WordPress

Architectural Description and Diagrams

Aisha Abdulwahab
aabdulwa@depaul.edu

Katie Ruane
kruane2@depaul.edu

David Zwerdling
dzwerdli@depaul.edu

Version 1
March 6, 2023

1 Version History

Version	Date	Author	Comments
1	February 20, 2023	A. Abdulwahab, K. Ruane, and D. Zwerdling	Initial version
2	March 6, 2023	A. Abdulwahab, K. Ruane, and D. Zwerdling	Amended with Analysis.

Contents

2 Introduction

This report analyzes the software architecture of WordPress, a content management system. In one survey[11] the system was identified to be running at least 40-43% of websites[1] on the Internet. The system is open-source and its internal PHP and runtime are also used in user-exposed templates. These facts together enable small-scale providers and webmasters to easily build websites.

Our goal is to have a better understanding of the WordPress architecture, the qualities attributes that the WordPress team prioritizes while creating the software, and the programming languages and patterns that are employed for the system. This report has the following objectives:

- Read through and understand the available documentation to comprehend its underlying architecture.
- After analyzing documentation, identify the architectural requirements, style, and quality attributes of the chosen system.
- Examine the code and model the system using one of the architectural frameworks covered in class.
- Using the knowledge we have learned in class over the quarter, give our feedback on how well the WordPress system was designed architecturally.

3 System Overview

3.1 Scope

WordPress is a web-oriented content management service (CMS)[2]. In this role, WordPress comprises the following functionality:

- Site administration, in that the platform contains administrative elements and thus does not require an external tool to administer.
- Interacting with web content, in that content is able to be both created and read by users of various types.
- Site extensibility, in that there is a highly functional plug-in capability. Since the system can be arbitrarily extended, we will only cover the most common cases here.

Having been developed for nearly two decades, the architecture is very mature, and worth examining in the context of a long-lived free open source software project worked on by hundreds of developers.

3.2 Context

Most WordPress installations perform the task of serving content over the internet. The system's main task is to listen on HTTP for page requests. When a page request arrives, it is routed by the system to the appropriate site site in the network. WordPress then produces the appropriate response – typically themed HTML, CSS, and referenced files on the requested page – and sends the appropriate data back to the client where the page is rendered for the user.

In addition to the Internet, the system also interacts with the host's local file system and a MySQL[3] server. Together, these elements determine the behavior of the system when a request arrives including themes, templates, and plug-ins.

[1] In this survey a website was essentially identified as a 2nd-level domain.

[2] WordPress now also comprises an entire brand, but these are aspects of WordPress with which this analysis will not concern. Instead it will exclusively focus on the technical aspects of the free open source php-based CMS platform under that name. This analysis will also ignore the free/paid hosting provider service under the brand, for example.

[3] Even though they are distinct products, this analysis will consider both MySQL and MariaDB as referred to by "MySQL". WordPress is compatible with both.

The network has connections to the local filesystem as well as a database. The architecture does not impose a limit on the type of network: it is possible to run WordPress on the internet, as a wiki on a corporate LAN, or even as a local service available only to the local host.

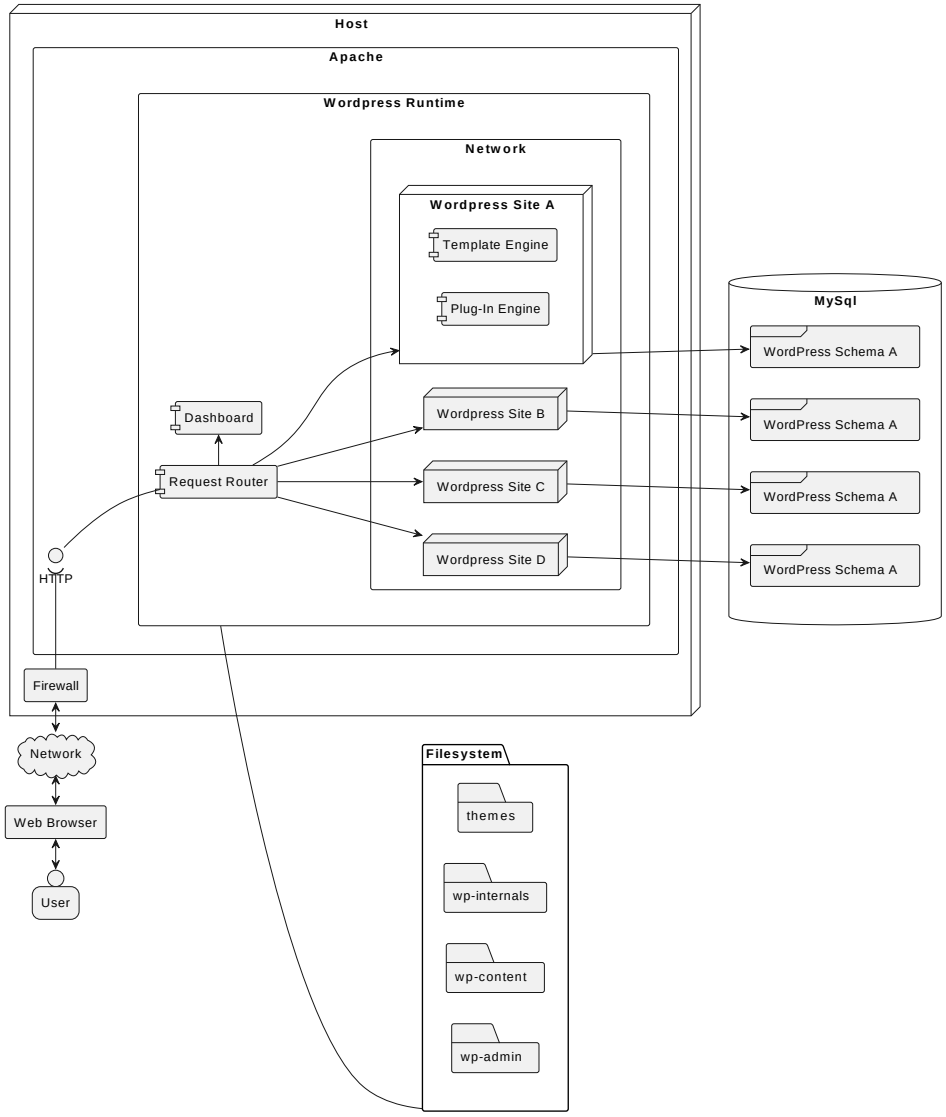

Figure 1: A hypothetical install of WordPress connected to a network with multiple sites within.

3.3 Audience

This document is intended for developers, open-source contributors, project managers, and consumers involved in the continuous development, maintenance, and use of a WordPress website. It will provide the audience with a detailed understanding of the underlying architecture of the WordPress software system to gain better insight into how all components interact to deliver a functional and robust platform. By reading this document developers and open-source contributors can make informed decisions about how to design and implement various functionalities that will best deliver an optimal user experience. Project managers and consumers will be able to best determine if their website is scalable, maintainable, and aligned with business requirements. This document aims to provide the audience knowledge to support decisions about identifying potential issues or

areas of improvement in the system architecture. The purpose is to be a comprehensive overview of the WordPress software architecture to support all audience members involved in the development, maintenance, or usage of a WordPress website.

