StatsSync Pipeline - TECHNICAL REPORT

Automated Practice-Stats Synchronization Using GitHub Actions

Author: Amjad Ali Kudsi Date: November, 2025

1. Introduction

This report describes the full design, reasoning, and implementation of a system that updates three practice statistics in a portfolio website and GitHub profile automatically. The system was designed to provide clean stat updates, after implementing a series of minor cleaning and optimizing strategies like: avoid unnecessary deployments, remove hidden elements from the main profile, and preserve a single point of manual input.

The final solution uses two GitHub repositories - a controller repository and a profile repository. They are linked through a custom workflow that propagates values from the controller repository to the profile repository. The system provides consistent GitHub Pages deployments, and clear workflow notifications.

2. Background Context

My personal portfolio website contains three visible counters:

1. Total number of projects completed
2. Total CodeSignal practice questions solved
3. Total LeetCode questions solved

These values were hard coded into the JavaScript file of the portfolio. Updating them required manual editing of the source files and redeploying the site, based on my daily study sessions. This resulted in repetitive and unnecessary work.

The long-term goal was a system that required changing only three values once per day while allowing both the GitHub profile and the portfolio website to update automatically.

Because CodeSignal and LeetCode do not provide direct access to these counters through any public or unrestricted application program interface, it was necessary to store the values manually in a controlled location. GitHub serves as this controlled location, since it can store structured data and expose that data over a public and stable URL through GitHub Pages.

3. Initial Architecture

This section describes the earlier approach that was used before transitioning to the two-repository system.

3.1 Overview

The initial design placed both the source of truth and the automated output inside one repository. The GitHub profile README contained a hidden block of JSON inside a details tag. A GitHub Actions workflow extracted that JSON, created a stats.json file, and committed the file back into the same repository. GitHub Pages then served that file.

The portfolio site fetched that stats.json file and updated the counters dynamically.

3.2 Workflow Steps

When the user updated the README, a push event triggered the workflow. The workflow steps were:

1. Check out the repository
2. Locate the lines between:

```
<!-- stats:start -->
 and
<!-- stats:end -->
```

3. Write the JSON content into stats.json
4. Validate the JSON
5. Commit stats.json back to the repository
6. Trigger GitHub Pages deployment through the commit

This resulted in two commits to the same repository for every single update:
• Commit A: the user editing the README
• Commit B: the workflow committing the generated stats.json

Both commits landed on the same branch, which was also the GitHub Pages source branch.

3.3 GitHub Pages Cancelled Deployments

GitHub Pages automatically starts a deployment on every push to the active Pages branch. When Commit A arrived, GitHub started page building. When Commit B arrived a moment later, GitHub started a new page build for B and cancelled the build for A, which resulted in some cancelled jobs.
This is normal behavior because GitHub Pages uses a queue mechanism. If a newer deployment event enters the queue, older deployments for the same branch are cancelled.

As a result, the user always received a cancellation email for the earlier deployment.

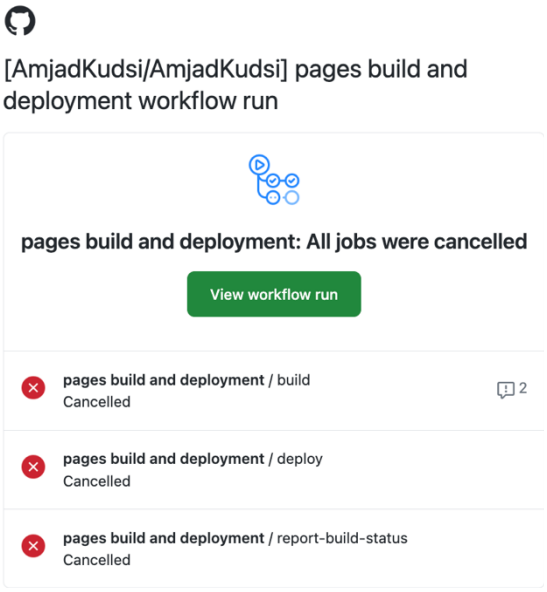


Fig. Email notification - cancelled jobs

4. Design Requirements for the Improved Architecture

The new design had the following explicit requirements:

1. Update the three numbers in one place only
2. No hidden values inside the profile README
3. Only one commit per update in the profile repository
4. Only one GitHub Pages deployment per update
5. Predictable notifications with no cancellations
6. Compatible with Shields dynamic badges
7. Compatible with client side fetching for the portfolio site
8. Automatic propagation of updates with no manual copying

5. Final Architecture

The final design uses a two-repository structure. Only one repository contains GitHub Pages and only one repository receives commits that affect deployment. The other repository serves as the controller.

