Report WP4-A4:

Platform code documentation

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency.

Co-funded by the European Union

Result

Report on platform code documentation

Related to

WP4-A1: Documentation of the platform programming code, creation of the user manual

Statement of originality

This report contains original unpublished work, except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both.

Disclaimer

This report contains material, which is the copyright of TET Consortium Parties. All TET Consortium Parties have agreed that the content of the report is licensed under a Creative Commons Attribution Non-Commercial Share Alike 4.0 International License. TET Consortium Parties does not warrant that the information contained in the Deliverable is capable of use, or that use of the information is free from risk and accept no liability for loss or damage suffered by any person or any entity using the information.

Copyright notice

Note

For anyone interested in having more information about the project, please see the website at: https://tet-erasmus.eu/

Document version history

Version	Date	Author	Comment
1.0	10.3.2025	Tomaž Požrl, UNILJ	Initial version
1.1	12.5.2025	Tomaž Požrl, UNILJ	Extended version
2.0	10.6.2025	Tomaž Požrl, UNILJ	Final version

Table of contents

1. Introduction

The Evolving Textbook (TET) aims to develop an innovative online platform for higher education textbooks that evolves through time with active participation from students. This project leverages advanced information and communication technologies (ICT) to create a dynamic and interactive educational resource. The TET platform will support content creators, teachers, and learners by providing tools for content creation, structuring, connectivity with other resources, and storage. It will facilitate easy browsing, presentation, and sharing of relevant educational materials in both classrooms and online or blended learning environments. Furthermore, the platform will include a recommender system to suggest pertinent content, enhancing the learning experience.

The University of Ljubljana (UNILJ) is the leading organization and coordinator for this work package.

This document provides comprehensive documentation for the programming code of the TET platform as adapted and customized for higher education study material storage. It aims to assist technical and programming staff in understanding the implementation, architecture, and integration of the system. This documentation is important for ensuring maintainability, facilitating troubleshooting, and supporting future development or enhancements of the platform.

1.1 Target audience

This documentation is intended for technical and programming staff responsible for the development, deployment, and maintenance of the TET platform. This includes:

- **Developers:** For understanding and modifying the codebase or extending features**.**
- **System administrators:** For deploying, configuring, and managing the platform in production**.**
- **Technical managers**: For overseeing implementation and aligning development with project goals.

It assumes familiarity with programming concepts, web development technologies, and the Laravel framework.

1.2 Overview of the TET platform

TET platform is built upon BookStack, an open-source, self-hosted knowledge management system designed for organizing content in a simple, hierarchical format. Its intuitive interface makes it well-suited for non-technical users, while its robust backend architecture supports scalability and extensibility. In this project, BookStack has been adapted to function as a digital repository for higher education study materials, providing a collaborative and co-creation environment for teaching staff and students.

Key features of the TET platform:

- **Hierarchical content organization**: Content is structured into Books, Chapters, and Pages, mimicking a traditional textbook layout.
- **Role-based permissions**: Fine-grained control over who can view, edit, or manage content.
- **Rich text editing**: A WYSIWYG editor for easy content creation and formatting.
- **Search functionality**: Allows users to quickly locate content, even in large repositories.
- **Version control**: Tracks changes to content, enabling rollbacks if needed.
- **Customizable interface**: Adapts to institutional branding and needs.

1.3 Purpose of the platform in this project

In the context of this project, the TET platform serves as a scalable and user-friendly solution for organizing and sharing educational materials. Its primary objectives include:

- **Centralized content management**: Providing a unified location for storing study materials, reducing duplication and improving access.
- **Collaboration**: Allowing educators to co-author and manage content collaboratively.
- **Ease of use**: Ensuring that both technical and non-technical users can efficiently interact with the platform.
- Secure access: Protecting sensitive data with role-based permissions and secure authentication mechanisms.

By leveraging BookStack, the TET platform fosters a seamless flow of knowledge between teaching staff and students, while its modular design ensures adaptability for future requirements.

In this project, the platform serves as a dedicated digital repository tailored for the **"Evolving Textbook"** model, advancing far beyond standard BookStack functionality. Building on BookStack's robust foundation, the TET platform introduces key adaptations to support the requirements of higher education and collaborative textbook development. These

enhancements specifically address the dynamic, co-created nature of modern educational materials, ensuring that the platform not only stores content but actively supports its evolution over time.

Key customizations and unique features developed for the TET platform include:

- **Extended user roles and granular permission models** specifically designed to reflect the diverse responsibilities of project partners, educators, and students involved in collaborative authoring, peer review, and content moderation.
- **Custom workflows for content review and approval**, allowing teaching staff to manage the lifecycle of chapters and pages from draft to published state.
- **Enhanced content analytics and activity tracking**, enabling the monitoring of contributions, usage patterns, and collaborative engagement across multiple institutions.
- **Automated versioning and change-logging** tailored for the textbook development process, supporting transparent co-creation and easy rollback to previous states.
- **Integration with project-specific repositories and data sources**, streamlining the import and export of reference materials and datasets used in the evolving textbook.
- **Localization and theming features** reflecting project branding, multi-language support, and institution-specific requirements.
- **Custom modules and API extensions** to facilitate interoperability with other TET project outputs, including learning analytics dashboards and MOOC platforms.

The platform has been customized to meet the following objectives:

- **Streamline the collaborative creation, storage, and ongoing revision** of educational materials in a structured, textbook-oriented format.
- **Support project partners and teaching staff** with tools for content quality assurance, peer review, and attribution.
- **Empower students to actively participate in content enrichment**, annotation, and feedback, making the textbook a living document.
- **Facilitate transparent tracking of changes and contributions** for project reporting and research purposes.
- **Ensure compatibility with project workflows** and external systems, supporting sustainable adoption across diverse educational contexts.

These targeted enhancements make the TET platform a true enabler of collaborative, high-quality digital textbook development—aligned with the goals and innovative spirit of the TET project.

1.4 Tools and technologies used

The implementation and customization of the BookStack platform rely on the following tools and technologies:

- **Backend**:
  - Laravel framework: A PHP-based framework that powers the platform's core logic,
  - PHP: Provides the necessary environment for Laravel to run efficiently.
- **Frontend**:
  - Blade / Typescript: Enables dynamic, interactive components in the user interface,
  - Bootstrap: Ensures a responsive and accessible design across devices.
- **Database**:
  - MySQL: Used for persistent storage of content, user data, and metadata.
- **Server**:
  - Apache as the web server.
- **Operating system**:
  - Ubuntu 22.04.
- **Hosting**:
  - Cloud-based infrastructure (DigitalOcean).
- **Version control**:
  - Github for managing source code.

1.5 Structure of the documentation

This documentation is organized to provide a clear and thorough understanding of the platform's codebase and functionality. The chapters are structured as follows:

- **System architecture**: Describes the high-level architecture of the platform and its components.
- **Codebase structure**: Provides an overview of the directory structure and naming conventions.
- **Modules overview**: Details the core modules and their interactions.
- **Code implementation Details**: Explains how the core functionalities are implemented.
- **Interfaces and integration**: Describes APIs, third-party integrations, and communication protocols.

- **Development and deployment**: Provides guidance on setting up the development environment and deploying the platform.
- **Testing**: Covers testing strategies and examples.
- **Future improvements**: Suggests potential enhancements for the platform.
- **Glossary**: Defines core technical terms.
- **Appendices**: Code snippets and platform configuration files.

This chapter establishes the foundation for understanding the rest of the documentation. Each subsequent section builds upon this introduction to provide a comprehensive view of the platform's programming and development.

2. System architecture

The system architecture of the TET platform has been designed to ensure scalability, maintainability, and usability, particularly as it has been customized for use in higher education. This chapter provides an overview of the system's high-level architecture, explains the key components, and details the dependencies required to run the platform effectively.

2.1 High-level system architecture diagram

The TET platform follows a standard multi-layered architecture, ensuring a clear separation of concerns. Below is an overview of how the layers interact (Fig. 1).

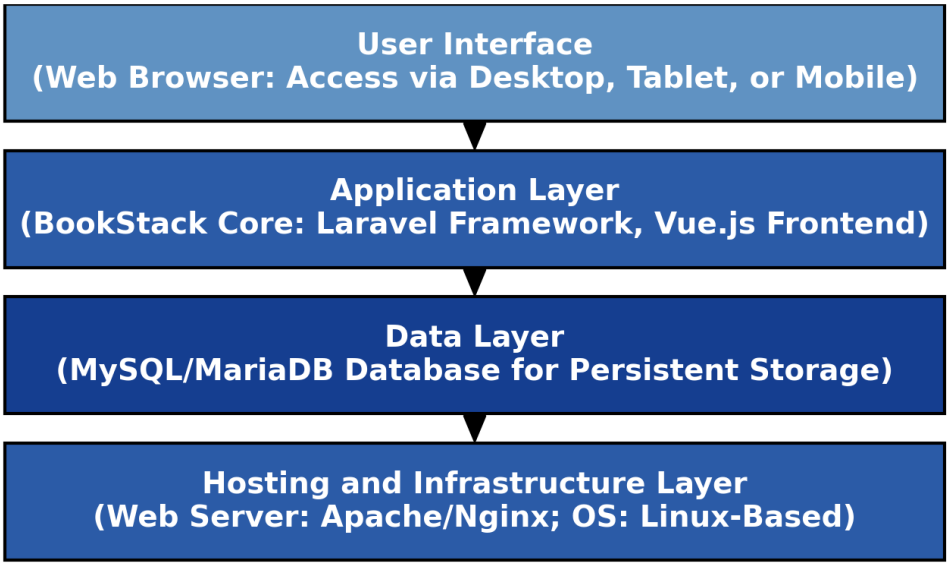

Figure 1: TET platform high-level system architecture

This architecture ensures smooth user interactions at the top layer while efficiently handling requests, processing content, and storing data at the lower layers. The infrastructure layer provides the foundation for hosting and securing the system.

2.2 Key components

User interface (UI)

The user interface serves as the primary entry point for all users—students, educators, and administrators. It is built using plain Blade/Typescript for dynamic, reactive components

alongside Laravel Blade templates for server-rendered views, ensuring a seamless and responsive experience across all devices.

Key features include:

- **Hierarchical Navigation**: The content is organized in a textbook-like structure (`Books > Chapters > Pages`) that allows users to navigate intuitively through complex educational materials.
- **Advanced Search & Rich Content Editing**: Integrated advanced search capabilities enable users to quickly locate content. A robust WYSIWYG editor supports rich text formatting, making it straightforward for educators to create and update materials.
- **Customization and Localization**: The interface supports customizable themes and branding options, allowing institutions to align the platform's appearance with their identity. It also includes localization features to ensure accessibility for a diverse user base.
- **Mobile Optimization**: The UI is optimized for smaller screens, providing a consistent and seamless experience on smartphones and tablets. Interactive components ensure that changes are rendered in real time without full page reloads.
- **User-Centric Design**: Designed with simplicity in mind, the interface minimizes the learning curve for non-technical users. Its clean and intuitive layout enables effortless navigation and interaction with the platform.

Application layer

The application logic layer is the core of the system, built on the Laravel framework to ensure scalability, security, and maintainability. It processes the platform's business logic by handling critical tasks such as user authentication, role-based permissions, content management, and API integrations. The layer is organized into several dedicated modules within the /app directory, each responsible for specific functionalities:

- **Access module**: Manages authentication and authorization, including services for registration, login, password resets, multi-factor authentication (MFA), and external authentication mechanisms (e.g., OpenID Connect). Controllers like `RegistrationController`, `LoginController`, and `ForgotPasswordController` work in tandem with services such as `RegistrationService` and `LoginService` to secure user access.

- **Activity module**: Tracks user interactions, notifications, and system events. Controllers and models within this module handle activities such as logging, favourites, comments, and webhooks, ensuring that user actions are recorded and managed effectively.
- **API module**: Facilitates RESTful API interactions, allowing external applications to interact with the platform. This module includes dedicated controllers and middleware for secure, token-based API access, ensuring consistent and reliable integration with third-party systems.
- **Core App module**: Contains the foundational application logic, including controllers (e.g., `HomeController`), service providers, and utilities that initialize and maintain core settings. This module underpins the overall functionality and configuration of the platform.
- **Additional modules**: Other modules, such as User Management, Content Management, and System Administration, are similarly organized. They follow a modular design where:
    - **Controllers** handle incoming requests and delegate operations to the appropriate services.
    - **Models** manage database interactions, ensuring data consistency and integrity.
    - **Services** encapsulate reusable business logic (e.g., search indexing, file uploads) to promote code reusability and easier maintenance.

The modular organization of the application layer ensures that business logic is clearly separated and encapsulated, enabling efficient development and easier future enhancements. Moreover, this structure supports strict access control, ensuring that only authorized users can interact with sensitive content based on their assigned roles and permissions.

Data layer

The data layer is responsible for managing the storage and retrieval of all content, metadata, and user-related data. It relies on MySQL or MariaDB to provide a robust and scalable relational database system tailored to the platform's needs.

Key aspects include:

- **Hierarchical content structure:** The database schema is designed to mirror the platform's organization of educational materials. Core entities such as Books, Chapters, and Pages are stored in relational tables with clearly defined relationships, enabling efficient querying and retrieval.
- **User management and access control:** Dedicated tables manage user accounts, roles, and permissions, enforcing security at the database level. Many-to-many relationships, facilitated by pivot tables, allow flexible role assignments, ensuring that only authorized users access sensitive content.
- **Search indexing:** Full-text search indexing is implemented to enhance performance, particularly in large repositories. This indexing supports fast and accurate content discovery across extensive datasets.
- **Database migrations:** Laravel-managed migrations ensure that schema changes are seamlessly applied across development, staging, and production environments, maintaining consistency and easing future updates.
- **Integration with Laravel Eloquent:** The data layer integrates with Laravel's Eloquent ORM, providing an intuitive and consistent interface for data manipulation, which simplifies complex queries and reinforces data integrity.

This structured approach ensures that the data layer is not only scalable and secure but also adaptable to future enhancements.

Hosting and infrastructure layer

The hosting and infrastructure layer lays the foundation for deploying and running the TET platform in a secure, scalable, and efficient manner. It supports both on-premises and cloud-hosted setups, adapting to the specific needs of different institutions.

Key components include:

- **Web server:** The platform is served using Apache or Nginx, with `mod_rewrite` enabled to handle URL routing efficiently. These web servers are optimized for handling concurrent requests and for the fast delivery of static assets.
- **Operating system:** Linux-based systems - particularly Ubuntu or Debian - are recommended due to their proven stability, performance, and extensive community support.

- **SSL/TLS security:** Secure communication is ensured by using SSL/TLS certificates. This is particularly important for deployments that involve sensitive data, such as student records or institutional information.
- **Containerization and virtualization:** For enhanced scalability and simplified deployment, Docker containers are employed to isolate and manage the application environment. Tools like Docker Compose facilitate orchestration and seamless updates.
- **Cloud hosting options:** The platform can be deployed on cloud services such as AWS, DigitalOcean, or Azure. These cloud-based hosting options offer additional scalability and resilience, making it easier to meet varying performance and availability requirements.
- **Security best practices:** This layer incorporates robust security measures including firewalls, strict access controls, and encrypted connections to protect the platform from unauthorized access and potential vulnerabilities.

This structured approach ensures that the TET platform can be deployed in a variety of environments while maintaining optimal performance, security, and scalability.

2.3 System dependencies

The TET platform relies on various dependencies to ensure its smooth operation. These dependencies are carefully chosen to align with Laravel's ecosystem and modern web development standards.

Backend dependencies

- **PHP**: platform requires PHP 8.0 or higher to support the latest Laravel features.
- **Composer**: Used to manage PHP dependencies, ensuring all required libraries are up-to-date.
- **Laravel framework**: The backbone of the application logic, providing MVC architecture, routing, and database management.

Frontend dependencies

- **Node.js and npm**: Required for building and managing JavaScript/TypeScript components and other frontend assets.
- **Bootstrap**: CSS framework that ensures a consistent and responsive design across devices.

Database dependencies

- **MySQL or MariaDB**: These relational database management systems store all content, metadata, and user data.

Server dependencies

- **Web server**: Apache (with `mod_rewrite`) or Nginx: Configured to serve the TET platform efficiently.
- **Operating system**: Linux (Ubuntu 20.04 LTS or higher recommended): Recommended for its stability, performance, and compatibility with the platform.

Optional dependencies

- **Redis**: Can be used to cache frequently accessed data, improving the platform's performance for large-scale deployments.
- **Docker**: Enables containerized deployments, simplifying scaling and updates.

By understanding the system architecture and dependencies, developers and administrators can effectively deploy, maintain, and scale the TET platform. This structure ensures that the platform can handle the needs of higher education while remaining flexible for future enhancements.

