**VANIER COLLEGE**

**420-941 WEB SERVICES**

**NOVEMBER 2024**

**REPORT**

**BIKE SHARING WEB SERVICES**

**REST AND SOAP IMPLEMENTATIONS**

**BY:**

**NICOLAS LAUZON (2495040)**

**Table of Content:**

[1. INTRODUCTION 1](#_Toc181452975)

[1.1. Business Case 1](#_Toc181452976)

[1.2. Objective 1](#_Toc181452977)

[1.3. Developed Bike Sharing Management Tools 1](#_Toc181452978)

[1.4. Report Content 2](#_Toc181452979)

[2. DESIGN 2](#_Toc181452980)

[2.1. Web Service Architecture 2](#_Toc181452981)

[2.2. Data and Model Classes 4](#_Toc181452982)

[2.3. Calculation of Distance Between Station 6](#_Toc181452983)

[2.4. Web Services 6](#_Toc181452984)

[2.5. Resources 7](#_Toc181452985)

[3. TEST CASES 8](#_Toc181452986)

[3.1. REST Implementations 8](#_Toc181452987)

[3.1.1. Login, Logout and Account View 9](#_Toc181452988)

[3.1.2. Station Details 11](#_Toc181452989)

[3.1.3. Distance Between Two Stations 12](#_Toc181452990)

[3.1.4. Stations Within the Radius of a Given Location 13](#_Toc181452991)

[3.1.4 JSON Handling 14](#_Toc181452992)

[3.2. SOAP Implementations 16](#_Toc181452993)

[3.2.1 Login Method 17](#_Toc181452994)

[3.2.2 User Info Method 18](#_Toc181452995)

[3.2.3 User Ride List Method 18](#_Toc181452996)

[4. CLIENT APPLICATION 20](#_Toc181452997)

[5. CONCLUSION 21](#_Toc181452998)

**List of Tables**

[Table 1. Fields of the StationInfo Class 5](#_Toc181452999)

[Table 2. Fields of the StationStatus Class 5](#_Toc181453000)

[Table 3. Fields of the User Class 5](#_Toc181453001)

[Table 4. Fields of the Ride Class 5](#_Toc181453002)

[Table 5. Field of the RideList Class 6](#_Toc181453003)

[Table 6. REST Web Services Description 7](#_Toc181453004)

[Table 7. SOAP Web Services Description 7](#_Toc181453005)

[Table 8. REST Web Services for JSON Data Extraction 15](#_Toc181453006)

**List of Figures**

[Figure 1. Web Service Architecture 3](#_Toc181453007)

[Figure 2. UML Diagram of the REST and SOAP Implementations 4](#_Toc181453008)

[Figure 3. Client Application Architecture 4](#_Toc181453009)

[Figure 4. BixiRESTMainPage.html Web Page 8](#_Toc181453010)

[Figure 5. Call of the Web Service for the Dropdown Selection of Station 9](#_Toc181453011)

[Figure 6. Call of the Web Service for the List of Stations 9](#_Toc181453012)

[Figure 7. BixiRESTMainPage.html Web Page after User Login 9](#_Toc181453013)

[Figure 8. BixiRESTMainPage.html Web Page after Account Functionality Activated 10](#_Toc181453014)

[Figure 9. Login, Logout, Services and Account JavaScript Functions 11](#_Toc181453015)

[Figure 10. Call of the Web Service for User Checkup at Login 11](#_Toc181453016)

[Figure 11. Show – Station Details – REST Web Service 12](#_Toc181453017)

[Figure 12. Call – Station Details – REST Web Service 12](#_Toc181453018)

[Figure 13. Show – Distance Estimate Between Two Station – REST Web Service 13](#_Toc181453019)

[Figure 14. Call – Distance Between Two Stations – REST Web Service 13](#_Toc181453020)

[Figure 15. Show – Station List from a Given Location, Within a Given Radius 14](#_Toc181453021)

[Figure 16. Call – Station List from a Given Location, Within a Given Radius 14](#_Toc181453022)

[Figure 17. Calls – Data Extraction from JSON Parsing – REST Web Service 15](#_Toc181453023)

[Figure 18. JSON Parsing of the StationInfo Data – REST Web Service 16](#_Toc181453024)

[Figure 19. JSON Parsing of the StationDetails Data – REST Web Service 16](#_Toc181453025)

[Figure 20. WebBixi Web Service Description Language Information 17](#_Toc181453026)

[Figure 21. SOAP Login Method Application on Postman 17](#_Toc181453027)

[Figure 22. SOAP User Info Method Application on Postman 18](#_Toc181453028)

[Figure 23. SOAP Ride List Method Application on Postman, Part 1 of 4 19](#_Toc181453029)

[Figure 24. SOAP Ride List Method Application on Postman, Part 2 of 4 19](#_Toc181453030)

[Figure 25. SOAP Ride List Method Application on Postman, Part 3 of 4 20](#_Toc181453031)

[Figure 26. SOAP Ride List Method Application on Postman, Part 4 of 4 20](#_Toc181453032)

[Figure 27. Client Application Consumption Results, Part 1 of 2 21](#_Toc181453033)

[Figure 28. Client Application Consumption Results, Part 2 of 2 21](#_Toc181453034)

# 1. INTRODUCTION

## 1.1. Business Case

Pools of communal resources shared by members of the community leads to the practice of reducing consumption, promoting more sustainable living and environmental well-being. Notably, the transportation sector benefits greatly from public transit and pooling of many means of transportation such as cars and bicycles. Multiple users share the use of means of transportation at a reasonable cost, instead of relying on costly, single-owned vehicles.

Managing communal resources is however essential to ensure success. Users of these communal resources must have the tools to locate these resources, check their availability and book them for their personal use.

## 1.2. Objective

The objective of this work is to develop web services that emulate simple versions of tools for managing communal resources and user account information. The developed web services focus on bicycle pooling, as implemented by BIXI in Montreal, and data publicly available from that company is used on test cases.

This document addresses tests of REST and SOAP implementations of web services for bike sharing management tools. REST implementations focus on communal resources management (i.e., the bicycles), while SOAP implementation focus on user account management.

## 1.3. Developed Bike Sharing Management Tools

Using REST implementations, the following tools have been developed:

* The listing of all stations on the bike share network, where to retrieve and return bikes: This tool allows users to access an overview of stations close to their points of origin and destination.
* The details of a given station of the network: This tool provides users an account of the status of the resources at the station, such as the number of total bikes available, of electric bikes available, of disabled bikes, and of docks available and disabled.
* The distance between two stations: This tool allows users to estimate the amount of effort to be made between the origin station, where a bike is retrieved, to the destination station, where the bike is returned.
* The stations present within a radius from a given location: This tool provides to users a more focused view of stations from where they may be located, for the purpose of finding a station close to their points of origin and destinations.

The results from all the REST implementations can be viewed from the dynamic elements of a single web page deployed on the server with the web services.

Using SOAP implementations, the following tools have been developed:

* A login method to verify whether a username and password inputted by a person match the credentials of an entry of the records of users. The method returns the user ID if the inputs are valid. This method is replicated as a REST implementation as well, returning the username if valid.
* A user account information viewer method that returns the information details of that user. The user is identified in that method by their ID.
* A bike ride list viewer method that returns the list of bike rides performed by a user. The user is identified in that method by their ID.

In its current stage, the results from all the SOAP implementations can be viewed using Postman.

## 1.4. Report Content

Section 2 of the report addresses the design requirements serving as the foundation of the web services developed for the project.

Section 3 presents the display on a web page of the results from the consumption of the REST web services, while the results from the consumption of the SOAP services are shown using Postman. This section also presents the results of a preliminary investigation of directly consuming web services and parsing JSON data from Bixi, for the purpose of using such data instead of currently hardcoded data in a subsequent phase of the project.

Section 4 presents the results of a client application consuming the REST web services of the project for viewing on a simple console.

Finally, Section 5 provides concluding remarks.

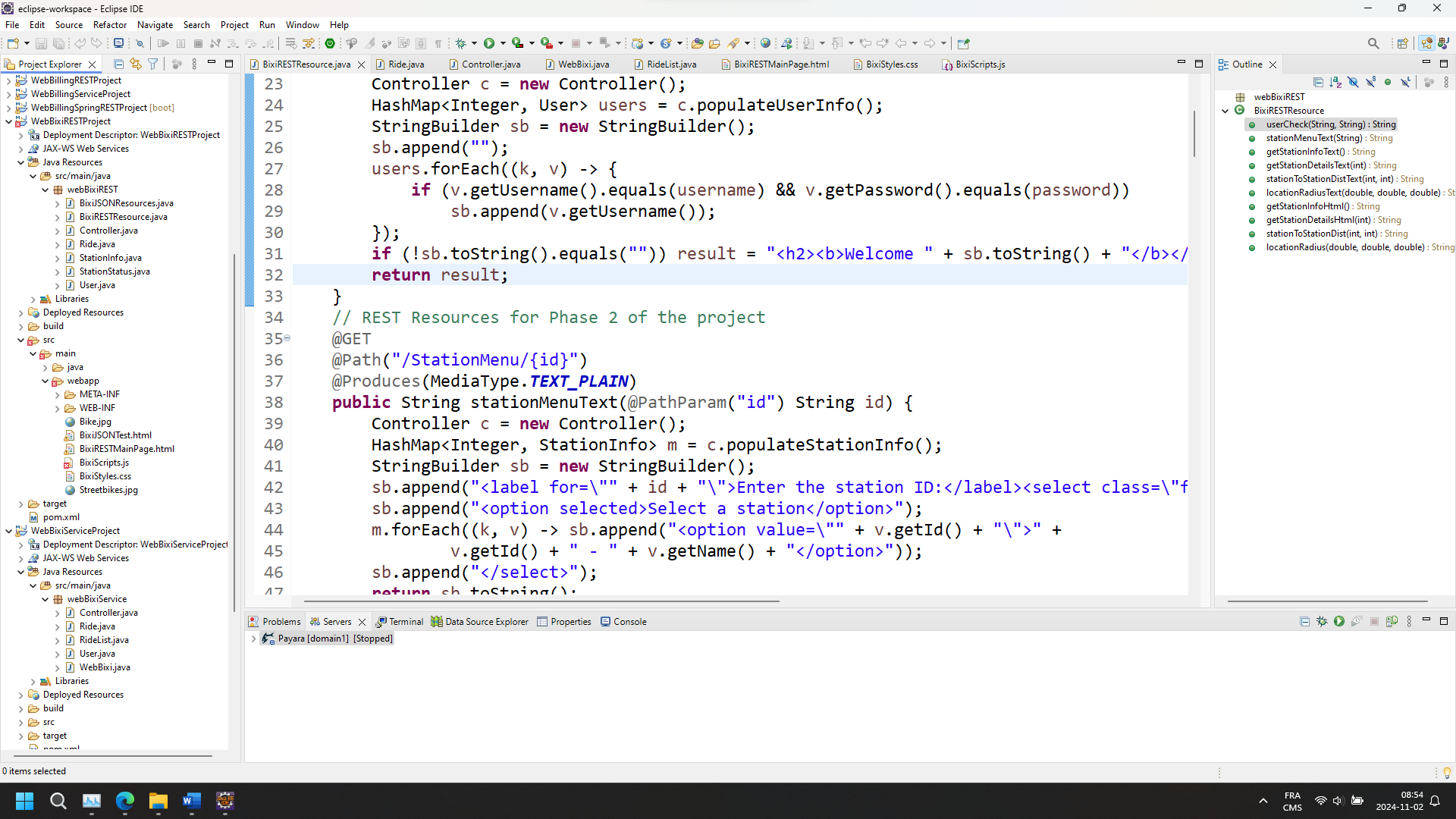
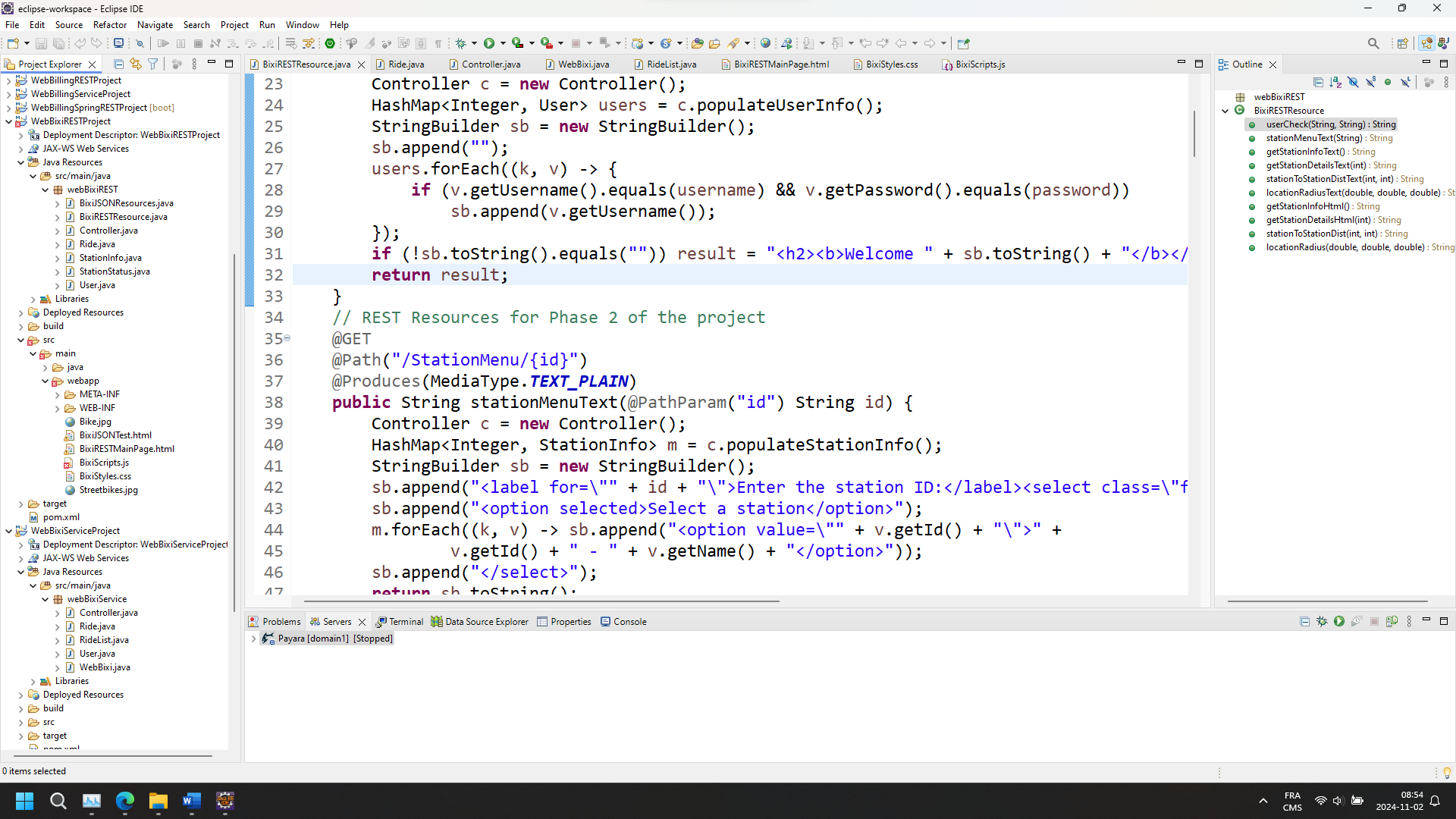
# 2. DESIGN

