Automate Processes Within Django Contribution Workflow

GSoC 2025 Proposal for Django by Adya

Table of Contents:

1. Abstract

1.1 An Overview of Django Contribution Workflow Automation

The project idea aims to streamline Django's contribution and development process. The contribution workflow within Django has many manual, lengthy, scrupulous and error-prone processes involved in identifying active PRs, PR validation as per Django code style, following contribution rule, deft review and feedback on PRs, managing aging & stale PR queue, etc. Thus costing valuable time for contributors and maintainers. The active community and high contribution rate in Django lead to long review queues for maintainers.
More successful contributions mean more updates to the Django codebase, which consequently requires regular releases. Django releases are a sensitive manual process that necessitates consistent and procedural assurance of all relevant components and details.
There's a significant need to update both PR management and Releases management for a more efficient and streamlined workflow.

These manual processes consume 20% to 30% spare time of contributors and maintainers. These actions and tasks are currently handled by Trac API, GitHub Actions automation pipeline, scripts and Django releases team. But with some updates in these workflows as per suggestions and inputs from the community, we can build a more automated, faster and monitored contribution pipeline for the Django community, promoting deft contribution, quality code, and better productivity.

By engaging in the community for a decent long time, I learned that contributing to Django is neither very easy nor impossible, it requires lots of dedication, testing, communication, and collaboration. Specifically, this proposal requires a strong understanding of contributing processes, existing Django's GitHub Actions pipeline, Django release process and Django Trac. I believe I gained these by spending months in the community, getting inputs & guidance, working on the community's suggested ticket, releases app , testing on my local forked setup and also working on a few personal projects.

As per the official proposal idea description on the Django GSoC wiki page

- Mentor: **Lilly Foote**
- Project size: Medium

1.2 Existing Django's Workflows and it's Limitation

Before outlining the proposed implementation, it's important to first review the existing contribution workflows and pipeline at a high level. The contribution workflow within Django are handled by several automated and manual processes depending on their sensitivity. Django uses a modular approach and steps that uses the Trac API, CICD pipeline, GitHub actions, and manual human intervention for validation with a mix of manual and automated steps. This project proposal would seek to automate the possible manual processes or add ease in manual processes, updates pipeline and codebase. I gathered it from the Django codebases, docs, references, community forums and Discord.

Here are some points that I understood:

- If someone finds a new issue/bug and wants to contribute on it, he will first query on the Trac website whether a similar issue/ticket has already been created. If not, then he could create a ticket and wait for its triage; after that confirmation, he could proceed to work on it.

- **Pull Request Management and Monitoring**: Django uses a dual-system workflow of *GitHub* and *Trac to manage contributions*.
  Trac (code.djangoproject.com) tracks acceptance, features, bugs and discussions. GitHub (github.com/django/) handles codebase and code changes via standard pull requests.

  Contributors have to refer to Trac ticket numbers (e.g., Refs #1111) in their GitHub PRs title to maintain traceability. Each open PR undergoes a rigorous testing process where maintainers provide feedback and decide for merging. The Django Trac API is designed for fetching data (no write operations). It queries the Trac server for ticket details, comments, and status flags.
  The API endpoints (exposed via views) allow both the Django website and external tools (such as GitHub Actions) to query Trac data.
  Overview of foremost GitHub Actions files

File	Trigger	Key Jobs	Automation Scope
benchmark.yml	Runs only when the "benchmark" label is added and on these PR events: `labeled`, `synchronize`, `opened`, `reopened`	- Measure the performance impact of code changes - Compare benchmarks between the previous commit (`HEAD^`) and the current commit (`HEAD`) - Fail CI/CD build if `PERFORMANCE DECREASED` is detected - Attach benchmark results to the PR as a	N/A

		step summary	
docs.yml	Push to main/docs changes	- The workflow defines two jobs: `docs` and `blacken-docs` - The workflow runs when a pull request is opened, updated, or reopened, but only if changes affect files in the docs/ directory (or its subdirectories) - Navigates to the docs directory and runs sphinx-build	N/A
labels.yml	Triggers on PR events: `edited`, `opened`, `reopened`, `ready_for_review`. Uses `pull_request_target` for security.	- Check PR title for Trac ticket references (e.g., #1234). - **Remove Label**: If the title has a ticket ID but a label exists. - **Add Label**: If the title lacks a ticket ID and the label is missing.	- No Ticket Validation: Doesn't check if #1234 corresponds to a real Trac ticket. - Add **type** label like [Bug], [New feature], [Cleanup/optimization] using Trac meta data
linters.yml	Runs whenever a pull request is **opened**, **updated**, or **reopened** but ignores docs/ dir	automates code quality checks using various linters: - flake8: PEP8 compliance & syntax errors - isort: for proper importing and displaying diff - black: standard code formating	-

- **Django Releases**: Django uses a time-based release schedule, with feature releases every eight months or so. Django release versions are numbered in the form A.B or A.B.C (**MAJOR.MINOR.PATCH(MICRO)**) The MAJOR version number changes when team make big collective, significance changes in modules and APIs, it may include breaking changes, MINOR version when the team add some functionality in a backwards-compatible manner. PATCH is for any bug fixes. Internally, Django also uses a 5 elements format: (MAJOR, MINOR, MICRO, STATUS, ITERATION). The **release manager** keeps the schedule, release plan, track of who's working on what issues, and nagging developers who are in danger of missing deadlines.

Release process schedule: Feature freeze / Alpha ➜ Beta ➜ RC ➜ Final.
As the release date approaches, Django enters a “feature freeze” period. During this phase, only bug fixes and critical improvements are accepted. Once the feature freeze is in place, Django issues a series of release candidates. These are essentially preview versions of the final release. Each release candidate is thoroughly tested by the Django community. After the **RC testing** release management team ensures *version bumping, changelog validation, dependency checks, documentation updates, deprecation warnings and some other attachments* (in terms of security) team then publishes an updated version.

Ovieriew of Django release app
`https://github.com/django/djangoproject.com/tree/main/releases`

1. Provides an admin interface for managing releases and their details.
2. **Release Management**: Tracks Django releases with details like version, release date, end-of-life (EOL) date, and status (alpha, beta, final, etc.), Supports Long-Term Support (LTS) releases and their lifecycle.
3. **Handle artifacts**: Manages file uploads for release artifacts (e.g., .tar.gz, .whl, .checksum).
4. `templatetags/release_notes.py`: Provides custom template tags for generating release notes links and fetching the latest micro-release.
5. `Fixtures`: Preloaded data (doc_releases.json) for historical releases.

1.3 Goals

There’s a need to upgrade and automate the **key** contribution and development workflows by taking inputs from fellow contributors, members and forum discussions.
I tried and with observing all those inputs, suggestions, feedback from seniors to me, the Django PRs lifecycle, the Django Trac website, Natalia’s checklist generator, and my open-source contributions experience, I am setting vital goals:

1. **More Automated PR Management:** Streamlining and automating tasks associated with handling pull requests (PRs) using GitHub Actions, CICD pipeline, scripts and Trac API.
   The updated workflow will provides automated validation of Pull Requests (PRs), faster reviewing and feedback pipeline, ensuring requirements of test coverage, attached documentation, and automated Stale PRs monitoring and handling . It ensures that PRs are efficiently reviewed, labeled, updated, and quality ensured, reducing feasible manual workload.

   - File to modify:
     `django/.github/workflows/labels.yml`
   - Files to create:
     `django/.github/workflows/aging_pr.yml`
     `django/.github/workflows/pr_review.yml`

2. **Automated Releases:** Automating the release management of Django with systematic and procedural approach that involves updating the release app, models, provide details tracking facility, and templates resultant eliminating feasible manual and error-prone tasks and ensure consistency, completeness, and enhanced collaboration in releasing processes.
   My goal is to develop a more automated, efficient, and details tracking releasing process by merging `https://github.com/nessita/checklist-generator` with testings into `https://github.com/django/djangoproject.com/releases` app
   - I Will update the:

`https://www.djangoproject.com/` repository's releases app:
`https://github.com/django/djangoproject.com/tree/main/releases`

See the detailed description of each goal below in the Solution section.

1.4 Motivation

I am an **AI and Python developer** and spend most of my time working and learning open-source models, tools, and technologies and giving feedback and contributions to them. Django is my primary tech stack and I keep up with the community on social media platforms. So I started my GSoC preparation with the **sole purpose of contributing and getting involved in Django**.