3. Codebase structure

A clear and well-organized codebase is essential for maintaining, extending, and debugging the platform. This chapter provides a detailed explanation of the directory and file structure, naming conventions, and the logical organization of modules in the TET platform.

3.1 Directory and file structure

The TET platform codebase is organized into a hierarchical directory structure to separate concerns, improve maintainability, and make it easier for developers to locate files and modules. Below is an overview of the directories and their purposes:

- **Top-level files:**
  - **Environment files**: Centralized configuration settings for various environments - `.env`, `.env.bak`, `.env.example`.
  - **Git and version control**: Files used for version control and metadata.
  - **Licence and versioning files**
  - **Command-line tools**: Laravel and platform CLI
  - **Docker & Composer**: Used for containerized deployment and PHP dependency management.
  - **JavaScript settings**: Manages frontend dependencies and build processed.
  - **PHP tools**: Used for coding standards, static analysis, and testing configuration.
- **`/app`:**

  Contains the core application logic, organized into several modules:
  - **Access**: Authentication, user access control, MFA, external authentication (OIDC, SAML2), and notifications.
  - **Activity**: User activity tracking, notifications, comments, favourites, and logs.
  - **Api**: Handles API requests, authentication, and documentation for external integrations.

- **App**: Core logic including the base model, `HomeController`, and application initialization.
- **Config**: System configuration files for caching, mail, storage, and logging.
- **Console**: Artisan commands for administrative tasks.
- **Entities**: Management of the core content structure (Books, Chapters, Pages), including controllers, models, queries, and export tools.
- **Exceptions**: Defines custom exception classes and centralizes error-handling logic, ensuring consistent responses and easy debugging.
- **Facades**: Provides Laravel-style "static" interfaces to underlying services (e.g. Cache, Notification), making calls more expressive and testable.
- **Http**: Houses HTTP-related classes:
  - Controllers for routing requests to actions
  - Middleware for request filtering (e.g. auth, throttle)
  - Requests for input validation and authorization
  - Resources for transforming models into JSON
- **Permissions**: Encapsulates all permission-checking logic, defining abilities, policies, and role mappings to enforce fine-grained access control.
- **References**: Manages in-text linking and embedded attachments (e.g. images, files), resolving `![alt](ref)` syntax to actual storage URLs or page anchors.
- **Search**: Integrates and configures search engines; handles indexing, query parsing, and relevance scoring for Books, Chapters, and Pages.
- **Settings**: Defines the storage and retrieval of system-wide settings (in database or config), with helpers for defaults, caching, and runtime overrides.
- **Theming**: Allows custom look & feel through theme discovery, asset publishing (CSS/JS), and view overrides without touching core views.
- **Translation**: Loads and manages localization files, handles language selection per-user, and provides helpers for translating UI strings and content.

- **Uploads**: Contains logic for handling file uploads: request processing, storage (local/S3), validation, and cleanup of orphaned files.
- **Users**: Manages user entities beyond authentication: profile settings, avatars, preferences, and user-specific content (e.g. recently viewed).
- **Util**: A grab-bag of reusable helper classes and static utilities (e.g. date formatting, URL builders, array helpers) to dry up code across modules.

Each submodule focuses on a specific domain like error handling, HTTP interactions, role-based permissions, cross-linking content, search functionality, system settings, theme customization, localization, file uploads, user management, and utility functions.

- **`/bootstrap`:**

  Contains bootstrapping scripts and caching configurations to speed up application start up.

- **`/database`:**

  Holds database migrations, factories (for test data generation), and seeders to ensure schema changes are trackable and easily deployable.

- **`/dev`:**

  Provides development resources including:

  - **api**: API request/response templates.
  - **build**: Build scripts for compiling JS and CSS assets.
  - **docker**: Docker deployment files.
  - **docs**: Documentation for development workflows.

- **`/lang`:**

  Contains language files for interface translations, organized into categories (e.g., common, editor, errors, pagination, preferences).

- **`/public`:**

  The web-accessible directory, containing:

- **Web entry files**: `index.php`, `.htaccess`, `web.config`
- **Static assets**: Images (app logos, default avatars, book covers, loading gifs), icons, minified JS/CSS in the dist folder, third-party libraries in libs, and a dedicated uploads folder for user content.

- **`/resources`:**

  Houses frontend assets including:

  - **Icons**: Platform-specific images.
  - **js**: JavaScript code for handling frontend behaviour (code editors, UI components, markdown, WYSIWYG editors, utility scripts).
  - **sass**: CSS styling files.
  - **views**: Laravel Blade templates used to generate HTML pages.

- **`/routes`:**

  Defines the platform's API routes (`api.php`) and web routes (`web.php`), mapping URLs to controllers and actions.

- **`/storage`:**

  Stores logs, cached data, backups, custom fonts, framework cache files, and user uploads. Organized into subdirectories such as **app**, **backups**, **clockworck**, **fonts**, **framework**, **logs**, and **uploads**.

- **`/themes`:**

  Directory for custom themes and branding files.

- **`/tests`:**

  Contains automated tests written using PHPUnit for unit, feature, and integration testing.

- **`/vendor`:**

  Includes third-party libraries and dependencies installed via **Composer**. Direct modifications are discouraged to maintain compatibility with future updates.

This structured organization ensures that developers can easily navigate and work with the codebase, minimizing the risk of errors and duplication.

3.2 Naming conventions

To maintain consistency and readability throughout the codebase, the following naming conventions are adhered to:

- **Files and Classes:**
  File names and class names use PascalCase. For example, a controller for managing books is named `BookController.php`.
- **Variables:**
  Variables are written in camelCase to differentiate them from classes and constants. For instance, `$bookTitle` or `$userRole`.
- **Constants:**
  Constants use uppercase letters with underscores for separation, such as `DEFAULT_PAGE_LIMIT`.
- **Routes:**
  Route names follow a kebab-case convention for URL readability, e.g., `/create-book` or `/manage-users`.
- **Database Tables and Columns:**
  Table names use `snake_case` to align with Laravel's Eloquent ORM conventions, e.g., `user_roles` or `page_content`.

By following these conventions, the codebase remains uniform and easier to understand for both new and experienced developers working on the platform.

3.3 Organization of modules

The platform's functionality is modularized to isolate concerns, promote reusability, and facilitate future enhancements. Each module corresponds to a specific feature or domain within the platform. Below is a brief explanation of all the modules and how they are organized:

- **Authentication and user access module**
  Manages user login, registration, authentication, and security features.

- **Content management module**
  Handles the creation, editing, and organization of Books, Chapters, and Pages.
- **User management module**
  Manages users, roles, and permissions.
- **Search & indexing module**
  Manages search functionality across books, chapters, pages, and users.
- **API & external integrations module**
  Exposes API endpoints for third-party applications to interact with BookStack.
- **Theming & customization module**
  Handles UI customization, theming, and HTML injection.
- **Localization & translation module**
  Manages language support and translations.
- **Notification & activity tracking module**
  Logs user actions, sends notifications, and manages webhooks.
- **System configurations & settings module**
  Handles system-wide settings, caching, logging, and storage.
- **Backend administration module**
  Manages system commands, maintenance tasks, and CLI utilities.
- **Database management module**
  Manages database migrations, data population, and backups.
- **Frontend module**
  Manages the user interface, frontend behaviour, and styles.
- **Testing & development module**
  Handles automated testing, API development, and build scripts.

By modularizing the codebase in this way, developers can focus on specific parts of the platform without affecting others, making the system easier to maintain and extend. This chapter outlines the codebase's structure, conventions, and modular design, providing developers with a solid foundation for navigating and working with the platform. The structure

ensures that the platform remains organized, maintainable, and scalable for future development efforts.

4. Modules overview

The TET platform is composed of various modules, each designed to handle a specific aspect of its functionality. These modules work in harmony to provide a robust, scalable, and user-friendly system for managing educational content. This chapter provides an overview of the modules developed, their purposes, and how they interact to form a cohesive platform. Each module has been designed to address a specific requirement while ensuring smooth integration with other modules.

4.1 Authentication and user access module

The **Authentication & User Access module** is a core component of the platform, responsible for managing user authentication, authorization, and access control. It ensures that users can securely log in, register, and interact with the system while maintaining strict security policies and permissions. By leveraging Laravel's built-in authentication framework, the module integrates advanced security features such as role-based access control (RBAC), session management, and multi-factor authentication (MFA) to protect user data and system integrity.

Key features

1. User authentication & session management

- Supports user **login**, **registration**, and **logout** with secure session handling.
- Users are authenticated using **hashed passwords** stored in the database.
- Laravel's built-in authentication guards (such as **session guard**) ensure users remain logged in securely.
- Session expiration and timeout mechanisms help prevent unauthorized access.

2. Role-based access control (RBAC)

- Implements a **hierarchical role-based permission system**, restricting access based on assigned user roles.
- The primary roles include:
  - **Admin** – Full control over user management, settings, and content.
  - **Teacher** – Can create, edit, and delete content but cannot manage users.
  - **Student** – Can create, edit, and delete their own content.

- **Viewer** – Read-only access, ensuring no unauthorized modifications.
- Permissions are **stored in the database** and validated through **Laravel's Gate and Policy classes**, enabling dynamic access control.

3. Secure password hashing & authentication

- Uses **bcrypt hashing** to store passwords securely.
- The **login process** compares the entered password with the hashed version stored in the database.
- Implements **password reset functionality**, allowing users to recover their accounts securely.
- **Brute-force protection mechanisms** (such as rate limiting and login attempt restrictions) help prevent unauthorized access.

4. Multi-factor authentication (MFA)

- Adds an extra layer of security by requiring users to verify their identity with:
  - **Time-based One-Time Password (TOTP)**
  - **Email-based authentication codes**
  - **Third-party authentication apps (Google Authenticator, Authy, etc.)**
- MFA is managed via the `MfaController` and `MfaTotpController`, ensuring smooth integration into the login process.

5. External authentication support (OAuth, LDAP, SAML, OIDC)

- Users can log in using external authentication providers:
  - **LDAP authentication** for enterprise environments.
  - **SAML & OpenID Connect (OIDC)** for single sign-on (SSO) support.
  - **Social authentication** (Google, GitHub, etc.) via OAuth.
- Managed through the `LdapService`, `Saml2Controller`, and `OidcController`.

6. User invitation & account management

- **Admins can invite new users** via email using the `UserInviteService`.
- Supports **self-registration** with email confirmation using the `EmailConfirmationService`.
- Users can update their profiles and reset passwords securely.

7. Access control middleware

- Laravel **middleware** ensures only authorized users can access certain routes.
- Example middleware includes:
  - `auth` → Ensures users are authenticated before accessing protected routes.
  - `role:admin` → Restricts access to admin-only pages.
  - `throttle` → Prevents excessive login attempts to mitigate brute-force attacks.

How the module works

1. Authentication flow

1.1 User registration/Login

- User submits credentials (email + password).
- The system validates credentials using `LoginService`.
- If valid, Laravel's **session-based authentication** grants access.
- If MFA is enabled, a secondary authentication step is required.

1.2 Session management

- Upon successful login, a session is created.
- Sessions are stored in `/storage/framework/sessions`.
- Expired sessions are automatically cleared.

1.3 Logout

- The session is destroyed upon logout.
- The user is redirected to the login page.

2. Authorization & Access control

2.1 Role-based permissions

- Permissions are stored in the `EntityPermission` and `RolePermission` models.
- Admins assign roles via the `PermissionController`.
- Laravel's Gate policies check permissions before granting access.

2.2 Middleware enforcement

- Protected routes are guarded with middleware:

```
    Route::middleware(['auth', 'role:admin'])->group(function
() {
        Route::get('/admin', [AdminController::class,
'index']);
    });
```

- This ensures only users with the admin role can access certain features.

2.3 User invitation & email confirmation

- New users receive an email with a registration link.
- Email confirmation is handled by the `ConfirmEmailController`.

Key code structures

The module is implemented using Laravel's authentication framework and consists of models, controllers, services, and middleware.

1. Models (database layer)

- `/app/Users/User.php` → Defines user attributes and relationships.
- `/app/Permissions/EntityPermission.php` → Handles permission assignments.
- `/app/Permissions/RolePermission.php` → Stores role-based access rules.

2. Controllers (business logic & API)

- `/app/Access/RegistrationController.php` → Manages user sign-up.
- `/app/Access/LoginController.php` → Handles login requests.
- `/app/Access/ForgotPasswordController.php` → Manages password resets.
- `/app/Access/MfaController.php` → Handles multi-factor authentication.
- `/app/Permissions/PermissionController.php` → Controls role assignments.

3. Services (application logic)

- `/app/Access/RegistrationService.php` → Manages user sign-up logic.

- `/app/Access/LoginService.php` → Handles authentication logic.
- `/app/Access/UserInviteService.php` → Sends email invitations.
- `/app/Access/EmailConfirmationService.php` → Confirms email addresses.

4. Middleware (access enforcement)

- `/app/Http/middleware/AuthMiddleware.php` → Ensures authentication.
- `/app/Http/middleware/RoleMiddleware.php` → Enforces role-based permissions.
- `/app/Http/middleware/MfaMiddleware.php` → Checks for MFA authentication.

5. Views (user interface)

- `/resources/views/auth/login.blade.php` → Login page.
- `/resources/views/auth/register.blade.php` → Registration page.
- `/resources/views/auth/mfa.blade.php` → Multi-factor authentication prompt.
- `/resources/views/users/profile.blade.php` → User profile settings.

Benefits of the module

- **For end users (students, educators, contributors):**
  - Ensures a secure and reliable login experience, protecting personal and institutional data.
  - Offers self-service registration and password reset, minimizing dependence on administrators.
  - Provides multiple authentication options (password, MFA, single sign-on) for convenience and enhanced security.
  - Guarantees only authorized access to content and actions, protecting privacy and user contributions.
  - Supports external authentication providers, making it easy to log in with existing institutional or social accounts.
- **For administrators:**

- Enables granular, role-based access control—defining exactly who can manage users, settings, or content.
- Simplifies onboarding and management with user invitations, bulk creation, and email confirmations.
- Offers clear audit trails through permissions models and middleware enforcement, supporting compliance.
- Reduces the risk of unauthorized access via session management, brute-force protection, and MFA enforcement.
- Facilitates integration with enterprise identity systems (LDAP, SAML, OIDC) for unified user management.

- **For technical teams and platform maintainers:**
  - Leverages Laravel's robust and well-supported authentication framework, ensuring maintainability and security best practices.
  - Supports scalable and modular user management, making it easy to extend or adapt for new requirements.
  - Centralizes authentication and access logic, simplifying troubleshooting and future development.
  - Provides reusable middleware and service components for consistent enforcement of access policies across the platform.
  - Enables rapid updates or integration of additional authentication mechanisms as security standards evolve.

Conclusion

The **Authentication & User Access module** forms the backbone of the platform's security framework. It provides:

- Secure user authentication using bcrypt hashing.
- Granular role-based access control for managing permissions.
- Multi-Factor Authentication (MFA) for enhanced security.
- External authentication support (LDAP, SAML, OIDC).
- Session management & middleware-based enforcement for secure access.

By implementing these security mechanisms, the module ensures data integrity, controlled access, and scalability, making it a robust solution for managing users and permissions within the platform.

4.2 Content management module

At the core of the platform, the **Content Management module** is the backbone of its functionality, enabling users to create, edit, organize, and manage content in a structured and efficient manner. The module follows a hierarchical structure of Books, Chapters, and Pages, allowing for intuitive content organization. By integrating rich editing tools, media support, version control, and permissions management, the module ensures that content is accessible, well-structured, and easily navigable.

The Content Management Module leverages Laravel's MVC architecture, where:

- **Models** represent Books, Chapters, Pages, and related content entities.
- **Controllers** handle user interactions such as content creation, editing, deletion, and organization.
- **Views** render the user interface, ensuring an intuitive and seamless experience for users.
- **Services** manage core logic, including content formatting, exporting, and indexing.

Key features

1. Hierarchical content structure

- The platform adopts a multi-tiered approach to content organization:
    - **Books** serve as containers for broad topics.
    - **Chapters** help divide books into meaningful sections.
    - **Pages** store the actual content, such as text, images, or embedded media.
- This hierarchy ensures that content remains structured, making it easy to navigate and reference.

2. WYSIWYG editor for rich text formatting

- Users can create and edit content with an intuitive What You See Is What You Get (WYSIWYG) editor.
- The editor includes:
    - Text formatting options (bold, italics, headings, bullet points).
    - Tables, hyperlinks, and lists to improve content structuring.
    - Code block support for embedding technical scripts.
    - Math equation support for academic and scientific content.

- The editor integrates with Markdown and WYSIWYG modes, offering flexibility based on user preference.

