Software Description

June 2, 2015

Contents

1 Software Outline

Shokuhin will consist of the following Modules:

- Recipe Viewer
- Recipe Editor
- Recipe Search
- MP3 Player

Shokuhin will include the following features:

- Recipes
- Timer Bar
- Image Association
- Text To Speech
- HTML Export
- Cloud Storage/Synchronisation
- Mobile Application

Shokuhin will potentially have the following extensions:

- Voice Commands
- YouTube Viewer

2 Module Descriptions

2.1 Recipe Viewer

The Recipe Viewer will constitute the main function of Shokuhin. It is likely to be indirectly opened by the user via the Recipe Search function, rather than directly.

The main focus of the viewer should be the method of a recipe selected by Recipe Search. It should make use of all available space to ensure that there is little distraction.

The Viewer could be created in two ways. In either instance, a list of Ingredients should be supplied in an appropriate manner.

The first method would rely on the Recipe Method being represented as a list of steps. The GUI space would contain two text areas, one to display the current step, and one to display the next step. This would allow emphasis to be placed on the current step, as it can be displayed in a prominent font style, and the separation inherently encourages focus. The text area for the next step would allow the user to be able to look ahead by one step, albeit in a less emphasised manner to ensure it does not take focus from the user.

The second method would be based on the Recipe Method being a continuous document. The GUI space would contain a single text area, displaying the recipe in full. This does not aid focus on the current step, but would instead allow the user to scroll back and forth through the recipe with more ease.

It is possible that both display methods could be implemented, with the user selecting their preferred method. This preference should be stored as a persistent file.

Finally, a Recipe should be able to adjust measurements of Ingredients based on the number of servings, if the functionality to support this has been implemented (i.e. fixed measurements).

2.2 Recipe Editor

The Recipe Editor will allow a user to create, modify as well as remove a recipe. When creating a recipe, Shokuhin should ensure that a Recipe with the new Title does not already exist. If it does exist, then it should prevent the user from creating the new recipe and committing it to the hard disk.

The Recipe Editor will be accessible in two ways. Firstly, it should be accessible by the user directly in 'Create' mode, where the Recipe Editor is not associated with an existing function, and simply allows the user to create a new Recipe.

Secondly, it should be accessible by the user indirectly in 'Edit' mode. This would be triggered by the user selecting a Recipe in the Recipe Search function, then choosing to edit it. The Recipe Search function would then pass the Recipe to the Recipe Editor function in order to edit it, thereby loading all the Recipe's information in the Recipe Editor's fields. In this mode, the function will have to take special care in the case of editing the Title of a Recipe. It should ensure that the new name doesn't already exist, but it must also rename the existing recipe file. Additionally, keeping the same Title should ensure that the respective Recipe file can be overwritten in this special case.

2.3 Recipe Search

The Recipe Search will allow a user to search for recipes based on any sensible combination of search criteria. For example, a user should be able to search for a Breakfast dish that has been rated 4 out of 5, contains the tag 'quick', and lists 'egg' in its list of Ingredients.

The Recipe Search function should allow the user to either 'Open' or 'Edit' a Recipe. Opening a Recipe will pass the Recipe to the Recipe Viewer, allowing the Recipe to be displayed and used in full. Editing a Recipe will pass the Recipe to the Recipe Editor, allowing it to be modified as the user chooses.

It is in Recipe Search where Image Association could be utilised.

2.4 MP3 Player

The MP3 Player will allow a user to play MP3 files from a library that either the User specifies, or creates as a directory.

The user will then be able to create a playlist of songs, such that can either play a subsection of their library, or play the entire library if they do not wish to create a playlist.

The interface must provide a play, pause, stop, previous and next function. Additionally, the ability to scrub through a song should be present.

The MP3 Player must be able to provide a selection of features (i.e. play, pause, previous, next) as functions on the Timer Bar, allowing quick access to these controls from any Module. These controls should only be available when the MP3 Player module has been initialised, and should be removed when the Module is closed.

3 Feature Descriptions

3.1 Recipes

Recipes are data files that will be persistently stored on disk through serialisation. They will consist of several attributes.

Firstly, they will consist of a Title, which represents the name of the Recipe, and will act as a Primary Key for the recipe. This is to ensure when written to disk, recipes can be saved with their Title as their filename. As such, a Title should be permanently attached to a Recipe.

Secondly, they will contain a list of Ingredients. This should allow for both common Metric and Imperial measurements to be used, as well as colloquial measurements such as 'to taste' or '1 pinch of'.
Potentially, recipes could be exclusively Metric or Imperial, which would require conversion, as well as identification of measurements that should not be converted, such as 'teaspoon'.
As an alternative, measurements could be entirely determined by the user, although this would inhibit the implementation of functionality such as scaling servings.

Thirdly, a Recipe should contain a method. This can be represented in two ways. Firstly, as a list of steps, such that the Recipe Viewer can emphasise the current step, or as a continuous document, such that the Recipe Viewer displays the Recipe in its entirety.

Next, a Recipe should contain a list of courses it is suitable for. This should allow it to be a selection of either Breakfast, Lunch, Dinner, Dessert or Snack. If it does not suit a category, such as a bread recipe, it should be classed as General, and General alone.

Next, a Recipe should contain a list of Tags. These tags should be used to provide keywords that the user personally chooses. These should be comma separated.

