Advanced Professional WordPress Developer Certification - Exam Guide

Produced by VIP | Enterprise WordPress

Section 1: WordPress Core (~15% of the exam)

1.1 Hooks (Actions and Filters)
Knows when and how to use hooks to extend or modify WordPress behavior.

- Differentiate between actions (for side effects) and filters (for data transformation).
- Identify and use important core hooks.
- Write efficient custom hooks and avoid performance pitfalls.
- Debug hook priorities and conflicts.

1.2 Core REST API Usage and Customization
Knows how to consume, authenticate, and extend the WordPress REST API safely.

- Use core endpoints for CRUD operations.
- Apply authentication methods (cookies, OAuth, application passwords).
- Add validation and sanitization for requests.
- Extend core endpoints with custom fields or routes.

1.3 Options API
Knows how to store and retrieve site-wide settings efficiently.

- Get, set and delete options correctly.
- Differentiate between autoloaded and non-autoloaded options.
- Optimize performance when working with large option sets.

1.4 Transients API
Knows how to cache data temporarily to improve performance.

- Create, retrieve, and delete transients.
- Use transients for expensive operations (e.g., API calls, queries).
- Understand site-wide vs multisite transient storage.
- Handle transient expiration and cache invalidation.

1.5 Cron / Event API

Knows how to schedule and manage tasks in WordPress at scale.

- Register custom cron events.
- Distinguish between WP-Cron and system cron jobs.
- Handle missed or duplicate events.
- Optimize cron performance in high-traffic environments.

1.6 Settings API

Knows how to build secure and extensible admin settings pages.

- Register settings, sections, and fields.
- Validate and sanitize settings input.
- Integrate with block-based settings pages.
- Organize admin UX with tabs, custom pages, or options groups.

1.7 Template Hierarchy

Knows how WordPress resolves templates in both classic and block themes.

- Navigate classic PHP-based template hierarchy.
- Use `theme.json` in block themes.
- Override core templates safely.
- Debug template selection.

1.8 Permalinks and Rewrite Rules

Knows how permalinks are generated and managed in WordPress.

- Register custom rewrite rules for CPTs/taxonomies.
- Flush rewrite rules correctly and avoid unnecessary flushes.
- Troubleshoot common permalink issues.
- Handle multilingual or complex permalink structures.

1.9 Multisite

Knows when and how to implement WordPress Multisite effectively.

- Understand multisite architecture and shared database structure.
- Evaluate when to use multisite vs multiple installs.
- Manage site creation, roles, and permissions.
- Recognize limitations (plugins/themes, domain mapping).

1.10 Roles and Capabilities

Knows how to configure user access with maximum security in mind.

- Create custom roles and capabilities programmatically.
- Restrict access via capabilities instead of roles where possible.
- Avoid storing critical security logic in the database.
- Use least-privilege principles when designing roles.

1.11 WP_Query

Knows how to construct efficient and secure database queries with WP_Query.

- Build custom queries with arguments (meta, taxonomy, date).
- Handle pagination correctly.
- Distinguish between `WP_Query`, `query_posts`, and `get_posts`.
- Optimize queries with indexes and caching.

1.12 Taxonomies

Knows how to leverage core and custom taxonomies for content organization.

- Use categories and tags effectively.
- Register custom taxonomies with labels and rewrite rules.
- Query taxonomy terms in performant ways.
- Handle pitfalls of large or deeply nested taxonomies.

1.13 Post Meta

Knows how to store, retrieve, and query metadata efficiently.

- Use `get_post_meta` and `update_post_meta`.
- Handle serialized data safely.
- Optimize queries on large meta datasets.
- Use meta queries in `WP_Query` appropriately.
- Recognize risks of frequent meta updates (performance, locking).

1.14 Block Editor Basic Architecture
Knows how the Gutenberg editor and block system are structured.

- Register blocks with `registerBlockType`.
- Use `block.json` for block metadata.
- Define block attributes and implement save/render functions.
- Connect JavaScript (editor) with PHP (server-side).

1.15 Custom Post Types (CPTs)
Knows when and how to implement CPTs effectively.

- Register CPTs with `register_post_type`.
- Integrate CPTs with custom taxonomies.
- Define rewrite rules and capabilities for CPTs.
- Understand appropriate use cases.

