Fall 2015

Web Crawler

Michael Schmidt
Governors State University

Follow this and additional works at: http://opus.govst.edu/capstones

Part of the Databases and Information Systems Commons

Recommended Citation
http://opus.govst.edu/capstones/148

For more information about the academic degree, extended learning, and certificate programs of Governors State University, go to http://www.govst.edu/Academics/Degree_Programs_and_Certifications/

Visit the Governors State Computer Science Department
This Project Summary is brought to you for free and open access by the Student Capstone Projects at OPUS Open Portal to University Scholarship. It has been accepted for inclusion in All Capstone Projects by an authorized administrator of OPUS Open Portal to University Scholarship. For more information, please contact opus@govst.edu.
Web Crawler

By

Michael Schmidt

B.S. Computer Science, Governors State University, 2013

GRADUATE CAPSTONE SEMINAR PROJECT

Submitted in partial fulfillment of the requirements

For the Degree of Master of Science,

With a Major in Computer Science

Governors State University
University Park, IL 60484

2015
# Table of Contents

Web Crawler .............................................................................................................................................. 1

1 Project Description ................................................................................................................................. 2
   1.1 Project Abstract ................................................................................................................................. 2
   1.2 Competitive Information .................................................................................................................... 2
   1.3 Relationship to Other Applications/Projects .................................................................................... 2
   1.4 Assumptions and Dependencies ....................................................................................................... 2
   1.5 Future Enhancements ....................................................................................................................... 2
   1.6 Definitions and Acronyms ............................................................................................................... 3

2 Technical Description ............................................................................................................................... 3
   2.1 Project/Application Architecture ......................................................................................................... 3
   2.2 Project/Application Information flows .............................................................................................. 3
   2.3 Interactions with other Projects (if Any) ............................................................................................ 4
   2.4 Interactions with other Applications ................................................................................................ 4
   2.5 Capabilities ....................................................................................................................................... 4
   2.6 Risk Assessment and Management .................................................................................................. 4

3 Project Requirements ................................................................................................................................. 4
   3.1 Identification of Requirements ........................................................................................................... 4
   3.2 Operations, Administration, Maintenance and Provisioning (OAM&P) ........................................... 5
   3.3 Security and Fraud Prevention .......................................................................................................... 5
   3.4 Release and Transition Plan .............................................................................................................. 5

4 Project Design Description ........................................................................................................................ 5

5 Print Screens ........................................................................................................................................... 6

6 Acknowledgements ................................................................................................................................... 8

7 References ............................................................................................................................................... 8

8 Appendix ............................................................................................................................................... 8
1 Project Description

1.1 Project Abstract

A web crawler is a piece of code that travels the Internet and collects data from various web pages, also known as web scraping. Some web crawlers are autonomous and require no instructions once started. This project will focus on a user driven web crawler where user input will direct where the crawler goes and how the collected data is analyzed. Web scraping replaces the need for manual data entry and more easily reveals trends among data collected. It can also aggregate information from multiple sources into one central location. While this application provides three specific examples of web crawling/scraping, it could be easily altered to better suit additional markets and/or needs.

1.2 Competitive Information

Identify competitor products, applications, or services that this project will directly compete against, and whether your team has the potential to be the first to market this new project application or capability.

Web crawlers are currently being used in various markets to collect and analyze information. This information can provide the ability to gain an edge over competitors such as showing the average price of their products. Web crawling has been around for over a decade but can be applied to new and emerging markets. With a few small changes in code most websites can be scraped for information.

1.3 Relationship to Other Applications/Projects

There are many open source web crawlers available for public use. This one is unique in that it was written specifically for three websites in particular. It would need to be modified to be used for any other sites.

1.4 Assumptions and Dependencies

- This project depends on both the jsoup and MongoDB Java libraries. Should these libraries change or lose functionality, code changes would be required to keep the application running smoothly.
- This project also depends on the websites it scrapes for information. If the HTML code changes on any of the pages, the application’s code would need to be updated in order to continue working.
- Future sites desired to be supported would each need to be looked at and coded individually.

