**Author: Aditi Gupta - argupta**

**Task 1:**

1. **Screen shots** of input, MD5 and SHA-256 output, both in hex and base 64

**Input for MD5**

A screenshot of a calculator

Description automatically generated

**Output of MD5 in Hexadecimal and Bas64**

A close-up of a text

Description automatically generated

**Input of SHA-256**

A screenshot of a calculator

Description automatically generated

**Output of SHA-256 in Hexadecimal and Bas64**

A black text on a white background

Description automatically generated

1. **Code snippets** of computation of each hash

/\*  
\* Author: Aditi Gupta - argupta  
\* Last Modified: September 22, 2023  
Explanation:  
This is a Java Servlet (ComputeHashes) that computes cryptographic hashes (e.g., MD5, SHA-1) for a given input text.  
The doGet method is invoked when a client sends an HTTP GET request to the specified URL (/ComputeHashes).  
It retrieves the text and hashType parameters from the request, which represent the text to be hashed and the hash algorithm to use.  
The computeHash method is called to calculate the hash value based on the selected hash type.  
The computed hash is then displayed in an HTML response, along with the original text and hash type.  
Two representations of the hash are shown: hexadecimal and Base64-encoded.  
The code handles potential exceptions, such as when the specified hash algorithm is not available (NoSuchAlgorithmException).  
\* \*/  
  
@WebServlet(name = "ComputeHashes", value = "/ComputeHashes")  
public class ComputeHashes extends HttpServlet {  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {  
 response.setContentType("text/html");  
 PrintWriter out = response.getWriter();  
  
 // Get the text and hashType parameters from the request  
 String text = request.getParameter("text");  
 String hashType = request.getParameter("hashType");  
  
 // Compute the hash for the provided text and hash type  
 String hashResult = computeHash(text, hashType);  
  
 // Generate an HTML response to display the hash result  
 out.println("<html><head><title>Hash Result</title></head><body>");  
 out.println("<h1>Hash Result</h1>");  
 out.println("<p>Original Text: " + text + "</p>");  
 out.println("<p>Selected Hash Type: " + hashType + "</p>");  
 out.println("<p>Hexadecimal Hash: " + hashResult + "</p>");  
 out.println("<p>Base64 Hash: " + DatatypeConverter.*printBase64Binary*(hashResult.getBytes()) + "</p>");  
 out.println("</body></html>");  
 }  
  
 // This method computes a hash of the provided text using the specified hash type  
 private String computeHash(String text, String hashType) {  
 try {  
 // Create a MessageDigest instance for the specified hash type  
 MessageDigest md = MessageDigest.*getInstance*(hashType);  
  
 // Compute the hash of the text as bytes  
 byte[] hashBytes = md.digest(text.getBytes());  
  
 // Convert the hash bytes to a hexadecimal string  
 return DatatypeConverter.*printHexBinary*(hashBytes);  
 } catch (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 return "Hashing Failed";  
 }  
 }  
}

### Task 2:

1. **Screen shots** of input page(s) and output page(s).

**Input page 1:** index.jsp – Asking for user input for a particular question.

A white background with black text

Description automatically generated

**Input page 2:** After selecting a response: For example, I have selected ‘A’ here.

A screenshot of a computer

Description automatically generated

**Input page 3:** submit.jsp – Submitting a particular responseA close-up of a question

Description automatically generated

**Output page:** result.jsp – Looking at the results of response submitted, the output page gives the number of times a response was selected and a link to another question.

A black text on a white background

Description automatically generated

1. **Code snippets** for producing clicker output.