1.16 Media Library
Knows how the media library architecture works and how to extend the media library

- Understands the data model (attachment posts and post meta)
- Intermediate image sizes
- Knows the architectural difference between list (PHP) and Grid (Backbone.js) views.
- Can extend via APIs/hooks

1.17 Interactivity API
Understands what the interactivity API is and what it can be used for

- Enable reactive, stateful behavior in blocks.
- Build simple client-side interactions without extra frameworks.
- Combine server-rendered data with front-end interactivity.
- Follow accessibility and performance best practices.

Section 2: Custom Development (~15% of the exam)

2.1 Block Editor
Knows how to develop with the block editor, block themes, and custom blocks.

- Write a block theme using `theme.json` and block templates.
- Build custom blocks with `registerBlockType`.
- Create block variations and reusable block patterns.
- Distinguish between static and dynamic blocks.

- Ensure backward compatibility with the classic editor.

2.2 Internationalization (i18n)
Knows how to make WordPress projects translation-ready and multilingual-friendly.

- Use WordPress translation functions.

- Prepare strings and escape output properly for translation.
- Load text domains for plugins/themes.
- Work with `.po` and `.mo` files.
- Understand the role and limitations of multilingual plugins.

2.3 Dependencies in WordPress Code
Knows how to manage dependencies and load assets correctly.

- Enqueue scripts and styles with `wp_enqueue_script` and `wp_enqueue_style`.
- Handle dependencies, versioning, and load order.
- Register and deregister assets responsibly.
- Use action and filter hooks appropriately when loading assets.
- Understand strategies for plugin dependency management.

2.4 WordPress Standards
Knows and applies official WordPress coding standards.

- Follow coding standards for PHP, JavaScript, CSS, and accessibility.
- Use PHPCS and other tooling to enforce consistency.
- Write maintainable, readable, and community-compliant code.

2.5 Code Organization
Knows how to structure WordPress code and templates for clarity and maintainability.

- Organize custom plugin functionality into logical file structures.
- Use consistent naming conventions for templates.
- Separate concerns (logic vs presentation vs configuration).
- Apply WordPress philosophies like “decisions, not options.”

2.6 Activation and Deactivation
Knows how to manage plugin lifecycle events safely.

- Register options, roles, or custom tables on activation.
- Clean up options, roles, or custom data on deactivation/uninstall.
- Flush rewrite rules when appropriate.
- Ensure reversibility and safe failure states.

2.7 Correct Hook Use for Loading Plugin Code
Knows how to load plugin logic only in the proper context.

- Use hooks to defer execution until WordPress is ready.
- Avoid running license checks, API calls, or heavy logic on every request.
- Prevent execution during unintended contexts (REST API calls, WP-CLI).
- Scope code correctly to admin, frontend, or login contexts.

2.8 Running Code in the Correct Context
Knows how to differentiate execution contexts and scope code accordingly.

- Detect WP-CLI context.
- Run WP-CLI commands only during CLI execution.
- Prevent CLI logic from running during HTTP requests (frontend or admin).
- Separate context-specific logic into isolated functions or files.

2.9 PHP Sessions and Caching Issues
Knows why PHP sessions are discouraged in WordPress and their impact on caching.

- Recognize how PHP sessions break page/object caching.
- Avoid session use in plugins/themes.
- Use alternatives like cookies, transients, or the options API.
- Design solutions that scale with enterprise caching layers (CDN, Varnish).

Section 3: Security (~15% of the exam)

3.1 Identifying and Preventing Injection Vulnerabilities
Knows how to avoid SQL, XSS, and command injection in WordPress code.

- Apply sanitization and escaping functions correctly.
- Recognize common injection vectors (e.g., unsanitized meta queries, direct DB queries).
- Distinguish between injection types and their attack surfaces.

3.2 Secure Handling of User Input and Output

Knows how to validate, sanitize, and escape user data in all contexts.

- Use WordPress APIs like `sanitize_text_field()` and `esc_html()`.
- Apply output escaping for different contexts (HTML, JS, URLs).
- Identify unsafe patterns in form handling and submissions.

3.3 WordPress Functions for Sanitization, Validation, and Escaping

Knows which core functions to use in different contexts.

- Select the correct escaping function (`esc_url()` vs `esc_html()`).
- Enforce validation rules in admin and public forms.
- Apply context-aware sanitization consistently.

3.4 WordPress-Specific Injection Vectors

Knows common places where WordPress code can be exploited.

- Secure poorly structured `WP_Query` or `meta_query` arguments.
- Validate and sanitize search inputs.
- Audit plugin code for unsafe query patterns.

3.5 Access Control Best Practices

Knows how to enforce permissions in a secure, least-privilege way.

