COIMBATORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Autonomous | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with "A" Grade

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

U23CSP11 - OPEN SOURCE SOFTWARE PRACTICES

LABORATORY

2025 - 2026

CIET COIMBATORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Autonomous | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with "A" Grade

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of work done by Mr. / Ms. …… in the **U23CSP11 - OPEN SOURCE SOFTWARE PRACTICES LABORATORY** of this institution, as prescribed by the Anna University for the Fifth Semester Computer Science and Engineering, during the year 2025-2026.

Ms. R.NITHYADEVI M.E.,
ASSISTANT PROFESSOR
Department of Computer Science & Engineering,
Coimbatore Institute of Engineering and Technology,
Coimbatore – 641109.

Dr. K. PUSHPALATHA M.E., Ph.D.,
HEAD OF THE DEPARTMENT
Department of Computer Science & Engineering,
Coimbatore Institute of Engineering and Technology,
Coimbatore – 641 109.

Submitted for the University Practical Examination held on ……..…….. at Coimbatore Institute of Engineering and Technology, Coimbatore – 641 109.

Register Number:

Internal Examiner **External Examiner**

INDEX

S.NO.	DATE	LIST OF EXPERIMENTS	PAGE NO.	MARKS	SIGNATURE
1.		Exploration and Comparative Study of Open Source Licenses: MIT, GPL, Apache, and BSD			
2.a)		Version Control with Git – Practice Using Git CLI			
2.b)		Version Control with GitHub Desktop			
3.a)		GitHub Collaboration – Fork a Repository, Create Branches			
3.b)		GitHub Collaboration -Raise Issues, and Create Pull Requests			
4.		Build and Debug OSS Projects using Maven and VS Code			
5.		Bug Tracking and Issue Management using GitHub Projects			
6.a)		Documentation Contribution Using Markdown			
6.b)		Documentation Contribution Using MkDocs			
6.c)		Documentation Contribution Using Sphinx			
7.		Translation/localization-contribute translations using weblate, transifex and POEditor			

8.		Contribution Discovery- Use Up for Grabs and First Timers only platform to find beginner friendly issues			
9.		React .js Framework Basics			
10.		Create the GitHub Account to demonstrate CI/CD pipeline using Cloud Platform.			

Ex No : 1
Date :

Exploration and Comparative Study of Open Source Licenses: MIT, GPL, Apache and BSD

Aim:

To understand the importance and role of open source licenses, explore the terms and conditions of MIT, GPL, Apache, and BSD licenses and compare these licenses based on permissions, conditions, and limitations.

MIT License:

The MIT License is one of the most popular open-source software licenses.

It's known for being very short, simple, and permissive — meaning it gives almost complete freedom to use, copy, modify, and distribute the software, as long as we follow one small rule.

1. Permissions

Under the MIT License, we have **four main permissions**:

Permission	**Meaning**
Commercial Use	We can use the software in commercial projects — even sell it as part of our product or service.
Modification	We can modify or change the source code however we like — add new features, fix bugs, or use parts of it in another project.
Distribution	We can freely distribute copies of the software — original or modified versions — to anyone.
Private Use	We can use the software privately, without sharing it publicly or telling anyone.

2. Conditions

- The MIT License has only **one main condition**:
- We must include the **original license text** and **copyright notice** in any copies or substantial portions of the software.
- Whenever we **share** or **distribute** the software (either original or modified), we must **keep the MIT License file** and **the original author's name and copyright notice**.

- This ensures the original author is **credited** for their work.

3. Limitations

The MIT License includes **two important disclaimers** — these are legal protections for the author.

Limitation	Meaning
No Warranty	The software is provided "as is." The author does **not guarantee** that it works correctly or is free of errors.
Liability Disclaimer	If something goes wrong — like data loss, system damage, or security issues — the author is **not responsible** for any losses or damages.

Example Summary

Imagine we find an MIT-licensed Python library on GitHub.
We can:

- Use it in our commercial app
- Modify it to fit our needs
- Redistribute our version
- Keep the original license and author's name with our code
- Blame the author if it breaks something
- Expect any kind of support or warranty

GNU General Public License (GPL):

The GNU GPL is a "copyleft" license.

If we use, modify, or distribute GPL-licensed code, we must also release our code under the GPL.

1. Permissions

Under the GPL, we have these rights:

Permission	Meaning
Use	We can use the software for any purpose — personal, educational, or commercial.
Study	We can read and study the source code to understand how it works.
Modify	We can modify the software to fit our needs.
Distribute	We can share copies (original or modified), as long as we follow the GPL terms.

So like the MIT License, it's very permissive in what we can do, but the **conditions** are much stronger.

2. Conditions

Condition	Description
Source Code Access	If we distribute a GPL program (original or modified), we must also provide or make available the **complete source code**.
Same License (Copyleft)	Any modified version or derivative work must also be released under the **same GPL license**. We cannot make it proprietary.
License Notice	We must include the **original copyright notice** and the **GPL license text**.
No Additional Restrictions	We can't impose extra restrictions on users beyond what the GPL allows (e.g., we can't prevent others from sharing or modifying it).

Example:
If we take GPL software, modify it, and sell it — that's fine
But we must:

- Provide your source code
- Release it under GPL
- Include the license text and copyright notice
- We **cannot** make it closed-source or restrict others from using your version.

3. Limitations

Just like MIT, the GPL also has **no warranty or liability**.

Limitation	Meaning
No Warranty	The software is provided "as is," with no guarantee that it works correctly.
No Liability	The author is not responsible for any damage, loss, or issues caused by the software.

Example Scenario

Imagine we find a **GPL-licensed C++ library** for image processing.

- Use it in your company project
- Modify it to improve performance

- Redistribute it

- You must **share your modified code** if you distribute it
- You must **keep it under GPL**, not your own custom or closed license
- You cannot restrict others from modifying or redistributing your version

Apache License 2.0:

The **Apache License 2.0** is a **permissive open-source license** — similar to the MIT License, but **more detailed** and includes **explicit patent protection**.

It's maintained by the **Apache Software Foundation (ASF)**.

1. Permissions

Under the Apache 2.0 License

Permission	**Meaning**
Commercial Use	Use the software in commercial products or services — even sell it.
Modification	Modify, adapt, or extend the software as you like.
Distribution	Distribute the original or modified version.
Private Use	Can use it privately without sharing.
Patent Use	Granted a license to use any **patents** that the contributors might hold related to the software.

2. Conditions

The **Apache License** allows broad use, but it has several **requirements** when redistributing the software.

Condition	**Description**
Include License and Notice	Must include a copy of the Apache License 2.0 and a NOTICE file (if provided) in your distribution.
State Changes	If we modify the code, we must clearly state that we have changed it (e.g., "This file has been modified from the original").
Include Attribution	Must give credit to the original authors (usually through the NOTICE file).

Provide Source of License	Must say that the software is licensed under the Apache License 2.0.
Patent Termination	If we sue someone for patent infringement using this software, our patent license is automatically terminated.

Key Difference: Patent Clause

- This is the biggest difference between **Apache** and **MIT** or **GPL**.
- When we use software under the Apache License, the contributors **grant us a license to use any of their patents** related to the software.
- But if **we file a patent lawsuit** claiming the software violates your patent, we **lose that license**.
- This protects developers and users from patent-related legal issues.

3. Limitations

Like other open-source licenses, the Apache License also includes legal disclaimers:

Limitation	**Meaning**
No Warranty	The software is provided "as is," without any guarantees of performance.
No Liability	The author or organization is not responsible for damages or issues caused by using the software.

Example Scenario

Let's say we use **Apache-licensed software** (like Apache Kafka or Android) in your project.

- Use it in company's product
- Modify the code
- Sell the software

But we must:

- Include the **Apache License 2.0** text
- Keep the **NOTICE file** (if any) intact
- Mention if you made changes

- Credit the original authors

And we can't:

- Sue the original authors for patent infringement
- Expect them to fix bugs or provide warranty

BSD License (2-Clause or 3-Clause):

- **Permissions**: Use, modify, distribute, commercial use.
- **Conditions**: Include license, no endorsement clause (3-Clause).
- **Limitations**: No warranty.

Read and Retrieve the License Texts

- Visit the following links or use command line tools like wget or curl to download the license texts:
- MIT License
- GPLv3
- Apache 2.0
- BSD 3-Clause

Comparison Table :

Feature	MIT	GPLv3	Apache 2.0	BSD 3-Clause
Permissions	Yes	Yes	Yes	Yes
Conditions	Include license	License derivatives as GPL	State changes, include NOTICE	Include license, no endorsement
Copyleft	No	Yes (strong)	No (permissive)	No
Commercial Use	Yes	Yes	Yes	Yes
Modification Allowed	Yes	Yes	Yes	Yes
Patent Grant	No	Yes	Yes	No
License Compatibility	High	Low	Moderate	High
Simplicity	Very simple	Complex	Moderate	Simple

Result:

The difference between permissive and copyleft licenses and choosing an appropriate license for a given software project has been studied.

Ex No:2.a) **Version Control with Git – Practice Using Git CLI**
Date :

Aim:

To practice and understand basic version control operations using the Git Command Line Interface (CLI) such as initializing a repository, staging files, committing changes, creating branches, merging, and pushing changes to a remote repository (GitHub).

Software Required:

- Git Command Line Interface (CLI)
- GitHub Account
- Visual Studio Code / Command Prompt

Procedure:

1. Install Git-Download and install Git on your system and verify that it has been installed correctly and Configure Git User Information - Set your username and email so that your commits can be properly identified.

2. Create Project Directory -Create a new folder for your project and navigate into it.

3. Initialize Git Repository -Initialize a new Git repository inside the project folder.

4. Create and Add Files - Create a new file in the project folder. Check the repository status to see the untracked files and add the file to the staging area.

5. Commit Changes - Save the staged file to the repository by committing it with a proper commit message.

6. Create and Switch Branch -Create a new branch for development or new features and switch to that branch.

7. Modify File and Commit Again -Edit the file in the branch, stage the changes, and commit them with an appropriate message.

8. Merge Branch with Main -Switch back to the main branch and merge the changes from the feature branch into it.

9. Push Repository to GitHub -Connect the local repository to a remote repository on GitHub and push all committed changes so that the repository is updated online.

Program (Git CLI Commands):

Step 1: Configure Git
git --version
git config --global user.name "ABC"
git config --global user.email "ABC@example.com"

Step 2: Create project folder
mkdir git_practice
cd git_practice

Step 3: Initialize repository
git init

Step 4: Create and add a file
```
echo "<h1>Hello Git</h1>" > index.html
git status
git add index.html
```

Step 5: Commit changes
```
git commit -m "Initial commit with index.html"
```

Step 6: Create and switch to new branch
```
git branch feature
git checkout feature
```

Step 7: Modify file and commit
```
echo "<p>Feature branch update</p>" >> index.html
git add index.html
git commit -m "Updated index.html in feature branch"
```

Step 8: Merge feature branch to main
```
git checkout main
git merge feature
```

Step 9: Add remote repository and push
```
git remote add origin https://github.com/yourusername/git_practice.git
git push -u origin main
```

Output:

Result:

Basic Git operations such as initialization, committing, branching, merging, and pushing were successfully performed using the Git Command Line Interface (CLI).The local repository was successfully synchronized with a remote GitHub repository.

Ex No:2.b)
Date :

Version Control with GitHub Desktop

Aim:

To understand and practice basic version control operations using **GitHub Desktop** including creating repositories, committing changes, branching, merging, and syncing with a remote GitHub repository.

Software Required:

- GitHub Desktop Application
- GitHub Account
- Text Editor (VS Code, Notepad++, etc.)

PROCEDURE :

1. **Install GitHub Desktop**
   Download and install GitHub Desktop on your system.

2. **Sign in to GitHub**
   Open GitHub Desktop and sign in using your GitHub account.

3. **Create a New Repository**
   Create a new repository by providing the repository name, description, and local path. Initialize it with a README file.

4. **Add Files**
   Open the local repository folder and add new files for the project.

5. **Commit Changes**
   Stage the added files and commit them with an appropriate commit message.

6. **Create Branch**
   Create a new branch for development or new features and switch to it.

7. **Make Changes in Branch**
   Edit files in the branch, stage the changes, and commit them with a message.

8. **Merge Branch to Main**
   Switch back to the main branch and merge the changes from the feature branch into it.

9. **Push to Remote Repository**
   Upload all committed changes from the local repository to the remote GitHub repository.

PROGRAM / STEPS IN GITHUB DESKTOP:

1. **Create Repository:**

   File → New Repository → git_desktop_practice → Local Path → Initialize with README → Create Repository

2. **Add File:**

   Create index.html in local folder

   GitHub Desktop → Changes → Stage the file → Commit

3. **Create Branch:**

   Branch → New Branch → Name feature → Create Branch

4. **Edit and Commit in Branch:**

   Add content to index.html

   GitHub Desktop → Changes → Commit

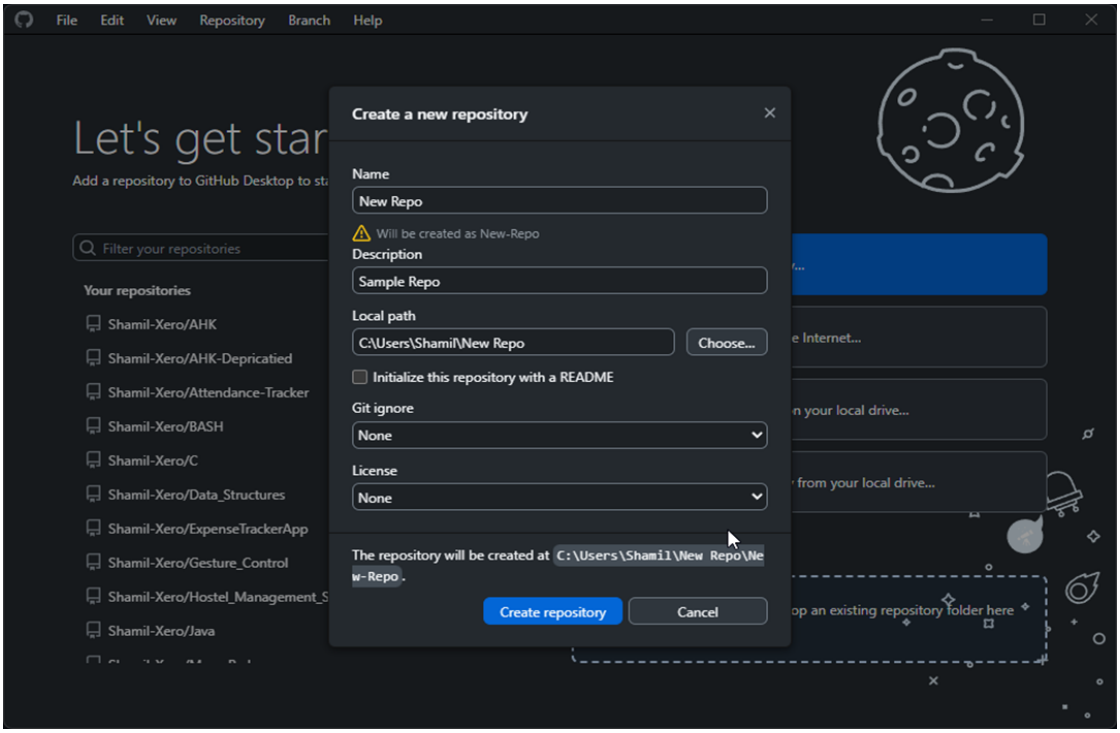
5. **Merge Branch:**

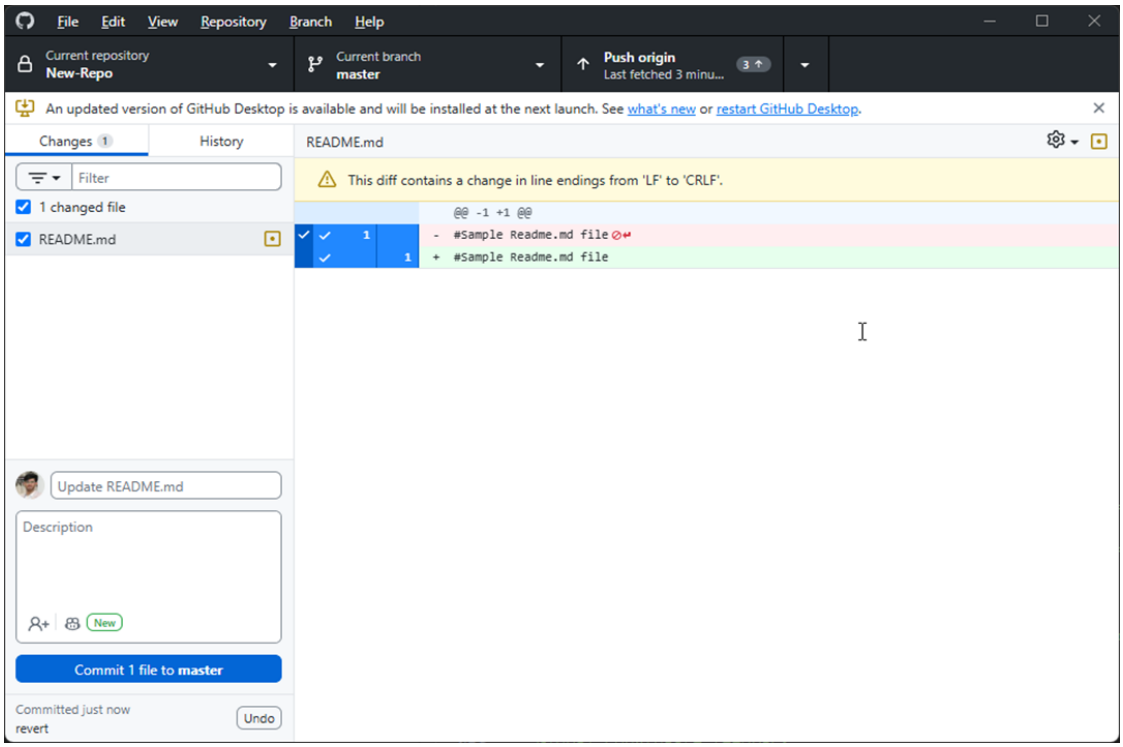
   Switch to main → Branch → Merge into Current Branch → Merge feature

