

CS 458

Software Validation and Verification



Introduction to Unit Testing

Project #4

Okan Şen	21202377
Hassam Abdullah	21701610
Mustafa Tuna Acar	21703639

Introduction to Unit Testing

1 Code Details/Information	2
1.1 Screenshot of Web Application	2
Login Page	2
Registration Page	3
Survey Page	4
1.2 Excerpts of Web Page Code	5
1.3 Screenshot of Test Code and Test Cases	7
1.3.1 Sign up Page Works Properly	7
1.3.2 Log in Page Works Properly	8
1.3.3 FrontPage Works Properly	10
1.3.4 The User Data is Saved/Retrieved Successfully	11
1.3.5 User Being Notified of Their Symptoms Progress	12
1.4 Excerpts of Test Code	12
2 UML Diagrams	14
2.1 Class Diagram	14
2.2 Use Case Diagram	15
2.3 Sequence Diagram	16
3 Coverage of Unit Tests Calculation	17
4 Unit Testing Experience	17
4.1 Development Velocity	17
4.2 Code Quality	17

1 Code Details/Information

1.1 Screenshot of Web Application

Login Page

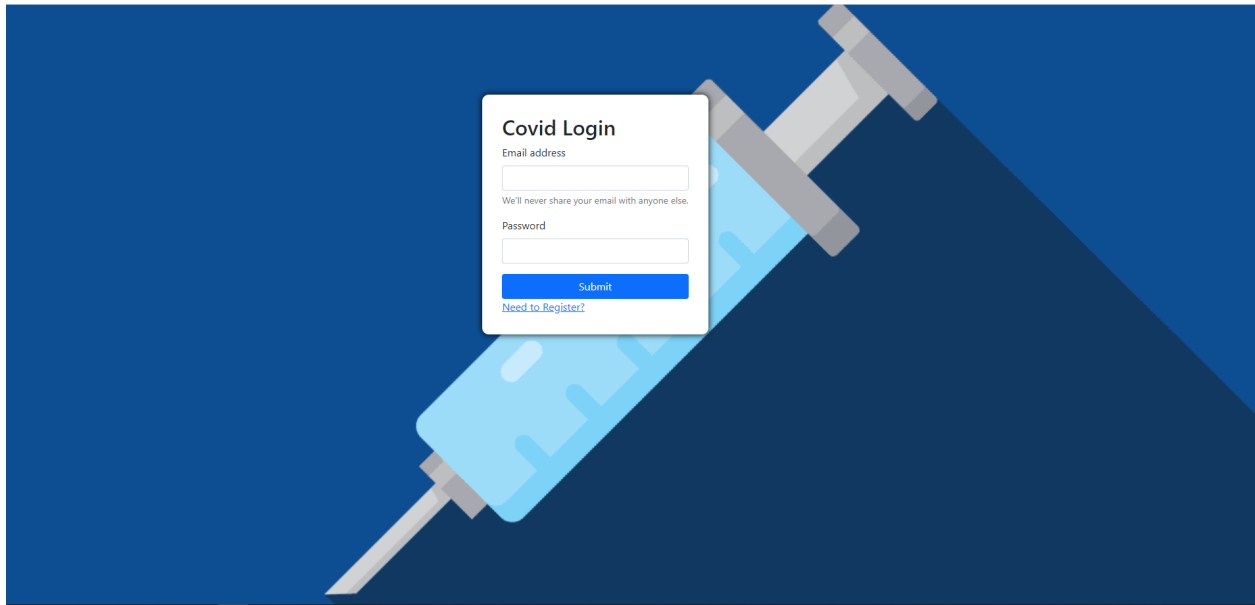
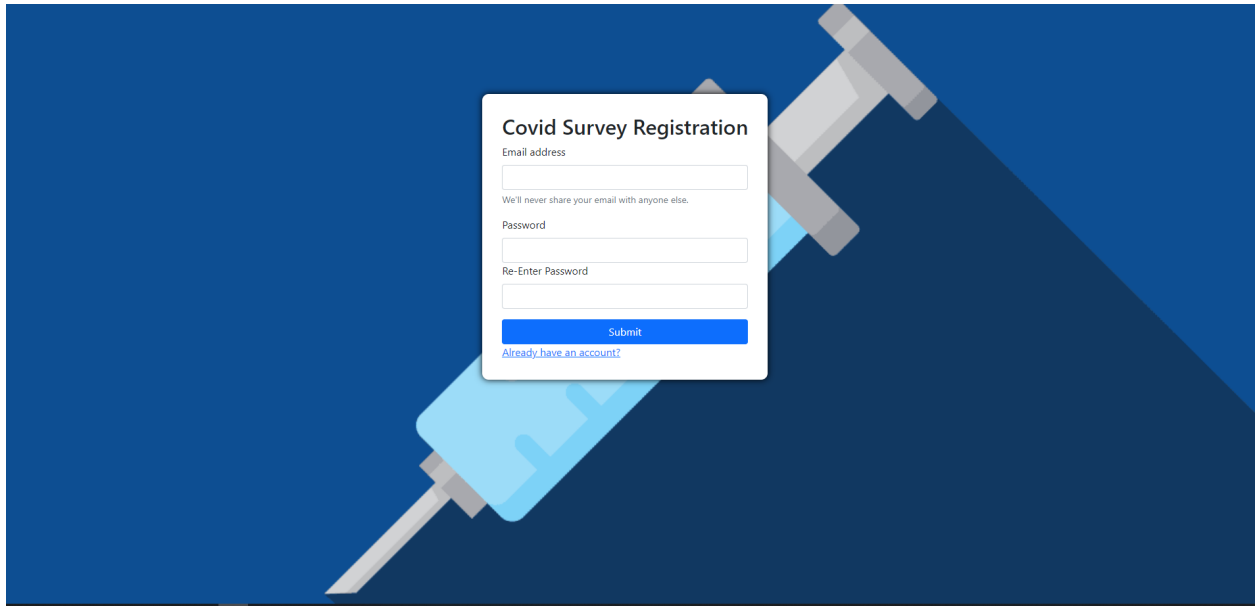


Figure 1: Login Page

Figure 1, shows the main login page from where the user can redirect to either the registration page by clicking on the “Need to register?” link or login successfully by entering valid credentials and getting redirected to the Survey page.

