JAROSLAW HIRNIAK

SEEKER

DOCUMENTATION

Contents

1 Project

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.1 Motivation

The World Wide Web is a very rich source of information and many excellent search engines exist that help to find interesting information in the matters of milliseconds. However, they only provide links to the source of information and if user wants to compare data from many sources or gather it for analysis purpose they are left with a daunting task of collecting all this information by hand.

This is repetitive and boring task, something that humans do not enjoy, but computers excel in. It would be of significant help to any researchers that need to compare vast sections of information from multitude of sources to have a tool that would take care of the first part of the analysis, which is collection of relevant data, and let the researcher to concentrate on what is important and hard - the reasoning about the data.

The goal of this project is to research feasibility of implementing a system for querying big data sets (e.g. collection of 100 pages report from standardization committee) and presenting only relevant sections of it (e.g. only few paragraphs of interest filtered from across a thousand of related documents stored in the database). The last step of the project is to implement a proof of concept system that possibly could be developed into fully functional system.

1.2 Feasibility and technologies

The first part of the project is going to be to investigate the nature of the problem and if it can be efficiently solved. If it can be solved then suitable technologies will be suggested and used in the proof of concept system.

1.3 Proof of concept

The project idea originated from a problem common to the researches in minority languages at the University of Edinburgh, who faced issue of gathering data from variety of sources, for instance from European Charter for Regional or Minority Languages website for comparative analysis and producing various reports based on that data. However, the problem is that data is presented in PDF reports (and is not annotated in anyway), the layout of reports vary (depending on the cycle, country issuing it, and other factors), and average length of a report is over 100 pages. Thus, any researcher interested in using valuable data from reports must face first daunting extraction of interesting information sections from multiple sources.

1.4 Data sources

The primary intended source of data are long documents, often being reports, with usual length from 50 to 100 pages, in different formats like DOC(X), ODT, PDF, TXT, etc. and being unannounced beside in the most optimistic case having similar structure across range of documents. For example, for the proof of concept there are 651 PDF documents listed on the page, of which 309 documents are in English (some documents are produced also in French or country producing report official language), of which 300 documents are of actual interest (rest includes charter advertisements or documents that do not present valuable source of information). There are 25 countries producing reports, each adhering to general layout, but with subtle differences coming from the tool used or typesetting conventions, which should be taken into account during parsing PDF into JSON[1].

[1] Thus, the optimal parser would be context aware (and such parser was implemented for this project).

1.5 Stakeholders

Each of the stakeholders is important part of the system without which the system could not be considered successful, all with different needs, and assumptions about the system functionality.

- Researchers - people who are using service often for their research and rely on the correctness and up-to-date properties of the system;

- Consumers - people who need to access data occasionally and do not have need to register an account in the service, or even do not care if the document published 2 days ago is included in the search;
- Administrators - people having rights to add sources, respond to tickets, etc., and responsible for the service maintenance;
- Developer - person responsible for the project development and code base maintenance.

1.6 *Project Name*

The project name *Seeker* comes from "Seeker, Reaper" poem by George Campbell Hay. You can listen to the peom being recited on YouTube here.

2 Requirements

2.1 Summer 2014 plan

The goal for 8 week summer project was set to

- carry out research and recommend appropriate solution (architecture, full software stack, and persistence layer);
- Implement a proof of concept functionality to prove the possibility of developing efficient solution for the stated problem, this is, ability to query European Committee for Minority Languages and receive combined documents composed of elements specified in the query.
- Deploy the proof of concept on the university server for evaluation and presentation.

2.2 Data format

The main source of the data are going to be the European Charter for Regional or Minority Languages expert reports[1].

The goal of the project is to account for the future grow and to include also documents from Galeic Plans produced by various bodies in Scotland [2].

The data needs to be assumed to do not have any common format or fields.

[1] http://www.coe.int/t/dg4/education/minlang/Report/default_en.asp

[2] http://www.gaidhlig.org.uk/bord/en/our-work/gaelic-language-plans.php

2.3 Use cases

During the initial meeting on 26 May 2014 the below use cases were identified.