## 2.1. Web Service Architecture

The java program projects developed to implement both the REST and SOAP web services loosely follows a Model-View-Controller architecture, whereby:

* The Model component codes classes defining the structure of the data to be collected from a local database or an API.
* The View component codes the dynamic elements rendered to users on the single web page, as responses to their requests.
* The Controller component codes the methods for retrieving data and performing manipulations such as the calculation of distances between two stations or the retrieval of stations within a given radius and location.

The web service architecture within the REST and SOAP projects folders is illustrated in   
Figure 1.



(a) REST implementation (b) SOAP implementation

Figure 1. Web Service Architecture

The following classes have been created to implement the web services:

* BixiRESTResource and WebBixi classes: Acting as the View components of the REST and SOAP implementation programs, respectively, and using all the necessary classes of the Jakarta libraries to construct the web services queries. The methods within that class call on methods within the Controller class.
* Controller class: Acting as the Controller component of both the REST and SOAP implementation programs, incorporating methods that retrieve data and other function related to business logic.
* StationInfo, StationStatus, User, Ride and RideList classes: Acting as the Model components of the programs.

The UML diagram illustrating the components of the classes and their relationships is shown in Figure 2.

For REST implementations, a single web page is BixiRESTMainPage.html is created. Styling of that web page is defined in the BixiStyles.css file, while the JavaScript codes calling on the web services to feed the dynamic elements of the web page are stored in the BixiScripts.js file.

For SOAP implementations, Postman is employed to view the results.

As a test, a client application has been built to demonstrate the access of the REST web services from outside of the server. The client application consists of a simple class with a main method (clientBixiREST), as shown in Figure 3.



Figure 2. UML Diagram of the REST and SOAP Implementations

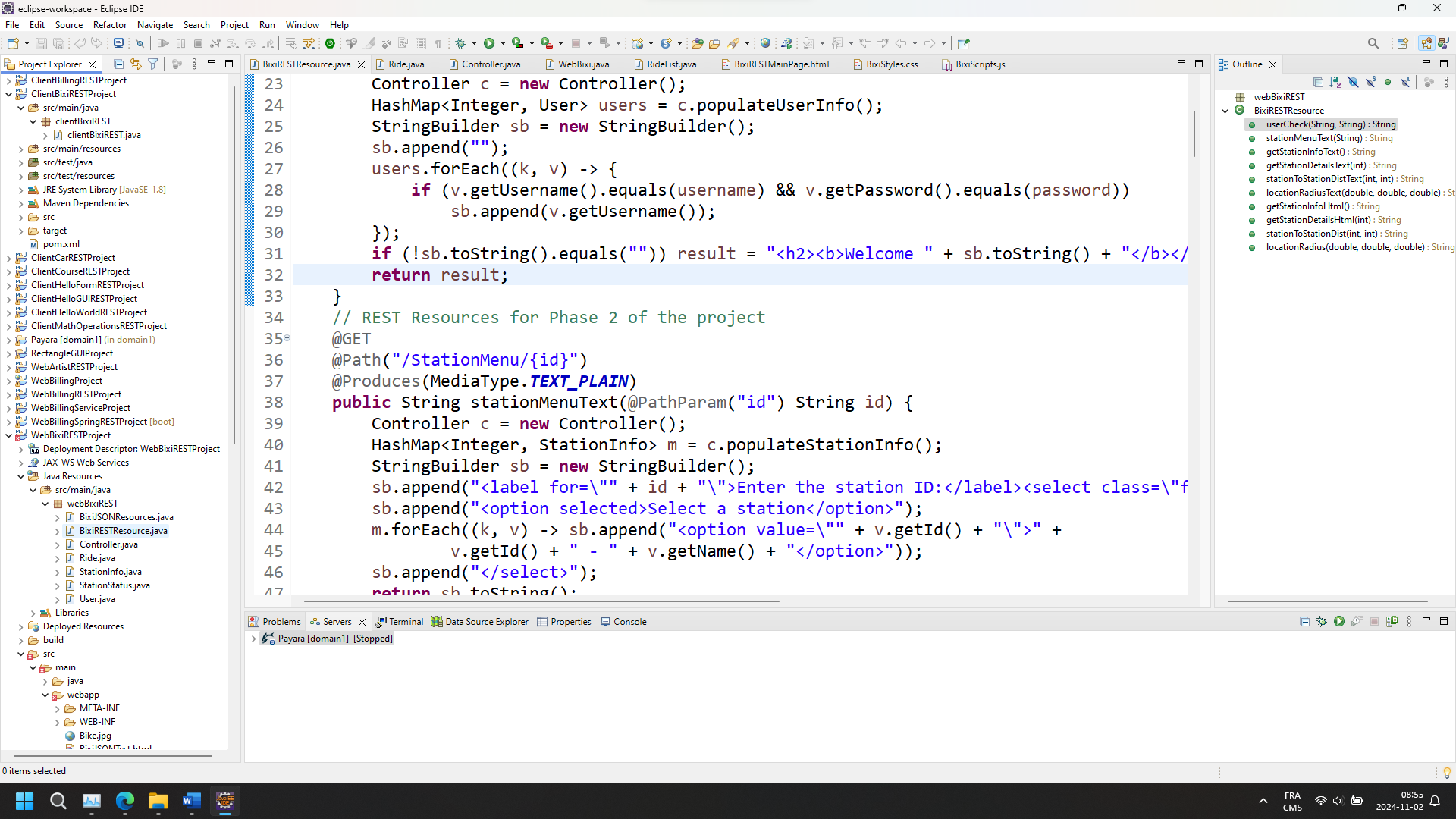


Figure 3. Client Application Architecture

## 2.2. Data and Model Classes

The listing of the stations of the BIXI network, along with status information (e.g., number of bikes available and disabled) are publicly available, from these two links, respectively:

* https://gbfs.velobixi.com/gbfs/en/station\_information.json
* https://gbfs.velobixi.com/gbfs/en/station\_status.json

The data content from these two links has dictated the structure of the StationInfo and StationStatus classes implemented in support for the REST web services The fields of these two classes are described in Tables 1 and 2, respectively.

Table 1. Fields of the StationInfo Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **Type** | **Name** | **Description** |
| Private | int | id | A unique identifier for each station |
| Private | String | name | The name of the station, typically the intersection of streets |
| Private | double | lat | The latitude of the station |
| Private | double | lon | The longitude of the station |
| Private | int | capacity | The number of bike docks at the station |

Table 2. Fields of the StationStatus Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **Type** | **Name** | **Description** |
| Private | int | id | The same identifier in StationInfo class |
| Private | int | aBikes | The total number of available bikes at the station |
| Private | int | aEbikes | The number of electrical bikes available at the station |
| Private | int | dBikes | The number of disabled bikes at the station |
| Private | int | aDocks | The number of available docks at the station |
| Private | int | dDocks | The number of disabled docks at the station |
| Private | long | check | The date-time stamp of the last change in the status of the station |

The User, Ride and RideList classes are implemented in support of the SOAP web services, and in support as well of a REST web service in the case of the User class. The fields of these three classes are described in Tables 3, 4 and 5, respectively.

Table 3. Fields of the User Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **Type** | **Name** | **Description** |
| Private | int | userID | A unique identifier for each user |
| Private | String | username | The name used by a user for login |
| Private | String | password | The password used by a user for login |
| Private | String | usertype | A qualifier for the status a user (occasional, regular or platinum) |
| Private | String | address | The user’s address |
| Private | String | email | The user’s email |

Table 4. Fields of the Ride Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **Type** | **Name** | **Description** |
| Private | int | rideID | A unique identifier for each ride |
| Private | int | userID | The identifier of the user for the ride |
| Private | int | origID | The identifier of the station from where the ride started |
| Private | int | destID | The identifier of the station to where the ride ended |
| Private | long | aDocks | The date and time the ride started |
| Private | long | dDocks | The date and time the ride ended |

Table 5. Field of the RideList Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **Type** | **Name** | **Description** |
| Private | ArrayList<Ride> | rides | Intended to contain the rides performed by a given user. That fields facilitates the transformation of the data in that field into an XML format. |

For all classes, default and field constructors have been implemented, as well as the getters and setters of all fields. The structure employed as support for the data carried by the objects of these two classes is a HashMap (i.e., one for each class).

For the sake a simplicity, all the required data have been hardcoded in the program, in the Controller class. However, a preliminary investigation of directly consuming Bixi web services, and parsing the JSON objects to extract the data has been undertaken.

The code implementing that investigation is located in the BixiJSONResources.java file. The BixiJSONTest.html web page provides a form to call the web services within that code, and includes a dynamic element to show the results from these web services. These two files appear in Figure 1, as a part of the project.

## 2.3. Calculation of Distance Between Station

Calculation of distance between two stations can be done by first converting their respective latitude and longitude into Universal Transverse Mercator (utm) coordinates, which express a location by a Northing and an Easting value in meters. Second, a distance between two stations can be calculated on the basis of their respective utm coordinates.

Conversion from latitude-longitude to utm coordinates can be done using an API developed by NOAA, as explained at: https://geodesy.noaa.gov/web\_services/ncat/lat-long-height-service.  
shtml. That API is however relatively slow and is not practical for use when hundreds of calculations of distances must be made.

Within the program, the local variables of the distanceCalc method in the Controller class defines the coordinates of the four corners of the area encapsulating all the stations of the BIXI network, as well as the distance between these corners, as derived from their utm coordinates. Calculation of the distance between two stations is then accomplished by interpolation within that area.

## 2.4. Web Services

Six REST web services have been implemented within the BixiRESTResource class, and are described in Table 6. Three SOAP web services have been with the WebBixi class, and are described in Table 7.

