



University  
of Windsor

# FINAL PROJECT REPORT

Course: Advanced Computing Concepts  
(COMP8547)

**Variant: Car Rental Price Analysis**

**Team: Lancer**

## Section 1

### Team members

Timil Prajapati - 110157182

Ayush Patel - 110137866

Gravin Patel- 110156150

Smit Patel - 110156386

Sanket Kothiya -110156177

**Submitted to: Dr Olena Syrotkina**

## **Task:**

### **Car Rental Price Analysis**

The idea of the project is to analyze and understand car rental prices to get the best deal for the user:

1. Choose at least three websites to crawl (e.g. enterprise.ca, carrental.ca, expedia.com, etc.).
2. Specify the details of your search (e.g. Price, Rating, Pick-up & Return Location, dates, etc.).
3. Implement required features (see Final Project Information.pdf, Instructions).

### **Contribution of All Team Member:**

SR.	MEMBERS	STUDENT NO	TASKS
1	Timil Prajapati	110157182	Spell Checker [SpellChecker.java] Data Validation [UserManager.java] Integration with Command Line Interface
2	Ayush Patel	110137866	Search Frequency [SearchQuery.java] Web Crawler [Main.java]
3	Gravin Patel	110156150	Page Ranking [PageRanking.java] HTML Parser
4	Smit Patel	110156386	Word Frequency Count [WordFrequencyInFile.java] Finding Patterns using RegEx [UserManager.java] Rating Based Sorting Integration with Command Line Interface
5	Sanket Kothiya	110156177	Word Completion [WordCompletionTries.java] URL Extractor [URLExtractor.java]

## 1. Data Validation

### - Timil Prajapati: [UserManager.java]

The provided code snippet consists of three methods in Java for data validation. Here's an explanation of each method:

- **isEmailValid(String email):**
  - **Purpose:** Validates the format of an email address.
  - **Implementation:** Uses a regular expression to match the email format.
- **isPasswordStrong(String password):**
  - **Purpose:** Validates the strength of a password.
  - **Implementation:** Uses a regular expression to ensure the password meets certain criteria
- **isPasswordUsed(String password):**
  - **Purpose:** Checks if the password has already been used in the database.
  - **Implementation:** Executes a SQL query to count occurrences of the password in the user table.

```
public boolean isEmailValid(String email) { 2 usages new *
    // Simple validation for email format
    return email != null && email.matches( regex: "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$");
}

public boolean isPasswordStrong(String password) { 2 usages new *
    // Simple validation for password strength
    return password != null && password.matches( regex: "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[!@#$%^&*]).{8,}$");
}

public boolean isPasswordUsed(String password) { 2 usages new *
    String sql = "SELECT COUNT(*) FROM users WHERE password = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString( parameterIndex: 1, password);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt( columnIndex: 1) > 0;
        }
    } catch (SQLException e) {
        System.err.println("SQL error: " + e.getMessage());
    }
    return false;
}
```

- The isUsernameExists method checks if a given username already exists in the users table.
- It uses a PreparedStatement to safely execute a SQL query that counts the number of rows with the specified username.
- If the count is greater than 0, it returns true; otherwise, it returns false.
- Proper exception handling ensures any SQL errors are caught and reported.

```

public boolean isUsernameExists(String username) { 1 usage new *
    String sql = "SELECT COUNT(*) FROM users WHERE username = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString( parameterIndex: 1, username);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt( columnIndex: 1) > 0;
        }
    } catch (SQLException e) {
        System.err.println("SQL error: " + e.getMessage());
    }
    return false;
}

```

[Fig-1.1: Source Code for Data Validation]

```

*****
*                                     *
*      Welcome to Lancer Cars!      *
*                                     *
*****
*                                     *
*   Your trusted car rental service  *
*                                     *
*****

Enter 'login' to log in, 'signup' to sign up, or 'exit' to quit: signup
Enter username (email): test1232uwindsor.ca
The username must be a valid email address. Please try again.
Enter username (email): test123@uwindsor.ca
Enter password: Uwindsor@12343
Signup successful.
Enter 'login' to log in, 'signup' to sign up, or 'exit' to quit: login
Enter username (email): test123@uwindsor.ca
Enter password: Uwindsor@12343
Login successful.
Do you want to perform the web scraping? (yes/no): no
Scraping Skipped
Search a car (or write exit):

```

[Fig-1.2: Output for Data Validation]

## 2. Pattern Finder using RegEx:

- Smit Patel: [UserManager.java]

**Method:** isValidEmail

**Purpose:** To validate if the provided email string follows a standard email format.

**Regex:** `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$`

- `^`: Asserts the position at the start of the string.
- `[a-zA-Z0-9._%+-]+`: Matches one or more characters that can be a letter (uppercase or lowercase), digit, dot, underscore, percent, plus, or hyphen.
- `@`: Matches the @ symbol.
- `[a-zA-Z0-9.-]+`: Matches one or more characters that can be a letter (uppercase or lowercase), digit, dot, or hyphen.
- `\.`: Matches the dot (.) character. The backslash is used to escape the dot, which is a special character in regex.
- `[a-zA-Z]{2,6}`: Matches between 2 to 6 letters (uppercase or lowercase), representing the domain suffix (e.g., .com, .net).
- `$`: Asserts the position at the end of the string.

**Condition:** `email != null && email.matches(regex)`

- Ensures the email is not null.
- Uses matches method to check if the email matches the specified regex pattern.