3. Media upload & embedding

- Users can upload images, videos, PDFs, and attachments to enhance content.
- The system provides an internal media library, allowing users to reuse uploaded assets across multiple pages.
- Supports external media embedding, enabling content creators to insert YouTube videos, external PDFs, and other online resources.
- All media files are stored in `/app/Uploads`, ensuring secure file handling and access control.

4. Version control & page revisions

- The platform maintains historical versions of **Books**, **Chapters**, and **Pages**.
- Users can:
  - View previous versions of content.
  - Compare changes between different revisions.
  - Roll back to an earlier version if necessary.
- This feature ensures that accidental deletions or unwanted changes can be reversed.

5. Content permissions & access control

- The module integrates with the **Permissions** system, ensuring that only authorized users can create, edit, or delete content.
- Role-based permissions control:
  - Who can edit **Books**, **Chapters**, and **Pages**.
  - Who can view content.
  - Who can delete or restore content from the recycle bin.
- Implemented using Laravel's Gate and Policy system, allowing flexible and scalable role management.

6. Exporting & printing support

- Users can export **Books**, **Chapters**, or **Pages** into various formats:
  - PDF (for offline reading and printing).
  - Markdown (for portability).
  - HTML (for external web embedding).

- This is managed through the `BookExportController`, `ChapterExportController`, and `PageExportController`.

7. Internal linking & references

- Users can create cross-references between **Books**, **Chapters**, and **Pages**, ensuring easy content navigation.
- This is handled by the **References** module, which:
  - Tracks internal links and automatically updates references when content is moved or renamed.
  - Prevents broken links within the platform.

How the module works

1. **Content creation & editing**
   - Users create content via the WYSIWYG or Markdown editor.
   - When content is saved, it is stored in the database, maintaining relationships between **Books**, **Chapters**, and **Pages**.
   - Each content entry is assigned a unique ID, ensuring that relationships are preserved even if content is reorganized.
2. **Hierarchical structure enforcement**
   - The system enforces hierarchical relationships using foreign keys in the database:
     - **Bookshelves** contain **Books**.
     - **Books** contain **Chapters**.
     - **Chapters** contain **Pages**.
   - Laravel's Eloquent ORM manages these relationships, ensuring data integrity.
3. **Versioning & change tracking**
   - Every time content is updated, a new version is stored.
   - Users can view previous versions, compare changes, and restore older versions.
   - This prevents data loss and allows for content recovery.
4. **Permissions & access control**
   - The system checks user roles before allowing edits or deletions.
   - Middleware ensures only authorized users can perform actions on content.
   - Example: An **Editor** can edit a **Page**, but a **Viewer** cannot.

5. **Exporting & file handling**
   - When exporting content, the system generates a file in the requested format.
   - Laravel's **Blade templating engine** is used to format HTML exports.
   - File storage is handled in `/storage/uploads/`, ensuring organized file management.
6. **Search & content discovery**
   - The module integrates with the **Search** module, ensuring real-time indexing of content.
   - Users can search for Books, Chapters, and Pages using keywords and tags.

Key code structures

1. Models (database representation)

- `/app/Entities/Bookshelf.php` → Represents **Bookshelves**.
- `/app/Entities/Book.php` → Represents **Books**.
- `/app/Entities/Chapter.php` → Represents **Chapters**.
- `/app/Entities/Page.php` → Represents individual **Pages**.
- `/app/Entities/PageRevision.php` → Tracks version history.

2. Controllers (business logic & API)

- `/app/Entities/BookshelfController.php` → Manages **Bookshelves**.
- `/app/Entities/BookController.php` → Manages **Books**.
- `/app/Entities/ChapterController.php` → Handles **Chapter** operations.
- `/app/Entities/PageController.php` → Handles **Page** creation and editing.
- `/app/Entities/PageRevisionController.php` → Handles version history.
- `/app/Entities/BookExportController.php` → Manages **Book** exports.

3. Services (application logic)

- `/app/Entities/Tools/ContentFormatter.php` → Handles content rendering.
- `/app/Entities/Tools/ExportService.php` → Manages export operations.

4. Middleware (access control)

- `/app/Http/middleware/ContentPermissionMiddleware.php` → Ensures access control.

5. Views (user interface)

- `/resources/views/entities/book.blade.php` → **Book** view template.
- `/resources/views/entities/chapter.blade.php` → **Chapter** template.
- `/resources/views/entities/page.blade.php` → **Page** template.
- `/resources/views/editor.blade.php` → WYSIWYG editor interface.

Benefits of the module

- **For educators and content creators:**
  - Enables rapid development and organization of complex educational materials using a familiar book–chapter–page hierarchy.
  - Supports collaborative authoring with real-time editing, version history, and rollback features.
  - Provides rich text and media embedding tools, simplifying the integration of images, videos, equations, and interactive content.
  - Reduces administrative workload by automating content structuring and permission management.
  - Ensures consistency and quality through built-in templates and export functions.
- **For students and end users:**
  - Delivers an intuitive, easily navigable structure for course materials, improving comprehension and engagement.
  - Allows quick content discovery through advanced search, tagging, and internal linking.
  - Offers access to multimedia and interactive elements for a richer learning experience.
  - Guarantees content reliability with transparent revision history and up-to-date materials.
  - Protects sensitive content through robust, role-based permissions.

- **For administrators and technical teams:**
  - Facilitates centralized management of all educational content, users, and access policies.
  - Eases onboarding of new contributors through standardized workflows and access controls.
  - Integrates seamlessly with other modules (search, export, permissions), ensuring platform scalability and flexibility.
  - Supports compliance, backup, and archival needs through export and versioning features.
  - Leverages Laravel's MVC architecture for maintainability, enabling efficient updates and future enhancements.

Conclusion

The **Content Management module** is the foundation of the platform, ensuring that users can create, structure, and manage content effectively. By combining hierarchical organization, rich-text editing, media integration, version control, and permissions, the module enables a dynamic, secure, and user-friendly content creation experience.

This robust system ensures that users can:

- Structure their content efficiently using a Books, Chapters, Pages approach.
- Create rich, formatted content using the WYSIWYG editor.
- Embed multimedia and external resources for a richer experience.
- Maintain content history and restore previous versions when needed.
- Control access permissions to ensure secure content management.

By integrating with the Search and Export modules, the Content Management system enhances accessibility and portability, making it a powerful and scalable solution for content-driven applications.

4.3 User Management module

The **User Management module** is a central administrative tool that empowers platform administrators to create, manage, and monitor user accounts, assign roles and permissions, and enforce access control policies. It ensures that every user on the platform—whether an

administrator, educator, or student—has access only to the content and tools appropriate to their role, promoting both security and clarity in the user experience.

By combining robust user account management, role-based permissions, and comprehensive activity tracking, the module provides the foundation for maintaining an organized, scalable, and secure user environment.

Key features

1. Adding, editing, and deactivating user accounts

- **Administrators** can create new users manually or via automated onboarding (e.g., email invitations).
- Existing user accounts can be edited to update key details:
  - Name
  - Email address
  - Assigned roles
  - Profile images or avatars
- Users can be deactivated instead of deleted, preserving historical data and activity logs for auditing purposes.
- Bulk operations support the management of large user populations, streamlining administrative tasks for educational institutions or enterprise environments.

2. Assigning roles and permissions

- The module uses a **Role-Based Access Control** (RBAC) system to manage what users can see or do:
  - **Admin** – Full control over user management, settings, and content.
  - **Teacher** – Can create, edit, and delete content but cannot manage users.
  - **Student** – Can create, edit, and delete their own content.
  - **Viewer** – Read-only access, ensuring no unauthorized modifications.
- Custom roles can be created by administrators to tailor access policies to organizational needs.
- Roles are linked to permissions such as `edit-pages`, `delete-books`, or `manage-users`, providing granular control over user capabilities.

3. Monitoring user activity logs

- All significant user actions—such as logins, content edits, permission changes, and role assignments—are logged and timestamped.
- Administrators can review activity logs to audit behaviour, trace changes, or investigate security concerns.
- Activity data can be filtered by user, action, or time period, and used to generate custom reports to monitor engagement or detect anomalies.

How the module works

The User Management module is built on Laravel's authentication system and enhanced by custom models, controllers, and middleware.

Database structure

- Users are stored in the `users` table.
- Roles and permissions are stored in the `roles` and `permissions` tables.
- Relationships are managed through pivot tables:
  - `role_user` connects users to roles.
  - `permission_role` connects permissions to roles.
- This structure enables many-to-many relationships, allowing multiple roles per user or multiple users per role.

Access enforcement

- The system uses middleware to restrict access based on assigned roles:
  - `auth` ensures users are logged in.
  - Custom middleware like `can:manage-users` or `role:admin` enforce role-specific access.
- Example:

```
    Route::middleware(['auth', 'role:admin'])->group(function
() {
        Route::get('/users', [UserController::class,
'index']);
    });
```

Role and permission management

- **PermissionController** handles creating and assigning permissions.

- **UserController** and **RoleController** (within the `/Users` and `/Permissions` namespaces) manage the user-role-permission lifecycle.
- Controllers interact with models like:
    - `User`, `Role`, `EntityPermission`, `RolePermission`, and `JointPermission`.

User interface (UI)

- Views located in `/resources/views/users/` provide:
    - A user list with search and filters
    - Forms for editing user profiles and assigning roles
    - Dashboards for managing permissions and activity logs

Key code structures

1. Models

- `/app/Users/User.php` – Core user model.
- `/app/Permissions/RolePermission.php` – Role-specific permissions.
- `/app/Permissions/EntityPermission.php`, `/JointPermission.php` – Control access to platform entities.

2. Controllers

- `/app/Users/UserController.php` – Manage user accounts and profile settings.
- `/app/Permissions/PermissionController.php` – Assign and manage permissions.

3. Middleware

- `/app/Http/middleware/CheckRole.php` – Restricts route access based on role.
- `/app/Http/middleware/Authenticate.php` – Ensures user authentication.

4. Views

- `/resources/views/users/index.blade.php` – List and filter users.
- `/resources/views/users/edit.blade.php` – Edit user details and assign roles.

- `/resources/views/permissions/index.blade.php` – View and manage roles and permissions.

Benefits of the module

For administrators:

- Provides centralized tools for managing users at scale.
- Enables the creation of tailored access rules to fit organizational policies.
- Supports auditing and accountability through detailed logs and role histories.

For users:

- Ensures that each user has access only to relevant tools and content, improving usability and minimizing confusion.
- Protects user privacy and platform integrity by preventing unauthorized access.
- Provides a seamless, personalized experience across the platform.

Conclusion

The **User Management module** is critical to maintaining the platform's security, structure, and usability. With tools for account lifecycle management, role assignment, and activity tracking, it ensures that users interact with the platform in a controlled, transparent, and efficient manner.

Its tight integration with the Permissions and Authentication modules, paired with intuitive administrative interfaces and Laravel-powered logic, makes it an essential feature for institutions managing large user bases across roles and departments.

Whether managing a small class or an entire institution, this module empowers administrators to maintain order, uphold security, and ensure a positive user experience for everyone.

4.4 Search and Indexing module

The **Search and Indexing module** is a vital component of the platform, designed to provide users with powerful, intuitive, and fast content discovery capabilities. Whether educators are managing vast repositories of academic content or students are trying to find specific materials within their courses, this module ensures quick and accurate retrieval of relevant information.

By combining full-text search, field-based filtering, real-time suggestions, and optional integration with external search engines, the module significantly enhances the usability, responsiveness, and scalability of the platform's content exploration capabilities.

Key features

1. Full-text search with field-specific filters

- Performs full-text searches across key content types: **Bookshelves**, **Books**, **Chapters**, **Pages**, and **Users**.
- Scans titles, body content, tags, and metadata to return highly relevant results.
- Allows users to narrow down results by applying filters:
    - **Content type** (**Bookshelf**, **Book**, **Chapter**, **Page**)
    - **Author** or **creator**
    - **Date** or **last modified**
    - Associated **tags** or **categories**
- Search results are ranked by relevance, ensuring the most important matches appear first.

2. Intelligent search suggestions & autocomplete

- As users begin typing in the search field, the system:
    - Suggests matching titles and tags.
    - Completes words or phrases to speed up search query formulation.
- Autocomplete reduces typing effort and enhances user experience, especially when users don't recall the exact content title.
- Suggestions are pulled dynamically from recent and popular queries as well as indexed content, helping users explore relevant materials faster.

3. Integration with external search engines (Elasticsearch support)

- For large-scale deployments, the platform can integrate with Elasticsearch, providing:
    - Faster indexing and retrieval
    - Fuzzy matching (for typos and similar words)
    - Synonym handling
    - Advanced phrase-based queries

- This integration is optional and configurable, allowing institutions to choose between:
    - Laravel's native database-based search
    - External search engines for high-performance indexing

4. Real-time indexing & search accuracy

- The module ensures search indexes are always up to date:
    - Whenever content is created, updated, or deleted, the index is refreshed.
    - This real-time updating ensures users always search the latest available data.

How the module works

The Search and Indexing module is built using Laravel's native search capabilities, which can be extended through Laravel Scout. Scout provides a flexible interface to perform full-text searches using either a database driver or an external search engine like Elasticsearch.

Indexing workflow:

1. Content creation, update, or deletion triggers an update in the search index.
2. Model observers or event listeners capture changes in key entities: **Bookshelves**, **Books**, **Chapters**, **Pages**, and **Users**.
3. The data is pushed into the index (either local DB or external system).

Search workflow:

1. User enters a query into the search bar.
2. The system:
    - Tokenizes the input into searchable terms.
    - Applies filters and content type restrictions.
    - Searches against indexed fields like title, tags, and content body.
3. Results are scored and ranked:
    - Title matches are prioritized.
    - Metadata and body content are considered with weighted scoring.
4. Results are presented with highlights, filters, and context snippets, allowing users to quickly locate relevant matches.

API-based search support

- The platform provides an API endpoint for search, allowing:
  - External applications to perform search queries.
  - Programmatic filtering and retrieval of search results.
- Defined in `/routes/api.php`, the `SearchApiController` handles:
  - Search query parsing
  - JSON-formatted results
  - Role-based search restrictions (e.g., only show content a user has permission to view)

Key code structures

1. Folders

- `/app/Search` → Main logic and controllers for search
- `/app/References` → Supports internal linking between search results and related content
- `/routes/api.php` → Defines search-related API routes

2. Controllers

- `/app/Search/SearchController.php` → Handles UI-based search requests (web interface).
- `/app/Search/SearchApiController.php` → Handles API-based search queries and JSON responses.

3. Models and traits

- Content models such as **Bookshelf**, **Book**, **Chapter**, **Page**, and **User** implement searchable traits to be indexed.
- These models define which fields (title, tags, body, etc.) are searchable.

4. Views

- `/resources/views/search/index.blade.php` → Renders the search interface.
- `/resources/views/search/partials/results.blade.php` → Displays search results with context and filters.

5. Optional integrations

- If **Elasticsearch** is enabled:

- Configuration is handled in `config/scout.php`.
- Custom indexing behaviour can be defined in Searchable model traits.

Benefits of the module

- Educators can quickly locate and update content across multiple Bookshelves, Books, Chapters, and Pages without manual browsing.
- Students benefit from **fast, responsive search** results that allow them to explore and discover content effectively.
- **Scalability**: As content repositories grow, the system remains responsive through real-time indexing and external search engine support.
- **Flexibility**: Filters and autocomplete provide a personalized, efficient experience that adapts to user behaviour and content structure.

Conclusion

The **Search and Indexing module** plays a central role in making the platform navigable, responsive, and user-friendly. With a combination of real-time full-text search, field-specific filtering, and scalable integration with external engines, it ensures that users can quickly access relevant materials, no matter the size or complexity of the content repository.

By maintaining a live index of Bookshelves, Books, Chapters, Pages, and Users - and intelligently ranking results based on relevance - the module supports the platform's mission to offer a structured, accessible, and engaging educational experience.

4.5 API & External Integrations module

The **API module** serves as the platform's gateway for integration with third-party tools, custom applications, and external systems. By exposing core functionality through a RESTful API, this module enables seamless interoperability, making the platform extensible, automatable, and adaptable to a wide range of use cases. From integrating with Learning Management Systems (LMS), to automating backend processes, or powering custom user interfaces (like mobile apps), the API module is the key to connecting BookStack with the broader digital ecosystem.

Key features of the module

1. Endpoints for managing content, users, and permissions

The API provides a robust set of endpoints for interacting with nearly all critical aspects of the platform:

- Content management
  - Full CRUD (Create, Read, Update, Delete) operations for:
  - Books, Chapters, and Pages
  - Bookshelves and Page Revisions
  - Allows external systems to create and organize educational materials, retrieve content dynamically, or automate content publishing workflows.
- User management
  - Endpoints to create, retrieve, update, and deactivate users
  - Access user profiles, roles, and activity data
  - Synchronize users with external systems such as Single Sign-On (SSO) providers or LMSs
- Permissions management
  - APIs to assign or update roles and permissions for users
  - Helps maintain consistent access control policies across integrated systems

These endpoints follow RESTful conventions, ensuring developers can interact with them in a predictable and intuitive way.

