CS 458
Software Verification & Validation
2019-2020 Spring

PROJECT #4

Berk Yıldız 21502040

Contents

1. Implementation

The website consists of four pages: Homepage, sign up, profile and symptoms. User can login the system from homepage. They can create account on sign up page. Profile page stands for the entrance of demographic information. Finally user can check his/her status regarding the Covid-19 on symptoms page. Website is scalable for all kind of devices.

The website is implemented by JavaScript and HTML. Project consists of six classes: *main.js, index.html, profile.html, signup.html, symptoms.html and test.java.* HTML creates the skeleton of the website and *main.js* brings interactivity into webpage by including the methods for the operations. *test.java* class stands for unit testing. The source code is in the project folder.

a. Page Properties

i. Homepage

User can login to system by entering valid username and password. This page throws 4 alerts for four conditions: Blank username and password, blank username, blank password and wrong username or password. Successful login directs user to the profile page.

ii. Sign Up

This is the page which user creates an account by choosing username and password. It is not allowed choose an existing username. Also page throws alert for missing fields.

iii. Profile

This is the page which user enters his/her demographic information. User should fill in all the fields before submission, otherwise page throws an alert to warn user.

iv. Symptoms

This is the page where user enters symptoms and learn about his/her conditions. Page throws three alerts to the user: You are healhty, visit nearest hospital and watch your condition. User can enter information for seven main symptoms. These are body temperature, difficulty in breathing, cough, muscle pain, chills, sore throat and loss of taste or smell. User should enter his/her body temperature in celcius. User should rate his/her breathing, cough, chills, muscle pain and sore throat

between 0 and 10. Just entering “yes” or “no” is enough for loss of taste or smell. I gathered the symptoms from website of WHO. If body temperature is greater than 37.9 celcius or there is loss of taste or smell, page directly advices for visiting a hospital regardless of other symptoms. If user rates other symptoms greater than 5, page advices to visit hospital again. If body temperature is normal and symptoms are less than 2, page states the user that he/she is healthy. For the all other conditions, pages advices to watch condition at home.

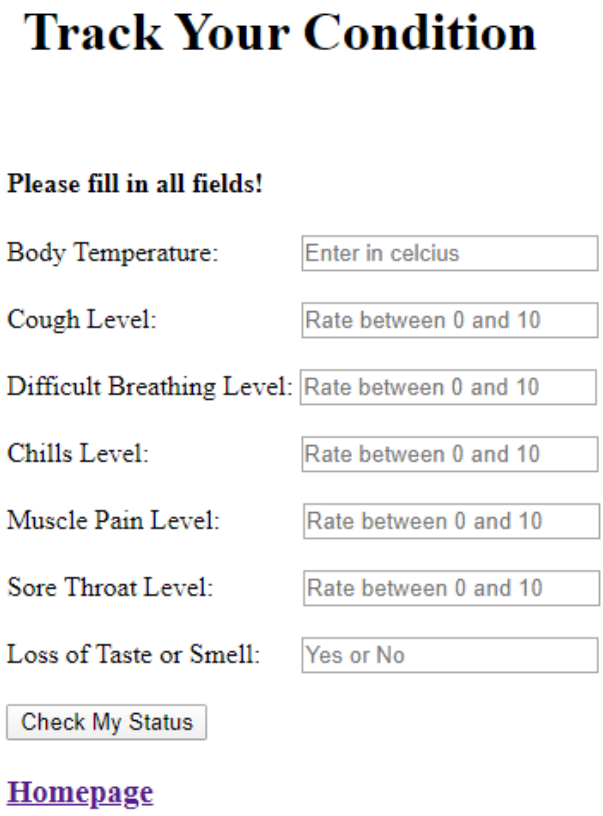
b. Desing of Pages

i. Homepage

ii. Sign Up

iii. Profile

iv. Symptoms



2. Unit Tests

There are four functions in the *main.js* file which handles website operations. These methods are *getInfo(), signUp(), updateProfile()* and *checkStatus().*

getInfo() function handles the login operations.

signUp() function handsles the sign up operations.

updateProfile() function handles profile and demographic information operations.

checkStatus() function handlses the symptom and advice operations.

