GSoC 2025 Proposal - Bring django-template-partials into core

proposal.md

GSoC 2025 Proposal - Bring django-template-partials into core

Project Information

- **Project Title:** Django Templates: Bring django-template-partials into core
- **Mentor:** Carlton Gibson
- **Difficulty:** Medium
- **Size:** 350 hours

Project Overview

Problem Statement

Django's template system currently lacks built-in support for reusable named inline partials. While Django provides template inheritance, includes, and template tags for code reuse, there is no native way to define and reuse template fragments within the same template multiple times. The third-party package django-template-partials fills this gap, but integrating it into Django core would make this functionality available to all Django users without requiring additional dependencies.

Modern web development technoloies, especially with libraries like HTMX, benefit greatly from the ability to render specific template fragments and keep related markup together. This improves the "Locality of Behavior" principle, where related UI components and their behavior are defined close to each other.

Proposed Solution

Design

I propose adding the functionality of `django-template-partials` directly into Django core. This would include native support for the `{% partialdef %}` tag, which allows developers to define reusable template fragments, and the `{% partial %}` tag to include those fragments wherever needed.

Additionally, I suggest implementing a partial template loader that supports the `#partial-name` syntax, making it easier to access specific partials within templates. The solution would also offer inline rendering support, allowing partials to be rendered immediately at their definition point when required. To enhance template organization, I recommend adding support for `{% endpartialdef partial_name %}`, which will improve readability. Along with these features, a documentation would be provided, offering clear instructions and examples to guide users. Finally, a migration path will be included to help those already using the django-template-partials library transition smoothly to the new core functionality.

Benefits

From my experience with Django templates, I believe this feature will bring several advantages. First, it will help create cleaner templates by eliminating the need to repeat the same code in different parts of a template. Instead, we can define it once and use it everywhere. This feature also brings a modern component-based approach to Django templates, which aligns with how most developers think these days, without needing a JavaScript framework. For those who work with HTMX and Django, this feature will improve integration, making it easier to define and target specific parts of a page. It will also keep related code together, so developers won't have to jump between files to understand how a page works. Additionally, it will reduce the need for many small template files, keeping the templates folder cleaner and more manageable. By removing one more dependency, deployment will be simpler, and there will be fewer potential version conflicts.

Implementation Plan

This proposed details from my understanding to the problem. It may be need edits and improvements to ensure alignment with the project's objectives.

Addressing Issues

get_nodes_by_type() Compatibility

The issue with template inheritance in the commit 71128aa will be addressed by properly implementing the `get_nodes_by_type()` method on `TemplateProxy`. I plan to ensures that the TemplateProxy correctly participates in the node traversal process used by template inheritance and other template features.

Template Rendering Signal

To fix the issue #54 with template rendering signals, I'll ensure that when a partial is rendered, the appropriate signals are fired:

Template Locality

We'll maintain the principle of template locality by ensuring partials are stored within the template's context and not in a global registry. This design ensures that partials are scoped to their defining template, maintaining locality of behavior.

Testing

Testing will be an important part of this project to ensure everything works correctly. I will implement two main types of tests: **unit tests** to verify individual components, function, and methods works correctly and **integration tests** to check how different parts work together. I will use popular testing tools that match the project's technology stack and maintain a good test coverage. All tests will be well-documented and easy to run, making it simple for other developers to understand and contribute to the project.

Documentation

Documentation will be a key component of this project to ensure long-term usability and maintainability. Good documentation helps both users and developers understand, use, and extend the project. Throughout the development process, I will maintain clear and structured documentation, including setup guides, usage instructions, and contribution guidelines, following the project's standards. It will include code snippets, diagrams, and examples where helpful.

Project Timeline

Community Bonding Period (May 8 - June 1, 2025)

- Deep dive into Django's template system internals
- Analyze django-template-partials codebase thoroughly
- Meet with mentor Carlton Gibson to align on project goals
- Set up development environment and tools
- Create detailed implementation architecture document
- Draft implementation roadmap with deliverables
- **Deliverable:** Architecture document and implementation plan approved by mentor

Phase 1: Core Implementation (June 2 - June 29, 2025)

Week 1-2 (June 2 - June 15)

- Implement basic template tags (`partialdef`, `partial`)
- Create initial implementation of `TemplateProxy`
- Begin integration with Django's template engine
- Handle template inheritance compatibility
- **Deliverable**: First PR draft opened for early feedback

Week 3-4 (June 16 - June 29)

- Implement `get_nodes_by_type` support for template inheritance
- Add named end tag support (`{% endpartialdef partial_name %}`)
- Address inline partial rendering challenges
- Refine implementation based on mentor feedback
- **Deliverable**: Core tag implementation completed

Phase 2: Secondary Features (June 30 - July 13, 2025)

Week 5-6 (June 30 - July 13)

- Implement template loader for `#` syntax
- Add configuration support in template engine
- Implement default settings in `global_settings.py`
- Add signal handling for template rendering (fixing issue #54)
- Complete integration with Django's template engine
- **Deliverable**: Feature-complete implementation ready for midterm

Midterm Evaluation (July 14 - July 18, 2025)

- Submit comprehensive progress report
- Review implementation with mentor
- Prepare code for evaluation
- Adjust timeline if necessary based on feedback
- **Milestone**: Mentor submits evaluation by July 18, 18:00 UTC

Phase 3: Testing & Optimization (July 19 - August 3, 2025)

Week 7-8 (July 19 - August 3)

- Develop unit test for template tags and loader
- Create integration tests focusing on Django compatibility

- Implement edge case and error handling tests
- Optimize code based on test results
- **Deliverable:** Comprehensive test suite

Phase 4: Documentation & Refinement (August 4 - August 24, 2025)

Week 9-10 (August 4 - August 17)

- Update key Django documentation files
- Create migration guide from django-template-partials
- Draft release notes for Django 5.1
- Add usage examples with HTMX
- **Deliverable:** Complete documentation package

Week 11 (August 18 - August 24)

- Address feedback from Django core developers
- Prepare final PR for submission
- **Deliverable:** PR ready for merge review

Final Evaluation (August 25 - September 8, 2025)

- Submit final work product by August 25, 18:00 UTC
- Document accomplishments and future work recommendations
- Complete final evaluation
- **Milestone:** Mentor submits final evaluation by September 8, 18:00 UTC

About Me

Background

I am a Senior Computer Science Student and Django Backend Developer with a passion for open source contribution and web development. I have hands-on experience building Django applications and contributing to the Django via multiple Pull Requests.

As a Backend Django Developer at Egypt Metro Platform, I implemented a Django backend metro management system which includes API, business logic, and metro administration (Admin Panel) integrating with frontend flutter and AI model. You can view the live application here. Additionally, I have completed a **3 Summer Training** at the Information Technology Institute, mastering **full-stack web development technologies** such as **HTML, CSS, Bootstarp, JavaScript, React, UI/UX, Python, Django, PostgreSQL, RESTful APIs,** and more. This experience has equipped me with skills in both backend and frontend development.

Contact Information

- Name: Ahmed Nassar
- GitHub: AhmedNassar7
- LinkedIn: nasssar
- Portfolio: Live
- Gmail: a.moh.nassar00@gmail.com
- Django Forum: AhmedNassar7
- Discord: nassar15
- Location: Cairo, Egypt
- Timezone: (UTC+2) Eastern European Time (EET)
- Preferred Language: English