3.4 Statement of Purpose

The popular content management system WordPress, which powers millions of websites worldwide, is thoroughly examined in this architectural document. We investigated the WordPress architecture in-depth as a team, and we have supplied diagrams that make it easier to comprehend the internal workings of the system. Various aspects of the WordPress architecture, including its core elements, modules, plugins, and code structure, have been examined. To give a complete picture of the system, both its functionality and quality attributes have been examined. The investigation has shown that the WordPress architecture offers several advantages, including a modular design, flexibility, and usability that have helped it become so popular. Nevertheless, the investigation has also pointed out other flaws that may be strengthened to improve the application's overall performance, such as its heavy reliance on plugins and the possible security risks they provide. For individuals looking to better understand WordPress architecture, this architectural document will be an invaluable reference.

4 Glossary

- **Block**
  *Block*s are a feature of the WordPress template subsystem. A relatively modern addition, older styles of composing page templates are still available, though deprecated. When composing a page template, the designer can select from the following elements, or create their own. At runtime, these components are transformed into parsable HTML.

  New template blocks can be also be added, and existing blocks can be created by merging existing blocks using the "block group" block.

- **CMS**
  Content management systems, generally, assist the creation and delivery of web-oriented content. They are used to create multimedia for download by users.

- **Category**
  Arbitrary, mutually exclusive, site-defined groups of content pages.

- **Hooks**
  Actions and filters are collectively referred to as *hooks* because they allow plug-in developers to attach to events of various kinds within the system.

- **LAN**
  The *local area network* of the server in question. Typically, WordPress is configured to serve content to the Internet, but there is a use case for a fire-walled intranet or local install, such as corporate or personal information portals.

- **MariaDB**
  A free open-source software alternative to MySQL. *MariaDB* aims to be a drop-in replacement for MySQL. This product was created after MySQL implemented no-free licensing.

- **MIME-type**
  Defines the contents of a file transferred via browser over the internet. E.g.: 'text/javascript','text/html', 'image/png' etc.

- **MySQL**
  A database platform. In the WordPress install, stores posts, comments. users and roles, along with other dynamic elements.

- **Multisite**
  An installation which hosts multiple sites via the network feature is termed *multisite*.

- **Network**
  Each WordPress installation is capable of hosting multiple sites, termed *network*s.

- **Network Intermediary**
  A *network intermediary* receives and interprets network traffic. In this paper we use network intermediaries to refer to firewalls, load balancers, and memory caches. Network intermediaries are capable of mitigating certain performance-related aspects of the WordPress system.

- **PHP**
  The (recursive acronym) *PHP: Hypertext Preprocessor* is a programming language in which WordPress is developed. While it is a general-purpose programming language, it specializes in producing HTML-formatted responses to HTTP requests.

- **Publish**
  In WordPress terminology, "publish" means to make a piece of content public.

- **Publisher**
  An organization or individual publishing content to the internet. This is the primary, broad-case user of the system.

- **RBAC**
  Role-based authorization control enables systems administrators to allocate system rights based on user roles. While some systems allow a modular approach to RBAC, WordPress uses a hierarchical approach modeling a content publishing organization. See Appendix B. This does not allow piecemeal allocation of rights based on fine-grained capabilities, but can be extended to do so using plug-ins.

- **Site**
  WordPress designates a user space boundary at the site level. This means users within a network may not have capabilities on other sites within the network, which allows Super-Admins (See Appendix B) to designate CMS-oriented responsibilities to Admins, Authors, Editors, and Contributors per site. Networks can be configured to route site-specific requests using domain names or URL paths.

- **Tag**
  An arbitrary, non-exclusive, site-defined way to group content pages.

- **Taxonomy**
  WordPress *taxonomies* represent methods of classification of content pages. That is, not the classification themselves, but the meta-category in which the categorization takes place. Analogously, "breed of dog" may be a taxonomy, whereas "German Shepherd" may be a 'term', whereas "My dog, Max" may be classified into that term. Similarly "age of dog" could be a perpendicular method of classification, represented by a separate taxonomy.

- **Term**
  WordPress *terms* represent categories within a classification. Pages can be assigned terms by each taxonomy defined in a site.

- **WYSIWYG**
  What you see is what you get editors allow non-technical users to produce documents which appear the way the user has specified by allowing the user to edit a format of the document that closely resembles the final product of the document. For example, modern word processors allow a user to select text and make it bold, immediately showing the change in the document the user is editing. This is in contrast to documents which include, and are displayed differently than, their markup – such as HTML.

- **XSS**
  Cross-site scripting is a form of attack in which a user inputs malicious code into a page via publicly accessible forms. E.g.: A user may add JavaScript code to a comment box, and submit the content to the server. If no filtering exists, the content may be served to other users viewing the page, executing the JavaScript and performing arbitrary actions which are not necessarily sanctioned by the site owner.

5 Requirements and Stakeholders

5.1 Stakeholders

A stakeholder is a person, group, or organization that has an interest or concern in an organization or project. Stakeholders can directly or indirectly impact a project and can be either internal or external to the organization.

According to the WordPress Governance project[1], in the context of a WordPress application, stakeholders can include:

- **Consumers/End users**
  Those who develop and maintain websites using the WordPress application.
- **Site Owners**
  The owners and operators of WordPress-powered websites.
- **WordPress Developers**
  The people who develop and maintain the WordPress software.
- **Theme and Plug-in Developers**
  The individuals that develop WordPress plugins and themes to enhance its functionality.
- **Hosting Providers**
  Organizations that provide hosting for WordPress-powered websites.
- **Open-Source Community**
  The community of developers and users who contribute to developing and maintaining the WordPress software.
- **Investors**
  The people who have contributed in the development and growth of WordPress.
- **Government Agencies**
  Regulating bodies such as the Federal Trade Commission or the European Union that may regulate and impact the development and use of WordPress.
- **Past, Present, and Future Innovators**
  These are individual designers, engineers, and other groups of people working together to change the WordPress system by integrating it with other software

5.2 Overview of Requirements

Requirements for WordPress aim to create a seamless experience for its users from the moment it is installed. The WordPress's Content Management System will provide a quick and easy way to build and modify a webpage. With the support of plug-in repository, users will be able to tailor the web development process to their unique needs. The system creates an organizational hierarchy of roles with various capabilities that will affect the overall management of the website. Storing and accessing website data will be possible with the secured connection to an external database where users can immediately access. WordPress will also consistently update security so users can feel comfortable knowing their website content is safe.

5.2.1 Functionality

WordPress's main built-in functionality is centered around serving dynamically-generated web content. In the most common workflow, an author will create or modify a URL-addressed web page, which is then served as a response to subsequent web-users http requests. Administrators can install plug-ins to enhance the functionality of the networks[4] or sites.

Here are the top-level tasks the platform is capable of performing:

- **Administration**
  WordPress can be managed at the site network level using administration integrated into the WordPress Dashboard. That is, once a WordPress system is installed, there are no other tools necessary to add users, install plugins, update itself, change settings, or stand-up additional WordPress sites.

- **Content Management**
  Contributors can add, remove, or modify content available on a given site.

  WordPress systems listen for HTTP requests and serve content to web browsers such as HTML, JavaScript, CSS, and other MIME type content. The pages in question may be a content page created by an editor, the dashboard used to administrate the site, or a plugin configuration page – everything is done through the web browser-based interface.

  The code which is ultimately run by the client's machine will have additional analysis as well, which do serve as technical decisions relating to the decision to make everything web oriented.

