<!--

SYNC IMPACT REPORT

Version Change: 0.0.0 → 1.0.0
Modified Principles: Initial establishment of all principles
Added Sections:

- Core Principles (7 principles specific to WordPress theme development)
- WordPress Coding Standards
- Development Workflow
- Governance
  Removed Sections: None (initial creation)
  Templates Status:
- ✅ plan-template.md: Compatible (constitution checks will enforce WordPress standards)
- ✅ spec-template.md: Compatible (user stories align with WordPress features)
- ✅ tasks-template.md: Compatible (task structure supports WordPress development)
  Follow-up TODOs: None

Rationale for MAJOR version (1.0.0):

- Initial ratification of constitution with complete WordPress theme development principles
- Establishes governance structure for professional WordPress theme project
  -->

RenalInfo WordPress Theme Constitution

Core Principles

I. WordPress Coding Standards Compliance (NON-NEGOTIABLE)

All code MUST strictly adhere to official WordPress Coding Standards:

- **PHP**: Follow WordPress PHP Coding Standards
- **HTML**: Follow WordPress HTML Coding Standards
- **CSS**: Follow WordPress CSS Coding Standards
- **JavaScript**: Follow WordPress JavaScript Coding Standards
- **Accessibility**: Follow WordPress Accessibility Coding Standards

Rationale: Coding standards ensure consistency, maintainability, security, and compatibility within the WordPress ecosystem. Non-compliance creates technical debt and security vulnerabilities.

Enforcement: All code MUST pass WordPress Coding Standards checks via PHP_CodeSniffer with WordPress rulesets before merging.

II. Theme Review Guidelines Adherence

Theme MUST comply with WordPress.org Theme Review Requirements:

- Use WordPress core functionality and avoid plugin territory
- Implement proper theme support features (`add_theme_support()`)
- Provide complete accessibility (WCAG 2.1 Level AA minimum)
- Include proper licensing (GPL-compatible)
- Sanitize, validate, and escape all data (security-first approach)
- Use proper template hierarchy and WordPress functions
- No hard-coded URLs or site-specific content
- Implement internationalization (i18n) for all strings

Rationale: Theme Review Guidelines represent WordPress community best practices and security standards. Compliance ensures theme quality, security, and potential WordPress.org directory submission.

Enforcement: Pre-release checklist MUST include Theme Check plugin validation with zero errors.

III. Template Hierarchy & Modularity

Theme structure MUST follow WordPress template hierarchy with modular, reusable components:

- Utilize proper WordPress Template Hierarchy
- Create modular template parts using `get_template_part()`
- Implement block templates for Full Site Editing (FSE) compatibility when applicable
- Separate concerns: templates for structure, functions for logic, styles for presentation
- Use hooks and filters for extensibility without template modification

Rationale: Proper template hierarchy ensures WordPress core compatibility, child theme support, and maintainability. Modularity reduces code duplication and simplifies updates.

Enforcement: Every template file MUST have a documented purpose. Code reviews MUST verify proper template part usage and hook implementation.

IV. Security-First Development

Security MUST be built into every feature from the start:

- **Input Validation**: Validate all user inputs and data sources
- **Output Escaping**: Escape all output using appropriate WordPress functions (`esc_html()`, `esc_attr()`, `esc_url()`, `wp_kses()`)
- **Sanitization**: Sanitize data before storage using WordPress sanitization functions
- **Nonces**: Implement nonces for all forms and AJAX requests
- **Capability Checks**: Verify user capabilities before privileged operations
- **SQL Safety**: Use `$wpdb->prepare()` for all database queries
- **File Inclusion**: Validate and sanitize all file paths

Rationale: WordPress themes are frequent security vulnerability targets. Proactive security measures protect site owners and maintain theme reputation.

Enforcement: Security review MUST be performed before any code merge. All database queries, user inputs, and outputs MUST be reviewed for security compliance.

V. Performance & Optimization Standards

Theme MUST prioritize performance with measurable optimization:

- **Asset Loading**: Enqueue scripts and styles properly using `wp_enqueue_script()` and `wp_enqueue_style()`
- **Lazy Loading**: Implement lazy loading for images and heavy resources
- **Minification**: Minify and concatenate CSS/JS for production
- **Caching**: Support WordPress caching mechanisms and transients API
- **Database Efficiency**: Optimize queries, use WP_Query properly, avoid unnecessary queries
- **Image Optimization**: Implement responsive images with srcset and sizes
- **Critical CSS**: Inline critical CSS for above-the-fold content

Performance Targets:

- Google PageSpeed Insights score ≥ 90 (mobile & desktop)
- Largest Contentful Paint (LCP) < 2.5s
- First Input Delay (FID) < 100ms
- Cumulative Layout Shift (CLS) < 0.1

Rationale: Performance directly impacts user experience, SEO rankings, and site success. WordPress themes heavily influence site performance.

Enforcement: Performance testing MUST be conducted for major features. Performance regression blocks deployment.

VI. Accessibility & Inclusivity (WCAG 2.1 Level AA)

Theme MUST be fully accessible to all users:

- Semantic HTML5 markup with proper heading hierarchy
- ARIA labels and roles where appropriate
- Keyboard navigation support for all interactive elements
- Sufficient color contrast ratios (4.5:1 for normal text, 3:1 for large text)
- Focus indicators for keyboard navigation
- Skip links for navigation
- Screen reader compatibility
- Responsive text sizing (no fixed px for font sizes)
- Form labels properly associated with inputs

Rationale: Accessibility is a legal requirement in many jurisdictions and a moral imperative. WordPress has a strong commitment to accessibility.

