Author: Shivam Patel

Andrew ID: shpatel

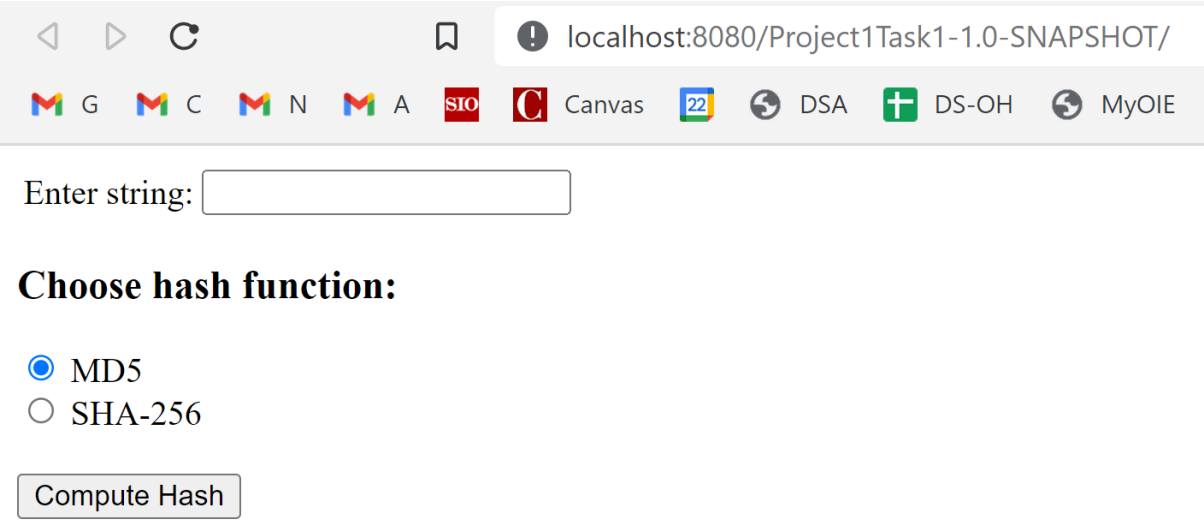
Last Modified: September 23, 2022

Project 1

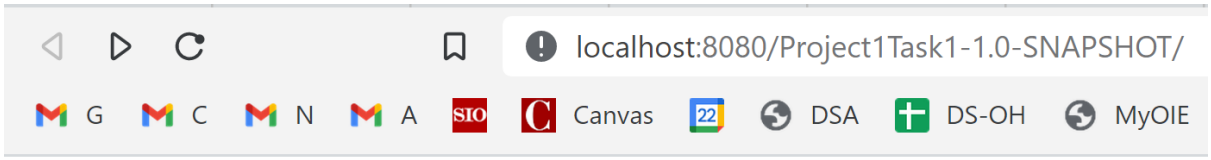
Project 1 - Task 1

Screenshots:

Initial Input Page



MD5 Output

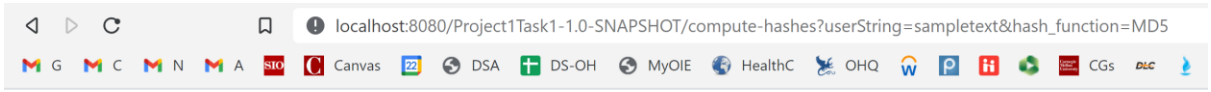


Enter string: sampletext

Choose hash function:

- (•) MD5
- ( ) SHA-256

Compute Hash



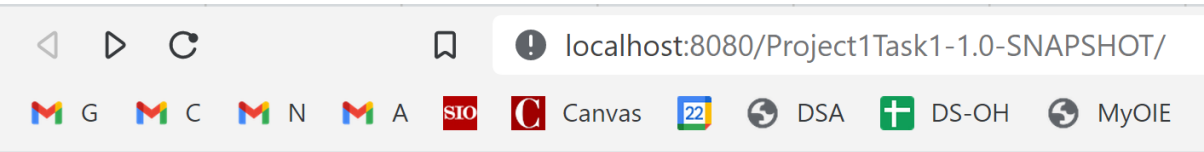
Original Text: sampletext

Hash Function: MD5

Hexadecimal Hash Value: 9BA4A43CE8C7C6A4DE68B58F9A444A49

Base 64 Hash Value: m6SkPOjHxqTeaLWPmkRKSQ==

SHA256 output

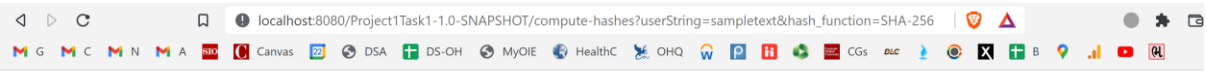


Enter string: sampletext

Choose hash function:

- ○ MD5
- ◉ SHA-256

Compute Hash



Original Text: sampletext

Hash Function: SHA-256

Hexadecimal Hash Value: A5871E22AE2DC0DCFFDAEBCFFA9D12AB2278E22FEAA5CBE2891EBA56D52678F5

Base 64 Hash Value: pYceIq4twNz/2uvP+p0SqyJ44i/qpcviiR66VtUmePU=

Code Snippets:

Computation of MD5

```
/***
 * computes the MD5 hash value of the string passed
 * into the function, converts the hash into hexadecimal and
 * base64 notations and return these two notations.
 * @param user_input String input from the user whose hash values are to be
computed
 * @return A String array of two values. First is the hexadecimal notion
 *         of the MD5 hash of user_input and the second is the base64
notion
 *         of the MD5 hash of user_input
 */
public String[] computeMD5(String user_input) {

    // String array to store the hexadecimal and base64 notion of MD5 hash
of user_input
    String[] hashes = new String[2];

    // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
    try {
        // Access MessageDigest class for MD5
        MessageDigest md = MessageDigest.getInstance("MD5");

        // Compute the digest
        md.update(user_input.getBytes());

        // Store digest as a byte array for further use
        byte[] digest = md.digest();

        // Compute hexadecimal notion of MD5 hash value of user_input
        String hex_hash =
javax.xml.bind.DatatypeConverter.printHexBinary(digest);

        // Compute base64 notion of MD5 hash value of user_input
        String b64_hash =
javax.xml.bind.DatatypeConverter.printBase64Binary(digest);

        // Add the hexadecimal and base64 notions to hashes String array
        hashes[0] = hex_hash;
        hashes[1] = b64_hash;
    }
    // Handles No MD5 Algorithm exceptions
    catch(NoSuchAlgorithmException e) {
        // Print error message in console
        System.out.println("No MD5 available" + e);
    }
    // Return the hexadecimal and base64 notions of MD5 hash of user_input
    return hashes;
}
```

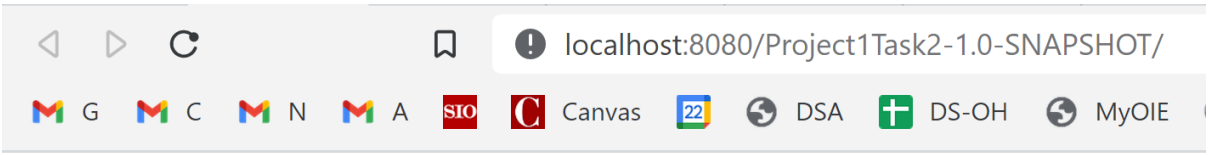
Computation of SHA256

```java
/***
 * computes the SHA256 hash value of the string passed
 * into the function, converts the hash into hexadecimal and
 * base64 notations and return these two notations.
 * @param user_input String input from the user whose hash values are to be
computed
 * @return A String array of two values. First is the hexadecimal notion
 *         of the SHA256 hash of user_input and the second is the base64
notion
 *         of the SHA256 hash of user_input
 */
public String[] computeSHA256(String user_input) {

    // String array to store the hexadecimal and base64 notion of SHA256
hash of user_input
    String[] hashes = new String[2];

    // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
    try {
        // Access MessageDigest class for SHA256
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        // Compute the digest
        md.update(user_input.getBytes());

        // Store digest as a byte array for further use
        byte[] digest = md.digest();

        // Compute hexadecimal notion of MD5 hash value of user_input
        String hex_hash =
javax.xml.bind.DatatypeConverter.printHexBinary(digest);

        // Compute base64 notion of MD5 hash value of user_input
        String b64_hash =
javax.xml.bind.DatatypeConverter.printBase64Binary(digest);

        // Add the hexadecimal and base64 notions to hashes String array
        hashes[0] = hex_hash;
        hashes[1] = b64_hash;
    }
    // Handles No SHA-256 Algorithm exceptions
    catch(NoSuchAlgorithmException e) {
        // Print error message in console
        System.out.println("No SHA-256 available" + e);
    }
    // Return the hexadecimal and base64 notions of SHA256 hash of
user_input
    return hashes;
}
```

Project 1 - Task 2

Screenshots:

Input Page



State Information

Created by Shivam Patel

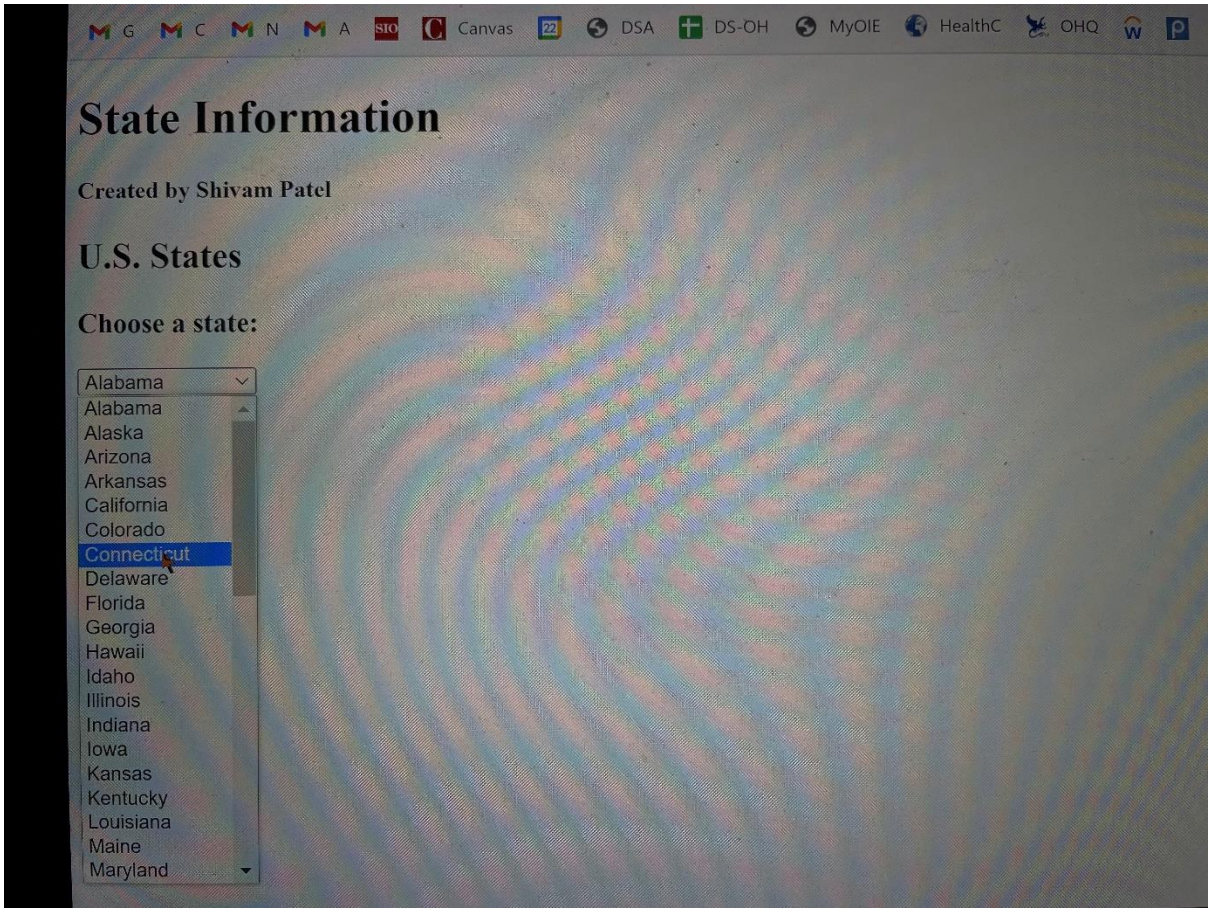
U.S. States

Choose a state:

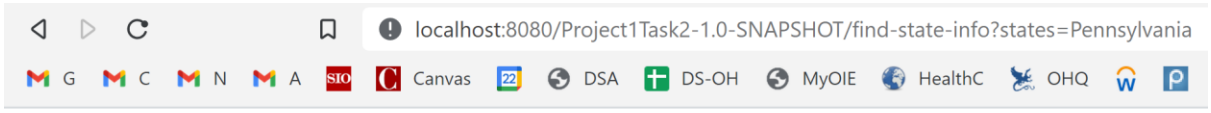
Alabama

Submit

Drop Down Menu (Could not take a screenshot of the drop down from my PC, hence took a photo from my phone and added it here. The drop down used to disappear as soon as I tried to take a screenshot in the laptop, hence I used the phone)



Pennsylvania Output



State: Pennsylvania

Population: 13002700

Nickname: Keystone State

Capital: Harrisburg

Song: Pennsylvania

Flower: Mountain Laurel

Credit: https://statesymbolsusa.org/categories/flower

Flag:

Credit: https://www.states101.com/flags

Continue

New York Output

State: New York

Population: 20201249

Nickname: Empire State

Capital: Albany

Song: I love New York

Flower: Rose

Credit: https://statesymbolsusa.org/categories/flower

Flag:

Credit: https://www.states101.com/flags

Continue

Code Snippets:

Scrapping of nickname

```java
/***
 * Function to scrape the nickname of the US State as selected by the user
 * @param user_state The name of the US State whose nickname is to be found
 * @return Scrapped nickname of the US State
 */
public String getStateNickName(String user_state) {

    // Stores the result of the Jsoup scrapped URL for US States nicknames
    // URL for scrapping nickname: https://www.britannica.com/topic/List-of-nicknames-of-U-S-States-2130544
    Document doc_nicknames;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_nicknames = Jsoup.connect("https://www.britannica.com/topic/List-of-nicknames-of-U-S-States-2130544").get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential elements that contain
    // nickname of the US State
    Elements nicknames = doc_nicknames.select("tbody").select("tr").select("td");

    // Stores the scrapped nickname of the US State
    String state_nickname = "";

    // For every element in nicknames
    for(int i = 0; i < nicknames.size(); i++) {
        // If the desired US State is found in the element
        if (nicknames.get(i).text().equals(user_state)) {
            // Get the nickname of the US state
            state_nickname = nicknames.get(i + 1).text();
        }
    }

    // Return the scrapped nickname of the US State
    return state_nickname;
}
```

Scrapping of capital

```java
/***
 * Function to scrape the capital of the US State as selected by the user
 * @param user_state The name of the US State whose capital is to be found
 * @return Scrapped capital of the US State
 */
public String getStateCapital(String user_state) {

    // Stores the result of the Jsoup scrapped URL for US States capitals
    // URL for scrapping capitals: https://gisgeography.com/united-states-map-with-capitals/
    Document doc_capitals;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_capitals = Jsoup.connect("https://gisgeography.com/united-states-map-with-capitals/").get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential elements that contain
    // capitals of the US State
    Elements capitals = doc_capitals.getElementsByClass("kt-inside-inner-col").select("p");

    // Stores the scrapped capital of the US State
    String state_capital = "";

    // For every element in capitals
    for(int i = 0; i < capitals.size(); i++) {
        // Create an array of string split leading to name of capital
        String[] p_split = capitals.get(i).text().split("\\)");
        // If length of string in the split if > 5 (to avoid unnecessary elements further)
        if (p_split.length > 5) {
            // For every string split
            for (int j = 0; j < p_split.length; j++) {
                // If the desired US State is found in the element
                if (p_split[j].split("\\(")[0].contains(user_state)) {
                    // Get the capital of the US state
                    state_capital = p_split[j].split("\\(")[1];
                }
            }
        }
    }
    // Return the scrapped capital of the US State
    return state_capital;
}
```