Table 6. REST Web Services Description

| **Description** | **Called Method** |
| --- | --- |
| Verifying the username and password at login |  |
| Listing stations in a dropdown selection |  |
| Listing stations in a table |  |
| Details for a selected station |  |
| Distance between two selected stations |  |
| List of stations from a given location, within a given radius |  |

Table 7. SOAP Web Services Description

| **Description** | **Called Method** |
| --- | --- |
| Verifying the username and password at login |  |
| Show the user information |  |
| Show the rides performed by the user information |  |

## 2.5. Resources

Implementation of the project has required the inclusion of the following dependencies:

* org.glassfish.jersey.containers:
* org.glassfish.jersey.inject
* com.sun.activation
* com.sun.xml.ws
* jakarta.json

# 3. TEST CASES

## 3.1. REST Implementations

To facilitate testing of the REST web services tools, a welcome page giving access to them has been developed and can be accessed from:

http://localhost:8080/WebBixiRESTProject/BixiRESTMainPage.html

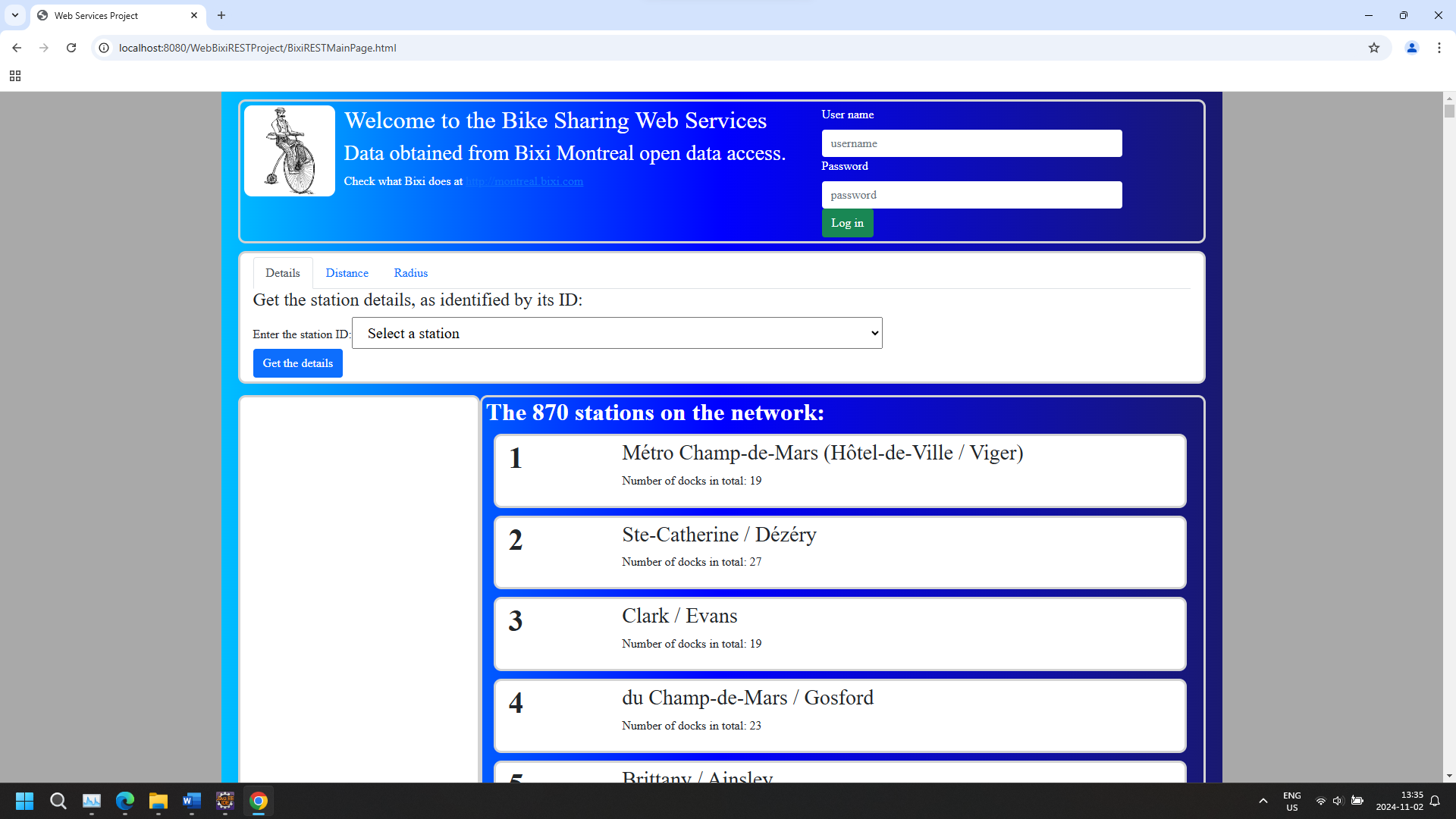
Obviously, the IP address and port in the link above must match those of the server where the program is deployed. The web page is illustrated in Figure 4. The first two rows of the web page are static, with the exception of the login area, which is dynamic. The two columns in the third and last rows are dynamic and are updated depending on the request made by the user.

The second row of the web page consists of a series of tabs attached to web services, namely getting:

* The details of a selected station (Section 3.1.2);
* The distance between two selected stations (Section 3.1.3); and
* The list of stations from a given location, within a given radius (Section 3.1.4).

For the first two web services, a dropdown selection is built from a web service returning a list with the id and name of every station. The call of that web service is established in JavaScript (BixiScripts.js), as illustrated in Figure 5.

By default, the list of stations, with their names and numbers of bike docks, is uploaded when the web page is called on a browser. The call of that web service, established in JavaScript, is illustrated in Figure 6. That list is located in the dynamic element of the right column of the third row of the web page. That list can be replaced by a more focused list from a given location, within a given radius (Section 3.1.4), but can always be called back at the user’s request.



**Dynamic**

**elements**

**Static**

**elements**

**Dynamic**

**elements**

Figure 4. BixiRESTMainPage.html Web Page

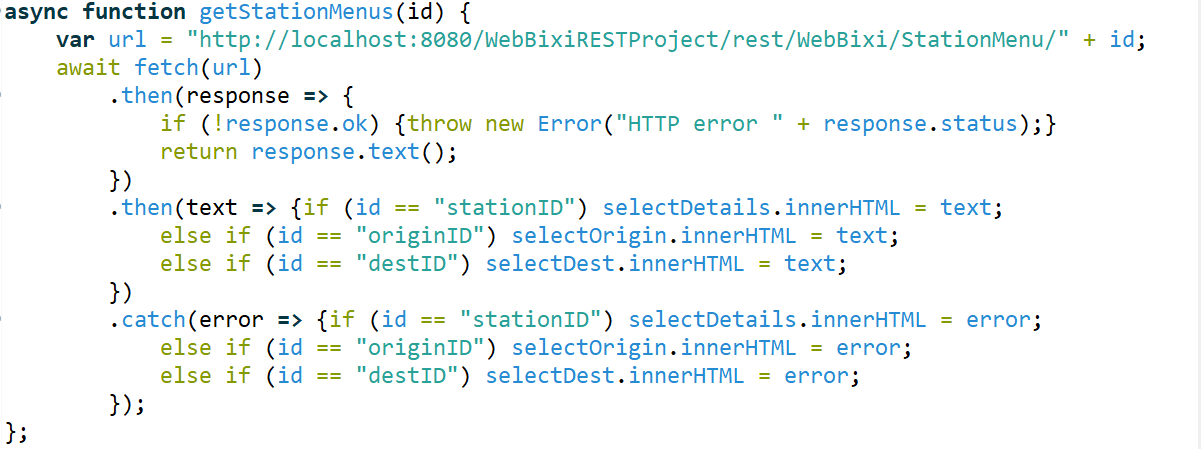


Figure 5. Call of the Web Service for the Dropdown Selection of Station



Figure 6. Call of the Web Service for the List of Stations

### 3.1.1. Login, Logout and Account View

The login area of the web page (Figure 4) manages the access to user account information. If no one logs in, no account information is available. After a person logs in, as illustrated in Figure 7, functionalities are offered in the upper right corner of the web page.

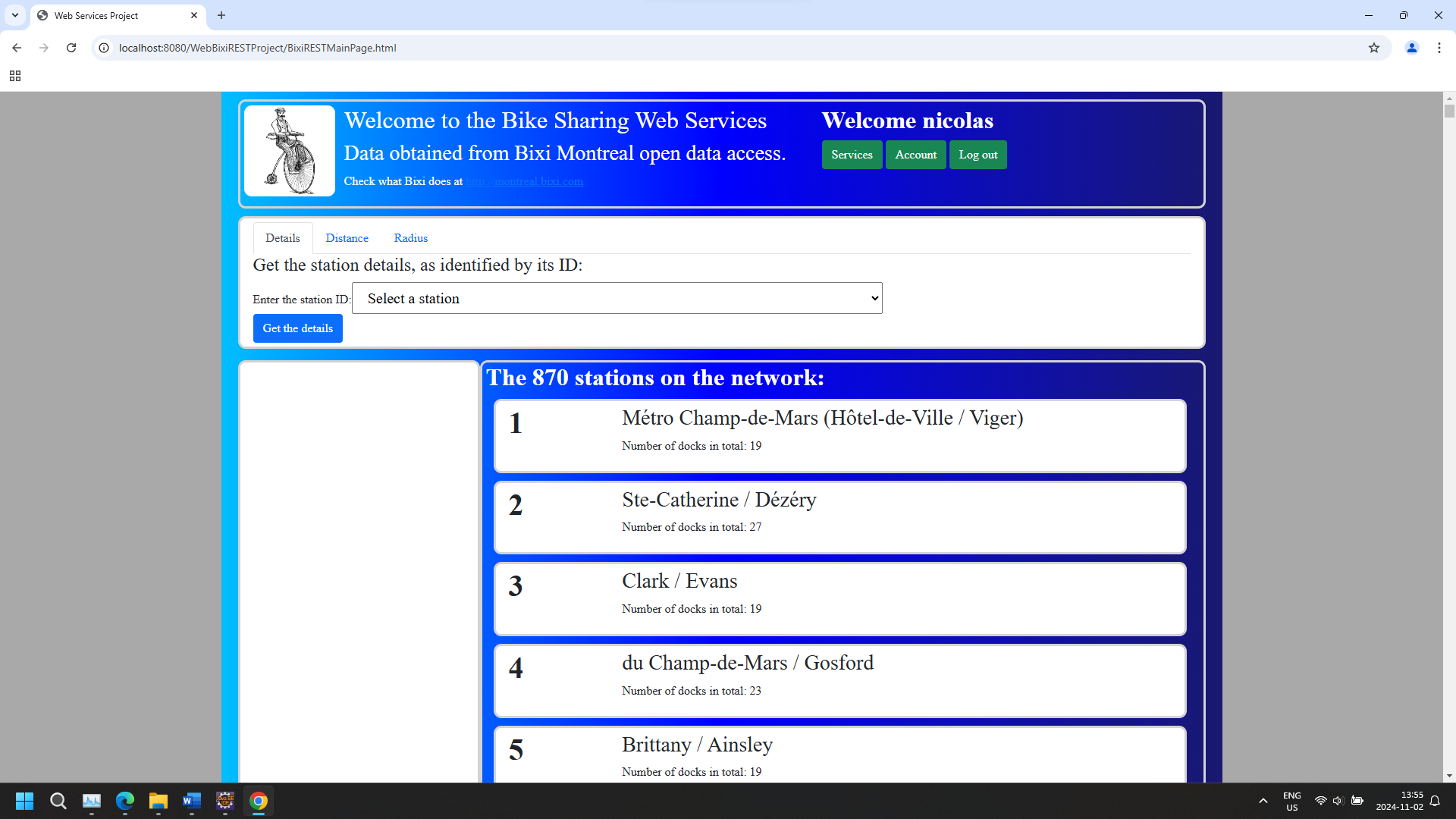


Figure 7. BixiRESTMainPage.html Web Page after User Login

The included functionalities are:

* Log out: By clicking on that button, the web page returns to its initial state, with a login area.
* Services: By clicking on that button, the web page displays the usual third row seen on Figures 4 and 7, allowing the user to access the REST web services focused on communal resources (i.e., the stations and bicycles).
* Account: By clicking on that button, the web page displays a third row allowing the user to access the web services focused their account information.

For now, the web services focused on account information are SOAP implementation testable on Postman only, as described in Section 3.2. On the web page, the account display is as shown in Figure 8.