- Use `current_user_can()` for authorization checks.
- Restrict access by role and capability.
- Apply least-privilege design to custom code.

3.6 Roles and Capabilities System for Least Privilege

Knows how to use roles and capabilities to enforce access securely.

- Create and modify roles programmatically.
- Secure CPTs and REST endpoints with `map_meta_cap()`.
- Avoid granting unnecessary admin-level access.

3.7 Securing AJAX and REST API Endpoints

Knows how to protect custom endpoints against abuse.

- Use `check_ajax_referer()` for nonce validation.

- Enforce capability checks in callbacks.
- Use `permission_callback` for REST routes.

3.8 Nonce Usage and CSRF Protection

Knows how to prevent CSRF attacks with WordPress nonces.

- Generate nonces with `wp_nonce_field()` or `wp_create_nonce()`.
- Verify with `check_admin_referer()` or `wp_verify_nonce()`.
- Apply CSRF protection to both admin forms and front-end AJAX.

3.9 Secure WordPress Configuration Practices

Knows how to harden core configuration and file system settings.

- Lock down `wp-config.php` with server rules.
- Configure salts, keys, and file permissions.
- Disable file editing in wp-admin.

3.10 Limiting Attack Surfaces (File Editing, XML-RPC, etc.)

Knows how to reduce unnecessary exposure in WordPress.

- Disable plugin/theme editors.
- Block or restrict XML-RPC if not needed.
- Harden unused features and APIs.

3.11 Keeping Core, Plugins, and Themes Updated

Knows why updates are critical for security and how to manage them.

- Define update policies for production environments.
- Use WP-CLI or management tools for updates at scale.
- Monitor vulnerabilities in outdated components.

3.12 Nature of Server-Side Request Forgery (SSRF)

Knows what SSRF is and how it can affect WordPress.

- Understand how attackers exploit internal services.
- Identify SSRF risks in imports, image previews, and plugins.
- Recognize patterns of SSRF exploitation.

3.13 SSRF Exploits via Misconfigured Code or Plugins

Knows how insecure remote requests can lead to SSRF.

- Avoid unsafe use of `wp_remote_get()` or `file_get_contents()`.
- Prevent requests to internal IP ranges.
- Restrict outbound requests to approved domains.

3.14 Mitigating SSRF Risks

Knows how to defend WordPress environments against SSRF.

- Filter and validate remote request URLs.
- Implement hosting/firewall protections.
- Block loopback requests where not required.

3.15 Types of DoS and DDoS Attacks

Knows common denial-of-service techniques and their impact.

- Identify volumetric and application-layer DoS.
- Understand brute force and wp-cron exploitation.
- Recognize resource exhaustion tactics.

3.16 Limiting Resource Exhaustion

Knows how to prevent overuse of server resources.

- Apply rate limiting and throttling.
- Use CAPTCHA or login attempt limits.
- Implement CDN-based or host-level mitigation.

3.17 Spotting Abuse and Brute Force Attacks

Knows how to detect signs of automated attack activity.

- Analyze logs for repeated failed logins.
- Use security plugins for brute force detection.
- Monitor unusual traffic spikes or patterns.

3.18 Logging and Audit Trails in WordPress

Knows what to log and why for forensic security.

- Track logins, role changes, and admin actions.

- Store logs securely for analysis.
- Use logging frameworks or monitoring plugins.

3.19 Implementing Logging for Authentication, Changes, and Errors
Knows how to capture critical system and user events.

- Log failed/successful logins.
- Track plugin/theme updates and code changes.
- Capture PHP and WordPress errors systematically.

3.20 Monitoring and Responding to Suspicious Activity
Knows how to detect and react to potential compromises.

- Set up log monitoring and alerting.
- Use intrusion detection tools.
- Respond to breaches with incident handling steps.

3.21 WordPress Coding Standards for Secure Development
Knows how coding standards enforce better security practices.

- Follow WordPress Coding Standards (WPCS).
- Use PHPCS for linting and enforcement.
- Structure code to minimize security risks.

3.22 Role of Static Analysis Tools
Knows how to integrate automated security checks into workflows.

- Use static analysis to detect unsafe code patterns.
- Integrate tools like PHPStan.
- Review and address flagged issues proactively.

3.23 Incorporating Security Checks into CI Pipelines
Knows how to automate security in continuous integration workflows.

- Set up automated vulnerability scans.
- Enforce code quality/security gates.
- Use WPScan, PHPStan, or equivalent tools in CI/CD.