Model used for producing clicker output:

/\*  
\* Explanation:  
\* Author: Aditi Gupta - argupta  
\* Last Modified: September 22, 2023  
ClickerModel is a Java class that represents the model component of a simple voting or polling application.  
It maintains a static optionMap which is a map to store the vote counts for each voting option.  
The recordVote method is used to record a user's vote for a given option.  
It takes an option parameter, representing the option that the user voted for.  
It uses getOrDefault to increment the vote count for the specified option. If the option doesn't exist in the map, it initializes it to 1.  
The getResults method retrieves the voting results.  
It returns a copy of the optionMap to prevent external modifications to the map.  
The clearResults method clears the voting results.  
It empties the optionMap to reset the vote counts, allowing a fresh round of voting.  
This class serves as the data model for the voting application, responsible for storing, recording, retrieving, and clearing voting results.  
\* \*/  
public class ClickerModel {  
 // A static map to store vote counts for each option  
 public static Map<String, Integer> *optionMap* = new HashMap<>();  
  
 // Method to record a vote for a given option  
 public void recordVote(String option) {  
 // Increment the vote count for the specified option, or initialize to 1 if it doesn't exist  
 *optionMap*.put(option, *optionMap*.getOrDefault(option, 0) + 1);  
 }  
  
 // Method to retrieve the voting results  
 public Map<String, Integer> getResults() {  
 // Return a copy of the optionMap to prevent direct modification  
 return new HashMap<>(*optionMap*);  
 }  
  
 // Method to clear the voting results  
 public void clearResults() {  
 // Clear the optionMap to reset the vote counts  
 *optionMap*.clear();  
 }  
}

Servlet used for producing clicker output:

/\*Explanation:  
\* Author: Aditi Gupta - argupta  
\* Last Modified: September 22, 2023  
This is a Java Servlet (ClickerServlet) that handles user voting and result retrieval for a simple clicker application.  
The servlet is mapped to two URL patterns: "/submit" and "/getResults."  
The ClickerModel instance is created to handle the underlying business logic of recording votes and managing results.  
The doGet method is invoked when a client sends an HTTP GET request to the servlet.  
It retrieves the "option" parameter from the request, which represents the user's vote choice.  
If no option is selected (option is null), it redirects the user back to the index page (/index.jsp).  
If an option is selected, it records the user's vote using the ClickerModel and forwards the request to the "submit.jsp" view, displaying the selected option.  
If the request is for getting results (/getResults), it retrieves the voting results from the ClickerModel, sets them as an attribute for the "result.jsp" view, and then clears the results in the model to start fresh.  
If the request is not for getting results, it redirects the user back to the index page.\*/  
@WebServlet(name = "ClickerServlet", urlPatterns = {"/submit", "/getResults"})  
public class ClickerServlet extends HttpServlet {  
 private static final long *serialVersionUID* = 1L;  
 private final ClickerModel clickerModel = new ClickerModel();  
  
 // This method handles HTTP GET requests  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String option = request.getParameter("option");  
  
 // Code to not register a vote if no answer is selected  
 if (option == null) {  
 // If no option is selected, redirect back to the index page  
 request.getRequestDispatcher("/index.jsp").forward(request, response);  
 return;  
 }  
  
 // Record the user's vote using the ClickerModel  
 clickerModel.recordVote(option);  
  
 // Set the selected option as an attribute for the view  
 request.setAttribute("option", option);  
  
 // Forward the request to the "submit.jsp" view  
 request.getRequestDispatcher("/submit.jsp").forward(request, response);  
  
 // Check if the request is for getting results  
 if (request.getServletPath().equals("/getResults")) {  
 // Get the voting results from the ClickerModel  
 Map<String, Integer> results = clickerModel.getResults();  
  
 // Set the results as an attribute for the view  
 request.setAttribute("results", results);  
  
 // Forward the request to the "result.jsp" view  
 request.getRequestDispatcher("/result.jsp").forward(request, response);  
  
 // Clear the results in the ClickerModel after displaying them  
 clickerModel.clearResults();  
 } else {  
 // If it's not a request for getting results, redirect back to the index page  
 request.getRequestDispatcher("/index.jsp").forward(request, response);  
 }  
 }  
}

### Task 3:

1. **Screen shots** of two uses of the input page (two different sets of input data) and the corresponding output pages.

**Input page 1: prompt.jsp – Clicking on the radio button of photo**

A screenshot of a computer

Description automatically generated

**Ouptut** : result.jsp- After submitting this input, we get the photo of the player and the flag of the team which he belongs to.

