University of Huddersfield

MEng Group Project

Cryptic Crossword Solver

Development Document

Authors:
Mohammad Rahman
Leanne Butcher
Stuart Leader
Luke Hackett

Supervisor:
Dr. Gary Allen

Examiner:
Dr. Sotirios Batsakis

Friday, 9th May 2014

Contents

List of Figures

List of Tables

Chapter 1

Development

1.1 Web Service & Servlets

The core system is wrapped around a RESTful web service, that allows users from various devices to submit clues to be solved. Within this section both the web service and the servlet implementations will be discussed and presented.

1.1.1 Web Service

During the project analysis phase the decision was taken that the system's functionality will be delivered via a web service. The web service was developed using Java Enterprise Edition (Java EE) and Tomcat 7.

The main reason for using Java is that the web service could easily make use of the various packages that are provided by the chosen natural language processing library — Apache OpenNLP written in Java.

The web service has solely been produced using the Java EE platform and does not use any additional frameworks or libraries such as Apache Axis. The reason being is that the Java EE platform will run on any machine that is capable of running the Java virtual machine without any additional configuration.

Although Apache Axis (for example) provides additional functionality in configuring the web service, it was decided that the project was to focus upon the solving of a clue. Therefore a 'standard web service' setup would easily meet the requirements of the project.

Another design decision was taken to ensure that the web service followed a RESTful style of communication. The mains reason for this is that some of the target devices (i.e. mobile and tablet platforms) do not support SOAP based communication without additional plug-ins. RESTful web services also provide a number of advantages over their SOAP-based counter parts, as was highlighted within the research section.

1.1.2 Servlets

The servlet design has been split across two classes — `Servlet` and `Solver`.

The `Servlet` class extends the standard `HttpServlet` class and provides common functionality. The `Servlet` class provides a base for all system Servlets to use the common functionality.

The `Servlet` class is able to if a given request is from a JSON, XML or Ajax background. For example if a client was to make a request to the web service through a web browser utilising an Ajax request, then the `isAjaxRequest` method would return `true`.

For illustrative proposes the `isAjaxRequest` method is shown below.

```
  protected boolean isAjaxRequest(HttpServletRequest request) {
    String ajax = request.getHeader("x-requested-with");
    return ajax != null && ajax.toLowerCase().contains("xmlhttprequest");
  }
```

The servlet also contains two customised methods – one to handle errors, and the other to handle a good response – that are able to send a response back to the requesting client based upon a number of factors.

For example the methods are able to convert the return data into either XML or JSON depending upon what the client has asked for. The `sendError` method will also set the HTTP status code correctly, allowing the client to correctly authenticate the response.

Finally the `Servlet` class overrides the `init` method, which is automatically called as part of the object construction. This method will initialise resources at servlet creation (i.e. first run-time within tomcat) rather than during the first call to the servlet.

In doing this, Tomcat will take more time initially starting up, however the user will notice that their queries are dealt with much quicker. In this project the `init` method has been used to initialise the various in-memory dictionaries and thesauri.

The second class is the `Solver` servlet and handles all request that are specifically for solving a given clue. The `Solver` servlet accepts both GET and POST requests, with each requiring the clue, the length of the solution and the solution pattern.

The `Solver` servlet upon receiving a request will validate the input parameters, based upon a number of criteria including presence checks and regular expressions. The code snippet below is an example validation rule, that will validate the solution pattern against a regular expression.

```
 private boolean isPatternValid(String pattern) {
    // Pattern string regular expression
    final String regex = "[0-9A-Za-z?]+((,|-)[0-9A-Za-z?]+)*";
    boolean match = Pattern.matches(regex, pattern);

    // Pattern String must be present and of a valid format
    return isPresent(pattern) && match;
  }
```

In order for validation to pass, the solution pattern must not be empty and must match to the regular expression stated in the method.