6. **Push to GitHub:**

   Repository → Push → Sync changes to remote repository

Output:





Result:

The repository was successfully created and managed using **GitHub Desktop**. Files were added, committed, branched, merged, and successfully pushed to the remote repository on GitHub.

Ex No:3.a) **GitHub Collaboration – Fork a Repository, Create Branches**
Date :

Aim:

To practice collaborative workflows in GitHub by forking a repository, creating branches, making changes, and understanding how contributors can work together efficiently.

Requirements:

- GitHub account (https://github.com/)
- Git installed locally
- GitHub Desktop (optional)
- Text/code editor (e.g., VS Code)

Procedure:

1. Fork a Repository
   - Go to the repository you want to contribute to on GitHub.
   - Click the Fork button to create a personal copy of the repository in your GitHub account.
2. Clone the Forked Repository (Optional)

   Clone the forked repository to your local system to make changes using GitHub Desktop or Git CLI.
3. Create a New Branch
   - In your forked repository, create a new branch for your changes (e.g., feature-update).
   - This ensures that the main branch remains stable while you work on new features.
4. Make Changes in the Branch
   - Edit files or add new files in your branch.
   - Commit the changes with a descriptive message.
5. Push the Branch to Your Fork
   - Upload the committed branch from your local repository to your forked repository on GitHub.
6. Create a Pull Request
   - Go to the forked repository on GitHub.
   - Click Compare & Pull Request to propose merging your branch into the original repository.
   - Add a description of your changes and submit the pull request for review.

7. Collaborate and Review

- Repository owners can review your pull request, suggest changes, and merge it if approved.
- This completes the collaborative workflow in GitHub.

Program:

Fork a Repository

1. Navigate to a public GitHub repository (e.g., https://github.com/octocat/Spoon-Knife).
2. Click on the "Fork" button (top-right).
3. The forked repo will appear under GitHub profile.
4. Clone the forked repository to system:

git clone https://github.com/your-username/Spoon-Knife.git

cd Spoon-Knife

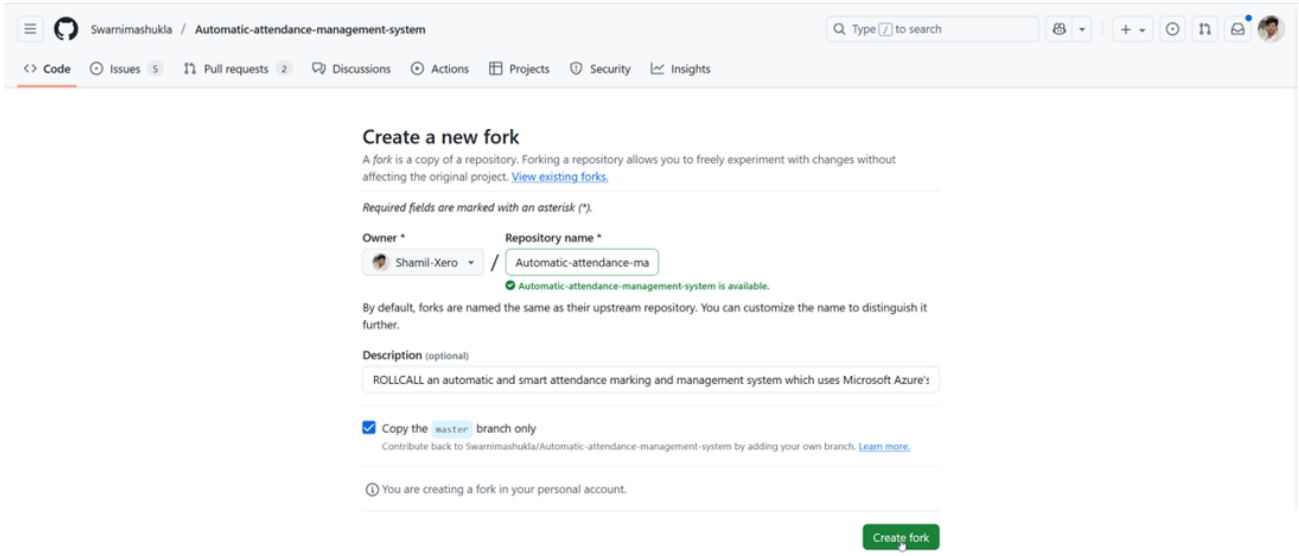
Create a Branch

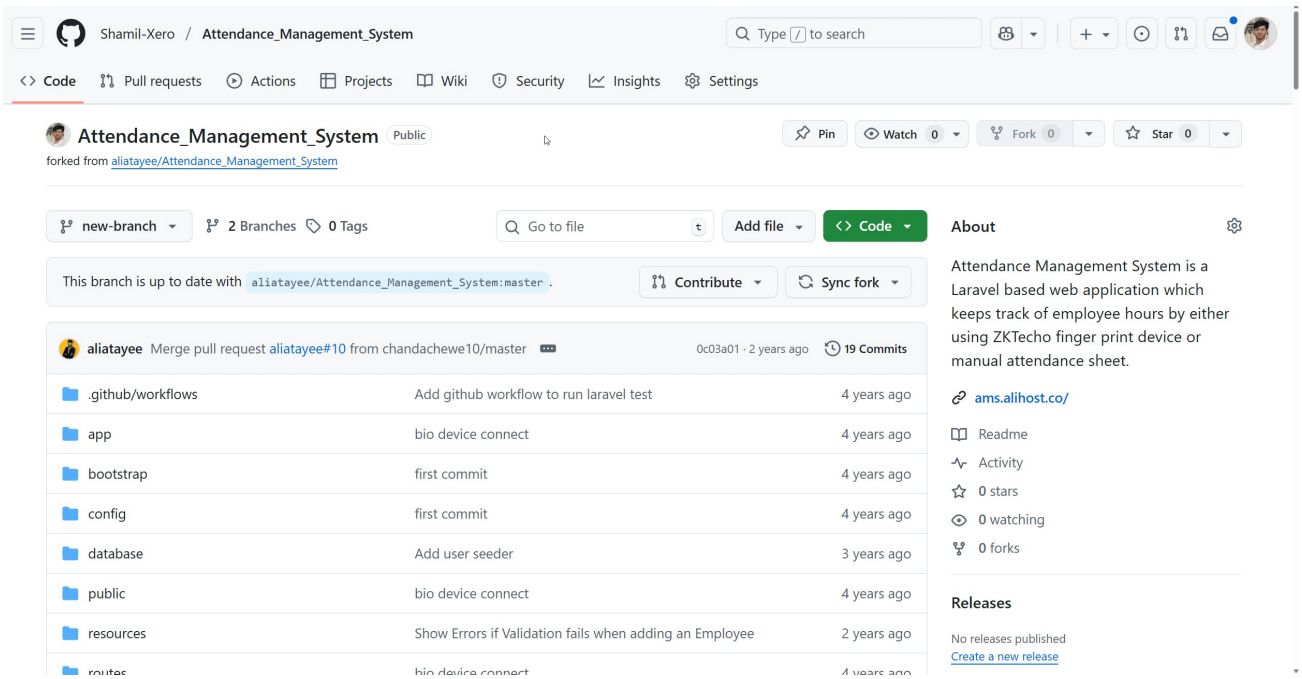
1. Create and switch to a new branch:
   git checkout -b feature-update
2. Make changes in code or files.
3. Stage and commit changes:
   git add .

   git commit -m "Added a feature update"
4. Push the branch to your GitHub repository:

   git push origin feature-update

Output:





Result:

To practice collaborative workflows in GitHub by forking a repository, creating branches has been done successfully

Ex No:3.b) **GitHub Collaboration –Raise Issues, and Create Pull Requests**
Date :

Aim:

To practice collaborative workflows in GitHub by **raising issues** to report bugs or suggest enhancements and **creating pull requests** to contribute changes to a repository.

Software Required:

- GitHub Account
- Web Browser
- GitHub Desktop or Git CLI (optional)

Procedure :

1. **Access the Repository**

   Navigate to the repository on GitHub where you want to report an issue or contribute.

2. **Raise an Issue**

   ○Go to the **Issues** tab of the repository.

   ○Click **New Issue**.

   ○Provide a descriptive **title** and detailed **description** explaining the bug, enhancement, or suggestion.

   ○Submit the issue to notify the repository maintainers.

3. **Fork the Repository (Optional)**

   If you want to contribute code to fix the issue or add a feature, click the **Fork** button to create your own copy of the repository.

4. **Create a New Branch**

   In your forked repository (or local clone), create a new branch specifically for addressing the issue (e.g., fix-bug or feature-update).

5. **Make Changes**

   Edit or add files in the branch to fix the issue or implement the requested feature.

6. **Commit Changes**

   Stage and commit your changes with a descriptive commit message explaining the update.

7. **Push Branch to GitHub**

Upload the branch with your changes to your forked repository.

8. **Create a Pull Request**

   Navigate to your forked repository on GitHub.

   Click **Compare & Pull Request** to propose merging your changes into the original repository.

   Reference the issue (e.g., “Fixes #issue-number”) and add a detailed description.

   Submit the pull request for review.

9. **Collaboration and Review**

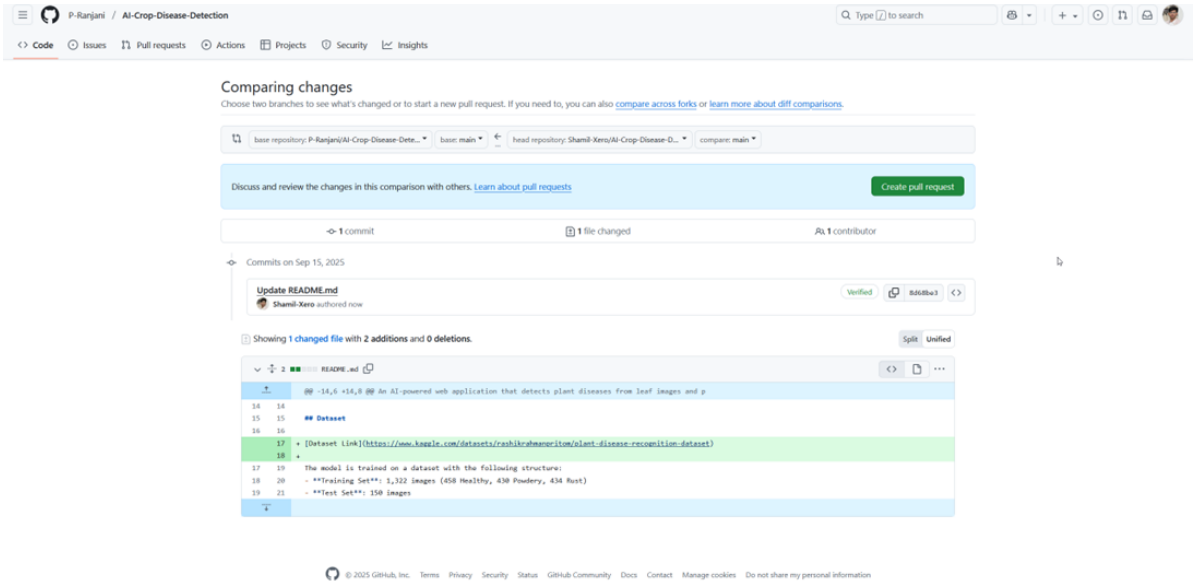
   Repository maintainers review the pull request, provide feedback, and merge the changes if approved.

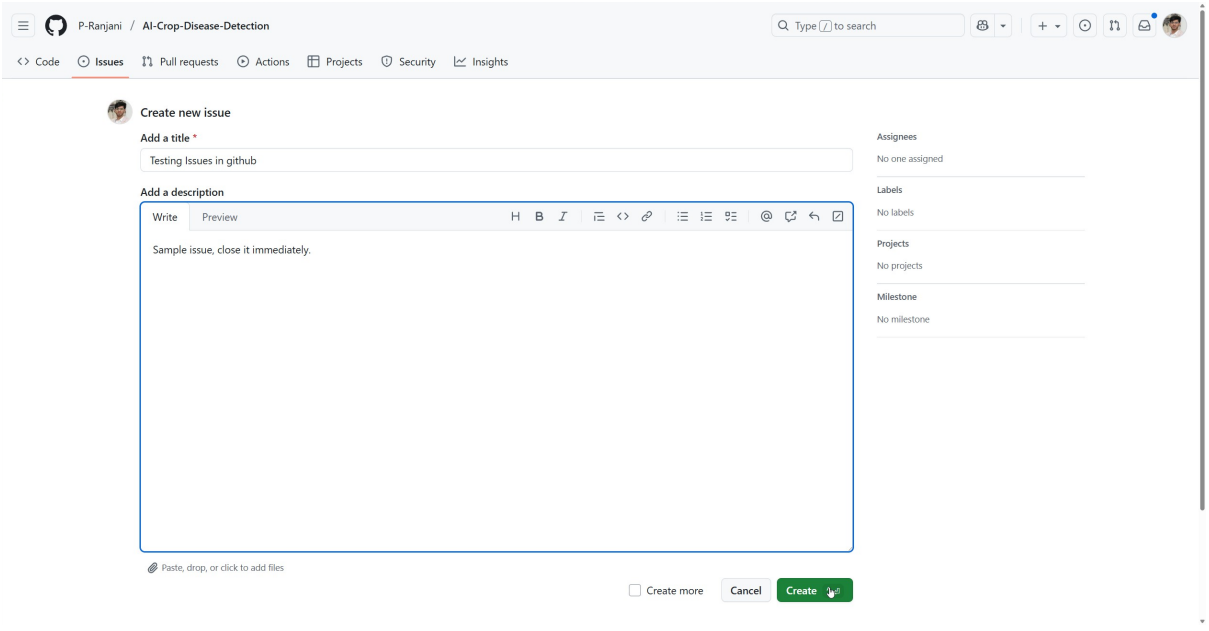
Program :

```
# Clone the forked repository

git clone https://github.com/your-username/forked-repo.git

cd forked-repo

# Create a new branch for the issue

git checkout -b fix-issue-branch

# Make changes in files

# (Edit or add files using a text editor)

# Stage and commit changes

git add .

git commit -m "Fixed issue #<issue-number>"

# Push branch to GitHub

git push origin fix-issue-branch

# Clone the forked repository

git clone https://github.com/your-username/forked-repo.git

cd forked-repo

# Create a new branch for the issue

git checkout -b fix-issue-branch
```

```
# Make changes in files

# (Edit or add files using a text editor)

# Stage and commit changes

git add .

git commit -m "Fixed issue #<issue-number>"

# Push branch to GitHub

git push origin fix-issue-branch
```

Output:





Result:

Thus the pull request is submitted to propose changes back to the original repository and learned how GitHub issues and pull requests facilitate collaboration and version control in real projects.