**Method:** isPasswordStrong

**Purpose:** To validate if the provided password string meets the criteria for a strong password.

**Regex:** `^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*]).{8,}$`

- `^`: Asserts the position at the start of the string.
- `(?=.*[a-z])`: Positive lookahead to ensure at least one lowercase letter.
- `(?=.*[A-Z])`: Positive lookahead to ensure at least one uppercase letter.

- `(?=.*\\d)`: Positive lookahead to ensure at least one digit.
- `(?=.*[!@#$$%^&*])`: Positive lookahead to ensure at least one special character from the set `!@#$$%^&*`.
- `.{8,}`: Ensures the password is at least 8 characters long (matches any character except a newline, at least 8 times).
- `$`: Asserts the position at the end of the string.

**Condition:** `password != null && password.matches(regex)`

- Ensures the password is not null.
- Uses `matches` method to check if the password matches the specified regex pattern.

```
public boolean isEmailValid(String email) { 2 usages new *
    // Simple validation for email format
    return email != null && email.matches(regex: "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$");
}

public boolean isPasswordStrong(String password) { 2 usages new *
    // Simple validation for password strength
    return password != null && password.matches(regex: "(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[!@#$$%^&*]).{8,}$");
}
```

[Fig-2.1: Source Code for Pattern Finder using RegEx]

```
Enter 'login' to log in, 'signup' to sign up, or 'exit' to quit: signup
Enter username (email): timil@panta.com
Enter password: panta2
Your password is not strong enough. Please follow these guidelines:
1. At least 8 characters long
2. Includes both upper and lower case letters
3. Includes at least one digit
4. Includes at least one special character (e.g., !@#$$%^&*)
Enter password: Panta@1234321
Signup successful.
Enter 'login' to log in, 'signup' to sign up, or 'exit' to quit:
```

[Fig-2.2: Output for Pattern Finder using RegEx]



### 3. Web Crawler and HTML Parser:

- Ayush Patel, Gravin Patel: [Main.java]

HTML parsing and web scraping are crucial components of the car rental website, enabling the extraction and integration of external data to enhance the website's functionality. These techniques allow the site to gather real-time information on rental vehicles, pricing, and availability from various sources, providing users with accurate and up-to-date information.

#### HTML Parsing:

HTML parsing is used to read and extract structured data from HTML documents. In the context of the car rental website, this involves retrieving information from web pages of partner rental agencies or third-party services. The HTML parser traverses the HTML document, selecting elements based on tags, classes, or IDs. It extracts relevant data such as vehicle names, prices, images, and availability information.

```
Href: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpIn=4675417&locn=Winnipeg+%2F
Title: Toyota Corolla
Image URL: https://mediaim.expedia.com/cars/19/92c43819-9328-400a-8516-6b16b7971736.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Total price with Tax: $66
Rating: 2.90
Car Page Direction Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpIn=4675417&locn=Winnipeg+%2F
-----
Href: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpIn=4675417&locn=Winnipeg+%2F
Title: Toyota Camry
Image URL: https://mediaim.expedia.com/cars/19/8a46595b-5936-44d8-9e79-b3d28012405e.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Total price with Tax: $69
Rating: 2.90
Car Page Direction Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpIn=4675417&locn=Winnipeg+%2F
-----
Href: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpIn=4675417&locn=Winnipeg+%2F
Title: Toyota Yaris
Image URL: https://mediaim.expedia.com/cars/19/c181d325-4079-4d28-be54-32cfc9d159d7.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Total price with Tax: $75
Rating: 2.90
```



```
Total price with Tax: $1,341
Passenger capacity: 5
Transmission type: Automatic
Rating: 1.50
Car Page Direction Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=59211
-----
Href: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=59211136locn=Toronto+%2F
Title: Toyota Yaris
Specification: Mini
Image URL: https://media.aim.expedia.com/cars/19/ac8739ca-f1c0-4933-8de7-6563a35845fd.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Price per day: $999
Total price with Tax: $1,341
Passenger capacity: 4
Transmission type: Automatic
Rating: 1.50
Car Page Direction Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=59211
-----
Total cars: 246
Csv file generate successfully and stored scraping data>>

Process finished with exit code 0
```

[Fig-3: Output for Web Crawling of a website]

#### 4. Spell Checker:

- Timil Prajapati: [SpellChecker.java]

##### Purpose

The code snippet performs spell checking on a user's query and ranks products based on the query. It also provides an option for the user to sort the products by rating or view the full list.

##### 1. Spell Check Condition

- Checks if the query length is greater than 3 characters.
- If true, proceed to spell check.

##### 2. Get Spell Check Suggestions

- Calls `getSuggestionWord(query)` to get a list of spelling suggestions for the query.
- If the suggestions list is empty, prints "No suggestions available."
- If the first suggestion matches the query, prints "Word is Correct."

##### 3. Page Ranking

- Creates an instance of the `PageRanking` class.
- Calls `getTopItems("cars1.csv", query)` to get a list of top products based on the

query.

- Prints "Top 10 Products:" and displays the top products using ``printProductsTable(topProducts)``.

#### 4. User Input for Sorting Options

- Prompts the user to press 1 to sort by rating or 2 to view the full list of products.
- Continuously reads user input until a valid option ("1" or "2") is entered.
- If an invalid option is entered, prints "Invalid option. Try Again."
- ``getSuggestionWord(query)``: Method to fetch spelling suggestions, assumed to be implemented elsewhere.
- ``PageRanking`` class: Contains logic for ranking products based on the query.
- ``printProductsTable(topProducts)``: Method to print the top products in a formatted table.
- ``getInput()``: Method to capture user input.