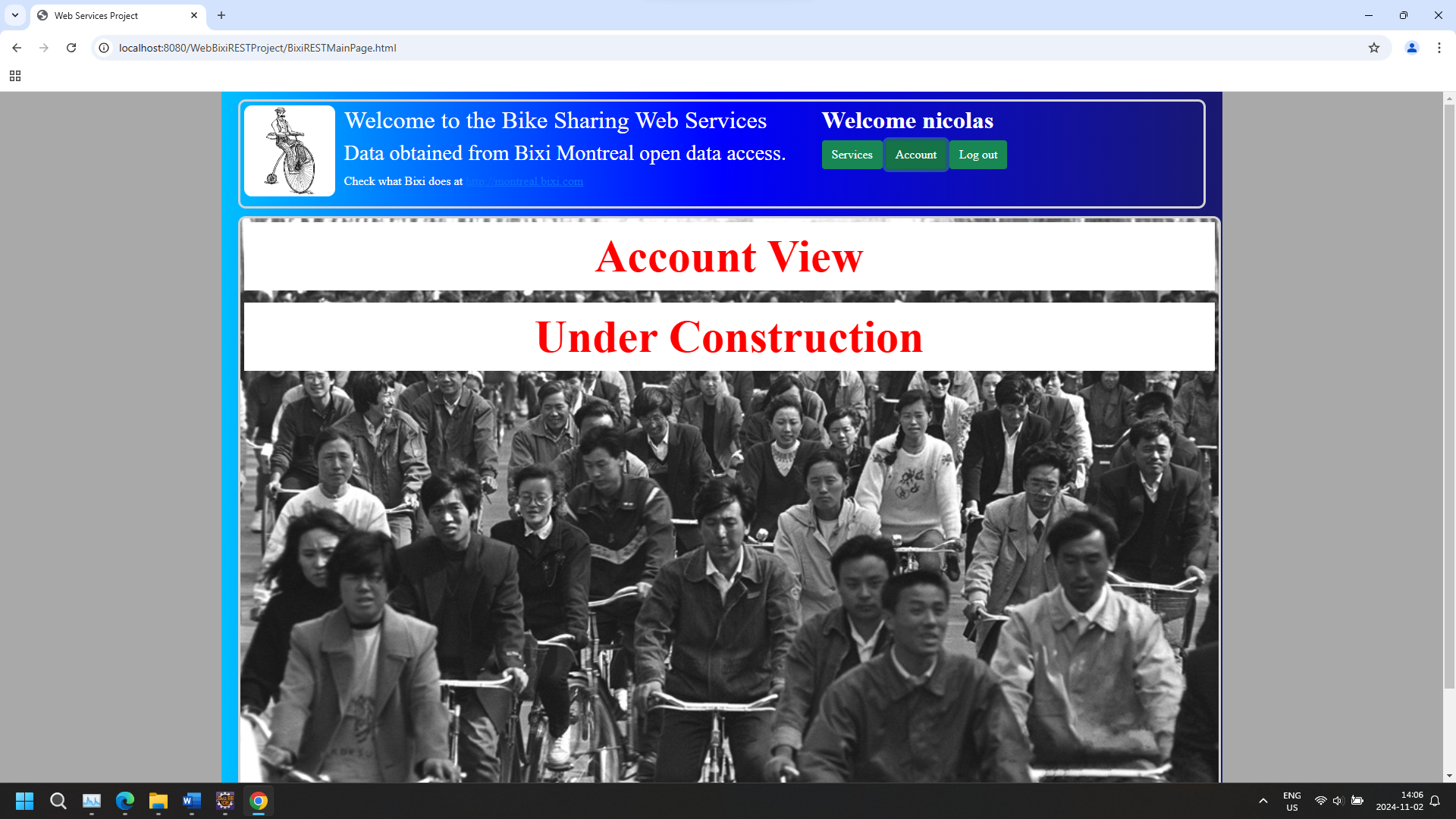


Figure 8. BixiRESTMainPage.html Web Page after Account Functionality Activated

The call of the functionalities in JavaScript managing user login and account is illustrated in   
Figures 9 and 10.

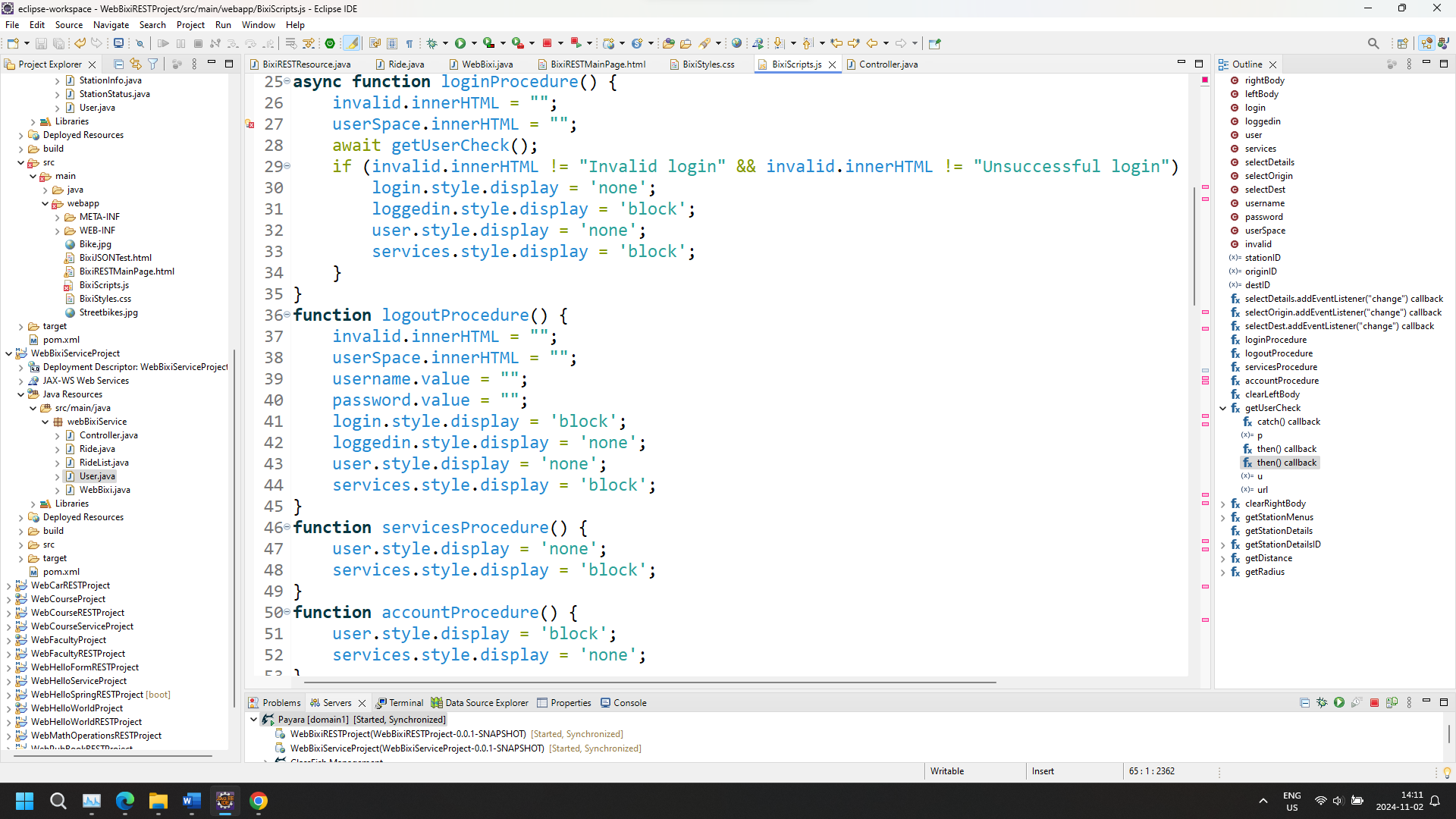


Figure 9. Login, Logout, Services and Account JavaScript Functions

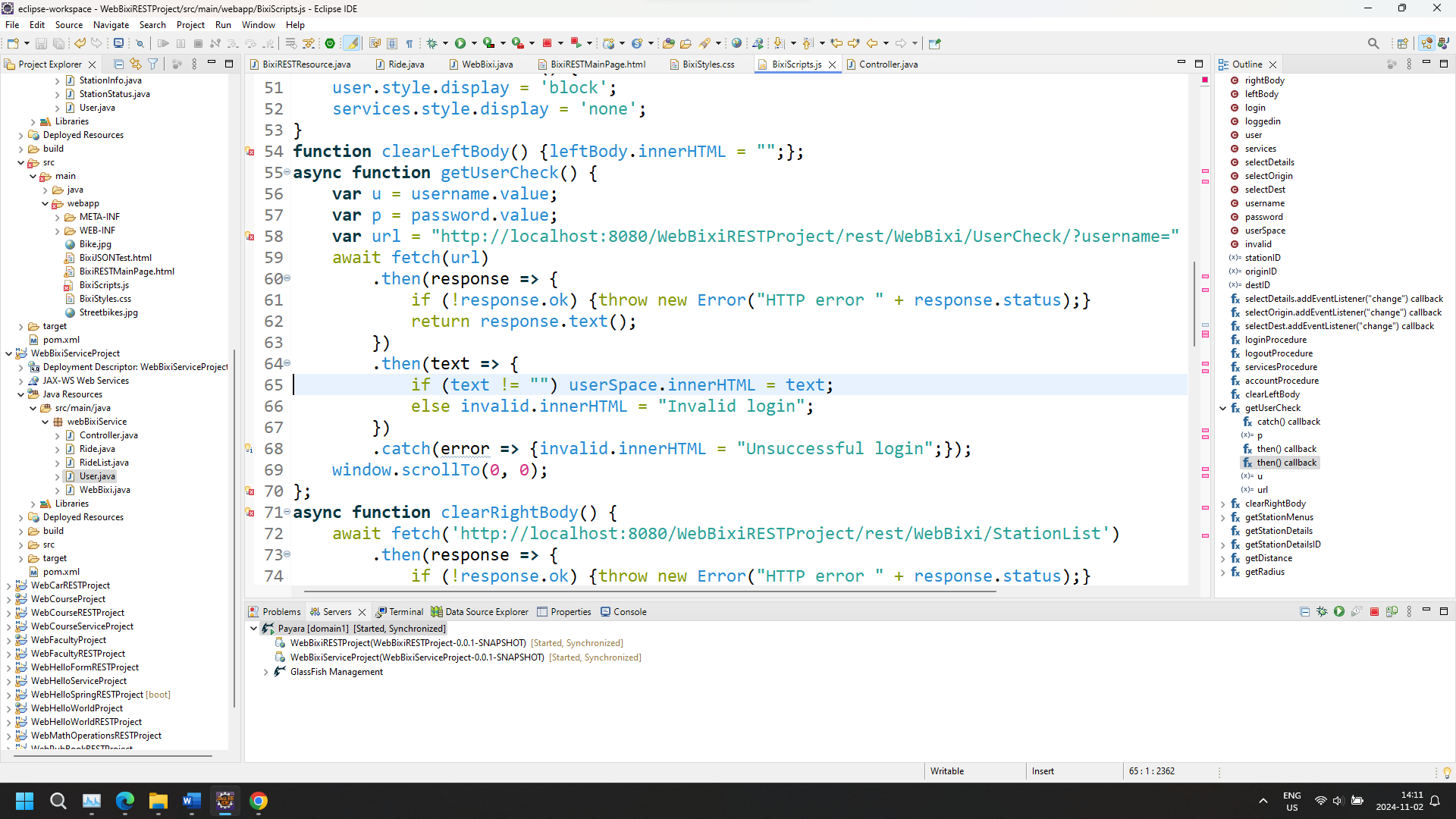


Figure 10. Call of the Web Service for User Checkup at Login

### 3.1.2. Station Details

As a reminder, that web service is available whether or not a user is logged in.

Details for a given stations can be done by two means:

* Clicking on a station in the list in the right column of the third row of the web page;
* Selecting a station from the dropdown selection in the Details tab in the second row of the web page.

The station details then appear on the left column of the third row of the web page, as illustrated in Figure 11. A Clear button allows the user to remove these details.

The call of that web service is established in JavaScript, as illustrated in Figure 12.

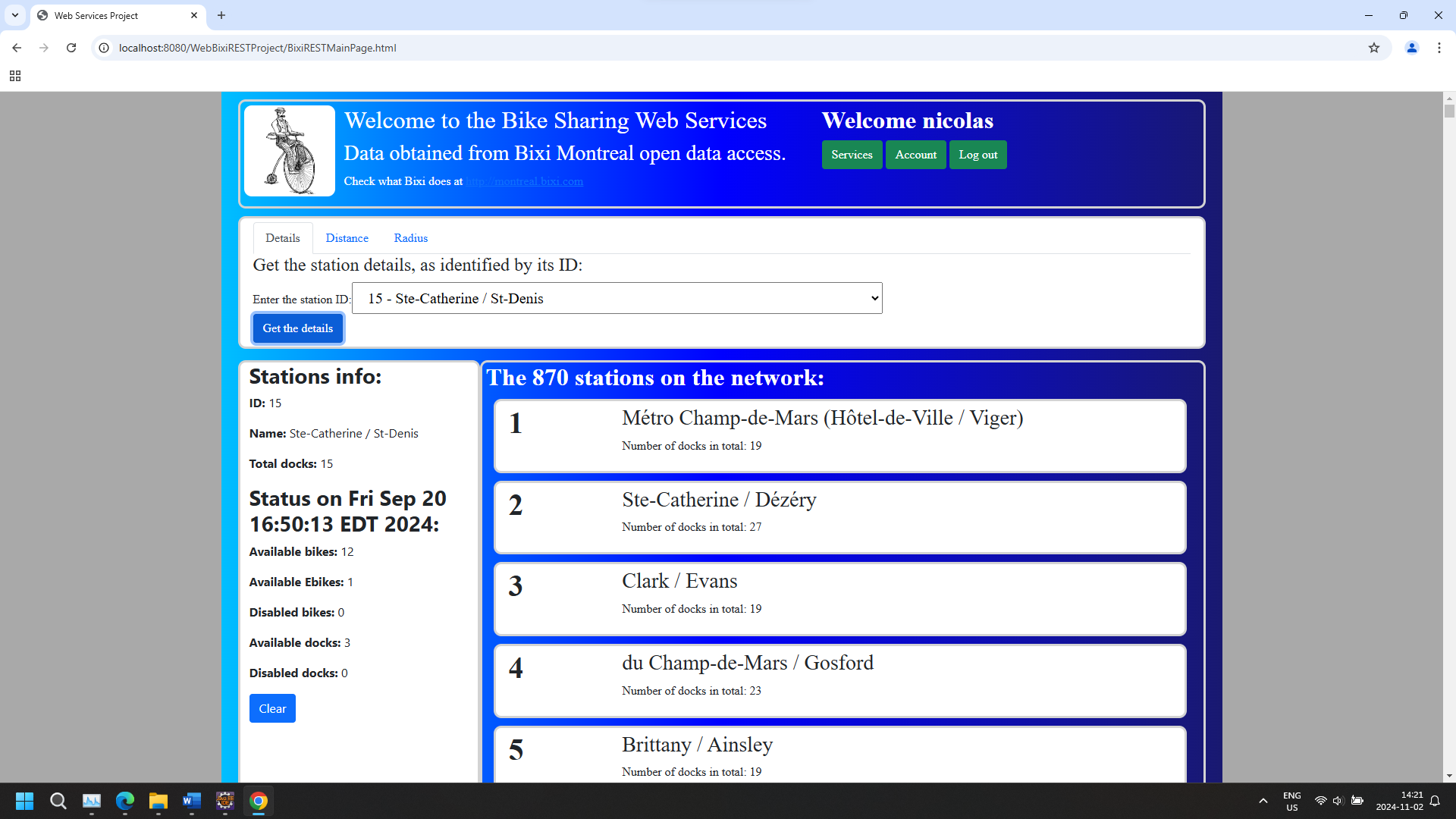


Figure 11. Show – Station Details – REST Web Service



Figure 12. Call – Station Details – REST Web Service

### 3.1.3. Distance Between Two Stations

As a reminder, that web service is available whether or not a user is logged in.