1.5 Future Enhancements

Discuss any planned or possible evolution of the project. Some project will need to be delivered in phases; this will be the place to document your plan.

More websites can be supported for data scraping in future releases. Every website uses different html id tags and therefore will need to be coded on a case-by-case basis.
1.6 Definitions and Acronyms
Web crawler – an application or piece of code that travels about websites in an attempt to gather information

Web scraper – a web crawler that stores pieces of information taken directly from the websites it has crawled

2 Technical Description
There are two major components that make up this application: a Mongo database and the application code. The web scraper code is written in Java and is comprised of a data access layer and business layer. It uses the jsoup library to easily pull out HTML elements from the webpages it traverses. There are several classes that are used in the business layer to grab pages from the web. The data access class uses the MongoDB library and connects the application to the MongoDB server. This allows for the saving and retrieval of information in the database.

2.1 Project/Application Architecture

2.2 Project/Application Information flows
There is not a UI at this time but one could be added for easier use. The command line is used to take user input and present the user with the options that are available to them. From the main menu there are three options: crawl for data, view data, and analyze data. The crawl for data menu allows the user to choose from three different websites to scrape for data. Once an option has been chosen, the site is scraped and the data is saved to the MongoDB server. If a user wants to view the data in the database, they can choose to do so from the main menu. The database menu has three options to choose from. The user can choose to print out the database names, the collection names, or the collection contents. There will be one database used for this application.
and three separate collections, one for each of the websites. The last option on the main menu allows the user to perform analytics on the data that has been stored into the MongoDB server.

2.3 Interactions with Other Projects (if Any)
There are currently no interactions with other projects at this time but could be added in future releases.

2.4 Interactions with Other Applications
At this time there are no interactions with other applications. The only outside interactions this application has are with the Internet and MongoDB server.

2.5 Capabilities
Provide a list of the capabilities need to support this Project. Example: a database application must provide capabilities to support business application such as retrieving/adding/deleting/updating user data.
This application has a database that is used to store and retrieve data and so the business layer must allow the user to easily access it. The data access class allows for the addition and retrieval of data. At this time, the user is not able to delete or update data. The business reason behind this is that the information should be unaltered from the website it has been scraped from. A future option may allow users to delete information that is stored on the MongoDB server in order to free up space in the database, but the user will never be allowed to edit saved information.

2.6 Risk Assessment and Management
Web crawling and scraping both land in a grey area of computing. Information that is gathered should be used responsibly in order to avoid possible legal actions taken by the websites crawled. All crawling and scraping that is performed in this project is purely for research and is legal in regards to all laws. Web crawling/scraping is legal so long as the information being collected does not violate copyright laws and/or is not used for financial gains. Web crawling laws should be monitored to ensure the legality of this application in the future.

3 Project Requirements

3.1 Identification of Requirements
This section provides a brief explanation of the use of named and enumerated requirements to identify and number requirements. The following format is an example:

<GSU-GS_FA2015-1 WebScraping-1>
The project must allow users to scrape websites for data.
Implementation: Mandatory

<GSU-GS_FA2015-1 DBActions-1>
The project must allow users to view or print data stored in the database.
Implementation: Mandatory

<GSU-GS_FA2015-1 DBActions-2>
The project must allow users to delete data stored in the database.
Implementation: Optional (This is not being implemented)

The project must allow users to perform analytics on data stored in the database. Implementation: Mandatory

3.2 Operations, Administration, Maintenance and Provisioning (OAM&P)
Throughout the use of this application, backups should be taken regularly of the MongoDB server. This will prevent the loss of data collected. The application may have to be updated should any of the supported websites’ HTML change.

3.3 Security and Fraud Prevention
Access to this application is open to all. Since the information gathered is public knowledge, there is no need for application security. Users are not allowed to edit information stored in the MongoDB. Integrity of data is upheld by ensuring that data is pulled directly from the websites and saved directly to the MongoDB.