A person wearing a baseball cap

Description automatically generated

**Input page 2: prompt.jsp – Clicking on the radio button of data**

A screenshot of a computer

Description automatically generated

**Output:** data.jsp - After submitting this input, we get data about that particular player.

A screenshot of a computer

Description automatically generated

1. **Code snippets** from the Java code that screen scrapes, queries the API, and produces output.

General explanation about the model:

/\*  
Explanation:  
\* Author: Aditi Gupta - argupta  
\* Last Modified: September 22, 2023  
BaseballModel is a class that encapsulates the business logic for the web application.  
It provides methods to search for baseball player image URLs, scrape player data, and search for baseball team flag image URLs.  
The doSearch method searches for a baseball player's image URL by screen scraping a website.  
The scrapePlayerData method scrapes additional player data using the player's first and last name.  
The doFlagSearch method searches for a baseball team's flag image URL by screen scraping another website.  
The fetch method is a private utility method used to make HTTP GET requests and retrieve the response as a string.  
The class uses external libraries such as Jsoup for HTML parsing and Gson for JSON parsing.  
Data is extracted from web pages using string manipulation and HTML structure analysis.  
The results are stored in a Map called playerDataMap, which can be accessed by the servlet for further processing.  
 \*/

**Screen Scraping code for the player photo:**

public class BaseballModel {  
  
 Map<String, String> playerDataMap = new HashMap<>();  
  
 // This method performs a search for a baseball player's image URL  
 public String doSearch(String firstName, String lastName) throws UnsupportedEncodingException {  
 // URL encode the searchTag, e.g., to encode spaces as %20  
 firstName = URLEncoder.*encode*(firstName, "UTF-8");  
 lastName = URLEncoder.*encode*(lastName, "UTF-8");  
  
 String response = "";  
 String baseballReferenceURL = "https://www.baseball-reference.com/players/";  
  
 // Create a URL for the page to be screen scraped  
 if(firstName != null && lastName != null) {  
 if(lastName.length() >= 5) {  
 baseballReferenceURL = baseballReferenceURL + lastName.substring(0, 1).toLowerCase() + "/" + lastName.substring(0, 5).toLowerCase() + firstName.substring(0, 2).toLowerCase() + "01.shtml";  
 } else {  
 baseballReferenceURL = baseballReferenceURL + lastName.substring(0, 1).toLowerCase() + "/" + lastName.toLowerCase() + firstName.substring(0, 2).toLowerCase() + "01.shtml";  
 }  
 } else {  
 baseballReferenceURL = null;  
 }  
  
 // Fetch the page  
 response = fetch(baseballReferenceURL);  
  
 // Search the page to find the picture URL  
 int cutLeft = response.indexOf("players open");  
 cutLeft = response.indexOf("media-item multiple", cutLeft);  
 String s = " src=";  
 cutLeft = response.indexOf(s, cutLeft) + s.length() + 1;  
  
 // If not found, then no such photo is available.  
 if (cutLeft == -1) {  
 System.*out*.println("pictureURL= null");  
 return null;  
 }  
  
 // Look for the jpg extension  
 int cutRight = response.indexOf("jpg", cutLeft) + 3;  
  
 // Now snip out the part from positions cutLeft to cutRight  
 // and prepend the protocol (i.e., https).  
 String pictureURL;  
 if (cutRight > cutLeft) {  
 pictureURL = response.substring(cutLeft, cutRight);  
 } else {  
 pictureURL = null;  
 }  
 return pictureURL;  
 }

**Code to query the API-** The name entered by the user is used to get the MLB ID for that player. Then the ID is scarped for that player from: <https://razzball.com/mlbamids/>. The ID scarped is used to form the API, which in turn is used to get the player data.