2. Secure API access with token-based authentication

- Secures API endpoints via token-based authentication
- Developers or system administrators can:
  - Generate personal access tokens for specific users or integrations
  - Limit token permissions to specific scopes (e.g., content read-only, user management)
  - Set token expiration or revoke tokens for added security
- All API interactions occur over HTTPS, ensuring encrypted communication of sensitive data

This architecture ensures that only authorized and authenticated clients can access the platform programmatically, reducing risk while enabling powerful integrations.

3. API versioning to ensure backward compatibility

- The API supports versioning (e.g., `/api/v1/`) to ensure that:

  - Existing integrations are not broken when new features or changes are introduced
  - Developers can build stable integrations and update at their own pace
- Older API versions can be maintained and supported during transition periods
- Future enhancements or architectural improvements can be rolled out as new API versions, minimizing disruption

How the module works

The API module is implemented using Laravel's routing, controller, and middleware layers, following RESTful design principles. Each API request is handled by a dedicated controller method, with middleware used to enforce authentication, rate limiting, and permission checks.

Request flow example

- `GET /api/v1/books` → Retrieves a paginated list of all books, supports optional filters like `?author=Tomaz&tag=IoT`
- `POST /api/v1/users` → Creates a new user account (admin permissions required)
- `PUT /api/v1/pages/{id}` → Updates the content of a specific page
- `DELETE /api/v1/pages/{id}` → Deletes a page, only if the token has deletion permissions

All responses follow standard HTTP status codes, and error messages are provided in a developer-friendly format (JSON), making debugging and integration smooth and transparent.

Key code structures

1. Folders

- `/app/Api` → Contains the controllers, resources, and authentication logic for API requests
- `/routes/api.php` → Defines all API endpoints, grouped and versioned
- `/storage/app` → Used for caching or storing temporary API results, file imports, or media
- `/public/uploads` → Hosts files accessible via the API (e.g., user-uploaded images or attachments)

- `/dev/api` → Provides API testing templates, such as example cURL requests or Postman collections

2. Controllers

- `/app/Api/BookApiController.php`
- `/app/Api/ChapterApiController.php`
- `/app/Api/PageApiController.php`
- `/app/Api/BookshelfApiController.php`
- `/app/Api/UserApiController.php`

These controllers map directly to REST actions like `index()`, `store()`, `update()`, `destroy()`

3. Middleware

- Enforces authentication, scopes, and rate limits
- Ensures read/write permissions are validated for each token and endpoint

Benefits of the module

For developers and integrators

- Provides clean, RESTful endpoints that are easy to document and consume
- Enables integration with external tools such as:
  - Learning Management Systems (LMS)
  - Institutional authentication providers
  - Custom analytics platforms
  - Mobile or desktop clients

For administrators

- Automate routine tasks such as:
  - Bulk content uploads
  - User synchronization
  - Role provisioning
- Build custom management interfaces or dashboards tailored to specific institutional workflows

For the platform as a whole

- Ensures future scalability and flexibility
- Encourages innovation through custom integrations and extensions
- Helps maintain a modular, decoupled architecture that's open to ecosystem-wide adoption

Conclusion

The **API module** transforms the platform from a standalone application into a powerful, extensible hub within an educational or organizational ecosystem. By exposing core content and user management functionalities through a secure, well-documented, and versioned API, it enables automation, system integration, and the development of customized applications.

Whether you're building a mobile companion app, connecting to an LMS, or automating content workflows, the API module ensures your platform remains open, adaptable, and future-ready.

4.6 Theming and Customization module

The **Theming and Customization module** empowers platform administrators and developers to tailor the platform's visual appearance and user interface to reflect institutional branding, enhance usability, or simply provide a more engaging user experience. This module provides the infrastructure needed to customize layouts, colours, styles, and even inject custom HTML or scripts, giving full control over the look and feel of the platform without altering its core functionality.

Whether adapting the interface for a university, a training program, or a branded deployment, this module ensures that the platform can be visually aligned with organizational identity and user preferences.

Key features of the module

1. Custom themes and UI styling

Administrators and developers can create custom themes using the provided structure:

- Define custom colour schemes, fonts, icons, and layouts.
- Extend the default platform appearance without modifying core code.

Themes are stored in the `/themes` directory, making them easy to organize, activate, or switch.

2. Custom HTML and script injection

The module allows injection of custom HTML, CSS, or JavaScript into key parts of the interface.

- Add custom headers, footers, banners, or modals.
- Embed tracking scripts (e.g., Google Analytics) or live chat widgets.

Injection is managed through the `ThemeService` in `/app/Theming`, which ensures that injected content is securely and cleanly integrated into Blade templates.

3. SASS-Based Styling and Asset Compilation

Custom styles can be written using SASS in `/resources/sass`, allowing the use of variables, nesting, and mixins.

Assets are compiled into minified CSS and JS files and stored in `/public/dist` for optimized frontend performance.

The styling layer allows full control over:

- Button styles
- Page layouts
- Typography
- UI components like alerts, menus, and tabs

4. Blade template customization

The UI is rendered using Laravel Blade templates found in `/resources/views`.

- Templates for every major section (e.g., books, chapters, pages, users) can be extended or overridden to match custom needs.
- Developers can create layout extensions or insert custom logic into views without affecting system updates.

How the module works

The module is structured around Laravel's view rendering engine, asset pipeline, and theme service injection mechanisms. It provides a clean separation between application logic and UI customization, ensuring that updates to the core platform do not overwrite visual customizations.

Workflow overview:

- `ThemeService` initialization
  - At runtime, the `ThemeService` loads custom HTML, scripts, or styles based on configuration or active themes.

- This allows the platform to dynamically inject content without code-level changes.

- Custom Styles via SASS
  - Developers modify or extend SASS files in `/resources/sass`.
  - Run asset compilation (npm run dev or npm run production) to build final assets.
  - Output is placed in `/public/dist`, which is automatically linked to in views.

Blade template overrides

- Custom templates can be created by publishing or extending default views in `/resources/views`.
- Developers can override headers, footers, layouts, or individual page components.
- Themes Folder (Reserved for Future Use)
  - The `/themes` directory is reserved for organizing complete theme packages.
  - Each theme may include its own styles, scripts, and views to be loaded as a cohesive unit.

Key code structures

1. Folders

- `/app/Theming` → Contains the `ThemeService.php`, which manages dynamic content injection.
- `/resources/views` → Blade templates for all frontend UI components, from navigation to content rendering.
- `/resources/sass` → SASS source files for styling, organized by component or layout.
- `/public/dist` → Compiled and minified assets (CSS/JS) for frontend use.
- `/themes` → Reserved for complete theme definitions (styles, views, layouts).

2. Services

`ThemeService.php`

- Allows custom HTML, JS, or CSS snippets to be registered and inserted into specific parts of the UI.
- Provides methods like:
  - `registerHeadContent()`
  - `registerBodyContent()`

- `registerScripts()`

3. Views

Layout and interface files like:

- `/resources/views/layout.blade.php`
- `/resources/views/common/header.blade.php`
- `/resources/views/pages/show.blade.php`

Can be extended using Laravel's `@extends` and `@section` directives.

Benefits of the module

For institutions and organizations

- Easily align the platform with branding guidelines, including colours, logos, and UI elements.
- Add custom analytics, accessibility tools, or widgets to meet institutional needs.

For developers and designers

- Enjoy full control over styling and layout, without hacking core files.
- Work with modern web tools (SASS, Blade, Webpack) for efficient theming workflows.

For end users

- Benefit from a clean, branded, and responsive interface tailored to their organization.
- Experience a cohesive visual identity across institutional systems and learning tools.

Conclusion

The **Theming and Customization module** ensures that the platform is not just functional but aesthetically aligned with the identity and needs of its users. By offering tools for HTML injection, SASS-based styling, Blade templating, and dynamic theming, it provides the flexibility needed to craft a personalized and engaging user experience.

Whether applying a university's branding, embedding useful scripts, or simplifying the interface for specific user groups, this module allows the platform to evolve visually without compromising on stability or maintainability.

4.7 Location and Translation module

The **Localization and Translation module** ensures that the platform is accessible to users from diverse linguistic backgrounds by providing multi-language support and seamless language switching capabilities. This module enables administrators to translate the platform's interface, messages, and notifications into various languages, making it suitable for international deployments, multicultural classrooms, and global collaboration.

By supporting both system-wide translations and user-specific language preferences, the module enhances the platform's usability, inclusivity, and adaptability across different regions and audiences.

Key features of the module

1. Multi-language interface support

- The entire platform interface—including buttons, labels, navigation items, system messages, and form elements—can be translated into multiple languages.
- Each language is represented as a PHP file containing key-value pairs under the /lang directory.
- Supported languages can be activated or deactivated via platform settings, giving administrators control over which locales are available to users.

2. User-specific language preferences

- Users can choose their preferred language through their profile settings.
- Once selected, the platform dynamically loads the appropriate language files, rendering the entire interface in the chosen language for that user.
- Preferences are saved in the database, ensuring the language setting persists across sessions.

3. Organized and modular translation files

Translation files are modular and grouped by context, improving maintainability and clarity. Examples include:

- `auth.php` – Login and registration messages
- `pagination.php` – Pagination labels
- `entities.php` – Labels for Books, Pages, Chapters
- `errors.php` – System error messages
- `settings.php` – Configuration-related text

All translation files are stored in `/lang/{locale}` directories (e.g., `/lang/en`, `/lang/sl`, `/lang/de`), allowing easy expansion into new languages.

4. Dynamic language switching

- Language can be switched on the fly based on:
  - User preferences
  - Browser locale
  - URL parameters or environment settings
- The active locale is managed by Laravel's localization system and automatically applied to all views, notifications, and messages.

5. Developer support for custom translations

- Developers can easily add new translations or override existing ones without modifying core files.
- Untranslated keys gracefully fall back to a default language (typically English), ensuring the interface remains functional even with partial translations.

How the module works

The **Localization and Translation module** is built on Laravel's native localization framework, using the `Lang` and `App::setLocale()` mechanisms to load and apply translations dynamically.

Translation workflow:

- On each request, the platform checks the user's language preference.
- Laravel's localization engine loads the relevant files from /lang/{locale}.
- Blade templates and controllers call translations using `__('key')` or `@lang('key')`.
- Notifications, validation messages, and UI components are rendered in the active locale.

Language File Structure Example:

```
/lang/
  └── en/
      ├── auth.php
      ├── errors.php
      ├── pagination.php
      └── settings.php
  └── sl/
      ├── auth.php
```

```
    ├── errors.php
    ├── pagination.php
    └── settings.php
```

Key code structures

1. Folders

- `/app/Translation` → Contains core logic for loading, managing, and applying translations. May include services to define supported locales and override translation logic.
- `/lang` → Stores all translation files, organized by language code and context category.

2. Services and utilities

- `LanguageManager.php` (hypothetical service) could manage:
  - Supported language list
  - Locale switching logic
  - Language preference persistence
- Laravel's `App::setLocale()` is used to set the current locale globally.

3. Blade templates

Use built-in localization directives like:

```
{{ __('auth.login') }}
@lang('settings.language')
```

Benefits of the module

For end users

- Enables users to interact with the platform in their preferred language, improving comfort, comprehension, and engagement.
- Especially important for students and educators in multilingual regions or international programs.

For administrators

- Provides the tools to localize the platform for institutional or regional deployments.
- Reduces the barrier to adoption by ensuring the platform is linguistically inclusive.

For developers and translators

- Offers a clean, modular structure for adding or updating translations.
- Easily integrates with external tools such as **Crowdin** or **POEditor** for collaborative translation workflows.

Conclusion

The **Localization and Translation module** is essential for delivering a globally accessible and inclusive learning experience. By supporting multiple languages, dynamic switching, and modular translations, the module ensures that the platform can be used confidently across diverse linguistic contexts.

Its seamless integration with Laravel's localization engine and clean file structure allows developers and administrators to expand language support with ease, making the platform truly adaptable to international, institutional, and user-specific needs.

4.8 Notifications and Alerts module

The **Notifications and Activity Tracking module** is designed to keep users consistently informed about important actions and events occurring across the platform. Whether it's a student being alerted to newly published content, an educator receiving feedback on shared materials, or an administrator monitoring platform activity, this module ensures that key updates and events never go unnoticed.

By combining email notifications, in-platform alerts, and detailed activity logging, the module helps maintain an engaging, accountable, and transparent environment for all user roles. It enhances the user experience and system integrity by bridging communication gaps and providing insight into platform usage.

Key features of the module

1. Email and in-platform notifications

Supports two main notification channels:

- Email notifications: Sent to the user's registered address, alerting them to critical changes or personal updates even when they are offline.
- In-platform alerts: Displayed as visual notifications within the UI (e.g., bell icons, banners), allowing users to see relevant updates when logged in.

Common notification triggers include:

- Content creation or updates
- Mentions in comments
- Page approvals or deletions
- Role or permission changes

2. Configurable notification preferences

Users can customize their notification settings based on:

- Event type (e.g., content updates, mentions, new user invites)
- Channel (email vs in-platform)
- Frequency (immediate, daily digest, or none)

Notification preferences are respected at the individual user level, allowing each user to tailor their communication experience. Settings are accessible via the user profile or settings menu, ensuring ease of configuration.

3. Activity logging and audit trails

The system records all significant user actions in the activity log, including:

- Logins and logout times
- Edits and deletions of Books, Chapters, and Pages
- Comments, favourites, and tag changes
- Role assignments or system setting changes

Activity logs are stored in the database and written to file logs (`/storage/logs`), supporting:

- User-level history review
- Administrative auditing
- Security monitoring and forensic tracking

4. Webhooks for external integrations

The module can trigger webhooks on specific events (e.g., content changes, new user creation), enabling integration with external systems such as:

- Learning Management Systems (LMS)
- Analytics dashboards
- Notification platforms like Slack or Microsoft Teams

How the module works

The module is powered by Laravel's Notification System and integrates tightly with the platform's event-driven architecture. When an event occurs (e.g., a new comment is posted or a page is updated), an event listener:

- Identifies the relevant notification class
- Determines the target users
- Dispatches the notification via email, in-platform alert, or webhook

Email notifications

- Use Laravel's Mail system and the platform's SMTP configuration (e.g., Gmail, SendGrid) to send messages.
- Templates are defined using Blade view files, allowing customizable email layouts.

In-platform notifications

- Powered by JavaScript/TypeScript components for real-time display in the interface.
- Alerts appear in the notification's dropdown, highlighting unread updates.

Activity tracking

- Actions are recorded using the Activity model in `/app/Activity/Activity.php`.
- Controllers log user actions with relevant metadata (e.g., user ID, timestamp, content ID).
- Logs are stored both in the database and optionally in file logs at `/storage/logs/laravel.log`.

Key code structures

1. Folders

- `/app/Activity`
  - Contains models (`Activity.php`, `View.php`, `Watch.php`, etc.)
  - Controllers such as `AuditLogController.php` for displaying logs
- `/app/Notifications`
  - Houses Laravel Notification classes
  - Defines notification templates and dispatch logic

- `/storage/logs`
    - Stores application logs including user activity and system-level events

2. Controllers

- `/app/Activity/AuditLogController.php` → Displays logs in the admin interface
- `/app/Activity/FavouriteController.php`, `CommentController.php`, `TagController.php` → Log user interactions and trigger notifications

3. Models

- `Activity` – Stores user events
- `Comment`, `Favourite`, `Watch` – Track user interaction with content

4. Notification classes

- Located in `/app/Notifications`
    - `ContentUpdatedNotification.php`
    - `UserMentionNotification.php`
    - `RoleChangedNotification.php`

5. Views

- Notification UIs integrated into standard page templates via `/resources/views/partials/notifications.blade.php` and TypeScript components

Benefits of the module

For educators and Content Creators

- Stay informed about changes to content they manage
- Get immediate alerts when mentioned or when collaborative updates occur
- Monitor student engagement with their materials

For students

- Receive notifications when new learning materials are published
- Get reminders for important updates or responses to their comments
- Stay connected to the platform without constantly checking manually

For administrators

- Track system usage and user behaviour
- Monitor key system events (e.g., logins, role assignments, deletions)
- Gain insight into how users engage with content and features

Conclusion

The **Notifications and Activity Tracking module** strengthens platform communication by delivering timely, relevant, and personalized updates to users while simultaneously offering administrators a comprehensive view of platform activity. With real-time alerts, flexible preferences, and robust logging, this module ensures that users remain engaged, informed, and secure.