Java Extension Pack of Visual Studio Code allowed me to write and run unit tests by JUnit on JavaScript. This extension pack includes core extensions like Language Support for Java, Debugger for Java and Java Test Runner.

a. getInfo() Unit Tests for Login Operations

- **Blank Username and Password**

```
JunitTesting test = new JunitTesting();

 @Test
 public void testBlankUsernamePassword() {
     String output = test.getInfo("", "");
     String expected = "Username and password cannot be blank!";
     assertEqual(expected, output);
 }
```

- **Blank Username**

```
@Test
    public void testBlankUsername() {
        String output = test.getInfo("", "abc");
        String expected = "Username cannot be blank!";
        assertEqual(expected, output);
    }
```

- **Blank Password**

```
@Test
    public void testBlankPassword() {
        String output = test.getInfo("aaa", "");
        String expected = "Password cannot be blank!";
        assertEqual(expected, output);
    }
```

- **Wrong Username or Password**

```
@Test
    public void testWrongUsernamePassword() {
        String output = test.getInfo("xyz", "1234");
        String expected = "Username or password is wrong!";
        assertEqual(expected, output);
    }
```

- **Successful Login**

```
@Test
    public void testSuccessfulLogin() {
        String output = test.getInfo("123", "abc");
        String expected = "Successful Login";
        assertEqual(expected, output);
    }
```

b. signUp() Unit Tests for Sign Up Operations

- **Blank Username and Password**

```
@Test
    public void testSignUpBlankUsernamePassword() {
        String output = test.signUp("", "");
        String expected = "Please enter username and password!";
        assertEqual(expected, output);
    }
```

- **Blank Username**

```
@Test
    public void testSignUpBlankUsername() {
        String output = test.signUp("", "sasasa");
        String expected = "Please enter username!";
        assertEqual(expected, output);
```

- **Blank Password**

```
@Test
    public void testSignUpBlankPassword() {
        String output = test.signUp("aaaa", "");
        String expected = "Please enter password!";
        assertEqual(expected, output);
    }
```

- **Existing Username**

```
@Test
    public void testSignUpExistingUsername() {
        String output = test.signUp("123", "");
        String expected = "Username exists. Please choose different o
ne.";
        assertEqual(expected, output);
    }
```

- **Successful Sign Up**

```
@Test
    public void testSignUpSuccess() {
        String output = test.signUp("xyz", "test");
        String expected = "Account created. Please login from homepag
e.";
        assertEqual(expected, output);
    }
```

c. updateProfile() Unit Tests for Profile Operations

- **Blank fields (all fields)**

```
@Test
    public void testUpdateProfileBlankFields() {
        String output = test.updateProfile("", "","","","","");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Full Name**

```
@Test
    public void testUpdateProfileBlankFullName() {
        String output = test.updateProfile("", "40","Male","Turkey","
Ankara","Cankaya");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Age**

```
@Test
    public void testUpdateProfileBlankAge() {
        String output = test.updateProfile("abc def", "","Male","Turk
ey","Ankara","Cankaya");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Gender**

```
@Test
    public void testUpdateProfileBlankGender() {
        String output = test.updateProfile("abc def", "40","","Turkey
","Ankara","Cankaya");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Country**

```
@Test
    public void testUpdateProfileBlankCountry() {
        String output = test.updateProfile("abc def", "40","Male","",
"Ankara","Cankaya");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank City**

```
@Test
    public void testUpdateProfileBlankCity() {
        String output = test.updateProfile("abc def", "40","Male","Tu
rkey","","Cankaya");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank District**

```
    @Test
    public void testUpdateProfileBlankDistrict() {
        String output = test.updateProfile("abc def", "40","Male","Tu
rkey","Ankara","");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Successful Update**

```
@Test
    public void testUpdateSuccessfulUpdate() {
        String output = test.updateProfile("abc def", "40","Male","Tu
rkey","Ankara","Cankaya");
        String expected = "Your information has been saved.";
        assertEqual(expected, output);
    }
```

d. checkStatus() Unit Tests for Symptom and Advice Operations

- **Blank Fields (All Fields)**

```
@Test
    public void testCheckStatusBlankFields() {
        String output = test.checkStatus("", "","","","","","");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Breathing Level**

```
@Test
    public void testCheckStatusBlankBreathingLevel() {
        String output = test.checkStatus("", 38,5,5,5,5,"No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Body Temperature**

```
@Test
    public void testCheckStatusBlankBodyTemperature() {
        String output = test.checkStatus(5, "",5,5,5,5,"No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Chills Level**

```
@Test
    public void testCheckStatusBlankChillsLevel() {
        String output = test.checkStatus(5, 38,"",5,5,5,"No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Muscle Pain Level**

