IntimateTales Shared Dependencies

1. WordPress Core Features and Plugin Integration:
    - ACF (Advanced Custom Fields) Integration: Evident through the use of `acf-json` files.
    - Custom Post Types: Such as character_post_type and story_post_type.
    - Custom Taxonomies: Such as taxonomy_format, taxonomy_genre, etc.
2. Configurations:
    - Plugin Configuration File: config.php
    - Theme Configuration File: functions.php
3. Settings:
    - User Settings: Evident through user-settings.json
    - Design Settings: design-settings.json
    - Interaction Settings: interaction-settings.json
    - Payment Settings: payment-settings.json
    - SMTP (Email) Settings: smtp-settings.json
4. ACF Option Pages and Field Groups:
    - Various option pages and field groups created with ACF to define different settings and custom fields.
5. Function Names:

on_activation() on_deactivation() load_textdomain() enqueue_scripts_and_styles() run() get_template_part() load_template() locate_template() login_user() register_user() logout_user() authenticate_user() render_login_form() render_signup_form() render_pw_reset_form() hash_password() verify_password() generate_nonce() verify_nonce() collect_user_feedback() process_feedback_submission() load_dashboard() update_user_info() initiate_roleplay() save_roleplay_decision() send_email_notification() send_sms_notification() send_pair_request() accept_pair_request() decline_pair_request() generate_story() save_generated_story() update_challenge_status() load_leaderboard() add_points_to_user() redeem_reward() register_new_user() setup_user_profile() display_tutorial() display_all_categories() search_for_category() intimate_tales_theme_setup() toggle_dark_mode()

6. Dependency Management:
    - New Dependencies: New dependencies are added to the application through the Composer dependency management tool. The new dependency is added to the composer.json file and installed using the `composer install` command.
    - Outdated Dependencies: Outdated dependencies are updated or replaced using the `composer update` command. This command checks the composer.json file for any outdated dependencies and updates them to their latest versions.
    - Unused Dependencies: Unused dependencies are removed from the application by removing their entries from the composer.json file and running the `composer`

`update` command.

7. Dependency Optimization:
    - Reducing Dependencies: The application aims to minimize the number of dependencies to reduce the application's footprint and improve performance. This is achieved by only adding necessary dependencies and regularly reviewing and removing unused dependencies.
    - Version Control: The application uses specific versions of dependencies to ensure compatibility and stability. This also helps in reducing the risk of introducing breaking changes.
    - Autoloading: The application uses Composer's autoloading feature to only load the necessary files when they are needed. This helps in improving the performance of the application.