Its deep integration with the platform's content and user systems makes it an essential component for promoting accountability, enhancing collaboration, and supporting a vibrant, well-informed learning environment.

4.9 System Configuration and Settings module

The **System Configuration and Settings module** is the backbone of the platform's operational environment, responsible for managing core system settings, runtime configuration, performance optimization, and logging. This module ensures the platform is both stable and flexible, allowing administrators to tune behaviour, enable features, monitor performance, and troubleshoot issues without modifying application code.

Whether you're adjusting site-wide preferences, managing environment-specific behaviour, or diagnosing errors, this module provides the essential tools to configure, monitor, and maintain the platform at scale.

Key features of the module

1. Centralized system settings management

Platform-wide settings are managed in a unified interface, covering areas such as:

- Site name and branding
- Default language and time zone
- User registration options
- Email server configuration
- Privacy and security preferences

Settings are stored in the database and loaded via Laravel's configuration cache, enabling real-time updates without restarting the application.

2. Runtime configuration and environment control

Supports the use of .env files and /app/Config logic to manage environment-specific configurations such as:

- Database connection
- Cache drivers
- Storage paths
- API keys and third-party service credentials

Developers and system administrators can define configurations that are environment-aware (e.g., dev vs production), ensuring consistent behaviour across deployments.

3. Caching for improved performance

- Utilizes Laravel's caching mechanisms to store configuration data, compiled views, and route definitions for fast access.
- Cache files are stored in `/storage/framework/cache`, reducing database queries and speeding up request processing.
- Administrators can clear and rebuild caches via Artisan commands or admin tools:

```
php artisan config:cache
php artisan route:cache
```

4. Logging and error tracking

All system events, warnings, and errors are logged to files located in `/storage/logs`.

- Includes logs for user actions, system events, authentication attempts, and exceptions.
- Organized by date, enabling efficient debugging and issue tracing.

Supports integration with external monitoring tools or log aggregators such as:

- Sentry, Loggly, or centralized ELK stacks (Elasticsearch, Logstash, Kibana)

5. Maintenance and debugging tools

- Provides maintenance mode toggle via Artisan or UI (e.g., php artisan down / up).
- Debugging features such as detailed exception pages and error stacks are conditionally enabled via APP_DEBUG in the .env file.

- Tracks system status and component availability to assist with diagnostics and uptime monitoring.

How the module works

This module leverages Laravel's robust configuration, logging, and caching systems, providing abstractions and management interfaces that are customized for BookStack's needs.

Settings management workflow:

- Admin updates a setting via the UI or CLI.
- The `SettingsService` (in `/app/Settings`) stores the change in the database.
- Updated values are cached for fast retrieval, with the option to persist them to config files.
- Changes take effect immediately without restarting the platform.

Logging workflow:

- A system event (error, warning, info) triggers Laravel's Log facade.
- The message is written to the appropriate log file in `/storage/logs/laravel.log`.
- Logs can be viewed manually or aggregated externally for further analysis.

Caching workflow:

- Frequently used data (e.g., routes, views, settings) is compiled into optimized files in `/storage/framework`.
- These are automatically read on each request, bypassing unnecessary runtime computations.

Key code structures

1. Folders

- `/app/Config`
  - Custom configuration files and logic
  - Extends Laravel's config loading mechanism
- `/app/Settings`
  - `Setting.php` model for database-stored configuration

- `SettingsService.php` for retrieving and modifying settings
- `/storage/framework`
  - Caches for config, routes, sessions, views
- `/storage/logs`
  - Log files for system operations and debugging

2. Models and services

- `Setting` – Represents system settings stored in the database.
- `SettingsService` – Manages setting retrieval, validation, and updates.

3. Artisan commands

Commonly used commands:

```
php artisan config:cache
php artisan view:clear
php artisan log:clear
php artisan down --reason="System maintenance"
```

Benefits of the module

For administrators

- Provides a centralized, intuitive interface for managing critical system settings.
- Reduces reliance on developer intervention by exposing configurable behaviour through the admin panel.
- Supports efficient troubleshooting and monitoring via real-time logs.

For developers

- Offers flexible, environment-specific configurations using Laravel's .env and config file structure.
- Simplifies deployment, debugging, and maintenance workflows with built-in tools for caching and error tracking.

For the platform

- Ensures consistent behaviour, stability, and performance across different environments.
- Makes the system adaptable and extensible, supporting long-term scalability and third-party integrations.

Conclusion

The **System Configuration and Settings module** provides the administrative and infrastructural backbone of the platform. From managing environment variables and caching layers to tracking logs and system behaviour, it ensures that the platform operates smoothly, efficiently, and securely.

With its tight integration into Laravel's core systems and its extensible, service-driven architecture, this module offers a powerful toolkit for both day-to-day operation and long-term maintenance, making it indispensable for administrators and developers alike.

4.10 Backend Administration (Console and Maintenance) module

The **Backend Administration (Console & Maintenance) module** provides powerful tools for managing, maintaining, and automating the platform's core operations from the command line. Designed for system administrators and DevOps teams, this module allows for the execution of routine and advanced maintenance tasks, enabling efficient platform upkeep, troubleshooting, and performance tuning—all without the need for direct interaction with the web interface.

By harnessing Laravel's Artisan CLI framework and extending it with custom BookStack commands, this module ensures the platform remains stable, clean, and optimized in all operational contexts, from single-server deployments to complex CI/CD pipelines.

Key features of the module

1. Command-line system administration

The platform includes a suite of Artisan commands to handle essential administrative tasks:

- Cache clearing and rebuilding
- Permission regeneration
- Database migration and seeding
- User and role management
- Maintenance mode toggling

These tasks can be executed manually or integrated into automated deployment scripts, reducing overhead for administrators.

2. Automated maintenance and clean-up

Regularly scheduled tasks help maintain system health, such as:

- Cleaning up old activity logs and revisions
- Optimizing storage and clearing unused uploads
- Checking system status and configuration consistency

Commands can be scheduled using cron jobs or Laravel's Task Scheduler, ensuring continuous maintenance without manual intervention.

3. BookStack-specific CLI utilities

Located in the /bookstack-system-cli tool, the platform extends Laravel's CLI with custom commands tailored to BookStack operations:

- Managing users, content, and settings via CLI
- Exporting or importing content
- Running diagnostics or integrity checks

These utilities provide fine-grained control over platform behaviour and streamline tasks that would otherwise require UI-based workflows.

4. Maintenance mode management

The platform can be placed into maintenance mode during updates or critical operations:

- Uses php artisan down and php artisan up commands
- Displays a customizable maintenance message to end users

Optionally, administrators can whitelist IPs during maintenance to allow continued access for testing.

How the module works

This module is tightly integrated with Laravel's console system, utilizing the Artisan command-line tool and custom command classes defined in the `/app/Console` directory.

Command registration workflow:

- Commands are defined in `/app/Console/Commands`.
- Registered in `Kernel.php`, which binds them to Artisan and enables scheduling.
- When executed, each command:
    - Performs its assigned logic (e.g., clearing logs, exporting data)
    - Outputs success/error messages to the terminal

- Administrators can automate commands using scheduled tasks (`schedule()` method in `Kernel.php`) or system-level cron jobs.

Custom CLI tool:

The bookstack-system-cli is a standalone utility or script that offers additional BookStack-focused commands and administrative utilities. It can interact directly with core services and database layers for batch operations or system configuration.

Key code structures

1. Folders

- `/app/Console` → Contains the Artisan command classes, such as `CleanRevisions`, `SendReminders`, `RegeneratePermissions`. Includes `Kernel.php` for scheduling and command registration
- `/bootstrap` → Bootstraps the Laravel application for CLI execution
- `/artisan` → Laravel's command-line entry point script (`php artisan`)
- `/bookstack-system-cli` → BookStack-specific CLI tool offering enhanced command utilities and system scripts

2. Example commands

- `php artisan bookstack:regenerate-permissions` → Rebuilds permission tables
- `php artisan bookstack:cleanup-revisions` → Removes excess page revisions
- `php artisan bookstack:create-user` → Adds a new user via CLI
- `php artisan config:cache` → Caches the configuration for better performance
- `php artisan queue:work` → Processes queued jobs

Benefits of the module

For system administrators

- Enables direct, scriptable access to system functions
- Supports automation of repetitive tasks and batch operations
- Facilitates headless deployments, scaling, and CI/CD integration

For developers

- Provides a clean and consistent way to extend the platform's functionality with custom Artisan commands
- Enables fast debugging and configuration without relying on the frontend interface

For platform maintenance

- Ensures the platform remains optimized, secure, and clean through scheduled and manual tasks
- Reduces downtime during updates with controlled maintenance workflows

Conclusion

The **Backend Administration (Console & Maintenance) mod**ule is a powerful toolkit for maintaining and scaling the platform efficiently and reliably. By combining Laravel's Artisan CLI with custom BookStack-specific utilities, it provides deep control over system operations—from routine maintenance to advanced configuration.

Whether running on a single server or deployed as part of a cloud-native infrastructure, this module ensures that administrators can monitor, manage, and maintain the platform confidently and effectively, empowering smooth operations and long-term sustainability.

4.11 Database Management module

The **Database Management module** provides the foundation for the platform's data structure and integrity. It handles everything from schema evolution and initial data population to test data generation and system backups. This module plays a crucial role in both the development and deployment lifecycle, enabling developers, administrators, and DevOps teams to manage the platform's database in a controlled, repeatable, and secure manner.

By leveraging Laravel's powerful database tools—migrations, seeders, and factories—along with backup and restore mechanisms, this module ensures that the platform's data is both structured and safeguarded, ready to support scalable and reliable deployments.

Key features of the module

1. Database schema management via migrations

Migrations define and track the structure of database tables, including:

- Table creation
- Indexing and constraints

- Column updates and removals

Stored in `/database/migrations`, each migration is versioned and timestamped, ensuring consistent schema evolution across environments.

Commands like `php artisan migrate`, `migrate:rollback`, and `migrate:fresh` provide developers with precise control over schema changes.

2. Initial and sample data population with seeders

- Seeders insert predefined or sample data into the database:
  - Default roles and permissions
  - System settings
  - Demo users and content
- Useful for bootstrapping development environments, demo setups, or resetting test instances.
- Located in `/database/seeders`, they can be run via:

```
php artisan db:seed
php artisan db:seed --class=UserSeeder
```

3. Test data generation with factories

Factories in `/database/factories` define blueprints for generating fake data using Laravel's faker library.

Ideal for automated testing, load testing, and prototyping.

Example:

```
User::factory()->count(10)->create();
```

Factories are integrated into unit and feature tests, allowing developers to simulate realistic data scenarios without affecting production databases.

4. Backup and recovery support

The `/storage/backups` directory is reserved for storing database and system backups. Administrators can generate backups using:

- Custom Artisan commands
- External backup services or Laravel-compatible packages (e.g., Spatie Backup)

Supports periodic, automated backups for:

- Disaster recovery

- Migration to new servers
- Compliance with data retention policies

How the module works

The module uses Laravel's Eloquent ORM and migration system to manage the schema, seeders to populate it, and factories to simulate realistic data. Backups can be handled manually or via integration with scheduling and cloud storage solutions.

Migration workflow:

- Developers define schema changes in migration files (e.g., `create_books_table.php`).
- Changes are tracked and versioned using Laravel's migrations table.
- Running php artisan migrate applies new migrations to the current environment.

Seeding workflow:

- Seeders populate tables with default or mock data.
- Called independently or chained together in the DatabaseSeeder class.
- Can be scoped to environments (e.g., only run on `local/dev`).

Backup workflow:

- Backup command exports the database to `/storage/backups/`.
- Optionally compressed and encrypted for secure storage.
- Can be automated via cron or Laravel's scheduler.

Key code structures

1. Folders

- `/database/migrations` → Schema definition and versioning
- `/database/seeders` → Data insertion scripts for initializing databases
- `/database/factories` → Data blueprints for test generation
- `/storage/backups` → Storage location for database dump files or encrypted backups

2. Artisan Commands

- `php artisan migrate` – Apply migrations

- `php artisan migrate:rollback` – Undo last migration
- `php artisan db:seed` – Run seeders
- `php artisan backup:run` (via Spatie package or custom) – Create backup

Benefits of the module

For developers

- Enables safe schema evolution through version-controlled migrations.
- Supports realistic testing through factories and seeders.
- Simplifies development setup with repeatable database initialization.

For administrators

- Provides control over backup and restore operations.
- Ensures data protection and recovery capabilities.
- Facilitates smooth deployments and updates across environments.

For the platform

- Maintains data integrity and consistency through schema enforcement.
- Reduces risk of downtime or data loss through proactive backup strategies.
- Supports continuous improvement and agile development workflows.

Conclusion

The **Database Management module** is critical to the stability, security, and maintainability of the platform. With tools for managing schema changes, populating data, generating test records, and backing up the system, it ensures the database layer remains robust, versioned, and recoverable.

By aligning with Laravel's best practices and extending them with BookStack-specific workflows, this module empowers both developers and administrators to work confidently and efficiently—protecting data, accelerating development, and ensuring operational resilience.

4.12 Frontend (UI & Interactive Elements) module

The **Frontend (UI & Interactive Elements) module** governs the look, feel, and interactivity of the platform's user interface. It brings together Blade templates, JavaScript behaviours, CSS

styles, and third-party libraries to deliver a polished, intuitive, and responsive experience for all users—whether they are administrators managing users, educators authoring content, or students navigating materials.

This module is built with a modern frontend architecture that emphasizes modularity, performance, and accessibility, ensuring that the platform is both visually appealing and highly functional across devices and use cases.

Key features of the module

1. Responsive and modular user interface

- The platform's interface is crafted using Laravel Blade templates, providing:
  - Modular components like headers, sidebars, menus, and content cards
  - Layout inheritance and reusable sections for streamlined UI development
- The design is responsive by default, ensuring optimal usability across desktops, tablets, and mobile devices.

2. Rich frontend behaviour with JavaScript

- Interactive functionality is powered by custom Blade/TypeScript components located in `/resources/js`.
  - Dynamic forms and content editors
  - Modal windows, collapsible menus, and tabbed navigation
  - Live search, real-time notifications, and user tagging
- JavaScript services manage behaviours such as:
  - AJAX data fetching
  - DOM manipulation
  - Event broadcasting and animations

3. Sass-based styling and theming

- All styles are defined using Sass (SCSS), stored in `/resources/sass`, which offers:
- Variables and mixins for reusable design patterns
- Nested structures for modular and maintainable styling
- Theming capabilities that align with the platform’s customization tools
- Final stylesheets are compiled and minified into `/public/dist` for production use.

4. Integration of third-party frontend libraries

- Located in `/public/libs`, the module includes key third-party libraries like:
  - TinyMCE for WYSIWYG editing
  - Draw.io integration for diagram editing
  - `Clipboard.js`, `Highlight.js`, and other UI tools
- These libraries are seamlessly integrated into the platform's frontend to extend functionality and improve user productivity.

5. Custom editors and interactive elements

- The platform supports multiple content editing modes:
  - WYSIWYG editor
  - Markdown editor
  - Code editor with syntax highlighting
- Rich input components are used throughout the interface, including:
  - Permission toggles
  - User selectors
  - Tag inputs and searchable dropdowns

How the module works

The Frontend Module combines server-side rendering (Blade) with client-side interactivity (Blade and JS components). Assets are compiled via modern frontend tooling (e.g., Laravel Mix, Webpack, Vite), enabling fast development and deployment.

Rendering workflow:

1. Blade templates in `/resources/views` define the layout and structure.
2. Sass files are compiled into optimized CSS and placed in `/public/dist`.
3. Blade and JS components are bundled and served to the browser from `/public/dist/app.js`.
4. Third-party libraries from `/public/libs` are loaded as needed.

Component interaction:

- Blade components communicate via events or props to create dynamic experiences.

- Blade templates serve as the base structure, injecting interactive elements via `@include` and `@yield` directives.
- Real-time updates and in-page interactions are achieved through AJAX and websockets (when enabled).