Ex.No : 4 **Build and Debug OSS Projects using Maven and VS Code**
Date :

Aim:

To setup, build, and debug a simple open-source Java project using **Maven** and **Visual Studio Code (VS Code)**.

Software Requirements:

- Java JDK (version 11 or above)
- Maven (latest stable version)
- Visual Studio Code with:
    - Java Extension Pack
    - Debugger for Java

Procedure:

1. Setup Maven Project

Open a terminal and create a Maven project

2. Open Project in VS Code

- Launch VS Code.
- Open the generated SimpleApp folder.
- Install the recommended Java extensions when prompted.

3. Explore Project Structure

- src/main/java: source code
- src/test/java: test cases
- pom.xml: project configuration file

4. Build the Project

In terminal or VS Code:
mvn clean compile

5. Run the Project

Create a Main class or use the default one in App.java:

```
package com.example;

public class App {

    public static void main(String[] args) {

        System.out.println("Hello from Maven Project!");

    }

}
```

Run using:
mvn exec:java -Dexec.mainClass="com.example.App"

6. Debug in VS Code

- Open App.java, add a **breakpoint** on the System.out.println() line.
- Click **Run > Start Debugging** (or press F5).
- Observe variable states in **Debug Panel**.

Sample Program:

```
package com.example;

public class App {

    public static void main(String[] args) {

        String name = "VS Code + Maven";

        System.out.println("Hello, " + name);

    }

}
```

Commands to upload a project to GitHub from the VS Code terminal:

Initialize a Git repository in project folder:

```
    git init

   git add .

   git commit -m "Initial project commit"

 git remote add origin <repository_url>

git push -u origin main
```

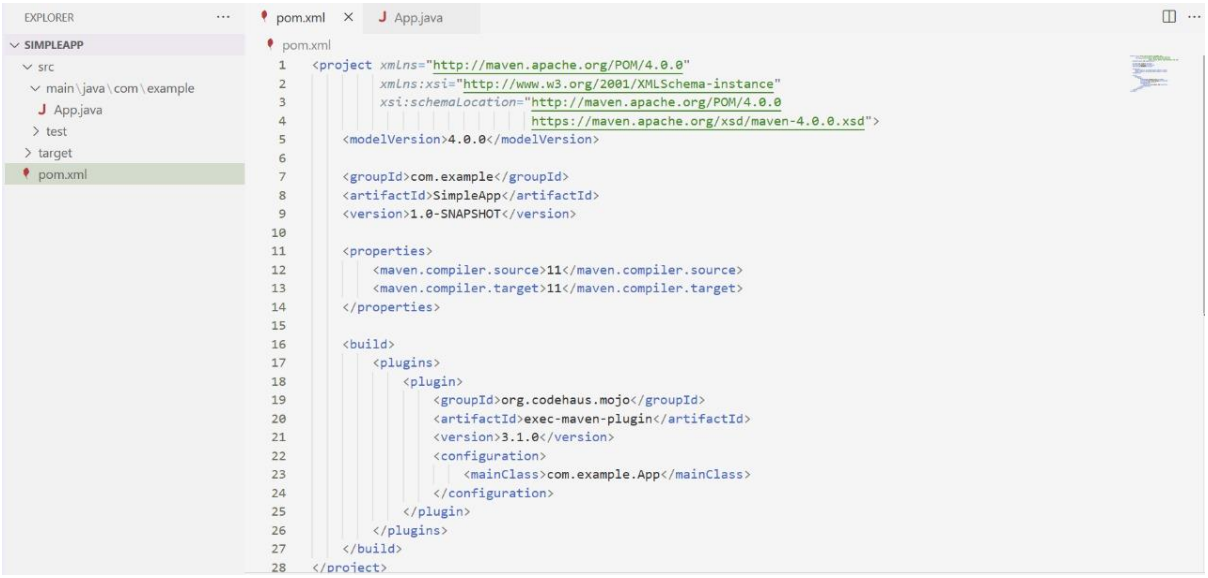
git status

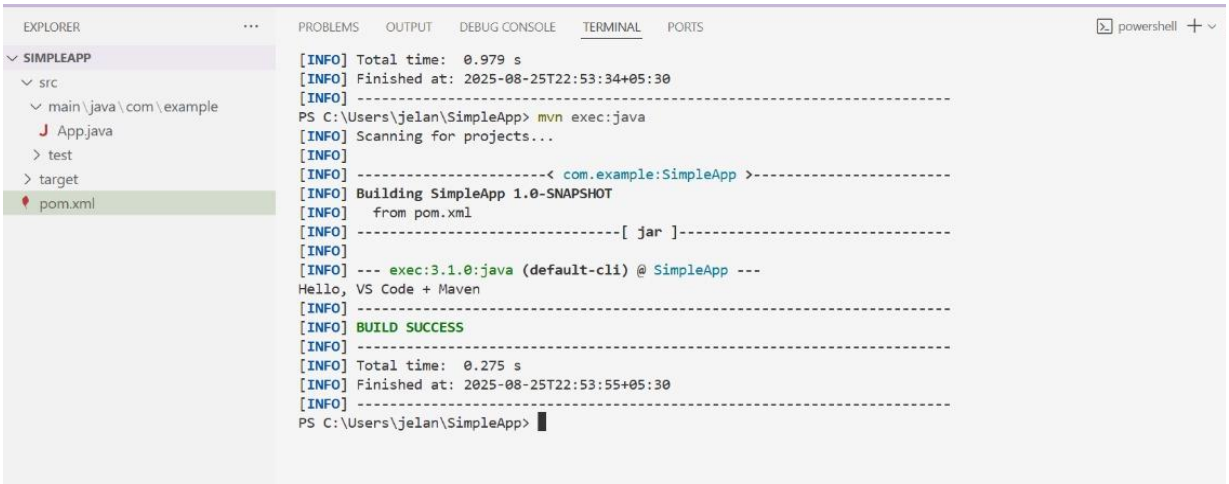
git add .

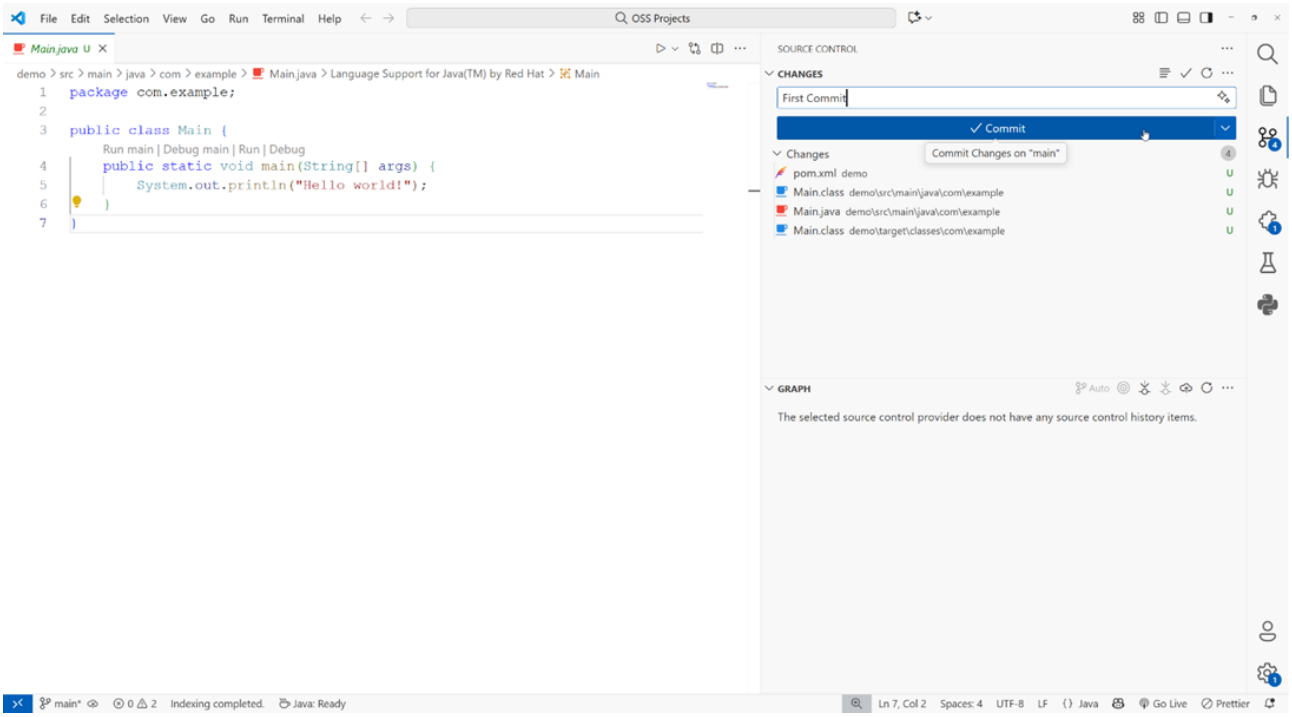
git commit -m "edited"

git push origin main

Output:







Result:

The open-source Maven-based project was successfully created, built, and debugged using Visual Studio Code.

ExNo: 5 **Bug Tracking and Issue Management using GitHub Projects**
Date :

Aim:

To track and manage software bugs and feature requests using **GitHub Issues** and **GitHub Projects** (Kanban board).

Software Required:

- GitHub account
- Web browser
- Existing GitHub repository (or create a new one)

Procedure:

1. Create a Repository (if not already existing):

- Go to https://github.com
- Click **+ New repository** → Name it (e.g., bug-tracker-demo) → Initialize with a README.

2. Enable GitHub Projects:

- Go to your repository.
- Click **Projects → New Project**
- Select **Board (Kanban)** view → Name it (e.g., "Bug Tracker Board") → Click **Create**.

3. Create Columns:

By default, three columns will be created:

- To do
- In progress
- Done

4. Create and Manage Issues:

- Go to the **Issues** tab → Click **New issue**
- Title: Fix broken login button
- Add a description, labels (e.g., bug, high priority), and assign.
- Submit the issue.

- Click the **Projects** section on the issue's sidebar → Assign it to the created board and column (e.g., "To do").

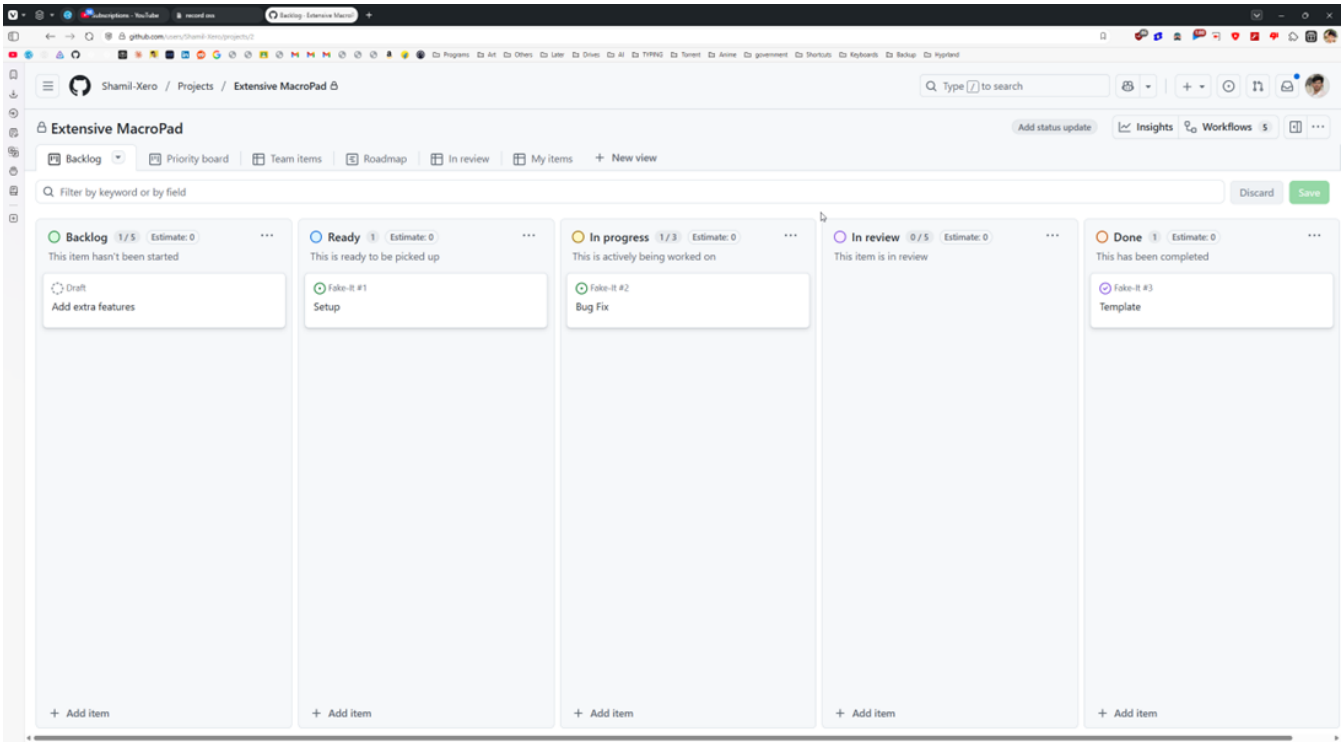
5. Track and Update Issue Status:

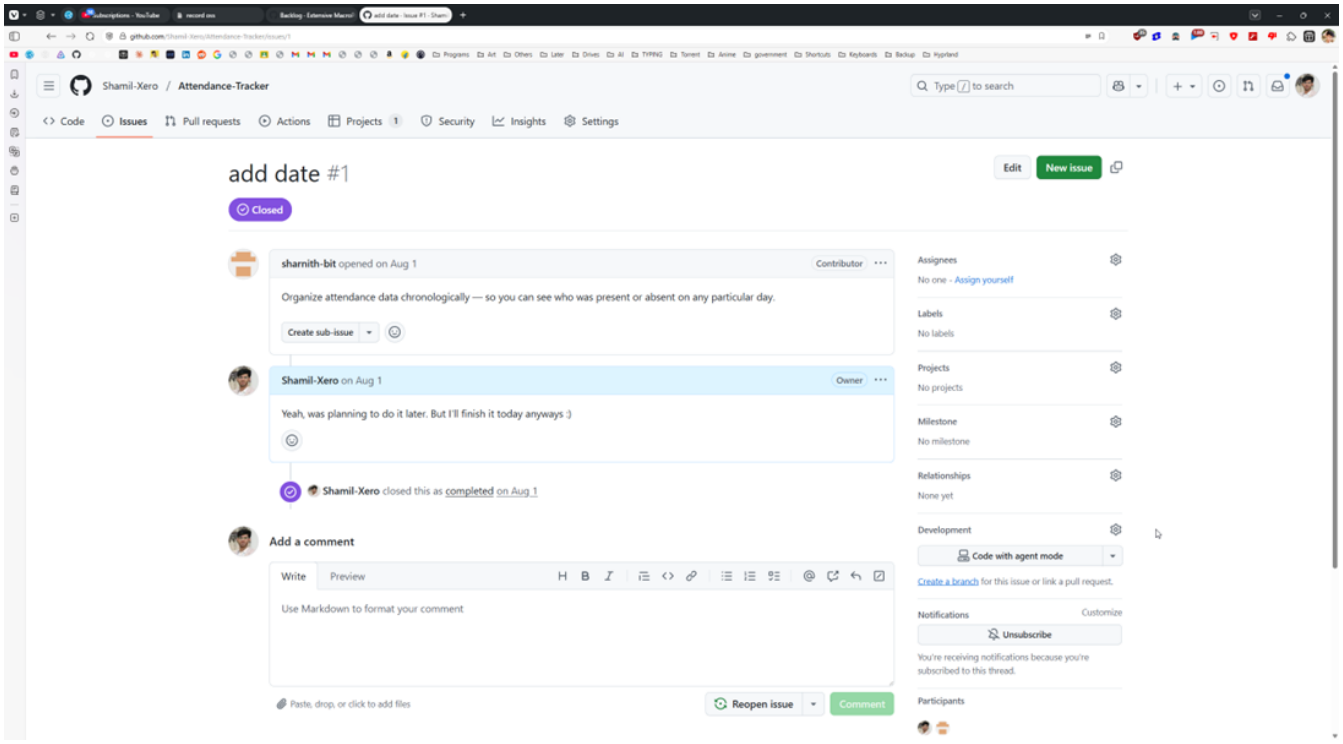
- Go back to the **Project Board**.
- Drag the issue card from **To do → In progress**.
- After resolving, drag it to **Done**.