Next, a Recipe should contain times that indicate its Total time, Preparation time, and Cooking time. This should be limited to a value of Hours and Minutes.

Penultimately, a Recipe should contain a Rating, that a user can personally allocate. This should be an integer value between 1 and 5.

Finally, a recipe should contain a number of servings that it provides.

3.2 Timer Bar

The Timer Bar is a static function in Shokuhin, which is represented as toolbar at the bottom of the GUI. In being static, it persists across tabs, and is therefore not part of the tab interface.

The Timer Bar consists of a Timer, which will allow a user to select a quantity of seconds, minutes and hours, then start the Timer. The Timer will be able to be stopped at any time. If the Timer is allowed to reach Zero on all fields, then it must produce an audiovisual indication that the Timer has expired. When this indication has been stopped, the Timer should return to its former state.

Additionally, the Timer Bar provides the functionality to push controls onto the Timer Bar, as well as remove them. This will be used by Modules such as the MP3 Player to provide internal functionality without activating the Module's tab.

3.3 Image Association

Image Association is a feature of Shokuhin relating to Recipes.

By creating a directory for Images adjacent to the directory for Recipes, it is possible for the user to associate images with a recipe, by using a shared file name. For example, a recipe titled 'Lemon Meringue Pie' in the Recipes directory would be inherently associated with an image titled 'Lemon Meringue Pie' in the Images directory.

As such, it would be possible to display an image when also displaying a recipe, which could be used in multiple ways. One example would be to display a click-able gallery of Recipes which would load its associated Recipe when selected. Another example would be to simply display an image of the Recipe when a Recipe is selected in the Recipe Search Module.

Alternatively, images could be associated by links to images contained as a field in Recipes. This would mean that a user doesn't have to create a copy of any images they may have just to place in the Images folder, but conversely, the coding to implement image associated would be less complex.

3.4 Text to Speech

The Text to Speech feature will allow the steps in the method of a Recipe to be read out. As such, this would be implemented in the Recipe Viewer Module.

3.5 HTML Export

The HTML Export feature will allow a recipe to be exported into a HTML document that could be displayed in a web browser. This is to ensure that Recipes can be shared with and viewed by non-users of the software.

3.6 Cloud Storage/Synchronisation

The Cloud Storage feature will allow new recipes to be downloaded from a HTML server. This ensures that users of the system can pool their recipes together to

create a shared database.

3.7 Mobile Application

The Mobile Application will provide a subset of the functions and features in Shokuhin to allow usage of it through a mobile device. This may work well with Cloud Storage in order to push Recipes to the mobile device.

4 Extension Descriptions

4.1 Voice Commands

To ease use of the software when cooking whilst cooking a Recipe, voice commands could be used to carry out certain functions, such as proceeding to the next step in a recipe.

4.2 YouTube Viewer

A Module could be created that allows access to YouTube in order to find new recipes without opening a separate web browser.

5 Reusable Code

5.1 Serialisation

5.1.1 Writing an Object to a File

In this example, 'doc' is a class that implements 'Serializable'. As such, it can be written as an object. A serializable class should largely contain data fields such as Strings. It shouldn't contain Objects, particularly Swing Objects, as this may cause serialization to fail. It can however contain methods related to those data fields.

```
String fileDir = "C://Windows/Program Files/";
String fileName = "C://Windows/Program Files/text.doc";

//Create files from the directory and filename
File createDir = new File(fileDir);
File createFile = new File(fileName);

   //This creates all directories specified in createDir
   createDir.mkdirs();
   //This creates the specific file in the created directory
   createFile.createNewFile();

   //Write the Object to a file.
   //Based on code from
      http://www.javapractices.com/topic/TopicAction.do?Id=57
   file = new FileOutputStream(fileName);
   buffer = new BufferedOutputStream(file);
   output = new ObjectOutputStream(buffer);
   //End of 'Based on Code'

   //write the serializable 'doc' Object to file
   output.writeObject(doc);
   output.close();
```

5.1.2 Reading a File into an Object

Here, a file is read in from the Hard Drive, and processed as an Object. It must be cast to the specific type of Object you intend on using.

```
//Based on code from
   http://www.javapractices.com/topic/TopicAction.do?Id=57
//newFile is the Object that will store the Object read in
InternalDocument newFile = null;

//The parameter here points to the file to be read in
InputStream file = new FileInputStream("C:/Windows/test.doc");
BufferedInputStream buffer = new BufferedInputStream(file);
ObjectInput input = new ObjectInputStream(buffer);
//End of 'Based on Code'

//Read the File into the newFile Object
newFile = (InternalDocument)input.readObject();
input.close();
```

5.2 Downloading a File from a HTTP Server

Here, 'file' represents a full filepath including filename, which represents a new file on the hard drive to save the downloaded file to. As with serialisation, all its directories should already exist before using 'createNewFile'.

'fileName' represents the name of the file to be downloaded from the Server.

```
file.createNewFile();
URL url = new URL("http://123.45.678.90/~spastakia/Manabu/" +
   fileName);
ReadableByteChannel channel = Channels.newChannel(url.openStream());
FileOutputStream output = new FileOutputStream(file);
output.getChannel().transferFrom(channel, 0, Integer.MAX_VALUE);
output.close();
```