- **Plug-Ins**
  Through a robust standards-based interface, WordPress's functionality can be extended through plugins. There is a centralized repository for free and commercial plugins called the WordPress Plugin Directory, but the system also enables administrators to install plugins manually or add external repositories. Plugin directories are externally hosted software and will not be covered in this paper, except where they interface with the WordPress CMS platform.

5.2.2 Use Cases

The system models the CMS workflow[6]. Essentially all functions are related to supporting an organization's capability of writing, editing, and publishing content.

- **System Installation**
  - ✦ Users should be able to download, install, and run the system on their own devices.
  - ✦ Plug-ins can be easily installed from repositories or local files.
  - ✦ Users can create more than one site on a single installation.
  - ✦ The system can update itself once installed.

- **Content Management**
  - ✦ Users can use PHP as a building block to design a dynamically-generated webpage, to the individual's preference.[5]
  - ✦ Users can easily develop, install, and distribute site templates which modify the structures of their site webpages.
  - ✦ Users can easily develop, install, and distribute site themes which modify the appearance of webpages.
  - ✦ Additional site media can quickly and easily be added to a site.

[4] In WordPress terminology, "site network" or "network", is a single instance of the software which is capable of managing separate sites, but each 'site' within a network is otherwise conceptually self-contained.

[5] In the context of web interfaces.

- ✦ The quantity of site media should not affect the performance of the installation.
- ✦ Changes to each site installation should be easy and prompt.
- ✦ Users may self-register and log-in to sites.
- ✦ A user may identify themselves to other users on a site.
- ✦ Users can comment on pages.

- **Organizational Support**
  - ✦ The system supports a web-content-creation workflow.
  - ✦ The system is pre-configured with necessary permissions and roles for web-content-creation.
  - ✦ Site administrator permissions are separated from other roles.
  - ✦ Multi-site management permissions are separated other roles.
  - ✦ Administrators can add or modify user roles.

5.2.3 Additional Requirements

In addition, the system has some features which assist in the technical aspect of performing such tasks.

- **External Systems**
  - ✦ Database should house the website content, tags, comments, user data, configuration options, and plugin tables to enable easy search queries and immediate access of all stored data for all users throughout the world.
  - ✦ All user and website data are stored in a MySQL database with 1 GB of free storage and the availability of tiered storage options which users may select and alter at any time quickly with no loss of data.
  - ✦ Data can be quickly added, edited or deleted from storage; changes to the data should be immediately reflected on the server, the webpages and the users overall account.
  - ✦ Websites should be able to be easily imported to other hosted web services.
  - ✦ Site Health screen is available on the user dashboard that displays critical information and technical aspects of their site.

- **Security**
  - ✦ Security updates will be backwards compatible.
  - ✦ Security vulnerabilities can be identified and patched.
  - ✦ Automatic background updates are utilized to ensure all websites are kept safe and to ensure that users do not need to continuously monitor the health and safety of their websites around the clock.

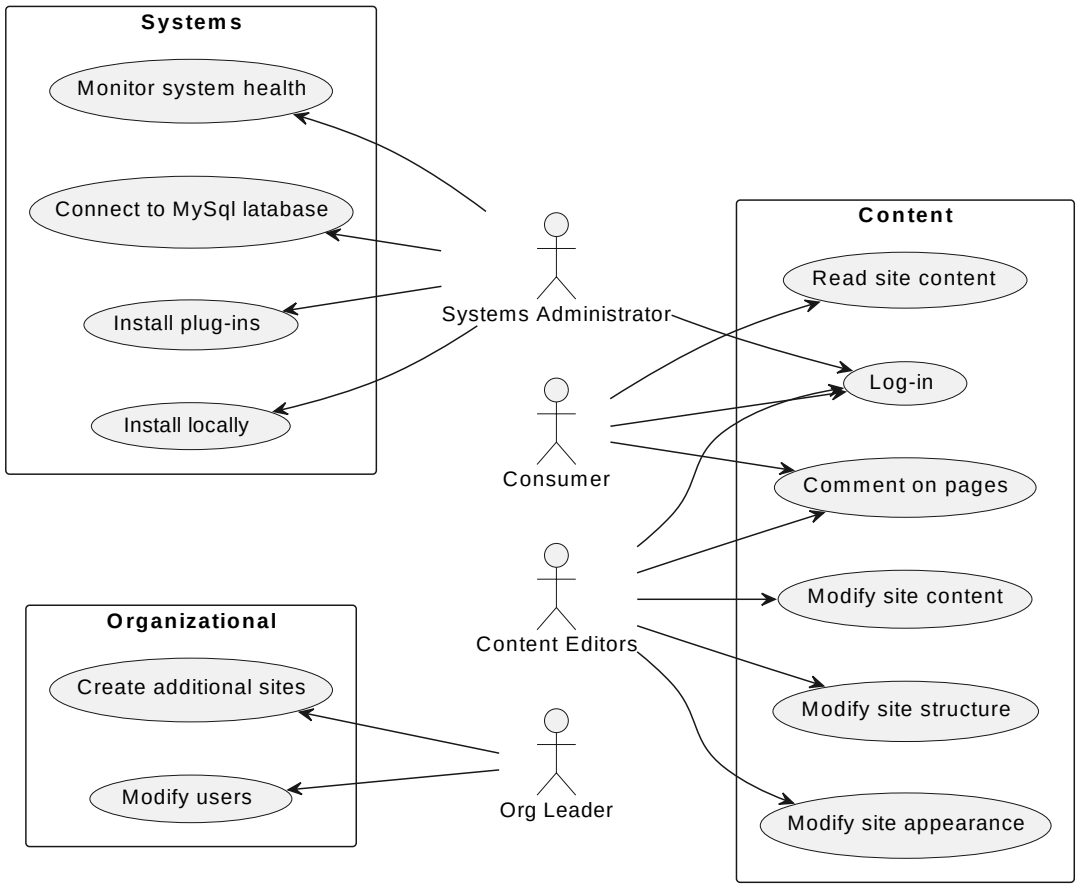

Figure 2: WordPress supports a large cross section of the most common needs of CMS builders and users.

6 System Qualities

6.1 Modifiability

By making their content generation pipeline based on dynamic elements - files located on the host and in the database, administrators of a WordPress install are a few uploaded files away from altering the way their content appears to users.

In terms of development, WordPress embraces the standard platforms of the web, having enabled an incredible quantity of themes and templates. If the default configuration isn't capable of the functions needed, plug-ins can extend the platform. The plug-in interface is event-oriented, a well-known pattern that allows developers to get started quickly.

The platform's built-in dashboard system makes it easy for administrators to modify arbitrary aspects of the install itself.

- **Tactics**
  Because of the wide range of sites, users, and administrators in the sytstem's userbase, WordPress employs a wide range of tactics to promote modifiability. The majority of modifiability capabilities are oriented towards plug-in developers, but because of the enormous number of open-source developers, we cannot neglect the internal design of the system here:

  ✦ **Module Cohesion**
  Internally, the structure of the system follows a highly module design. Each module is broken

up into fewer than approximately 15 internal components, with more complex modules being subdivided further. See Section 7.2.