6. Close the Issue:

- Navigate back to the **issue**.
- Comment on the fix (e.g., "Resolved in commit abc123") and click **Close issue**.

Output:





Result:

A project board was successfully created on GitHub. Issues were created, tracked, moved across columns, and closed after resolution using GitHub Projects.

ExNo:6 a) **Documentation Contribution Using Markdown**
Date :

Aim:

To practice creating and contributing documentation in **Markdown** format for a GitHub repository, enabling clear and structured project documentation.

Software Required:

- GitHub Account
- Web Browser
- Text Editor (VS Code, Notepad++, or any Markdown editor)
- Optional: GitHub Desktop

Procedure :

1. **Access the Repository**
   Navigate to the GitHub repository you want to contribute documentation for.

2. **Fork or Clone the Repository**

   Fork the repository to your GitHub account, or clone it to your local system.

3. **Create a New Branch**

   Create a separate branch for documentation changes (e.g., docs-update).

4. **Create or Edit Markdown Files**

   Create a new Markdown file (e.g., README.md, CONTRIBUTING.md) or edit existing ones.

   Use Markdown syntax to add headings, lists, tables, links, images, and code blocks.

5. **Preview the Markdown**

   Use your editor or GitHub preview to verify the formatting and appearance.

6. **Commit Documentation Changes**

   Stage and commit the Markdown files with a descriptive commit message, such as "Added documentation for setup and usage."

7. **Push Branch to GitHub**

   Upload the branch with your documentation changes to your forked repository.

8. **Create a Pull Request**

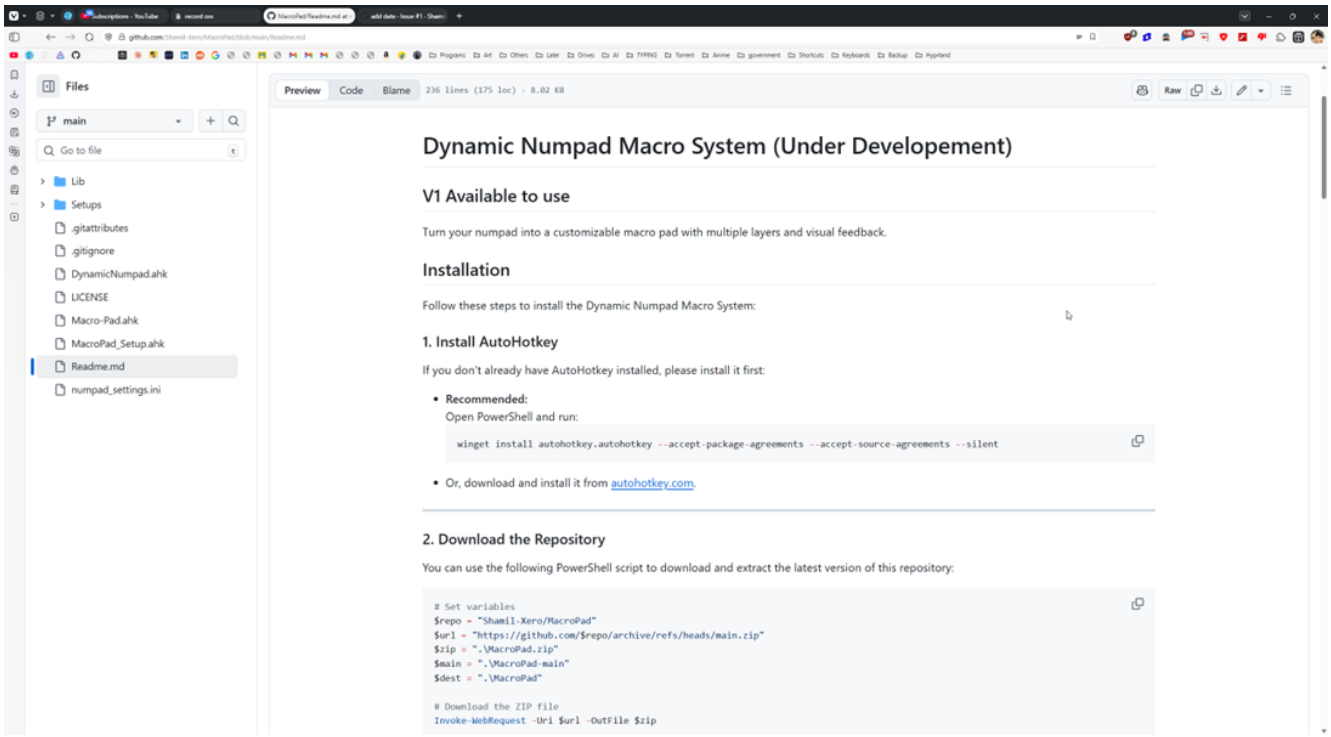
   Propose merging your documentation changes into the main repository.

   Add a clear description of the updates and submit the pull request for review.

9. **Collaborate and Review**

Repository maintainers review the pull request and merge the documentation if approved.

Output:



Result:

Thus an experiment to create or update documentation in readme.md file has been executed successfully.

ExNo:6 b) **Documentation Contribution Using MkDocs**
Date :

Aim:

To create and contribute project documentation using **MkDocs**, a static site generator for Markdown-based documentation, and to publish it for collaboration on GitHub.

Software Required:

- Python (3.x) installed
- MkDocs installed (pip install mkdocs)
- GitHub Account
- Text Editor (VS Code, Sublime, etc.)

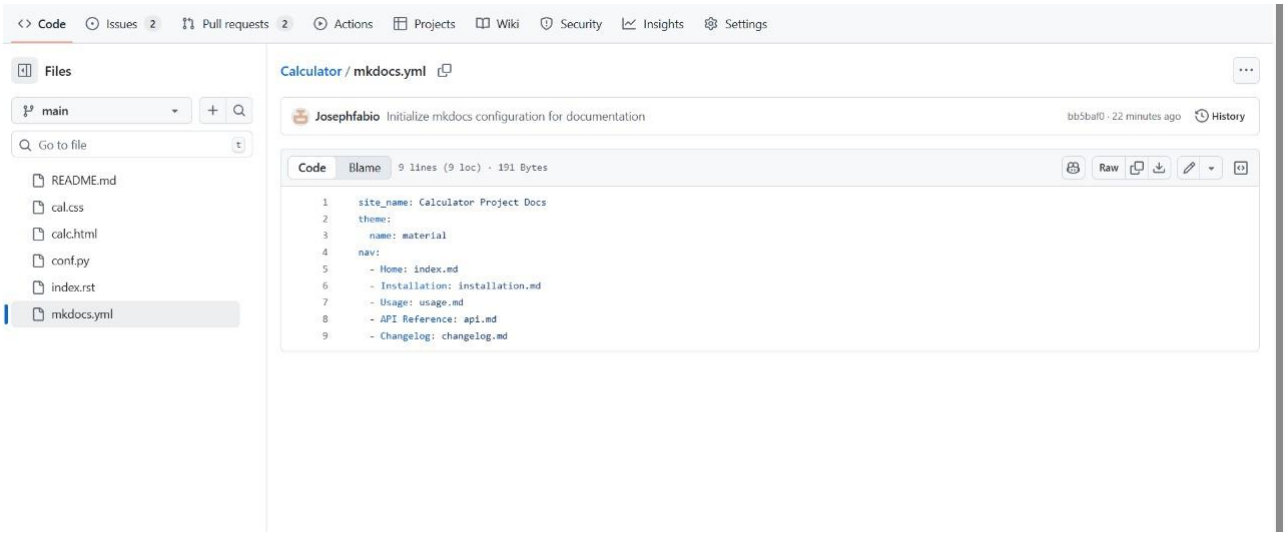
Procedure :

1. **Install MkDocs**
   Install MkDocs on your system using Python's package manager.
2. **Create a New MkDocs Project**
   Initialize a new documentation project with MkDocs, which creates a directory structure including docs folder and mkdocs.yml configuration file.
3. **Add Documentation Content**
   Create or edit Markdown files inside the docs folder. Include headings, paragraphs, lists, tables, images, and code snippets as needed.
4. **Configure Navigation**
   Update the mkdocs.yml file to define the navigation structure for the documentation site.
5. **Preview the Documentation Locally**
   Use MkDocs' built-in server to preview the documentation site locally in a browser to ensure formatting and structure are correct.
6. **Commit Changes to Local Repository**
   Initialize a Git repository (if not already), add the MkDocs project files, and commit your changes with a descriptive message.
7. **Push to Remote Repository**
   Connect your local repository to a remote GitHub repository and push the MkDocs project files.
8. **Publish Documentation (Optional)**
   Use mkdocs gh-deploy to deploy the documentation site to GitHub Pages for public access.
9. **Collaborate via Pull Requests**
   If contributing to an existing project, create a branch, push your changes, and submit a pull request for review.

Program:

```
# Install MkDocs

pip install mkdocs

# Create a new MkDocs project

mkdocs new my-project-docs

cd my-project-docs

# Add/edit Markdown files in the docs/ folder

# Preview the documentation locally

mkdocs serve

# Initialize Git repository (if needed)

git init

git add .

git commit -m "Initial MkDocs documentation"

# Push to GitHub

git remote add origin https://github.com/your-username/my-project-docs.git

git push -u origin main

# Deploy to GitHub Pages

mkdocs gh-deploy
```

Output:



Result:

Thus an experiment to create a structured documentation site using MkDocs has bben done and Markdown files organized under docs/ and navigation configured in mkdocs.yml.Documentation previewed locally and deployed to GitHub Pages.

ExNo:6.c) **Documentation Contribution Using Sphinx**
Date :

Aim:
To create and contribute project documentation using **Sphinx**, a Python-based documentation generator, enabling well-structured, professional documentation that can be published in multiple formats.

Software Required:

- Python (3.x) installed
- Sphinx installed (pip install sphinx)
- Text Editor (VS Code, Sublime, etc.)
- GitHub Account (for version control and collaboration)

Procedure :

1. **Install Sphinx**

   Install Sphinx on your system using Python's package manager.

2. **Initialize Sphinx Project**
   Use the Sphinx quickstart to create a new documentation project. This generates folders and files including conf.py (configuration) and index.rst (main documentation file).

3. **Create Documentation Files**
   Add content in reStructuredText (.rst) files inside the docs directory. Organize sections for introduction, usage, modules, or any other project details.

4. **Configure Documentation Settings**
   Edit conf.py to set project information, theme, extensions, and other settings for the documentation site.

5. **Build and Preview Documentation Locally**
   Use Sphinx build commands to generate HTML or PDF versions of the documentation and preview them in a web browser.

6. **Commit Documentation to Local Repository**
   Initialize a Git repository if needed, stage the Sphinx project files, and commit changes with a descriptive message.

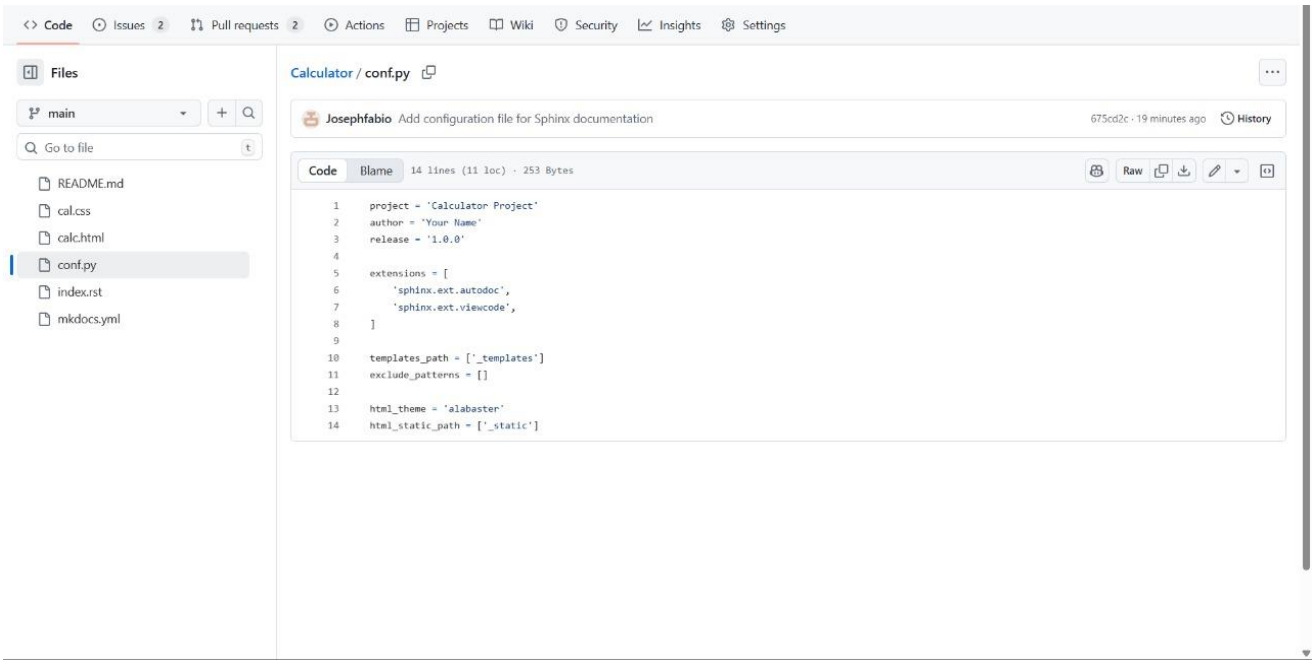
7. **Push to Remote Repository**
   Connect the local repository to GitHub and push your Sphinx documentation files.