Key code structures

1. Folders

- `/resources/js`
  - Blade components (`components/`)
  - Editor logic (`editor/`, `markdown/`, `wysiwyg/`)
  - Utility scripts (`services/`, `animations/`, `ajax.js`, etc.)
- `/resources/sass`
  - Global styles, variables, layout and component-specific styles
- `/resources/views`
  - Blade templates for pages, partials, and UI components
- `/public/dist`
  - Compiled CSS/JS assets for frontend performance
- `/public/libs`
  - Static JS/CSS assets from third-party libraries

2. Notable components and scripts

- `Editor.js` – Manages WYSIWYG and Markdown content editing modes
- `AjaxService.js` – Handles dynamic content fetching and form submissions

Benefits of the module

For end users

- Provides a visually consistent and intuitive interface across devices
- Enhances content interaction with dynamic, responsive elements
- Supports accessibility and usability through modern UI patterns

For developers

- Offers a clean and maintainable architecture for extending or customizing the UI

- Enables rapid development using Blade
- Ensures frontend assets are optimized for performance and maintainability

For institutions and organizations

- Supports branding and theming through custom styles and layouts
- Delivers a user-centric experience that aligns with modern UX expectations
- Allows seamless integration of third-party tools and scripts

Conclusion

The **Frontend (UI & Interactive Elements) module** is what brings the platform to life—transforming backend logic into an elegant, user-friendly interface. It provides all the tools necessary to build and deliver an engaging experience through well-structured templates, dynamic JavaScript components, clean styling, and third-party integrations.

Whether enhancing content editing, improving navigation, or fine-tuning the look and feel of the application, this module ensures that the platform not only works flawlessly—but feels modern, intuitive, and empowering for every user.

4.13 Testing and Development module

The Testing and Development Module is essential for ensuring the reliability, scalability, and maintainability of the platform. It provides a robust environment for automated testing, API development, and asset building, empowering developers to write clean code, catch bugs early, and deploy with confidence.

This module supports both back-end (PHP/Laravel) and front-end (JS/CSS) development workflows, and it plays a crucial role in maintaining code quality, feature stability, and continuous integration readiness.

Key features of the module

1. Automated testing framework

- Uses PHPUnit (configured via `phpunit.xml`) for backend testing of:
  - Models and database relationships
  - Controllers and services
  - Authentication, permissions, and APIs
- Located in `/tests`, the structure supports:
  - **Unit tests** – For isolated logic and utility functions

- **Feature tests** – For user-facing workflows and system interactions
- **Browser tests** (optional) – Simulate real user behaviour in headless environments

- Tests can be executed via:

```
php artisan test
vendor/bin/phpunit
```

2. API development and testing tools

- `/dev/api` provides sample API requests and response templates, making it easier for developers to:
  - Explore and test endpoints
  - Document API behaviours
  - Share API usage examples with third-party integrators
- Supports rapid iteration during API development with mock request flows and expected outputs.

3. Build and asset compilation scripts

- `/dev/build` contains scripts for:
  - Compiling and minifying frontend assets (CSS/JS)
  - Running Webpack/Laravel Mix or Vite
  - Cleaning build directories
- Ensures that assets are optimized, versioned, and production-ready, improving performance and cache control.

4. Developer environment tools

- The `/dev/docker` directory offers Docker-based setups for:
  - Local development environments
  - Database containers (MySQL, MariaDB, etc.)
  - Web servers (NGINX, Apache)
- Enables quick project spin-up and consistent development environments across teams

How the module works

The **Testing and Development module** is structured around **Laravel's testing framework**, modern frontend build pipelines, and support scripts for **developer operations**.

Testing workflow:

1. Test cases are written in `/tests` using PHPUnit conventions.
2. Developers run tests locally or in CI pipelines to validate functionality.
3. Failing tests are logged and output with detailed context for easy debugging.

API testing workflow:

1. Developers reference `/dev/api` templates to create test calls (e.g., via Postman or curl).
2. Requests are sent to `/api/v1/...` routes and validated against expected results.
3. API behaviour is documented alongside test cases for easier integration support.

Build workflow:

1. Frontend code in `/resources/js` and `/resources/sass` is compiled using scripts in `/dev/build`.
2. Output is stored in `/public/dist`.
3. Build scripts include options for:
   - Development (`npm run dev`)
   - Production (`npm run build`)
   - Linting and cleaning

Key code structures

1. Folders

- `/tests` – Unit and feature test classes (e.g., `UserTest.php`, `PagePermissionsTest.php`)
- `/dev`
- `/api` – API request/response examples
- `/build` – Frontend build and asset management scripts
- `/docker` – Dev container configurations for environment setup
  - `phpunit.xml` – Global test configuration (DB setup, environment settings, test discovery paths)

2. Sample test types

- **Unit test example:**

```
public function testUserHasRole()
{
    $user = User::factory()->create();
    $role = Role::factory()->create();
    $user->roles()->attach($role);
    $this->assertTrue($user->hasRole($role->name));
}
```

- **Feature test example:**

```
public function testAdminCanAccessSettingsPage()
    {
        $admin = User::factory()->create(['system_role' =>
'admin']);
        $response = $this->actingAs($admin)->get('/settings');
        $response->assertStatus(200);
    }
```

Benefits of the module

For developers

- Provides a **safe environment** for experimentation and iterative development.
- Enables **test-driven development** and reduces bugs in production.
- Simplifies **frontend builds** with reusable scripts and Docker environments.

For QA and CI/CD

- Supports **automated testing pipelines** for quality assurance.
- Ensures changes do not break existing features (regression testing).
- Provides a testable interface for APIs and external integrations.

For the platform

- Enhances long-term maintainability through **test coverage and reproducibility**.
- Reduces technical debt by encouraging clean, testable code.
- Improves **developer onboarding** by offering ready-to-use dev setups and testing patterns.

Conclusion

The **Testing and Development module** is the platform's engine for quality assurance, developer productivity, and delivery speed. By enabling robust automated testing, easy API prototyping, and streamlined asset builds, it ensures that code is reliable, scalable, and ready for production. From day-to-day development to automated deployment pipelines, this module is essential for maintaining a modern, resilient, and developer-friendly platform.

4.14 Interaction between modules

The platform is built on a modular yet deeply interconnected architecture, where each module plays a specialized role while contributing to the system's unified operation. These interactions ensure a fluid, responsive, and secure user experience, while maintaining flexibility, scalability, and administrative control.

1. Authentication & user access → Foundation for secure interactions

- This module acts as the gateway to all other modules, ensuring that only authenticated and authorized users can access platform features.
- It works hand-in-hand with the Content Management, User Management, and API modules to enforce role-based access, such as restricting administrative functions or editing capabilities to certain roles.
- Middleware enforces these rules both on web and API routes, guaranteeing granular security across modules.

2. Content management ↔ Search and indexing

- Every time Books, Chapters, or Pages are created, edited, or deleted, the Search and Indexing module is triggered to reindex content.
- This ensures that users always retrieve accurate, real-time search results aligned with the latest content updates.
- The internal References module also keeps links intact when content is renamed or reorganized, maintaining search relevance and navigation integrity.

3. User management ↔ Notifications and activity tracking

User creation, role changes, and content interaction events are logged and monitored by the Activity Tracking module, which in turn can trigger notifications.

For example, when a user is mentioned in a comment or added to a content permission group, the Notifications module delivers an in-platform alert or email update.

These interactions ensure users remain informed, accountable, and connected.

4. API module ↔ Core functional modules

- The API module exposes the functionality of the Content Management, User Management, and Permissions modules to external systems.
- It allows for:
    - Programmatic content creation and retrieval
    - Automated user management
    - External permission syncing or audit log extraction
- The API is protected by the Authentication module, ensuring secure token-based access.

5. Frontend (UI & interactive elements) ↔ All backend modules

- The Frontend module pulls data from backend services—like content, users, permissions, and settings—and presents it in a responsive, interactive interface.
- It interacts with the Theming and Customization module for layout and branding, and leverages JavaScript and Blade for dynamic behaviours like editing, live search, or content previews.
- It also relies on Localization and Translation to render the interface in the user's preferred language.

6. System configuration & settings → Global control layer

- This module provides centralized management of system-wide behaviour, influencing how all other modules function.
    - It defines default settings, such as language, registration policies, and caching behaviour.
    - Logging and performance configurations affect how the Activity Tracking and Console & Maintenance modules perform diagnostics and audits.

7. Backend administration (console & maintenance) ↔ Infrastructure backbone

- The Console module provides CLI commands for regenerating permissions, cleaning revisions, seeding test data, and performing backups—affecting nearly all other modules.
- Automated tasks and Artisan commands streamline platform upkeep and trigger system-level events like reindexing search data or clearing configuration caches.

8. Database management ↔ All data-driven modules

- Every module that creates, reads, or modifies data depends on the structures managed by the Database Management module.
- Migrations, seeders, and backups support safe updates, test generation, and data recovery for modules such as:
    - Content Management (Books, Pages)
    - User Management (Users, Roles)
    - Activity Logs and Permissions

9. Theming and customization ↔ Frontend & user experience

- This module works in parallel with the Frontend to apply custom styles, layout changes, branding, and UI enhancements.
- It allows institutions to inject custom HTML, CSS, or scripts—affecting how users interact with content and system elements visually.

10. Localization and translation ↔ Global user experience adaptation

- The Localization module ensures all content in the UI, notifications, and system messages reflects the user's selected language.
- It interfaces with all user-facing modules (Frontend, Content, Notifications) to ensure global accessibility and inclusivity.

Conclusion

These interconnected modules form a cohesive and robust platform, where each part contributes to the larger goal of delivering a secure, adaptable, and user-centric digital space. From backend configuration and automation to frontend interaction and user engagement, every module is designed to collaborate and communicate, ensuring that the platform remains modular in design but unified in experience.

This chapter provides a detailed overview of the platform's modular architecture, emphasizing how each component contributes to its functionality. By understanding the role and interactions of these modules, developers and administrators can effectively manage, maintain, and extend the system.

5. Code implementation details

This chapter delves into the technical implementation of the TET platform's core features, focusing on how specific functionalities are programmed and integrated. It also provides an overview of external libraries and APIs utilized in the system, as well as error handling and logging mechanisms.

5.1 Authentication and authorization

The platform employs a robust authentication and authorization system to ensure secure access and proper user role management.

The **Authentication** system in the platform ensures secure access to user accounts by implementing modern security practices. User credentials are securely stored using bcrypt, a robust hashing algorithm designed to protect passwords from unauthorized access. Laravel's built-in `Auth` system simplifies the handling of essential authentication functionalities, including user login, registration, password resets, and session management. For organizations requiring additional security, the platform supports multi-factor authentication (MFA), providing an extra layer of protection against unauthorized access.

The **Authorization** system complements authentication by controlling what each user can access within the platform. Role-based permissions are implemented using Laravel's `Gate` and `Policy` classes, which allow for precise definition and enforcement of user privileges. Roles such as Admin, Editor, and Viewer are assigned to users and stored in the database, with each role granting specific permissions tailored to their responsibilities. Middleware is used to intercept requests, ensuring that only users with the necessary permissions can access restricted routes and perform sensitive actions. This combination of robust authentication and fine-grained authorization ensures that the platform remains both secure and highly customizable to organizational needs.

5.2 Data storage and retrieval (database schema)

The platform uses a relational database (MySQL/MariaDB) to store content, user data, and metadata. The database schema is optimized for the hierarchical structure of Bookshelves, Books, Chapters, and Pages.

The platform's database schema is carefully designed to support its functionality and hierarchical content structure. The `users` table is central to managing user accounts, storing essential details such as hashed passwords and assigned roles. Content is organized across

three core tables: `books`, `chapters`, and `pages`, which mirror the platform's hierarchical structure. Each of these tables includes fields for metadata, such as titles, descriptions, timestamps, and content versions, ensuring comprehensive data management. Additionally, the `roles` and `permissions` tables define the various access levels and permissions associated with each role, forming the foundation for the platform's role-based authorization system. For enhanced search capabilities, the `search_index` table can optionally store pre-processed content, enabling efficient full-text searches. The table below briefly describes all database tables in detail.

Table Name	**Description**
`activities`	Logs user activity across the platform for auditing and analytics.
`api_tokens`	Stores API access tokens for users or integrations.
`attachments`	Holds metadata and file references for uploaded attachments.
`books`	Represents the main container for content, consisting of chapters and pages.
`bookshelves`	Organizes books into groups or collections.
`bookshelves_books`	Pivot table linking books to bookshelves (many-to-many relationship).
`cache`	Stores temporary cache data to improve performance.
`chapters`	Subsections within books that organize pages.
`comments`	User comments made on pages.
`deletions`	Tracks soft-deleted entities for potential restoration.
`email_confirmations`	Stores email confirmation tokens for verifying user emails.

entity_permissions	Defines specific permissions for users or roles on content items.
failed_jobs	Logs background jobs that have failed.
favourites	Tracks which content items users have marked as favourites.
images	Stores metadata and references for uploaded images.
jobs	Background jobs queued for asynchronous execution.
joint_permissions	Precompiled permission sets for faster access control checks.
mfa_values	Stores multi-factor authentication data.
migrations	Tracks database migrations applied to the system.
page_revisions	Historical versions of pages, used for versioning and rollback.
pages	Individual content items that make up chapters or books.
password_resets	Stores tokens for password reset requests.
permission_role	Links permissions to roles (many-to-many relationship).
references	Tracks links and references between content entities.
role_permissions	Maps specific permissions to roles.
role_user	Links users to their assigned roles.
roles	Defines roles that group permissions for users.
search_terms	Logs search queries made by users.
sessions	Stores active session data for logged-in users.

settings	Holds application-wide configuration settings.
social_accounts	Stores linked social login account information.
tags	Metadata labels assigned to entities for categorization.
user_invites	Stores pending invitations to register and join the platform.
users	Holds user account information.
views	Tracks view counts and metadata for content items.
watches	Tracks user subscriptions to content changes.
webhook_tracked_events	Stores events triggered for webhooks.
webhooks	Configuration for outgoing webhook integrations.

The relationships between these tables are established using foreign keys to maintain the hierarchical organization of Books, Chapters, and Pages. This approach ensures that each chapter is associated with a specific book and each page with a specific chapter, preserving content integrity. Similarly, users are linked to roles through pivot tables, enabling a many-to-many relationship that allows flexibility in assigning multiple roles to a single user or sharing roles across multiple users.

Data retrieval is managed through Laravel's Eloquent ORM (Object-Relational Mapping), which provides an intuitive interface for querying and manipulating data. The ORM simplifies complex database interactions, making it easier for developers to work with the schema. Additionally, optimized queries are utilized to fetch hierarchical content efficiently, ensuring fast and reliable performance even when dealing with large volumes of data. This thoughtful database design ensures the platform's scalability and maintainability while maintaining its responsiveness and usability.

5.3 Content management features

The core functionality of the platform revolves around managing content. This includes creating, editing, organizing, and searching for content.

The **Content Uploading** feature allows users to seamlessly upload files, such as PDFs, images, and videos, through the frontend interface. These files are stored securely in the `storage` directory, organized to ensure easy access and management. Laravel's `Storage` facade is utilized to handle file uploads, offering a flexible and consistent API for managing files. This approach ensures compatibility with various storage options, whether the platform is using local storage or integrating with cloud-based solutions like AWS S3. The system is designed to scale with the needs of the institution, allowing efficient handling of large volumes of media.

Tagging provides a powerful way to categorize and organize content. Tags are stored in a dedicated `tags` table and are linked to individual content items (such as Books, Chapters, or Pages) via pivot tables. This relational setup allows for a many-to-many relationship, where a single content item can have multiple tags, and each tag can be associated with multiple items. Users can easily add, edit, or remove tags directly through the platform, providing a dynamic way to group and search for related materials. By enabling better categorization, tagging enhances the usability of the platform for both educators and learners.

The **Searching** functionality is a cornerstone of the platform, making it easy for users to find the content they need quickly. Full-text search can be implemented using Laravel Scout, which offers seamless integration with the database or external search engines like Elasticsearch for more advanced capabilities. Search queries are processed to include keyword matching, filters for specific fields, and relevance scoring, ensuring that results are accurate and prioritized according to user intent. The platform automatically updates search indexes whenever content is created, updated, or deleted, ensuring that the search results remain up-to-date without manual intervention. This robust search system enables users to navigate even extensive repositories with ease, making content discovery both efficient and intuitive.

5.4 External Libraries and APIs Used

The TET platform keeps its external dependencies lean and focused on core needs, ensuring both performance and extensibility. For file and media storage, it relies on **League Flysystem** to abstract filesystem operations and supports adapters for **AWS S3**, **Azure Blob**, **Rackspace**, **SFTP**, **WebDAV**, and more. Outgoing emails are handled through **SwiftMailer** via **Laravel's** mailer, allowing integration with any SMTP service such as **Gmail**, **SendGrid**, or **Mailgun**.

Authentication is extended with industry-standard protocols: **OAuth 1** and **2** are managed through **Laravel Socialite** and its **SocialiteProviders** extensions (covering **GitHub**, **Google**,

Slack, Azure AD, **Discord**, **GitLab**, **Okta**, **Twitch**, and others), while SAML-based single sign-on is supported via the **OneLogin PHP SAML Toolkit** and **LDAP** directories through packages like **adldap2/adldap2-laravel**. To bolster security, the TET platform offers two-factor authentication using the **PragmaRX Google2FA** bridge for time-based one-time passwords.