// This method scrapes player data using the first and last name  
public Map<String, String> scrapePlayerData(String firstName, String lastName) throws UnsupportedEncodingException {  
 firstName = URLEncoder.*encode*(firstName, "UTF-8");  
 lastName = URLEncoder.*encode*(lastName, "UTF-8");  
 String playerName = firstName + " " + lastName;  
 String response = "";  
  
 // Send an HTTP GET request to the website  
 String html = fetch("https://razzball.com/mlbamids/");  
 // Fetch the page  
  
 // Search the page to find the picture URL  
 int cutLeft = html.indexOf(firstName+" "+lastName);  
 cutLeft = html.indexOf("</a>", cutLeft);  
 cutLeft = html.indexOf("</td>", cutLeft);  
 String s = "<td>";  
 cutLeft = html.indexOf(s, cutLeft) + s.length();  
  
 // If not found, then no such photo is available.  
 String result;  
 if (cutLeft == -1) {  
 System.*out*.println("Element not found");  
 return null;}  
  
 // Look for the </td>  
 int cutRight = html.indexOf("</td>", cutLeft);  
  
  
 // Now snip out the part from positions cutLeft to cutRight  
 if (cutRight > cutLeft) {  
 result = html.substring(cutLeft, cutRight);  
 } else {  
 result = null;  
 }  
  
 if (result != null) {  
 // Extract player data using an API  
 try {  
 // Define the URL of the API  
 String apiUrl = "https://lookup-service-prod.mlb.com/json/named.player\_info.bam?sport\_code=%27mlb%27&player\_id=%27" + result + "%27";  
 System.*out*.println(apiUrl);  
 String api = "";  
 api = fetch(apiUrl);  
  
 // Parse the JSON response using Gson  
 JsonParser jsonParser = new JsonParser();  
 JsonObject jsonData = jsonParser.parse(api.toString()).getAsJsonObject();  
 JsonObject playerInfo = jsonData.getAsJsonObject("player\_info")  
 .getAsJsonObject("queryResults")  
 .getAsJsonObject("row");  
  
 // Extract specific fields  
 String fullName = playerInfo.get("name\_display\_first\_last").getAsString();  
 String teamName = playerInfo.get("team\_name").getAsString();  
 String primaryPosition = playerInfo.get("primary\_position\_txt").getAsString();  
 String jerseyNumber = playerInfo.get("jersey\_number").getAsString();  
 String noOfThrows = playerInfo.get("throws").getAsString();  
 String playerId = playerInfo.get("player\_id").getAsString();  
 String activeStatus = playerInfo.get("active\_sw").getAsString();  
 String birthCountry = playerInfo.get("birth\_country").getAsString();  
  
 playerDataMap.put("Full Name", fullName);  
 playerDataMap.put("Team Name", teamName);  
 playerDataMap.put("Primary Position", primaryPosition);  
 playerDataMap.put("Jersey Number", jerseyNumber);  
 playerDataMap.put("Throws", noOfThrows);  
 playerDataMap.put("Player Id", playerId);  
 playerDataMap.put("Active Status", activeStatus);  
 playerDataMap.put("Birth Country", birthCountry);  
  
 return playerDataMap;  
 } catch (JsonSyntaxException e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("Player not found or element not found");  
 }  
 return playerDataMap;  
}

**Screen Scraping code for the flag of that player’s team:**

The data scraped from the API is used to get the team’s name. After getting the team name, a new URL is formed using this team name. This URL is used to screen scrape the flag of that particular player’s team.

// This method performs a search for a baseball team's flag image URL  
public String doFlagSearch(String firstName, String lastName) throws UnsupportedEncodingException {  
 firstName = URLEncoder.*encode*(firstName, "UTF-8");  
 lastName = URLEncoder.*encode*(lastName, "UTF-8");  
 Map<String, String> map1 = scrapePlayerData(firstName, lastName);  
 String teamName = map1.get("Team Name");  
 String response = "";  
 String baseballFlagURL = "https://www.flags.com/";  
  
 // Create a URL for the page to be screen scraped  
 if(teamName != null){  
 baseballFlagURL = baseballFlagURL + teamName.toLowerCase().replace(" ", "-") + "-flag/";  
 } else {  
 baseballFlagURL = null;  
 }  
  
 // Fetch the page  
 response = fetch(baseballFlagURL);  
  
 // Search the page to find the picture URL  
 int cutLeft = response.indexOf("productView-imageCarousel-main-item slick-current");  
 String s = " href=";  
 cutLeft = response.indexOf(s, cutLeft) + s.length() + 1;  
  
 // If not found, then no such photo is available.  
 if (cutLeft == -1) {  
 System.*out*.println("pictureURL= null");  
 return null;  
 }  
  
