Department of Computer Engineering
Bilkent University

CS 458 Project #3

Project Report

İdil YILMAZ - 21703556
Doğukan Ertunga KURNAZ - 21702331
Sarp Ulaş KAYA - 21801992
Berk Kerem BERÇİN - 21803190

Instructor: Haluk Altunel
Teaching Assistant(s): Cihan Erkan

April 26, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Software Verification and Validation course CS458.

1 Introduction

As a project for learning Test Driven Development (TDD), as a team of four, we built a simple web application. The aim was to follow TDD steps and develop an application that finds the country of a given location, calculates the device's distance to the Geometric North Pole and calculates the distance of the device (or a user-determined location) to the Moon's core.

In other projects of the course, we first built the applications and then wrote tests in order to make sure that the applications run correctly. However, in TDD, we should convert the requirements to test cases before developing the application. Then, we should write the simplest code that passes the chosen test case. Repeating this process for all test cases results in an application that passes all the test cases and therefore satisfies the requirements.

Firstly, based on the requirements, we thought of several tests which can make sure that the application runs error-free when they are passed. Then, we select test cases one by one, write test code for the test with Selenium, and write minimal code pieces that can make the test pass. After we make sure that the code passes the test, we refactor the code in order to make it readable and get rid of duplications. This process is repeated for all of the test cases and finally, we had a working application that satisfies the requirements.

The following sections include diagrams and implementation details of our application, the test cases we chose in order to satisfy the requirements along with their details and implementations, and a general evaluation of our first Test Driven Development experience.

2 UML Diagrams

2.1 Activity Diagram

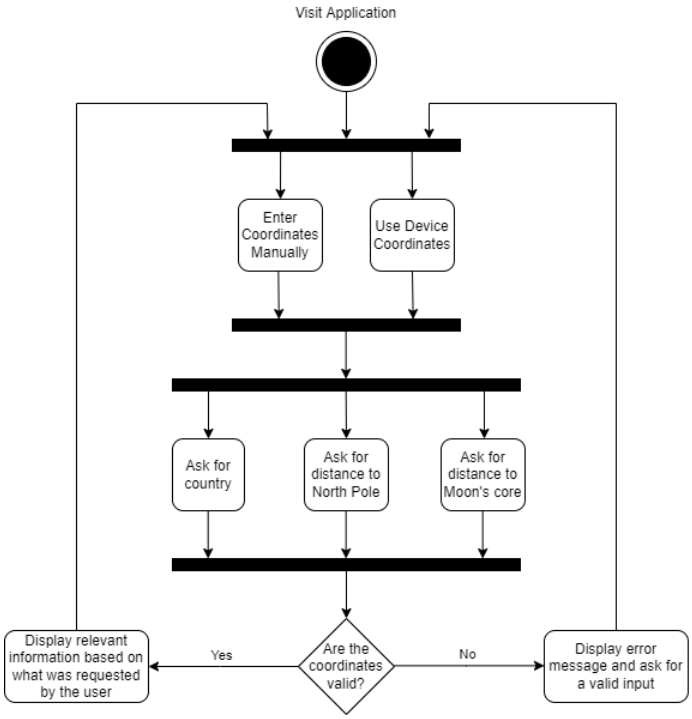

Figure 1: Activity Diagram.

2.2 State Diagrams

2.2.1 Find Country

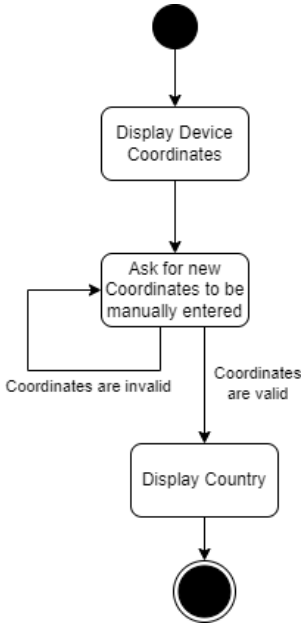

Figure 2: State Diagram of finding the correct country based on user inputs.

2.2.2 Display Distance to the North Pole

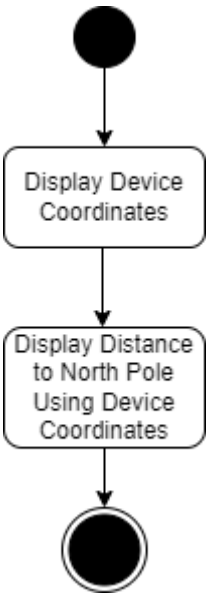

Figure 3: State Diagram of finding the distance to the North Pole using device GPS.

2.2.3 Display Distance to Moon's Core

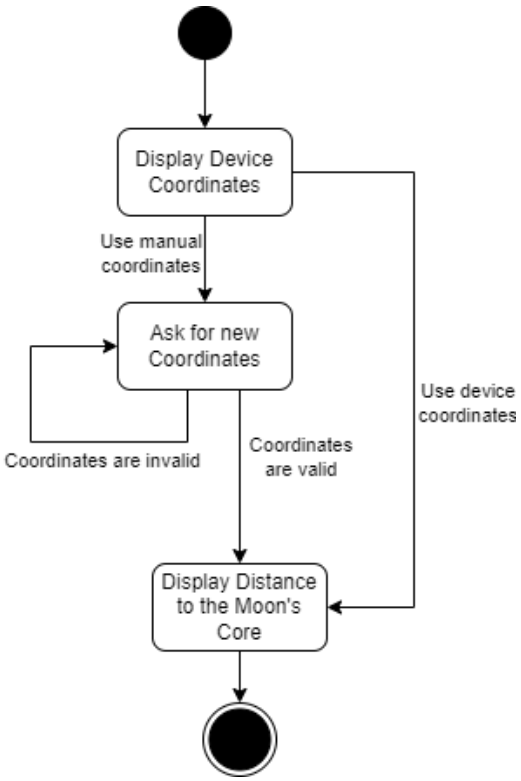

Figure 4: State Diagram of finding the distance to the Moon's core using device GPS or user inputs.