I chose to *automate processes within the Django contribution workflow* because it falls under my proficiency in automating things (inspired by my AI agents' works), my knowledge I gained during engagement in the Django community and my experience in open-source contributions.

Proof of Works

1. With my keen interest in the idea of 'Automate processes within Django contribution workflow, I have also done **research** to understand this topic in depth. Please check out my research document here.

2. To understand the practical requirements for Django contribution, and working with related skills, I have also built a small **project "Dashboard"** using CICD pipeline and scripts on Django inspired by the proposal topic that manage and monitor the contribution workflow. You check out my live project here and codebase here

3. For some of my other relevent projects and my Django contributions check below in the About section

2. The Solution and Deliverables: New Workflow

I will update and automate the current workflows with structured and different phase approach with the guidance and collaboration of mentors and stakeholders.
Let me outline the approach, it will require more than one pull request. *(and may require some alteration or new ideas based on future community inputs*).

2.1 Automated PR Management and Monitoring

Key Features and Implementation Plan for PR Monitoring & Management:

- **Aim's Overview**: Automated PR title labelling, style check and validation of PR code, creating a deft PR pre-reviewing pipeline, feedback loop, pipeline for faster merging, managing stale PRs, and updating the Django releases workflow.

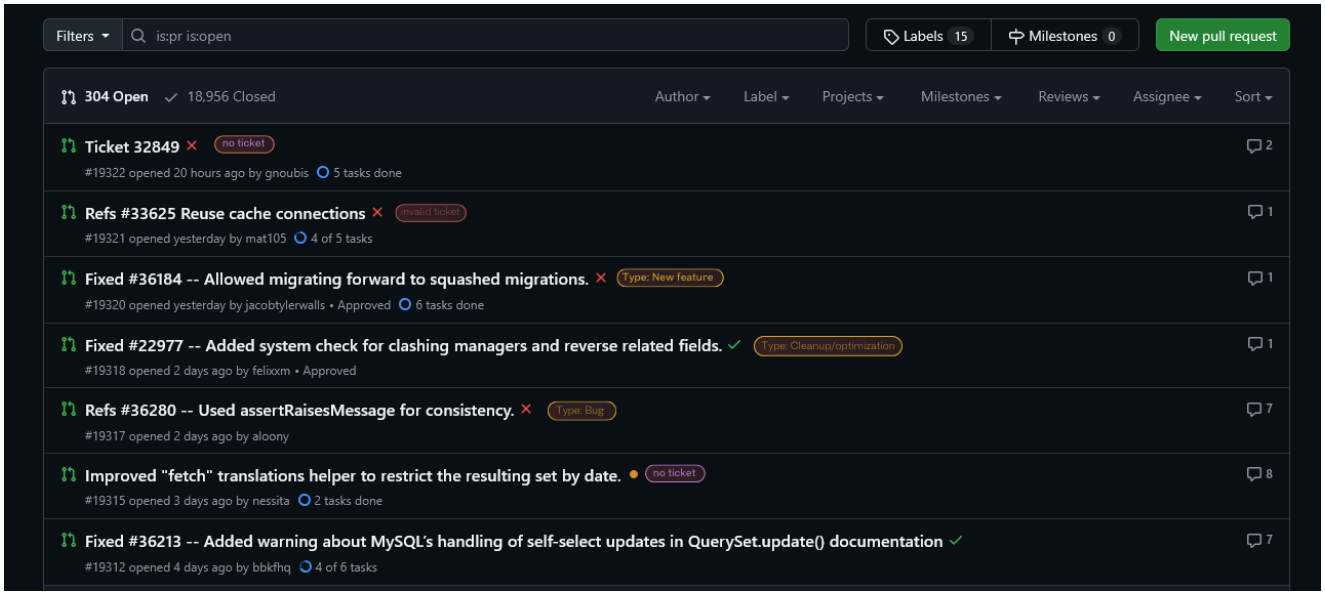
1. Enhanced and Validated PR-Ticket Labeling:
Automating the checking of PR title for ticket with validation and featureful labelling with Trac ticket integration. This will be achieved through some updates in the `labels.yml` file of GitHub/workflows.

- Existing `labels.yml` check whether a PR title has a ticket number, if not it will add a [no ticket] label

- It does not validate the ticket and doesn't have any integration of Trac ticket's metadata.
- In updated `labels.yml`:
    - **Ticket validation:** In a PR title, check if a ticket is mentioned; if not mentioned, pipeline will add [no ticket] label; if mentioned, it will try to validate it and if ticket is invalid/not-found it will add [invalid ticket] label.
    - **Add type label for better understanding PR context for reviewer:** If validation is successful, it will use Trac API to bring `type: bug, new feature, or cleanup/optimization` meta data and label it. It will give a better context to the reviewer about PR.

      We can also fetch other ideal ticket metadata and add other feature label as suitable or as maintainers would suggest as per their requirements
    - It will help to mitigate any wrong ticket mentioning by contributor and help them to correct the mistake, ultimately promoting toward safe reviewing and merging. I used the reference https://gist.github.com/rixx/422392d2aa580b5d286e585418bf6915 to understand the Trac API call, but need some mentorship for efficient Trac API call setup

Here's how updated labelling will look like:




```yaml
name: Trac Ticket Validation
on:
  pull_request_target:
    types: [edited, opened, reopened, ready_for_review]

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  pull-requests: write
  issues: write

jobs:
  ticket_validation:
    name: Validate Trac Tickets and apply labels
    runs-on: ubuntu-latest
    steps:
      - name: Validate and label
        uses: actions/github-script@v7
        env:
          TRAC_API: "https://code.djangoproject.com/jsonrpc" # need help to correct it
        with:
          script: |
            const title = context.payload.pull_request.title;
            const regex = /#[0-9]+[ ,:]?/gm;
            const label = "no ticket";
```

```
            const hasMatch = regex.test(title);
            const labels = context.payload.pull_request.labels.map(l => l.name);
            const owner = context.repo.owner;
            const repo = context.repo.repo;
            const pr_number = context.payload.pull_request.number;
            console.log(`=> Pull Request Title: ${title}`);
            console.log(`=> Labels on PR: [${labels}]`);

            async function validateTicket() {
              if (!hasMatch) {
                if (!labels.includes(label)) {
                  await addLabel(label);
                }
                return;
              }

              const ticketId = title.match(regex)?.[0].replace(/[ ,:]/g, '').substring(1);

              try {
                const response = await fetch(process.env.TRAC_API, {
                  method: 'POST',
                  headers: { 'Content-Type': 'application/json' },
                  body: JSON.stringify({
                    jsonrpc: "2.0",
                    method: "ticket.get",
                    params: [parseInt(ticketId)],
                    id: 1
                  })
                });
                if (!response.ok) throw new Error(`HTTP ${response.status}`);

                const data = await response.json();
                if (data.error) {
                  await handleInvalidTicket(ticketId, data.error.message);
                  return;
                }
                if (!data.result?.[3]?.type) {
                  throw new Error('Invalid ticket data structure');
                }
                const ticketType = data.result[3].type.toLowerCase();
                await handleValidTicket(ticketType);
              } catch (error) {
                console.error('Validation failed:', error);
                await handleInvalidTicket(ticketId, error.message);
              }
            }

            async function handleInvalidTicket(ticketId, reason) {
              await removeLabel('no ticket');
              if (!labels.includes('invalid ticket')) {
                await addLabel('invalid ticket');
              }
              await postComment(
                `**Ticket Validation Failed**\n` +
                `Could not verify #${ticketId}: ${reason}\n` +
                `• Verify ticket exists: https://code.djangoproject.com/ticket/${ticketId}\n` +
                `• Create new ticket: https://code.djangoproject.com/newticket`
              );
            }

            async function handleValidTicket(ticketType) {
              await removeLabel('no ticket');
              await removeLabel('invalid ticket');

              const TYPE_LABELS = {
                'defect': 'type: Bug',
                'enhancement': 'type: New Feature',
                'task': 'type: Cleanup/Optimization'
              };
              const label = TYPE_LABELS[ticketType] || 'type: Other';
              await syncLabels(label);
            }

            async function syncLabels(newLabel) {
              const typePrefix = 'type: ';
              const currentLabels = labels.map(l => l.name);

              for (const label of currentLabels.filter(n => n.startsWith(typePrefix))) {
                if (label !== newLabel) await removeLabel(label);
              }

              if (!currentLabels.includes(newLabel)) {
                await addLabel(newLabel);
              }
            }

            async function addLabel(label) {
```

```
            await github.rest.issues.addLabels({
              owner, repo, issue_number: pr_number,
              labels: [label]
            });
          }

          async function removeLabel(label) {
            try {
              await github.rest.issues.removeLabel({
                owner, repo, issue_number: pr_number,
                name: label
              });
            } catch (error) {
              if (error.status !== 404) throw error;
            }
          }

          async function postComment(message) {
            await github.rest.issues.createComment({
              owner, repo, issue_number: pr_number,
              body: message
            });
          }

          await validateTicket();
```