5.1 Roles of the Two Repositories

Repository A: Controller
This repository contains the source of truth. It stores a single file, stats.json, with the three values. It holds the workflow that copies these values to the profile repository.

Repository B: Profile

This repository hosts the GitHub profile README and GitHub Pages. It contains the final stats.json file used by the profile badges and the portfolio site. It does not contain hidden values or automation code. It receives updates only through the controller workflow.

5.2 Data Flow

1. The user edits stats.json in Repository A
2. A GitHub Actions workflow in Repository A runs
3. The workflow clones Repository B using a personal access token
4. The workflow copies the new stats.json into the root of Repository B
5. The workflow commits and pushes the change
6. GitHub Pages deploys Repository B
7. The portfolio site fetches the updated stats.json from GitHub Pages
8. Shields dynamic badges read the same JSON through their dynamic JSON function

5.3 Pages Hosting Behavior

Repository B is the only repository that uses GitHub Pages. Since only the workflow in Repository A pushes changes to Repository B, each update results in only a single commit to Repository B. GitHub Pages receives one push and produces exactly one deployment with no cancellations.

6. Implementation Details

6.1 Repository A

Repository A functions as the source of truth for the practice statistics. This design isolates the editable data from the repository that triggers GitHub Pages, which ensures predictable deployments and clean commit history.

6.1.1 Structure of Repository A

Repository A contains:

- A single file named stats.json
- A GitHub Actions workflow located at .github/workflows/sync-stats-to-profile.yml

The stats.json file has the structure:

```
{
  "projects": 40,
  "codesignal_practices": 374,
  "leetcode": 200
}
```

These values represent the counters displayed in the GitHub profile as dynamic Shields badges and in the portfolio website as animated counters.

6.1.2 Why stats.json is the Source of Truth

Storing the values directly in a JSON file provides:

- Minimal complexity
- Machine readable structure
- Compatibility with both Shields dynamic JSON badges and client side JavaScript
- Easy manual updates

It also avoids embedding values inside README files, which would create hidden blocks or require parsing Markdown.

6.1.3 Trigger Conditions in Repository A

The workflow in Repository A triggers when stats.json changes:

```
on:
  push:
    paths:
      - stats.json
```

This causes the workflow to run only when practice values are manually updated. It never triggers on unrelated commits. This targeted trigger prevents unnecessary synchronization.

6.2 Permissions and Secret Management

GitHub Actions provides an internal token called GITHUB_TOKEN. This token grants write access only to the repository where the workflow runs, and it cannot push to other repositories. Fine grained personal access tokens support selecting specific repositories and permissions, which allows controlled cross repository writes.

Since Repository A must update Repository B, a personal access token was created with write access restricted only to Repository B. The token was then stored in Repository A under Secrets and variables to keep it secure and prevent exposure.

6.3 Workflow in Repository A

The workflow in Repository A does the following:

1. Checks out Repository A
2. Clones Repository B using the personal access token
3. Copies stats.json from A into B
4. Checks if the file changed

5. Commits and pushes if there is a change

6.3.1 Checkout Step

The workflow begins by checking out Repository A. GitHub documentation specifies that actions require checkout to interact with repository content.

```
- name: Checkout controller repo
  uses: actions/checkout@v4
```

6.3.2 Cloning Repository B

Repository B is cloned using an authentication URL. This allows push permission based on the stored token.

```
git clone https://x-access-token:${{
secrets.PROFILE_REPO_PAT }}@github.com/AmjadKudsi/AmjadKudsi.git profile
```

6.3.3 Copying Data

The updated file from Repository A is copied:

```
cp stats.json profile/stats.json
```

6.3.4 Detecting File Changes

The workflow tests whether stats.json changed:

```
if git diff --quiet -- stats.json; then
  echo "changed=false" >> $GITHUB_OUTPUT
  exit 0
fi
```

If no changes are detected, the workflow exits early, preventing unnecessary pushes and Pages builds.

6.3.5 Committing and Pushing

If there is a change, the workflow commits the updated file:

```
git config user.name "github-actions[bot]"
git config user.email "41898282+github-
actions[bot]@users.noreply.github.com"
git add stats.json
```

```
git commit -m "Update stats.json from controller repo"
git push
```

Repository B now has exactly one commit per update cycle.

6.4 Repository B Structure

Repository B is the GitHub profile repository named:

AmjadKudsi / AmjadKudsi

GitHub renders the README of the repository whose name matches the username as the user profile README.