The distance is calculated between two stations selected from the dropdown lists in the Distance tab of the second row of the web page. An example of results showing on the left column of the third row of the web page is illustrated in Figure 13.

The call of that web service is established in JavaScript, as illustrated in Figure 14.



Figure 13. Show – Distance Estimate Between Two Station – REST Web Service



Figure 14. Call – Distance Between Two Stations – REST Web Service

### 3.1.4. Stations Within the Radius of a Given Location

As a reminder, that web service is available whether or not a user is logged in.

By entering the latitude and longitude of a location, and a radius in meters from the fields in the Radius tab in the second row of the web page, a focused list of stations is created and shown in the right column of the third row of the web page, as illustrated in Figure 15.

The call of that web service is established in JavaScript, as illustrated in Figure 16.

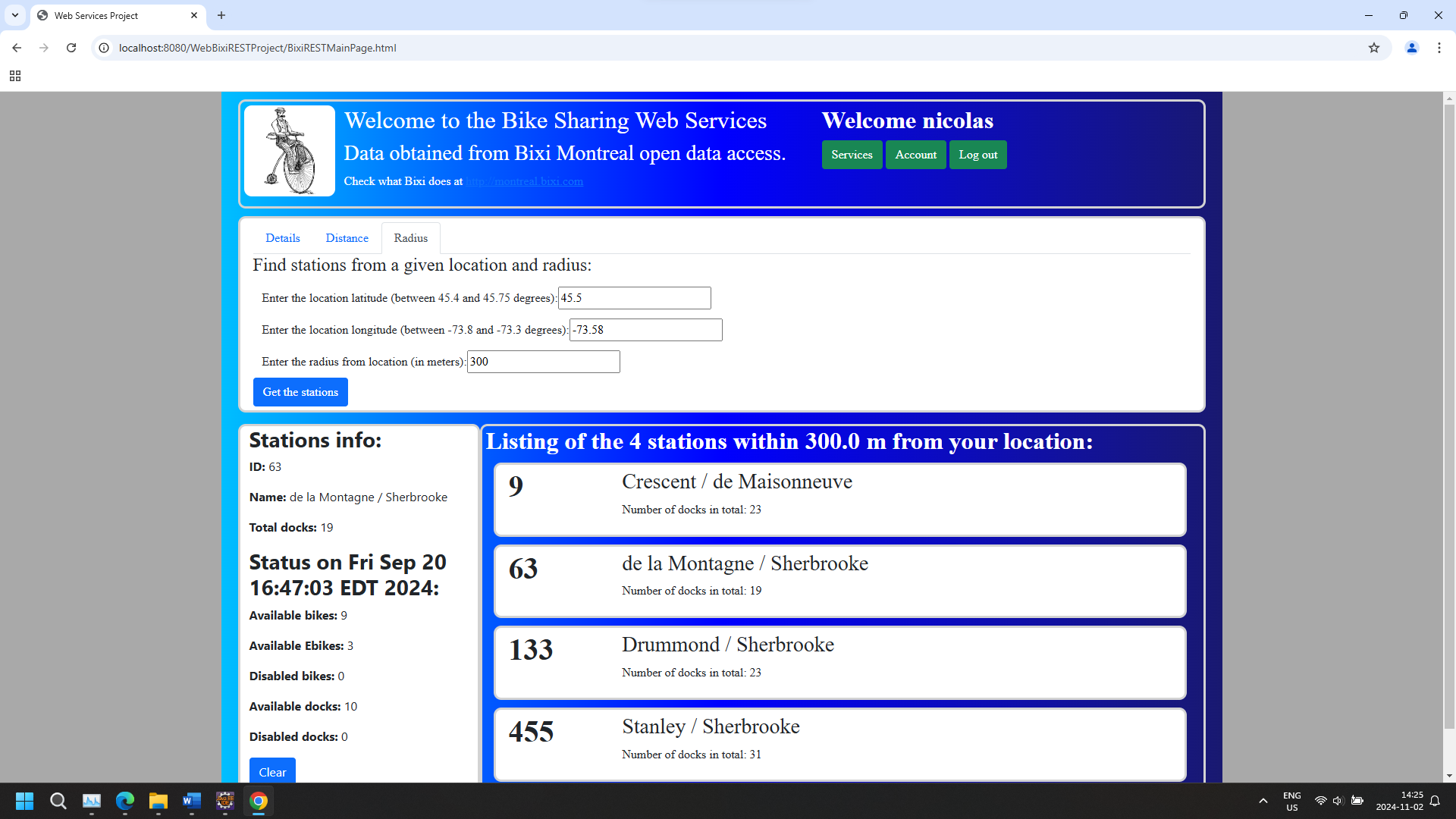


Figure 15. Show – Station List from a Given Location, Within a Given Radius

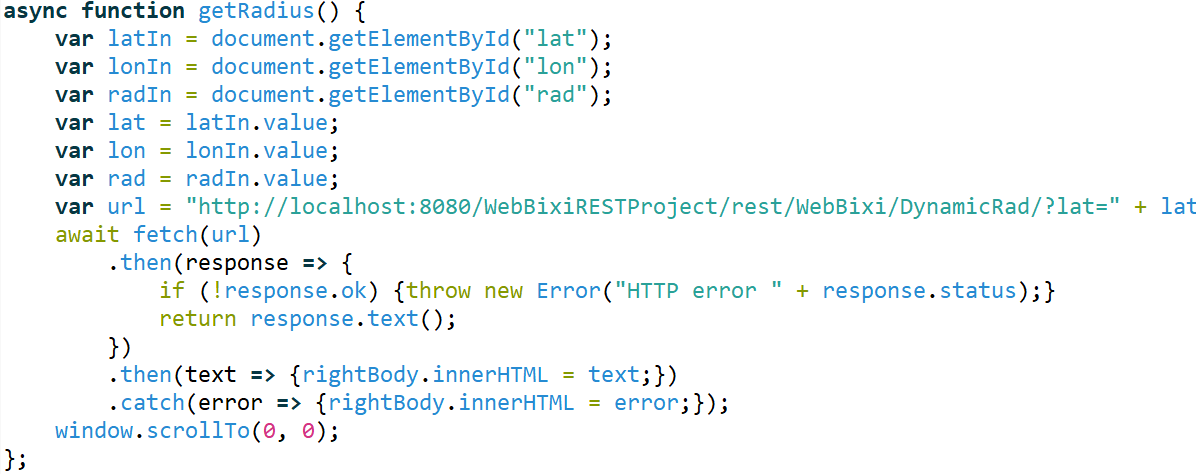


Figure 16. Call – Station List from a Given Location, Within a Given Radius

### 3.1.4 JSON Handling

As indicated in Section 2.2, the data supporting the web services of the project are obtained from:

* https://gbfs.velobixi.com/gbfs/en/station\_information.json
* https://gbfs.velobixi.com/gbfs/en/station\_status.json

As an experiment, parsing has been applied in web services in the BixiJSONResources.java file to extract the data from the JSON objects obtained from these two links. These services, all defined for application in a GET request, are described in Table 8.

The front end running the web services from BixiJSONResources can be accessed from:

http://localhost:8080/WebBixiRESTProject/BixiJSONTest.html

Table 8. REST Web Services for JSON Data Extraction

| **Description** | **Called Method** |
| --- | --- |
| Station info data extraction |  |
| Station details data extraction |  |

The BixiJSONTest.html web page provides a form to call these two web services, and includes a dynamic element to show the results from them. The call of these web service is established in JavaScript within the html file, as illustrated in Figure 17.

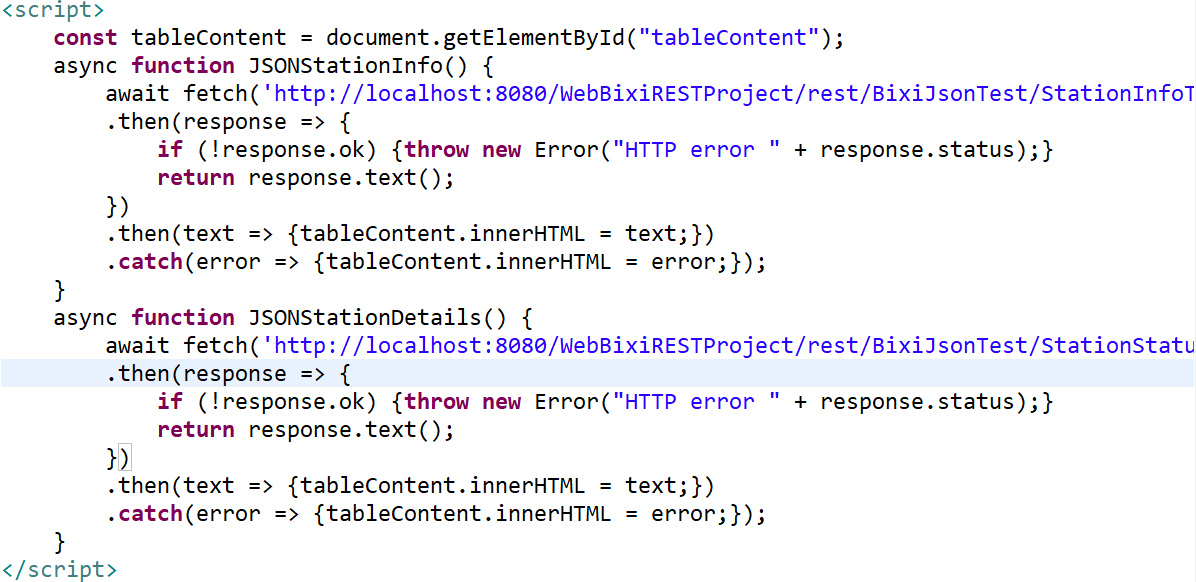


Figure 17. Calls – Data Extraction from JSON Parsing – REST Web Service

The results of these JavaScript calls are showing in Figures 18 and 19, for the data extraction of JSON station info and details, respectively.

As stated in Section 2.2, the program uses hardcoded data for the REST web services detailed in Sections 2 and 3.1. Using data extracted from JSON objects obtained from Bixi is an objective of a subsequent phase of the project. It is the expectation that data extraction from JSON objects can be achieved from SOAP implementations, with the data saved in memory in the server for use by the REST web services afterwards.