- ✦ **Binding Deference**
  The plug-in subsystem categorizes bindings in twain: actions and filters[6]. Plug-ins can dynamically register for each based on strings at runtime. In this way, events are bound at-will by plug-ins, allowing them to have as much complexity as necessary, while also preserving system speed at a reasonable level.
- ✦ **Encapsulation**
  The system encapsulates extensible behavior into three categories:
  - ✶ **Templates**
    Strictly speaking, WordPress templates represent the per-page formatting using HTML[7]. Because they control the root file for the page a user is viewing in their browser, this gives nearly limitless control to template developers to modify content. Templates can be attached by types, tags, taxonomies, categories, and authors. Modifying a template does not necessitate modifying the actual content presented in the page.
    Templates are further composed of template blocks. See [2].
  - ✶ **Themes**
    Themes are, effectively, groups of template files, and can modify the style of an entire site. This enables users to modify their entire site with a single add-on. E.g.: a theme developer could package a number of templates, images, CSS files, and audio into a single theme and distribute it for use by site administrators.
    The default installation of WordPress includes a number of themes, named by year of development. See [8].
  - ✶ **Plug-Ins**
    Plug-ins represent the ultimate extension of the system. Using plug-ins, a developer can hook into arbitrary actions and events, and alter the incoming data (through filters) or execute arbitrary code based on events (actions) the system fires. Writing plug-ins requires knowledge of PHP, but installing existing plug-ins does not.

- **Scenario**
  To make a previously built theme easier for WordPress users to use, a WordPress plug-in developer updates it, the modification is done successfully and available for use less than two hours after the update with no downtime.

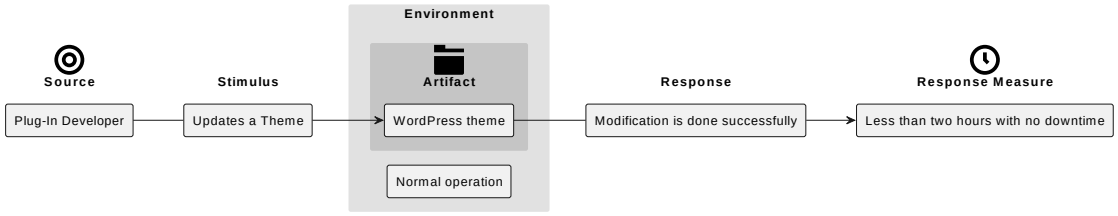

Figure 3: WordPress has a streamlined process to adding plug-ins.

- ✦ Source of stimulus: Plug-in Developer
- ✦ Stimulus: Updates a theme
- ✦ Artifact: WordPress theme
- ✦ Environment: Normal operation (after deployment)
- ✦ Response: Modification is done successfully
- ✦ Response measure: Less than two hours with no downtime

[6]This division of event types can also be viewed as a form of encapsulation.
[7]Colloquially, themes and templates are conflated, with the term "theme" essentially encapsulating both.

6.2 Usability

WordPress's tens of thousands of themes and plugins make it simple for users to create content. Administrators may simply format posts and pages, add media, and publish them to the web with a few clicks.

WordPress is also simple to set up and update. WordPress may be installed with just one click thanks to the many one-click installers that many web hosts provide. Alternately, if administrators are more technical, they can configure their server, build a database, upload WordPress through FTP, and then launch the installer.

Internationalization has also been key to the system's development. More than 70 different languages are supported by WordPress, making it simple for users to post content in their native tongue on the web.

- **Tactics**
  Generally, the usability tactics WordPress employs are centered around its use as a CMS. As the system evolved, it also incorporated tactics with special regards to systems administration:

  - ✦ **Maintain Task Model**
    To be used as a CMS, the system models the process of templating, creating, editing, and reading content. It does this by providing web pages for each task and providing access to each based on its user model.

    - ✶ **Creating and Editing Content**
      The system provides built-in WYSIWYG editors for content managers to easily create and edit pages. This contrasts with more complex unspecialized systems which perhaps involve editing text files, compiling, and uploading them to a server. In contrast, all WordPress asks of the user is to authenticate and access the appropriate page.
    - ✶ **Page Templating**
      By providing built-in templates, the meta-process of creating templates for users to percieve content in a consistent manner.

      The process of creating page templates is also modeled by the block subsystem. Here content areas with predictable HTML are made easily composable into existing pages. See [2]. In turn, the templating engine supports easily creating content by eliminating portions of the repetitive page creation process.
    - ✶ **Reading Content**
      The system supports users consuming the content of a site by supporting the generally accepted modern practice of accessing a website, and responding with content which web browsers are expected to be able to interpret.

      The system also supports a broad range of languages, enabling easy internationalization of content.

  - ✦ **Maintain User Model**
    The system models the structure of organizations requiring CMS capabilities in its RBAC system. See Section 5.2.2 and Appendix B.

  - ✦ **Maintain System Model**
    The system models each stage of the systematic process of page generation, for use by plug-in developers[3]. In this model, a developer can register for events (with so-called "hooks") and inject behavior determined by the arbitrary code in the plug-in. E.g.: A plug-in may hook into events that fire when a page is published. Default hooks are listed in [5], [4] and plug-ins can create new hooks.

- **Scenario**
  An administrator quickly alters the appearance of pages on the website using a new template that is uploaded and implemented within 2 seconds.

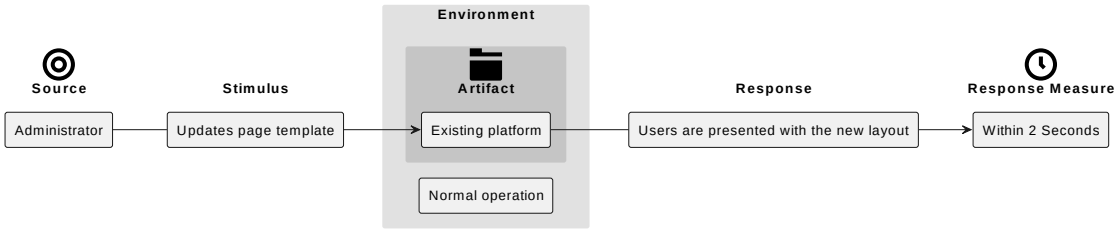

Figure 4: Support for the CMS workflow is a main draw of WordPress.

- ✦ Source of stimulus: Administrator
- ✦ Stimulus: Updates page template
- ✦ Artifact: Existing platform
- ✦ Environment: Normal operation
- ✦ Response: Users are presented with the new layout
- ✦ Response Measure: Within 2 seconds

6.3 Security

WordPress has role-based authorization control that a team can take advantage of if they do not all need the same level of access to the website for security reasons. The site is managed by administrators, content is handled by editors, written by authors and contributors, and profiles managed by subscribers through their profile.

The community developing WordPress is vigilant regarding security issues. An internal survey[9] of recent vulnerabilities suggests identified-to-patched time of about one month. Developers have a well-defined system for triaging and fixing vulnerabilities, as well as a "bug bounty" program.

- **Tactics**
  The system employs role-oriented security now typical among security-conscious internet sites:
  - ✦ **Authenticating and Authorizating Actors**
    The system provides a role-based authentication and authorization system, with typical CMS-oriented roles. See Section 5.2.2 and Appendix B.
  - ✦ **Limit Access**
    The system provides administrators to restrict access to important pages. This is especially important in WordPress due to the integrated nature of the security administration – if pages were not restricted, unauthenticated users may be able to access and modify important systems pages from public pages, since administrative panels themselves are part of the public-facing system. By default, unauthenticated users only have access to the pages which have been specifically published by appropriate users. The system also provides plug-in developers the ability to modify these patterns.
  - ✦ **Validate Input**
    XSS attacks are prevalent on public-facing websites which accept arbitrary content from anonymous internet users – a large portion of WordPress's site base. To prevent this, the system employs a system of so-called "filters". The filter subsystem is oriented around preventing malicious content from being writted to the database directly.