#### **Flow of the function:**

##### 1. Spell Check:

- Check if query length > 3.
- Fetch and handle suggestions.

##### 2. Product Ranking:

- Rank and display top products based on the query.

##### 3. User Interaction:

- Prompt user for sorting/viewing options.
- Validate user input and respond accordingly.

```
// Perform spell check only if query has more than 3 letters
if (query.length() > 3) {
    List<String> suggestions = getSuggestionWord(query); // Assuming this method is implemented elsewhere

    if (suggestions.isEmpty()) {
        System.out.println("No suggestions available.");
    } else if (query.equals(suggestions.get(0))) {
        System.out.println("Word is Correct");

        PageRanking pageRanking = new PageRanking(); // Adjust as necessary to fit your setup
        List<Product> topProducts = pageRanking.getTopItems( filePath: "cars1.csv", query);

        System.out.println("Top 10 Products:");
        printProductsTable(topProducts);

        String choice = "";
        while(true) {
            System.out.println("\nPress 1 to sort by rating or 2 to view the full list of products:");
            choice = getInput().trim();
            if(choice.equals("1") || choice.equals("2")) {
                break;
            } else {
                System.out.println("Invalid option. Try Again");
            }
        }
    }
}
```

```

}
if (choice.equals("1")) {
    List<Product> allProducts = pageRanking.readProductData( filePath: "cars1.csv");
    String finalQuery = query;
    List<Product> filteredProducts = allProducts.stream()
        .filter(product -> product.name.toLowerCase().contains(finalQuery))
        .collect(Collectors.toList());

    // Using Quick Sort to sort the products by rating
    quickSort(filteredProducts, low: 0, high: filteredProducts.size() - 1);

    System.out.println("Products sorted by rating:");
    printProductsTable(filteredProducts);
} else if (choice.equals("2")) {
    List<Product> allProducts = pageRanking.readProductData( filePath: "cars1.csv");
    String finalQuery1 = query;
    List<Product> filteredProducts = allProducts.stream()
        .filter(product -> product.name.toLowerCase().contains(finalQuery1))
        .collect(Collectors.toList());
    System.out.println("Full list of products:");
    printProductsTable(filteredProducts);
} else {
    System.out.println("Invalid option.");
}
} else {
    System.out.println("Spelling is Wrong. Did you mean " + suggestions.get(0) + "?");
    // No need to display top 10 products if the word is incorrect
}
```

[Fig-4.1: Source code for Spell Checker]

```

Search a car (or write exit): chevrolate
This query is searched for 0 times.
This word occurred 0 times in CSV file.
Spelling is Wrong. Did you mean chevrolet?
Search a car (or write exit): chevrolet
This query is searched for 1 times.
This word occurred 15 times in CSV file.
Word Suggestions:
CHEVROLET SPARK
CHEVROLET SILVERADO
CHEVROLET SILVERADO
CHEVROLET MALIBU
Word is Correct
Top 10 Products:
+-----+
Name: Chevrolet Spark
Image Link: https://mediaim.expedia.com/cars/43/75834ed8-0075-4218-8967-57bb19bcee97.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&loen=Winnipeg+%2F

```

[Fig-4.2: Output for Spell Checker]

## 5. Word Completion:

- Sanket Kothiya: [\[WordCompletionTries.java\]](#)

This code provides a method for suggesting words based on a given prefix using a Trie data structure and a CSV file as a data source. Here's a summary of each part:

### 1. Main Method:

Functionality: Retrieves word suggestions for valid inputs.

Implementation:

- Calls ``getWordsWithPrefix(query)`` to get a list of words matching the prefix ``query``.
- Prints "Word Suggestions:" if any results are found.
- Iterates through the list of results and prints each suggested word.

### 2. ``getWordsWithPrefix`` Method:

Functionality: Retrieves a list of words with the given prefix from a CSV file.

Implementation:

- Loads the CSV file (``cars1.csv``) into a Trie using ``loadTrieFromCSV(csvFile)``.
- Uses the ``searchPrefix(prefix)`` method of the Trie to find and return words matching the prefix.

In summary, the code reads words from a CSV file into a Trie data structure and provides word suggestions based on a given prefix by querying the Trie.

```
// Provide word suggestions for all valid inputs
List<String> results = getWordsWithPrefix(query);
if (results.size() >= 1) {
    System.out.println("Word Suggestions: ");
    for (String result : results) {
        System.out.println(result);
    }
}
```

```
// Method to get the list of words with a given prefix
public static List<String> getWordsWithPrefix(String prefix) { 3 usages new *
    String csvFile = "cars1.csv";
    Trie trie = loadTrieFromCSV(csvFile); // Load data into the Trie
    return trie.searchPrefix(prefix); // Get the list of words with the given prefix
}
```

[Fig-5.1: Source Code for Word Completion]

```
Search a car (or write exit): audi
This query is searched for 1 times.
This word occurred 6 times in CSV file.
Word Suggestions:
AUDI A4
AUDI Q3
Word is Correct
Top 10 Products:
+-----+
Name: Audi A4
Image Link: https://mediaim.experia.com/cars/6/c5caf75c-ffe1-44c6-a3d7-d52ddc30f9de.jpg?impolicy=resize&crop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+M
Price: $78
Rating: 4.45
+-----+
Name: Audi Q3
```

[Fig-5.2: Output for Word Completion]

## 6. Word Frequency Count in File:

- Smit Patel: [WordFrequencyInFile.java]

The Word Frequency Count feature is designed to analyze the frequency of words used within the content of the car rental website. This feature helps in understanding user behavior, optimizing content for search engines, and enhancing the overall user experience by identifying popular keywords and trends.

