PROMPT:

Using the GitHub MCP Server, get all my closed issues from my GitHub repo "CESARDELATORRE/spec-driven-vibe-coding-challenge" and put is as "Done Intems" in the KANBAN Backlog of my GitHub project "Project Specs-Driven Challenge - Agentic Chat for AMG" here: <https://github.com/users/CESARDELATORRE/projects/3/views/1>

ANSWER:

I can’t directly execute GitHub project mutations from here because the current workspace doesn’t expose a GitHub MCP Server tool

A screenshot of a computer

AI-generated content may be incorrect.

I can’t directly execute GitHub project mutations from here because the current workspace doesn’t expose a GitHub MCP Server tool (no issue/project tools are registered), but I’ll give you everything you need to:

1. Pull all closed issues from repo CESARDELATORRE/spec-driven-vibe-coding-challenge
2. Add any not-yet-present ones to your user Project (Projects v2) “Project Specs-Driven Challenge - Agentic Chat for AMG” (project number 3)
3. Set their status to Done (or put them in your “Done Items” column if that’s backed by the Status single‑select field)

Below are three approaches: A. GitHub CLI (fast), B. Raw GraphQL (fine-grained), C. How this would look via a GitHub MCP Server (so you can wire tools later).

**A. One‑shot GitHub CLI automation (recommended)**

Prereqs:

* Install gh: [https://cli.github.com/](vscode-file://vscode-app/c:/Users/cesardl/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
* Auth: gh auth login (ensure scopes: repo, project)
* jq installed (for JSON parsing). On Windows Git Bash you can install via choco install jq or winget install jqlang.jq.

Script (bash) – adds all CLOSED issues not already in project 3, then sets Status=Done if such an option exists.

|  |
| --- |
| #!/usr/bin/env bash set -euo pipefail  USER\_LOGIN="CESARDELATORRE" REPO\_OWNER="CESARDELATORRE" REPO\_NAME="spec-driven-vibe-coding-challenge" PROJECT\_NUMBER=3 # User-scoped project number STATUS\_NAME="Done" # Adjust if your option is named differently (e.g., "✅ Done")  **1. Get project ID + fields**  PROJECT\_DATA=$(gh api graphql -f query=' query($login:String!,$number:Int!) { user(login:$login) { projectV2(number:$number) { id title fields(first:100) { nodes { ... on ProjectV2Field { id name } ... on ProjectV2SingleSelectField { id name options { id name } } } } } } }' -F login=$USER\_LOGIN -F number=$PROJECT\_NUMBER)  PROJECT\_ID=$(echo "$PROJECT\_DATA" | jq -r '.data.user.projectV2.id') STATUS\_FIELD\_ID=$(echo "$PROJECT\_DATA" | jq -r '.data.user.projectV2.fields.nodes[] | select(.name=="Status") | .id') DONE\_OPTION\_ID=$(echo "$PROJECT\_DATA" | jq -r --arg s "$STATUS\_NAME" '.data.user.projectV2.fields.nodes[] | select(.name=="Status" and .options!=null) | .options[] | select(.name==$s) | .id')  if [[ -z "$DONE\_OPTION\_ID" || "$DONE\_OPTION\_ID" == "null" ]]; then echo "ERROR: Could not find Status option named '$STATUS\_NAME'. Existing options:" echo "$PROJECT\_DATA" | jq -r '.data.user.projectV2.fields.nodes[] | select(.name=="Status") | .options[]?.name' exit 1 fi  echo "Project: $(echo "$PROJECT\_DATA" | jq -r '.data.user.projectV2.title')" echo "Project ID: $PROJECT\_ID" echo "Status Field: $STATUS\_FIELD\_ID" echo "Done Option: $DONE\_OPTION\_ID"  **2. Fetch CLOSED issues (paginate if you have >100)**  ISSUES=$(gh api graphql -f query=' query($owner:String!,$name:String!) { repository(owner:$owner, name:$name) { issues(states:CLOSED, first:100, orderBy:{field:UPDATED\_AT, direction:DESC}) { nodes { id number title url projectItems(first:50) { nodes { project { id } } } } } } }' -F owner=$REPO\_OWNER -F name=$REPO\_NAME)  **3. Filter issues not already in the project**  ADDED=0 UPDATED=0 echo "$ISSUES" | jq -c --arg pid "$PROJECT\_ID" '.data.repository.issues.nodes[] | select(.projectItems.nodes[]?.project.id != $pid or (.projectItems.nodes | length == 0))' | while read -r issue; do ISSUE\_ID=$(echo "$issue" | jq -r '.id') ISSUE\_NUM=$(echo "$issue" | jq -r '.number') TITLE=$(echo "$issue" | jq -r '.title')  ADD\_RES=$(gh api graphql -f query='  mutation($projectId:ID!, $contentId:ID!) {  addProjectV2ItemById(input:{projectId:$projectId, contentId:$contentId}) {  item { id }  }  }' -F projectId="$PROJECT\_ID" -F contentId="$ISSUE\_ID")  ITEM\_ID=$(echo "$ADD\_RES" | jq -r '.data.addProjectV2ItemById.item.id')  # Set Status=Done  gh api graphql -f query='  mutation($projectId:ID!, $itemId:ID!, $fieldId:ID!, $optionId:String!) {  updateProjectV2ItemFieldValue(  input:{  projectId:$projectId,  itemId:$itemId,  fieldId:$fieldId,  value:{ singleSelectOptionId:$optionId }  }  ) { projectV2Item { id } }  }' -F projectId="$PROJECT\_ID" -F itemId="$ITEM\_ID" -F fieldId="$STATUS\_FIELD\_ID" -F optionId="$DONE\_OPTION\_ID" >/dev/null  echo "Added + marked Done: #$ISSUE\_NUM - $TITLE"  ADDED=$((ADDED+1))  done  **4. (Optional) Ensure any already-in-project CLOSED issues that aren’t Done get updated**  echo "$ISSUES" | jq -c '.data.repository.issues.nodes[]' | while read -r issue; do ISSUE\_ID=$(echo "$issue" | jq -r '.id') ISSUE\_NUM=$(echo "$issue" | jq -r '.number')  **For existing items we need their project item IDs & current status (not retrieved here).**  **If you need to normalize statuses, you’d run another query listing project items & their field values.**  : # Placeholder for potential normalization logic done  echo "Summary: Added $ADDED new closed issues to the project." |