    Similar to the hooks subsystem, plug-ins can listen for named events listed in [5] and [4], and alter the content prior to the content's progression to the database. E.g.: When a comment is submitted, a plug-in would be notified of the content and enabled to escape any HTML or JavaScript in the message, which would then appear as text literals in subsequent displays of the comment message body.

- **Scenarios**
  A malicious user uploads JavaScript intended to be redistributed by a site to its readers in the body of a page comment. The system is configured with a community-developed plug-in leveraging the filters API, which identifies and removes JavaScript in comments. The plugin acts imperceptably for the task.

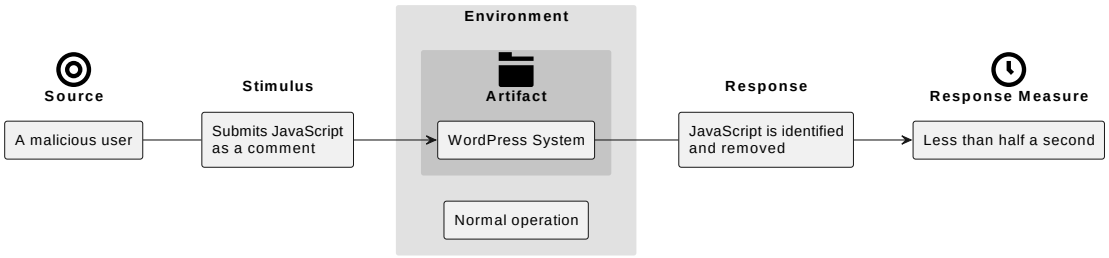

Figure 5: The hooks system supports a CMS-oriented data pipeline model.

- ✦ Source of stimulus: A malicious user
- ✦ Stimulus: Submits JavaScript as a comment
- ✦ Artifact: WordPress System
- ✦ Environment: Normal operation
- ✦ Response: The JavaScript is removed
- ✦ Response Measure: Less than half a second.

7 Architectural Views

7.1 Components & Connectors

The system interacts with four aspects of the host system:

- **Firewall/Reverse Proxy**
  The system exposes all functionality via port 80.
- **PHP**
  The system requires a PHP installation on the host node for runtime compilation and evaluation.
- **Filesystem**
  The system stores Templates, Themes, Plug-Ins, and the program executable files in a directory structure.
- **Database**
  The system expects a MySQL system to store various stateful elements of the installation.

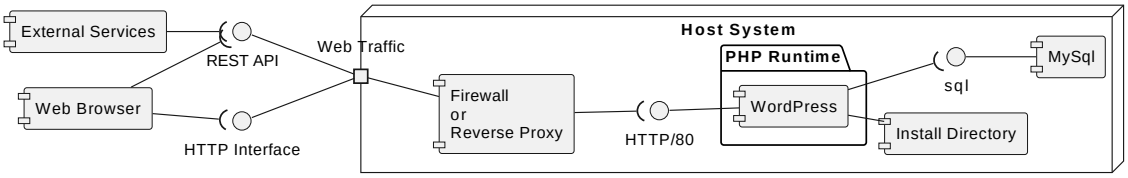

Figure 6: The WordPress system is fairly encapsulated from a systems administrator perspective.

7.2 Modules

Because of the large size of the code-base[10], this module listing is slightly abridged[8]. The views here are partitioned by top level for legibility.

A comprehensive listing of built-in themes can be found at [8]; a comprehensive listing of template blocks can be found at [2], respectively.

Figure 7: The top-level organization of the system.

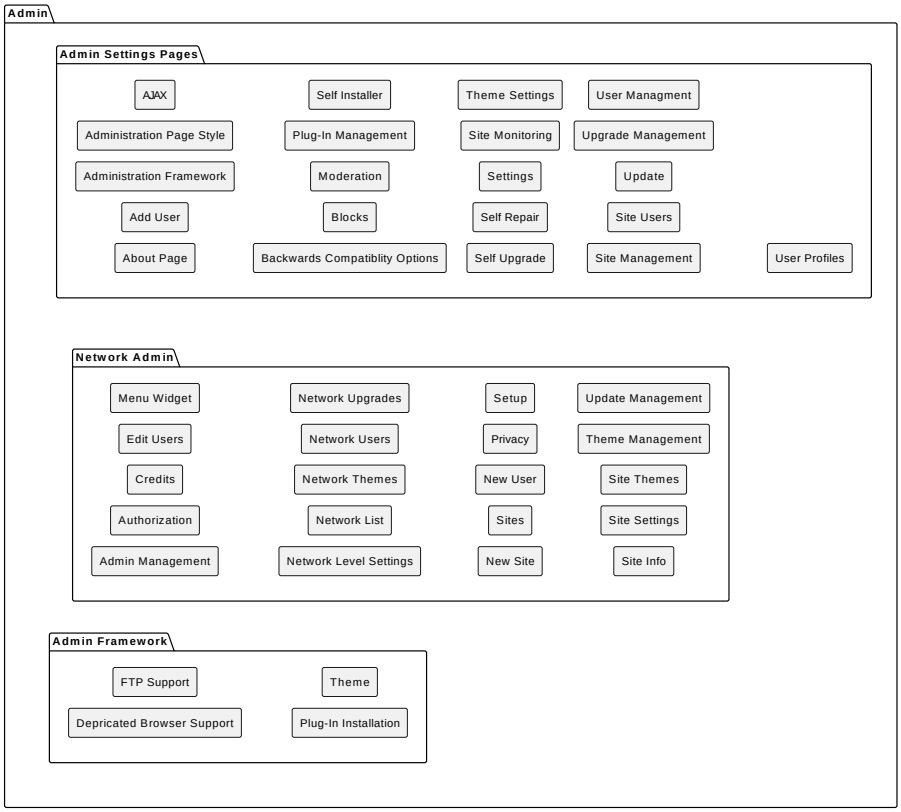

Figure 8: Administration functions are divided into HTTP-accessible page groups and the internal framework.

[8]Mainly, the modules seen here are composites of more, smaller, components. Additionally, this listing omits deprecated components.

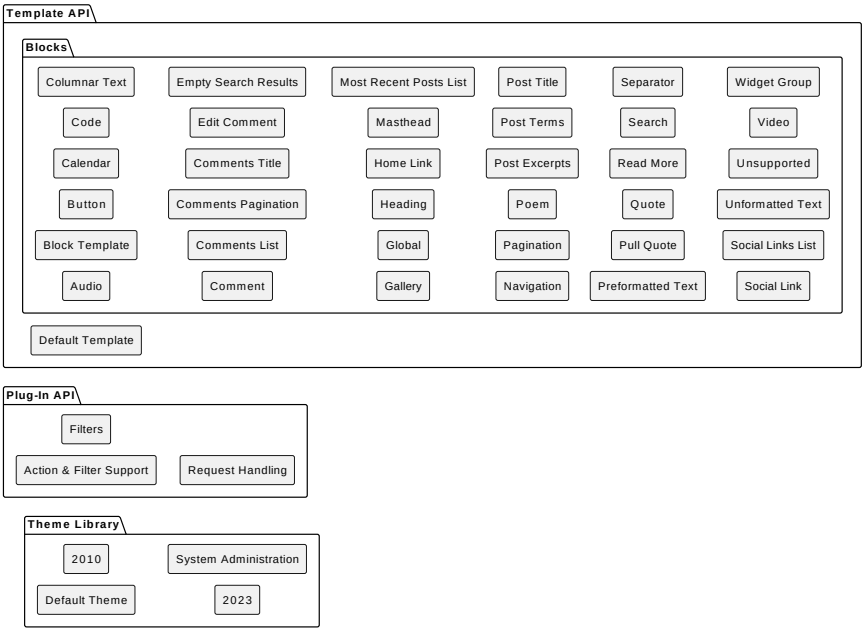

Figure 9: Customizing an install is supported in various forms.