Scrapping of state song

```java
/***
 * Function to scrape the song of the US State as selected by the user
 * @param user_state The name of the US State whose song is to be found
 * @return Scrapped song of the US State
 */
public String getStateSong(String user_state) {

    // Stores the result of the Jsoup scrapped URL for US States songs
    // URL for scrapping songs: https://www.50states.com/songs/
    Document doc_songs;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_songs = Jsoup.connect("https://www.50states.com/songs/").get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its
details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential
elements that contain
    // song of the US State
    Elements songs = doc_songs.select("dl");

    // Stores the scrapped song of the US State
    String state_song = "";

    // For every element in songs
    for(int i = 0; i < songs.size(); i++) {
        // If the desired US State is found in the element
        if (songs.get(i).select("dt").text().equals(user_state)) {
            // Create a new set of elements to scrape multiple songs for
the US which has more than one song
            Elements songs_dds = songs.get(i).select("dd");

            // For every song element
            for(int j = 0; j < songs_dds.size(); j++) {
                // Scrape the first song and come out of the loop, since we
need only one song
                state_song = songs_dds.get(j).text();
                break;
            }
        }
    }
    // Return the scrapped song of the US State
    return state_song;
}
```

Scrapping of flower URL (have also scrapped the name of the flower as an additional task)

```java
/***
 * Function to scrape the flower name and flower image link of the US State as selected by the user
 * @param user_state The name of the US State whose flower name and flower image link is to be found
 * @return String array with two strings flower name and flower image link of the US State
 */
public String[] getStateFlower(String user_state) {

    // Stores the result of the Jsoup scrapped URL for US States flowers
    // URL for scrapping flowers: https://statesymbolsusa.org/categories/flower
    Document doc_flowers;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_flowers = Jsoup.connect("https://statesymbolsusa.org/categories/flower").get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential elements that contain
    // flowers of the US State
    Elements flowers = doc_flowers.select(".view-symbols > .view-content ul li");

    // Stores the flower name and flower image of the US State
    String[] state_flower_info = new String[2];

    // Stores the flower name the US State
    String state_flower_name = null;

    // Stores the flower image link of the US State
    String state_flower_image = null;

    // For every element in flowers
    for(int i = 0; i < flowers.size(); i++) {
        // Find and select a subset of elements with ccs query div
        Elements flowers_divs = flowers.get(i).select("div");
        // If the desired US State is found in the element
        if (flowers_divs.get(2).text().equals(user_state)) {
            // Get the first flower image link of the US state
            state_flower_image = flowers_divs.get(1).select("img").attr("src");
            // Get the first flower name of the US state
            state_flower_name = flowers_divs.get(3).text();

            // Break since we need details of only one flower
            break;
        }
    }

    // Add flower name and flower image link in state_flower_info
    state_flower_info[0] = state_flower_name;
    state_flower_info[1] = state_flower_image;
```

```
    // Return the scrapped flower information of the desired US state
    return state_flower_info;
}
```

Scrapping of Flag URL

```
/***
 * Function to scrape the flag image link of the US State as selected by
the user
 * @param user_state The name of the US State whose flag image link is to
be found
 * @return Scrapped flag image link of the US State
 */
public String getStateFlag(String user_state) {

    // Stores the result of the Jsoup scrapped URL for US States flags
    // URL for scrapping flags: https://www.states101.com/flags
    Document doc_flags;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_flags = Jsoup.connect("https://www.states101.com/flags").get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its
details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential
elements that contain
    // flags of the US State
    Elements flags = doc_flags.select("div.content > div.row > .col-md-
3.col-sm-4.col-xs-6");

    // Stores the scrapped flag image link of the US State
    String state_flag_image = "";

    // For every element in flags
    for(int i = 0; i < flags.size(); i++) {
        // If the desired US State is found in the element
        if (flags.get(i).select("a").text().equals(user_state + " ")) {
            // Get and form the complete image link of the flag of the US
state
            state_flag_image = "https://www.states101.com" +
flags.get(i).select("p > a > img").attr("src");
        }
    }

    // Return the scrapped flag image link of the US State
    return state_flag_image;
}
```