- countOccurrences(query) counts the occurrences of the query word in the CSV file.
- Prints the word occurrence count in the file.
- getSuggestionWord(query) and getWordsWithPrefix(query) (methods assumed to be implemented elsewhere) generate suggestions based on the query.
- If there are suggestions, they are printed.
- Searches for a pattern in a text using the Boyer-Moore algorithm and returns the count of occurrences.
- Reads the entire content of a CSV file and returns it as a lowercase string.
- Counts how many times a given word appears in the CSV file content.
- Prompts the user to enter a word to search for in the CSV file.
- Uses countOccurrences to find the word's frequency.
- Continues to prompt for new words until the user decides to stop.

Hence, this java program reads a CSV file and uses the Boyer-Moore algorithm to count the occurrences of a user-specified word in the file. It includes methods for reading the file, implementing the search algorithm with bad character heuristics, and displaying search results. The main method allows users to input a word, displays how many times it occurs, and offers to search for additional words until the user opts to exit.



```

int frequency = getSearchCount(query); // Assuming this method is implemented elsewhere
System.out.println("This query is searched for " + frequency + " times.");

int wordInFileFreq = countOccurrences(query);
System.out.println("This word occurred " + wordInFileFreq + " times in CSV file.");

// Provide word suggestions for all valid inputs
List<String> results = getWordsWithPrefix(query);
if (results.size() >= 1) {
    System.out.println("Word Suggestions: ");
    for (String result : results) {
        System.out.println(result);
    }
}
}

```

[Fig-6.1: Source Code for Word Frequency Count in File]

```

-----+
Search a car (or write exit): audi
This query is searched for 2 times.
This word occurred 6 times in CSV file.
Word Suggestions:
AUDI A4
AUDI Q3
Word is Correct
Top 10 Products:
-----+
Name: Audi A4
Image Link: https://media. Expedia.com/cars/6/c5caf75c-ffe1-44c6-a3d7-d52ddc30f9de.jpg?impolicy=resizecrop&w=fit&h=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpLn=4675417&locn=Winnipeg%28YW6+--+Winnipeg+James+Armstrong+Richardson+In
Price: $78
Rating: 4.45
-----+
Name: Audi Q3
Image Link: https://media. Expedia.com/cars/6/c5caf75c-ffe1-44c6-a3d7-d52ddc30f9de.jpg?impolicy=resizecrop&w=fit&h=165&rw=165

```

[Fig-6.2: Output for Word Frequency Count in File]

## 7. Search Frequency:

### - Ayush Patel: [SearchQuery.java]

This code snippet is an enhancement of the previous code, providing additional functionalities to count the frequency of word searches and occurrences in a CSV file. Here is a summarized explanation:

#### 1. Frequency of Searches:

- Functionality: Counts how many times the query word has been searched.
- Implementation: Calls `getSearchCount(query)` to get the search count of the



query word.

- Prints the search count.
- avlTree: Instance of AVLTree for storing search queries.
- data: List to store rows from the CSV file.
- searchCounts: HashMap to store counts of search queries.
- Static Block: Executes when the class is loaded.
- Reads the CSV file cars1.csv, skips the header row, and adds each row to data.
- Inserts each value from the CSV file into the avlTree.

This Java program uses an AVL Tree to store and manage search queries from a CSV file, with functionality to track and retrieve the frequency of each query. It reads the CSV file, inserting each entry into the AVL Tree while updating a search count map. The `getSearchCount` method updates the AVL Tree and count map based on user input, and the `main` method provides a user interface for querying and displaying search results, including previously searched terms and their frequencies.