3.4 Release and Transition Plan
Future application updates can be obtained from a Git repository such as GitHub.

4 Project Design Description
There are two major design pieces for this application: the application code and the MongoDB server.

The MongoDB server is hosted locally and can be started using a command line shell. Once the MongoDB server is started, connections to it can be made. One connection is maintained while the application is running. Once the user exits the application, this connection is closed to the MongoDB server.

Application code runs within NetBeans and uses the command line to display options to the user. The code itself consists of a business layer and a data access layer. Only the data access layer can open and close connections to the running MongoDB server. The business layer contains all of the objects used to scrape the Internet for information. The various menus available to the user are described above in section 2.2.

Below are the prompts the user will encounter:

Web Crawler - Main Menu
Please make a selection
1. Scrape Web for Data
2. View Database
3. Analyze Scraped Data
4. Exit

Selection 1 will bring up the following menu:
Please choose one of the following to scrape:
1. Movie titles from Movies.com
2. 5 Day weather forecast from Weather.com
3. Week 11 NFL scores from NFL.com

Selection 2 will bring up the following menu:
Please choose a database option:
1. Print list of databases
2. Print list of collections
3. Print contents of collection

Selection 3 will bring up the following menu:
Please choose an operation to perform:
1. Print all movies containing the string 'the'
2. Average temperature over next 5 days
3. Average points scored by NFL home team

As of now all menu options are hard coded into the application. In the future the user may have more analytics to perform on data stored in the MongoDB. Another future feature may allow users to cycle through the websites. For example, NFL scores can only be gathered from week 11. In the future it may ask the user which week they would like to scrape data from and then dynamically insert that into the URL used for crawling.

5 Print Screens

MongoDB running from command prompt:
Movie titles pulled from Movies.com:

Average NFL home team points for week 11:
6 Acknowledgements

I would like to extend great thanks to Dr. Park and all that she has taught me over the years. My foundational database skills were formed in her class and have come a long way since my undergraduate studies.

I would also like to thank my friends and family for always supporting me in my pursuit of a graduate degree. I could not have done this without you guys!

7 References


8 Appendix

Business class code

package webcrawler;

import MSchmidtDataAccess.DataAccess;
import MSchmidtBusiness.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

/**<*
 * @author Mike Schmidt
 */
public class WebCrawler {
  
  public static void main(String[] args) throws Exception {
      int menu = 1;
      int webCrawl = 0;
      int dbSelection = 0;
      int operSelection = 0;
      List<String> movieData = new ArrayList<>();
      List<String> weatherData = new ArrayList<>();
      List<String> NFLData = new ArrayList<>();
      //DataAccess da = new DataAccess();
      Scanner input = new Scanner(System.in);

      while (menu != 4) {

printMenu();
menu = input.nextInt();

switch (menu) {
    case 1:
        printWebCrawl();
        webCrawl = input.nextInt();
        switch(webCrawl) {
            case 1:
                movieData = crawlWeb(webCrawl);
                break;
            case 2:
                weatherData = crawlWeb(webCrawl);
                break;
            case 3:
                NFLData = crawlWeb(webCrawl);
                break;
        }
        break;
    case 2:
        printDatabaseMenu();
        dbSelection = input.nextInt();
        //printDBContents(da, dbSelection);
        break;
    case 3:
        printAnalyticsMenu();
        operSelection = input.nextInt();
        switch(operSelection) {
            case 1:
                performAnalytics(movieData, operSelection);
                break;
            case 2:
                performAnalytics(weatherData, operSelection);
                break;
            case 3:
                performAnalytics(NFLData, operSelection);
                break;
        }
        break;
    case 4:
        break;
}
}

private static List<String> crawlWeb(int webCrawl) throws Exception {
    List<String> movieList = new ArrayList<>();
    List<String> tempList = new ArrayList<>();
    List<String> NFLList = new ArrayList<>();
    Domain domain;
    Anchor anchor;
    switch(webCrawl) {
        case 1:
            domain = new Domain("http://www.movies.com/movie-times/60452-movie-times");
            anchor = new Anchor(domain, "http://www.movies.com/movie-times/60452-movie-times");
            break;
case 2:
    break;

    case 3:
    domain = new Domain("http://www.nfl.com/scores/2015/REG11");
    anchor = new Anchor(domain, "http://www.nfl.com/scores/2015/REG11");
    break;

default:
    return new ArrayList<>();

