Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**13**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Conduct a code review of your semester project to identify and address potential issues before integration. |
| 2 | Implement automated tools to perform static code analysis also find cyclomatic complexity of any feature or module in your project.Choose a static analysis tool (e.g., pylint, pytest,radon & selenium). |
| 3 | In this task, your goal is to perform a simple validation of a web page using Selenium. Start by installing the Selenium WebDriver for your preferred browser (e.g., Chrome). Identify a specific element on a webpage (e.g., a button, input field) and create a Selenium script in Python or another language of your choice to interact with and validate that element. For instance, you can automate clicking a button and checking if the expected result, such as a new page or a pop-up, occurs. Finally, document the steps taken and any challenges faced during the process. This task provides a foundational understanding of Selenium for web automation. |
| 4 | Deploy the module of semester project on github |
|  |  |

Submitted On:

02/01/2024

(Date: DD/MM/YY)

**Task No 1:** Conduct a code review of your semester project to identify and address potential issues before integration.

**Solution:**

**Code Structure and Readability**

Modularity: Are functions and modules appropriately divided for clarity and reusability?

Naming Conventions: Do variables, functions, and classes follow consistent and meaningful naming conventions?

Comments and Documentation: Are comments provided where necessary to explain complex logic or unclear code? Is the documentation up-to-date?

Code Formatting: Is the code consistently formatted, following a style guide?

**Security**

Input Validation: Are user inputs validated thoroughly to prevent SQL injection, XSS, and other security vulnerabilities?

Authentication and Authorization: Is the authentication process robust? Are sensitive operations restricted to authorized users only?

Data Encryption: Are sensitive data like passwords and payment information properly encrypted?

Session Management: Is session handling secure to prevent unauthorized access?

**Performance**

Database Queries: Are database queries optimized, avoiding unnecessary calls or large data retrievals?

Caching Strategies: Are caching mechanisms implemented where appropriate to improve performance?

Resource Usage: Does the code efficiently use server resources like memory and CPU?

Error Handling: Are errors handled gracefully without impacting performance?

**Functionality**

Feature Completeness: Does the app cover all specified functionalities, meeting requirements and user stories?

Edge Cases Handling: Are edge cases and boundary conditions considered and handled appropriately?

Testing: Is there sufficient unit testing, and have edge cases been covered in the tests?

Usability: Is the UI/UX intuitive and responsive across different devices and browsers?

**Maintainability**

Code Duplication: Are there instances of duplicated code that could be refactored into reusable functions or classes?

Dependency Management: Are dependencies managed properly, ensuring versions are up-to-date and compatible?

Version Control: Are commits descriptive, and does the version history reflect logical progression?

**Miscellaneous**

Accessibility: Is the app accessible to users with disabilities?

Scalability: Is the app designed to scale with increased user loads?

Internationalization and Localization: Does the app support multiple languages and regions?

By systematically reviewing the codebase based on these aspects, you can identify potential issues and address them proactively before integration.

**Task No 2:** Implement automated tools to perform static code analysis also find cyclomatic complexity of any feature or module in your project.Choose a static analysis tool (e.g., pylint, pytest,radon & selenium).

**Solution:**

export async function login(req, res) {

  const { username, password, role } = req.body;

  const userData = await userModel.findOne({ username });

  if (!userData || role !== userData.role) {

    return res.status(404).json({ message: 'user not found' });

  }

  const isPasswordCorrect = await comparePassword(password, userData.password);

  if (!isPasswordCorrect) {

    return res.status(401).json({ message: 'Incorrect Password.' });

  }

  const user = { username: username };

  const accessToken = generateToken(user);

  const refreshToken = jwt.sign(user, process.env.REFRESH\_TOKEN\_SECRET);

  refreshTokens.push(refreshToken);

  res.json({

    accessToken: accessToken,

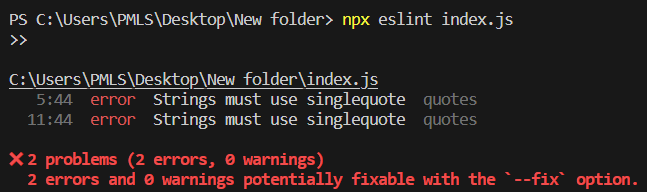
    refreshToken: refreshToken,

    username: username,

  });

}

**Output:**



**Task No 3:** In this task, your goal is to perform a simple validation of a web page using Selenium. Start by installing the Selenium WebDriver for your preferred browser (e.g., Chrome). Identify a specific element on a webpage (e.g., a button, input field) and create a Selenium script in Python or another language of your choice to interact with and validate that element. For instance, you can automate clicking a button and checking if the expected result, such as a new page or a pop-up, occurs. Finally, document the steps taken and any challenges faced during the process. This task provides a foundational understanding of Selenium for web automation.

**Solution:**

const { Builder, By, until } = require("selenium-webdriver");

async function validateButtonFunctionality() {

  const driver = await new Builder().forBrowser("chrome").build();

  try {

    await driver.get("https://example.com");

    const button = await driver.wait(

      until.elementLocated(By.id("myButton")),

      10000

    );

    await button.click();

    await driver.wait(until.elementLocated(By.className("popup")), 10000);

    const isPopupDisplayed = await driver

      .findElement(By.className("popup"))

      .isDisplayed();

    if (isPopupDisplayed) {

      console.log("Validation successful! Pop-up is displayed.");

    } else {

      console.log("Validation failed! Pop-up is not displayed.");

    }

  } catch (error) {

    console.error("Validation failed:", error);

  } finally {

    await driver.quit();

  }

}

validateButtonFunctionality();

**A close-up of a website

Description automatically generatedOutput:**

**A screenshot of a computer

Description automatically generated**

**Task No 4:** Deploy the module of semester project on github

**Solution:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**