2.3 Use-Case Diagram

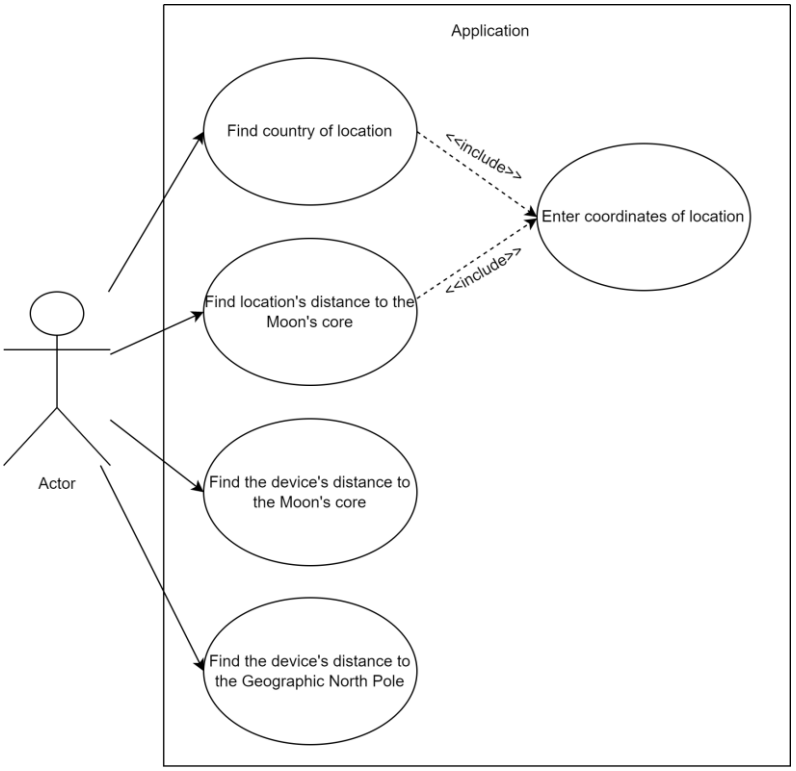

Figure 5: Use-Case Diagram.

2.4 Sequence Diagrams

2.4.1 Find Country

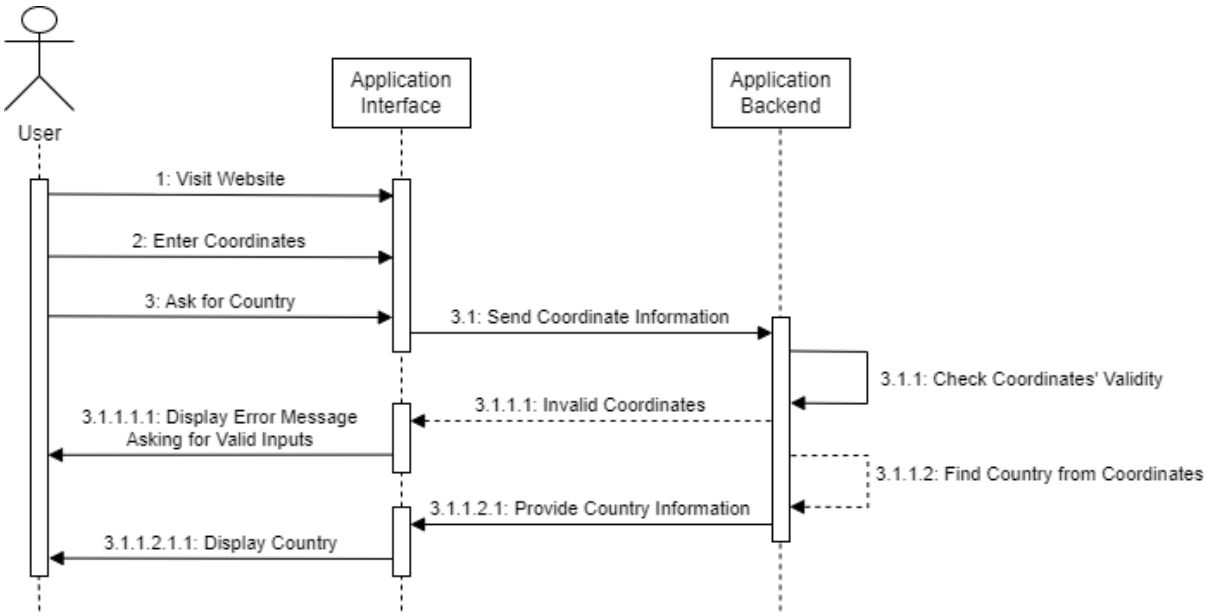

Figure 6: Sequence Diagram of finding the correct country based on user inputs.