In summary, this code snippet counts the frequency of word searches and occurrences in a CSV file and provides word suggestions based on a prefix by querying a Trie data structure.

```

public class IntegratedFinalFile { new *
    private static void handleSearchAndSuggestions() throws InterruptedException { 1 usage new *

        int frequency = getSearchCount(query); // Assuming this method is implemented elsewhere
        System.out.println("This query is searched for " + frequency + " times.");

        int wordInFileFreq = countOccurrences(query);
        System.out.println("This word occurred " + wordInFileFreq + " times in CSV file.");

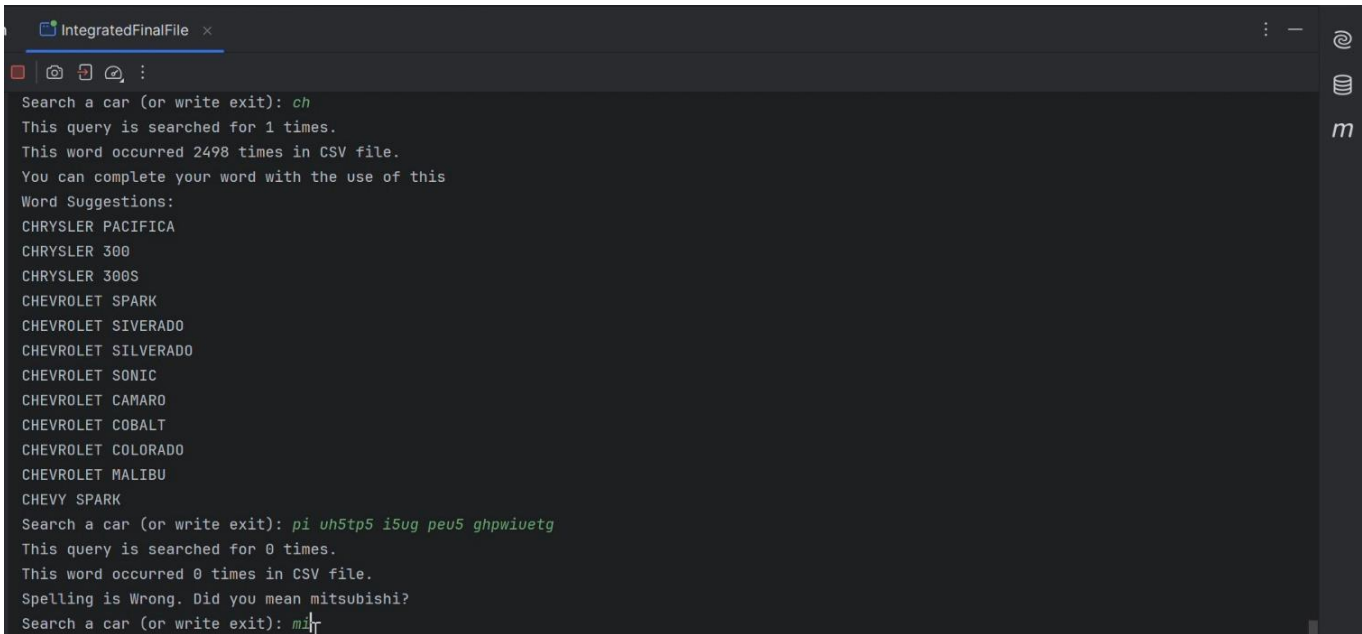
        // Provide word suggestions for all valid inputs
        List<String> results = getWordsWithPrefix(query);
        if (results.size() >= 1) {
            System.out.println("Word Suggestions: ");
            for (String result : results) {
                System.out.println(result);
            }
        }
    }
}

// Public method to get the search count for a given query
public static int getSearchCount(String query) { 3 usages
    query = query.toLowerCase().trim(); // Convert query to lower case and trim spaces
    boolean found = false; // Flag to track if any matching data is found in the CSV file
    // Iterate through each entry in the data list to find matches for the query
    for (String[] entry : data) {
        // Check if any cell in the current entry contains the search query (case-insensitive)
        String finalQuery = query;
        if (Arrays.stream(entry).anyMatch(s -> s.toLowerCase().contains(finalQuery))) {
            found = true; // Set found flag to true if match is found
            break; // Exit loop early since match is found
        }
    }

    // If matching data is found, update the AVLTree and searchCounts
    if (found) {
        avlTree.insert(query); // Insert the search query into the AVLTree for frequency tracking
        searchCounts.put(query, searchCounts.getOrDefault(query, defaultValue: 0) + 1); // Update search count for the query
        return avlTree.findFrequency(query); // Return the frequency count of the search query
    } else {
        return 0; // Return 0 if no matching data is found
    }
}
}

```

[Fig-7.1: Source Code for Search Frequency]




```

IntegratedFinalFile x
Search a car (or write exit): ch
This query is searched for 1 times.
This word occurred 2498 times in CSV file.
You can complete your word with the use of this
Word Suggestions:
CHRYSLER PACIFICA
CHRYSLER 300
CHRYSLER 300S
CHEVROLET SPARK
CHEVROLET SILVERADO
CHEVROLET SILVERADO
CHEVROLET SONIC
CHEVROLET CAMARO
CHEVROLET COBALT
CHEVROLET COLORADO
CHEVROLET MALIBU
CHEVY SPARK
Search a car (or write exit): pi uh5tp5 i5ug peu5 ghpwivetg
This query is searched for 0 times.
This word occurred 0 times in CSV file.
Spelling is Wrong. Did you mean mitsubishi?
Search a car (or write exit): mits

```

[Fig-7.2: Error handling if invalid string is inserted]



```

Name: Audi Q3
Image Link: https://mediaim.expedia.com/cars/42/4d7af24d-3b1c-42e0-9df2-a6b6a8f0a4bc.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg
Price: $125
Rating: 4.25
+-----+
Search a car (or write exit): audi
This query is searched for 2 times.
This word occurred 6 times in CSV file.
Word Suggestions:
AUDI A4
AUDI Q3
Word is Correct
Top 10 Products:
+-----+
Name: Audi A4
Image Link: https://mediaim.expedia.com/cars/6/c5caf75c-ffe1-44c6-a3d7-d52ddc30f9de.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg
Price: $78

```

[Fig-7.2: Output for Search Frequency]

## 8. Page Ranking:

- Gravin Patel: [PageRanking.java]

This Java code is part of a larger program that appears to handle product data, provide word suggestions, and sort products by rating. Here's a summary of the main functions:

### 1. readProductData(String filePath) Function:

- This function reads product data from a CSV file specified by filePath.
- It creates a list of Product objects from the CSV file.
- Each line of the CSV is split by commas, and a Product object is created if there are at least two values in the line.
- The Product objects are added to a productList, which is returned at the end.