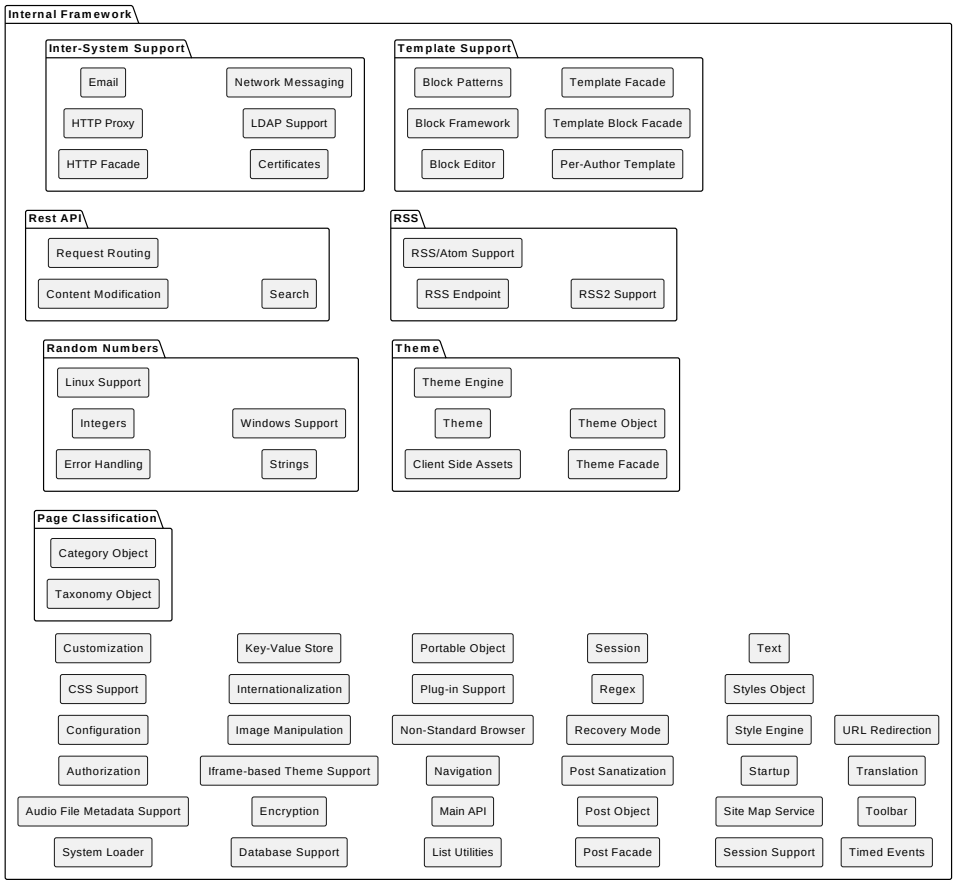

Figure 10: The internal framework is used throughout the system.

8 Architectural Analysis

8.1 Drivers

- **Design Purposes**
  - ✦ **Modeling a CMS**
    The system is designed to facilitate CMS workflows. Conforming to the CMS and CMS-administrative processes is a primary driver in the architecture of the system. Given this use case may not involve a highly technical population, one manifestation of this driver is the need to keep the core software relatively easy to use, which influences which features are *not* included as much as which are.
  - ✦ **Preserving Incumbency**
    WordPress's source code is open, and licensed under the GPL. This means that it is straightforward for developers with competing ideas to create a competing system[9]. Given that the system's initial release dates to 2003, it seems that the developers of the system have an interest in ensuring the system remains popular. The system-adjacent VPS and valuable branding are also manifestations of this driver.

    How does this manifest in the software specifically? One aspect is the plug-in interface: There is a fairly limited interface which requires detailed knowledge of the system, and the documentation on the process of integrating a plug-in is clear and straightforward. With a low barrier to entry for extending the system, the developers seem to have prioritized the popularity of the software.
- **Use Cases**
  See Section 5.2.2
- **Quality Attributes**
  See Section 6
- **Constraints**
  - ✦ **Resource Use Limitations**
    Devices serving WordPress may be severely constrained. While there is no official disclosure regarding the minimum specifications for RAM and CPU, some system-adjacent sites suggest a system with 512MB RAM and 1GHz CPU clock. The system has few reasons to refactor its codebase to take advantage of the many-cored processors of today's consumer- or datacenter-grade hardware.

    The system would also be restricted to a fewer number of users if this change took place. E.g.: One common 'homelab'-style deployment is to run the system on a single-board-computer. Increasing the resource requirements is effectively in conflict with its goal of preserving incumbency.
  - ✦ **Standards-Oriented**
    The system is open and leverages web standards to limit design work and improve compatibility. This helps the system eliminate certain design decisions, but certainly acts as a restriction in some ways. E.g.: The system utilizes REST and JSONP for frontend API calls. These standards are highly compatible, but recent industry focus has lead some to believe they introduce some inefficiency in some cases due to the serialization boundary.
  - ✦ **Modeling a CMS**
    As much as it is a purpose for the system to model the CMS workflow, it is also a constraint. There may be decisions which would benefit other drivers, but conflict with this need. E.g.: Many WordPress installations are operated as single operator on a single server. In an effort to facilitate this non-CMS-style workflow, the architects could choose to incorporate a built-in database, simplifying the current 3-layer model. This would simplify setup for less technical users in these groups.

[9] WordPress is itself a fork of an earlier system b2/cafelog.

8.2 Styles and Patterns

WordPress communicates with clients using standardized web technologies. This lends it to be used with various other modular components which can aid in some deficiencies in WordPress's otherwise monolithic style. It is possible to use WordPress within a network infrastructure of a scale which this analysis does not intend to describe.

Client/Server: Monolithic, Layered

In the simplest deployment, two components will execute all necessary functionality for the WordPress system: The PHP/WordPress runtime, and a MySQL schema. These two components are tightly coupled, and it is atypical for other systems to directly access a schema setup for a WordPress installation[10]. For these reasons, they are considered a single component in this model.

Components external to the WordPress system such as user browsers and API calls are expected to connect via HTTP. These are the system's clients, which consume the data served over its public interfaces. Clients are not configured to access the WordPress database directly.

