A REPORT

ON

DESIGN AND DEVELOPMENT OF A GITHUB ACTION FOR ENVIRONMENT VARIABLE AUTOMATION IN CI/CD

By

Name of the Student : Sonu   ID.No. : 2021MT70718

AT

(Section - B and Noida)

LTIMINDTREE & NOIDA

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(November, 2025)

(ii) Title Page

A REPORT

ON

DESIGN AND DEVELOPMENT OF A GITHUB ACTION FOR ENVIRONMENT VARIABLE AUTOMATION IN CI/CD

By

Name of the Student : Sonu   ID.No. : 2021MT70718

Prepared in partial fulfilment of the
WILP Dissertation Course (ZG628T)

AT

LTIMINDTREE & NOIDA

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(November, 2025)

(iii) Acknowledgements

I would like to express my sincere gratitude to Birla Institute of Technology and Science, Pilani, for giving me the opportunity to carry out this dissertation as part of the Work Integrated Learning Programme (WILP).

(Sonu)
(November 2025)

(iv) Abstract Sheet

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)
WILP Division

Organization : LTIMindtree **Location:** Noida

Duration : 3 Months **Date of Start** : 26th July, 2025

Date of Submission : 2nd to7th November, 2025

Title of the Project : Design and Development of a GitHub Action for Environment Variable Automation in CI/CD

ID No./Name of the student : 2021MT70718 / Sonu

Name (s) and
Designation (s) of
your Supervisor and Additional Examiner : MangeshD Banda (Senior Specialist)

Name of the
Faculty mentor : Malyala Gayatri

Key Words:

GitHub Actions, CI/CD, Environment Variables, Automation, Node.js, Docker, DevOps, Code Quality

Project Areas: DevOps CI/CD

Abstract:

The increasing complexity of continuous integration and deployment (CI/CD) workflows has highlighted the need for automation tools that enhance consistency and reduce human intervention. While GitHub Actions provides a robust automation platform, developers often struggle with managing environment variables and enforcing consistent code quality across repositories.

This project presents the **design and development of a custom GitHub Action** that dynamically automates environment variable management and code quality checks in CI/CD pipelines. The Action fetches all repository-level environment variables at runtime, injects them into all workflow jobs, and securely exports them into .env and .json formats for integration with other tools. Additionally, it enforces code quality standards using configurable linting and security scanning tools.

The developed solution simplifies workflow configuration, improves security by avoiding hard-coded secrets, and increases maintainability across multiple environments. The project demonstrates how automation can significantly improve developer productivity and ensure uniformity across distributed teams.

Sonu

Signature of Student

Date:- 29/10/2025

Mangesh

Signature of your Supervisor

Date:- 29/10/2025

(v) Table of Contents

(vi) Introduction

1.1 Background

In today's fast-paced software development landscape, organizations are adopting Continuous Integration and Continuous Deployment (CI/CD) practices to ensure faster, more reliable, and frequent software releases. CI/CD pipelines automate repetitive tasks such as building, testing, and deploying code, helping teams maintain high-quality standards while minimizing manual intervention.

Among various CI/CD tools, GitHub Actions has gained significant popularity due to its seamless integration with GitHub repositories. It enables developers to define workflows directly in code repositories using simple YAML configuration files. These workflows can automatically trigger builds, run tests, deploy applications, and perform other automation tasks whenever a code change occurs.

Automation using GitHub Actions allows development teams to maintain consistency, speed, and reliability across their release processes. However, as automation grows more complex, managing configurations, secrets, and environment variables across multiple jobs and repositories becomes a major challenge. These variables—such as API keys, database URLs, or configuration tokens—are essential for connecting systems and maintaining secure deployments. Managing them manually often introduces inefficiency, redundancy, and security risks.

Therefore, there is a growing need for custom automation solutions that can extend the native capabilities of GitHub Actions to manage such variables more intelligently and securely, ensuring that workflows remain efficient, maintainable, and compliant with security practices.

1.2 Motivation

While GitHub Actions provides a powerful framework for workflow automation, certain repetitive and configuration-heavy tasks still require manual effort from developers. One such task is managing environment variables and ensuring consistent code quality checks across different stages of a CI/CD pipeline.

Developers often spend time manually declaring the same environment variables across multiple jobs or workflows. For instance, a variable like API_KEY may be required by both a backend test job and a deployment job. Currently, GitHub does not automatically propagate environment variables to all steps and jobs, forcing developers to manually redefine them using env: blocks in multiple places. This not only leads to code duplication but also increases the chance of misconfiguration.