2. Stale PRs Managemnt

An updated and automated way to handle stale PRs with polite reminders to contributors. Instead of flagging stale PRs directly after the inactivity criteria period. Will implement a **three-phase approach:**

- New `stale_pr.yml` pipeline will check PRs that received a review, their related ticket is updated, and there is no any activity from 3 months to avoid prematurely flagging before start working on it..
- **First reminder**: 3 months inactivity → notifying with a message: *"Are you still working on this...?"*
- **Second reminder**: After one additional month (4 months total) → reminder with comment and notice for stale label and auto-close
- If no response after one week from the second reminder → Put [stale] label and close the PR
- Inactivity criteria will resume if the contributor replies or make any update after any of two first reminders
- Also, provide a manual intervention option to make changes in stale managements for better control

  (The inactivity period criteria and reminder style can be changed according to the suggestions by community for a better and polite environment)

```yaml
name: Stale PR Management

on:
  schedule:
    - cron: '0 0 * * 6'  # Runs weekly on Saturday at midnight

permissions:
  pull-requests: write
  issues: write

jobs:
  check_stale_prs:
    name: Check and Manage Stale PRs
    runs-on: ubuntu-latest
    steps:
      - name: Fetch and Process PRs
        uses: actions/github-script@v7
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          script: |
            const owner = context.repo.owner;
            const repo = context.repo.repo;
            const now = new Date();
```

```
            // Helper function to calculate a date before now
            function getDateBeforeNow(days = 0) {
              const date = new Date(now);
              date.setDate(date.getDate() - days);
              return date;
            }

            // Inactivity thresholds
            const threeMonthsBeforeNow = getDateBeforeNow(90);
            const fourMonthsBeforeNow = getDateBeforeNow(120);
            const oneWeekBeforeNow = getDateBeforeNow(7);

            // Helper function to paginate open PRs
            async function getAllOpenPRs() {
              let allPRs = [];
              let page = 1;
              const perPage = 100;
              while (true) {
                const { data: prs } = await github.rest.pulls.list({
                  owner,
                  repo,
                  state: "open",
                  per_page: perPage,
                  page: page
                });
                if (prs.length === 0) break;
                allPRs = allPRs.concat(prs);
                page++;
              }
              return allPRs;
            }

            async function processPRs() {
              const pullRequests = await getAllOpenPRs();
              console.log(`Total PRs fetched: ${pullRequests.length}`);

              for (const pr of pullRequests) {
                const prNumber = pr.number;
                const updatedAt = new Date(pr.updated_at);

                // Fetch PR labels
                const { data: labels } = await github.rest.issues.listLabelsOnIssue({
                  owner,
                  repo,
                  issue_number: prNumber
                });
                const labelNames = labels.map(l => l.name);

                // Fetch PR timeline to check for contributor activity (comments & events)
                const { data: timeline } = await github.rest.issues.listEvents({
                  owner,
                  repo,
                  issue_number: prNumber,
                  per_page: 100
                });
                const lastContributorActivity = timeline
                  .filter(event => event.actor && event.actor.type === "User")
                  .map(event => new Date(event.created_at))
                  .sort((a, b) => b - a)[0] || updatedAt;

                // Rule 1: 3 months inactivity → Add reminder comment
                if (lastContributorActivity < threeMonthsBeforeNow && !labelNames.includes('stale')) {
                  await github.rest.issues.createComment({
                    owner,
                    repo,
                    issue_number: prNumber,
                    body: `👋 This PR has been inactive for 3 months. Are you still working on this?
If you're facing issues, discuss them in our forum for help. Kindly give updates.`
                  });
                  console.log(`PR #${prNumber} received a reminder comment.`);
                }

                // Rule 2: 4 months inactivity → Add warning comment
                if (lastContributorActivity < fourMonthsBeforeNow && !labelNames.includes('stale')) {
                  await github.rest.issues.createComment({
                    owner,
                    repo,
                    issue_number: prNumber,
                    body: `⚠️Stale Notice: Dear author, this PR has been inactive for 4 months. If no
action is taken soon, it may be marked as stale and closed. Kindly give your updates, You can also
handover to a fellow contributor.`
                  });
                  console.log(`PR #${prNumber} received a warning comment.`);
                }

                // Rule 3: 1 week inactivity after warning → Mark as stale & close
                if (lastContributorActivity < oneWeekBeforeNow && !labelNames.includes('stale')) {
```

```
                // Add the stale label first
                await github.rest.issues.addLabels({
                  owner,
                  repo,
                  issue_number: prNumber,
                  labels: ['stale']
                });
                console.log(`PR #${prNumber} marked as stale.`);

                // Then add the closing comment
                await github.rest.issues.createComment({
                  owner,
                  repo,
                  issue_number: prNumber,
                  body: `🚨 This PR has been inactive for too long. Making it 'Stale' and Closing
due to inactivity. If you are still working on this, You can easily reopen the PR or ask a maintainer
for help..`
                });

                // Finally, close the PR
                await github.rest.pulls.update({
                  owner,
                  repo,
                  pull_number: prNumber,
                  state: "closed"
                });
                console.log(`PR #${prNumber} marked as stale and closed.`);
              }
            }
          }

          await processPRs();
```

3. Faster and Automated PR Feedback and Reviewing Management

Creating a new pipeline `pr_review.yml` to add an automation of preliminary testing and reviewing of codes on same factors to give an immediate result/overview of PR to both contributor and reviewer, ultimately promoting faster review and merging.

Updated pipeline will have various steps and jobs for testing the following factors in a PR:

- **Code Check CI Status**: Checking if the contributor follows the contribution guideline for code style and docs formatting. The `linters.yml` and `docs.yml` are already handling it, so I tried to integrate their results and not to make a duplicate job and also check to ensure correct library and dependencies and finding any broken code.
- **Check tests:** Some tests are written (test-coverage)
- **Documentation check**: Whether the contributor added the documentation of his changes, check for `versionchanged/versionadded` notes when required, whether the line length of documentation is valid, and deprecation warnings included (if related)
- **Release note:** Check for release note (if related)
- If a PR *passed* all tests. Label the PR with [ready for review] label and use its logic for finding an SME reviewer.

- This pipeline is quite delicate. The role of mentor and volunteers will be very crucial for mentorship and guidance to set up the more sophisticated, updated and accurate pipeline. It could be changed to check for any more factors as suggested by the community and fellow contributor
- It will help contributors to find and solve issues on time and save potential spare time of both contributors and maintainers

```yaml
name: PR Review

on:
  pull_request:
    types: [opened, synchronize, reopened]
    paths-ignore:
      - 'docs/releases/**'
  workflow_dispatch:

concurrency:
  group: "${{ github.workflow }}-${{ github.ref }}"
  cancel-in-progress: true
```

```yaml
permissions:
  contents: read
  pull-requests: write

jobs:
  # Tried to reuse CI results instead of duplicating checks
  ci-status-check:
    runs-on: ubuntu-latest
    outputs:
      ci_status: ${{ steps.check-ci.outputs.status }}
    continue-on-error: true
    steps:
      - name: Check CI Status
        id: check-ci
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const pull = await github.rest.pulls.get({
              owner, repo, pull_number: number
            });
```