The `Solver` servlet class will initialise the solving of a clue if the three inputs are deemed to be valid. The `solveClue` method will utilise the Clue manager class, that will handle the distributing of the clue to the various solvers.

This has been designed so that the servlet and the solving processes are upon separate threads. This prevents tomcat from freezing, and allows it to handle requests from users.

Once all the solvers have finished executing, the `Solver` servlet will produce an XML document based upon the various elements. Once the XML document has been created, will be sent back to the client as either XML or JSON.

1.2 User Interface

In order for users to use the system a simple and powerful user interface was required. The design reasoning's behind the user interface were described in the **??** design section, which can be found on page **??**.

The user interface has been designed utilising a fall-back system, which is a standard web development approach. The majority of the user interface is delivered by the server utilising JavaServer Pages (JSP) language.

The JSP language essentially extends from XML, and allows for html-like code to be written so that when complied with Java, a full server-side page is rendered. Within this project an additional JavaServer Pages Standard Tag Library was used. The library – xml – provided additional functionality so that JSP was able to directly utilise XML within it's rendering technique.

An example code snippet has been presented below from the `solver.jsp` file.

```
<x:choose>
  <x:when select="$solution/trace">
    <p>Solution Trace:</p>
    <ol>
      <x:forEach select="$solution/trace" var="trace">
        <li><x:out select="$trace"/></li>
      </x:forEach>
    </ol>
  </x:when>
  <x:otherwise>
    <p>Solution Trace Unavailable.</p>
  </x:otherwise>
</x:choose>
```

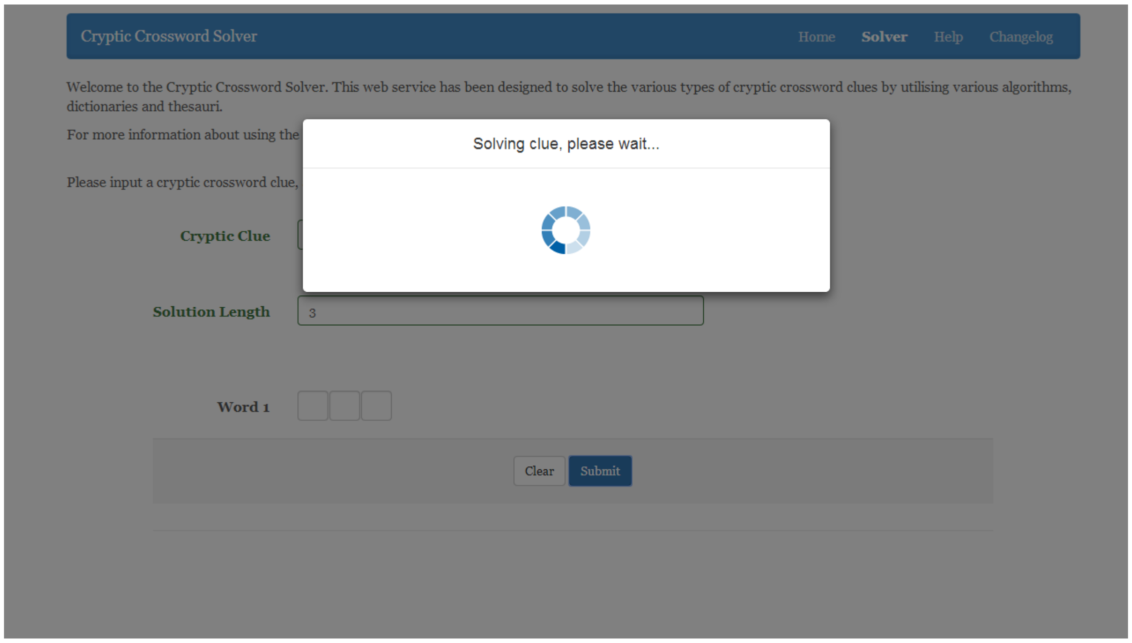
The code snippet above shows use of the XML library, as denoted by the '`x`' name space to certain elements. The code is utilising XPATH to find all possible traces that are listed within the trace element (the trace element is an array).