Registration Page



Covid Survey Registration

Email address

We'll never share your email with anyone else.

Password

Re-Enter Password

[Already have an account?](#)

Figure 2: Registration Page

This is the website's Registration Page where the user can enter his email/password to receive a confirmation that his account has been registered in the database after the Submit button has been pressed. The user can also choose to press the "Already have an account?" link to go back to the login page.

Survey Page

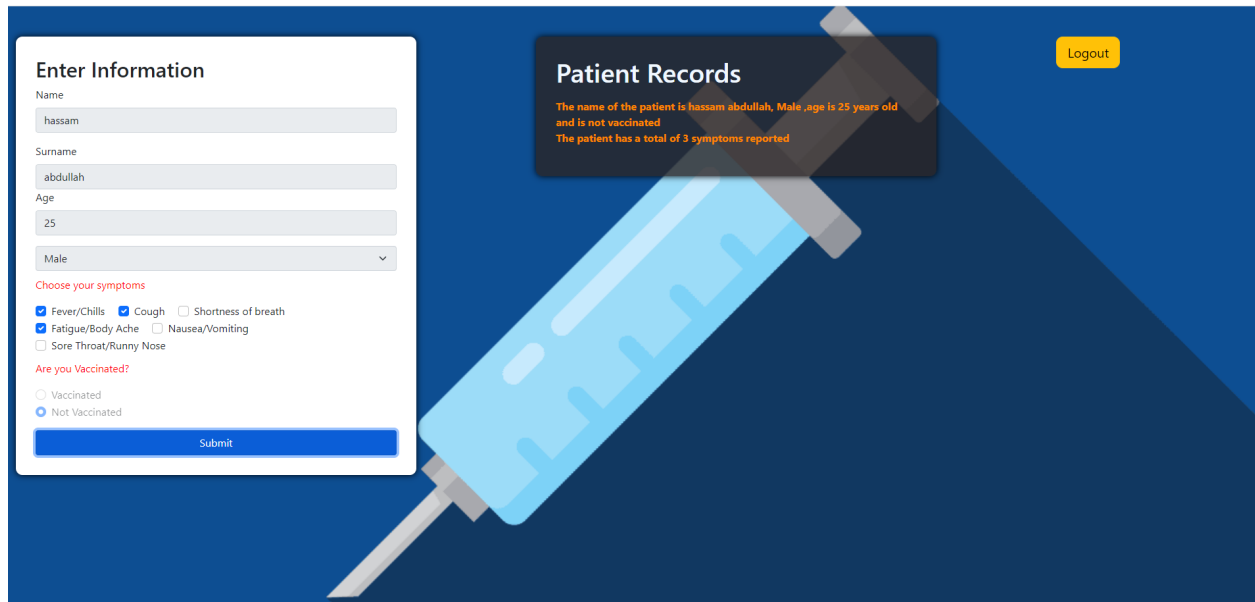
A screenshot of a web application's Survey Page. The background is dark blue with a large, light blue syringe graphic. On the left, a white 'Enter Information' form contains input fields for Name (hassam), Surname (abdullah), Age (25), and Gender (Male). It also has checkboxes for symptoms: Fever/Chills, Cough, Shortness of breath, Fatigue/Body Ache, Nausea/Vomiting, and Sore Throat/Runny Nose. A 'Submit' button is at the bottom. On the right, a dark grey 'Patient Records' box displays text: 'The name of the patient is hassam abdullah, Male ,age is 25 years old and is not vaccinated' and 'The patient has a total of 3 symptoms reported'. A yellow 'Logout' button is in the top right corner.

Figure 3: Survey Page

This is the website's Survey page and the user is redirected to this after he enters valid credentials in the Login page that are verified in the pseudo-database created by localStorage variables. The user can enter the information form given and the Patient Records will be updated. After the first submission, the user can only toggle his symptoms which will be compared to the symptoms that the user had inputted earlier after he logs out and will be alerted if symptoms are getting worse or better.

Important excerpts of code containing PHP/HTML/javascript etc can be viewed below that are integral to the functioning and understanding of the website.

