Blackout Calendar Generator

Devin Raposo, Nathan Robinson, Tamara Wertheim

December 9th, 2017

Project Report Submission - Final

CEN 4065 Software Architecture & Design

Instructor: Dr. Fernando Gonzalez

Department of Software Engineering

Florida Gulf Coast University

Ft. Myers, FL 33965

The Hertz Corporation

Ft. Myers, FL 33965

Contents

1. Introduction

The Hertz Corporation's [5] various Innovation teams led by Felipe Velosa meet weekly at the Florida Gulf Coast University [6] Emergent Technologies Institute [7] to engage with and solve for difficult problems the Hertz Corporation faces in the modern era of vehicle rental services. Each team at the beginning of internship employment were given a project which is meant to provide a potential solution to typical problems a customer with Hertz might face throughout the overall vehicle rental timeline, from pickup to return. The developers of the project described herein were designated the Operations team, an internal Hertz workforce-facing team whose primary goal is to assist other members of the Hertz corporate workforce with achieving their goals. The Operations team were assigned a core project to complete in addition to other, smaller tasks needed to be carried out: a tool which would programmatically automate the creation of the Hertz Corporation's yearly Blackout Calendar [4] webpages through the usage of two independent Microsoft Excel [2] spreadsheet files. Previously, this page would be created yearly by hand by Hertz's Digital Content team, a process which is intensely tedious and laborious.

1.1 Purpose

This Software Design Document describes the design details of the program that will be created by The Hertz Innovation Operations team. The core program is to be a Blackout Calendar Generator which parses date from two Microsoft Excel files—one of which contains an index of relevant Hertz rental locations and the other of which contains the yearly blackout dates— and creates a correctly-formatted and populated HTML [10] page without any CSS [8] or JavaScript [9] treatment.

1.2 Scope

The scope of the project covers the parsing of preformatted Microsoft Excel files, properly formatting the parsed data, and the creation of HTML pages from the formatted data. The scope does not include networking, security, hardware, or user interaction with the program itself. The program will be written in Java, shall utilize the POI XSSF [1] Apache API to read from local Microsoft Excel files and automatically generate the blackout calendar HTML webpage within the same file directory upon application execution.

1.3 Definitions, Acronyms, Abbreviations

Term	Definition
1.3.1 HTML	Hypertext Markup Language
1.3.2 XLSX	Excel Microsoft Office Open XML Format Spreadsheet File
1.3.3 POI XSSF	Java API To Access Microsoft Excel Format Files

Figure 1. Term Definitions

1.4 Summary

The Hertz Innovation Operations team is responsible for creating a portable Java application that will parse through preformatted Microsoft Excel files, format the blackout data, and then generate a HTML page with the properly formatted data. The HTML page can then be uploaded to the Hertz servers as is.

2. Design Overview

2.1 Description of Problem

Every year The Hertz Content team is employed to create a HTML page from two Microsoft Excel files. This HTML page contains all of the blackout data for the calendar year and typically takes around 6 months to hardcode by hand. The Hertz Innovation Operations team will create a Java application which will take the Microsoft Excel files, parse them, format the parsed data, and then dump the formatted data into a HTML page.

2.2 Technologies Used

The technologies used include: The Java Runtime Environment, POI-XSSF and Microsoft Excel.