In the initial design, the profile README contained an unwanted hidden JSON block inside HTML comments or a details tag. This content was removed in the final design, since Repository A now stores the source of truth and populates Repository B automatically.

Shields provides a dynamic JSON badge feature that reads values from a remote JSON resource. The badge syntax follows the Shields documentation, which describes the dynamic JSON endpoint features.

Example badge:

```
<img
src="https://img.shields.io/badge/dynamic/json?url=https://amjadkudsi.github.io/AmjadKudsi/stats.json&query=$.projects&label=Projects&color=1f6feb&style=flat" />
```

Three badges are displayed for:

- Projects
- CodeSignal practices
- LeetCode

Each badge automatically updates when stats.json changes.

6.5 Portfolio Integration

The portfolio website uses a fetch request to load stats.json from:

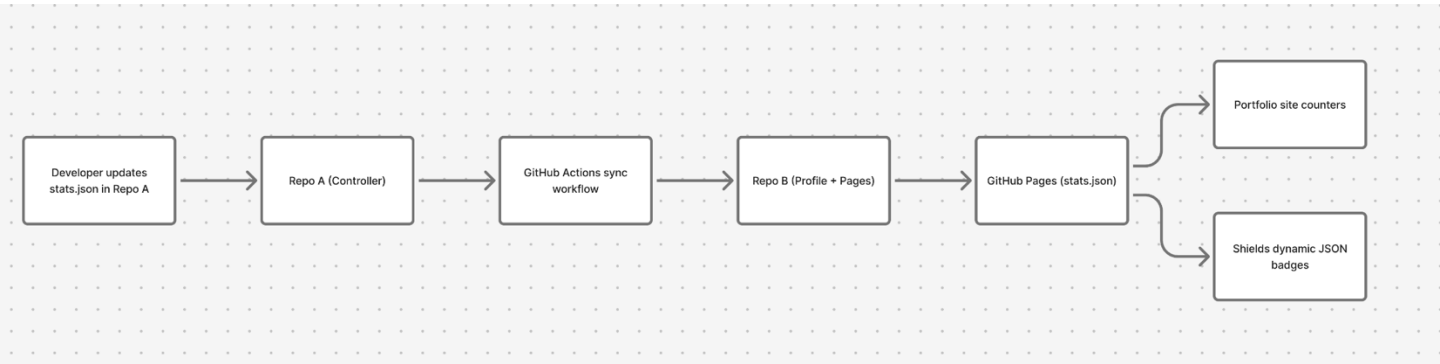
https://amjadkudsi.github.io/AmjadKudsi/stats.json

This JSON is then passed to an animateCounter function that increments numbers smoothly for visual effect.

Since stats.json is hosted through GitHub Pages, it is publicly accessible, fast to retrieve, and version controlled.

7. System Behavior (Deployment Timeline)

The deployment timeline per update is:



Figma

Since there is only one commit in Repository B per update, GitHub Pages performs only one build. No deployments are cancelled.

8. Benefits of the Final System

The new architecture offers the following advantages:

- No hidden data inside the profile README
- Fully automated propagation of stats from controller to profile
- Clean workflow history with no cancellation emails
- Single source of truth for daily updates
- Dynamic profile badges without manual editing
- Synchronization logic isolated in one place
- GitHub Pages receives exactly one commit per update, preventing concurrency cancellation
- Portfolio and profile always show consistent numbers

This system is significantly more maintainable than the initial single repository version.

9. Future Enhancements

Possible future work includes:

LeetCode Automation - If the LeetCode API stabilizes or third-party endpoints become available, a scheduled workflow in Repository A could fetch the LeetCode solved count programmatically.

CodeSignal Summaries - Similarly, if CodeSignal releases public API endpoints for problem counts, these can be integrated into the same synchronization pipeline.

Historical Tracking - Repository A could store past values in a history folder and generate charts using GitHub Actions.

Scheduled Updates - The workflow could run daily on a schedule and fetch external data instead of requiring manual updates.

10. Conclusion

The two-repository architecture achieves all the goals set for the automated update system. It reduces noise, avoids hidden structures inside the profile README, enables controlled propagation of JSON updates, and provides stable and predictable deployments in the profile repository.

The design is fully scalable and future proof, and it removes unnecessary workflow cancellations and email notifications while preserving the convenience of making only one manual edit per day.

This system maintains:

- Clean Pages deployments
- Clean commit history
- Automated portfolio updates
- Automated badge updates
- A single well-organized source of truth

It is suitable for long term maintenance and easy to expand with new metrics.