Figure 4: Javascript Code in login.js

Figure 5: HTML code in login.html

```

<script type="text/javascript">

$(document).ready(function () {

    var check1 = 0;
    var check2 = 0;
    var check3 = 0;
    var check4 = 0;
    var check5 = 0;
    var check6 = 0;

    var vaccinated = 0;
    var saveUserName = localStorage.getItem("login") + "name";
    var saveSurname = localStorage.getItem("login") + "surname";
    var saveAge = localStorage.getItem("login") + "age";
    var saveVac = localStorage.getItem("login") + "vac";
    var saveGender = localStorage.getItem("login") + "gender";
    var saveSeverity = localStorage.getItem("login") + "severity";
    var alreadyFilled = localStorage.getItem("login") + "filled";
    var onceSubmitted = localStorage.getItem("login") + "submitted";

    isLogged();
    disableIfFilled();
    setUserInfo();

    function setUserInfo() {

        var name = localStorage.getItem(saveUserName);
        var surname = localStorage.getItem(saveSurname);
        var age = localStorage.getItem(saveAge);
        var severity = localStorage.getItem(saveSeverity);
        var vac = localStorage.getItem(saveVac);
        var userGender = localStorage.getItem(saveGender);

        if (localStorage.getItem(alreadyFilled) == 1) {
            document.getElementById("login-error").innerText = "This form has been submitted already! Alter symptoms you are feeling today!";
            document.getElementById("details").innerText = "The name of the patient is " + name + " " + surname + ", " + userGender + " ,age is " + age + " years old" + " and is " + vac + "in" +
                "The patient has a total of " + severity + " symptoms reported";
        }

    }

    function disableIfFilled() {
        if (localStorage.getItem(alreadyFilled) == 1) {
            document.getElementById("name").disabled = true;
            document.getElementById("surname").disabled = true;
            document.getElementById("age").disabled = true;
            document.getElementById("gender").disabled = true;
            document.getElementById("flexRadioDefault1").disabled = true;
            document.getElementById("flexRadioDefault2").disabled = true;
        }
    }
});

```

Figure 6: Javascript code using JQuery in front.html

Figure 4, shows the logic that deals with form validation in the Login page of the website. There is a similar file for the Registration page of the website with identical logic. The localStorage variables saved in the registration page are checked against the user input in the Login page and validated, as seen in Figure 4.

Figure 5, shows the standard HTML code prevalent in HTML files related to all the pages, using Bootstrap to ensure responsiveness.

Figure 6, shows the JQuery code written for the Survey page after the user has successfully logged in from the Login page. The figure shows the logic related to saving the user's data after the user has submitted his details so that they may be checked in the next session after he logs out and to alert the user if his symptoms show deterioration or improvement.

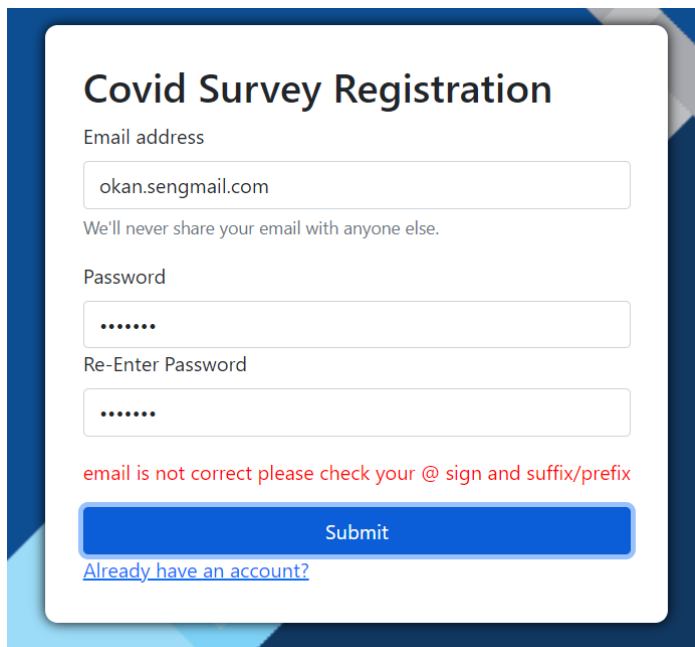
1.3 Screenshot of Test Code and Test Cases

The following test cases have been implemented to ensure the integrity of the web page's functionalities:-

- 1) Sign up page works properly (no same e-mails, e-mail is properly formatted)
- 2) Log in page works properly (e-mail has not registered notification, e-mail is properly formatted)
- 3) The front page has fields that are properly formatted and must be filled only once. All fields are blocked once submitted and cannot be changed except for symptoms checkboxes.
- 4) The user data is saved and retrieved for the next login session successfully
- 5) The user is notified if his symptoms are starting to get worse or better.

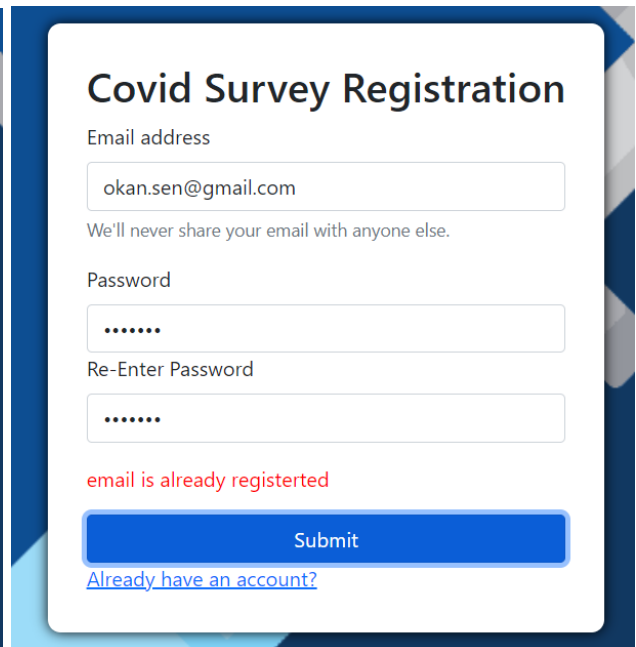
1.3.1 Sign up Page Works Properly

The signup page does not accept invalid emails, and the passwords must be the same strings. Other than that, the email should not have already been registered to the website before.