2.3 System Architecture

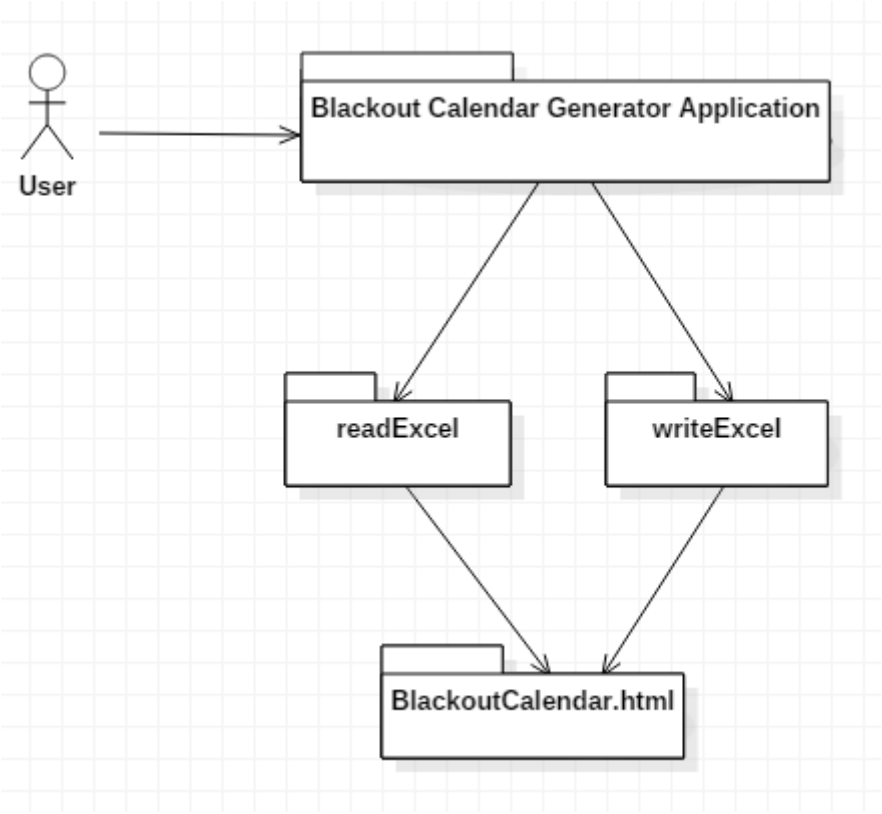

Figure 2. Context Diagram of the System

Figure 1 depicts the high-level architecture. The system is composed of a few main components:

- Blackout Calendar Generator Application - The application to be launch by the user.
- dates.xlsx – The Microsoft Excel file responsible for storing residential area numbers and corresponding blackout dates; this file is to be built by hand according to the specification anew yearly to generate a new calendar for the next year.
- locs.xlsx – The Microsoft Excel file responsible for storing Hertz rental location names, the city that location is in, the state, the residential number, and a link to the location's page.
- readExcel - The Java function which reads from both Excel files and stores that data in an accessible fashion.
- writeExcel - The Java function which writes that data and all other formatting as previously defined by the original Blackout Calendar [4] into a clean, accessible HTML file.
- BlackoutCalendar.html - The HTML file which is created once the program finishes.

2.4 System Operation

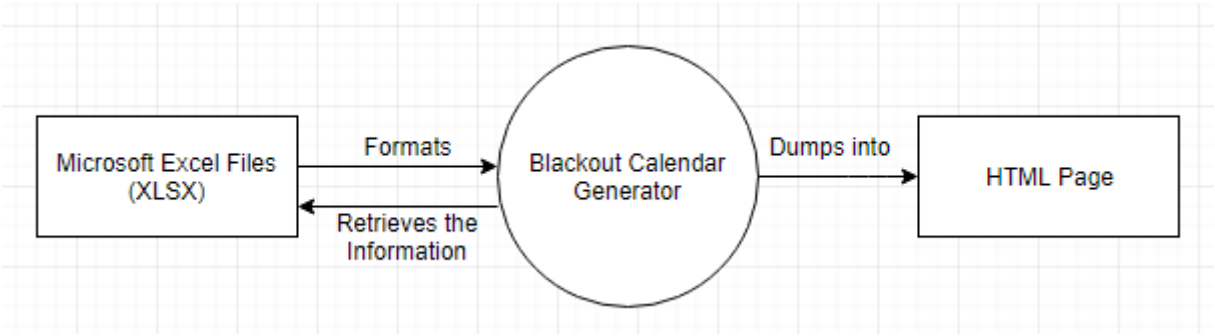

Figure 2. Context Diagram of the System

Figure 2 shows how the software is to operate. The Blackout Calendar Generator retrieves data from the preformatted Excel files. The data is then formatted and dumped into an HTML page.

3. System Architecture

The system's design is extremely simple and quick. It performs its core task—to generate a blackout calendar for the United States, Puerto Rico/Virgin Islands, and Canada from two Excel files—and completes execution. To do this, the program first queries location, city, state, link, and residential area number data from the location Excel file, which is to be named 'locs.xlsx' at all times. It removes trailing spaces ahead of and behind string characters and also recognizes all United States, Puerto Rico, Virgin Islands, and Canada state/providence abbreviations as suitable substitutions for spelling out the entire name of the state/providence. It stores all of this data in one of three arrays of states, one for each country represented by this calendar.

Next, the tool grabs the blackout dates and corresponding residential numbers using Apache's POI XSSF API and couples the dates found to the state arrays by city if the residential number for that city matches one of the residential numbers which is covered by that particular blackout date. At the end of this process, if a city has not been found to contain any blackout dates, it is removed from its corresponding state array for the sake of simplicity when generating the final HTML page. If a state has no cities which contain blackout states, that entire state is removed before generating the HTML page. The states are then sorted in alphabetical order, as are the cities and locations within each individual state, and the blackout dates are sorted in chronological order for each city before moving to the final blackout calendar generation.