Api call for the population

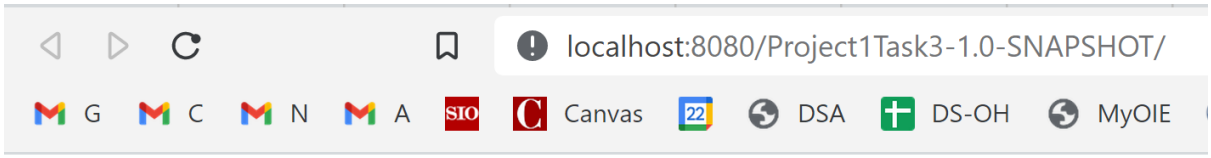
```
/***
 * Function to scrape the population of the US State as selected by the
user
 * @param user_state The name of the US State whose population is to be
found
 * @return Scrapped population of the US State
 */
public String getStatePopulation(String user_state) {

    // Stores the scrapped population of the US State
    String state_population = "";

    // Source: https://stackoverflow.com/questions/43698645/read-local-
file-in-intellij-idea-javaee-web-application-deployed-with-tomcat
    // Create a URI object to read the CSV file containing FIPS values of
the states
    URI uri;
    try {
        // Read the CSV file
        uri =
Objects.requireNonNull(StateInformationServlet.class.getClassLoader().getRe
source("fips.csv")).toURI();
    }
    // Handles URI syntax exceptions and throws a Runtime exception along
with its details
    catch (URISyntaxException e) {
        throw new RuntimeException(e);
    }

    // Stores the fips of all states (to be cleaned later in the code)
    String fips_values;
    try {
        // Get the FIPS values of all states in one string
        fips_values = String.valueOf(Files.readAllLines(Paths.get(uri)));
    }
    // Handles IO exceptions and throws a Runtime exception along with its
details
    catch (IOException e) {
        throw new RuntimeException(e);
    }

    // Format fips values to be cleaned further
    fips_values = fips_values.substring(1,fips_values.length() - 1);

    // Create a split based on commas
    String[] fips_split = fips_values.split(",");

    // Stores the fips of the desirved US state
    String user_state_fips = "";

    // For every string in fips split
    for (int i = 0; i < fips_split.length; i++) {
        // If the desired US state is found
        if (fips_split[i].contains(user_state)) {
            // Store the US state fips and come out of the loop
            user_state_fips = fips_split[i + 1];
            break;
        }
```

```
    }

    // Creates and stores the link to be scrapped for finding the
population of the desired US state
    // The website requires and API call. API was got from
https://api.census.gov/data/key_signup.html
    // API Key = a5c5a24598a6575a96967635c42f210918dd111f
    String population_link =
"https://api.census.gov/data/2020/dec/pl?get=NAME,P1_001N&for=state: " +
user_state_fips + "&key=a5c5a24598a6575a96967635c42f210918dd111f";

    // Stores the result of the Jsoup scrapped URL for US States population
    // URL for scrapping population: population_link (as above)
    Document doc_population;
    try {
        // Using Jsoup, connect to the URL, scrape it and store its data
        doc_population =
Jsoup.connect(population_link).ignoreContentType(true).get();
    }
    // Handles IO exceptions and throws a Runtime exception along with its
details
    catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Using Jsoup selectors, scrape the Document and store the potential
elements that contain
    // population of the US State
    Elements population = doc_population.select("body");

    // Scrape the population of the desired US state
    state_population = population.text().split(",")[4].split("\"")[1];

    // Return the scrapped population of the US State
    return state_population;
}
```

Project 1 - Task 3

Screenshots:

Input Page



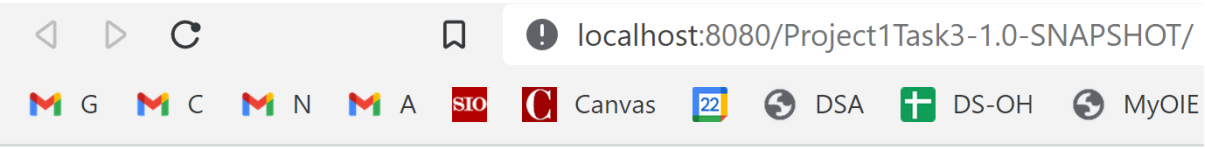
Distributed Systems Class Clicker

Submit your answer to the current question:

- A
- B
- C
- D

Submit

Output page 1 vote

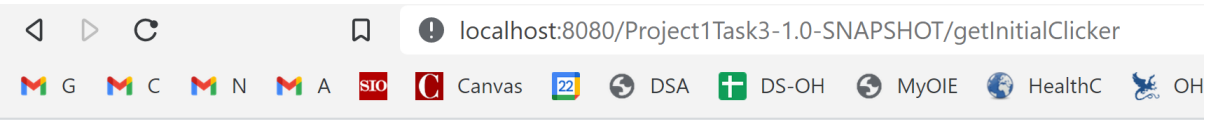


Distributed Systems Class Clicker

Submit your answer to the current question:

- (●) A
- ( ) B
- ( ) C
- ( ) D

[Submit]



Distributed Systems Class Clicker

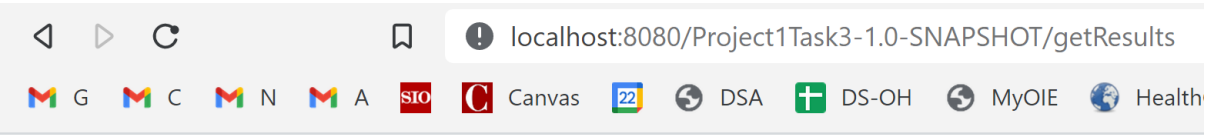
Your "A" has been registered.

Submit your answer to the current question:

- ( ) A
- ( ) B
- ( ) C
- ( ) D

[Submit]

Results Page

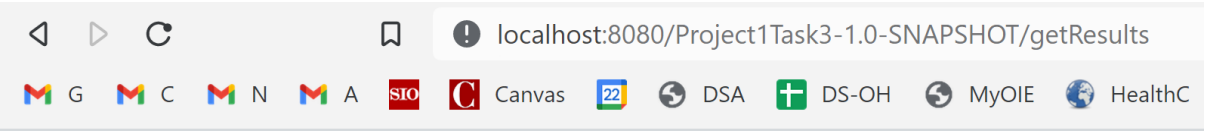


Distributed Systems Class Clicker

The results from the survey are as follows

A: 1

Results Page (again after resetting the counter above)