For each of the traces found they will be printed out into the standard HTML output. However if a solution has not been found a simple predefined message will be presented.

The server side rendering that has been described above is known as the fall-back option. This option will work on all browsers and operating systems. The reason for this is that the rendering is controlled by the server, and hence can be fully tested.

Many modern browsers will support JavaScript in some form of fashion. This allows for various additional functionality to be provided to enhance the user's web browsing experience.

The cryptic crossword website features a JavaScript override, that will override the default server-side rendering and provide it's owner rendering. For example when a user clicks upon the 'solve' button, the web site will display a message informing the user that the clue is currently being solved as shown in figure 1.1. Under a non-JavaScript supporting browser this would simply look like a normal page request that is taking it's time.



Welcome to the Cryptic Crossword Solver. This web service has been designed to solve the various types of cryptic crossword clues by utilising various algorithms, dictionaries and thesauri.

For more information about using the

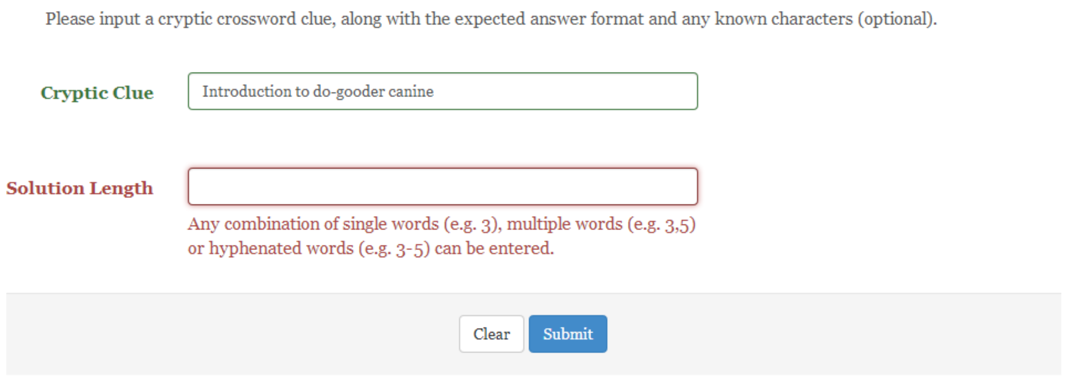
Please input a cryptic crossword clue,

Solving clue, please wait...

3

Figure 1.1: Message informing the user that a clue is currently being solved

The JavsScript engine will also provide some basic forms of validation, to prevent the user from waiting a long time simply to find out that there was a validation error. The validation occurs upon a key press, and hence the feedback is instant as shown in figure 1.2.



Please input a cryptic crossword clue, along with the expected answer format and any known characters (optional).

Introduction to do-gooder canine

Any combination of single words (e.g. 3), multiple words (e.g. 3,5) or hyphenated words (e.g. 3-5) can be entered.

Figure 1.2: Live validation feedback ensures users are entering correct data

The JavaScript engine will also make POST requests to the server. This reduces the total amount of work the server will be required to do, as it only has to return the requests (rather than render HTML).

The fall-back system that was previously described is that if for some reason the JavaScript engine fails to load, or is incompatible with the device, then the JavaScript will not load. However, the functionality of the site will still continue to work, as the browser will 'fall-back' to the standard server-side validation and rendering.

Glossary of Terms

The following section contains a glossary with the meanings of all names, acronyms, and abbreviations used by the stakeholders.

Term/Acronym	**Definition**
The Guardian	A newspaper with a website featuring cryptic crosswords
Blackberry	A mobile phone platform by Blackberry
iOS	A mobile phone platform by Apple
Android	A mobile phone platform by Google
NLP	Natural Language Processing
SRS	Software Requirements Specification
App	Short for application

Bibliography