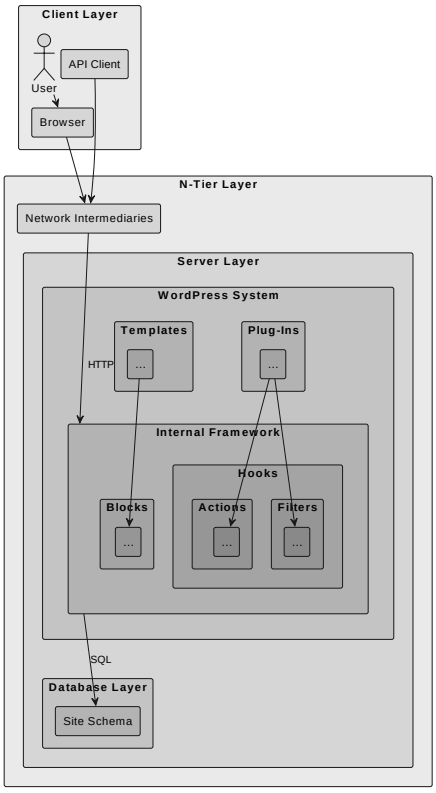

Figure 11: All client calls are mediated by the runtime.

Closer examination of the system reveals additional architectural patterns:

- **3/N-Tier**
  WordPress necessarily has a companion MySQL schema. In this way, the server architecture implements a 3-tier architecture.

  Because of the typically static nature of the content hosted on system sites, it is also common to place an in-memory cache for faster content access. Load balancing client-server requests is also possible. This layered but integrated approach may be classified by some as an N-tier architecture.

[10] E.g.: schema access is assumed to be exclusive to a single runtime at once.

- **Kernel/Plug-In**
  The system enables 3rd-party code to intermediate client requests within the runtime component.

- **Pipe/Filter**
  Plug-ins can subscribe to events as a page request is processed by the system, revealing an internal data pipeline.

8.3 Rationales

These architecture styles and patterns are designed to support a wide range of quality attributes, making it a popular choice for building websites and web applications. A few of the most enabled qualities include:

- **Modifiability**
  The Kernel/Plugin architecture allows developers to add or modify functionality without changing the core codebase. This makes it easy to extend and customize WordPress to meet specific needs. The layered architecture style promotes modifiability by providing a clear separation of concerns between the different layers. Changes to one layer typically do not affect the others, making it easier to modify and maintain the application.

- **Usability**
  The WordPress monolithic, layered architecture is designed to be user-friendly, with a simple and intuitive user interface that makes it easy for non-technical users to create and manage content. This architecture style is easy to understand and does not have all the complexities involved in a distributed system.

- **Interoperability**
  WordPress is built using the client-server architecture style that incorporates open standards and public APIs, which makes it easy to integrate with other systems and applications. Developers can create custom integrations that can improve the functionality and usability of the system

8.4 Alternative Architectures

WordPress is a CMS that uses a client-server architectural style. Requests are sent from a client to the server, which houses the WordPress application and database. After processing these requests, the server provides the client with the proper response.

- **Peer-to-Peer**
  The peer-to-peer (P2P) model is one alternative architectural style for creating web apps. There is no central server or authority in a P2P scheme. Instead, every node in the network functions as both a client and a server and is capable of direct communication with every other node. This makes it possible for more distributed and decentralized systems, which can be advantageous in circumstances where there is no central authority or where scaling is an issue. Despite the fact that the P2P model might be helpful in some situations, it is not well suited to the requirements of a CMS like WordPress. Here are a few reasons:

  - ✦ **Centralized control**
    With a single point of control for handling content, user accounts, and site settings, WordPress is intended to be a centralized system. For the system to remain secure and consistent, this is crucial Since there is no centralized authority in a P2P system, enforcing rules or standards may be challenging. Distributing content causes an issue modeling the CMS workflow because publishers expect to have control over the content.
  - ✦ **Scalability**
    The standards based approach WordPress employs allows a layered approach, lending to use with load balancers and memory caches. This enables horizontal scaling, using more

servers to deal with growing workloads or traffic. A P2P system would require coordination to distribute the process of page generation, adding complexity to the system.

- ✦ **Database Management**
  WordPress uses a single database schema to store and organize information. In a P2P system, data may be dispersed across numerous nodes with various database systems, but this would limit publisher's ability to control their content, or increase the complexity of the network.
- ✦ **Security**
  WordPress comes with built-in security tools like access controls and user authentication. Since there is no centralized body in charge of managing security in a P2P system, these are more challenging to execute.

In conclusion, the P2P model is not an appropriate architectural approach for creating a CMS like WordPress, despite the fact that it may have some advantages in specific situations. WordPress' client-server architecture is better suited to a CMS' centralized administration, scalability, database management, and security needs.

- **Services**
  Breaking up components of WordPress into services could mitigate certain aspects of the system, specifically relating to performance. In a multi-site installation, this is somewhat possible already: Administrators have the ability to dedicate separate database schemas to separate sites, which can be allocated on different MySQL servers. However, it is still the case in this implementation that updates must be done all at once: the system is still unitary.

  The typical use of load-balancers and in-memory caches which large-scale deployments employ may be interpreted as an application of this approach.

  Narrowing the focus to the WordPress system, there are two main reasons why it may not be preferential to implement this approach:

  - ✦ **Complexity**
    Breaking up the monolithic internal structure of the system into components would require substantial refactoring. Plug-Ins, represent a substantial barrier: Hooks must be registered centrally, so would need to be resolved somehow. Without involving complex voting protocols, this alone would still represent a single point of communication. Additionally, the state of the system can be queried at the time that a hook is activated, demanding complete synchronization between all nodes.

    It may be possible for templates to be distributed since their outputs should be idempotent. In practice, however, since they are capable of executing arbitrary PHP, they would probably encounter similar issues to plug-ins.
  - ✦ **Diminishing Returns of Scale**
    While performance on WordPress may not be superior to other platforms, most users do not encounter traffic sufficient to demand the scalability provided by SOA or microservice architectures.

  In conclusion, the monolithic model is "good enough" compared to a service-oriented architecture. The main reason this approach is unlikely to have advantages for system stakeholders is the diminishing returns to scale those stakeholders will perceive.

8.5 Challenges and Limitations

A three-tier architecture provides several benefits to the WordPress system, but it also comes with some limitations that can affect certain quality attributes. Developers and organizations need to consider these limitations when designing and implementing their systems to ensure they meet their requirements and objectives. Proper planning, design, and testing can help mitigate these limitations and ensure a successful implementation. A few drawbacks to consider include:

- **Performance**
  Performance limitations can occur due to the communication overhead between the layers. Network latency and message passing can add to the overall response time of the system, leading to slower performance.

- **Scalability**
  This type of architecture system can be challenging to scale horizontally, as each layer may need to be scaled separately. This can lead to complexity and increased costs, making it difficult to achieve high levels of scalability.

- **Security**
  Three-tier architectures can be vulnerable to security threats, as the communication between the layers can be intercepted or compromised. Ensuring the security of each layer can be challenging, and a vulnerability in one layer can potentially compromise the entire application.

- **Availability**
  There can be availability limitations due to the monolithic nature of the system. If one of the layers encounters a failure, the entire system may become unavailable, leading to downtime.

- **Maintainability**
  Maintaining a three-tier architecture can be challenging, especially if changes need to be made across multiple layers. Changes in one layer can potentially impact other layers, leading to unexpected errors and downtime.

A References

[1] WordPress Governance Authors. *Identifying the Stakeholders of WordPress*. Feb. 2023. URL: https://wpgovernance.com/research/identifying-stakeholders-of-wordpress.