The screenshot shows the 'Covid Survey Registration' form. The 'Email address' field contains 'okan.sengmail.com'. Below the field, a red error message states: 'email is not correct please check your @ sign and suffix/prefix'. The 'Password' and 'Re-Enter Password' fields are filled with dots. A blue 'Submit' button is at the bottom, with a link '[Already have an account?](#)' below it.

Figure 7: Invalid email



The screenshot shows the 'Covid Survey Registration' form. The 'Email address' field contains 'okan.sen@gmail.com'. Below the field, a red error message states: 'email is already registered'. The 'Password' and 'Re-Enter Password' fields are filled with dots. A blue 'Submit' button is at the bottom, with a link '[Already have an account?](#)' below it.

Figure 8: Email already registered

Covid Survey Registration

Email address

okan.sen@gmail.com

We'll never share your email with anyone else.

Password

...

Re-Enter Password

.....

Passwords do not match

Submit

[Already have an account?](#)

Figure 9: Passwords do not match

Covid Survey Registration

Email address

okan.sen5@gmail.com

We'll never share your email with anyone else.

Password

.....

Re-Enter Password

.....

You have been successfully registered

Submit

[Already have an account?](#)

Figure 10: Successful registration

1.3.2 Log in Page Works Properly

The login form also does not accept invalid emails, emails that are not registered to our website, and wrong email and password combinations.

Covid Login

Email address

test@bilkent.edu.tr

We'll never share your email with anyone else.

Password

.....

Email does not exist

Submit

[Need to Register?](#)

Figure 11: Unregistered email

Covid Login

Email address

okan.sen@gmail.com

We'll never share your email with anyone else.

Password

.....

Password is incorrect

Submit

[Need to Register?](#)

Figure 12: Wrong password combination

Covid Login

Email address

We'll never share your email with anyone else.

Password

email is not correct please check your @ sign and suffix/prefix

Submit

[Need to Register?](#)

Figure 13: Invalid email

Application	
Manifest	
Service Workers	
Storage	
Local Storage	
file://	
Session Storage	
IndexedDB	
Web SQL	
Cookies	
Trust Tokens	
Cache	
Cache Storage	
Application Cache	
Background Services	
Background Fetch	
Background Sync	
Notifications	
Payment Handler	
Periodic Background Sync	

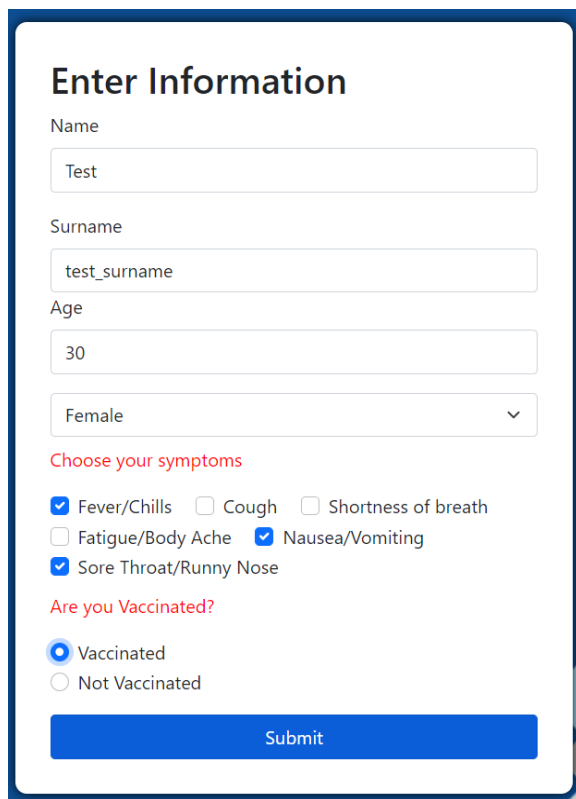
Key	Value
okan.sen1@g...	not vaccinated
okan.sen3@g...	2
okan.sen1@g...	Male
okan.sen@gm...	0
okan.sen@gm...	Sen
okan.sen@gm...	0
okan.sen1@g...	1
okan.sen1@g...	300
okan.sen@gm...	1
okan.sen1@g...	asd
okan.sen2@g...	asd
okan.sen2@g...	1
okan.sen2@g...	0
okan.sen2@g...	vaccinated
okan.sen@gm...	Male
okan.sen3@g...	23
okan.sen3@g...	asd
okan.sen3@g...	0
okan.sen2@g...	Male
okan.sen2@g...	53
okan.sen2@g...	asd
okan.sen@gm...	vaccinated

Figure 14: Saved Data

For this project, we saved the user data as saved data, which saves information locally, and is easy to implement for testing purposes. Since we have already implemented a database for Project 1, we did not want to overcomplicate this project with the database as the purpose of this project is unit testing and responsiveness rather than implementing a database.