Section 4: Performance (~15% of the exam)

4.1 Performance Throughout the Request Cycle
Knows how performance is affected from DNS resolution to browser rendering.

- Understand the WordPress execution lifecycle.
- Identify bottlenecks at each stage (server, PHP, DB, front-end).
- Optimize both server-side and client-side performance.

4.2 Full Page Caching Strategies in WordPress
Knows when and how to use page caching for scale.

- Compare plugin-level, server-level, and CDN caching.
- Handle cache invalidation correctly.
- Measure and validate cache effectiveness.

4.3 Unbounded Queries
Knows why unbounded MySQL queries degrade performance.

- Recognize queries without limits or constraints.
- Apply pagination or batching strategies.
- Optimize query patterns for scale.

4.4 Using NOT IN
Knows how NOT IN impacts MySQL query performance.

- Identify performance penalties of NOT IN queries.
- Apply alternative query strategies where possible.

4.5 Indexing
Knows how to use indexes to speed up queries.

- Spot missing indexes after imports/migrations.
- Understand when to use primary, composite, or unique indexes.
- Diagnose index usage with query analysis tools.

4.6 Using LIKE
Knows why LIKE queries are slow and how to avoid them.

- Recognize inefficiencies in post meta queries.
- Use exact key/value matches instead of wildcards.
- Apply indexing to improve pattern matching.

4.7 Using EXPLAIN

Knows how to analyze queries with EXPLAIN to find bottlenecks.

- Interpret EXPLAIN output (filesort, full table scans, index usage).
- Diagnose unintended index usage.
- Adjust queries or indexes based on EXPLAIN results.

4.8 Post Meta Performance

Knows performance challenges of bloated post meta usage.

- Avoid storing large or inappropriate data in post meta.
- Optimize queries with meta joins.
- Consider alternative storage (custom tables).

4.9 Auto-Loaded Options

Knows how autoloaded options affect page load.

- Identify unnecessary autoloaded options.
- Limit the size and number of autoloaded values.
- Optimize options usage during site initialization.

4.10 Uncached Functions

Knows which WordPress functions can be expensive without caching.

- Identify functions that repeatedly query the database.
- Cache results where appropriate.
- Replace uncached calls with more efficient patterns.

4.11 Sitemap and Archive Performance

Knows how deep crawling affects performance.

- Understand sitemap and archive query limits.
- Apply pagination or query restrictions.
- Optimize for SEO crawlers without overloading the DB.

4.12 External HTTP Requests

Knows performance implications of server-side external requests.

- Identify blocking requests during page generation.
- Cache responses from external APIs.
- Offload or defer requests where possible.

4.13 Query Offloading (e.g., Elasticsearch)

Knows when to offload queries to external search systems.

- Identify queries too complex for MySQL (full-text, faceted).
- Understand when MySQL is sufficient.
- Integrate Elasticsearch or similar tools for scalability.

4.14 Persistent Object Cache

Knows the role of persistent object caching for performance.

- Differentiate between persistent and non-persistent caching.
- Implement Redis or Memcached via drop-ins.
- Understand cache invalidation with object cache.

4.15 Transients

Knows how transients behave with and without object caching.

- Store temporary data with expiration.
- Understand limitations when persistent cache is not enabled.
- Use transients appropriately for non-critical caching.

4.16 Shutdown Hook

Knows how to use the shutdown hook for post-response tasks.

- Execute actions after response delivery.
- Defer logging, cleanup, or async tasks.
- Avoid performance impact on page loads.

4.17 Profiling Page Generation and Hook Execution

Knows how to benchmark WordPress execution and identify bottlenecks.

- Use Query Monitor and custom timing code.

- Profile actions, filters, and template loading.
- Trace slow plugin/theme operations.

4.18 Deferring or Caching Remote API Calls
Knows strategies to reduce latency from external APIs.

- Cache responses with transients or object cache.
- Prefetch or schedule calls instead of real-time requests.

- Avoid blocking user-facing requests.

4.19 Caching Expensive Backend Operations at the Right Layer
Knows how to choose the right caching strategy for heavy operations.

- Cache complex queries, calculations, or API results.
- Apply caching at object, transient, or page level as needed.
- Handle cache invalidation safely and efficiently.

[TODO - Front End Performance Subject areas to follow here, sections 4.20 - 4.30]

Section 5: Change Management (~10% of the exam)

5.1 Difference Between Unit, Integration, and Acceptance Testing
Knows the purpose and role of each testing type in the development lifecycle.