Enforcement: All features MUST pass automated accessibility testing (WAVE, axe DevTools). Manual keyboard navigation testing MUST be performed.

VII. Internationalization & Localization (i18n/l10n)

Theme MUST support translation and localization:

- All user-facing strings MUST be translatable using WordPress i18n functions
- Use text domain consistently: `renalinfo`
- Implement proper context with `_x()` when needed
- Support RTL (Right-to-Left) languages with RTL stylesheet
- Use `wp_localize_script()` for JavaScript translations
- Load text domain in `after_setup_theme` action
- Provide POT file for translators

Rationale: WordPress is global software. Internationalization ensures theme accessibility to non-English speakers and supports WordPress's global mission.

Enforcement: All strings MUST use i18n functions. Automated checks MUST verify text domain consistency. RTL testing MUST be performed before release.

WordPress Coding Standards

Technology Stack

- **WordPress Version**: 6.0+ (maintain compatibility with two latest major versions)
- **PHP Version**: 8.0+ (minimum 7.4 for backward compatibility)
- **MySQL Version**: 5.7+ or MariaDB 10.3+
- **HTML**: Semantic HTML5
- **CSS**: Modern CSS3 with fallbacks, preprocessors allowed (SCSS/LESS)
- **JavaScript**: ES6+ with transpilation for browser compatibility
- **Build Tools**: Node.js 18+, npm/yarn for asset compilation

Required Theme Files

All themes MUST include these files as per Theme Handbook:

- `style.css` (with proper theme header)
- `index.php`
- `functions.php`
- `screenshot.png` (1200x900px)
- `readme.txt`
- `LICENSE` (GPL v2 or later)

Recommended Structure

```
renalinfo/
├── assets/
│   ├── css/
│   ├── js/
│   ├── images/
│   └── fonts/
├── inc/
│   ├── customizer.php
│   ├── template-functions.php
│   ├── template-tags.php
│   └── classes/
├── template-parts/
│   ├── header/
│   ├── footer/
│   ├── content/
│   └── navigation/
├── languages/
├── style.css
├── functions.php
├── index.php
├── header.php
├── footer.php
├── sidebar.php
├── single.php
├── page.php
├── archive.php
├── search.php
├── 404.php
└── screenshot.png
```

Development Workflow

Environment Setup

Development MUST use local WordPress environment:

- **Recommended Tools**: Local by Flywheel, XAMPP, Docker (official WordPress Docker images), or WP-CLI
- **Version Control**: Git with `.gitignore` excluding `node_modules/`, compiled assets
- **Code Quality**: PHP_CodeSniffer with WordPress-Core, WordPress-Extra rulesets
- **Debugging**: Enable `WP_DEBUG`, `WP_DEBUG_LOG`, `SCRIPT_DEBUG` in development

Development Process

1. **Planning**: Document features in `/specs/` following spec-template.md
2. **Research**: Reference WordPress Codex, Developer Handbook, and Core source code
3. **Implementation**: Follow constitution principles and coding standards
4. **Testing**: Manual testing, Theme Check plugin, accessibility testing
5. **Code Review**: Peer review for standards compliance and security
6. **Documentation**: Update inline documentation and readme.txt

Git Workflow

- **Branching Strategy**: Feature branches from `main`
- **Branch Naming**: `###-feature-name` (e.g., `001-custom-post-types`)
- **Commit Messages**: Clear, descriptive commits following conventional commits
- **Pull Requests**: Required for all changes with checklist verification

Quality Gates

Before merging, ALL of the following MUST pass:

- ☑ PHP_CodeSniffer with WordPress rulesets (zero errors)
- ☑ Theme Check plugin (zero errors, minimal warnings)
- ☑ Accessibility testing (WAVE/axe DevTools)
- ☑ Cross-browser testing (Chrome, Firefox, Safari, Edge)
- ☑ Responsive testing (mobile, tablet, desktop)
- ☑ Performance testing (PageSpeed Insights ≥ 90)
- ☑ Security review (sanitization, escaping, validation)
- ☑ Internationalization verification (all strings translatable)

Governance

Constitution Authority

This constitution supersedes all other development practices and preferences. When conflicts arise between personal coding style and constitutional requirements, the constitution prevails.

Amendment Process

Constitutional amendments require:

1. Written proposal with rationale in `/specs/constitution-amendments/`
2. Impact analysis on existing codebase
3. Team review and approval
4. Version bump following semantic versioning
5. Migration plan for existing code (if applicable)
6. Update to all dependent templates and documentation

Versioning Policy

Constitution follows semantic versioning:

- **MAJOR** (X.0.0): Backward-incompatible governance changes, principle removals/redefinitions
- **MINOR** (0.X.0): New principles added, material expansions to existing guidance
- **PATCH** (0.0.X): Clarifications, typo fixes, non-semantic refinements

Compliance Review

- All pull requests MUST include constitution compliance checklist
- Monthly audits to verify ongoing compliance
- Violations MUST be documented and justified in plan.md Complexity Tracking section
- Unjustified violations block merge

Living Documentation

This constitution is a living document. Developers MUST:

- Reference this document before starting new features
- Propose amendments when principles conflict with WordPress ecosystem evolution
- Update constitution when WordPress core changes impact best practices
- Maintain alignment with official WordPress Developer Handbook

Version: 1.0.0 | **Ratified**: 2025-10-16 | **Last Amended**: 2025-10-16