2.4.2 Display Distance to the North Pole

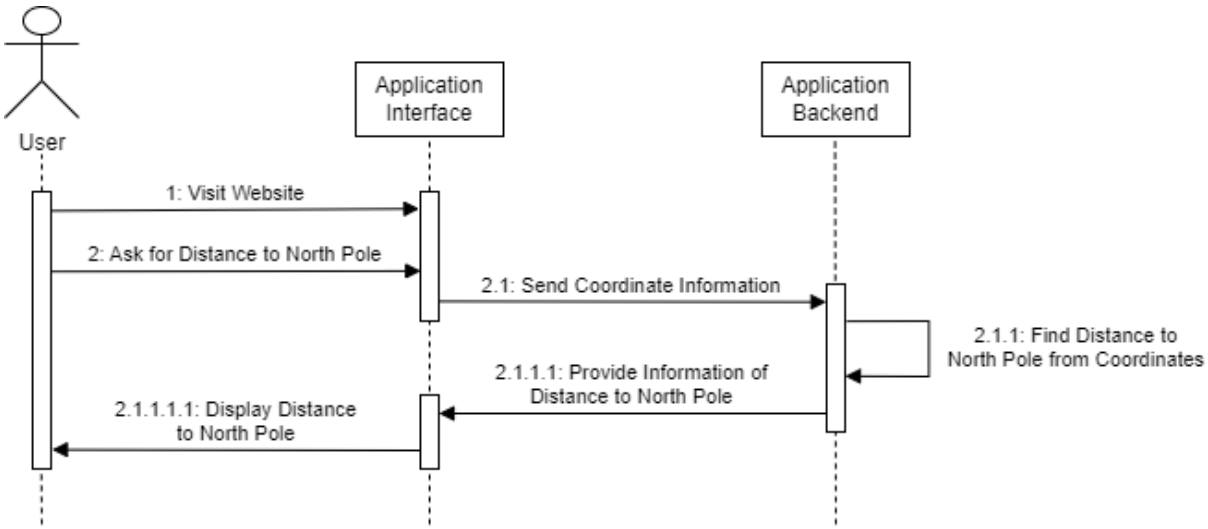

Figure 7: Sequence Diagram of finding the distance to the North Pole using device GPS.

2.4.3 Display Distance to Moon's Core

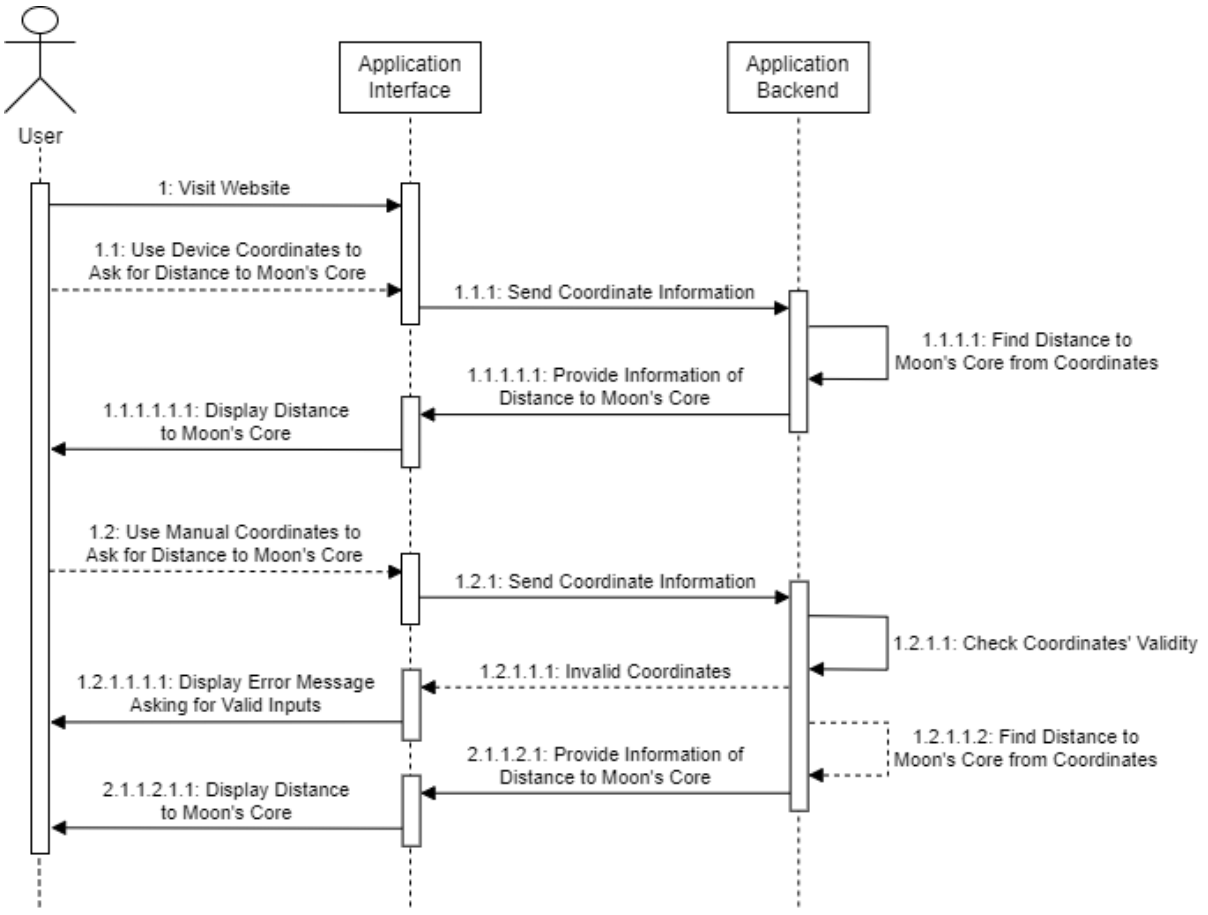

Figure 8: Sequence Diagram of finding the distance to the Moon's core using device GPS or user inputs.

3 Implementation Details

3.1 Screenshots of the Application UI

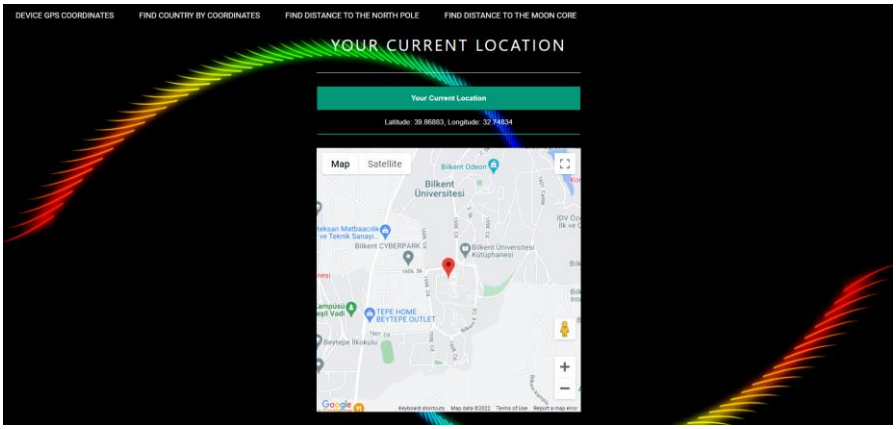

Figure 9: User's current location displayed on the application UI.

