Contributing to the Global Governance Frameworks Website

Welcome to the technical contribution guide for the Global Governance Frameworks website! This guide will help you understand our codebase, development workflow, and how to make meaningful contributions to the project.

🎯 Project Overview

The Global Governance Framework website is built with modern web technologies and serves as a platform for sharing governance frameworks, implementation tools, and fostering global collaboration. Our mission is to create tools, patterns, and frameworks that enable different governance systems to interoperate and evolve together.

Core Technologies

- **Frontend**: SvelteKit (latest)
- **Styling**: Tailwind CSS with custom design system
- **Internationalization**: Custom i18n implementation supporting English and Swedish
- **Form Handling**: Formspree integration
- **Content Management**: Markdown-based content with dynamic routing
- **Build Tool**: Vite
- **Package Manager**: npm

🚀 Getting Started

Prerequisites

- Node.js (v16 or higher)
- npm (comes with Node.js)
- Git
- A code editor (VS Code recommended)

Quick Setup

1. **Clone the repository:**

```
git clone git@github.com:GlobalGovernanceFramework/governance-framework-site.git
cd governance-framework-site
```

2. **Install dependencies:**

```
npm install
```

3. **Start the development server:**

```
npm run dev -- --open
```

4. **View the site:** Open your browser to `http://localhost:5173`

⚠️ Important: Tailwind CSS Configuration Issues

Current Status: Our Tailwind CSS setup appears to have configuration conflicts and may not be working optimally. We have two conflicting config files (`tailwind.config.js` and

`tailwind.config.ts`) and the codebase shows heavy reliance on inline styles instead of Tailwind utilities.

Evidence of Issues:

- Duplicate Tailwind config files with different approaches
- Heavy use of inline `style` attributes instead of Tailwind classes
- Inconsistent color system implementation
- Missing proper Tailwind v4 configuration

If You Want to Help Fix This: This would be a **high-impact contribution**! The ideal solution would be:

1. **Consolidate to a single config file** (preferably TypeScript)
2. **Implement Tailwind v4 properly** with our design system
3. **Convert inline styles to Tailwind utilities** throughout the codebase
4. **Set up proper CSS custom properties** for our color palette
5. **Ensure all Tailwind plugins work correctly**

Current Workaround: If you encounter Tailwind issues during development:

```
# Try this fix first
npm uninstall tailwindcss postcss autoprefixer @tailwindcss/typography @tailwindcss/f
npm install -D tailwindcss postcss autoprefixer @tailwindcss/typography @tailwindcss/
npx tailwindcss init -p
```

For New Contributors:

- You can still contribute effectively using inline styles (as seen in existing code)
- Follow the color palette defined in the configs
- We'll migrate to proper Tailwind utilities once the configuration is fixed

Want to Tackle This Challenge? If you have experience with Tailwind CSS v4 configuration and would like to help modernize our styling approach, please:

- Join our **#dev-design** channel on Discord (under 🤝 WORKSPACES category) to discuss the approach
- Open a GitHub issue to coordinate with other contributors
- Share your ideas and get feedback before starting

This would significantly improve our development experience!

📁 Project Structure

Understanding our project structure is crucial for effective contribution:

```
src/
├── lib/
│   ├── components/          # Reusable Svelte components
│   │   ├── Header.svelte   # Main navigation
│   │   ├── Footer.svelte   # Site footer
│   │   └── ...             # Other UI components
│   ├── content/            # Markdown content files
│   │   ├── frameworks/     # Framework documentation
│   │   ├── ggf-os/        # GGF Operating System docs
│   │   └── translations/   # Translation guides
│   ├── i18n/              # Internationalization
```

```
│   │   ├── en/            # English translations
│   │   ├── sv/            # Swedish translations
│   │   └── index.js       # i18n system core
│   ├── posts/             # Blog posts
│   └── utils/             # Utility functions
├── routes/                # SvelteKit pages
│   ├── frameworks/        # Framework pages
│   ├── blog/             # Blog functionality
│   └── ...               # Other pages
└── app.html              # HTML template
```

Key Directories Explained

- `src/lib/components/` : Reusable UI components used throughout the site
- `src/lib/content/` : All markdown content, organized by type and language
- `src/lib/i18n/` : Translation files and internationalization logic
- `src/routes/` : SvelteKit's file-based routing system
- `static/` : Static assets like images, PDFs, and downloads

🛠️ Development Workflow

1. Setting Up Your Development Environment

Create a new branch for your feature:

```
git checkout -b feature/your-feature-name
```

2. Making Changes

Adding New Pages

To add a new page, create a `+page.svelte` file in the appropriate `src/routes/` directory:

```
<!-- src/routes/your-new-page/+page.svelte -->
<script>
  import { t } from '$lib/i18n';
  // Your page logic
</script>

<svelte:head>
  <title>Your Page Title</title>
</svelte:head>

<div class="container mx-auto px-4 py-8">
  <h1 class="text-3xl font-bold mb-6">{$t('yourPage.title')}</h1>
  <!-- Your content -->
</div>
```