- Define unit, integration, and acceptance testing.
- Identify when to use each type.
- Explain how testing contributes to stability and reliability.

5.2 Writing and Running Unit Tests with PHPUnit
Knows how to create and execute unit tests for WordPress code.

- Set up PHPUnit in a WordPress environment.
- Write test cases for functions and classes.
- Mock dependencies effectively.
- Run tests locally and in CI pipelines.

5.3 Performing Integration Testing

Knows how to test plugin and theme interactions in a real environment.

- Create integration tests spanning multiple components.
- Include database, hooks, and plugin interactions.
- Isolate integration concerns.
- Validate real-world code behavior.

5.4 Structuring and Conducting User Acceptance Testing (UAT)

Knows how to validate features against user workflows.

- Design UAT scenarios around business logic.
- Capture user behavior in scripts.
- Document acceptance criteria clearly.
- Ensure end-user requirements are met.

5.5 Automating Tests in CI/CD Pipelines

Knows how to integrate testing into continuous delivery workflows.

- Configure pipelines to run tests on push/pull requests.
- Ensure tests run in isolated environments.
- Automate both unit and integration testing.
- Enforce test coverage requirements.

5.6 Safe Deployment Practices

Knows principles of secure and reliable WordPress deployments.

- Use atomic or zero-downtime deploys.
- Separate configuration from code.
- Ensure deployments are repeatable and reversible.
- Minimize downtime during updates.

5.7 Deploying Updates with Version Control and CI/CD

Knows how to use Git workflows with automated deployments.

- Integrate Git with deployment tools.
- Trigger builds on branches/tags.
- Manage secrets and environment variables securely.
- Automate WordPress updates with CI/CD pipelines.

5.8 Configuring CI/CD Systems for WordPress Deployment
Knows how to set up GitHub Actions, GitLab CI, or similar tools.

- Create workflows for automated deploys.
- Write build and deployment scripts.
- Secure workflows with proper credentials.
- Control access to environments.

5.9 Environment-Specific Configuration
Knows how to separate logic between production, staging, and development.

- Use environment variables
- Prevent cross-environment misconfigurations.
- Configure environment-aware plugins/themes.
- Store sensitive data securely.

5.10 Importance of Pre-Deployment Staging Environments
Knows how staging protects production during changes.

- Replicate production conditions in staging.
- Test deployments and migrations safely.
- Validate changes before going live.
- Integrate staging with CI/CD pipelines.

5.11 Git Branching Strategies
Knows how to structure collaborative workflows with Git.

- Apply Git Flow, trunk-based, or feature branching.
- Choose strategies based on team/project needs.
- Maintain release and hotfix branches.
- Avoid long-lived divergent branches.

5.12 Managing WordPress Projects in Git
Knows how to structure repositories for collaborative development.

- Organize repos for plugins, themes, or monorepos.
- Use submodules or subtrees when needed.
- Support multi-team workflows.
- Standardize repo structure for maintainability.

5.13 Managing and Reviewing Pull Requests
Knows how to ensure quality and clarity during code reviews.

- Conduct peer reviews effectively.
- Enforce coding standards with CI checks.
- Provide clear documentation in PRs.
- Communicate feedback constructively.

5.14 Resolving Merge Conflicts and Maintaining History
Knows how to handle conflicts while keeping a clean Git history.

- Resolve merge conflicts safely.
- Understand rebase vs. merge workflows.
- Maintain readable, consistent commit logs.
- Support rollback and auditability.

5.15 Rolling Back Failed Deployments
Knows how to revert safely when deployments go wrong.

- Use Git-based rollbacks.
- Roll back plugins/themes with WP-CLI or tooling.
- Maintain deployment version history.
- Minimize downtime during rollback.

5.16 Versioning Plugins/Themes and Managing Changelogs
Knows how to communicate and track updates effectively.

- Apply semantic versioning consistently.
- Write clear changelogs for users and clients.
- Manage version updates in Git and deployments.
- Ensure compatibility with WordPress releases.

5.17 Backup and Restore as Part of Deployment
Knows how to safeguard and recover WordPress environments.

- Set up automated file and database backups.
- Test restores in staging environments.
- Restore partial or full backups with minimal downtime.
- Integrate backup routines into deployment workflows.

5.18 Deployment Logs and Monitoring Tools
Knows how to track and diagnose deployments.

- Read and interpret deployment logs.
- Identify and fix failed processes.
- Monitor application behavior post-deploy.
- Detect regressions early through monitoring.