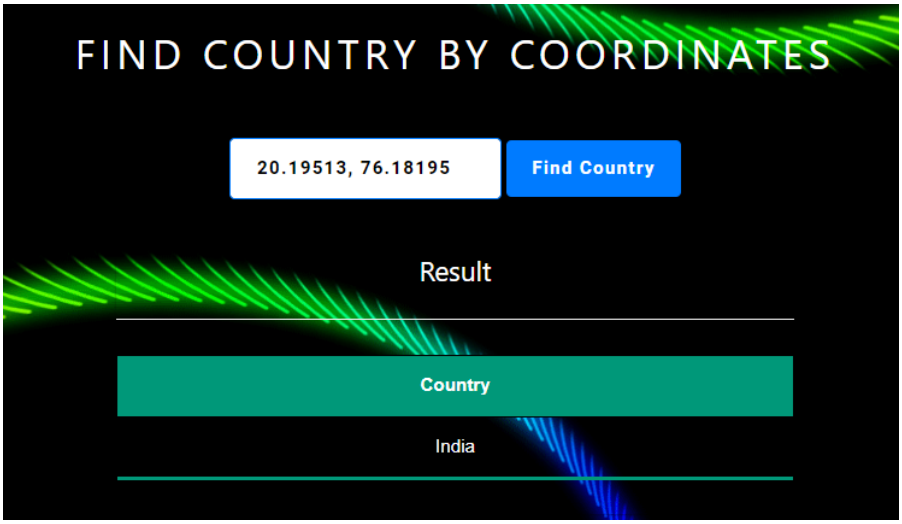

Figure 10: Results of Find Country functionality displayed based on the user inputs.

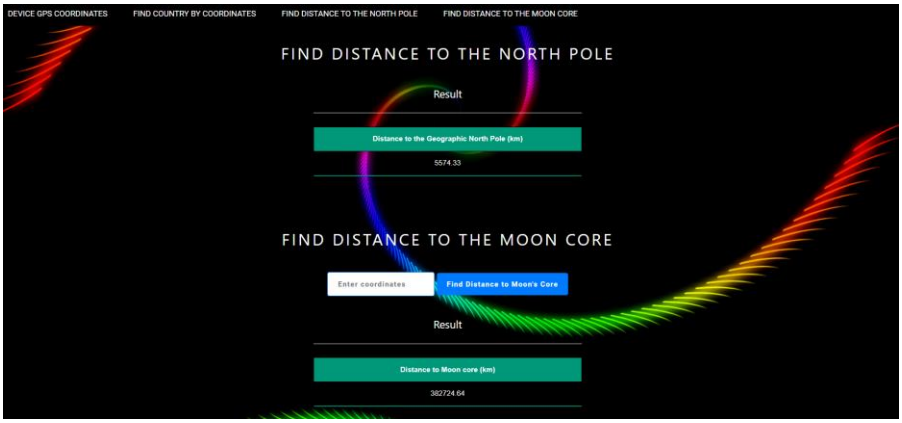

Figure 11: Distances to the North Pole and to the Moon's core are automatically displayed based on device coordinates.

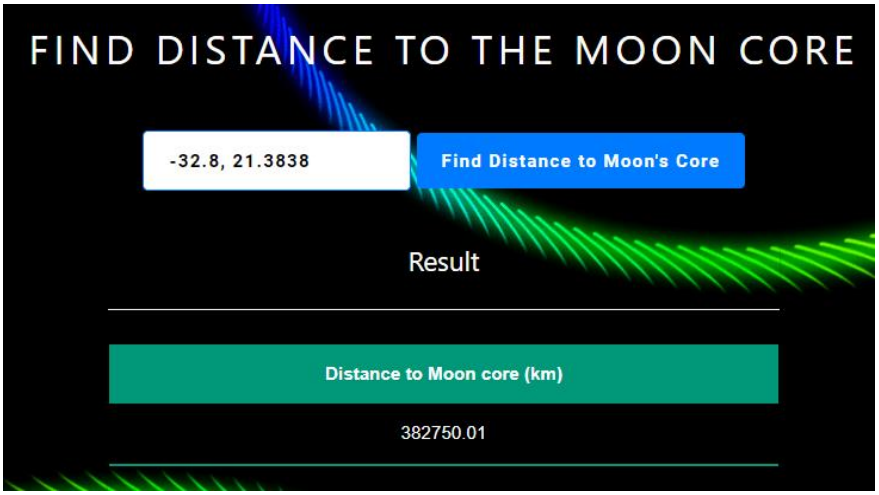

Figure 12: Distance to the Moon's core displayed based on user-specified coordinates.

3.2 Code Refactors

3.2.1 Find Country

We had implemented a function called findCountry that used Geoapify's Geocoding API [1] to pinpoint what country the coordinates that it took as function inputs belonged to.

```
//calculate the coordinates country
function findCountry(cor) {
  //if cor has no comma, error
  if (cor.indexOf(",") == -1) {
    alert("Error: Please enter valid coordinates (latitude, longitude)");
    return;
  }
  //console.log(cor);
  //split cor to lat and lng
  var lat = cor.split(",")[0];
  var lon = cor.split(",")[1];
  //google's reverse geocode api is paid so i used different one
  fetch(`https://api.geoapify.com/v1/geocode/reverse?lat=${lat}&lon=${lon}&apiKey=9a14581af46544eb9f289e74d579ce10`)
    .then(response => response.json())
    .then(result => {
      if (result.features.length) {
        console.log(result.features[0].properties.country);
        //insert the country name to the result div
        document.getElementById("result_country").textContent = result.features[0].properties.country;
      } else {
        console.log("No address found");
      }
    });

}
```

Figure 13: Old version of the Find Country function before testing and refactors.