The full code is provided below (after conclusion section) on page no. 20 or 21

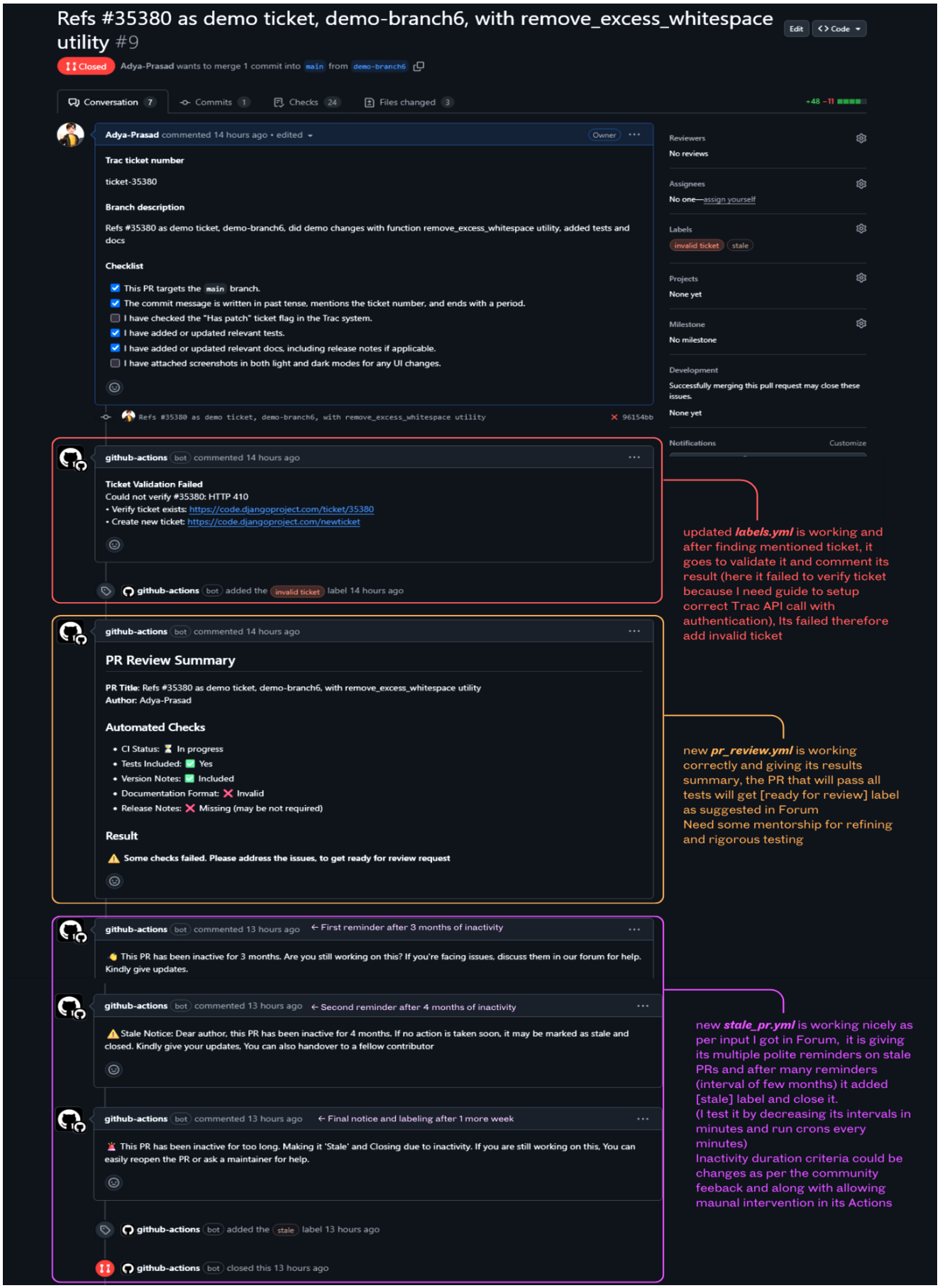
During practical contribution, it will be good to try to add automated available SME reviewer checking feature (with consent and guide under Django team)

```
Required Labels:
- no ticket
- invalid ticket
- bug
- new feature
- cleanup/optimization
- stale
- ready for review
```

- **Testing strategy**:
    a. GitHub Actions testing is quite challenging, When I asked and searched about it in community I didn't find much, this is the key area where I need guide from stakeholders.
    b. We can test it using the GitHub Actions Toolkit
    c. **I used**: Running all GitHub Actions locally on multiple and variety of PR with intentional mistakes
    d. Running the updated GitHub actions manually over the historical PR of Django to test its accuracy in a sandbox environment

1. Validating the ticket labeling	**2. Stale PRs monitoring**	**3. Automated PR Feedback and Review Process**
a. Debugged and updated many times as per the performance b. Tested and verified with `yaml` linter tool c. Tested locally multiple times on multiple and variety of PR with intentional mistakes	a. Debugged and updated many times as per the performance b. Tested and verified with `yaml` linter tool c. Tested locally multiple times on multiple and variety of PR with intentional mistakes d. Tested with temporary changes (decreasing to minutes) in duration criteria and cron schedule every minute for rapid testing,	a. Debugged and updated many times as per the performance b. Tested and verified with `yaml` linter tool c. Tested locally multiple times on multiple and variety of PR with intentional mistakes

Demonstration Screenshot of whole updated GitHub Actions workflows (during my testings):



2.2 Automated Releases

4. More Enhanced and Verified Releases

Updating the release management will be a systematic approach for automating and streamlining the release app for the **release team**. I learnt about Django's release process from:
https://docs.djangoproject.com/en/dev/internals/howto-release-django/
https://docs.djangoproject.com/en/dev/internals/release-process/

I will update the `release` app of the `Djangoproject`
The existing Django release app handles different version releases by tracking its details like LTS support, EOL, Release Date, and MAJOR.MINOR.MICRO format with its Status and Iterations. Also, manage the artifacts of each release. But lacks some essential features and automation and also unable to handle and manage different kinds of releases.

So I will merge the `github.com/nessita/checklist-generator` app within the `release` app of the `Djangoproject` with tests and documentation that provide more features and automated release management to the release team:
Benefits:

- Release Management System
    - Manages different types of Django releases (Feature, Security, Bug Fix, Pre-releases)
    - Tracks version numbers, release dates, and end-of-life dates
    - Handles both LTS (Long Term Support) and regular releases
- Security Management
    - Tracks CVE (Common Vulnerabilities and Exposures) entries
    - Manages security issues and their fixes across different Django versions
    - Coordinates security release processes
- Release Process Automation
    - Generates checklists for release procedures
    - Creates release announcements and blog posts
    - Manages release artifacts (tarballs, wheels, checksums)

Structure of djangoproject release app **before** merging and updating:

```
releases/
    ├── fixtures/
    │   └── doc_releases.json
    ├── migrations/
    │   ├── 0001_squashed_0004_make_release_date_nullable.pyy
    │   ├── 0005_release_artifacts.py
    │   └── __init__.py
    ├── templatetags/
    │   ├── __init__.py
    │   ├── data_format.py
    │   ├── generator_extras.py
    │   └── release_notes.py
    ├── __init__.py
    ├── admin.py
    ├── app.py
    ├── context_processors.py
    ├── logging.py
    ├── models.py
    ├── test.py
    ├── urls.py
    ├── utils.py
    └── views.py
```

Structure of djangoproject release app **after** merging and updating:

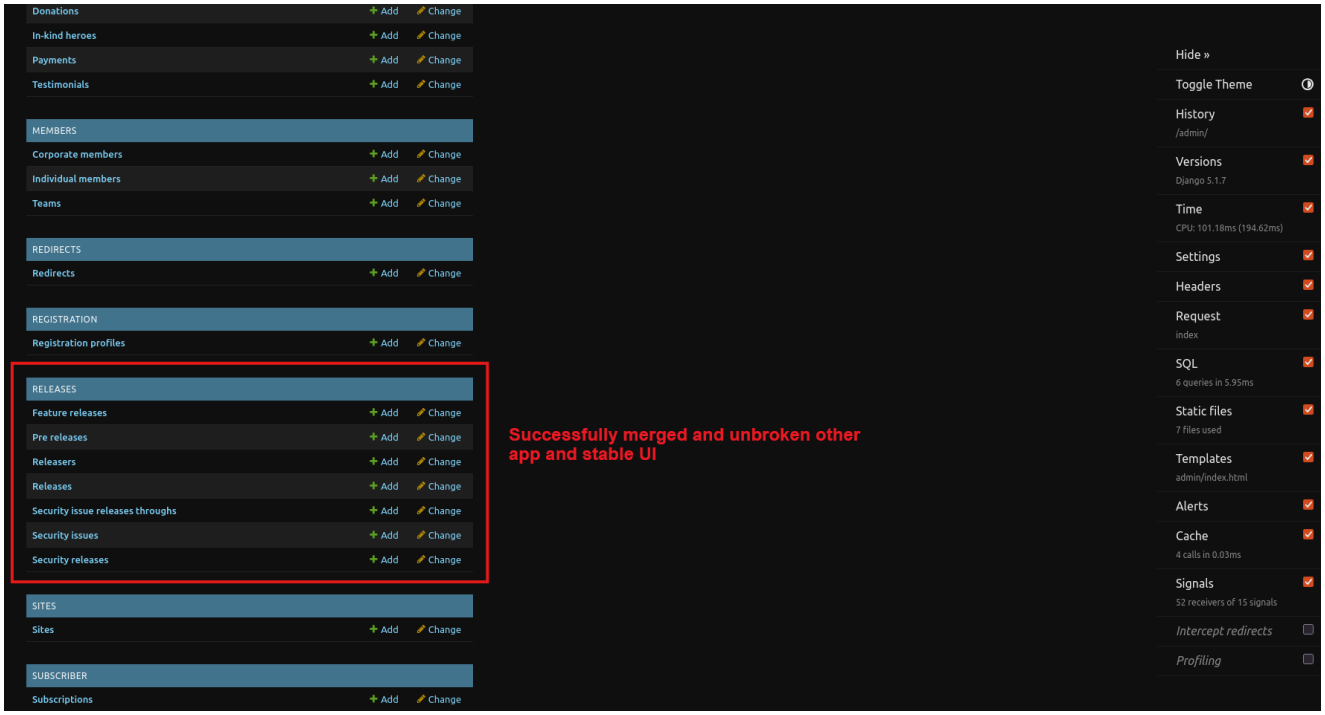
```
releases/
```

```
    ├── fixtures/
    │   └── doc_releases.json
    ├── migrations/
    │   ├── 0001_squashed_0004_make_release_date_nullable.pyy
    │   ├── 0005_release_artifacts.py
    │   ├── Not generated two files (may be this is causing error)
    │   └── __init__.py
    ├── templates/generator
    │   ├── _build_release_binaries.md
    │   ├── _close_security_pr.md
    │   ├── _cve_publication.md
    │   ├── _make_release_public.md
    │   ├── _relaser_info.rst
    │   ├── _stub_release_notes.md
    │   ├── release-security-skeleton.md
    │   ├── release-skeleton.md
    │   ├── release_alpha_blogpost.rst
    │   ├── release_beta_blogpost.rst
    │   ├── release_final_blogpost.rst
    │   ├── release_rc_blogpost.rst
    │   ├── release_security_archive.rst
    │   ├── release_security_blogpost.rst
    ├── templatetags/
    │   ├── __init__.py
    │   ├── data_format.py
    │   ├── generator_extras.py
    │   └── release_notes.py
    ├── __init__.py
    ├── admin.py
    ├── app.py
    ├── context_processors.py
    ├── logging.py
    ├── models.py
    ├── test.py
    ├── urls.py
    ├── utils.py
    └── views.py
├── requirements/ # updated to add some required dependencies
```

🌟I worked on both repositories for some time and understand their feature and codebase. Then start merging it.
To initiate merging in my setup, I first ran the base tests of the project, then started to write some test cases, run it for assurance. For actual file merging, I first started with merging `tests.py` and then `admin.py` and others
I successfully able to merge it and run and use the admin panel (it runs smoothly without breaking another functionality)

Screenshot of updated `release` app features:

Screenshot of merged and updated `Djangoproject` admin panel:



Note: After merging, the release app is running, but some releases fields are showing programming errors (HTTP/1.1" 500). **Mentorship** on a better alternate merging approach or debugging the current particular issues would be great. I would also take feedback for efficient testing, especially for security releases.
You can check the all updated and merged file here in my repository:
https://github.com/Adya-Prasad/djangoproject.com-checklist-merged

Updated `admin.py`

```python
from django.contrib import admin
from django.http import HttpResponse
from django.template.loader import render_to_string

from .models import (
    FeatureRelease,
    PreRelease,
    Release,
    Releaser,
    SecurityIssue,
    SecurityIssueReleasesThrough,
    SecurityRelease,
)


def render_checklist(request, queryset):
    assert len(queryset) == 1, "A single item should be selected"
    [instance] = queryset
    context = {
        "instance": instance,
        "releaser": instance.releaser,
        "slug": instance.slug,
        "version": instance.version,
        "title": instance.__class__.__name__,
        **instance.__dict__,
    }
    if (release := getattr(instance, "release", None)) is not None:
        context["release"] = release
    if (data := getattr(instance, "get_context_data", None)) is not None:
        context.update(data)
    checklist = render_to_string(instance.checklist_template, context, request=request)
    return HttpResponse(checklist, content_type="text/markdown")
```

Kindly see the full updated admin.py file below (after conclusion section) on page no. ~ 26

➜ Key Stakeholders

My project may require the participation, guidance and consent of the following Django Internals team. By spending such time in the Django Community, I learned that the teams are usually busy with their regular workloads, It will be better to request and ping when required and also take feedback from other community volunteers.

- Mergers team
- Releaser team
- Triage & review team
- Mentor: **Lily Foote** and repository maintainers

3. Schedule, Milestone & Timeline

Community bonding and guidance on contribution approach (1 Week, 08 May - 14 May)

Before starting coding, it is recommended to do some preparations, auditing workflows and community bonding

- Get connected with Mentors for their guidance.
- Gathering any remaining guidance
- **Connecting with stakeholders and proposing them to come on board and provide guidance to ensure unbroken updation.**
- Figure out if there is any better approach to the problem or if the solution could be improved in any way or any new idea required.
- Start Implementing Phases (2 Phases): Each idea implementation will have definite time spans to save both stakeholders' and my time and ensure efficient completion. Timeline below is estimated and can be slightly changed during contribution
- On average, I will be putting in around 5 to 6 hours a day

[Phase 1: 7 weeks]

3.1. *First Milestone:* Automated PR Ticket Labeling and Validation (2 weeks, 15 May - 29 May)

- Setup the Project
- Creating Trac ticket if required
- Gathering inputs from the community
- Learn about the correct way of using Trac API
- Creating a full updated `labels.yml` file
- Testing and feedback from stakeholders
- Creating its docs
- Creating and merging final PR

3.2. *Second Milestone:* Stale PR Management (2 weeks, 30 May - 12 June)

- Reusing the previous setup or setting up the new project as per the condition
- Creating a new `stale_pr.yml` file

- Gathering inputs from community, opinion of volunteers from a forum posts on stale period and polite way to notify & closing PR and implementing the collectives.
- Taking feedback and confirmation on testing from stakeholders and mentor
- Creating its docs
- Creating and merging final PR

3.3. *Third Milestone:* Automated PR Review Pipeline (3 weeks, 13 June - 3 July)

- Reusing the previous setup or setting up the new project as per the condition
- Creating a new `pr_review.yml` file draft and showing it to the mentor and stakeholder
- Gathering inputs from the community on the draft, collecting any suggestion of changes and any new feature wanted and implementing the draft again
- Showing the updated draft again
- Testing the final draft
- Creating its docs
- Making and merging final PR

Deliverables: 3 PRs respectively for `labels.yml`, `stale_pr.yml` and `pr_review.yml`

[GSoC halfway evaluation will be completed]
[Mid term evaluation date: 5 July]

Maybe I will need a break for a week (10 days) for some possible exams of my department (in July)

[Phase 2: 4 weeks]

3.4. *Fourth Milestone:* More Enhanced and Streamlined Releases Process (4 weeks, 14 July - 11 August)

- Setting up the djangoproject repository and checklist generator repository
- Taking guidance, recommendations from the author and mentorship for a better alternate merging approach
- Creating Trac ticket if already not
- Writing and merging test cases and running it
- Merging all other files and creating some new folders and files
- Testing and using the merged and updated releases app
- Testing UI and proper functioning
- Publishing demo for testing for by releases team and taking their feedback
- Updating again as per releases team feedback
- Final testing and approving the new releases app
- Creating docs, screenshots and writing blogs and guides on it
- Creating and merging the PR into the codebase

Deliverable: One more PR on Merged checklist generator and updated release app
Total 4 PRs

10 Days break for any possible institute's assignments and projects

3.5. Buffer Week: Checking Full Implementation, Any Failures and Feedbacks, (21 August - 27 August)

Objective: Finalising Production-Ready Workflow

- End-to-End-Testing: Checking for any failings, incompatibility, and errors
- If required, may create blog post, tutorial, or references and screen cast.