Use Case ID:	0
Use Case Name:	Database query
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Consumer
Description:	Actor wants to extract information from the database relating one or multiple documents that were present on source listed web pages.
Preconiditions:	• Actor on the query page (being also the landing page)
Postconditions:	• The result is displayed in the browser and can be retrieved in prefered format (pdf • txt) • the search refined • or modified and then retrieved.
Priority:	100
Frequency of Use:	100
Basic Flow:	1. Go to the search page 2. Insert query in the prefered format (library index, query language, etc.) 3. Refine search
Exceptions:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	1
Use Case Name:	Information Retrieval
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	System
Description:	Actor process all source listed documents and updates the database accordingly.
Preconiditions:	• Regular update in time of small server activity (e.g. 2am) • Update requested by administrator
Postconditions:	• The database contains the latest document information • If data accessed during update user notified that data may be inconsistent by a warning as well warning was saved in all generated documents
Priority:	100
Frequency of Use:	10
Basic Flow:	1. Activate at predefined time 2. Fetch all documents 3. Extract information 4. Update aggregates accordingly
Exceptions:	Update already in progress
Special Requirements:	Cannot be requested one after another
Assumptions:	
Notes and Issues:	

Use Case ID:	2
Use Case Name:	Save generated output to a file
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Consumer
Description:	Actor saves generated data to a file in specified format.
Preconiditions:	• Legal format specified • Data generated
Postconditions:	• The download process of a generated document started
Priority:	75
Frequency of Use:	50
Basic Flow:	1. Generate the desired output 2. Select file format 3. Save to disk
Exceptions:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	3
Use Case Name:	Save generated output to a file
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Consumer
Description:	Actor wants to register on the website.
Preconiditions:	• User name not registered • All required data provided
Postconditions:	• The actor account exist in the system
Priority:	25
Frequency of Use:	25
Basic Flow:	1. Go to the sign up page 2. Fill the form 3. Sign up 4. Verify 5. Log in
Exceptions:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	4
Use Case Name:	Update the source listing
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Administrator
Description:	Actor wants to update/delete existing listing or insert a new one.
Preconiditions:	• Listing already not specified • Listing not being subdomain of existing listing
Postconditions:	• Data can be fetched from the listing source
Priority:	50
Frequency of Use:	5
Basic Flow:	1. Log in 2. Go to the source listing page 3. Request an action 4. Fill the data
Exceptions:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	5
Use Case Name:	Request listing to be included in the source listing
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Consumer
Description:	Actor wants that his listing would be included in the retrieval process.
Preconiditions:	• Listing already not specified • Listing not being subdomain of existing listing
Postconditions:	• Data can be fetched from the listing source
Priority:	40
Frequency of Use:	2
Basic Flow:	1. Go to the suggest listing page 2. Insert link 3. Normalize link 4. Test if valid 5. Notify about the result 6. Notify administrator to take action
Exceptions:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	6
Use Case Name:	Report Bug
Date Created:	26 May 2014
Date Last Updated:	26 May 2014

Actors:	Consumer
Description:	Actor wants to report a bug which he or she encountered either during usage of the system or observed basing on the experience (can be constructive to improve the quality of service).
Preconiditions:	• Error occurred • User at suggestions page
Postconditions:	• Ticket saved in the system and present on the ticket board (visible to administrators)
Priority:	60
Frequency of Use:	10
Basic Flow:	1. Error occurred and feedback form was presented 2. User filled the form and clicked submit 3. Ticket shown in the system
Exceptions:	
Special Requirements:	Must project from XSS and code injections
Assumptions:	Input is normalized
Notes and Issues:	

3 Requirements analysis

3.1 Objectives

Based on the gathered requirements the following objectives were identified.