Adding Components

Create reusable components in `src/lib/components/` :

```
<!-- src/lib/components/YourComponent.svelte -->
<script>
  export let title = '';
```

```
  export let description = '';
</script>

<div class="component-container">
  <h2 class="text-xl font-semibold">{title}</h2>
  <p class="text-gray-600">{description}</p>
</div>

<style>
  .component-container {
    /* Your styles using Tailwind classes */
    background-color: white;
    border-radius: 0.5rem;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    padding: 1.5rem;
  }
</style>
```

Working with Markdown Content

Add new framework content in `src/lib/content/frameworks/`:

```
<!-- src/lib/content/frameworks/en/your-framework.md -->
# Your Framework Title

## Introduction

Your framework introduction...

## Core Principles

1. **Principle One**: Description
2. **Principle Two**: Description

## Implementation

Implementation guidelines...
```

3. Internationalization

Adding Translations

Add translation keys to the appropriate JSON files:

```
// src/lib/i18n/en/common.json
{
  "header": {
    "yourNewItem": "Your New Item"
  }
}
```

```
// src/lib/i18n/sv/common.json
{
  "header": {
    "yourNewItem": "Ditt nya objekt"
```

```
  }
}
```

Using Translations in Components

```
<script>
  import { t } from '$lib/i18n';
</script>

<h1>{$t('common.header.yourNewItem')}</h1>
```

4. Styling Guidelines

⚠️ **Current State**: Our styling system has some configuration issues that present both challenges and opportunities:

Current Approach

Due to Tailwind configuration conflicts, much of our current codebase uses inline styles. While not ideal, this approach works and maintains consistency.

Color Palette (Use these values in styles)

```
:root {
  --primary-blue: #2B4B8C;
  --secondary-purple: #6B5CA5;
  --earthy-green: #2D5F2D;
  --warm-gold: #DAA520;
  --dark-gold: #B8860B;
}
```

Current Component Styling Pattern

```
<!-- Current approach used throughout the codebase -->
<div style="background-color: #2B4B8C; color: white; padding: 1rem; border-radius: 0
  Content
</div>

<!-- Goal: Convert to Tailwind utilities (help needed!) -->
<div style="background-color: #1e293b; color: white; padding: 1rem; border-radius: 0
  Content
</div>
```

Styling Guidelines for Contributors

For Quick Contributions:

- Follow existing inline style patterns
- Use our color palette consistently
- Maintain responsive design with CSS media queries

For Tailwind Enthusiasts:

- Help us fix the Tailwind configuration (high-impact contribution!)
- Convert inline styles to proper Tailwind utilities
- Implement our design system as Tailwind custom classes

Responsive Design Pattern

```
<div style="
  display: grid;
  grid-template-columns: repeat(1, minmax(0, 1fr));
  gap: 2rem;
">
  <!-- Mobile: single column -->
</div>

<style>
  @media (min-width: 768px) {
    div {
      grid-template-columns: repeat(2, minmax(0, 1fr));
    }
  }
</style>
```

5. Testing Your Changes

Local Testing

```
# Run development server
npm run dev

# Build for production (test)
npm run build

# Preview production build
npm run preview
```

Content Validation

- Ensure all links work correctly
- Verify translations are complete
- Test on multiple screen sizes
- Check accessibility with screen readers

🎨 Design System

Typography Scale

- **Headings**: Use semantic heading tags with appropriate styling
- **Body**: Use readable font sizes for main content
- **Small text**: Use smaller sizes for captions and metadata

Spacing Guidelines

- **Sections**: Use consistent vertical spacing between sections
- **Components**: Use consistent internal padding
- **Elements**: Use consistent vertical rhythm

Layout Patterns

- **Container**: Maximum width with auto margins and padding

- **Grid**: Use CSS Grid or flexbox for layouts
- **Cards**: Consistent shadow and border-radius patterns

📝 Content Guidelines

Markdown Best Practices

- Use semantic heading hierarchy (H1 → H2 → H3)
- Include clear section breaks
- Add descriptive alt text for images
- Use consistent formatting for code blocks

Framework Documentation Structure

```
# Framework Title

## Executive Summary
Brief overview and key benefits

## Core Principles
Fundamental principles with explanations

## Implementation Guidelines
Step-by-step implementation process

## Tools and Resources
Links to relevant tools and materials

## Case Studies
Real-world examples and success stories

## Appendices
Additional technical details and references
```

🔧 Advanced Development

Adding New Framework Visualizations

1. Create SVG assets in `static/frameworks/your-framework/`
2. Add interactive components in `src/routes/frameworks/visuals/your-framework/`
3. Use D3.js or other visualization libraries as needed