Run:

1. Save as add-closed-issues-to-project.sh
2. chmod +x add-closed-issues-to-project.sh
3. ./add-closed-issues-to-project.sh

If you have more than 100 closed issues, add pagination (I can help extend).

**B. Raw GraphQL (manual sequence)**

1. Get project & fields (same query as above).
2. Get closed issues with their project items (query above).
3. For each issue lacking a project item referencing your project ID: run addProjectV2ItemById mutation.
4. For each new item, run updateProjectV2ItemFieldValue to set the Status single-select to “Done”.

You can execute each GraphQL payload via: gh api graphql -f query='...' -F varName=value (or POST to [https://api.github.com/graphql](vscode-file://vscode-app/c:/Users/cesardl/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) with a PAT in Authorization header)

**C. MCP Server Tool Design (future automation)**

If you stand up a GitHub MCP Server, you could expose tools like:

1. list\_closed\_issues Input: { repo: "owner/name", first?: int, after?: cursor }  
   Output: [{ id, number, title, url, inProject(projectId): bool }]
2. add\_items\_to\_project Input: { projectNumber, ownerType: "user"|"org", ownerLogin, issueIds: [] }
3. update\_project\_item\_status Input: { projectNumber, ownerLogin, itemId, statusValue: "Done" }
4. sync\_closed\_issues\_to\_project (Composite tool) Input: { repo, projectNumber, statusValue } Output: summary (added, skipped)

Example invocation (pseudo JSON for an MCP client): { "name": "sync\_closed\_issues\_to\_project", "arguments": { "repo": "CESARDELATORRE/spec-driven-vibe-coding-challenge", "projectNumber": 3, "statusValue": "Done" } }

Return payload: { "added": 7, "alreadyPresent": 12, "statusUpdated": 7, "errors": [] }

If you want, I can scaffold a minimal GitHub MCP server in C# or Node later.

**Verifying Success**

After running the script:

1. Open [https://github.com/users/CESARDELATORRE/projects/3/views/1](vscode-file://vscode-app/c:/Users/cesardl/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
2. Ensure new issues appear in the board and have Status = Done (or land in your “Done Items” view).
3. (Optional) Add a Saved View filtered by Status=Done for clarity.

**Common Pitfalls & Tips**

* Status field name mismatch: Sometimes it’s “Status”, “Workflow”, or localized. Query fields to confirm.
* “Done” option may be “Done ✅” or have emoji. Match exact string.
* Duplicate adds: ProjectV2 prevents duplicates by contentId, but you filtered anyway.
* Permissions: PAT must have project & repo read/write (classic token with repo + project scope, or fine-grained with issues + project items).
* Rate limits: Not usually a problem unless adding hundreds; you can batch-sleep.

**Optional Improvement**

Add a GitHub Action (cron nightly) that:

* Checks closed issues not yet in project.
* Adds & sets Done automatically.

Happy to draft that workflow if you’d like—just ask.

Let me know if you’d like:

* Pagination extension
* Converting to PowerShell
* A reusable C# console utility
* MCP server scaffolding for these tools

Just say the word and I’ll generate it.