For document export, BookStack integrates **Dompdf** (via the **barryvdh/laravel-dompdf wrapper**) and **wkhtmltopdf** through **Snappy**, enabling on-demand PDF generation from HTML content. When communicating with external services—such as webhooks or custom APIs—it uses **Guzzle** for robust HTTP requests, alongside utility libraries like **phpdotenv** for environment configuration, **phpseclib** for cryptographic routines, **Carbon** for date handling, and **Monolog** for logging.

On the front end, rich-text editing is powered by **TinyMCE** and **Lexical**, while live **Markdown** editing leverages **CodeMirror** paired with markdown-it and its task-list plugin. Drag-and-drop reordering comes courtesy of **Sortable.js**, icons are provided by Google's **Material** Icons, and browser-based storage uses Jake Archibald's **idb-keyval** for **IndexedDB**. Finally, the TET platform can embed the diagrams.net editor via a CDN script and allows administrators to inject custom JavaScript or CSS directly into the page head for additional customization.

This carefully curated selection of libraries and APIs ensures the TET platform remains a streamlined, high-performance documentation platform without unnecessary bloat..

5.5 Error handling and logging mechanisms

Effective **error handling and logging mechanisms** are essential for maintaining the reliability and stability of the platform. These mechanisms are designed to identify, address, and document issues as they arise, ensuring that the system remains maintainable and responsive to user needs. By providing meaningful feedback to users and detailed logs for developers, the platform minimizes downtime and facilitates efficient troubleshooting.

Error handling

The platform's error handling is built on Laravel's robust exception handling system, which provides a framework for catching and managing errors across the application.

- **Centralized exception handling**: Laravel's exception handler ensures that errors are captured consistently throughout the application. When an error occurs, it is automatically logged, and the appropriate response is sent to the user.

- **Custom exception classes**: For specific scenarios, such as unauthorized access or invalid input, custom exception classes are implemented. These classes provide tailored error handling, allowing the platform to respond appropriately to unique cases.
- **User-friendly error messages**: End-users are shown clear and meaningful error messages that help them understand the issue without exposing sensitive technical details. For example, an invalid login attempt might prompt the user to check their credentials, while developers receive a more detailed log of the error.

This structured approach to error handling ensures that issues are managed efficiently while maintaining a secure and user-friendly experience.

Logging mechanisms

The platform employs Laravel's built-in logging system, powered by the Monolog library, to record and categorize errors, warnings, and system events. This logging infrastructure is critical for tracking the platform's behaviour and diagnosing issues.

- **Log storage and configuration**: By default, logs are stored in the `storage/logs` directory, organized by date for easy access. Administrators can configure the logging system to use external services, such as Logstash or AWS CloudWatch, for centralized log management and analysis.
- **Log levels**: The platform uses standard log levels, including DEBUG, INFO, WARNING, and ERROR, to categorize and prioritize log entries. This allows developers and administrators to focus on critical issues while still having access to detailed information for lower-priority events.
- **Structured logging**: Each log entry includes timestamps, error codes, and contextual information, making it easier to trace the root cause of issues during debugging.

The logging system ensures that all significant events are documented, providing valuable insights into the platform's operation and performance.

Monitoring and alerts

To enhance the platform's observability, external monitoring tools can be integrated to track application performance and capture runtime errors in real time.

- **Performance tracking**: Tools like **Sentry** or **New Relic** can be used to monitor key metrics, such as response times, resource usage, and error rates. These tools help administrators identify bottlenecks and optimize performance proactively.

- **Error alerts**: Alerts can be configured to notify administrators of critical issues as they occur. For instance, an error resulting in a service outage might trigger an email or message via platforms like **Slack** or **Microsoft Teams**, enabling a rapid response.
- **Dashboard integration**: Monitoring tools often include dashboards that provide an overview of system health, making it easier for administrators to detect trends and address potential problems before they impact users.

Benefits of error handling and logging mechanisms

The combination of robust error handling, detailed logging, and proactive monitoring ensures the platform's reliability and maintainability. For developers, these mechanisms provide the information needed to identify and resolve issues quickly, reducing development and debugging time. For administrators, real-time alerts and logs enhance their ability to monitor system health and respond to critical events.

Users benefit from clear error messages that help them resolve minor issues independently, while the platform's logging infrastructure prevents these issues from escalating into larger problems. Overall, these mechanisms contribute to a stable, secure, and user-friendly platform that can adapt and scale effectively over time.

This chapter provides a technical overview of how the platform's core features are implemented, highlighting the use of modern tools and best practices. By understanding these details, developers can effectively maintain and extend the platform while ensuring its stability and performance.

6. Interfaces and integration

The interfaces and integration chapter explores the mechanisms that allow the TET platform to connect with external tools and systems, enabling seamless interoperability and enhanced functionality. This includes the design and functionality of the platform's APIs, which expose core features for programmatic access, and the integration of third-party services that extend the platform's capabilities. By providing well-documented endpoints and supporting flexible integrations, TET platform ensures that institutions can customize and expand the platform to meet their unique needs.

6.1 APIs and endpoints (with sample requests and responses)

The **platform API** exposes the platform's core functionality, allowing developers to interact with Books, Chapters, Pages, and user accounts programmatically. Built using RESTful principles, the API is straightforward to use, with clear endpoints, predictable request formats, and consistent response structures.

Key API features:

- **Content management**: Endpoints for creating, updating, retrieving, and deleting Books, Chapters, and Pages.
- **User management**: APIs to manage user accounts, roles, and permissions.
- **Search and metadata retrieval**: Tools for querying the platform's content and metadata programmatically.

Example API usage:

1. Retrieving a list of books:

Endpoint: `GET /api/v1/books`

Request:

```
GET /api/v1/books HTTP/1.1
Host: tet-erasmus.com
Authorization: Bearer YOUR_API_TOKEN
```

Response:

```
[
  {
    "id": 1,
    "name": "Introduction to Programming",
    "description": "A beginner's guide to programming concepts",
    "created_at": "2025-01-20T14:55:00Z",
    "updated_at": "2025-01-25T08:15:00Z"
  },
  {
    "id": 2,
    "name": "Advanced Mathematics",
    "description": "Comprehensive material on calculus and algebra",
    "created_at": "2025-01-22T10:30:00Z",
    "updated_at": "2025-01-25T08:15:00Z"
  }
]
```


2. Creating a new page:

Endpoint: `POST /api/v1/pages`

Request:

```
POST /api/v1/pages HTTP/1.1
Host: tet-erasmus.com
Authorization: Bearer YOUR_API_TOKEN
Content-Type: application/json

{
  "book_id": 1,
  "chapter_id": 2,
  "name": "Variables and Data Types",
  "html": "<p>Understanding variables and their types in
programming...</p>"
}
```

Response:


```
{
  "id": 25,
  "book_id": 1,
  "chapter_id": 2,
  "name": "Variables and Data Types",
  "created_at": "2025-01-25T09:10:00Z",
```

```
  "updated_at": "2025-01-25T09:10:00Z"
}
```

3. Searching for content:

Endpoint: `GET /api/v1/search?query=programming`

Request:

```
GET /api/v1/search?query=programming HTTP/1.1
Host: tet-erasmus.com
Authorization: Bearer YOUR_API_TOKEN
```

Response:

```
{
  "results": [
    {
      "id": 5,
      "type": "page",
      "name": "Introduction to Programming",
      "url": "/books/1/pages/5"
    },
    {
      "id": 10,
      "type": "page",
      "name": "Programming Basics",
      "url": "/books/2/pages/10"
    }
  ]
}
```

The API documentation is hosted within the platform or on services like **ReadTheDocs**, providing developers with details on all endpoints, required parameters, and expected responses. Authentication is managed using **token-based authentication**, ensuring secure access to API resources.

6.2 Third-party integrations

TET platform supports integration with a curated set of third-party tools and services, chosen for reliability and compatibility. These integrations enhance storage, authentication, search, communication, analytics, and interoperability with educational ecosystems.

Key integrations:

1. **Cloud storage services**
   - AWS S3 via **League Flysystem** adapter, ideal for scalable media storage.
   - **Google Cloud Storage** & **Azure Blob Storage** through respective **Flysystem** adapters.

2. **Single sign-on (SSO)**

   - OAuth 1 & 2 via **Laravel Socialite** and **SocialiteProviders** (GitHub, Google, Microsoft, Slack, Discord, etc.).
   - SAML SSO using **OneLogin PHP SAML Toolkit**.
   - LDAP integration through for directory-based authentication.

3. **Search engines**
   - Elasticsearch connectivity via custom Laravel Scout driver or direct API calls using **Guzzle**.
   - **Algolia** integration for instant, typo-tolerant search.

4. **Email delivery**
   - **SwiftMailer** managed by Laravel's mailer, compatible with SMTP services like **SendGrid**, **Mailgun**, **Postmark**, or **Gmail SMTP**.
   - Transactional email APIs via direct HTTP calls using Guzzle when higher throughput or tracking is required.

5. **Analytics platforms**
   - **Google Analytics** via JavaScript snippet or server-side measurement protocol.
   - **Matomo** integration through JavaScript embed or PHP tracking API for self-hosted analytics.

6. **PDF generation**
   - **Dompdf** for simple HTML-to-PDF rendering.
   - wkhtmltopdf via Snappy for advanced PDF layouts.

7. **Two-Factor Authentication**
   - TOTP-based 2FA with **PragmaRX Google2FA**.

8. **Front-end editors & utilities**
   - **TinyMCE** & **Lexical** for rich-text editing.
   - **CodeMirror** + markdown-it (with task-list plugin) for live Markdown editing.
   - **Sortable.js** for drag-and-drop reordering.
   - **Google Material Icons** for UI icons.
   - **IndexedDB** storage for client-side persistence.

9. **Embeds & custom scripts**
   - **diagrams.net** embed script via CDN for in-app diagram creation.
   - Custom JS/CSS injection capability through admin settings for additional third-party widgets or styling.

This targeted suite of integrations ensures the TET platform remains versatile and scalable, aligning with institutional requirements without unnecessary overhead.

Benefits of interfaces and integration

The combination of well-documented APIs and seamless third-party integrations makes TET platform a flexible and scalable solution. The API enables developers to automate tasks, create custom tools, and connect the platform to a broader ecosystem of educational technologies. Meanwhile, third-party integrations provide enhanced functionality, such as cloud storage, improved search, and user analytics, empowering administrators to tailor the platform to their specific needs.

By supporting these interfaces, TET platform ensures that it can adapt to evolving requirements, integrate smoothly into existing workflows, and remain a robust solution for managing educational content.

7. Development and deployment

The development and deployment chapter provides a guide for setting up the TET platform's development environment, building the application, and deploying it in production. Additionally, it outlines the use of CI/CD pipelines (if applicable) to streamline the process of testing, building, and deploying updates. By following these instructions, developers and administrators can ensure a reliable and scalable deployment while maintaining an efficient development workflow.

7.1 Instructions for setting up the development environment

Setting up the development environment for platform involves installing the required dependencies, configuring the application, and preparing the local environment for testing and development.

1. System requirements

- **Operating system**: Linux-based (Ubuntu/Debian recommended) or Windows.
- **PHP**: Version 8.0 or higher.
- **Composer**: Dependency manager for PHP.
- **Node.js and npm**: For building frontend assets.
- **Database**: MySQL/MariaDB for local testing.
- **Web server**: Apache or Nginx.

2. Steps to set up the environment

- Clone the repository:

```
git clone https://github.com/BookStackApp/BookStack.git
cd BookStack
```

- Install PHP dependencies using Composer:

```
composer install
```

- Install Node.js dependencies and build assets:

```
npm install
npm run dev
```

- Copy the example environment file and update it with local settings:

```
cp .env.example .env
```

Configure `.env` with database credentials, application URL, and other local settings.

- Generate the application key:

```
php artisan key:generate
```

- Run database migrations to set up the schema:

```
php artisan migrate
```

- Start the development server:

```
php artisan serve
```

- Access the application in a browser at `http://localhost:8000`.

3. Testing the environment

Run the test suite to ensure the local environment is functioning correctly:

```
php artisan test
```

7.2 Build and deployment processes

The deployment process involves preparing the application for a production environment, including optimizing performance and securing the server.

1. Preparing the build

- Compile frontend assets for production:

```
npm run production
```

- Optimize the application for deployment:

```
php artisan optimize
```

2. Setting up the production server

- Install and configure a web server (Apache or Nginx) to serve the application.
- Secure the server with SSL/TLS for HTTPS connections using a tool like Let's Encrypt.
- Set appropriate file permissions for the storage and bootstrap/cache directories to ensure the application can write to these locations.

3. Deploying the application

- Upload the built application files to the server or pull them directly from the Git repository.
- Configure the `.env` file for the production environment, including database credentials, caching settings, and email server details.
- Run database migrations and seeders (if necessary):

```
php artisan migrate --force
```

- Restart the web server to apply changes.

4. Post-deployment tasks

- Clear application caches to ensure the latest configurations are applied:

```
php artisan config:clear
php artisan cache:clear
```

- Monitor logs and test the application in the production environment to verify successful deployment.

7.3 CI/CD pipelines

For projects requiring frequent updates or complex deployments, a **Continuous Integration/Continuous Deployment (CI/CD)** pipeline can streamline the process and reduce manual effort. Tools such as GitHub Actions, GitLab CI, or Jenkins can be used to automate testing, building, and deployment.

1. Pipeline overview

A typical CI/CD pipeline for the platform includes the following stages:

- Build: Installs dependencies, compiles assets, and prepares the application for deployment.
- Test: Runs the application's test suite to ensure changes do not introduce regressions.
- Deploy: Pushes the application to the production or staging server.

2. Example CI/CD pipeline configuration (GitHub Actions):

Create a `.github/workflows/deploy.yml` file:

```
name: Deploy TET platform

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up PHP
      uses: shivammathur/setup-php@v2
      with:
        php-version: 8.0

    - name: Install dependencies
      run: composer install --no-dev --optimize-autoloader

    - name: Install Node.js
      uses: actions/setup-node@v2
      with:
        node-version: 16

    - name: Build assets
      run: npm install && npm run production
```

```
      - name: Deploy to server
        run: rsync -avz --exclude='node_modules' ./
user@yourserver:/path/to/tetplatform
        env:
          SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
```

3. Benefits of CI/CD

- Automates repetitive tasks, saving time and reducing errors.
- Ensures that code is thoroughly tested before deployment, improving reliability.
- Provides a streamlined and consistent deployment process, reducing downtime.

7.4 Benefits of development and deployment workflows

A well-defined development and deployment process ensures that the platform remains maintainable, secure, and scalable. Setting up a structured development environment allows developers to work efficiently, while optimized deployment practices minimize disruptions during updates.

For teams adopting CI/CD pipelines, the automation of testing and deployment reduces manual intervention, accelerates delivery cycles, and ensures a consistent, high-quality application. This comprehensive approach to development and deployment empowers institutions to make the most of the TET platform while maintaining operational excellence.

8. Testing

The testing chapter outlines the strategies, frameworks, and methodologies used to ensure the stability, reliability, and quality of the TET platform. Testing is an integral part of the development and maintenance process, helping identify and address potential issues before they impact end-users. This chapter describes the testing strategies, provides examples of unit testing, and explains the balance between automated and manual testing procedures.

8.1 Testing strategies and frameworks used

The testing approach for the TET platform is built around a combination of **unit tests**, **integration tests**, and **end-to-end tests**, ensuring comprehensive coverage of the platform's functionality. The following strategies are implemented:

1. Unit testing

- Focuses on testing individual components or functions in isolation to verify their behaviour.
- Ensures that specific pieces of the application, such as models or utility functions, work as expected.

2. Integration testing

- Validates how different components of the platform interact with one another, such as controllers interacting with models and views.
- Ensures that data flows correctly through the application's layers.

3. End-to-end (E2E) testing

- Simulates real-world user interactions to validate the platform's functionality as a whole.
- Covers critical workflows, such as content creation, user authentication, and permission management, ensuring that the platform performs well under typical usage scenarios.

4. Regression testing

- Rechecks existing features to ensure that new updates or changes do not introduce unexpected bugs.
- Often automated to efficiently verify functionality across multiple builds.

Frameworks used

- **PHPUnit**: The primary framework for writing and running unit and integration tests in Laravel.
- **Laravel Dusk**: Used for E2E testing, providing tools to simulate user interactions in a browser environment.
- **Mockery**: Facilitates mocking of dependencies during testing, ensuring that tests remain isolated and predictable.

8.2 Unit testing examples

Unit tests in focus on verifying the correctness of individual components, such as models and utility functions. Below is an example of a unit test for verifying the creation of a Book in the platform.