 // Look for the jpg extension  
 int cutRight = response.indexOf("jpg", cutLeft) + 3;  
  
 // Now snip out the part from positions cutLeft to cutRight  
 // and prepend the protocol (i.e., https).  
 String imageURL;  
 if (cutRight > cutLeft) {  
 imageURL = response.substring(cutLeft, cutRight);  
 } else {  
 imageURL = null;  
 }  
 return imageURL;  
}

**Output code: Baseball Servlet.java**

/\*  
Explanation:  
\* Author: Aditi Gupta - argupta  
\* Last Modified: September 22, 2023  
This code defines a Servlet (BaseballServlet) for handling HTTP GET requests.  
It uses the BaseballModel class to interact with the business logic.  
The init method initializes the BaseballModel when the servlet is created.  
In the doGet method, the servlet processes GET requests from clients.  
It checks if the user is on a mobile device and sets the appropriate DOCTYPE.  
It retrieves the first name and last name parameters from the request.  
Depending on the action parameter, it performs searches for image URLs and player data using the BaseballModel.  
It sets attributes on the request object to pass data to the view.  
Finally, it forwards the request to the appropriate view based on the nextView variable.  
\*/  
  
@WebServlet(name = "BaseballServlet", urlPatterns = {"/getResult", "/getData", "/processData"})  
public class BaseballServlet extends HttpServlet {  
  
 BaseballModel bbm = null; // The "business model" for this app  
  
 @Override  
 public void init() {  
 bbm = new BaseballModel();  
 }  
  
 @Override  
 protected void doGet(HttpServletRequest request,  
 HttpServletResponse response)  
 throws ServletException, IOException {  
  
 // Get the first name and last name parameters if they exist  
 String firstName = request.getParameter("firstName");  
 String lastName = request.getParameter("lastName");  
  
 // Determine what type of device our user is  
 String ua = request.getHeader("User-Agent");  
  
 boolean mobile;  
 // Prepare the appropriate DOCTYPE for the view pages based on user agent  
 if (ua != null && ((ua.indexOf("Android") != -1) || (ua.indexOf("iPhone") != -1))) {  
 mobile = true;  
 /\*  
 \* This is the latest XHTML Mobile doctype. To see the difference it  
 \* makes, comment it out so that a default desktop doctype is used  
 \* and view on an Android or iPhone.  
 \*/  
 request.setAttribute("doctype", "<!DOCTYPE html PUBLIC \"-//WAPFORUM//DTD XHTML Mobile 1.2//EN\" \"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd\">");  
 } else {  
 mobile = false;  
 request.setAttribute("doctype", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org/TR/html4/loose.dtd\">");  
 }  
  
 String nextView = null;  
 String action = request.getParameter("action");  
  
 /\*  
 \* Check if the first name and last name parameters are present.  
 \* If not, then give the user instructions and prompt for input.  
 \* If they are present, then do the search and return the result.  
 \*/  
 if (firstName != null && lastName != null && firstName.trim().length() > 0 && lastName.trim().length() > 0) {  
 if (action != null) {  
 if (action.equals("getResult") || action.equals("getData")) {  
 // Perform a search for image URLs and other data  
 String pictureURL = bbm.doSearch(firstName, lastName);  
 String imageURL = bbm.doFlagSearch(firstName, lastName);  
 request.setAttribute("imageURL", imageURL);  
 request.setAttribute("pictureURL", pictureURL);  
 // Pass the user inputs (firstName and lastName) also to the view.  
 request.setAttribute("firstName", firstName);  
 request.setAttribute("lastName", lastName);  
  
 if (action.equals("getData")) {  
 // Scrape additional player data  
 bbm.scrapePlayerData(firstName, lastName);  
 nextView = "data.jsp";  
 } else {  
 nextView = "result.jsp";  
 }  
 } else {  
 nextView = "prompt.jsp";  
 }  
 }  
 } else {  
 // No first name and last name parameters, so choose the prompt view  
 nextView = "prompt.jsp";  
 }  
 // Transfer control over to the correct "view"  
 RequestDispatcher view = request.getRequestDispatcher(nextView);  
 view.forward(request, response);  
 }  
}