- Extendibility - the project should be implemented and handled in a way that potential development into a full system should be relatively easy, i.e.
  - It should be *documented* (this file) and the source code should be commented that the project modules and structure should be easy to understand by a developer unfamiliar with the project;
  - Popular technologies should be used that the potential group of developers would not be too much limit factor;
- Schemaless - documents (sometimes even from the same source) will have different structure, so it is important to not restrict the database to store one kind (structure) of information, thus relational database seem as unfit solution and one of NoSQL approaches should be adopted;
- Open source - all technologies and tools used should be open source and the project itself should be published as open source that anyone interested should be able to use it.

3.2 Structure

Based on the objectives and requirements it was concluded that the project should be split up into the following components:

- Front-End: client side code for forming queries and presenting results.
- Back-end: receiving queries, query processing, storing searchable data in the database and document copies.
- Extractor: compiling documents from different formats (e.g. PDF, DOC, TXT) without any annotations into JSON format with meta-data like contextual information from the website (e.g. document cycle, country producing, language) and structured document data, i.e. create index of sections, analyse context and

information in sections and paragraph in order to create index etc.

- Web crawler: collecting documents for the extractor. Can be implemented as simple downloader of all files with given extension from a list of websites or general web crawler with proper seed for the documents of interest (e.g. start at MinLang committee website and Gaelic plans and follow up to 3 links).

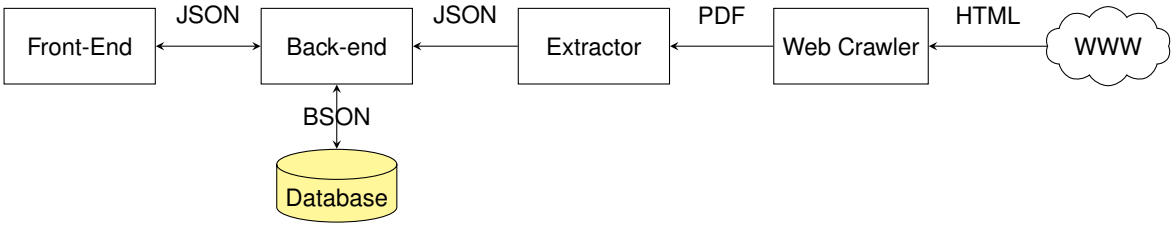


Figure 3.1: System components

3.3 Front-end

The main functions of the front-end are

1. query formulation by a not necessarily computer science inclined users and
2. presentation of the query evaluation results downloadable for off-line analysis.

Because this looks exactly like SPA[1], the design should be based on MVC[2] framework, and use standard interaction libraries for intuitive feel and great reactiveness. Generally, the feel of using the application should be similar to using a desktop program.

[1] SPA - Single Page Application is a web application that fits on one page and provide more fluid experience similar to a desktop application.

[2] Model-View-Controller design where our data is stored in the model, presented to the user via view, and modified via the controller.

3.4 Back-end

The back-end has two main roles:

1. executing the query specified by the user in the front-end and returning the results,
2. providing point of communication of extractor to insert and update the database.

The back-end must be able to communicate asynchronously with multiple users at the same time responding to their requests. As some requests are likely to take at least few seconds to execute the user should be prompted about the query evaluation result.

3.5 Database

Database should be able to store annotated by extractor documents in a form of raw text which will vary in the structure and

type of annotation as the documents from which they will come will vary. However, in the same time it should allow for query formulation based on the existing properties which may exist only for specific documents or even not at all. Also it should not be a problem to insert a document of new format into the database as it may be quite frequent (depending on the crawler configuration).

The record themselves should be documents and contain meta information as from where the document was downloaded, when, and other information depending on the type of document such as for MinLang Committee documents cycle, country producing the report, the language in which the report was written, etc. The record should also contain the stripped down version of the text in document, including the knowledge of document structure (i.e. division into chapters, sections, and paragraphs) as those are also likely to be used in the query formulation (i.e. selecting sections of interest from the range of documents). Moreover, the fields will be of varying type and will mainly include numbers, references (e.g. section number to a field of text), dates, and both short and long chunks of text (short for e.g. titles, and long ones for paragraphs and sections).