Distributed Systems Class Clicker

There are currently no results

Code Snippets

Output Page – Servlet (Java code)

```
/***
 * This servlet will reply to HTTP POST requests via this doPost method
 * Handles urlPatterns = {"/getInitialClicker", "/getFutureClicker"}
 * @param request Represents the request from HttpServletRequest
 * @param response Represents the response from HttpServletRequest
 * @throws IOException Exception during IO operations
 * @throws ServletException Represents servlet exceptions
 */
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {

    // Source: CMU 95702 Fall 2022 Lab2-InterestingPicture Code
    // Used for compatability of the code on different devices

    // Determine what type of device our user is
    String ua = request.getHeader("User-Agent");

    // prepare the appropriate DOCTYPE for the view pages
    if (ua != null && ((ua.indexOf("Android") != -1) ||
(ua.indexOf("iPhone") != -1))) {
        /*
         * 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 {
        request.setAttribute("doctype", "<!DOCTYPE HTML PUBLIC \"-
//W3C//DTD HTML 4.01 Transitional//EN\"
\"http://www.w3.org/TR/html4/loose.dtd\">");
    }

    // Get and store the answer selected by the user in the current round
    String answer = request.getParameter("answer");

    // Add the answer to the Sorted Map to increment its could by 1 in the
    // Sorted Map that is storing the counts of each answer
    model.incrementAnswerCount(answer);

    // Set attributes for futureClicker.jsp file
    // The attribute set is the selected answer by the user in the current
round
    request.setAttribute("selectedAnswer", answer);

    // Set the next View to be output.jsp displaying the scrapped
information
    RequestDispatcher view =
request.getRequestDispatcher("futureClicker.jsp");

    // Forwards HttpServletRequest request, HttpServletResponse response to
the View
    view.forward(request, response);
}
```

Output Page – View (JSP code)

```
<html>
<head>
  <title>Clicker</title>
</head>
<body>
<form action="getFutureClicker" method="post">
  <h1>Distributed Systems Class Clicker</h1>
  <h3> Your "<%= request.getAttribute("selectedAnswer")%>" has been
registered.</h3>
  <h3>Submit your answer to the current question:</h3>
  <input type="radio" id="a" name="answer" value="A">
  <label for="a">A</label><br>
  <input type="radio" id="b" name="answer" value="B">
  <label for="b">B</label><br>
  <input type="radio" id="c" name="answer" value="C">
  <label for="c">C</label><br>
  <input type="radio" id="d" name="answer" value="D">
  <label for="d">D</label><br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Results Page – Servlet (Java code)

```java
/***
 * This servlet will reply to HTTP GET requests via this doGet method
 * urlPatterns = {"/getResults"}
 * @param request Represents the request from HttpServletRequest
 * @param response Represents the response from HttpServletRequest
 * @throws IOException Exception during IO operations
 * @throws ServletException Represents servlet exceptions
 */
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

    // Gets and store the HTML formatted output to be presented on
results.jsp
    String results = model.getResultCount();

    // Reset the Sorted Map storing the count of answers
    // This is done because the /getResults url pattern was invoked
    model.resetResults();

    // Source: CMU 95702 Fall 2022 Lab2-InterestingPicture Code
    // Used for compatability of the code on different devices

    // Determine what type of device our user is
    String ua = request.getHeader("User-Agent");

    // Prepare the appropriate DOCTYPE for the view pages
    if (ua != null && ((ua.indexOf("Android") != -1) ||
(ua.indexOf("iPhone") != -1))) {
        /*
         * 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 {
        request.setAttribute("doctype", "<!DOCTYPE HTML PUBLIC \"-
//W3C//DTD HTML 4.01 Transitional//EN\"
\"http://www.w3.org/TR/html14/loose.dtd\">");
    }

    // Set attributes for results.jsp file
    request.setAttribute("results", results);

    // Set the next View to be output.jsp displaying the scrapped
information
    RequestDispatcher view = request.getRequestDispatcher("results.jsp");