**The three codes for giving the output are as follow:**

1. **prompt.jsp:** Asking for user input

<%@**page** contentType="text/html" pageEncoding="UTF-8"%>  
  
<html>  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>JSP Page</title>  
</head>  
<body>  
<p>Give me a first name and last name of a baseball player.</p>  
<form action="processData" method="GET">  
 <label for="firstName">First Name:</label>  
 <input type="text" name="firstName" value="" /><br>  
  
 <label for="lastName">Last Name:</label>  
 <input type="text" name="lastName" value="" /><br>  
  
 <label>Select an Action:</label><br>  
 <input type="radio" name="action" value="getResult" checked>Photo<br>  
 <input type="radio" name="action" value="getData">Data<br>  
  
 <input type="submit" value="Submit" />  
</form>  
</body>  
</html>

1. **result.jsp :** Used to display the player’s photo and his team’s flag photo

<%@ **page** import="java.util.Map" %>  
<%@ **page** import="com.example.project1task3.BaseballModel" %>  
<%@ **page** import="com.example.project1task3.BaseballServlet" %>  
<%@**page** contentType="text/html" pageEncoding="UTF-8"%>  
<%= request.getAttribute("doctype") %>  
  
<html>  
<head>  
 <title>Interesting Picture</title>  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
</head>  
<body>  
<%  
 String firstName = request.getParameter("firstName");  
 String lastName = request.getParameter("lastName");  
  
 if (request.getAttribute("pictureURL") != null) {  
%>  
<h1>Here is a picture of <%= firstName %> <%= lastName %> </h1><br>  
<img src="<%= request.getAttribute("pictureURL")%>"><br><br>  
<h1>Here is a picture of flag of his team</h1><br>  
<img src="<%= request.getAttribute("imageURL")%>" style="height: 200px; width: 200px;"><br><br>  
<% } else { %>  
<h1>A picture of <%= firstName %> <%= lastName %> could not be found</h1><br>  
<% } %>  
  
<form action="processData" method="GET">  
 <label for="firstName">First Name:</label>  
 <input type="text" name="firstName" value="" /><br>  
  
 <label for="lastName">Last Name:</label>  
 <input type="text" name="lastName" value="" /><br>  
  
 <label>Select an Action:</label><br>  
 <input type="radio" name="action" value="getResult" checked>Photo<br>  
 <input type="radio" name="action" value="getData">Data<br>  
  
 <input type="submit" value="Submit" />  
</form>  
</body>  
</html>

1. **data.jsp :** Displays the result of quering the API , that is, the player’s data

<%@ **page** import="java.util.Map" %>  
<%@ **page** import="com.example.project1task3.BaseballModel" %>  
<%@ **page** import="com.example.project1task3.BaseballServlet" %>  
<%@**page** contentType="text/html" pageEncoding="UTF-8"%>  
<%= request.getAttribute("doctype") %>  
  
<html>  
<head>  
 <title>Interesting Picture</title>  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
</head>  
<body>  
<%  
 BaseballModel bbm = new BaseballModel();  
%><%  
 java.util.Map<String, String> playerDataMap = bbm.scrapePlayerData(request.getParameter("firstName"), request.getParameter("lastName"));  
 if (playerDataMap != null && !playerDataMap.isEmpty()) {  
%>  
<h2>Player Data:</h2>  
<ul>  
 <% for (java.util.Map.Entry<String, String> entry : playerDataMap.entrySet()) { %>  
 <li><strong><%= entry.getKey() %>:</strong> <%= entry.getValue() %></li>  
 <% } %>  
</ul>  
<% } %>  
  
<form action="processData" method="GET">  
 <label for="firstName">First Name:</label>  
 <input type="text" name="firstName" value="" /><br>  
  
 <label for="lastName">Last Name:</label>  
 <input type="text" name="lastName" value="" /><br>  
  
 <label>Select an Action:</label><br>  
 <input type="radio" name="action" value="getResult" checked>Photo<br>  
 <input type="radio" name="action" value="getData">Data<br>  
  
 <input type="submit" value="Submit" />  
</form>  
</body>  
</html>