Custom Components with Interactivity

```
<!-- Example interactive component -->
<script>
  import { onMount } from 'svelte';
  import { browser } from '$app/environment';

  let data = [];
  let loading = true;

  onMount(async () => {
    if (browser) {
```

```
      // Load data and initialize component
      loading = false;
    }
  });
</script>

{#if loading}
  <div class="loading-spinner">Loading...</div>
{:else}
  <!-- Your interactive content -->
{/if}
```

Working with Static Assets

- **Images**: Place in `static/images/`
- **Downloads**: Place in `static/downloads/`
- **Framework tools**: Place in `static/frameworks/tools/`

🤝 Contribution Process

1. Planning Your Contribution

Before starting:

- Check existing issues and discussions
- Propose new features in GitHub issues
- Coordinate with maintainers for large changes

2. Development Checklist

- ☐ Code follows project conventions
- ☐ All translations are complete
- ☐ Responsive design works on all devices
- ☐ Accessibility standards are met
- ☐ Content is well-structured and clear
- ☐ Links and navigation work correctly
- ☐ No console errors or warnings

3. Submitting Your Contribution

```
# Ensure your branch is up to date
git checkout main
git pull origin main
git checkout your-branch
git merge main

# Push your changes
git push origin your-branch

# Create a pull request on GitHub
```

Pull Request Guidelines

- **Title**: Clear, descriptive title

- **Description**: Explain what changes were made and why
- **Screenshots**: Include before/after screenshots for UI changes
- **Testing**: Describe how you tested your changes

📚 Resources

Documentation

- SvelteKit Documentation
- Tailwind CSS Documentation
- Our Style Guide

Development Tools

- Svelte Extension for VS Code
- Tailwind CSS IntelliSense

Community

- GitHub Discussions
- Discord Community - Join **#dev-design** for technical discussions

🔐 Security and Best Practices

Security Guidelines

- Never commit sensitive data (API keys, passwords)
- Validate all user inputs
- Follow OWASP security guidelines
- Use environment variables for configuration

Performance Best Practices

- Optimize images before adding them
- Use lazy loading for heavy components
- Minimize bundle size with tree-shaking
- Follow SvelteKit's performance recommendations

🌱 What to Contribute

High-Impact Areas

1. 🎨 **Tailwind CSS Configuration Fix ⭐ MOST NEEDED ⭐**
   - Resolve configuration conflicts between `.js` and `.ts` files
   - Implement Tailwind v4 properly with our design system
   - Convert inline styles to Tailwind utilities throughout codebase
   - Set up proper CSS custom properties integration
2. **Framework Documentation**: Help expand and improve governance framework content
3. **Internationalization**: Add support for new languages
4. **Interactive Tools**: Create visualization and simulation tools
5. **Accessibility**: Improve site accessibility and usability
6. **Performance**: Optimize loading times and user experience

Good First Issues

- **Fix Tailwind CSS configuration** (high impact for experienced developers)
- Convert inline styles to Tailwind utilities
- Fix broken links or typos
- Add missing translations
- Improve mobile responsiveness
- Add alt text to images
- Update outdated dependencies

Ongoing Needs

- **Tailwind CSS modernization** (urgent technical debt)
- Content review and editing
- New framework implementations
- Community tools and features
- Documentation improvements
- Testing and quality assurance

💡 Getting Help

If you need assistance:

1. **Check Documentation**: Review this guide and linked resources
2. **Search Issues**: Look for similar questions or problems
3. **Join Our Discord**: Connect with the developer community
4. **Create GitHub Issues**: For bugs, feature requests, or questions
5. **Pair Programming**: Reach out for collaborative coding sessions

👥 Developer Community

Discord Server: Join our development community at https://discord.gg/Zx4hMJf4JU

- **#dev-design channel** (under 🤝 WORKSPACES category) for technical discussions, code reviews, and architecture decisions
- Real-time chat with other developers and designers
- Voice channels for pair programming sessions and technical meetings
- Quick help for setup issues and development questions

GitHub Discussions: Use repository discussions for:

- Technical questions and solutions
- Feature proposals and architectural discussions
- Pull request coordination
- Documentation feedback

When to Use Which Channel

- **Discord #dev-design**: Quick questions, real-time collaboration, brainstorming
- **GitHub Issues**: Bug reports, feature requests, formal proposals
- **GitHub Discussions**: In-depth technical discussions, documentation questions
- **Pull Requests**: Code reviews, implementation discussions

🎉 Recognition

We value all contributions to the Global Governance Framework project. Contributors will be:

- Listed in our contributors section
- Acknowledged in release notes
- Invited to contributor events and discussions
- Given priority support for their own projects

Thank you for contributing to a more connected and collaborative world through global governance frameworks!

Need immediate help? Join our Discord community in the **#dev-design** channel or create an issue on GitHub.