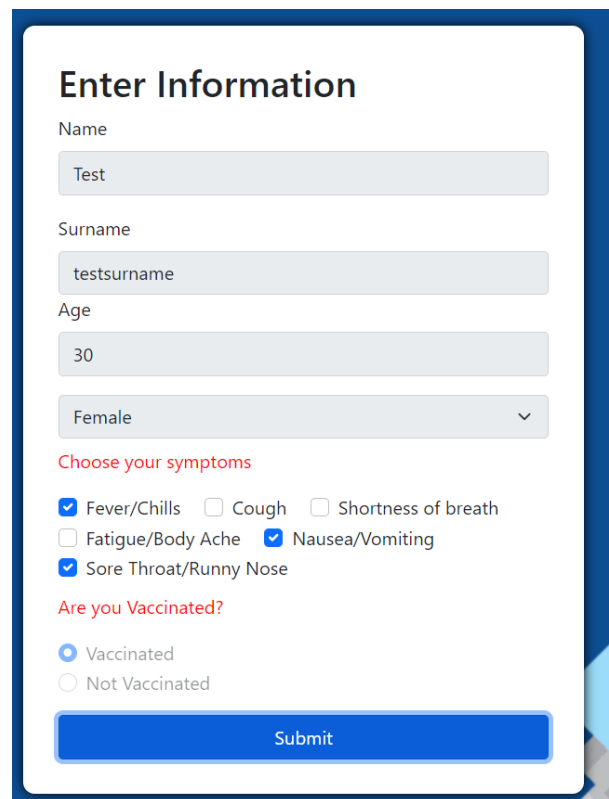
1.3.3 FrontPage Works Properly

Once the user is logged in, they are asked to provide information about themselves such as; name, surname, age, symptoms, gender, and vaccination status. Once they submit this info, they cannot edit their entries, except their symptoms information which is to track their illness progress.



The screenshot shows a web form titled "Enter Information". It contains several input fields: "Name" with the value "Test", "Surname" with "test_surname", "Age" with "30", and a "Gender" dropdown menu set to "Female". Below these is a section titled "Choose your symptoms" with six checkboxes: "Fever/Chills" (checked), "Cough" (unchecked), "Shortness of breath" (unchecked), "Fatigue/Body Ache" (unchecked), "Nausea/Vomiting" (checked), and "Sore Throat/Runny Nose" (checked). Another section titled "Are you Vaccinated?" has two radio buttons: "Vaccinated" (selected) and "Not Vaccinated" (unselected). At the bottom is a blue "Submit" button.

Figure 15: Before submitting info



This screenshot shows the same "Enter Information" form as Figure 15, but after the data has been submitted. All input fields (Name, Surname, Age, Gender, symptoms checkboxes, and vaccination radio buttons) are now greyed out, indicating they are no longer editable. The "Submit" button remains visible at the bottom.

Figure 16: After submitting info (notice greyed out fields)

Enter Information

Name
123

Surname
_!?!^

Age
22

Male

Choose your symptoms

☐ Fever/Chills ☒ Cough ☐ Shortness of breath
☐ Fatigue/Body Ache ☒ Nausea/Vomiting
☐ Sore Throat/Runny Nose

Are you Vaccinated?

☐ Vaccinated
☒ Not Vaccinated

Name or surname must only contain letters

Submit

Figure 17: Name/Surname fields require strings

Enter Information

Name
test

Surname
test

Age
test

Male

Choose your symptoms

☐ Fever/Chills ☒ Cough ☐ Shortness of breath
☐ Fatigue/Body Ache ☒ Nausea/Vomiting
☐ Sore Throat/Runny Nose

Are you Vaccinated?

☐ Vaccinated
☒ Not Vaccinated

Age must be an integer

Submit

Figure 18: Age field requires an integer

As usual, the fields do not accept special characters or numbers. The age field must also get an integer input.

1.3.4 The User Data is Saved/Retrieved Successfully

Each user's information is retrieved when they log in. Here are several instances of users.

Patient Records

The name of the patient is Okan Sen, Male ,age is 27 years old and is vaccinated

The patient has a total of 2 symptoms reported

Figure 19: User 1 Data

Patient Records

The name of the patient is Hassam Abdullah, Male ,age is 25 years old and is vaccinated

The patient has a total of 3 symptoms reported

Figure 20: User 2 Data

1.3.5 User Being Notified of Their Symptoms Progress

All the users' progress is tracked with dynamic messages on the dashboard. If the user's symptoms are worsened, the system advises them to see a doctor or take care of themselves, and if the symptoms are getting better, they are encouraged with positive messages.

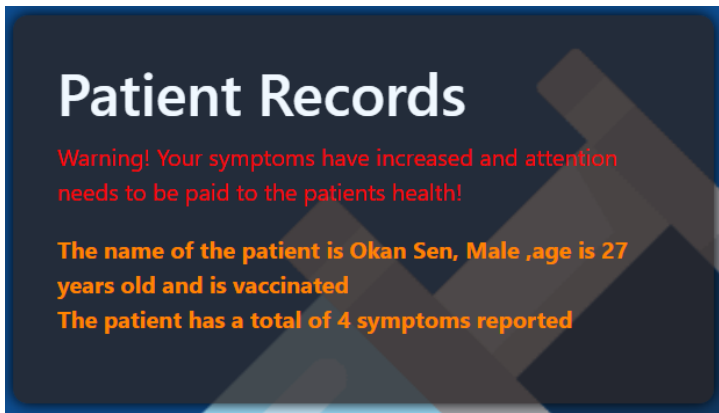


Figure 21: Worsened symptoms



Figure 22: Patient getting better

1.4 Excerpts of Test Code

For this project, we used Java to test the webpage we have created. We used JUnit and Selenium for automated unit testing, integrated to Eclipse.

```
public class JUnit_Selenium_Test_Class {  
  
    static WebDriver driver;  
  
    @BeforeClass  
    public static void BrowserOpen() {  
  
        System.setProperty("webdriver.chrome.driver", "C:\\webdrivers\\chromedriver92\\chromedriver.exe");  
        driver= new ChromeDriver() ;  
        driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);  
    }  
}
```