[2] WordPress Official Documentation Authors. *Blocks List*. Jan. 2023. URL: https://wordpress.org/documentation/article/blocks-list/.

[3] WordPress Official Documentation Authors. *Introduction to Plugins*. Jan. 2023. URL: https://wordpress.org/documentation/article/introduction-to-plugins/.

[4] WordPress Official Documentation Authors. *Plugin API/Action Reference*. Jan. 2023. URL: https://codex.wordpress.org/Plugin_API/Action_Reference.

[5] WordPress Official Documentation Authors. *Plugin API/Filter Reference*. Jan. 2023. URL: https://codex.wordpress.org/Plugin_API/Filter_Reference.

[6] WordPress Official Documentation Authors. *Wordpress Features*. Jan. 2022. URL: https://wordpress.org/about/features/.

[7] WordPress Official Documentation Authors. *Wordpress Roles and Capabilities*. Jan. 2023. URL: https://wordpress.org/documentation/article/roles-and-capabilities/.

[8] WordPress Official Documentation Authors. *Work with Themes*. Jan. 2023. URL: https://wordpress.org/documentation/article/worik-with-themes/#default-themes.

[9] WordPress Bug Bounty Contributors. *WordPress Bug Bounty Program*. Feb. 2023. URL: https://hackerone.com/wordpress?type=team.

[10] WordPress Developers. *wordpress-develop*. Jan. 2023. URL: https://github.com/WordPress/wordpress-develop/tree/trunk/src.

[11] W3Techs. *Usage Statistics and Market Share of WordPress, January 2023*. Jan. 2023. URL: https://w3techs.com/technologies/details/cm-wordpress.

B User Roles

WordPress implements an hierarchical RBAC modeling a CMS workflow. The user-accessible functions of a WordPress system are broken down into roles[7], with each mostly having one essential function. Single users are given a role by an administrator or super-admin. The architecture's capabilities are broken out by permission level.

- **Super-Admin**
  Manage site networks. Super-admins can create sites and also have the ability to manage plugins, themes, and users across all sites in the site network installation. Super-Admin abilities are a superset of Admins.
- **Admin**
  Admins are site-level users responsible for managing plugins, themes, and other users at the site level. Admin abilities are a superset of Editors.
- **Editor**
  Editors are responsible for managing pages or comments written by other users. Editor abilities are a superset of Authors.
- **Author**
  Authors can create and manage their own posts. Author abilities are a superset of Contributors.
- **Contributor**
  Contributors are capable of creating content but are not allowed to publish it directly, necessitating interaction with an editor or author-enabled user. Contributor abilities are a superset of Subscribers.
- **Subscriber**
  Subscribers can edit a profile which is used when leaving comments.
- **Web User**
  Unauthenticated users can access content which has been published by Authors. This role is not explicitly enumerated by the default configuration.

Categorized by aspects identified in Section 5.2.2, this is a comprehensive list of all platform capabilities:

- **Administration**
  - ✦ **Subscribers**
    - ✶ Log in
    - ✶ Edit profile information
  - ✦ **Super-Admin**
    - ✶ Set up multi-site host
    - ✶ Manage administrators list by site
  - ✦ **(Site-level) Administrators**
    - ✶ Install site-level themes, templates, and plugins
    - ✶ Add and modify users
    - ✶ Backup and restore site data
- **Content Management**
  - ✦ **Web clients/Users**
    - ✶ Download and run website content

- ✶ Comment on a page

✦ **Editor**

- ✶ Manage posts
- ✶ Moderate site comments
- ✶ Manage site media
- ✶ Moderate site Comments

✦ **Author**

- ✶ Publish posts

✦ **Contributor**

- ✶ Create or edit private pages

✦ **Web User**

- ✶ Read Published Content

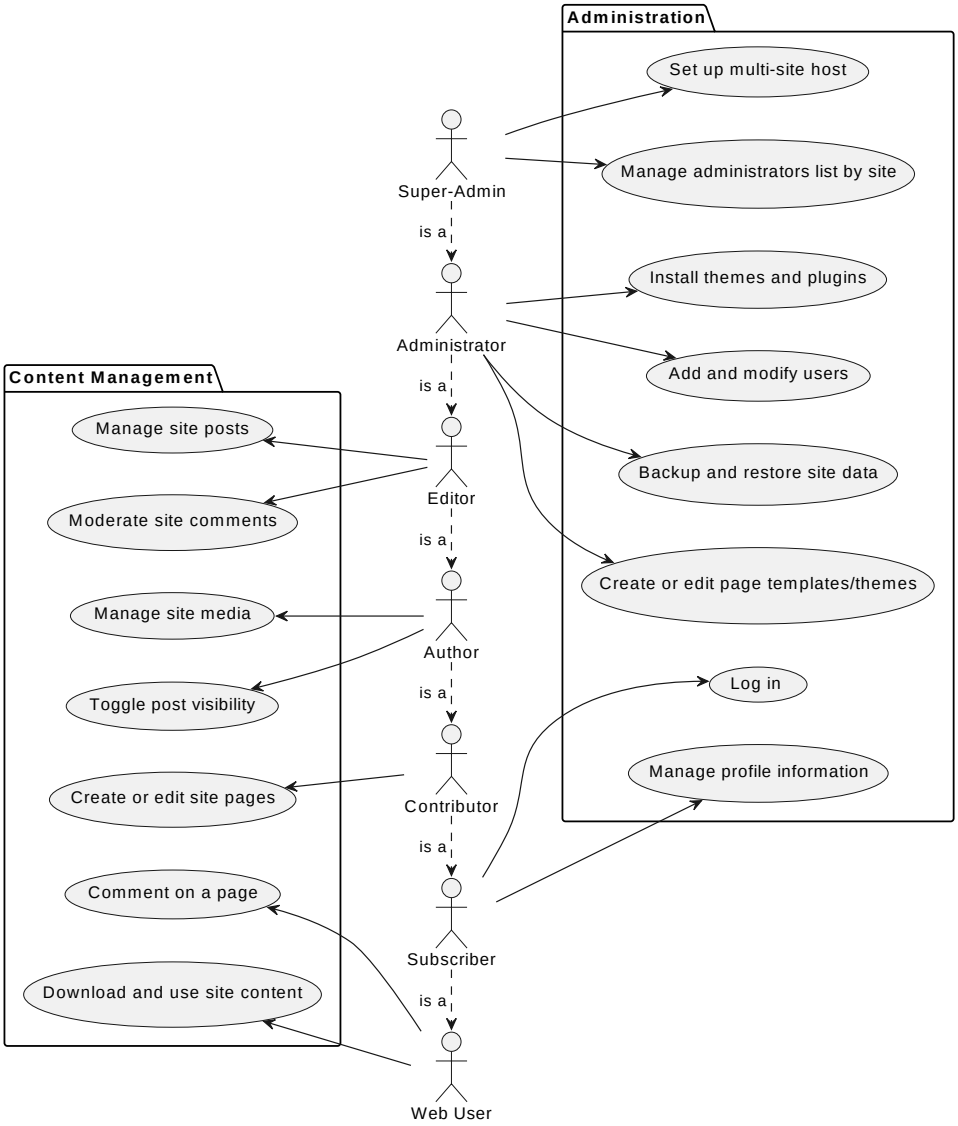

Figure 12: User roles are hierarchical in WordPress.