## **2. Word Suggestion and Product Handling:**

- If there are no word suggestions, it prints "No suggestions available."
- If the user's query matches the first suggestion, it indicates the word is correct.
- It then creates a PageRanking object and retrieves the top products related to the query from the "cars1.csv" file.
- The top 10 products are printed in a table format.
- The user is prompted to press '1' to sort by rating or '2' to view the full list of products.

## **3. Sorting and Filtering Products:**

- If the user chooses to sort by rating:
- The program reads all product data from "cars1.csv".
- It filters the products to include only those whose names contain the query (case-insensitive).
- It sorts the filtered products by rating using Quick Sort.
- The sorted products are printed in a table format.

This code snippet shows the integration of reading product data from a CSV file, handling user input, and providing sorted product information based on a search query.

```

class PageRanking { new *

    // Function to read product data from CSV file
    public static List<Product> readProductData(String filePath) { 3 usages new *
        List<Product> productList = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] values = line.split( regex: ",");
                if (values.length >= 2) {
                    productList.add(new Product(values[0], values[1], values[4], values[2], values[3], frequency: 0));
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return productList;
    }
}

```

```

        if (suggestions.isEmpty()) {
            System.out.println("No suggestions available.");
        } else if (query.equals(suggestions.get(0))) {
            System.out.println("Word is Correct");

            PageRanking pageRanking = new PageRanking(); // Adjust as necessary to fit your setup
            List<Product> topProducts = pageRanking.getTopItems( filePath: "cars1.csv", query);

            System.out.println("Top 10 Products:");
            printProductsTable(topProducts);

            String choice = "";
            while(true) {
                System.out.println("\nPress 1 to sort by rating or 2 to view the full list of products:");
                choice = getInput().trim();
                if(choice.equals("1") || choice.equals("2")) {

```

[Fig-8.1: Source Code for Search Frequency]

```

+-----+
Search a car (or write exit): chevrolet
This query is searched for 2 times.
This word occurred 15 times in CSV file.
Word Suggestions:
CHEVROLET SPARK
CHEVROLET SILVERADO
CHEVROLET SILVERADO
CHEVROLET MALIBU
Word is Correct
Top 10 Products:
+-----+
Name: Chevrolet Spark
Image Link: https://media.istockphoto.com/photos/43/75834ed8-0075-4218-8967-57bb19bcee97.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+-+Winnipeg+James+Armstrong+Richardson+Int
Price: $68
Rating: 4.30
+-----+
Name: Chevrolet Spark
Image Link: https://media.istockphoto.com/photos/45/cf3108ca-e380-45f3-8a92-a59c4b343f80.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+-+Winnipeg+James+Armstrong+Richardson+Int
Price: $68
Rating: 4.25
+-----+
Name: Chevrolet Malibu

```

[Fig-8.2: Output for Search Frequency]

## 9. URL Extractor:

- Sanket Kothiya: [URLExtractor.java]

This Java class, `URLExtractor`, includes a method to extract URLs from a given text.

Here's a summary of the functionality:

### 1. `extractURLs(String text)` Method:

- Purpose: To extract and return a list of URLs found within the input text.
- Initialize a List: An `ArrayList` named `urls` is created to store the URLs found in the text.
- Define a URL Pattern: A regular expression (`urlPattern`) is defined to match URLs. This regex can match both HTTP and HTTPS URLs, as well as URLs starting with "www.".
- Compile the Pattern: The regular expression is compiled into a `Pattern` object using `Pattern.compile(urlPattern)`.
- Create a Matcher: A `Matcher` object is created by calling `pattern.matcher(text)`. This matcher will scan the input text for substrings that match the regex.
- Find Matches: A while loop is used to find all matches in the text using



`matcher.find()`. For each match found, the matched substring (URL) is added to the `urls` list using `matcher.group()`.

- Return the List: After all matches are found and added to the list, the `urls` list is returned.

```
public class URLExtractor { new *

    // Method to extract URLs from a given text
    // This method takes a String as input and returns a list of URLs found within the text
    public static List<String> extractURLs(String text) { 1usage new *
        // List to store the found URLs
        // We use an ArrayList to store the URLs because it allows dynamic resizing and provides fast access
        List<String> urls = new ArrayList<>();

        // Regular expression to match URLs
        // This regex is designed to match both HTTP and HTTPS URLs, as well as URLs starting with "www."
        String urlPattern = "http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\\)]|(?:%[0-9a-fA-F][0-9a-fA-F]))+|"
            + "www\\.?(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\\)]|(?:%[0-9a-fA-F][0-9a-fA-F]))+";

        // Compile the regular expression into a pattern
        // This step is necessary because it converts the string representation of the regex into a Pattern object
        Pattern pattern = Pattern.compile(urlPattern);

        // Create a matcher to find matches of the pattern in the text
        // The matcher will scan the text and try to find substrings that match the regex
        Matcher matcher = pattern.matcher(text);

        // Loop through all matches found in the text
        // The find() method of the Matcher class returns true if a substring matching the pattern is found
        while (matcher.find()) {
            // Add each found URL to the list
            // The group() method of the Matcher class returns the matched substring
            urls.add(matcher.group());
        }
    }
}
```

