application has no internet connection.
* The search engine is down.

Hasher:

* Hasher cannot reach a link.
* Hasher retrieves an unhashable page source code.
* Hasher times out while reaching a link.
* Connection to a link is restricted.

SearchGenerator:

* Search cannot start a search thread due to OS restrictions.

Search:

* Search can’t create a notification.

# Subsystem Services

*kwListingService:* Retrieves and lists the entered keyword by the user.

*kwRetrievalService:* Provides the keywords selected by the user, if any.

*propertiesService:* Provides the search properties information.

*searchListingService:* Retrieves and lists searches initiated by the user.

*searchInitService:* Initializes the search function and provides required parameters.

*searchRetrievalService:* Retrieves actual search instance data for viewing purposes.

*searchGenService:* Retrieves the search parameters and creates a search instance then starts it.

*queryService:* Provides the search functions of Google Web Search Engine.

*hashingService:* Provides the hashing services to hash webpage source codes.

*comparisonService:* Provides the comparison service for comparing an existing previous search iteration’s ‘link:hash’ dictionary with the current one and marking each result as whether they are updated or not.

*picklingService:* Provides the ‘pickling’ service, using the default Python module ‘pickle’ to persistently store ‘link:hash’ dictionaries for searches.

*hashStorageService:* Stores and manages the dictionaries kept as pickle files.

# References

The following is an example of listing a book in this section. Check the text to see how it is cross referenced (The whole document is based on [1]).

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. *IEEE 100: the authoritative dictionary of IEEE standards terms*. Institute of Electrical and Electronics Engineers, 2000
3. “12.1. pickle - Python object serialization¶.” *12.1. pickle - Python object serialization — Python 3.6.4 documentation*, docs.python.org/3/library/pickle.html.