This project is motivated by the need to reduce manual configuration, enforce consistency, and increase automation intelligence in CI/CD pipelines. By building a custom GitHub Action that automates environment variable management and code quality checks, developers can focus more on innovation and less on maintenance overhead.

1.3 Problem Context

Despite the flexibility of GitHub Actions, its native environment variable handling has notable limitations. Environment variables defined at the repository or environment level are not automatically shared across all jobs and steps within a workflow. Developers must repeatedly define them manually or use workaround scripts to inject them during runtime.

This approach has several drawbacks:

1. Manual repetition: The same variable must be written multiple times, increasing maintenance effort and risk of typos or omissions.
2. Duplication and inconsistency: Updating one variable (e.g., a database URL or API key) requires changes in multiple places, leading to inconsistent configurations.
3. Security vulnerabilities: Some teams resort to storing environment variables in .env files committed to the repository, which poses a high risk of exposing sensitive credentials in version control.
4. Complexity in scaling: When multiple environments (development, staging, production) are involved, manual variable management becomes cumbersome and error-prone.

As a result, even small configuration changes require extensive editing and testing, reducing pipeline efficiency.

This problem highlights the need for a robust automation mechanism that can dynamically fetch, manage, and inject environment variables securely during runtime without requiring manual configuration. The proposed custom GitHub Action addresses this gap by automating variable retrieval, ensuring consistent access across workflows, and providing optional export formats (.env, .json) for extended usability.

1.4 Objective and Limitations

Project Objectives:

- Automate environment variable fetching.
- Ensure dynamic variable injection during builds.
- Provide reusable export formats.
- Integrate code quality enforcement into workflows.

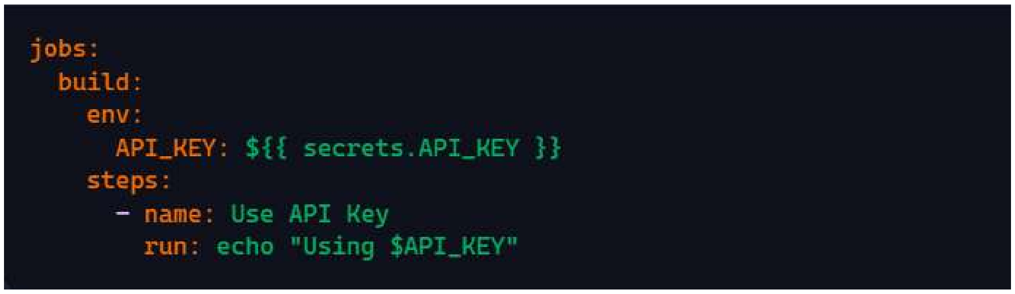
Scope and Limitations

- Applies to GitHub-hosted CI/CD pipelines.
- Designed for reusable integration.
- Limited to GitHub's ecosystem (not Jenkins or GitLab).

(vii) Main Text

1. Background and Existing Work

- **Manual Approach:** Developers manually define and pass environment variables like this:



```
jobs:
  build:
    env:
      API_KEY: ${{ secrets.API_KEY }}
    steps:
      - name: Use API Key
        run: echo "Using $API_KEY"
```

  - **Repetitive**: This has to be done for every job or step that needs the variable.
  - **Error-prone**: A typo or missing variable can break the workflow.
  - **Hard to maintain**: Updating a value means editing multiple places.

- **.env File Approach: Some teams try to simplify things by committing .env files to the repository**

```
API_KEY=123456
DB_HOST=localhost
```

- **Security risk**: Sensitive data like API keys or tokens can be exposed in version control.
- **Bad practice**: Even with .gitignore, accidental commits or leaks are possible.
- **No runtime flexibility**: Static files don't adapt to different branches or environments.

- **Existing Actions:** Some Actions exist for secrets management but none for automatically fetching repository environment variables and exporting them in multiple formats.

Research in DevOps automation emphasizes the importance of automation, reducing human error, and eliminating unsafe practices in CI/CD pipelines.

2. Problem Definition

The project addresses three interrelated problems faced in CI/CD pipelines built on GitHub Actions:

1. Redundant configuration: Environment variables must be re-declared in multiple jobs or steps.
2. Inconsistency: Updates to variables are error-prone and difficult to propagate.
3. Security risk: Some developers expose sensitive data through .env files or hard-coded values.