    // Forwards HttpServletRequest request, HttpServletResponse response to
the View
    view.forward(request, response);
}
```

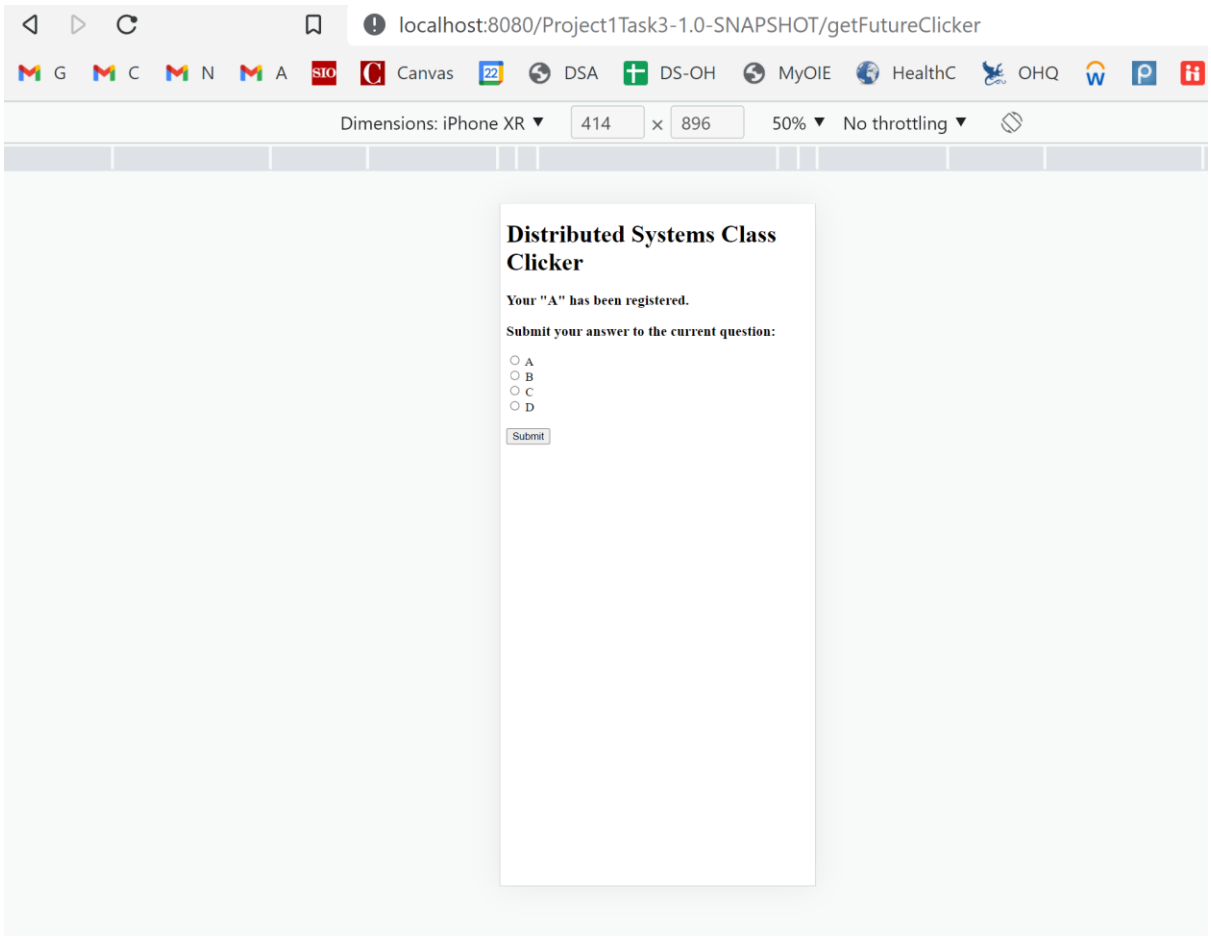
Results Page – View (JSP code)

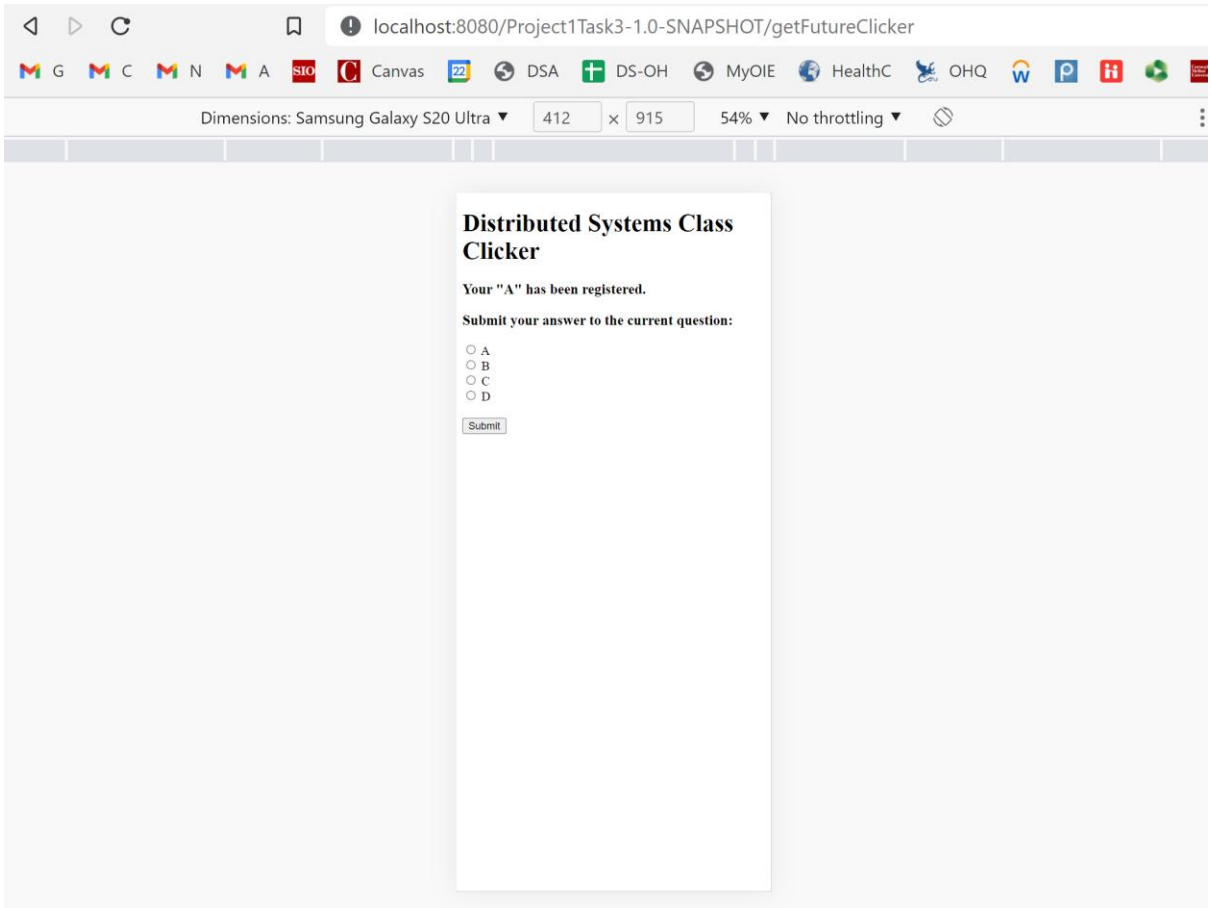
```
<html>
  <head>
    <title>Clicker</title>
  </head>
  <body>
    <p><%= request.getAttribute("results")%></p>
  </body>
</html>
```

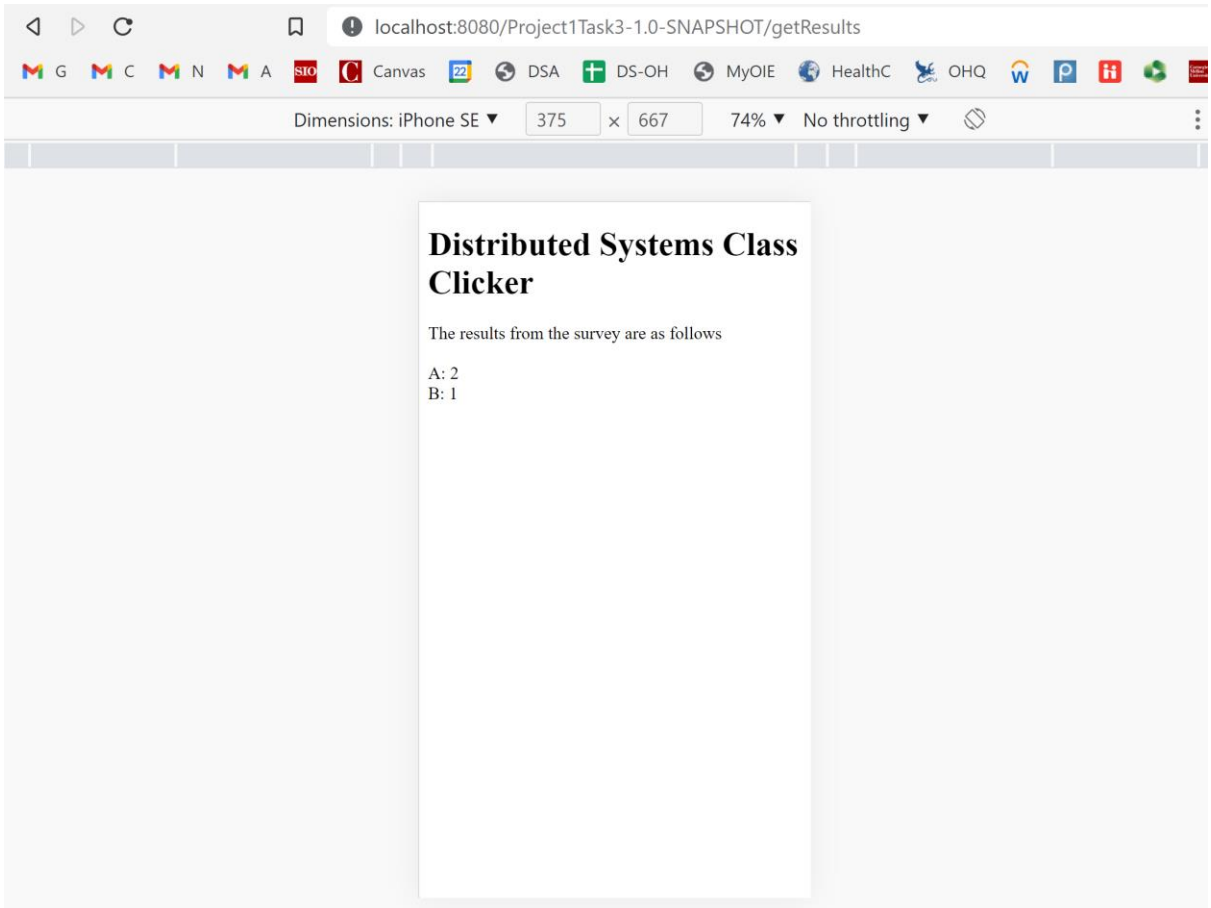
Results Page – Model