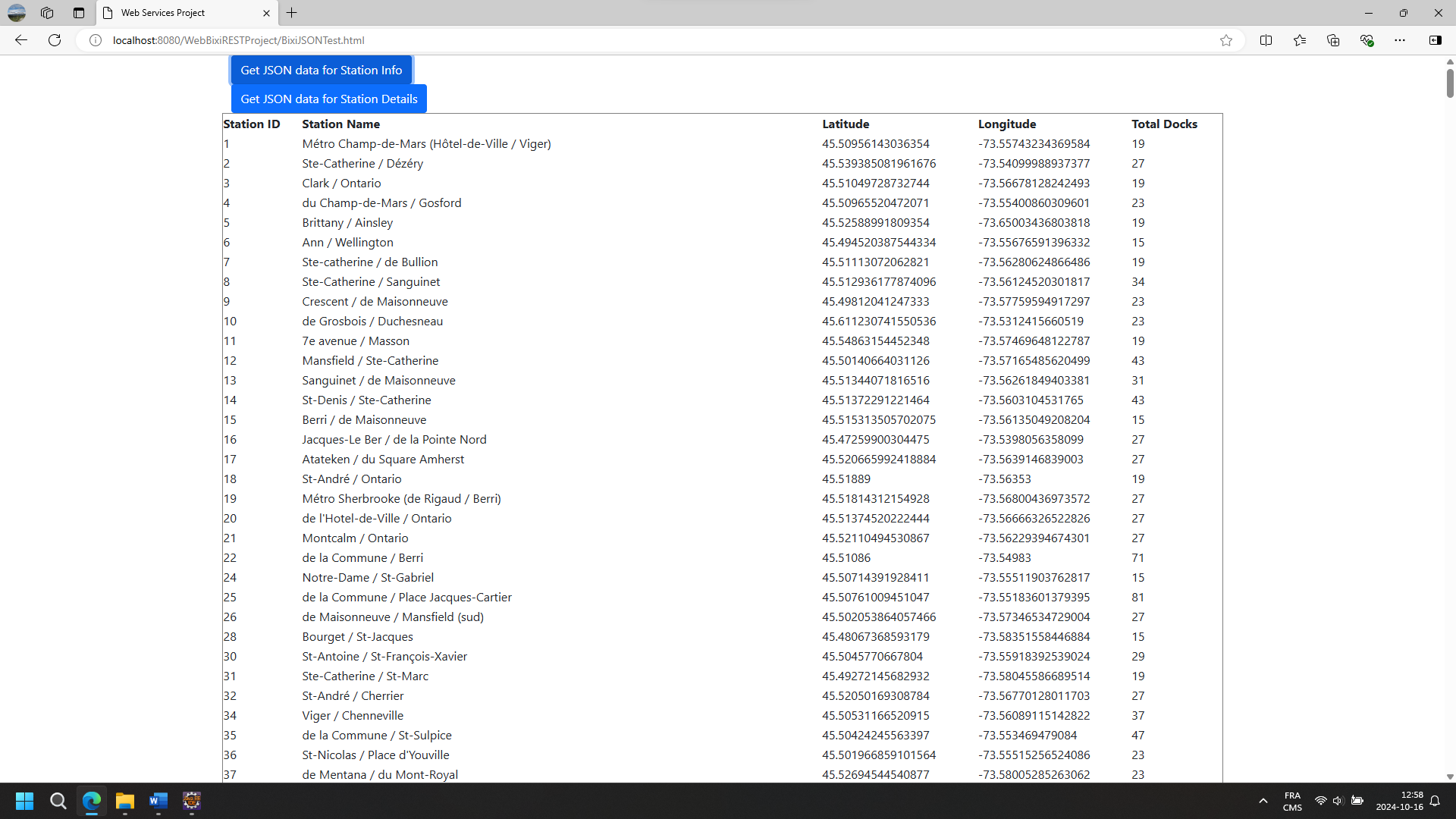


Figure 18. JSON Parsing of the StationInfo Data – REST Web Service

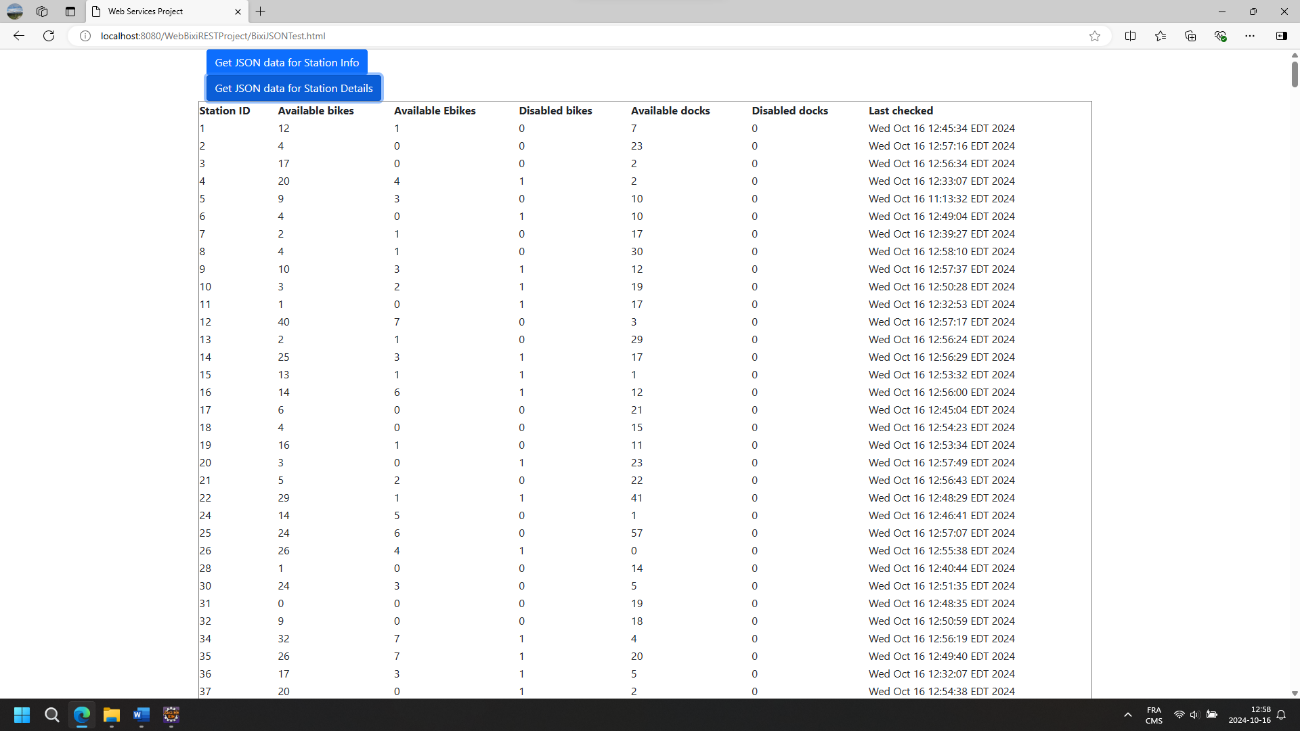


Figure 19. JSON Parsing of the StationDetails Data – REST Web Service

## 3.2. SOAP Implementations

Writing of the body of the requests made in Postman to consume SOAP services are supported by the wsdl (web service description language) information obtained from the WebBixi deployment on the server, and shown in Figure 20.

Postman is employed for viewing the results of the SOAP services for now; however, it is the ultimate objective to offer a view of these results within the BixiRESTMainPage.html web page.

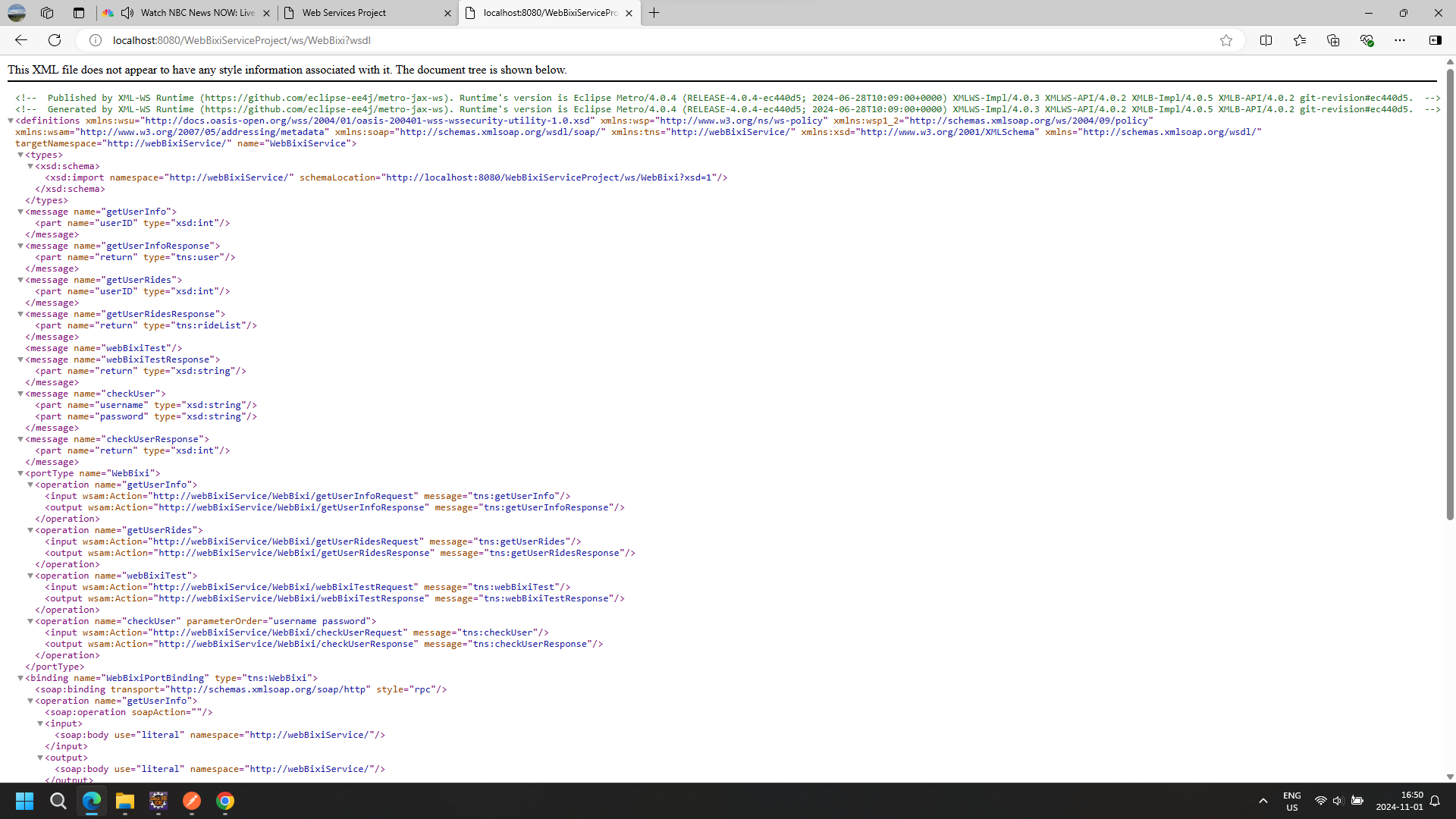


Figure 20. WebBixi Web Service Description Language Information

### 3.2.1 Login Method

This method is intended to validate the credentials entered by a person on the login area, returning a valid user ID if the credentials are valid, or an invalid user ID (-1) if invalid. With valid credentials, user account information can be made available at the user request, searchable from the user ID. The login area remains in place, waiting for valid credential otherwise. A valid result is shown in Figure 21.

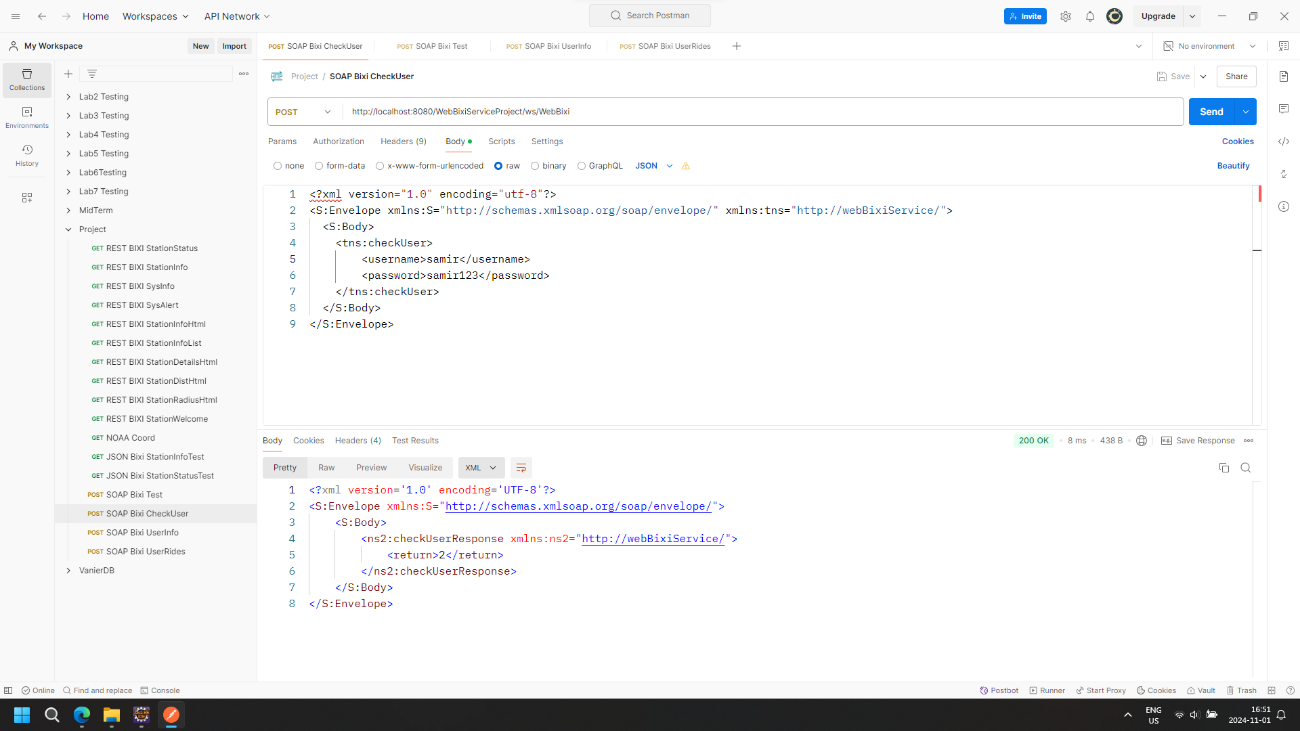


Figure 21. SOAP Login Method Application on Postman

### 3.2.2 User Info Method

This method allows a user to access their information, as stored on the server. This method is only a read operation, but can ultimately be complemented with edit capabilities on a web page to conduct an update operation at the user’s request. An example of the results from that method is shown in Figure 22.