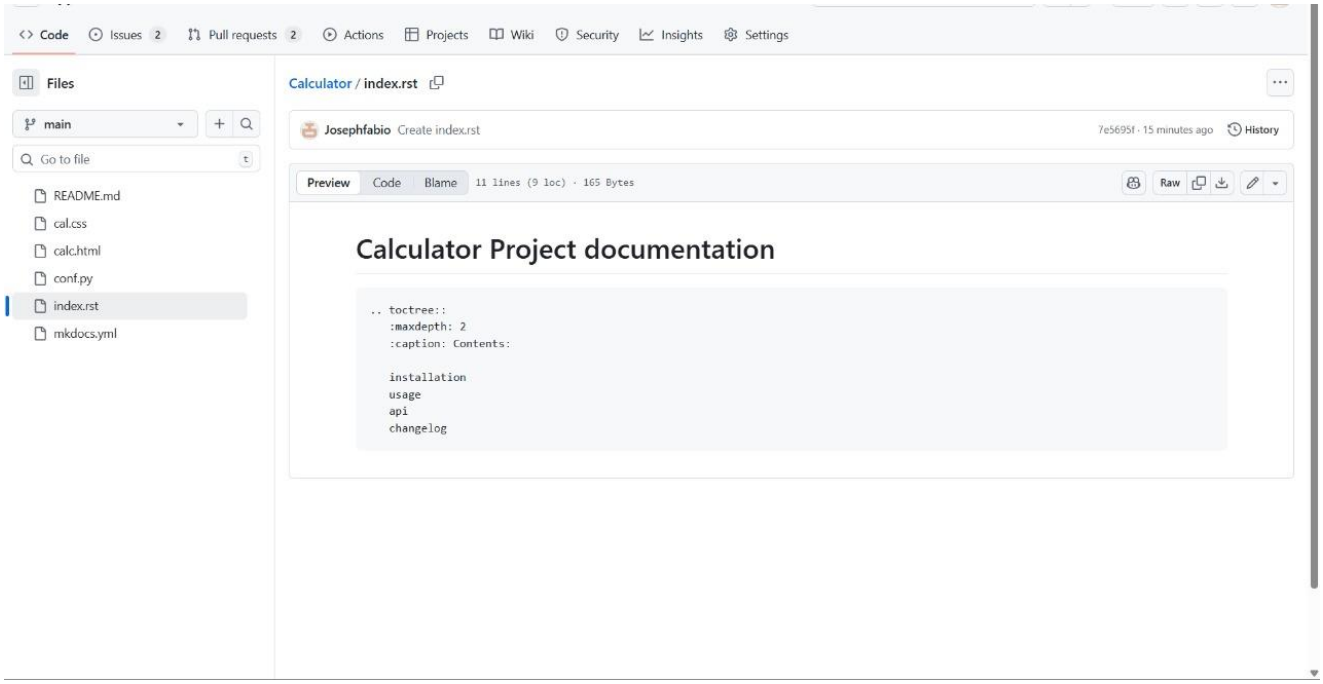
8. **Collaborate via Pull Requests**
   If contributing to an existing project, create a separate branch for your documentation updates, push the branch, and submit a pull request for review.

Commands:

```
# Install Sphinx
pip install sphinx
# Initialize Sphinx project
sphinx-quickstart
# Add or edit .rst files in the docs/ directory
# Build HTML documentation
make html
# Preview documentation in browser
open _build/html/index.html  # or navigate to local path
# Initialize Git repository (if needed)
git init
git add .
git commit -m "Initial Sphinx documentation"
# Push to GitHub
git remote add origin https://github.com/your-username/project-docs.git
git push -u origin main
```

Output:





Result:

Thus an experiment to create a structured documentation project using Sphinx has been done and Pushed the documentation project to GitHub and contributed via pull request if collaborating.

Ex.No : 7

Date :

Translation/localization-contribute translations using weblate, transifex and POEditor

Aim

To learn how to contribute translations to open-source software hosted on GitHub using graphical user interfaces (Weblate, Transifex, and POEditor)

- Understand the localization workflow for open-source projects.
- Learn to join translation platforms (Weblate, Transifex, POEditor) linked with GitHub repositories.
- Perform translations using only web-based GUIs.
- Submit and review translations that get synchronized to GitHub.

Pre-requisites

- Basic GitHub account.
- Accounts on:
- Weblate
- Transifex
- POEditor
- A sample or real open-source GitHub project integrated with one of these platforms.

Procedure:

Contributing with Weblate (GUI)

Steps:

1. **Join Project**
    - Go to the project's Weblate URL (example: https://hosted.weblate.org/projects/project-name/).
    - Sign in or create an account.
2. **Select Language**
    - Click target language.
3. **Translate**

    Click "Translate" → The editor opens with:

- **Source String** (original text).
- **Translation Box** (enter translation).
- **Context** (comments, screenshots).
- **Suggestions** (from Translation Memory or Machine Translation).

4. **Save & Review**
    - Click “Save” after each translation.
    - Mark as “Needs Review” if unsure.
5. **Sync to GitHub**
    - Wait for the project’s automated bot to push changes to the GitHub repo (usually visible as a commit like “Translated using Weblate”).

Contributing with Transifex (GUI)

Steps:

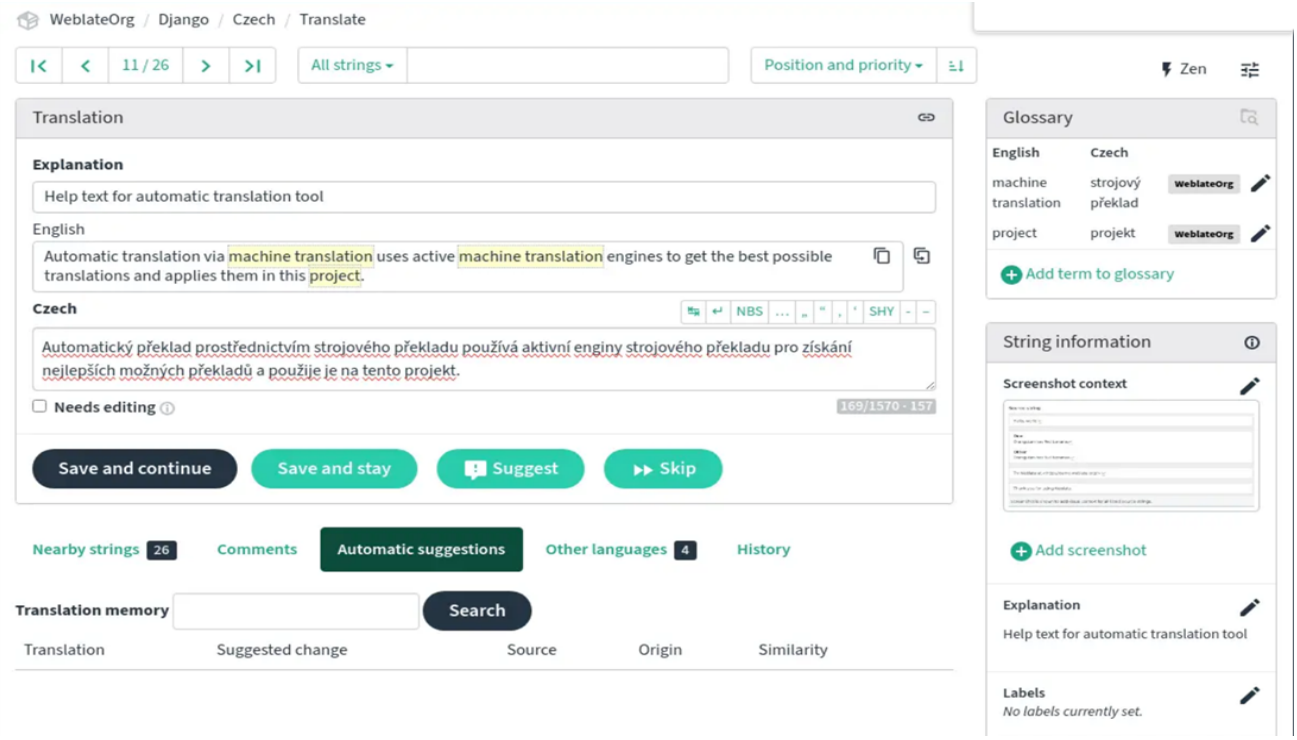
1. **Join Project**
    - Go to the Transifex project page (example: https://www.transifex.com/organization/project-name/).
    - Sign in or create an account.
2. **Choose Resource & Language**
    - Pick the file/resource to translate.
    - Select target language.
3. **Web Editor**
    - Source string on the left, translation box on the right.
    - Add translation, check glossary suggestions.
4. **Save**
    - Click “Save Translation.”
5. **Sync**
    - Transifex syncs translations to GitHub automatically when the project maintainer triggers an export.

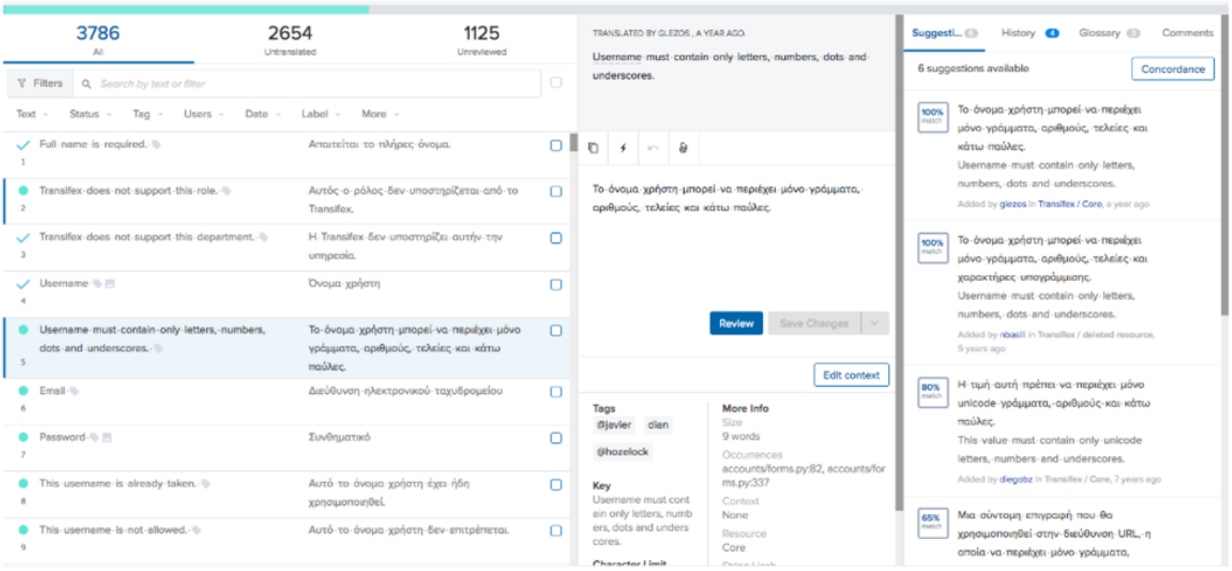
Contributing with POEditor (GUI)

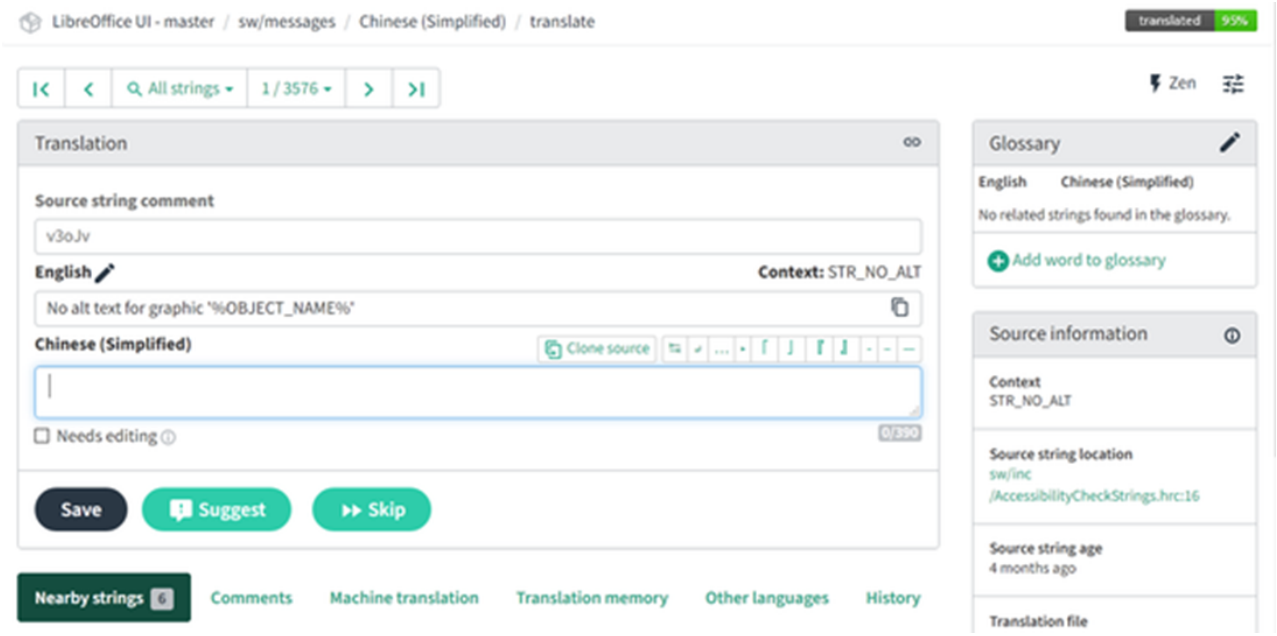
Steps:

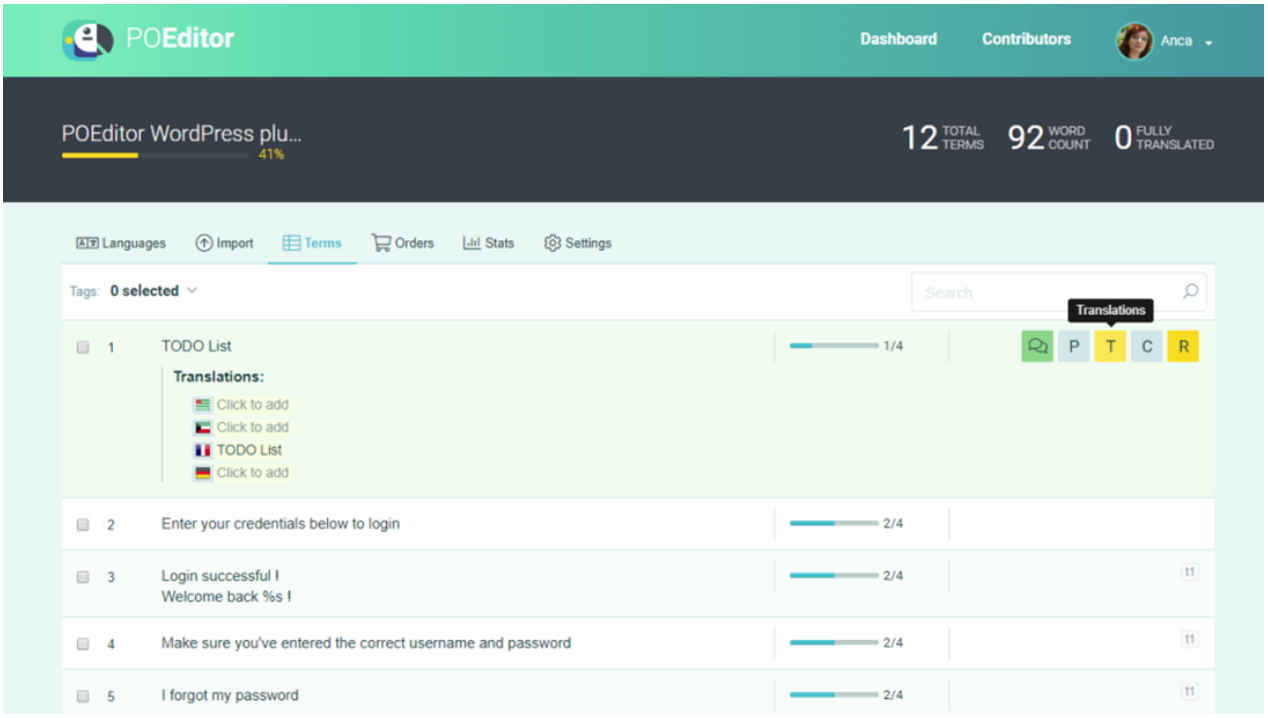
1. **Join Project**
    - Open project invitation link or public project page.
    - Sign in or create an account.
2. **Select Language**
    - Choose the language we want to contribute to.
3. **Translate**
    - Click on each term, type translation in the text field.
    - Use MT suggestions if available.
4. **Save**
    - POEditor saves automatically.
5. **GitHub Sync**
    - Maintainer exports translations to GitHub (automatic if integration is set).