```
/***
 * It computes the result of the answers and their counts to be printed in
results.jsp
 * @return The string value containing HTML tags that would be printed on
results.jsp
 */
public String getResultCount() {

    // Stores the formatted string output (in HTML form)
    String result = "";

    // If the map is not empty
    if (answer_map.size() != 0) {

        // Start creating the result
        result = result + "<h1>Distributed Systems Class Clicker</h1>" +
                "The results from the survey are as follows <br><br>";

        // Source: https://www.geeksforgeeks.org/iterate-map-java/
        // For every key value pair that is present in the map
        for (Map.Entry<String,Integer> entry : answer_map.entrySet()) {
            // Add the key and value to the results
            result = result + entry.getKey() + ": " + entry.getValue() +
"<br>";
        }
    }
    // If the map of the count of answers is empty
    else {
        result = result + "<h1>Distributed Systems Class Clicker</h1>" +
                "There are currently no results";
    }

    // Return the formatted results
    return result;
}
```

Output – Mobile Devices




localhost:8080/Project1Task3-1.0-SNAPSHOT/getFutureClicker
Dimensions: Samsung Galaxy S20 Ultra
412
915
54%
No throttling
Distributed Systems Class Clicker
Your "A" has been registered.
Submit your answer to the current question:
A
B
C
D
Submit


localhost:8080/Project1Task3-1.0-SNAPSHOT/getResults
Dimensions: iPhone SE
375
667
74%
No throttling
Distributed Systems Class Clicker
The results from the survey are as follows
A: 2
B: 1