}

WebPage webPage = new WebPage(anchor);
webPage.loadWebPageIntoDoc();

switch (webCrawl) {
    case 1:
        Elements movies = webPage.getDocument().select("a");
        for(Element e : movies) {
            if(e.attr("id").contains("MovieName") && !movieList.contains(e.text())) {
                movieList.add(e.text());
            }
        }
        for(String s: movieList) {
            System.out.println(s);
        }
        return movieList;

    case 2:
        int counter = 0;
        Element weather = webPage.getDocument().select("div#feed-tabs").first();
        Elements dates = weather.select("h4");
        Elements temps = weather.select("strong.temp");
        System.out.println("5 Day Forecast");
        for (int i = 0; i < 5; i++) {
            tempList.add(temps.get(i).text().substring(0, 2));
            System.out.println(dates.get(i).text() + " - " + temps.get(i).text());
        }
        System.out.println("\n");
        return tempList;

    case 3:
        Elements NFLScores = webPage.getDocument().select("div.scorebox-wrapper");
        for(Element e : NFLScores) {
            System.out.println(e.select("span.date").text());
            System.out.println(e.select("p.team-name").first().text() + " - " + e.select("p.total-score").first().text());
            System.out.println(e.select("p.team-name").last().text() + " - " + e.select("p.total-score").last().text());
            System.out.println("\n");
            NFLList.add(e.select("p.total-score").last().text());
        }
        System.out.println("\n");
        return NFLList;

    default:
        System.out.println("Invalid selection. Returning to main menu.");
        return new ArrayList<>();
}
private static void printDBContents(DataAccess da, int dbSelection) {
    Scanner input = new Scanner(System.in);
    switch (dbSelection) {
    case 1:
        da.printDatabases();
        break;
    case 2:
        da.printCollections();
        break;
    case 3:
        System.out.println("Please enter name of collection");
        da.printCollection(input.next());
        break;
    default:
        System.out.println("Invalid selection. Returning to main menu.");
        return;
    }
}

private static void performAnalytics(List<String> scrapeData, int operSelection) {
    switch (operSelection) {
    case 1:
        System.out.println("Movie titles containing the string 'the':");
        for (String s : scrapeData) {
            if (s.contains("The") || s.contains("the")) {
                System.out.println(s);
            }
        }
        System.out.println("\n");
        break;
    case 2:
        System.out.println("The average temperature over the next five days is:");
        int totTemp = 0;
        for(String s : scrapeData) {
            totTemp += Integer.parseInt(s);
        }
        System.out.println(totTemp/scrapeData.size() + " degrees");
        System.out.println("\n");
        break;
    case 3:
        System.out.println("The average score for home teams is:");
        int totScore = 0;
        for(String s : scrapeData) {
            totScore += Integer.parseInt(s);
        }
        System.out.println(totScore/scrapeData.size());
        System.out.println("\n");
        break;
    default:
        System.out.println("Invalid selection. Returning to main menu.");
        }
}
public void insertDataIntoDB(Elements elements) {
    