Therefore, three things become imminent from the above description

- the database must be schemaless as it is impossible to define it upfront and moreover it needs to be flexible enough to account for different documents that needs to be stored;
- the database must allow for efficient query formulation as they are likely to involve at least few constraints some of which will not be satisfiable due to the type of document (e.g. the section we are looking for does not exist in most of the documents, except the short range of them)[3] and
- the documents should be stored in a format that can be easily rendered back into documents to present results back to the user and allow for creation new documents out of query result for off-line analysis.

[3] This last point also gives a hint how the database should be indexed.

3.5.1 *Example query*

Example query relating information gathered from MinLang committee reports could look like

1. Show all Committee of Experts recommendations under article 7b.
2. Show all sections relating to Scottish Gaelic.
3. Show all sections mentioning word school, relating to Scottish Gaelic, after 2012.

3.6 *Extractor*

The goal of extractor is to compile a document from given format into format that can be queried quickly enough using information both from the website and the document and place documents from the web crawler into the database.

3.7 *Web Crawler*

Web Crawler should either

- download all files in specified extensions (optionally matching given pattern) from the list of websites or
- threat that list as a seed and download all files in the specified extensions up to certain depth (or until exhaustion of links or allocated time).

4 Design

4.1 Database

and will be composed of varying fields from strings, ints, urls, dates, to paragraphs or sections of text. Therefore, the database should allow storing schemaless information and reasonably efficient query processing. It would be optimal if the database would use JSON format to store data as it would simplify the overall design[1]. Document database seems ideal

[1] JSON is a standard format for data exchange on the Internet and has many advantages compared to other formats as simplicity, easy extensibility, interoperability, and number of open source tools available for its processing.

4.1.1 Options

The following database technologies were taken into consideration:

- Traditional RDMS such as PostreSQL and MySQL
- Key-value store such as Riak and Redis
- Document database such as MongoDB and CouchDB
- Graph database such as Neo4J and Infinite Graph

Traditional RDMS has drawback of dedicated to the outlined schema and every new document (not to mention every new source of documents) has a chance to have different organization than the previous ones.

Key-value store has not enough good query languages and

Document database has both flexible structure of aggregates (so we are not dedicated to one schema) and powerful query language (which can be implemented using incremental update)

Graph database has ability to perform rich queries regarding relationships, but is not as powerful when it comes to queries relating the content, can answer queries like 1 and 2 in O(1) time, but update is expensive.

4.1.2 Choice

MongoDB has been chosen for storing documents because

- it has big community of supporters and developers;

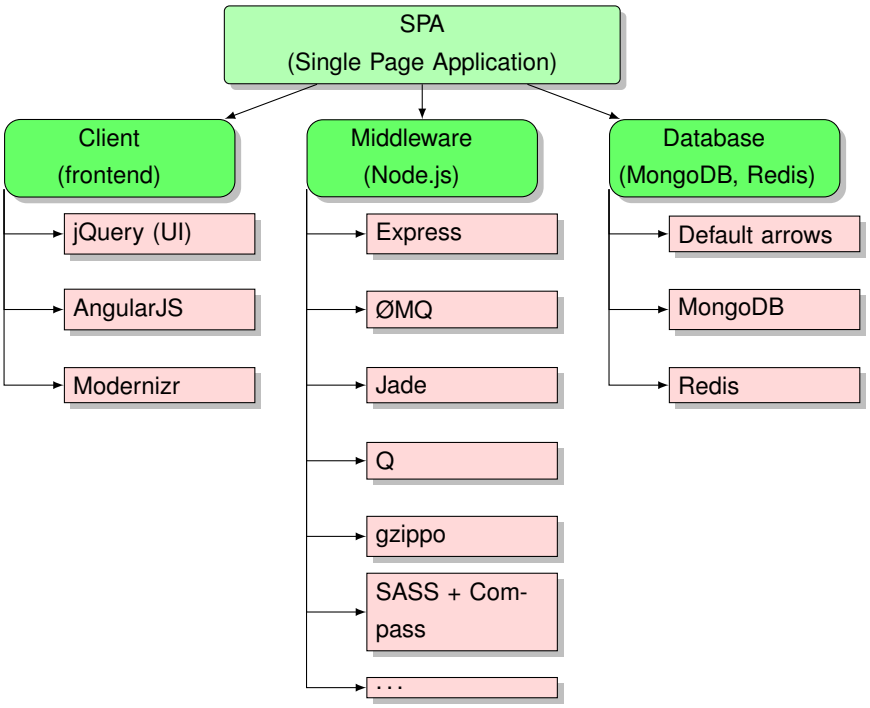
- it is well-documented;
- naturally blends with JavaScript when implementing query (JQuery notation) and data from the database (JSON);
- can store and operate on big documents of varying schema easily
- has rich query language with greater capabilities than the need outlined in the example queries subsection;
- when used with Node.js increases programmers productivity significantly (one language, centralized view)