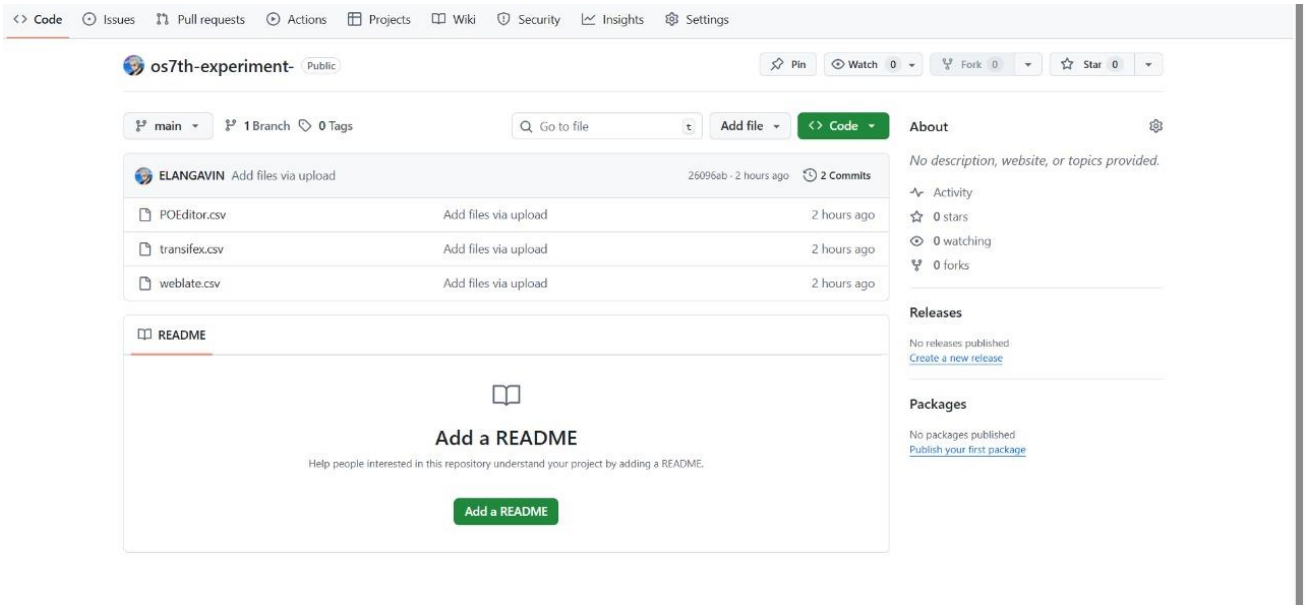
Output:


WeblateOrg / Django / Czech / Translate
11 / 26
All strings
Position and priority
Zen
Translation
Explanation
Help text for automatic translation tool
English
Automatic translation via machine translation uses active machine translation engines to get the best possible translations and applies them in this project.
Czech
Automatický překlad prostřednictvím strojového překladu používá aktivní enginy strojového překladu pro získání nejlepších možných překladů a použije je na tento projekt.
Needs editing
169/1570 · 157
Save and continue
Save and stay
Suggest
Skip
Nearby strings 26
Comments
Automatic suggestions
Other languages 4
History
Translation memory
Search
Translation
Suggested change
Source
Origin
Similarity
Glossary
English
Czech
machine translation
strojový překlad
WeblateOrg
project
projekt
WeblateOrg
Add term to glossary
String information
Screenshot context
Add screenshot
Explanation
Help text for automatic translation tool
Labels
No labels currently set.


3786
2654
1125
Filters
Full name is required.
Απαιτείται το πλήρες όνομα.
Transifex does not support this role.
Αυτός ο ρόλος δεν υποστηρίζεται από το Transifex.
Transifex does not support this department.
Η Transifex δεν υποστηρίζει αυτήν την υπηρεσία.
Username
Όνομα χρήστη
Username must contain only letters, numbers, dots and underscores.
Το όνομα χρήστη μπορεί να περιέχει μόνο γράμματα, αριθμούς, τελείες και κάτω παύλες.
Email
Διεύθυνση ηλεκτρονικού ταχυδρομείου
Password
Συνθηματικό
This username is already taken.
Αυτό το όνομα χρήστη έχει ήδη χρησιμοποιηθεί.
This username is not allowed.
Αυτό το όνομα χρήστη δεν επιτρέπεται.
Review
Save Changes
Edit context
Suggestions
History
Glossary
Comments
6 suggestions available
Concordance


LibreOffice UI - master / sw/messages / Chinese (Simplified) / translate
translated 95%
All strings
1 / 3576
Zen
Translation
Source string comment
v3oJv
English
Context: STR_NO_ALT
No alt text for graphic '%OBJECT_NAME%'
Chinese (Simplified)
Clone source
Needs editing
0/390
Save
Suggest
Skip
Nearby strings 6
Comments
Machine translation
Translation memory
Other languages
History
Glossary
English
Chinese (Simplified)
No related strings found in the glossary.
Add word to glossary
Source information
Context
STR_NO_ALT
Source string location
sw/inc
/AccessibilityCheckStrings.hrc:16
Source string age
4 months ago
Translation file


POEditor
Dashboard
Contributors
Anca
POEditor WordPress plu...
41%
12 TOTAL TERMS
92 WORD COUNT
0 FULLY TRANSLATED
Languages
Import
Terms
Orders
Stats
Settings
Tags: 0 selected
Search
Translations
1 TODO List 1/4
Translations:
Click to add
Click to add
TODO List
Click to add
2 Enter your credentials below to login 2/4
3 Login successful !
Welcome back %s ! 2/4
4 Make sure you've entered the correct username and password 2/4
5 I forgot my password 2/4



Result:

Thus the translations to open-source projects using GUI-based workflows in Weblate, Transifex, and POEditor, with commits appearing in GitHub is contributed successfully.

ExNo:8
Date :

Contribution Discovery- Use Up for Grabs and First Timers only platform to find beginner friendly issues

Aim:

To explore and identify beginner-friendly open-source contribution opportunities using Up for Grabs and First Timers Only platforms, and to understand how to evaluate issues suitable for first-time contributors.

Software Required:

•Web browser (Chrome / Firefox)

•GitHub account

•Internet connection

•GitHub Desktop or Git CLI

Procedure:

Open-source contribution allows developers to collaborate on public projects and improve real-world software. However, beginners often struggle to find suitable entry points. Platforms like Up for Grabs and First Timers Only curate repositories that welcome new contributors.

Up for Grabs (https://up-for-grabs.net): Features a searchable list of projects with labels like good first issue or beginner friendly, allowing filtering based on language.

First Timers Only (https://www.firsttimersonly.com): Promotes repositories that encourage first-time contributors with clear contribution guidelines and supportive onboarding.

Using Up For Grabs

1. Open Up For Grabs in the browser.
2. Browse through the projects listed.
3. Use the **search box** or filters to find a project in a familiar technology (e.g., Python, JavaScript).
4. Open a project's page and review the issues tagged as *Up-For-Grabs*.
5. Note down at least **two beginner-friendly issues** .

Using First Timers Only

1. Visit First Timers Only.
2. Read about the initiative to understand its purpose.
3. Navigate to projects linked on GitHub or other platforms with **First Timer** issues.

4. Select a project and open a beginner-friendly issue.

5. Record the issue title, link, and a short summary of what the issue is about.

Step 1: Create or Log in to a GitHub Account

- Visit https://github.com.
- Sign up for a free account or log in to your existing one.

Step 2: Access Contribution Discovery Platforms

- Open the following sites:
- https://up-for-grabs.net
- https://www.firsttimersonly.com

Step 3: Search for Beginner-Friendly Projects

- Browse through the list of repositories.
- Filter projects based on your preferred programming language (e.g., Python, JavaScript, Java).
- Select an issue labeled as:
- Good first issue
- first-timers-only
- help wanted

Step 4: Analyze the Selected Issue

- Read the issue description carefully.
- Check repository documentation for contribution guidelines (CONTRIBUTING.md).
- Understand the scope of the task.

Step 5: Fork and Clone the Repository

git clone https://github.com/<username>/<repository-name>.git

Step 6: Create a New Branch

git checkout -b fix-beginner-issue

Step 7: Make Changes and Commit

- Edit files as required to address the issue.
- Then commit the changes:

  git add .

  git commit -m "Resolved beginner issue #123"

  git push origin fix-beginner-issue

Step 8: Submit a Pull Request

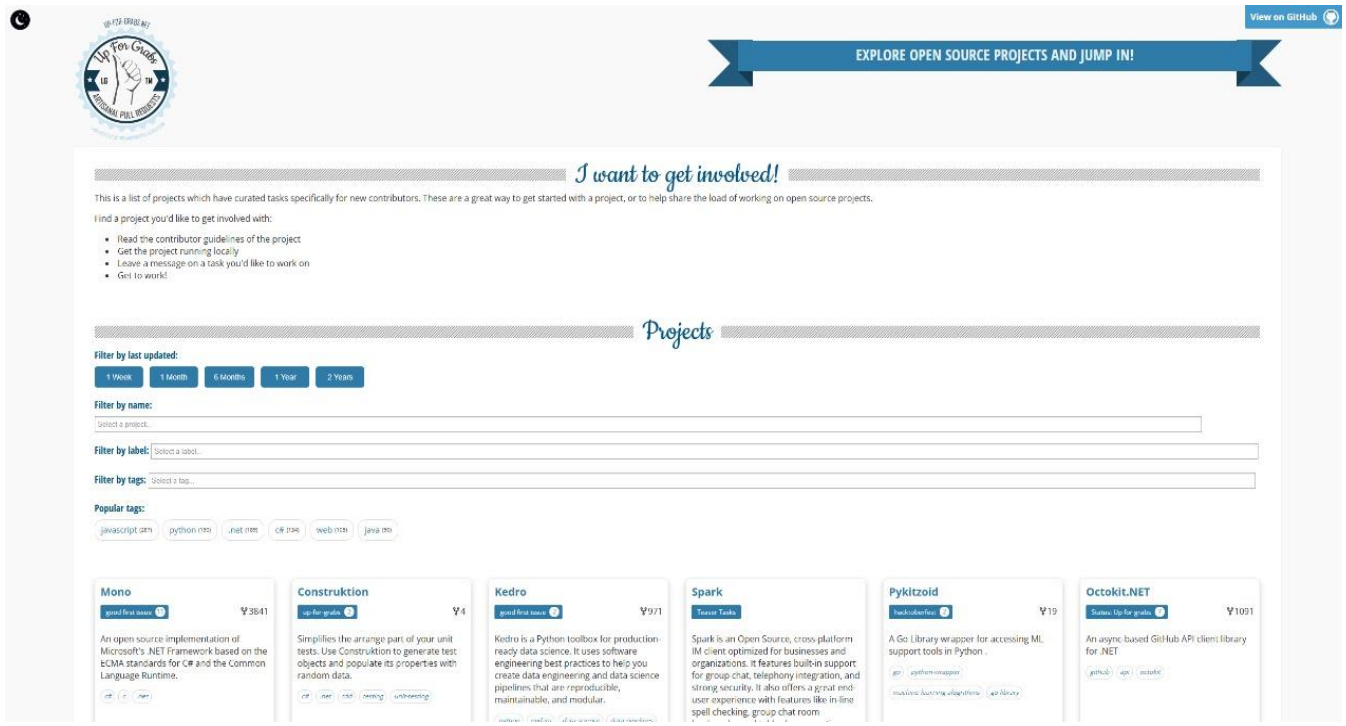
- Navigate to your GitHub repository.
- Click Compare & pull request.
- Write a short summary and submit the PR for review.

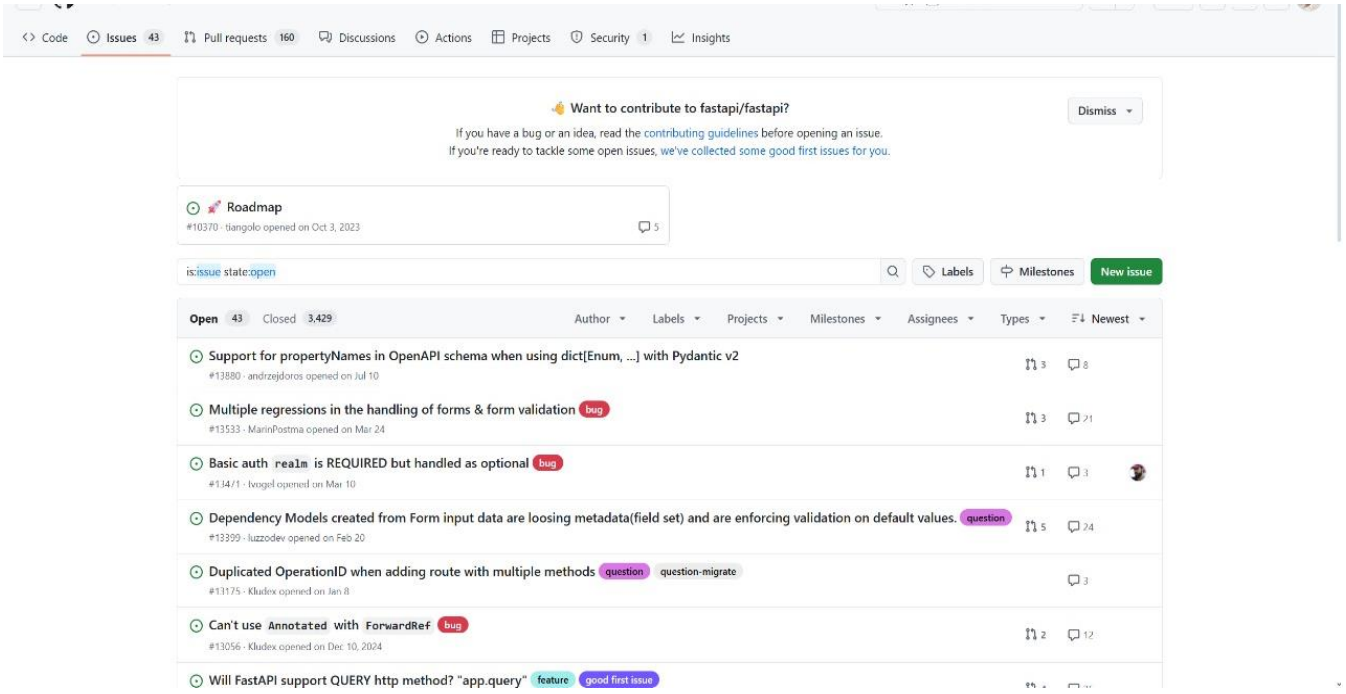
Step 9: Wait for Maintainer Review

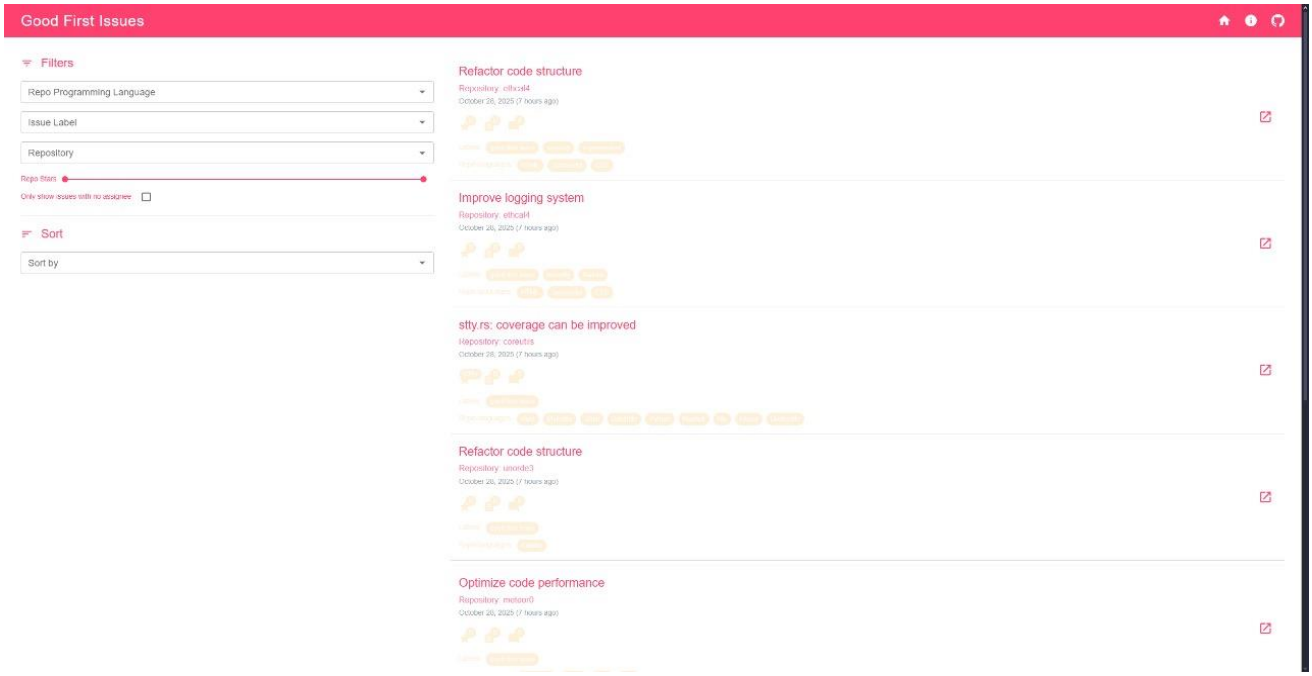
Once the maintainer reviews your contribution, it may be approved and merged into the main project.