```
@Test
    public void testCheckStatusBlankMusclePainLevel() {
        String output = test.checkStatus(5, 38,5,"",5,5,"No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Sore Throat Level**

```
@Test
    public void testCheckStatusBlankSoreThroatLevel() {
        String output = test.checkStatus(5, 38,5,5,"",5,"No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Cough Level**

```
@Test
    public void testCheckStatusBlankCoughLevel() {
        String output = test.checkStatus(5, 38,5,5,5,"","No");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Blank Loss of Taste of Smell**

```
@Test
    public void testCheckStatusBlankTasteSmell() {
        String output = test.checkStatus(5, 38,5,5,5,5,"");
        String expected = "Please fill in all fields!";
        assertEqual(expected, output);
    }
```

- **Healthy Advice**

```
@Test
    public void testCheckStatusHealthy() {
        String output = test.checkStatus(1, 36,0,0,1,0,"No");
        String expected = "You look healthy. Be careful though. Stay
at home.";
        assertEqual(expected, output);
    }
```

- **High Body Temperature Advice**

```
@Test
    public void testCheckStatusHighBodyTemp() {
        String output = test.checkStatus(1, 40,0,0,1,0,"No");
        String expected = "Visit the nearest hospital";
        assertEqual(expected, output);
    }
```

- **Loss of Taste or Smell Advice**

```
@Test
    public void testCheckStatusLossTasteSmell() {
        String output = test.checkStatus(1, 36,0,0,1,0,"Yes");
        String expected = "Visit the nearest hospital";
        assertEqual(expected, output);
    }
```

- **Visit Hospital Advice**

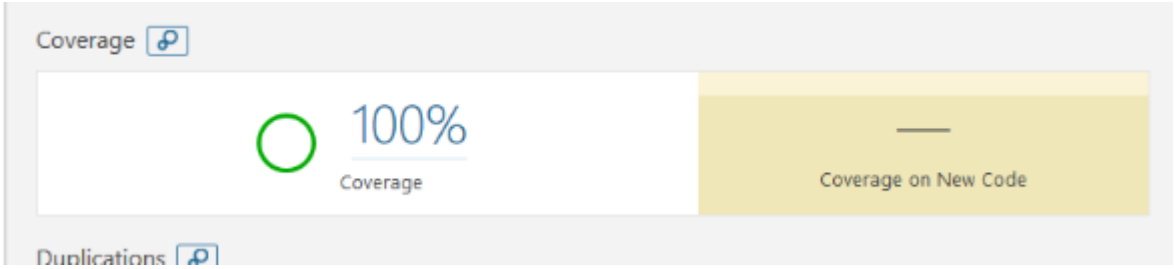
```
@Test
    public void testCheckStatusHeavy() {
        String output = test.checkStatus(6, 37,6,8,9,10,"No");
        String expected = "Visit the nearest hospital";
        assertEqual(expected, output);
    }
```

- **Watch Condition Advice**

```
@Test
    public void testCheckStatusWatchCondition() {
        String output = test.checkStatus(1, 36,4,2,5,6,"No");
        String expected = "Watch your condition. Stay at home!";
        assertEqual(expected, output);
    }
```

e. Test Coverage

I could able to analyze test coverage of unit tests with Sonarlint extension of Visual Studio Code. This extension allowed me to connect project with SonarQube. The test coverage of the unit tests were about 60% in my first analysis. Already it is an small project and does not have thousands of lines of code. So by increasing the scope of my test cases I finally reached the 100% coverage by including all code blocks into testing process.



3. Evaluation of Unit Test Experience

Unit test is the smallest possible component which is testable in a software. The smallness of unit tests provides some advantages. These advantages are that it is easy to design and execute unit tests which make the development process agile. Easy design and execution provides a faster feedback mechanism which helps developer to take faster action for solving problems. When we consider the development process, debugging and bug fixing take more time than writing the code. Unit testing reduces the time on debugging because it allows to find location of bugs much more easier. A reduction in debugging time directly have positive effects on development velocity. However unit testing should be supported by integration tests because only unit tests can not be sufficient for finding all bugs in the system. In terms of code quality unit testing provides improvements because it helps to identify low-level bugs and defects before sending the code to integration tests or production. Also it helps to identify boundary conditions of the code in earlier stages. Clenaer and smooth code arrives to further steps by the help of unit testing.