Test case: Verifying book creation

File: `tests/Unit/BookTest.php`

```
namespace Tests\Unit;

use Tests\TestCase;
use App\Models\Book;

class BookTest extends TestCase
{
    / @test */
    public function it_creates_a_book_successfully()
    {
        // Arrange
        $data = [
            'name' => 'Test Book',
            'description' => 'A book for testing purposes',
        ];

        // Act
        $book = Book::create($data);

        // Assert
        $this->assertInstanceOf(Book::class, $book);
        $this->assertEquals('Test Book', $book->name);
        $this->assertEquals('A book for testing purposes', $book-
```

```
>description);
    }
}
```

This test verifies that the `Book` model correctly handles data creation and ensures the object properties are properly assigned.

8.3 Automated and manual testing procedures

Testing combines automated and manual approaches to ensure thorough coverage across all functionality.

Automated testing

- **Focus**: Automates repetitive and predictable tests, such as unit tests, regression tests, and integration tests.

- **Tools**: Utilizes PHPUnit for backend logic and Laravel Dusk for E2E scenarios.

- **Examples of automated tests**:

- Verifying the correct functionality of API endpoints (e.g., retrieving a list of Books).
- Testing role-based permissions to ensure unauthorized users cannot access restricted features.
- Simulating user workflows, such as creating and editing content.

- **Advantages**:

- Saves time on repetitive tasks.
- Identifies bugs early in the development process.
- Ensures consistency by running tests automatically in CI/CD pipelines.

Manual testing

- **Focus**: Validates user experience and detects edge cases or visual issues that automated tests might miss.

- **Procedures**:

- Conduct exploratory testing of new features to evaluate usability and design.
- Perform smoke testing after deployments to verify that critical functionalities are working as expected.

- Review edge cases, such as unexpected input formats or rare workflows.

- **Examples of manual tests**:

- Checking the responsiveness of the platform on various devices (desktop, tablet, mobile).
- Validating the accuracy of content search and tagging.
- Ensuring the platform's visual elements align with institutional branding.

Combining automated and manual testing

Both automated and manual testing complement each other to provide a robust testing framework. Automated testing ensures repeatability and efficiency, while manual testing captures the nuances of real-world usage scenarios. For example:

- After running automated regression tests, manual exploratory tests can focus on new features or areas not yet automated.
- Post-deployment, smoke testing verifies the production environment while automated tests ensure backend stability.

8.4 Benefits of testing workflows

The structured approach to testing ensures that the platform remains reliable, secure, and user-friendly. Key benefits include:

- **Improved quality**: Early detection of bugs reduces the likelihood of issues affecting end-users.
- **Reduced downtime**: Rigorous testing minimizes the risk of regressions or deployment failures.
- **Enhanced user trust**: A stable platform fosters confidence among users, improving adoption and engagement.

By leveraging automated frameworks like PHPUnit and Laravel Dusk alongside thorough manual testing procedures, the TET platform achieves a high standard of quality assurance, ensuring that it performs well in both development and production environments.

9. Future improvements

Looking forward, our goal is to evolve TET into a living, adaptive environment where creation, discovery and assessment happen in seamless harmony. By sharpening search and discovery tools, enriching collaborative spaces with real-time feedback and discussions, and layering in intelligent organization frameworks, we can make every piece of content—notes, quizzes, simulations—effortlessly accessible and instantly editable by the people who know it best: instructors and learners.

At the same time, we're setting our sights on deeper integrations and richer experiences: embedding TET within existing course ecosystems, enabling truly interactive and multimedia-driven lessons, and opening the platform to global classrooms through multilingual workflows and gamified engagement. Underpinning all of this is a commitment to scalability, robust governance and performance—so that whether a single seminar or an entire university system, TET scales to meet the ambition of its community. In combining these near-term refinements with bold, forward-looking initiatives, we plan transform TET from a repository into a collaborative engine that fuels pedagogical innovation at every level.

9.1 Recommendations for enhancements

1. Advanced search capabilities

- Enhance search functionality to help students and teachers quickly locate relevant study materials.
- **Faceted search**: Enable filtering by tags, subjects, contributors, or content type to streamline resource discovery.
- **Search result highlights**: Display excerpts from content where the search term appears, providing immediate context.
- **Synonym support**: Automatically include results for related terms, making searches more intuitive for students.
- Integrate advanced search engines like Elasticsearch to handle large repositories efficiently, particularly for institutions with vast collections of co-created materials.

2. Improved collaborative tools

- Expand real-time editing to include **live cursors** and a **change history view**, enabling teachers and students to track contributions during co-creation.

- Introduce **in-document discussions** where users can add comments or feedback on specific sections, supporting active collaboration in developing materials.
- Allow resolution of comment threads to facilitate organized and goal-oriented teamwork.

3. Enhanced content organization

- Provide customizable content hierarchies, allowing teachers to organize materials by courses, semesters, or topics.
- Add **content templates** for teachers and students to standardize formatting for specific purposes (e.g., lecture notes, assignments, or study guides).
- Enable automatic cross-referencing between related Books, Chapters, or Pages to link study resources seamlessly.

4. Offline access and synchronization

- Introduce an offline mode to allow students and teachers to download study materials and contribute to content creation without requiring an active internet connection.
- Implement auto-sync functionality to update content when users reconnect, ensuring that all co-created materials remain consistent across devices.

5. Analytics and engagement tracking

- Provide analytics dashboards tailored for higher education:
- Teachers can monitor how often materials are accessed, identify popular content, and gauge student engagement.
- Students can view personalized dashboards tracking their progress through shared study materials.
- Enable exportable reports to help educators refine content and address gaps in knowledge areas based on student interactions.

6. More robust permissions system

- Introduce **collaboration-specific roles**, such as “Reviewer” or “Contributor,” to define granular permissions for content co-creation.
- Allow flexible role configurations within courses or projects to encourage a blend of teacher-guided and student-led initiatives.

9.2 Areas for further development

1. Integration with learning management systems (LMS)

- Develop seamless integrations with LMS platforms like **Moodle**, **Canvas**, or **Blackboard**, allowing study materials co-created on TET platform to be directly embedded into course structures.
- Sync teacher and student accounts between TET platform and LMS platforms for streamlined user management.
- Support standards like **SCORM** or **xAPI** to allow interoperability between platforms and richer tracking of content use.

2. Interactive and multimedia content support

- Extend support for interactive materials, such as quizzes, annotations, and multimedia-enhanced chapters.
- Integrate tools like **H5P** to enable teachers and students to co-create dynamic and engaging resources.
- Introduce collaborative annotations on videos, images, or documents to promote group discussions and knowledge sharing.

3. Localization and multi-language support for education

- Facilitate collaborative translation workflows where teachers and students can work together to translate content into multiple languages for global accessibility.
- Allow students to switch between language versions of study materials, ensuring they have access in their preferred language.
- Enhance RTL (right-to-left) language support for content creation and presentation.

4. Gamification to encourage engagement

- Introduce gamification elements, such as badges or leader boards, to motivate students to participate in co-creating and contributing to study materials.
- Allow teachers to set milestones or challenges, encouraging students to collaborate and contribute quality content to the repository.

5. Scalability for collaborative environments

- Optimize the platform to handle high volumes of users and content, ensuring smooth performance even during peak usage in large-scale educational settings.

- Provide tools for institutions to deploy multi-tenant instances for separate faculties, departments, or course groups while sharing the same backend infrastructure.

6. Content revision workflows for education

- Create workflows for reviewing and approving co-created content before it is published, enabling teachers to provide feedback on student contributions.
- Include a “suggest edits” feature where students can propose changes without immediately altering the material, allowing teachers to review and accept modifications.

9.3 Benefits of future improvements

By focusing on higher education use cases, these improvements will enhance the TET platform as a tool for collaborative learning and teaching. Students and teachers will benefit from advanced search and organization tools that make it easier to navigate and contribute to shared resources. Enhanced collaboration features will encourage more active participation from students, turning study material creation into a co-learning experience.

Integration with LMS platforms and the introduction of analytics will help educators monitor and refine their teaching methods, while scalability and offline support ensure that TET platform remains a reliable solution for institutions of any size. These future enhancements will empower teachers and students to co-create rich, engaging educational content, fostering collaboration, innovation, and deeper engagement in the learning process.

10. Glossary

The glossary chapter provides definitions of key technical terms used throughout the TET platform and its documentation. This section serves as a reference for teachers, students, and administrators, ensuring that users can better understand the technical terminology related to the platform, its functionality, and its development.

10.1Definitions of technical terms

Term	Definition
API (Application Programming Interface)	A set of rules and protocols that allows one application to interact with another. Platform's API enables developers to programmatically access and manage Books, Chapters, Pages, users, and other platform features.
Authentication	The process of verifying a user's identity to allow access to the platform. The platform uses secure methods, such as password hashing and multi-factor authentication, to ensure that only authorized users can log in.
Authorization	The process of determining what actions a user is allowed to perform within the platform based on their role or permissions. For example, an Editor can create and edit content, while a Viewer can only read content.
Book, Chapter, Page	The hierarchical structure used in the platform to organize content: Books are the highest-level containers, Chapters are subdivisions, and Pages are the most granular units of content.
Caching	A technique for storing frequently accessed data temporarily to improve performance and reduce server load. In the platform, caching may be used for content, search results, or configuration data.
CI/CD (Continuous Integration/Continuous Deployment)	A development practice where code changes are automatically tested, built, and deployed to staging or production environments, ensuring updates are reliable and efficiently delivered.
CMS (Content Management System)	A platform used for creating, organizing, and managing digital content. The platform is a CMS tailored specifically for educational content, allowing teachers and students to collaboratively build and manage study materials.

Term	Definition
Cron Job	A scheduled task that runs automatically at specified intervals. In the platform, cron jobs might be used for activities like sending email notifications or running backups.
Dependency	A software library or package that the platform relies on to function. For example, Laravel is a dependency that provides the core framework for the platform.
Environment file (.env)	A configuration file used to store system-specific settings, such as database credentials, email server details, and API keys. The .env file is critical for customizing the platform installations.
Full-text search	A search method that scans all text within a document or database to find matches for a query. the platform's full-text search allows users to locate content across Books, Chapters, and Pages quickly.
HTML (HyperText Markup Language)	The standard language used to create web pages. In the platform, Pages are stored in HTML format, enabling rich formatting and media embedding.
Middleware	A layer of logic that processes incoming requests before they reach the application or its routes. The platform uses middleware to handle tasks like authentication and permission checks.
Migration	A structured way to make changes to the database schema, such as adding tables or modifying fields. Laravel migrations are used in the platform to manage the database consistently across environments.
Multi-factor authentication (MFA)	An additional layer of security requiring users to provide two or more forms of verification (e.g., a password and a one-time code) to access the platform.
Permission	A specific capability assigned to a user role, such as the ability to create, edit, delete, or view content. Permissions ensure that users can only access or modify what they are authorized to.
Responsive design	A web design approach that ensures the platform’s interface adapts to different screen sizes and devices, such as desktops, tablets, and smartphones.
Role	A user classification in the platform that determines their permissions and level of access to the platform. Common roles include Admin, Editor, and Viewer.

Term	Definition
SSL/TLS (Secure Sockets Layer / Transport Layer Security)	Encryption protocols that secure communication between the platform and users' browsers, protecting sensitive data from interception.
Token-based authentication	A secure method for granting access to the platform API. Users or systems receive a token that must be included with API requests to verify their identity.
Version control	A system that tracks changes to content over time, allowing users to review, compare, and revert to previous versions if needed. The platform includes built-in version control for Pages.
WYSIWYG (What You See Is What You Get)	A content editor that allows users to format text, insert media, and structure content visually, without requiring knowledge of coding. The platform uses a WYSIWYG editor to simplify content creation for teachers and students.
Webhook	An automated message sent to an external system when a specific event occurs in the platform, such as creating a new Page or updating content. Webhooks enable integration with other platforms.
YAML (YAML Ain't Markup Language)	A human-readable data serialization format often used for configuration files. The platform uses YAML for certain configuration options, such as defining custom themes.

The glossary provides teachers, students, and administrators with an accessible reference for technical terms related to the platform. By simplifying complex concepts and explaining platform-specific terminology, this section ensures that users can navigate and utilize the platform effectively, regardless of their technical expertise. It fosters a shared understanding among all stakeholders, enabling better collaboration and smoother adoption of the platform.

Appendices

The Appendices provides a collection of practical resources for developers working with the platform. This includes key code snippets from the platform itself, examples of API interactions, and configuration files. These resources serve as a hands-on reference for understanding and extending the platform's functionality in higher education settings, where teachers and students collaboratively create and manage content.

Code snippets for key features

Below are examples of the platform **code snippets** that demonstrate core platform functionality. These are based on Laravel and show how various features are implemented within the application.

1. Retrieving a list of books in a controller

The platform uses Laravel's MVC architecture. Here's an example of a controller method to retrieve a list of all books:

```
namespace App\Http\Controllers;

use App\Models\Book;

class BookController extends Controller
{
    public function index()
    {
        // Retrieve all books from the database
        $books = Book::all();

        // Return a view with the list of books
        return view('books.index', compact('books'));
    }
}
```

Explanation:

This code retrieves all books from the database using Laravel's `Book::all()` Eloquent query and passes the data to the `books.index` view.

2. Creating a new page programmatically

This snippet demonstrates how to create a new page for a book and chapter programmatically:

```
use App\Models\Page;
use App\Models\Chapter;

$chapter = Chapter::find(1); // Get the chapter where the page will be added

$page = new Page();
$page->name = 'New Page Title';
$page->html = '<p>This is the content of the page.</p>';
$page->chapter_id = $chapter->id;
$page->book_id = $chapter->book_id;
$page->creator_id = auth()->id(); // Assign the current authenticated user as the creator
$page->save();

echo "Page created successfully with ID: " . $page->id;
```

Explanation:

This code creates a new `Page` object, assigns it to a specific chapter and book, and then saves it to the database. The `creator_id` links the page to the authenticated user.

3. Checking user permissions

The platform implements role-based permissions using middleware and policies. Below is an example of checking a user's permissions to edit a page:

```
use App\Models\Page;

$page = Page::find(1);

if (auth()->user()->can('update', $page)) {
    echo "User has permission to edit this page.";
} else {
    echo "User does not have permission to edit this page.";
}
```

Explanation:

The `can` method checks the authenticated user's permissions based on policies defined for the `Page` model. This approach ensures secure role-based access control.

4. Using middleware to enforce authentication

To secure routes, the platform uses middleware to require authentication. Here's an example of defining a route with middleware in `web.php`:

```
use Illuminate\Support\Facades\Route;

Route::middleware(['auth'])->group(function () {
    Route::get('/books', [BookController::class, 'index'])-
>name('books.index');
    Route::get('/books/{id}', [BookController::class, 'show'])-
>name('books.show');
});
```

Explanation:

The `auth` middleware ensures that only authenticated users can access the `/books` routes. Unauthenticated users will be redirected to the login page.

Configuration files

Below are examples of key configuration files used to customize and deploy the platform:

1. Example `.env` file

```
APP_NAME=TetPlatform
APP_ENV=production
APP_KEY=base64:YOUR_APP_KEY
APP_DEBUG=false
APP_URL=http://bookstack.example.com

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tet
DB_USERNAME=root
DB_PASSWORD=yourpassword

MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
```

```
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
```

Key settings:

- `APP_ENV` defines the environment (local, production).
- `DB_*` sets up the database connection.
- `MAIL_*` configures email delivery for notifications and alerts.

2. Example `docker-compose.yml` file

```
version: '3'

services:
  app:
    image: tetplatformapp/tet
    ports:
      - "80:80"
    environment:
      - DB_HOST=db
      - DB_DATABASE=tet
      - DB_USERNAME=root
      - DB_PASSWORD=yourpassword
    depends_on:
      - db
    networks:
      - tet-network

  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=yourpassword
      - MYSQL_DATABASE=tet
    volumes:
      - db-data:/var/lib/mysql
    networks:
      - tet-network

volumes:
  db-data:
```

```
networks:
  tet-network:
    driver: bridge
```

Key points:

- Defines two services: `app` (the platform) and `db` (MySQL).
- Uses volumes for database persistence.
- Configures environment variables for database settings.

These code snippets and configuration files provide practical examples of how to work with the platform, whether you're customizing the application, deploying it to production, or interacting with its API. By understanding these foundational elements, administrators and developers can fully leverage the platform's capabilities for creating and managing higher education study materials.

Lead Partner

UNIVERZA V LJUBLJANI

Partners

POLITECHNIKA RZESZOWSKA im. IGNACEGO ŁUKASIEWICZA

IN SUPREMÆ DIGNITATIS 1343 UNIVERSITÀ DI PISA

KTH VETENSKAP OCH KONST