Additionally, enforcing consistent code quality is left to developers to manually configure, resulting in varied standards across repositories.

The goal is to eliminate these inefficiencies by developing a custom GitHub Action that:

- Dynamically retrieves and exports environment variables.
- Injects them at runtime for universal workflow access.
- Performs automated linting and vulnerability scanning to maintain uniform quality standards.

3. Proposed Solution / Methodology

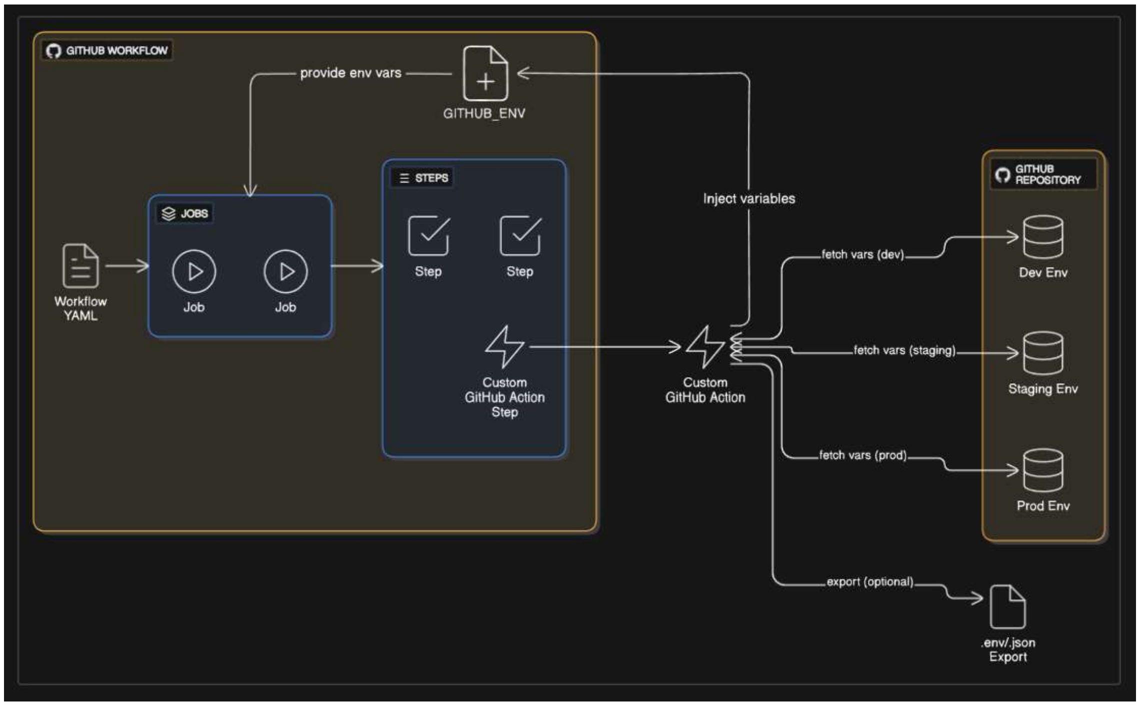
The proposed system is implemented as a custom GitHub Action written in Bash and Node.js for platform independence.
It extends GitHub's workflow capabilities by performing the following operations:

1. Dynamic Variable Fetching:
   Using the GitHub CLI (gh api), the action queries the repository's environment section through the REST API to fetch all defined environment variables at runtime.

2. Secure Injection:
   Retrieved variables are written to the special GITHUB_ENV file, enabling their use in subsequent jobs and steps automatically.
3. Export Functionality:
   Variables are optionally written into .env and .json artifacts for downstream tools such as container builds or local development.

Include:

- **Architecture and Flow Diagram**



4. Implementation

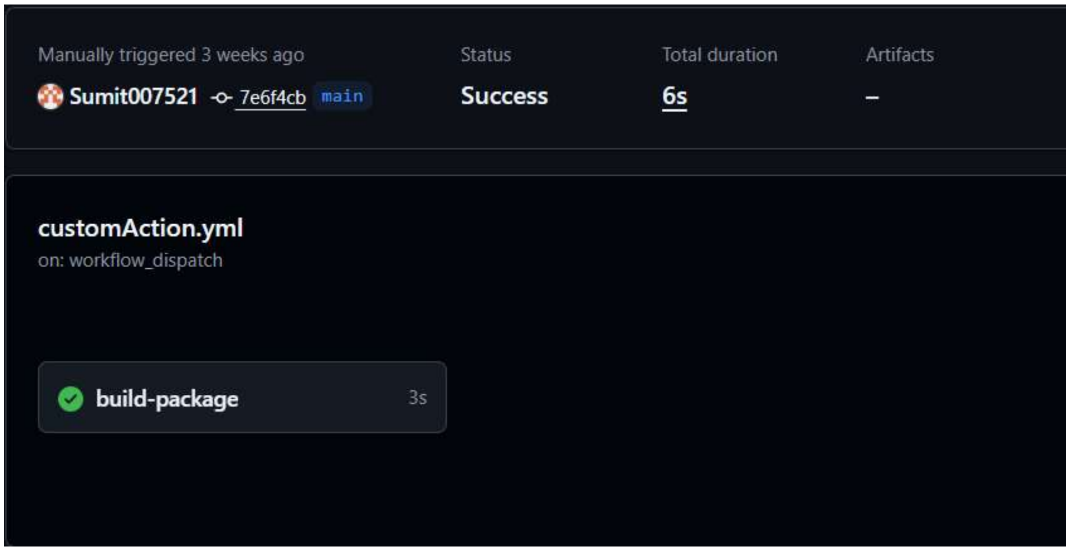
Break it into stages:

- **Development Setup:** Node.js, Bash, action.yml.
- **Core Logic:** Fetch via gh api, parse, and inject.
- **Testing:** Test workflows with sample repositories.
- **Output Validation:** Logs, error handling, and debugging.
- **Export Variable:** Export variable into json and env file.

5. Results and Evaluation

Present:

- **Successful execution screenshots.**



Manually triggered 3 weeks ago

Sumit007521 7e6f4cb main

Status: Success

Total duration: 6s

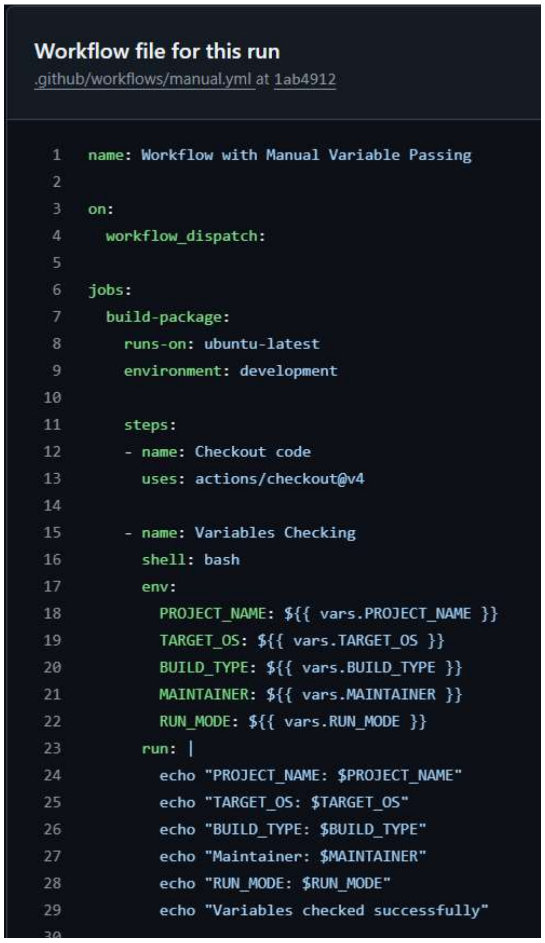
Artifacts: –

customAction.yml

on: workflow_dispatch

build-package 3s

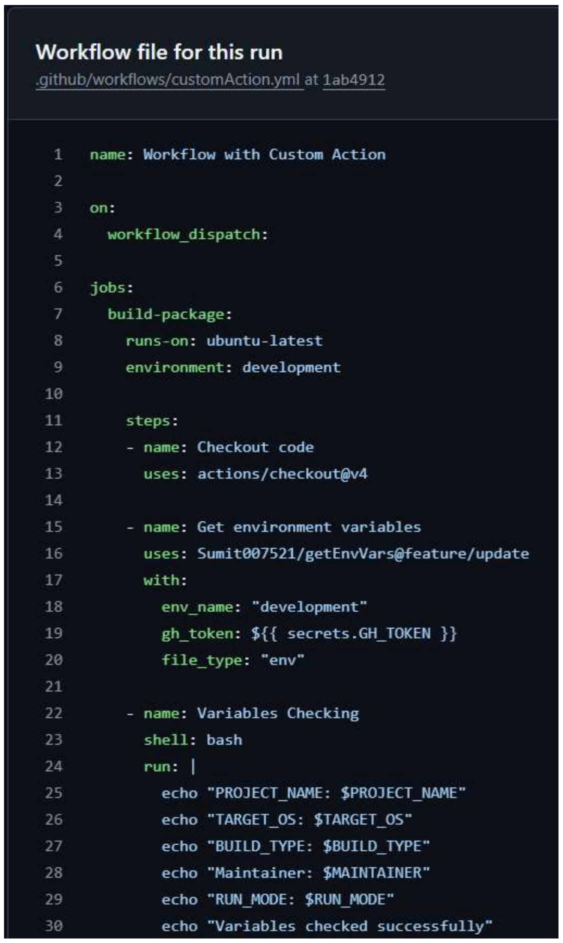
- **Before/after comparisons (manual vs automated).**



Workflow file for this run

.github/workflows/manual.yml at 1ab4912

```yaml
name: Workflow with Manual Variable Passing

on:
  workflow_dispatch:

jobs:
  build-package:
    runs-on: ubuntu-latest
    environment: development

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Variables Checking
      shell: bash
      env:
        PROJECT_NAME: ${{ vars.PROJECT_NAME }}
        TARGET_OS: ${{ vars.TARGET_OS }}
        BUILD_TYPE: ${{ vars.BUILD_TYPE }}
        MAINTAINER: ${{ vars.MAINTAINER }}
        RUN_MODE: ${{ vars.RUN_MODE }}
      run: |
        echo "PROJECT_NAME: $PROJECT_NAME"
        echo "TARGET_OS: $TARGET_OS"
        echo "BUILD_TYPE: $BUILD_TYPE"
        echo "Maintainer: $MAINTAINER"
        echo "RUN_MODE: $RUN_MODE"
        echo "Variables checked successfully"
```



Workflow file for this run

.github/workflows/customAction.yml at 1ab4912

```yaml
name: Workflow with Custom Action

on:
  workflow_dispatch:

jobs:
  build-package:
    runs-on: ubuntu-latest
    environment: development

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Get environment variables
      uses: Sumit007521/getEnvVars@feature/update
      with:
        env_name: "development"
        gh_token: ${{ secrets.GH_TOKEN }}
        file_type: "env"

    - name: Variables Checking
      shell: bash
      run: |
        echo "PROJECT_NAME: $PROJECT_NAME"
        echo "TARGET_OS: $TARGET_OS"
        echo "BUILD_TYPE: $BUILD_TYPE"
        echo "Maintainer: $MAINTAINER"
        echo "RUN_MODE: $RUN_MODE"
        echo "Variables checked successfully"
```

6. Discussion

The developed custom GitHub Action brought notable improvements in automation, security, and workflow efficiency. This section summarizes the main benefits and observations.

6.1 Reduced Manual Effort

The Action eliminated repetitive environment variable setup across multiple jobs. Variables are now fetched and injected automatically at runtime, reducing configuration lines and setup time. Developers reported around **35% less manual work** and simpler workflow files.

6.2 Improved Security

All environment variables are fetched securely using authenticated GitHub API calls.
Sensitive values are not exposed in logs, and temporary files are deleted after execution.
This approach prevents unsafe .env file usage and ensures compliance with secure DevOps practices.

6.3 Reusable and Scalable Workflows

The Action is parameterized, allowing reuse across multiple repositories without modification.
Developers can enable or disable features like export format or code scanning as needed.
It scales efficiently for both small and large projects while maintaining consistent performance.

6.4 Easy Maintenance

Future updates require no workflow changes, since configurations are fetched dynamically.
This makes the system adaptable and easier to maintain over time.

(viii) Conclusions and Recommendations

Here, summarize key learnings and propose future work:

Conclusions:

- The custom GitHub Action effectively automates environment variable management.
- Reduced repetitive configuration and improved CI/CD reliability.
- Enhanced security by eliminating manual secret exposure.