Finally, the tool begins to generate the HTML blackout calendar based on the original blackout calendar supplied to the team by Autumn Earle from the Digital Content team and on the data read in from the two Excel files. The tool begins by populating the page with any and all initial formatting code and text as was originally specified from the 2017 Blackout Calendar. It separates the countries, states, cities, and locations with tables. The tool uses simple loops to query through the data retrieved from the Excel files and print that data into the HTML file in such a way that it is identical to the original, only with code to automate aspects of generating the page which would otherwise by extremely laborious and tedious. Upon printing the states from all three countries, the tool's execution has completed.

4. Data Design

4.1 Data Description

The program contains five classes: Locations, which contains a string for the location name and a link to that location's webpage; Dates, which contains a Date value for the start date and a Date value for the end date; Cities, which contains a string for the city name, an ArrayList of Locations for the different locations in a given city, an ArrayList of Dates for the different start and end dates in a given city, and an ArrayList of Strings for the different residential numbers which that city is represented by in the locs file; States, which contains an ArrayList of Cities for each of that state's cities as well a String for the state name; and the main (public) class BlackoutGenerator, which contains all functions. Thus, an instantiation of a State inherently contains instantiations of Cities, Dates, and Locations. There are three instantiations of the State class for the three countries represented: states, prStates (for Puerto Rico), and canadaStates. For all intents and purposes, these are the three main data bodies which constitute the majority of the program. There were three glossaries of United States state names, Canadian provinces, as well as one for Puerto Rico/Virgin Islands to handle translation from abbreviations found in files to the full state name. There are also glossaries for United States, Puerto Rico/Virgin Islands and Canadian state/province abbreviations. Also included are a handful of temporary variables which act as bridges for editing data before it is included in one of the three State instantiations. Each of these State instantiations sits at the top layer of the main BlackoutGenerator class, so they are global to all that class' functions.

4.2 Data Dictionary

```
class Cities
{
    String city;
    ArrayList<Locations> locs;
    ArrayList<Dates> dates;
    ArrayList<String> resNums;
}
```

```
class Dates
{
    Date startDate;
    Date endDate;
}
class Locations
{
    String loc;
    String link;
}
class States
{
    ArrayList<Cities> cities;
    String state;
}
public class BlackoutGenerator
{
     public enum Areas
    {
        UNITED_STATES,
        //Puerto Rico/St. Thomas U.S.V.I.//
        PRST,
        CANADA
    }
    public static String [] stateNames =
    {
        "Alabama", "Alaska", "Arizona", "Arkansas", "California",
"Colorado", "Connecticut", "Delaware", "Florida", "Georgia", "Hawaii",
"Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan",
"Minnesota", "Mississippi", "Missouri", "Montana", "Nebraska",
"Nevada", "New Hampshire", "New Jersey", "New Mexico", "New York",
"North Carolina", "North Dakota", "Ohio", "Oklahoma", "Oregon",
"Pennsylvania", "Rhode Island", "South Carolina", South Dakota",
"Tennessee", "Texas", "Utah", "Vermont", "Virginia", "Washington",
```

```
        "West Virginia", "Wisconsin", "Wyoming"
    };
    public static String [] prNames =
    {
        "Puerto Rico", "Virgin Islands",
    };
    public static String [] canadaNames =
    {
        "British Columbia", "Manitoba", "New Brunswick", "Nova
Scotia", "Ontario", "Prince Edward Island", "Quebec", "Saskatchewan"
    };
    public enum StateAbrev
    {
        AL, AK, AZ, AR, CA, CO, CT, DE, FL, GA, HI, ID, IL, IN, IA,
KS, KY, LA, ME, MD, MA, MI, MN, MS, MO, MT, NE, NV, NH, NJ, NM, NY,
NC, ND, OH, OK, OR, PA, RI, SC, SD, TN, TX, UT, VT, VA, WA, WV, WI, WY
    }
    public enum PRAbrev
    {
        PR, VI,
    }
    public enum CanadaAbrev
    {
        BC, MB, NB, NS, ON, PE, QC, SK
    }
    public static ArrayList<States> states;
    public static ArrayList<States> prStates;
    public static ArrayList<States> canadaStates;
    public static DateFormat df;//for handling correct date formatting
    public static void main(String[] args);
    public static void readExcel(); //read from both Excel files
   public static String fixSpaces(String string);//fix trailing spaces
    public static String fixDates(String string);//fix date formatting
    public static void writeHTML(); //write the Excel data to HTML
```

```
   public static ArrayList<String> parseRes(String string); parse res
area numbers from the locs file
```