[Final term evaluation date: 28 August]

4. About Me

- Name: **Adya Prasad**
- Email: adyaprasad.info@gmail.com, adyaprasad108@gmail.com
- My Django forum profile: https://forum.djangoproject.com/u/adyaprasad/ [member]
- My Portfolio**:** https://adya-prasad.github.io/portfolio/
- **My Articles:** https://adyaprasad.hashnode.dev/simplest-way-to-build-jenkins-cicd-pipeline-on-kubernetes, https://dev.to/adyaprasad/how-to-build-and-deploy-microservices-in-kubernetes-with-ingress-2007
- Country of residence: **India**
- Timezone: **Indian Standard Time (UTC + 5:30) [Asia/Kolkata]**
- Study: **VBSP University** and **Harvard CS50x**
- Languages: **English** and **Hindi**
- Primary Stack: **Python, Artificial Intelligence** and **DevOps**
- Profiles: LinkedIn profile, GitHub profile
- Work Experience: **Developer at NowTreat Healthcare, Freelance Python web developer, Contributor at Unsloth AI**

4.1. My Journey So Far in GSoC Django

I have been already associated with the Django community for a long time through social platforms but actively participating for the last 2 months through the Django forum and Discord server.

I started my Django GSoC journey by reading a lot of Django and GSoC official docs, contribution guidelines, COC, references, code rules, some GitHub gist and more (I mentioned my read resources in my research).
I learned a lot and loved talking with very cool and supportive Django community members and leaders, and I want to continue this journey and become a **Django fellow**.

I go through PRs that have been reviewed and contribution guidelines to understand the PR cycle, Django's specific processes and where we can make improvements.

4.2 My Past Contributions

- My proposal requires working on the local setup for PR testings created and updated GitHub actions files. I created dozens of varieties of PR (with ticket, without ticket, with docs and without docs, without tests, intentional failing tests, and more) locally for testing, hence not getting much time to contribute and merge real PR. But I work on some:
- Worked on ticket #36254 'Django Configurable Content-Type Parsing topic', but it got removed from the Django page. I learn about how Django parses contents through http and how `request.data` can enhance this processing. I learn more things about content-parsing; you can see that here in my research
- Learnt about automation in Django by working ticket #35380 locally (it was recommended to me in the Forum), I learned about triggering a command, generating screenshots, running selenium tests: `./runtests.py --selenium=firefox,chrome --headless --screenshots`, and more
- I worked on first ticket on #36244, which was invalid and it was my newbie mistake
- In summary, I didn't merged any PR yet but I can assure I have strong understanding of Django contribution workflows, PR lifecycle, and releases process.
- But I have contributed to other open source application and wrappers, like Unsloth AI and few more, you can check in my portfolio (mentioned above)

My profile has been promoted to "Trust Level: **Member**" [new user → basic user → member]

- Also,you can check out my **research** on current proposal idea: Automate Processes Within Django Contribution Workflow (see here)
- I also helps a few fellow GSoc contributors to provide relevant resources, docs and in solving setup errors in Django Community and Discord server that give me idea about Django community workflow

4.3. My Projects and Works Related to Automation:

Automated Contribution Management Dashboard	**Automated Friendly Bot: Yaar**	**Missing Word Prediction and Attention Visualizer with BERT**
Tech Stack: Django, YAML Configurations, GitHub API, GitHub Actions, Jenkinsfile A Dashboard that shows the status and ongoing contribution workflows of the repository using GitHub API and tokens. When a user enters his repo details, it will check and analyse its status and show all metrics and PR status and labels. Inspired by automate contribution workflow. View source code Check live project	*Tech Stack*: NLP, Django, JSON, Docker This was my Harvard CS50 last project, I built an adaptive AI Bot system that engages with students casually by mimicking real-life friends who are talking with messages and ready to help by giving automated responses. Backend integrated with context understanding with NLP, JSON and response pipeline. View project	*Tech Stack:* BERT, Transformer, Tokenizer, Encoding, Flask One way to create language models is to build a Masked Language Model, where a language model is trained to predict a "masked" word that is missing from a sequence of text. Uses BERT to predict masked words. The project also automatically generates diagrams on visualizing attention scores to understand and monitor prediction workflows View source code Check live project

5. Conclusion

As outlined in Django's GSoC 2025 project ideas, this proposal directly addresses the critical need to automate error-prone manual processes in Django's contribution workflow and making contributions faster.

Manual Process	***Automated Solutions***	***Positive Impact***
PR monitoring and management	Updated and new GitHub actions pipelines and script	• Reduced the PR merging time by 20-30% • Fasten the Django releases process surely more than by 10%
Releases preparation for different kind of releases and tracking the related components	Updated and more featureful Django project release app for all kind of releases	