When developing session and account management functionality it would be beneficial to implement the concept of polyglot persistence and for these two use key-value database, which creates separation of the concers as well as suits better the case.

5 Architecture

5.1 Overview

5.1.1 Application



Description

Client:

- jQuery (UI)
- AngularJS
- Modernizr

Middleware:

- Express - main framework for message passing and service the content to the clients
- ØMQ - for message passing, mainly for efficient Router/Dealer protocol implementation
- Jade - for more productivity in templating
- gzippo - for serving gzipped content

- SASS + Compass - styling, plus features like building sprites out of files

Database:

- MongoDB - document data store for document management *TODO*
- Redis - key-value data store for session and user data management

5.1.2 *Full layer*

Picture showing Yeoman, Grunt, Karma testing with Jasmin, etc. code: sudo npm install -g yo # installs yeoman, grunt, bower, etc. globally sudo npm install -g generator-angular-fullstack # install angular generator (btw the most popular generator) yo angular-fullstack seeker (projectname)# create full stack locally with jade tempalates [?] Would you like to use Sass (with Compass)? Yes âŤĆ [?] Would you like to include Twitter Bootstrap? Yes âŤĆ [?] Would you like to use the Sass version of Twitter Bootstrap? Yes âŤĆ [?] Which modules would you like to include? âŤĆ âĂčâňć angular-resource.js âŤĆ âňć angular-cookies.js âŤĆ âňć angular-sanitize.js âŤĆ âňć angular-route.js mongo with mongosee passport All yeses

5.1.3 *Code management and deployment process*

5.2 *Full stack*

5.2.1 *Client facing*

5.2.2 *Middleware*

5.2.3 *Persistence data store*

5.3 *System design*

5.4 *Query language*

5.4.1 *Elements*

- *source documents* (e.g. Committee of Experts reports)
- *document specifiers* (e.g. cycle, country producing the document)
- *selectors* (i.e., languages, sections, words)
- *organizer* specifying how to structure the output document (e.g. arrange by section numbers, in each section by countries alphabetically)
- *template* choose how to style the output document

5.4.2 Dependencies as tree

1. *Source documents* must be root.
2. *Document specifiers* can be children of each another, but can occur only once on any path from root to leaves, can appear multiple times.
3. *Selectors* (i.e. languages, sections, words) can be children of any node, and they are always terminal (leaves).
4. *Organizers*, *templates*, and other similar options which can be added in the future must be children of root, and in case of *organizers* they must be aware of all root's children except itself to display available options.

5.4.3 Query evaluation stages

1. Select relevant documents using:
   - producer - group (e.g. committee)
   - producer - origin, i.e. countries (e.g. UK)
   - cycles - time (e.g. last)
2. filter the document content using selectors (i.e. sections, languages, filters- words, phrases, etc.)
3. reorganize resulting JSON using organizers
4. render using template

5.5 Implementation

5.5.1 Core

5.5.2 Rendering

5.5.3 Auxiliary modules

5.6 Deployment

5.6.1 Hosting Environment

5.6.2 Connecting to the server

5.6.3 Managing Modules

5.6.4 Deployment Cycle

5.6.5 Testing

5.6.6 Persistence

5.7 Data

5.8 *Perspectives*