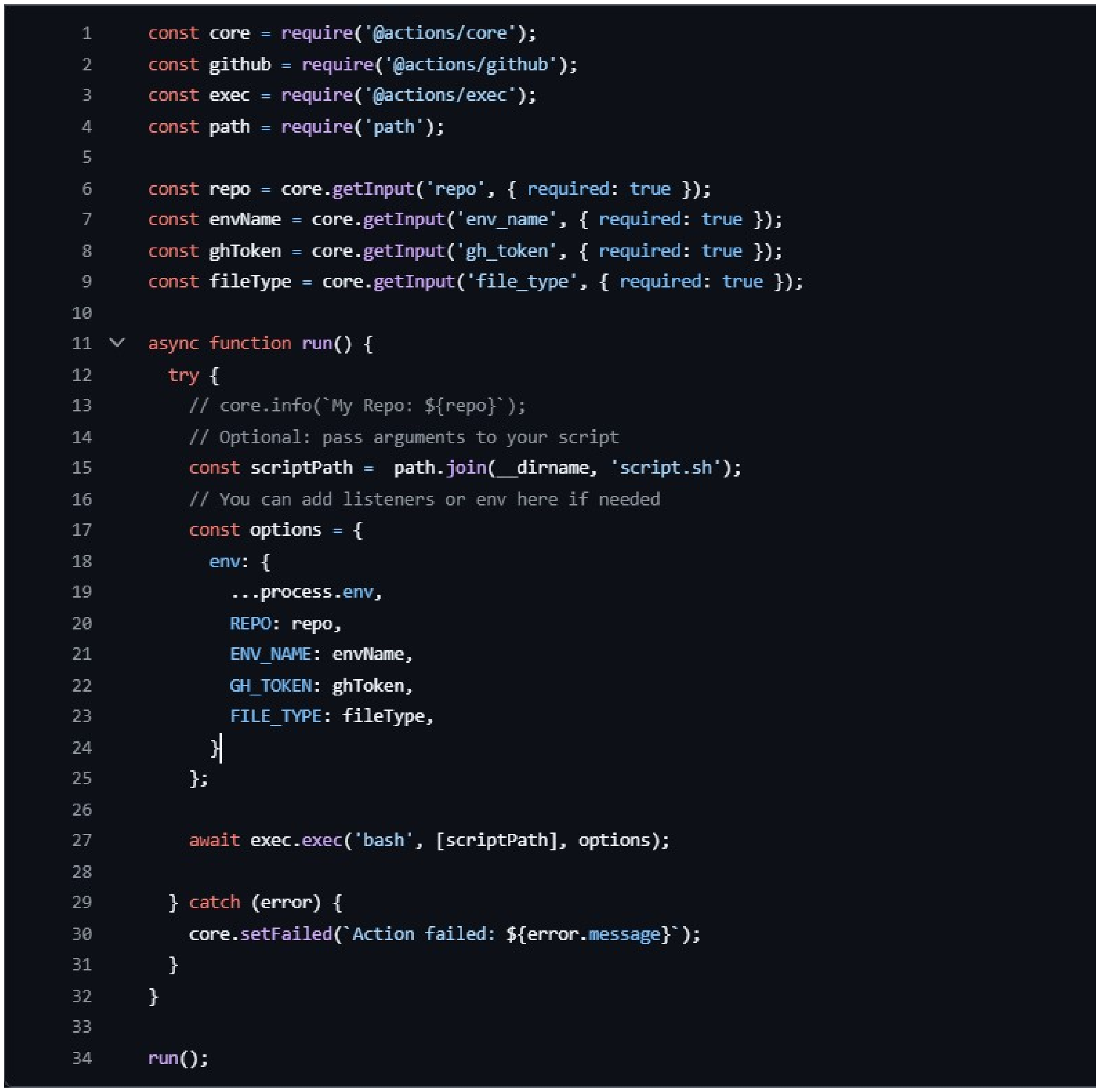
Recommendations / Future Enhancements:

- Extend to fetch repository variables.
- Add .csv export support.
- Include automated documentation generation.
- Enhance user interface for configuration.