During our tests of this functionality, we found out that while the calculations were correct and the correct country was displayed as a result for each valid input, what this function lacked was the handling of certain edge cases. Specifically, the first of these was the case of inputs that were separated by a comma (,) as the expected input had

to be, but were not coordinates. For instance, an input such as “abc, def” did not generate the expected error message and the function still tried to find which country it belonged to. The second group on the other hand included the type of inputs that did resemble valid coordinates, but were not as they were out of the possible range of coordinates. For instance an input such as “-1934.129341, 2477.967402” did not generate an error message as it was supposed to, being outside of the [-90, 90] range for the latitude and the [-180, 180] range for longitude. To fix these issues, we implemented additional if statements to handle these cases and display the appropriate error messages without executing the rest of the function.

```
37  //calculate the coordinates country
38  function findCountry(cor) {
39    //if cor has no comma, error
40    if (cor.indexOf(",") == -1) {
41      alert("Error: Please enter valid coordinates (latitude, longitude)");
42      return;
43    }
44    //if lattitude or longitude is not a number, error
45    if ( isNaN(cor.split(",")[0]) || isNaN(cor.split(",")[1])) {
46      alert("Error: Please enter valid coordinates (latitude, longitude)");
47      return;
48    }
49    if ( (cor.split(",")[0] < -90) || (cor.split(",")[0] > 90) || (cor.split(",")[1] < -180) || (cor.split(",")[1] > 180) ) {
50      alert("Error: Invalid range");
51      return;
52    }
53    //console.log(cor);
54    //split cor to lat and lng
55    var lat = cor.split(",")[0];
56    var lon = cor.split(",")[1];
57    //google's reverse geocode api is paid so i used different one
58    fetch(`https://api.geoapify.com/v1/geocode/reverse?lat=${lat}&lon=${lon}&apiKey=9a14581af46544eb9f289e74d579ce10`)
59      .then(response => response.json())
60      .then(result => {
61        if (result.features.length) {
62          console.log(result.features[0].properties.country);
63          //insert the country name to the result div
64          document.getElementById("result_country").textContent = result.features[0].properties.country;
```

Figure 14: Relevant excerpts of the new version of the Find Country function after testing and refactors.

3.2.2 Find Distance to the North Pole

Based on the results of our tests, we found no refactors to be necessary for this functionality.

3.2.3 Find Distance to Moon’s Core

Using the functions of the suncalc library [2] for JavaScript, we implemented a method to display the distance to the Moon’s Core from the given coordinates.

```
//find distance to the moon core
function findDistanceMoonCore(mooncor) {
  //if mooncor has no comma, error
  if (cor.indexOf(",") == -1) {
    alert("Error: Please enter valid coordinates (latitude, longitude)");
    return;
  }
  //first get the input coordinates then split them
  var lat = mooncor.split(",")[0];
  var lon = mooncor.split(",")[1];
  console.log(mooncor, lat, lon);
  //then calculate the distance to the moon core
  import('./suncalc.js').then(() => {
    const moon = SunCalc.getMoonPosition(new Date(), lat, lon);
    console.log(moon);
    document.getElementById("result_mooncore").textContent = moon.distance.toFixed(2);
  });
}
```

Figure 15: Old version of the Find Distance to Moon Core function before testing and refactors.

The problems with this method, as we later discovered thanks to our tests of the functionality, were the lack of sufficient error handling similar to that of the initial version of our findCountry method, and the inaccuracies of the library that we had used. The first of these had seemingly emerged as a result of us using the findCountry method as a template to change and build upon accordingly, lacking the error handling for the same cases of invalid and out-of-range inputs. As we had done for the findCountry method before, we simply added a couple of additional if statements to appropriately handle these types of inputs.

The second issue was more frustrating to deal with, as we did not know at first that it originated from the library itself rather than how we had used it. Only upon discovering the many issues opened inside the repository concerning the various mistakes related to the calculations of the library and its results being different than both the expected results of our test cases and the various sources on the internet [3, 4] did we realize this fact. Luckily, even though we now had to correct mistakes that we had not directly made, the issue with the calculation did not seem to be too complex, as the result was always a certain amount lower than the real value due to a factor that the author had seemingly forgotten, this amount being approximately 5330 kilometers. Adding this value to the library's results, we then saw during the tests that we were thus able to decrease the error of the calculations from about 2% to about 0.1-0.2%. Not only did we then introduce this change to the aforementioned function, which ran once the button to calculate the distance to the Moon's core from the given coordinates was clicked, but also to another function that ran once the website was visited in order to automatically display the distance to the Moon's Core from the device coordinates.

```
93  //find distance to the moon core
94  function findDistanceMoonCore(mooncor) {
95    //if mooncor has no comma, error
96    if (mooncor.indexOf(",") == -1) {
97      alert("Error: Please enter valid coordinates (latitude, longitude)");
98      return;
99    }
100   //if lattitude or longitude is not a number, error
101   if ( isNaN(mooncor.split(",")[0]) || isNaN(mooncor.split(",")[1])) {
102     alert("Error: Please enter valid coordinates (latitude, longitude)");
103     return;
104   }
105   if ( (mooncor.split(",")[0] < -90) || (mooncor.split(",")[0] > 90) || (mooncor.split(",")[1] < -180) || (mooncor.split(",")[1] > 180) ) {
106     alert("Error: Invalid range");
107     return;
108   }
109   //first get the input coordinates then split them
110   var lat = mooncor.split(",")[0];
111   var lon = mooncor.split(",")[1];
112   //console.log(mooncor, lat, lon);
113   //then calculate the distance to the moon core
114   import('./suncalc.js').then(() => {
115     const moon = SunCalc.getMoonPosition(new Date(), lat, lon);
116     //console.log(moon);
117     document.getElementById("result_mooncore").textContent = (moon.distance + 5330).toFixed(2);
118   });
119 }
```

Figure 16: New version of the Find Distance to Moon Core function after testing and refactors.

```
$info.textContent = `Latitude: ${lat.toFixed(5)}, Longitude: ${lng.toFixed(5)}`;
import('./suncalc.js').then(() => {
  const moon = SunCalc.getMoonPosition(curTime, globLat, globLng);
  var corrected = moon.distance + 5330;
  console.log("Current Time: " + curTime + "\n" + "Moon distance: " + corrected + "\n" + "Coordinates: " + globLat + "," + globLng);
  document.getElementById("result_mooncore").textContent = corrected.toFixed(2);
});
findDistanceNorthPole();
$info.classList.remove('error');
},
```

Figure 17: Relevant excerpts of the function that calculates the initial distance to the Moon's core based on the device coordinates after it was refactored.