Section 6: Debugging (~10% of the exam)

6.1 Debug Bar
Knows how to inspect queries, hooks, and request info in the admin toolbar.

- View database queries and execution time.
- Inspect hooks firing on a page load.
- Review request/response details in admin.

6.2 Query Monitor
Knows how to identify slow queries, HTTP calls, and enqueued assets.

- Trace database query performance.
- Debug HTTP requests and responses.
- Inspect conditionals, hooks, and loaded assets.

6.3 PHP Error Logs
Knows how to read and interpret PHP errors, warnings, and notices.

- Access PHP error logs in different environments.
- Configure error reporting for development vs production.
- Distinguish fatal errors from notices.

6.4 Xdebug
Knows how to step through WordPress code with Xdebug.

- Set breakpoints and step into functions.
- Trace function calls and execution order.
- Profile performance and memory usage.

6.5 APM Tools (New Relic, Datadog)
Knows how to monitor performance with application monitoring tools.

- Identify slow transactions and bottlenecks.
- Trace backend performance metrics.
- Monitor WordPress processes in real-time.

6.6 Safe Debugging in Non-Production Environments
Knows how to replicate production issues safely in staging or dev.

- Configure staging to match production.
- Reproduce errors without impacting live users.
- Debug configurations before deployment.

6.7 Browser Developer Tools
Knows how to debug front-end performance and JavaScript issues.

- Inspect DOM changes and styling.
- Analyze network requests and load times.
- Debug JavaScript errors in console.

6.8 Request and Response Headers
Knows how to debug caching, redirects, and authentication via headers.

- Inspect authentication headers.
- Diagnose caching behavior (e.g., cache hits/misses).
- Trace redirect loops via headers.

6.9 cURL
Knows how to test endpoints and APIs from the command line.

- Send GET/POST requests to WordPress endpoints.
- Simulate headers, cookies, and authentication.
- Inspect raw responses for debugging.

6.10 Graphical HTTP Clients (Postman, Insomnia)
Knows how to test REST API endpoints with GUI tools.

- Send custom REST or AJAX requests.

- Modify headers, tokens, and payloads.
- Save request collections for reuse.

6.11 Host File Entries

Knows how to route local domains to staging or dev servers.

- Edit hosts file to map domains to IPs.
- Test staging environments with live domains.
- Debug DNS-related issues locally.

6.12 Debugging with Actions and Filters

Knows how to use hooks for inspection and overriding logic.

- Add debug callbacks to actions and filters.
- Capture data passing through hooks.
- Temporarily override default behavior.

6.13 Terminal-Based Tools

Knows how to analyze server performance using CLI utilities.

- Familiarity with common terminal commands used directly on a webserver

6.14 WP-CLI for Debugging

Knows how to inspect and query WordPress data with WP-CLI.

- Familiarity of common WP_CLI commands e.g. wp option get, wp db query, `wp cron event list`.
- Debug scheduled tasks and runtime state.
- Interact with database directly.

6.15 WordPress Shell

Knows how to debug interactively with wp shell or WP Console.

- Evaluate PHP expressions in real time.
- Inspect variables and objects.
- Test function output without modifying code.

6.16 Custom Debug Code
Knows how to log or dump values for troubleshooting.

- Use `error_log()` and `var_dump()`.
- Add conditional debug statements.
- Remove temporary code after debugging.

6.17 Custom Response Headers
Knows how to send headers for debugging workflows.

- Add debug headers to WordPress responses.
- Inspect headers in browser tools.
- Use headers for API testing.

6.18 Custom WP-CLI Commands for Debugging
Knows how to build WP-CLI commands to expose site internals.

- Write custom CLI commands.
- Query runtime state programmatically.
- Output structured debug information.

6.19 Custom Logs
Knows how to maintain application-specific debug logs.

- Write to custom log files.
- Organize debug output by feature.
- Rotate and manage log size.

6.20 Extending Query Monitor
Knows how to add custom panels for debugging themes/plugins.

- Create Query Monitor extensions.
- Display plugin-specific runtime info.
- Provide targeted debug views.

6.21 Local Development Environment Debugging
Knows how to set up and debug in a local WordPress environment.

- Use Docker, wp-env, or custom stacks.

- Sync data and configs with production.
- Profile local performance.

6.22 Static Analysis Tools
Knows how to catch errors before runtime using static analysis.

- Use PHPStan, Psalm, and PHPCS.
- Detect unsafe code and deprecated functions.
- Integrate static analysis into CI.

6.23 Database GUI Tools
Knows how to safely inspect and edit WordPress data via GUIs.

- Browse and query WordPress tables.
- Safely update and export data.
- Visualize table structures.