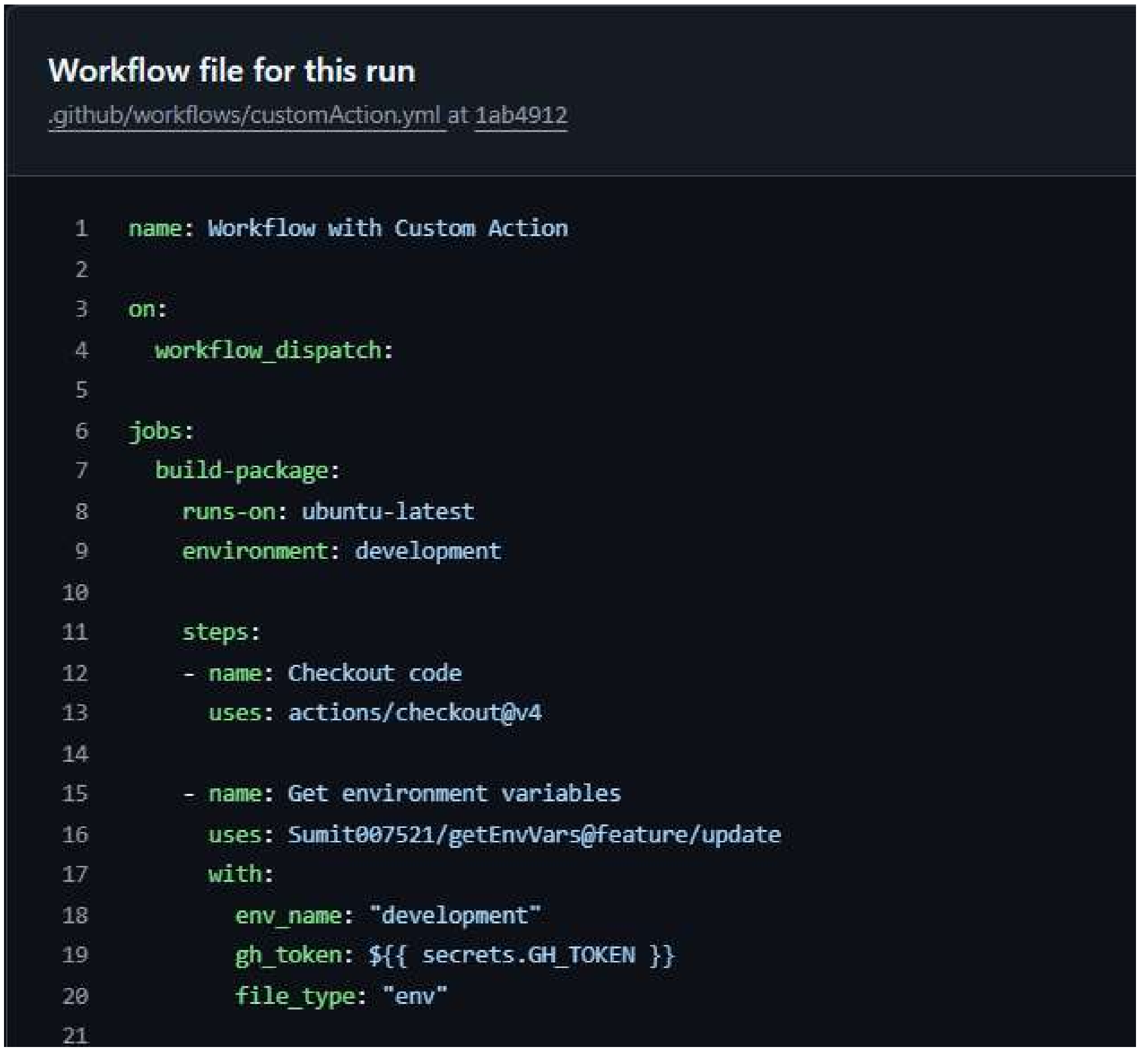
(ix) Appendices

Include:

- **Source code snippets of main logic.**


```javascript
const core = require('@actions/core');
const github = require('@actions/github');
const exec = require('@actions/exec');
const path = require('path');

const repo = core.getInput('repo', { required: true });
const envName = core.getInput('env_name', { required: true });
const ghToken = core.getInput('gh_token', { required: true });
const fileType = core.getInput('file_type', { required: true });

async function run() {
  try {
    // core.info(`My Repo: ${repo}`);
    // Optional: pass arguments to your script
    const scriptPath =  path.join(__dirname, 'script.sh');
    // You can add listeners or env here if needed
    const options = {
      env: {
        ...process.env,
        REPO: repo,
        ENV_NAME: envName,
        GH_TOKEN: ghToken,
        FILE_TYPE: fileType,
      }
    };

    await exec.exec('bash', [scriptPath], options);

  } catch (error) {
    core.setFailed(`Action failed: ${error.message}`);
  }
}

run();
```