5. Component Design

There are two main functions represented within this program: `readExcel()` and `writeHTML()`. In `readExcel()`, first the locs file is opened using Java's native FileInputStream [11] functionality. It creates an XSSFWorkbook [12] from this file, then an XSSFSheet [13] is created from that workbook. Next, an Iterator [14] is made to iterate through the Rows [15] of the Excel spreadsheet which we've loaded in. The rows are iterated through with this Iterator, using a Cell [16] to query the individual columns from the locs file. The program checks if a state name found is an abbreviation and queries its abbreviation atlas to convert this to a full state name for final presentation purposes. This also lets the program know which country the state is in; if it's not an abbreviation, the program needs to determine this by checking the full state name against the repository of full state names for United States, Puerto Rico/Virgin Islands, and Canada. Next, the function checks to see if that state has already been added to the ArrayList of states for its corresponding country, and adds if it is not. It does the same for the city found, as well as the location. If the city is not found, an ArrayList for that city is allocated. If the location is not found, a location is allocated from that data. Next, the function reads from the dates file using the same data types as with the locs file. The date file consists of a column with residential area numbers separated by commas, a column with starting blackout dates, and a column with ending blackout dates. The program separates each of the residential numbers from this column using an ArrayList of Strings generated from a function called `parseRes(String string)`. The function then uses a loop to go through all of the cities and check if a city there contains an instance of one of those res numbers, which would indicate that that blackout date applies to that city. If it does, it adds that starting and ending blackout date to the Dates ArrayList in the Cities instantiation; otherwise it disregards that city and continues on. Now that the program knows which cities have blackout dates and which don't, the function uses a loop to check for cities and states which don't have blackout dates and delete them. Finally, the function uses the Collections [17] class' sort function to sort the ArrayLists according to a specific member field for presentation purposes. Thus, the State ArrayField is sorted by state name alphabetically, and each of the City ArrayFields within each State is sorted by city name alphabetically; each of the Location names within those cities is sorted alphabetically, and the dates within each city is sorted chronologically.

The second main function is `writeHTML()`, and is comparatively much simpler. It uses a PrintWriter [18] to print raw HTML code to a file titled ‘BlackoutCalendar.html’. It populates tables formatted according to the original specification from the previous 2017 calendar with the data pulled from both the dates and locs files which were saved in the three States instantiations using the PrintWriter’s print function. For legibility’s sake, just as it in the original it switches between grey and white background for city table rows. After going through and printing all of the saved data for each of the three countries represented, the function is now complete and the program execution has finished.

6. Human Interface Design

This application features no visual user interface. All it takes is to run the application, which will produce the Blackout Calendar after a few seconds of execution. Of course, both Excel files are opened in Excel; thus, the user will have to engage with Microsoft Excel's user interface to make changes to those files if they so wish to.

7. References

1. POI-HSSF and POI-XSSF - Java API To Access Microsoft Excel Format Files. (n.d.). Retrieved December 09, 2017, from https://poi.apache.org/spreadsheet/

2. Excel 2016-get it now with an Office 365 subscription. (n.d.). Retrieved December 09, 2017, from https://products.office.com/en-us/excel

3. Java SE Downloads. (n.d.). Retrieved December 09, 2017, from http://www.oracle.com/technetwork/java/javase/downloads/index.html

4. Hertz Gold Plus AnyDay Reward Periods 2017. (n.d.). Retrieved December 09, 2017, from https://www.hertz.com/rentacar/misc/index.jsp?targetPage=2017_AnyDay_Rewards_Points.jsp

5. (n.d.). Retrieved December 09, 2017, from https://www.hertz.com/rentacar/reservation/

6. Florida Gulf Coast University. (n.d.). Retrieved December 09, 2017, from https://www2.fgcu.edu/

7. Florida Gulf Coast University. (n.d.). Retrieved December 09, 2017, from https://www2.fgcu.edu/Eng/eti.html

8. Cascading Style Sheets home page. (n.d.). Retrieved December 09, 2017, from https://www.w3.org/Style/CSS/Overview.en.html

9. JavaScript. (n.d.). Retrieved December 09, 2017, from https://www.javascript.com/

10. Home. (n.d.). Retrieved December 09, 2017, from https://html.com/

11. (2017, October 09). Retrieved December 09, 2017, from https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html

12. (n.d.). Retrieved December 09, 2017, from https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html

13. (n.d.). Retrieved December 09, 2017, from https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html

14. (2017, October 09). Retrieved December 09, 2017, from https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

15. (n.d.). Retrieved December 09, 2017, from https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/Row.html

16. (n.d.). Retrieved December 09, 2017, from https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/Cell.html

17. (2017, October 09). Retrieved December 09, 2017, from https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

18. (2017, October 09). Retrieved December 09, 2017, from https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html