}

public static void printMenu() {
    System.out.println("Web Crawler - Main Menu");
    System.out.println("---------------------");
    System.out.println("Please make a selection");
    System.out.println("1. Scrape Web for Data");
    System.out.println("2. View Database");
    System.out.println("3. Analyze Scraped Data");
    System.out.println("4. Exit");
    System.out.print("\n");
}

private static void printAnalyticsMenu() {
    System.out.println("Please choose an operation to perform:");
    System.out.println("1. Print all movies containing the string 'the'");
    System.out.println("2. Average temperature over next 5 days");
    System.out.println("3. Average points scored by NFL home team");
}

private static void printDatabaseMenu() {
    System.out.println("Please choose a database option:");
    System.out.println("1. Print list of databases");
    System.out.println("2. Print list of collections");
    System.out.println("3. Print contents of collection");
}

private static void printWebCrawl() {
    System.out.println("Please choose one of the following to scrape:");
    System.out.println("1. Movie titles from Movies.com");
    System.out.println("2. 5 Day weather forecast from Weather.com");
    System.out.println("3. Week 11 NFL scores from NFL.com");
}

DataAccess Class
package MSchmidtDataAccess;

import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import org.bson.Document;
/**
 * @author Mike Schmidt
 */
public class DataAccess {
    private final MongoClient mongo;
    private final MongoDatabase db;

    public static void main(String[] args) throws Exception {
        int menu = 0;
        int dbPrint = 0;
        Scanner input = new Scanner(System.in);

        while (menu != 3) {
            printMenu();
            menu = input.nextInt();
            switch (menu) {
                case 1:
                    printViewMenu();
                    dbPrint = input.nextInt();
                    printDatabase(dbPrint);
                    break;
                case 2:
                    insertDocument();
                    break;
                case 3:
                    break;
                default:
                    System.out.println("Invalid selection.");
                    break;
            }
        }
    }

    public DataAccess() {
        this.mongo = new MongoClient("localhost", 27017);
        this.db = mongo.getDatabase("webcrawler");
    }

    private static void printDatabase(int dbPrint) {
        DataAccess da = new DataAccess();
        Scanner input = new Scanner(System.in);
        switch(dbPrint) {
            case 1:
                da.printCollections();
                break;
            case 2:
                System.out.println("Which collection would you like to print?");
                da.printCollection(input.next());
                break;
        }
        da.closeConnection();
    }

    private static void insertDocument() {
        DataAccess da = new DataAccess();
        Document document = new Document();
        Scanner input = new Scanner(System.in);
        List<String> dataToInsert = new ArrayList<>();
        char quit = 'y';
        }
boolean addMore = true;
String docTag, docValue;

System.out.println("Which collection would you like to insert into?");
String colName = input.next();
MongoCollection col = da.db.getCollection(colName);

while(addMore) {
    System.out.println("Enter document tag name");
    docTag = input.next();
    System.out.println("Enter document value");
    docValue = input.next();
    document.put(docTag, docValue);
    System.out.println("Would you like to add more to this document? y/n");
    quit = input.next().charAt(0);
    if (quit == 'n') {
        addMore = false;
    }
} 
col.insertOne(document);
da.closeConnection();

public void printCollections() {
    System.out.println("Collections: ");
    MongoIterable<String> colNames = db.listCollectionNames();
    for (String s: colNames) {
        System.out.println(s);
    }
}

public void printDatabases() {
    System.out.println("Databases: ");
    MongoIterable<String> dbs = this.mongo.listDatabaseNames();
    for (String s: dbs) {
        System.out.println(s);
    }
}

public void closeConnection() {
    this.mongo.close();
}

public void printCollection(String collection) {
    MongoCollection coll = db.getCollection(collection);
    BasicDBObject query = new BasicDBObject();
    FindIterable curs = coll.find(query);
    for (Object i: curs) {
        System.out.println(i.toString());
    }
}

public static void printMenu() {
    System.out.println("MongoDB Interface");
    System.out.println("---------------------");
System.out.println("Please make a selection");
System.out.println("1. View Database");
System.out.println("2. Insert Data");
System.out.println("3. Exit");
System.out.println(")

public static void printViewMenu() {
    System.out.println("Please make a selection");
    System.out.println("1. View Collections");
    System.out.println("2. Print Collection");
    System.out.println(");
}