6.24 Remote Database Debugging with SSH Tunnels
Knows how to securely connect to remote DBs for troubleshooting.

- Configure SSH tunnels for DB access.
- Query staging/production databases.
- Protect credentials during connections.

6.25 Site Health and Site Info
Knows how to use built-in diagnostics in WordPress admin.

- Review Site Health status and recommendations.
- Inspect environment details via Site Info.
- Identify misconfigurations or missing modules.

6.26 IDE Customization for Debugging
Knows how to configure IDEs for WordPress debugging.

- Integrate Xdebug in IDEs.
- Set breakpoints and debug sessions.
- Install WordPress-specific extensions.

6.27 Local WordPress Installations

Knows how to manage and troubleshoot local installs.

- Debug with wp-env, Local, or Docker.
- Handle configuration mismatches.
- Test plugins/themes in isolation.

6.28 Debugging Object Cache Issues

Knows how to diagnose cache inconsistencies and stale data.

- Inspect cache groups and keys.
- Debug persistent cache backends.
- Resolve cache invalidation issues.

6.29 Debugging Redirect Loops

Knows how to diagnose and resolve redirect bugs.

- Trace redirect logic with headers.
- Inspect wp_redirect() usage.
- Identify plugin or server-level loops.

6.30 Database Connection Issues

Knows how to troubleshoot failed DB connections.

- Verify credentials in `wp-config.php`.
- Check DB server availability.
- Debug socket and timeout errors.

6.31 Optimizing Slow Database Queries

Knows how to analyze and fix inefficient queries.

- Use EXPLAIN to interpret execution plans.
- Apply indexing strategies.
- Identify slow queries with Query Monitor.

6.32 Database Indexing and Table Types

Knows how schema design affects performance.

- Understand primary/secondary indexes.

- Interpret autoincrement behavior.
- Compare MyISAM vs InnoDB tradeoffs.

6.33 Counting and Classifying Tables
Knows how to analyze table counts and types in WordPress.

- List database tables by type.
- Count custom vs core tables.
- Identify unused or orphaned tables.

6.34 Assessing Table Size and Growth
Knows how table size impacts performance and backups.

- Measure table sizes and growth trends.
- Diagnose oversized log/meta tables.
- Plan cleanup or archiving strategies.

6.35 Options Table Performance Issues
Knows how to debug bloated or autoloaded options.

- Identify large autoloaded values.
- Audit the options table for performance risks.
- Optimize autoload usage.

6.36 Debugging in Production Safely
Knows how to debug live systems with minimal risk.

- Use read-only or limited logging techniques.
- Minimize overhead during production debugging.
- Roll back temporary debug changes quickly.

6.37 Intermittent/Difficult-to-Reproduce Bugs
Knows how to capture and reproduce rare issues.

- Use logging to capture intermittent behavior.
- Apply traffic/session recording.
- Analyze patterns across requests.

6.38 Bypassing or Disabling Code/Plugins
Knows how to isolate issues by disabling components.

- Temporarily bypass plugin code.
- Disable hooks or themes selectively.
- Confirm sources of conflicts.

6.39 Capturing and Replaying HTTP Requests
Knows how to reproduce API/AJAX bugs with captured requests.

- Capture request payloads and headers.
- Replay with cURL and other tools.
- Compare expected vs actual responses.

6.40 Out-of-Memory Errors
Knows how to diagnose and fix PHP memory exhaustion.

- Increase memory limits.
- Profile memory usage.
- Identify memory-heavy plugins or queries.

6.41 Debugging REST API Requests
Knows how to inspect and debug API calls in WordPress.

- Inspect permissions and authentication.
- Check response codes and payloads.
- Trace custom REST routes.

6.42 PHP Stack Traces
Knows how to read stack traces to identify root causes.

- Interpret call order in trace logs.
- Map traces to source code.
- Identify misbehaving functions.

6.43 Debugging WordPress Core Code
Knows how to step through and inspect core files.

- Trace execution paths in core.

- Add temporary debug hooks to core (safely).
- Contribute patches for reproducible bugs.

6.44 Debugging Server-Side HTTP Requests
Knows how to debug WordPress remote requests.

- Inspect calls from `wp_remote_get`/`wp_remote_post`.
- Diagnose failures or latency.
- Apply caching to reduce impact.

6.45 Cookie Issues
Knows how to debug authentication/session cookie problems.

- Inspect cookies in browser tools.
- Debug cookie expiration and persistence.
- Resolve conflicts across plugins or domains.

