**Assignment: Continuous Integration with GitLab**

This assignment is designed to balance technical learning with practical exposure. It introduces CI concepts without requiring deep technical expertise, preparing students for future roles where CI/CD is essential.

**Overview**

In this assignment, you will gain hands-on experience with Continuous Integration (CI) by setting up a pipeline on GitLab to run two security scans on a JavaScript program. This activity will introduce you to key CI concepts, such as pipelines, stages, and jobs, and provide a foundation for understanding how CI can improve software quality.

**Learning Objectives**

1. Understand the purpose of CI in software development.
2. Set up and trigger a basic CI pipeline on GitLab.
3. Perform two security scans:
   1. A **JSHint** scan to identify potential errors and enforce coding standards in JavaScript files, improving code quality and maintainability.
   2. The **secret detection analyzer** to scan for sensitive information, such as API keys or passwords, to prevent accidental exposure and enhance application security.
4. Use Git to push code to a GitLab repository.
5. Reflect on the role of CI in collaborative development.

**Assignment Instructions**

**Part 1: Setup**

1. **Write a JavaScript Program:**
   * Create a JavaScript file (validatepalindrome.js) that:
     + Accepts an array of strings as input.
     + Use a function to determine if each string is a palindrome. A palindrome is a word, phrase, number, or sequence of characters that reads the same backward as forward, ignoring capitalization and spaces. A string of length one or zero is a palindrome.
     + Return a list of strings that are palindromes.
     + Recommend: pre-populate the array as a constant or as a var in order to simplify the programming.
   * Example input and output:
     + Input: ["racecar", "hello", "Level", “I”, "world", "mad am", "12321", "not a palindrome"]
     + Output: racecar, Level, I, mad am, “\12321]
     + Output should be displayed on the console using console.log()
   * Tips:
     + Divide the program into functions
     + Use the toLowerCase() and replace() functions for cleaning the input.
       - Example: in\_string.toLowerCase(); // converts in\_string to all lower case
       - Example: new\_string.replace(" ", ""); // replaces spaces with nulls, eliminates spaces
     + Use the split(), reverse(), and join() functions to reverse a string.
       - Example: return in\_string.split('').reverse().join(''); // This code reverses in\_string, for example, if in\_string = “Mike” this code will produce “ekiM”
2. **Initialize Git:**
   * Commit your validatepalindrome.js program to your local Git repository.
3. **Push to GitLab:**
   * Create an account on GitLab at [www.gitlab.com](http://www.gitlab.com).
   * When registering, you can say that you will be the only person using GitLab.
   * Create a new project on GitLab.
     + Group should be INFO465
     + Project name should be “simple\_pipeline\_build”
     + Do not select a template.
   * Two options to add your .js program
     + Option 1: Add the GitLab remote to your local repository and push your code:
       - Will need to create a SSH key. Instructions are posted on Canvas in the file “ssh\_key\_for\_gitlab.docx”
       - git remote add origin <GitLab-repository-URL>
       - git push -u origin main
     + Option 2: Use the GitLab GUI to add and commit a file.
       - Within your project, click on the + sign.
       - Select “Upload file”

**Part 2: Configure CI Pipeline**

1. **Add the .gitlab-ci.yml File:**
   * Create a .gitlab-ci.yml file in your project directory:
     + Click the **"+" (plus)** button then **"New file"**
     + In the "File name" field, enter “.**gitlab-ci.yml”**.
     + Define the CI/CD pipeline. Add the code from .git-ci.yml which is posted on Canvas.
       - The code creates a pipeline defines two jobs, secret\_detection and security\_scan, both in the test stage. The secret\_detection job uses a security analyzer to detect sensitive information in the code and generates a JSON report. The security\_scan job runs JSHint to check JavaScript files for code quality issues, saving the results as an artifact and allowing the job to fail without stopping the pipeline.
     + Commit the file.
2. **Run the Pipeline:**
   * Check your GitLab project’s CI/CD > Pipelines section to monitor the pipeline execution.
   * Verify that the pipeline completes successfully.

**Part 3: Submission**

Submit the following:

1. A screenshot of your pipeline showing a successful run.
2. Your JavaScript file (validatepalindrome.js).
3. Download the results (artifacts) from your run:
   * Go to Pipelines, select your most recent pipeline run by clicking on the status (should be Passed)
   * Click on each of the jobs, then click on “Download” in the Job Artifact section on the right-hand side of the screen.
4. A brief analysis (100-150 words) of the scan results:
   * Identify any sensitive information detected by the secret detection job and explain how to address it.
   * Highlight key findings from the JSHint report and propose changes to improve code quality.
   * What happens if you alter your .js program and add this line of code?

var test\_ssn = ‘222-33-4444’? Include your thoughts in the analysis.

1. A reflection (100-150 words) addressing:
   * What you learned about CI.
   * How CI can improve software development quality and collaboration.

**Rubric**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exemplary (90-100)** | **Proficient (75-89)** | **Developing (50-74)** | **Incomplete (<50)** |
| **JavaScript Program** | Fully functional and meets the specification. | Functional with minor issues. | Significant errors or incomplete implementation. | Missing or non-functional. |
| **GitLab Setup** | Code successfully pushed and visible on GitLab. | Code pushed with minor setup issues. | Errors in pushing code; setup incomplete. | Code not pushed to GitLab. |
| **CI Pipeline Execution** | Pipeline executes successfully without errors. | Pipeline executes with minor errors or warnings. | Pipeline setup incomplete or fails to execute. | No evidence of pipeline execution. |
| **Reflection** | Insightful, detailed, and connects CI concepts to practice. | Adequate but lacks depth or connection to CI concepts. | Superficial or vague reflection. | Missing or fails to address the reflection prompts. |
| **Submission Quality** | All required files and screenshots submitted; well-organized. | Most files submitted but minor omissions. | Several required components missing. | Incomplete submission or disorganized. |