[Fig-9.1: Source Code for Search Frequency]

```
ating: 4.25
-----+
ame: Chevrolet Malibu
image Link: https://media.iamexpedia.com/cars/38/221c13d6-9b20-4f10-b3c6-53767ae8db09.jpg?impolicy=resize&crop&ra=fit&rh=165&rw=165
ink: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
rice: $73
ating: 4.45
-----+
ame: Chevrolet Malibu
image Link: https://media.iamexpedia.com/cars/38/d9f2b096-9ccb-4162-9b2b-4c14e41e73bd.jpg?impolicy=resize&crop&ra=fit&rh=165&rw=165
ink: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
rice: $73
ating: 4.45
-----+
ame: Chevrolet Silverado
image Link: https://media.iamexpedia.com/cars/19/77009bb8-60a1-402d-a1da-8c1375508a41.jpg?impolicy=resize&crop&ra=fit&rh=165&rw=165
ink: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
rice: $59
ating: 2.90
-----+
ame: Chevrolet Malibu
image Link: https://media.iamexpedia.com/cars/39/84f9a5bc-ac97-494e-9aa8-7935d8ed7507.jpg?impolicy=resize&crop&ra=fit&rh=165&rw=165
ink: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dpln=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
rice: $83
ating: 4.40
-----+
```

[Fig-9.2: Output for Search Frequency]



## 10. Rating Based Sorting using Quick Sort:

- Smit Patel: [\[IntegratedFinalFile.java\]](#)

This Java code snippet handles the user's choice to sort products by rating. Here's a summary of the functionality:

### 1. Checking User Choice:

- The code checks if the user's choice is "1", indicating they want to sort products by rating.

### 2. Reading and Filtering Products:

- Read All Products: It reads all product data from the CSV file "cars1.csv" using the `readProductData` method of the `PageRanking` class.
- Filter Products: It filters the products to include only those whose names contain the query string (case-insensitive). This is done using Java Streams and the `filter` method, converting product names to lowercase and checking if they contain the query.

### 3. Sorting Products by Rating:

- Quick Sort: It sorts the filtered products by rating using the Quick Sort algorithm. The `quickSort` method is called with the filtered products list, the starting index (0), and the ending index (`filteredProducts.size() - 1`).

### 4. Displaying Sorted Products:

- Print Sorted Products: It prints the sorted list of products using the `printProductsTable` method.

This part of code is used in a context where users can search for products by name and then choose to sort the results by rating. After the user inputs their choice, the program reads the product data, filters it based on the search query, sorts the filtered list by rating, and prints the sorted products.

This code effectively combines reading, filtering, sorting, and displaying product data based on user input and the specified query.

```

if (choice.equals("1")) {
    List<Product> allProducts = pageRanking.readProductData( filePath: "cars1.csv");
    String finalQuery = query;
    List<Product> filteredProducts = allProducts.stream()
        .filter(product -> product.name.toLowerCase().contains(finalQuery))
        .collect(Collectors.toList());

    // Using Quick Sort to sort the products by rating
    quickSort(filteredProducts, low: 0, high: filteredProducts.size() - 1);

    System.out.println("Products sorted by rating:");
    printProductsTable(filteredProducts);
}

```

[Fig-10.1: Source Code for Search Frequency]

```

Press 1 to sort by rating or 2 to view the full list of products:
1
Products sorted by rating:
+-----+
Name: Chevrolet Malibu
Image Link: https://media.iam.expedia.com/cars/39/84f9a5bc-ec97-494e-9aa8-7935d8ed7507.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
Price: $83
Rating: 4.60
+-----+
Name: Chevrolet Malibu
Image Link: https://media.iam.expedia.com/cars/39/24d002a1-55b3-4b47-9faf-fdd6407a3fdf.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
Price: $83
Rating: 4.60
+-----+
Name: Chevrolet Silverado
Image Link: https://media.iam.expedia.com/cars/15/97eb8156-ff12-4935-ac5a-5be73a7feb9.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
Price: $121
Rating: 4.55
+-----+
Name: Chevrolet Malibu
Image Link: https://media.iam.expedia.com/cars/38/221c13d6-9b20-4f10-b3c6-53767ea8db09.jpg?impolicy=resizecrop&ra=fit&rh=165&rw=165
Link: https://www.carrentals.com/carsearch/details?date1=8%2F7%2F2024&date2=8%2F8%2F2024&time1=1030AM&time2=1030AM&dp1n=4675417&locn=Winnipeg+%28YWG+--+Winnipeg+James+Armstrong+Richardson+Int
Price: $73

```

[Fig-10.2: Output for Search Frequency]

## References:

### [1] Regular Expressions in Java.

[https://books.google.ca/books?hl=en&lr=&id=1eZDDwAAQBAJ&oi=fnd&pg=PP1&dq=regex+in+java&ots=H02QkbT6KK&sig=CHO-IDCmz3tKzmTk3V920\\_iquonU#v=onepage&q=regex%20in%20java&f=false](https://books.google.ca/books?hl=en&lr=&id=1eZDDwAAQBAJ&oi=fnd&pg=PP1&dq=regex+in+java&ots=H02QkbT6KK&sig=CHO-IDCmz3tKzmTk3V920_iquonU#v=onepage&q=regex%20in%20java&f=false)