4 Test Cases and Important Excerpts of Test Code

4.1 Basic Site Navigation and Interaction

4.1.1 Test Case Description

Before testing the relatively more specific functionalities of the web application, we agreed to make sure that the navigation elements and the interactive map worked correctly, since they are the first things that the user is greeted by once they visit the website and the test cases are relatively simple. Invoking a few clicks on certain elements of the interactive map and the navigation buttons proved to be sufficient, and not much was needed in terms of code to achieve the test results as they were clearly visible on the website during the tests unlike the rest of the functionalities that we tested later, which required more than observing the website due to the calculations that were involved.

4.1.2 Relevant Code Excerpts of the Test Case

```
        // MAP CONTROL TESTS
        WebElement zoomIn;
        WebElement zoomOut;

        WebElement fullScreenMap;

        zoomIn = driver.findElement(By.xpath("//*[@id=\"map\"]/div/div/div[13]/div/div[3]/div/button[1]"));
        zoomOut = driver.findElement(By.xpath("//*[@id=\"map\"]/div/div/div[13]/div/div[3]/div/button[2]"));

        fullScreenMap = driver.findElement(By.xpath("//*[@id=\"map\"]/div/div/div[8]/button"));

        sleepFor(1000);

        //fullscreen
        fullScreenMap.click();
        sleepFor(1500);
        fullScreenMap.click();
        sleepFor(1500);

        //Zoom in and out
        zoomIn.click();
        sleepFor(1500);
        zoomOut.click();
        sleepFor(1500);

        // PAGE NAVIGATION
        WebElement homeTab;
        WebElement countryTab;
        WebElement northPoleTab;
        WebElement moonCoreTab;

        homeTab = driver.findElement(By.xpath("/html/body/div[1]/div/a[2]"));
        countryTab = driver.findElement(By.xpath("/html/body/div[1]/div/a[3]"));
        northPoleTab = driver.findElement(By.xpath("/html/body/div[1]/div/a[4]"));
        moonCoreTab = driver.findElement(By.xpath("/html/body/div[1]/div/a[5]"));

        countryTab.click();
        sleepFor(1500);
        northPoleTab.click();
        sleepFor(1500);
        moonCoreTab.click();
        sleepFor(1500);
        homeTab.click();
        sleepFor(1500);
```

Figure 18: Script for testing the website's navigation tabs and interactive map.

4.2 Find Country

4.2.1 Test Case Description

As mentioned and shown earlier, the Find Country functionality requires the user to enter the coordinates manually, rather than using the device GPS which the website uses to display their current location. As a result, we decided to simply pinpoint a few locations throughout the world and note which countries they were in so that we could see whether Find Country gave the correct result when they were entered. Before doing so, however, we also tested the functionality with a few different types of invalid inputs to see whether it gave the proper error messages or not.

First, we tested the functionality with no inputs, leaving the field for the coordinates empty. Then, we ran the test with a series of various invalid inputs, not supposed to resemble coordinates. Then, we entered a set of invalid inputs that did resemble coordinates but were in an invalid range. These tests allowed us to see which types of inputs we had not accounted for while implementing the functionality, and we refactored our code accordingly afterwards. Finally, we ran the tests with the set of valid inputs mentioned earlier in order to see whether the correct results (countries) were obtained.

4.2.2 Relevant Code Excerpts of the Test Case

```
120         //FIND COUNTRY
121         WebElement coordinateField;
122         WebElement findCountryBtn;
123         WebElement countryResult;
124
125         coordinateField = driver.findElement(By.xpath("//*[@id=\"myText\"]"));
126         findCountryBtn = driver.findElement(By.xpath("//*[@id=\"results\"]/div[1]/button"));
127         countryResult = driver.findElement(By.xpath("//*[@id=\"result_country\"]"));
128
129         countryTab.click();
130         sleepFor(1500);
131
132         //valid and invalid coordinate inputs
133         findCountryBtn.click(); //no coordinates given
134         sleepFor(1000);
135         driver.switchTo().alert().accept();
136         sleepFor(500);
137         if (countryResult.getText().isEmpty())
138             System.out.println("Empty inputs for finding country test succeeded.");
139         else
140             System.out.println("Empty inputs for finding country test failed.");
141         sleepFor(500);
142
143         coordinateField.sendKeys("abc, def"); //invalid inputs
144         sleepFor(1000);
145         findCountryBtn.click();
146         sleepFor(1000);
147         driver.switchTo().alert().accept();
148         sleepFor(500);
149         if (countryResult.getText().isEmpty())
150             System.out.println("Invalid inputs for finding country test 1 succeeded.");
151         else
152             System.out.println("Invalid inputs for finding country test 1 failed.");
153         sleepFor(500);
```