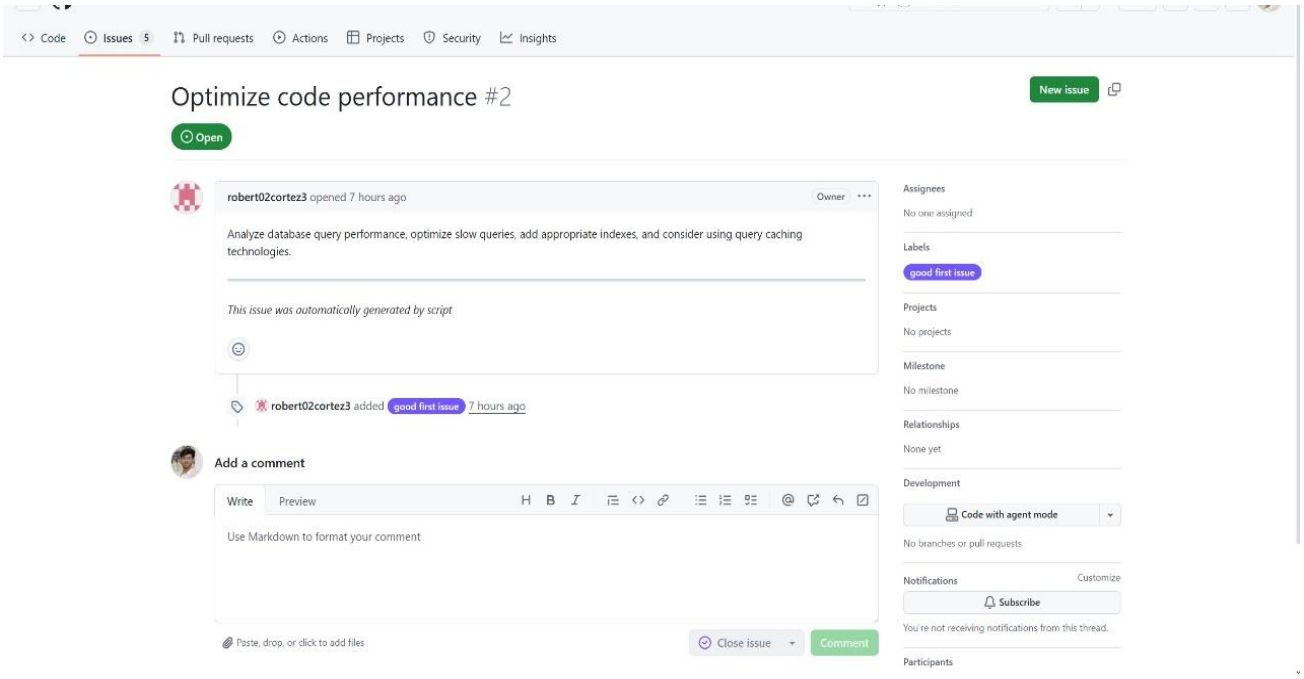
Platform Used	Repository Name	Language	Issue Title	Status
Up for Grabs	firstcontributions/first-contributions	Markdown	Add your name to CONTRIBUTORS.md	Open

Output:


View on GitHub
EXPLORE OPEN SOURCE PROJECTS AND JUMP IN!
I want to get involved!
This is a list of projects which have curated tasks specifically for new contributors. These are a great way to get started with a project, or to help share the load of working on open source projects.
Find a project you'd like to get involved with:
Read the contributor guidelines of the project
Get the project running locally
Leave a message on a task you'd like to work on
Get to work!
Projects
Filter by last updated:
1 Week
1 Month
6 Months
1 Year
2 Years
Filter by name:
Filter by label:
Filter by tags:
Popular tags:
Mono
Construktion
Kedro
Spark
Pykitzoid
Octokit.NET


Code
Issues 43
Pull requests 160
Discussions
Actions
Projects
Security 1
Insights
Want to contribute to fastapi/fastapi?
Dismiss
If you have a bug or an idea, read the contributing guidelines before opening an issue.
If you're ready to tackle some open issues, we've collected some good first issues for you.
Roadmap
#10370 · tiangolo opened on Oct 3, 2023
is:issue state:open
Labels
Milestones
New issue
Open 43
Closed 3,429
Author
Labels
Projects
Milestones
Assignees
Types
Newest
Support for propertyNames in OpenAPI schema when using dict[Enum, ...] with Pydantic v2
#13880 · andrzejdoros opened on Jul 10
Multiple regressions in the handling of forms & form validation bug
#13533 · MarinPostma opened on Mar 24
Basic auth realm is REQUIRED but handled as optional bug
#1147/1 · tvogel opened on Mar 10
Dependency Models created from Form input data are loosing metadata(field set) and are enforcing validation on default values. question
#13399 · luzzodev opened on Feb 20
Duplicated OperationID when adding route with multiple methods question question-migrate
#13175 · Kludex opened on Jan 8
Can't use Annotated with ForwardRef bug
#13056 · Kludex opened on Dec 10, 2024
Will FastAPI support QUERY http method? "app.query" feature good first issue





Result:

Thus open-source projects and identify beginner-friendly contribution opportunities using community-driven platforms has been explored successfully.

ExNo:9
Date :

React .js Framework Basics

Aim:

To introduce the fundamentals of the React.js framework and demonstrate the concept of **component-based architecture** by building a simple React application.

Procedure:

Step 1: Setup React Project

1. Open terminal/command prompt.
2. Run the following command to create a React app:

```
npx create-react-app my-first-react-app

cd my-first-react-app

npm start
```

3. The app will run on http://localhost:3000/.

Step 2: Create Components

1. Inside src/, create a folder named components.
2. Create a file Header.js with the following code:

```
function Header() {
 return (
   <header>
  <h1>Welcome to My React App</h1>
 </header>
 );
}
export default Header;
```

3. Create another file Footer.js:

```
function Footer() {
 return (
   <footer>
  <p>© 2025 My React App</p>
 </footer>
  );
}

export default Footer;
```

4. Create one more component Message.js:

```
function Message(props) {
    return <p>Hello, {props.name}! This is a React component.</p>;
```

```
}
export default Message;
```

Step 3: Use Components in App.js

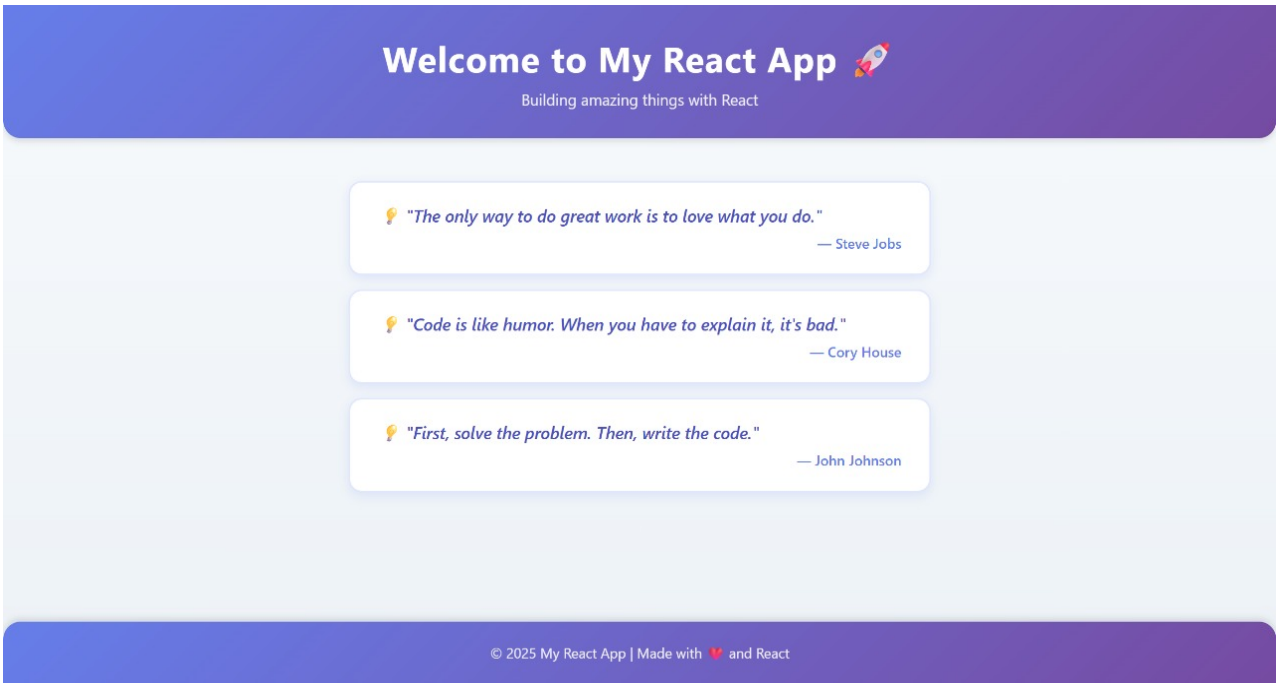
Modify App.js as follows:

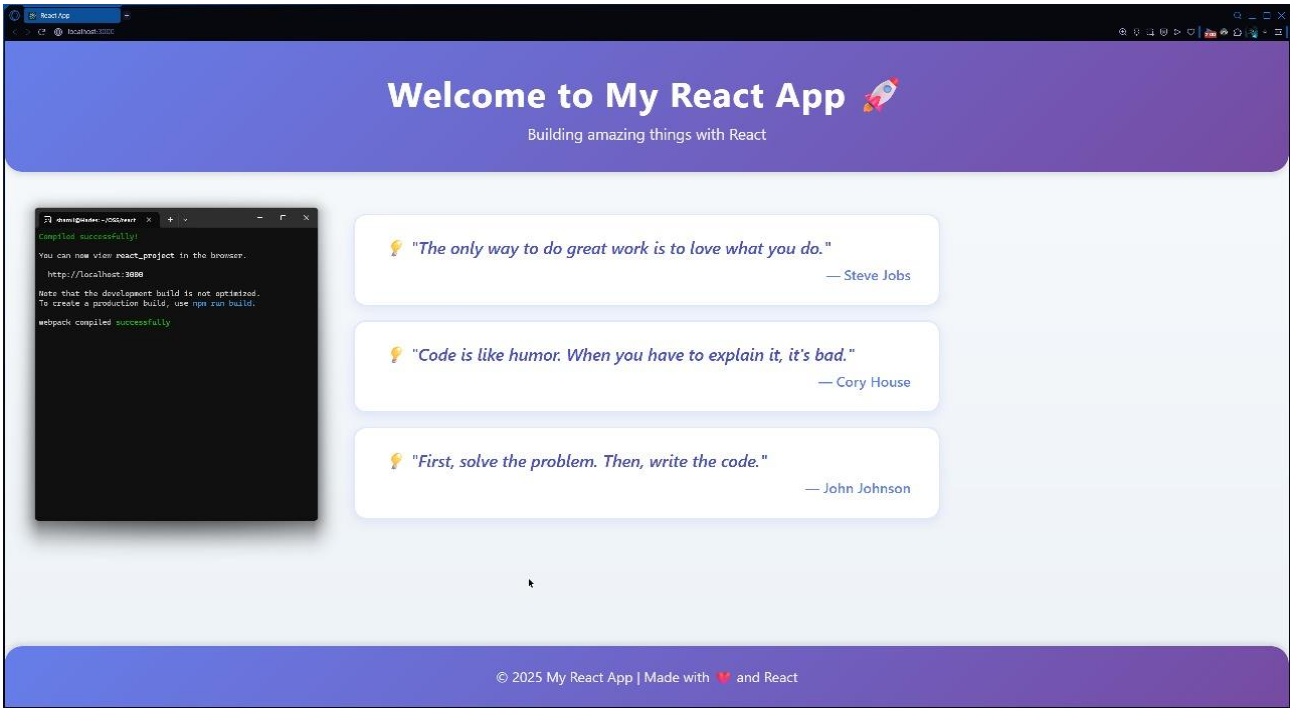
```
import Header from './components/Header';
import Footer from './components/Footer';
import Message from './components/Message';

function App() {
  return (
    <div>
      <Header />
      <Message name="ABC" />
      <Message name="Student" />
      <Footer />
    </div>
  );
}

export default App;
```

Step 4: Run and Verify

1. Save all files and ensure the development server is running (npm start).
2. Open http://localhost:3000/ in the browser.
    - A header at the top.
    - Two Message components with different names.
    - A footer at the bottom.

Output:





Result:

A simple React app has been built with **Header, Footer, and Message components**.

ExNo:10 **Create the GitHub Account to demonstrate CI/CD pipeline using Cloud**
Date : **Platform**

Aim:

To create a GitHub account and implement a Continuous Integration and Continuous Deployment (CI/CD) pipeline using a cloud platform (such as GitHub Actions, AWS, Azure, or Google Cloud) for a sample project.

Theory:

Continuous Integration (CI) and Continuous Deployment (CD) are key practices in DevOps that enable faster, more reliable software delivery.

Continuous Integration (CI): The process of automatically building and testing code every time a developer commits changes to a shared repository.

Continuous Deployment (CD): The process of automatically deploying the tested application to production or a staging environment.

CI/CD Benefits:

- Early detection of integration issues
- Automated testing and deployment
- Faster release cycles
- Improved collaboration between developers and operations

Popular CI/CD Tools:

- GitHub Actions
- Jenkins
- GitLab CI
- CircleCI
- AWS CodePipeline
- Azure DevOps Pipelines.

Software / Tools Required:

- GitHub account
- Web browser (Chrome / Firefox)
- Internet connection
- Cloud Platform account (AWS / Azure / Google Cloud / GitHub Actions)
- Sample project (HTML, Python, or Node.js app)

Procedure:

Step 1: Create a GitHub Account

- Visit the GitHub website (https://github.com/).
- Click on the "Sign Up" button and follow the instructions to create GitHub account.

Step 2: Create a Sample GitHub Repository

- Log in to GitHub account.
- Click the "+" icon in the top-right corner and select "New Repository."
- Give repository a name (e.g., "my-web-pages") and provide an optional description.
- Choose the repository visibility (public or private).
- Click the "Create repository" button.

Step 3: Set Up a Google Cloud Platform Project

- Log in to Google Cloud Platform account.
- Create a new GCP project by clicking on the project drop-down in the GCP Console (https://console.cloud.google.com/).
- Click on "New Project" and follow the prompts to create a project.

Step 4: Connect GitHub to Google Cloud Build

- In GCP Console, navigate to "Cloud Build" under the "Tools" section.
- Click on "Triggers" in the left sidebar.
- Click the "Connect Repository" button.
- Select "GitHub (Cloud Build GitHub App)" as the source provider.
- Authorise Google Cloud Build to access GitHub account.
- Choose GitHub repository ("my-web-pages" in this case) and branch.
- Click "Create."