Full code of `pr_review.yml`

```yaml
name: PR Review

on:
  pull_request:
    types: [opened, synchronize, reopened]
    paths-ignore:
      - 'docs/releases/**'
  workflow_dispatch:

concurrency:
  group: "${{ github.workflow }}-${{ github.ref }}"
  cancel-in-progress: true

permissions:
  contents: read
  pull-requests: write

jobs:
  # Tried to reuse CI results instead of duplicating checks
  code-ci-status-check:
    runs-on: ubuntu-latest
    outputs:
      ci_status: ${{ steps.check-ci.outputs.status }}
    continue-on-error: true
    steps:
      - name: Check CI Status
        id: check-ci
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const pull = await github.rest.pulls.get({
              owner, repo, pull_number: number
            });
            const sha = pull.data.head.sha;
            const checkRuns = await github.rest.checks.listForRef({
              owner, repo, ref: sha
            });
            const relevantWorkflows = checkRuns.data.check_runs.filter(
              run =>
                (run.name.includes('Tests') ||
                 run.name.includes('Linters') ||
                 run.name.includes('flake8') ||
                 run.name.includes('isort') ||
                 run.name.includes('black') ||
                 run.name.includes('JavaScript tests')) &&
                run.status === 'completed'
            );
            if (relevantWorkflows.length === 0) {
              core.setOutput('status', 'pending');
              return { status: 'pending' };
            }
            const allPassing = relevantWorkflows.every(
              run => run.conclusion === 'success'
            );
            const status = allPassing ? 'success' : 'failure';
            core.setOutput('status', status);
```

```
          return { status };

  # Test coverage check
  test-coverage:
    runs-on: ubuntu-latest
    outputs:
      has_tests: ${{ steps.check-tests.outputs.has_tests }}
    continue-on-error: true
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Check for test files
        id: check-tests
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const files = await github.rest.pulls.listFiles({
              owner, repo, pull_number: number
            });
            const pythonFiles = files.data.filter(
              file => file.filename.endsWith('.py')
            ).filter(
              file =>
                !file.filename.includes('/tests/') &&
                !file.filename.includes('test_')
            );
            const testFiles = files.data.filter(
              file => file.filename.endsWith('.py')
            ).filter(
              file =>
                file.filename.includes('/tests/') ||
                file.filename.includes('test_')
            );
            const hasTests = testFiles.length > 0 ||
                             pythonFiles.length === 0;
            core.setOutput('has_tests', hasTests ? 'true' : 'false');
            if (pythonFiles.length > 0 && !hasTests) {
              await github.rest.issues.createComment({
                owner, repo, issue_number: number,
                body: "⚠️ **Test Coverage Check**\n" +
                      "This PR modifies Python code but doesn't appear " +
                      "to include any test files. Please consider adding tests " +
                      "to cover your changes."
              });
            }
            return { has_tests: hasTests };

  # Documentation checks - combined into a single job for efficiency
  documentation-checks:
    runs-on: ubuntu-latest
    outputs:
      has_version_notes: ${{ steps.check-version-notes.outputs.has_version_notes }}
      doc_checks_valid: ${{ steps.check-doc-format.outputs.all_valid }}
    continue-on-error: true  # Allow this job to fail without stopping the workflow
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.13"
      - name: Check for version notes
        id: check-version-notes
        uses: actions/github-script@v7
        with:
          script: |
            const fs = require("fs");
            const { owner, repo, number } = context.issue;
            const files = await github.rest.pulls.listFiles({
              owner, repo, pull_number: number
            });
            const djangoFiles = files.data.filter(
              file => file.filename.endsWith(".py")
            ).filter(
              file =>
                file.filename.startsWith("django/") &&
                !file.filename.includes("tests/")
            );
            let needsVersionNotes = djangoFiles.length > 0;
            let hasVersionNotes = false;
            if (needsVersionNotes) {
              for (const file of djangoFiles.concat(
```

```
            files.data.filter(
              file =>
                file.filename.endsWith(".txt") ||
                file.filename.endsWith(".rst")
            ).filter(
              file => file.filename.startsWith("docs/")
            )
          )) {
            try {
              if (fs.existsSync(file.filename)) {
                const fileContent = fs.readFileSync(
                  file.filename, "utf8"
                );
                if (
                  fileContent.includes(".. versionadded::") ||
                  fileContent.includes(".. versionchanged::")
                ) {
                  hasVersionNotes = true;
                  break;
                }
              }
            } catch (error) {
              console.error(`Error reading file ${file.filename}: ${error}`);
            }
          }
        }
        core.setOutput("has_version_notes",
          hasVersionNotes ? "true" : "false");
        if (needsVersionNotes && !hasVersionNotes) {
          await github.rest.issues.createComment({
            owner, repo, issue_number: number,
            body: "⚠️ **Version Notes Check**\n" +
                  "This PR modifies Django core files but doesn't appear " +
                  "to include version notes (`.. versionadded::` or " +
                  "`.. versionchanged::`). Please add appropriate version " +
                  "notes in docstrings or documentation files."
          });
        }
        return { has_version_notes: hasVersionNotes };

    - name: Check documentation formatting and deprecation
      id: check-doc-format
      uses: actions/github-script@v7
      with:
        script: |
          const fs = require("fs");
          const { owner, repo, number } = context.issue;
          const files = await github.rest.pulls.listFiles({
            owner, repo, pull_number: number
          });
          const docFiles = files.data.filter(
            file =>
              (file.filename.endsWith(".txt") ||
               file.filename.endsWith(".rst")) &&
              file.filename.startsWith("docs/")
          );
          let lineLengthValid = true;
          let invalidFiles = [];
          for (const file of docFiles) {
            try {
              if (fs.existsSync(file.filename)) {
                const fileContent = fs.readFileSync(
                  file.filename, "utf8"
                );
                const lines = fileContent.split("\n");
                const longLines = lines.map((line, idx) =>
                  line.length > 80 ? idx + 1 : null
                ).filter(idx => idx !== null);
                if (longLines.length > 0) {
                  lineLengthValid = false;
                  invalidFiles.push(
                    `${file.filename}: Lines ${longLines.join(", ")}`
                  );
                }
              }
            } catch (error) {
              console.error(`Error reading file ${file.filename}: ${error}`);
            }
          }
          const pythonFiles = files.data.filter(
            file => file.filename.endsWith(".py")
          ).filter(
            file => file.filename.startsWith("django/")
          );
          let deprecationFound = false;
          let deprecationValid = true;
          let deprecationData = { documented: true, tested: true };
```

```
            for (const file of pythonFiles) {
              try {
                if (fs.existsSync(file.filename)) {
                  const fileContent = fs.readFileSync(
                    file.filename, "utf8"
                  );
                  if (
                    fileContent.includes("RemovedInDjango") ||
                    fileContent.includes("warnings.warn(")
                  ) {
                    deprecationFound = true;
                    deprecationData.documented =
                      fileContent.includes(".. deprecated::");
                    break;
                  }
                }
              } catch (error) {
                console.error(`Error reading file ${file.filename}: ${error}`);
              }
            }
            if (deprecationFound) {
              const testFiles = files.data.filter(
                file => file.filename.endsWith(".py")
              ).filter(
                file => file.filename.includes("/tests/")
              );
              deprecationData.tested = testFiles.some(file => {
                try {
                  if (fs.existsSync(file.filename)) {
                    const fileContent = fs.readFileSync(
                      file.filename, "utf8"
                    );
                    return fileContent.includes("test_deprecation") ||
                           fileContent.includes("assertWarns(");
                  }
                  return false;
                } catch (error) {
                  console.error(`Error reading file ${file.filename}: ${error}`);
                  return false;
                }
              });
              deprecationValid =
                deprecationData.documented && deprecationData.tested;
            }
            const allValid = lineLengthValid && deprecationValid;
            core.setOutput('line_length_valid', lineLengthValid ? 'true' : 'false');
            core.setOutput('deprecation_valid', deprecationValid ? 'true' : 'false');
            core.setOutput('invalid_files', JSON.stringify(invalidFiles));
            core.setOutput('deprecation_data', JSON.stringify(deprecationData));
            core.setOutput('all_valid', allValid ? 'true' : 'false');
            return {
              line_length_valid: lineLengthValid,
              deprecation_valid: deprecationValid,
              invalid_files: invalidFiles,
              deprecation_data: deprecationData,
              all_valid: allValid
            };

      - name: Summarize documentation checks
        id: summarize-doc-checks
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const lineLengthValid = ${{ steps.check-doc-format.outputs.line_length_valid }} ===
"true";
            const deprecationValid = ${{ steps.check-doc-format.outputs.deprecation_valid }} ===
"true";
            const invalidFiles = JSON.parse(${{ steps.check-doc-format.outputs.invalid_files }});
            const deprecationData = JSON.parse(${{ steps.check-doc-format.outputs.deprecation_data
}});
            const allValid = ${{ steps.check-doc-format.outputs.all_valid }} === "true";

            // Set the output for the job
            core.setOutput("all_valid", allValid ? "true" : "false");

            // Post comments if there are issues
            if (!lineLengthValid && invalidFiles.length > 0) {
              await github.rest.issues.createComment({
                owner, repo, issue_number: number,
                body: `⚠️ **Documentation Line Length Warning**\nThe following documentation files
contain lines longer than 80 characters:\n${invalidFiles.join("\n")}\n\nPlease ensure all
documentation lines are at most 80 characters long.`
              });
            }

            if (!deprecationValid) {
```

```
              let message = `⚠️ **Deprecation Warning**\nThis PR appears to include code that raises
deprecation warnings.`;

              if (!deprecationData.documented) {
                message += `\n- Please ensure deprecations are documented with \`.. deprecated::\` in
the docstring.`;
              }

              if (!deprecationData.tested) {
                message += `\n- Please ensure deprecation warnings are tested with appropriate test
cases.`;
              }

              await github.rest.issues.createComment({
                owner, repo, issue_number: number,
                body: message
              });
            }

  # Check for release notes if needed
  release-notes:
    runs-on: ubuntu-latest
    outputs:
      has_release_notes: ${{ steps.check-release-notes.outputs.has_release_notes }}
    continue-on-error: true  # Allow this job to fail without stopping the workflow
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Check for release notes
        id: check-release-notes
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const [files, prData] = await Promise.all([
              github.rest.pulls.listFiles({
                owner, repo, pull_number: number
              }),
              github.rest.pulls.get({
                owner, repo, pull_number: number
              })
            ]);
            const significantChanges = files.data.filter(
              file => file.filename.endsWith(".py")
            ).filter(
              file =>
                file.filename.startsWith("django/") &&
                !file.filename.includes("tests/") &&
                !file.filename.includes("django/utils/") &&
                !file.filename.includes("django/conf/")
            );
            const prTitle = prData.data.title.toLowerCase();
            const isFeature = prTitle.includes("feature") ||
                              prTitle.includes("add") ||
                              prTitle.includes("new") ||
                              prTitle.includes("implement");
            const isBugfix = prTitle.includes("fix") ||
                             prTitle.includes("bug") ||
                             prTitle.includes("issue") ||
                             prTitle.includes("solve");
            const releaseNotesModified = files.data.some(
              file =>
                file.filename.startsWith("docs/releases/") ||
                file.filename.includes("release_notes")
            );
            const needsReleaseNotes =
              (isFeature || isBugfix) &&
              significantChanges.length > 0;
            const hasReleaseNotes = releaseNotesModified;
            core.setOutput("has_release_notes",
              hasReleaseNotes ? "true" : "false");
            if (needsReleaseNotes && !hasReleaseNotes) {
              await github.rest.issues.createComment({
                owner, repo, issue_number: number,
                body: "⚠️ **Release Notes Reminder**\n" +
                      "This PR appears to include significant changes that " +
                      "should be documented in the release notes. Please consider " +
                      "adding appropriate entries to `docs/releases/`."
              });
            }
            return { has_release_notes: hasReleaseNotes };

  # PR summary and label management
  pr-review-summary:
    runs-on: ubuntu-latest
```

```
    needs:
      - ci-status-check
      - test-coverage
      - documentation-checks
      - release-notes
    if: always()
    steps:
      - name: Generate PR Review Summary
        uses: actions/github-script@v7
        with:
          script: |
            const { owner, repo, number } = context.issue;
            const pull = await github.rest.pulls.get({
              owner, repo, pull_number: number
            });
            const ciStatus = "${{ needs.ci-status-check.outputs.ci_status }}";
            const hasTests = "${{ needs.test-coverage.outputs.has_tests }}";
            const hasVersionNotes = "${{ needs.documentation-checks.outputs.has_version_notes }}";
            const docChecksValid = "${{ needs.documentation-checks.outputs.doc_checks_valid }}";
            const hasReleaseNotes = "${{ needs.release-notes.outputs.has_release_notes }}";
            const allChecksPass =
              (ciStatus === "success" || ciStatus === "pending") &&
              hasTests === "true" &&
              (hasVersionNotes === "true" || hasVersionNotes === "") &&
              (docChecksValid === "true" || docChecksValid === "") &&
              (hasReleaseNotes === "true" || hasReleaseNotes === "");
            let summary = "## Here is Your PR Summary\n\n";
            summary += "**PR Title**: " + pull.data.title + "\n";
            summary += "**Author**: " + pull.data.user.login + "\n\n";
            summary += "### Automated Checks\n\n";
            summary += "- CI Status: " +
              (ciStatus === "success"
                ? "✅ Passing"
                : (ciStatus === "pending"
                  ? "⏳ In progress"
                  : "❌ Failing")) + "\n";
            summary += "- Tests Included: " +
              (hasTests === "true" ? "✅ Yes" : "❌ No") + "\n";
            summary += "- Version Notes: " +
              (hasVersionNotes === "true"
                ? "✅ Included"
                : (hasVersionNotes === ""
                  ? "❌ Problem detected, may be docs.yml failed!"
                  : "❌ Missing")) + "\n";
            summary += "- Documentation Format: " +
              (docChecksValid === "true"
                ? "✅ Valid"
                : (docChecksValid === ""
                  ? "➖ N/A"
                  : "❌ Invalid")) + "\n";
            summary += "- Release Notes: " +
              (hasReleaseNotes === "true"
                ? "✅ Included"
                : (hasReleaseNotes === ""
                  ? "➖ N/A"
                  : "❌ Missing (may be not required)")) + "\n\n";
            if (allChecksPass) {
              summary += "### Verdict\n\n" +
                "✅ **All checks passed. This PR is ready for review request!**\n";
              await github.rest.issues.addLabels({
                owner, repo, issue_number: number,
                labels: ["ready for review"]
              });
            } else {
              summary += "### Result\n\n" +
                "⚠️ **Some checks failed. Please address the issues " +
                "to get ready for review request.**\n";
            }
            await github.rest.issues.createComment({
              owner, repo, issue_number: number,
              body: summary
            });
            core.summary.addRaw(summary).write();
```