Figure 19: First part of the script for testing the Find Country functionality.

```
155         coordinateField.clear();
156         coordinateField.sendKeys("xya/0t9ru.p"); //invalid inputs
157         sleepFor(1000);
158         findCountryBtn.click();
159         sleepFor(1000);
160         driver.switchTo().alert().accept();
161         sleepFor(500);
162         if (countryResult.getText().isEmpty())
163             System.out.println("Invalid inputs for finding country test 2 succeeded.");
164         else
165             System.out.println("Invalid inputs for finding country test 2 failed.");
166         sleepFor(500);
167
168         coordinateField.clear();
169         coordinateField.sendKeys("180 190"); //invalid inputs
170         sleepFor(1000);
171         findCountryBtn.click();
172         sleepFor(1000);
173         driver.switchTo().alert().accept();
174         sleepFor(500);
175         if (countryResult.getText().isEmpty())
176             System.out.println("Invalid inputs for finding country test 3 succeeded.");
177         else
178             System.out.println("Invalid inputs for finding country test 3 failed.");
179         sleepFor(500);
180
181         coordinateField.clear();
182         coordinateField.sendKeys("1620.19, 19500.28"); //invalid inputs
183         sleepFor(1000);
184         findCountryBtn.click();
185         sleepFor(1000);
186         driver.switchTo().alert().accept();
187         sleepFor(500);
188         if (countryResult.getText().isEmpty())
189             System.out.println("Invalid inputs for finding country test 4 succeeded.");
190         else
191             System.out.println("Invalid inputs for finding country test 4 failed.");
192         sleepFor(500);
```

Figure 20: Second part of the script for testing the Find Country functionality.

```
194         coordinateField.clear();
195         coordinateField.sendKeys("39.86883, 32.75162"); //valid inputs
196         sleepFor(1000);
197         findCountryBtn.click();
198         sleepFor(1000);
199         if (countryResult.getText().equals("Turkey"))
200             System.out.println("Valid inputs for finding country test 1 succeeded.");
201         else
202             System.out.println("Valid inputs for finding country test 1 failed.");
203         sleepFor(500);
204
205         coordinateField.clear();
206         coordinateField.sendKeys("20.195134281975815, 76.1819519536014"); //valid inputs
207         sleepFor(1000);
208         findCountryBtn.click();
209         sleepFor(1000);
210         if (countryResult.getText().equals("India"))
211             System.out.println("Valid inputs for finding country test 2 succeeded.");
212         else
213             System.out.println("Valid inputs for finding country test 2 failed.");
214         sleepFor(500);
215
216         coordinateField.clear();
217         coordinateField.sendKeys("-32.8, 21.3838"); //valid inputs
218         sleepFor(1000);
219         findCountryBtn.click();
220         sleepFor(1000);
221         if (countryResult.getText().equals("South Africa"))
222             System.out.println("Valid inputs for finding country test 3 succeeded.");
223         else
224             System.out.println("Valid inputs for finding country test 3 failed.");
225         sleepFor(500);
```

Figure 21: Third part of the script for testing the Find Country functionality.

4.3 Find Distance to the North Pole

4.3.1 Test Case Description

As this functionality works completely automatically unlike the others, requiring no inputs from the user and running calculations solely based on the device coordinates, the test case was simple to write as the only thing to test was whether or not the displayed result was accurate. Thus, as we did for the application's backend, we wrote a function for our test script that found the haversine distance between two points and checked whether the function yielded the same result for the distance between the device and the North Pole as the website did.

4.3.2 Relevant Code Excerpts of the Test Case

```
227         //FIND DISTANCE TO NORTH POLE
228         WebElement deviceLocation;
229         WebElement northPoleResult;
230
231         deviceLocation = driver.findElement(By.xpath("//*[@id=\"info\"]"));
232         northPoleResult = driver.findElement(By.xpath("//*[@id=\"result_northpole\"]"));
233
234         double deviceLat = Double.parseDouble(deviceLocation.getText().substring(10, 18));
235         double deviceLong = Double.parseDouble(deviceLocation.getText().substring(31, 39));
236
237         northPoleTab.click();
238         sleepFor(1500);
239
240         String haversResult = String.format(Locale.US, "%.2f", haversine(deviceLat, deviceLong, 90, 135 ));
241         if (northPoleResult.getText().equals(haversResult))
242             System.out.println("Distance to north pole test succeeded.");
243         else
244             System.out.println("Distance to north pole test failed.");
245
```

Figure 22: Script for testing the Find Distance to the North Pole functionality.

4.4 Code Refactors

4.4.1 Test Case Description

As the functionality itself was among the others, the test case for finding the distance to the Moon's core was the most complex and lengthy process among the rest of the tests that we ran. Having to account for many different factors while being as accurate as possible, since none of us are physicists or astronomers, we had naturally used external libraries [2, 5] that calculated the relevant information for us. Writing the tests in Java and the application in HTML and JavaScript, we had to use similar but different libraries for testing [5] and implementation [2], and comparing the results during testing allowed us to see the inaccuracies of the JavaScript library [2] and refactor our code accordingly. The most important thing to note here is that we strove for as little difference between the expected results of the test library [5] and the results displayed on the website, since finding the exact distance would be almost impossible, especially under the time constraints we had and the experience we had not. This practice of ours can be observed by looking at the print statements we wrote as well, where we displayed the results of our test library and our website back-to-back in each case rather than checking if they were the same or not, and decided at the end that accuracy of higher than 99% was sufficient. Additionally, as we did for the Find Country functionality, we tested the functionality with several sets of valid and invalid inputs, which also led to the discoveries of missing alerts and inaccurate results, followed by further necessary refactors to our code.