Step 5: Create a CI/CD Configuration File

- In GitHub repository, create a configuration file named **cloudbuild.yam**l. This file defines the CI/CD pipeline steps.
- Add a simple example configuration to copy web pages to an Apache web server. Here's an example:
- ***steps:***
- ***name: 'gcr.io/cloud-builders/gsutil'***
- ***args: ['-m', 'rsync', '-r', 'web-pages/', 'gs://your-bucket-name']***
- Replace 'gs://your-bucket-name' with the actual Google Cloud Storage bucket where Apache web server serves web pages.
- Commit and push this file to GitHub repository.

```
name: CI/CD Pipeline

on:

  push:

    branches: [ main ]

  pull_request:

    branches: [ main ]



jobs:

  build:

    runs-on: ubuntu-latest

    steps:

    - name: Checkout code

      uses: actions/checkout@v3

    - name: Set up Python

      uses: actions/setup-python@v4

      with:

        python-version: '3.9'
```

```
- name: Install dependencies
  run: |
    pip install -r requirements.txt || echo "No dependencies"
- name: Run tests
  run: |
    echo "Running tests..."
    python app.py
- name: Deploy (Simulated)
  run: echo "Deployment step executed successfully!"
```

Step 6: Trigger the CI/CD Pipeline

- Make changes to web pages or configuration.
- Push the changes to GitHub repository.
- Go to GCP Console and navigate to "Cloud Build" > "Triggers."
- An automatic trigger for repository. Click the trigger to see details.
- Click "Run Trigger" to manually trigger the CI/CD pipeline.

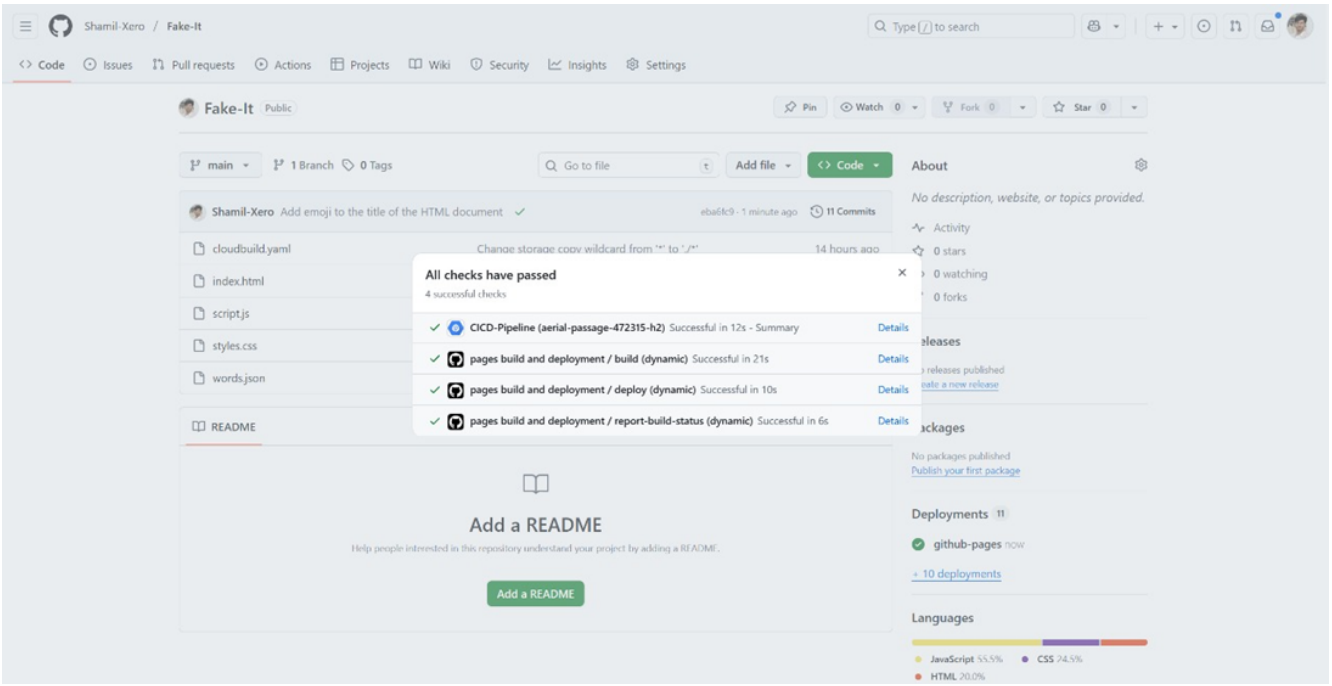
Step 7: Monitor the CI/CD Pipeline

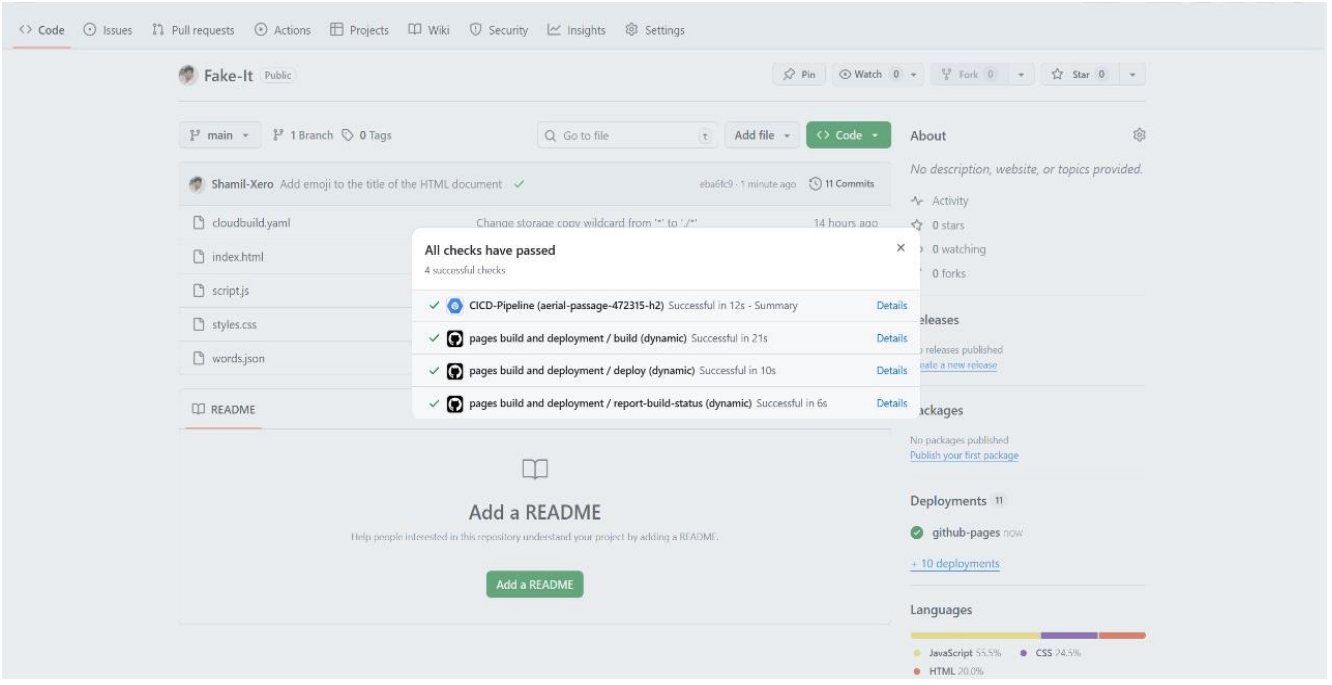
- In the GCP Console, navigate to "Cloud Build" to monitor the progress of build and deployment.
- Once the pipeline is complete, web pages will be copied to the specified Google Cloud Storage bucket.

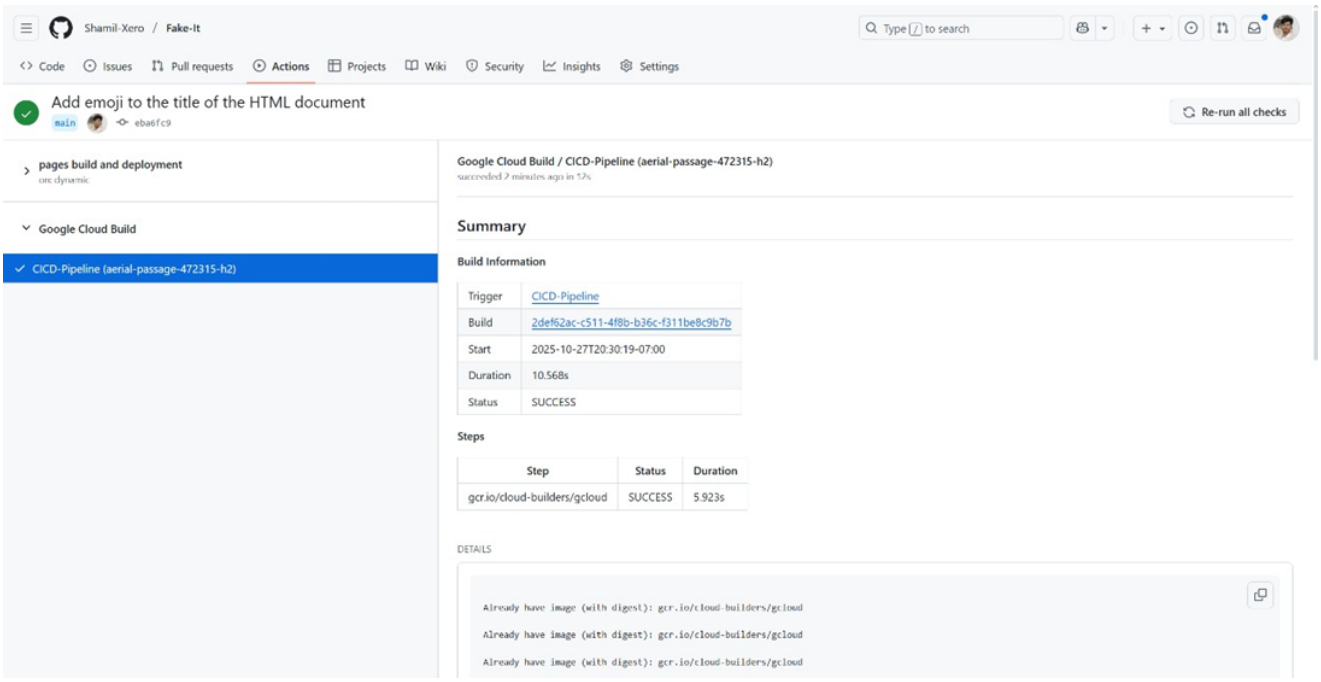
Step 8: Access Deployed Web Pages

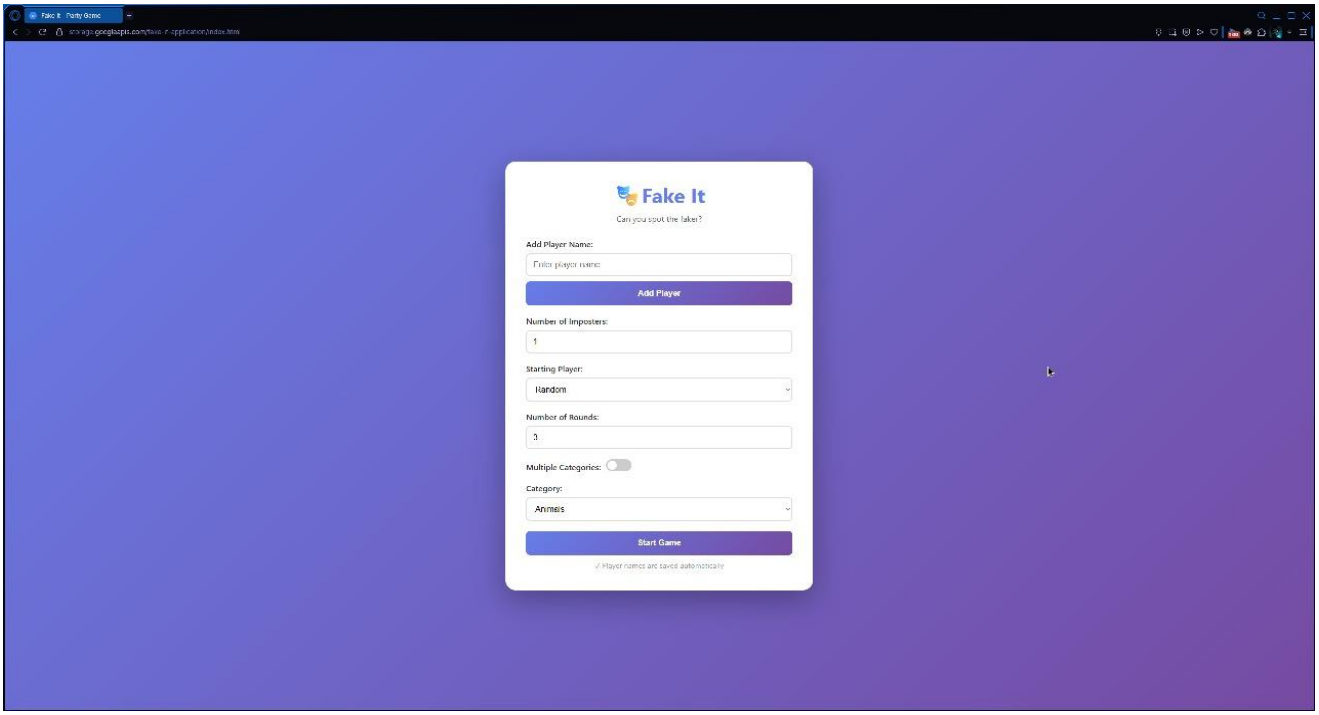
- Configure Apache web server to serve web pages from the Google Cloud Storage bucket.
- Access deployed web pages by visiting the appropriate URL.

Output:


Shamil-Xero / Fake-It
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Fake-It
Public
Pin
Watch 0
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
Shamil-Xero Add emoji to the title of the HTML document
eba6fc9 · 1 minute ago
11 Commits
cloudbuild.yaml
Change storage copy wildcard from '*' to '/*'
14 hours ago
index.html
script.js
styles.css
words.json
README
All checks have passed
4 successful checks
CICD-Pipeline (aerial-passage-472315-h2) Successful in 12s - Summary
Details
pages build and deployment / build (dynamic) Successful in 21s
Details
pages build and deployment / deploy (dynamic) Successful in 10s
Details
pages build and deployment / report-build-status (dynamic) Successful in 6s
Details
Activity
0 stars
0 watching
0 forks
releases
releases published
eate a new release
ackages
No packages published
Publish your first package
Deployments 11
github-pages now
+ 10 deployments
Languages
JavaScript 55.5%
CSS 24.5%
HTML 20.0%
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README


Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Fake-It
Public
Pin
Watch 0
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
Shamil-Xero Add emoji to the title of the HTML document
eba6fc9 · 1 minute ago
11 Commits
cloudbuild.yaml
Change storage copy wildcard from '*' to '/*'
14 hours ago
index.html
script.js
styles.css
words.json
README
All checks have passed
4 successful checks
CICD-Pipeline (aerial-passage-472315-h2) Successful in 12s - Summary
Details
pages build and deployment / build (dynamic) Successful in 21s
Details
pages build and deployment / deploy (dynamic) Successful in 10s
Details
pages build and deployment / report-build-status (dynamic) Successful in 6s
Details
Activity
0 stars
0 watching
0 forks
releases
releases published
eate a new release
ackages
No packages published
Publish your first package
Deployments 11
github-pages now
+ 10 deployments
Languages
JavaScript 55.5%
CSS 24.5%
HTML 20.0%
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README





Result:

To create a GitHub account and set up a basic CI/CD pipeline on GCP. Connect GitHub repository to GCP, configure CI/CD using Cloud Build, and automatically deploy web pages to an Apache web server has been done successfully.