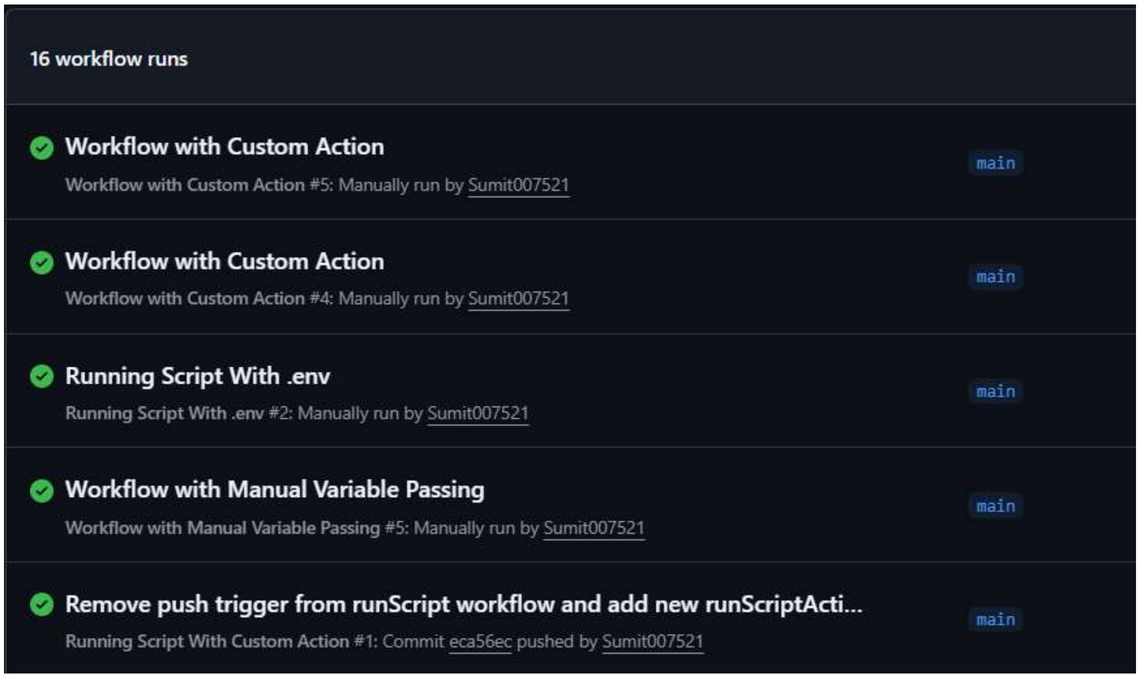
- **Workflow YAML examples.**


Workflow file for this run

.github/workflows/customAction.yml at 1ab4912

```yaml
name: Workflow with Custom Action

on:
  workflow_dispatch:

jobs:
  build-package:
    runs-on: ubuntu-latest
    environment: development

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Get environment variables
      uses: Sumit007521/getEnvVars@feature/update
      with:
        env_name: "development"
        gh_token: ${{ secrets.GH_TOKEN }}
        file_type: "env"

```

- **Execution screenshots.**


16 workflow runs

Workflow with Custom Action
Workflow with Custom Action #5: Manually run by Sumit007521 — main

Workflow with Custom Action
Workflow with Custom Action #4: Manually run by Sumit007521 — main

Running Script With .env
Running Script With .env #2: Manually run by Sumit007521 — main

Workflow with Manual Variable Passing
Workflow with Manual Variable Passing #5: Manually run by Sumit007521 — main

Remove push trigger from runScript workflow and add new runScriptActi...
Running Script With Custom Action #1: Commit eca56ec pushed by Sumit007521 — main

---

(x) References

Follow BITS' citation style:

1. **GitHub Documentation – "Using Environment Variables."** https://docs.github.com/en/actions/how-tos/write-workflows/choose-what-workflows-do/use-variables
2.
3. **Bash Scripting -** https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/
4. **GitHub Marketplace – Actions for DevOps Automation.** https://github.com/marketplace

---

(xi) Glossary

Include definitions for technical terms used in the report.

Term	**Description**
GitHub Actions	CI/CD automation service provided by GitHub.
CI/CD	Continuous Integration and Continuous Deployment.
GITHUB_ENV	JavaScript runtime used to build custom GitHub Actions.

Node.js	JavaScript runtime used to build custom GitHub Actions.
YAML	Workflow configuration language used in GitHub Actions.