### [2] A study on Page Ranking Algorithm.

[https://d1wqtxts1xzle7.cloudfront.net/100150642/pdf-libre.pdf?1679481893=&response-content-disposition=inline%3B+filename%3DA\\_Comparative\\_Study\\_Of\\_Page\\_Ranking\\_Algo.pdf&Expires=1721859934&Signature=fifgtaQGUUVKeNPz-CZ4p7LmGs0uRu9zpRUCU9p~mDG-tAsrn-u~myFKy5MqnvQ-hHDvkAO4DBJLqaBDuWeAQqOnXW778CKpMJFEglcZOQZ~s2EmMCSkIG6XMJzNyEePoELWazAl8HrR38-3mglHtzHWo4khBXVqthqlxWsJmzKnsC4ca4beQChoeGEogRcddOxezQhI8rKYreU3Z0DvELV6hR3e0DMcir7ddvKnxB1JVZO~WhrgAEKYcvzHngRCGEqeQ0WntZW-ToTsWCgEr7AjpgCIM2AYV2YV0VZc8Xyl~0MB-qH1fzBxC7j0p4pdyh5ryh27rmd5zK6za0HIQ\\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA](https://d1wqtxts1xzle7.cloudfront.net/100150642/pdf-libre.pdf?1679481893=&response-content-disposition=inline%3B+filename%3DA_Comparative_Study_Of_Page_Ranking_Algo.pdf&Expires=1721859934&Signature=fifgtaQGUUVKeNPz-CZ4p7LmGs0uRu9zpRUCU9p~mDG-tAsrn-u~myFKy5MqnvQ-hHDvkAO4DBJLqaBDuWeAQqOnXW778CKpMJFEglcZOQZ~s2EmMCSkIG6XMJzNyEePoELWazAl8HrR38-3mglHtzHWo4khBXVqthqlxWsJmzKnsC4ca4beQChoeGEogRcddOxezQhI8rKYreU3Z0DvELV6hR3e0DMcir7ddvKnxB1JVZO~WhrgAEKYcvzHngRCGEqeQ0WntZW-ToTsWCgEr7AjpgCIM2AYV2YV0VZc8Xyl~0MB-qH1fzBxC7j0p4pdyh5ryh27rmd5zK6za0HIQ_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)

### [3] Data Structure and Algorithms in Java.

[https://www.google.ca/books/edition/Data\\_Structures\\_and\\_Algorithms\\_in\\_Java\\_I/F2vqEAAAQBAJ?hl=en&gbpv=1&dq=A\\_VL+and+red+black+trees+and+its+algorithm+in+java&printsec=frontcover](https://www.google.ca/books/edition/Data_Structures_and_Algorithms_in_Java_I/F2vqEAAAQBAJ?hl=en&gbpv=1&dq=A_VL+and+red+black+trees+and+its+algorithm+in+java&printsec=frontcover)

### [4] Selenium WebDriver Quick Starter.

[https://www.google.ca/books/edition/Selenium\\_WebDriver\\_Quick\\_Start\\_Guide/48t1DwAAQBAJ?hl=en&gbpv=1&dq=hashmap+in+java&printsec=frontcover](https://www.google.ca/books/edition/Selenium_WebDriver_Quick_Start_Guide/48t1DwAAQBAJ?hl=en&gbpv=1&dq=hashmap+in+java&printsec=frontcover)

### [5] Optimize Boyer Moore Algorithm Using Parallel Processing.

[https://d1wqtxts1xzle7.cloudfront.net/60842177/OPTIMIZE\\_BOY\\_thesis20191008-73295-elqdw1-libre.pdf?1570594169=&response-content-disposition=inline%3B+filename%3DOPTIMIZE\\_BOYER\\_MOORE\\_ALGORITHM\\_USING\\_PAR.pdf&Expires=1721859972&Signature=T9T0T0FQo-37CFYe75UDDHZ96yMzZGPIS26qZ7HgxQqNtbP67gM2Tcz2cBovR5wBFjDpLKswJRql63mb6E5hI-LGDKtiE5-gSmk6AIqh7XZe-b~BtG49QCI0NtAIYwYHoDCKWdEak3dzXXwDwTwPUisiHxHqjeMUq4yqvPZrLGLbeKuNdjEQ7~H31F0qKYyDO35MW4qn2jBOCyzVUKq3BJLr2F-qvq9MctewwSVrn965B82zA7RwFaUH8GtodqJs1j6Yax4inb7yI31I7vYx86zctWz6z9CYRbycQaQUQ7vntLhR86lwWgwSd-aHQGLue-PxpIufF~bl8Hpqzoi2eA\\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA](https://d1wqtxts1xzle7.cloudfront.net/60842177/OPTIMIZE_BOY_thesis20191008-73295-elqdw1-libre.pdf?1570594169=&response-content-disposition=inline%3B+filename%3DOPTIMIZE_BOYER_MOORE_ALGORITHM_USING_PAR.pdf&Expires=1721859972&Signature=T9T0T0FQo-37CFYe75UDDHZ96yMzZGPIS26qZ7HgxQqNtbP67gM2Tcz2cBovR5wBFjDpLKswJRql63mb6E5hI-LGDKtiE5-gSmk6AIqh7XZe-b~BtG49QCI0NtAIYwYHoDCKWdEak3dzXXwDwTwPUisiHxHqjeMUq4yqvPZrLGLbeKuNdjEQ7~H31F0qKYyDO35MW4qn2jBOCyzVUKq3BJLr2F-qvq9MctewwSVrn965B82zA7RwFaUH8GtodqJs1j6Yax4inb7yI31I7vYx86zctWz6z9CYRbycQaQUQ7vntLhR86lwWgwSd-aHQGLue-PxpIufF~bl8Hpqzoi2eA_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)