6.46 Client-Side JavaScript Debugging
Knows how to debug JS issues in WordPress themes/plugins.

- Use browser console and breakpoints.
- Debug script dependencies and load order.
- Trace front-end rendering bugs.

6.47 Debugging Headless/Decoupled WordPress
Knows how to debug API-driven and headless architectures.

- Trace data flow between WordPress and frontend frameworks.
- Debug CORS and authentication issues.
- Inspect API payloads powering headless frontends.

Section 7: Scalability & Architecture (~10% of the exam)

7.1 Building Performant Pages with Large Datasets
Knows how to design WordPress features that handle large volumes of posts, users, or meta data efficiently.

- Optimize loops and queries for large datasets.

- Apply efficient code design to reduce memory and CPU load.
- Recognize backend limits and plan around them.

7.2 Optimizing Queries, Pagination, and Indexing

Knows how to make content rendering scalable through query and DB optimizations.

- Use pagination to limit results.
- Restrict fields returned in queries.
- Avoid unbounded loops.
- Apply proper indexes or custom tables.

7.3 Multi-Level Caching Strategies

Knows how to apply caching at object, fragment, and full-page levels.

- Use persistent object caches.
- Cache expensive template fragments.
- Implement full-page caching with plugins or CDNs.
- Balance caching layers to reduce server load.

7.4 Vertical vs. Horizontal Scaling

Knows the trade-offs between scaling up and scaling out WordPress infrastructure.

- Differentiate CPU/RAM upgrades (vertical) vs multi-server scaling (horizontal).
- Recognize when vertical scaling is cost-effective.
- Plan for horizontal scaling at higher traffic levels.

7.5 Designing High-Traffic Architectures

Knows how to architect WordPress for scale with modern infrastructure.

- Use load balancers to distribute traffic.
- Deploy WordPress in containers (Docker, Kubernetes).
- Design stateless application layers.
- Handle shared assets and sessions.

7.6 CDN Integration

Knows how to configure CDNs to offload traffic and improve global performance.

- Integrate CDNs like Cloudflare, Akamai, or Fastly.
- Configure caching for static assets.

- Optimize DNS and routing through CDN.
- Tune cache behaviors to reduce origin load.

7.7 Edge Caching and Cache-Control Headers

Knows how to push caching closer to users with edge rules and headers.

- Configure cache-control headers (`max-age, stale-while-revalidate`).
- Enable HTML caching at the CDN edge.
- Apply edge rules for performance-critical paths.

7.8 Offloaded Search Services

Knows how to implement scalable search using external engines.

- Integrate Elasticsearch or OpenSearch.
- Index WordPress content efficiently.
- Build faceted or full-text search interfaces.
- Offload heavy search queries from MySQL.

7.9 Decoupling Features to Reduce Load

Knows how to offload background and resource-intensive tasks from the web tier.

- Move analytics, search, or reporting into separate services.
- Implement background processing with queues.
- Use microservices for non-critical workloads.

7.10 Infrastructure Monitoring and Autoscaling

Knows how to monitor and scale WordPress hosting environments dynamically.

- Use tools like New Relic or Datadog for infrastructure metrics.
- Define thresholds for CPU, memory, and response time.
- Configure auto-scaling groups or container clusters.
- Plan for traffic spikes and failover.

Section 8: Disaster Recovery (~10% of the exam)

8.1 Safe Database Restores

Knows how to restore full or partial WordPress databases without causing data loss or integrity issues.

- Restore databases with WP-CLI, phpMyAdmin, or SQL commands.
- Ensure schema compatibility during restores.
- Avoid overwriting current data unintentionally.
- Validate data integrity post-restore.

8.2 Restoring the Codebase

Knows how to roll back the WordPress codebase to a safe, stable version.

- Restore code from backups or Git.
- Ensure file integrity and dependency consistency.
- Match codebase with database schema.
- Validate environment stability after rollback.

8.3 Identifying and Cleaning Corrupted Data

Knows how to detect and repair corrupted or compromised database entries.

- Find malformed content or orphaned metadata.
- Diagnose inconsistent taxonomy relationships.
- Clean corrupted data without further disruption.
- Document and validate cleanup actions.

8.4 Writing Scripts for Recovery and Remediation

Knows how to script database fixes and migrations during recovery.

- Write scripts to patch or transform data.
- Handle serialized data safely.
- Repair user accounts and relationships.
- Restore missing content from logs or external sources.