Figure 23: Initializing web driver and JUnit testing with @BeforeClass

```

@Test
public void testCase1() throws IOException, InterruptedException {

    driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);
    driver.get("D:\\Program Files (x86)\\Eclipse\\eclipse saves\\JUnit_Selenium_Testing\\");
    driver.manage().window().maximize();

    // TEST CASE 1
    // Sign-up page works properly (no same e-mails, e-mail is properly formatted)
    System.out.println("\n\n\n");
    System.out.println("Test case 1 junit selenium");

    // Find fields and button from the web page
    WebElement reg_id_field = driver.findElement(By.id("email1"));
    WebElement reg_pass_field1 = driver.findElement(By.id("password1"));
    WebElement reg_pass_field2 = driver.findElement(By.id("password2"));

    WebElement submit_btn = driver.findElement(By.xpath("html/body/div/div/div/form/div[

    // The test e-mails and passwords are stored in this array by reading from files.
    String test_case_1_inputs[] = new String[100];
    String test_case_1_filepath = "D:\\Program Files (x86)\\Eclipse\\eclipse saves\\JUnit

    readFiles(test_case_1_filepath, test_case_1_inputs);

```

Figure 24: Test Case 1

With JUnit, we can create multiple tests using “@Test”. In this way, we create multiple units of test cases. When the code is run, it traverses the first test until the last one. As it finishes test cases, each is marked as successful or failed.

```

@Test
public void testCase3() throws IOException, InterruptedException {

    // TEST CASE 3
    // Front page needs fields to be filled, all fields except symptoms cannot be
    // changed once submitted. The test submits info for one user then logs out
    // and submits new info for a second user and logs out for test case 4.
    System.out.println("\n\n\n");
    System.out.println("Test case 3 junit selenium");

    // Find fields and button from the web page
    WebElement front_name_field = driver.findElement(By.id("name"));
    WebElement front_surname_field = driver.findElement(By.id("surname"));
    WebElement front_age_field = driver.findElement(By.id("age"));
    Select front_gender_select = new Select(driver.findElement(By.id("gender")));

    WebElement submit_btn = driver.findElement(By.xpath("html/body/div/div/div/fc

```

Figure 25: Test Case 3

After implementing JUnit, apart from the definitions of the units, and running the project differently, writing the code is the same as a normal Selenium automation code. There is no difference involved, and this makes JUnit easy to use.

2 UML Diagrams

2.1 Class Diagram

Viral Paradigm Standard (Metaform Aca(01nextUML))