4.4.2 Relevant Code Excerpts of the Test Case

```
246         //FIND DISTANCE TO MOON'S CORE
247         WebElement moonCoreField;
248         WebElement moonCoreBtn;
249         WebElement moonCoreResult;
250
251         moonCoreField = driver.findElement(By.xpath("//*[@id=\"mooncor\"]"));
252         moonCoreBtn = driver.findElement(By.xpath("/html/body/div[2]/main/div[8]/div/button"));
253         moonCoreResult = driver.findElement(By.xpath("//*[@id=\"result_mooncore\"]"));
254
255         moonCoreTab.click();
256         sleepFor(500);
257
258         System.out.println(moonCoreDist(deviceLat, deviceLong));
259         System.out.println(moonCoreResult.getText());
260
261         // valid and invalid coordinate inputs
262         moonCoreBtn.click(); // no coordinates given
263         sleepFor(1000);
264         driver.switchTo().alert().accept();
265         sleepFor(500);
266         System.out.print("Result of moon core distance from empty inputs: ");
267         System.out.println(moonCoreResult.getText());
268         sleepFor(500);
269
270         moonCoreField.sendKeys("abc, def"); // invalid inputs
271         sleepFor(1000);
272         moonCoreBtn.click();
273         sleepFor(1000);
274         driver.switchTo().alert().accept();
275         sleepFor(500);
276         System.out.print("Result 1 of moon core distance from invalid inputs: ");
277         System.out.println(moonCoreResult.getText());
278         sleepFor(500);
```

Figure 23: First part of the script for testing the Find Distance to the Moon's Core functionality.

```
280         moonCoreField.clear();
281         moonCoreField.sendKeys("xya/0t9ru.p"); // invalid inputs
282         sleepFor(1000);
283         moonCoreBtn.click();
284         sleepFor(1000);
285         driver.switchTo().alert().accept();
286         sleepFor(500);
287         System.out.print("Result 2 of moon core distance from invalid inputs: ");
288         System.out.println(moonCoreResult.getText());
289         sleepFor(500);
290
291         moonCoreField.clear();
292         moonCoreField.sendKeys("180 190"); // invalid inputs
293         sleepFor(1000);
294         moonCoreBtn.click();
295         sleepFor(1000);
296         driver.switchTo().alert().accept();
297         sleepFor(500);
298         System.out.print("Result 3 of moon core distance from invalid inputs: ");
299         System.out.println(moonCoreResult.getText());
300         sleepFor(500);
301
302         moonCoreField.clear();
303         moonCoreField.sendKeys("1620.19, 19500.28"); // invalid inputs
304         sleepFor(1000);
305         moonCoreBtn.click();
306         sleepFor(1000);
307         driver.switchTo().alert().accept();
308         sleepFor(500);
309         System.out.print("Result 4 of moon core distance from invalid inputs: ");
310         System.out.println(moonCoreResult.getText());
311         sleepFor(500);
```

Figure 24: Second part of the script for testing the Find Distance to the Moon's Core functionality.

```
313         System.out.println();
314         moonCoreField.clear();
315         moonCoreField.sendKeys("39.86883, 32.75162"); // valid inputs
316         sleepFor(1000);
317         moonCoreBtn.click();
318         sleepFor(1000);
319         System.out.print("Result 1 of moon core distance from valid inputs: ");
320         System.out.println(moonCoreResult.getText());
321         System.out.print("Expected Result: ");
322         System.out.println(moonCoreDist(39.86883, 32.75162));
323         sleepFor(500);
324
325         moonCoreField.clear();
326         moonCoreField.sendKeys("20.195134281975815, 76.1819519536014"); // valid inputs
327         sleepFor(1000);
328         moonCoreBtn.click();
329         sleepFor(1000);
330         System.out.print("Result 2 of moon core distance from valid inputs: ");
331         System.out.println(moonCoreResult.getText());
332         System.out.print("Expected Result: ");
333         System.out.println(moonCoreDist(20.195134281975815, 76.1819519536014));
334         sleepFor(500);
335
336         moonCoreField.clear();
337         moonCoreField.sendKeys("-32.8, 21.3838"); // valid inputs
338         sleepFor(1000);
339         moonCoreBtn.click();
340         sleepFor(1000);
341         System.out.print("Result 1 of moon core distance from valid inputs: ");
342         System.out.println(moonCoreResult.getText());
343         System.out.print("Expected Result: ");
344         System.out.println(moonCoreDist(-32.8, 21.3838));
345         sleepFor(500);
```

Figure 25: Third part of the script for testing the Find Distance to the Moon's Core functionality.

5 Conclusions and Evaluation of TDD Experience

As group members, we had previous experience with writing automated tests but none of us had any previous experience with Test Driven Development. TDD worked well in this project and we were able to get what we aimed for, software that meets the requirements. At first, we thought that TDD would take us more time compared to the first two projects where we developed the application and wrote the tests afterwards. However, TDD really speeded things up for us in this project. In our opinion, the application requiring only a few simple features was the reason behind that.

We wrote simple requirements for each feature e.g., the test for finding the country of a given location. After that, it was simple to write the minimal code to satisfy the test case and pass the redline, in our example finding the correct country for the given location. All required features were similar, they were all about geographic locations and distances and we wrote tests and codes for them one by one. Refactoring the code was not as hard as we thought it would be because it was a small-scale application with similar features.

In the previous projects, writing the whole application and then failing a test was challenging because sometimes we had to review the whole code and things could get very complex and costly. Preparing tests for testing each small feature while still developing the code made things easier for us because when the test failed, we knew where to look for it, and passing the test was the green line for that feature.

We think that the application being simple is not the only reason for TDD working well. Large-scale projects can also be developed more comfortably with this approach because it results in a program with more readable and understandable code as well as fewer bugs and errors.

In conclusion, our first experience with TDD was very satisfying and we did not regret developing the project with TDD. We think that it can be applied in many scenarios to make things easier for developers.

6 References

[1] “Geocoding API.” Geoapify. Accessed: April 26, 2022. Available: https://www.geoapify.com/geocoding-api

[2] “SunCalc.” mourner. Accessed: April 26, 2022. Available: https://github.com/mourner/suncalc

[3] “Moon Distances for Ankara, Turkey.” timeanddate. Accessed: April 26, 2022. Available: https://www.timeanddate.com/astronomy/moon/distance.html

[4] “How Far is The Moon from Earth?” The Sky Live. Accessed: April 26, 2022. Available: https://theskylive.com/how-far-is-moon

[5] “commons-suncalc.” Shredzone. Accessed: April 26, 2022. Available: https://shredzone.org/maven/commons-suncalc/index.html