Here is the full updated release app `admin.py`:

```
from django.contrib import admin
from django.http import HttpResponse
from django.template.loader import render_to_string

from .models import (
    FeatureRelease,
    PreRelease,
    Release,
    Releaser,
```

```
    SecurityIssue,
    SecurityIssueReleasesThrough,
    SecurityRelease,
)


def render_checklist(request, queryset):
    assert len(queryset) == 1, "A single item should be selected"
    [instance] = queryset
    context = {
        "instance": instance,
        "releaser": instance.releaser,
        "slug": instance.slug,
        "version": instance.version,
        "title": instance.__class__.__name__,
        **instance.__dict__,
    }
    if (release := getattr(instance, "release", None)) is not None:
        context["release"] = release
    if (data := getattr(instance, "get_context_data", None)) is not None:
        context.update(data)
    checklist = render_to_string(instance.checklist_template, context, request=request)
    return HttpResponse(checklist, content_type="text/markdown")


@admin.register(Release)
class ReleaseAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {"fields": ["version", "is_active", "is_lts"]}),
        ("Dates", {"fields": ["date", "eol_date"]}),
        ("Artifacts", {"fields": ["tarball", "wheel", "checksum"]}),
    ]
    list_display = (
        "version",
        "show_is_published",
        "is_lts",
        "date",
        "eol_date",
        "major",
        "minor",
        "micro",
        "show_status",
        "iteration",
    )
    list_filter = ("status", "is_lts", "is_active")
    ordering = ("-major", "-minor", "-micro", "-status", "-iteration")

    def get_form(self, request, obj=None, change=False, **kwargs):
        form_class = super().get_form(request, obj=obj, change=change, **kwargs)
        extensions = {"tarball": ".tar.gz", "wheel": ".whl", "checksum": ".asc,.txt"}
        for field, accept in extensions.items():
            widget = form_class.base_fields[field].widget
            widget.attrs.setdefault("accept", accept)
        return form_class

    @admin.display(
        description="status",
        ordering="status",
    )
    def show_status(self, obj):
        return obj.get_status_display()

    @admin.display(
        boolean=True,
        description="Published?",
        ordering="is_active",
    )
    def show_is_published(self, obj):
        return obj.is_published

    # Adding list_display and ordering from admin2.py, if needed.
    # If the admin1.py has all needed, then it is not required.
    # list_display = ["version", "date", "is_lts"]
    # ordering = ["-version"]

class ReleaserAdmin(admin.ModelAdmin):
    list_display = ["user", "key_id", "key_url"]

class ReleaseChecklistAdminMixin:
    list_display = ["version", "when", "releaser"]
    list_filter = ["releaser"]
    actions = ["render_checklist"]
    readonly_fields = ["blogpost_link"]

    def queryset(self, request):
        return super().get_queryset(request).select_related("release")
```

```
    @admin.action(description="Render checklists for selected releases")
    def render_checklist(self, request, queryset):
        return render_checklist(request, queryset)

class PreReleaseAdmin(ReleaseChecklistAdminMixin, admin.ModelAdmin):
    list_display = ["feature_release"] + ReleaseChecklistAdminMixin.list_display
    list_filter = ["feature_release"] + ReleaseChecklistAdminMixin.list_filter

class FeatureReleaseAdmin(ReleaseChecklistAdminMixin, admin.ModelAdmin):
    list_display = ReleaseChecklistAdminMixin.list_display + ["tagline"]

class SecurityReleaseAdmin(ReleaseChecklistAdminMixin, admin.ModelAdmin):
    list_display = ["versions", "when", "releaser"]
    search_fields = ["affected_branches"]
    ordering = ["-when"]
    readonly_fields = ["hashes_by_versions"]

class SecurityIssueAdmin(admin.ModelAdmin):
    list_display = ["cve_year_number", "summary", "severity", "commit_hash_main"]
    list_filter = ["severity", "release"]
    search_fields = ["cve_year_number", "summary", "description", "commit_hash_main"]
    ordering = ["-cve_year_number"]
    readonly_fields = ["hashes_by_branch", "releases"]

class SecurityIssueReleasesThroughAdmin(admin.ModelAdmin):
    list_display = ["security_issue_cve_year_number", "release_version", "commit_hash"]
    list_filter = ["release__version"]
    search_fields = [
        "security_issue__cve_year_number",  # Note: using the correct field name from the model.
        "release__version",
        "commit_hash",
    ]
    ordering = ["-security_issue__cve_year_number", "release__version"]

    @admin.display(
        description="CVE Year Number",
        ordering="security_issue__cve_year_number",
    )
    def security_issue_cve_year_number(self, obj):
        return obj.security_issue.cve_year_number if obj.security_issue else None

    @admin.display(
        description="Release Version",
        ordering="release__version",
    )
    def release_version(self, obj):
        return obj.release.version if obj.release else None


# Ensure each model is registered
admin.site.register(Releaser, ReleaserAdmin)
admin.site.register(FeatureRelease, FeatureReleaseAdmin)
admin.site.register(PreRelease, PreReleaseAdmin)
admin.site.register(SecurityRelease, SecurityReleaseAdmin)
admin.site.register(SecurityIssue, SecurityIssueAdmin)
admin.site.register(SecurityIssueReleasesThrough, SecurityIssueReleasesThroughAdmin)
```