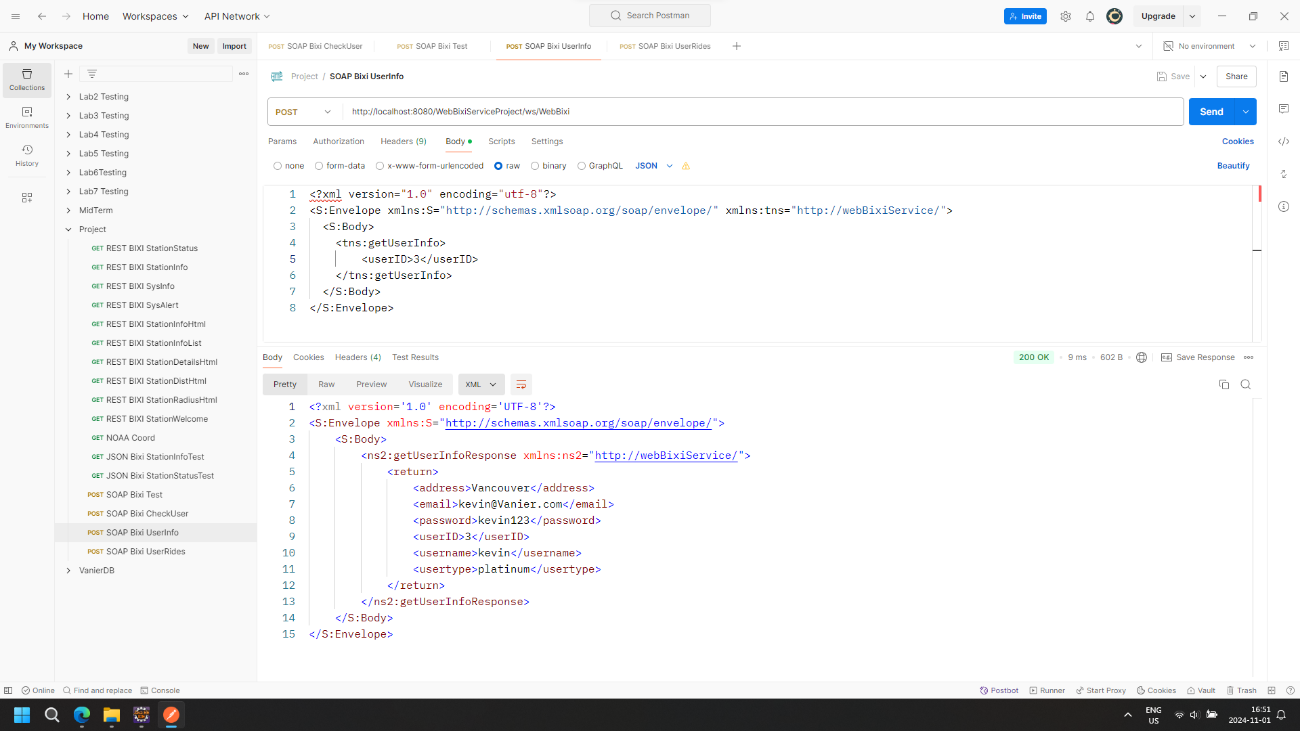


Figure 22. SOAP User Info Method Application on Postman

### 3.2.3 User Ride List Method

This method provides a user with a history of their usage of the bike sharing resources. Seeing a whole unfiltered list can be cumbersome. However, functionalities can ultimately be implemented on a web page to offer a filtered view of the results. An example of the results from that method is shown in Figures 23 to 26.

Rides are the equivalent of transactions performed by the user. A user may decide to terminate a membership to the bike sharing service, if their transactions are too few considering the recurring costs of maintaining that membership. The history of rides may also be used to check fraudulent transactions (i.e., rides that have not been performed by the user).