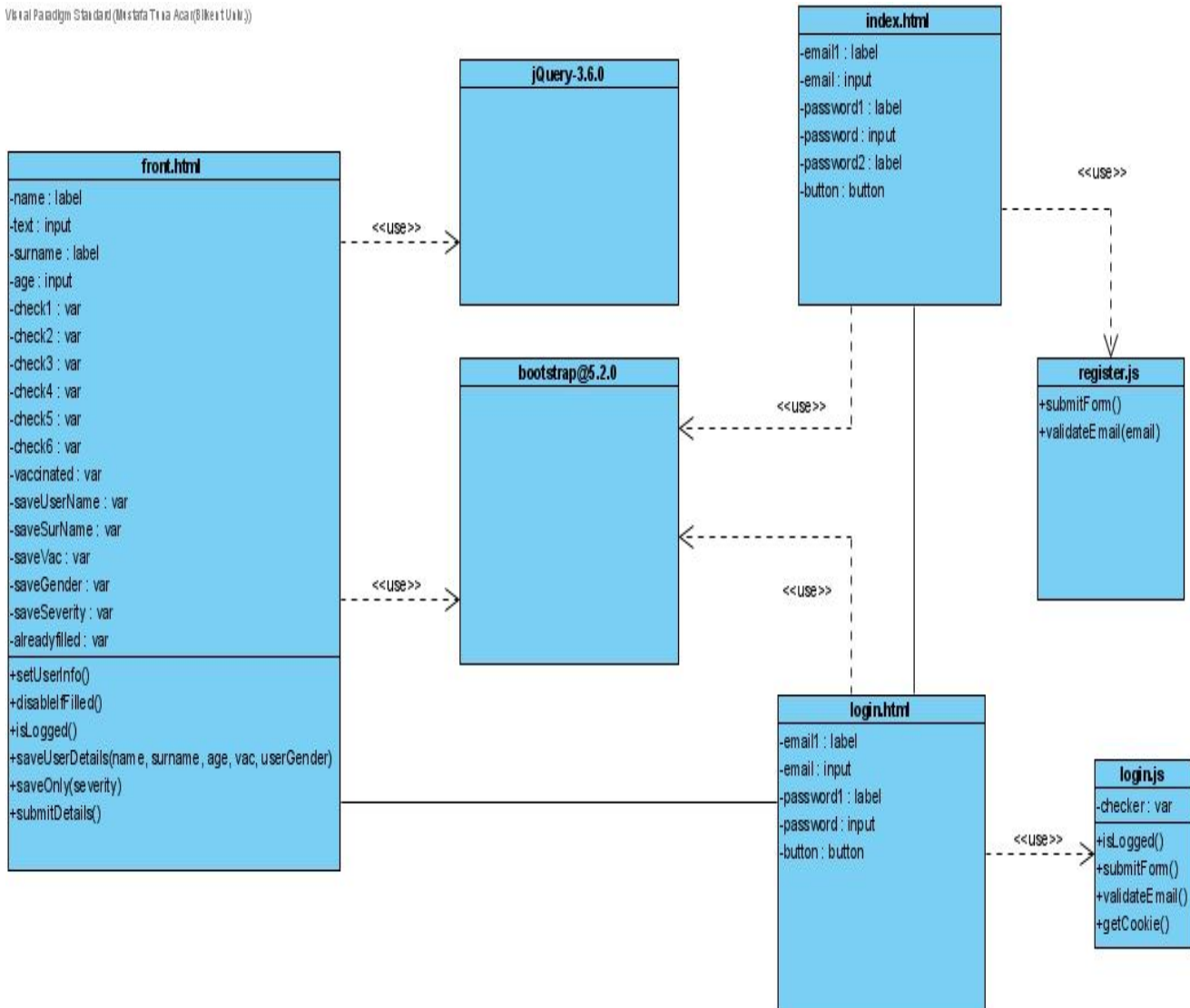


Figure 18: Class Diagram

2.2 Use Case Diagram

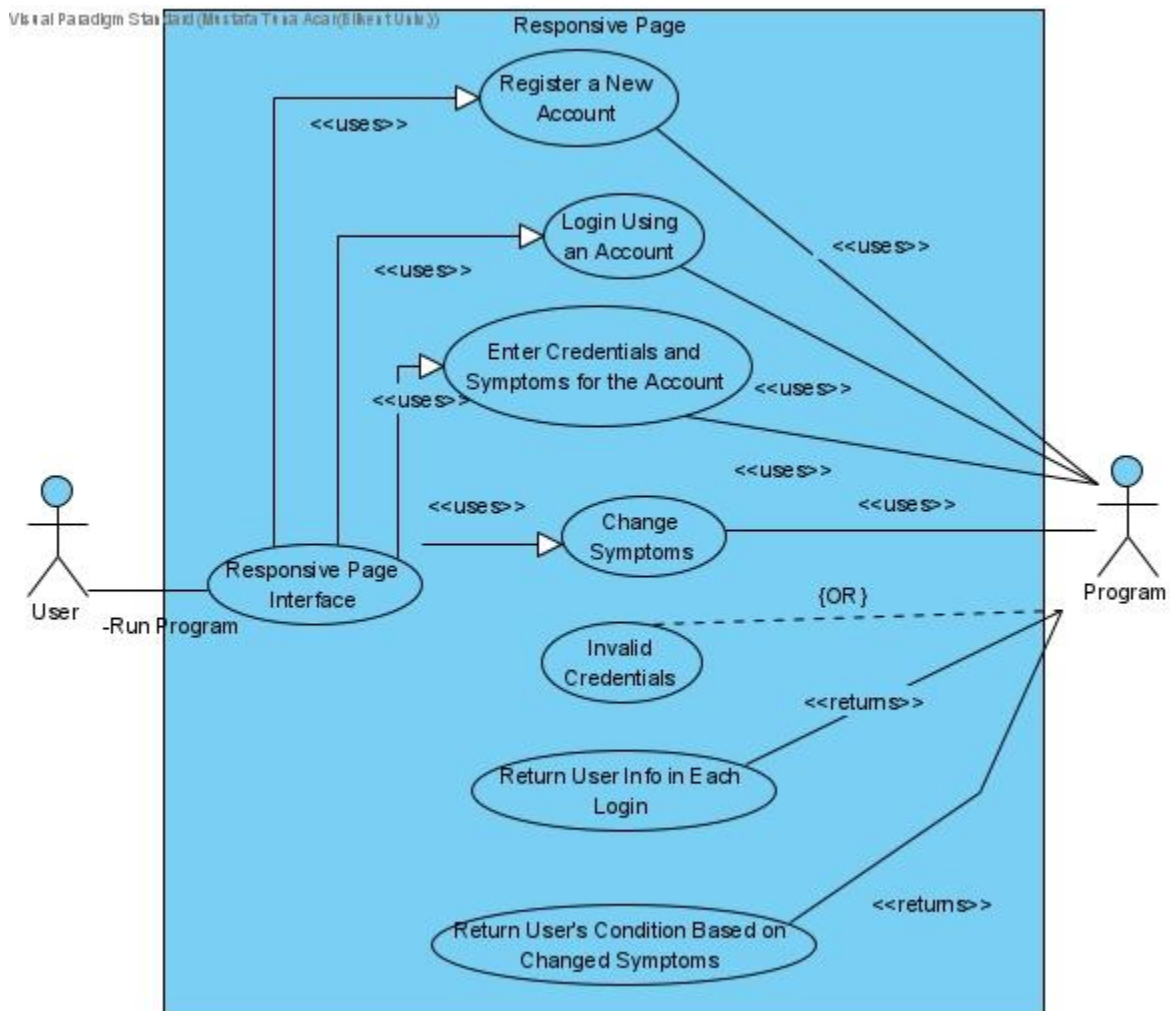


Figure 19: Use Case Diagram

2.3 Sequence Diagram

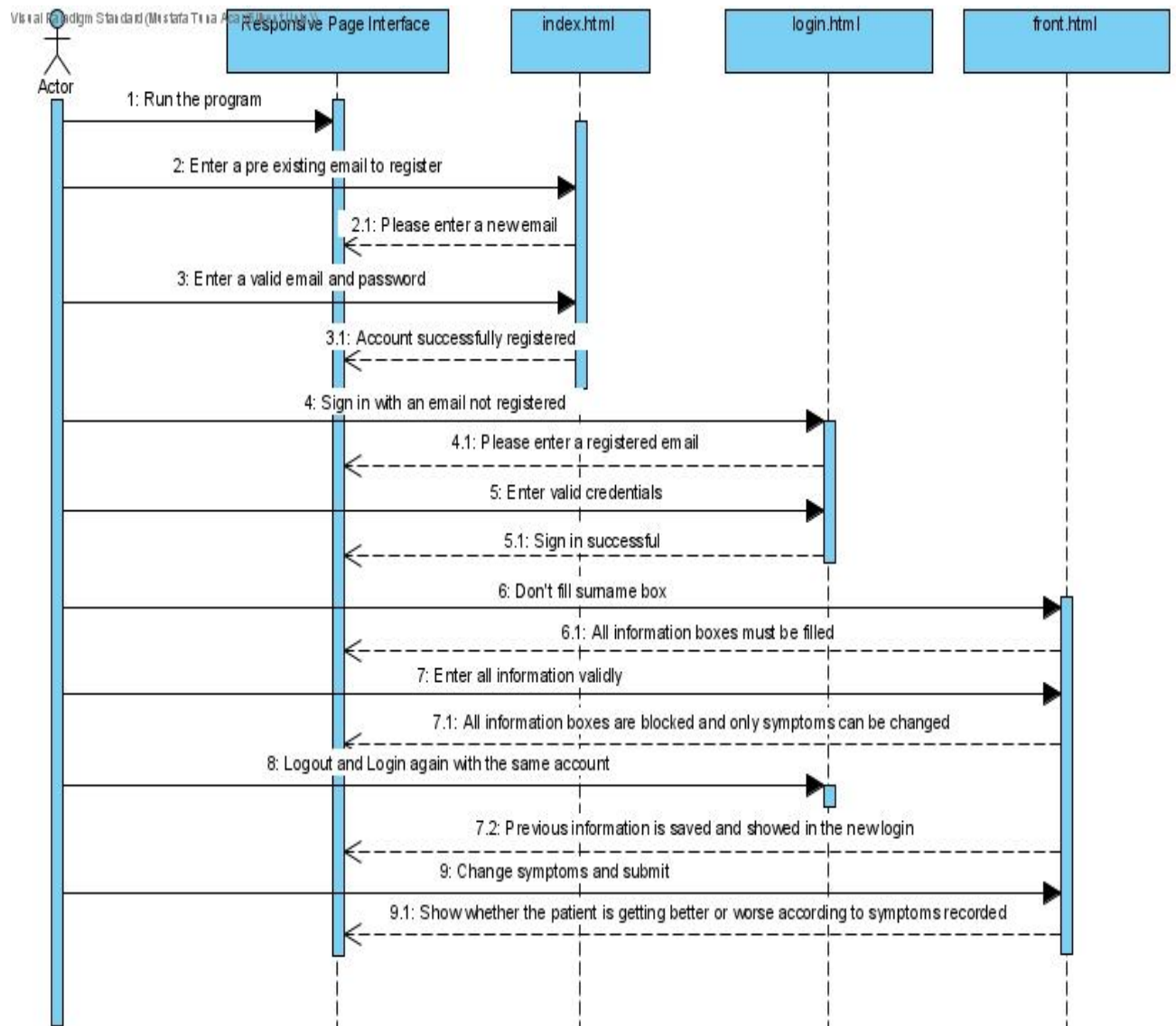


Figure 20: Sequence Diagram

3 Coverage of Unit Tests Calculation

Using Sonarqube we found that we had a correlation with our calculations as seen below;

$$\text{Coverage} = (\text{Covered Conditions} + \text{Covered Lines}) / (\text{Conditions to Cover} + \text{Lines to Cover})$$

- Covered Conditions = conditions_to_cover - uncovered_conditions
- Covered Lines = lines_to_cover - uncovered_lines
- Conditions to Cover = total number of conditions (conditions_to_cover)
- Lines to Cover = total number of executable lines (lines_to_cover)

$$\text{Coverage} = (5 + 280) / (5 + 349) = 0.80 \text{ Meaning that we have a coverage of 80\%.}$$

4 Unit Testing Experience

4.1 Development Velocity

There was definitely an improvement in development velocity by utilizing unit tests. An important factor in the benefits obtained was that our project was of a smaller scale and therefore did not have that many functionalities and therefore fewer unit tests. The unit tests helped us to localize our errors and have a methodological approach in solving them. A unit test that called several functions was broken into a few simple tests and whenever that test failed it was easy to identify the problem. This enabled us to dynamically fix our code and eliminate lag during the coding process.

Additionally, the way JUnit progresses through test cases, and how it marks them as successful or failed is really useful for bigger scaled projects. We can easily see in which test case an error has occurred and quickly fix it, rather than having to observe all of the testings. It speeds up the development exponentially.

4.2 Code Quality

Unit testing helped increase the code quality via the following functionalities:

- a) Enabled us to check if the code was working and that it was not throwing any errors during the runtime process and it is running successfully.
- b) It enabled us to pinpoint functions in the code and check if the function was returning correct values as intended and also whether the web application was in the intended state after the function had been called during the runtime. In some cases, the code was rewritten and the functions were rewritten to ensure the functionality of the web application.

- c) In addition, it was checked whether the functions were throwing correct errors as well. Certain error conditions were simulated and it was seen whether the functions handled the simulated error conditions correctly and were giving the intended errors.