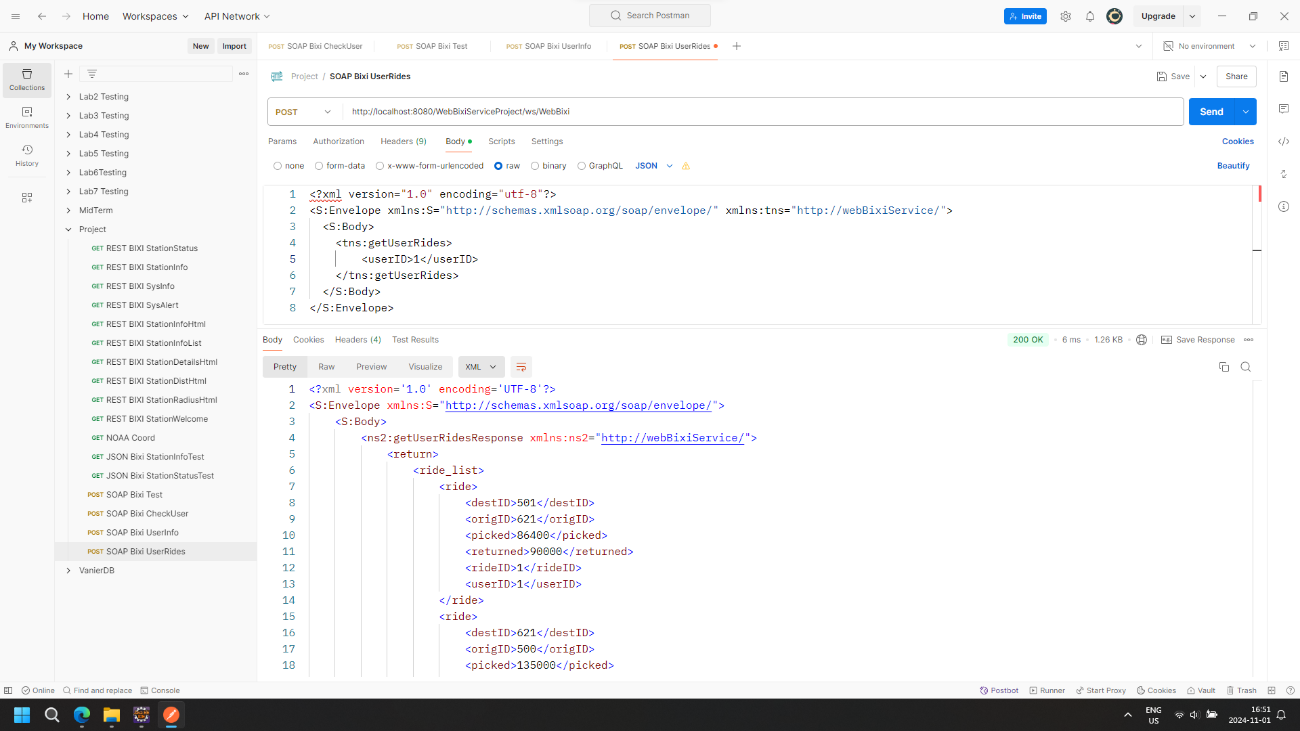


Figure 23. SOAP Ride List Method Application on Postman, Part 1 of 4

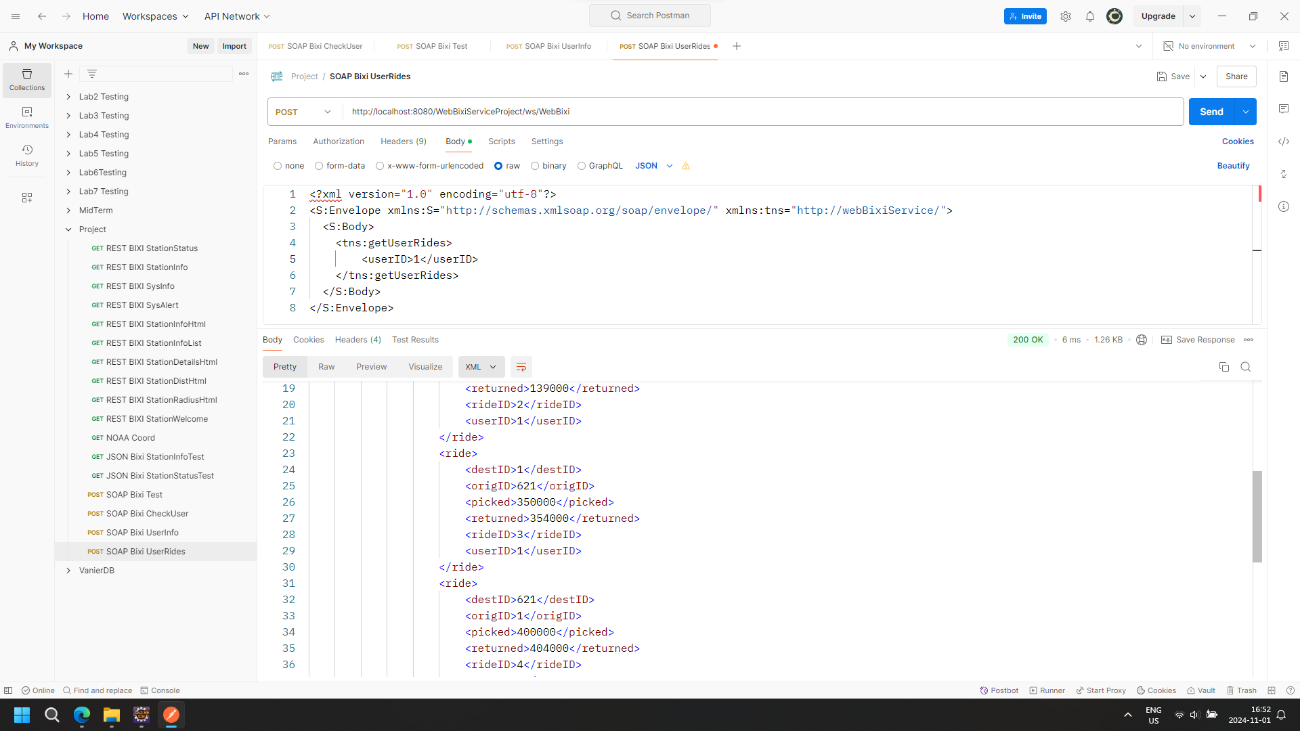


Figure 24. SOAP Ride List Method Application on Postman, Part 2 of 4

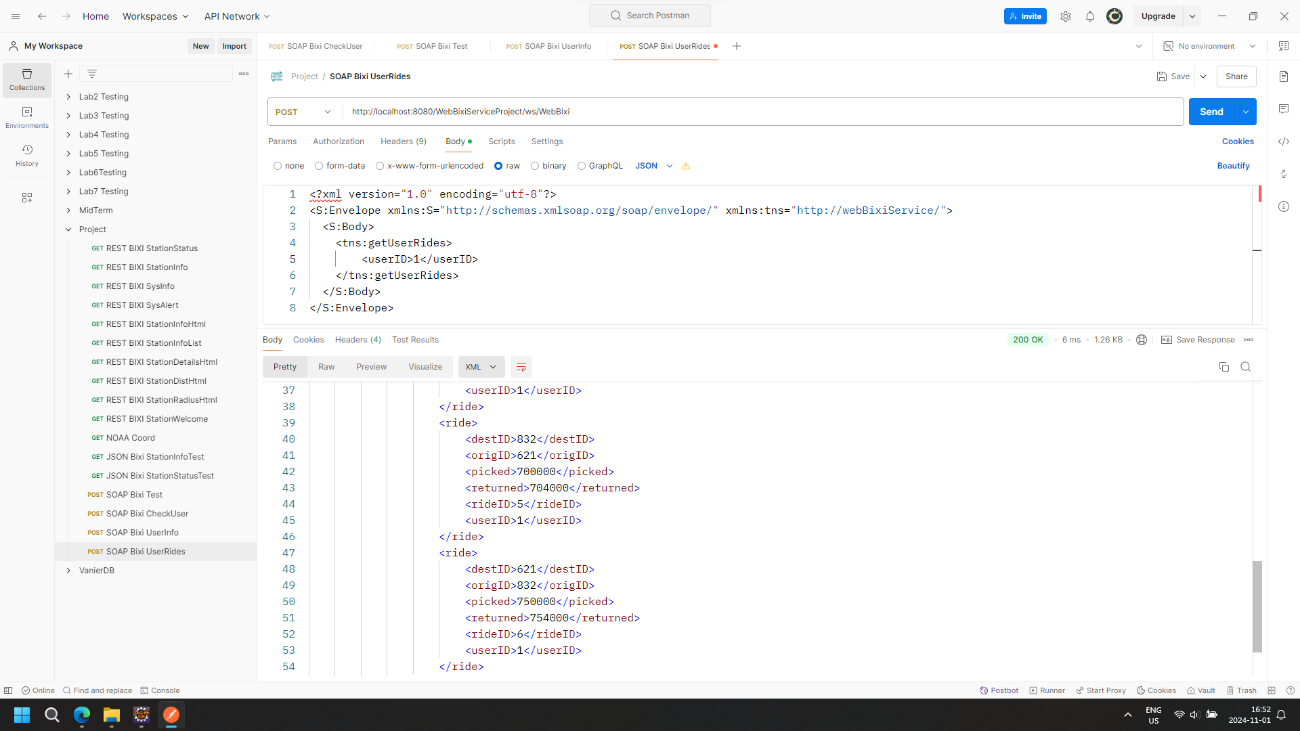


Figure 25. SOAP Ride List Method Application on Postman, Part 3 of 4

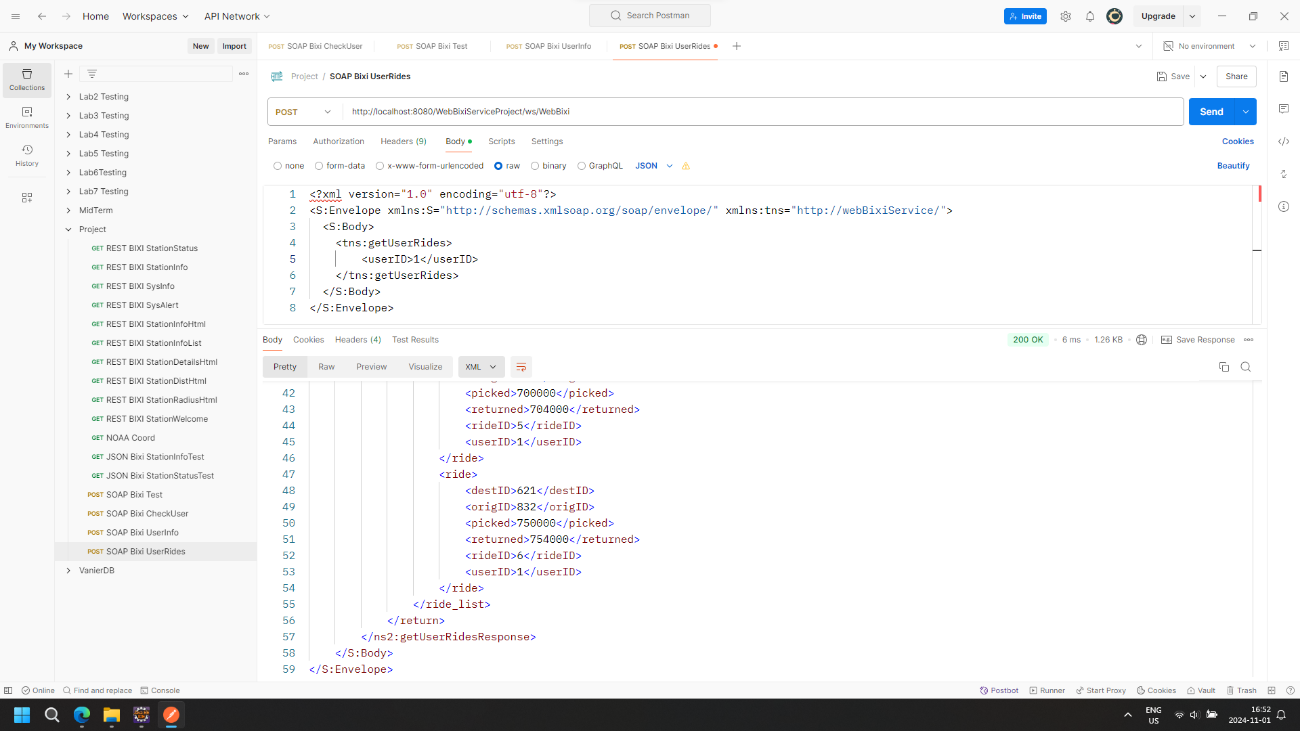


Figure 26. SOAP Ride List Method Application on Postman, Part 4 of 4

# 4. CLIENT APPLICATION

As a test, a client application has been built to demonstrate the access of the web services detailed in Sections 2 and 3.1, from outside of the server. The results of the test are shown in Figures 27 and 28.

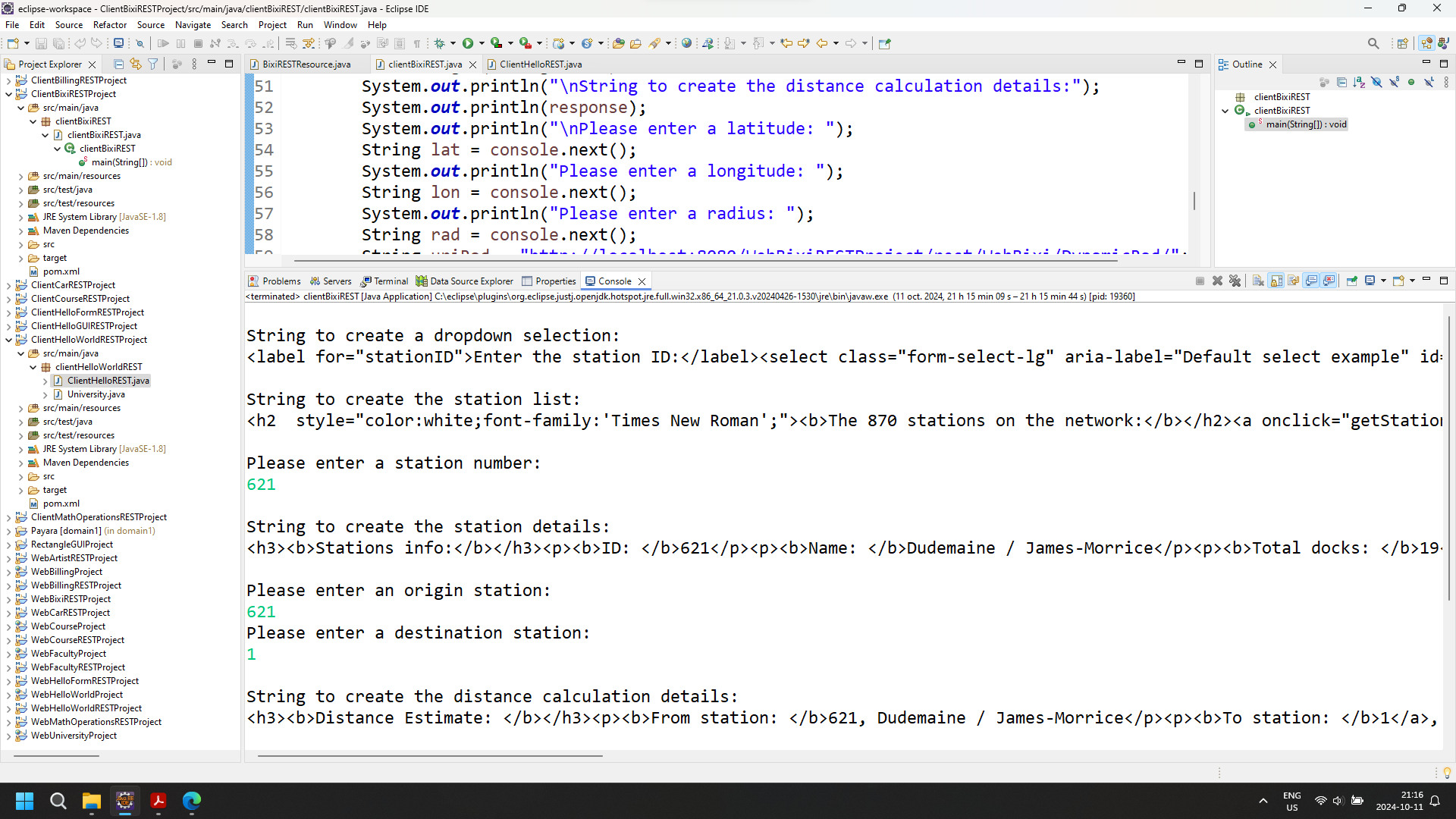


Figure 27. Client Application Consumption Results, Part 1 of 2

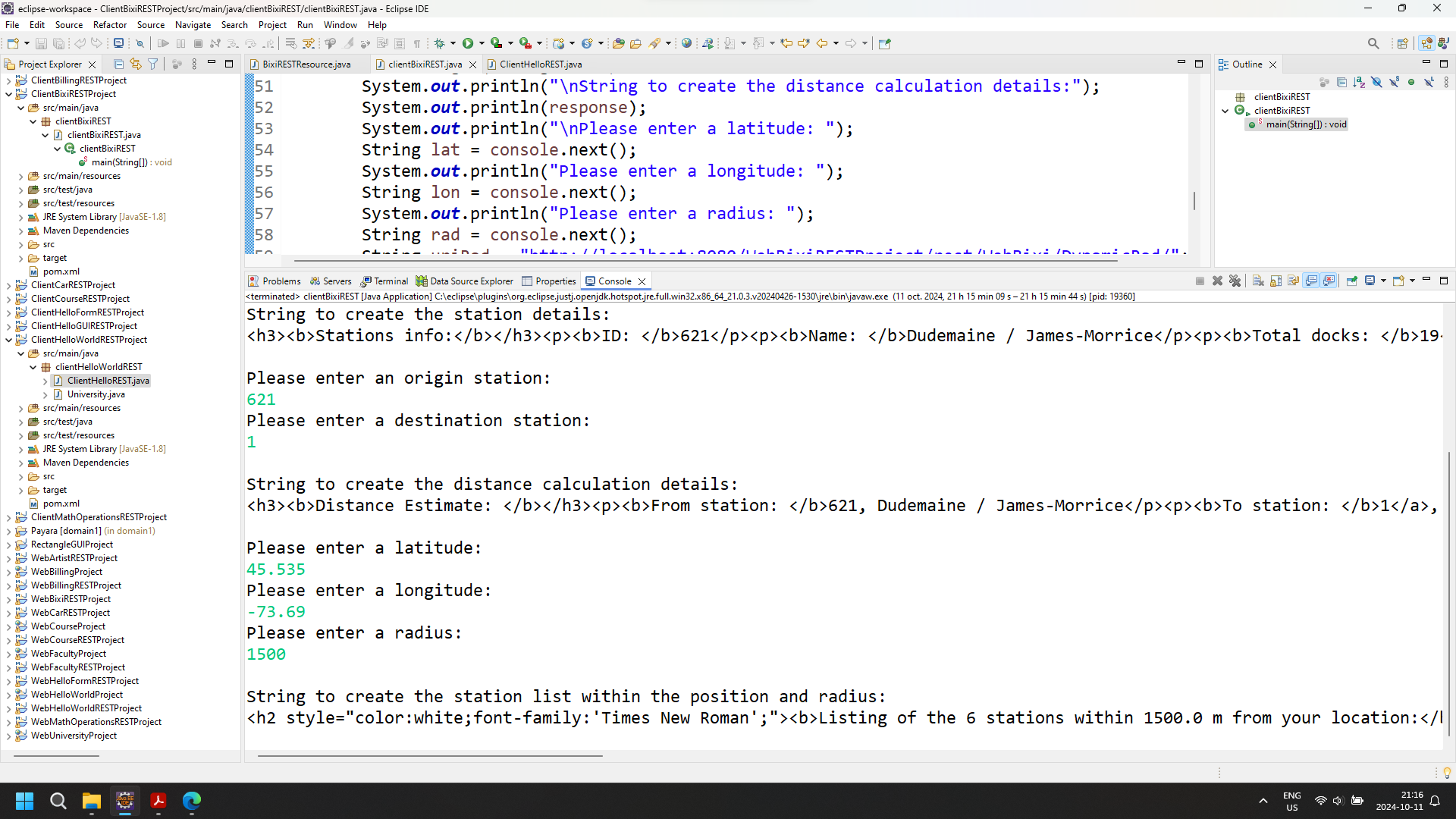


Figure 28. Client Application Consumption Results, Part 2 of 2

# 5. CONCLUSION

This work has demonstrated the ease of rapidly implementing web services to support the business objective of a bike sharing enterprise. This ease of implementation can be applied to many business objectives, for many types of enterprises. This work emphasizes the strategy in software development to start with basic concepts and progressively refine trough multiple development cycles.

In the context of this work, it is agreed that the use of SOAP implementation is rather limited. Subsequent development of this work can entail the viewing the results of SOAP services into web pages, as done for REST web services.